TTDS Group Project - Job Search Engine
Group_Id: 4

Student Numbers: s2556430, s2561987, s2567498, s2583940, s2603149, s2603217

Abstract

The Advanced Job Search Engine is a comprehensive platform designed to efficiently connect job seekers with relevant opportunities. Leveraging web crawling tools and custom scrapers, the system collects job data from various UK and EU websites on a daily basis, ensuring up-to-date and comprehensive listings. A PostgreSQL database stores the collection of about 1.3 million job postings , while preprocessing techniques prepare it for indexing. The indexing process, facilitated by Redis, enables fast and accurate search operations using advanced functionalities like boolean, phrasal, and proximity searches. Developed with FastAPI, the backend API offers seamless query processing, while user feedback and evaluation metrics attest to the system's high success rates and precision. The user-friendly single-page web application interface enhances the overall experience. This work contributes to the advancement of job search technologies, making job hunting more efficient for both job seekers and employers. The Job Search Engine can be accessed at http://34.105.159.121:8000/

1 Introduction

Creating an information retrieval system for job postings involves designing a platform that can efficiently and accurately match job seekers with relevant job opportunities. The goal is to develop a system that can parse, index, and retrieve job postings from a variety of sources, providing users with the most relevant and up-to-date information. This paper outlines the development of such a system, focusing on the challenges of dealing with diverse job data formats, the implementation of advanced search algorithms, and the integration of user feedback to improve search accuracy. Through this work, we aim to contribute to the improvement of job search technologies, making it easier for individuals to find suitable employment and for employers to connect with qualified candidates. In the subsequent sections, we describe the system design and implementation processes.

2 Data Collection and Storage

The first task in creating this search engine was to collect the data that our users would be searching through. The data collection process for our job search engine is vital to ensure that our platform provides up-to-date and comprehensive listings for job seekers across the UK and the European Union (EU). This section outlines the methodology, tools, and procedures employed in gathering data from various sources, including websites within the UK job market and the official EU jobs portal.

2.1 Data Collection Methodology

The data collection process is divided into two main components:

1. Scrapy Framework for UK and EU Sites:
    - We utilize the Scrapy framework, a powerful web crawling and scraping tool, to collect job listings from multiple websites within the UK and the EU.

- The Scrapy spiders are designed to extract relevant job information such as job id, title, company, location, date posted and job description from each target website.
- Separate spiders are developed for UK and EU sites to ensure compatibility with different website structures and data formats.
- The data collected by Scrapy spiders are stored in a PostgreSQL database for further processing and integration into our job search engine platform.

2. Custom Scraper for EU Official Jobs Website:

   - In addition to scraping job listings from various UK and EU websites, we have developed a custom scraper specifically tailored to extract data from the official EU jobs website.
   - This custom scraper is designed to navigate the structure of the EU jobs portal and retrieve job listings along with relevant details.
   - Similar to the Scrapy spiders, the data obtained from the EU official jobs website scraper is stored in the same PostgreSQL database for consolidation and utilization within our platform.
   - As this website is dynamically rendered, we found a way to directly collect the data from the backed API using session cookies. This approach makes the scraping much faster while still obeying `robots.txt`.

2.2 Execution and Schedule

- The data collection process is initiated and managed by the `main.py` function, which orchestrates the execution of both Scrapy spiders and the custom EU jobs website scraper.
- To prevent indefinite execution and ensure efficiency, a timeout of 14,400 seconds (equivalent to 4 hours) is set within the `constants.py` file. This timeout mechanism helps mitigate the risk of prolonged and unpredictable execution times.
- The data collection is scheduled on a daily basis for maintaining our system with up to date job listings, as of now the data collection is scheduled to be recurring at 12:00AM daily. We also have a flexibility to execute the data collection instantly at any point of time.

2.3 Data

Since the data collection involves entries from various websites, we needed a way to consistently represent the data. Thus, we defined our data model, shown in table 1.

Data Model	
Field	**Description**
`id`	The serial id as generated by the database
`job_id`	The job id, prefixed with the site it was collected from
`link`	The job's link
`title`	Job title
`company`	Company that posted the job
`location`	The best approximation of the job's location. Not always provided, and if so, it is assumed using context from the website.
`date_posted`	The date that the job was posted
`description`	The job listing description
`timestamp`	Automatically added field in order to auto-delete entries older than X amount of days. X is currently set to 20 days.

Table 1: Jobs Data Model

For the data storage we used a PostgreSQL instance, inside a Docker container. Although new data is constantly being added, at the time that this report is being written, the database contains just over **1.3 million** entries/jobs which is roughly equivalent to **4GB** of data.

A problem we faced is that many job posting websites do not allow data collection from 3rd parties, as specified by their `robots.txt` file, a constraint that we wanted to respect. As we result, our data sources were limited and to address this issue, we decided to collect data from both the UK and Europe sites, as previously mentioned. Many of the job postings are written in the language of their respective origin country. Thus, for our system we implemented multi-language support, providing job entries in various languages.

It is important to note that some jobs with identical titles and descriptions may appear multiple times in our system. This occurrence is not due to an error in our system but rather stems from the practice of job posting websites reposting the same job under different IDs. This duplication is a characteristic of the ground truth data source we rely on for job postings. In light of this, we made a deliberate decision not to artificially filter out these duplicate job postings in our system. By maintaining these duplicates, we ensure that our system accurately reflects the data provided by the source, providing users with a true representation of the job postings available.

3 Preprocessing

Since the language collections encompass various languages across the European Union, the initial preprocessing stage involves identifying the language of the data. This language identification task is accomplished using the LangDetect library, renowned for its robust language detection capabilities. When the LangDetect library encounters ambiguous text or text lacking clear language indicators, which is a rare occurrence but possible, it raises an exception indicating that no language could be confidently detected. In such cases, the text is marked as unknown, bypassing further processing steps specific to known languages. This precaution ensures that text with uncertain language characteristics does not undergo language-specific preprocessing steps that could lead to inaccurate results or processing errors.

