Link the job applicants and recruiters through linked data

*

Chandan Javaregowda
CIDSE
Arizona State University
Tempe, Arizona, US
cjavareg@asu.edu

Nachiappan Lakshmanan
CIDSE
Arizona State University
Tempe, Arizona, US
nlakshm3@asu.edu

Mayank Kataruka
CIDSE
Arizona State University
Tempe, Arizona, US
mkataruk@asu.edu

Harsh Kavdikar
CIDSE
Arizona State University
Tempe, Arizona, US
hkavdika@asu.edu

***Abstract*—The current job market is experiencing an exponential growth in online recruitment and traditional hiring methods are becoming obsolete. The problem with online recruitment is that a particular company can receive resumes in various formats and the data is unstructured, and it is very difficult to extract meaningful information from it. Also, the recommendation systems which are present right now are keyword based, thus ignoring the semantic aspect of the data at hand. Through various data sources available, like skills of candidates, location data, hiring trends of a company, we can create a structured web of data by linking all of them to provide accurate recommendations both for a job applicant and a recruiter.**

I. Introduction

Due to the exponential growth of the use of the internet for various kinds of applications, even the job marketplace has moved on from traditional hiring to an online recruitment model. Due to this, every company receives an enormous number of job applications everyday throughout the world. Since these applications come in different styles and structures, finding a relevant candidate might not be an easy task. Through the semantic web, we can make the best use of disparate data sets available like candidates' resumes, job descriptions, skills,locations, school and university data, and previous trends and make meaningful sense of all the linked data.

We will collect the above mentioned data sources and create ontologies out of these data-sets to create a linked web of data. From this we can retrieve accurate and streamlined search for an applicant or recruiter. For example, with the resume data and job description data, we can achieve almost accurate match of a candidate based on that individual's skill-set, relevant experience, domain knowledge, and location preferences of the candidate. With location and other data sources, we can visualize the hiring trends of a particular company in that particular location. For example, if Google has recently opened a new research centre in Phoenix, the hiring trend at Phoenix can be visualized and represented in the form of a graph. This data driven application will be helpful for broad range of audience, like a candidate can look for a perfect the job, and a recruiter can find the perfect candidate. Apart from this, we will visualize the job scenario at a particular location in the world.

Our team includes Chandan Javaregowda, Mayank Kataruka, Nachiappan Lakshmanan and Harsh Kavdikar. Chandan will look for skillset data which will have all the current trending skillset each company is looking for. Company-related datasets will be collected by Mayank, where the data should provide information about location, and company's previous years hiring and job openings related data. Harsh will be handling the data cleansing part for removing unnecessary details and information in the data-sets. Nachiappan will be converting raw data to RDF data. After the data collection, cleansing and formatting is done, as a team we will create ontologies to link the given data sets and create multiple SPARQL queries to retrieve meaningful results.

II. Project Flow

The project will consist of multiple stages as follows. First step will be data collection and pre-processing which involves collecting various datasets relating to resume data, job description data and location data. Second step of the project involves ontology creation and linking data where different ontologies are created for each dataset which describe the relationship between datasets, then merge them together to form a single ontologies. The next step involves formatting the data into n-triple format by applying ontology rules. The fourth step will be write SPARQL queries to query the n-triple datasets and obtain meaningful result. The last step would be create an application to represent this data.

III. Data collection and Pre-processing

For the purpose of this project, we used multiple open data sources available online. For the job description data, we utilized kaggle datasource. For finding information regarding candidates, we used collection of different data from kaggle and github.We obtained location information like latitude, longitude of major cities from simplemaps.com. To cleanse the data to retrieve pertinent fields and to discard trivial data, we developed a python script which iterate over all the resumes and extracts the relevant data. By using the common fields from Resume data and Job description data, we will create ontologies providing meaning and semantics by linking both the datasets. For example, we can link the current skills data from a resume dataset to the required skills of a job description dataset. Another example would be to link the number of job openings at a particular location and semantically linking the best fit candidate for those job openings, which will involve linking of Jobs, Resume and Location data.

