Smart CV Analyser

Assignment Element 010-1

Coursework Project Proposal

MOD003484

2361648

2411228

2410684

February 21, 2025

Contents

1 Introduction 1

2 Task Allocation List 2

3 Project Requirements 3

3.1 Functional Requirements 3

3.2 Non-functional Requirements 5

4 Project Design Solution 6

4.1 Wireframe Diagram 6

4.2 Use Case Diagram 6

4.3 Navigation Maps 6

4.4 Flow Charts 6

4.5 Pseudocode 6

4.6 Class Diagram 6

4.7 Usability 6

4.8 Prototype 6

5 Evaluation 7

Reference List 8

1. Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

2. Task Allocation List

3. Project Requirements

Our client has requested us to design a software solution that simplifies and speeds up their recruitment process by automating CV review and sorting and providing a ranked list of CV summaries. Based on the Live Brief provided by TrackGenesis and the following presentation, we have decided on what we believe to be the essential requirements for this software.

3.1 Functional Requirements

The following list is the minimum and essential requirements needed for the system to function correctly:

1. **Job profile:** The program must allow the user to set up a job profile to use as a working environment for adding and keeping a job description and candidate resumes. The reason why we decided to implement profiles was to enable an easy way to save and resume work between multiple sessions, including adding resumes from new applicants to a previously generated ranking list without having to re-upload all of them. We have also made a reasonable assumption that the company might be hiring for multiple positions at the same time, and the user could benefit from having multiple saved job descriptions that they could later reuse without having to manage their storage themselves.

2. **Job description data input:** The user should be able to add a job description to a job profile either through a text field as plain text, or by uploading a PDF, DOCX, or TXT file.

3. **Job description parsing:** Because the client indicated optimisation and automation as their priorities, we have decided to use NLP to parse job descriptions to extract desired criteria for each job, instead of having the user set up their own criteria manually. This

should simplify and hasten the setup of new job profiles. Parsing of job descriptions should result in a set of criteria which are then stored for each profile.

4. **Resume data input:** The user needs to be able to upload multiple CV files in different formats (PDF, DOCX, TXT) at the same time, which are then saved on the profile and assigned an candidate ID. The resumes should be stored in case the user wants to view them, and to avoid the need of uploading them multiple times in case they need to be parsed again (e.g. if a job description is updated).

5. **Resume data parsing:** CV files for each candidate need to have text extracted before it can be parsed into data that's usable by our data matching and ranking system. The workflow of handling resumes and job descriptions needs to include these steps to cover the file formats requested by our client:

   (a) **Plain text extraction from CV files:** this can be achieved by reading TXT files natively in Java, using the *Apache PDFBox* library for PDF files, and the *Apache POI* library for DOCX files. However, the *Apache Tika* library acts as a wrapper for aforementioned libraries and can detect all 3 file types and extract text from all of them using just one function.

   (b) **Natural Language Processing:** Extracted text is then passed on for processing via the *Stanford CoreNLP* library. Ultimately, processed data has to result in *entities* (e.g. Name, Programming Languages, Company) with assigned *attributes* (e.g. John, Python, Google) that are then put into *categories* (e.g. Personal Info, Skills, Work Experience).

6. **Data matching and ranking:** Parsed data needs to be stored in individual JSON files and then compared to the job description JSON file that has the same structure to hold entities and attributes. The CV files are assigned a *relevance score*, the value for each match is determined by it being an exact or partial match.

7. **Result summary display and storage:** The program needs to be able to save data

of each job profile and resume into JSON files that are encrypted using *AES-256 encryption*, and then accessed by the program and displayed in the GUI as a list that is ranked by the relevance score.

3.2 Non-functional Requirements

These are additional requirements that do not affect the basic functionality, but alter it in ways that provide additional benefits to the User.

1. **Security:** Because TrackGenesis is a Scotland based company, and due to the nature of data being gathered and processed by the program, we have made the assumption that it should be compliant with the *United Kingdom General Data Protection Regulation (Art. 32.1.a)* (2023) and encrypt parsed data before it is stored. Therefore we decided to implement *AES-256 encryption* using *Java Cryptography Architecture (JCA)* as it provides an easily accessible built-in way to encrypt the JSON data *(Oracle, 2025)*.

2. **Usability:** The program should have an intuitive GUI that allows the user to easily set up job descriptions, upload CV files and view ranked summaries. We decided to implement a GUI is to avoid users having to familiarise and remember terminal commands, which is less intuitive and slower than a very basic GUI. This should improve the speed of resume screening further, which is one of the main stated goals of the client.

3. **Scalability and performance:** The user should be able to upload a large amount of complex CV files at the same time and not experience a major slowdown in data processing. Already processed data should be saved on the job profile and not processed repeatedly if additional CVs need to be uploaded. The data should be well organised and presented in a way that's intuitive and easy to navigate.

4. Project Design Solution

4.1 Wireframe Diagram

4.2 Use Case Diagram

4.3 Navigation Maps

4.4 Flow Charts

4.5 Pseudocode

4.6 Class Diagram

4.7 Usability

4.8 Prototype

5. Evaluation

One part of designing the project, which presented a significant challenge, was selecting the correct tools for our purposes, primarily the Java libraries required for extracting raw text from resumes and then parsing the data via NLP.

At first, we were looking into the recommendations presented to us via the Live Brief, which included some Python libraries. After doing some research on those, we were informed that despite the Live Brief, we were required to use Java exclusively. This was a slight setback for us, but it simulated a very real possibility of requirements changing mid-project and shows the importance of careful requirements gathering.

Our Client emphasised efficiency and automation as a requirement, so we needed to pick Java libraries that were fast, accurate, and able to process a large number of complex resumes. Plain text extraction from common file formats like PDF and DOCX is quite straightforward and fast, so we went with Apache Tika as recommended in the Live Brief. However, the NLP solution required more consideration. After doing some research, we have decided to use Stanford CoreNLP as recommended in the Live Brief. OpenNLP would offer easy integration with Tika (both developed by the Apache Software Foundation), however, multiple sources regard CoreNLP as the faster and more accurate libraries according to their benchmarks (Schmitt et al. 2019; Nanavati and Ghodasara 2015). Having gone through these considerations we are therefore confident in our choice of tools for the purposes of this program.

Reference List

Nanavati, Jay and Yogesh Ghodasara (2015). "A Comparative Study of Stanford NLP and Apache Open NLP in the view of POS Tagging". In: *International Journal of Soft Computing and Engineering (IJSCE)*. Accessed: 14 February 2025. URL: https://www.ijsce.org/wp-content/uploads/papers/v5i5/E2744115515.pdf.

Schmitt, Xavier, Sylvain Kubler, Jeremy Robert, Mike Papadakis, and Yves LeTraon (2019). "A Replicable Comparison Study of NER Software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate". In: *Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. Accessed: 14 February 2025. URL: https://www.researchgate.net/publication/337977695_A_Replicable_Comparison_Study_of_NER_Software_StanfordNLP_NLTK_OpenNLP_SpaCy_Gate.

United Kingdom General Data Protection Regulation (Art. 32.1.a) (2023). Accessed: 16 February 2025. URL: https://www.legislation.gov.uk/eur/2016/679?view=plain.