Knowledge Traceability in Higher Education

Jesper Nilsson - ejeino-7@student.ltu.se
Johan Rodahl Holmgren - ojaohe-3@student.ltu.se
Robin Danielsson - robdan-7@student.ltu.se
Tommy Andersson - anetom-6@student.ltu.se
Wilma Krutrök - wilkru-7@student.ltu.se

16 mars 2020

Table of content

1. Introduction

This report is part of an assigned project in the course “*Project in computer science and engineering*”. The purpose of the project is to give students, teachers and student counsellors a tool for viewing and evaluating information about the knowledge given in different courses. The tool itself will hopefully provide students with a wider understanding of how different courses are connected and why parts of them are important for future studies.

To measure knowledge the project is using knowledge components, KC:s. A knowledge component is a short description of the knowledge combined with a value that is specifying the level of knowledge. The level is called taxonomy level. Using this information teachers will be able to see how changing a part of a course would affect other courses given to the same students.

All code for the project will be maintained at a github repository [1].

1.1 Background

The project owner Jan van Deventer stated a problem with students not having the knowledge needed when a course starts. To solve this problem Jan wanted a software for teachers to easily see what the students know and how long time has passed since they took a course with a specific “knowledge component”. The software should also be used by students to keep track of the importance of learning specific parts of courses.

One attempt to solve this problem has been done. The solution was using Google Sheets and Matlab which turned out to be very inefficient. One big challenge was that teachers named the knowledge components differently, resulting in chaos. The program was not able to see if something was misspelled, used shortening or other similar things.

1.2. Problem description

The main problems are divided into different parts. One is that some courses overlap while others have a missing link, i.e. the knowledge required for one course is not given in previous courses. Certain prerequisites are not obvious between courses since some details are not written in the course description. When interviewing the teachers it became apparent that most of them had, at least once, encountered the problem where most students had not been given the knowledge required for some part in a course.

1.3. Assignment

To solve the problem a software will be implemented. The software will be web based and interactive. Users should be able to search for programs and courses to see how the courses are connected and what knowledge component each course requires and develops. This will motivate the students, and help administrators and teachers to plan the optimal course order. Teachers should be able to see what knowledge components they should teach and on what level.

1.4. Delimitation

Due to the limited time set the project group will not focus on collecting the data that is needed for the solution to work. This job will be placed on the examiners together with a group of engaged teachers. The system will not be fully integrated with the system the university is currently using.

2. System Design

The software is planned to be implemented as a Java program utilizing a graph database named Neo4J [3]. In that way classes and objects can be implemented as related components. Java also got the libraries needed to communicate with Neo4j. Courses and knowledge components will be implemented as objects with different parameters and represented as nodes in Neo4j. Courses will be connected to knowledge components containing the taxonomy level. It will also make it possible to make the connections between the different objects clearer and easier to change.

All the data for courses and knowledge components will be stored in a Neo4J database with the same structure as in the Java objects. The reason for choosing Neo4J is because it is fast with graph relationships and the way it is constructed makes it easy to create and connect nodes. Storing the data in a database will give a good overview and access to the data. Connection to the front-end program from the Java program will go through a server.

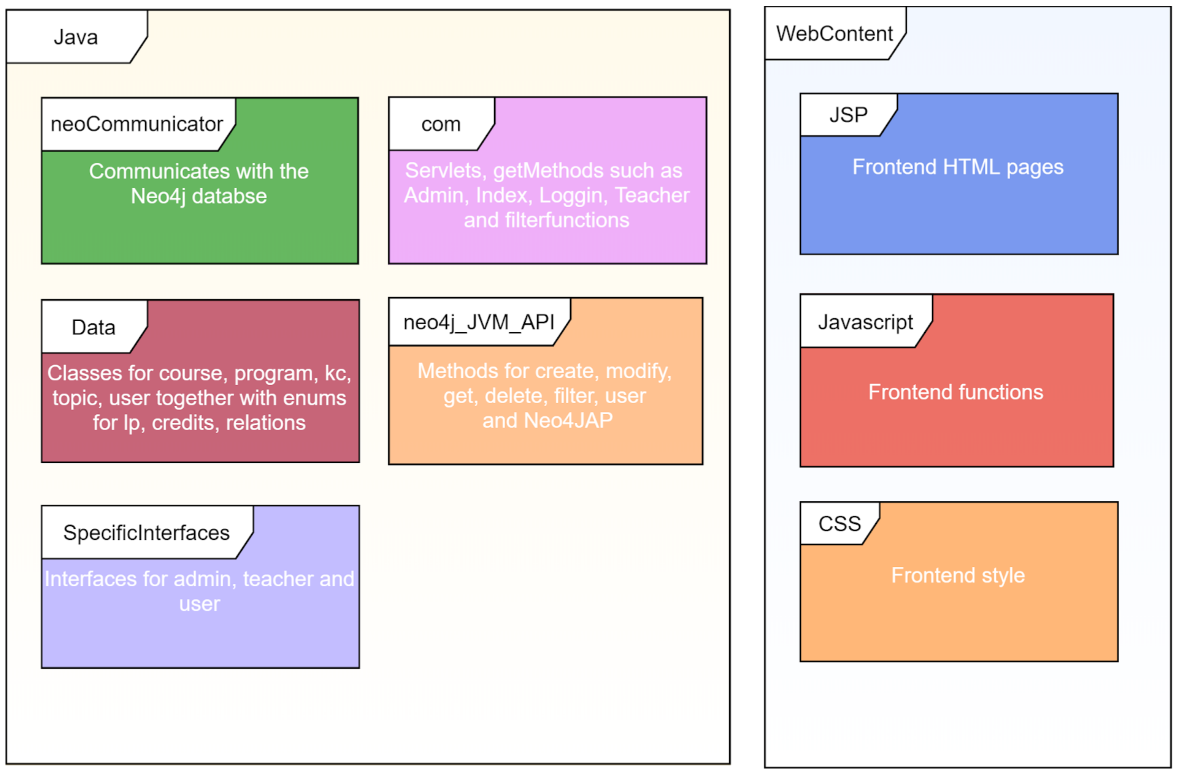

Java
neoCommunicator
Communicates with the Neo4j databse
com
Servlets, getMethods such as Admin, Index, Loggin, Teacher and filterfunctions
Data
Classes for course, program, kc, topic, user together with enums for lp, credits, relations
neo4j_JVM_API
Methods for create, modify, get, delete, filter, user and Neo4JAP
SpecificInterfaces
Interfaces for admin, teacher and user
WebContent
JSP
Frontend HTML pages
Javascript
Frontend functions
CSS
Frontend style

Figure 1: Overview of the program structure by a module diagram

Figure 1 is an overview of our system design. The user will access the system using a web browser. The Java program will contain several JSP (Java Server Pages) classes running in a Tomcat container and connect to the web browser using HTTP requests. All database communication will be handled by the Java program.

3. Implementation

3.1. Working module

SCRUM has been used every time the group has worked together. For these meetings a SCRUM master has been elected which controls the pace of the meetings. Every person must answer three questions: What have you done? Is there something you want help with? What will you do until the next meeting? The person should answer truthfully for the development to carry on. If complications occur, it is important that the team works together to find the best solution, instead of blaming a group member. The meeting should take around 15 minutes if not something critical has happened since the last meeting.

GitHub issues have been used for group members to know which function is most critical to finish for the development to move on. New issues will be made if the group members find that some functions are missing. Each issue will have a weight and the lower the digit the higher the priority, where 0 is the highest priority. When a function is tested and considered safe it will be closed.

For some tasks extreme programming is used. This is a method that is used to improve the quality of software by working in pairs or smaller groups when writing code. The method will also promote better team work, communication and understanding of the software.

3.1.1 Reflections on working module

All members of the group have taken responsibility for the development of the project. During the meetings with the project owner all members of the group have been active and talked about how the project is moving forward. During SCRUM meetings the group has discussed the planning of tasks. One exception to SCRUM meetings is when some, more urgent topics, has been brought up between meetings. Common discussions were held regarding the scope of the project and how a specific feature should be implemented and if features were necessary.

Roles were not assigned to all members early on. Each member has instead earned their role by showing certain knowledge and skills about some specific topic. Although, these roles have not been concrete. All members have had the opportunity to try new things and learn from other members with other roles. This helped when the knowledge gap between members meant that the project got stalled if one member had to leave.

