Bilkent University

Computer Engineering

Senior Design Project

T2309

Expert Developer Finder

Analysis and Requirement Report

Team Members
Ali Eren Günaltılı - 21801897 - eren.gunaltili@ug.bilkent.edu.tr
Bora Altınok - 21902206 - bora.altinok@ug.bilkent.edu.tr
Ceyda Şahin - 21903448 - ceyda.sahin@ug.bilkent.edu.tr
Ege Ergül - 21902240 - ege.ergul@ug.bilkent.edu.tr
Tuna Dalbeler - 21802539 - tuna.dalbeler@ug.bilkent.edu.tr

Supervised by Eray Tüzün
Innovation Expert Eray Tüzün
Course Instructed by Erhan Dolak and Tağmaç Topal

March 13th, 2023

1. Introduction

1.1 Purpose of the system

With the COVID-19 pandemic, the entire world shifted its infrastructures towards a more remote environment and adjusted its policies accordingly. IT jobs got affected the most by this transition and many IT companies are planning on going entirely remote working even after the end of the pandemic [1]. This entire remote working strategy causes late adaptation of a junior developer to the working environment and colleagues. In a physical office, junior developers are more likely to communicate with senior developers and can get help from them when they are having difficulties with the code base which is new for them. On the other hand, knocking on the doors of senior developers may not be easy in remote environments for junior developers since the process takes a long time and the response ratio is expected to be less. Response time to their questions is crucial for junior developers as it accelerates the onboarding process. Thus, the time that companies spend on their fresh starters in the onboarding process, is likely to decline. From a company's point of view, minimizing the onboarding time is a great opportunity to save money and make more profit. From the developers point of view, the process of adjusting to a new code base can be frustrating and getting the correct help rapidly will facilitate their jobs. For these concerns, we are implementing the Expert Developer Finder.

Expert Developer Finder is a Visual Studio Code extension that has a single primary functionality of finding the best, the most knowledgeable experts for a specified code part. The purpose of Expert Developer Finder is to address the above-mentioned concerns: reducing the time and therefore the cost of on-boarding process, helping developers to adjust to a new codebase and as a consequence, result in healthier projects.
Also it should be noted that the user target is not limited to junior developers or fresh starters. Even senior developers can use the system as well. Since there is a rapid turnover of proficiency (i.e. every file is constantly modified and the expert of a given codebase changes rapidly), even senior developers can have difficulty adjusting to a codebase and may need help to get used to it faster. Last, of all, such a system is vital for the reusability of the software as well. Consulting people who are not experts in the given code base can result in even more harm than it causes good. Thus, we believe that finding the correct expert for a code base is vital. In this report, the proposed product regards these concerns and comes up with a renovative and innovative solution.

1.2 Design goals

The top three design goals of ExDev are performance, maintainability and usability.

1.2.1 Usability

Since ExDev has one main functionality, we want to retrain from a complicated user interface and use-flow. That is, creating an account and starting to get help from experts should be straightforward and users should be able to figure out using our system without needing any further guidance / tutorials. Otherwise, if the user experience of our system is poor, users may feel irritated and overwhelmed and may stop using our system. Therefore, usability shows itself as the most important design choice.

To achieve a good user experience, our frontend team will be learning the basic principles of design. Also, our system's front end will make a responsive design for the web management tool, always prioritizing the *mobile first approach*. To test the usability of our system, the feedback that we will be receiving from real developers will be considered.

1.2.2 Maintainability

In the following times of this semester, after we are ready to release a beta version of ExDev, we are hoping to get our system used by real developers for getting their feedback; and based on their feedback, we will be improving our system until the end of this semester. Therefore, we will have to modify our system in a limited time. This is why the maintainability of our system, and our code becomes a priority.

In our design, we are aiming to achieve high cohesion and loose coupling among the project files/ modules and good usage of encapsulation (needed for facilitated debugging). This way, a change will not cause unintended consequences and crash some other part of the system. Decompositions of the systems will be done as thoroughly as possible to ensure this goal.

To test the maintainability of our system, we will check how long it takes for our team to respond to a reported bug by the real users. We estimate a one-week range for fixing a bug.

1.2.3 Performance

Since the entire value that ExDev offers is coming from the promise we make about speeding up the on-boarding process by providing help to developers faster, performance is the third most crucial aspect of our design. We will be mining the entire history of big projects and then, we will run queries on an artifact traceability graph that will include various artifacts from the entire history. In other words, we will be working with great amounts of data. Thus, following conventional and intuitive methods may result in a bad performance. Therefore, we will be putting the effort in finding ways to speed up the performance of our system so that it will meet the performance requirements.

Our performance goals are as follows: creating the artifact traceability graph should finish over a night (approximately 8 hours). Returning results to a query (making a recommendation) should not take more than 5 minutes. None of the functionalities of the web management tool should take up more than 30 seconds. To test the performance of ExDev, we will compare the actual run-time values of the above-mentioned operations with these values.

1.3 Definitions, acronyms, and abbreviations

In the rest of the report, the following names will be used to refer to specific components of our system. To maintain consistency and prevent any further confusion, these terms should be noted at the beginning of the report, not at the glossary..

ExDev: This term is a shorter version of Expert Developer Finder. ExDev and Expert Developer Finder will be used interchangeably to refer to our system.

Web Management Tool: This term is used to refer to the web frontend of our project. That is, in the analysis report, we had mentioned that there will be a web part of our system where new users will be able to register to our system and create / join repositories. The frontend of this web part will be referred to as the *Web Management Tool*.

Server: This term will be used to refer to the component that will connect every sub-system of ExDev. That is, it will be the server who will read/write data from/to MongoDb and return a response to the Web Management Tool. Also, it will be again the server who will get the query request from VS Code extension, then trigger the recommendation engine, get the recommended experts and return the VS Code extension. More detail about this structure will be provided in section 3.2. However, it is important to note that the server is not interchangeable with the *recommendation engine* (see below).

Recommendation Engine: This term will be used for referring to the sub-system where or system runs queries on the artifact traceability graph and finds the most knowledgeable experts.

Please note that the term *backend* results in confusions in our system (e.g. when we say backend, one can understand both the server or the recommendation engine). Thus, we are retraining from using this term. However, the backend of our project can be considered as the combination of server and the recommendation engine.

1.4 Overview

The rest of this report is structured as follows: In section two, the current alternatives of ExDev are mentioned. In section three, the proposed software architecture is described including the subsystem decomposition, hardware to software mapping, persistent data management and access control and security choices. In section four, our subsystems are explained in more detail. In section five, the test cases of our project are described. In section six, various factors that we considered during our design process are evaluated. In section seven, information on how we manage the team-work is provided. This reports ends with sections eight: glossary and section nine: references.

2. Current software architecture

Recall that the problem that Expert Developer Finder intends to overcome is that it is hard and time consuming to find the most knowledgeable expert for a given code part. Also, when the true expert is not found, the information gathered from the wrong experts can be misleading, causing an unhealthy software development process. Currently, the best practices to avoid these problems are the efforts to figure out the best expert manually (by asking) and to check out the last editor of the code base with “git blame” command or GitLens.

2.1 Manual Process to Find the Best Expert

Manual process to find the best expert is the initial solution addressing the problem declared above. Consider a junior developer who started working in a repository recently. When they struggle adjusting to the repository, most intuitively they will ask questions to their team members, colleagues, team leads and/or their supervisors. The developer that junior developers had consulted to can either redirect them to other more knowledgeable developers or can try to help them as much as they could. However, there are a few fallbacks in this method. Firstly, if the developer who answers the questions of the junior developer is not the true expert of that specific area, the information he/she provides to the junior developer may not be sufficient or completely accurate. Or the junior developer and the other developer who helps the junior developer may together figure out the code part together but this situation most probably takes a long time, wasting both the developer’s and the incorrect expert’s time. Another scenario is that the junior asks about a code part to their colleagues/ team-leads and etc. and they may be redirected. The developer whom the junior developer was redirected to may also redirect them to others. This may go on like this and the whole process takes a long time. Thus, the manual process for finding the best expert is not addressing this problem as it has more than many fall backs and insufficiencies.

2.2 Git Blame and GitLens

Another current attempt to overcome the above-defined problem is checking out the last modifier of the code part of interest. This solution is a less intuitive and a less popular solution when compared to the previous solution. This is because git blame and GitLens are unfamiliar commands and tools to many developers.

Git Blame [2] is a git command that shows the last commit’s author that modified the given code part. GitLens [3] is a tool that has many functionalities, one of which is to show the last modifier of the specified code part. Another useful feature that GitLens offers is that with using GitLens, developers can see every commit that acted on a file and compare each commit of that file manually. By using these features, junior developers can find a developer to whom they may consult. However, like the previous solıution, this method also has its fallbacks. First of all, when junior developers reach to the last modifier of a file to ask questions about that file by using git blame command or GitLens, they may reach developers who don’t have sufficient knowledge about that file. For example, a developer might have only modified method A in the file B.js. If the junior developer has questions about the whole file, reaching to developer C who has the most recent commit editing file B may not be enough since developer C isn't knowledgeable about the rest of the file. Another possible (yet possibly less likely) scenario is as follows: The developer A is responsible for "cleaning" the folder B. He/she formats every single file in folder B and deletes the unused print statements. Thus, even though developer A has no understanding about the content of these files, the most recent modifier of those files is them. Such edge cases may result in inconveniences. It is possible to come up with many possible alternative scenarios. Yet, it is more than safe to state the following claim: "The last editor of a code part is **not** necessarily the most knowledgeable expert of that code part". Thus, this solution also is not error-prone and completely overcomes the predefined problem.

To conclude, although there are two possible methods that address the problem of finding the best expert for a given code part, both of these methods have their own fallbacks. Thus, it is necessary to come up with a more concrete solution that will directly solve this issue.

3. Proposed software architecture

3.1 Overview

Expert Developer Finder is a Visual Studio Code extension that has a single primary functionality of finding the best, the most knowledgeable experts for a specified code part. Yet, Expert Developer Finder also has other secondary functionalities to help developers get familiar with a new code base more rapidly.