The preprocessing pipeline begins with case folding, where all letters in the text are converted to lowercase. Following case folding, the text undergoes cleaning to remove special characters, HTML entities, symbols, and unknown alphanumeric characters. This cleaning process is achieved using a pre-defined regular expression pattern. The pattern captures and removes these non-alphabetic and non-numeric elements, leaving behind only meaningful tokens for further processing. After cleaning, the text is tokenized into individual tokens or words using the cleaned and standardized text.

Next, language-specific stop words are removed from the tokenized list based on the language identified earlier in the preprocessing pipeline. Stopwords, such as common articles, prepositions, and conjunctions, are filtered out using NLTK's stopwords corpus, which provides language-specific stopwords lists for various languages. Once stop words are removed, stemming is applied to the remaining tokens using the Snowball stemmer system. Snowball stemmers, available through NLTK, offer language-specific stemming algorithms that reduce words to their root forms, considering the previously detected language.The combined use of case folding, regex-based cleaning, tokenization, stop words removal, and stemming ensures that the text data is appropriately processed and prepared for indexing.

4 Indexing

The inverted index constructed in our system follows a positional indexing approach, where not only the presence of terms in documents is recorded but also their specific positions within each document. This positional information is crucial for advanced search functionalities such as phrase queries and proximity searches, where the relative positions of terms within documents are significant for determining relevance and accuracy in search results. In the inverted index structure, each term (key) maps to a nested dictionary containing document IDs (keys) and their corresponding positions (values) encoded as a string separated by commas, as it is the best option for speed and memory usage within the selected database system. For example, the term "engineer" might be associated with positions 0, 5, and 12 within "doc1," and positions 3 and 8 within "doc2." Similarly, the term "teacher" could be associated with positions 2 in "doc1" and positions 7 and 15 in "doc3." This level of granularity enables precise document retrieval based on the positional context of terms within documents.

Furthermore, the inverted index also stores the unique identifiers of documents, commonly referred to as document IDs or job IDs. These IDs are crucial for identifying and retrieving complete documents or document snippets containing specific terms. Including document IDs in the inverted index ensures efficient

document retrieval based on search queries and facilitates seamless integration with other parts of the system, such as document retrieval engines or data analytics processes. The initial index construction involves processing a large dataset efficiently. We split the dataset into manageable chunks, utilizing batch processing capabilities. This approach optimizes memory usage and allows for parallel processing of chunks, enhancing the overall indexing speed.

Moreover, this approach was updated to integrate with the continuous addition of new data. The inverted index-building process in our search system adopts an individual document indexing approach that is triggered whenever new job data is added. This live indexing process ensures timely updates to the index, aligning with adding or modifying job data. As new documents are added or existing ones are modified, the system triggers the indexing process for each document individually, promptly reflecting changes in the inverted index and supporting real-time data availability for searches.

Redis was chosen for indexing due to its speed and ability to handle a large number of index entries efficiently within memory storage constraints. Compared to MongoDB, which primarily relies on disk storage, Redis's in-memory data structure store excels in managing a high volume of index data with low latency for read and write operations. Each term is key in Redis, and its value is a hash map containing document IDs as fields and positions within the document as values. This structure supports fast lookups during search operations. Redis pipelines (pipe) are utilized to group Redis commands (such as HSET) and execute them in a single network round-trip. This reduces latency and improves overall indexing performance, especially when dealing with large volumes of data.

When jobs expire or are removed from the system, maintaining index integrity becomes crucial. To handle this scenario, we utilize document IDs to identify and remove expired or removed entries from the index. We use Redis commands such as HDEL to iteratively remove the entries associated with expired or removed documents. This approach ensures that the index remains relevant and up-to-date, optimizing search performance and managing index size efficiently.

5 Back-end

In order to connect the logic from the Information Retrieval (IR) system with the front end we created a back-end API, using the FastAPI framework, which excels in speed compared to other traditional frameworks like Flask [ND23]. Therefore this API serves as an entry point for the queries and routes the queries to the corresponding types of search, including boolean, phrasal, proximity and ranked search, depending on the query syntax. When a query is retrieved the syntax of the query is examined for special operators, mainly boolean operators (AND, OR, NOT) or "" or #(). After that, the preprocessing steps described in Section 3 are performed. The system then routes the query to the boolean search if a special operator is detected. Otherwise, the query is routed to the ranked search.

The API provides three endpoints to three main services: search, query suggestion, and feedback submission. Please refer to the Appendix C for more information on the usage of the web API.

5.1 Ranked Search

For the ranked search a modification of the TFIDF algorithm is employed, as we considered the TFIDF algorithm would achieve great success in retrieving the job postings for the introduced query. This intuition comes from the fact that many job postings have common words like graduate or experience, that can not be removed as stopwords as a user can search for them. TFIDF weights these common less informative terms less than others enhancing the relevance of the retrieved documents. However, recency is an important factor when searching for job vacancies and the traditional TFIDF does not take into account this factor. Consequently, we introduce a factor to the TFIDF score to penalize older documents. This factor is calculated using Equation 1 where *dat* is the date factor and d is the number of past days since the publication of the job. These date factors are cached and added to a dictionary once every day, matching with the live index updates. For each term in the query that appears in a document, the TF and IDF are multiplied together along with the document's date factor to calculate a score for that term-document pair, resulting in the TFIDF formula represented in Equation 2. If a document contains multiple terms from the query, the scores for each term-document pair are summed to update the document's total score.

$$dat = \frac{1}{1 + \log(1 + d/30)} \tag{1}$$

$$\text{Score}[q, d] = \sum_{t \in q \cap d} (1 + \log_{10} \text{tf}(t, d)) \times \log_{10} \left(\frac{N}{\text{df}(t)} \right) \times dat \tag{2}$$

Additionally, to improve the retrieval we leverage query expansion mechanisms. Initially, we implemented Rocchio's Algorithm, nevertheless, the nature of the job postings limited the search performance especially when a common term was introduced in the query. Thus, Rocchio's Algorithm was discarded. After considering other options, we selected a thesaurus-based query expansion as many common jobs like *consultant*, *data* and *internship* can easily be matched to synonyms improving the query results without sacrificing query performance. For this approach, we built a job posting thesaurus because other open-source thesaurus implementations were too general to improve the relevance of the retrieved documents.

