Code Review for Novice Programmers

N. Keijzer, J.J. Korpershoek, T. Leemreize, J. Prins

April 19, 2018

Contents

Chapter 1

Introduction

For the design project of the bachelor programme Computer Science at the University of Twente, students are asked to develop and deliver a production-ready system. For this project ten weeks are available to the students. This report describes the development for the assignment “code review for novice programmers”.

1.1 Problem Statement

In the first module of the bachelor programme Creative Technology the students are introduced to programming. Currently the students' only way to ask for help is during the assigned hours or outside of the assigned hours on Slack or Blackboard from either the student assistants or the professor. The main issue with Slack or blackboard is that there is no real way to discuss your code aside from posting a screenshot or text copy of the code with another separate message describing the issue or question. This makes it hard to clearly indicate what you are talking about as it is not possible to attach comments to specific lines of the code.

For novice programmers receiving feedback line by line is more useful than receiving feedback over an entire file. This feature is lacking in the current method of giving feedback.

Because of the problems mentioned above, the client is in need of a system in which students can receive line-based feedback on their programs.

1.2 Stakeholders

This section will list the stakeholders for this project. These people were interviewed during the requirements engineering phase to find out what the current and future use cases are. The requirements for this project are mostly the result of the meetings with A. Fehnker. The wishes of M. Huisman and L. Ferreira Pires are either the same as those of A. Fehnker or are out of the scope of the project.

The visions of M. Huisman and L. Ferreira Pires for the project are:

- Automated grading of projects
- Automated feedback on code
- Fast manual grading of a lot of projects
- Faster sign off during tutorials

The primary stakeholder and client is:

A. Fehnker
University of Twente, Faculty EEMCS
Building Zilverling, room 3120
PO Box 217
7500 AE Enschede
The Netherlands

The secondary stakeholders of this project are:

M. Huisman
University of Twente, Faculty EEMCS

Building Zilverling, room 3039
P.O. Box 217
7500 AE Enschede
The Netherlands

L. Ferreira Pires
University of Twente, Faculty EEMCS
Building Zilverling, room 4098
PO Box 217
7500 AE Enschede
the Netherlands

1.3 Report Outline

In chapter 2 the possible pitfalls of the team are laid out together with the right methods to resolve these pitfalls. Next in chapter 3 the requirements engineering phase is documented with all the decisions made along the way. In chapter 4 the global and detailed design are presented alongside with the justification for the choices made. Chapter 5 contains a list of problems encountered during the implementation phase alongside with the solutions found for these problems. Chapter 6 presents all the different tests together with their results. Chapter 7 encompasses the different tasks during the project. Finally, in chapter 8 the report is concluded with the results of the project, final thoughts about the project dynamics and a list of unmet, non-critical requirements which could be added in the future.

Chapter 2

Risk Analysis

In this chapter the risk analysis of the project will be discussed. The risk analysis tries to map out the possible pitfalls of the project group with solutions for these pitfalls. The risk analysis is formatted using the Project Management Body of Knowledge (PMBOK) guidelines[1]. Table 2.1 is the result of the risk analysis using the PMBOK guidelines. This table will be used to help come to decisions throughout the project when unexpected events happen.

Problem	**Precautions**	**Impact**	**Solution**
A member may unexpectedly have to leave the group	Develop a project plan where the basic system can be implemented with just 75% of the workforce, define optional extensions that can be added when there is time left after the basic implementation. Make a clearly defined plan of the tasks of every member such that it is easy to know what the tasks of this member were.	5/5	Distribute the work equally over the remaining students such that the increased workload is kept to a minimum.
The laptop of a member may break	Make sure every member makes backups of their work after every meaningful change and it is easy to migrate the work to a new environment.	2/5	Force the member to get a replacement device, otherwise assume the member has to leave the group
A member may forget to bring his laptop	Set up a daily reminder for this	1/5	Send him back home to pick up his laptop and make him work overtime
We might underestimate the amount of work needed to deliver the project	Plan for this, set out to have a basic implementation that should be completely finished within 7 weeks and then define optional extensions that can be done if there is time left.	4/5	Worst case scenario only finish the basic implementation. Work more to finish the basic implementation in time if necessary.

Figure 2.1: Risk Analysis

Chapter 3

Requirements Engineering Phase

In this chapter the requirements engineering phase will be discussed. First off, the methodology of the project for the requirements phase will be covered. Next, the requirements phase will be laid out using the MoSCoW method [2]. Finally, useful scenarios will be presented to give an outline of some use cases of the project.

3.1 Methodology

The purpose of the requirements engineering phase is to find out what the client wants, who the stakeholders are, what the stakeholders want, and to extract requirements from that information.
The requirements engineering phase is divided in three steps. The first step is requirements inception, in this step meetings with the client and stakeholders are held to find out what they want. The second step is requirements analysis, in this step requirements are extracted from those meetings. Furthermore, extra requirements that are necessary for the system to function are also added. In the third and final step the requirements are listed as user stories.

3.2 Requirements

The MoSCoW[2] method is used to describe the requirements for the system that will be delivered. All user stories listed under must will be implemented as basic functionality of the system. The base system will only fulfill these requirements. All user stories listed as should will be implemented if time allows it. All user stories listed under could will be implemented if there is time left after implementing the must and should requirements. Realistically the would user stories will not be implemented in the given time frame. For each of the listed categories testing and documentation will be done, and thus testing and documenting features from must requirements has priority over starting with a should requirement. Likewise, testing should requirements has priority over starting with could requirements.

3.2.1 Definitions

Term	Definition
Module	A structure in which a number of people and courses can be added
Course	A structure in which a number of exercises can be added
Exercise	A structure in which projects can be started
Project	A number of code documents that belong together, which can correspond to the work of a project group in the tutorials
User	Anyone using the system, i.e. student, teaching assistant, and lecturer
Student	A student, has the least number of rights within a group
Reviewer	Someone who reviews the code in a course, i.e. teaching assistant or lecturer
Moderator	Someone who can manage everything in a course, i.e. a lecturer
Administrator	Someone who can create new modules
Comment	A remark on a piece of code or another comment
Thread	A comment and the responses to that comment

