# JobFinder

CS411 Group 10

by
Yusuf Mulyo, Kayce McCue, Nivedita Natarajan

# Abstract

Applying to jobs is usually a time consuming task and often people aren't motivated or forget to apply at all. With this Google Chrome extension app, one has the option to apply through the "new tab" page of the google chrome browser. So whenever the user clicks on the new tab page, they are reminded to apply, and they also have an option to search using keyword, and location options. This app aims to not only search for jobs each time you click on a new tab, but after the first search, recommended jobs will pop up for suggestions.

# Introduction

Initially, we aimed to create a chrome extension that would act as a "Common Application" for jobs. The information is imported from Linkedin or manually stored it in our database, and the app would autofill the correct boxes on a job application with the information. Since executing a basic chrome extension took a significant amount of time, we decided to switch to a chrome extension app instead. This chrome extension would pop up our customized web page on click of the new tab. The webpage has a search bar that takes in a job keyword and location. Job recommendations will also be posted based on the user's profile.

# Implementation

To implement this project, we decided to use Python Flask because that was the language all members in our group were most comfortable with. We used MongodB to store our information on the database. We used the Jooble API to access current job applications that we can call for recommendations.

## Phase 1 : Project Pitches

In the first Project Homework, as a Project Proposal, we proposed that the app would utilize the Indeed and LinkedIn API to make the API calls, but since we ran into troubles with the Indeed's API, we switched over to Jooble. The third party login authentication using OAuth would be made using LinkedIn API, and we decided to use MongoDB as the database to store user information.

GitHub link: https://github.com/ymulyo/CS411group10

## Phase 2 : User Stories

For this assignment, we came up with user stories as a means to describe the functionality of our app. In our user story we said that "As a user I want to be able to use my LinkedIn account

to import data. As a user without LinkedIn, I would still like to use this application." To address this, we came up with the option of third-party authentication using LinkedIn, and an option to sign up for an account.

GitHub folder : https://github.com/ymulyo/CS411group10/tree/master/User%20Stories

## Phase 3 : Prototype

For this phase, we came up with a sample extension to learn about the functionality of a Chrome extension. We read the documentation and read up on how to use json for different sections of manifest.json. We also learnt a little bit of javascript and made a sample extension with a logo. For the other part of the prototype, we made a sample webpage using simple HTML The webpage at this step accepted a keyword and location and makes the API call from Jooble, and shows the results to the user. We also used JSON for browser-server communication because it used attribute-value pairs and array data types which is better in readability and great for beginners as well. Since the Jooble API's website had a lot of documentation on how to make the calls with JSON, we stuck with that.

Both the webform, and the API calls were edited and integrated into the final project files.

## Phase 4 : Data

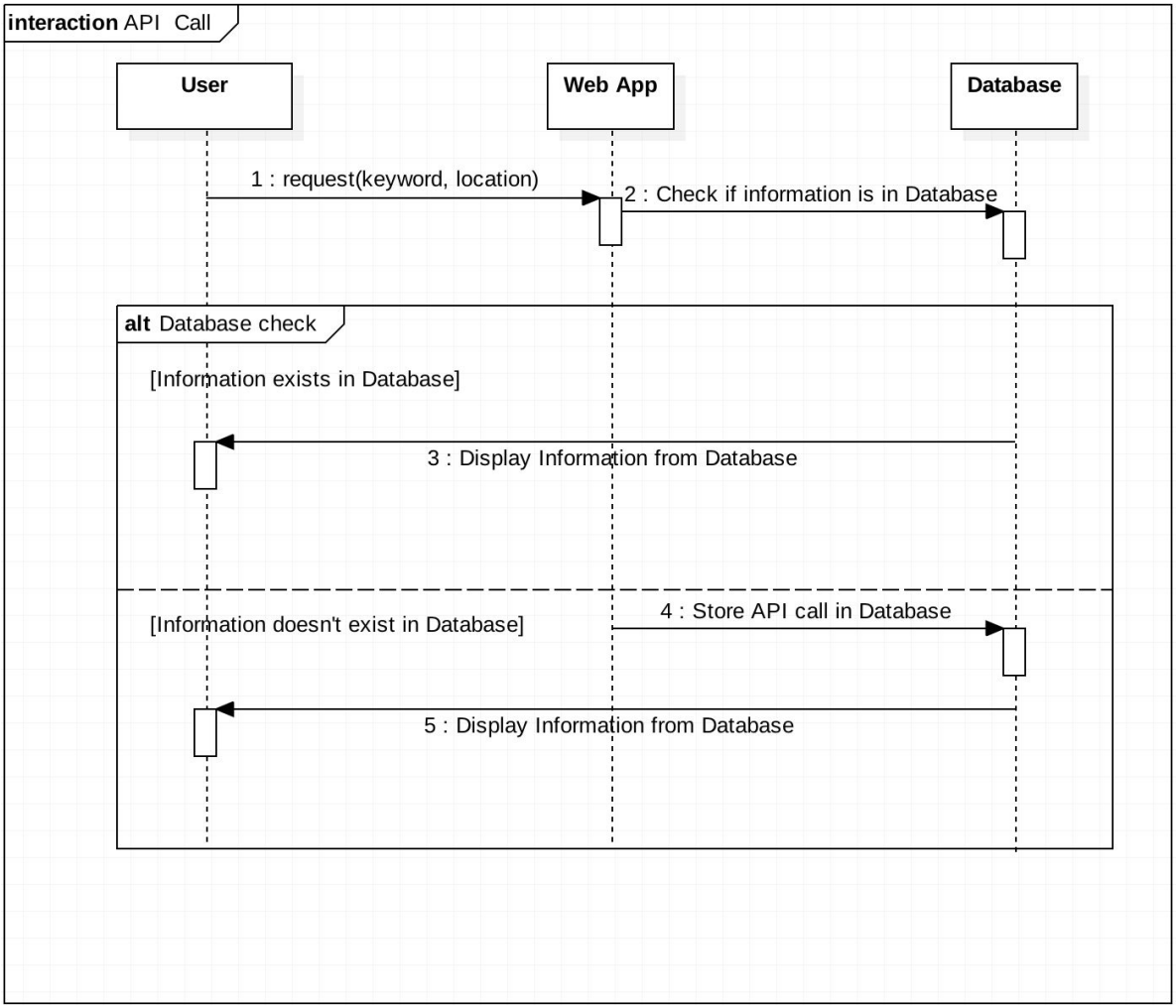
For this phase, we used a python microframework called Flask through which we linked the OAuth and LinkedIn API call to store the API keys. We decided to use Python rather than Javascript because the micro framework is supported in Python and could just be easily imported. We gave the functionality to login and logout buttons as well.