3.1.1.1 Jesper

Jesper is one of the group members. During the project he pushed the group to obtain high goals for the project. Since the project started the only possible end has been to complete the project. Jesper has not been assigned a specific role by the group, but during the implementation he has been flexible and helped other group members. He has worked a lot with extreme programming where experience and knowledge have been shared. When looking at the whole picture of the project, Jesper has been involved in most of it. Major things to mention is the responsibility for the filter container together with Wilma and responsibility for the admin page together with Tommy.

3.1.1.2 Johan

Johan is the Git master and is responsible for the Github repository. Handling any question or issue regarding Git as a platform to ease the development process. Johan also instructed the group on proper procedures utilizing git, ultimately maintaining and deciding the development structure to meet end goals. Also responsibility for the relevant server infrastructure during development. Additionally Johan has assisted the development.

3.1.1.3 Robin

In the absence of the Git master, someone had to learn how to properly merge branches and maintain the repository. Robin has been the secondary Git master with the task to offload Johan and learn how to use the Git terminal. This role has also included encouraging the others to use proper commit messages and branch names. Other responsibilities have been to make sure everyone maintains, at least, some rudimentary coding conventions and best practices such as to not hardcode strings and values instead of variables and generic data types.

3.1.1.4 Tommy

Tommy was elected to be the SCRUM master of the project and therefore was in charge of the meetings and made sure that everyone's opinion was heard. Outside that there was not much that distinguished that role from the rest of the group. His main focus was planning, discussing ideas and talking with the project owner. Tommy has mostly coded in pairs with someone but has done some solo coding and code reviewing, mostly in the back end. The biggest part of the program Tommy worked with was a collaboration with Jesper on the admin page.

3.1.1.5 Wilma

As one of the group members Wilma has been part of project development and planning tasks and meetings with the project owner. Regarding the implementation she has worked dynamically and has helped where needed. Her responsibilities were for the filter container together with Jesper, the teacherpage and the style for the front end.

3.2. Project implementation

To work with the project the group has regular meetings where the SCRUM master directs the meeting, assigns tasks and has the final word when voting if a stalemate. All tasks are assigned as issues in GitHub together with the documentation. The Git master has the responsibility to merge everything into the master branch and to make sure everything is correct. Once a week a follow up meeting with the project owner is scheduled. At the meeting the work since last time is presented and feedback is received. After the feedback the group together with the project owner discuss what should be done until the next meeting.

For sprint 1 the group has been working together in the assigned project room at the university. This work style has made it easy to work in small groups of two or three and ask other group members for help, which also made it possible to work together or individually depending on the task. The group has found this to be an efficient method because everyone has been synchronized and learned from each other to optimize the workload.

Things that could have done better is documentation over what has been done, is tested and ready to use. Communication is something that could have worked better since communication is essential in teams and can always be improved.

Sprint 2 started earlier since most from sprint 1 was done before the deadline. The working method was similar to sprint 1 and the group sat together and programmed in pairs. On sprint 2 the group members specialized themselves more in the code, some coded the admin page and some coded the canvas of the system. This strayed from the path that the group members should be able to jump in if someone gets sick.

Documentation is still something that could have been improved and testing was only done manually and is limited.

3.2.1 Sprint 1

For sprint 1 the main focus is to make sure that Neo4J could be used for the project. Since Neo4J is schemaless it is important to implement a schema structure for the data. To make sure all data storage follows the same structure an API was implemented. In the API a set of methods are implemented and these methods are the only one that is allowed to be used when communicating with the database. Using this method potential bugs should be minimized if implemented correctly, and the system will be well prepared for implementing the front end.

The planned stories for sprint 1 (Story 1-4) have been completed. For each week the group planned around seven meetings to work on the sprint. Every meeting started and ended with a SCRUM meeting where everyone was asked what they wanted to work with and if any problems had occurred. After every SCRUM meeting time was allocated for discussion and re-planning before continuing with working on the planned tasks. Some time has been spent for creating test-benches and finding bugs.

In the middle of the sprint, the group decided to have a bigger planning meeting. During the meeting changes were made to the system design regarding the API and how the system should interact with the web interface. Since no group member has any previous experience with using JSP and Tomcat, time has been spent setting it up and making sure that the Java code is usable. After gaining some understanding of JSP and Tomcat, the group made drawings for the graphical view of the frontend and decided what files the project should contain.

3.2.2 Sprint 2

The main focus of Sprint 2 has been writing Javascript and HTML/CSS. Since the members of the group are inexperienced with these languages some time has been spent on learning. Major challenges have been the communication with the Java server program and formatting of data when transfering.

The planned stories for Sprint 2 (Story 5 - 8) have been completed. For each week the group has planned 7-8 meetings for development. During these meetings the group has used the assigned working room and been programming together. Every meeting started and ended with a SCRUM meeting.

3.2.3 Personal reflections

For the implementation in sprint 1 and sprint 2 each member reflected on the work done and the estimated time which is presented in this section.

3.2.3.1 Jesper

During the project the group has worked with tasks that members got zero or very little experience to work with. Easy things like styling a small HTML-div can easily take a lot more time than predicted. Working with Java on the server is much easier to estimate since everyone in the group has more knowledge about Java. The planned tasks for Sprint 1 finished faster than expected, and more time was spent on the tasks for Sprint 2.

During Sprint 1 Jesper spent a lot of time working in pairs. This was good since experience and knowledge easily was shared between the programmers. During Sprint 2 Jesper was responsible for the filter container together with Wilma and the admin page together with Tommy. Apart from that he implemented most of the sign in functionality, and also the graphical admin view when making changes to programs. Most of the time was spent on the admin page.

3.2.3.2 Johan

This project quite quickly ran into the ninety-ninety-rule, when a big feature was done and only a small implementation remained it was not apparent that the time to implement would be far greater for the smaller features. Mostly Story 5 implementation of the graph was grossly underestimated, time taken was roughly 100 hours, for only the 2 first tasks. The problem was never the amount of code needed to be worked on but the time to solve the theoretical problems proposed by graphics on a web page and mapping abstract points to each other. Then when a big feature was complete, the group was ensured that it is almost done, 90% of the work is completed but 90% of feature still remains. Aside from that most development proceeded smoothly and according to plan, if not at time poorly operated planning preceding with bad protocol.

Mostly during the first sprint the work was going very smoothly but often in disjunktion between group members made testing and subsequent fixes made a big mess. Git master had to fix merge issues that took a considerable amount of time and often swamped actual development time from git master. As the group improved in utilizing git, structure was established and the group felt comfortable working with the repository, the workload then shifted more towards development rather than administering conflicts. It is however to save time for developing the graph, testing was mostly scraped.

3.2.3.3 Robin

Estimating the time needed for one task is not complicated, it is the estimation of multiple tasks that is difficult. Time will quickly add up and delay the next task a little further than the previous one, until the work done during one week is substantially less than what was planned. Meetings, discussions and planning take time as well, which can even further delay the work progress.

Too much time has been spent on testing different designs and code structures. The final version of the graphic canvas was not difficult to implement, but the work that was put into testing different implementations would not have been necessary if the implementation were known from the start.

Some code from sprint 1 had to be patched up and fixed as well. Some mistakes were made when it comes to sending data between the web browser and the server. The time spent on finding bugs was not much, but it definitely slowed down the development of sprint 2.

3.2.3.4 Tommy

It was hard to estimate the time for all the tasks before the project started since the group had very limited experience with this type of project. Since there was not a clear idea what was going to be implemented, a lot of time was spent on planning. Most of the time planning was not so accurate, either tasks were finished too early or later than anticipated. This worked in the end but it was a tight fit. Neo4J was easier to work with than expected but JSP on the other hand took longer time to set up.

During sprint 1 all Tommys efforts were put into the backend and more precise API, this was due all the frontend needs data from the backend. In this phase the group worked in pairs, this was to even the skillcap between the group members. For sprint 2 Tommy worked for the most part together with Jesper on the Admin page, this included both the planning and the implementation. Other tasks were to test the system and work with getting data from the backend to the frontend.

3.2.3.5 Wilma

The sprints were divided into backend and frontend and therefore the first estimation was that the two parts would take equally long time. This estimation turned out to be good, mostly because in both sprints the group needed to learn new things they had not worked with before. For the backend the group was able to learn how the database Neo4J worked together with servlets and for the frontend the group learned to work with JavaScript.

The most work with the backend was done some time before the deadline for sprint 1 and therefore the group started on sprint 2 in parallel with testing the implemented functions in sprint 1. It has been difficult to estimate big tasks but the group divided these into smaller parts which was easier to estimate.

For sprint 1 Wilma has worked with the API for Neo4J and with parts of the data types. In sprint 2 most time was put into the teacher page, style for the frontend and the filter container together with Jesper.