3.2.2 User Stories

Must

- As a user I must be able to log in and out of the system

- As an administrator I must be able to create a module
- As a moderator I must be able to add a course to a module
- As a moderator I must be able to add users to a module
- As a moderator I must be able to add exercises to a course
- As a user I must be able to create a project in an exercise of a course of which I am a member
- As a user I must be able to upload code to a project of which I am a member
- As a user I must be able to create a comment on a project on which I have that right
- As a user I must be able to view a personal page with comments that are interesting to me
- As a reviewer I must be able to see projects in a module of which I am member
- As a reviewer I must be able to comment on a project in a module of which I am a member
- As a moderator I must be able to assign different rights to user groups within a module, so that they can have limited or extended rights
- As a moderator I must be able to assign different rights to user groups within an exercise, so that they can have limited or extended rights
- As a comment I must be able to be attached to a line of code
- As a comment I must be able to be attached to another comment
- As a system I must be able to show code with syntax highlighting and line numbers

Should

- As a user I should be able to close a comment thread I started
- As a moderator I should be able to add other users to a project
- As a user I should be able to remove a user from a project I created
- As a user I should be able to remove a comment that I made
- As a user I should be able to remove a project I created.
- A user should be able to comment on the project of another user if I have that right
- As a reviewer I should be able to attach a tag to a comment
- As a reviewer I should be able to close any comment thread
- As a moderator I should be able to control the visibility level of comments of different user groups (i.e. who can see what)
- As a moderator I should be able to control the visibility level of projects (i.e. who can see what)

- As a moderator I should be able to add users to any project in a course
- As a moderator I should be able to remove a user from any project in a course
- As a moderator I should be able to remove any project in a course
- As a moderator I should be able to remove any comment in a course
- As a moderator I should be able to choose a template for user rights in an exercise
- As a moderator I should be able to create a new template for user rights
- As a moderator I should be able to make the comments of different user groups invisible until I approve them
- As a comment I should be unapproved if my author has limited permissions to post comments
- As a moderator I should be able to approve unapproved comments
- As a moderator I should be able to remove unapproved comment

Could

- As a moderator I could be able to make a comment sticky to make it appear at the top of a thread
- As a system I could be able to receive comments generated by a tool (i.e. checkstyle, etc.)
- As a user I could be able to upvote/downvote comments
- As a user I could be able to connect my project to a git repository to receive the code from there
- As a moderator I could be able to add a deadline to an exercise
- As a moderator I could be able to add a final verdict to a project
- As a user I could be able to view a dashboard for a project

Would

- An external tool would be able to act as a reviewer

3.3 Scenarios

The functionality of the front end is described with the use of scenarios. Scenarios are short stories that describe a specific action that the user can do with CoDR.

Personas [3] have not been used as it was determined that the role of a user is of greater importance to the project than the personality of the user. However, actors in the scenarios have been given a name for the sake of readability.

3.3.1 Requesting Feedback on a Piece of Code

Alice is working on an exercise during a tutorial. She has made an initial solution but it is not working. This is because she gets an error on line 7 that she does not understand. She goes to the website of CoDR and logs in using her University of Twente account. She clicks through the module, course and exercise that she is currently working on. She then creates a project and uploads the file she has just made. Once the file is uploaded, she selects line 7 and places a comment explaining what she does not understand and which error she gets. After a while Bob (a teaching assistant) replies to her comment with an explanation of what that line of code does and what goes wrong in her implementation. Alice adapts her code and replies to the teaching assistant that she is thankful for Bob's help.

3.3.2 Peer Review

During a tutorial session the teacher asks students to give feedback on each others projects. Alice is supposed to give feedback on Charlie's work. She first uploads code to her own project of the current exercise so that Charlie can give feedback on her code. She then browses to Charlie's project and opens the latest revision of his file. There she reads through the code and places comments on lines that she considers incorrect or has improvements on. After a while Charlie reacts to some of her comments and asks for clarification as to why Alice considers it wrong.

3.3.3 Creating an Exercise

Dave is a teacher and wants to create a new exercise for the students. He goes to the website of CoDR and logs in with his University of Twente account. After logging in he browses to the current module and course and creates a new exercise. He decides that for this exercise users should not be able to see each others projects, therefore he changes the rights that students have on the projects in this exercise so that users can only see their own projects. He then adds a description and name to the exercise and updates it on the website.

Chapter 4

Design

In this chapter the design choices behind the implementation of CoDR will be discussed. The choices made during the design phase will impact the look and feel of the application. The discussion of the design of CoDR has been split up in two sections: global design and detailed design. In the global design section of this chapter the design choices behind the base framework of CoDR will be justified. In the detailed design section design choices about individual components of the system will be justified.

4.1 Global design

4.1.1 Framework

Node.js with the Express web framework was chosen as the base framework for CoDR. This choice was made as it is easy to quickly develop a web application in Express and JavaScript. This is a preferable feature for a project with a limited timespan. Another viable alternative that was considered for Node.js and Express was the Python framework Django. Python is, similarly to JavaScript, easy to learn and quick to develop in. Django was however not chosen to be the framework to develop in as research into the frameworks showed that Node.js performs and scales better than Django[4][5]. Another benefit of Node.js is that Node.js programs are written in JavaScript, which means that both the front end and back end of the application can be written in the same language. This allows the developers to switch between the front end and back end parts easily.

Front end Frameworks

Pug Pug was used as an alternative to simple HTML pages. Because Pug allows us to insert variables directly into the HTML pages.

CSS Bootstrap was used as the CSS framework, because this is commonly used and some measure of experience was already available to some members of the project group.

JavaScript React[6] was used to dynamically update the content of a website so that the user receives a new comment on the current page as fast as possible.

webpack[7] In order to be able to use SASS[8] for useful features on top of plain CSS, ES6[9] and React[6] JSX[10] a JavaScript build chain was needed. One of the most used packages for building and transpiling JavaScript is webpack. In this project all ES6, JSX and SASS is compiled to JavaScript for different web browsers by webpack in combination with Babel and a SASS-loader for SASS compilation.

4.1.2 Front and Back end

During the early stages of the design phase it was decided to split the application into two parts, namely the front end and the back end parts. By splitting up these parts it would be possible to use different programming languages for the front end and back end.

By having a separate front end it is possible to replace the front end in the future by a more polished front end, may the product owner decide to do so. In addition to allowing for the front end to be replaced it is also possible to have multiple front ends for the application by splitting it from the back end. This would allow for a desktop application or mobile application to be developed, which would use the same back end as the website.

4.1.3 User Authorization

All user authorization is done as close as possible to the source of data in the system, which is the database. It was chosen to embed this feature into the database to ensure that authorization is always enforced, such that users are never allowed to view data they are not authorized to view. In a system which might be used for submitting exams, security is of utmost importance so a lot of thought has gone into this part to ensure that user authorization is always enforced. The security has been implemented using a rights system, which will be explained in detail in section 4.2.2.

