# AI for Agent Recruiting

## Final Report

Name: Kerry Agabi.

Student Number: C00249804.

Supervisor: Greg Doyle.

# Introduction

The aim of this project was to complete the development and implementation of an agent recruitment web application with the added functionality of predicting employee attrition. The application was built using ReactJS for the front end, Firebase for the back end, and Python for the machine learning model. Logistic Regression was employed as the algorithm for the attrition prediction model, and the application was integrated with a Flask server hosted on PythonAnywhere.

The goal of this report is to offer an insight and overview of the development process that assisted in the launch of the web application to production servers. This report will concentrate on the discovery of new technologies, frameworks, and all other tools that aided the completion of this project.

## Project Description:

The Recruitment Agency web application was developed from both standpoints of a job seeker and a recruiter. The main objective for this application was:

- To develop a user-friendly web application for agent recruitment
- Implement a machine learning model to predict employee attrition.
- Integrate the attrition prediction model into the web application.
- Deploy the application on a secure and scalable platform.

### Jobs Section:

This section is for non-registered users to view the available jobs in the web applications before registering an account with the application. Users can get a feel of the fluidity of the application and its various components.

### Resources:

This section is for the users to view external articles and pages to get some knowledge and insight on information surrounding the recruitment lifestyle from both the perspective of a job seeker and a recruiter.

### Dashboard:

This is a section that allows registered users to view their profile information with also the ability to update the information, which is reflected back to the database.

### Jobseeker Menu:

This is a section of the web application that allows a registered user to access the full functionality of the web application from a job seekers perspective, such as:

- Viewing Job listings that match the user's career information.
- Viewing previous and current job applications.
- Ability to update dashboard credentials.
- Ability to reset password
- Update login credentials

### Recruiter Menu:

This is a section of the web application that allows a registered user to access the full functionality of the web application from a Recruiter's perspective, such as:

- Post a job listing
- View current applications from each Job Listing.
- Ability to update dashboard credentials.
- Ability to reset password
- Update login credentials

## Technical Project Overview:

Front-end ReactJS:

The front end of the agent recruitment web application was built using ReactJS, a popular JavaScript library for building user interfaces. ReactJS enabled the development of reusable UI components, improving code maintainability and separating each important functionality. The front-end design incorporated responsive layouts, ensuring compatibility across various devices and screen sizes. I personally have a small experience using ReactJS from an intern innovation program, but this was a huge learning curve for me, in terms of learning the recent React hooks and adapting to changes in the version of ReactJS

Firebase, a Backend-as-a-Service (BaaS) platform, was employed for the web application's back-end. Firebase provided a real-time database for storing and syncing application data, as well as authentication services for secure user access management. Using Firebase streamlined the back-end development process, allowing for rapid deployment and scaling. Firebase was a new platform for me as a developer as I'm used to the more traditional Maria DB or PHPmyadmin. I had to get complacent with understanding JSON tree structure like of the real-time database and also learn how the collection, fields, and documents worked in firestore.

Python was used to develop the logistic regression model for attrition prediction. The model utilized historical employee data, including variables such as demographics, work experience, and performance metrics. The Scikit-learn library was employed for data preprocessing, feature selection, model training, and evaluation.

A Flask server, a lightweight web framework in Python, was employed to integrate the attrition prediction model into the web application. The Flask server acted as a bridge between the ReactJS front-end and the Python-based machine learning model, exposing RESTful API endpoints for communication. This integration facilitated real-time attrition risk assessment for candidates and employees during the recruitment process.

The web application was deployed on PythonAnywhere, a cloud-based hosting platform that supports Python web applications. PythonAnywhere simplified the deployment process, offering a scalable and secure environment for the Flask server and the attrition prediction model.

These Python entities were very easy to integrate and I worked very will with this programming language and web framework due to my experience in machine learning and Python from previous years. I was able to adapt my existing knowledge to cross-check the most accurate evaluation of the logistic regression model.

Other tools used in the development of this project include:
- Github - standard version control
- Visual Studio Code - code editor
- Star UML - Modelling language builder.

In summary, the front end of the application was developed using ReactJS, incorporating a responsive design and user-friendly interface for the agent recruitment process. Firebase was used for the back end, providing real-time data synchronization and user authentication services. The attrition prediction model was integrated into the application through a Flask server hosted on PythonAnywhere, allowing for seamless communication between the front end and the machine learning model.

## Mistakes:

The first well known mistake was having the functionality worked up in my head very quickly but not having a notion of how the UI design would come out at the other end.

I was overly ambitious in integrating two different frameworks into one not realizing there would be a security https issue hosting my Flask server on pythonanywhere and calling it from ReactJS.

At the start of the development, I uploaded the job seeker's information into the real-time database and job listing information in the firestore collection which brought some problems when I tried to fetch both pieces of information simultaneously.

I ignored the importance of a role system in this web application which cost me dearly near the end of the deadline because I refused to prioritise user authentication. Once the single sign-in was handled by Firebase and the user was secure and authenticated I got complacent.

## Solutions:

I migrated all the jobseeker's information to the firebase firestore database. All the data used in the web application are now stored under the same firestore entity just different collections and document fields.

I got the inspiration for the UI from tutorials and observing similar recruitment agencies' applications

I installed the CORS dependency to handle smooth integration of the different framework and modified the web header settings in pythonanywhere to ensure https connectivity is a must for the flask server.

## What I achieved:

I successfully developed a web application where users can:

- Sign up.
- Log in and log out successfully.
- Advertise a job.
- Delete a job.
- Update a job.
- View the job listings.
- Update their dashboard.
- View jobs that are outputted from the matching algorithm.
- Apply for a job.
- Predict Churn.
- View job applications based on the matching score of the matching algorithm.

## What I did not achieve:

- A competent role management system.
- An admin page.
- A CV parser.
- Use of Work experience details to convey the matching algorithm.
- User-friendly front-end application to throw back to machine learning component.

- Ericode finder API

## What I would do differently starting again:

- Implement a role system for both recruiter and job seeker straight away
- Create a Figma to visualize the end product.
- Work in more iterations.
- Have more technical meetings with my supervisor.
- Collect user feedback after each iteration.
- Deploy more Test Driven Development.
- Research the latest versions and the impact they have on dependencies in ReactJS because the developers update their versions quite so often.