3.2 Subsystem decomposition

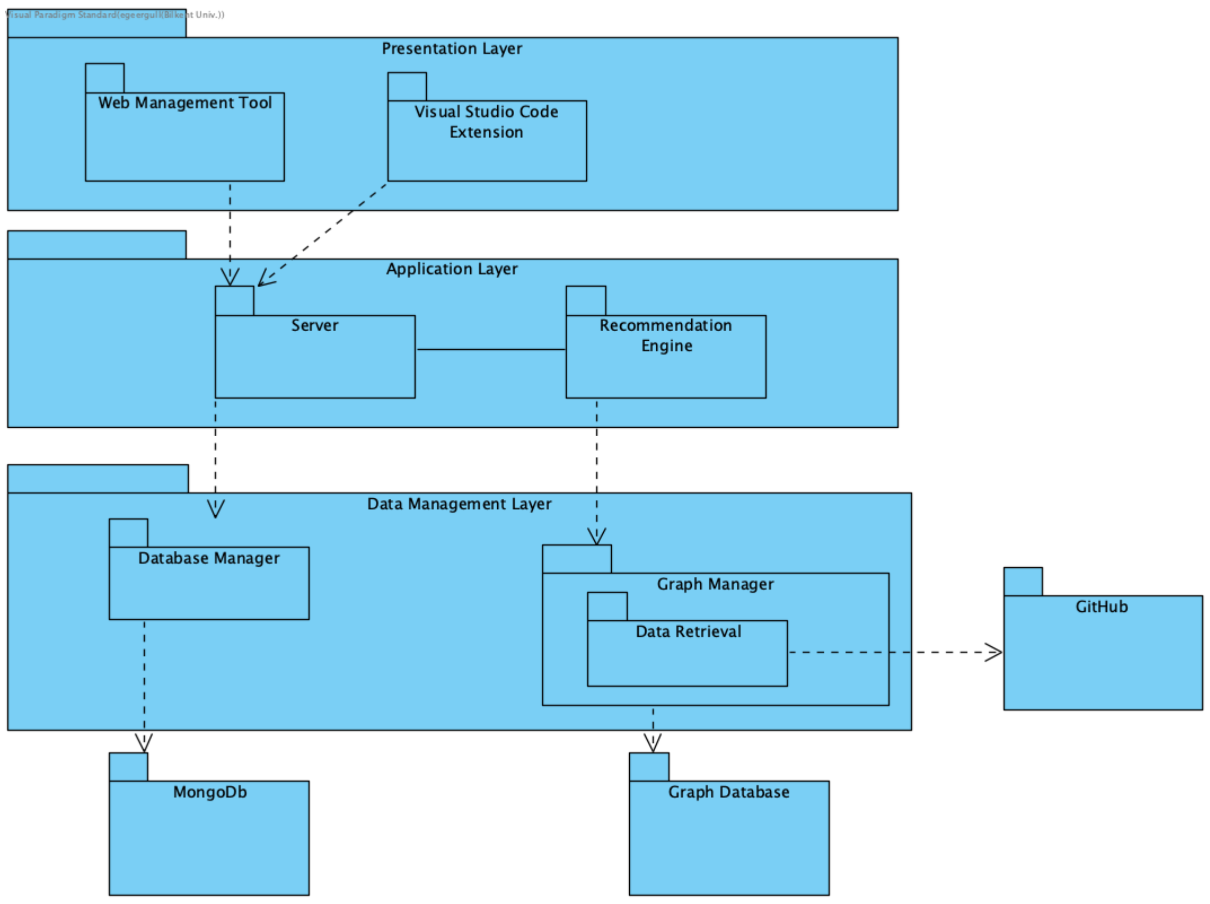

Figure 1. Subsystem decomposition of ExDev

In order to decompose our subsystems, we used a multi-tier design. To retain a high level of abstraction, which has previously been mentioned as one of our top design priorities, we chose to employ three layers. When there are three layers, there is a significant reduction in coupling between subsystems, which may slightly impact the performance but beneficial to maintainability (see 3.1.5 for the desired design trade-offs). The frontend is represented by the topmost layer, while the backend is represented by the bottom two levels.

The top level layer is the presentation layer where the frontend components of ExDev take place, namely the web management tool and the VS Code extension. There are two subsystems in this layer, both of which are different projects that will have a different build. Therefore, there is no dependency between them. Web management tool will be the frontend code which is a subsystem by its own. Similarly, VS Code extension is the frontend code which is a subsystem by its own too. These subsystems will be communicating with the server (see below) only.

The second layer is the application layer, which is responsible for the logic of our application. In other words, the application layer acts like the brain of our system. According to the requests it receives from the presentation layer, it takes the necessary actions. The application layer has two subsystems in it: Database manager and the recommendation engine.

The server is the subsystem that is responsible for communicating with the presentation layers. For example, a user can use the web management tool to register. The register request will be sent to the server where the server will handle this request. Server will also be communicating with the database manager subsystem, which will eventually manage the MongoDb database. Last of all, the server will be communicating with the recommendation engine. For example, when a query from the VS Code extension arrives to the server, the server will pass this query to the recommendation engine, then wait for the results and send the results back to the VS Code.

The recommendation engine is the second subsystem that takes place in the application layer. This layer will be responsible for making a recommendation. The recommendation algorithm and its application/ logic will take place here. It will be in touch with the graph manager subsystem, which will be the interface of the graph.

The last layer is the data management layer. In this layer, there are two subsystems: Database manager and the graph manager. These subsystems can be considered as the interfaces of two different databases. The application layer will use these “interfaces” for managing the database, which will create a high-level abstraction. The database manager subsystem is responsible for making CRUD operations and running various queries on the MongoDb database. An example query could be like this, get me the repositories of this user. The graph manager subsystem is the last subsystem of ExDev. It is very similar to the database manager. The only difference is that the graph manager manages the graph database Neo4j, not the MongoDb database. It is responsible for doing the CRUD operations and running various queries on the graph database. An example query would be like this: return me the number of connections from user node A, and the file node B. It is also seen that the graph manager is also in contact with GitHub as well. This part is done by the data retrieval subsystem located in the graph manager subsystem. This is because the last functionality of the graph manager subsystem is to create and update the graph database by fetching data from GitHub REST API. That is, when the recommendation engine asks the graph manager to create the artifact traceability graph of a given repository, this subsystem will mine that repository’s history by using the GitHub REST API and then create a graph in the graph database.

With this multi-tier design approach, we aimed to achieve a high abstraction between the units. By bearing in mind the KISS principle (keep it simple and stupid), we wanted each subsystem to have a highly-specialized and specific responsibility. This way, development of these subsystems can be assigned to different team members easily. Furthermore, modifying a sub-system will be possible without affecting other subsystems. This way, maintaining our system will be easier, which is the second design goal we had in mind.

3.3 Hardware/software mapping

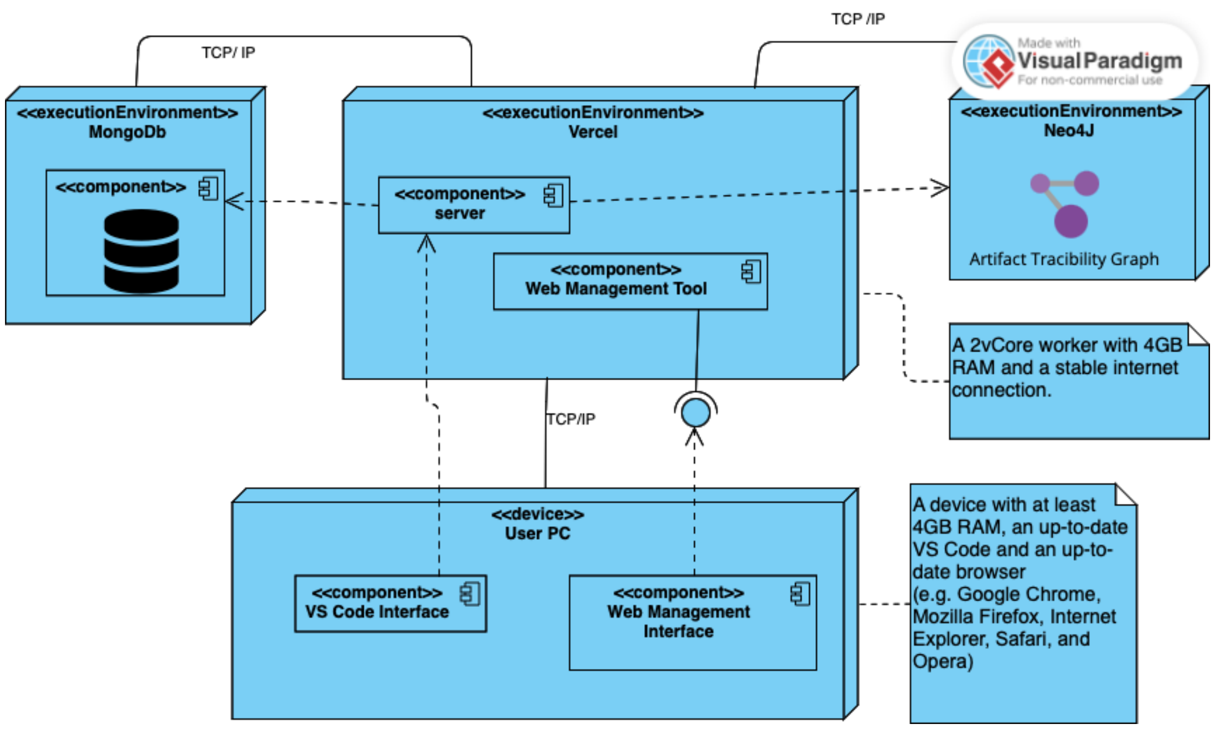
TCP/ IP
TCP /IP
Made with Visual Paradigm For non-commercial use
<<executionEnvironment>> MongoDb
<<component>>
<<executionEnvironment>> Vercel
<<component>> server
<<component>> Web Management Tool
<<executionEnvironment>> Neo4J
Artifact Tracibility Graph
A 2vCore worker with 4GB RAM and a stable internet connection.
TCP/IP
<<device>> User PC
<<component>> VS Code Interface
<<component>> Web Management Interface
A device with at least 4GB RAM, an up-to-date VS Code and an up-to-date browser (e.g. Google Chrome, Mozilla Firefox, Internet Explorer, Safari, and Opera)

Figure 2. Hardware to Software Mapping Diagram

ExDev does not require any specialized hardware components that will be designed for a specific purpose. Therefore, in this section, we will be mentioning the hardware requirements of the devices where our system will be running.

We will be developing the web management tool of ExDev with React v16.14.0. Hence, for being able to run the web management tool, devices with web browsers who can support React v16.14.o are needed. Some of such browsers that we promise that will support our web management tool are Google Chrome, Mozilla Firefox, Internet Explorer, Safari, and Opera in their latest versions. It is possible to use our web management tool from both computers or mobile phones. For running web management tools from a computer, a keyboard, a mouse and a monitor are needed whereas for running web management tools on a mobile phone, a working touch screen is needed.