3.3 Resource and activity planning

The project has been divided into two themes, backend and frontend. Backend will be covered in sprint 1 where stories and tasks considering storing, searching and modifying data in the database will be handled. For sprint 2 the frontend will be made including the user interface.

3.3.1. Sprint 1

The theme for sprint 1 will be the backend for the system. This includes how the Java program is communicating with the database.

Story 1

1: Title
Write to the Neo4J database using the server program.

2: Intended use
Will be used to write data from the Java program into the database. One example is creating a new course or a new knowledge component.

3: Desired properties
It is important that the writing is working correctly and does not overwrite data that exists in the database. The structure for the nodes that are stored need to be correct and the functionality must be implemented in an efficient way.

4: Test case
Run a test bench that is creating new nodes and stores them into the database using all available functions.

Tasks

1.1
Description: Store a new course in the database
References tasks: none
Estimated time: 6 hours
Estimated risk : 8/10

1.2
Description: Store a new KC in the database
References tasks: none
Estimated time: 6 hours
Estimated risk : 8/10

1.3
Description: Create and store a new user
References tasks: none
Estimated time: 5 hours
Estimated risk : 8/10

1.4
Description: Store a new topic
References tasks: none
Estimated time: 4 hours
Estimated risk : 7/10

1.5
Description: Store a new course program
References tasks: none
Estimated time: 10 hours
Estimated risk : 7/10

Risk for the story: 8/10

Story 2

1: Titlel
Search in the database

2: Intended use
Search the database for data. This will be used when a user searches for a program and all the courses relations. This will also be used when admin and teachers are to login to modify courses or add new ones.

3: Desired properties
It returns correct values with no possible way to alter other functions. No value (null) shall be returned if no data was found.

4: Test case
Rigorous search tests to see that the database returns the correct values. Stress test it with input that should be illegal.

Tasks

2.1
Description: Search for courses in the database
References tasks: 1.1
Estimated time: 3 h
Estimated risk: 2/10

2.2
Description: Search for KC in the database
References tasks: 1.2
Estimated time: 2 h
Estimated risk: 1/10

2.3
Description: Search for users in the database
References tasks: 1.3
Estimated time: 4 h
Estimated risk: 5/10

2.4
Description: Search for topics in the database
References tasks: 1.4
Estimated time: 3 h
Estimated risk: 2/10

Risk for the story: 5/10

Story 3

1: Title
Modifying database content

2: Intended use
When a course or knowledge component is to be altered, a connection to the database is established and the desired data is modified.

3: Desired properties
There will be no alternation if there exists no such entry in the database to begin with. The user must also be verified and have the rights to modify the database.

4: Test case
Manual testing that includes deleting and modifying objects in the database. The same tests must also be done as a user without permission.

Tasks

3.1
Description: Modify course content in the database
References tasks: 2.1
Estimated time: 3h
Estimated risk : 7/10

3.2
Description: Modify KC content in the database
References tasks: 2.2
Estimated time: 2h
Estimated risk: 7/10

3.3
Description: Modify existing user credentials
References tasks: 2.2
Estimated time: 4h
Estimated risk: 9/10

3.4
Description: Modify course program content.
References tasks: 1.5
Estimated time: 3h
Estimated risk: 7/10

Risk for the story: 9/10

Story 4

1: Title
Setup the Neo4J server.

2: Intended use
The database that will store all the data for the system. This includes courses, knowledge components, student programs, system users, topics and more.

3: Desired properties
Needs a lot of memory and high reliability. It also needs to be configured with correct access from a security viewpoint.

4: Test case
Run tests to check if it is accessible from only the server computer and not from other computers.

Tasks

4.1
Description: Install the server software on the server machine
References tasks: none
Estimated time: 5 hours
Estimated risk: 10/10

4.2
Description: Give the correct access
References tasks: none
Estimated time: 3 hours
Estimated risk: 10/10

Risk for the story: 10/10

3.3.2. Sprint 2

The theme for this sprint is the system frontend. Here the web pages will be set up and the user interface will be in focus.

Story 5

1: Title
Displaying the Graph

2: Intended use
Let users see results in an “easy to the eye” graph, with isolation of data.

3: Desired Properties
Courses and knowledge components should be displayed clear and plain on the page.

4: Test case
Third party program intended for testing on frontend. Other than that manual testing by viewing and testing results.

Tasks

5.1
Description: Display Course order on page
References tasks: 2.*, 4.*
Estimated time: 15 h
Estimated risk : 5/10

5.2
Description: Connect courses and knowledge components
References tasks: 5.1
Estimated time: 8 h
Estimated risk : 5/10

5.3
Description: application protocol to communicate to server, fetch new and publish data
References tasks: 5.2
Estimated time: 6 h
Estimated risk : 4/10

Risk for the story: 5/10

Story 6

1: Title
Logging in

2: Intended use
Logging in an existing user by using the individuals username and password.

3: Desired properties
Two text bars where you write username and password and a button for logging in.

4: Test case
Registering an account then trying to log in if it fails then something is wrong, either username, password or something else that needs a closer look.

Tasks

6.1
Description: Logging in
References tasks: 2.*
Estimated time: 6 h
Estimated risk : 4/10

6.2
Description: Register Account
References tasks: 1.3
Estimated time: 8 h
Estimated risk : 8/10

Risk for the story: 8/10

Story 7

1: Title
Search data from database via web browser

2: Intended use
Users should be able to display data from the database for example when searching for a program, course or knowledge component.

3: Desired properties
The functions for this story should not affect other parts of the program. It should be simple and user friendly.

4: Test case
Run a test program that searches for a student program and get the desired overview over the course plan.

Tasks

7.1
Description: Search for knowledge component or course
References tasks: 2.*
Estimated time: 10 h
Estimated risk: 2/10

7.2
Description: Search for program with specialization
References tasks: 2.*
Estimated time: 6 h
Estimated risk: 2/10

Risk for the story: 2/10

Story 8

1: Title
Modify data in database via web browser

2: Intended use
Make it possible for users to add/change courses and knowledge components. It should also be possible to add or remove relationships between courses and knowledge components.

3: Desired properties
Should be implemented correctly to go via Java program functions and not directly to the database. Need to be user friendly and catch errors.

4: Test case
Run tests with different inputs and compare with expected results.

Tasks

8.1
Description: Add course
References tasks: 1.*
Estimated time: 10 h
Estimated risk: 5/10

8.2
Description: Remove knowledge component from course
References tasks: 2.*, 3.2
Estimated time: 8 h
Estimated risk: 6/10

8.3
Description: Edit information for knowledge component
References tasks: 2.*, 3.2
Estimated time: 8 h
Estimated risk: 5/10
Risk for the story: 6/10

3.3.3. Reflections, planning, implementation and time tracking

In general the work with the sprints has gone well. The planned tasks for each sprint have been completed and resulted in a system that can be demonstrated. The most difficult task was to plan the detailed picture of the whole system. The plan changed during the development process when the group started to understand how the system should work and what problems there were to solve. In the beginning parts were very abstract and difficult to plan in detail.

Sprint 1 was almost completed one week before the deadline so the group started to work in parallel on sprint 2 earlier to keep the workflow. Before the beginning of sprint 2 the plan for the whole project was reviewed and updated due to more insight about the overall structure. For the majority of time all members of the group have had something to do and been able to work with different functions at the same time.

Some functionality that has taken more time than estimated in section 3.3 was the graphical view and to set up the server environment. The methods for Neo4J communications and implementing the data types did on the other hand take less time than estimated.

When developing the system the group has been working together in the same room. That made it possible to have meetings in person and to decide who does what and to discuss upcoming problems right away. One drawback was that written documentation therefore did not become a priority. Another difficulty was that when talking about time the same words did not have the same meaning for every group member. This led to more waiting time than necessary and not the optimal allocation of resources.

4. Result

4.1. Delivery

The finished project was demonstrated at the seminar on March 12. Since this is a big project the group was not able to deliver a fully functional product that is ready to launch. To make the product as good as possible and easy to use it needs more time. The solution demonstrated in the seminar still delivers what the group agreed with the supervisor to develop.

4.2. Testing

4.2.1. Test strategy

When writing code the programmer has been responsible for only commiting code that is working correctly to the Git repository. For testing functions and methods that depend on each other testbenches have been made. Further on in the project when functionality was accessed through the browser it has been easier to test functionality and debug to find code that was not working as intended. Before the demonstration useful data was put into the database using the functionality in the browser. Using this data the group planned what functionality to demonstrate and ran the demonstration multiple times to make sure it did not contain any bugs.

