On-Demand Programming Recommendation System using Knowledge Graphs

Seyed Maziar Sojoudian

A Thesis
in
The Concordia Institute
for
Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master of Applied Science (Quality Systems Engineering) at
Concordia University
Montréal, Québec, Canada

March 2021

© Seyed Maziar Sojoudian, 2021

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Seyed Maziar Sojoudian**
Entitled:	**On-Demand Programming Recommendation System using Knowledge Graphs**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Quality Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair
Dr. Rachida Dssouli

______________________________ Examiner
Dr. Rachida Dssouli

______________________________ Examiner
Dr. Roch Glitho

______________________________ Thesis Supervisor
Dr. Jamal Bentahar

Approved by ______________________________
Dr. Mohammad Mannan, Graduate Program Director

March 23, 2021 ______________________________
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

On-Demand Programming Recommendation System using Knowledge Graphs

Seyed Maziar Sojoudian

With the increasing advancement of the internet and mobile technology, we are facing the information overload phenomenon. One of the solutions to this information overload is to filter data for the end-users. The information filtering process that provides more personalized results constitutes the main component of a recommendation system. Recommendation systems aim to provide closer results to the users' preferences. Therefore, if users have access to content that meets their needs, higher user satisfaction would be obtained. One of the domains that can benefit from recommendation systems is helping programmers write more efficient codes and develop faster by presenting them with solutions or code samples relating to their requirements. Although major repositories such as Stackoverflow and GitHub are trying to overcome this problem, there are still considerable shortcomings regarding the problem formulation and personalized results. In this thesis, we propose an on-demand programming assistance system that first helps developers present their problems. Then, through a natural language processing (NLP) module, the platform extracts valuable data from the presented problem. The users' questions which are asked on our platform form knowledge objects from which a knowledge graph is constructed following an efficient data model created on a graph database. With regard to the extracted valuable data from a knowledge object, the search module provides results from the Stackoverflow and the GitHub APIs. The end-users who ask their questions on the platform can save search results for the future or express their feelings about the results by marking them as useful libraries. Besides, our on-demand programming assistance platform provides a list of developers who have experience in different end-users' problems. After interaction, the end-users can mark those developers as experts and a sub-graph of the expert developers is appended to the knowledge graph. The platform collects 191 real-world programming problems for eight different programming languages via its powerful data model and represents the problem remarkably in the graph database in the form of nodes and edges.

The proposed recommendation system relies on the constructed knowledge graph to provide the end-user recommendation list containing libraries and experts from similar knowledge objects in the knowledge graph. Two main recommendation techniques, namely collaborative-based and content-based filtering, are used to create a robust recommendation system. The content-based recommendation method is used when an end-user is new to the system or there are no similar knowledge objects in the knowledge graph. The Jaccard index similarity, a weighting algorithm, and two different similarity measurement algorithms are used to build the mentioned content-based recommendation system. Moreover, when there is enough information regarding the knowledge object, the collaborative method is employed to solve the recommendation problem. The cosine similarity algorithm is utilized to apply collaborative-based recommendations on the knowledge graph. The two main algorithms, the Jaccard index similarity and the cosine similarity, were tested in different situations. First, they were applied in algorithms' standard forms, second with the proposed optimized form via gaining benefits from auxiliary data on the knowledge graph. These auxiliary data are nodes and edges, which help provide more filtering on the results. The proposed method brings more accurate results in comparison with standard baseline algorithms.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Jamal Bentahar, for his constant support throughout my journey at Concordia University.

I would like to express my gratitude to my HumanitiAI colleagues for assisting me and sharing their experiences. Nicolas, Patrick, Titash, I consider myself very lucky to have the opportunity to collaborate with all of you.

I would also like to thank my fellow friends and teammates for their collaboration to create the On-demand programming assistance. Hafsa and Zeinab. It was fun being your teammate.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

People rely on other people's advice and recommendation in everyday life by hearing their comments about news, travel, and more common habits [3]. This human social mechanism is assisted and supplemented by recommendation systems, which assist clients in sifting through websites, movies, music, shopping centers, and so forth to find the most appropriate and useful content [3]. Hence, recommendation systems are one of the promising tools to find the closest information to users needs in the overwhelming information era [4]. A typical recommendation system aims to make information filtering and is used to deal with the issue of information overload [4]. It's used to find appropriate or customized items based on the user's interests [4].

1.1 Problem Statement

Although recommendation systems have been successful in the past few years, more efforts are required to achieve higher customer satisfaction. Moreover, there are domains and areas which are not investigated thoroughly within the recommendation systems community. One of these domains is filtering massive online information on the internet's resources to help developers code more efficiently and faster. The IT industry still suffers from a lack of systems that provide help to programmers and answer their coding problems. Creating a platform to meet this need is highly desirable. The problem is how to design an on-demand programming assistance platform to help those who are striving to reach their goals in terms of programming. This platform should first benefit from the knowledge of other programmers using an appropriate data structure, i.e., a knowledge graph constructed through other programmers' questions, problems, and solutions. These solutions are derived results from available programming sources, in particular Stackoverflow and GitHub. Furthermore, suppose a new developer asks a previously answered question or asks a similar problem. In that case, the recommendation system will provide the most similar solution that exists on the knowledge graph to the new user. On the other hand, the on-demand programming assistance platform will recognize expert developers based on their activity in the platform. These expert developers are another source of recommendation. Moreover, if a user needs help from another developer, they can contact experts through the platform. If the expert was beneficial in solving the programming problem, the user could mark them as a useful developer. Later, those useful developers will be recommended for similar issues.

1.2 Research Objectives

This thesis's main objective is to contribute to designing on-demand programming assistance to help developers code more efficiently and improve programming teams' performance. A robust recommendation system that can recommend well-suited code examples, mainly from GitHub or discussions from Stackoverflow plays a notable role in achieving programmers' goals such as bringing a project or a task to a successful conclusion. Hence, the recommendation system should be applicable to suggest the best candidates for the end-users. Two main techniques are widely used in recommendation systems: collaborative-based filtering and content-based filtering. To achieve better results, our objective is to provide a comprehensive platform that combines the two techniques. Different algorithms will be applied to data for each technique. Many factors influence the selection of appropriate algorithms, including whether or not there is sufficient historical data. It is challenging to know which approaches are best without trying them on real data. As a result, various methods should be evaluated when taking data characteristics into account to determine which technique performs better in recommending the best solution for programming problems. The aim is to have a comprehensive recommendation system that can provide the desirable programming solution according to the developers' issues.

1.3 Research Contributions

The contributions of this thesis are divided into three main points:

- **Data modeling and constructing the knowledge graph:** As we mentioned in the previous section, our goal in this thesis is to recommend the best candidate in the dataset to the end-users by using a robust data model. The data model represents how data will be stored in our database. Since we are using the graph data structure, the data modeling has more influence on the results. Then the knowledge graph will be created based on the data model. We will investigate the first proposed data model for the beginning phase and then the second data model to store more helpful information on our knowledge graph.

- **Data gathering:** After creating the data model, the knowledge graph will be feed with real data. These real data are real questions from developers extracted from the Stackoverflow website. Those questions were converted to a more understandable way and were added to the database. The current knowledge graph is containing 191 real-world programming problems for eight different programming languages. It is noticeable that the knowledge graph has 1716 nodes and 3544 relationships between those nodes, which means that the data is in production and real-world problem scale.

- **Designing and Implementing new algorithms for a comprehensive recommendation system** When the required data are prepared, and the knowledge graph is constructed, different algorithms for the two main recommendation techniques will be applied to the data. The ultimate purpose is to propose a method that can apply more filtering to both content-based and collaborative-based methods to provide more accurate results.

1.4 Assumptions

In this thesis, problems that end-users are asking in the platform are known as Knowledge Objects. These knowledge objects are constructing the knowledge graph. Besides, knowledge objects can have different ways to express in Agile concept [5]. Still, we consider all the problems asked in our platform in this thesis as the User Story in Agile concept [5]. Another assumption is that all the programming teams and organizations in the knowledge graph are agreed to share information related to their team members with other teams and organizations. In real-world situations, this information can be protected, and data privacy will be available for them not to share their data with others.

1.5 Thesis Structure

The organization of this thesis is as follows. Chapter 2 studies the background and relevant related work. In the background section, necessary information, definitions, terms, and technologies, which are essential to understand this thesis, are presented. Then, related work that benefited other cutting-edge research in recommendation systems is detailed. In Chapter 3, we introduce the proposed framework. The technologies and modules used to create this platform are presented. The data model's design is then explained, and how knowledge objects are received as the end-user input is described. How those knowledge objects are stored in the knowledge graph in our graph database is discussed. Details about the data stored on the knowledge graph after feeding the graph through real-world problems are also provided. Communicating with the graph database via Cypher query is finally investigated. In Chapter 4, we present the recommendation engine and explain how the expert developers and useful libraries graph is constructed. This chapter shows how both content-based filtering and collaborative-based filtering techniques along with various algorithms are used to make comprehensive recommendations. Conclusions and future work are finally discussed in Chapter 5.

Chapter 2

Background and Related Work

In this chapter, important concepts, terms, and technologies, which are essential to understand the rest of the thesis, are introduced. These foundations are discussed in the background section (Section 2.1). Then, in Section 2.2, relevant proposals in the related work are discussed.

2.1 Recommendation Systems

On our daily basis usage of the internet, we are taking advantage of the recommendation systems, even though we are not aware of this use. We live in the big data era, which causes overwhelming information and uncountable information resources [6]. Therefore living without the recommendation systems is not imaginable. For example, when a user reads the news, if the news agency website or application can not understand users' preferences about the news headlines, it would be a discouraging experience for the clients. Also, the news agency can not engage the user with their content. Here, the recommendation system will help to solve the problem by referring the closest news article to the user's taste and preferences. It will result in user engagement for the news agency. Other examples of the daily basis for facing the recommendation systems, which are the most famous examples, are online movies and music streaming platforms. These platforms are trying to help their users to find content that meets their expectations. Afterward, user satisfaction will be achieved from the recommendation system, and the platform will have its revenue.

In the recent decade, the development of digital communication and digital trading, which has resulted in more digital life, has been increased. People meet their needs online through the internet, and web-based applications, such as ordering food, taking a taxi for transportation, shopping online for groceries or other goods, through using this development. Even finding a perfect match as a partner, husband, or wife is now is done using dating applications. Recommendation systems are taking place in all these examples. To recommend the best possible candidate for users' preferences.

After we learned what the recommendation system is, it is time to discover how they help us solve the mentioned problems by knowing different recommendation system approaches. Mainly recommender systems have two different methods to solve recommending the best item that fits people's personal needs: collaborative filtering and content-based filtering Dong2017 [7]. By combining these two methods, we can have a robust recommendation system called hybrid recommendation system that improves previous methods' problems.

2.1.1 Collaborative Filtering Recommendation Systems

The collaborative filtering approach is one of the recommendation systems types, which has been studied for more than a decade to deal with the data overload to make more accurate predictions and recommendations [8].

With the Internet's success and the inclusive use of the internet by society, an enormous amount of information has been created. These massive information resources have some issues besides their benefits, such as difficulties finding relevant information for the user, which is usually specified as "information overload." Many different solutions have been provided to solve this issue. Before our modern and digital time, people would trust in the opinion of their friend's network to find suitable chose between friends who have a closer taste to them in a particular subject. For example, ask a friend who knows our preferences and has good knowledge of many movies to suggest a new movie. Collaborative filtering is implemented and developed based on this theory [8]. A collaborative filtering system needs to collect thousands of users' preferences and feelings. These preferences and feelings are stored as ratings by users for items. Now, the system will prove a recommendation for an active user by finding users with similar preferences and using their taste to make a recommendation. Platforms with multi-value rating data are using this methodology, and they are good examples to prove the methodology [8]. In other words, collaborative filtering-based methods use historical data, previous activities, and opinions, like purchasing/viewing records or user rating for a specific item, and make a recommendation with no consideration on the content of an item [3] [6].

The collaborative filtering approach for the recommendation systems has a different way to solve the recommendation issue by recommending the best item in the platform or a ranked list of top items that exist in the platform for the user. This list is referred to as a top-N recommendation. This list is usually between 1 and 20 items and mostly implemented from a web-based application for the recommendation [8].

Collaborative filtering algorithms are the main methods to recommend items on famous e-commerce websites. For example, 20% - 40% of purchases on the Amazon website are communing from the recommendation system, and Netflix's 60% of DVD renting is also coming from the recommendation system [6].

Collaborative filtering has two main techniques: memory-based and model-based methods [6].

A. Memory-based Collaborative Filtering

In the memory-based collaborative filtering, method algorithms acquire similar connections and relations among the clients or items regarding the user-item scoring matrix. After that, suggest the items with the highest rate among other recommendation items in lists and are highly scored by users comparable to the active user [9]. To predict the rating score for new items that are not rated yet, the memory-based collaborative filtering method uses historical rating scores on other items from users directly [6]. Two algorithms have been proposed within the memory-based collaborative filtering system: user-based collaborative filtering and item-based collaborative filtering.

i: User-based Collaborative Filtering Algorithm. The theory of user-based collaborative filtering is based on deterioration, which says clients with similar rating history should have similar interests; therefore, the algorithm can predict the rating score for the active user's not rated items regarding a similar user's rating score on specific items [6]. This algorithm has three steps to provide recommendations for the active user: (1) calculate the similarity between users; (2) find the nearest neighbors; and (3) predict the ratings. The similarity calculation process between the active user and other users will be done in the first step. Second, based on the calculated similarity, the nearest neighbors to the active user will be selected. In the last step, from historical preferences data of the similar neighbor users, rating of specific items from the active user will be predicted [6].

ii: Item-based Collaborative Filtering Algorithm. Regarding the item-based collaborative filtering recommendation approach, the algorithm should first determine the relationship between items by evaluating the user-item matrix and employing these relationships to make a recommendation for clients [10]. The same steps used in user-based collaborative filtering should be taken to solve the item-based collaborative filtering problem, namely (1) calculate the similarity between items; (2) find the nearest neighbors; and (3) predict the ratings .

B. Model-based Collaborative Filtering

The accomplishment of the memory-based recommendation systems is uncomplicated because its algorithm is more clever and more understandable. When collaborative filtering deals with large datasets of users and items, the memory-based recommendation system is not appropriate for the recommendation problem. Another recommendation model named model-based collaborative filtering is suggested to fix this weakness. Before generating any recommendation, model-based collaborative filtering needs to detect excellent model parameters in a learning phase. The model-based collaborative filtering can anticipate the rating of users immediately right after the learning phase is done. This approach has three different ways to recommend items to users: a: matrix factorization model, b: non-negative matrix factorization model, and c: singular value decomposition (SVD) [6].

Technical Challenges in Collaborative Filtering Recommendation Systems

Collaborative filtering recommendation systems generally meet major issues due to the growth in the data volume and different data types. In the following, we will investigate different technical challenges in this recommendation system method.

(1) Data sparsity. The user-item matrix always has many unknown rating scores in the matrix, and it has more than 99% sparsity. Enormous data sparsity is caused by an increased amount of common ratings among items too few or none. In the similarity calculation, there is a big deflection, which resulted in recommendation performance. Therefore data sparsity should be considered before choosing an algorithm, or it should be included in the suggesting algorithm to solve the issue [6].

(2) Cold start. Collaborative filtering algorithms are based on historical data. When a new user is registered on the system or a new item is added to the system, there is no historical information for them, no preferences information for the user, and no rating for the item. Consequently, the system can not recommend an item for the new user, or a new item can not be recommended by the system for users. For the cold start problem, content-based filtering algorithms are usually taking place to solve the issue. We will talk about content-based filtering algorithms in the next section [6].

(3) Scalability. In online platforms like social networks and e-commerce websites, data is growing geometrically by the increase in the number of users or items. This issue has resulted in a high computational cost for the system to make a recommendation. The model-based approaches solve this problem by train model parameters offline to increase an online recommendation system's performance through user modeling, similarity calculating and features extracting [6].

(4) Diversity. Collaborative filtering recommendation systems are more successful in recommending famous and high rated items to the current user. Hence, a good recommendation system should have the ability to discover and recommend items that are difficult to be found by the user. At the same time, the recommended item should meet users' needs [6].

(5) Interpretability. Evaluating recommendation system success is crucial to be calculated and achieve an acceptable result. The better recommendation means the system has better interpretability of the items and the users in the system. Thus, the algorithm's excellence can not be measured entirely with Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). Because they can not describe the recommendation result very well, and they can not specify if the suggested items fit with users' needs [6].

2.1.2 Content-based Filtering Recommendation Systems

Recommended items are selected regarding their correlation and relationship between the content of the items that exist on the dataset by Content-based filtering. In the book recommendation example, the book's content and user purchased history of a particular book will be considered. A chain of attributes of the book will be used to recommend more books with similar content [11].

In other words, the Content-based filtering method will suggest items to the active user regarding similarity count from previous positive rates [12] [13] [14] [15]. Another example, when a user likes websites with the following keywords: mobile, pen driver, and RAM, the Content-based filtering suggests a website relevant to the electronic world [15].

Description of an Item and an active user's profile play a key role in Content-based filtering recommendation systems. Recommendation approach in Content-based filtering method is implemented on similarity count [15]. Different recommendation candidate items will be compared with user rating history on other items to find the best match. The tf-idf is the most widely used algorithm for Content-based filtering. User profile creation usually is focused on two different kind type of information 1. A user's preference model. 2. User's interaction history with the recommender system. To qualifying the item inside the platform, the Item profile is mainly employed tf-idf algorithms. Users profile is generated with gaining the benefit of the weighted vector of item features in the Content-based filtering recommendation system. Weights would define the importance of the features (value, validity) for a user. Different techniques can calculate this importance from individually rated content vectors. Content-based filtering techniques have three steps: (1) extract the characteristics of the recommendation items; (2) compare item attributes with the active user's preferences; and (3) suggest items based on features that meet the user's needs [15].

While item attributes and user profiles are acknowledged, the primary responsibility for content-based filtering is to detect if the active user will like a particular item. Classification algorithms such as rule induction, nearest neighbors methods, Rocchio algorithm, linear classifiers [16], and probabilistic methods or heuristic methods will take place to determine if an active user will like a particular item [17] [15].

Content-based Filtering Benefits

First, through limited ratings, which the active user uses to create their personal profile, user independence is provided by the Content-based filtering recommendation framework [15]. Second, by describing how the recommendation process occurs, the Content-based filtering recommendation system provides transparency for active users [15]. Third, the content-based filtering recommendation system is an appropriate method to suggest unexplored items. Newly registered users in the platform will gain benefit from this [15].

Content-based Filtering Challenges

First, extracting attributes for specific items in different areas is a difficult process. Second, Since it suffers from the problem of overspecialization, this approach advocates the same type of products. Third, users do not usually participate in giving rate to items; it is inexplicable to collect users' feedback; therefore, it is not feasible to detect if the recommendation is correct or not [15].

2.1.3 Hybrid Recommendation Systems

The Hybrid recommendation approach has better performance. In the information filtering domain, two methods, as mentioned earlier, collaborative filtering and content-based filtering algorithms, are the most popular. Those two techniques have their challenges and advantages. The purpose of the hybrid recommendation system fundamentally is to increase accuracy through aggregating collaborative filtering and content-based filtering algorithms [15].

Two vital weaknesses in recommendation systems, cold start and the sparsity are solved regarding the Hybrid recommendation techniques. Netflix is a perfect example of this method. By comparison over the active users' search and finding similar preferences of similar users (collaborative filtering), along suggests movies with common attributes with films that the active user has rated with the total score (content-based filtering) [15].

Collaborative filtering and content-based filtering algorithms can be combined in different forms. In the following, we will study these different forms. The first form aggregates collaborative filtering and content-based filtering after evaluating them individually to return the improved recommendation across the board. Second, to resolve the cold start issue in collaborative filtering and overspecialization issue in content-based filtering, the next method is suggested by integrating content-based filtering into collaborative filtering. The third approach mixes beneficial features from collaborative filtering and content-based filtering algorithms to develop a unified model. This method, called the unified utility system, has the characteristics of both aforementioned algorithms, which can highly improve recommendation results [15].

2.1.4 Graph Databases

One of the vital factors in online recommendation systems is how we should store data and how we should be retrieve required information from those data sources because dealing with live data on a large scale is an expensive process. Recently graph databases are preferred in many areas to Relational Database Systems (RDBS), domains like Chemistry, Biology, Semantic web, social networks, and recommendation systems. Those are all domains in which graph databases can express them in a more natural design [1]

When it comes to big data, relational database systems have some disadvantages: storing, retrieving, and complex data manipulation. On the other hand, in highly connected data, graph databases are optimized [1].

Fundamentally edges and vertices are constructing a graph. [18]. In subjects that illustrate the relationship among the entities is vital in the data model designing, technologies like graph databases can play a key role in being efficient in data modeling. Property graphs, which are directed multi-graphs, are labeled, attributed, and most systems and designs support them. Figure 1 denoted interactions between people and objects in a property graph [1].

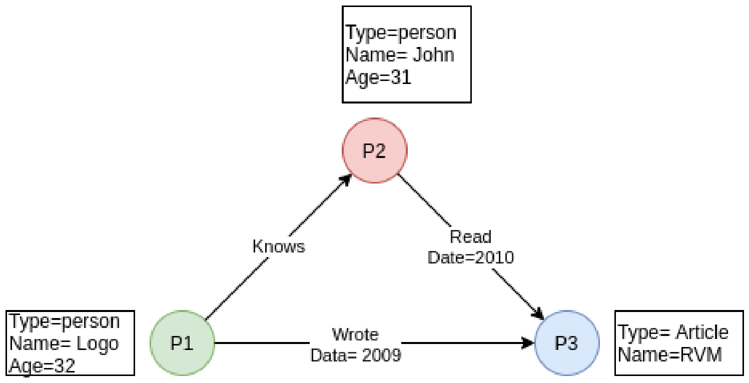


Figure 1: An example of graph, structured by edges and nodes [1].

Since the multi-graphs are the most complex in implementation due to construction with the subset of the property graphs, they can model all other graph types more efficiently [1]. Therefore, for dense and interrelated datasets, situations, graph databases are an optimized and efficient solution [18]. Graphs are represented by nodes and edges (edges are also known as relationships). These nodes and edges can have their own properties, and edges connect all nodes as the relation that is a highly well-organized structure for this dynamic data model. It is powerful for high-speed traversing edges across vertices [1]. This traversing is localized, and it is not investigating unrelated data. As a result, graph databases are powerful in detecting correlations and patterns [1], which makes them a highly robust candidate for recommendation systems.

2.1.5 Knowledge Graphs

Recently, because of the advantages of schema-less nature, knowledge graphs are employed extensively. This schema-less structure lets knowledge graphs grow smoothly via adding new relationships and entities. To give semantic to the textual information, knowledge graphs with a labeled directed graph are denoting knowledge in the graphs. Edges and nodes are constructing a knowledge graph (like graphs in graph databases). For instance, each node illustrates items, entities, users, and so forth, and edges connect nodes. These edges represent nodes' relation. Knowledge graphs are assisting in extracting meaningful knowledge from information. An example of data processing to extract knowledge from information can be like "Li Ming's height is 177cm". This sentence (we consider this sentence as information) can answer the question of "Li Ming's" height. Clearly, that was only the first step to extract useful information. The more complex question can be answered using the aforementioned technique [19]. This technique extracts information through abstracting and converting the information and the data in a given context [20]. Summarizing knowledge from information and digging the information from the data can be done via defining characteristics and classification summary [19]. As a result, general knowledge of the current class on information can construct desirable judgments, accurate predictions, and smart decision making [19].

2.2 Related Work

In this part, recently related works have been reviewed, and their methods and approaches to the recommendation problems will be discussed. The recommendation problems have been denoted in Knowledge Graph and graph-based recommendation systems and how they are modeling data in knowledge graphs constructed in the graph databases. We will review three recently published papers in this domain in order of, first, "`A Graph Approach for Job Recommendation at Scale`", second, "`Efficient Approach for Social Recommendations Using Graphs on Neo4j`" and "`BetterChoice: A Migraine Drug Recommendation System Based on Neo4j`". We will discuss each paper in detail.

Different solutions based on knowledge graphs constructed in graph databases that are solving recommendation issues in different domains will be studied in this section. We will discuss how different proposed solutions will model data and combine different recommendation methods together in other problems in distinct situations.

2.2.1 A Graph Approach for Job Recommendation at Scale

Recruitment is one of the largest issues that both companies and people have. Supplying human resources between millions of candidates is vital for organizations since they have to find the best fits for their empty positions or new opportunities for people to find the closest job to their skills and interest. Shalaby et al. in [21] tried to find a solution for this problem via a graph-based approach, which is also suitable for large-scale job-seeking websites in the real world. Because content analysis of resumes and job textual information are more considered to solve this problem before, which sometimes end up with an unsuccessful result and it is suffering from the scalability or cold start problem of the collaborative filtering techniques, they proposed a scalable item-based recommendation system to suggesting job to people. Using graph databases and directed multi-edge graph models of jobs illustrating the diversity of behavior and contextual similarity signals, issues like sparsity and scalability have been overcome. The item-based collaborative filtering has been used through their popularity between recommendation techniques used in this domain, especially by employing user-item interaction to extract similar items [21]. Regarding the aforementioned, there are two main challenges in constructing the recommendation system for this domain:

(a) Scalability: A graph-based structure has been proposed to achieve an accurate model based on the position-position relation through variable-length neighborhood sizes. To enhance efficiency for the future additional updates in data jobs have been divided into two different categories. First, active position for architecting and renewing the online graph. Second, leverage expired jobs in offline environments.

(b) Job Similarities/Sparsity: like movie recommendation systems, user behaviors are recorded as user-item interaction in the form of ratings. Diversity of similarity metrics among the item and clients are computed based on these ratings via cosine or correlation-based measures techniques [22]. Since relying only on users' rating will cause failure for item-item similarity computation because of the high data sparsity level [23]. Thus, to ultimately collect the relationship between jobs and reduce the scattered data issue, different data sources in the job domain have been analyzed and developed.

(c) Cold-Start: when user-job interaction does not exist in the platform, the low-quality recommendation result will happen. This is known as the "cold-start" problem and will become more significant in dynamic systems like job recommendations because new clients and or job opportunities are added to the platform at high scores.

To overcome the aforementioned challenges, a modern item-based job opportunities recommendation system has been proposed. Through a unified marketable graph-based architecture by improving various contextual and behavioral reactions. This platform includes three steps. First, they construct a homogeneous graph consisting of job opportunities while nodes with various edges connecting those nodes to collect aggregated behavioral plus contextual signs. Second, by weighted aggregation of all their association edges, a weighted directed edge is generated concerning all job pairs. Third, the model user expectations and produce recommendations utilizing propagation-based search techniques on the graph [21]. Since the method used in this research uses the advantages of both collaborative filtering and content-based filtering, it is considered as a hybrid recommendation system approach. Sparsity and low-quality recommendations, challenges in collaborative filtering are trying to be solved using graph-based designs that utilize connection investigation approaches from the graph theory [24]. Recommendation structures based on graphs are distinguished based on how the graph is built and traversed for making the recommendation process. Nonhomogeneous graph-based standards construct a bipartite graph of clients and goods [25]. However, homogeneous models only consist of users [26] or items [23] to be represented as nodes.

The collaborative filtering approach can be expressed as a link prediction problem in a user-item bipartite graph, where edges reflect the interaction between users and objects [21]. Graph theory can be used to predict a non-existing edge between a client and an object using neighbor-based and path-based linkage methods [24]. To have a better scalable recommendation system for a job opportunities website, they simply develop an item-based graph [21]. A homogeneous graph-based approach is suggested for clients as the nodes plus a degree of predictability as to the edges between users [26]. They traverse routes of each connected node, starting from the client, to identify a client who has scored the items of concern, unlike the nearest neighbor search [21]. The authors create an item-based graph in this paper as well, but they use a local search technique preferably than a global propagation-based method on the graph. Besides, They support different edges across nodes and measure an aggregate asymmetric interaction rating related to job content, application co-apps, plus co-clicks to capture the popularity of jobs as well as their content and interaction similarities [21]. Hence, homogeneous Graph-Based Recommendation architecture has been chosen to develop the recommendation system. Jobs outline nodes, and edges denote different similarity rates among sets of job opportunities in this recommendation graph [21]. They measure the similarity values from various information roots that record user actions, in addition to resumes and job opportunities content that they choose to construct an item-based graph preferably than a user-item graph or user-based, as given the number of users in their suggestion pool, this allows for more extra scalability. They model the actions of users of specific signs, such as their work applications, and explicit signals, such as their clicks. They can calculate various job-job co-statistics like co-apps, which shows how many users applied to both jobs for any set of positions, and co-clicks, which shows how many users clicked on both when they appeared in the user query result collection for any pair of jobs [21].

Furthermore, They suggested innovative methods for a salable and robust job recommendation framework to introduce advanced recommendation systems' vast potential into the job opportunities search domain. The method includes a multi-graph of jobs linked by edges of similarity optimized to capture their complete relationship based on user activity and work content. Furthermore, they make use of the vast amount of tacit and explicit online data that is freely accessible. They then reached a level of accuracy that was highly acceptable. It also shows that with a wide edge across the classical collaborative filtering approach, our proposed method achieves higher Expression of Interest while requiring just a third of the amount of sent emails.

As mentioned in the paper, this approach is not good enough for their needs because of the cold-start and distribution skew, even though they had a low Mean-Square-Error. They could solve the cold-start problem with a better data model through natural language processing or Natural language understanding (because the data is in textual structure and it is easy to apply mentioned method on the textual data). And then, they could create a content-based filtering recommendation that would save them from the cold-start effects. On the other hand, they only used the PageRank algorithm, and they did the same process, for example, for the active users, for two or in some mentioned scenario more than two steps. They did the technical process on the data, which are only for the last 180 days, not for all data in the dataset. Moreover, from the tools and methods they used, we can know that their approach is offline, they will make the recommendation list for each user offline, and then they will show the recommendation list to the user if they are offline or through sending email to the users. More importantly, the activity score is not exact; they did not specify the activity value range.

They could use different algorithms like different similarity measurement algorithms to find which items (jobs) are closer to the user preferences, and also, they could use native graph databases to represent data they received from the careerbuilder in the form of the graph and apply their method on it.

2.2.2 Efficient Approach for Social Recommendations Using Graphs on Neo4j

Social networks are becoming a fundamental value in internet businesses and the path to knowledge that offers recommendations that strongly reduce comprehensive data to the items that completely solve the user's problems and weaknesses. Low accurate recommendations are still existed due to the cold start problems. Virk et al. in [2] suggested an additional advantageous situation for these schemes: consumers can encode more knowledge about their relationships than they can ultimately tell whom they trust. They have also suggested transitivity in social networks along with trust from which we can obtain more specific recommendations. Here we will discuss those solutions they proposed, and also, we will investigate their results. In the last few decades, the web has grown exponentially, resulting in a detonation of knowledge, and we have also seen a rapid increase in the number of mobile phones [2]. When customers have many details, it is difficult for them to find correct and trustworthy data. As the number of options available grows exponentially, the problem becomes inundated with data [27]. Here recommendation system can perform an important role. In this research, Virk et al. in [2] introduced Trust Aware Social Recommendation system to employ in problem and reduce the recommendation accuracy. Smart applications that use trust data that one person expressly trusts to another person and individual consumer information in social systems to provide personalized recommendations are known as trust-aware recommendation mechanisms [2]. Previous research into trust-aware frameworks has shown that trust-based platforms' ability to deliver reliable predictions blended with their shilling attack vigor makes them a better choice than conventional recommendation frameworks [28]. So, they also devised a method for manipulating the graph with the effect, i.e., the influence of one individual on another's selection, based on trust data. The graph that is interpreted in the document is referred to as the graph of impact. The influence graph extends social trust and motivation between related customers and this impact besides the modern theory of recommending transitivity persuasively and competently in recommending products. Trust is specifically mentioned by the customers in whom they believe. In general, the authors compare users' similarities to provide them with accurate and reliable suggestions. It is possible to calculate the Pearson correlation in order to decide how many users are close to each other. [2].

