Job Search System using Semantic Web Technologies

*

Parikshith Kedilaya Mallar
CIDSE
Arizona State University
Tempe, Arizona, US
pkedilay@asu.edu

Raghavan Sampige Sreenivasa
CIDSE
Arizona State University
Tempe, Arizona, US
rsampige@asu.edu

Rishika Bera
CIDSE
Arizona State University
Tempe, Arizona, US
rbera1@asu.edu

Sayali Suryakant Tanawade
CIDSE
Arizona State University
Tempe, Arizona, US
ssuryaka@asu.edu

***Abstract*—With the increasing trend in online recruitment, many platforms like LinkedIn, Indeed and Glassdoor have emerged. Currently an individual interested in jobs must visit multiple platforms and filter by location and go through all the jobs listed to see which job description fits best. It can be very cumbersome to do this on multiple sites. Also, new jobs get added regularly which might require the user to constantly check sites until a desired job is obtained. We plan to create a structured web of data that gathers job profiles from multiple sites using metadata like location, job title, salary, experience and provide them in one place where users can filter by metadata attributes. Building this involves two parts. First part is to build the ontology for detecting jobs and using a reasoner to check the correctness of the ontology. Second part is linking to the Linked Open Data and querying this to retrieve the information.**

I. Introduction

Social media event detection is becoming a popular trend in semantic web mining applications with the profusion of social media platforms. Main application of this is to check for events like concerts, campaigns, news topics, jobs nearby. Currently the authors of this paper are faced with a challenge of securing a job amidst a crashing economy (due to the COVID-19 pandemic) and we want to make it easier for people like us to search for jobs. There has been an increasing trend in online recruitment with development of platforms like LinkedIn, Glassdoor and Indeed. Companies post enormous job postings everyday on these platforms. Since there are many platforms, users will have to login to each of them and check for their desired jobs. This can be a very tedious process. We want to create a platform where all the job postings are available to the users in one place and provide flexibility of filtering the jobs based on the metadata that are used to tag jobs.

We are collecting the data from open source sites like Kaggle and Google open source data. For the purpose of this project, we have collected job data from Indeed and monster.com. The major challenge is that the data is heterogeneous and there is no consistent structure to the data. Through semantic transformation we are solving the issue of data heterogeneity using Resource Description Framework (RDF) format. This approach offers better relevant data over the existing job recommendation systems.

First step in the project is to develop an ontology after looking at these data by noticing similar fields/attributes in them. After creating ontology we will link that to the Linked Open Data (LOD). We will use Apache Jena to build triples, ontology and query using ARQ(SPARQL 1.1).

II. Problem Definition

The problem addressed in this paper is to identify jobs by querying job datasets linked with metadata from numerous job aggregation websites. Metadata contains information about job title, company, location, salary, job type and posted date. The project works on a dataset that is already gathered and is not concerned with crawling the sites to find new postings or detecting new jobs getting posted.

III. Motivation and Related Literature

There has been a considerable amount of work done in this type of application and have been quite popular areas of research. An enormous amount of content is generated everyday. Maintaining and retrieving relevant content is a challenge. In [1], problem of detecting social events using the Flicker and Facebook website data and by identifying photographs as events has been addressed. The solution is implemented by using metadata fields about photographs like photoID,title, tags, description, date, time and geo-location. Paper also explains how ontology for these event's domain is implemented and events are detected using Protégé. After knowledge representation they are using HermiT reasoner for reasoning. As a last step paper discusses how media is retrieved using SPARQL query [1]. We are trying to use a similar approach as used in this paper to solve the problem with job posting platforms. We have metadata about the job postings from LinkedIn and Monster websites like job title, location and salary. Social Event Detection discussed in [1] is real inspiration for the solution we are trying to implement with job platforms.

[2, 3, 5] explain how Linked Open Data (LOD) is highly heterogeneous and very difficult to interpret as there is no systematic structure to it. In [2], the authors collected world war data and created a project called WarSampo which uses event-based approach to harmonize and create an LOD service. The purpose is that all the data which is collected on the web about war history to study history in a contextualized way where linked datasets enrich each other. The purpose of the Linked Open Data is that the idea of the WarSampo portal is to provide a variety of different perspectives to war history. For example, most datasets have their own perspectives, where the user can first search data of interest and then get linked data related to the other resources found. The idea is that the perspectives enrich each other via Linked Data. When the user clicks on an event, it is highlighted, and the historical place, time span, type, and description for the selected event are displayed. Photographs related to the event are also shown. The photographs are linked to events based on location and time.