5.2 Search - Boolean, Proximity, Phrase

Incorporating Boolean, phrasal, and proximity searches into a job posting information retrieval system significantly improves the ability to match users with relevant job opportunities. These advanced search techniques offer an improved approach to query formulation, allowing for precise job searches.

Boolean search uses logical operators like AND, OR, and NOT in capital letters to refine searches. This method lets users precisely define what their search should include or exclude, making it possible to narrow down or broaden search results efficiently. For example, a search could combine keywords to find postings that mention multiple specific skills or use exclusion to filter out irrelevant fields or skills the user does not have. To implement this type of search the operands are pushed into a stack and then poped when a non-special token is found, applying the corresponding set operations.

On the other hand, phrasal search matches exact phrases by treating the search query as a single unit. This approach is especially relevant for finding job postings with specific titles or qualifications. It delivers high precision but at the cost of flexibility, as it will only return postings that contain the exact sequence of words. In the created system the user can introduce a phrasal search by surrounding the search with "".

Proximity search builds on the idea of phrasal search but adds the concept of term proximity. It allows users to specify how close the search terms should be within the text, offering a middle ground between the rigidity of phrasal searches and the broadness of separate keyword searches or ranked searches. This is particularly useful for finding postings where certain qualifications or skills are mentioned in close relation to each other, enhancing the contextual relevance of the search results and providing the user with more flexibility. The syntax of this type of query is a # followed by the proximity and then the terms surrounded by ().

The implementation of phrasal and proximity is very similar, both of them search for the intersection of docs containing the terms and then iterate over the postings in search of a combination that matches the maximum distance, which is 1 for phrasal and the proximity for the proximity search.

These search approaches, especially phrasal and proximity searches, demand more computational resources because they require a cross-product of common document postings to determine the proximity or exact match of terms within the documents. This process can be more time-consuming than simple ranked searches because it involves additional computations to assess the distance between terms or to confirm the presence of an exact phrase.

Furthermore, boolean, phrasal, and proximity searches can be combined in various ways to tailor the search process even more closely to the user's needs. Combining these methods allows for sophisticated query formulation, enabling users to pinpoint job postings that meet very specific criteria. However, this flexibility and precision come with increased computational demands, as the system must perform more complex analyses of the text data to accommodate the nuanced requirements of these search queries.

By offering these advanced search capabilities, a job posting information retrieval system can greatly improve the relevance and precision of search results, helping users find job opportunities that closely match their skills and experiences. However, the enhanced functionality also implies higher computational costs, reflecting the balance between search efficiency and the complexity of processing advanced search queries.

5.3 Query Suggestion and Spell Checker

Aiming to provide the user with the best user experience and make the job search easier and more accessible we have developed a spell-checking service for queries in English and a query suggestion mechanism.

The spell-checker detects the written language for the query and then suggests potential improvements. We implemented a spell-checking mechanism using three libraries: SpellChecker, Speller, and TextBlob, and integrated weighted preferences into our approach. Leveraging the strengths of each library allowed us to significantly improve the accuracy of spell-checking. For instance, corrections suggested by multiple libraries carry more weight and are more likely to be chosen, ensuring a higher level of accuracy. However, in cases where there is no consensus among libraries regarding a correction, our code intelligently utilizes the assigned weights to make informed decisions.By assigning weights to corrections from these libraries, we achieved a balanced and effective strategy for handling spelling errors in textual data. This adaptable approach proved invaluable in addressing a wide range of spelling variations and errors commonly encountered in natural language text.

In addition to query expansion, we explored query suggestion strategies to recommend relevant queries to users. Initially, we experimented with a model based on the BERT architecture. However, the output from this model was found to be overly generic and lacked significant utility for end users, as it did not provide specific and actionable suggestions. To improve the query suggestion experience, we adopted a different approach by retrieving job titles existing in the database that start with the characters introduced by the user. This strategy yielded much more useful and targeted results for query suggestions, enhancing the overall user experience without introducing significant delays between typing and receiving query suggestions. This approach strikes a balance between relevancy and speed, offering meaningful suggestions to users as they interact with the system.

6 Front-end

A single-page web application was built to provide access to our search system. The technologies used for this application are Vue.js for page development, TailwindCSS for styling, and Vite for serving. These technologies were chosen because of their ease of developing and hosting.

The web application provides a user-friendly interface inspired by popular web search engines such as Bing or Google. It was decided to follow this design to provide users with a familiar interface. This will reduce the friction that new users encounter with new services.

The web application consists of three views: welcome, results and feedback. The welcome page provides a single text box where the user inputs the query. When the user begins to write a query, a list of query suggestions retrieved from the API is shown. Then, the results page is displayed when the user submits the query by hitting the enter key. The results page shows again the query input bar and a list of clickable results. Every request displays the first ten results. An infinite scroll feature was implemented to allow users to load more results by simply scrolling down. This was achieved using the back-end provided pagination system. Navigation links are located at the top of the results view for users to go to the home view, download the currently loaded results in a CSV file, or go to the feedback page. The download feature allows users to save the current set of results locally in a CSV file format. This feature is akin to the "Save Jobs to Account" functionality available in search engines like LinkedIn, where users can save or download job listings for future reference or offline viewing. The feedback page contains a simple form that allows the user to rate the service and provide written feedback.

The application is resilient to API errors and provides users with feedback on known and unknown errors. The known errors include HTTP-code 404 which happens when no results are found, and HTTP-code 400 which occurs when a Boolean query is invalid.

7 Integration

The integration of the inner components of the search engine is a complex task. Achieving fast and secure communication between the components assures a better performance, thus improving the experience for the final user. Special attention was put to the system architecture and its deployment to achieve the correct

integration of its components.

7.1 Architecture

The system comprises five main components: a relational database, a non-relational database, one web crawler, one HTTP API, and one single-page application. The components are divided by role in three architecture layers: persistence, computing, and user interface.