4.2 Detailed Design

4.2.1 Database

In appendix B the class diagram of the database is shown. This diagram describes the tables and columns that are needed in the database. The arrows indicate references to other columns/tables. An explanation from some columns is given below.

Comments

The parent_id, file_id and line_range are noted as optional. This means in this case that a comment can either have a parent_id, which means that it is a reaction on another comment, or it can have a file_id and a line_range, in which case it is a comment on a part of a file. The deleted field indicates that the comment has been deleted, this is so that the comment can later also be undeleted.

line_range The format of this parameter is a set of line numbers separated by commas. An example: 1,2,3,5,8 to indicate lines 1, 2, 3, 5 and 8.

Templates_content

The columns project_right and comment_right indicate which permissions the specified group has for the respective parts of the application. The rights fields can be interpreted as a bit mask, the interpretation of this bit mask is detailed in section 4.2.2.

Functions

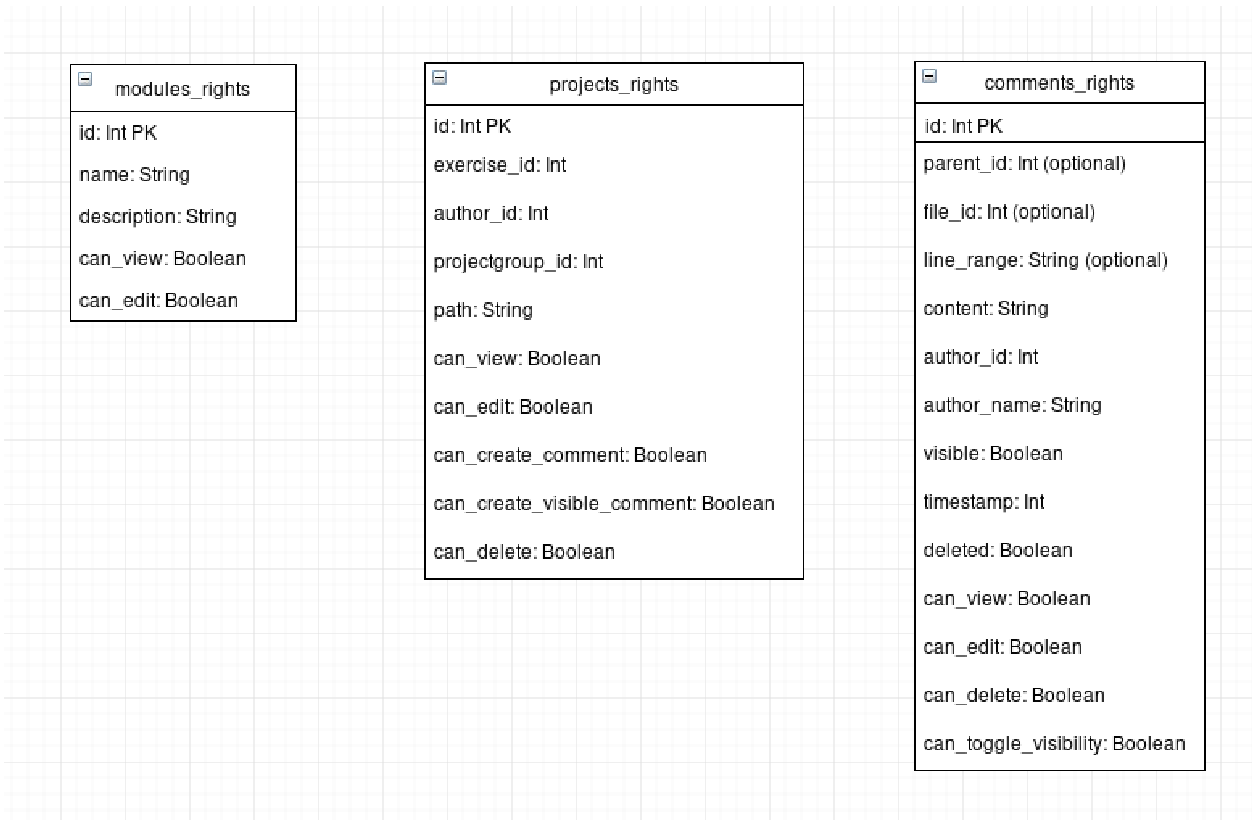


Figure 4.1: Result templates of functions that can be used to get a project or comment with the rights the current user has on that object.

In figure 4.1 three database functions that return tables are shown that can be used to get a comment or project including the rights that the current user has on it. A database function is a function that returns all data from multiple tables as a single table containing all data. In figure 4.1 this means that for example the function modules_rights returns five fields containing data about a module and the rights a user has for the given module. This is useful because then the database can do most of the rights checking, and the back end only has to check whether the rights fields are true or false.

Users

user_level indicates the global level of the user. The user is a global administrator if the level is 0.

4.2.2 Back End

For the design of the back end multiple choices had to be made. In the introduction of this chapter it was mentioned that the front- and back end are split up. As a result there is a lot of freedom in how the back end is implemented as it just has to act as an API. The choices made regarding how the back end is implemented can be found in this chapter.

API Design

The choice was made to use Swagger as a means to define an API specification for the back end. By making use of Swagger as an API specification tool it made it easy to create a nice and clear API specification that can be shown as an HTML page. The API specification can be found in appendix C. As an extension of the choice to use Swagger for the API specification, a Node.js module (swagger-node [11]) was found that could convert this specification into paths for the back end. This made it possible to automatically resolve API paths to Node.js code which would in turn process the request and send back a response.

The API itself was designed by first making a list of all use cases that were necessary to use the system from a user's perspective. These use cases were then split up into requests that would need to be made for each specific use case. These requests were then used to define each API endpoint with input and output specifications. REST was used for the API specification as using REST to specify an API can reduce the errors and improve success rate and satisfaction of API client developers[12].

Picking a Database

To support this system a database has to be used in order to efficiently save and manipulate data. To communicate with this database from the Node.js code a framework has to be used. The choice was made to use pg[13] as a means to communicate with the database as this was the largest database communication framework for Node.js.

For the database type there were several options. The first option would be an ORM[14]. This is effectively an interface that manages the database through objects. These objects can then be created and used as representations of database columns. This makes it very easy to create database entries and update/delete/fetch their data. The downside of this approach is that it is not as flexible as managing ones own database, which is the main reason that the choice was made to not use an ORM. The use of an ORM made it impossible to use the designed rights system because it works on top of the database itself.