DBPedia project [3] has extracted all the details of the data from Wikipedia. The metadata from it is out in the open as Resource Description Framework triples. The data that is being extracted is heterogeneous and it lacks proper structure and organization. The events of the datasets from the semantic web is highly unstructured and it needs to be filtered appropriately in order to access in a way so that people can query the information datasets properly. The main challenge is to provide RDF datasets as the related work is not always publicized or available. Another important thing is that we need to recognize events and rank them according to their relevance [3].

As discussed in [4, 5] there has been some progress made in detecting events on social media. [4] addresses some of the challenges faced in event detections. It states that the main problems are short and noisy contents, diverse and fast changing topics and large amounts of data and we need different strategies to handle each of these cases. As stated in [1], it also mentions that we can gather information based on the metadata available in the images posted on social media. But as discussed in [2] as well, with different types of contents posted, we have to rely on the users to provide us with the proper data and there is no way to ensure this as of now. This paper further discusses some of the uses of event detection as disaster detection, traffic detection, infectious disease outbreak detection and news detection. The underlying method in each of the uses is to gather data from various sources in real time and read the metadata associated with the data and accordingly classify the events and notify the users.

Y. Pandey and S. Bansal in their work [5] discuss how semantic web can be used in emergency management. Predicting when natural disaster can happen is nearly impossible. The Safety Check application is designed to help people who are affected by natural or man made emergency situations. First data is extracted from different sources like emergency alerts, weather alerts and contact data for geographical location that is affected in disaster. Then extracted data is used to create a semantic data model. Goal is to reach out to a large number of people as quickly and efficiently as possible. Semantic web technologies used to implement best suited solutions to handle this high volume data. This application is using Facebook's graph APIs to get people data. Web crawlers are used to gather information about cities affected in disaster. Main advantage of this is that the app is highly extensible.

James Allan, Ron Papka, Victor Lavrenko in their work highlight the challenges and workaround that can be used in online new event detection and tracking [6]. They talk about how the system should make decisions about a story before looking at the subsequent ones. Single pass clustering algorithm and a novel threshold model is proposed that can solve a huge pending gap that is present in today's event detection problems. The current world problems in event detection followed by linking the current new event with an old event has been highlighted.

To begin with, a little history on how event detection and tracking was originally proposed is discussed. Event detection being a specialization of the bigger domain Topic Detection and Tracking(TDT) is highlighted. Some research findings on TDT done by some of the prestigious institutions across the globe are presented. The major challenges and drawbacks of this research is addressed and a solution for event domain is proposed by the trio [6].

As per the work in [6], in-depth analysis about Evaluation Corpus, Evaluation Measures, Events, New Event Detection and Tracking is necessary for Social media event detection were discussed . Experiments were performed using the single pass clustering algorithm and were compared against the other algorithms already in use and the improvement in the accuracy of event detection and tracking. Threshold values are plotted on a Detection Error Tradeoff curve to reveal the results of the new algorithm. Using this a hypothesis is drawn on Detection Threshold and the performance is measured against Dimensionality and Failure Analysis was conducted on previously occurred events. A new Tracking algorithm based on Information Filtering is proposed for the purposes of Event tracking. When tested against some datasets, the findings were surprisingly better. Another feature of Adaptive Tracking was used to track events over time. The problem of events changing over time and the existing algorithms losing track of was a part of this research.

The major takeaway from all the papers that we have reviewed is that the data needs to be categorized well as data is non uniform. Building ontology is of the utmost importance in our application.

IV. Project flow

The major steps involved in the project are as follows:

- We gathered job related data from multiple sources.
- Cleanse data by removing null values and irrelevant values.
- Analyze data and create ontology.
- Create instances of data conforming to the ontology.
- Host instances on Fuseki server.
- Query on Fuseki server to fetch required results.