IV. Ontology Creation and Linking Data

After completing the data cleansing, the data is now present in csv format. This csv data must be converted into n-triple format data to import them as instances in the ontologies. For this purpose, Google's open source tool OpenRefine with RDF extension was used. OpenRefine, formerly called Google Refine and before that Freebase Gridworks, is a standalone open source desktop application for data cleanup and transformation to other formats, the activity known as data wrangling. It is similar to spreadsheet applications (and can work with spreadsheet file formats); however, it behaves more like a database[8].

It operates on rows of data which have cells under columns, which is very similar to relational database tables. An OpenRefine project consists of one table. The user can filter the rows to display using facets that define filtering criteria (for example, showing rows where a given column is not empty). Unlike spreadsheets, most operations in OpenRefine are done on all visible rows: transformation of all cells in all rows under one column, creation of a new column based on existing column data, etc. All actions that were done on a dataset are stored in a project and can be replayed on another dataset. Openrefine tool can be accessed from URL http://openrefine.org/. The RDF extension required to convert tabular csv data to n-triple format can be downloaded from the link https://github.com/stkenny/grefine-rdf-extension/blob/orefine/README.md . In order to convert csv file format to n-triple format, the csv file is uploaded to openrefine tool and the rules are imported from the Ontologies. There are various in built ontologies that can be used or they can be uploaded from the system.

To create the ontologies Protégé is used, which is a free, open source ontology editor and a knowledge management system. Protégé provides a graphic user interface to define ontologies. It also includes deductive classifiers to validate that models are consistent and to infer new information based on the analysis of an ontology. Like Eclipse, Protégé is a framework for which various other projects suggest plugins. This application is written in Java and heavily uses Swing to create the user interface. Protégé recently has over 300,000 registered users. According to a 2009 book it is "the leading ontological engineering tool"[7].

This application consists of three Ontologies JobDescription.owl, Location.owl and Person.owl for JobDescription, Location and Resume data set respectively. The Person ontology contains three classes Applicant, ApplicantLocation and SkillSet. The Location ontology contains only a class i.e. Location and the instances of this class are defined in the same Ontology. JobDescription ontology contains five classes City, Company, Department, DesiredSkills and Jobs. The Department has two sub classes ComputerScience and Others which are disjoint with each other.

The next step is to merge these Ontologies to into single Ontology. In order to merge these Ontologies they have to be imported in Protege and then merged to form a new Ontology. The instances of Classes which are common in different Ontologies should only appear once and in order to achieve this their IRI's must be same across all the Ontologies.That's why IRI's of the instances of Location, City, ApplicantLocation classes are same and the IRI's of SkillSet and DesiredSkills are same.

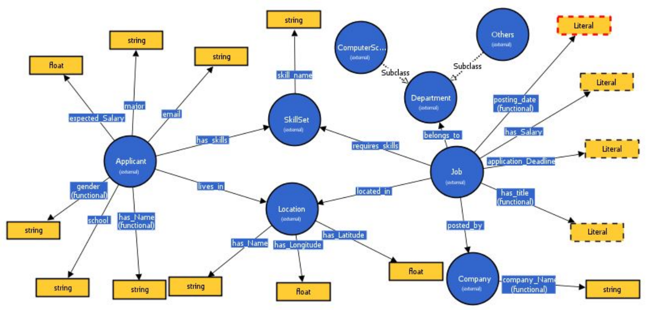


Fig. 1. Ontology

V. Semantic Querying

Once we created all the n-triple format files for all three of our datasets, we hosted them on three individual cloud instances(AWS EC2 servers). We used fuseki server to host the dataset and expose the data on port number 3030 of each server. An example endpoint of the data looks like http://ec2-54-172-125-252.compute-1.amazonaws.com:3030. We wrote multiple SPARQL queries to fetch the data needed according to the business logic. Below are few of the important SPARQL queries used along with the explanation of each.