Now, let us take a look at how this researcher builds their recommendation framework. The algorithm is applied to practice the user as an individual user in previous recommendation systems as it does not connect with any other user. Still, when the social recommendation system was first developed, it did not consider the client to be an autonomous individual; instead, it used various algorithms to extract social relations between users. In conventional recommendations, multiple vulnerabilities were examined in which clients were considered self-sufficient, so this drawback was resolved by social recommendation. Previously, users used to make decisions based solely on their own actions [2]. The fundamental downside to this filtering strategy is user privacy and user independence. Collaborative filtering is another technique besides content-based filtering. Data filtering, in which client similarity value is determined by comparing how one user classified an item to another user scored the same product or rating of a specific item. Both are identical, and their choices are similar to each other according to this method. So, if one user has rated some item favorably in the future, another user of similar preference will receive the product's recommendation. Again, the downside to this tactic was that users were treated as an individual party. Information is encoded in various formats. Details of ratings given to products by diversified users are listed in the user-item matrix. This dataset typically does not have many entries as it is extensively broad. Since the user would not rate each item, the user defines just a few scores for such items of interest. Because of the smaller number of ranking entries in the dataset, the following matrix is sparse when the dataset has few entries. Another issue that arises is when the client has recently registered. If clients are new to recommending the system or unspecified web pages on which they are searching for feedback, they will not have rated any item before, so similar users will not be identified. A cold start issue occurs when a user is new to the recommendation system and has not yet entered any ratings for his likes or dislikes, from which the recommendation system will infer any suggestions for that new user [2]. Researchers suggested a method in which hyper-edge is used to monitor both the current issues, data sparsity in datasets, and cold start problem for users' influence. Nevertheless, besides the lack of knowledge and cold-start on trust and social relationships between users, another problem arises, which traditional guidelines face. Let us say one user p wants to purchase a book; he or she can be influenced by his trusted users when looking for a book, and he or she can consider their options when purchasing a book. So the views or feedback from his own trusted users are useful to the user p for confidence diffusion. As a result, the transitivity algorithm was proposed as a way to link more users. Consequently, a trust-aware recommendation system with many core interests in trust (rather than rating) utility is likely to achieve only minor gains in implementation. Truth, the latest confidence-based models thought of the precise effect of ratings as it were. That is, not much manipulation of the utility of ratings. In their proposed solution, they used Neo4j to manipulate social graphs. They analyzed their conclusions or final recommendations by keeping those test data separate to compare findings with distorted outcomes [2]. It is time to study how the Neo4j database helps the authors in this research to solve the problem. It is a viral graph database server, and it is the Cypher Query Language (CQL). Like another popular one, SNAP, there are numerous other graph databases. Neo4j is more beneficial and is thus utilized to manage graphs and obtain essential graphs in the proposed method. There should be some link between users and item ratings in datasets to suggest something socially. To apply an algorithm to suggest something, coordination between users and items in datasets should be established, as shown in Figure 2, relationships in a socially linked network between users and items. Each blue node knows and rates a few items as the user summons them [2].

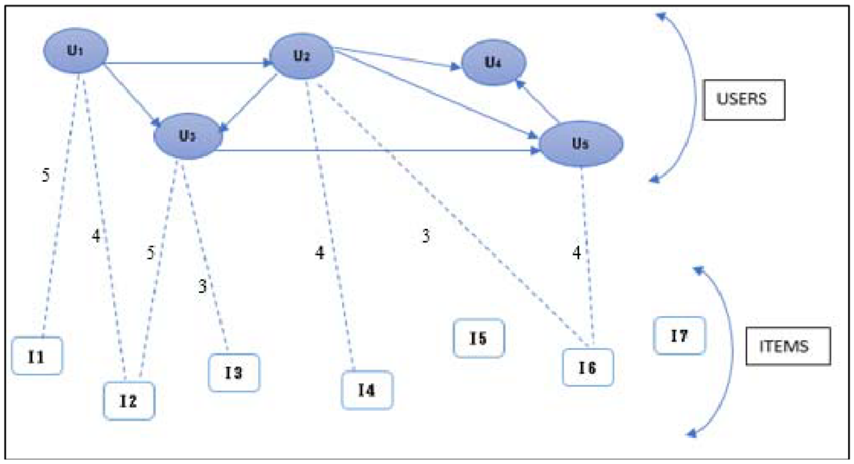


Figure 2: Connections between users and the rating of the item in the dataset [2].

In Figure 2, on the other hand, the dabbed line indicates the rating relation between users and objects, while the strong coordinated line refers to the accompanying link between pairs of clients. The speckled line integer displays the rating value for a particular item provided by the corresponding user. When a user wants recommendations, the recommendation predictions analyze the informal group and user-item rating data, and then they propose highly anticipated rating objects [2].

Hyper-edge is the transitivity concept in graphs. As a social influence, this transitive closure is registered. In social graphs, the principle of hyper-edge social influence is manipulated. Researchers assess the associativity of the nodes of a graph [2]. In a graph, a hyper-edge associate's various adjoining nodes with the help of transitivity [29]. Each node in social graphs is to some degree affected by some other node, either directly or indirectly. Owing to the presence of fast neighbors, this influence on social networks affects a user's leadership alignment [30]. Users may provide or get recommendations for using graphs in these social networks based on this impact. Previously, only neighboring nodes that are immediate to that user were examined to impact the provision of recommendations for social recommendations [2]. In the authors' approach, they have seen the relationship between users through the influence of each other with the help of transitivity, which means that if user A trusts user B and user B trusts user C, then there would be a link between user A and C [2]. So user A can trust the definition of hyper-edge with user C [2]. In graphs, transitive completion is a binary relation that defines the reachability of nodes in social graphs [31]. It would be easier to consider social trust and power instead of merely considering similarities in collaborative filtering. It will be like icing on the cake if we consider all of the potential variations when recommending products to consumers. In the mathematical equation, if a relationship Rn is defined on "`a Rn b`" and "`b Rn c`", users are intransitive closure, then there is a relation between "`a Rn c`". This can be thought of as a matrix containing details about the number of hops needed to get from point x to point z [31].

In the end, the authors in this research suggested a methodology to overcome the problems of formal recommendations by using the idea of hyper-edges on social graphs. Cold start issues and sparsity issues were content and collaborative-based filtering limitations. The user-user matrix and the user-item matrix are used to describe networks of interest and trust. They have improved user confidence by allowing writers to provide more user-user matrix entries and user-item matrix entries, eliminating issues with the cold start and sparsity. To analyze and manipulate social graphs, Neo4j is used. They have found through studies that this proposed method outperforms the state-of-the-art recommendation system. Social tags seem to be very appealing because they can enhance user confidence by providing more tag-based information. By further allocating weights to the nodes in social graphs, we can also adjust this algorithm [2].

There are some though about their work. First, their data was in another data structure, and it was CSV format data, then import those data into the graph. This process allowed them to calculate MAE and RMSE first when it was in CSV and then when after manipulating the data, so then they could have an evaluation for what they did. Their proposed method only predicted the relationship between the users and the items nodes through the Link prediction algorithm. They could use a more efficient algorithm because they will lose more significant similarities by only using one algorithm. Moreover, in the end, their method is an offline method which is not proper for a real-world web-based problem to recommend items to users.

2.2.3 BetterChoice: A Migraine Drug Recommendation System Based on Neo4j

Stark et al. employed a graph-based recommendation system using Neo4j to suggest more effective drugs to patients. This subsection will explore how they model the data into the graph and deal with the recommendation problem. Hospitals have access to vast quantities of patient information. This extensive data set could be used in many ways, such as tailoring patient treatment to their particular characteristics to achieve personalized healthcare. However, there is often a lack of available resources for medical experts to reach the aggregated data from existing databases for a particular issue at the point of care when appropriate [32]. Furthermore, as more drugs, research, and treatment advice become available for medical practitioners, deciding which prescription to offer a patient based on a vast amount of medical history, symptoms, or test results is becoming increasingly tricky [32].

Throughout the world, migraine is a common illness. It not only has a direct impact on people's lives, but it also leads to high costs, such as inability to work or varying sufficient medication-taking times to find the best drug for a patient [32]. Solving the latter factor may help improve patients' lives and reduce the influence of other effects [32]. As a result, this paper introduces a drug recommendation architecture based on the highly scalable native graph database Neo4j. The approach discussed here employs simulated patient data to assist doctors in gaining a better understanding of which treatment best suits a migraine patient, taking into account their unique characteristics. Their assessment demonstrates that the suggested method works as planned. This ensures that only medicines with the highest relevance scores and no reported associations with the patient's conditions, narcotics, or pregnancy can be prescribed [32]. Therefore, to help the decision-making process during therapy, a recommendation system for medicinal application may also be used [32]. The BetterChoice was then developed as a drug recommendation framework for migraines that provides medical facilities with the opportunity to optimize a patient's clinical outcome by using data analytics on a smaller scale than Watson Health [32].

There are two approaches to developing drug recommendation engines: ontology and rule-based approaches, as well as data mining and machine learning, approaches [32]. Then it is time to study the proposed solution. The proposed method is a prototype for providing physicians with information about their patients' opioid treatment results. The outcome of previous prescriptions issued to identical patients is often used. The doctor can choose one of her patients from a database that has been synchronized from the Electronic Medical History and run the algorithm on them [32]. The algorithm takes into account a patient's current health status, medical history, current and previous prescriptions, and allergies [32]. Patients with similar experiences, treatment received, and experience after leaving the facility are grouped together by the engine, which scans the database for patients with similar requirements [32]. The doctor sees the five drugs as a comfort level for the least number of migraine days per month, the number of days, and the number of prescriptions until the algorithm is complete [32]. Based on the prior treatment results, the doctor will now be able to make a more educated and accurate decision about continuing a patient's care. It is important to emphasize that the prototype is not meant to take the doctor's place, but rather to assist her/him [32].

To propose a solution, they have to study what benefit they gained from Neo4j. All relevant information for the recommendation framework will be processed in the graph database Neo4j in order to create a network of patients, drugs, and other components listed above. To guarantee scalability without compromising performance, they chose Neo4j. Neo4j, unlike a traditional relational database, stores data as nodes and edges instead of columns and rows. [32]. In this concept, data points (nodes, vertices) are connected to one another through relationships (edges), allowing the user to create a network of relationships between nodes quickly. Each patient, drug, disease, and allergy will be stored as a single node and connected to other nodes, depending on the case of use. Besides, A node can also have several characteristics in order to describe itself better. With the language Cypher, it will be possible to retrieve coefficients of similarity via graph traversals. All the features mentioned above make Neo4j an excellent prototype tool [32].

Then, the implementation of the recommendation engine should be studied. In general, the purpose of a recommendation system is to construct a user's individual list of items based on their specific characteristics. Therefore, it is possible to use recommendation ratings, prediction algorithms, or other methods to classify items that are not yet known to the individual user but may be necessary to him/her. The user's list was topped by the products with the highest forecast scoring or recommendation ranking [33]. The three key types of recommendation techniques are collaborative filtering (user-based), content-based filtering (item-based), and hybrid approaches. Collaborative filtering is the one used for that project. The guiding principle is to predict objects based on the behavior of a specific group of users, of which the target user is a member. The patient is then assigned to a patient group, and the decision is based on the group's interests. The data is collected, and a prediction score is computed. Recommendations are the top-rated pieces. This method is rational, as it aims to compare patients with similar characteristics [32].

In conclusion, this paper proposes a novel method for a drug recommendation engine based on a graph database. A traditional collaborative filtering algorithm is used to rate patients based on their similarity of characteristics (e.g., allergies, prescriptions, migraine type). Our review revealed that the algorithm makes recommendations as planned. The new drug advisory system will also reduce workload and increase transparency, resulting in stronger physician decision-making and enhanced patient outcomes [32].

Besides the effort and progress they had, they only mentioned that they just used the collaborative-filtering method because the data was imported from another database. As a result, they had historical data of patients. Hence their method is not online. It means that the data is not growing, and new information about a patient or a drug will not be added to their platform when making recommendations. The data that existed in the Neo4J database is static. On the other hand, collaborative-filtering was implemented via a Query-based method which checks some conditions and then calculates the scores or computes the similarity. Like for similarity, their algorithm is finding similarity based on gender. If a patient is male, so then he will be similar to all other men (not all men are similar to each other, or there are more to extract with use of Neo4j). It shows us they didn't model the data set very well in Neo4j, or they didn't benefit from the algorithms like community detection, similarity measurement algorithms, and moreover. More importantly, it was a great job to do all the recommendation process in native Neo4j by Cypher Query, but they could use better algorithms.

2.2.4 Personalized Recommendations using Knowledge Graphs: A Probabilistic Logic Programming Approach

Another significant related work is what Catherine and Cohen proposed in [34] using external knowledge graphs. Benefiting from a general-purpose probabilistic logic platform, they built three various techniques to create a recommendation framework with infra-structured knowledge graphs named ProPPR. The first method benefited just from the relationships in the graph, and it is named EntitySim. Then to boost the model generalization abilities, they extended the former model with benefits from the type of entities, and they named this model TypeSim. The third model they introduced was GraphLF, a latent factor model using the graph advantages that aggregate the robust features of latent factorization through graphs. They applied the previously mentioned methods on two vast and popular datasets, Yelp and MovieLens-100K. They proved if the data in the knowledge graph become spare, the result will be leveraged, and The knowledge graph is a valuable source of data. However, its usefulness decreases as the number of training examples peruse increases. Finally, they demonstrated that their methods achieved a significant improvement in inefficiency.

2.3 Conclusion

In this chapter, first, we studied the background knowledge required to understand this thesis. After defining the recommendation problem, techniques and methods to solve it are presented. Collaborative filtering, content-based filtering, and hybrid recommendation systems are the main solutions discussed in this chapter. In the related work section, we reviewed three recent published methods that solve the recommendation problem. Moreover, how the problem is modeled and how graph databases and knowledge graphs are being exploited have been discussed. In the next chapter, we will present our approach of data collection and knowledge graph construction.

Chapter 3

Data Modeling, Gathering and Preparation

3.1 Introduction

This chapter is about how our application environment works and how our platform can help us create a suitable data model and feed our database, which will result in the construction of the knowledge graph. In Chapter 2, we reviewed some definitions, platforms, algorithms, and libraries. In this chapter, our goal is to explain what is on-demand programming assistance, the benefits of using this application, and its relation to our thesis. Next, we will present different components and sub-components of the system and how they communicate with each other to build the platform. We will also discuss the used data structure and design the data model to formulate human questions and tasks as the database's knowledge graph. We will then analyse our knowledge graph and how it will help us get better results. Thereafter, we will discuss feeding this knowledge graph through a data gathering process to create our recommendation engine. The technologies used to create our on-demand programming assistance platform will be detailed.

3.2 On-Demand Programming Assistance

Every day, computer programmers are facing new challenges. Even senior developers need to update their skills about specific solutions for repetitive problems. It can have different reasons. Sometimes frameworks, operating systems, and libraries are changing all of their structure in major updates, which means it can cause downtime even for stable applications with exclusive standard code. This issue can be categorized into dependency changes. Besides, all senior and junior developers need to know about cutting-edge solutions. In different circumstances, high skilled developers need to increase their code efficiency. To solve those issues, they have to spend hours on the Internet to find the best practices. On the other hand, junior developers or new grad students who are joining the industry don't have enough experience about their responsibilities and tasks they have to deliver. Therefore they also spend even more time on the Internet for their question on websites like Stackoverflow or they might review different solutions and codes on platforms like GitHub to compare and find which one is more appropriate to their problem

IT companies are trying to save the time spent in searching and surfing the web for more appropriate developers' solutions via a web-based application. We call this application on-demand programming assistance. This section will discuss this web-based application and its different modules and the technologies, databases, programming languages, and how they work and communicate together to solve this issue. Figure 3 illustrates our on-demand programming assistance application diagram in detail. All the end-user interaction with the system as asking a question or entering a user story to the system and all process flow to preparing the results are illustrated. Besides, each module interacts with other modules, and their communication to different databases is also shown in Figure 3.

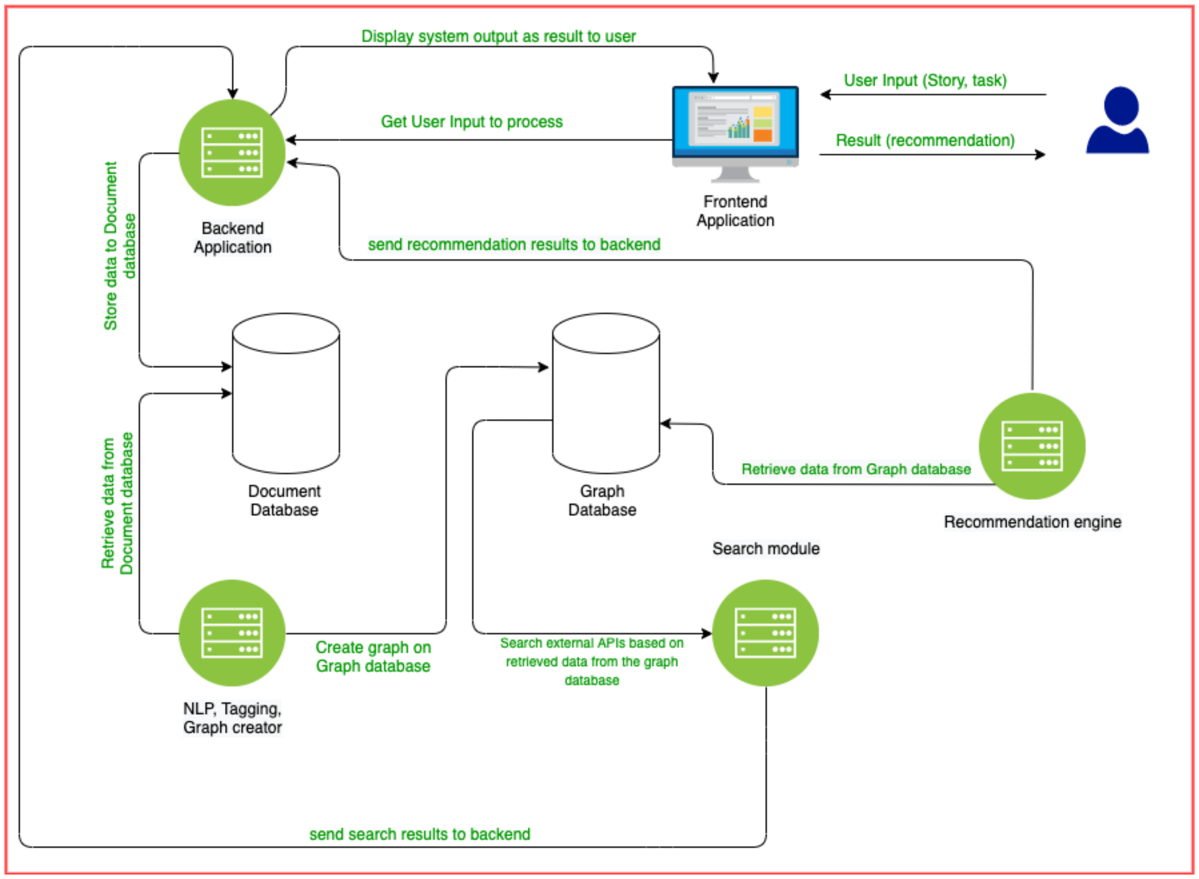

Display system output as result to user
User Input (Story, task)
Get User Input to process
Result (recommendation)
Backend Application
Frontend Application
Store data to Document database
send recommendation results to backend
Retrieve data from Document database
Document Database
Graph Database
Retrieve data from Graph database
Recommendation engine
Search module
Create graph on Graph database
Search external APIs based on retrieved data from the graph database
NLP, Tagging, Graph creator
send search results to backend

Figure 3: The on-demand programming assistance application diagram

As illustrated in Figure 3, this application has four modules: first, Backend application; second NLP and NLU, and the Graph creator module; third, search module; and fourth the recommendation engine. We will explain all of these components in the mentioned order. And Recommendation engine will be discussed in Chapter 4.

3.3 Technologies and Development

Different technologies and tools were used to create this application environment. In the following, we will study the reason why we chose them and how they are helping us achieve a better recommendation solution.

3.3.1 Databases

We have two different databases in this platform, MongoDB and Neo4j. Since dealing with big data is always a big concern for developers, it was decided to use MongoDB as the main database to store user preferences, team and organization information, user authentication information, knowledge object information, and user engagement like mark a library or a code repository or an expert developer as the useful library, repository or expert developer. This information is collected on different collections. Collections are like tables in relational databases like MySQL and MS-SQL; for example, user information is stored on the "`beta_users`" collection. Therefore MongoDB has a huge contribution to this platform. All the modules are communicating and querying this MongoDB and operating CRUD (create, read, update, delete) operation on its different collections. The second database we will investigate is Neo4j. Because Neo4j is a graph database, which means data is represented in the form of nodes and edges, and it has built-in graph algorithms, it is beneficial for the problems like recommendation systems. First, because traversing the graph even in high-scale applications when there are too many nodes and edges will be efficient, like when dealing with a small graph. Also, because Neo4j is a native graph database, it is efficient to create a knowledge graph on this database and gain the graph data structure's required benefits.

3.3.2 Data Modeling and Database Design

Semantic presentation of data that can show information in the form of graphs will be represented in the below image, Figure 4. This will be our knowledge graph data model, and it is just a pattern to illustrate how data will be gathered in the Neo4j database. This form of representing has a huge benefit, which will cause explaining information and facts in a more understandable way for humans.

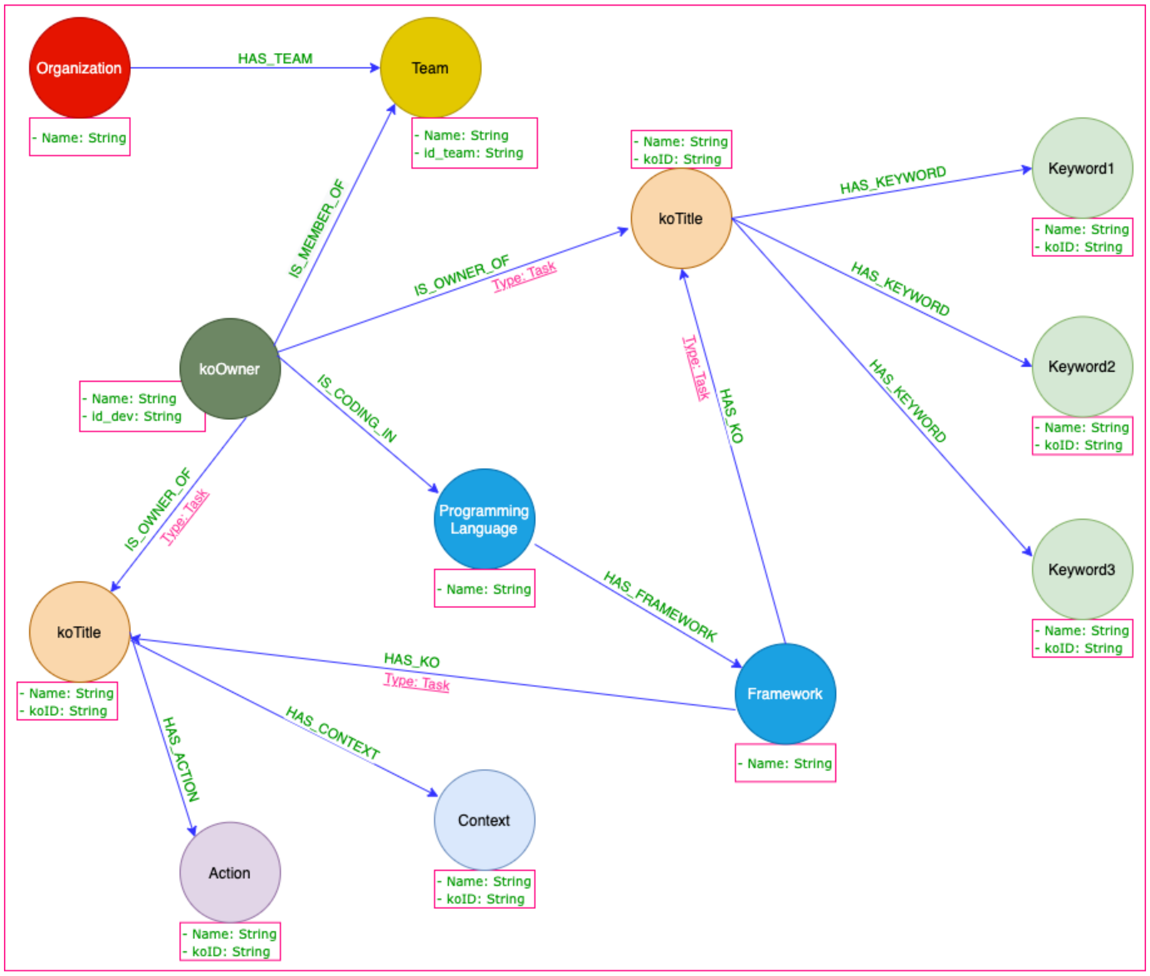


Figure 4: The Knowledge graph data model on the Neo4j

From Figure 4, we can see different entities, nodes, and relationships (i.e., edges). An organization node is referring to the company or organization that the user is working with. Next, a team node is referring to the team in the organization to which the user belongs, such as the Backend team or the Frontend team. The koOwner node is pointing to the current user's username. Its name is coming from Knowledge Object Owner (a Knowledge Object in our platform is equal to a user story, task, or question). Each user can have more than one programming language and framework. Frameworks are dependent on the programming language; for example, the Python programming language in this platform has two frameworks, Flask and Django. Also, a knowledge object can have only a programming language, and it can still have meaning without a framework. Other than the Programming Language node and the Framework node, if a knowledge object type is a Story or a Task, it can have other relations or nodes. For example, if a knowledge object is a Task, it will have Keyword nodes, and keywords can be more than one node. The order from Keyword nodes $Key_0, \ldots, andKey_n$ shows their degree of importance in the Task body like key0, key1, and key2, where key0 has a higher degree of importance than key1, and key1 has a higher degree of importance than key2. On the other hand, if the knowledge object is a Story, the Action node and the Context node will be created and connected to the knowledge object head, the koTitle node. The Action node is the most important verb or word in the user story. And the Context is the node which is defining the user story's circumstances and conditions. Finally, the type of the knowledge objects can be defined on a relationship from the koOwner to the koTitle as an entity called Type or from another relationship from the Framework to the koTitle again as an entity labeled Type. A real-world graph database that is fed with real data is shown in Figure 5.

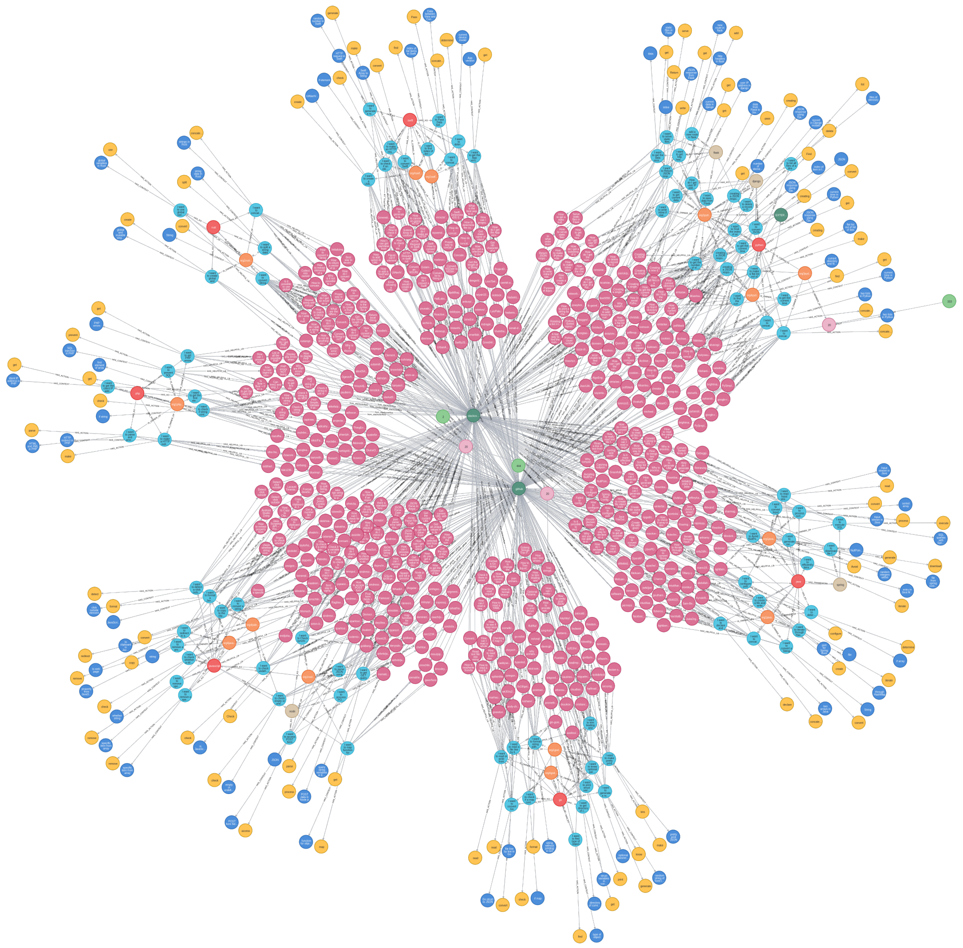


Figure 5: An example of knowledge graph fed with real data

In Figure 5, edges are relationships between nodes which have the following semantics. Dark green nodes are showing source nodes; light blue nodes are representing the Knowledge Object nodes; light red nodes are representing library nodes; pink nodes are denoting teams; light green nodes are illustrating organizations; orange nodes are showing developers; brown nodes are showing the frameworks; yellow nodes are displaying the actions for each Knowledge Object; dark blue nodes are showing the context of each Knowledge Object; and red nodes are depicting the programming languages.

3.3.3 Web Frameworks

We have two different web frameworks. First, we have to talk about Flask because the core module and search module and communication with Neo4j are happening using this powerful framework. It was chosen from other web frameworks because it is highly customizable, and in the future, it can be used for large-scale, and it was easy to create REST APIs. Then, NodeJS Express is the other web framework chosen to create Knowledge Objects on MongoDB and update its information on the database.

3.3.4 The Third-Party Application for NLP and NLU

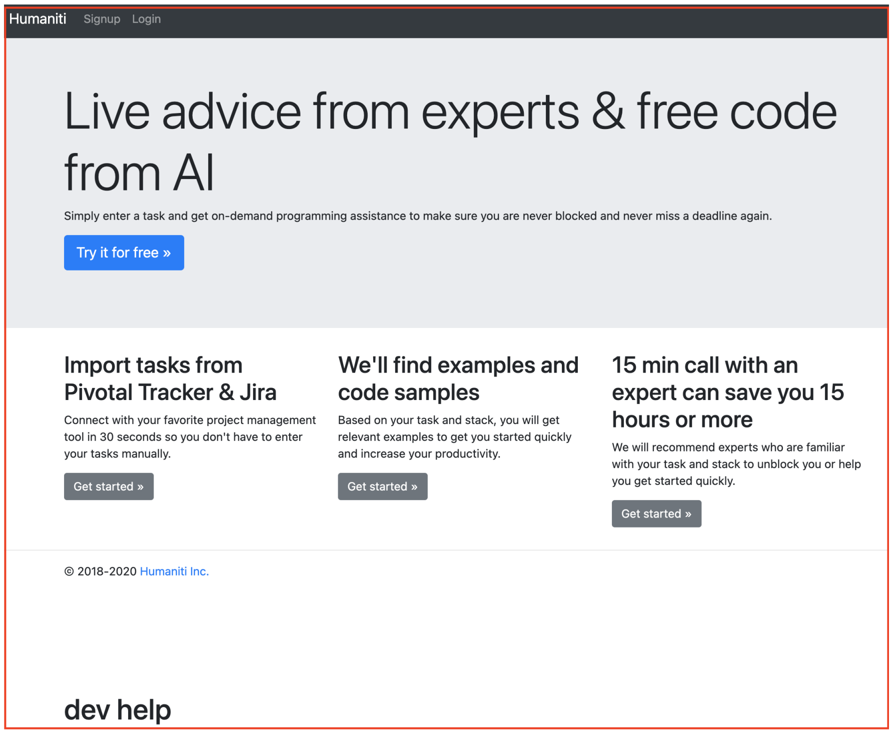
This platform uses two different third-party applications for entity extraction and matching entities with user data. First for the entity extraction "`Dandelion API`" [35]. And for the matching entities "`spacy`" [36]. Using these third-party applications, the action and the context for each knowledge Object will be extracted.

3.4 On-Demand Programming Assistance Modules

On-demand programming assistance is a platform to help developers to develop better and faster code. They can ask their questions on this platform. They can also import their agile system user story or task to find the best solution for them. In the result, they can see discussion on the Stackoverflow. And they can see code examples on the GitHub code repositories. Also, they can find other developers who were working on the same issues or a related issues. Hence, they can communicate through On-demand programming assistance via email and ask their senior or experienced developers questions to reach this application goal, making the developer work faster and more efficiently. In the following sub-sections, we will discuss these On-demand programming assistance modules.

3.4.1 Backend Application and User Interface

As the heart of the On-demand programming assistance ecosystem, a module called Quarterback Python performs the leading role in communicating with users (both developers, team leaders, and admins), receiving and storing data to the multiple databases from web pages inputs, and demonstrating the results on the user interface. It is clear from the name that Quarterback Python is written in Python version 3, and the framework chosen to build the main structure was the Flask web framework. There are different options for web frameworks like Django and ..., but Flask was more customizable. Because the management team wanted to make the web application closer to the customer needs, the development team could create applications easier and faster with the Flask. The Vue.js is again chosen between different frontend frameworks due to its fast and easy development system, and in addition, it was a super powerful tool for the user interface. Figure 6 is the main page of the application, which will illustrate after running this application. Neo4j and MongoDB databases are essential to launching this application.