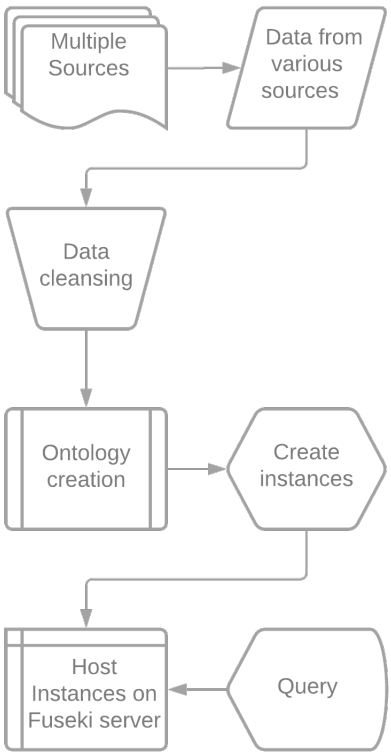

Fig. 1. Project flow

Fig. 1 shows the flow of the processes that were involved in implementing this system.

V. Data collection and Processing

We collected most of the job related data of job posting websites monster.com and indeed.com from kaggle.com and Google open source data. We analyzed the csv data for similar attributes/properties. The common attributes/properties for jobs were company name, location, salary, application link, job title, job description, posted-on date and industry to which the job belongs to. We have retained all the fields except for job description as it is difficult to filter on this field. Along with these we have added job-type field which has three values: Full-Time, Part-Time and Internship. We mocked this field on mockroo.com. Apart from that we also generated latitude and longitude details of cities and names of company CEOs and number of employees of the company. We noticed that data was inconsistent with empty columns and duplicate values. To eliminate these inconsistencies we used openrefine to cleanse the data.

Further, we created three different datasets to host them on three different Fuseki servers running on AWS EC2 instances. We wrote a java program to separate the attributes mentioned above into their respective datasets. In the first dataset, we have company related fields which are company_id, company_name, ceo_name and employee_count. In the second dataset, there are two end points. In the first endpoint, we have city_id, city_name, latitude and longitude and in the second, we have data for city and state mapping. In the third dataset, we have company, location, salary, application_link, job_title, posted_on_date, job_type and industry. Company field is mapped to first dataset using company_id as the key. Location field is mapped to second dataset using city_id as the key.

VI. Approach and High-level System Design

After designing the datasets, we designed our ontology. Here, we created three different ontologies. We created three ontologies using Protege which is an open source tool. First owl file, Company.owl has 2 classes Company and JobsInCompany. The object property in this is company_has_jobs which is a one-to-many relationship between Company and JobsInCompany classes. And this class has data properties for ceo_name, company_id, company_name and number_of_employees. Location.owl has one main class Location and it has three subclasses City, Country and State. There are And data properties are has_latitude, has_longitude, location_id, and location_name. Job.owl has five classes for JobCompany, Industry, Job, JobType and JobLocation. Industry has many sub-classes which represent various sectors to which a job can belong to. JobType has sub-classes for Fulltime, Internship and Parttime. There are object properties belongs_to_industry which is many-to-many relationship between Job and Industry, located_in which is a one-to-one relationship between Job and JobLocation, posted_by which is a one-to-one relationship between Job and JobCompany and type which is a one-to-one relationship between Job and JobType. Data properties are application_link, company_name, has_description, has_salary, has_title, location_name and posted_on.

We have defined domains and ranges for the properties. Once we developed these three owl files, we visualized our ontology which can be seen in Fig. 2. After visualizing our ontology, the next step is to generate instances of all the classes and properties in our ontology. For this we used openrefine. In openrefine, we import our csv files. In the file, we can map columns to IRIs in the ontology. We stored these instances in a Turtle files which have the extension ".ttl".

VII. Setting up fuseki servers

Once all the ttl files were generated for each of the datasets, we launched three EC2 instances on Amazon AWS platform. Apache Jena Fuseki is an open source framework that runs SPARQL queries. It runs as an operating system service or as a java web application and a standalone server. Fuseki offers two forms, web application and as a docker container. For the scope of this project we are using the web application. Fuseki provides SPARQL 1.1 protocols for query and update and SPARQL graph store protocol. We downloaded "apache-jena-fuseki-3.16.0.tar.gz" files into each of the EC2 instances and extracted them. To run fuseki server we type the command "java -jar fuseki-server.jar". We can access this server on