Due to the fact that an ORM layer would make it very difficult to implement the desired database system it was better to use a database without an extra ORM layer. Therefore, a choice had to be made regarding which database should be used. There were multiple database options to choose from. For the purposes of this project and the way the database needs to be set up the options have been limited to these relational databases:

- PostgreSQL[15]
- MySQL[16]
- SQLite[17]

In the end the choice was made to use PostgreSQL due to the fact that it is open source, multiple project members have previously worked with it, has extensive support for JSON serialization and the residential expert (Maurice van Keulen) at the University of Twente recommends PostgreSQL. The design of the database will be explained further in the database design section 4.2.1.

User Authentication

For logging in to the system it was preferable to use existing login credentials of the University of Twente as it makes the login process of the application easy to use for the target group. To make this work there were two options, log in using the University of Twente account of the student or link the application to Canvas [18] and let students access the application through Canvas. After some research and contact with students that linked their application to Canvas in their design project from 2017 this was deemed to be a hard approach as it was very complicated to get developer access to Canvas. Therefore, the choice was made to use the University of Twente login system which makes use of SAML[19]. To make use of SAML authentication the passport-saml framework[20] was used in combination with passport[21]. This framework was chosen because it was the most popular SAML framework for Node. s.

Rights System

The rights system is used to check if the current user may view a specific object or execute a specific action. The requirement from the client was that different exercises should have different right levels, in order to ensure that some exercises can be peer-reviewed and that for other exercises users cannot see each other's projects. Rights can be defined on the exercise level, so that the user can use a rights template on an exercise to specify which specific user groups are allowed to do in a specific project. As the templates are defined on an exercise level, anything above the exercise level, such as modules or courses, do not adhere to this template but follow a different, simpler, rights system. It was decided to specify the rights system with this level of granularity to allow complete customization of the rights system. This level of customization allows the system to be used for many setups, such as peer review sessions and exam submissions.

The decision was made to define the rights system in the database as opposed to a common practice to define rights in the code of the application, since this way the rights can be updated while the program is running and the rights system is included if a backup of the database is made.

Global Definition

Templates Templates can be created and added to an exercise and thus specify the rights of users inside all projects that belong to that exercise. The benefit of using templates is that they can be shared between different exercises (even across modules) so that the rights do not always have to defined again for each exercise.

Modules and Courses Within modules and courses the rights system is less complicated: moderators of the module (that the course belongs to) have all rights, and so do global administrators. The rest of the users do not have any rights to modify modules or courses.

Creating Modules Only a global administrator has rights to create modules.

Definitions in the Database The bit layout of this mask is as follows:

Project

Part of the bits	Name of that part	Applies if
First 5	Member	the user is a member of a project
Second 5	Other	the user is not a member of a project

Where the bits in each part correspond to the following rights:

Right name	Explanation
View	View the project
Edit files	Edit, create files in the project
Create comment	Create a comment in the project
Create visible comment	Create visible comments in a project. Requires create comment.
Delete	Delete the project

Comment

Part of the bits	Name of that part	Applies if
First 4	Owner	the user is the author of a comment
Second 4	Member	the user is a member of the project the comment belongs to
Third 4	Other	the user is not the creator of the comment

Where the bits in each part correspond to the following rights:

Right name	Explanation
View	View the comment
Edit	Edit the comment
Delete	Delete the comment
Toggle visibility	Toggle the visibility of the comment. Requires Edit rights.

4.2.3 Front End

For the front end design three methods of specifying the layout are used. Firstly, the general flow of the front end is discussed. Secondly, for all main functions of the application a description of how that function is accessed is shown. Finally, mock-ups were made to show what the front end will look like in some of the views.

Flow of Content Front end

To fully understand the front end design a general knowledge of the client-sided data flow is needed. The data flow will be clarified using an example.

1. The user requests an HTML page to the web server
2. The web server converts the Pug files to HTML files
3. The client loads the HTML page, therefore creating another request to the web server for the necessary JavaScript files
4. The JavaScript files utilize the React framework to create a request to the API to load the data from the back end
5. The React framework processes the data from the API and renders the processed data to the user

Descriptions of Different Functions

In this section different functions of CoDR are described in a format that loosely resembles a scenario. This format is chosen because it accurately describes what different buttons do and how the user should execute a certain task.

Entering the Website Required rights: None.

1. Go to the main page
2. Redirected to University of Twente log in
3. After logging in redirected to main page

Creating a Module Required rights: Administrator.

1. Enter the website
2. Click the plus button on the right
3. Enter a name and description
4. Click the save button

Adding Users to a Module Required rights: Administrator

1. On the main (or modules) page click on the desired module
2. Click in the sidebar on settings
3. Click the plus on the right side
4. Select a user and a group for that user (this group will define the rights that the user will have in this module)

Creating a Project Group Required rights: Member of module.

1. On the main (or modules) page click on the desired module
2. Click in the sidebar on Project Groups
3. Create a new project group

Adding a User to a Project Group Required rights: Member of a project group.

1. On the main (or modules) page click on the desired module
2. Click in the sidebar on Project Groups
3. Select a user in the selector field in the table
4. Click the plus
5. Click the save button

Creating a Course Required rights: Administrator or module Moderator.

1. On the main page of a module click the plus button on the right
2. Enter a name and description
3. Click the save button

Creating an Exercise Required rights: Administrator or module Moderator.

1. In the main page of a course click the plus button on the right
2. Enter a name and description and select a rights template

Editing an Exercise Required rights: Administrator or module Moderator.

1. In the main page of an exercise click in the sidebar on settings
2. Click the edit button
3. Enter desired name or description
4. Select the desired rights template
5. Click the save button

Creating a Project Required rights: Member of a module.

1. In the main page of an exercise click the plus button on the right
2. Select a project group for that project
3. Click the save button

Uploading a File Required rights: Can edit a project.

1. In the main project page, click on the upload button in the sidebar
2. Select a file or drag a file into the drop zone
3. Click the upload button

Creating a Comment Required rights: Can create a comment in a project.

1. In the main project view click on a file
2. Select the lines you want to comment on using the check boxes on the left
3. Click the Add Comment button
4. Enter comment text
5. Click the save button

View User Dashboard Required rights: None.

1. Click the user name in the top right

Replying to a Comment Required rights: Can create a comment in a project.

1. In the main project view click on a file
2. Click the reply icon in a comment
3. Write a reply
4. Click the save button

Editing a Template Required rights: Administrator.

1. Click the Admin tab in the navigation bar
2. Click the rights template in the left sidebar
3. Select a template to edit
4. Click the edit button
5. Change the desired options
6. Click the save button to save the changes

Deleting an Object Required rights: Can delete the specified object.

