2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS) | 979-8-3315-0546-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CSITSS64042.2024.10816975

Hierarchical Data Storage Model Representation in MongoDB for a Resume Ranking Application

Sahil Kumar Gupta
Department of Computer Science and Engineering
Amrita school of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22081@bl.students.amrita.edu

Raushan kumar Gupta
Department of Computer Science and Engineering
Amrita school of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22086@bl.students.amrita.edu

Ishwor Acharya
Department of Computer Science and Engineering
Amrita school of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22088@bl.students.amrita.edu

Mohit Kumar Rauniyar
Department of Computer Science and Engineering
Amrita school of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
bl.en.u4cse22089@bl.students.amrita.edu

Divya KV
Department of Computer Science and Engineering
Amrita school of Computing, Bengaluru
Amrita Vishwa Vidyapeetham, India
kv_divya@blr.amrita.edu

***Abstract*—Introducing an innovative resume ranking website, representing Hierarchical Data Storage in MongoDB. By allowing candidates to fill out job-specific forms, the project assess their skills against customized criteria set by each company. Each skill criteria is stored in MongoDB in parent-child node formation making it easy for retrieval with minimum time complexity. Utilizing a point-based system, primary, secondary, and least priority skills are prioritized tailored to the requirements of each job opening. The goal is to make candidate selection, ensuring that the most qualified individuals rise to the top, facilitating efficient and effective hiring decisions for businesses.**

***Index Terms*—MongoDb, Trees, AVL tree, Linked list, ArrayList**

I. Introduction

The work present a sophisticated system designed to streamline the recruitment process for companies that ensure the selection of the best-fit candidates efficiently. Leveraging the power of data structures, the platform allows companies to define their specific skill requirements, categorizing them into primary, secondary, and least priority sets, each assigned with corresponding points. These skill sets are organized within a tree data structure, offering flexibility for companies to tailor their job postings according to their unique needs. Candidates applying for a job select their skills from the predefined sets, and the system automatically assigns points based on the skill's category, optimizing the evaluation process.

As candidates submit their applications, their names and corresponding points are stored in an AVL tree, sorted based on points for easy retrieval. When companies request the top candidates for a particular job opening, the system efficiently retrieves the top candidates with the highest points from the AVL tree[15]. This approach drastically decreases the time and effort traditionally required for manual candidate screening, ensuring that hiring managers have access to a curated list of top candidates tailored to their job requirements. The work revolutionizes the recruitment process, enabling companies to make data-driven hiring decisions swiftly and effectively.

II. Literature Survey

Numerous studies on document management system optimization have been conducted in the last several years. Multiple investigations were conducted to determine the optimal approach for storing data in a database to minimize access and updating times.

Ibrahim, Abbas Kh [1] highlights the importance of data structures in a database. The paper mentioned how to use tree data structure to store data in mongo db instead of traditional json file approach. Doing this will significantly reduce the access time and have a great effect on social networking platforms such as Facebook and Twitter, which are processing millions of data on daily basis.

Teng Lv, Ping Yen, Hongwu Yuan and Weimin He [2] worked together to find a way to store and process data in JSON formats.The study explored the benefits and drawbacks of storing JSON data in a variety of formats, from a straightforward data file format to incorporating a tree model, and ultimately identified a workaround that stores JSON text using the linked list data structure. This article provides great

information on capturing and maintaining key with value pairs and relationships between different keys in JSON documents by relationships using the Linked List data structure.

A novel hybrid storage model that functions more like a tree data structure and has faster processing and retrieval times was proposed by Yao, Jiaqui [3]. This paradigm creates an inverse index between the nodes and the trees, and stores the nodes and their relationships independently. Doing so significantly reduces the multiple recursive queries and input functions. This article is a new idea for storing data in a database for the project.

Yoshi and Eitan [4] compared the searching times of various families of trees and came to the conclusion that a complete binary tree is more efficient because it stores the values in such a way that half of the trees won't be looked at at once during the search.

Ellis Carla Schlatter [5] came up with a solution for the problem of simultaneous search and insert in AVL trees. They designed an algorithm that uses locks to make sure that either the searching or inserting process takes control of the node while it is performing its task and then unlocks it when done. In this way, any conflict arising during concurrent search and insertion operations can be avoided. The provided text gives an insight to consider as we run the developments in parallel processing environment.