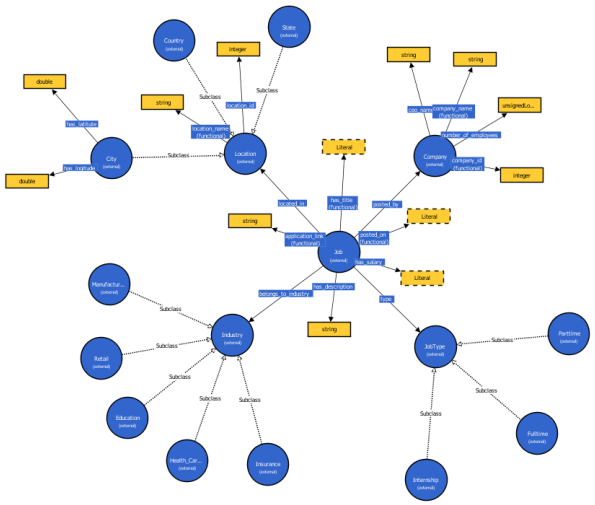

Fig. 2. Ontology

port 3030. We are going to write multiple SPARQL queries according to the business logic.

VIII. Implementation

After setting up the Fuseki server to host the datasets on AWS EC2 instances we developed a Java application to query the data from datasets. We created a Springboot application to implement REST APIs. We created SPARQL queries dynamically based on the user inputs on the user interface. The query is run on fuseki servers and data is returned to the spring boot application. The data is then routed to the user interface which renders in the user interactive format. We are using HTML, CSS, JavaScript, NodeJS, ExpressJS for front-end development. All the client-server interaction happens through REST API calls. We have three different AWS servers which run multiple instances of fuseki servers that have datasets stored in .ttl format.

The endpoints of AWS EC2 servers are http://ec2-3-129-207-101.us-east-2.compute.amazonaws.com, http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com, http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com. The fuseki servers are running on port 3030. The datasets are Locations, Jobs, Companies. We have performed visualisation on the following queries- Number of jobs based on a particular location, Number of jobs posted by a particular company, industry, job title and job type.

IX. SPARQL queries implemented

Our project has three different endpoints, so we have implemented Federated SPARQL queries where we can specify the different services where our datasets are hosted. Below are few of the sample queries we have implemented.

A. Query to fetch job related data

We need all the details related to the Job to be displayed on the front end user application. Here we fetch the details

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX jobs: <http://www.semanticweb.org/SER531/ontologies/Team-7/Jobs#>

SELECT DISTINCT ?company_id ?city_id ?salary ?link ?title ?date ?type ?industry
{
SERVICE <http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com:3030/Jobs> {
    SELECT ?company_id ?city_id ?salary ?link ?title ?date ?type ?industry
    WHERE {
        ?job jobs:posted_by ?company_id ;
             jobs:located_in ?city_id ;
             jobs:has_salary ?salary ;
             jobs:application_link ?link ;
             jobs:has_title ?title ;
             jobs:posted_on ?date ;
             jobs:type ?type ;
             jobs:belongs_to_industry ?industry .
    }
  }
}
```

Fig. 3. Query1

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX companies: <http://www.semanticweb.org/SER531/ontologies/Team-7/Companies#>

SELECT ?company_id ?company_name ?ceo ?employees
{
SERVICE <http://ec2-3-129-207-101.us-east-2.compute.amazonaws.com:3030/Companies> {
    SELECT ?company_id ?company_name ?ceo ?employees
      WHERE {
        ?company companies:company_id ?company_id ;
                 companies:company_name ?company_name ;
                 companies:ceo_name ?ceo;
                 companies:number_of_employees ?employees.
    }
  }
}
```

Fig. 4. Query2

of jobs such as company id, location id, salary, application link, job title, date posted on, job type and industry the job belongs to. We apply various filters applied by the user in the front end on job title, job type and industry. We attach the filters dynamically in the java application based on the query parameters sent to the API call. After fetching the results from the service, we return the jobs to the front end to display to the user. This query is shown in Fig. 3.

B. Query to fetch company related data

To get all the fields related to company, we have a query with service as company dataset endpoint. This query returns all the fields like company id, company name, CEO name and number of employees in the company. Our java application filters based on company name that is entered by the end user in the front end. This query is shown in Fig. 4.

C. Query to fetch location related data

Location related data is fetched using the query is shown in Fig. 5. Here we use the location endpoint to fetch all location related data like city id, city name, latitude and longitude of where the city is located.