For the database storage of user information, we created a new python file to include PyMongo support that adds this information into the MongoDB Database. MongoDB is an open source document-oriented database. For new users trying to register, they will enter their username and password and they will be stored in the database. The password is encrypted using bcrypt.

To print the appropriate results, before calling the API (and wasting the limited API calls we have), it checks the database to see if the same call was made before. If it was made, then it would fetch the data directly from the database or else, it would make a new API call and save it to the database. We also connected the keyword and location calls to each user.

Code: https://github.com/ymulyo/CS411group10/tree/master/docs

The image below is the sequence diagram for the above process.




## Phase 5 : Cleaning up, integration, and final result

For this phase, all that was left for us to implement was to style the webpage using CSS, and integrate all the separate chunks of code that we had produced so far. We took a long time here because we were still figuring out the different capabilities of the extension documentation and utilities. There were too many files to deal with and too many options were ruled out. At this stage we were still trying to implement the clipboard feature of the extension, to store information entered by the user and to try to load the information into the webpage. We explored different code snippets and looked up different extensions on the chrome extension store that at least remotely tried to mimic our functionality, but even the basic clipboard feature of the existing extension, was still in BETA mode, and was not fully developed. Furthermore we didn’t have enough documentation on the chrome website, neither could we download the extension package as a set of code files, so we couldn’t see the source code either.

At this point we decided that the best way to move was to switch into looking at the functionality of the web page rather than getting stuck with the chrome extension. Not wanting to completely discard the extension idea after having spent a lot of time on it, we decided to move forward with the ‘new tab’ page idea described previously, and ditch the clipboard/ autofill functionality.

Integrating the code also proved to be a painful task, because by this time we had run out of time and noone in the team had any experience in javascript. Connecting the front end to the back end turned out to be a huge issue. The app worked on some runs, but refused to work on other runs. In other words, it became erratic.

# Evaluation

Our end goal was to make a webpage that opens either with the click of the extension icon, or with a new tab, lets us login with LinkedIn or login with an account created on the website, lets us enter a jobsearch keyword and location and get results based on the search and then the recommended list would get updated, and would start recommending results, every time the webpage was loaded by the user. As mentioned previously, integration was a difficult task at hand. Our group struggled a lot with connecting the front and back end since we didn’t have enough experience in creating web applications before - it became a steep learning curve. But still we managed to make a webpage that made the API calls and the user could successfully login with an account that they created. We were able to tick off all the goals laid out by us except one. The only goal we weren’t able to achieve was probably making the recommended jobs list, and creating hyperlinks to them. That functionality wasn’t appearing and it was getting hard to debug the issue. We also weren’t really savvy with making web pages, so the end result wasn’t the most visually pleasing, nevertheless we got all the project deliverables done on time, and were able to make a ‘web app’.

Possible improvements to this app would be to actually create the clipboard database and make it into an autofill kind of function. Even without the clipboard, with some cleaning on the front end UI, and integrating the recommended jobs successfully, the user would be able to look at their recommendations each time they click on a new tab or the icon. We could have optimized the database results by creating a timestamp to rule out previously stored data that could have been outdated. When finding the right time, the lower the threshold you set, the more likely it is that the app is up to date, but that also meant more API calls.

This app has the potential to become really elaborate and visually pleasing, and also very helpful to a user who wants to conveniently apply to jobs. If other job search website API’s are linked, this would truly be a robust extension that surely no one would forget to use thanks to the new tab popup functionality.

# Conclusion

We were able to successfully create a JobFinder Application, and make it into a chrome extension. With more fine-tuning and functionality, this app has the potential to really make a change in the user's lives. With the help of better resources and documentation, (and more time), if the above recommendations were to be implemented, this app would be ready to be uploaded to the chrome extension app store, and be utilized by people world-wide.

The extension folder can be found at:
https://github.com/ymulyo/CS411group10/tree/master/ext%202