4.2.2. Regression risks

When updating the software there is a risk for features to stop functioning. Based on how the user input is handled in functions together with the information in the database, seen in system design - section 2, the following regression risks are considered.

Local regression

Changing a function can result in a bug appearing in the function. One such thing is that the input to the database is not secured and allows incomplete queries. It looks like the data is added to the database when it is not. This error is hard to discover, but the current solution to mitigate the possibility of this kind of regression is the usage of an interface between the server and the rest of the system. However, if this interface were to have a risk of regression the rest of the system could be at risk as well.

Another local regression risk is caused by any inconsistency in variable names between the database and the client. A direct consequence is that the values can not be accessed from the desired variables or the returned values will not be stored or used in the wanted way. But this problem should also be prevented by the server API, since only one part of the software will be responsible for interfacing with the database.

Unmasked regression

It is important that all functions are designed to minimize bugs. Meaning it should be able to handle non valid input. Otherwise a new function can introduce new bugs in previous implemented functions. Unmasked regression can occur when the input used is changed and the function is not handling the new input. If a function is designed only for specific inputs it must throw an exception if the input is invalid.

For example, if a function is designed to iterate over an array, it must throw an exception if an empty array is the input. To make this work it is important that each function have a specification for the inputs it can handle, and also that it is tested for all the inputs. All the communication to the database goes through the Neo4J API and this reduces the risk for unmasked regression in the database. The package Graph could possibly unmask bugs for relations in the database or in functions for the package Neo4J API.

Remote regression

A certain function could be edited in a way that breaks some other function because the behaviour is different. Whenever some function is changed, that function must retain the same behaviour as specified in the specification. If the behaviour is different it could cause unwanted results in other functions that rely on the code that was altered. E.g. if component A writes to the database while component B reads from it, there is a possibility that a change in A causes the output to be altered, or even corrupted, thus breaking B. Therefore, the database API must write and read in a unified fashion in order to minimize the risk of bugs between the rest of the software and the server.

These regressions show that it is important to have a well defined structure over the database and how the system should interact with it. It is also important to specify what all functions take as parameters, does and returns. All invalid inputs should be handled, or at least be clearly stated in the documentation of what the function can and cannot handle.

4.2.3. Strategy for regression testing

To test the system to avoid regression a strategy is to have a set of predefined input and output. When a new function is implemented the set is tested to see if the right output is given to guarantee no new bugs have been introduced. When a bug is discovered it will be fixed and put on a high risk list to be checked regularly to see that it does not create or handle new bugs.

Traceability

A selection of tests are made to ensure that the software behaves like it is supposed to. Tests for the documented quality risks will also be used. To be able to conduct these tests, activity charts and sequence charts for the functions are needed.

In software development, a third party application (automatic testing) will help in identifying issues with current development. All code will be placed in a development branch where it will be tested with a set of predetermined variables and types. Before the test the function will be analyzed to see how the function should behave and the test result will be noted. If all checks are clear the function will be considered stable and put in a compile Branch for further testing.

Change analysis

Since the code interface with the database, any mistake has the possibility to grow and eventually reach the database. If the graph were to crash or produce an anomaly, data could be stored in the database if the process has teacher privileges. A change in the graph node classes could also cause unwanted behaviour. The graph could be rendered incomplete, or not at all if the mistake is severe enough.

In order to prevent these problems every possible output a changed function or object can generate needs to be analyzed. The output must then be used as input to every component that utilizes said function. E.g. if an if-statement in a function creates a new code path, that code path will have its own set of possible outputs that may be different from what some other function expected. The same situation can be created by a thrown error that did not exist before. It is therefore important to test for regressions when the specification of a function or class is altered in a way that does not affect functionality unless there is some initial error.

Quality risk analysis

There is a moderate risk that bugs that are damaging appear inside the web application or backend system. Additional risk is asserted in the fact that there is not enough user support to make the application useful in some cases, this however is a low damage but high risk. If the project is completed earlier than expected, the final application might have this issue resolved. Conflict of data storage; since having multiple database solutions and neo4j being quite unfamiliar and utilizing the other database, there is a high risk of conflicting entries unless the final application solves the issue by other means.

The issue of knowledge components is that it has to be arbitrarily inserted by teachers via system administrators. Main issue is, even with good fundamental structure duplication of KCs might occur. Additionally the issue of inserting all KCs might be too large of a task with low risk.

To avoid risking the quality for the user and to reduce the business risk it is important to have good documentation. To do so the functions should be well documented, containing a description together with stating the input, output and assumptions. It is also important to have well descriptive names for functions and variables. The predefined input should cover the expansion of the software to guarantee the low business risk.

Cross-functional testing

Cross-functional testing is done by testing different functions which should not affect each other in a test suite. By doing this the expected result would be the same as running the functions in isolation. If the result is not the same, it means that the functions contain a bug. Using this function a tester can focus on areas of the software that contained bugs in the previous test. This will ensure that no new bugs were created when the old ones were fixed. The software will be web-based and handle multiple users at once. When handling more than one user it is always important to lock global variables when using them. The strategy that will be used is a wide search for bugs, for example check global variables or functions used by other functions. If a bug is discovered it will be fixed and added to the high risk list.

4.2.4. Development description

To test the system for regression stories for automated unit level and automated system level testing is described. The stories are divided into tasks, with estimated time to finish, priority to be finished and risk. Where risk is defined as a summary of different requirements, implementation and dependency risks.

Automated unit level testing

Unit level testing is small parts of a code tested in isolation to make sure that the modules are working properly. This will make it easier to detect bugs and this kind of testing is usually performed by developers.

Story 9

Title: Test of the Neo4J API.
Intended use: Automatically create, edit and delete nodes and relationships using the developed API.
Desired properties: Will cover all possible inputs for the API
Test case: Run the test with both valid and invalid input
Estimated time: 18h, **Risk:** 8/10, **Prio:** High

Tasks

9.1
What to do: Create a script that starts the Neo4J module.
Estimated time: 4h, **Risk:** 8/10, **Dependency:** No dependency

9.2
What to do: Make a script to create, modify and remove KC, program and courses in the module Neo4J.
Estimated time: 9h, **Risk:** 8/10, **Dependency:** No dependency

9.3
What to do: Create a script that checks if the data in the database is correct after creating, modifying and removing KC, program and courses in the module Neo4J.
Estimated time: 5h, **Risk:** 1/10, **Dependency:** No dependency

Automated system level testing

System testing involves several modules in one test. It could be used to test a function for the user to execute that will start functions in other modules of the system to give the asked output.

Story 10

Title:
Automatic test of all functions involved when getting a graph from the database containing a student program.
Intended use: Run a request for all available programs to make sure that all inputs give the expected output.
Desired properties: Will be easy to set up for testing.
Test case: Test should run with different input, both correct and input that should not work.
Estimated time: 8h, **Risk:** 4/10**, Prio:** High

Tasks

10.1
What to do: Develop a script to call the system with all possible inputs and some invalid input.
Estimated time: 4h, **Risk:** 4/10, **Dependency:** No dependency

10.2
What to do: Use the script from above to check that the graphical view is working.
Estimated time: 4h, **Risk:** 4/10, **Dependency:** No dependency

4.3. Ethics

The system stores for example passwords for users which is sensitive information. Therefore it is important to provide the users security that can guarantee to keep the sensitive data secure. The planned implementations in sprint 1 and sprint 2 do not store any personal data. Further implementations may store information such as individual programs and student grades. Implementing such functionality must increase the security for the whole system, because it may be sensitive for the users. This kind of data can be seen as harmless, but it can also be used by organizations for tracking human behavior and targeted advertisement.

To obtain the highest quality for security it is important to test the system and make sure that passwords and other potentially sensitive data only can be accessed by an authorized user, as well as hashing the password in case it does. When a user is logging in, the server is responsible to compare the hashed password in the database with the hash from the user input, and in the client.

Before using parts of the system with sensitive information all methods have been tested to see that no information is accessed in the wrong way. If this is not done right and does not cover all exposed functionality then the sensitive information is at risk. Therefore it is important to keep track of what has been tested and what has not.

4.4. Further implementations

4.4.1 Improvements

Before taking the system into use there are some functionality that need to be discussed and modified to fit the desired specific use cases. These functionalities are described in detail in the appendix B. The reason there are functions not completed or not optimized is due to lack of time or the knowledge needed are missing. The system handles terms like courses, knowledge components and study programs. These need to have a clear definition to be implemented in a useful way. To be able to make a demonstration the group needed to define terms and make assumptions. Therefore some functions in the system need to be redefined to fit all universities.