D. Query to combine all the services together when user filters on city and company fields

We form federated query to combine all the services together. Since we have city id and company id in job service,

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX locations: <http://www.semanticweb.org/SER531/ontologies/Team-7/Locations#>

SELECT ?city_id ?city_name ?lat ?long
{
SERVICE <http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com:3030/Locations> {
    SELECT ?city_id ?city_name ?lat ?long
      WHERE {
        ?city locations:location_id ?city_id ;
              locations:location_name ?city_name ;
              locations:has_latitute ?lat;
              locations:has_logitude ?long.
    }
  }
}
```

Fig. 5. Query3

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX companies: <http://www.semanticweb.org/SER531/ontologies/Team-7/Companies#>
PREFIX jobs: <http://www.semanticweb.org/SER531/ontologies/Team-7/Jobs#>
PREFIX locations: <http://www.semanticweb.org/SER531/ontologies/Team-7/Locations#>

SELECT DISTINCT ?company_name ?city_name ?salary ?link ?title ?date ?type ?industry
WHERE {
SERVICE <http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com:3030/Locations> {
            ?location locations:location_id ?city_id ;
                      locations:location_name ?city_name .
            }
FILTER(regex(?city_name, 'san', 'i'))
SERVICE <http://ec2-3-129-207-101.us-east-2.compute.amazonaws.com:3030/Companies> {
            ?company companies:company_id ?company_id ;
                 companies:company_name ?company_name .
            }
FILTER(regex(?company_name, 'ka', 'i'))
SERVICE <http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com:3030/Jobs> {
    SELECT ?company_id ?city_id ?salary ?link ?title ?date ?type ?industry
        WHERE {
            ?job jobs:posted_by ?company_id ;
                 jobs:located_in ?city_id ;
                 jobs:has_salary ?salary ;
                 jobs:application_link ?link ;
                 jobs:has_title ?title ;
                 jobs:posted_on ?date ;
                 jobs:type ?type ;
                 jobs:belongs_to_industry ?industry .
        }
    }
}
```

Fig. 6. Query4

we need to fetch city name and company name from location service and company service respectively. Since we have filters for city and company, we first fetch city name and then company name and then implicitly join on job service data. Fig. 6 shows this query.

E. Query to combine all the services together without filters on city and company fields

This is similar to the previous query. But since we do not have filter on company and city, we fetch first 100 jobs and then implicitly join on city service and location service. This is shown in Fig. 7.

F. Query to get all company related data for visualization

If a user has entered any string in the company field, we show the user details of the company like ceo name, number of employees and the number of jobs posted by that company. This is shown in Fig. 9.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX companies: <http://www.semanticweb.org/SER531/ontologies/Team-7/Companies#>
PREFIX jobs: <http://www.semanticweb.org/SER531/ontologies/Team-7/Jobs#>
PREFIX locations: <http://www.semanticweb.org/SER531/ontologies/Team-7/Locations#>

SELECT DISTINCT ?company_name ?city_name ?salary ?link ?title ?date ?type ?industry {
SERVICE <http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com:3030/Jobs> {
    SELECT ?company_id ?city_id ?salary ?link ?title ?date ?type ?industry
    WHERE {
        ?job jobs:posted_by ?company_id ;
             jobs:located_in ?city_id ;
             jobs:has_salary ?salary ;
             jobs:application_link ?link ;
             jobs:has_title ?title ;
             jobs:posted_on ?date ;
             jobs:type ?type ;
             jobs:belongs_to_industry ?industry .
      }
    LIMIT 100
    }
SERVICE <http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com:3030/Locations> {
    ?location locations:location_id ?city_id ;
              locations:location_name ?city_name .
}
SERVICE <http://ec2-3-129-207-101.us-east-2.compute.amazonaws.com:3030/Companies> {
    ?company companies:company_id ?company_id ;
             companies:company_name ?company_name .
    }
}
```

Fig. 7. Query5

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX companies: <http://www.semanticweb.org/SER531/ontologies/Team-7/Companies#>
PREFIX jobs: <http://www.semanticweb.org/SER531/ontologies/Team-7/Jobs#>
PREFIX locations: <http://www.semanticweb.org/SER531/ontologies/Team-7/Locations#>

SELECT ?company_name ?ceo_name ?employees ?c {
    SERVICE <http://ec2-3-129-207-101.us-east-2.compute.amazonaws.com:3030/Companies> {
    SELECT ?company_id ?company_name ?ceo_name ?employees
    WHERE {
            ?company companies:company_id ?company_id ;
                     companies:company_name ?company_name ;
                     companies:ceo_name ?ceo_name ;
                     companies:number_of_employees ?employees .
            FILTER (regex(?company_name, 'ka', 'i'))
        }
    }
    SERVICE <http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com:3030/Jobs> {
    SELECT ?company_id (count(?job) as ?c)
    WHERE {
            ?job jobs:posted_by ?company_id.
       }
    GROUP BY(?company_id)
    }
}
```

Fig. 8. Query6

G. Query to get all the cities in the state

We have written a query to show the user jobs nearby, if the user has entered any city in the location filter. Here we find out the state the city belongs to and we get a list of all the cities in that state. This query is shown in Fig. 8.