1. Query to fetch all the Jobs related data

This query fetches all the details related to Jobs like Job Title, application deadline, required skills to apply to the job, required specialization and graduate level, salary , location , posted by which company, posting date among other details. This query also incorporates multiple filters, for example to fetch jobs from particular location you can apply location filter from the UI, the backend Java Jena code dynamically constructs a filter to our main SPARQL Query and queries the results based on the filter applied. This filter can be applied to any field from the front-end and the Jena backend takes care of applying the filter and fetching the results based on whatever filter is applied to whichever field of the dataset.

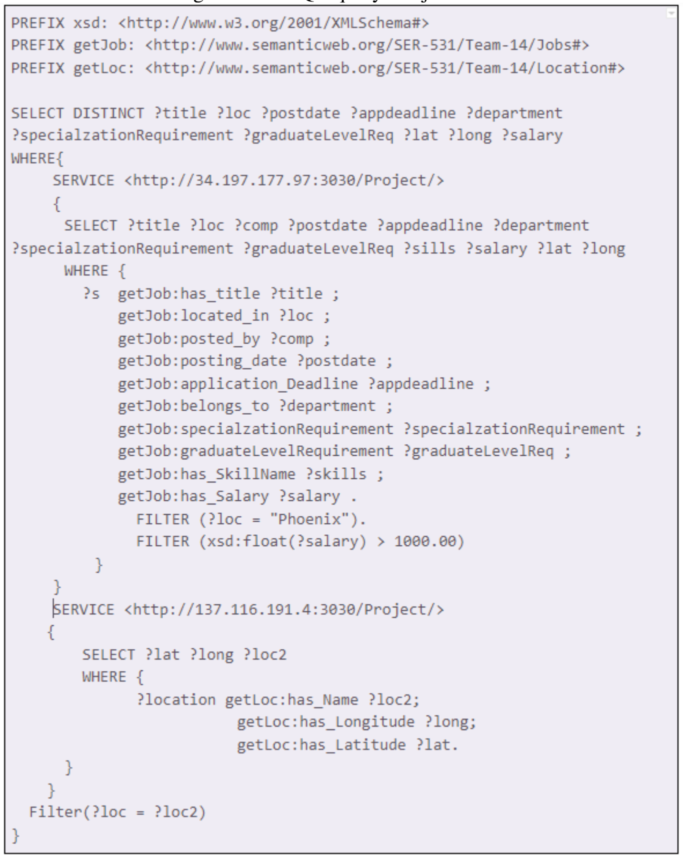


Fig. 2. SPARQL query for job data

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX getJob: <http://www.semanticweb.org/SER-531/Team-14/Jobs#>
PREFIX getLoc: <http://www.semanticweb.org/SER-531/Team-14/Location#>