In addition to redefining the different terms in the system there are other things that also could be improved. More time could be put into the front end to make the webpage even more user friendly. It is also important to implement more security to minimize errors. For example add more checks at the admin page so the same course is not created twice or at the teacher page so needed knowledge components can not be deleted.

4.4.2 Possible implementations

In this subsection possible implementations for the system are described. These are all summaries and more detailed information about them will be in the Blacklog(Appendix C).

The system is built so it can be modified in the future, a few things that can be implemented in the future is user login for a personal reading order. This would help students to check how they progress in their studies and if they have to do some modification to their own reading order.

Implementing *Teaching activity* (TA) and *Intended learning outcome* (ILO), TA is a way for both student and teachers to see where the students gain their KC:s. A TA will specify how each KC is developed, by for example refer to a lab assignment or a seminar. ILO is a description for how the knowledge in a course will be used more practically, like solving a set of problems. Adding all ILO from the courses in a program will give a better understanding of what the student will be able to do at the end of the program.

In the future this tool can be used to optimize courses but to do so more structure is needed for the KC:s and TA:s. To do this further analysis needs to be done and KC:s needs a better definition.

5. Conclusions

This project was a learning experience for the group. Apart from working in a bigger group containing five to six people, working together in a development platform of this size is new. In the middle of the project one group member left due to personal reasons. Since the group had little experience with web development the members have found methods for easily finding information and examples of how to solve different tasks. The project gave experience of working with a project owner, planning, team work, system design, and also programming with usage of libraries.

Due to the limited time set for development of the project some things will be less prioritized. Security is often the easiest to sort away. Some security is implemented, like passwords being hashed.. Testing is also very time consuming and easy to not prioritize. All functionality available in the browser is tested and works for correct and expected inputs, but some functionality is not tested for invalid inputs. During both Sprint 1 and Sprint 2 some smaller testbenches were created but were not covering all the functionality.

The project is at a stage where it could be used for demonstration of the concept. If the database were filled with more courses, programs and knowledge components it could actually be used as a product, but with questionable stability. The purpose is not to utilize it at this stage, but to future development, possibly by assigning another project group.

Neo4J was selected early as the database for the project. It would be interesting to compare with usage of another type of database such as a relational database. The usage of Neo4J is adapted for handling huge amounts of relations. This property of Neo4J could be seen as a big advantage for the project, but since the nodes in Neo4J are converted to Java objects and then converted to JSON before sending to the browser many advantages with the database are lost. The usage of Java could also be reflected on. Perhaps usage of other frameworks would have saved development time, even though more time would have been spent on learning.

The project was experienced as abstract and big compared to other projects that have been done before by the group members. The hardest part was to implement vague ideas from the stakeholders.

6. References

[1] J. Holmgren et al. “Knowledge Traceability in Higher Education”,[Online]. github.com. Available: https://github.com/ojaohe-3/d0020e [accessed: March. 16, 2020]

[2] Refsnes Data. [Online]. w3schools.com. Available: https://www.w3schools.com/ [accessed: March 16, 2020]

[3] Neo4j, Inc. [Online]. Neo4j.com. Available: https://www.neo4j.com/ [accessed: March 16, 2020]

Appendix

A. UML

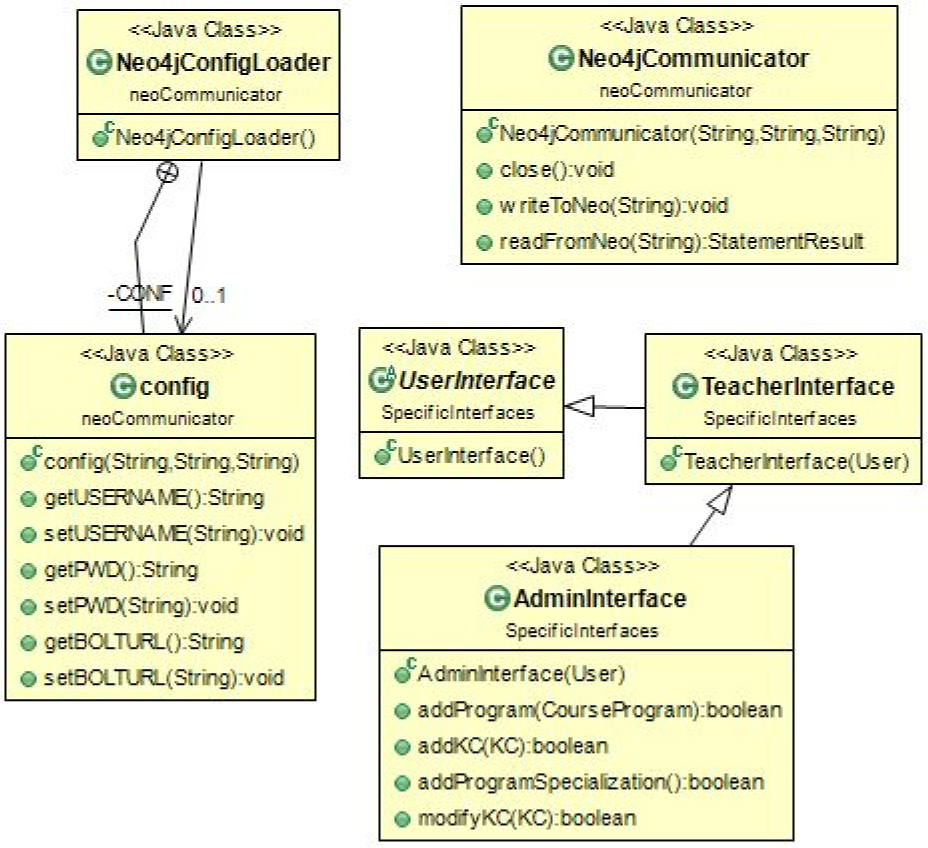
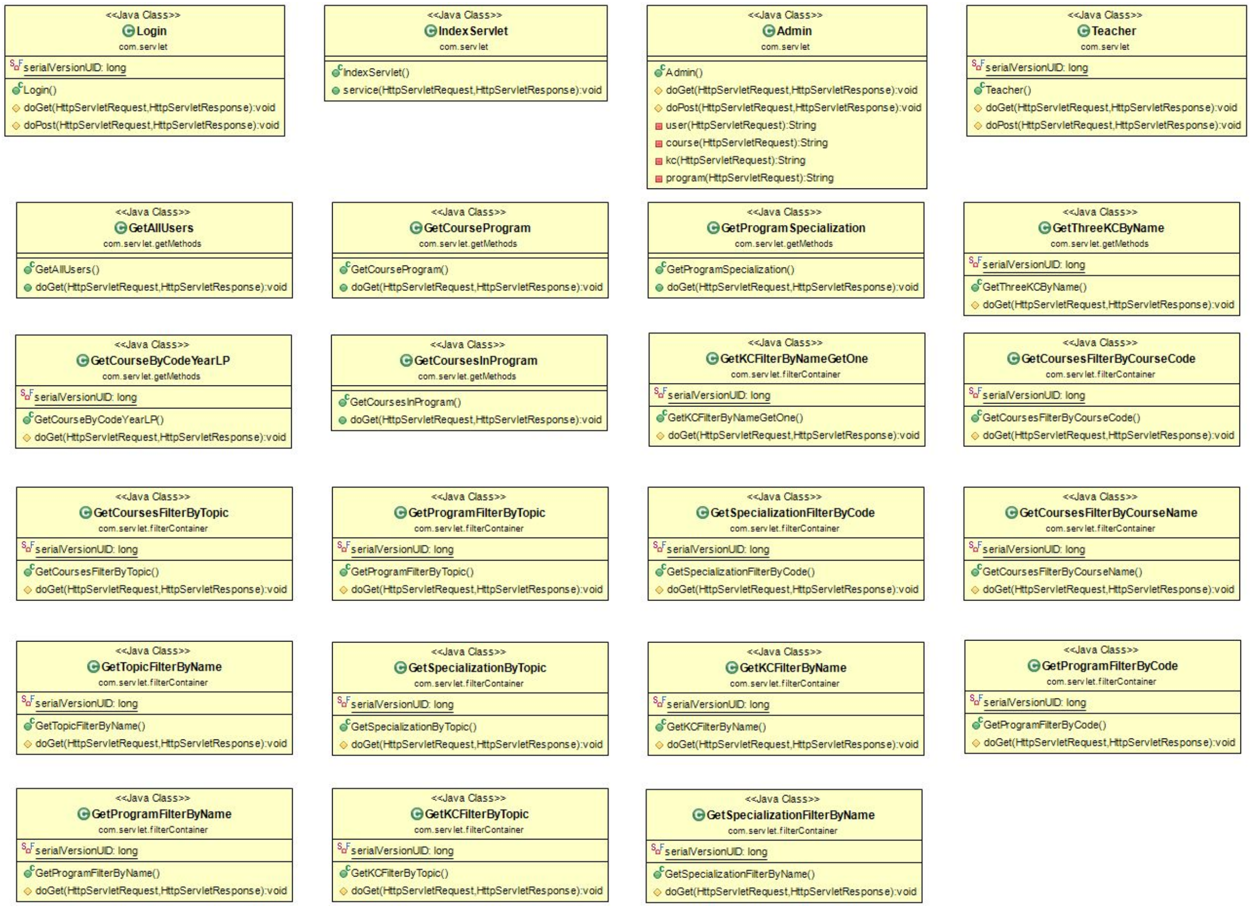


Figure 1: Java Specific Interfaces and Neo4J Communicator package.