The persistence layer consists of two database components: a relational PostgreSQL database and a non-relational Redis database. This layer is responsible for storing and persisting the collected information by the web crawler. The relational database stores the job information, scrapped by the crawler, and the non-relational database stores the search index information. Redis was chosen as the index storage because of faster database transactions by keeping the most frequent queried terms in memory.

The computing layer comprises two Python applications: the web crawler and the HTTP API. This layer oversees doing all the processing required to crawl or retrieve information.

The single-page web application and the HTTP API build the user interface layer. This layer provides a public HTTP API and Graphic User Interface (GUI). These two interfaces enable access to the search engine.

7.2 Deployment

Docker containers [Mer14] were used to deploy the system. The CloudLab [DRM+19] experimental platform was used during the development on which the system's components were deployed. They were deployed on separate nodes to convert them into serverless functions in the Google Cloud Platform (GCP). The system showed very high latency when this approach was implemented on GCP. This happened due to the volume of data being transferred over the network.

Thus, it was decided to use Docker Compose to host all the services on an individual GCP Virtual Machine, which solved the latency problem. After confirming that this was the best course of action, the services and the stored data were migrated To a GCP Virtual Machine (provisioned with plenty of resources). The deployment of the search engine was completed once the services were started using docker-compose, and the public IP of the virtual machine, along with the application ports, were exposed to the internet.

8 Evaluation

The evaluation of our job search retrieval system encompasses both user feedback utilization and theoretical metrics application to comprehensively assess system performance. On the front end, we have integrated a user feedback feature allowing users to provide direct feedback, enabling us to gather valuable insights into search relevance, user interface experience, and overall satisfaction levels. This feedback loop ensures that user opinions and experiences are taken into account for continuous system refinement and improvement. This process was utilized extensively during user testing sessions we conducted within our social student circle, where user feedback was actively gathered to fine-tune and improve both front-end and back-end aspects of the system.

In evaluating our job search retrieval system theoretically, we focused on testing it using the top 10 searched job titles to simulate real-world scenarios accurately [Bro24]. The detailed results of these tests are provided in Appendix B for reference.

The high Success Rate at K=1 (SR at K=1) score of 0.9 indicates that the first job listed in the search results was highly relevant in almost all cases, demonstrating the system's ability to prioritize relevant content effectively. The Mean Precision @K=15 (Mean P @K=15) value of approximately 0.79 at K=15 signifies the system's consistent retrieval of relevant job postings within the top 15 results, contributing to enhanced user satisfaction and result quality. This high precision value suggests that users can trust the system to provide accurate and relevant job suggestions early in their search process, improving overall user experience and search result quality. Furthermore, the Mean Reciprocal Rank (MRR) of 0.925 reflects the system's proficiency in ranking relevant job postings higher in the list, ensuring prompt and relevant content presentation to users. MRR is a metric that measures how well the system ranks relevant items higher in the list, with a higher value indicating better ranking performance.

Additionally, the Mean Normalized Discounted Cumulative Gain at K=15 (NDCG@15) score of approximately 0.946562797 is a critical metric that evaluates the quality of the ranked search results in a more nuanced manner. NDCG@15 considers both the relevance of retrieved job postings and their positions in the ranked list, incorporating the diminishing returns of relevance as we move down the list. The high NDCG@15 score indicates that the system is successful in not only retrieving relevant items but also ensuring that they are positioned prominently within the top 15 results.

In our exploration of evaluation measures using a combination of pseudo-user behaviour and theoretical metrics, we estimated the Rank-Biased Precision (RBP) for our system.[MZ08] RBP is a measure that evaluates the utility of search results based on the likelihood of a user examining further results beyond the top-ranked items. It takes into account user behaviour of exploring multiple result pages and assigns higher weights to relevant documents encountered earlier in the ranking. A parameter "p" (persistence factor) in Rank-Biased Precision (RBP) represents the probability that a user will continue examining search results beyond a certain point. This persistence factor is crucial in modelling user behaviour and evaluating the utility of search results. To put it in context, at p = 0.95, there is a roughly 60% likelihood that a user will explore a second set of 10 results and a 35% chance that they will move to a third set of 10 results. We estimated mean RBP values for different p values, indicating how users interact with search results at varying exploration depths. The results showed a Mean RBP at P=0.2 of 0.7774268 and a Mean RBP at P=0.8 of 0.8639271. These values suggest that at lower probabilities of examining deeper into search results (P=0.2), the system still retains a significant portion of its utility, while at higher probabilities (P=0.8), the utility is further enhanced, emphasizing the importance of presenting relevant content early in the search results for user satisfaction and system effectiveness.

Many traditional measures like DCG and RBP rely on a position-based user browsing model. However, research has demonstrated that these models often fail to accurately approximate real user behavior[CMZG09]. To counter this, we also used Expected Reciprocal Rank (ERR) to evaluate the system. ERR considers the diminishing returns of relevance as users move down the ranked list, reflecting real-world user behaviour. Users are more likely to engage with and consider top-ranked items as highly relevant, and their interest diminishes as they move further down the list. The Mean ERR value across 10 queries of approximately 0.87 indicates that, on average, the system effectively ranks relevant items higher in the list, leading to a more satisfactory user experience. It suggests that users are more likely to encounter relevant content early in their search process, aligning well with user expectations and improving overall search system performance.

While evaluating the system, we observed limitations, especially lower precision values for search titles like Teaching assistant & Human resources. These terms are generic and have higher probabilities of appearing in other job titles. For instance, the term "assistant" is common in various roles like Research Assistant, impacting search result relevance. Conversely, less generic titles like Receptionist exhibit higher precision, showing the system's effectiveness. It's important to note that while this limitation is observed in the system, it is more due to the nature of the data rather than a flaw in the retrieval system itself.

9 Future Work and Conclusion

In conclusion, the job retrieval system has been successfully implemented with a strong focus on efficiency and user satisfaction. Looking ahead, there are several key areas for future work and enhancements. Firstly, extending the system's capabilities to support job searches across continents beyond Europe will broaden its global appeal and effectiveness.