SELECT DISTINCT ?title ?loc ?postdate ?appdeadline ?department
?specialzationRequirement ?graduateLevelReq ?lat ?long ?salary
WHERE{
    SERVICE <http://34.197.177.97:3030/Project/>
    {
     SELECT ?title ?loc ?comp ?postdate ?appdeadline ?department
?specialzationRequirement ?graduateLevelReq ?sills ?salary ?lat ?long
     WHERE {
       ?s  getJob:has_title ?title ;
           getJob:located_in ?loc ;
           getJob:posted_by ?comp ;
           getJob:posting_date ?postdate ;
           getJob:application_Deadline ?appdeadline ;
           getJob:belongs_to ?department ;
           getJob:specialzationRequirement ?specialzationRequirement ;
           getJob:graduateLevelRequirement ?graduateLevelReq ;
           getJob:has_SkillName ?skills ;
           getJob:has_Salary ?salary .
             FILTER (?loc = "Phoenix").
             FILTER (xsd:float(?salary) > 1000.00)
        }
    }
    SERVICE <http://137.116.191.4:3030/Project/>
    {
       SELECT ?lat ?long ?loc2
       WHERE {
             ?location getLoc:has_Name ?loc2;
                        getLoc:has_Longitude ?long;
                        getLoc:has_Latitude ?lat.
       }
    }
  Filter(?loc = ?loc2)
}
```

2. Query to get count of Job in a particular location
This query gets us the count of Jobs available in a particular location along with latitude and longitude of that location which is used to visualize the number of applicants according to location.

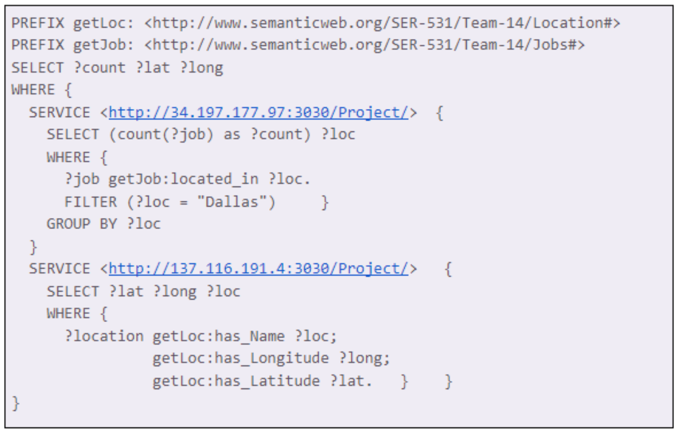


Fig. 3. SPARQL federated query for Job and Location data

```
PREFIX getLoc: <http://www.semanticweb.org/SER-531/Team-14/Location#>
PREFIX getJob: <http://www.semanticweb.org/SER-531/Team-14/Jobs#>
SELECT ?count ?lat ?long
WHERE {
  SERVICE <http://34.197.177.97:3030/Project/>  {
    SELECT (count(?job) as ?count) ?loc
    WHERE {
      ?job getJob:located_in ?loc.
      FILTER (?loc = "Dallas")     }
    GROUP BY ?loc
  }
  SERVICE <http://137.116.191.4:3030/Project/>   {
    SELECT ?lat ?long ?loc
    WHERE {
      ?location getLoc:has_Name ?loc;
                getLoc:has_Longitude ?long;
