User Manual

Smart Job Alert Project For:

Team: Server Side Squad

Members: Michael Kurzyna, Tyler Martin, Vivian Le, Arlind Collaku

Introduction:

The smart job alert system we were developing for our client, Jobs for Humanity, provides multiple functions that can assist in matching Jobs for Humanity's customers with job openings based on their unique skills and needs. However, due to time constraints, our system is not fully developed to the point where it can be deployed to interact with end users. Keeping this in mind, we are creating this user manual with the perspective that a new developer team will take on our project as it stands and continue development until the project satisfies all requirements, similar to how the project was introduced to our team. While our ideal user manual would detail how an end user can interact with our system to obtain useful results; we are detailing how a developer would be able to interact with our system to obtain useful results that can be used to continue the system's development.

- Detail all necessary steps needed to deploy/install your system. Provide all necessary technical specifications (2 pts)

Tools Needed:

To properly run our system, you will need multiple applications and tools, as detailed below:

- Coding environment / Compiler
    - It has proved to be beneficial, especially in our case, to utilize a user-friendly coding environment, as our system includes multiple file types and must include functions from multiple python libraries. While software is not required to edit our code, it can keep the libraries needed synced and allow for seamless database communication. At the bottom line, you need a way to edit, compile, and execute the programs associated with our system.
    - Our client recommended the software *Eclipse*, as that is the software the previous developer working on our project used.
- MySQLWorkbench - Database access software
    - In order for our program to interact with Jobs for Humanity's data, you will need to install the software MySQLWorkbench and connect to Jobs for Humanity's server, using the credentials that they have provided. Of course, this implies that you have permission and would thus have acces to JFH's database.
    - This software is also useful to test code, as you can query a new table based on specific qualifiers to ensure code is working properly.
- Python - Programming Language
    - As our system runs on the python programming language, you will have to install python on your system to compile and execute the program. Python version 2.9+ has worked previously for our group.
- Tabulate - Python Package
    - Our system utilizes the tabulate python package in order to organize data into tables, so the python package must be installed for our program to work properly.
- Project files
    - Our system runs on around 10 separate files, which must all be in one python project file, so the files can communicate and our program can work properly.

Steps:

The steps to install and deploy our system are detailed below:

1. Install the Python programming language onto your system, ensure that python installs with the “pip” command, as pip will be needed to install the Tabulate python package.
2. Run the command, “pip install tabulate” to install the tabulate python package onto your system.
3. Open the project files in your programming environment, ensure all project files are together in one project, that tabulate is installed properly, and that the project has no compiling errors.
4. Install and open MySQLWorkbench, enter the credentials provided by Jobs for Humanity in order to connect to the live database.
5. From this point, the system should be working as intended and have access to the live database, allowing the user to run the program to obtain metrics and parsed, filtered data; run the program to complete the deployment.

Main Features:

As our project stands, it provides developers with a wide array of functions to manipulate Jobs for Humanity’s data in order to assign relations to users and open jobs within the system. Upon running our main program, the user (developer) will be greeted with a menu, listing the options our main program has currently implemented. These options include:

1. Filter Jobs Based on Location
    a. This will create a list of jobs based on a candidate’s listed location in the database, returning a table of jobs that match the candidate’s location. The program also takes into account remote jobs, so if a candidate has no listed current location this filter will simply return all the remote jobs that remain open in the database.
2. Filter Jobs Based on Desired Function
    a. This function will create a list of jobs based on a candidate’s listed desired function, returning a table of jobs that match the candidate’s chosen function. If the user has not listed a desired function, this function will not work.
3. Filter Jobs Based on Community/Cause
    a. This function will create a list of jobs based on a candidate’s listed community or cause, returning a table of jobs that match what was input by the candidate. If the user has not listed a community or cause, this function will not work.
4. Run All 3 Filters
    a. This function will run all three above functions at once, returning a table of jobs that match all of the collected information about a specific candidate. This function has been extremely useful when updating the above functions to ensure everything is still working properly.
5. Get Experience Year for Each Candidate
    a. This function will determine a candidate’s length of previous experience. This is done by parsing a candidate’s resume, as in a normal resume experience is listed as such: “Manager Experience 2017 - 2019.” With this information our program

was unable to determine the candidate has 2 years of experience, so this function was created to determine a candidate's length of experience from inputted resume data. This function returns a table of a candidate's length of experience along with their corresponding candidate ID.