1. Click the delete button

Mock-ups

These mock-ups were made early in the design to specify how the front end would look in different pages. Some details in these mock-ups have been changed later in the application because of implementation advantages or because the functionality they provided was not realized. For example, the notifications system was not implemented so the buttons related to the notification system have not been included in the final system. The same goes for the search bar. The drop-down menu for the profile has been moved to the main navigation bar because there was enough room there. It was also decided that the Log out button in the sidebar was not necessary.

Courses See figure A
These mock-ups were made during the requirements-engineering phase. Later in this phase it was decided to add another hierarchy layer above courses, namely modules. Therefore, this courses page is more a mock-up for what is currently the module page.

Course See figure A
In this view the Members and Rights button has been changed to a Settings button where the exercise can be changed and another rights template can be used.

File See figure A
The main structure of the file page is the same, however, in implementation some difficulties were phased with creating this exact layout. See section 5.2 for a more detailed description of these problems.

Profile See figure A
The functionality in the sidebar has been cut down, because those items were not must requirements. The rest of the layout is similar.

Controllers

This section will describe how the front end is set up. It will list the views that are used, what those views will show, which URLs those views have and which controllers handle them.

Before accessing the application, the user will have to log in using the external University of Twente login service.

Modules Shows all modules of which the current user is a member.
For administrator, shows all modules.

URL /

Controller modules

Module Shows all courses within the selected module. It also has a settings page where users in the module can be managed and a page where the user can manage their project groups for the module.

URL /modules/{module_id}

Controller modules

Course Shows all exercises that belong to this course.

URL /modules/{module_id}/courses/{course_id}

Controller courses

Exercise Shows all projects within the selected exercise that the current user has access to. The user also has the option to create a new project here. This page also contains a settings page where the exercise can be edited.

URL /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}

Controller exercises

Project Shows the project overview with in the sidebar the files that belong to the project and in the main part the recent comments on that project.

URL /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}

Controller projects

File Shows the files in the sidebar that belong to the project and in the main part the currently open file with comments.

URL /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files/{file_id}

Controller projects

Admin Shows the administration page. On this page templates and right groups can be managed.

URL /admin

Controller admin

Profile Page Shows the profile page of the selected user. Here the relevant recent comments for that user are shown.

URL /profile/{user_id}

Controller profile

Chapter 5

Implementation Phase

In this chapter the methods used to aid the development during the implementation phase as well as any problems and solutions that were encountered will be discussed.

5.1 Methodology

During the development phase the development team was split in two, allowing the workload to be distributed evenly. When working on a project in separate teams communication is key. Bad communication could lead to bugs in the code or project members doing the same task twice. In order to prevent this from happening precautionary measures were made.

5.1.1 Tracking Issues

For any issues that occurred during the development of the application issues on the GitHub[22] repository were made. By having a list of all unsolved issues on GitHub a clear overview is made of all things which have to be fixed, while also tracking the progress of the fix for the issue. GitHub also allows issues to be marked with labels, allowing the submitter of the issue to mention the importance and the related software component of the issue. In addition to creating issues on GitHub, issues were also shared verbally with the development team to ensure everyone was up to date with the latest state of the application.

5.1.2 Source Code Version Control

Aside from submitting issues to GitHub the code for the application was also hosted on GitHub, which uses Git[23] as version control system. By using GitHub the code base of the application is hosted in a central place, allowing all members of the development team to access it from any location or device. Git allows users to create separate branches for different features, enabling developers to work seamlessly on different features at the same time. These branches can then later on be merged into the master branch after having been tested, ensuring the master branch only contains working code. As the development team had been split up in two parts using version control greatly aided the development phase. Issues that could have otherwise been encountered due to people working on the same file were avoided.

5.2 Problems and Solutions

During the implementation phase of the project several issues which were unaccounted for during the design phase were encountered. In this section these problems will be listed as well as the solution found for each issue.

Problem	**Solution**
Selecting lines in a file to create a comment lacked the functionality to select lines which are not adjacent to each other.	The line selection interface was changed to have a checkbox for every line, allowing the user to select any possible combination of lines.
Front end does not know the user name, but has to display it in the top right corner. Simply fetching it from the API leads to a big overhead since this means an extra API call for every page.	Store the user name and some other user info in a cookie and display the data from the cookie, this way the number of API calls is greatly limited. A disadvantage of this approach is that the user name is not immediately updated when the value changes in the database.
To make the correct requests to the API the React[6] code needs to know what the parameters in the current URL are, but those parameters are evaluated in the express code of the front end.	It is possible to store the parameters in an attribute of the script tag that loads the React[6] script into a page template. This way the scripts can fetch them from the DOM[24].
Several pages were loading slowly, i.e. they took long to respond when actions like clicking a button or expanding a hidden section were conducted.	This was due to inefficient use of the React[6] framework, there were some classes that were updated too often because they were too big. The solution was to split up some classes into smaller parts so that only the necessary parts are updated.

Table 5.1: Problems and solutions

Chapter 6

Testing

In this chapter the testing phase will be discussed. The testing phase is one of the most important phases in software development. It creates insurances that the system is of high standards and is production ready. Therefore, several sort of tests were made to ensure CoDR is in fact of high quality. The tests can be divided into three main parts: white box tests, black box tests and acceptance tests. The white box testing phase includes system tests for the front and back end and swagger validation for the back end. The black box testing phase includes unit tests and swagger tests for the back end and selenium tests for the front end.

6.1 Black box Testing

Black box testing is an important part to ensuring that all the different system components work well together [25]. The black box testing is divided into different parts for the front end and for the back end. For the front end selenium tests were made. For the back end unit tests using the rest-assured framework [26] and swagger tests were made.

6.1.1 Selenium Testing

Selenium is a testing framework which acts as a normal user on a website [27]. It can automate user interaction to a website. Therefore, it can quickly test the front end.

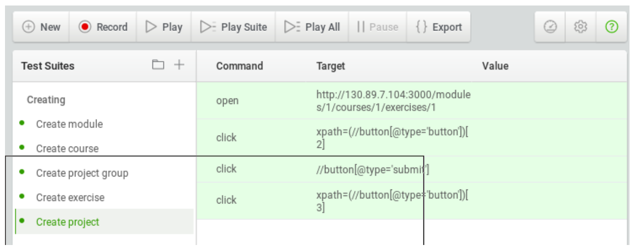
Several selenium tests were conducted which are detailed below. The selenium tests do not cover the full extent of the features of the website, for example deleting items and uploading files is not included. This is because selenium has trouble handling confirms and alerts. These features were tested manually.