Additionally, improving the system's query suggestion capabilities through the implementation of more custom-trained models, given the current computational limitations, will enable the delivery of more relevant and personalized suggestions, thereby enhancing the overall user experience. Furthermore, incorporating robust user monitoring features such as Click-Through Rate (CTR) analysis and Session Duration tracking will provide deeper insights into user behaviour and preferences. Introducing features like resume parsing, skill matching, and personalized job recommendations based on user profiles will further enhance the system's functionality and engagement, ensuring a comprehensive and seamless job search experience.

This project has not only highlighted the inherent limitations and uncertainties in developing a retrieval system but has also provided valuable insights into effectively addressing these challenges. It encouraged exploration beyond traditional concepts, fostering holistic development and innovative solutions in information retrieval and user experience realms.

10 Individual Contributions

10.1 Harish - s2583940

My primary contribution to the project revolves around parsing, data preprocessing, constructing inverted indexes, and developing additional features such as query spell check and suggestions using BERT for performance evaluation. A significant challenge encountered in this task was the multi-lingual nature of the data. To address this, I conducted research on various libraries supporting multiple languages for stemming and stop word removal. Despite utilizing LangDetect, certain languages remained unsupported or unrecognized, necessitating alternative approaches to handle exceptions. In constructing the inverted index for the entire dataset, I applied concepts learned during coursework 1, albeit contending with significant computational resource and memory requirements.

For query spell check and suggestion, I opted against using Levenshtein or similar modules for spell checks due to their computational expense, instead employing a weighted system leveraging existing modules. While this approach lacks support for multiple languages compared to Levenshtein, its reduced computational demands justified the decision. Furthermore, for the query suggestion component, I implemented a pre-trained BERT model with optimization techniques, filtering results to include only synsets of query terms identified as nouns. I opted for a pre-trained model over a custom one to mitigate computational and memory constraints, but was replaced by the method, currently employed in the system due to its accurate results.

10.2 Carlos - s2603149

My contribution was primarily focused on back-end development and collaborated in indexing and to a lesser extent front-end development. I utilized the FastAPI framework for its noteworthy performance benefits, ensuring a robust and efficient API service. This choice was pivotal in enabling the seamless integration of various search functionalities that I implemented, designed, tested, debugged and optimized such as boolean, phrasal, proximity, and ranked searches, including query expansion and the finally adopted query suggestion mechanism, which is a selection on titles starting with the characters introduced in the query. This approach achieved more realistic results, than employing BERT-based models. Additionally, for the indexing, I modified the preprocessing pipeline fixing some bugs and changing the storing database from MongoDB, which was the initial approach to Redis. This was done by collaborating with Ricardo. Moreover, I integrated the indexing with the scrapping. Furthermore, some of the front-end features such as the infinite scroll and the suggestion list were developed by me. Finally, I debugged the system as a whole in collaboration with Ricardo.

10.3 Loucas - s2567498

My main focus of this project was the data sourcing, collection and management along with the orchestration of the infrastructure and deployment of the final system.

Initially, I searched for various sources to collect data from, trying to abide by `robots.txt` and finally landed on the ones we eventually used. I used the Scrapy library to extract data from the static websites and developed a custom scrapping module for dynamic websites. I utilized concurrency and session cookies to speed up the process. I then developed code that could periodically extract the data from the specified websites and store it in a PostgreSQL instance. The module also supports automatic deletion of old entries. The final data collection module can collect up to 50 jobs per second (depending on internet speed). The initial data collection took around 3 days which yielded over 1 million entries, which allowed the rest of the team to start working a sample of our actual dataset. Then, the periodic data collection added any new entries posted on the websites. In addition to the data collection modules, I made sure to standardize the data model in order for the rest of the team to have a consistent format. Naturally, I was in charge of the storage and the initial prepossessing of the data during collection.

I was also in charge of integrating the different services into one repository and deploy the code on dummy servers on CloudLab. Towards the end of the project, I handled the migration and deployment of the codebase and data on Google Cloud. I researched on the best course of action on how to handle the Google Cloud credits and made a cost analysis of the different possible services we could use. Along with other teammates, we tried some solutions which proved to be too expensive and finally, I made the informed decision of deploying the code as a monolithic application in a Google Cloud VM.

10.4 Ricardo - s2556430

My contribution to the project is of significant impact as it involves designing, developing, testing, and implementing various components.

Firstly, I contributed by implementing an indexing algorithm used as an initial seeder for the index database. This algorithm iterated over millions of documents to create the initial index used by the system. This index and the data collected by the crawler provided enough information to begin the development of the HTTP API.

I worked closely with Carlos throughout the development of the HTTP API application. I contributed by setting up the start and closing of the application, adding connection pools to both databases to avoid disconnections and providing the app with resiliency, adding search features and error handling, optimizing code, and cleaning and refactoring the codebase for higher maintainability. I proposed the usage of the 'GitFlow' git workflow to provide more control over code changes and releases. I was responsible for merging and pushing code into the repository for everyone to maintain an updated version.

I worked closely with Loucas on the deployment of the application. I provided docker images for local development and the deployment of apps. I made an initial deployment on the GCP cloud run of the back-end and front-end applications. I provided initial versions of docker-compose files for local execution of the solution which provided a base for the final production release files.

I developed the front-end main features while Sreehitha and Carlos contributed by adding feedback, query suggestions, and infinite scroll features. I worked on the styling of the single-page application with attention to the responsiveness of it. I overlooked the code cleanliness and readability throughout the development of the application, contributing to faster bug fixing.

10.5 Sreehitha - s2603217

My initial contribution to the project began with collaborating with Loucas on data scraping for researching job sites, ensuring adherence to ethical guidelines. Subsequently, I transitioned to implementing some user features within the system, focusing on enhancing user experience and functionality by utilizing the features in Vue. Once the system reached a stable state, Dharma and I collaborated on evaluation and conducting user surveys to collect feedback and evaluate its performance using a variety of metrics, ensuring robustness and efficiency. Additionally, I played a role in integrating the project report, consolidating our development and contributions for dissemination and documentation purposes.

