Intelligent Web Applications - Project Proposal

Using LinkedIn, Glassdoor and DBpedia to review companies

Author:
Rex Valkering

February 9, 2015

1 Introduction

LinkedIn is a social network developed for professionals and employers. It aims to match job seekers with companies that employ their skills and interests, and is internationally used to develop a professional network.[1]

Users can create an account on the site and connect with colleagues and friends, after which they are continuously showed job offers and promotions. Occasionally, recruiters will send them a message, inviting them for an interview or asking them to keep in touch. Users can also click on job offers or companies showing up in their newsfeed to gather more information.

These methods may be effective for communication between job seekers and employers, but shows only limited information to the user. The information shown is put together by the company itself, and the representation of a company is highly dependent on the people who run it and recruiters. With a semantic web application, it may be possible to give a more complete overview of information on companies LinkedIn presents to the user.

One organisation that independently gathers information about companies is Glassdoor. Glassdoor is a website that lets users write company reviews, interview experiences and salary reports, so job searchers can get a more complete and independent overview of information about the companies they're interested in.[2]

Another organisation that collects information about companies is DBpedia. The DBpedia dataset contains a large multi-domain ontology about *things*, among which over 241,000 organisations.[3] This creates the opportunity to use this data to find and detect companies in LinkedIn data, and integrate this with Glassdoor data.

We propose to create an intelligent web application that gathers information from the user's LinkedIn account and integrates it with DBpedia company information and Glassdoor data to give a more complete overview of the companies they've been contacted by.

2 Application design

The application first requires the user to log into LinkedIn, so it can acquire an API key. Then, the application crawls the user's profile and message inbox to find and detect companies and company names. The application presents these companies in a structured overview. Once the user clicks on a company, it will show a quick overview of information about the company, such as user ratings, average salary, and job openings.

2.1 Components

The application described requires a number of components for both backend and front end application functionality. The following paragraphs describe each component in detail.

2.1.1 Application front end

The application requires a website interface written in HTML, CSS and Javascript. The application will consist of two pages: one page where the user can authenticate their LinkedIn account and permit the

[1]About Us — LInkedIn: `https://www.linkedin.com/about-us?trk=hb_ft_about`
[2]About Us — Glassdoor: `http://www.glassdoor.com/about/index_input.htm`
[3]The DBpedia Data Set — DBpedia: `http://wiki.dbpedia.org/Datasets`

application to access their data, and one interactive page where the user can view all the collected and integrated data.

The user should be able to click on displayed companies to learn more about them, but also search for companies in case their interests aren't listed.

2.1.2 Access to LinkedIn and Glassdoor API's

The user's API key will be stored in a server-side session, which it can use until the session or key expires. For Glassdoor, an API partner ID and API key are required, which can be registered through the Glassdoor site.

2.1.3 API with data processing functionality

The application requires a small API for data collection and processing. The application can then query the API for information to display interactively. This API should be publicly accessible, in case other websites wish to use it as well.

The API should provide the ability to find companies of interest for a LinkedIn user. It will do this by collecting LinkedIn information about a user, such as friends, social streams and communications. It will then attempt to identify companies in that information, by cross-validating the information with LinkedIn's companies API and DBpedia's list of known institutions. It can then display this information for the user to interact with.

The API should provide the ability to display relevant information about a company. It will do this by quering DBpedia, Glassdoor and LinkedIn for information about the company and its job offers, integrating it, and then showing it to the user. The information queried should be personalised to the user (e.g. a professional in computer science ideally wouldn't see reviews of jobs in marketing, and a user in the Netherlands shouldn't see job applications in the United States).

2.2 Data integration

Once a company has been identified on LinkedIn, it should be associated with companies in the DBpedia Database and Glassdoor API. This is not a trivial task. The Glassdoor API only allows querying companies by company name, occupation and location, while companies often have their page connected to LinkedIn. We can use DBpedia to find possible permutations of the company name, the company location, and then cross-check the results by comparing location and LinkedIn ID.

The data can be integrated using RDF and RDFS, and temporarily stored or communicated using Turtle or JSON-LS. Job reviews and interview experiences will be related to fields of experience and locations, in order to find the best match for users.

2.3 Challenges

When developing web applications, programmers often encounter problems and challenges that may heavily affect the efficiency or capabilities of their application. While most of these challenges only occur on frequently used web applications, it may prove useful to consider these and their solutions beforehand.

The project proposal contains two *artificial intelligence* tasks: detecting company names in messages, and integrating data from different websites. The algorithms for these tasks may be computationally expensive, causing the user to wait for a while. One method to solve this is to provide the data in parts, continuously updating the application as long as there is new data available, so the user can start using and reading information before the processing is complete. Another method that can be used is data caching. Once a company's data has been processed, the created relations can be stored on the server to serve to users when requested once more. This has two downsides: relations can become outdated quickly, and this may not fully benefit users with different backgrounds, as provided data may be personalised.

Another challenge is making the web application optimal for mobile devices. Since the objective of this project is not to create a mobile-friendly application, this is not a priority. However, it would be wise to design the application in such a way that this may be easily achieved, by using a front end framework that supports mobile websites.

Finally, security and privacy are common issues in web applications, though not the objective of this project. Even though HTTPS is recommended, it will not be implemented for this project, as SSL certificates are not free and using a self-signed certificate is not user-friendly. In order to respect privacy, user data will not be stored, through the API's mentioned may require some user info to be communicated, such as IP address and browser info.