<<Java Class>>
Login
com.servlet
serialVersionUID: long
Login()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
IndexServlet
com.servlet
IndexServlet()
service(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
Admin
com.servlet
Admin()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
user(HttpServletRequest):String
course(HttpServletRequest):String
kc(HttpServletRequest):String
program(HttpServletRequest):String
<<Java Class>>
Teacher
com.servlet
serialVersionUID: long
Teacher()
doGet(HttpServletRequest,HttpServletResponse):void
doPost(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetAllUsers
com.servlet.getMethods
GetAllUsers()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCourseProgram
com.servlet.getMethods
GetCourseProgram()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetProgramSpecialization
com.servlet.getMethods
GetProgramSpecialization()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetThreeKCByName
com.servlet.getMethods
serialVersionUID: long
GetThreeKCByName()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCourseByCodeYearLP
com.servlet.getMethods
serialVersionUID: long
GetCourseByCodeYearLP()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCoursesInProgram
com.servlet.getMethods
GetCoursesInProgram()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetKCFilterByNameGetOne
com.servlet.filterContainer
serialVersionUID: long
GetKCFilterByNameGetOne()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCoursesFilterByCourseCode
com.servlet.filterContainer
serialVersionUID: long
GetCoursesFilterByCourseCode()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCoursesFilterByTopic
com.servlet.filterContainer
serialVersionUID: long
GetCoursesFilterByTopic()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetProgramFilterByTopic
com.servlet.filterContainer
serialVersionUID: long
GetProgramFilterByTopic()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetSpecializationFilterByCode
com.servlet.filterContainer
serialVersionUID: long
GetSpecializationFilterByCode()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetCoursesFilterByCourseName
com.servlet.filterContainer
serialVersionUID: long
GetCoursesFilterByCourseName()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetTopicFilterByName
com.servlet.filterContainer
serialVersionUID: long
GetTopicFilterByName()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetSpecializationByTopic
com.servlet.filterContainer
serialVersionUID: long
GetSpecializationByTopic()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetKCFilterByName
com.servlet.filterContainer
serialVersionUID: long
GetKCFilterByName()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetProgramFilterByCode
com.servlet.filterContainer
serialVersionUID: long
GetProgramFilterByCode()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetProgramFilterByName
com.servlet.filterContainer
serialVersionUID: long
GetProgramFilterByName()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetKCFilterByTopic
com.servlet.filterContainer
serialVersionUID: long
GetKCFilterByTopic()
doGet(HttpServletRequest,HttpServletResponse):void
<<Java Class>>
GetSpecializationFilterByName
com.servlet.filterContainer
serialVersionUID: long
GetSpecializationFilterByName()
doGet(HttpServletRequest,HttpServletResponse):void

Figure 2: Java Servlet package.

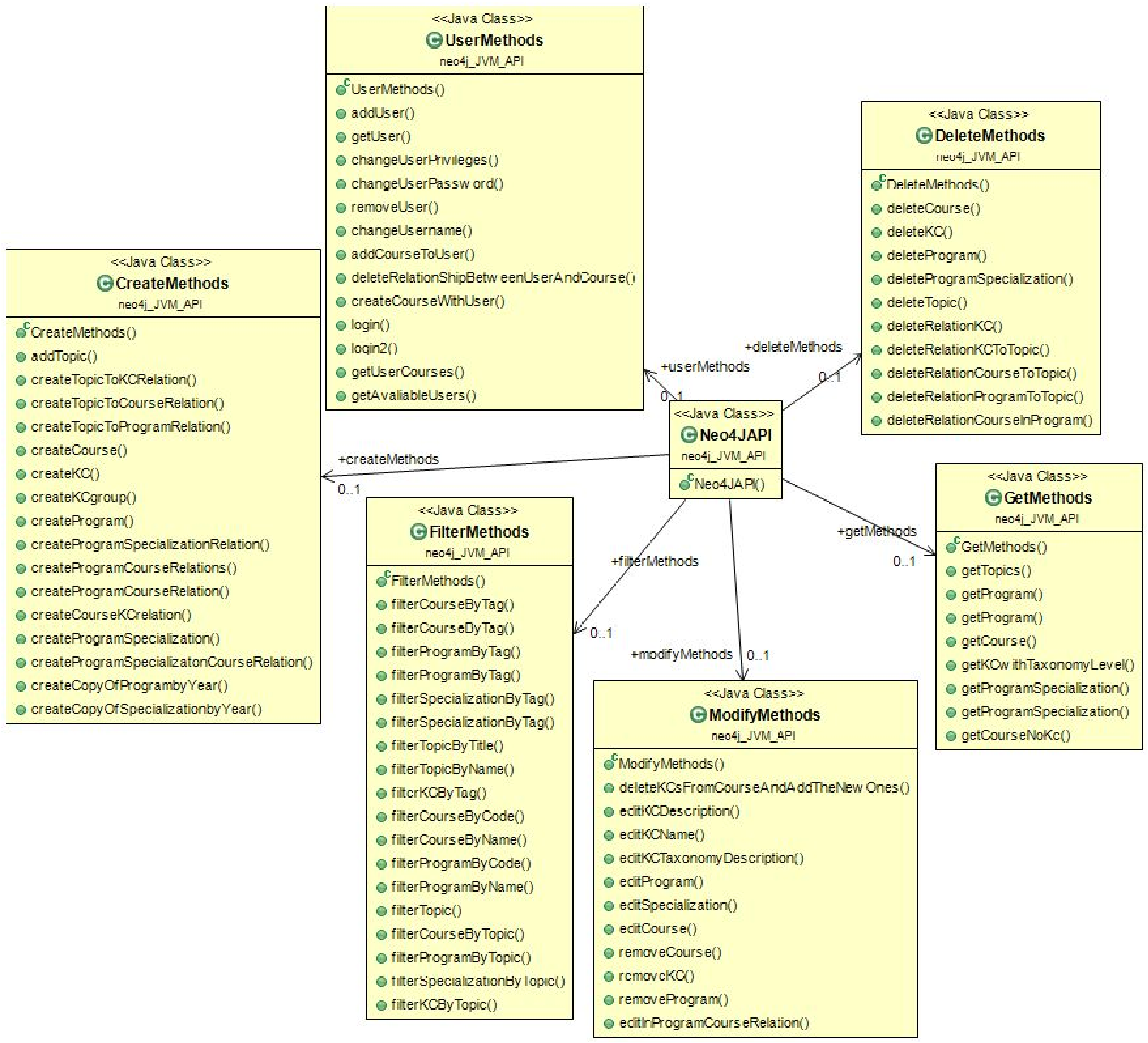


Figure 3: Java Neo4J package.