For being able to use the VS Code extension of ours, a computer with a keyboard, mouse and a monitor is needed. The extension will be developed with VS Code API version 1.76. Therefore, to use our extension, an up-to-date version of VS Code will be needed.

We estimated that a machine with at least 4GB RAM will be enough for being able to run both the web management tool and our extension at the same time. To come up with this number, we created a virtual machine with 4GB of RAM in our computers and tested our application on whether they are working or not. Both the web management tool and the extension could be runned in a virtual machine with 4 GB of RAM. However, we noticed that a virtual machine with 8GB of RAM performed better. Therefore, we recommend using a device with at least 8GB of RAM for a better performance.

The backend of our project will be hosted in a free hosting platform called Vercel. The subsystems of server, recommendation engine, database manager and graph manager will be one single executable project. In the figure 2, this executable is referred to as the server in the Vercel executable environment. The server is using Node.js v18.11.0. Also, the web management tool project will be another executable which will be hosted again with Vercel. The hardware device that will be hosting both of our executables is expected to be a 2vCore worker with 4GB RAM and a stable internet connection. Since these requirements are provided with the servers Vercel offers, we will make VErcel host and maintain our backend.

3.4 Persistent data management

ExDev will be using two databases: MongoDb and Neo4J.

MongoDb
In ExDev, information about users needs to be stored (e.g. usernames, passwords, repositories, information on which user is developer at which repository and who are the repository owners at a repository, etc.). For storing such data, we preferred using a relational database. Since MongoDb was a component of the MERN (MongoDb, Express, React, Node.js) stack that we used for the development of web management tools, we decided to use MongoDb as our relational database.

Our entity classes will be converted to a database scheme with a package called mongoose. CRUD operations will again be maintained with this package. CRUD operations will be maintained by the database maintainer subsystem in the data management layer. The database will be stored on the web (in the servers of MongoDb). Therefore, this database will not be stored in our servers or in the local machines of the users. This way, the accessed data will be the same for every user which will ensure the consistency of data.

Backing up the database will be left to MongoDb. For the beta version of ExDev, the free plan of MongoDb will be used, which will be taking daily snapshots of the database for backing up. The security of the database again will be in the responsibility of MongoDb, which offers end-to-end encryption even for its free plan.

Neo4J
Neo4j is a graph database. Neo4j will be used to keep our Artifact Traceability Graph (ATG). We will place all data about the project and relationships about data points in this Graph Database. The nature of the data can be intrinsically translated to a graph. We are using a library for connecting to a Neo4j instance. The Neo4j instance will reside in the cloud, specifically in the Neo4j AuraDB. On-premise installation of the DB is possible, but in the first step we will use the SaaS version of the Neo4j. Which, the free version should be enough for the Senior Design Project. We can open one database in the free version. We can probably merge a couple of projects into one database and we estimate that it can hold 1-2 large projects (around 25.000 nodes and 100.000 relations) or 10-15 mid sized projects (7.000 nodes) or 100-150 (around 1000 nodes) small projects.

Neo4j node creation and queries will be handled by the recommendation engine subsystem. To examine the systems we can consider an example like this. The user clicks “Find expert on this file” in VS Code. The request goes to the server and includes the file name. Server deconstructs the request and sends the information to the recommendation engine. Recommendation engine constructs the Cypher quarry. Cypher quarry is the quarry language for the Neo4j graph database. Then Neo4j returns the relevant nodes and relations to the recommendation engine. Then the engine handles the data and runs the algorithm for recommendation. Then returns the result to server and server sends to the VS Code frontend.

The MongoDB schemas are not very interesting but Neo4j is. Here is a proposed node and relations list.

Required Nodes	
File	
Commit	
Author	(Developer)
Change Set	(Pull Request) (PR)
Code Review	
Issue	(Requirement) (Bug)
Method	
Folder	
Optional Nodes	
Requirement	Can be included in issue
Bug	Can be included in issue
Test Case	Can be included in Method or File
Tester Designer Reviewer	Can be included in Author
Build	
Release	
Relations	
(Author) commited (Commit)	
(Author) added,deleted,modified, renamed (file)	

(Issue) opened, closed, participated	
(Folder) contains (Folder/File)	(File) contains (Method)
Other*	

Table 1: Neo4j Data Schema Draft

*Other: At the current situation we cannot determine all the relations. Our program's performance and accuracy depends on which relations we will include. Which (almost certainly) may change with tests and optimisations.

3.5 Access control and security

Recall that to use ExDev, first of all a user has to *initialize the repository* [1] who will become the *repository owner* [2]. Then, other users can *join the repository* [3] who will become the *developers* [4] of that repository. As it is seen, at this point there are two different user types and these two types have different roles, different access rights. Therefore in ExDev, we need to make special access control. This will be achieved by structuring the database accordingly. That is, in each repo, each repository owner's user id will be stored (recall that there can be more than one repository owner if the first repository owner promotes some developers). This way, when a request is sent to the server from the web management tool frontend, the id of the sender will be checked to detect if they are a repository owner of that repository or not. If yes, then they will be allowed to perform the operation. If not, then they will not be able to perform the requested operation and will be returned an error message with the status code of 403. With this strategy, we will manually ensure the access control. See **Table 1** for a detailed description of which user type can perform which action.

Table 2: Access Control Matrix

	Developer	**Repository Owner**
Promote /Demote Developers	✗	✓
View join requests	✓	✓
Accept/ Reject join requests	✗	✓
Ask for help in that Repository	✓	✓

[1] See the glossary for a detailed description
[2] See the glossary for a detailed description
[3] See the glossary for a detailed description
[4] See the glossary for a detailed description

Be recommended as an Expert for that repository	✓	✓
Configure Repository Settings	✗	✓
Change Repository Shared Pass	✗	✓
Leave the Repository	✓	✓
Delete Repository	✗	✓

In ExDev, security is an important issue that we take into consideration. This is because, in our application, we will be processing personal data of users and also we will be mining data from GitHub repositories even if it is a private one. Therefore, for being able to make any operation, every user needs to be authenticated. For ensuring that users are authenticated, we are using Javascript Web Tokens (JWT). That is, every user needs to register/ login first from our web management tool or from our Visual Studio Extension. Then, if the provided credentials are correct, our server returns them a JWT that will expire in one hour. After the token expires, users need to login again. After the users receive this token from our server, this token is saved to the local storage in a web browser if the user is currently using our web management tool; or it is saved to a global state in Visual Studio if the user is currently using our extension. From now on onwards, for every request that is being sent to our server will need a valid JWT. Otherwise, our server will respond with an error with a status code of 403. This way, we are manually ensuring only the authorized users can make operations and no unauthorized third parties can make operations.

The security of the data in MongoDb is ensured by MongoDb itself. Only additional security measure we take is that we are not saving the passwords of users as they are. We are first encrypting them, and then we store the encrypted passwords in our database. The other security measures that MongoDb take are as follows:

Encryption: Mongodb provides a default end-to-end encryption at rest. It means that data is encrypted before being written into the disk.

Auditing: Mongodb supports auditing functionality that allows logging the actions performed on the database. Logs can be used to track user activity and detect suspicious actions.

Network Security: Mongodb supports multiple security qualities that are ACLs (access control lists), VPC peering and IP whitelisting. These qualities protect the database against unauthorized access.

Neo4j: We do not need extensive security measures for Neo4j, because the End User will never interact directly with the Graph DB.

Last of all, ExDev requests Personal Access Tokens (PAT) from its users for being able to mine private GitHub repositories. On the other hand, when given the necessary access rights, these PATs can be very powerful and can be dangerous if they are exposed. Therefore, ExDev requests Read Only tokens from its user, ensuring that nothing in the repository can be changed or deleted with those tokens. They will be used only for mining the history of the repository. This is another informed judgment that we made for ensuring the security of our users' repositories.

4. Subsystem services

4.1 Presentation Layer

The presentation layer is the layer that interacts with user requests directly. The web management tool and the VS Code extension are the subsystems that take place in the presentation layer. Web Management tool implies the web application side of the Expert Developer where the user can manage configuration and registration operations. VS Code extension is the extension we are building. Web management tool and VsCode extension send requests and required queries to the server and return the response from the server to the end users. The main functionalities of the web management tool are; registration to the system, initialization and registration to a repository, and managing configuration settings for repositories. The main functionalities of the VsCode extension are operating the core functionality of the system and finding the expert developer for specified code parts.

4.1.1 Web Management Tool

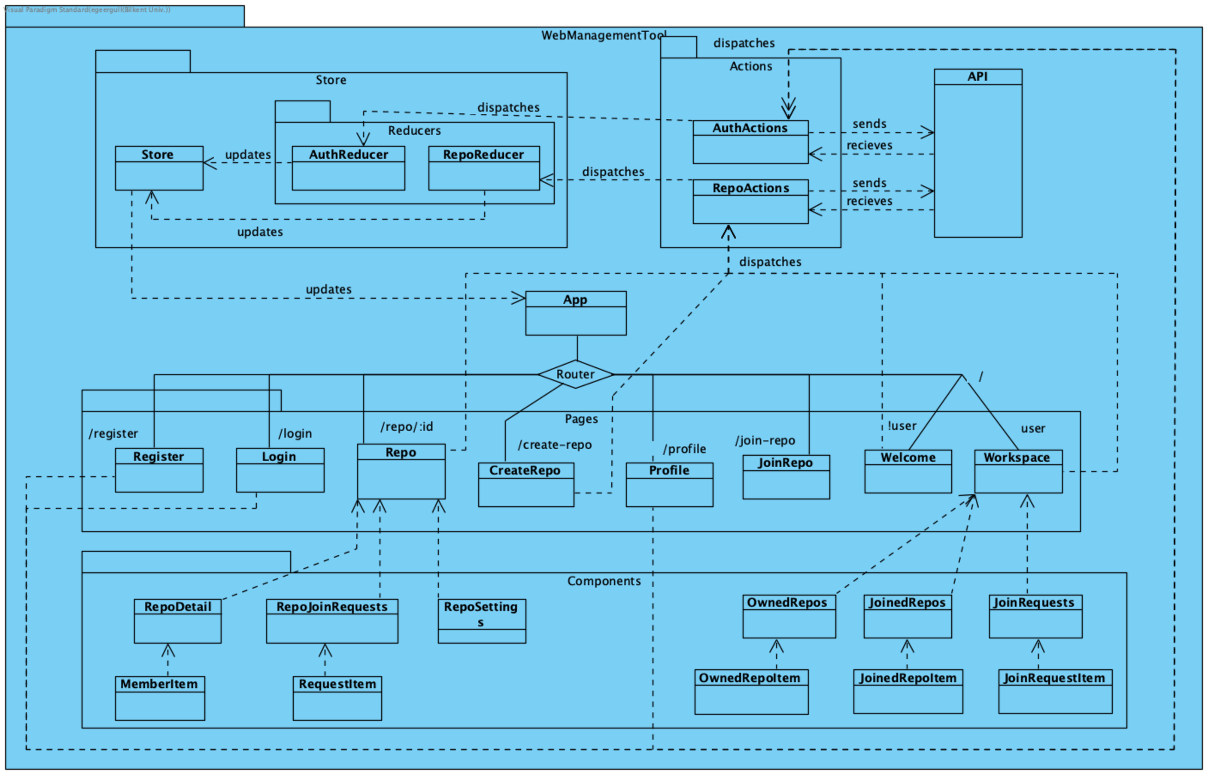

Figure 3. Web Management Tool Subsystem Details diagram

Web management tool is the web front-end side of ExDev where our users can register to our system, create/ join repositories and manage their repositories' settings. The web management tool is a react application that uses Redux for keeping track of the global states. There are 4 main folders and 2 main files in this sub-system (i.e. Actions, Reducers, Pages, Components, App, and API)