10.6 Dharmavenkatesan - s2569187

I played a pivotal role in the technical aspects of the project, specifically focusing on the indexing of the system. Working closely in collaboration with another team member, we devised and implemented efficient indexing protocols that significantly improved data retrieval and organization within the system. This involved designing algorithms, implementing data structures, and optimizing search functionalities to enhance overall system performance. Additionally, I led the evaluation phase of the project, conducting comprehensive research to identify and measure various metrics other than the ones used in course materials. Using these metrics, I assessed the system's effectiveness, user satisfaction, and adherence to project objectives, providing valuable insights into system effectiveness. In my project management role, I facilitated team collaboration, organized meetings, oversaw timelines, and monitored progress to ensure successful project outcomes. However, my involvement was not as extensive due to the team's highly motivational and independent nature. Team members readily supported each other in various areas when needed, reducing the necessity for my direct intervention.

References

[Bro24] Brother UK. World's most searched-for jobs. https://www.brother.co.uk/business-solutions/worlds-most-searched-for-jobs, 2024.

[CMZG09] Olivier Chapelle, Donald Metlzer, Ya Zhang, and Pierre Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, page 621–630, New York, NY, USA, 2009. Association for Computing Machinery.

[DRM+19] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. The design and operation of CloudLab. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, pages 1–14, July 2019.

[Mer14] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[MZ08] Alistair Moffat and Justin Zobel. Rank-biased precision for measurement of retrieval effectiveness. 27(1), dec 2008.

[ND23] Edward Nilsson and Dennis Demir. Performance comparison of rest vs graphql in different web environments: Node. js and python, 2023.

Appendices

A Architecture

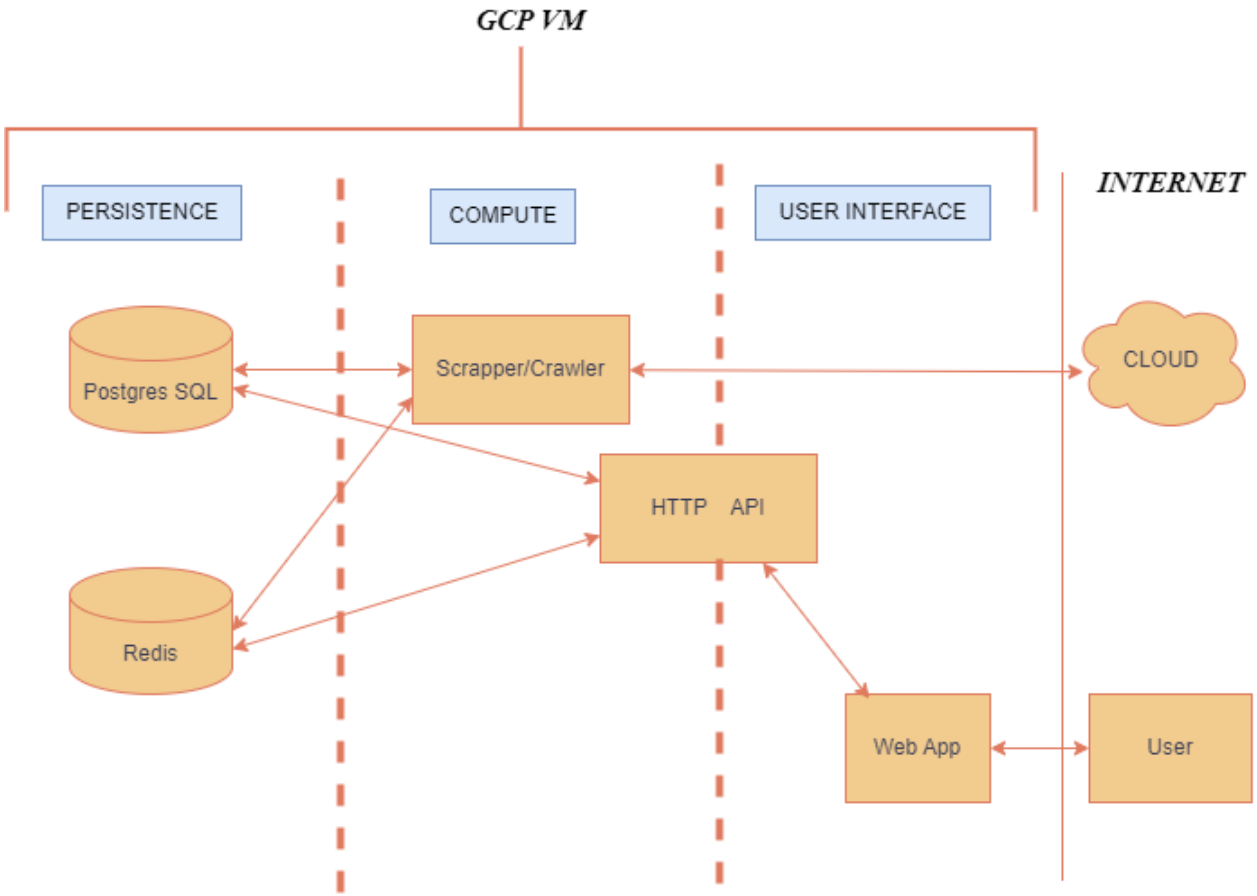

Figure 1: The system architecture of the search engine.