                getLoc:has_Latitude ?lat.   }    }
}
```

3. Query to fetch Applicant related data
This query fetches all the details related to Applicants like Name, emailId, expected graduation date, expected salary, skills, graduation level, University, specialization among other details. We can fire this query to filter applicants related data based on any field mentioned above. For example, if we want applicant details pertaining to one university or a particular location, we can select the filter from UI and the SPARQL query retrieves the fields matching the filters.

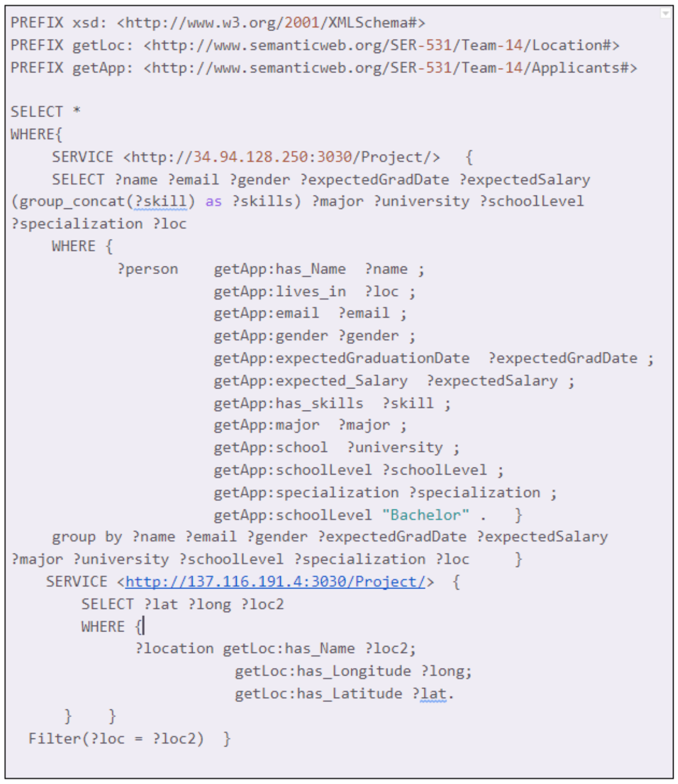


Fig. 4. SPARQL query for applicant data

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX getLoc: <http://www.semanticweb.org/SER-531/Team-14/Location#>
PREFIX getApp: <http://www.semanticweb.org/SER-531/Team-14/Applicants#>

SELECT *
WHERE{
    SERVICE <http://34.94.128.250:3030/Project/>   {
    SELECT ?name ?email ?gender ?expectedGradDate ?expectedSalary
(group_concat(?skill) as ?skills) ?major ?university ?schoolLevel
?specialization ?loc
    WHERE {
          ?person    getApp:has_Name   ?name ;
                     getApp:lives_in   ?loc ;
                     getApp:email  ?email ;
                     getApp:gender ?gender ;
                     getApp:expectedGraduationDate  ?expectedGradDate ;
                     getApp:expected_Salary  ?expectedSalary ;
                     getApp:has_skills  ?skill ;
                     getApp:major  ?major ;
                     getApp:school  ?university ;
                     getApp:schoolLevel ?schoolLevel ;
                     getApp:specialization ?specialization ;
                     getApp:schoolLevel "Bachelor" .   }
    group by ?name ?email ?gender ?expectedGradDate ?expectedSalary
?major ?university ?schoolLevel ?specialization ?loc     }
   SERVICE <http://137.116.191.4:3030/Project/>  {
       SELECT ?lat ?long ?loc2
       WHERE {
            ?location getLoc:has_Name ?loc2;
                      getLoc:has_Longitude ?long;
                      getLoc:has_Latitude ?lat.
     }    }
  Filter(?loc = ?loc2)  }
```