Actions and Reducers
These two files together create the Redux logic. The sub-system needs redux logic to manage the global states. Since react by its nature has a tree-like structure (i.e. from a single root react-component, many children components are born), maintaining global variables is hard. Without using a global-state manager like Redux, one would have to pass many parameters to each child to maintain the communication between different child components. To avoid this complexity, we decided to use Redux, so that the global states can be directly accessed by any child components. The actions folder has two files: AuthActions and RepoActions. In AuthActions, there are actions that are related to authentication and that will modify the user global variable. For example, sign in, signup, etc. In the RepoActions, there are actions that will modify the repositories of the user. For example, create repository, join repository, etc. These actions are dispatched by various pages or components. Then these actions dispatch various reduces. The reducer files are placed in the Reducers folder. The AuthReducer file is responsible for updating the user global variable stored in the store. Similarly RepoReducer is responsible for modifying the repositories of the user stored in the store.

API
The api file is responsible for the communication between the web management tool and the server. As actions require communication with the server, the actions send a request to the server via this api, and receives the response again via this api file. An example flow is as follows: a user tries to login to their accounts. The login function that is placed in the AuthActions file is triggered. This function sends a request to the server by providing the credentials entered from the user. The api sends this requests and waits for the response. Then the api returns the response to this function, and according to the response, UI is updated.

App
It has been mentioned that react by its nature has a tree like structure. The root component of this "tree" is the App file. This app file has a router inside. The router conditionally renders the correct cold component according to the route (i.e. the URL). The only extra conditional rendering is seen in the default "/" route. If the user has logged in, this route "/" will show the workspace of that user, else it will show the welcome screen.

Pages and Components
It has been mentioned that react by its nature has a tree like structure. In these folders, the children components of the tree-like structure take place. For reusability and maintainability of the code, such files are separated to two folders: Pages and Components. In the pages folder, the main pages of the web management tool take place. Each page corresponds to a route. For example, when the route is "/login", the login page will be rendered. However, in the components folder, smaller components of the pages take place. For example, the

Workspace page has 3 components: OwnedRepos, JoinedRepos and JoinRequests. These are the parts of that page which need extra logic. Therefore, to keep the code in the Workspace page simpler, they are separated into different files.

4.1.2. Visual Studio Code Extension

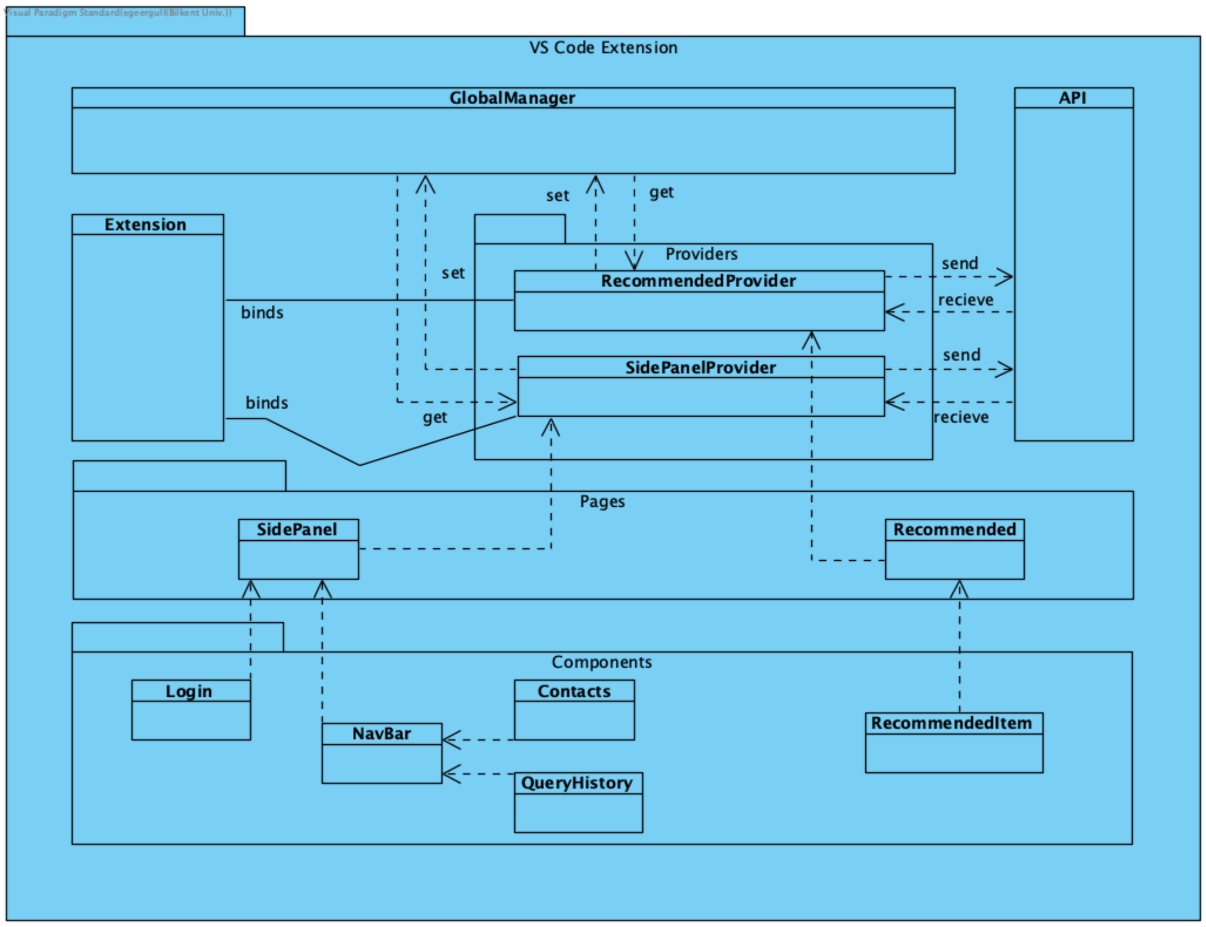

Figure 4. VS Code extension Subsystem Details diagram

The VS Code extension subsystem is where the code related to our extension takes place. This subsystem has 3 main folders (Providers, Pages and Components) and 3 main files (Extension, API, GlobalManager).

Extension
The extension file can be considered as the index.html file of a website. That is, once the extension is runned, the firstly read file is the extension where all the required bindings take place. In this file, we bind the RecommendedProvider and the SidePanelProvider to our extension.

GlobalManager
GlobalManager is a file that consists of a class. This class has properties like, user token, repository. These properties are saved into the extension and have set /get operations on them. By using this file, providers can save necessary data to the extension and get them when needed.

Providers
The providers folder has two files: RecommendedProvider and SidePanelProvider.

SidePanelProvider is the file that determines the content of the side panel (the collapsible part of VS Code on the left-side of the screen) on VS Code. In this side panel, we register a webview. This way, we will be able to customly design the side panel, without being restricted with the VS Code API. This file acts as the bridge between the webview and the extension file. Also, it is seen that this file interacts with the API file. This part is trivial. As our extension needs to be connected with our server (for operations like login, get recommendation, etc.), the API is needed. Last of all, sidepanelProvider is setting and getting from the GlobalManager file. This way, this provider can access/ modify global variables like user, etc.

RecommendedProvider is the file that determines the content of the tab that occurs when a query is runned (e.g. when a user asks for the experts of a file, etc.). Once a recommendation is made, this tab is created where the user is displayed a list of experts about their query. In this tab, we register a webview. This way, we will be able to customly design the side panel, without being restricted with the VS Code API. This file acts as the bridge between the recommendeds webview and the extension file. Also, it is seen that this file interacts with the API file. This part is trivial. As our extension needs to be connected with our server (for operations like adding the expert to the contacts, etc.), the API is needed. Last of all, recommendedProvider is setting and getting from the GlobalManager file. This way, this provider can access/ modify global variables like user, etc.

API
The api file is connecting to the server. It is sending requests and waiting for the response. Then, this file passes the responses to the related files. An example flow would be as follows: the user right-clicks on a file to get the recommended experts on that file. Then, the recommenders tab is created. The recommendedPanel immediately sends a request to the server with the information of the query. Then the api waits for the server's response. Once it receives the response, the api returns this response to the recommendedPanel. Finally, recommendedPanel updates its UI accordingly.

Pages and Components
It has been mentioned that webviews are used for the recommended tab and the side panel. In these webviews, a react-like environment is created with svelte files. That is, there is a root component for each provider and each root file has children components that are rendered conditionally. The root files are placed in the pages folder whereas the children components take place in the components folder. For example, the SidePanel page conditionally renders the Logging component if there is no user in the global state; else, it renders a navigation bar. According to the selection of tha navigation bar, the UI displays either the contacts or the query history.

4.2 Application Layer

The application layer is the decision mechanism of the system. As a middle layer in our architecture, it is the junction point of our system. By using this layering, utilizing modularity and better organized project structure is happening. The application layer also acts as an additional shelter to the data tier which had to be kept private. It has two components; Recommendation engine and Server.