H. Query to visualize count based on job title, industry and job type

We display the number of jobs available with the job title, industry and job type if these fields are entered by the user. This query performs aggregate operation and is shown in Fig. 10.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX locations: <http://www.semanticweb.org/SER531/ontologies/Team-7/Locations#>

SELECT ?cities {
SERVICE <http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com:3030/City> {
    SELECT ?state
    WHERE {
        ?state locations:has_city ?city .
      FILTER(regex(xsd:string(?city), 'san jose', 'i'))
    }
  }
SERVICE <http://ec2-3-134-101-50.us-east-2.compute.amazonaws.com:3030/City> {
    SELECT ?state2 ?cities
    WHERE {
        ?state2 locations:has_city ?cities .
     }
  }
  FILTER(?state2 = ?state)
}
```

Fig. 9. Query7

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX jobs: <http://www.semanticweb.org/SER531/ontologies/Team-7/Jobs#>

SELECT ?type ?c {
SERVICE <http://ec2-18-223-22-133.us-east-2.compute.amazonaws.com:3030/Jobs> {
    SELECT ?type (count(?job) as ?c)
    WHERE {
        ?job jobs:type ?type.
        FILTER (?type = 'Part-Time')
    }
    GROUP BY(?type)
    }
}
```

Fig. 10. Query8

X. Client side application

A simple web page was created with fields for company name, city, title, industry and job type using which user can filter the data as shown in Fig. 11. This application runs on local server on node.js. Based on the parameters entered by the user, we call APIs from the backend. The main part of the page displays the job related data. Here, the user can click on ”Apply Now” link to be taken to job application page (We have dummy links in the project. But real links can be added if available). We have visualization part on the left of the page where user can see company related data, cities around him (in the state where he is applying), number of jobs with the job title, industry and job type. Queries related to these can be seen in the ”SPARQL Queries Implemented” section.

XI. Evaluating the application and Challenges faced

We ran multiple queries on the application and analyzed the result that we obtained in the front end application. Since we had access to the data which is in the form of csv files, we were able to filter and obtain related in Microsoft Excel. The filtered rows in Excel match the data obtained in our application. We managed to check these for the visualization queries as well. One thing we noticed was that, as we keep adding more data, it takes longer for the queries to execute. Initially we tested with 2000 rows of data, and the federated query ran within 5 seconds. When we tested with 10000 rows of data, query

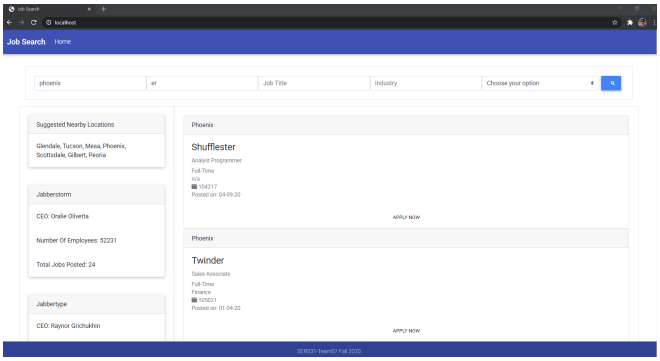

Fig. 11. User Interface

execution time took a significant hit and took as much as 30 seconds to process. Even though query execution time took a hit, the data obtained was accurate.

The main challenge we faced was to create three different datasets from the data that we had obtained from sources. As mentioned in the sections above, we managed to create different datasets. Other challenge that we faced was query evaluation time when we had a lot of data. We overcame this by limiting the query to fetch job to 1000 when no filters are applied by the user. This significantly improved the performance.

XII. Future work

The main objective of this project was to use the already generated data by job portals and consolidate them in one place to make it easy for users to search for jobs. This application can be extended by adding web crawler which checks all the job portals for new jobs when they are added. These new jobs can then be added to our system dataset. Also, we can keep checking the job links present in the system and we can keep removing expired links and remove jobs that are no longer available. If these features are implemented, our application will always have the latest jobs available.