4. Query to get count of Applicants in a particular location
This query gets us the count of Applicants in a particular location along with latitude and longitude of that location which is used to visualize the number of applicants according to location.

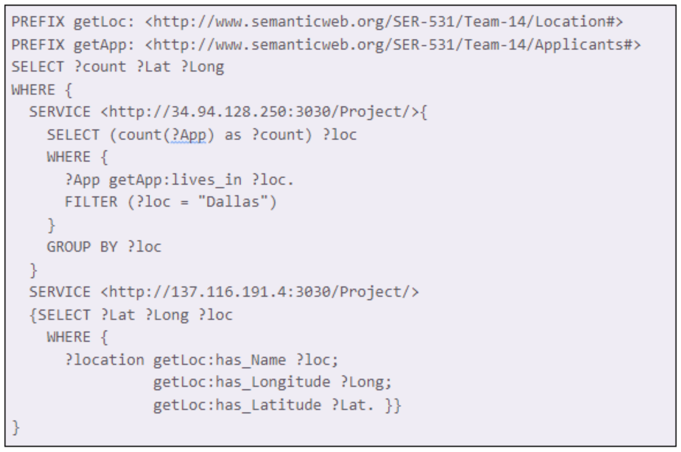


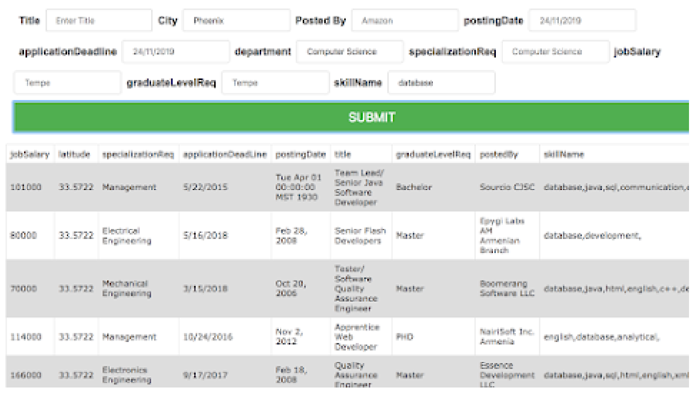
Fig. 5. SPARQL federated query for Applicant and Location data

```
PREFIX getLoc: <http://www.semanticweb.org/SER-531/Team-14/Location#>
PREFIX getApp: <http://www.semanticweb.org/SER-531/Team-14/Applicants#>
SELECT ?count ?Lat ?Long
WHERE {
  SERVICE <http://34.94.128.250:3030/Project/>{
    SELECT (count(?App) as ?count) ?loc
    WHERE {
      ?App getApp:lives_in ?loc.
      FILTER (?loc = "Dallas")
    }
    GROUP BY ?loc
  }
  SERVICE <http://137.116.191.4:3030/Project/>
  {SELECT ?Lat ?Long ?loc
    WHERE {
      ?location getLoc:has_Name ?loc;
                getLoc:has_Longitude ?Long;
                getLoc:has_Latitude ?Lat. }}
}
```

VI. Client Side Implementation

We have created tables to show all employers and applicants related data based on user input. User will be able to add as many filter they want based on any field. There are separate labels and text boxes created for each type of filters in the UI. Clicking the submit button will dynamically create sparql for any kind of user input. This will enable users to get refined output based on their preferences. This is a web based application where we have provided user with various different labels as a part of filter and submit button. We have also provided google maps at the bottom of the web page. For all the filtered query that are generated, latitude and longitude are also extracted that are marked on to the google maps. After clicking submit button, all the table data will be populated and google map will be visualized where for all the record there will be a markup pointer in the specified map location. These visualization will help both employer and applicant to get a clear view of result they are searching for.

Fig. 6. Snippet of User Interface


Title: Enter Title | City: Phoenix | Posted By: Amazon | postingDate: 24/11/2019
applicationDeadline: 24/11/2019 | department: Computer Science | specializationReq: Computer Science | jobSalary: Tempe
graduateLevelReq: Tempe | skillName: database

SUBMIT

jobSalary	latitude	specializationReq	applicationDeadLine	postingDate	title	graduateLevelReq	postedBy	skillName
101000	33.5722	Management	5/22/2015	Tue Apr 01 00:00:00 MST 1930	Team Lead/ Senior Java Software Developer	Bachelor	Sourcio CJSC	database,java,sql,communication,
80000	33.5722	Electrical Engineering	5/16/2018	Feb 28, 2008	Senior Flash Developers	Master	Epygi Labs AM Armenian Branch	database,development,
70000	33.5722	Mechanical Engineering	3/15/2018	Oct 20, 2006	Tester/ Software Quality Assurance Engineer	Master	Boomerang Software LLC	database,java,html,english,c++,de
114000	33.5722	Management	10/24/2016	Nov 2, 2012	Apprentice Web Developer	PHD	NairiSoft Inc. Armenia	english,database,analytical,
166000	33.5722	Electronics Engineering	9/17/2017	Feb 18, 2008	Quality Assurance Engineer	Master	Essence Development LLC	database,java,sql,html,english,xml