Live advice from experts & free code from AI

Simply enter a task and get on-demand programming assistance to make sure you are never blocked and never miss a deadline again.

Import tasks from Pivotal Tracker & Jira

Connect with your favorite project management tool in 30 seconds so you don't have to enter your tasks manually.

We'll find examples and code samples

Based on your task and stack, you will get relevant examples to get you started quickly and increase your productivity.

15 min call with an expert can save you 15 hours or more

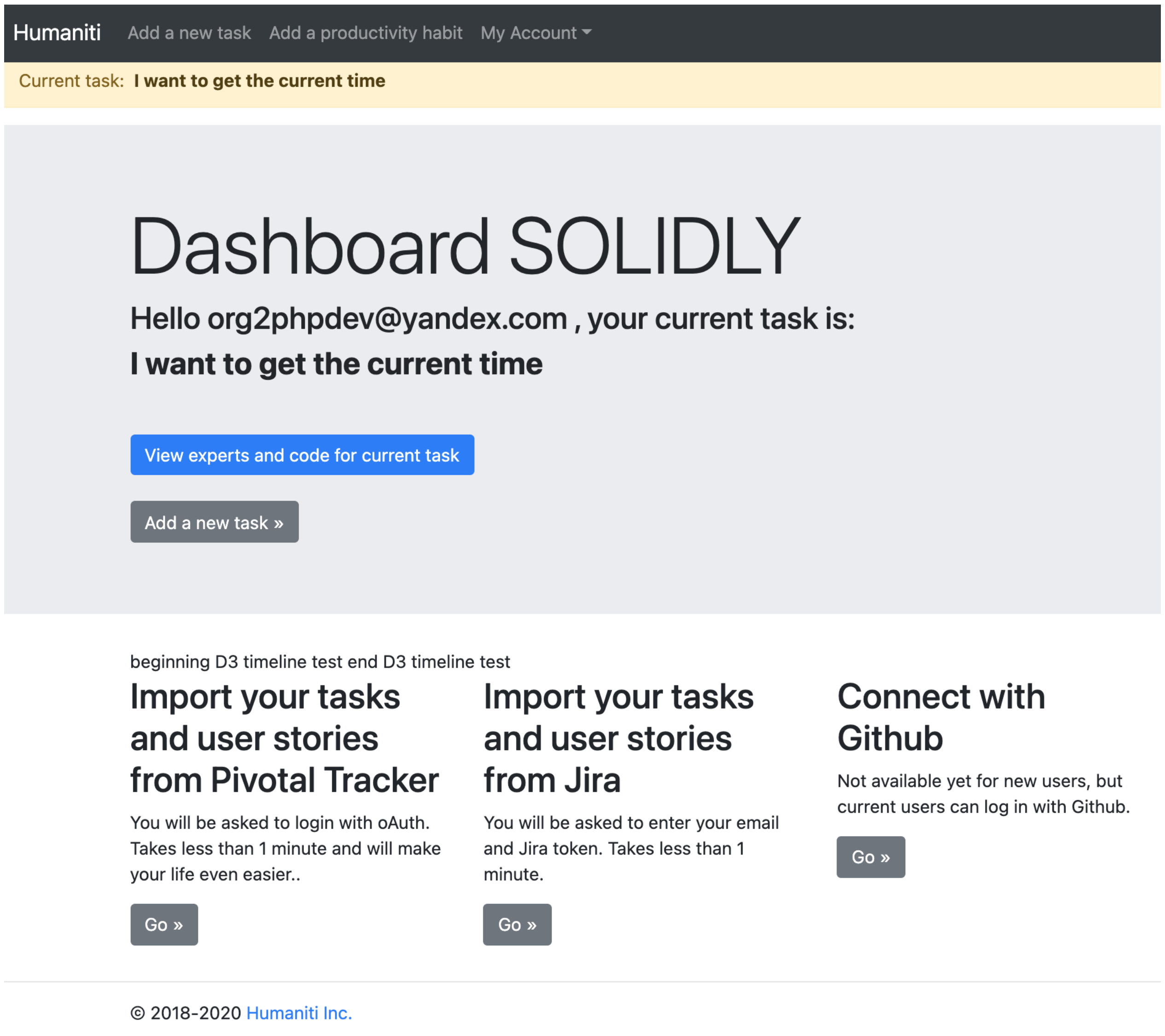
We will recommend experts who are familiar with your task and stack to unblock you or help you get started quickly.

© 2018-2020 Humaniti Inc.

dev help

Figure 6: The On-demand programming assistance web-applications' main page.

After login to the On-demand programming assistance, the user can navigate the "Add Task" tab figure 7. Here users should use a title for entering task or story title, which is the user's problem (here, by the user. We are pointing to a developer who is using our web application, On-demand programming assistance, to find solutions or help). Now we have to take a moment to explain Task and Story. In an Agile system, Story is the smallest component that should be delivered in the current agile sprint. A story can have sub-modules, which will be called Task. And these sub-modules will define how a story will be completed [5].



Current task: **I want to get the current time**

Dashboard SOLIDLY

Hello org2phpdev@yandex.com , your current task is:

I want to get the current time

beginning D3 timeline test end D3 timeline test

Import your tasks and user stories from Pivotal Tracker

You will be asked to login with oAuth. Takes less than 1 minute and will make your life even easier..

Import your tasks and user stories from Jira

You will be asked to enter your email and Jira token. Takes less than 1 minute.

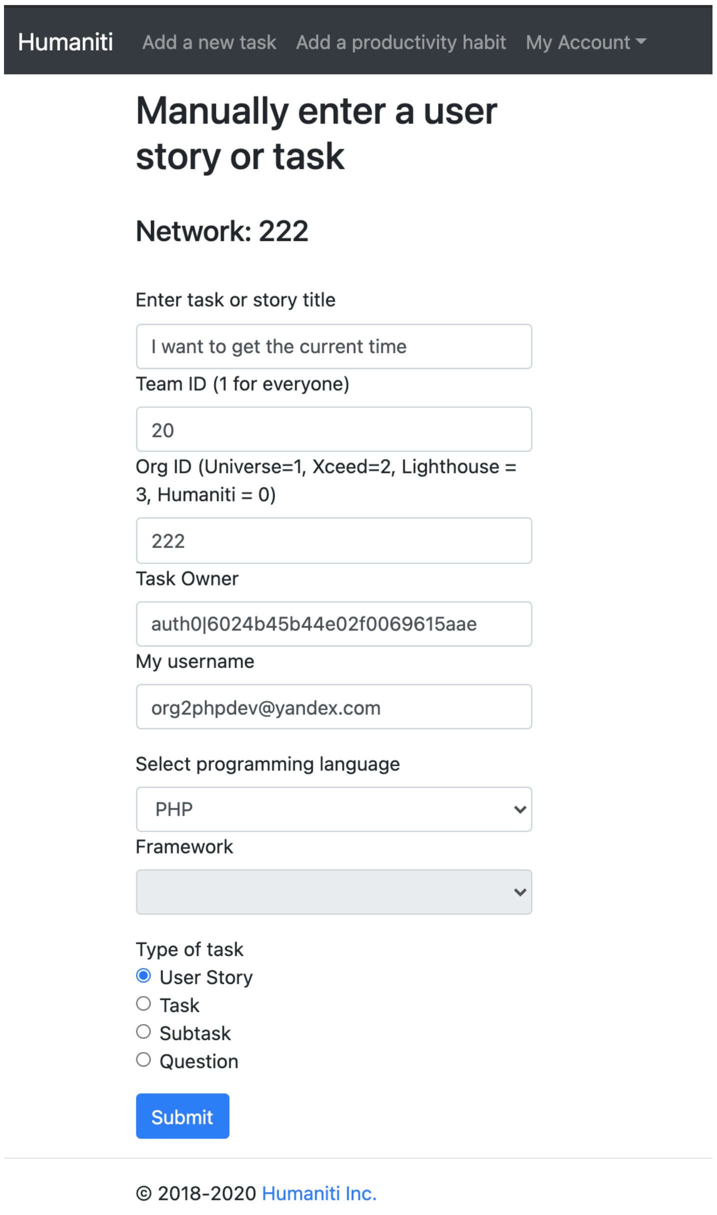
Connect with Github

Not available yet for new users, but current users can log in with Github.

© 2018-2020 Humaniti Inc.

Figure 7: The on-demand programming assistance main page.

For the test scenario (see Figure 8), the problem defined is “I want to get the current time”, the programming language is Python, and the framework is Flask. In a real-world use case, these options can take different values.



Manually enter a user story or task

Network: 222

Enter task or story title: I want to get the current time

Team ID (1 for everyone): 20

Org ID (Universe=1, Xceed=2, Lighthouse = 3, Humaniti = 0): 222

Task Owner: auth0|6024b45b44e02f0069615aae

My username: org2phpdev@yandex.com

Select programming language: PHP

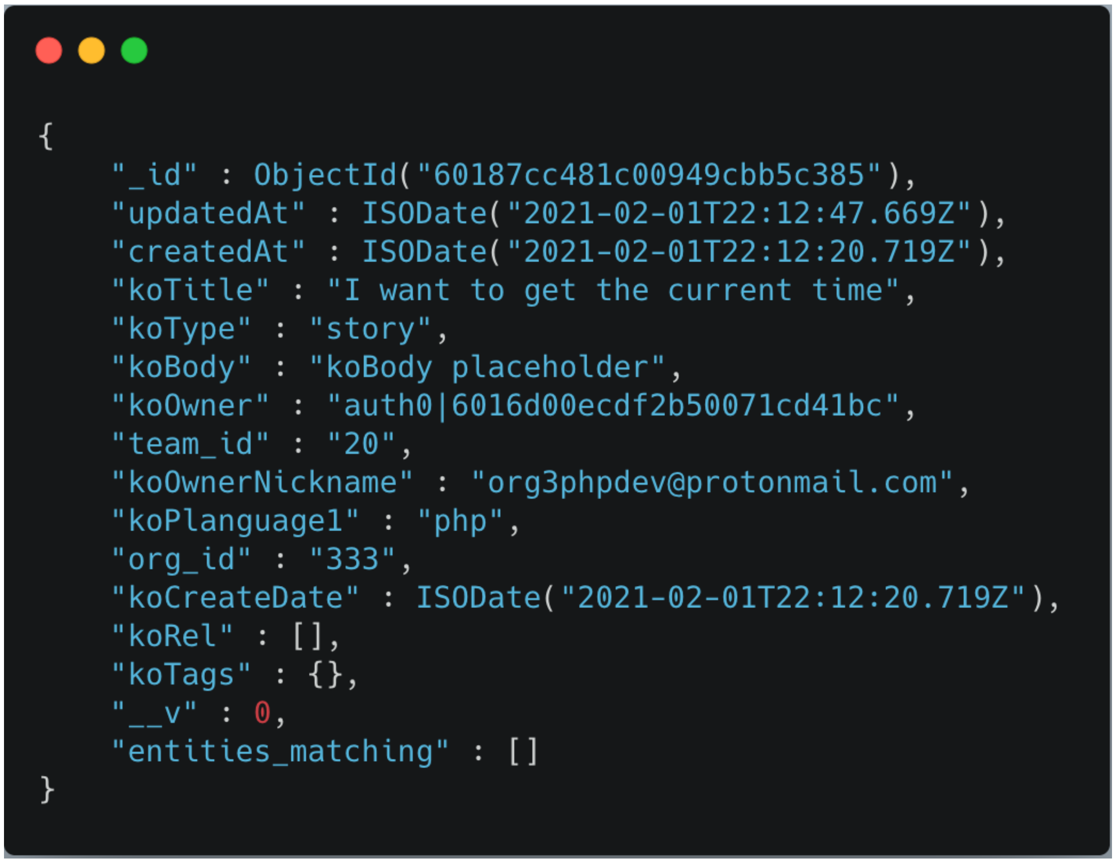
Framework:

Type of task: [✓ User Story] Task Subtask Question

© 2018-2020 Humaniti Inc.

Figure 8: Adding a new Knowledge Object on the web application.

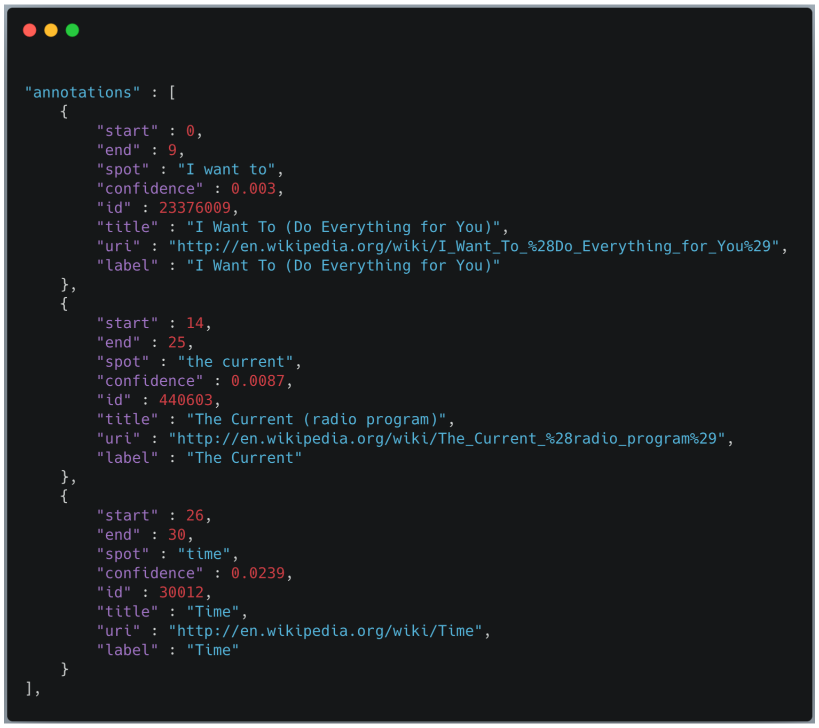
Besides, by querying kos collection in MongoDB, we can see the knowledge object information in detail. Information like task title, Knowledge Object, koOwner, and more. Also, in the query illustrated in Figure 9. This information should be the same as the data represented on the web app.



```
{
    "_id" : ObjectId("60187cc481c00949cbb5c385"),
    "updatedAt" : ISODate("2021-02-01T22:12:47.669Z"),
    "createdAt" : ISODate("2021-02-01T22:12:20.719Z"),
    "koTitle" : "I want to get the current time",
    "koType" : "story",
    "koBody" : "koBody placeholder",
    "koOwner" : "auth0|6016d00ecdf2b50071cd41bc",
    "team_id" : "20",
    "koOwnerNickname" : "org3phpdev@protonmail.com",
    "koPlanguage1" : "php",
    "org_id" : "333",
    "koCreateDate" : ISODate("2021-02-01T22:12:20.719Z"),
    "koRel" : [],
    "koTags" : {},
    "__v" : 0,
    "entities_matching" : []
}
```

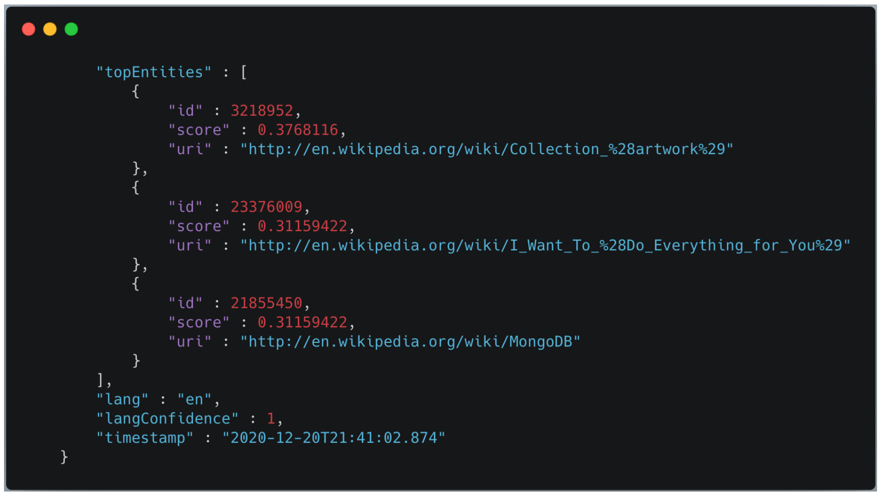
Figure 9: The mentioned Knowledge Object information in MongoDB.

There is more information about this new knowledge object in the database. We have to talk about them. First, there is a set of annotations which represents information like “`collection`” and “`mongodb`” from user input, Figure 10. Second, “`topEntities`” is extracted from the knowledge object, again like “`collection`” and “`mongodb`” but in numerical form and Wikipedia like to each entity Figure 11. The third and last is “`entities_matching`” which matches extracted entities from knowledge objects to our required entities Figure 12, for example as mentioned before, our web application required information like developer information, which is the username, use the user id, and more or programming language and so on.



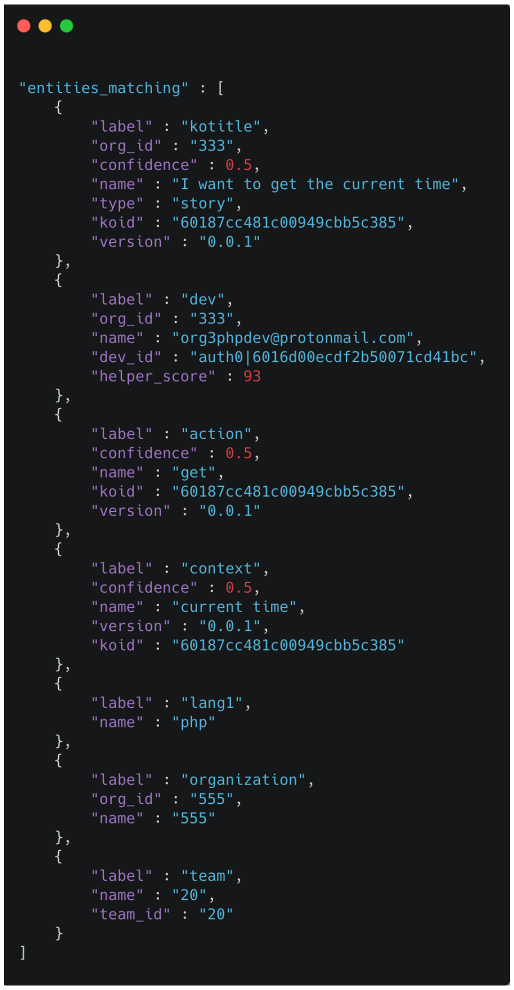
```
"annotations" : [
    {
        "start" : 0,
        "end" : 9,
        "spot" : "I want to",
        "confidence" : 0.003,
        "id" : 23376009,
        "title" : "I Want To (Do Everything for You)",
        "uri" : "http://en.wikipedia.org/wiki/I_Want_To_%28Do_Everything_for_You%29",
        "label" : "I Want To (Do Everything for You)"
    },
    {
        "start" : 14,
        "end" : 25,
        "spot" : "the current",
        "confidence" : 0.0087,
        "id" : 440603,
        "title" : "The Current (radio program)",
        "uri" : "http://en.wikipedia.org/wiki/The_Current_%28radio_program%29",
        "label" : "The Current"
    },
    {
        "start" : 26,
        "end" : 30,
        "spot" : "time",
        "confidence" : 0.0239,
        "id" : 30012,
        "title" : "Time",
        "uri" : "http://en.wikipedia.org/wiki/Time",
        "label" : "Time"
    }
],
```

Figure 10: Knowledge object annotations information in the kos collection



```
        "topEntities" : [
            {
                "id" : 3218952,
                "score" : 0.3768116,
                "uri" : "http://en.wikipedia.org/wiki/Collection_%28artwork%29"
            },
            {
                "id" : 23376009,
                "score" : 0.31159422,
                "uri" : "http://en.wikipedia.org/wiki/I_Want_To_%28Do_Everything_for_You%29"
            },
            {
                "id" : 21855450,
                "score" : 0.31159422,
                "uri" : "http://en.wikipedia.org/wiki/MongoDB"
            }
        ],
        "lang" : "en",
        "langConfidence" : 1,
        "timestamp" : "2020-12-20T21:41:02.874"
    }
```

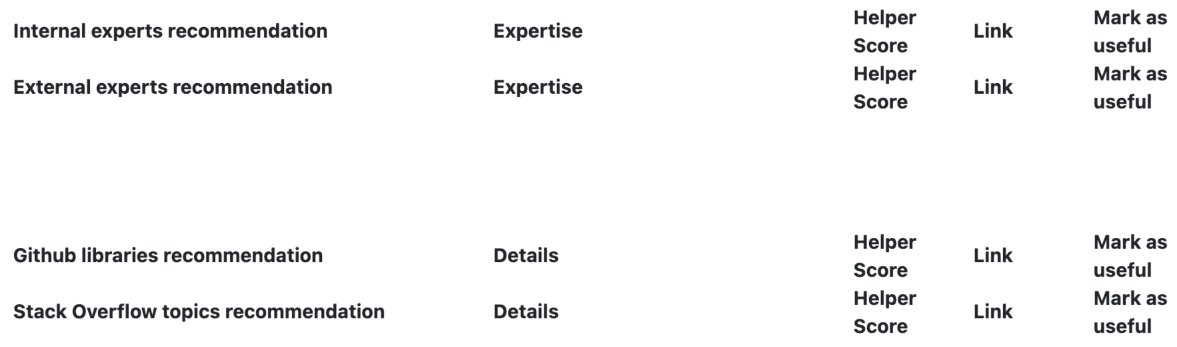
Figure 11: Knowledge object top entities information in MongoDB.



```
"entities_matching" : [
    {
        "label" : "kotitle",
        "org_id" : "333",
        "confidence" : 0.5,
        "name" : "I want to get the current time",
        "type" : "story",
        "koid" : "60187cc481c00949cbb5c385",
        "version" : "0.0.1"
    },
    {
        "label" : "dev",
        "org_id" : "333",
        "name" : "org3phpdev@protonmail.com",
        "dev_id" : "auth0|6016d00ecdf2b50071cd41bc",
        "helper_score" : 93
    },
    {
        "label" : "action",
        "confidence" : 0.5,
        "name" : "get",
        "koid" : "60187cc481c00949cbb5c385",
        "version" : "0.0.1"
    },
    {
        "label" : "context",
        "confidence" : 0.5,
        "name" : "current time",
        "version" : "0.0.1",
        "koid" : "60187cc481c00949cbb5c385"
    },
    {
        "label" : "lang1",
        "name" : "php"
    },
    {
        "label" : "organization",
        "org_id" : "555",
        "name" : "555"
    },
    {
        "label" : "team",
        "name" : "20",
        "team_id" : "20"
    }
]
```

Figure 12: The entities_matching information for the current Knowledge object in the MongoDB.

First, according to Figure 13, the recommendation section for experts recommended for the problem. These experts are developers who asked this problem/question. These developers can be internal developers who are defined developers working in the current user's company or external developers who exist on our knowledge graph, for example, volunteer developers who are using our platform to find their answers or look for solutions. Second, there will be recommended libraries in the same section, which are the GitHub repositories or Stackoverflow discussion topics related to the current knowledge object from the recommendation engine. Because at this time there is no other user except our current user, this section is empty, but just for now.



Internal experts recommendation	Expertise	Helper Score	Link	Mark as useful
External experts recommendation	Expertise	Helper Score	Link	Mark as useful

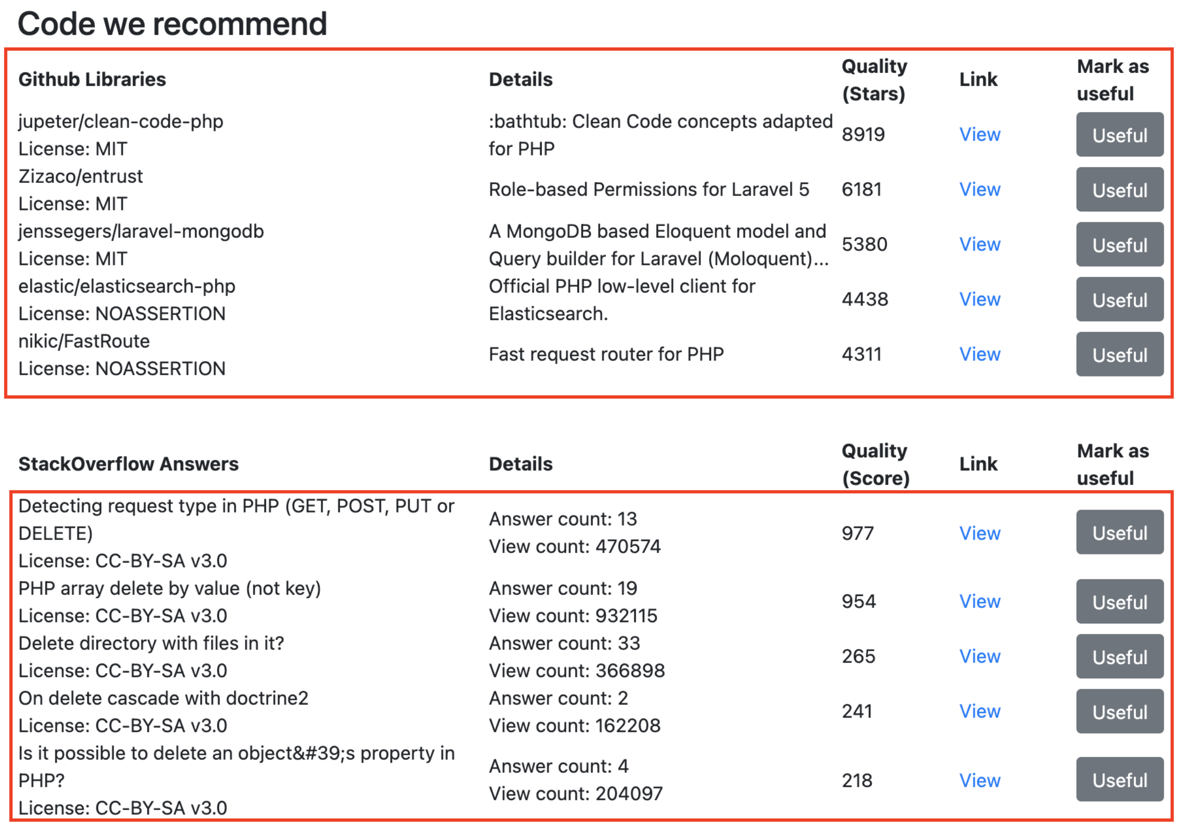
Github libraries recommendation	Details	Helper Score	Link	Mark as useful
Stack Overflow topics recommendation	Details	Helper Score	Link	Mark as useful

Figure 13: The empty recommendation section for the first Knowledge Object.

The second section of the result page is for illustrating search results on the GitHub for codes that are close to the problem and Stackoverflow for similar questions/problems. We will discuss the search module in detail, which has been named Fortytwo.

As denoted in Figure 14, the second section in the result page has two main parts. The first part is the results of querying from GitHub search API. This part is from Github public repositories. And it describes each code repository with three features: repository name and description from the repository readme file. And represent how many stars other Github users gave to the mentioned repository. The second part, as it is represented in Figure 14, is from Stackoverflow and it is showing questions and answers close to the problem the current user asked in our platform. The Stackoverflow section also has four features, including the topic title, details which contain answers count, and view count to the question. Then topic score, which is given by our On-demand programming assistance, and then the topic's link to view.

For both GitHub and Stackoverflow section, there is an option called Useful if a user clicks on this button, it will count as a positive signal, and useful marked code or topic will be added to our Graph database. It will be used by the recommendation module, which is responsible for the recommendation task. We will talk more about this in the chapter four.


Code we recommend

Github Libraries	Details	Quality (Stars)	Link	Mark as useful
jupeter/clean-code-php License: MIT	:bathtub: Clean Code concepts adapted for PHP	8919	View	Useful
Zizaco/entrust License: MIT	Role-based Permissions for Laravel 5	6181	View	Useful
jenssegers/laravel-mongodb License: MIT	A MongoDB based Eloquent model and Query builder for Laravel (Moloquent)...	5380	View	Useful
elastic/elasticsearch-php License: NOASSERTION	Official PHP low-level client for Elasticsearch.	4438	View	Useful
nikic/FastRoute License: NOASSERTION	Fast request router for PHP	4311	View	Useful

StackOverflow Answers	Details	Quality (Score)	Link	Mark as useful
Detecting request type in PHP (GET, POST, PUT or DELETE) License: CC-BY-SA v3.0	Answer count: 13 View count: 470574	977	View	Useful
PHP array delete by value (not key) License: CC-BY-SA v3.0	Answer count: 19 View count: 932115	954	View	Useful
Delete directory with files in it? License: CC-BY-SA v3.0	Answer count: 33 View count: 366898	265	View	Useful
On delete cascade with doctrine2 License: CC-BY-SA v3.0	Answer count: 2 View count: 162208	241	View	Useful
Is it possible to delete an object&#39;s property in PHP? License: CC-BY-SA v3.0	Answer count: 4 View count: 204097	218	View	Useful

Figure 14: Search module result for two different sources, Github and Stackoverflow.

Let's take a step back and study what is happening and how this result is represented here. If you remember it has been cited, the data will be stored on the graph database after clicking on the Submit knowledge object. The below Cypher Query shows data on the graph database, and the result will be mentioned in the following Figure 15.

```
MATCH (a:action)<-[:HAS_ACTION]-(k)-[:HAS_CONTEXT]->(c:context),
(k)<-[:IS_OWNER_OF]-(d:dev)-[:IS_CODING_IN]-(p:lang1),
(d)-[:IS_MEMBER_OF]->(t:team)<-[:HAS_TEAM]-(o:organization),
WHERE k.koid="60187cc481c00949cbb5c385"
RETURN a, k, c, p, d, t, o;
```

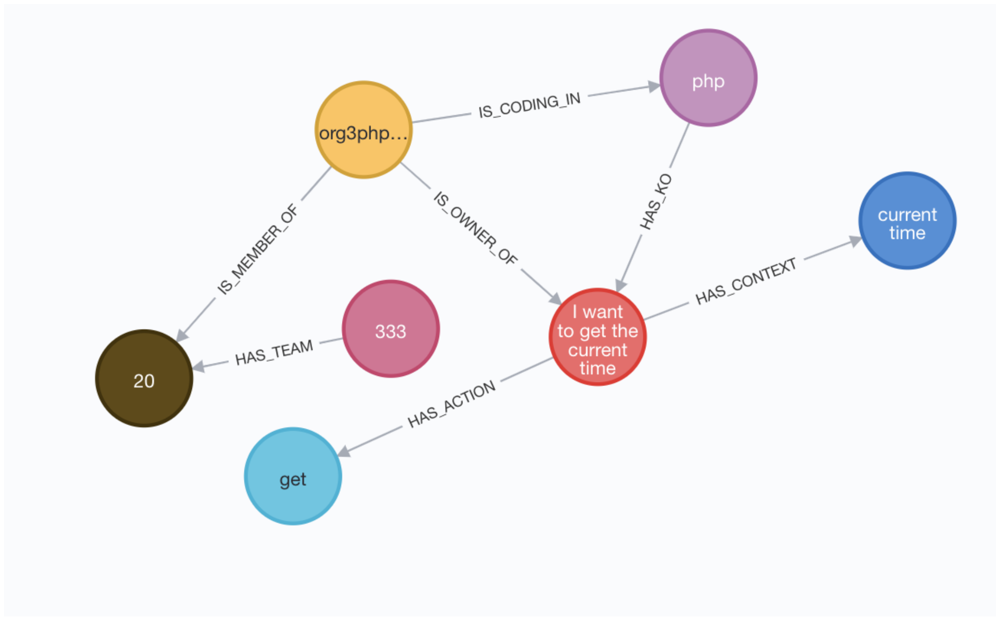


Figure 15: Illustrated knowledge object in the Neo4j graph database.

As noted before, the Neo4j graph database has been chosen by the development team to make use of graph-structured data structure and native graph-based algorithms. Knowledge object graph form will be like Figure 15. According to the data model in Figure 4 for this knowledge object also we will have the same relationships as edges and the same data in the shape of nodes. Because it is a user story and it will have only Story parts like "koTitle". But if we add a Task for this user application, we will create the required nodes for the data model task.