4.2.1 Server

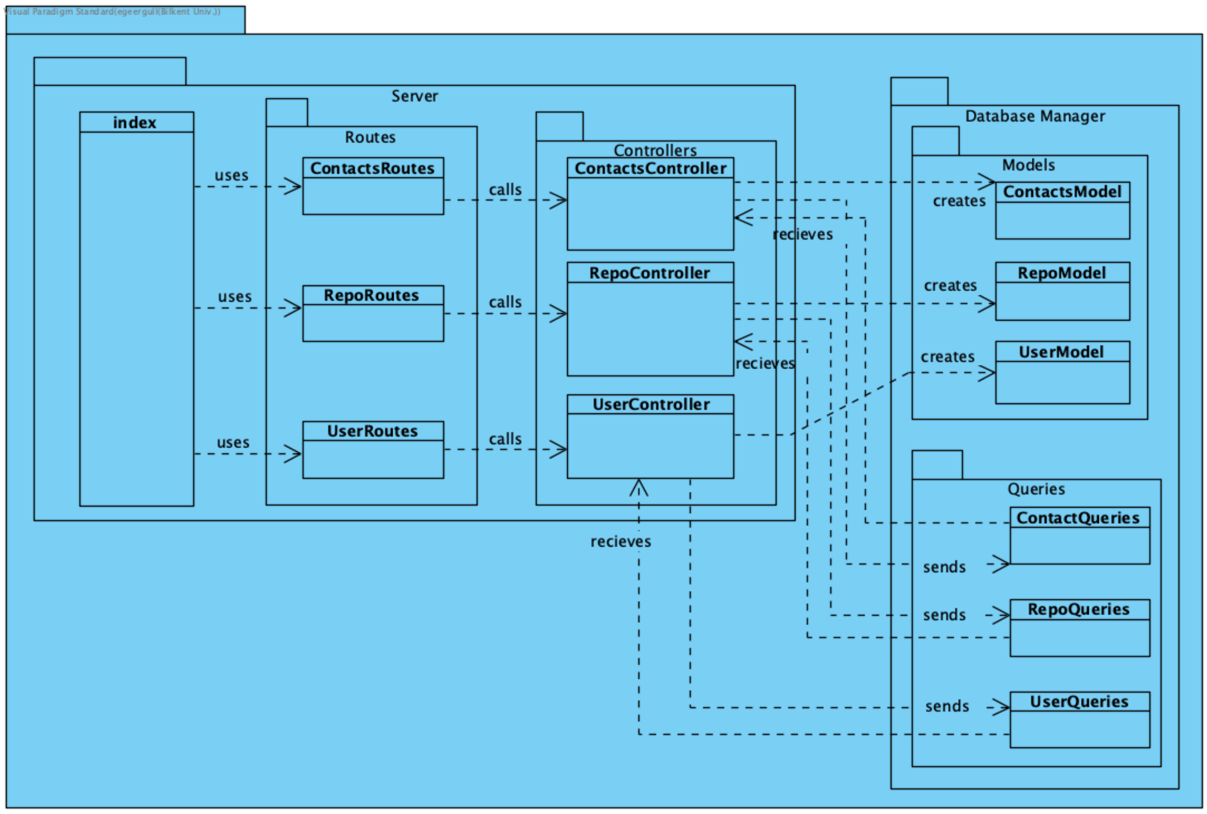

Figure 5. Server and Database Manager Subsystems detailed diagram together

The server is a subsystem of the application layer, an inevitable decision-maker that handles every request. The server is the main processor that takes place in the middle of every action as a crucial gateway. It connects the data management subsystem to the presentation layer. The presentation layer transmits the requests to the application to be handled.

The server of the system reaches the data tier or recommendation engine considering the reached request from the presentation tier. In our program, the initialization of the graph database and first processed recommendations are heavy operations to execute in real-time. Thus, this first initialization process does not appear in our layered architecture. Other than that, every request's flow is similar. The presentation tier and server then required subsystems of our architecture.

The server consists of two main folders (Routes and Controllers) and one main file (index).

Index
The index file is the file that is firstly runned by the project. The index file starts to listen to a port and binds the routers to itself.

Routes
Routes folder consists of 3 files: ContactsRoutes, RepoRoutes and UserRoutes. Each of these files binds the routes with the relevant functions. For example, UserRoutes file binds the route of {PORT}/user/signing to the function of signin implemented in the UserController file. This way, when the frontend sends a request to this route, the UserRoutes file will catch it and then execute the related function placed in the controller.

Controllers
Controller folders have 3 files: ContactsController, UserController, RepoController. The controller files have the implementation of specific functions. The logic that is needed for that function is placed in this file. Also, it is the controllers who communicate with the Data Manager Subsystem and the Recommendation Engine. One example scenario is as follows: once the router catches the request of a register, the register function of UserController is called. In this function, the controller firstly validates the sent inputs (checks if the email address is a valid email address, checks if the password is strong enough). If the validation fails, the controller sends back a response immediately. Else if the validation is successful, the controller first checks if a user with the same email exists by sending a query to the data manager. If the data manager returns “no such user exists in the database”, then the user can be registered to the system. Once again, the controller asks the data manager to create the user and save it to the MongoDb. As it can be understood here, the controller is responsible for operating the logic and manipulating the database via the data manager subsystem.

4.2.2 Recommendation Engine

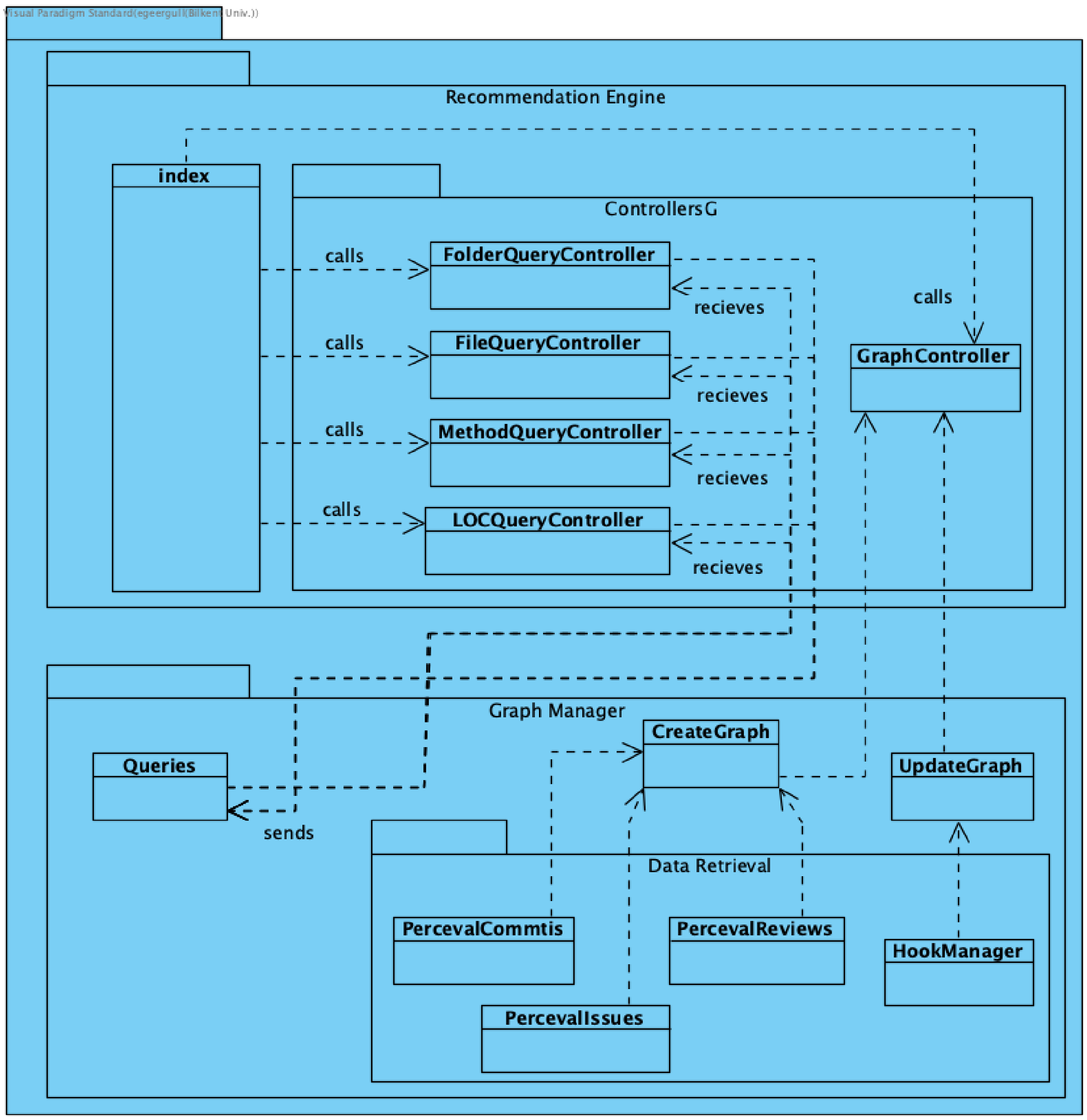

Figure 6. Recommendation Engine and Graph Manager Subsystems detailed diagram together

Recommendation Engine is the subsystem where all the logic of our recommendations lies. This subsystem is communicated via the server subsystem. That is, after getting the query request from VS Code extension, the server communicates with the recommendation engine, which first runs some queries on the artifact traceability graph to collect some data by communicating with the graph manager subsystem. Then, once it has the necessary data, the engine evaluates this data and in return, returns some experts to the server. The recommendation engine has 1 main file (index) and 1 main folder (controllers).

Index

Like the index file of the server, this index file also listens to a specific port. Since there aren't as many routes as the server, there is no separation of the routes file. That is, the requests coming to the specified routes are catched by the index file and the relevant controller functions are called.

Controllers
The Controller folder has 5 files: FolderQueryController, FileQueryController, MethodQueryController, LOCQueryController (that is Lines Of Codes Query Controller), and Graph Controller. The first 4 files are responsible for dealing with specific types of queries. For example, FileQueryController is responsible for returning experts for a given file. This controller can ask for some queries to the Graph Manager. One example query would be “give me the number of connections between file A and all of the users”. After getting every necessary data, this controller decides the most knowledgeable experts and returns them. GraphController is the controller that is responsible for creating and updating artifact traceability graphs. For example, when a repository is created, the server communicates with the recommendation engine. The recommendation engine calls the function “create graph” located in this controller. The functions in these controllers call the graph manager subsystem to actually manage the Neo4j graph database.