Creating

1. Module
2. Rights group
3. Course
4. Project group
5. Exercise
6. Project

Comments

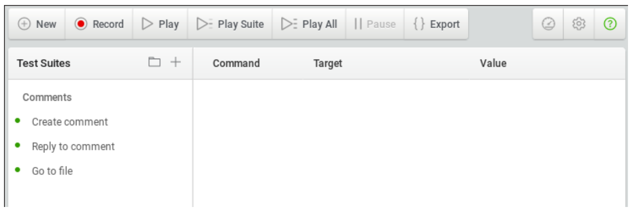
1. Create
2. Reply
3. Profile view -> go to file



Creating
- Create module
- Create course
- Create project group
- Create exercise
- Create project

Command	Target	Value
open	http://130.89.7.104:3000/modules/1/courses/1/exercises/1	
click	xpath=(//button[@type='button'])[2]	
click	//button[@type='submit']	
click	xpath=(//button[@type='button'])[3]	

Figure 6.1: Result of the selenium tests for creating several objects

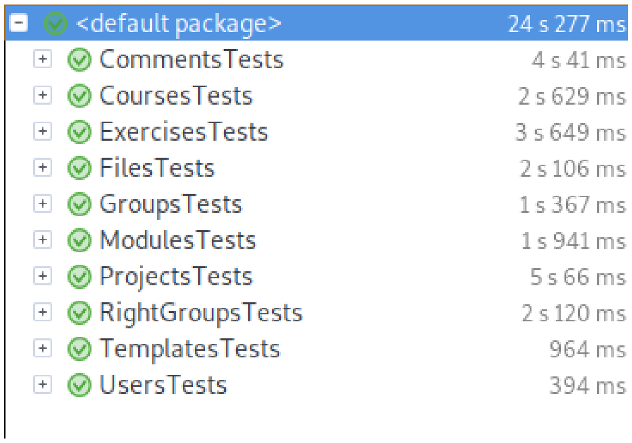


Comments
- Create comment
- Reply to comment
- Go to file

Figure 6.2: Result of the selenium tests for commenting on a file

6.1.2 Unit Testing

As was stated earlier, the rest-assured framework [26] was chosen to black box test the API. The rest-assured framework makes it easy to create automated tests for the API. For every API endpoint tests were made. The results of the tests are shown in 6.3



```
<default package>        24 s 277 ms
  CommentsTests           4 s 41 ms
  CoursesTests            2 s 629 ms
  ExercisesTests          3 s 649 ms
  FilesTests              2 s 106 ms
  GroupsTests             1 s 367 ms
  ModulesTests            1 s 941 ms
  ProjectsTests           5 s 66 ms
  RightGroupsTests        2 s 120 ms
  TemplatesTests              964 ms
  UsersTests                  394 ms
```

Figure 6.3: Result of all the unit tests

6.1.3 Swagger Testing

Swagger[11] provides useful testing utilities to create tests for an API. When developing the back end swagger provided a web interface that made it possible to easily generate requests for the API. The web interface gave a list of all API endpoints and their specification. It made it possible to change the request variables and then send the request. The reply from the server is shown in the interface in three different forms: raw data, formatted data and pretty printed data. This made it easy to check if the request had been processed properly and what return code and data the server had sent back. A result of this it was very easy to quickly test API endpoints during development.

6.2 White box Testing

White box testing is an important part to ensuring all the different system components work well independently. For the front end, manual system tests were executed and for the back end swagger tests were made.

6.2.1 System Testing

System tests are a vital part of testing the application. System tests allow the developer to individually test isolated components of the application. During the project phase all the team members performed manual system tests.

6.2.2 Swagger Input Validation

Swagger[11] provides automatic input validation that checks all requests made to the API and rejects them if their input does not match the expected input. It also shows an error when a response is sent that does not satisfy the response specified in the API. This makes it very easy to test both the front and back-end as an error is shown when a reply/response does not match its expected value.

6.3 Acceptance Testing

An acceptance test with real users was conducted. During this session the intent was to assign different roles (Teaching assistant, teacher, students) to different users and let them play out the scenarios as described in sections 3.3 and 4.2.3. Unfortunately the University of Twente log in system was not transferred to the production version before the session, this resulted in only a limited number of accounts being available. Therefore, the session has been changed to a demo with feedback from the available users. During this session several new ideas and improvements were found. These are discussed in section 6.3.1

6.3.1 New Requirements

- Named project groups
- In the project group view the projects that belong to the selected project group could be shown
- Select the lines to comment on by highlighting them
- Ability to change user name
- Temporary assume another (lower) role, for example as administrator assume the role of student temporarily to see what they can and cannot do
- Download a project
- Announcements, a message from a moderator or administrator that has to be read by all users in the object where it is posted, i.e. module, course, exercise etc
- Project summary, i.e. latest comments, changed files
- In profile view, instead of "Go to file", "Go to file <filename>"
- In profile view, replies to comments

- Color codes to show what type of user that made a comment, i.e. student, teaching assistant, moderator
- Show how many replies to a comment
- Show that another user is already responding to a comment

These results were taken into consideration, however due to time constraints these could not be implemented. See section 8.4 for more information.

Chapter 7

Responsibilities

To ensure an efficient work flow it was decided to split the team into two equal parts for the front end development and back end development. In tables 7.1 and 7.2 the tasks are laid out together with the person completing the task.

Item	Description	Jan-Jaap	Joost
Hierarchy	implement a blackboard-like hierarchy into our system	x	
File page	Create the layout and implementation of the file view		x
Commenting	Be able to create and reply to comments		x
Rights system	Create and implement the rights system	x	
Testing	Create selenium tests for the front end	x	
Documentation	Document the front end code	x	x

Table 7.1: Tasks front end

Item	Description	Noël	Jan-Jaap	Joost	Tom
UT login	Connect our system to the login system of the University of Twente				x
Database Schema	Setting up the database, creating the tables		x		x
Database Functions	Creating the database functions for authorization	x	x		
Implementing the controllers	Writing the actual code which contains the logic	x			x
Swagger	Creating Swagger definitions for the API	x			x
Testing	Create tests for the API endpoints	x		x	
Documentation	Document the back end code				x
Delivery	Setting up the Docker container for deliverables				x

Table 7.2: Tasks back end

Chapter 8

Discussion and Conclusion

8.1 Global Methodology

During the ten weeks of the project, daily work meetings were held to work on the tasks that were to be completed. Next to the daily meetings also weekly meetings with the Client/Supervisor 1.2 were held. These weekly meetings were useful to discuss our progress and receive feedback on the work that was delivered in the past week so that the project could keep going into the direction of a product that is in line with what the client wants.