Fahd, Djamel [6] illustrated the comparative study on various techniques for balancing the binary search tree and gave the conclusion that shows that the efficiency is different in various scenarios. They went into depth regarding the AVL tree, red black tree and b tree, which are the balanced versions of binary trees only. To wrap up, they presented a close lookup to what is the best version for which situation through the table view.

III. BACKGROUND

A. Adelson-Velsky and Landis (AVL) Tree

Among the diverse array of tree data structures, AVL tree stand out as self-balancing binary search trees with stringent height balance criteria.AVL trees keep the balance by making sure that a node's left and right sub-trees have height differences of no more than one. This balance condition guarantees O(log n) time complexity for efficient search, insertion, and deletion operations where 'n' represents the number of nodes in the tree.

The AVL tree's self-balancing property is maintained through rotation operations performed upon insertion or deletion, ensuring that the tree remains height-balanced. These rotations, including single rotations (left or right) and double rotations (left-right or right-left), preserve the tree's structural integrity while accommodating changes in the tree's shape due to insertions or deletions.

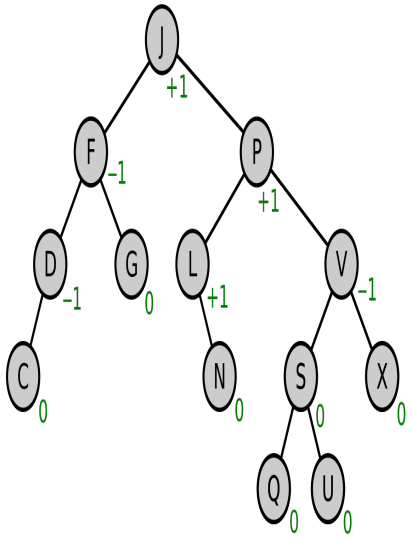

Fig. 1. Example of a AVL tree.

B. Queue

A queue is a simple linear data structure that has the approach of the First-In-First-Out (FIFO) principle. It operates on the basis that the element inserted first is the one to be removed first. Queues are mostly used in context where elements are added and removed in a sequential order, such as task scheduling, resource allocation, and breadth-first search algorithms.

A queue majorly supports two basic operations: enqueue and dequeue. Enqueue adds an element to the back of a queue, while dequeue deletes from the queue and returns the front element. This maintains FIFO behavior and ensures elements are processed in the order they are added.

Although there are many different underlying data structures that can be used to build queues, linked lists and arrays are the most popular options. Arrays offer constant access to their components, but may need to be enlarged if their capacity is exceeded. Linked lists provide efficient insertion, deletion, and dynamic memory allocation, but their overhead with memory allocation may make them more expensive.

C. Thymeleaf

Thymeleaf is a modern server-side Java template engine designed for web and standalone contexts. It provides a natural way to build dynamic web applications, allowing developers to create HTML pages with embedded server-side processing capabilities. Unlike traditional template engines, Thymeleaf templates can be viewed and edited as standard HTML files, making them easy to understand and maintain. One of the key features of Thymeleaf is its seamless integration with Spring Framework, although it can be used with other Java frameworks as well. With Spring Boot, in particular, Thymeleaf is the default choice for building web applications due to its

powerful features and ease of use.

Thymeleaf supports a wide range of template features, including expression languages, conditional statements, iteration, and internationalization. It also offers a robust set of utilities for processing forms, handling URLs, and working with fragments, allowing developers to build complex and interactive web pages with ease. One of the standout features of Thymeleaf is its support for natural templating, which allows developers to write HTML markup with minimal intrusion of template syntax. This makes Thymeleaf templates easy to read, write, and maintain, even for developers who are not familiar with the template engine.

IV. Methodology

This section outlines the systematic approach adopted to design and implement the tree based data storing algorithm in MongoDB to select the best fit candidate during the recruitment process of any company[18]. The following steps were adopted to achieve the objective of the project:

A. Requirement Analysis

Detailed and comprehensive study has been done on features and functionality of job selection platforms. Many websites were visited to find out the existing methodologies and functionalities. Based on the information collected, the solution has been proposed and implemented on the project.