4.3 Data Management Layer

The data management layer is the third and last layer of our program. It manages the connection between the application layer and the databases (MongoDb, Neo4j). It transmits queries that are provided by the application layer to return proper information from the store. To make it more protected and less visited, for efficiency and privacy considerations, it is hidden under two layers; Presentation and Application. The data management layer is composed of a database manager and a graph manager. Graph manager deals with the graph database Neo4J while database manager deals with MongoDB.

4.3.1 Database Manager

The database manager is there to control and communicate with the general NoSQL service which is MongoDB for our system. It is a bridge to reach objects stored in MongoDB. When a user's request comes to this layer and specifically the database manager part of the data management layer, it takes the request that is provided through layers and transforms that into the proper form to reach objects in the database service. Simple requirements of our system that can be generalized for many other software projects are handled by the database manager component, unlike our unique functionalities.

The Database Manager subsystem consists of two main folders: Models, Queries. (See Figure 5)

Models
The models folder has 3 files in it: ContactsModel, RepoModel, UserModel. Each model file shows the schema of database entities.

Queries

The queries folder has 3 files in it: ContactsQueries, RepoQueries, UserQueries. Each query file has specific queries to be runned on different tables of the MongoDb. For example, one query example of the RepoQueries file would be, “give me a list of repositories which this user id is a member of ".

4.3.2 Graph Manager

Graph manager is a component of the data layer to connect the graph database (namely Neo4j) to our system. Graph database is required to create the artifact traceability graph which is the main guidance to find expert developers for specific code segments. The recommendation engine runs a query that is transmitted from the server and makes an evaluation of the graph database via the graph manager. Every artifact traceability graph is distinguished by its unique repository id. The graph manager is responsible for reaching the objects of the artifact traceability graph. Due to the functionality distinction, it is a separate component from the database manager. In addition to that, the graph manager is also responsible for the heavy workload of the system, thus time constriction is not as strict as the database manager.

Creating the Graph
We will use the Perceval [4] and the GitHub API to collect information about the project. The Perceval can give us an workable, formatted version of the data which includes Git information, Author information, Issue information and Pull Request information. Also, all the information is collected by simply running the program. Then we will use that data to fill our graph. GitHub API is a second option that gives us some dirty and verbose information that needs to be worked on before turning into nodes and relations. It has an advantage of selective and detailed collection, but the graph manager needs to discover, check and parse all the GitHub links one by one. And, some additional development time needed for that system. We align to use Perceval instead of GitHub API because of this.

Updating the Graph
After the graph is initialized, we need to update the graph when new data is created. Graph Manager needs to understand if some new data is available, collect only the new data and update the graph with the new data. Our plan for understanding when the new data is created is using GitHub’s webhooks [5]. A Webhook can give us an alert when something is updated. Then we can fetch this data. We can also use a periodic check for updates. But, using webhooks can help us to update graphs as quickly as possible.

Working with Methods
As one of our goals, we want to keep the method metadata (not the method itself but who changed it, signatures, which commit changed it etc.) in our graph. Extracting this data is no straightforward task. Neither Perceval nor GitHub gives us that information. So, we need to construct an Abstract Syntax Tree to resolve this metadata. We plan to parse the source code with a library called “abstract-syntax-tree” [6]. This will (theoretically) give us the information we need. Then, we can create Method nodes to connect to the File nodes. We will keep which lines are changed in a particular commit in the commit data. Then, match the changed lines with the method metadata. Finally, we will link them.

For the relation of graph manager and the recommendation engine, see the figure 6. As it can be seen, the graph manager consists of several files.

Create Graph, PercevalCommits, PercevalReviews, PercevalIssues
The CreateGraph file is responsible for creating a graph. When the recommendation engine provides a URL to this file, a process is started which creates the graph. For creating a graph, the historical data of a repository is needed, which can be obtained by using the GitHub REST API. In the market, there is a good tool named Perceval which automates the data retrieval part. Yet, Perceval works with Python. However, our project is written with Node.js. Thus, the create graph file spawns python child processes. Each python process mines different artifacts from the GitHub REST API (commits, issues and reviews). Once all the data is mined, the create graph file creates the graph on Neo4.

UpdateGraph, HookManager
UpdateGraph file is responsible for keeping the artifact traceability graph up to date. This can be achieved by using hooks on GitHub. For example, every time a commit is pushed to the repository, our hook will trigger the updateGrahp file. This way, the graph will always be up to date.

Queries
This file will contain the queries that will be runned on Neo4j. One example query would be, “give me the number of connections between user A and folder B”. Every such query will be held in this file.

5. Test Cases (functional and non functional test cases, procedures and expected results)

Functional test cases test ‘what’ the product does. They check the operations and actions of an application. Non-Functional test cases test the non-functional requirements of our system. They test various topics like performance, usability, etc. The following template will be used to log our test scenarios in order to track and debug potential errors.

Test Scenario ID:
Satisfied Requirement:
Prerequisite:
Postrequisite:
Type: (Whether the tested action should actively complete a task or it should fail and get the required error messages)
Date Tested:
Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result

It should be noted that we grouped tests that are closely related to each other in a single table. Each row of such tables are distinct tests. That is, Tests 1.1 and 1.2 are different tests.

The total number of total functional tests are 32 and the total number of total non-functional tests are 18. In total there are 50 tests.

5.1. Functional Test Cases

1. Test Scenario ID: Login-User-1
   Prerequisite: Valid username and password
   Postrequisite: -
   Type: Positive
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
1.1	Verify if a user will be able to login with a valid username (e-mail address) and valid password.	a. User enters a valid e-mail address and password b. User clicks submit button	a. User input is visible from the GUI. Password field is visible as bullet signs. b. User is redirected to the Home Page.		
1.2	Verify if the password visibility changes when the eye icon is clicked	a. User enters their password. b. User clicks to the open-eye icon c. User clicks to the close-eye icon	a.Password field is visible as bullet signs. b. Password becomes visible as text c. Password becomes non-visible again (becomes bullet signs)		

2. Test Scenario ID: Login-User-2
   Prerequisite: -
   Postrequisite: -
   Type: Negative
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
2.1	Verify if a user cannot login with a valid e-mail	a. User enters an invalid e-mail address or password.	a. User input is visible from the GUI. Password		

	address and an invalid password.	b. User clicks submit.	field is visible as bullet signs. b. An error message is displayed.		
2.2	Display warning if any of the fields are blank and the submit button is clicked.	a. User leaves either the e-mail address or password field blank. b. User clicks submit.	a. User input is visible from the GUI. Password field is visible as bullet signs. b. A warning message is displayed.		
2.3	Display warning if network error occurs.	a. User fills out all the fields. b. User clicks submit. c. There is a network error or a problem connecting to the server.	c. A network error message is displayed to the user.		

3. Test Scenario ID: Register-User-1
   Prerequisite: -
   Postrequisite: -
   Type: Positive
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
3.1	Verify that a user cannot register without confirming their password.	a. User enters all necessary information and a valid e-mail address. b. User enters a password and confirms it by typing the same password. c. User clicks submit.	c. A warning message is displayed if the passwords don’t match or the user doesn’t confirm their password.		
3.2	Verify that a user cannot register without providing an actual email address.	a. User enters a string that doesn’t match the email format. b. User clicks submit.	b. A warning message is displayed ‘Enter a a valid email address’.		

4. Test Scenario ID: Logout-User-1
   Prerequisite: -
   Postrequisite: The user should be logged in.

Type: Positive
Date Tested:
Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
4.1	Verify that a user can log out successfully.	a. Logged in user clicks on ‘Log out’ from the Web interface.	a. The user is logged out of their account and cannot access their account without logging in again.		

5. Test Scenario ID: Create-Repository-User-1
   Prerequisite: -
   Postrequisite: -
   Type: Positive
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
5.1	Verify that the user can create a **private** repository as their own by providing a valid repository ur	a. Once the user is logged in, they navigate to the “Create” tab b. They enter a valid repository url and a shared pass c. User clicks “Create repository”.	c. The claimed repository is displayed under the Home Page - Owned Repositories.		
5.2	Verify that a user can create a public repository as their own as long as they are the owner of the repository.	a. Once the user is logged in, they navigate to the “Create” tab b. They enter a repository url and a shared pass and clicks “Create repository”. c. A GitHub repository check is performed to verify if the user is the owner of the claimed repository. d. The user is verified as the owner of the repository.	d. The claimed repository is displayed under the Home Page - Owned Repositories.		
5.3	Verify that the user can see if the “Create Repository” task is under progress, failed,	a. Authenticated user creates a repository by filling the necessary information from the	a. The repository creation task is resumed via the Graph Manager.		

	or complete.	"Create" tab of the web application. b. The user views the status of the task from the Home Page - Owned Repositories.	b. If the task is under progress or failed, it is indicated in the web interface. If it is complete, the "Open with VSCode" icon should be available to the user.		

6. Test Scenario ID: Create-Repository-User-2
   Prerequisite: -
   Postrequisite: -
   Type: Negative
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
6.1	Verify that a user cannot create a repository that is invalid / that does not exist on GitHub.	a. Once the user is logged in, they navigate to the "Create" tab b. User enters a repository url and a shared pass along with their PAT and clicks "Create repository". c. The repository url is invalid.	c. An error message is displayed "This is not a valid repository url — Try again**".**		

7. Test Scenario ID: Join-Repository-User-1
   Prerequisite: A valid user
   Postrequisite: -
   Type: Positive
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
7.1	Verify that a user can Join a repository by providing the valid repository owner, repository name, repository url along with the accurate shared pass (provided by	a. Once the user is logged in, they navigate to the "Join" tab. b. User fills in necessary information, specifically the accurate shared pass.	b. The joined repository is displayed under the Home Page - Joined Repositories.		

	repository owner).				
7.2	Verify that a joining user's request is delivered to the owner user's repository.	a. Once the user is logged in, they navigate to the "Join" tab. b. User fills in the necessary information. c. The user unchecks the shared pass box and sends a request to join the repository.	c. If the repository already exists in the ExDev system, the request is forwarded to the repository owner to confirm. The repository owner may accept or decline the request.		
7.3	Verify that a user can Join a repository if their request to join the repository is accepted by the repository owner.	a. Once the user is logged in, they navigate to the "Join" tab. b. User fills in the necessary information. c. The user unchecks the shared pass box and sends a request to join the repository. d. If the repository already exists in the ExDev system, the request is forwarded to the repository owner to confirm. e. The repository owner accepts the request.	d. The given repository should be valid and present in the ExDev system. User's Join Request is shown under Home Page as pending. e. The joined repository is displayed under the Home Page - Joined Repositories.		

