# AUTO JOBER

# Introduction

We all know applying for jobs is mundane and a pain, but nowadays, job applications are online looking very similar to one another. Hence, why not automate it?

For this project we turn to Python for its plethora of tools, libraries, and packages, and as well as being a simple to pick-up language.

This project uses Selenium and Tkinter to scrape web pages and create GUIs respectively.

We broke the project into two parts: the front-end and the back-end. The front-end will focus on the GUI where the user will interact with it to supply sufficient details for the back-end to function properly. The back-end will focus on automating input and searching to apply for jobs.

## Front-End

The big picture (at the time) was to have input fields, such as login credentials and to have a cover-letter/resume so when a job application requested it, it would be imputed in less than a second by the back-end stuff. Additionally, wherever there was any additional input needed by the user, there would be a popup window in which the information was stored and used if it needed to be used once more.

## Back-End

The ~~big~~ enormous picture was to:

- Look for jobs
- Click to apply for jobs
- Answer questions
- Submit application
- Repeat

Now, although this seems simple in nature, Selenium (the library used to automate user input on websites) is quite vague. To be clear, Selenium has all, if not most, proper tools, but websites are constantly changing which can make it difficult for a tool/software (like ours) to quickly depreciate over time (this will be covered more in ‘Difficulties and Challenges’).

## The Link

How will they be connected to one another?
Intuitively, the front end would speak to the back-end (i.e calling function from the back-end). However, it works the other way around. The back-end awaits for the user to start the automation, the back-end awaits for the user to answer questions when it needs it, the back-end awaits for the user. That is to say, the back-end communicates to the front-end.

## Purpose and Motivation

Unfortunately the program is not sentient so we can't ask it. However, we did birth this idea based upon the impending doom that one day we will all have to wake up to the harsh reality that is trying to apply to jobs for which you have no work experience for but require entry-level jobs to have a soul crushing lifetime of work under their belt. So why not shorten that process? 😃

Sadly, this is not a new problem, especially for new graduates. But all that is about to change!

# Timeline

## Meeting 1

In this meeting we meet online (due to COVID and campus restrictions).
Using the method of Top-Down Design, we looked at the bigger picture and broke it to pieces (back-end and front-end).
Once we had established the things we needed to work on, we broke ourselves into teams.
This is when we began to establish some functions and set up Github

## Meeting 2

This time we met up in-person and had a simple UI and some automation from the backend.
During this meeting we became more familiar with Selenium and Tkinter within each respective team.

## Meeting 3

In this meeting we finally got a pretty fleshed out GUI and completed the back-end (with some kinks that need to be worked out).
This led us to connecting the two ends, and BAM! We have a working program … for the most part.

# Instructions

## Setup

1. Download Python 3
2. Download Pip (a package installer for Python)
3. Download Selenium

## Running The Application

1. Run ‘webscraper_backend.py’ (this can either be done through an IDE or command line by typing the file name)
2. A GUI window should appear in front, and in the back Firefox should be opened
3. On GUI add your credentials for LinkedIn
4. Click the ‘Start’ button
5. You should then see stuff going on in Firefox
6. When prompted by the GUI, answer any questions

## How to stop the application?

Just exit out of Firefox.

# Team Member Roles

## Steve & Abigail : Backend

These two members focused on understanding the website's (LinkedIn) structure.
Then, using Selenium to be able to automate interactions with the website.

## Gavin & Jim: Frontend

These two members focused on designing the GUI using Python's built in library, Tkinter.
Additionally, they worked on functions that the back-end would need to call.

# Difficulties and Challenges

## Back-End: Depreciation

One of the main difficulties for this project was using the same line of code to work 100% of the time. Unlike most conventional programs, our program tried to generalize automation for something that is not static. That is to say, we were trying to write a program that, although it had a finite set of problems, had problems that could be entirely different the next day. This is mainly due to updates for the website, so even the name of classes could slightly change and our code won't work anymore.

What's the fix?
There really isn't one. However, we moved to finding more structural elements of the website instead of trying to find elements by name. In other words, we tried to find containers/dividers instead of trying to find elements by their ID or class names.

## Front-End: New Windows - Multitasking

A typical GUI only has 1 window, the main window, but for this project it was imperative that a new window present itself when the back-end needed information. But why? By intuition a user becomes quickly engaged with the program if something new appears. If the main window just updated, they might not even see it. Additionally, it creates a clean interface as the user can focus on the new task at hand, answering a question, which becomes/feels separate from the main window.
The challenge that had to be solved was how to create a new window and await for the user to be done with that window to continue the program.

The solution? The use of an unconventional method. As mentioned before, there is typically 1 window for GUIs, but Tkinter has a method (that is really underused) to wait for a new window to close.
Yes, it took us about a couple hours to find this 1 line of code to solve our issue

## Improvements

Readability of our code. This is absolutely vital and something we glanced over in exchange for time. Although we did make some effort to be readable there is much needed improvement.

## Conclusion

We needed more time.