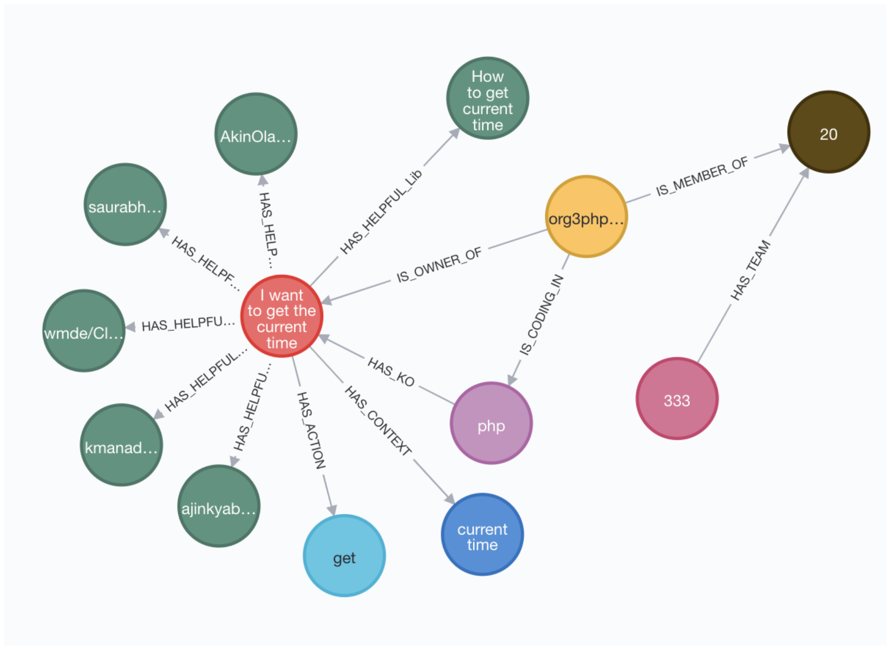


Figure 16: Illustrated knowledge object with useful libraries.

To continue testing our web application, data flow, and data gathering, we will click on a useful button for the first library from Github and the first topic from Stackoverflow in Figure 14 and then we will check the changes in the Neo4j database in Figure 14.

This useful button is significantly important and vital for the recommendation engine because it will be counted as a satisfaction signal for using a repository on the GitHub, question topic on Stackoverflow or another developer existed on the system which helps current users about the problem which we call it knowledge object. Hence, the recommendation engine can improve its suggestion through user previous preferences Figure 16.

To communicate with MongoDB for creating knowledge objects from user input and add information like add tags to the knowledge object, another backend application written in NodeJS programming language has been used.

This module is called Quarterback NodeJS. Furthermore, for other operations like updating data in the MongoDB database or delete we will use this application through web APIs on port 3000, for example, http://localhost:3000/ko/.

3.4.2 NLP, NLU, and Graph Creator Module

Another critical module in the system is the component responsible for the natural-language processing, natural-language understanding, and communicating and creating knowledge objects on the graph database. This module is called Kali, combined from three important submodules: natural-language understanding listener, Knowledge object creator, and Knowledge object search. We will discuss all of these three modules in the following.

Natural-Language Understanding Listener

After a client asks a question, it will be stored in MongoDB as a knowledge object, and it will be like Figure 9. Then instantly, an application will be triggered to grab that knowledge object and add tags on it through "`Dandelion API`" [35] also, via mentioned API, this submodule will add "`annotations`" and data collection in MongoDB will have data like in Figure 10.

Knowledge Object creator

Next, when the natural-language understanding listener creates the "`tags`" and "`annotations`", There is still one process to talk about before inserting data on the graph database creating "`topEntities`" for knowledge object. These entities are information like the programming language and then make "`entities_matching`" for required fields like information about user authentication information, programming language and framework organization information and more via using "`spacy`" library [36]. For studying all data fields you can check Figure 11 and Figure 12.

Search Knowledge Objects and graph database

The last submodule in the Kali module set is the search module. This application is responsible for querying graphs and provides some information for other modules like Fortytwo search. Or sometimes, in the platform, it is required to identify the knowledge object if it is a "`task`", or it is a "`user story`" to be distinguished from this application. This is an API-based application that other applications can communicate with over its rest APIs on http://localhost:4000/.

3.4.3 Search Module

After a knowledge object, that has been created, and the client wants to see the solutions for it, an API will be called to search for GitHub and Stackoverflow APIs. The application responsible for this search is called Fortytwo. It was denoted in the Figure 14 this search module output will be represented in the result page for the user. If user check the repositories from GitHub and discussions from Stackoverflow and they find these repositories or discussions useful, they can mark them useful. Therefore those useful repositories and discussions will be stored on MongoDB and Neo4j databases (see Figure 16 for instance).

3.5 Investigating the Knowledge Graph

Here we will go through more details of the knowledge graph, which is constructed via extracted data from user input. The graph will have useful libraries and discussions from two sources, namely Github and StackOverflow, regarding the graph's aforementioned manipulation. And also, it will have nodes for expert developers. And relationships between these libraries and expert nodes with old nodes. Therefore, the new data model will be like Figure 17.

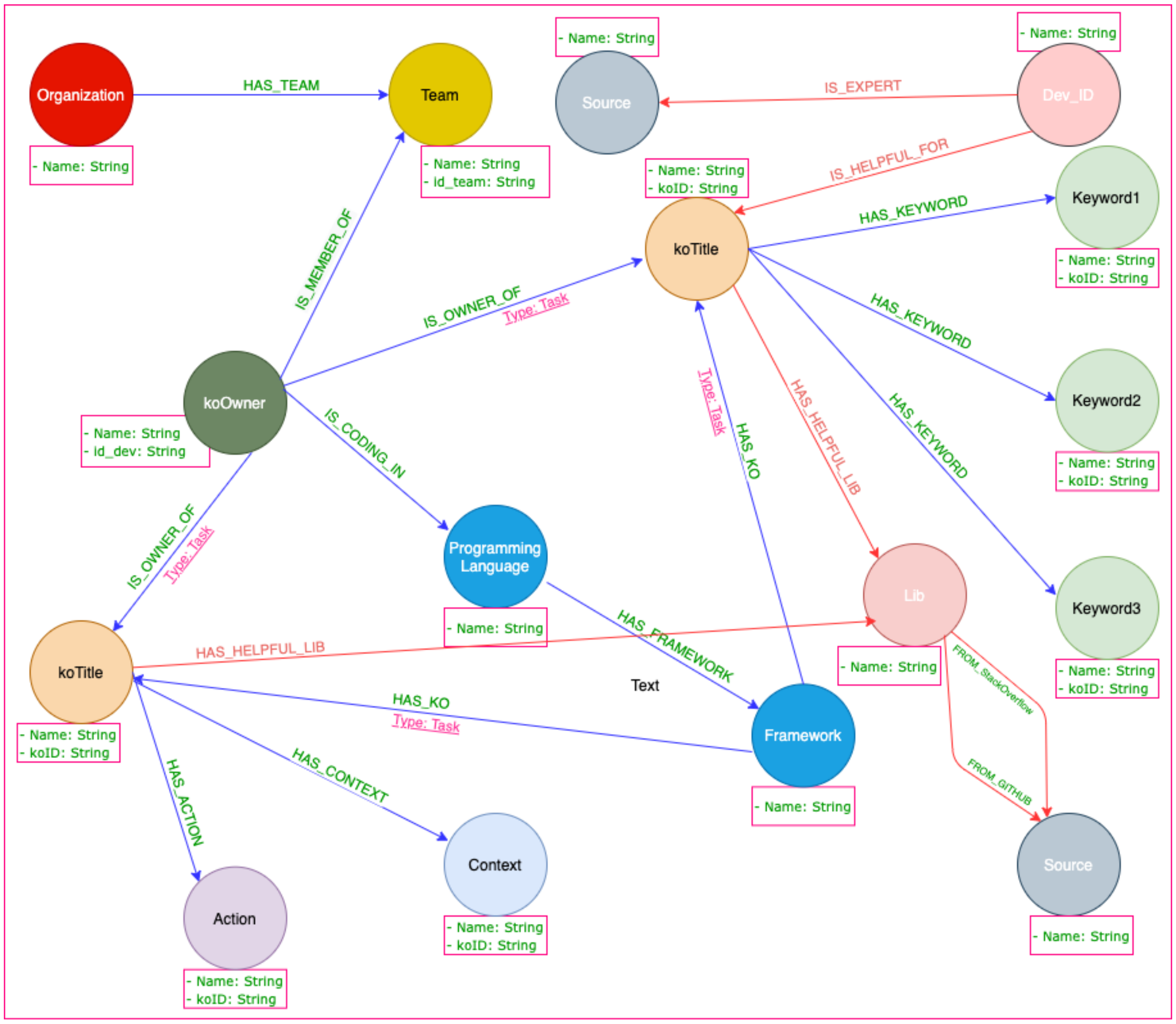


Figure 17: The new data model, after adding useful libraries and expert developers nodes.

Now, it's time to traverse the knowledge graph to gain more information about the data stored in the graph database. First, we will check the graph database to find all the organizations exited in the graph below Cypher Language Query (CQL). This Cypher Query is searching the graph for all nodes which have label property with their value is equal to the organization.

```
MATCH (o:organization) RETURN o
```

And results will be represented in Figure 18. As it is shown in Figure 18, now we have three different organizations, which are named 2, 333, 444. These names are just hypothetical names chosen to create the dataset for implementing this thesis. But this dataset creation process is an ongoing process, which means data will grow, and the graph will get larger than what it looks like now.

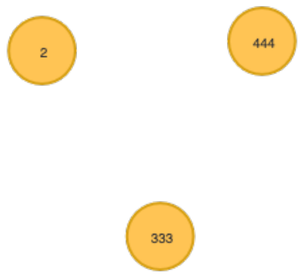


Figure 18: Finding all existing organizations in the graph

Next, we will traverse the Knowledge Graph in two more steps; we can see these aforementioned organizations with their teams and team members, which is illustrated in Figure 19. Red nodes represent developers, light blue nodes represent teams, and yellow nodes are denoting the organization information.

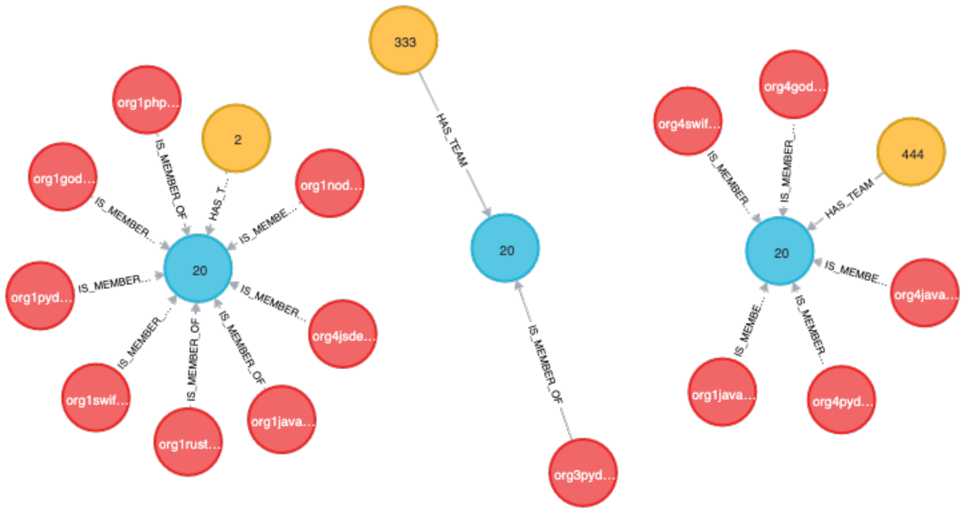


Figure 19: Finding organizations existed in the graph with their teams and members of those teams.

You can find in Figure 19 again there are three organizations. The first organization, which is named 2, and its node is colored yellow. This organization has a team with team ID 20, and the team ID node has a blue color. This team also has eight developers with red nodes and a directed edge from developer node to team node, which has the property key of the "IS_MEMBER_OF".

There is more interesting information in the graph. Information like the different programming languages we have on the graph is denied in Figure 20 with below Cypher Query.

```
MATCH (progLang:lang1) RETURN progLang
```

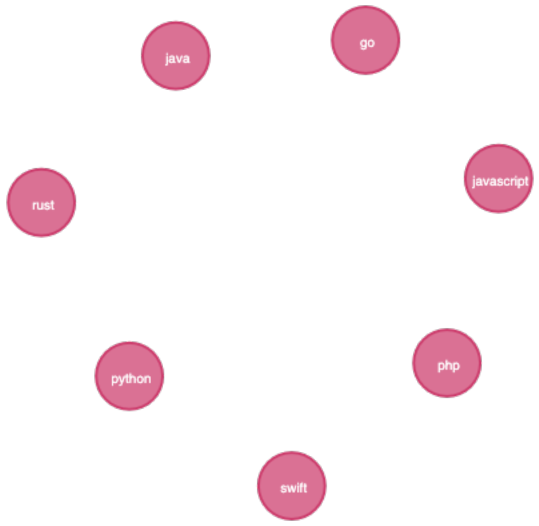
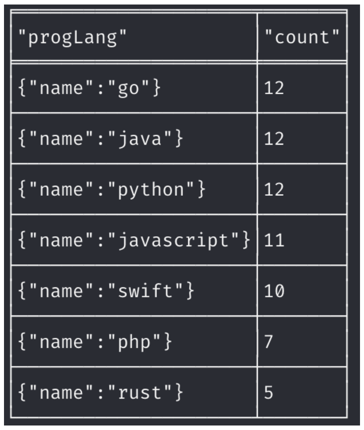


Figure 20: Finding all existing programming languages in the graph.

As we know from the aforementioned data model, the programming language has a relationship with the user story node, the KoTitle node. The relationship is "HAS_KO", and the edge direction is from the programming language node to the KoTitle node. Furthermore, if we count how many relationships each programming language with the value of "HAS_KO" has and compare the result, we can say the programming language with the higher edge count number is the most popular one in the graph. The Cypher Query for this operation is similar to the below query, and the result is shown in Figure 21. From the figure, we can notice that there are three programming languages with twelve edge which have "HAS_KO" value, and these three programming languages are the most popular in the knowledge graph. To explain how we retrieve this information by Cypher query, it is first will find all nodes which have "lang1" and also have "HAS_KO" relationship, then it will count relationship number and then it will sort the results.

```
MATCH (progLang:lang1)-[rel:HAS_KO]->(k:kotitle)
RETURN progLang, count(rel) as count ORDER BY count DESC
```



"progLang"	"count"
{"name":"go"}	12
{"name":"java"}	12
{"name":"python"}	12
{"name":"javascript"}	11
{"name":"swift"}	10
{"name":"php"}	7
{"name":"rust"}	5

Figure 21: Demonstrate all existing programming languages in the graph and find which ones are more popular than other programming languages.

Also, the graph representation for Figure 21 will be represented in Figure 22. This is graph database capability, which will construct the smaller graph to traverse and find answers.

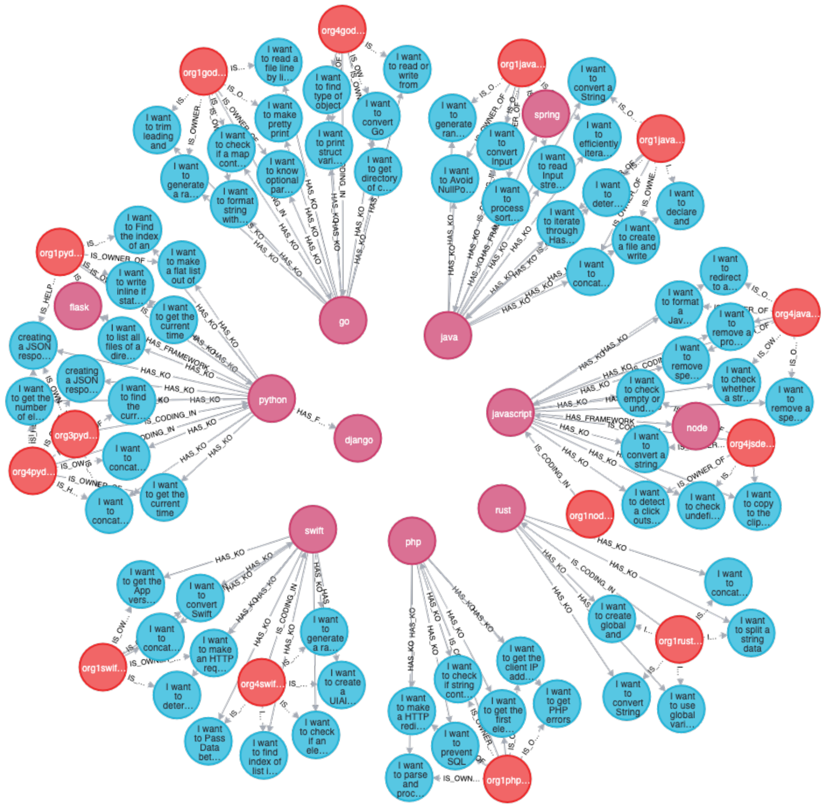


Figure 22: Graph representation of the most popular programming languages.

You can also find the most active user in the graph through a Cypher Query, which will find all developers and count the "`IS_OWNER_OF`" relationships. The developer who has the highest relationship count is the most active developer in terms of adding the user story to the On-demand programming assistance platform. The Cypher Query to retrieve this information is noted in the below code section, and the graph representation for that is illustrated in Figure 23. Also, the textual Cypher Query result has shown in Figure 24.

```
MATCH (developer:dev)-[rel:IS_OWNER_OF]->(k:kotitle)
RETURN developer, k, count(rel) as count ORDER BY count DESC
```

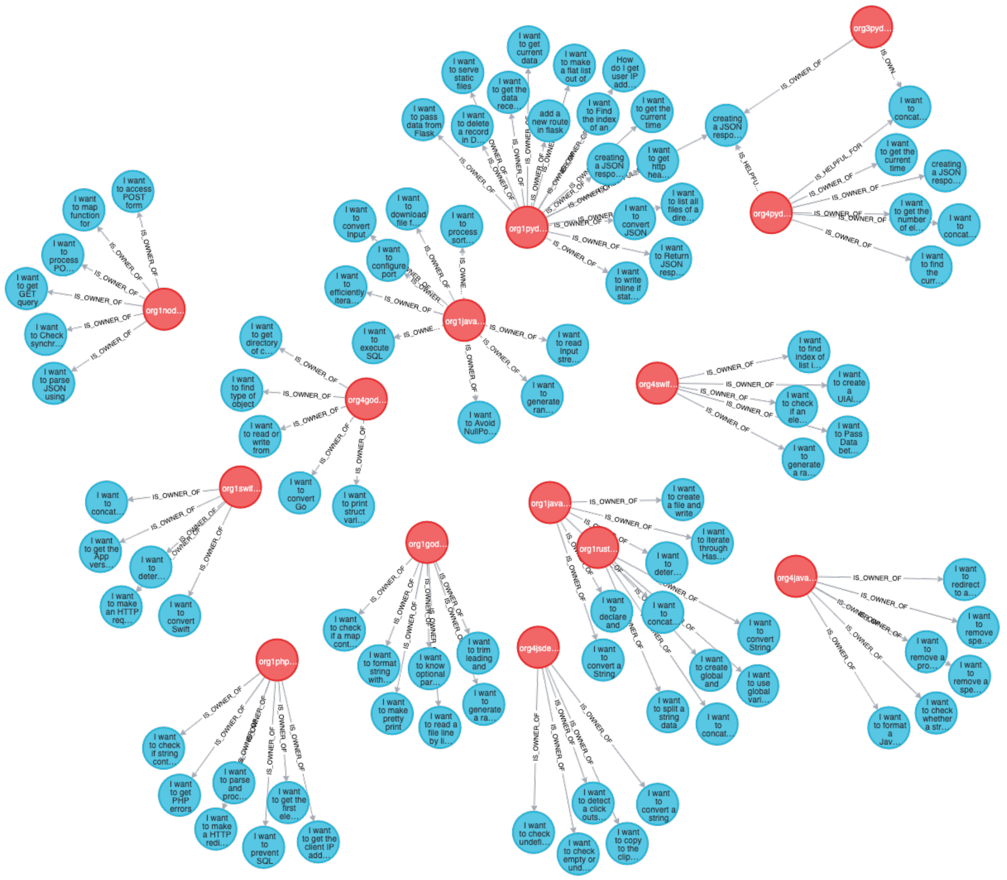
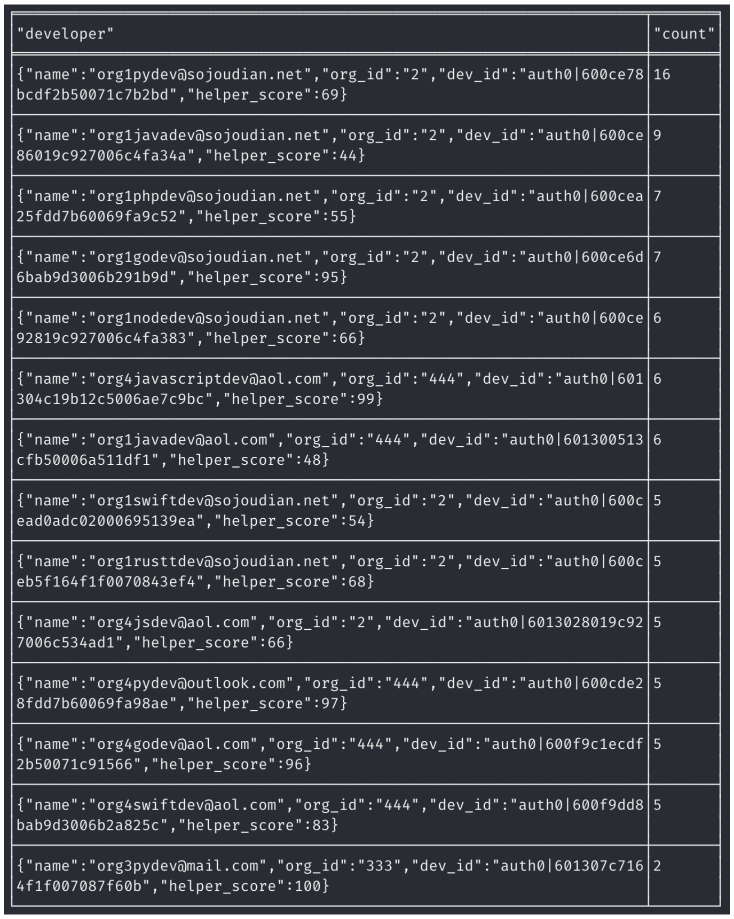


Figure 23: Showing all active developers with their knowledge object title node as the user story node.



"developer"	"count"
{"name":"org1pydev@sojoudian.net","org_id":"2","dev_id":"auth0|600ce78bcdf2b50071c7b2bd","helper_score":69}	16
{"name":"org1javadev@sojoudian.net","org_id":"2","dev_id":"auth0|600ce86019c927006c4fa34a","helper_score":44}	9
{"name":"org1phpdev@sojoudian.net","org_id":"2","dev_id":"auth0|600cea25fdd7b60069fa9c52","helper_score":55}	7
{"name":"org1godev@sojoudian.net","org_id":"2","dev_id":"auth0|600ce6d6bab9d3006b291b9d","helper_score":95}	7
{"name":"org1nodedev@sojoudian.net","org_id":"2","dev_id":"auth0|600ce92819c927006c4fa383","helper_score":66}	6
{"name":"org4javascriptdev@aol.com","org_id":"444","dev_id":"auth0|601304c19b12c5006ae7c9bc","helper_score":99}	6
{"name":"org1javadev@aol.com","org_id":"444","dev_id":"auth0|601300513cfb50006a511df1","helper_score":48}	6
{"name":"org1swiftdev@sojoudian.net","org_id":"2","dev_id":"auth0|600cead0adc02000695139ea","helper_score":54}	5
{"name":"org1rusttdev@sojoudian.net","org_id":"2","dev_id":"auth0|600ceb5f164f1f0070843ef4","helper_score":68}	5
{"name":"org4jsdev@aol.com","org_id":"2","dev_id":"auth0|6013028019c927006c534ad1","helper_score":66}	5
{"name":"org4pydev@outlook.com","org_id":"444","dev_id":"auth0|600cde28fdd7b60069fa98ae","helper_score":97}	5
{"name":"org4godev@aol.com","org_id":"444","dev_id":"auth0|600f9c1ecdf2b50071c91566","helper_score":96}	5
{"name":"org4swiftdev@aol.com","org_id":"444","dev_id":"auth0|600f9dd8bab9d3006b2a825c","helper_score":83}	5
{"name":"org3pydev@mail.com","org_id":"333","dev_id":"auth0|601307c7164f1f007087f60b","helper_score":100}	2

Figure 24: The textual representation for all active developers with their user story count.

3.6 Conclusion

In this chapter, we presented our on-demand programming assistance. Its components and how they work together to prepare the system for a proper recommendation for the end-user are elaborated. We reviewed the used technologies, development tools, and the settings of the database and web framework. The data model at the starting and graph feeding phases are discussed along with the construction process of our knowledge graph. Moreover, we studied how we can extract data from user issues, how the date can be presented when a useful signal is applied to it, and how this signal positively impacts our recommendation system. Real-world examples about exploiting the knowledge graph are included.

Chapter 4

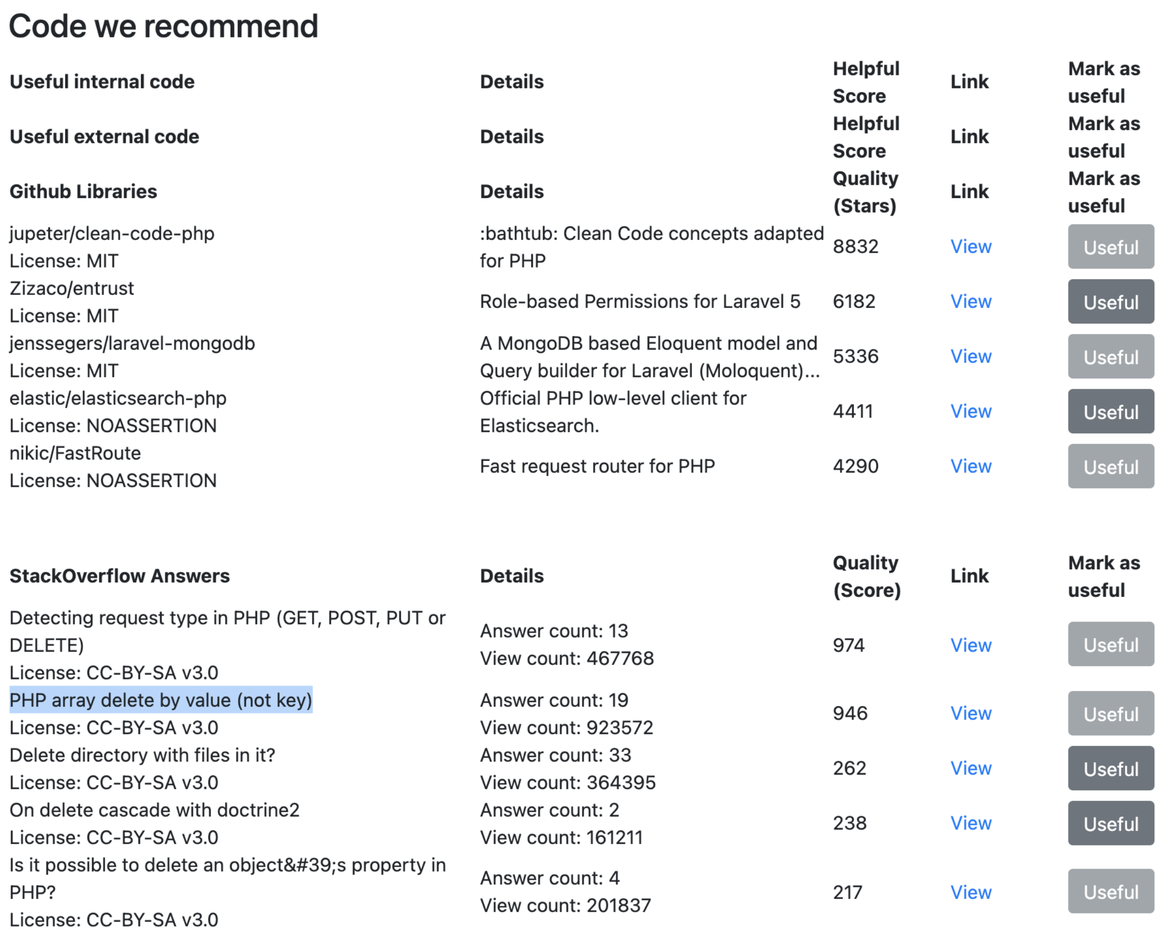
Recommendation Engine

4.1 Introduction

In the previous chapter, we discussed the on-demand programming assistance platform along with its various modules. This chapter will define what the recommendation in our platform is and what our platform should provide to the end-users as the recommendation results. Then, the construction of content-based filtering will be investigated, and we will discuss why it should happen first. Thereafter, collaborative-based filtering recommendations will be covered. Besides, the graph algorithm will be investigated and its implantation will be discussed.

4.2 Selecting Recommendation Candidates

The On-demand programming assistance platform aims to save developers time by providing them answers for their daily routing problems to finish their team's contribution. Developers will ask about their problems in terms of the user story or task. After that, our natural language processing and natural language understanding modules will extract action and context if the problem is the user story. If the problem is a task, our application will extract keywords from the user input. And then, it will construct the graph for the problem. As mentioned before, we know this problem as a Knowledge Object. After preparing the application with all required information, it can provide code repositories from Github and related discussion from the StackOverflow prepared by the On-demand programming assistance's Search module. These code repositories and discussions are known as libraries in our application. If an end-user click on the Useful button for a library, it will mark as a Useful library, and a node with library information will be created in the Knowledge graph, and an edge with "`HAS_HELPFUL_Lib`" will be created from the knowledge object node (or Kotitle node) to the mentioned library, and also another edge from this Library node to the source node will be created. The source node can be different from the Github node or the StackOverflow node depend on the library source. Figure 25 is an example for this case.



Code we recommend

Useful internal code	Details	Helpful Score	Link	Mark as useful

Useful external code	Details	Helpful Score	Link	Mark as useful

Github Libraries	Details	Quality (Stars)	Link	Mark as useful
jupeter/clean-code-php License: MIT	:bathtub: Clean Code concepts adapted for PHP	8832	View	Useful
Zizaco/entrust License: MIT	Role-based Permissions for Laravel 5	6182	View	Useful
jenssegers/laravel-mongodb License: MIT	A MongoDB based Eloquent model and Query builder for Laravel (Moloquent)...	5336	View	Useful
elastic/elasticsearch-php License: NOASSERTION	Official PHP low-level client for Elasticsearch.	4411	View	Useful
nikic/FastRoute License: NOASSERTION	Fast request router for PHP	4290	View	Useful

StackOverflow Answers	Details	Quality (Score)	Link	Mark as useful
Detecting request type in PHP (GET, POST, PUT or DELETE) License: CC-BY-SA v3.0	Answer count: 13 View count: 467768	974	View	Useful
PHP array delete by value (not key) License: CC-BY-SA v3.0	Answer count: 19 View count: 923572	946	View	Useful
Delete directory with files in it? License: CC-BY-SA v3.0	Answer count: 33 View count: 364395	262	View	Useful
On delete cascade with doctrine2 License: CC-BY-SA v3.0	Answer count: 2 View count: 161211	238	View	Useful
Is it possible to delete an object&#39;s property in PHP? License: CC-BY-SA v3.0	Answer count: 4 View count: 201837	217	View	Useful

Figure 25: Make a library mark as a useful library from the Search module output.

Now, the end-user, as we can see, marked "`PHP array delete by value (not key)`" as a helpful library, then the sub-graph for this process will be created in our knowledge graph. To find this sub-graph, we can use the below Cypher Query, and the result for this query is denoted in Figure 26 in the graph data structure form.

```
MATCH (l:lib {name: "PHP array delete by value (not key)"}),
(KO:kotitle)-[:HAS_HELPFUL_Lib]->(l),
(l)-[]-(s:source),
(d:dev)-[:IS_OWNER_OF]->(KO)
RETURN d, KO, l, s;
```

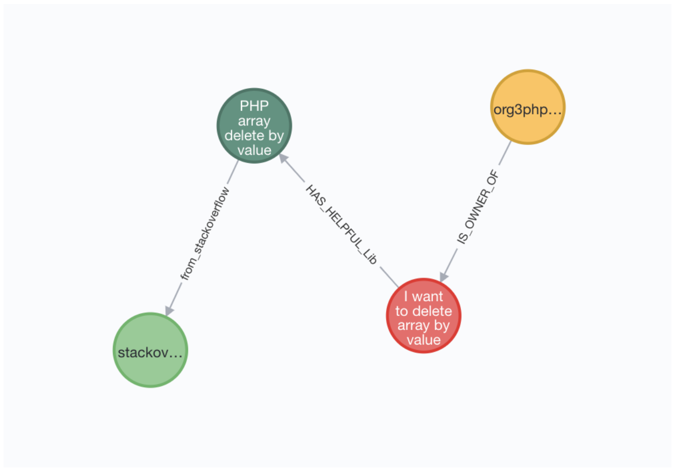
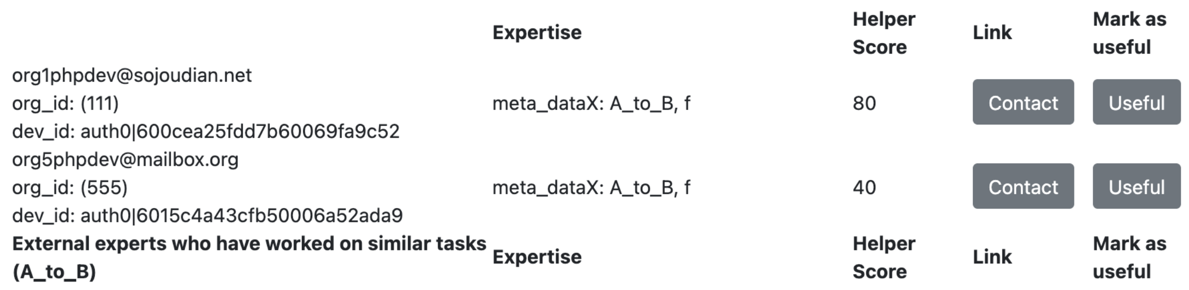


Figure 26: Graph view for a useful library and its relationships.