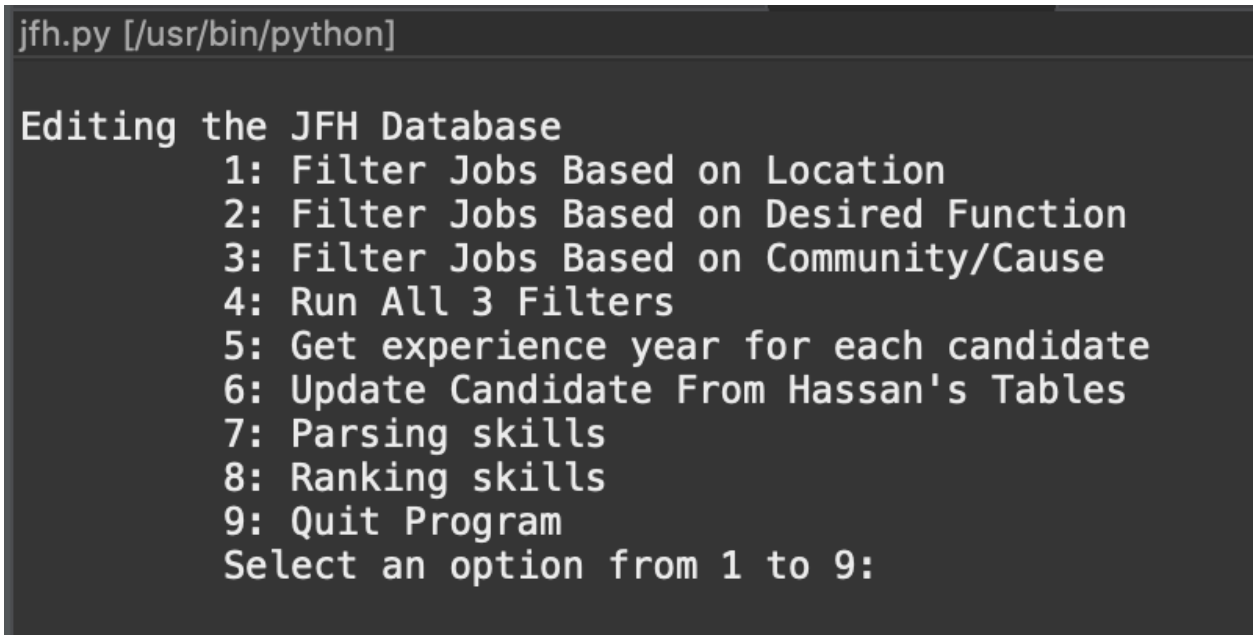
6. Update Candidate from Hassan's Tables
    a. This function simply runs an update script to keep the database as up-to-date as possible. This is done by checking candidate's creation date and checking if it is more recent than yesterday's date. If it is, the candidate is given an ID and added into Jobs for Humanity's database. This function should be ran at least once a day to keep the entire system updated and accurate.
7. Parsing Skills
    a. This function parses a candidate's skills using both our own created parser and the Smart Recruiter API parser. This function returns a table with all of the skills that the two combined parsers were able to retrieve from a candidate's resume, along with the candidate's corresponding ID.
8. Ranking Skills
    a. This function is yet to be developed and is currently used as a test method. When the program is fully developed and ready to be deployed for end users to interact with it, this function will use the data obtained from a combination of the above functions, and return a ranked list of open jobs that are determined to best match with a candidate based on the data in the database. However, as of now, this function is not ready for use.
9. Quit Program
    a. This function acts as the exit for our program, as upon running this function the program quits and will exit the developer from the menu interface.

Main Scenario Case Walkthrough:

The main scenario when using our program would be to keep the tables updated, using function 6 as described above. Our client has previously discussed intending on having this function operated on a daily basis. The steps to execute this function are detailed below:

1. Run the main project file, jfh.py
2. Once prompted with the menu, enter '6' into the console and press enter
3. The program will run the update function, returning the user to the menu upon completion
4. Once prompted with the menu, enter '9' into the console and press enter
5. The program will quit, with the database as up-to-date as possible

Example Screenshot of the Menu as detailed above:



Two Additional Scenario Walkthroughs:

As described previously, our program provides numerous functions for a developer to choose from, and some may be more relevant than others depending on what aspect of the program is currently being worked on.
One scenario we can imagine would be running all 3 filters on multiple candidates at once to obtain a table of matched jobs along with candidate's corresponding IDs. The steps to execute this function are detailed below:

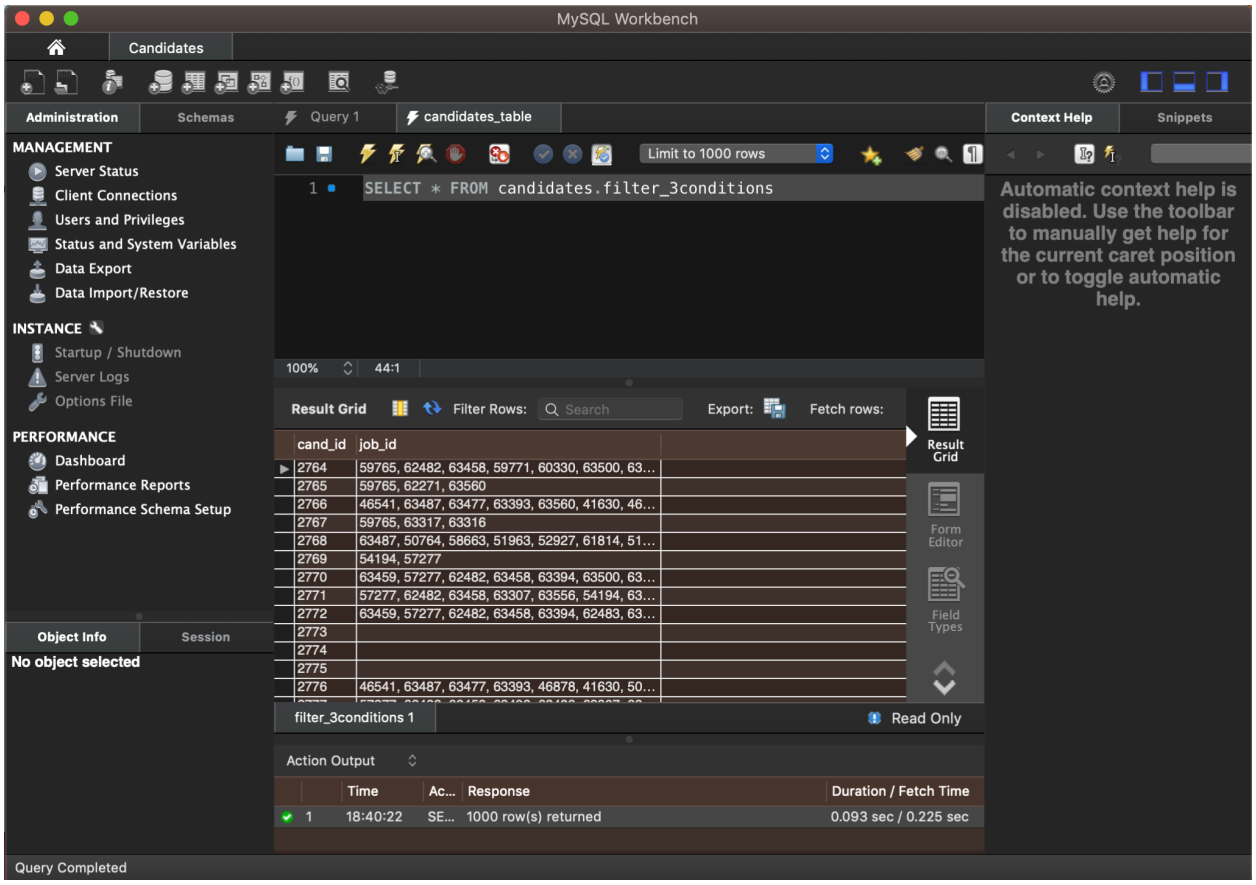
1. Run the main project file, jfh.py
2. Once prompted with the menu, enter '4' into the console and press enter
3. The program will run the 3 filters in succession, returning the user to the menu upon completion
4. Once prompted with the menu, enter '9' into the console and press enter
5. The program will quit, and the user can log into the JFH database using the credentials provided by JFH using MySQLDatabase to view the resulting table

Another scenario would be parsing a candidate's skills from their resume, using function 7 as described above. This would allow a developer to test the skill parser for accuracy and obtain a combined list of skills obtained using our developed parser and the Smart Recruiter API that has been implemented to grab skills that our parser may miss. The developer is running the function on multiple candidates at once in the detailed steps below:

1. Run the main project file, jfh.py
2. Once prompted with the menu, enter '7' into the console and press enter
3. The program will run the parsing function, returning the user to the menu upon completion
4. Once prompted with the menu, enter '9' into the console and press enter

5. The program will quit, and the user can log into the JFH database using the credentials provided by JFH using MySQLDatabase to view the resulting table

MySQLWorkbench viewing the resulting table of running function 4



What to accomplish next:

- The update job function only works with the candidate's work authorization and should be filtered further for the job's function and the communities they support. Also the resulting table needs to be updated.

- Update candidates should probably work to get the candidates skills and could score the jobs filtered for them

- Improve the scoring algorithm, currently the scoring algorithm only works with a list of jobs that match the candidate's job function and not their filtered list of jobs and does not use the skills gathered for jobs and candidates. This also could use the candidate's experience to provide a more accurate score

User stories to showcase what to further work on:

1. **As a developer I want to be able to create an update job function so that for each new job added to the database the job is added to a candidate's available job list where the new job is in the candidate's work authorization, job function, and supports their community**
    - Pre-conditions: database of jobs/users
    - Post-conditions: append jobs to candidate's list of available jobs if matched
    - User story size: 1
    - Priority level: Medium

2. **As a developer I want to be able to parse new candidate's skills and use these skills to score their available jobs so that the database does not need to be manually updated for new candidates**
    - Pre-conditions: parsing skills function, update function
    - Post-conditions: new candidates added to the scored table with their jobs scored
    - User story size: 3
    - Priority level: Medium

3. **As a developer I want to be able to create a scoring algorithm such that the skills of candidates are compared to the skills of the available job so that we can have an accurate list of a candidate's top jobs to email them**
    - Pre-conditions: Parsed skills for candidates/jobs
    - Post-conditions: Resulting scored table showing each candidate's jobs scored in descending order
    - User story size: 8
    - Priority level: Medium