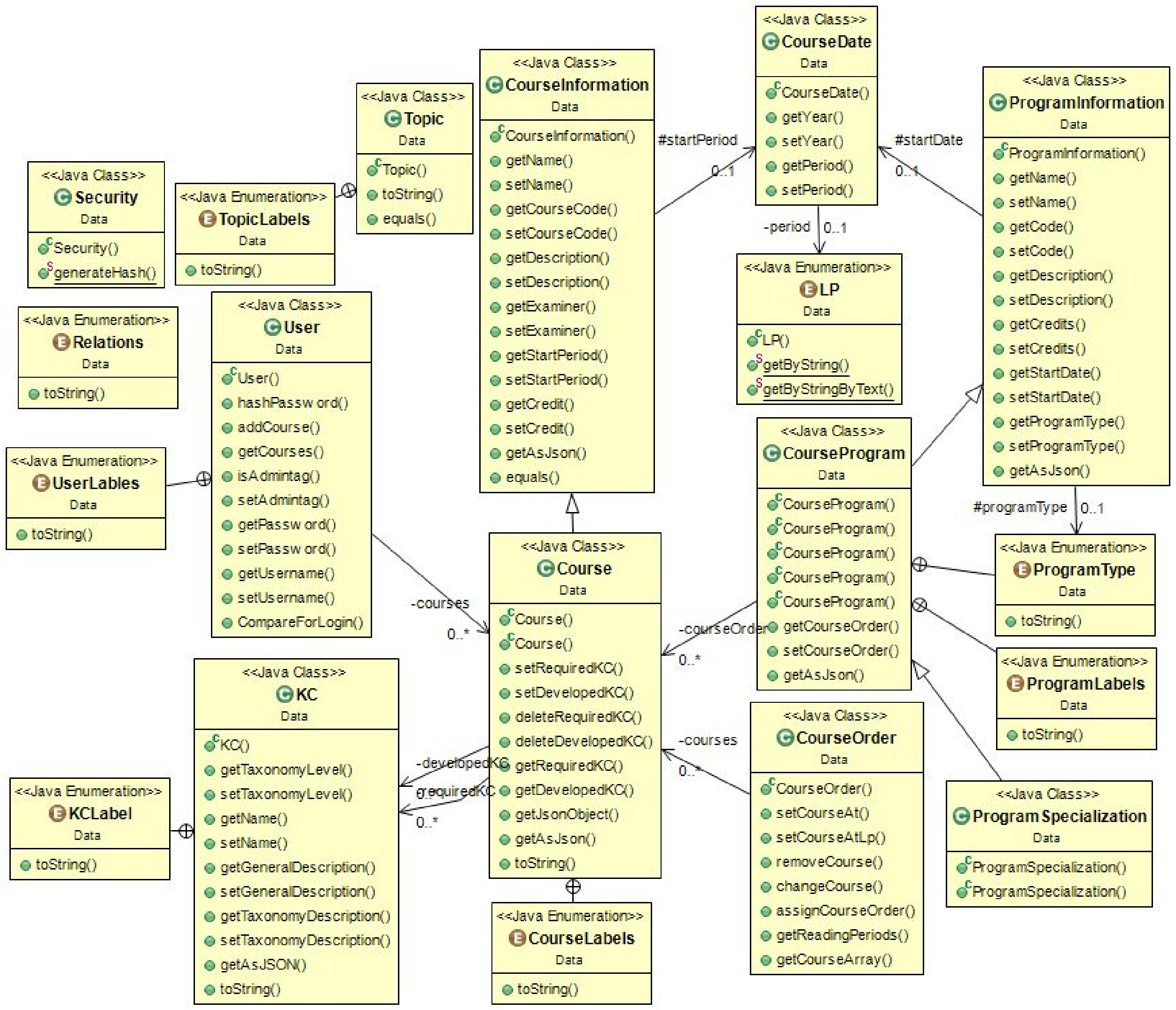
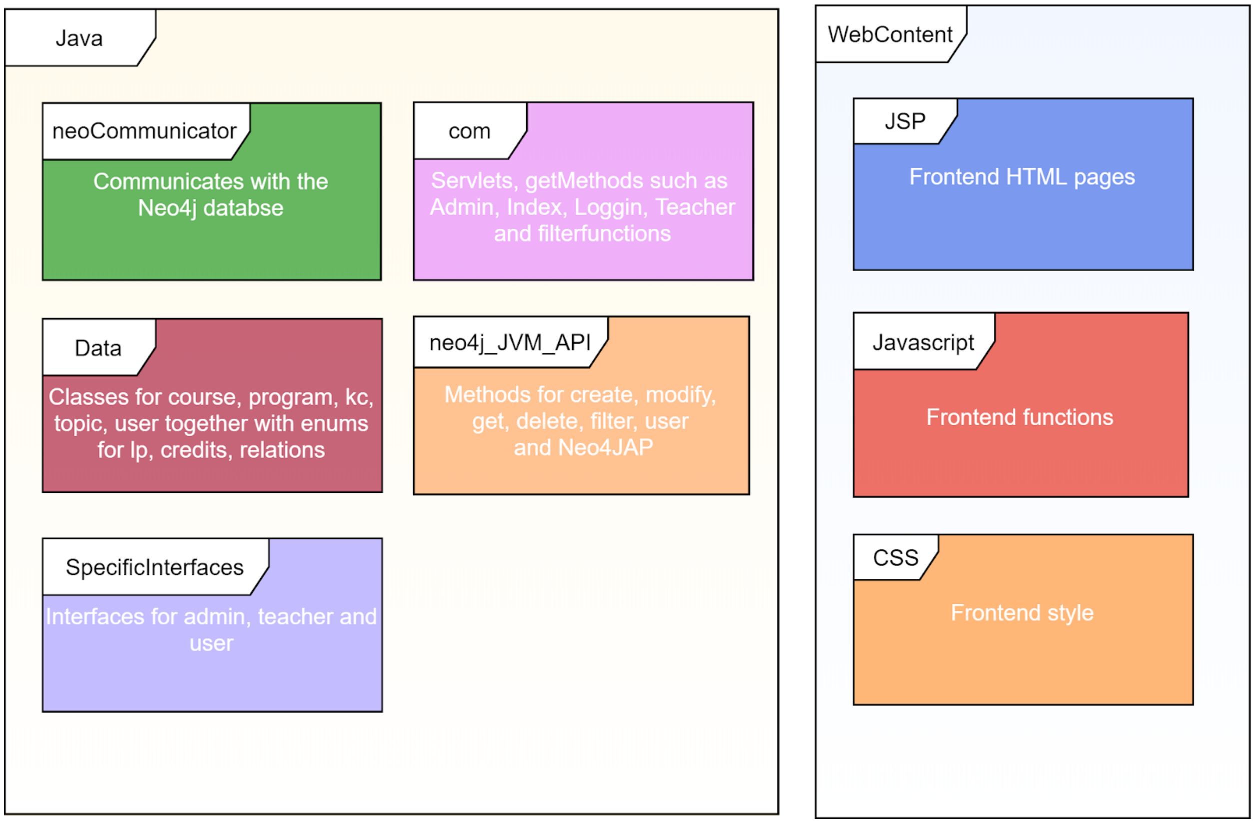


Figure 4: Java Data package.

B. Documentation for future

1. Introduction

This document describes how the project Study Planner is built and gives an introduction on how to continue working on the project. The code for the project can be found at github from the following link https://github.com/ojaohe-3/d0020e.



Java
neoCommunicator
Communicates with the Neo4j databse
com
Servlets, getMethods such as Admin, Index, Loggin, Teacher and filterfunctions
Data
Classes for course, program, kc, topic, user together with enums for lp, credits, relations
neo4j_JVM_API
Methods for create, modify, get, delete, filter, user and Neo4JAP
SpecificInterfaces
Interfaces for admin, teacher and user
WebContent
JSP
Frontend HTML pages
Javascript
Frontend functions
CSS
Frontend style

Figure 1: Module diagram to present packages in the system

Figure 1 describes the structure for the implemented system. The system is divided into two folders, Java that contains all the Java classes and WebContent that is for the HTML, JavaScript and CSS files.

In the Java folder there are five packages. First is the neoCommunicator packages are the files that communicate with the database. Between the neoCommunicator and the Data packages together with the com package is the neo4j_JVM_API. That package contains all methods that create queries to be sent to the database. The neo4j_JVM_API takes in data types that are defined in the package Data that contains all classes for the data types and enums, under section 3 are all data types described. The package com handles the communication between the java program and the web browser by converting the information to JSON.

This is where the folder WebContent comes in. The folder contains all JSP files building the web pages. The folder contains all JavaScript files handling all requests from the web page. The CSS folder contains all files for styling the web pages.

2. List of features to be improved

Due to lack of time and missing knowledge some assumptions and non optimal solutions were made. Below are descriptions on these and what should be done to optimize those.

- Selectable/optional courses
- Three taxonomy levels
- Error checks
- Admin page
- User friendly - More information
- Copy of program
- Course over more than one reading period
- Additional Security
- Hoverable KClines
- Alternative LP and credit systems

Selectable/optional courses
In order to map a program selectable/optional courses are mandatory. Due to lack of time this was left out in development. The problem with this function is that we need to draw out an empty course box, KC:s shall map to this box when a selected course is put in. The system should calculate if you can read the selected course from the developed KC:s you have acquired along the way. If the developed KC:s are not enough for the course an error message shall tell the user that you can not read the course because KC:s are missing. The system has not implemented selectable/optional courses nor checks for needed KC:s.