B. Selecting best Data Structure

Applying this Data Structure lead the data to be stored in an efficient and better way rather than storing it in traditional way. On working on different dataset using various data structure like linked list and array we come up with idea that it can be optimize with using tree and graph. By analysing the project we finalize tree data structure will be more suitable. Tree data structure have unbalanced and balanced tree and with the dataset where storing information related to candidate balanced tree known as AVL tree prove to be better.

C. Design

1) Algorithm Design: In this section, algorithm will filter the best candidate among the applied candidates for the particular job vacancy[19]. Dividing the priority skills for job into three category as primary, secondary and least priority is done. When a candidate applies for a job by filling the form, all the selected skill with points as award will be stored in AVL tree[13]. Now this tree will balanced itself in such a way the the candidate with high point will be on top of the tree and from there fetching the top require candidate for the job will be easier as compared to previous transformer[11] and NLP concept[12][14].

2) User Interface Design: The most difficult part is to finalise the user interface. Determining the user interface is not easy to design as visiting and exploring many platform related to it gives idea of making different interface for both applicant and company. Whenever company open a job-opening for any job role, it should be visible to candidate's page and after applying for the role by filling the form it will be stored in the database as tree. This approach of making interface help both the candidate and company as it is flexible and easy. The information collected from multiple similar webpages give an idea of two different interfaces for candidates and companies. Viewing job description and apply for the job portal is there on the applicant side page however, the opening of job vacancies and adding job description section is there on the company side page. This way the project ensures the smooth user experience and functioning of both candidate and company's roles.

D. Development

At the time of development, the webpage has been developed along with server side implementation and database connection. NoSQL database MongoDB is choosen to store the data but in a hierarchical way with indexes. Web framework Thymeleaf has been used on top of Spring-boot to make interactive and responsive user interfaces. Complete server side code is written on Java, making it easier to code using object oriented concepts. Coming to data structures part, tree data structure has been used to store the information in hierarchical model in documents of MongoDB.

E. Testing

In this phase, the overall project has been tested with different input data set to ensure the expected working of the website. Different company account were created and job vacancies were opened from each account. Similarly, multiple candidate accounts were created and applied for particular job with different skills chosen. At the end, top 5 candidate for particular list was fetched and analysed which ensured that the selection algorithm was implemented properly. As well as different user interactive tools present in the webpage were manually tested to ensure their proper functionality.

V. Result Analysis

After the completion of development phase, the system is tested with different sample inputs to find out the expected working of the designed system. The comparative study of the project based on different alogorithms and data structures is presented below:

A. COMPARATIVE STUDY

1) Self Balancing and Unbalanced Tree: Unbalanced trees, as the name suggests, lack a consistent balance between their branches, resulting in inefficient search, insertion, and deletion operations. In an unbalanced tree, one branch may grow longer than the others, leading to skewed structures and degraded performance. Common examples of unbalanced trees include binary search trees (BSTs) that are not actively balanced, where the tree's shape depends heavily on the order of insertions or deletions.

Conversely, self-balancing trees are designed to maintain balance automatically during insertions and deletions, ensuring

efficient operations regardless of the order in which elements are added or removed.The AVL tree is a popular self-balancing tree that assures the heights of any node's left and right subtrees deviate by no more than one. Other examples include red-black trees, B-trees, and splay trees, each with its own balancing criteria and performance guarantees.

TABLE I
SEARCH TIME COMPARISON IN TERMS OF TIME COMPLEXITY

Tree Division	Time complexity	
	Best Case	*Worst Case*
Balanced	O(1)	O(log(n))
Unbalanced	O(1)	O(n)

Table I shows the primary advantage of self balancing tree to perform well in worst case with time complexity of O(log(n)) compare to that of unbalanced binary tree with time complexity of O(n).

2) Traditional vs Hierarchical Approach of Data Storing in MongoDB: MongoDB is a document-oriented NoSQL database, and its data storage model revolves around collections of JSON-like documents[7][20]. One of the primary features of MongoDB is its flexible schema design, allowing papers in a collection to have different structures. This flexibility allows developers to store different data kinds and adapt the schema as application requirements evolve over time. Additionally, MongoDB supports nested documents and arrays, enabling sophisticated data structures within a single document.