B Evaluation

Teaching assistant	Cleaner	Teacher	Cabin crew	Project manager
SEND Teaching Assistants - Hyde	Cleaner, Totton	Trainee English Teacher	CABIN CREW TRAINING MANAGER DUB/STO	Head of Event Management
Residential Teacher	School Cleaner, Par	Trainee English Teacher	Air & Ground Steward, Carterton	Head of Event Management
Summer 2024 EFL Teacher in London	School Cleaner, Lanlivery	Trainee English Teacher	Cabin Crew/ Flight Attendant, Belfast	Team Leader Infog rance, Par
Summer 2024 EFL Teacher in London	School Cleaner	Trainee English Teacher	Cabin Crew/Flight Attendant, Manchester	Engineering Manager, Plymouth
Teacher of English with a Potential Teaching and Learning Responsibility - Full-time	School Cleaner	Trainee Mathematics Teacher	Cabin Crew / No Experience Required / Full Training	Project Director - with Great Benefits, Madrid
Teacher of English with a Potential Teaching and Learning Responsibility - Full-time	School Cleaner	Trainee Physical Education Teacher	Cabin Crew / Flight Attendant	Stellvertretende/r Serviceleitung (d/w/m) (Fachmann/-frau - Restaurants und Veranstaltungsgastronomie)
Teacher of English with a Potential Teaching and Learning Responsibility - Full-time	School Cleaner	Trainee Physical Education Teacher	Cabin Crew / Flight Attendant	Driver and Branch Sales Assistant - Flexible hours, Guildford
Equine Studies Lecturer	Domestic Assistant, Southampton	Trainee Physical Education Teacher	Cabin Crew / No Experience Required / Full Training	Assistant Area Head in Mechanical and Electrical Services
Equine Studies Lecturer	Domestic Assistant	Trainee Physical Education Teacher	Cabin Crew / Flight Attendant	Human Factors Specialist
Clinical Scientist - Nuclear Medicine physics	Cleaner, Totton	Science Teacher x 2, Widnes	Cabin Crew / No Experience Required / Full Training	Clinical Nurse Manager, Bristol
Clinical Scientist - Nuclear Medicine physics	Domestic/Cleaner, Birmingham	Trainee Geography Teacher	Cabin Crew / Flight Attendant	Finance Business Partner - Rebates & G&A
Clinical Scientist - Nuclear Medicine physics	Cleaner	Trainee Geography Teacher	Cabin Crew / No Experience Required / Full Training	Finance Business Partner - Rebates & G&A
Surgical First Assistant — Mersey and West Lancashire Teaching Hospitals NHS Trust	Cleaner - Huntingdon	Trainee Geography Teacher	Cabin Crew / No Experience Required / Full Training	Sr Project Manager / Assoc Project Director, barcelona
Surgical First Assistant — Mersey and West Lancashire Teaching Hospitals NHS Trust	Cleaner	yoga + mat Pilates teachers, London	Waiter / Waitress / Receptionist - Train as Cabin Crew	Finance Business Partner - Rebates & G&A
Tax Manager - Innovation Taxes	Cleaner	ESOL Lecturer, Westminster	Cabin Crew / No Experience Required / Full Training	Head of Brand

Results of the top 1-5 search queries with up to 15 results. The first row represents the queries used for search

Human resources	Estate agent	Receptionist	Graphic designer	Accountant
Full Stack Developer - Hiring Urgently, Helsinki	Estate Agent	Receptionist Hotel - Hiring Now, Braine-le-Comte	Graphic Designer, Miami	Senior Accountant - Fiduciaire de Taille Moyenne, Full Package H/F - with Growth Opportunities, Brussel
Operations Consultant – Ukraine Monitoring Initiative (SSA), Vienna	Estate Agent Sales Administrator, Leeds	Receptionist till Aqua Dental Express	Desktop Publisher to Hästens, Köping	Senior Accountant - Urgent Role, Athens
Deputy Co-ordinator, Combating Trafficking in Human Beings (S), Vienna	Estate Agent Sales Negotiator, Cheltenham	Receptionist A - Hiring Urgently, Athens	Graphic Designer	Accountant Finance Department (f/m/d) - Luxembourg - with Growth Opportunities, Marche-en-Famenne
Human Resources Coordinator, Madrid	Apprentice Estate Agent	Front desk / Receptionist - Start Immediately, Athens	Graphic Designer - Immediate Start, Athens	Principal Accountant, Oxford
Human Factors Specialist	Trainee Estate Agent, Birmingham	Receptionist — Hotel Oostende — Voltijdse job	UX / UI Design Lead	Group Accountant (INTERIM)
Human Resources Manager	Estate Agent	Medical Receptionist	Junior / Graduate Graphic Designer (Fashion Company)	GROUP MANAGEMENT ACCOUNTANT, Leicester
Mgr-Human Resources - Urgent Hiring, Madrid	Estate Agent Valuer / Lister	Front Office Receptionist - Hiring Fast, Athens	Junior / Graduate Graphic Designer (eCommerce)	Senior Accountant - Urgent Role, Municipal Unit of Tavros
Human Factors Specialist	Estate Agent Valuer / Lister	Receptionist - with Great Benefits, Brussel	Junior / Graduate Graphic Designer (Fashion Company)	SENIOR ACCOUNTANT — INTERNATIONALE GROUP IN ANTWERPEN, Antwerp
Human Factors Specialist	Estate Agent Valuer / Lister	Receptionist - Urgent Hire, Lisboa	Junior Graphic Designer	Accountant - Urgent Hiring, Athens
Human Resources Manager, Milano	Estate Agent Valuer / Lister	Receptionist for Founderver - Athens, Greece	Graphic Designer	Senior Accountant - International Company - Multiple Advantages - Hiring Now, Ixelles - Elsene
Human Resources Manager - Delta Hotels by Marriott Huntingdon	Trainee Property Lister	Receptionist sökes till kund i Landskrona	Graphic Designer / Marketing Content Designer	Junior Accountant, Mechelen
Human Resources Manager - Delta Hotels by Marriott Huntingdon	Estate Agent Branch Manager	Full Time Receptionist / Administrator	Graphic Designer / Marketing Content Designer	Senior Accountant
Human Resources Officer - Urgent Hiring, Municipality of Chania	Estate Agent Branch Manager	Full Time Receptionist / Administrator	Junior / Graduate Graphic Designer (Technology Company)	SENIOR ACCOUNTANT — OLEN — 2 DAGEN THUISWERK, Olen
Associate Human Rights Officer (JPO, P2), Geneva	Estate Agent Office Manager	Receptionist, barcelona	Design Manager	SENIOR ACCOUNTANT — OLEN — 2 DAGEN THUISWERK, Olen
Associate Human Rights Officer (JPO, P2), Geneva	Estate Agent Sales Negotiator	Receptionist, Dorking	Graphic Designer	Corporate Accountant, Llangefni