8. Test Scenario ID: Join-Repository-User-2
   Prerequisite: -
   Postrequisite: -
   Type: Negative
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
8.1	Verify that a user cannot Join a repository without providing the accurate shared pass (provided by the repository owner).	a. Once the user is logged in, they navigate to the "Join" tab. b. User enters an invalid shared pass.	b. An error message is displayed and the user is not joined to the repository.		
8.2	Verify that a user cannot Join a repository	a. Once the user is logged in, they navigate to the	e. User's Join Request is shown		

	if their request to join the repository is declined by the repository owner.	“Join” tab. b. User fills in the necessary information. c. The user unchecks the shared pass box and sends a request to join the repository. d. If the repository already exists in the ExDev system, the request is forwarded to the repository owner to confirm. e. The repository owner declines the request.	under Home Page as declined and user cannot view the repository.		
8.3	Verify that a user cannot Join a repository if it has not been created before	a. Once the user is logged in, they navigate to the “Join” tab. b. User provides a valid GitHub repository URL which has not been created in our system	b. User is shown an error message saying “This repository has not been created in our system!”		

9. Test Scenario ID: VSCode-Contacts-1
   Prerequisite: -
   Postrequisite: -
   Type: Positive
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
9.1	Verify that once user clicks add contact, changes are reflected to the database and user interface.	a. Users view the recommended experts tab where they see a list of recommended experts for a query. b. User clicks the 'Add Contact’ icon for a selected user.	b. The ‘contact’ relationship is added to the database and visible from the Contacts page.		
9.2	Verify that once user clicks remove contact, changes are reflected to the database and user interface.	a. User navigates to the Contacts tab from the ExDev VSCode extension. b. User clicks the 'Remove Contact’ icon for a selected user.	b. The ‘contact’ relationship is removed from the database and the Contacts page.		

10. Test Scenario ID: VSCode-Recommendation-1
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
10.1	Verify that in the user interface, a recommendation is available within the appropriate scope.	a. User right-clicks on the file explorer, on a file. b. User right-clicks on the file explorer, on a folder.	a. The right-click menu should show an option 'Recommend experts on this file'. b. The right-click menu should show an option 'Recommend experts on this folder'.		

11. Test Scenario ID: VSCode-Rating-1
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
11.1	Verify that if a user performs an expert developer rating, changes are reflected to the database.	a. User performs a rating for a selected expert developer.	a. The rating results should be updated or added to the database. Expert developer's score should be re-calculated accordingly.		

12. Test Scenario ID: VSCode-Select-Query
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
12.1	Verify that if a user right clicks on a file, they see an option to get recommendation on that file	a. User opens the file explorer and right clicks on a file	a. The pop-up menu includes an option like “Recommend me experts on this file”		
12.2	Verify that if a user right clicks on a file, they see an option to get recommendation on that folder	a. User opens the file explorer and right clicks on a folder	a. The pop-up menu includes an option like “Recommend me experts on this folder”		
12.3	Verify that if a user sees an option to get recommendation on that method, next to the signature of that method	a. User opens the file explorer and views a file b. User scrolls down until they see a method	b. The user sees on option like “Recommend me experts on this method” next to the signature of that method		

13. Test Scenario ID: VSCode-Get-Recommendations
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
13.1	Verify that if a user asks to get help on a file, they are recommended some experts	a. User opens the file explorer and right clicks on a file b. User clicks to “Recommend me experts on this file” option c. The test runner goes to the example database and looks to the tested repository for the number of recommended experts per query (let’s call this N)	b. A pop-up menu opens and N experts are displayed		
13.2	Verify that if a user asks to get help on a folder, they are recommended	a. User opens the file explorer and right clicks on a folder	b. A pop-up menu opens and N experts are		

	some experts	b. User clicks to "Recommend me experts on this folder" option c. The test runner goes to the example database and looks to the tested repository for the number of recommended experts per query (let's call this N)	displayed		
13.3	Verify that if a user asks to get help on a method, they are recommended some experts	a. User opens the file explorer and right clicks on a file b. User clicks to "Recommend me experts on this method" option c. The test runner goes to the example database and looks to the tested repository for the number of recommended experts per query (let's call this N)	b. A pop-up menu opens and N experts are displayed		

14. Test Scenario ID: Managing-Repo-Settings
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actua l O/P	Test Result
14.1	Verify that if a user in the role of a developer wants to access to the repository settings, they cannot	a. User logs in and then selects a repository from "Joined Repositories" b. User clicks to the settings under that repository	a. User is redirected to a page where they see the GUI related to that repository b. User is shown an error message saying "Only repository owners can view /modify repository settings"		
14.2	Verify that if a user in the role of a repository owner wants to access to the repository settings, they can	a. User logs in and then selects a repository from "Owned Repositories" b. User clicks to the settings under that	a. User is redirected to a page where they see the GUI related to that repository		

		repository	b. User is shown the repository settings		

15. Test Scenario ID: Delete-Repository
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
15.1	Verify that if a repository is deleted, members of that repository will no longer see that repository in their workspaces	a. Repository Owner logs in and views a repository b. Repository Owner deletes the repository c. Repository Owner returns to their workspace	c. Repository Owner does not see that repo under their owned repositories		

16. Test Scenario ID: Delete-User
    Prerequisite: -
    Postrequisite: -
    Type: Positive
    Date Tested:
    Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
16.1	Verify that a user cannot delete their account if they are a repository owner of any repository	a. A user who owns a repository logs in to their account b. User goes to the profile section and clicks to “Delete my account” button	b. User is shown an error saying “Please yield your repository owner status to some other developer for the following repositories: [A LIST OF OWNED REPOSITORIES]		

5.2. Non-Functional Test Cases

1. Test Scenario ID: Web-Performance-1

Satisfied Requirement: Performance
Prerequisite: -
Postrequisite: -
Date Tested:
Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
1.1	Verify that page load time is below 3s.	a. Users may navigate to any page including Login, Home Page, Create Page etc.	a. User’s waiting time should not exceed 3 seconds.		
1.2	Verify that the user can access their project from VSCode after creating the repository within 2 hours.	a. Authenticated user creates a repository by filling the necessary information from the “Create” tab of the web application. b. The user clicks on the “Open with VSCode” icon from the web interface.	a. The Graph Manager initializes the github repository as a Neo4j Artifact Graph. This process should not take more than 45 minutes for a mid-sized project. b. Once the graph initialization is complete, it should be visible from the web interface to the user with a latency below 1 hour.		

2. Test Scenario ID: Web-Usability-1
   Satisfied Requirement: Usability
   Prerequisite: -
   Postrequisite: -
   Type:
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
2.1	Verify that the user can access the register page from the login page, and the login page from the register page.	a. The user visits the login page. b. The user visits the register page.	a. A button or link is available to navigate to the register page. b. A button or link is available to		

			navigate to the login page.		

3. Test Scenario ID: Graph-Manager-Performance-1
   Satisfied Requirement: Performance
   Prerequisite: -
   Postrequisite: -
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actu al O/P	Test Result
3.1	Verify that graph initialization is queued within 1 hour from the create request.	a. Authenticated user creates a repository by filling the necessary information from the “Create” tab of the web application.	a. The new graph initialization task is queued within an hour.		
3.2	Verify that the graph is initialized under 45 minutes for a mid-sized project.	a. Authenticated user creates a repository by filling the necessary information from the “Create” tab of the web application.	a. The Graph Manager initializes the github repository as a Neo4j Artifact Graph. This process should not take more than 45 minutes for a mid-sized project.		
3.3	Verify that, in case new changes occur in the repository, a graph update task is queued within 5 hours from the change.	a. A new changeset (new issue, pull request, commit etc.) occurs in the GitHub repository that is present in the ExDev system.	a. The changeset is detected within 5 hours and a new graph update task is queued.		
3.4	Verify that graph update is performed under 45 minutes.	a. A new changeset (new issue, pull request, commit etc.) occurs in the GitHub repository that is present in the ExDev system. The changeset is detected within 5 hours and a new graph update task is queued.	a. The Graph Manager updates Neo4j Artifact Graph for the given repository. This process should not take more than 45 minutes for a mid-sized project.		

4. Test Scenario ID: Graph-Manager-Security-2
   Satisfied Requirement: Security
   Prerequisite: -

Postrequisite: -
Date Tested:
Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
4.1	Verify that graph transactions are atomic. They can end in one of the two i.e. Success or Failure.	a. A query is executed via the Graph Manager.	a. If the query is executed successfully, the Graph Manager should communicate the result to the Recommendation Engine. If not, The end-user is shown an error message.		
4.2	Verify that multiple transactions run without impacting each other and won’t hinder the database state.	a. Multiple users may try using the VSCode extension. This may require multiple concurrent graph transactions to occur.	a. The database state stays valid after a series of heavy transactions.		
4.3	Verify that committing changes to the graph are saved without any loss due to a power loss or crash.	a. The artifact traceability graph is updated (a new update/delete/create) occurs. b. During the update, there is a power loss, corrupted connection or Neo4j crash.	b. The updates are saved to the graph as soon as the problem gets eliminated.		
4.4	Verify that after a repository is deleted, the graph of it is also deleted	a. User deletes repository b. The tester checks Neo4J to see if the graph of that repository still exists	b. Tester cannot find such data		

5. Test Scenario ID: Recommendation-Engine-Performance-1
   Satisfied Requirement: Performance
   Prerequisite: -
   Postrequisite: -
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
5.1	Verify that the Recommendation	a. The user selects a code part and runs the VSCode	a. The Recommendation		

	Engine returns a result within 30 seconds.	extension. b. Recommended experts are displayed in the VSCode extension interface.	Engine classifies and refines the result from the Graph query and returns the recommendation result within 30 seconds. b. The recommendation is displayed by the user.		