MongoDB stores data as collections of documents, where each document is a JSON-like data structure composed of key-value pairs. Documents are stored in a binary representation called BSON (Binary JSON), which allows for efficient storage and retrieval of data.

The traditional approach of storing data in key-value pair decrease the access time[8] but when it comes to storing nested data or the data is required in sorted order, then AVL tree significantly helps in reducing time. If the data is stored in hierarchical order using AVL tree then each data can also be stored in key-value pair as well as in sorted fashion[9]. Each node contains all the required information and follow the BST fashion based on the unique field.

Suppose if the student detail is to be stored in database with student name, branch, student-id and many more then, we can simply make a node that has all the required field and stored in the tree[10]. When another node is created with another student data we will place it based on the sorted order of student-id and similarly for many other records. Now, whenever information of student with id 50 or something is to be fetched then it can be done almost within half of the time compared to going and searching into a JSON file.

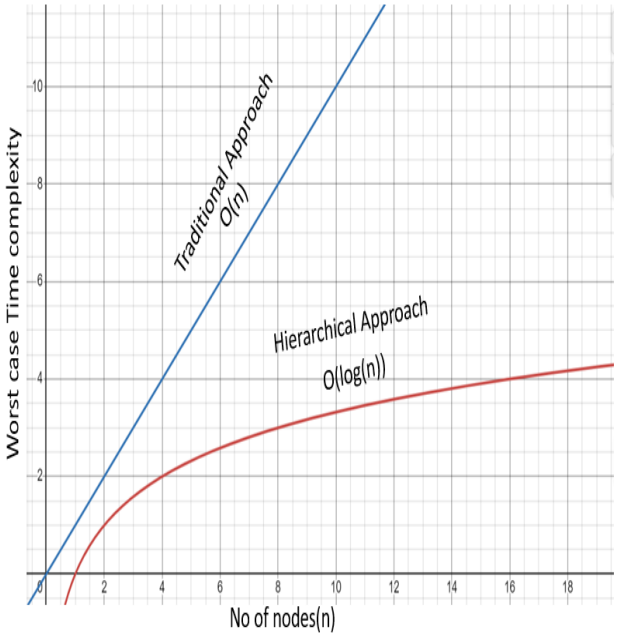

Fig. 2. Time complexity comparison between traditional and hierarchical approach of data storing in MongoDB

Figure 2 shows how the graph varies in array based traditional method and tree based hierarchical method of storing the data in MongoDB when there are 'n' number of records to store.

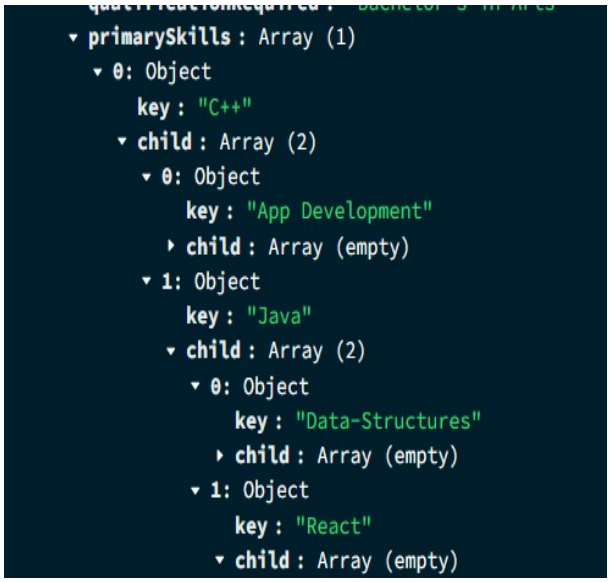

Fig. 3. Tree Representation in Database Document

Fig 3 depicts the list of all job skills mentioned by company in a job description which has been stored in the form of a map and object showcasing a tree like schema in MongoDB.