In the mention graph in Figure 26, the yellow node represents the developer, and the red node shows the knowledge Object, and the green node illustrates the library node, and the light green node denotes the source.

Besides, we have another basic search module, which is traversing the graph to find if a developer worked on the exact same problem. Or if another developer has a knowledge object with the same action or same context, our recommendation module will suggest them to the current user. Like Figure 27. This will only happen if the graph is not large enough to provide a proper recommendation. Or there is not enough information about our system's user story.



	Expertise	**Helper Score**	**Link**	**Mark as useful**
org1phpdev@sojoudian.net org_id: (111) dev_id: auth0|600cea25fdd7b60069fa9c52	meta_dataX: A_to_B, f	80	Contact	Useful
org5phpdev@mailbox.org org_id: (555) dev_id: auth0|6015c4a43cfb50006a52ada9	meta_dataX: A_to_B, f	40	Contact	Useful
External experts who have worked on similar tasks (A_to_B)	**Expertise**	**Helper Score**	**Link**	**Mark as useful**

Figure 27: Web view of useful experts.

The developers who marked as useful are called experts in our knowledge graph. And two edges will be created for them. One from the developer which is marked expert to the knowledge object title (koTitle) node named "`IS_HELPFUL_FOR`". The second edge from the expert developer to The Source node. This Source node can be different between "`EXTERNAL_EXPERT`" if the expert developer is from another organization or "`INTERNAL_EXPERT`" if the expert developer is from the same organization which the owner of the knowledge object title (koTitle) node is working for. And this second edge is named named "`IS_EXPERT`". The Cypher Query and Graph representation would be like below CQL and Figure 27.

```
MATCH (d:dev {dev_id:"auth0|60159b599b12c5006ae94723"})-
[Hrel:IS_HELPFUL_FOR]->(k:kotitle),
(d:dev {dev_id:"auth0|60159b599b12c5006ae94723"})-
[Srel:IS_EXPERT]->(s:source) RETURN d, Hrel, k, Srel, s
```

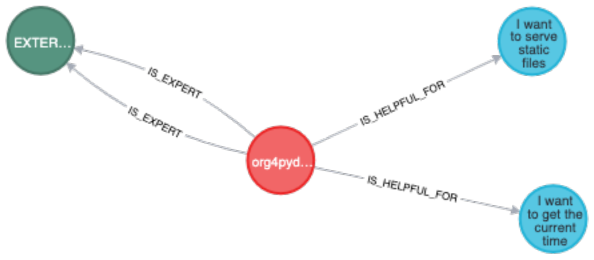


Figure 28: Expert developers graph representation with their relationships.

4.2.1 Constructing Expert Developers and Useful Libraries Graph

After discussing how we collect useful libraries and experts and then construct their nodes and relationships on our knowledge graph, we have to feed the graph with real-world issues and problems for different developers. The graph has more than twenty-five developers and more than 175 user stories, and more than 1075 libraries and 65 Knowledge Objects, which has "`IS_HELPFUL_FOR`" relationships (Figure 29). This Knowledge graph grows by adding new issues from the end-users and marking libraries and experts as helpful. We will build our recommendation systems based on this information.

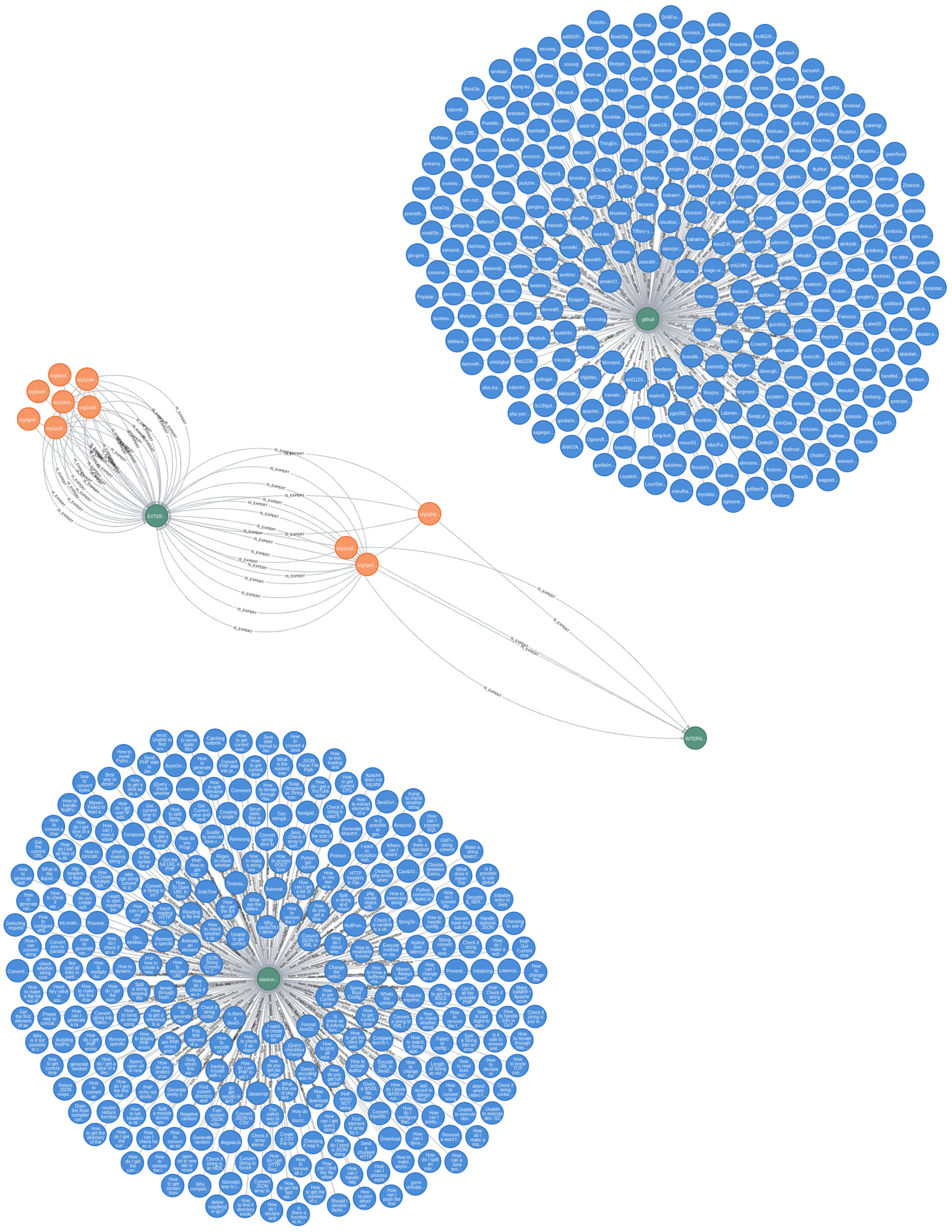


Figure 29: Example of expert developers and useful libraries.

In Figure 29 the blue nodes represent libraries, the green nodes denote different sources, and orange nodes illustrate developer information.

4.2.2 Recommendation Based on Expert Developers and Useful Libraries Graph

Social media like Twitter is trying to recommend people you may like to follow because, based on people you follow, twits you liked before subjects you are interested in. So, end-users will find more attractive and useful content, then they will spend more time on social media, and in the end, user satisfaction will be achieved for the twitter recommendation system. and this is the goal for such a system to provide the closest item for the end-users. In Our platform, On-demand programming assistance, the recommendation system is trying to provide two different recommendations. First, suggesting a developer or some developers as an expert developer who has experience in the current knowledge object or can be helpful about the current issue. Second, recommending libraries from both Github and StackOverflow can also help the current developer who asked the user story to finish their code faster and contribute more efficiently to their team. For both expert and library recommendations, our recommendation engine will provide a list with sorted items, and the item that has the highest score among the list will be on the top of the list. Furthermore, this recommendation will help both developers and organizations. Developers can have great progress regarding using these recommendations. Organizations will have more productive human resources.

4.3 Content-based Filtering Recommendation

Regrading the aforementioned explanations, after the end-user insert their user story to the On-demand programming assistance platform, this platform will provide some libraries that are provided from Stackoverflow and GitHub search API. The On-demand programming assistance platform will also provide expert developers, which are the result of the search on the Knowledge Graph. Normally searching the graph will be done by matching search keywords to provide those expert developers. This search result for expert developers will only appear if there is nothing close to the knowledge object or the developer is a newly registered user on the system. There is no information about him or her on the graph. After this newly registered user marks a developer or a library as useful, the recommendation system will be triggered to provide more and better results as libraries and experts. So, the current user can read those codes or discussion topics to understand better how to solve their problems. They can also connect to an expert who has experience in the problem and can help them solve the issue much faster.

As we explained, if the end-user is newly registered, they can only have the result from the keywords search based, which is not perfect as recommendation systems results are. And also, as we studied in Chapter 2, collaborative-based filtering recommendation systems are suffering from the cold start problems. Hence, at this point system will know there is no historical information on the Knowledge Graph. And then, it will switch to the content-based filtering recommendation system. In the following, we will investigate how does our platform suggests libraries and experts based on the content-based filtering recommendation system.

As we know, one of the best information filtering techniques used to tackle the overwhelming information overload problems is the recommendation systems [4]. It is used to identify related objects or customized items, to match user needs [4]. The content-based filtering technique recommends similar items based on the selected item characteristic, while collaborative filtering recommends items based on the preference of other users [4]. One of the purposes of using the content-Based Filtering recommendation system was to offer libraries and expert developers "`more like`" what they already liked. For the recently registered user scenario, we believe that if the user's chosen a specific library or expert developer, it will have many similar libraries or experts. Still, with limited resources, users may not pick all items. Our recommendation engine should provide a much closer library, expert out of all relevant libraries and experts list. Besides, it is imperative not to provide any inappropriate libraries or expert developers in the recommendation system. In such a scenario, first, regarding the current user's inputs, the search result will be provided by the On-demand programming assistance platform search module. Then, after the user-selected one of the libraries or experts from the search result, the recommendation list will then generated.

Before starting the construction of the content-Based filtering recommendation system, let talk about important information about the user story on the knowledge graph in Figure 30, which results from the below Cypher Query.

```
CALL db.schema()
```

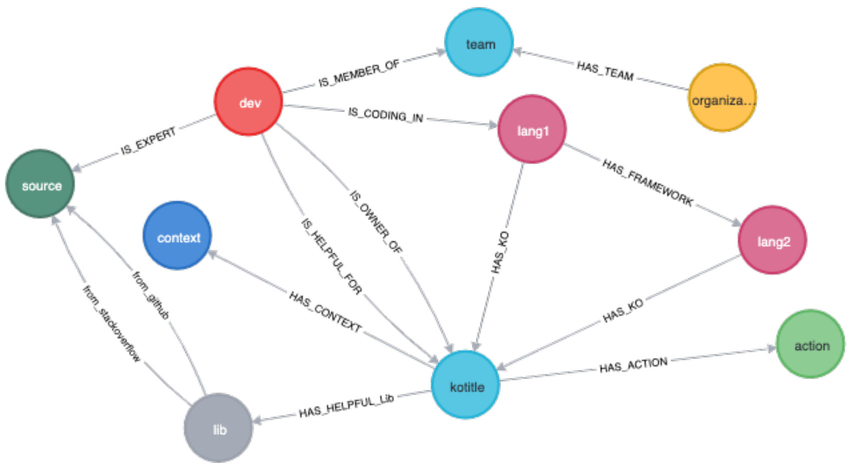


Figure 30: Graph view for a user story in Neo4j

Therefore, our important information will be, "`lang1`" node, which is representing the programming language which is connected to the current user node, "`dev`" node with the "`IS_CODING_IN`" and the relation edge is from the "`dev`" node to the "`lang1`" node.

Another important relationship is from the "lang1" node to the "kotitle" node, as "HAS_KO". And relation from "kotitle" node to "action" node with "HAS_ACTION" relationship. also , relation from "kotitle" node to "context" node with "HAS_CONTEXT" relationship.

If the end-user mark a library as a useful library, a "HAS_HELPFUL_Lib" relationship will be created between the current user story (Knowledge Object) node, which is "kotitle" node and "lib" node as Library node. Besides, if a developer is marked as useful by the current user, a IS_EXPERT" relationship will be created from the "dev" node (which is the developer who is marked as an expert, not the current user) to the "source" node. The "source" node can be different regarding to the source of the node; for example, it can be "INTERNAL_EXPERT", "EXTERNAL_EXPERT", "github" or "stackoverflow".

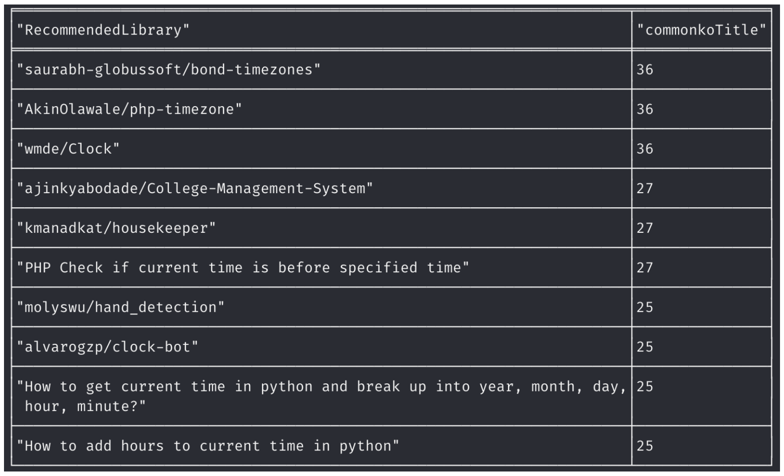
Different content-based filtering recommendation system based on the Cypher Query Language has been constructed, and at the end, we will choose the best among the others.

4.3.1 Content-based Filtering Recommendation, First Approach

The first content-based filtering recommendation technique we want to go through is finding libraries most similar to the library that the current user already marked as a useful library via below Cypher method. The result will be denoted in Figure 31.

```
//Content-Based
MATCH (l:lib)<-[:HAS_HELPFUL_Lib]-(k:kotitle)-[:HAS_HELPFUL_Lib]->(rec:lib)
WHERE k.name CONTAINS "current time" AND
rec.name <>"How to get current time in milliseconds in PHP?"
WITH rec.name as RecommendedLibrary,
COLLECT(k.name) AS koTitle, COUNT(*) AS commonkoTitle
RETURN RecommendedLibrary,commonkoTitle
ORDER BY commonkoTitle DESC LIMIT 10;
```

The aforementioned Cypher command wants to find the other library like what the current user just marked as a useful library, and then it will search for libraries that contain "`time`" as their keyword in the library description. After that, it will collect all other libraries with matched conditions, and in the end, it will count how many times a library has been marked as useful library (the "`HAS_HELPFUL_Lib`" relationship will help us to find and count and sort them) then it will sort libraries based on the number of co-occurrence of them. We only consider the top ten libraries with the most co-occurrence and make a list of them and represent them in descending order for the current user, so the user will know the top is the most popular one. Besides, they can see the "`commonkoTitle`" as the library score. Another important point to mention is the conditions we used were helping us to find libraries from both sources, Stackoverflow and GitHub. The visual illustration of the result will help us understand the Cypher Query much better in Figure 31.



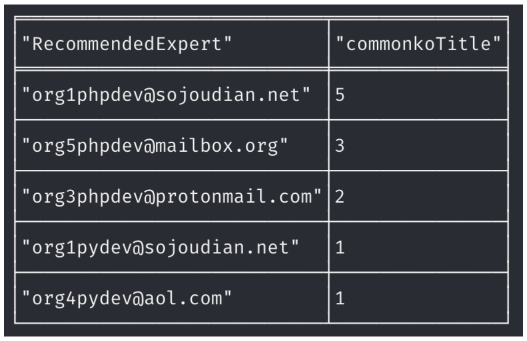
"RecommendedLibrary"	"commonkoTitle"
"saurabh-globussoft/bond-timezones"	36
"AkinOlawale/php-timezone"	36
"wmde/Clock"	36
"ajinkyabodade/College-Management-System"	27
"kmanadkat/housekeeper"	27
"PHP Check if current time is before specified time"	27
"molyswu/hand_detection"	25
"alvarogzp/clock-bot"	25
"How to get current time in python and break up into year, month, day, hour, minute?"	25
"How to add hours to current time in python"	25

Figure 31: Finding libraries most similar to the library that the current user already marked as a useful library in Neo4j

In this scenario, the current user asked a user story, and the content was "`I want to get the current time`" and the library title, which we know as library name, was "`How to get the current time in milliseconds in PHP?`" and the context for this user story was "`current time`". Hence we used "`current time`" to find similar libraries.

It had to be mentioned the same process will be done to find the expert developers for the user story. But in this case, the relationship we use is "`IS_HELPFUL_FOR`" the Cypher Query will be the below command, and the result is noted in Figure 32.

```
//Content-Based-1E
MATCH (d:dev)-[:IS_HELPFUL_FOR]->
(k:kotitle)<-[:IS_HELPFUL_FOR]-(rec:dev)
WHERE k.name CONTAINS "current time"
WITH rec.name as RecommendedExpert,
COLLECT(k.name) AS koTitle, COUNT(*) AS commonkoTitle
RETURN RecommendedExpert, commonkoTitle
ORDER BY commonkoTitle DESC LIMIT 10;
```



"RecommendedExpert"	"commonkoTitle"
"org1phpdev@sojoudian.net"	5
"org5phpdev@mailbox.org"	3
"org3phpdev@protonmail.com"	2
"org1pydev@sojoudian.net"	1
"org4pydev@aol.com"	1

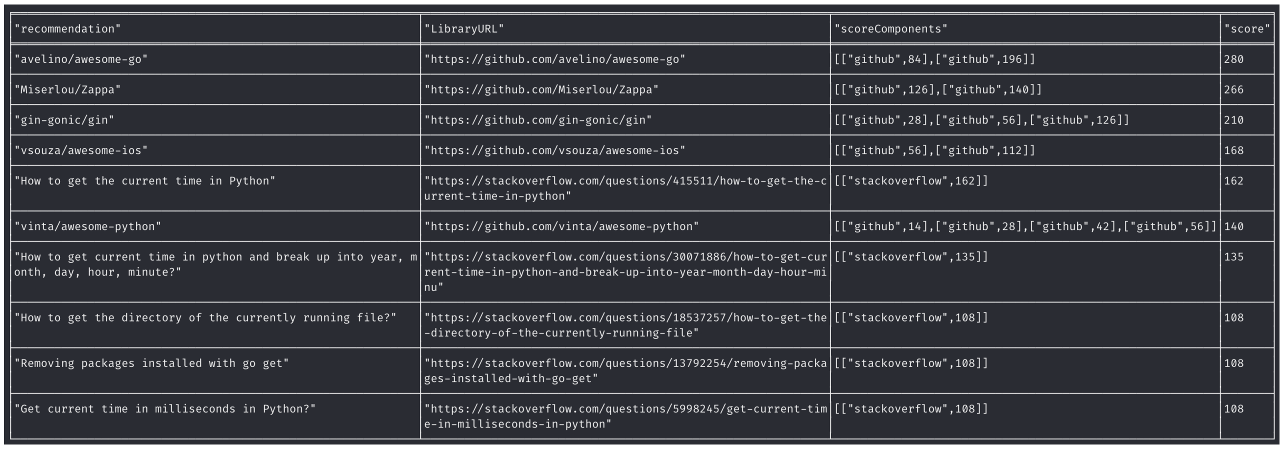
Figure 32: Finding expert developers similar to the expert developer that the current user already marked as a useful expert.

4.3.2 Content-based Filtering Recommendation, Second Approach

The second approach for the content-based filtering recommendation technique is more personalized. Suppose the current user has a user story in the past. Or the current user is trying to get the recommendations for the same user story by refreshing the result page. In that case, we will know what user stories they were working on and which libraries they liked for this user story. We can use existed records on the Knowledge Graph to recommend similar libraries and expert developers similar to those the user has already marked as a useful library or a useful expert developer.

```
//Content-Based-211
MATCH (d:dev {name: "org5phpdev@mailbox.org"})
-[rO:IS_OWNER_OF]->(k:kotitle)-[KrL:HAS_HELPFUL_Lib]->(l:lib),
(l)-[:from_stackoverflow|:from_github]->(s:source
<-[:from_stackoverflow|:from_github]-(rec:lib)
<-[RcK:HAS_HELPFUL_Lib]-(kRec:kotitle)
WHERE NOT EXISTS( (d)-[:IS_OWNER_OF]->
(k:kotitle)-[:HAS_HELPFUL_Lib]->
(rec)-[:from_stackoverflow|:from_github]->(s:source) )
AND k.name CONTAINS "current time"
WITH rec, [s.name, COUNT(*)] AS scores
RETURN rec.name AS recommendation, rec.library_id AS LibraryURL,
COLLECT(scores) AS scoreComponents,
REDUCE (s=0,x in COLLECT(scores) | s+x[1]) AS score
ORDER BY score DESC LIMIT 10;
```

The above Cypher query is a very time and performance-efficient query. The result in our scenario on a knowledge graph with 1681 nodes and 3488 relationships is less than 85 millisecond, and also it will search for both sources, Stackoverflow and GitHub. It means that we can run the query once to find the best possible result for both sources, Stackoverflow and GitHub once. Now, let get through the Cypher query in depth. The query first is finding the current user so it can find who we should recommend items to. Then we are trying to find all Knowledge Objects which has the keywords from the context. After that, it will find if there is a library with "`HAS_HELPFUL_Lib`" relationship but from aforementioned sources regarding "`[:from_stackoverflow|:from_github]`" command. Also, instead of specifying the source, we only used the "`source`" which is the label for all sources. In this case, it can vary from Stackoverflow to GitHub. Also, we will ignore all result which they contain the current user's username. And again, we used keywords in the context for this user story was "`current time`". Hence we used "`current time`" to find similar libraries. The similar libraries are coming from "`(rec:lib)`". And in the end, we will count all the repetition of specific sources to define scores and sort results in order of library scores. And we will use the "`(rec:lib)`" library name as recommendation library and "`(rec:lib)`", "`library_id`" as URL which is pointing to the repository on the GitHub or topic discussion on the Stackoverflow. And aggregation of the scores via "`COLLECT(scores)`" as the scoring component. So then, we can have better insight into how those libraries are picked and sorted. Also, we will use another Neo4j builtin functioned, called "`reduce()`". It returns the value resulting from the expression being added to each successive element of a list in accordance with the outcome of the calculation so far; this function will iterate through each element of "`s`" in the given list, run the expression on "`s`" and store the new partial result in the accumulator, taking into account the current partial result. And in the end, the last part of the command is for ordering scores and limiting it to op ten with the highest scores in descending order. The result will be denoted in Figure 33. It has to be mentioned the result is significantly acceptable for real end-users.



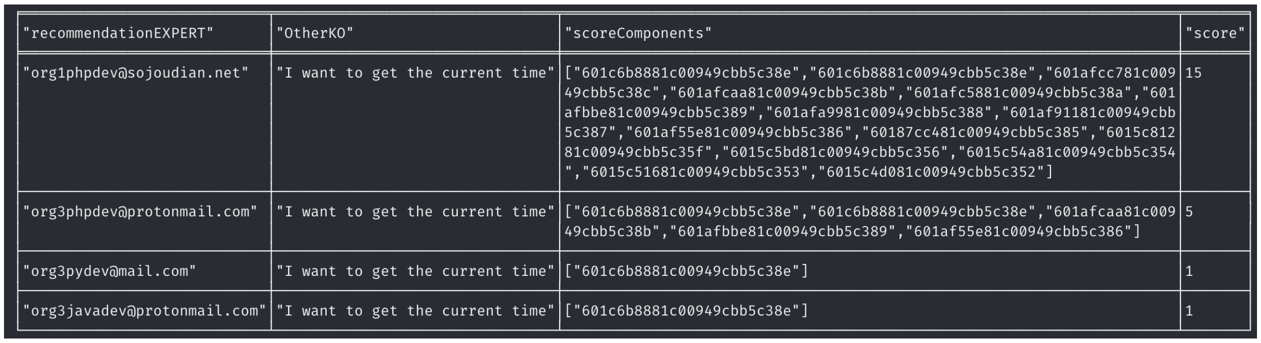
"recommendation"	"LibraryURL"	"scoreComponents"	"score"
"avelino/awesome-go"	"https://github.com/avelino/awesome-go"	[["github",84],["github",196]]	280
"Miserlou/Zappa"	"https://github.com/Miserlou/Zappa"	[["github",126],["github",140]]	266
"gin-gonic/gin"	"https://github.com/gin-gonic/gin"	[["github",28],["github",56],["github",126]]	210
"vsouza/awesome-ios"	"https://github.com/vsouza/awesome-ios"	[["github",56],["github",112]]	168
"How to get the current time in Python"	"https://stackoverflow.com/questions/415511/how-to-get-the-current-time-in-python"	[["stackoverflow",162]]	162
"vinta/awesome-python"	"https://github.com/vinta/awesome-python"	[["github",14],["github",28],["github",42],["github",56]]	140
"How to get current time in python and break up into year, month, day, hour, minute?"	"https://stackoverflow.com/questions/30071886/how-to-get-current-time-in-python-and-break-up-into-year-month-day-hour-minu"	[["stackoverflow",135]]	135
"How to get the directory of the currently running file?"	"https://stackoverflow.com/questions/18537257/how-to-get-the-directory-of-the-currently-running-file"	[["stackoverflow",108]]	108
"Removing packages installed with go get"	"https://stackoverflow.com/questions/13792254/removing-packages-installed-with-go-get"	[["stackoverflow",108]]	108
"Get current time in milliseconds in Python?"	"https://stackoverflow.com/questions/5998245/get-current-time-in-milliseconds-in-python"	[["stackoverflow",108]]	108

Figure 33: Second content-based filtering recommendation using the Cypher query and Neo4j

Now, It is time to apply the same method to find expert developers. So the implementation should be like below Cypher Query.

```
//Content-Based-221
MATCH (d:dev {name: "org5phpdev@mailbox.org"})-
[rO:IS_OWNER_OF]->(k:kotitle {koid:"601b0d4281c00949cbb5c38d"})
<-[KrE:IS_HELPFUL_FOR]-(e:dev),
(e)-[:IS_HELPFUL_FOR]->(Ok:kotitle)
<-[:IS_HELPFUL_FOR]-(rec:dev)-[recREL:IS_EXPERT]->(s:source)
WHERE rec.name <> d.name AND rec.name <>e.name
AND Ok.name CONTAINS "current time"
WITH Ok, rec, [recREL, COUNT(*)] AS scores, recREL.koid as recID
RETURN rec.name AS recommendationEXPERT, Ok.name AS OtherKO,
COLLECT(recID) AS scoreComponents,
REDUCE (s=0,x in COLLECT(scores) | s+x[1]) AS score
ORDER BY score DESC LIMIT 10;
```

Here, first, we found who is the user story owner and then what is the knowledge object ID for the user story, so then we can find who is already marked as an expert developer then we will use the information we gained from this user behavior to recommend more personalized suggestion to the current user. Then, after we found that who is an expert developer in our current user mind, we can find other experts like the previous expert developer. To have a better information filtering, we will avoid the result to provide a list which the current user or the current mark before as a useful developer are members of that via "`WHERE rec.name <> d.name AND rec.name <>e.name`" And again, we used keywords in the context for this user story was "`current time`". To build a scoring system, we will now count how many times a developer is marked as an expert before and sort our list regarding this information. The result of the aforementioned content-based filtering recommendation through the above Cypher Query is illustrated in Figure 34.



"recommendationEXPERT"	"OtherKO"	"scoreComponents"	"score"
"org1phpdev@sojoudian.net"	"I want to get the current time"	["601c6b8881c00949cbb5c38e","601c6b8881c00949cbb5c38e","601afcc781c00949cbb5c38c","601afcaa81c00949cbb5c38b","601afc5881c00949cbb5c38a","601afbbe81c00949cbb5c389","601afa9981c00949cbb5c388","601af91181c00949cbb5c387","601af55e81c00949cbb5c386","60187cc481c00949cbb5c385","6015c81281c00949cbb5c35f","6015c5bd81c00949cbb5c356","6015c54a81c00949cbb5c354","6015c51681c00949cbb5c353","6015c4d081c00949cbb5c352"]	15
"org3phpdev@protonmail.com"	"I want to get the current time"	["601c6b8881c00949cbb5c38e","601c6b8881c00949cbb5c38e","601afcaa81c00949cbb5c38b","601afbbe81c00949cbb5c389","601af55e81c00949cbb5c386"]	5
"org3pydev@mail.com"	"I want to get the current time"	["601c6b8881c00949cbb5c38e"]	1
"org3javadev@protonmail.com"	"I want to get the current time"	["601c6b8881c00949cbb5c38e"]	1

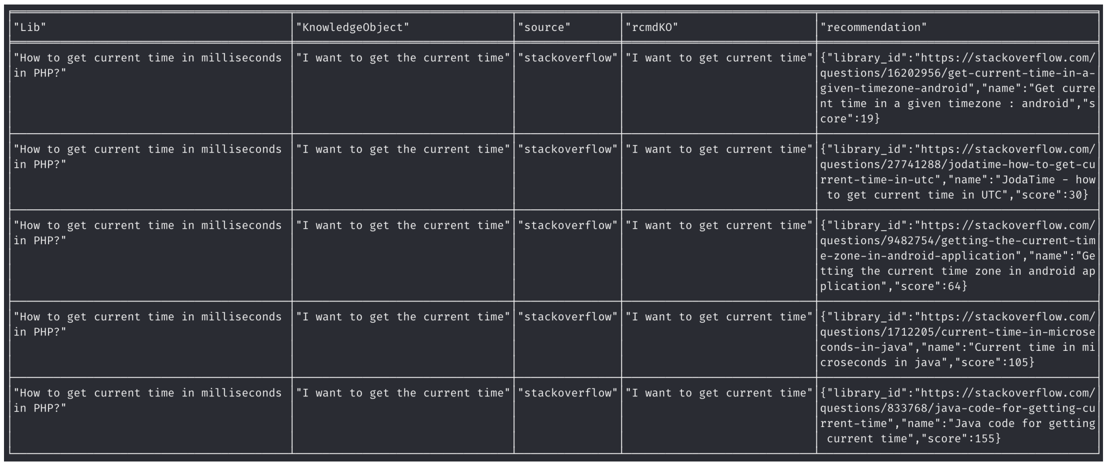
Figure 34: Recommending expert developers regarding the second content-based filtering recommendation using Cypher Query and Neo4j

4.3.3 Content-based Filtering Recommendation, Third Approach

In this method, we will continue the recommendation engine we made in the second approach. Still, using the Third technique, we will have more accurate and closer results to the end-user interests through a weighted procedure, which we will discuss more here. This approach will use more details and more complex traversing paths to provide better recommendation results in our knowledge graph. To have a better understanding of this method, let's study it step by step. First, let us take a look at the first step in the below Cypher Query representation.

```
//Content-Based-31219
MATCH (k:kotitle)-[KrL:HAS_HELPFUL_Lib]->(l:lib),
(l)-[:from_stackoverflow|:from_github]->(s:source)
<-[rREL:from_stackoverflow|:from_github]-(recLib:lib)
<-[RcK:HAS_HELPFUL_Lib]-(kRec:kotitle)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, COUNT(*) AS rcmLibs
WITH l, k, s, kRec, recLib, rcmLibs
RETURN l, k, s,  kRec, rcmLibs, recLib AS recommendation
```

Hence, using the above Cypher query, we are trying to find all libraries from both sources, Stackoverflow and GitHub based on the current knowledge object defined by knowledge object ID or koid. The textual result is depicted in Figure 35 and the associated graph representation is illustrated in Figure 36 where "`recLib`" counts all the libraries in the results.