8.2 Team Composition

All the members of the project group knew each other and worked together on previous projects throughout their bachelor careers at the University of Twente. Therefore, the team cohesion of the project group was great from the start, which made the communication within the team very natural and fluid. Having a team member with whom the other team members do not have a good relation could result in awkward situations or team members not giving criticism to avoid ruining the atmosphere. The good team atmosphere and practices enabled the team members to work together effectively. This led to a product that was of academic level and finished in time.

8.3 Conclusion

At the end of the time dedicated to the project all must requirements have been successfully completed, however not all requirements were finished due to the amount of work necessary to complete the must requirements. In the next section completed requirements have been marked with ✓, partially completed requirements are marked with ~ and requirements which have not been completed are marked with ×.

8.3.1 Requirements

Must

- ✓ As a user I must be able to log in and out of the system.
- ✓ As an administrator I must be able to create a module.
- ✓ As a moderator I must be able to add a course to a module.
- ✓ As a moderator I must be able to add users to a module.
- ✓ As a moderator I must be able to add exercises to a course.
- ✓ As a user I must be able to create a project in an exercise of a course that I am a member of.
- ✓ As a user I must be able to upload code to a project that I am a member of.
- ✓ As a user I must be able to create a comment on a project on which I have that right.
- ✓ As a user I must be able to view a personal page with comments that are interesting to me.
- ✓ As a reviewer I must be able to see projects in a module I am a member of.
- ✓ As a reviewer I must be able to comment on a project in a module I am a member of.
- ✓ As a moderator I must be able to assign different rights to user groups within a module, so that they can have limited or extended rights.
- ✓ As a moderator I must be able to assign different rights to user groups within an exercise, so that they can have limited or extended rights.
- ✓ As a comment I must be able to be attached to a line of code.
- ✓ As a comment I must be able to be attached to another comment.
- ✓ As a system I must be able to show code with syntax highlighting and line numbers.

As the must requirements are critical requirements in order for the project to be a success in the MoSCoW[2] model these requirements have been focused on and were completed first, as can be seen by the fact that all requirements have been completed.

Should

- × As a user I should be able to close a comment thread I started.
- × As a moderator I should be able to add other users to a project.
- ✓ As a user I should be able to remove a user from a project I created.
- ✓ As a user I should be able to remove a comment that I made.
- ✓ As a user I should be able to remove a project I created.
- ✓ A user should be able to comment on the project of another user if I have that right.
- × As a reviewer I should be able to attach a tag to a comment.
- × As a reviewer I should be able to close any comment thread.

- ✓ As a moderator I should be able to control the visibility level of comments of different user groups (i.e. who can see what.).
- ✓ As a moderator I should be able to control the visibility level of projects (i.e. who can see what).
- × As a moderator I should be able to add users to any project in a course.
- × As a moderator I should be able to remove a user from any project in a course.
- ✓ As a moderator I should be able to remove any project in a course.
- ✓ As a moderator I should be able to remove any comment in a course.
- ✓ As a moderator I should be able to choose a template for user rights in an exercise.
- ~ As a moderator I should be able to create a new template for user rights.
- ✓ As a moderator I should be able to make the comments of different user groups invisible until I approve them.
- ✓ As a comment I should be unapproved if my author has limited permissions to post comments.
- ✓ As a moderator I should be able to approve unapproved comments.
- ✓ As a moderator I should be able to remove unapproved comments.

The should requirements are requirements which are important but not necessary for the project to be a success. As these requirements are still important many of them have been completed after the must requirements had been completed, however it was not possible to complete them all during the time span of the project.

The partially completed requirement is partially completed because it is only possible for administrators to create a new template for user rights, while moderators are not able to do this. This was done due to the fact that allowing moderators to manage templates could result in moderators adjusting each other’s templates, possibly resulting in security flaws in the system.

Could

- × As a moderator I could be able to make a comment sticky to make it appear at the top of a thread.
- × As a system I could be able to receive comments generated by a tool (i.e. checkstyle, etc.).
- × As a user I could be able to upvote/downvote comments.
- × As a user I could be able to connect my project to a git repository to receive the code from there.
- × As a moderator I could be able to add a deadline to an exercise.
- × As a moderator I could be able to add a final verdict to a project.
- × As a user I could be able to view a dashboard for a project.

The requirements marked as could are requirements that are desirable for the project but not necessary. Most of these requirements are requirements which might increase the usability of the system or the overall experience of the customer. These requirements have not been implemented due to a lack of time.

Would

× An external tool would be able to act as a reviewer.

The would requirements are requirements which will not be implemented during this development phase due to them being too much work or out of scope for the current development phase. The would requirements could however be added to the product in the future and should be taken into consideration when developing the project. Similarly, to could requirements, these requirements have not been implemented due to a lack of time.

8.4 Future Work

The main points of future work are implementing the should, could and would requirements that were not implemented in this project, as was mentioned in section 8.3. Next to these requirements the results of the acceptance tests could be implemented to increase the usability, user experience and the use cases of the project. Finally, the project could be extended to provide features like faster grading, automatic feedback and faster sign-off, which are features mentioned by other stakeholders during the design phase.

Bibliography

[1] P. M. Institute, *A Guide to the Project Management Body of Knowledge (PMBOK® Guide)–Fifth Edition*. Project Management Institute, 2013.

[2] Wikipedia contributors, "Moscow method — Wikipedia, the free encyclopedia," 2018. [Online; accessed 16-April-2018].

[3] J. Pruitt and J. Grudin, "Personas: Practice and theory," in *Proceedings of the 2003 Conference on Designing for User Experiences, DUX '03*, 2003. Cited By :254.

[4] TechEmpower, "Framework benchmarks." `https://www.techempower.com/benchmarks/`. (Accessed on 04/16/2018).

[5] S. Teller, "Benchmarking node, tornado and django for concurrency." `https://swizec.com/blog/benchmarking-node-tornado-and-django-for-concurrency/swizec/1616`. (Accessed on 04/16/2018).

[6] F. O. Source, "React, a javascript library for building user interfaces." `https://reactjs.org/`. (Accessed on 04/17/2018).

[7] "bundle your scripts." `https://webpack.js.org/`. (Accessed on 04/18/2018).

[8] "Sass, css with superpowers." `https://sass-lang.com/`. (Accessed on 04/18/2018).

[9] "Ecmascript 6." `http://es6-features.org`. (Accessed on 04/18/2018).