Fig. 7. Jobs and Applicants visualization on a map

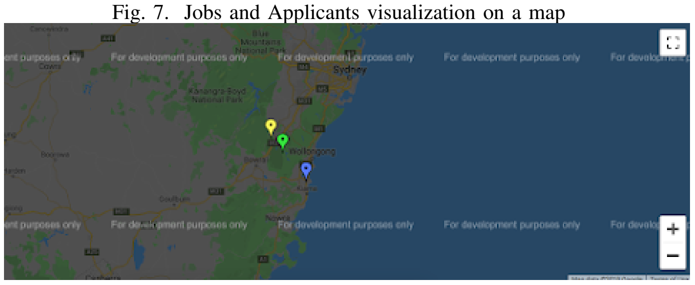
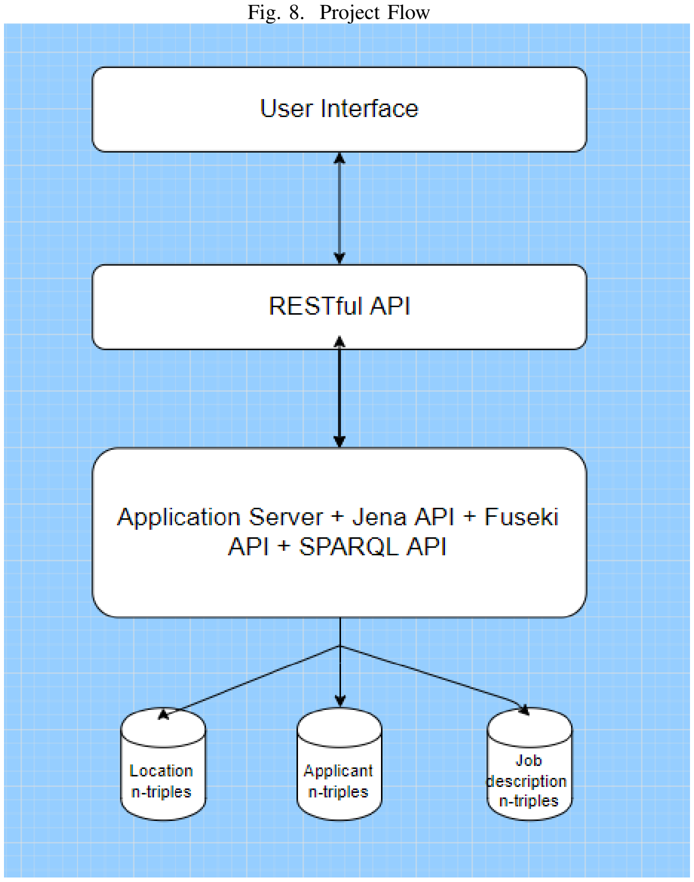


Fig. 8. Project Flow


User Interface
RESTful API
Application Server + Jena API + Fuseki API + SPARQL API
Location n-triples
Applicant n-triples
Job description n-triples

VII. Future Scope of the Project and Evaluation

The future scope involves extending the project to involve hiring trend of companies by determining the skillset of people they are hiring and the requirements they have for a particular job post which would help to provide better recommendation. Similarly, we can also add datasets regarding the employment history of an employee which would again help recruiters in getting better recommendation of the people looking for employment.

VIII. Conclusion

Job recommender system application was built using semantic linked web of data, which offers accurate results for both a recruiter and an applicant seeking a job. Since our application is using a semantic web model, the data retrieval process is more accurate and relevant compared to traditional keyword based recommender applications. Data visualization aspect of the application helps in better understanding of the job market based in a location, which is of great help for both recruiters and job applicants, thus by saving a lot of time which would have been wasted filtering the correct job or candidate.

References

[1] Abeer Zaroor , Mohammed Maree , Muath Sabha, "JRC: A Job Post and Resume Classification System for Online Recruitment," 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI) https://ieeexplore.ieee.org/document/8372026/metrics

[2] Walid Shalaby , BahaaEddin AlAila , Mohammed Korayem , Layla Pournajaf, “Help me find a job: A graph-based approach for job recommendation at scale,” 2017 IEEE International Conference on Big Data (Big Data) - https://ieeexplore.ieee.org/document/8258088
[3] Emily Bagarukayo ; Ezra Mwesigwa, “’Jobs256’ Mobile app linking job seekers to job opportunities,” 2017 IST-Africa Week Conference (IST-Africa) - https://ieeexplore.ieee.org/document/8102392
[4] Jaijanyani,‘’https://github.com/JAIJANYANI/Automated-Resume-Screening-System
[5] Kaggle ; Ezra Mwesigwa,‘’https://www.kaggle.com/madhab/jobposts
[6] Simplemaps.com ; https://simplemaps.com/data/world-cities
[7] Wikipedia: https://en.wikipedia.org/wiki/Protegesoftware
[8] Wikipedia: https://en.wikipedia.org/wiki/OpenRefine