6. Test Scenario ID: MongoDB-Security
   Satisfied Requirement: Security
   Prerequisite: -
   Postrequisite: -
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
6.1	Verify that the passwords are saved in an encrypted format	a. The testers views the MongoDb Users collection	a. Each user's password field is encrypted		
6.2	Verify that after a repository is deleted, all of its relevant information is erased from the MongoDb	a. User deletes a repository b. Tester looks at the MongoDb's repos collection to see if the repository still exists in the database c. Tester looks at the MongoDb's connections collection to see if any of the users have still any connections for that repository	b. The tester cannot find such data c. The tester cannot find such data		

7. Test Scenario ID: WebManagementTool-Security
   Satisfied Requirement: Security
   Prerequisite: -
   Postrequisite: -
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result

7.1	Verify that users are automatically logged out after 1 hour	a. The user logs in to their account b. The user waits for 1 hour c. The user tries to take any action	c. User is returned an error message saying “Your session has expired. Please login again!”		
7.2	Verify that web management tool cannot be accessed without logging in	a. Tester opens up the browser and browses to the following URLs: “http://localhost:3000/join-repo” , “http://localhost:3000/create-repo”, “http://localhost:3000/profile”	a. The tester is redirected to the welcome screen each time		
7.3	Verify that users with developer roles cannot access to pages which can only be accessed by repository admins	a. User logs in to their account b. Tester browses to the MongoDB and finds a repository id which the tested user is not a member of. Let’s name this id as VALID_REPO_ID c. Tester browses to the following URL: \`http://localhost:3000/repo/${VALID_REPO_ID}\`	c. Tester is shown an error message saying, “You are not a member of this repository”		

8. Test Scenario ID: Documentation Test
   Satisfied Requirement: Security
   Prerequisite: -
   Postrequisite: -
   Date Tested:
   Tested By:

Test Id	Description	Procedure	Expected O/P	Actual O/P	Test Result
8.1	Verify that every file in the code bases have comments	a. The tester checks every file in the following repositories: ExDevWeb, ExDevVS, RecommendationEngine	a. Tester sees comments on every file		

6. Consideration of Various Factors in Engineering Design

6.1 Public Health

Our project has no aspect related to public health. Therefore, we did not consider it during the design of our project.

6.2 Security

In our project, security is an important aspect. As we will be mining GitHub repositories for being able to make a recommendation, we need GitHub Personal Access Tokens (PATs) of that repo's members. Yet, as we will be only mining the data and not writing anything to the repositories, to ensure that nothing will be changed at the repository by our side, we are requesting for read only tokens. This way, we will not be able to write, change or delete anything at the repository. This way the security of the repository that has been shared with us is ensured.

Another important decision we made due to the security concerns is that as we will be collecting personal data from our users (e.g. email addresses, passwords, names, GitHub usernames, etc.) we need to ensure that their data will not be leaked and will not be accessible by unauthorized third parties. Therefore, we are establishing the communication of Web Management Tool - Server by using Javascript Web Tokens. This way, no third party is able to take unauthorized action.

6.3 Welfare

In our project design, we considered the welfare of our users. As a feature, we are having the experts rated by developers, and then we are considering these ratings before making future recommendations. At this point, we had to make a decision: either we were going to make these rating scores publicly available or not. We asked this question to developers via the survey we conducted last semester and with bearing in mind the response we received, we decided that we will be displaying only the good ratings. That is, only if an expert has a rating 4+ over 5, then we will display this information. This way, we are not disposing of poorly rated developers, which ensures the welfare of our users.

6.4 Global

Our project has no global aspect. Therefore, we did not consider it during the design of our project.

6.5 Cultural

Noting that our user profile is highly specific (targeting only developers), when considering culture, we can consider the developer culture. During the design process of our project, we bared in mind the developer culture. For example, one potential feature we considered was recommending experts on specific languages for language specific problems. Yet, we decided not to implement this feature as we are aware that developers tend to look at Stack Overflow for such problems. Trying to replace this habit with our feature would be like changing a well-rooted, fundamental component of developer culture. Thus, we decided not to implement this feature.

Also, again development culture was one of the key motivations of our project. Since helping each other is an essential part of development culture, we believe that our application is built on strong foundations. That is, in essence our project recommends experts to the developers who are in need of guidance and help and we assumed that experts would spend time helping the developers who are asking for their help. We justified our assumption with the survey we conducted last semester. Considering these two incidents, it can be concluded that our project considered coding culture a lot.

6.6 Social

During the design procedure of our application, we needed to make a decision on the following question: How will the developers start communicating with the experts? Our initial answer was “via email and Zoom”. However, after considering that our user profile will be developers only and we will want to establish fast synchronous communication, we added the option of communicating with Slack as well. This is because many teams are using Slack for their inter-communication. This can be proof that our project’s design process was affected by social factors.

6.7 Environmental

Our project has no aspect related to the environment. Therefore, we did not consider it during the design of our project.

6.8 Economic

In our design, we considered the economy as a factor. As starving-to-death broke students, we preferred only freely available services during the development of our project and again, we will be using freely available services when it comes to deploying our application. For example, we will deploy our web management tool with Vercel, instead of using Amazon Web Services. Also the Graph Database Neo4j that we will be using will be a free tool as well. Lastly, we are using the free version of MongoDb instead of the paid version. Therefore, it is possible to conclude that economy was a factor that we considered during our design process. However, it should be noted that even if we hadn’t considered the economy as a factor, the paid-versions of the services does not provide us any remarkable benefit. Thus, no sacrifices were made actually due to the economy.

From another perspective, the economy was one of the key motivations of our project. As we are making a project that will decrease the onboarding time, companies will make better use of newly hired employees. Thus, preferring to use our tool will cause them to save money in a way. Thus, it can be stated that the economy was highly considered during our design stage.

Factor Name	**Effect of Factor on Design (out of 10)**
Public Health	0
Safety	10
Welfare	7
Global	0
Cultural	10
Social	7
Environmental	0
Economic	8

7. Teamwork Details

7.1 Contributing and functioning effectively on the team

Each member is responsible for a different aspect of our system and each member can be said to be the leader of different subsystems of our application. Each member and their main responsibilities are listed below.

Ege Ergül
Ege is an ambitious designer. Therefore, he took the responsibility of the frontend of ExDev including web management tools and VS Code extension.

Bora Altınok
Bora is mainly responsible for the server. That is, he is responsible for managing MongoDb, and returning responses to web management tools and Vs Code.

Tuna Dalbeler
Tuna is mostly responsible for mining GitHub repositories and creating the artifact traceability graphs at Neo4j.

Ceyda Şahin

Ceyda is responsible for writing queries for the artifact traceability graph. For example, given a file name, she is working on methods to return the most knowledgeable users by using the artifact traceability graph.

Ali Eren Günaltılı
Ali Eren is responsible for the theoretical background of our recommendation engine. That is, which metrics should our recommendation algorithm use to return the best results.

Important Note: It should be noted that, even though each member is mainly responsible from one section of the system, no team members work alone in any of the above mentioned fields. In weekly group meetings, members exchange ideas and solve problems that each member encountered in that week together. From time to time, some members work in other fields to help a teammate depending on which part is more urgent or missing. Also tasks like writing a report or preparing a presentation are done by the entire team.

7.2 Helping creating a collaborative and inclusive environment

Since our team has never faced an *uncollaborative or non inclusive environment* there is no special effort that is being put to achieve a collaborative and inclusive environment. Each member is considering other team members' workloads and occupancy levels. When a particular team member is occupied for one week, other members cover his/her responsibilities. Thus, work is never interrupted and no team member is left with a difficult situation.

At the beginning of this semester, each member is affected on a different level by the earthquake disaster. At this point, every team member and our instructor Mr. Tüzün acted extra considerate towards each other. No team member was pressured about completing their tasks in those two weeks. This is a concrete example and proof of how each team member and our supervisor is taking an active role in creating an inclusive environment.

7.3 Taking lead role and sharing leadership on the team

During these two semesters, our team followed a method in which the team leader was changed biweekly. That is, the responsibility of maintaining the team, scheduling the meetings, and distributing the tasks was assigned to a different team member every two weeks. This way, the motivation of each team member is kept high all the time and every member gets equally involved with the project.

8. Glossary

Some of the terms we commonly used throughout this report are as follows. The correct identification of these keywords is important to avoid confusions and maintain the consistency throughout the entire project

Initializing a repository: It is important to note that when the word of repository is used, we imply the repository that our extension will be included in. Thus, the initialization of the repository doesn’t mean creating a new repository on GitHub but it means introducing a repository that already exists on GitHub to our system.

Joining a repository: Similarly to “initializing a repository”, joining a repository doesn’t mean joining a repository in GitHub. It means a developer can become a member of an initialized repository so that they will be able to use our extension for that repository.

Repository Owner: Repository owner in our system is a special developer who has more authority in our system. By default, the developer who initializes a repository to our system becomes the first repository owner for that repository.

Developer: Developer is a user who has created an account in our extension and joined a repository. Developers are the ones who “get help” from the expert developers.

Expert Developer: Expert developers are not any different types of developers. In essence, every developer is a potential expert developer. We created this term to distinguish the developers who “ask for help” and the other developers “who help the developers who had asked for help”. Thus, it is important to note that expert developers are in fact developers but this keyword is created only to emphasize the different roles a developer can get.

VS Code Extension: As the name suggests, this is the extension part of our system which will be available on the VS Code Extension market. Developers will be able to get recommendations from this part whereas they have to register or manage their repositories from the web management tool (see description above).

Artifact: Our algorithm will be mining the repositories that are stored on GitHub and acquire the artifacts to build the artifact graph. Some examples of such artifacts are as follows: Commits, Pull Requests, Opened/Closed Issues, Pull Request comments, Issue Comments, etc.

Artifact Graph: As the name suggests, artifact graph is a graph constructed out of artifacts (see description above). The graphs will be interconnected with each other according to various metrics. For a recommendation to be made, this graph will be used. The connections from the nodes of the artifact graph (artifacts) will be investigated and evaluated to determine the knowledgeability levels of expert developers.

9. References

[1] H. A. Çetin and E. Tüzün, “Analyzing developer contributions using artifact traceability graphs,” *Empirical Software Engineering*, vol. 27, no. 3, 2022.

[2] “Git-blame documentation,” *Git*. [Online]. Available: https://git-scm.com/docs/git-blame. [Accessed: 13-Nov-2022].

[3] “Gitlens features - supercharge git in VS Code,” *GitKraken*, 10-Nov-2022. [Online]. Available: https://www.gitkraken.com/gitlens/features. [Accessed: 13-Nov-2022].

[4] “Perceval.” CHAOSS, Mar. 01, 2023. Accessed: Mar. 12, 2023. [Online]. Available: https://github.com/chaoss/grimoirelab-perceval.

[5]   “Webhooks and events documentation,” *GitHub Docs*.
https://ghdocs-prod.azurewebsites.net/en/webhooks-and-events. (accessed Mar. 12, 2023).

[6]   “abstract-syntax-tree,” *npm*, Apr. 11, 2022.
https://www.npmjs.com/package/abstract-syntax-tree. (accessed Mar. 12, 2023).