"Lib"	"KnowledgeObject"	"source"	"rcmdKO"	"recommendation"
"How to get current time in milliseconds in PHP?"	"I want to get the current time"	"stackoverflow"	"I want to get current time"	{"library_id":"https://stackoverflow.com/questions/16202956/get-current-time-in-a-given-timezone-android","name":"Get current time in a given timezone : android","score":19}
"How to get current time in milliseconds in PHP?"	"I want to get the current time"	"stackoverflow"	"I want to get current time"	{"library_id":"https://stackoverflow.com/questions/27741288/jodatime-how-to-get-current-time-in-utc","name":"JodaTime - how to get current time in UTC","score":30}
"How to get current time in milliseconds in PHP?"	"I want to get the current time"	"stackoverflow"	"I want to get current time"	{"library_id":"https://stackoverflow.com/questions/9482754/getting-the-current-time-zone-in-android-application","name":"Getting the current time zone in android application","score":64}
"How to get current time in milliseconds in PHP?"	"I want to get the current time"	"stackoverflow"	"I want to get current time"	{"library_id":"https://stackoverflow.com/questions/1712205/current-time-in-microseconds-in-java","name":"Current time in microseconds in java","score":105}
"How to get current time in milliseconds in PHP?"	"I want to get the current time"	"stackoverflow"	"I want to get current time"	{"library_id":"https://stackoverflow.com/questions/833768/java-code-for-getting-current-time","name":"Java code for getting current time","score":155}

Figure 35: Textual result for the first step of weighted techniques for the Content-based filtering using Cypher Query and Neo4j limited to first top-five

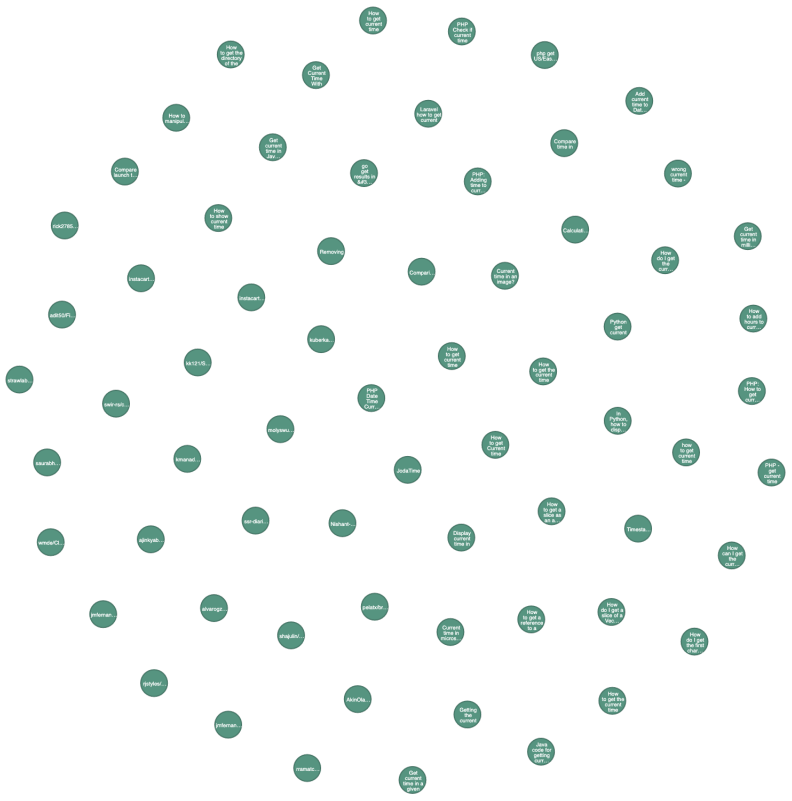


Figure 36: The graph representation for the first step of weighted techniques for the content-based filtering using Cypher Query and Neo4j

Now, in step two, throughout using "`OPTIONAL MATCH`", we will find all the libraries for the current knowledge object on the knowledge graph, which are marked as beneficial for the other Knowledge Objects with the same programming language. The technique is shown in the below Cypher Query. The graph representation for the aforementioned Cypher Query is represented in Figure 37.

```
//Content-Based-31219
MATCH (k:kotitle)-[KrL:HAS_HELPFUL_Lib]->(l:lib),
(l)-[:from_stackoverflow|:from_github]->(s:source)
<-[rREL:from_stackoverflow|:from_github]-(recLib:lib)
<-[RcK:HAS_HELPFUL_Lib]-(kRec:kotitle)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, COUNT(*) AS rcmLibs
//2 STEP Two
```

```
OPTIONAL MATCH (l)<-[:HAS_HELPFUL_Lib]-(k)
<-[:HAS_KO]-(L:lang1)-[Lrel:HAS_KO]->
(kRec:kotitle)-[RcK:HAS_HELPFUL_Lib]->(recLibT:lib)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, rcmLibs, L, COUNT(L) AS Ls
RETURN l, k, s,  kRec, L, Ls, recLib
```

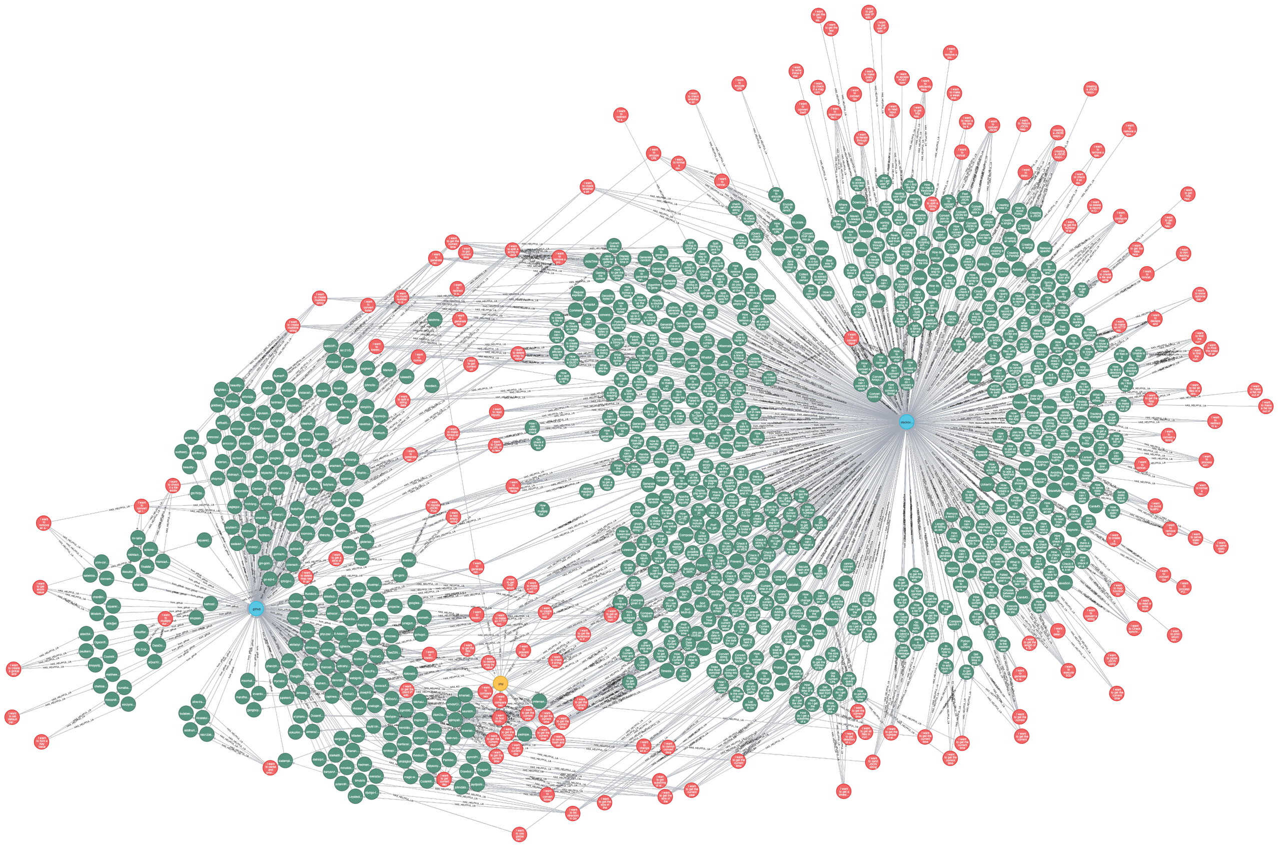


Figure 37: After using OPTIONAL MATCH for more filtering on the result, now we will have all libraries which are marked useful for similar Knowledge Objects.

In the graph illustrated in Figure 37, we can see all libraries represented in the shape of nodes with green color, and these library nodes are connected to the source nodes. The source nodes are represented in light blue color, as we know there are two source nodes for libraries the Stackoverflow and the GitHub. And the relationship between them is "`from_github`" or "`from_stackoverflow`". Besides, library nodes have another relationship in this graph which is "HAS_HELPFUL_Lib" between them and the Knowledge Objects. But the edge for this relationship is from the knowledge objects to the library nodes. knowledge object nodes are shown with red color. Hence the red nodes are demonstrating the Knowledge Objects. Regarding the Cypher Query, we know that the recommendation is based on the current Knowledge Object. And regarding the previous explanation, we know that all the Knowledge Objects has a programming language, now in Figure 37, all the knowledge Objects are related to the "php" programming language, because in the "OPTIONAL MATCH" we narrowed the result to the libraries which are already marked as the useful library for the Knowledge Objects with the same programming language with the current Knowledge Object.

The next step is the third step, we will first, find all knowledge object with similar "action" nodes then we will count all the knowledge object with the same "action" nodes as "rAs"then, in the second part of step three, we will find all "context" nodes similar to the "context" node of the current knowledge object and also count them as "rCS".This is demonstrating in the below Cypher Query.

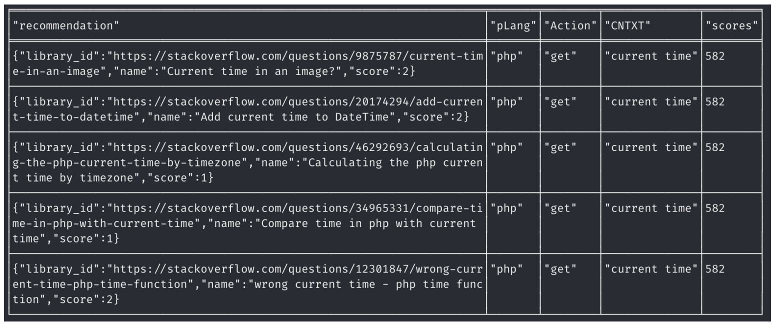
```
//Content-Based-31219
MATCH (k:kotitle)-[KrL:HAS_HELPFUL_Lib]->(l:lib),
(l)-[:from_stackoverflow|:from_github]->(s:source)
<-[rREL:from_stackoverflow|:from_github]-(recLib:lib)
<-[RcK:HAS_HELPFUL_Lib]-(kRec:kotitle)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, COUNT(*) AS rcmLibs
//2 STEP Two
OPTIONAL MATCH (l)<-[:HAS_HELPFUL_Lib]-(k)
<-[:HAS_KO]-(L:lang1)-[Lrel:HAS_KO]->
(kRec:kotitle)-[RcK:HAS_HELPFUL_Lib]->(recLibT:lib)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, rcmLibs, L, COUNT(L) AS Ls
//3 STEP Three
```

```
OPTIONAL MATCH (k)-[:HAS_ACTION]->(a:action),
(recLib)<-[RcK:HAS_HELPFUL_Lib]-(kRec)-[:HAS_ACTION]->(rAct:action)
WITH l, kRec, recLib, rcmLibs, L, Ls, rAct, COUNT(rAct) as rAs
OPTIONAL MATCH (k)-[:HAS_CONTEXT]->(c:context),
(recLib)<-[RcK:HAS_HELPFUL_Lib]-(kRec)-[:HAS_CONTEXT]->(rCon:context)
WITH l, kRec, recLib, rcmLibs, L, Ls, rAct, rAs, rCon, COUNT(rCon) as rCS
RETURN recLib AS recommendation, L.name as pLang,
rAct.name as Action, rCon.name as CNTXT,
(5*rcmLibs)+(4*Ls)+(3*rAs)+(3*rCS) AS scores ORDER BY scores DESC LIMIT 100
```

In the end, we will make a weighting system by gaining benefits from Cypher Query Language and Neo4j database to calculate the score for each library to rank them in order of their score moreover to have better recommendation results. It will be done via the below formula:

$$(5 * rcmLibs) + (4 * Ls) + (3 * rAs) + (3 * rCS)$$

We can see the result for this method in the above Cypher Query with textual representation form, which is limited to the first top-five items with the highest scores. Scores for the current database at the time of writing this thesis are between "`582`" and "`566`" amount sixty library nodes. And besides, the results are really satisfying for actual end-users. We can also have a graph representing the form for the outcome displayed in Figure 38.



"recommendation"	"pLang"	"Action"	"CNTXT"	"scores"
{"library_id":"https://stackoverflow.com/questions/9875787/current-time-in-an-image","name":"Current time in an image?","score":2}	"php"	"get"	"current time"	582
{"library_id":"https://stackoverflow.com/questions/20174294/add-current-time-to-datetime","name":"Add current time to DateTime","score":2}	"php"	"get"	"current time"	582
{"library_id":"https://stackoverflow.com/questions/46292693/calculating-the-php-current-time-by-timezone","name":"Calculating the php current time by timezone","score":1}	"php"	"get"	"current time"	582
{"library_id":"https://stackoverflow.com/questions/34965331/compare-time-in-php-with-current-time","name":"Compare time in php with current time","score":1}	"php"	"get"	"current time"	582
{"library_id":"https://stackoverflow.com/questions/12301847/wrong-current-time-php-time-function","name":"wrong current time - php time function","score":2}	"php"	"get"	"current time"	582

Figure 38: Weighted techniques for the content-based recommendation system

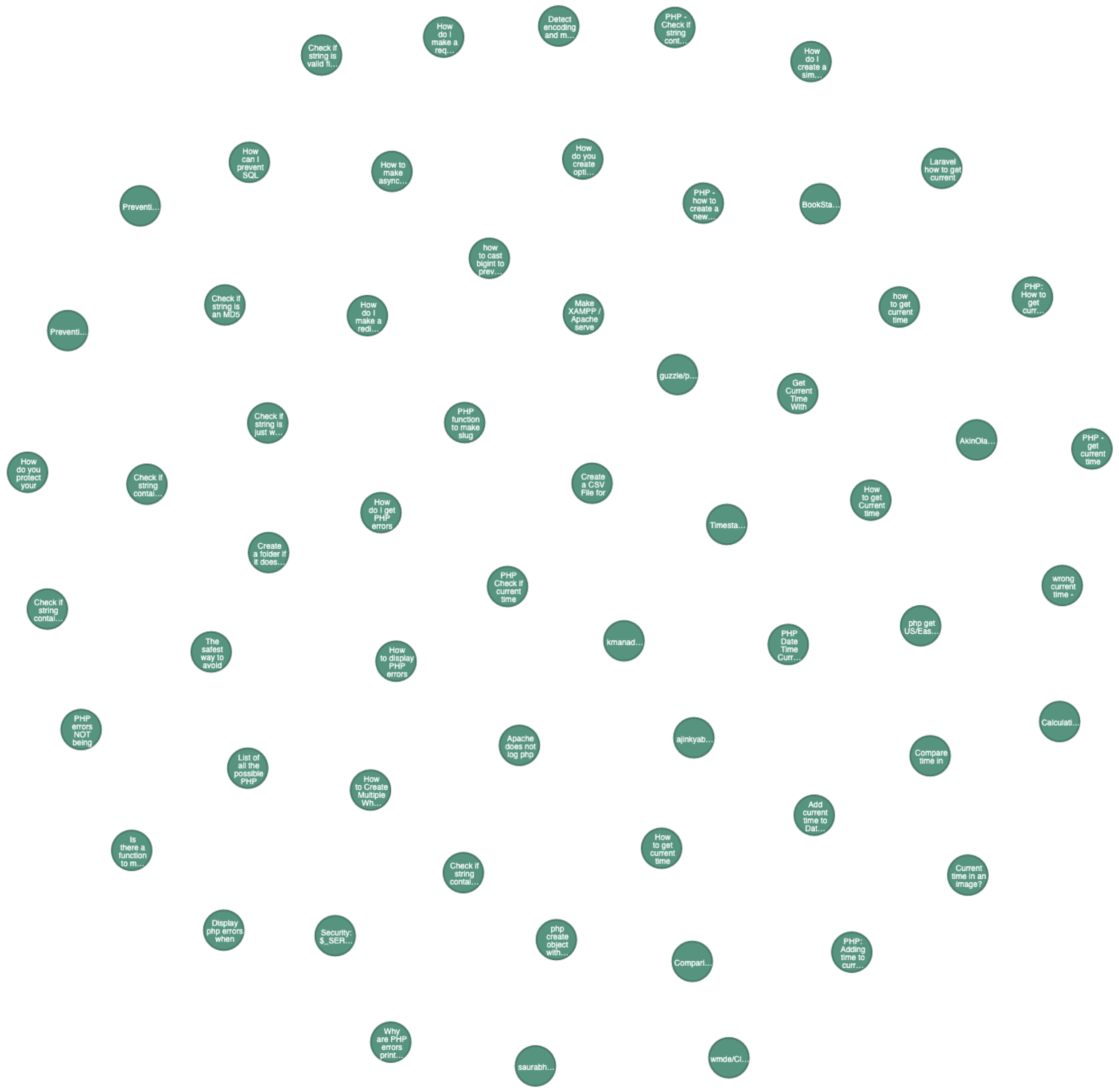
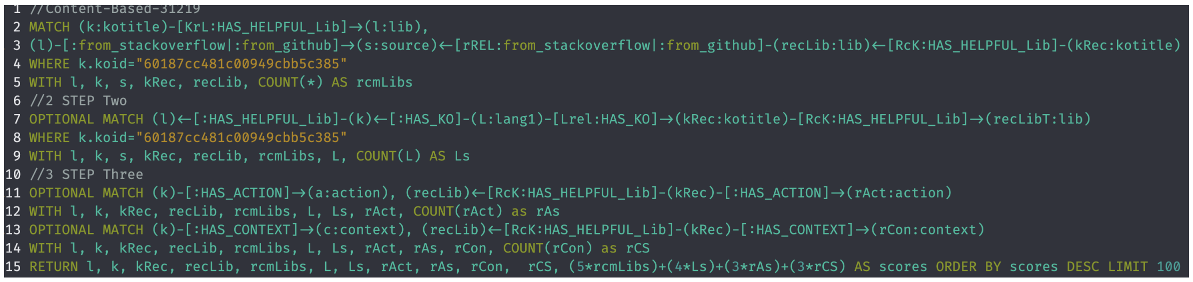


Figure 39: Weighted techniques for the content-based Recommendation system

Besides, if we change the last line of our Cypher Query like Figure 40, we can have a better graph with all related nodes and relationships, and it will help us to understand the method easier. The result of the mentioned Cypher query in Figure 40 is represented in Figure 41



```
//Content-Based-31219
MATCH (k:kotitle)-[KrL:HAS_HELPFUL_Lib]→(l:lib),
(l)-[:from_stackoverflow|:from_github]→(s:source)←[rREL:from_stackoverflow|:from_github]-(recLib:lib)←[RcK:HAS_HELPFUL_Lib]-(kRec:kotitle)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, COUNT(*) AS rcmLibs
//2 STEP Two
OPTIONAL MATCH (l)←[:HAS_HELPFUL_Lib]-(k)←[:HAS_KO]-(L:lang1)-[Lrel:HAS_KO]→(kRec:kotitle)-[RcK:HAS_HELPFUL_Lib]→(recLibT:lib)
WHERE k.koid="60187cc481c00949cbb5c385"
WITH l, k, s, kRec, recLib, rcmLibs, L, COUNT(L) AS Ls
//3 STEP Three
OPTIONAL MATCH (k)-[:HAS_ACTION]→(a:action), (recLib)←[RcK:HAS_HELPFUL_Lib]-(kRec)-[:HAS_ACTION]→(rAct:action)
WITH l, k, kRec, recLib, rcmLibs, L, Ls, rAct, COUNT(rAct) as rAs
OPTIONAL MATCH (k)-[:HAS_CONTEXT]→(c:context), (recLib)←[RcK:HAS_HELPFUL_Lib]-(kRec)-[:HAS_CONTEXT]→(rCon:context)
WITH l, k, kRec, recLib, rcmLibs, L, Ls, rAct, rAs, rCon, COUNT(rCon) as rCS
RETURN l, k, kRec, recLib, rcmLibs, L, Ls, rAct, rAs, rCon,  rCS, (5*rcmLibs)+(4*Ls)+(3*rAs)+(3*rCS) AS scores ORDER BY scores DESC LIMIT 100
```

Figure 40: The Cypher Query print result as nodes with their relationships in the form of the edges.

In the graph which is represented in Figure 41, the programming language is illustrated with yellow color as "`lang1: php`". As we can see in Figure 41, there is only one programming language that means all information is filtered based on this programming language. This programming language is connected to the Knowledge Graph nodes with "`HAS_KO`" Relationships, and the "`Knowledge Object`" nodes are displayed in red color. Besides, all the "`library`" nodes are shown in dark green color. Next is the "`action`" node, which is denoted in light green color, and it is connected to the knowledge object node via "`HAS_ACTION`" relationship. The last node in the graph is the "`context`" in blue color, and it is connected to the knowledge object by "`HAS_CONTEXT`". If we look at Figure 41, we will understand the method much easier.

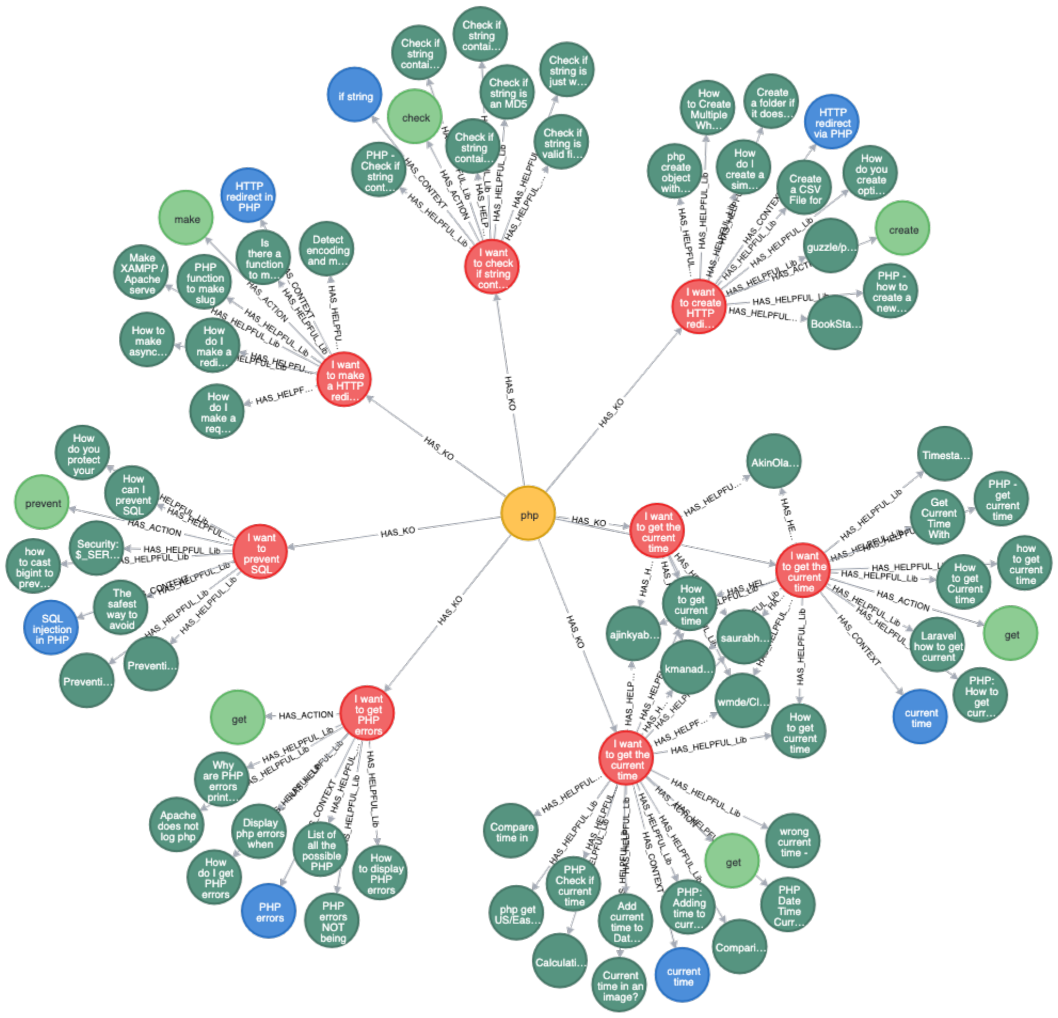
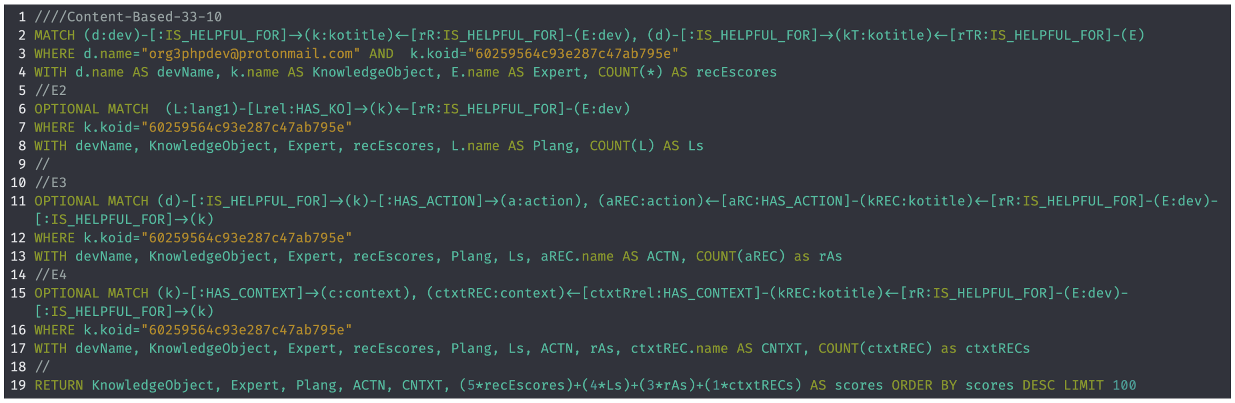


Figure 41: The result for previous Cypher Query

Now it is time to apply the same techniques for finding the expert developer in our Knowledge Graph. Let's consider this method to recommend better expert developers through a weighted content-based filtering recommendation method.

According to Figure 42 in our scenario, the current user has already marked an expert with "`org2phpdev@yandex.com`" as the username. We call it node "`d`" for knowledge object with the following knowledge object ID "`60259564c93e287c47ab795e`" and then we will find the other expert developers for the same Knowledge Object. We will find other Knowledge Objects in which first the expert developer are marked as an expert for them, and also the other expert developers are marked as the expert for them, and we will count these nodes as "`recEscores`".



```
////Content-Based-33-10
MATCH (d:dev)-[:IS_HELPFUL_FOR]→(k:kotitle)←[rR:IS_HELPFUL_FOR]-(E:dev), (d)-[:IS_HELPFUL_FOR]→(kT:kotitle)←[rTR:IS_HELPFUL_FOR]-(E)
WHERE d.name="org3phpdev@protonmail.com" AND  k.koid="60259564c93e287c47ab795e"
WITH d.name AS devName, k.name AS KnowledgeObject, E.name AS Expert, COUNT(*) AS recEscores
//E2
OPTIONAL MATCH  (L:lang1)-[Lrel:HAS_KO]→(k)←[rR:IS_HELPFUL_FOR]-(E:dev)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores, L.name AS Plang, COUNT(L) AS Ls
//
//E3
OPTIONAL MATCH (d)-[:IS_HELPFUL_FOR]→(k)-[:HAS_ACTION]→(a:action), (aREC:action)←[aRC:HAS_ACTION]-(kREC:kotitle)←[rR:IS_HELPFUL_FOR]-(E:dev)-
[:IS_HELPFUL_FOR]→(k)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores, Plang, Ls, aREC.name AS ACTN, COUNT(aREC) as rAs
//E4
OPTIONAL MATCH (k)-[:HAS_CONTEXT]→(c:context), (ctxtREC:context)←[ctxtRrel:HAS_CONTEXT]-(kREC:kotitle)←[rR:IS_HELPFUL_FOR]-(E:dev)-
[:IS_HELPFUL_FOR]→(k)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores, Plang, Ls, ACTN, rAs, ctxtREC.name AS CNTXT, COUNT(ctxtREC) as ctxtRECs
//
RETURN KnowledgeObject, Expert, Plang, ACTN, CNTXT, (5*recEscores)+(4*Ls)+(3*rAs)+(1*ctxtRECs) AS scores ORDER BY scores DESC LIMIT 100
```

Figure 42: The Cypher Query to find expert based on the weighting method

Then, we will find the programming language for the current knowledge Object, and we will count its repetition as "`Ls`" using the "`OPTIONAL MATCH`" method. After that, via another "`OPTIONAL MATCH`" we will try to find the "`action`" node for the current knowledge object and those other knowledge object which the expert "`d`" and all other experts are marked useful for them. And we will count the number of "`actions`" as "`rAs`". The next step is to do the same process for the "`context`" nodes. Hence we will first find the "`context`" for the current Knowledge Object. We will then find the "`context`" for the other Knowledge Objects which expert "`d`" and all other experts are marked useful for them. And we will make an alias for the count of the all "`context`" nodes as "`ctxtREC`". In the end, we will make our weighting system via the below formula.

$$(5 * recEscores) + (4 * Ls) + (3 * rAs) + (1 * ctxtRECs)$$

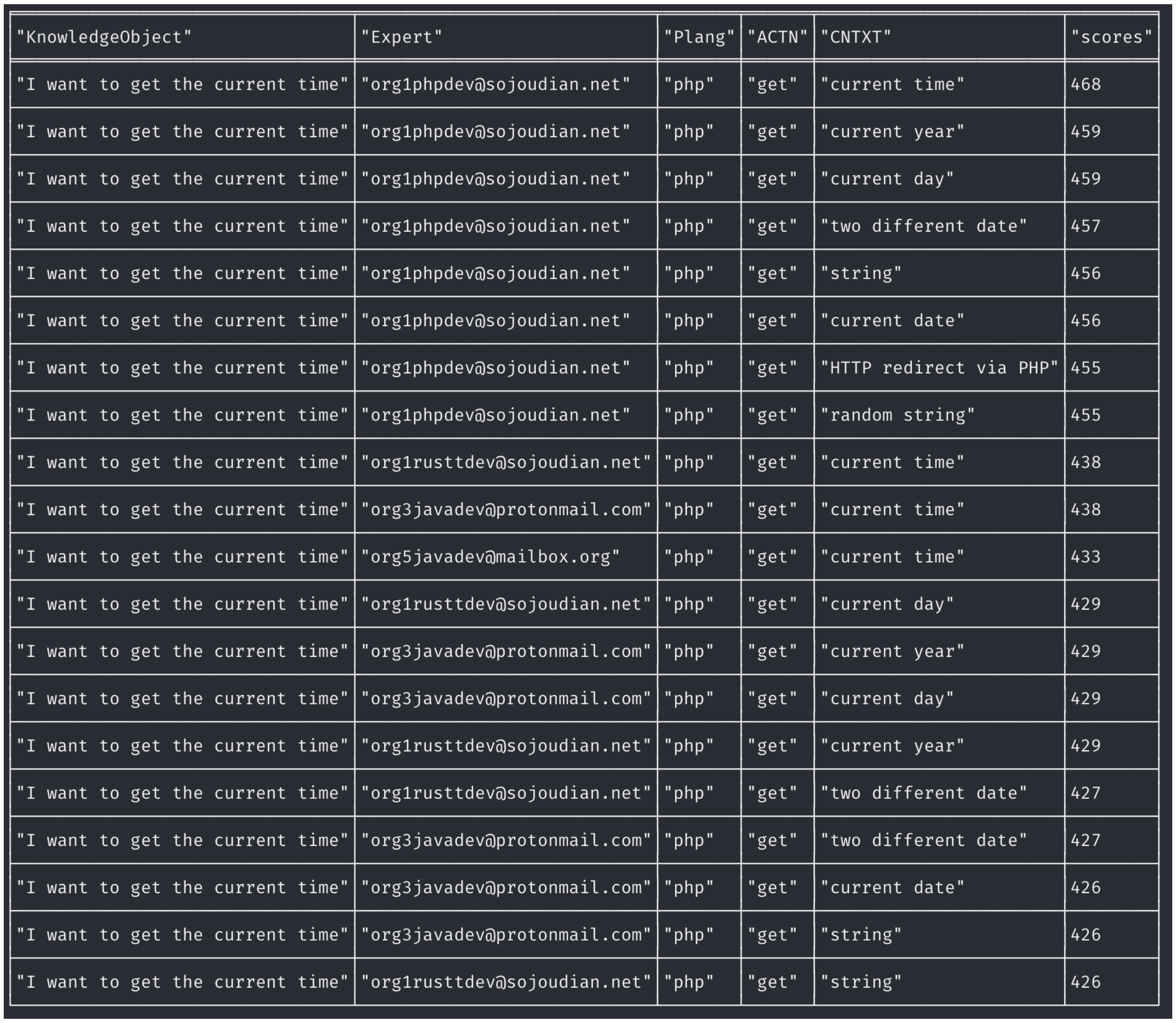
The full Cypher Query for the aforementioned method is denoted in the below query.

```
////Content-Based-33-10
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)<-[rR:IS_HELPFUL_FOR]-(E:dev),
(d)-[:IS_HELPFUL_FOR]->(kT:kotitle)<-[rTR:IS_HELPFUL_FOR]-(E)
WHERE d.name="org3phpdev@protonmail.com"
```

```
AND  k.koid="60259564c93e287c47ab795e"
WITH d.name AS devName, k.name AS KnowledgeObject,
E.name AS Expert, COUNT(*) AS recEscores
//E2
OPTIONAL MATCH  (L:lang1)-[Lrel:HAS_KO]->(k)<-[rR:IS_HELPFUL_FOR]-(E:dev)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores,
L.name AS Plang, COUNT(L) AS Ls
//E3
OPTIONAL MATCH (d)-[:IS_HELPFUL_FOR]->(k)-[:HAS_ACTION]->(a:action),
(aREC:action)<-[aRC:HAS_ACTION]-(kREC:kotitle)
<-[rR:IS_HELPFUL_FOR]-(E:dev)-[:IS_HELPFUL_FOR]->(k)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores, Plang,
Ls, aREC.name AS ACTN, COUNT(aREC) as rAs
//E4
OPTIONAL MATCH (k)-[:HAS_CONTEXT]->(c:context),
(ctxtREC:context)<-[ctxtRrel:HAS_CONTEXT]-(kREC:kotitle)
<-[rR:IS_HELPFUL_FOR]-(E:dev)-[:IS_HELPFUL_FOR]->(k)
WHERE k.koid="60259564c93e287c47ab795e"
WITH devName, KnowledgeObject, Expert, recEscores, Plang,
Ls, ACTN, rAs, ctxtREC.name AS CNTXT, COUNT(ctxtREC) as ctxtRECs
//
RETURN KnowledgeObject, Expert, Plang, ACTN, CNTXT,
(5*recEscores)+(4*Ls)+(3*rAs)+(1*ctxtRECs) AS scores
ORDER BY scores DESC LIMIT 100
```

The result can be investigated in various forms. We will study textual forms in Figure 43, which shows the result for the top twenty results. Regarding Figure 43 we can see that the scores are between 468 and 426. And it is sorted in order of how relevant they are. For example, the first recommended row is highly close to the current Knowledge

Object, and we can see the other context different from the current context. Still, they are highly similar, and the expert is marked useful for those Knowledge Objects. So then the current user will have more trust in the recommendation system. For example, the expert with "`org1phpdev@sojoudian.net`" username has related knowledge object with the same programming language "`php`" and related contexts. It has to be mentioned, the actual result is more than twenty, and the scores are between 468 and 48.



```
"KnowledgeObject"                 |"Expert"                       |"Plang"|"ACTN"|"CNTXT"                 |"scores"
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"current time"          |468
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"current year"          |459
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"current day"           |459
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"two different date"    |457
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"string"                |456
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"current date"          |456
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"HTTP redirect via PHP" |455
"I want to get the current time"|"org1phpdev@sojoudian.net"     |"php"  |"get" |"random string"         |455
"I want to get the current time"|"org1rusttdev@sojoudian.net"   |"php"  |"get" |"current time"          |438
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"current time"          |438
"I want to get the current time"|"org5javadev@mailbox.org"      |"php"  |"get" |"current time"          |433
"I want to get the current time"|"org1rusttdev@sojoudian.net"   |"php"  |"get" |"current day"           |429
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"current year"          |429
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"current day"           |429
"I want to get the current time"|"org1rusttdev@sojoudian.net"   |"php"  |"get" |"current year"          |429
"I want to get the current time"|"org1rusttdev@sojoudian.net"   |"php"  |"get" |"two different date"    |427
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"two different date"    |427
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"current date"          |426
"I want to get the current time"|"org3javadev@protonmail.com"   |"php"  |"get" |"string"                |426
"I want to get the current time"|"org1rusttdev@sojoudian.net"   |"php"  |"get" |"string"                |426
```

Figure 43: The result for finding expert developers based on the weighting method

4.3.4 Content-based Filtering Recommendation, Forth Approach

In this approach, we will use the Jaccard Index of similarity. But we will have some modifications to have more accurate results throughout using auxiliary data extracted from the graph. First, let's study what the Jaccard Index of similarity is. It is a measure used among two data sets to study similarity (or dissimilarity), originally developed by Jaccard [37]. The Jaccard index is defined as the proportion of the size of the intersection over union for any two finite sets, A and B (IoU or Intersection over Union), and it is shown in 1 [38].