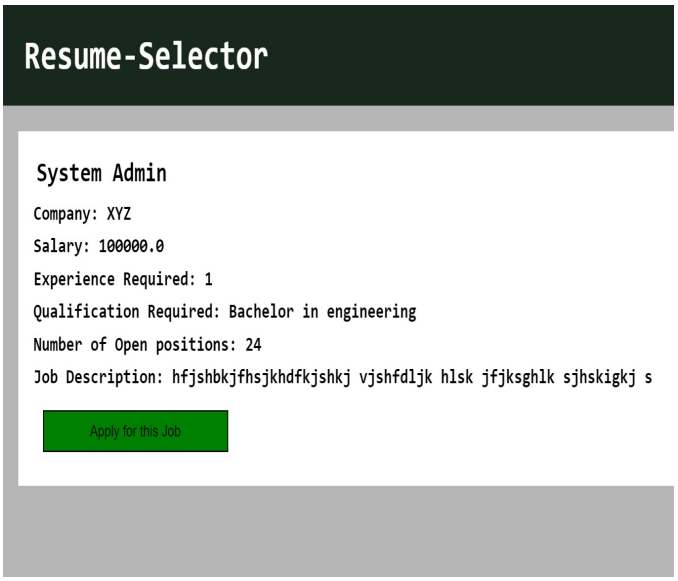

Fig. 4. Job Description on Candidate Dashboard

Fig 4 represents the job opening portal where candidate can view the job description and can apply for the job according to their preference.

Fig. 5. Form to create job openings

Fig 5 contains the job openings and is available to every registered company to open a new job vacancies for any position they require and set the required skills. The form takes in values for job role, salary, experience, skill set required, e.t.c.

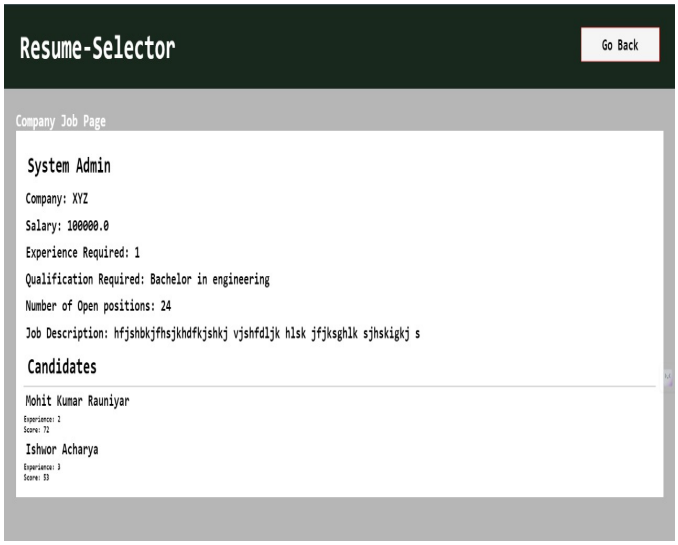

Fig. 6. Top Applicants for a Job

Fig 6 shows the job description created by company along with the flexibility to view the top 5 candidates best suited for the job vacancy.

VI. Future Work and Discussion

Future improvements to the job selection platform's functionality, user experience, and performance can be studied for industry-wide deployment. Here are some potential future works for the project:

1) Integration with External APIs and Services: Webpage can be connected with outside APIs and services to retrieve further information about applicants including their professional profiles on Linkedin or github repository where there works is displayed. Inclusion of APIs from employment forums or platforms for hiring is another idea in order to increase the pool of job listings that are accessible on the website.

2) Enhanced Security and Compliance Features: Supplementary security protocols, such as two-factor authentication (2FA) and data encryption, can be employed to safeguard confidential candidate and organizational data while assuring adherence to data protection laws, such as the CCPA or GDPR, with special attention to candidate data privacy and consent administration.

3) Enhancing performance with other data structures: Implementing other data structure instead of AVL tree can be fruitful based on the problem choice. Similarly, other methods can also be introduced to store data in the form of tree in database.

4) User Feedback and Iterative Improvements: Collect feedback from users (both candidates and hiring managers) to identify pain points and areas for improvement in the platform's functionality and user experience. Use agile methodologies to prioritize and implement iterative improvements based on feedback by users, making sure that the platform continues to grow to meet the requirements of its users.

VII. CONCLUSION

In conclusion, implementing tree data structures in MongoDB is a powerful solution for managing and categorizing skills within a job selection platform. This approach not only enhances the platform's flexibility and scalability but also improves the efficiency of candidate assessment and selection. With the added benefits of indexing and query optimization, MongoDB ensures quick and accurate retrieval of candidate data, facilitating better decision-making. As the platform evolves, the tree-based storage system will remain a crucial component, supporting the goal of creating a reliable and efficient marketplace for companies and talented professionals.