One of extension to this application can be a recommendation system. We can ask users to upload their resume and find out what job they are looking for and give recommendation with out them entering any filters. We can also keep a history of applications and predict what the user is interested in using machine learning.

XIII. Conclusion

Our job search system was build using all the technologies involved in semantic web which offers accurate results and is reliable. Since we are using metadata details to query, it is better than regular keyword search systems as it is more accurate and provides relevant results. Along with just the job details, our application also provides information of the company, nearby cities where jobs are available and number of jobs available.

REFERENCES

[1] Selvam, Sheba, Ramadoss Balakrishnan, and Balasundaram Ramakrishnan. "Social Event Detection–a systematic approach using Ontology and Linked Open Data with Significance to Semantic Links." INTERNATIONAL ARAB JOURNAL OF INFORMATION TECHNOLOGY 15, no. 4 (2018): 729-738.

[2] Hyvönen, Eero, Erkki Heino, Petri Leskinen, Esko Ikkala, Mikko Koho, Minna Tamper, Jouni Tuominen, and Eetu Mäkelä. "Publishing second world war history as linked data events on the semantic web." In Proceedings of the Digital Humanities Conference, pp. 571-3. 2016.

[3] Knuth, Magnus, Jens Lehmann, Dimitris Kontokostas, Thomas Steiner, and Harald Sack. "The DBpedia Events Dataset." In International Semantic Web Conference (Posters Demos). 2015.

[4] A. Nurwidyantoro and E. Winarko, "Event detection in social media: A survey," International Conference on ICT for Smart Society, Jakarta, 2013, pp. 1-5, doi: 10.1109/ICTSS.2013.6588106.

[5] Pandey, Yogesh, and Srividya K. Bansal. "A Semantic Safety Check System for Emergency Management." Open Journal of Semantic Web (OJSW) 4.1 (2017): 35-50.

[6] Allan J., Papka R., and Lavrenko V., "On-line New Event Detection and Tracking," in Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Melbourne, pp. 37-45, 1998.

[7] Firan C., Georgescu M., Nejdl W., and Paiu R.,"Bringing Order to Your Photos:Event-Driven Classification of Flickr Images based on Social Knowledge," in Proceedings of the 19th ACM International Conference on Information and Knowledge Management, Hanover, pp. 189-198, 2010.

[8] de Boer, V., van Doornik, J., Buitinck, L., Marx, M., Veken, T.: Linking the kingdom: enriched access to a historiographical text. In: Proc. of the 7th International Conference on Knowledge Capture (KCAP 2013). pp. 17–24. Association of Computing Machinery, New York (2013)

[9] M. Georgescu, N. Kanhabua, D. Krause, W. Nejdl, and S. Siersdorfer. Extracting event-related information from article updates in wikipedia. In Proceedings of the 35th European Conference on Advances in Information Retrieval, ECIR'13, pages 254–266, Berlin, Heidelberg, 2013. Springer-Verlag.

[10] Berners-Lee T, Hendler J, Lassila O. "The semantic web". Scientific American, 2001, 284(5): 28–37.

[11] Heiner Stuckenschmidt, Frank van Harmelen. Information Sharing on the Semantic Web.

[12] Natalya F. Noy, Deborah L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology.

[13] W3C Consortium Consortium. http://www.w3.org.

[14] RDF Primer. http://www.w3.org/TR/rdf rdf-primer, 2004.

[15] OWL Web Ontology Language Reference, 31 Mar. 2003, World Wide Web Consortium. www.w3.org/TR/owl owl-ref

[16] SPARQL Query Language for RDF.W3C Working Draft 4 October 2006. www.w3.org/TR/rdf rdf-sparqlsparql-query/.

[17] Reynolds, Dave. "Jena 2 inference support." https://jena.apache.org/documentation/inference/.

[18] Apache-Fuseki. https://jena.apache.org/documentation/fuseki2/

[19] Vindula Jayawardana, Ontology Generation and Visualization with Protégé. https://medium.com/@vindulajayawardana/ontology-generation-and-visualization-with-prot%C3%A9g%C3%A9-6df0af9955e0

[20] Josef Hardi, cellfie-plugin, https://github.com/protegeproject/cellfie-plugin/wiki/Grocery-Tutorial