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

The value for the Jaccard Index of similarity can range from 0 to 1, as an example, it can be defined as

$$0 \leq J(A,B) \leq 1$$

The more similarities between sample sets are expressed in the higher value of *J(A, B)*. [38].

First, we will apply the traditional Jaccard Index of similarity to our problem, finding libraries like what the current user already marked as a useful library. In the first step, we will find the current knowledge object and the library that is already marked as a useful library, and the knowledge object's programming language. We will then find other knowledge objects with the same programming language and libraries that are useful for them. Also, the count of this repetition will be aliased as "`intersection`". In step 2, we will look for other knowledge objects with the same knowledge object name, and then we will collect other knowledge objects context nodes as "`CNTXT`". The context node for other knowledge objects can be different from the current knowledge object. In step 3, action nodes for other knowledge objects with the same knowledge objects will be collected as "`ACTN`". Then the union we will calculate as the "`union`" according to 2, and then the

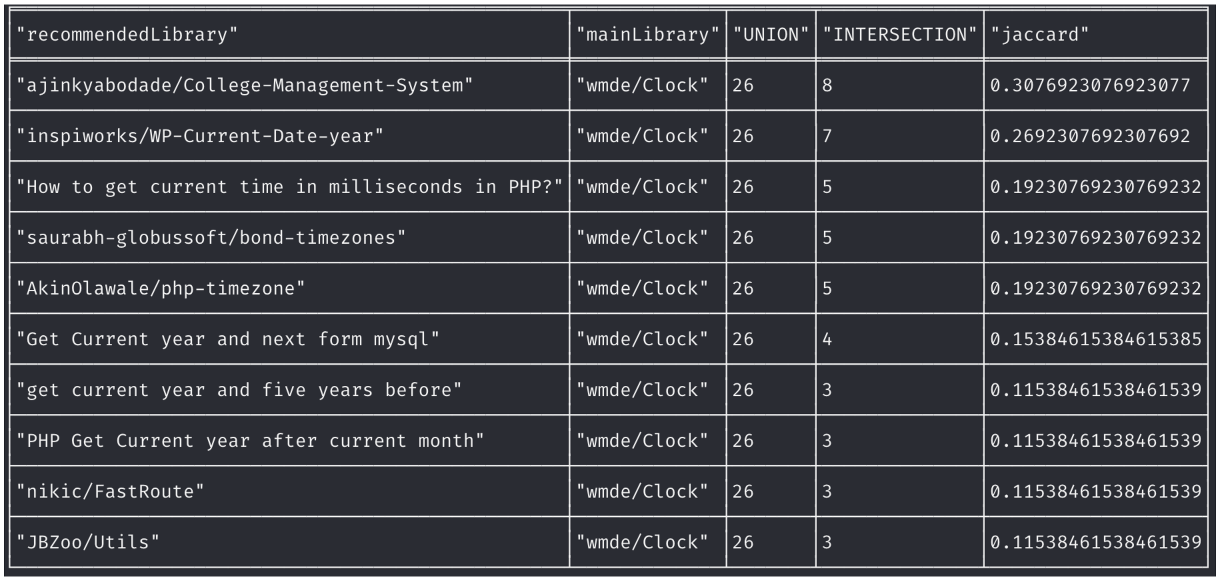
Jaccard Index of similarity will be calculated via 3.

$$ACTN + filter(xINCNTXTWHERENOTxINACTN)ASunion \tag{2}$$

$$((1.0 * INTERSECTION)/UNION)ASjaccard \tag{3}$$

Moreover, the full Cypher Query for the mentioned process is demonstrated in the below query. The results for the traditional Jaccard Index of similarity are denoted in 44.

```
//Content-Based-3434
MATCH (l:lib)<-[:HAS_HELPFUL_Lib]-(k:kotitle)-[:HAS_KO]-(q)
-[:HAS_KO]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(recL:lib)
WHERE k.koid="6028391fc93e287c47ab795f"
AND l.name= "wmde/Clock" AND recL.name <> "wmde/Clock"
WITH l, k, recL, COUNT(q) AS intersection, COLLECT(q.name) AS plang
//step2
MATCH (sc)-[:HAS_CONTEXT]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(SrecL:lib)
WHERE reck.name="I want to get the current time"
WITH l, k, recL, intersection, plang, sc.name AS SC, sc.koid as SCid
WITH l, k, recL, intersection, plang, SC, SCid, COLLECT(SC) AS CNTXT
//step3
MATCH (sa)-[:HAS_ACTION]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(TrecL:lib)
WHERE reck.name="I want to get the current time"
WITH l, k, recL, intersection, plang, CNTXT, COLLECT(sa.name) AS ACTN
WITH l, k, recL, intersection, plang, CNTXT, ACTN
WITH l, k, recL, intersection, plang,
ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN) AS union, CNTXT, ACTN
WITH DISTINCT recL.name AS recommendedLibrary, l.name AS mainLibrary,
MAX(SIZE(union)) AS UNION, MAX(intersection) AS INTERSECTION
RETURN recommendedLibrary, mainLibrary, UNION, INTERSECTION,
((1.0*INTERSECTION)/UNION) AS jaccard ORDER BY jaccard DESC LIMIT 10
```

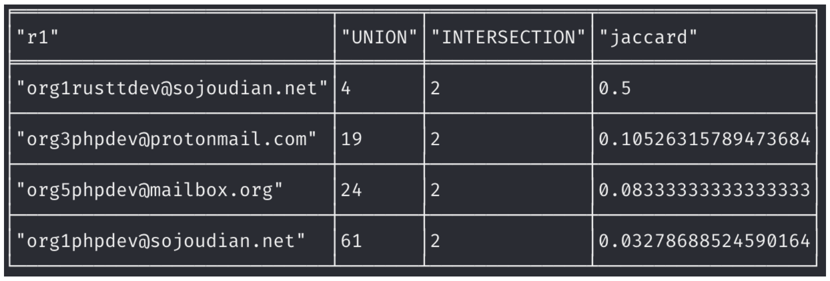


"recommendedLibrary"	"mainLibrary"	"UNION"	"INTERSECTION"	"jaccard"
"ajinkyabodade/College-Management-System"	"wmde/Clock"	26	8	0.3076923076923077
"inspiworks/WP-Current-Date-year"	"wmde/Clock"	26	7	0.2692307692307692
"How to get current time in milliseconds in PHP?"	"wmde/Clock"	26	5	0.19230769230769232
"saurabh-globussoft/bond-timezones"	"wmde/Clock"	26	5	0.19230769230769232
"AkinOlawale/php-timezone"	"wmde/Clock"	26	5	0.19230769230769232
"Get Current year and next form mysql"	"wmde/Clock"	26	4	0.15384615384615385
"get current year and five years before"	"wmde/Clock"	26	3	0.11538461538461539
"PHP Get Current year after current month"	"wmde/Clock"	26	3	0.11538461538461539
"nikic/FastRoute"	"wmde/Clock"	26	3	0.11538461538461539
"JBZoo/Utils"	"wmde/Clock"	26	3	0.11538461538461539

Figure 44: Results for the traditional Jaccard Index of similarity.

The same process will be applied to find the best candidates to recommend as expert developers. As a result, the Cypher Query will be expressed in the query below, and the results will be shown in 45.

```
//Content-Based-34-23
//step1
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_KO]-(q)
-[:HAS_KO]-(reck:kotitle)<-[:IS_HELPFUL_FOR]-(recD:dev)
WHERE k.koid="6028391fc93e287c47ab795f" AND d.name="org1phpdev@sojoudian.net"
WITH d, k, recD, reck, SIZE( ()-[:IS_HELPFUL_FOR]->(reck) ) AS intersection,
COLLECT(q.name) AS plang
//step2
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_CONTEXT]-(fc),
(recD:dev)-[:IS_HELPFUL_FOR]->(recK:kotitle)-[:HAS_CONTEXT]->(cS:context)
WHERE k.koid="6028391fc93e287c47ab795f"
AND reck.name="I want to get the current time"
WITH reck, recD, intersection, plang, collect(cS.name) AS CNTXT
//step3
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_ACTION]-(fa),
(recD:dev)-[:IS_HELPFUL_FOR]->(recK:kotitle)-[:HAS_ACTION]->(acS:action)
WHERE k.koid="6028391fc93e287c47ab795f"
AND reck.name="I want to get the current time"
WITH reck, recD, intersection, plang, CNTXT, COLLECT(acS.name) AS ACTN
WITH reck, recD, intersection, plang, CNTXT, ACTN
WITH reck, recD, intersection, plang,
ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN) AS union, CNTXT, ACTN
WITH DISTINCT recD.name as r1,
MAX(SIZE(union)) AS UNION, MAX(intersection) AS INTERSECTION
RETURN r1, UNION, INTERSECTION,
((1.0*INTERSECTION)/UNION) AS jaccard ORDER BY jaccard DESC LIMIT 10
```



"r1"	"UNION"	"INTERSECTION"	"jaccard"
"org1rusttdev@sojoudian.net"	4	2	0.5
"org3phpdev@protonmail.com"	19	2	0.10526315789473684
"org5phpdev@mailbox.org"	24	2	0.08333333333333333
"org1phpdev@sojoudian.net"	61	2	0.03278688524590164

Figure 45: Results for the traditional Jaccard Index of similarity.

Now let's consider how we apply this method to our knowledge graph to recommend based on the Jaccard Index of similarity technique and auxiliary data. Let's say we have a knowledge object with "6028391fc93e287c47ab795f" as knowledge object ID. First, we have to find this Knowledge Object, and we have to mention again we will recommend a useful library based on the library the current user is marked as useful. Hence we will consider library node with "wmde/Clock" as library name. The graph representation for this knowledge object and its relationships and connected nodes will help us better understand the knowledge object in Figure 46.

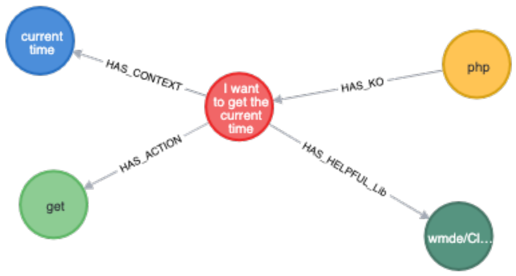


Figure 46: The graph representation for the mentioned knowledge object and its related nodes and relationships

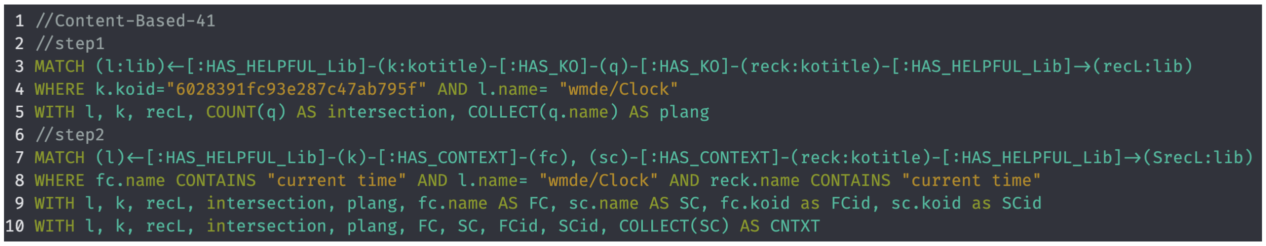
In the Figure 46 the blue node is representing "`context`" of the knowledge graph the light green node is denoting the "`action`" node. The yellow node is showing the "`lang1`" for the programming language. The dark green represents the "`lib`" node or library that the current user marked as a useful library.

Now, let's use the information we have about the current knowledge graph. The first node we want to use is the programming language or "`lang1`". We want to find all other libraries which are marked as the useful library for a knowledge object with the same programming language. We will also count all the number of these Knowledge Objects by using the below Cypher Query in Figure 47, and we will call it the "`intersection`".

```
//Content-Based-41
//step1
MATCH (l:lib)←[:HAS_HELPFUL_Lib]-(k:kotitle)-[:HAS_KO]-(q)-[:HAS_KO]-(reck:kotitle)-[:HAS_HELPFUL_Lib]→(recL:lib)
WHERE k.koid="6028391fc93e287c47ab795f" AND l.name= "wmde/Clock"
WITH l, k, recL, COUNT(q) AS intersection, COLLECT(q.name) AS plang
```

Figure 47: The Cypher Query for the aforementioned task

Then, the next step, step two, is to find all other libraries that are marked as the useful library for the other Knowledge Objects, but we are looking only for those which have a similar context. We will find those "`context`" nodes based on the aforementioned auxiliary data. Regarding 46 we know that the "`context`" node is containing "`current time`". We will define this kind of extra information as auxiliary data. Hence, we can have more accurate result through filtering recommendation result based on the information we gained from the "`context`" node and information about the library, which is already marked as a useful library like "`library name`" and the second knowledge object name based on the "`context`" content demonstrated in Figure 48. It has to be mentioned, at the end of step two, we will use the "`collect`" method from the Neo4j built-in functions to list all the "`context`" nodes as CNTXT.



```
//Content-Based-41
//step1
MATCH (l:lib)←[:HAS_HELPFUL_Lib]-(k:kotitle)-[:HAS_KO]-(q)-[:HAS_KO]-(reck:kotitle)-[:HAS_HELPFUL_Lib]→(recL:lib)
WHERE k.koid="6028391fc93e287c47ab795f" AND l.name= "wmde/Clock"
WITH l, k, recL, COUNT(q) AS intersection, COLLECT(q.name) AS plang
//step2
MATCH (l)←[:HAS_HELPFUL_Lib]-(k)-[:HAS_CONTEXT]-(fc), (sc)-[:HAS_CONTEXT]-(reck:kotitle)-[:HAS_HELPFUL_Lib]→(SrecL:lib)
WHERE fc.name CONTAINS "current time" AND l.name= "wmde/Clock" AND reck.name CONTAINS "current time"
WITH l, k, recL, intersection, plang, fc.name AS FC, sc.name AS SC, fc.koid as FCid, sc.koid as SCid
WITH l, k, recL, intersection, plang, FC, SC, FCid, SCid, COLLECT(SC) AS CNTXT
```

Figure 48: The Cypher Query for the fourth approach, step two.

In step three, we will do the same process for the "`action`" node. Here we will use more auxiliary data. The result will be narrow down via filtering recommended libraries with information we extracted from the current knowledge object "`context`" node.

library, and also "`action`" node. Step three use a two-step "`match`" showing in the below Cypher query:

```
MATCH (l)<-[:HAS_HELPFUL_Lib]-(k)-[:HAS_ACTION]-(fa),
(sa)-[:HAS_ACTION]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(TrecL:lib)
WHERE fa.name CONTAINS "get" AND l.name= "wmde/Clock"
AND reck.name CONTAINS "current time" AND (recL.name CONTAINS "current time")
```

In the above Cypher Query, first, we will find the "`action`" node. For the knowledge Objects which have the same "`action`" node and also their knowledge object has the content of the current knowledge object "`context`", after that all other libraries that are marked as a useful library will be collected as "`ACTN`". We will then have more filtering by narrowing down the results by limiting the libraries' collection by filtering them with the content of the current knowledge object "`context`" node.

Before calculating the Jaccard Index of Similarity, we will compute the Union of the content of "`action`" nodes and "`context`" nodes via the below Cypher Query:

```
ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN) AS union
```

Now it is time to calculate the Jaccard Index of Similarity because we defined the "intersection" and the "union" before, now we can quickly compute it regarding the below Cypher Query:

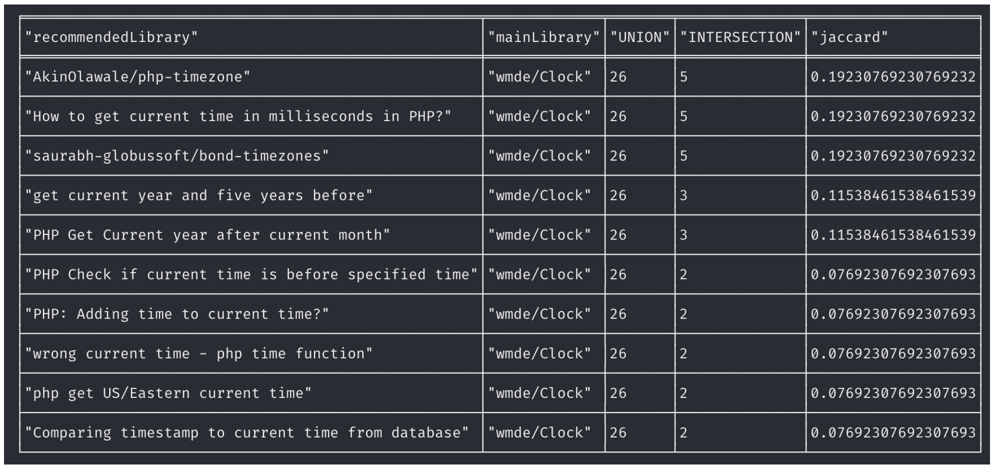
```
((1.0*intersection)/SIZE(union)) AS jaccard
```

And the full Cypher Query will be illustrating in the below query

```
//Content-Based-34342
MATCH (l:lib)<-[:HAS_HELPFUL_Lib]-(k:kotitle)-[:HAS_KO]-
(q)-[:HAS_KO]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(recL:lib)
WHERE k.koid="6028391fc93e287c47ab795f" AND l.name= "wmde/Clock"
WITH l, k, recL, COUNT(q) AS intersection, COLLECT(q.name) AS plang
//step2
MATCH (l)<-[:HAS_HELPFUL_Lib]-(k)-[:HAS_CONTEXT]-(fc),
(sc)-[:HAS_CONTEXT]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(SrecL:lib)
WHERE fc.name CONTAINS "current time"
AND l.name= "wmde/Clock" AND reck.name CONTAINS "current time"
WITH l, k, recL, intersection, plang,
fc.name AS FC, sc.name AS SC, fc.koid as FCid, sc.koid as SCid
WITH l, k, recL, intersection, plang,
FC, SC, FCid, SCid, COLLECT(SC) AS CNTXT
//step3
MATCH (l)<-[:HAS_HELPFUL_Lib]-(k)-[:HAS_ACTION]-(fa),
(sa)-[:HAS_ACTION]-(reck:kotitle)-[:HAS_HELPFUL_Lib]->(TrecL:lib)
WHERE fa.name CONTAINS "get" AND l.name= "wmde/Clock"
AND reck.name CONTAINS "current time"
AND (recL.name CONTAINS "current" OR recL.name CONTAINS "time")
WITH l, k, recL, intersection, plang, FC, FCid, SC, SCid, CNTXT,
fa.name AS FA, sa.name AS SA, fa.koid as FAid,
sa.koid as SAid, COLLECT(sa.name) AS ACTN
```

```
WITH l, k, recL, intersection, plang,
FC, FCid, SC, SCid, CNTXT, FA, SA, FAid, SAid, ACTN
WITH l, k, recL, intersection, plang,FC, FCid, SC, SCid, FA, SA, FAid, SAid,
ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN) AS union, CNTXT, ACTN
WITH DISTINCT recL.name AS recommendedLibrary, l.name AS mainLibrary,
MAX(SIZE(union)) AS UNION, MAX(intersection) AS INTERSECTION
RETURN recommendedLibrary, mainLibrary, UNION, INTERSECTION,
((1.0*INTERSECTION)/UNION) AS jaccard ORDER BY jaccard DESC LIMIT 10
```

The results for the aforementioned problem are demonstrating in 49. In the mentioned Figure, the first column is the recommended library. The second column is the library we are recommending other libraries based on this library. In the first step, we specified this library name. The third column is the Union, and the fourth column is the intersection, and the last column is the Jaccard Index of Similarity.



"recommendedLibrary"	"mainLibrary"	"UNION"	"INTERSECTION"	"jaccard"
"AkinOlawale/php-timezone"	"wmde/Clock"	26	5	0.19230769230769232
"How to get current time in milliseconds in PHP?"	"wmde/Clock"	26	5	0.19230769230769232
"saurabh-globussoft/bond-timezones"	"wmde/Clock"	26	5	0.19230769230769232
"get current year and five years before"	"wmde/Clock"	26	3	0.11538461538461539
"PHP Get Current year after current month"	"wmde/Clock"	26	3	0.11538461538461539
"PHP Check if current time is before specified time"	"wmde/Clock"	26	2	0.07692307692307693
"PHP: Adding time to current time?"	"wmde/Clock"	26	2	0.07692307692307693
"wrong current time - php time function"	"wmde/Clock"	26	2	0.07692307692307693
"php get US/Eastern current time"	"wmde/Clock"	26	2	0.07692307692307693
"Comparing timestamp to current time from database"	"wmde/Clock"	26	2	0.07692307692307693

Figure 49: The result for the content-based filtering using the Jaccard Index of Similarity in Neo4j

Now we have to do the same process to find the best candidate to recommend as expert developers. Hence, again we will use the Jaccard Index of Similarity for that. First, we will take a knowledge object with "`6028391fc93e287c47ab795f`" as the current. We assume the owner of the mentioned knowledge object is already marked another developer "`org1phpdev@sojoudian.net`" as an expert developer and the aforementioned knowledge object has "current time" as the "context" node, now we want first to find the other expert developers who are marked as the expert before for the same programming language and count the number of these result as the through the intersection through below Cypher Query. The graph representation for the result of the below query illustrates in 50.

```
//Content-Based-34-22
//step1
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_KO]-(q)-
[:HAS_KO]-(reck:kotitle)<-[:IS_HELPFUL_FOR]-(recD:dev)
WHERE k.koid="6028391fc93e287c47ab795f"
AND d.name="org1phpdev@sojoudian.net"
WITH d, k, recD, reck, SIZE(()-[:IS_HELPFUL_FOR]->(reck))
AS intersection, COLLECT(q.name) AS plang
```

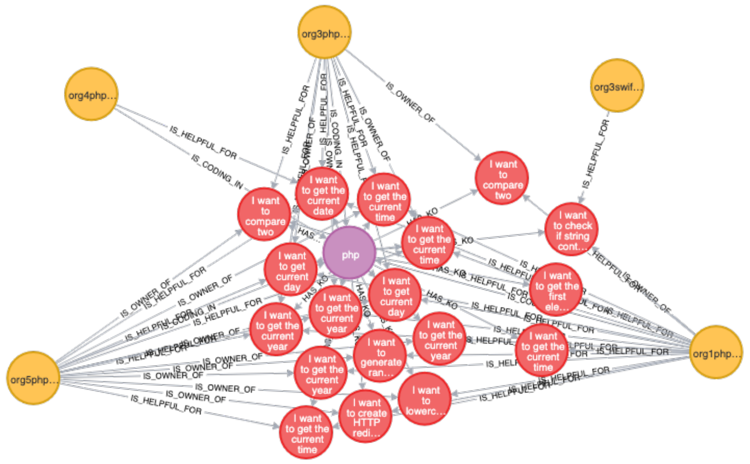

org3php...
org4php...
org3swif...
org5php...
org1php...
php
I want to compare two
I want to get the current date
I want to get the current time
I want to check if string cont...
I want to get current day
I want to get the first ele...
I want to get the current year
I want to generate ran...
I want to create HTTP redi...
I want to lowerc...
IS_HELPFUL_FOR
IS_OWNER_OF
IS_CODING_IN
HAS_KO

Figure 50: The graph result represents the Cypher query to find experts based on similar knowledge objects.

In 50 the yellow nodes denote developers, and the red nodes show knowledge object nodes, and the purple node is to show the programming language.

At the next step, step two, we will find the context nodes, and again here, we will use the auxiliary data to have better results. The flow to find the context begins with finding the current Knowledge Object's "context" node, and then we will find the context of those other knowledge object which has the expert developers with "IS_HELPFUL_FOR" relationship and count and store the result with "collect(cS.name) AS CNTXT". Furthermore, the full Cypher Query for step two will be denoted in the below query, and graph representation for that will be displayed in 51.

```
//step2
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_CONTEXT]-(fc),
(recD:dev)-[:IS_HELPFUL_FOR]->(recK)-[:HAS_CONTEXT]->(cS:context)
WHERE k.koid="6028391fc93e287c47ab795f"
AND (cS.name CONTAINS "current" OR cS.name CONTAINS "time")
WITH reck, recD, intersection, plang, collect(cS.name) AS CNTXT
```

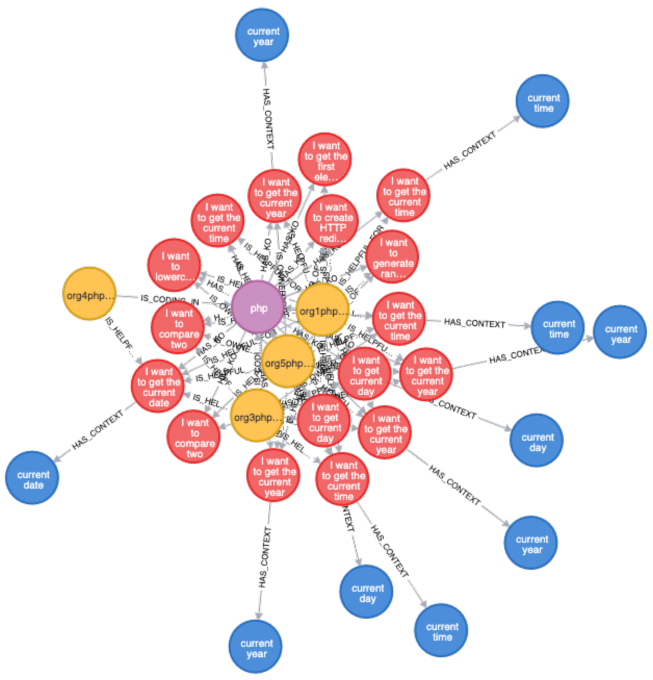
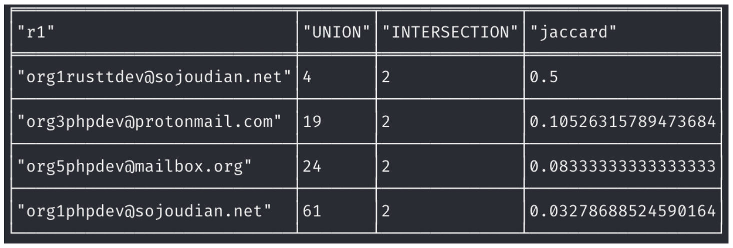


Figure 51: The result for the second step, benefiting from the context nodes.

In 51 the yellow nodes denote developers, and the red nodes show knowledge object nodes, and the purple node is to show the programming language. And blue nodes are representing the "context" nodes.