Results of the top 6-10 search queries with up to 15 results. The first row represents the queries used for search

Query No	Query	DCG@15	IDCG@15	nDCG@15
1	Teaching assistant	20.04325507	20.04325507	1
2	Cleaner	31.1643565	31.32382746	0.994909
3	Teacher	29.2077571	30.24180618	0.965807
4	Cabin crew	27.84597542	29.50326743	0.943827
5	Project manager	17.95372216	19.12204234	0.938902
6	Human resources	16.44254476	22.62850981	0.72663
7	Estate agent	25.88743088	27.13001273	0.954199
8	Receptionist	31.02597391	31.33242307	0.990219
9	Graphic designer	28.03504961	29.4753642	0.951135
10	Accountant	32.0537706	32.0537706	1
Mean		**25.9659836**	**27.28542789**	**0.946563**

Table 2: DCG, IDCG and nDCG @15 Scores for the Queries

Query No	Query	RBP at P= 0.2		RBP at P= 0.8		ERR
1	Teaching assistant	0.635382		0.961599		0.946964
2	Cleaner	0.867843		0.999988		0.948245
3	Teacher	0.85362		0.9936		0.948196
4	Cabin crew	0.793821		0.84		0.932446
5	Project manager	0.71933		0.998656		0.948241
6	Human resources	0.319398		0.006708		0.241385
7	Estate agent	0.943341		1		0.948245
8	Receptionist	0.964816		1		0.948245
9	Graphic designer	0.711901		0.83872		0.932414
10	Accountant	0.964816		1		0.948245
	Mean	**0.7774268**		**0.863927**		**0.874263**

Table 3: Rank Based Precision(RBP) and Excepted Reciprocal Rank(ERR) for the Queries

Query No	Precision @ K = 15			
1	0.466666667		Total relevant results at K=1	9
2	0.8		Total queries	10
3	0.866666667		**Success Rate at K = 1**	0.9
4	0.866666667			
5	0.6		Reciprocal Rank for Q1-Q10 EXCEPT Q6	1
6	0.6		Reciprocal Rank for Q6	0.25
7	0.933333333			
8	1		**Mean Reciprocal Rank**	0.925
9	0.8			
10	1			
Mean Precision @K = 15	**0.793333333**			

Table 4: Precision, Success Rate and Reciprocal Rank for the Queries

	Results No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Query																
Teaching assistant		1	1	0	0	1	1	1	1	1	0	0	0	0	0	0
Cleaner		1	1	1	1	1	1	1	0	0	1	0	1	1	1	1
Teacher		1	1	1	0	1	1	1	1	1	1	1	1	1	1	0
Cabin crew		1	0	1	1	1	1	1	1	1	1	1	1	1	0	1
Project manager		1	1	1	1	0	1	0	0	0	0	1	1	1	1	0
Human resources		0	0	0	1	0	1	1	0	0	1	1	1	1	1	1
Estate agent		1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
Receptionist		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Graphic designer		1	0	1	1	0	1	1	1	1	1	1	1	1	0	1
Accountant		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 5: Binary Relevance for Top 15 Results for the 10 Queries

	Results No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Query																
Teaching assistant		4	3	3	3	3	3	3	3	3	1	1	1	1	1	0
Cleaner		4	4	4	4	4	4	4	3	3	4	4	4	4	4	4
Teacher		4	4	4	0	4	4	4	4	4	4	4	4	4	4	3
Cabin crew		4	1	4	4	4	4	4	4	4	4	4	4	4	0	4
Project manager		4	4	3	3	1	3	0	0	0	1	1	1	4	2	2
Human resources		0	0	0	4	2	4	4	2	2	4	4	4	4	1	1
Estate agent		4	3	3	3	3	4	3	3	3	3	1	4	4	4	3
Receptionist		4	4	4	4	4	2	4	4	4	4	4	4	4	4	4
Graphic designer		4	2	4	4	2	4	4	4	4	4	4	4	4	4	1
Accountant		4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

Table 6: Relevance Grades for Top 15 Results for the 10 Queries

Note - Relevance Grades: **Perfect (4):** Documents or items marked as "Perfect" are highly relevant and closely match the user's query or information need with exceptional accuracy. They are considered the most relevant and valuable in the context of the task or domain.

Excellent (3): Items labelled as "Excellent" are very relevant and provide substantial information or value related to the user's query or intent. While they may not be perfect matches, they are highly valuable and useful.

Good (2): "Good" items are relevant and contribute positively to addressing the user's query or need. They provide useful information or functionalities, although they may not be as comprehensive or precise as "Perfect" or "Excellent" items.

Fair (1): "Fair" items have some relevance but may be less directly related to the user's query or need. They offer moderate value but may require additional context or refinement to be fully useful.

Bad (0): Items labelled as "Bad" are not relevant to the user's query or information need. They do not provide valuable information or may even be misleading or incorrect in the context of the task.

C Back-end API Specification

The API provides three endpoints to three main services: search, query suggestion, and feedback submission.

1. `GET /search/`
   Provides access to the main search service. Receives two URL-query arguments: *query* and *page*. The *query* parameter allows you to input the search query value and the *page* parameter allows you to indicate which page of results you are querying. The results are paginated to reduce the response time from the search service.
   Example request URL: `search/?query=Developer\&page=1`

2. `GET /suggest/`
   Access to the query suggestion service. Receives one URL-query parameter *query*, which specifies the search query to take into account while generating the query suggestions.
   Example request URL: `/suggest/?query=develp`

3. `POST /submit_feedback/`
   Registration of feedback. Provides access to the service that registers feedback into the database. Requires body with feedback object.
   Example request body:

```
{
  "rating": 0,
  "feedback": "string",
  "date": "string"
}
```

D Github

Code for this assignment can be found in https://github.com/loucaspapalazarou/ttds-jobs-monorepo.