To implement optional courses (a course student can choose freely) a definition of it needs to be implemented in the html/javascript as an empty box and for the admin to enter which reading period and year the optional course is so it can be drawn in the graphical view. For selectable courses (course students can choose from a few selected choices) a new relation in the database could be implemented to keep track of what courses are selectable for which program. At the admin page a function to enter a list of the selected courses for specific reading period and year should be made together with a box in the graphical view where the selectable courses from a list could be chosen by the student. To make sure that the required KC:s for a course are developed checks in the javascript should be implemented.

Three taxonomy levels
Early in the project it was decided that every KC should come with three different taxonomy levels. This was one way to structure the system. A better solution would be to make it more dynamic, some KCs maybe have two levels and others have ten levels.

To implement changes to this first changes must be made to javascript in the admin page where the KCs are created and the method called in the API package. Later on more smaller changes need to be made in all places where a group of KCs is collected from the DB. Like when clicking on a KC in filter container and so on.

Error checks
As the system is now there are very few error checks, the system trusts that the user knows what it is doing which is not always the case. This problem is split into user, teacher and admin.

User: Right now a user can remove courses that are needed in the program and a user can add an infinite number of the same course. It is also possible for the user to add a course even if the required KC:s do not match.

The first check needs to see what KC:s are needed in the future and then put restrictions on that course if that course KC:s are dependent on another course. The infinite add program check can be put into the canvas and check if the course is already present in the canvas if a user wants to use it.

Teacher: If a teacher removes a developed KC that is a required KC in another course there should be an error message that indicates that students no longer learn what it needs to learn in that course.

This check should be implemented in the description for the teachers.

Admin: The admin has a lot of responsibilities since there are no error checks at all except the number of data the system wants and the “;” that is the divider between data input.

All these checks should be in the admin page, the admin handles a lot of information on the system and there are bound to be errors when entering new courses, KC:s and programs.

Admin page
The admin page would be prioritized to make it more user friendly. The implemented solution was the easiest and fastest to implement.

To make changes to the admin page it is easiest to replace all the prompt() function calls to a form in the javascript. All functionality is already implemented so major focus should only be to make it more user friendly.

User friendly - More information
There could be more time put into the front end and how the web page looks and interacts with the user. More could be displayed to give the user even better understanding and help in planning their studies.

In the KC information box there could be included a list of courses where it is developed. In the course information box all developed and required KC:s could be added in a list.

Copy of Program
To make the system easy to maintain copies of a program to the next year is needed.

Course over more than one reading period
As implemented now all courses are assumed to run in one period. This is a problem in the graphical view since there is now way to see courses that run at 100% or 25%. Many programs contain courses that stretch over multiple reading periods.

To implement this, courses need to know their tempo. Using that information the graphical view can arrange the courses in the correct way.

Additional Security
More pronounced and thorough test of the security aspect of the web page, as of now login credentials does not have a hidden token making it vulnerable for CSRF attacks to just name one known exploitable point of the program out of several.

Hoverable KC lines
Make Lines more clear to and from where a kc is requested from or developed from. With clear color and additional information at the mouse it would be possible to see what that kc maps to.

Alternative LP and credit systems
The system is made for a course order with 60 credits per year and 15 credits per reading period. For a university where this is not applicable a redefinition of reading periods is needed.

The optimal would be to come up with a definition that could be used for different types of course orders. If this is not possible seperate versions of the system could be implemented. If a new definition wants to be implemented the data type should be updated together with the neo4j API methods, servlet methods and in the JavaScript files where programs are requested.

Graphical View
Developed KCs will only be visible from the latest source where it was developed. If the same KC is developed in multiple courses and a course that requires that KC is clicked it will only show the latest source.
The graphical view needs to search through all the courses instead of stop searching when it finds the first one.

Data types
The definition of the data types are described in the list below. This to give a clear picture of which parts that can be reused and which need to be redefined. The data types also contain labels that are enums to be used in the neo API.

CourseInformation - Contains the minimum information required for searching in the database for courses and receives information. This includes name, course code, year, lp and description.

Course - Inherits CourseInformation but is also containing all the developed and required KCs for the course.

LP - Enum that represents all possible study periods. One year is divided into four reading periods so the LP class contains ONE, TWO, THREE, FOUR and ERROR.

CourseDate - Contains of a year and a LP. Represent the date when the course starts.

ProgramInformation - Contains the minimum information required for searching in the database for programs.

CourseProgram - Contains a code, name, description, start date, credits and a type. Code is the program code which all programs have, name is the name of the program, description is used to explain the program. Start date is when that program started, every year a new program is made in case new courses have been added to a program. Credits is how many points the program is worth. Type is to differentiate a program and a specialization, a specialization is a special program.

ProgramSpecialization - Extends course program but changes the programType to SPECIALIZATION to separate it from main programs in the database.

KC - Contains a name, general description together with a taxonomy level and a description for that taxonomy level.

Relations - Enums for all possible relations in the database. DEVELOPED, REQUIRED, BELONGS_TO, IN_PROGRAM, SPECIALIZATION, CAN_EDIT. The relations are used in the queries to the database. Developed and required are between KC:s and courses, Belongs_to between topics and other nodes, In_program between a program and courses, Specialization is between program and a specialization while Can_edit is between user and course.

Topic - Have a title. All courses, KC and programs have a topic.
User - Have a username, password, a boolean that is true if admin and false otherwise.

C. Blacklog

In this document we will list methods and concepts that we tried to implement or skipped. With reason of bad implementation, lack of time or some other reason. This document also describes the path to future development.

List of unimplemented features

- Userpage
- Method to optimize reading order
- TA & ILO
- Remodeling KC
- Save changes in canvas
- Course evaluation

Userpage
The idea was that students could log in and make a customized program with all the selectable and optional courses put into their reading order. This was put on hold because selectable and optional courses were not implemented. We also weighed the efforts versus the benefits from this function and it was decided that this is a feature that is handy but not essential for users to use the system. This concept should be implemented in the future.

A new type of user, student, could be implemented in the system. The students should be connected to the program they are reading so they can see it on the first page. From this they should be able to choose optional, selectable and extra courses to see the graphical view. The customized course order could be saved in the user.

Method to optimize reading order
This is the most complicated method of all the methods in the system. The essence of it is to scan through a program and then try all the courses and make the most optimized reading order for students. This would be a very powerful tool to change current programs and specializations but also when new ones are made. In order for this to work we need to know exactly how the system works and needs to make a tool to analyze all the course nodes and KC:s nodes which we don’t have the proper skills for, or time.

This could be implemented with a powerful algorithm that checks all the required and developed KC:s. The algorithm structures the program with all the courses without required KC:s to the beginning of the program and then maps the developed KC:s with courses that require those KC:s. Other checks could be made but this is our thoughts about an optimized program.

TA & ILO
Teaching activity and Intended learning outcome. Teaching activity is the form of teaching. The knowledge components could be obtained with different teaching activities like presentations, seminarie, labs, exams, classes and so on. Intended learning outcome describes all skills the course should provide. ILOs can be for example problem solving, teamwork, and analytical thinking. Therefor different KC:s can lead to the same ILO so the same course can have different KC:s and TA:s under the same ILO. Our system has not implemented TA:s nor ILO:s.

To implement TA:s one could add a new type of node and java-class named TA together with a new relation to connect KC to TA. ILO could be implemented in the same way with a new type of node, a java-class and a new relation to connect KC:s and TA:s to ILO:s.

Remodeling KC
KC is a definition that the group and Jan Van Deventer came up with, the definition may be not correct since a very limited time has been set to research this term. As it is right now it can be used to show the user/teacher what a student learned from a course. To be able to optimize the course order of a program this term may need to be remodeled or redefined.

In order to do this more research needs to be done, how do you measure learned skills? Is it reasonable to do that? More insight into how programs and courses are created is needed. It may be off to a great start but more effort in this matter is needed.

Save changes in canvas
To be able to make a customizable course order there needs to be some function to save the current canvas. This can be used if you want to browse a course order later or on another computer.

For this to work, more functions in both the frontend and the backend are needed. Should the canvas be saved to a user profile or is it saved locally? If it is saved in a profile there needs to be a place in the database where that course order is saved.

Course evaluation
When a course window is opened it shows description of the course and required KC:s and developed KC:s. A link to the course evaluation could be shown in this window as well so students can easily access it. This could lead to more students filing the course evaluation but it should also lead to higher quality.

This would be simple to implement as a system that handle course evaluation already exist on Luleå tekniska universitet (LTU). The problem right now is that neither of the group members have access to this system. There may need some sort of login so that only the students who have taken a certain course can evaluate it.