In the last step, step three, we will find the current Knowledge Object's "action" node. Then we will find other Knowledge Objects "action" and we will count them and collect them as ACTN via "COLLECT(acS.name) AS ACTN". After that, we will compute the union between the "action" nodes and the "context" nodes via "ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN)" and we will make an alias for that as the "union". And in the end, the Jaccard Index of similarity will be calculated by "((1.0*intersection)/SIZE(union)) AS jaccard". Hence, the below query represent the full Cypher Query to reach the expert developers through the Jaccard Index of similarity and sorting the result regarding the Jaccard Index. The actual result is showing in 52.

```
//Content-Based-34-22
//step1
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_KO]-(q)
-[:HAS_KO]-(reck:kotitle)<-[:IS_HELPFUL_FOR]-(recD:dev)
WHERE k.koid="6028391fc93e287c47ab795f"
AND d.name="org1phpdev@sojoudian.net"
WITH d, k, recD, reck, SIZE( ()-[:IS_HELPFUL_FOR]->(reck) )
AS intersection, COLLECT(q.name) AS plang
//step2
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_CONTEXT]-(fc),
(recD:dev)-[:IS_HELPFUL_FOR]->(recK)-[:HAS_CONTEXT]->(cS:context)
WHERE k.koid="6028391fc93e287c47ab795f"
AND (cS.name CONTAINS "current" OR cS.name CONTAINS "time")
WITH reck, recD, intersection, plang, collect(cS.name) AS CNTXT
//step3
MATCH (d:dev)-[:IS_HELPFUL_FOR]->(k:kotitle)-[:HAS_ACTION]-(fa),
(recD:dev)-[:IS_HELPFUL_FOR]->(recK)-[:HAS_ACTION]->(acS:action)
WHERE k.koid="6028391fc93e287c47ab795f"
AND reck.name CONTAINS "current time"
WITH reck, recD, intersection, plang, CNTXT, COLLECT(acS.name) AS ACTN
WITH reck, recD, intersection, plang, CNTXT, ACTN
WITH reck, recD, intersection, plang,
ACTN+filter(x IN CNTXT WHERE NOT x IN ACTN) AS union, CNTXT, ACTN
RETURN  recD.name as r1, SIZE(union) AS UNION, intersection,
((1.0*intersection)/SIZE(union)) AS jaccard
ORDER BY jaccard DESC LIMIT 20
```



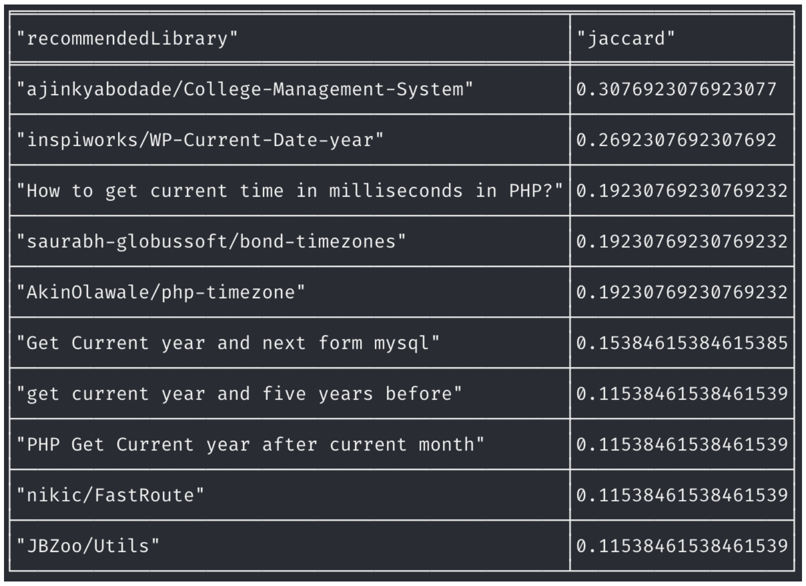
"r1"	"UNION"	"INTERSECTION"	"jaccard"
"org1rusttdev@sojoudian.net"	4	2	0.5
"org3phpdev@protonmail.com"	19	2	0.10526315789473684
"org5phpdev@mailbox.org"	24	2	0.08333333333333333
"org1phpdev@sojoudian.net"	61	2	0.03278688524590164

Figure 52: The result for the content-based filtering using the Jaccard Index of Similarity in Neo4j for finding expert developers

Although the Jaccard index similarity value is more for the traditional version than the proposed version with auxiliary data, if we compare the result, it will be shown that the second method is more accurate than the traditional version. For example, like 53 for the traditional method, we have "`ajinkyabodade/College-Management-System`", "`JBZoo/Utils`", "`nikic/FastRoute`", "`Get Current year and next form mysql`" that are highly dissimilar to what the end-user wants.

Also, there are "`How to get current time in milliseconds in PHP?`", "`AkinOlawale/php-timezone`", "`PHP Get Current year after current month`", which are close to the end-user need but still results can be more accurate.

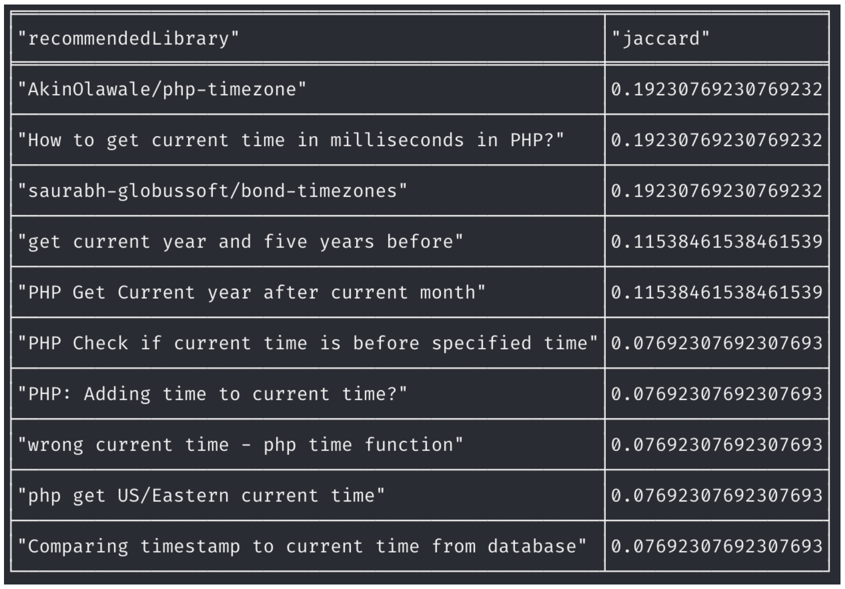
Moreover, the "`ajinkyabodade/College-Management-System`" is a library that was liked for different knowledge objects and had a high value of the Jaccard index similarity in the recommendation list, is not the perfect candidate for an end-user.



"recommendedLibrary"	"jaccard"
"ajinkyabodade/College-Management-System"	0.3076923076923077
"inspiworks/WP-Current-Date-year"	0.2692307692307692
"How to get current time in milliseconds in PHP?"	0.19230769230769232
"saurabh-globussoft/bond-timezones"	0.19230769230769232
"AkinOlawale/php-timezone"	0.19230769230769232
"Get Current year and next form mysql"	0.15384615384615385
"get current year and five years before"	0.11538461538461539
"PHP Get Current year after current month"	0.11538461538461539
"nikic/FastRoute"	0.11538461538461539
"JBZoo/Utils"	0.11538461538461539

Figure 53: The traditional Jaccard Index of similarity recommendation list.

On the other hands, when we are benefiting from auxiliary data like content of the "`context`" node and "`action`" node, we can have closer results to the problem and the library, which is already marked as a helpful library by the end-user. Although there are five identical recommendation items in both recommendation lists, the recommendation candidates' order is significantly critical, like 54. Besides, other recommendation candidates in the list for the second method with auxiliary data are highly leveraged via deriving a benefit from the "`context`" node and the "`action`". For example, "`PHP Check if current time is before specified time`", and other recommended libraries are remarkably close to the knowledge object and the library which is already liked.



"recommendedLibrary"	"jaccard"
"AkinOlawale/php-timezone"	0.19230769230769232
"How to get current time in milliseconds in PHP?"	0.19230769230769232
"saurabh-globussoft/bond-timezones"	0.19230769230769232
"get current year and five years before"	0.11538461538461539
"PHP Get Current year after current month"	0.11538461538461539
"PHP Check if current time is before specified time"	0.07692307692307693
"PHP: Adding time to current time?"	0.07692307692307693
"wrong current time - php time function"	0.07692307692307693
"php get US/Eastern current time"	0.07692307692307693
"Comparing timestamp to current time from database"	0.07692307692307693

Figure 54: The Jaccard index of similarity with auxiliary data recommendation list.

4.4 Collaborative-based Filtering Recommendation

As we studied in chapter two, another technique that tries to solve the recommendation problems is the collaborative-based Filtering technique, which has its own sub methods. The collaborative-based Filtering method's main idea is to deal with information overload to make more accurate predictions and recommendations [8]. Besides, we know that collaborative filtering requires historical data collected from a large number of users and users' feedback on the items that existed on the data collection. Our case study out data set is in the form of a Knowledge Graph, and we have 28 developers as our users. We have 187 user stories on our Knowledge Graph, which those developers asked on our On-demand programming assistance platform and also 1089 libraries from the Stackoverflow and the GitHub. There is also a feedback signal in the On-demand programming assistance, which we know as a "`useful`" library or developer. Our existing developers on the graph are marked as a useful developer 88 times (all of these numbers are when I'm writing this thesis, and it may be growing in the future). And the user feedback will be created on the graph by making a directed relationship from a knowledge object (or user story) to the library with the "`HAS_HELPFUL_Lib`" relationship. According to our knowledge from chapter two, we know that the method focused on using historical data, previous activities, and opinions, like records or purchasing/viewing records, user rating for a particular item, and making a recommendation regardless of the item content. Hence we need a rating system for our Knowledge Graph. We will add rate and score property keys to the graph on the "`HAS_HELPFUL_Lib`" relationship to make this rating system. The current user will add the rate based on their opinion, and the score will be the number of the "`HAS_HELPFUL_Lib` relationship for a library. For example, if a library has marked ten times as a useful library, so it will have ten "`HAS_HELPFUL_Lib` relationships from different Knowledge Objects. Furthermore, we want to find users with similar ratings on libraries and use their tastes to recommend. Respecting chapter two, we know that Collaborative filtering has two main techniques: memory-based and model-based, and these techniques have their own sub methods. Because our problem is to recommend the best libraries and expert developers to the current user, we choose memory-based Collaborative Filtering. This algorithm's focus is to compute the similarity between users or items [39]. Different algorithms can help us make our collaborative-based recommendation system. Still, we found the Cosine similarity much more suitable than the other similarity measurements algorithms regarding the test and data model on our Knowledge Graph.

The Cosine similarity measures the similarity of an inner product space between two vectors. The cosine of the angle between two vectors is determined as the Cosine similarity. This is also the same as the same normalized vectors' inner product, all of which have length 1. The Cosine similarity can be different from 1 to -1, in which 1 is totally similar, and -1 is highly dissimilar. The Cosine will be 1 if the cosine angle is 0°, and the direction is the same; here, the Cosine similarity will be calculate based on the orientation, not the magnitude. The Cosine similarity will be 0 if the cosine angle is 90°. In the positive space, where the result is neatly bound in [0,1], the cosine similarity is especially used [40].

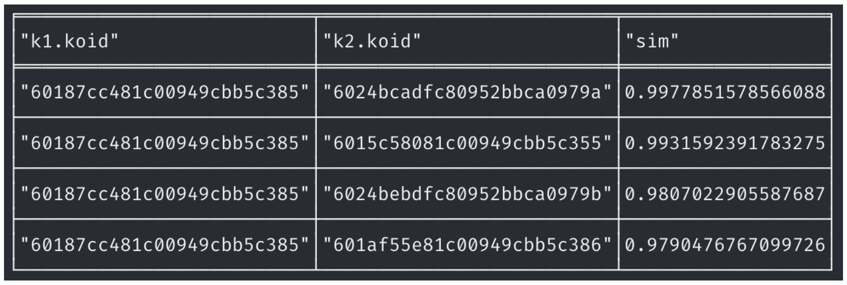
$$similarity(A, B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4}$$

We have two compute the cosine similarity for the current user and other users to recommend them library much closer to their preferences. This similarity calculation will be computed based on the current users' Knowledge Object. Therefore based on similar Knowledge Objects, we can recommend better libraries to the current users. In other words, if we find similar Knowledge Objects and library which was marked as a useful library for them, we can recommend those libraries to the current user for the current Knowledge Object. The process to calculate the Cosine Similarity in Neo4j will be explained as follows. We will first find the current knowledge object and the library that is already marked as a useful library for the current knowledge object with "x" as an alias of the "HAS_HELPFUL_Lib" relationship. Then we will find other Knowledge Objects which marked the same library as a useful library with "y" as an alias of the "HAS_HELPFUL_Lib" relationship. The count of the number of the libraries is aliased as "numberlibs". And the multiplication of the rate value of "x" and "y" then the sum of all of these multiplications will be aliased as "xyDotProduct". After that, we will calculate the square root for all "HAS_HELPFUL_Lib" relationship rate properties. And we will limit the result for the Knowledge Objects with more than two common libraries. Furthermore, the Cypher Query for that will be explained in the below query.

```
//Cosine similarity
MATCH (k1:kotitle {koid:"60187cc481c00949cbb5c385"})
-[x:HAS_HELPFUL_Lib]->(l:lib)<-[y:HAS_HELPFUL_Lib]-(k2:kotitle)
WITH COUNT(l) AS numberlibs, SUM(x.rate * y.rate) AS xyDotProduct,
SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rate) | xDot + a^2)) AS xLength,
SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rate) | yDot + b^2)) AS yLength,
k1, k2 WHERE numberlibs >= 2
```

```
RETURN k1.koid, k2.koid, xyDotProduct / (xLength * yLength) AS sim
ORDER BY sim DESC
LIMIT 100;
```

The above Cypher Query result will be denoted in Figure 55. The first column represents the current Knowledge Object, and the second column shows the other Knowledge Objects and their cosine similarity measurement to the current Knowledge Object.



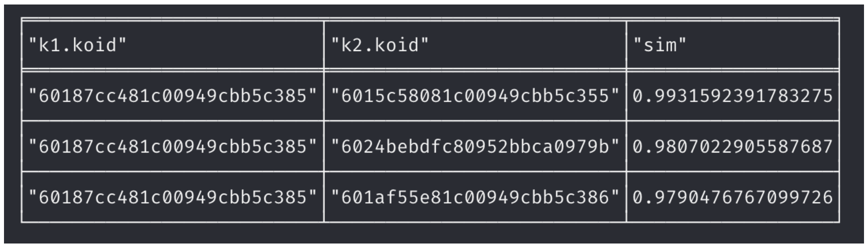
"k1.koid"	"k2.koid"	"sim"
"60187cc481c00949cbb5c385"	"6024bcadfc80952bbca0979a"	0.9977851578566088
"60187cc481c00949cbb5c385"	"6015c58081c00949cbb5c355"	0.9931592391783275
"60187cc481c00949cbb5c385"	"6024bebdfc80952bbca0979b"	0.9807022905587687
"60187cc481c00949cbb5c385"	"601af55e81c00949cbb5c386"	0.9790476767099726

Figure 55: The result for calculating the cosine similarity to find similar Knowledge Objects to the current one for useful library recommendation

If we use more information, which we can extract from the user story, we can have a better result. This extra information is called auxiliary data, like the content of the "`context`" node. In this case, context nodes' is containing the "`current time`". Hence, we can filter our results with this information to provide more accurate results. This filtering will happen through comparing the "`current time`" node data from the first Knowledge Object and the "`current time`" data of the second Knowledge Object. As we know, the "`rate`" property key is set on the relationship from the Knowledge Object and library, which is "`HAS_HELPFUL_Lib`". And the relationship between the "`context`" and the Knowledge Object does not have the "`rate`" property key. Furthermore, we can use this benefit of the knowledge graph and Graph database to calculate the Cosine similarity based on the knowledge object and context node.

And the below query will denote the Cypher Query for the mentioned process, and Figure 56 will be the result for the query.

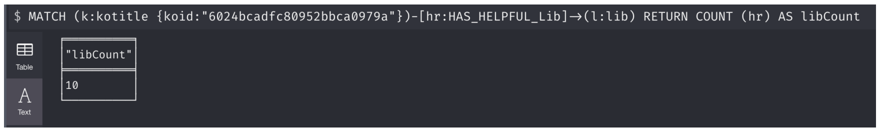
```
//Cosine similarity
MATCH (c1:context)<-[:HAS_CONTEXT]-
(k1:kotitle {koid:"60187cc481c00949cbb5c385"})
-[x:HAS_HELPFUL_Lib]->(l:lib)
<-[y:HAS_HELPFUL_Lib]-(k2:kotitle)-[:HAS_CONTEXT]->(c2:context)
WHERE c1.name="current time" AND c2.name="current time"
WITH COUNT(l) AS numberlibs, SUM(x.rate * y.rate) AS xyDotProduct,
SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rate) | xDot + a^2)) AS xLength,
SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rate) | yDot + b^2)) AS yLength,
k1, k2 WHERE numberlibs >= 2
RETURN k1.koid, k2.koid, xyDotProduct / (xLength * yLength) AS sim
ORDER BY sim DESC
LIMIT 100;
```



"k1.koid"	"k2.koid"	"sim"
"60187cc481c00949cbb5c385"	"6015c58081c00949cbb5c355"	0.9931592391783275
"60187cc481c00949cbb5c385"	"6024bebdfc80952bbca0979b"	0.9807022905587687
"60187cc481c00949cbb5c385"	"601af55e81c00949cbb5c386"	0.9790476767099726

Figure 56: The Cosine similarity measurements after filtering the results via adding context node to the method.

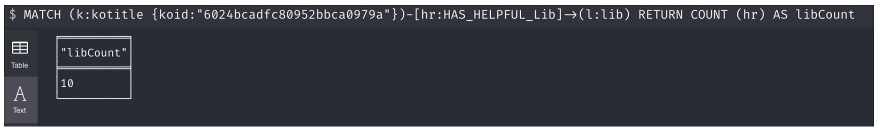
Although the Cosin Similarity in the results is less than the previous method, we have to consider the fact that at the end, we want to recommend libraries that are marked as useful libraries based on the knowledge object similarity. Hence, if the similarity value is less than the previous method, it is not a problem. First, because there is a new sorted list regarding the context node's content and because we used more factors to have a better outcome, we know if the similarity measurement factors increase, then the similarity value will decrease (basically, we are filtering the ordered list with more factors). Second, this method is saying recommend items (libraries) that are useful for other developers regarding their knowledge object and now using this technique, and we can see recommended knowledge objects have the better condition to be recommended like they have marked more libraries as a useful library in comparison with the previous technique. Furthermore, the result will be narrow down, but it will be much closer to the user problem. To investigate the results, in the previous method, the knowledge object with "`6024bcadfc80952bbca0979a`" ID was the first suggestion, which has 10 libraries, Figure 57. But in the new recommendation list, the first knowledge object with "`6024bcadfc80952bbca0979a`" ID has 11 libraries, Figure 58, with more filtering, so then if we apply more filtering through the extra data we extract from the knowledge object, we will have a more satisfying result. It has to be mentioned each knowledge object in the second list has a more useful library count than their pair knowledge Object in the first list.



```
$ MATCH (k:kotitle {koid:"6024bcadfc80952bbca0979a"})-[hr:HAS_HELPFUL_Lib]→(l:lib) RETURN COUNT (hr) AS libCount
```

```
"libCount"
10
```

Figure 57: The Cosine similarity measurements after filtering the results via adding context node to the method.



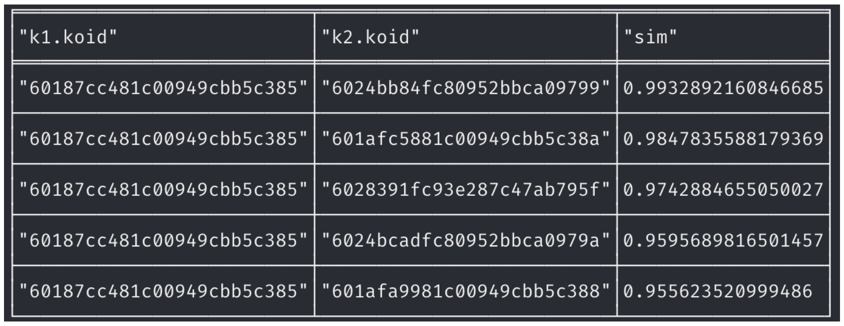
```
$ MATCH (k:kotitle {koid:"6024bcadfc80952bbca0979a"})-[hr:HAS_HELPFUL_Lib]→(l:lib) RETURN COUNT (hr) AS libCount
```

```
"libCount"
10
```

Figure 58: The Cosine similarity measurements after filtering the results via adding context node to the method.

The same process will be done for the expert developer recommendation based on the similarity of the current knowledge object and the other Knowledge Objects excited on the Knowledge Graph. But this time, the relationship that will be used to calculate the similarity measurement is the "`IS_HELPFUL_FOR`" relationship. The property key used is also the same as before rate property. We found the similarity of the other Knowledge Objects to the current Knowledge Object, and we can recommend the other expert developers they find useful for their user stories (Knowledge Objects) with more than two common expert developers. The Cypher Query for this process will be denoted in the below query, and the result will be illustrated in Figure 59.

```
//Cosine similarity
MATCH (k1:kotitle {koid:"60187cc481c00949cbb5c385"})
<-[x:IS_HELPFUL_FOR]-(d:dev)-[y:IS_HELPFUL_FOR]->(k2:kotitle)
WITH COUNT(d) AS numberlibs, SUM(x.rate * y.rate) AS xyDotProduct,
SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rate) | xDot + a^2)) AS xLength,
SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rate) | yDot + b^2)) AS yLength,
k1, k2 WHERE numberlibs >= 2
RETURN k1.koid, k2.koid, xyDotProduct / (xLength * yLength) AS sim
ORDER BY sim DESC
LIMIT 100;
```



"k1.koid"	"k2.koid"	"sim"
"60187cc481c00949cbb5c385"	"6024bb84fc80952bbca09799"	0.9932892160846685
"60187cc481c00949cbb5c385"	"601afc5881c00949cbb5c38a"	0.9847835588179369
"60187cc481c00949cbb5c385"	"6028391fc93e287c47ab795f"	0.9742884655050027
"60187cc481c00949cbb5c385"	"6024bcadfc80952bbca0979a"	0.9595689816501457
"60187cc481c00949cbb5c385"	"601afa9981c00949cbb5c388"	0.955623520999486

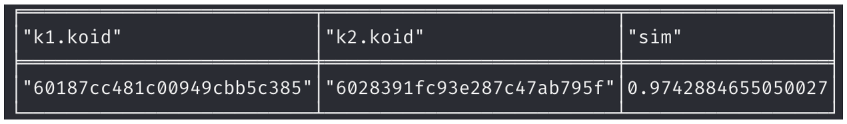
Figure 59: The result for calculating the cosine similarity to find similar Knowledge Objects to the current one for expert developer recommendation

Now we can apply more filtering to find more accurate results, like what we did for recommending the useful libraries. Again, we will gain benefits from auxiliary data, the content of the "`context`". The process for the mentioned method will be illustrated in the below Cypher query:

```
//Cosine similarity
MATCH (c1:context)<-[:HAS_CONTEXT]-
k1:kotitle {koid:"60187cc481c00949cbb5c385"})<-[x:IS_HELPFUL_FOR]-
(d:dev)-[y:IS_HELPFUL_FOR]->(k2:kotitle)-[:HAS_CONTEXT]->(c2:context)
```

```
WHERE c1.name="current time" AND c2.name="current time"
WITH COUNT(d) AS numberlibs, SUM(x.rate * y.rate) AS xyDotProduct,
SQRT(REDUCE(xDot = 0.0, a IN COLLECT(x.rate) | xDot + a^2)) AS xLength,
SQRT(REDUCE(yDot = 0.0, b IN COLLECT(y.rate) | yDot + b^2)) AS yLength,
k1, k2 WHERE numberlibs >= 2
RETURN k1.koid, k2.koid, xyDotProduct / (xLength * yLength) AS sim
ORDER BY sim DESC
LIMIT 100;
```

The Result for the above query will be denoted in Figure 60.



"k1.koid"	"k2.koid"	"sim"
"60187cc481c00949cbb5c385"	"6028391fc93e287c47ab795f"	0.9742884655050027

Figure 60: The Cosine similarity with more filtering using context node to find better expert developers.

4.5 Conclusion

In this chapter, we defined what should we recommend in our platform and what is the best recommendation item. Also, we explained how expert developers' graph and useful libraries graph are constructed. Then, we explained four different content-based filtering algorithms and which one is better to choose. We showed that the content-based filtering recommendation is used to deal with the cold-start problem and the collaborative-based filtering technique is used in the situations where we have enough historical data. We also showed that the proposed platform brings more accurate results in comparison with standard baseline algorithms.

Chapter 5

Conclusion

5.1 Conclusion

This thesis introduces a real-time graph-based recommendation system for programming tasks. It includes a deep investigation of the different recommendation techniques in order to help the end-users find well-suited solutions for their programming problems. To achieve this aim, we proposed an on-demand programming assistance platform equipped with a robust recommendation engine capable of recognizing the user's request and combining content-based filtering and collaborative-based filtering techniques. Using our programming assistance platform, end-users who are developers can ask their questions, and the platform can provide solutions from two major programming sources: namely, Stackoverflow and GitHub. The platform uses various technologies and tools to receive the end-users inputs and extract useful information from it. Also, an efficient data model is proposed to store the end-users inputs in a graph database. The aforementioned information includes different users' programming problems constructing the knowledge graph based on the data model. Moreover, the platform includes expert developers and libraries from Stackoverflow and GitHub modeled as a sub-graph of the original knowledge graph. The sub-graph is constructed based on another data model that leverages the main model. To evaluate the platform, 191 programming questions/knowledge objects were asked on the platform from 29 developers for 8 programming languages (including Java and Python). Various algorithms using content-based filtering and collaborative-based filtering methods are applied to the dataset to reach the best recommendation candidate list which is the closest to the end-users interests.

The traditional Jaccard index similarity algorithm is used in the content-based filtering recommendation process. Compared with standard baseline algorithms, our method showed more accurate results when auxiliary data extracted from the knowledge objects were used. For collaborative filtering, the cosine similarity algorithm was applied. Here again, more accurate results have been obtained when the auxiliary data were considered. The recommendation system is responding to the end-users in a real-time manner as it does not require a training phase.

5.2 Future Work

After building the pipeline that gets the knowledge objects automatically and constructing the knowledge graph, our first plan for future research is to investigate various data models and approaches to construct a richer graph. We aim to gather more programming problems in different programming languages and make more adequate recommendations.

Another significant path for further investigation is analysing and applying other graph algorithms, such as centrality algorithms and community detection algorithms on the knowledge graph and then comparing the results. Moreover, we can create a performance measurement and a performance comparison method for the mentioned algorithms. Finally, applying reinforcement learning to find the best policy for the recommendation system considering end-users defined rewards is a promising future problem.

Bibliography

[1] J. J. Miller, "Graph database applications and concepts with Neo4j," *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, vol. 2324, p. 36, 2013.

[2] A. Virk and R. Rani, "recommendations using graphs on Neo4j," *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 133–138, 2018.

[3] X. Su and T. M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Advances in Artificial Intelligence*, pp. 1–9, 2009.

[4] R. Wita, K. Bubphachuen, and J. Chawachat, "Content-Based Filtering Recommendation in Abstract Search Using Neo4j," *ICSEC 2017 - 21st International Computer Science and Engineering Conference 2017, Proceeding*, vol. 6, pp. 136–139, 2018.

[5] visual paradigm, "Theme vs epic vs user story vs task," https://www.visual-paradigm.com/scrum/theme-epic-user-story-task/, 2020, accessed: 2020-12-16.

[6] R. Chen, Q. Hua, Y. S. Chang, B. Wang, L. Zhang, and X. Kong, "A survey of collaborative filtering-based recommender systems: from traditional methods to hybrid methods based on social networks," *IEEE Access*, vol. 6, pp. 64 301–64 320, 2018.

[7] Y. Shi, M. Larson, and A. Hanjalic, "Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges," *ACM Computing Surveys, ACM 2014*, pp. 1–45, 2014.

[8] M. R. McLaughlin and J. L. Herlocker, "A collaborative filtering algorithm and evaluation metric that accurately model the user experience," *Proceedings of Sheffield SIGIR - Twenty-Seventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 329–336, 2004.

[9] Z. Yang, B. Wu, K. Zheng, X. Wang, and L. Lei, "A survey of collaborative filtering-based recommender systems for mobile internet applications," *IEEE Access*, pp. 1–15, 2016.

[10] Sarwar., K. Badrul, K. George, R. Joseph, and John., "Item-Based Collaborative Filtering Recommendation Algorithms," *International Conference on World Wide Web WWW'01*, pp. 1–11, 2001.

[11] P. Mathew, B. Kuriakose, and V. Hegde, "Book Recommendation System through content based and collaborative filtering method," *Proceedings of 2016 International Conference on Data Mining and Advanced Computing, SAPIENCE 2016*, pp. 47–52, 2016.

[12] M. J. Pazzani, "Framework for collaborative, content-based and demographic filtering," *Artificial Intelligence Review*, vol. 13, no. 5, pp. 393–408, 1999.

[13] S. Zhang, W. Wang, J. Ford, F. Makedon, and J. Pearlman, "Using singular value decomposition approximation for collaborative filtering," *Proceedings - Seventh IEEE International Conference on E-Commerce Technology, CEC 2005*, vol. 2005, pp. 257–265, 2005.

[14] Y. Shoham, "Fab:Content-Based, Collaborative Filtering for Recommendation," *Communications of the ACM*, vol. 40, no. 3, 1997.

[15] P. Thorat, R. M. Goudar, and S. Barve, "Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System," *International Journal of Computer Applications*, vol. 110, no. 4, pp. 31–36, 2015.

[16] M. De Gemmis, P. Lops, G. Semeraro, and P. Basile, "Integrating tags in a semantic content-based recommender," *RecSys'08: Proceedings of the 2008 ACM Conference on Recommender Systems*, pp. 163–170, 2008.

[17] C. Basu, H. Hirsh, and W. Cohen, "Recommendation as classification: Using social and content-based information in recommendation," *Proceedings of the National Conference on Artificial Intelligence*, pp. 714–720, 1998.

[18] M. A. Rodriguez and P. Neubauer, "Constructions from dots and lines," *Bulletin of the American Society for Information Science and Technology*, vol. 36, no. 6, pp. 35–41, 2010.

[19] D. Q. G. L. Udsk, Y. Duan, and Z. Lin, "Specifying architecture of knowledge graph with data graph, information graph, knowledge graph and wisdom graph," *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 327–332, 2017.

[20] G. Bellinger, D. Castro, and A. Mills, "Data, information, knowledge, and wisdom," 2004.

[21] W. Shalaby, B. E. Alaila, M. Korayem, L. Pournajaf, K. Aljadda, S. Quinn, and W. Zadrozny, "Help Me Find a Job: A Graph-based Approach for Job Recommendation at Scale," *arXiv*, pp. 1544–1553, 2017.

[22] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[23] H. Yildirim and M. S. Krishnamoorthy, "A random walk method for alleviating the sparsity problem in collaborative filtering," in *Proceedings of the 2008 ACM conference on Recommender systems*, 2008, pp. 131–138.

[24] H. Chen, X. Li, and Z. Huang, "Link prediction approach to collaborative filtering," in *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05)*. IEEE, 2005, pp. 141–142.

[25] T. Zhou, J. Ren, M. Medo, and Y.-C. Zhang, "Bipartite network projection and personal recommendation," *Physical review E*, vol. 76, no. 4, p. 046115, 2007.

[26] C. C. Aggarwal, J. L. Wolf, K.-L. Wu, and P. S. Yu, "Horting hatches an egg: A new graph-theoretic approach to collaborative filtering," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999, pp. 201–212.

[27] W. Li, Z. Ye, M. Xin, and Q. Jin, "Social recommendation based on trust and influence in sns environments," *Multimedia Tools and Applications*, vol. 76, no. 9, pp. 11 585–11 602, 2017.

[28] S. Ray and A. Mahanti, "Improving prediction accuracy in trust-aware recommender systems," in *2010 43rd Hawaii international conference on system sciences*. IEEE, 2010, pp. 1–9.

[29] S. Roy and B. Ravindran, "Measuring network centrality using hypergraphs," in *Proceedings of the Second ACM IKDD Conference on Data Sciences*, 2015, pp. 59–68.

[30] M. Cha, H. Haddadi, F. Benevenuto, and K. Gummadi, "Measuring user influence in twitter: The million follower fallacy," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 4, 2010.

[31] G. Bathla, H. Aggarwal, and R. Rani, "A graph-based model to improve social trust and influence for social recommendation," *The Journal of Supercomputing*, vol. 76, no. 6, pp. 4057–4075, 2020.

[32] B. Stark, C. Knahl, M. Aydin, M. Samarah, and K. O. Elish, "BetterChoice: A migraine drug recommendation system based on Neo4J," *2017 2nd IEEE International Conference on Computational Intelligence and Applications, ICCIA 2017*, vol. 2017-Janua, pp. 382–386, 2017.

[33] L. Lü, M. Medo, C. H. Yeung, Y.-C. Zhang, Z.-K. Zhang, and T. Zhou, "Recommender systems," *Physics reports*, vol. 519, no. 1, pp. 1–49, 2012.

[34] R. Catherine and W. Cohen, "Personalized recommendations using knowledge graphs: A probabilistic logic programming approach," *RecSys 2016 - Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 325–332, 2016.

[35] Dandelion, "The entity extraction api reference," https://dandelion.eu/docs, 2020, accessed: 2020-12-21.

[36] spacy, "spacy 101: Everything you need to know," https://spacy.io/usage/spacy-101, 2020, accessed: 2020-12-21.

[37] P. Jaccard, "Jaccard$_1902(method),''$ ***Bulletin de la Murithienne***$, vol. XXXVII, pp. 81-$
$-92., 1982.$

[38] V. Verma and R. K. Aggarwal, "A comparative analysis of similarity measures akin to the Jaccard index in collaborative recommendations: empirical and theoretical perspective," *Social Network Analysis and Mining*, vol. 10, no. 1, pp. 1–16, 2020. [Online]. Available: https://doi.org/10.1007/s13278-020-00660-9

[39] H. Liu, Z. Hu, A. Mian, H. Tian, and X. Zhu, "A new user similarity model to improve the accuracy of collaborative filtering," *Knowledge-Based Systems*, vol. 56, pp. 156–166, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2013.11.006

[40] "Understanding cosine similarity and its application," https://towardsdatascience.com/understanding-cosine-similarity-and-its-application-fd42f585296a.