Documentation for markw13a.co.uk (tier 2 job finder)

Technologies used: HTML, CSS, JavaScript, Node JS (with Express), MongoDB

Introduction

The aim of this website is to make it easier for non-EU citizens in the UK to find jobs that may be able to provide them with sponsorship for a work visa. I created it in order to help my girlfriend with her job search; there are a bewildering number of graduate opportunities advertised, but information on visa sponsorship is often not readily available.

Overview

At the most basic level, the inner-workings of the website are quite simple:

(1) Read in jobs from a list of websites
(2)Compare these against a database of companies able to sponsor a visa
(3)Discard all jobs from companies not found on this database

The companies database mentioned in (2) and (3) is generated from an Excel version of the UK government's "Register of licensed sponsors". It is currently hosted as a MongoDB on https://mlab.com, but I intend to transfer this to a database running on my web-server in order to reduce loading times.

It is important to note that, as the scraping process is too slow to be done on-demand, the website is currently set to execute steps [1-3] on a daily basis.

1

The application was designed to make adding or removing websites to scrape as simple as possible. The scraping process can be initiated by making a call to "runScrape.js" (found in the resources folder). runScrape's only purpose is to pass an array of Site objects to the main engine: siteScraper.

Site is defined in the resources folder. It contains the URL of the page to be scraped, its HTML contents (once obtained) and a method that defines how job information should be extracted. This method must be defined by the user before Site can be used in runScrape/siteScraper.

Using this Site object makes the program more flexible: siteScraper does not need to be hard-coded to deal with specific websites. Any required information specific to the page, such as its DOM structure, is handled by the user-defined Site object. New websites can then be scraped without any modifications to the program's overall structure.

2, 3

Once all pertinent information has been extracted from the given job listing, we must determine if the job might be able provide sponsorship for a work visa. At first, this would appear to be a simple case of querying our companies database with the company name extracted from the job listing, but there is an obvious flaw with this: we cannot guarantee that this will exactly match the company name provided in the database. BT, for example, might be listed as any of BT, B.T, British Telecoms or bt.

Fuse.js, an npm module that allows for “fuzzy matches”, was used to solve this problem. Although the module does provide a number of false positives, it is a sufficiently accurate out-of-the-box solution.

The front-end

The home page returned to the visitor is dynamically populated with the current contents of the website’s database. It was decided that reading from a regularly updated database would be preferable to on-the-fly scraping as this alternative is excruciatingly slow. The page is generated using an EJS template.

On visiting the site, the user is presented with a table containing jobs able to sponsor work visas in the UK (as identified by our back-end). It is possible to sort the data by clicking on the appropriate table heading, or to refine the results by entering a search term.

It is unfortunate that sorting the data or entering a search term forces the server to re-download the contents of our “opportunities” collection, rather than manipulating the data already present. As data passed to an EJS template does not seem to be accessible on the front-end, there does not seem to be an easy alternative to this.

Using jQuery to download and handle the information was considered, but I felt that potentially exposing database credentials to the end user was more serious than a, fairly innocuous, inefficiency.