REFERENCES

[1] Ibrahim, Abbas Kh, et al. "A Tree Method for Managing Documents in Mongodb." vol 83 (2020): 18351-18359

[2] Lv, Teng, et al. "Linked lists storage for JSON data." 2021 International Conference on Intelligent Computing, Automation and Applications (ICAA). IEEE, 2021

[3] Yao, Jiaqi. "An Efficient Storage Model of Tree-Like Structure in MongoDB." 2016 12th International Conference on Semantics, Knowledge and Grids (SKG). IEEE, 2016.

[4] Ben-Asher, Yosi, and Eitan Farchi. The cost of searching in general trees versus complete binary trees. Technical Report 31905, Technical Report Research Center, 1997

[5] Ellis, Carla Schlatter. "Concurrent search and insertion in AVL trees." IEEE Transactions on Computers 29.09 (1980): 811-817.

[6] Meguellati, Fahd Mustapha, and Djamel Eddine Zegour. "A survey on balanced binary search trees methods." 2021 International Conference on Information Systems and Advanced Technologies (ICISAT). IEEE, 2021

[7] Lee, Hyeopgeon, et al. "Study of MongoDB Architecture by Data Complexity for Big Data Analysis System." The Journal of Korea Institute of Information, Electronics, and Communication Technology 16.5 (2023): 354-361.

[8] Mok, Wai Yin. "A Conceptual Model Based Design Methodology for MongoDB Databases." 2024 7th International Conference on Information and Computer Technologies (ICICT). IEEE, 2024.

[9] Xiang, Wang, Zhao Jianfeng, and Liu Chunxiao. "Analysis and Application of Power Equipment Production Quality Based on Data Hierarchical Processing." 2022 IEEE 2nd International Conference on Data Science and Computer Application (ICDSCA). IEEE, 2022.

[10] Fresta, Matteo, et al. "Supporting a. csv-based Workflow in MongoDB for Data Analysts." 2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE). IEEE, 2023.

[11] James, Vinaya, Akshay Kulkarni, and Rashmi Agarwal. "Resume Shortlisting and Ranking with Transformers." International Conference on Intelligent Systems and Machine Learning. Cham: Springer Nature Switzerland, 2022.

[12] Minhas, Abdul Hanan, et al. "An efficient algorithm for ranking candidates in e-recruitment system." 2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM). IEEE, 2022.

[13] Kavitha, D., et al. "Screening and Ranking resume's using Stacked Model." 2023 International Conference on Advances in Electronics, Communication, Computing and Intelligent Information Systems (ICAECIS). IEEE, 2023.

[14] Ambareesh, S., et al. "Resume Shortlisting Using NLP." 2024 4th International Conference on Data Engineering and Communication Systems (ICDECS). IEEE, 2024.

[15] Decorte, Jens-Joris, et al. "Career path prediction using resume representation learning and skill-based matching." arXiv preprint arXiv:2310.15636 (2023).

[16] Pranav, H., E. Suryaa, and Manju Venugopalan. "Comprehensive C2 Analysis and Anomaly Detection in HTTP Traffic: A MongoDB-Based Approach." 2024 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI). IEEE, 2024.

[17] Praveen, Sivakami, Alysha Dcouth, and A. S. Mahesh. "NoSQL injection detection using supervised text classification." 2022 2nd International Conference on Intelligent Technologies (CONIT). IEEE, 2022.

[18] Rithani, M., and R. Venkatakrishnan. "Empirical Evaluation of Large Language Models in Resume Classification." 2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT). IEEE, 2024.

[19] Akhilesh, P., S. Karthick Bharadwaj, and Manju Venugopalan. "Semantic Similarity Analysis for Resume Filtering using PySpark." 2024 IEEE 9th International Conference for Convergence in Technology (I2CT). IEEE, 2024.

[20] Reddy, B. Akhileswar, et al. "AI-Driven Stress Analysis and Management: A Novel Approach Leveraging OpenAI and NoSQL Databases to Empower Students in Stress Management and Promote Overall Student Well-being and Academic Progression." 2024 International Conference on Computational Intelligence and Computing Applications (ICCICA). Vol. 1. IEEE, 2024.