[10] "Jsx in depth." `https://reactjs.org/docs/jsx-in-depth.html`. (Accessed on 04/18/2018).

[11] ron@swagger.io, "Swagger module for node.js." `https://github.com/swagger-api/swagger-node`. (Accessed on 04/17/2018).

[12] S. M. Sohan, F. Maurer, C. Anslow, and M. P. Robillard, "A study of the effectiveness of usage examples in rest api documentation," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 53–61, Oct 2017.

[13] B. C., "Postgresql client - pure javascript & libpq with the same api." `https://www.npmjs.com/package/pg`. (Accessed on 04/17/2018).

[14] "What is object/relational mapping?." `http://hibernate.org/orm/what-is-an-orm/`. (Accessed on 04/17/2018).

[15] P. G. D. Group, "an object-relational database management system with an emphasis on extensibility and standards compliance.." `https://www.postgresql.org/`. (Accessed on 04/17/2018).

[16] O. corporation, "Mysql is an open-source relational database management system.." `https://www.mysql.com/`. (Accessed on 04/17/2018).

[17] D. R. Hipp, "Sqlite is a relational database management system contained in a c programming library.." `https://www.sqlite.org`. (Accessed on 04/17/2018).

[18] I. inc., "Canvas." `https://canvas.instructure.com`. (Accessed on 04/17/2018).

[19] M. J. B. Campbell, C. Mortimore, "Rfc 7522 - security assertion markup language (saml) 2.0 profile for oauth 2.0 client authentication and authorization grants." `https://tools.ietf.org/html/rfc7522`, May 2015.

[20] M. Stosberg, "Saml 2.0 authentication with passport." `https://www.npmjs.com/package/passport-saml`. (Accessed on 04/17/2018).

[21] J. Hanson, "Simple, unobtrusive authentication for node.js.." `https://www.npmjs.com/package/passport`. (Accessed on 04/17/2018).

[22] T. Preston-Werner, "Github, the world's leading software development platform." (Accessed on 04/17/2018).

[23] L. Torvalds, "Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people." `https://git-scm.com/about`. (Accessed on 04/17/2018).

[24] T. R. Jonathan Robie, "What is the document object model." `https://www.w3.org/TR/DOM-Level-1/introduction.html`. (Accessed on 04/17/2018).

[25] M. Utting and B. Legeard, *Practical Model-Based Testing*. Practical Model-Based Testing, 2007. Cited By :347.

[26] J. Haleby, "Rest assured." `http://rest-assured.io/`. (Accessed on 04/16/2018).

[27] "Selenium - web browser automation." `https://www.seleniumhq.org/`. (Accessed on 04/16/2018).

Appendix A

Mock-ups

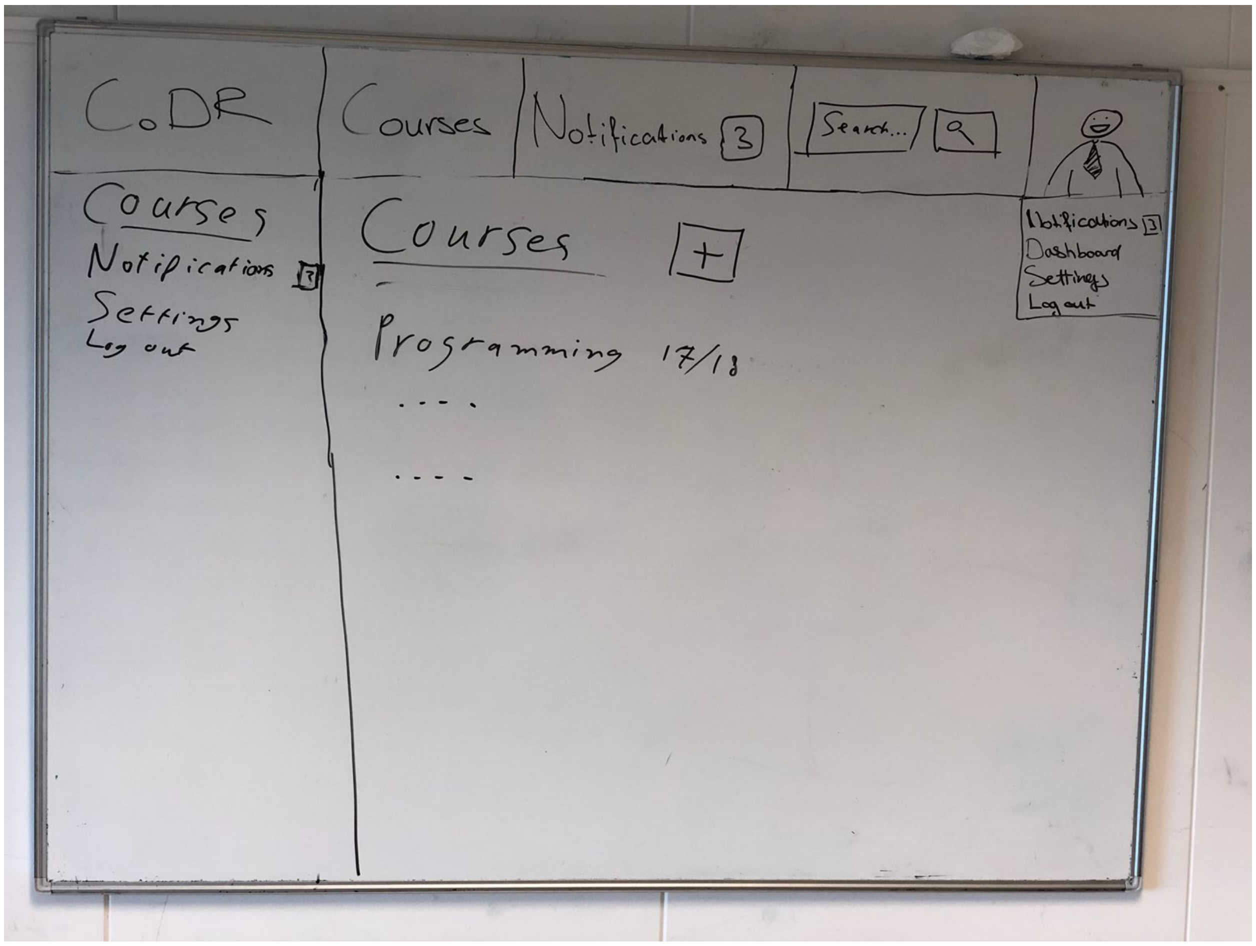


Figure A.1: The overview of the courses page with different exercises

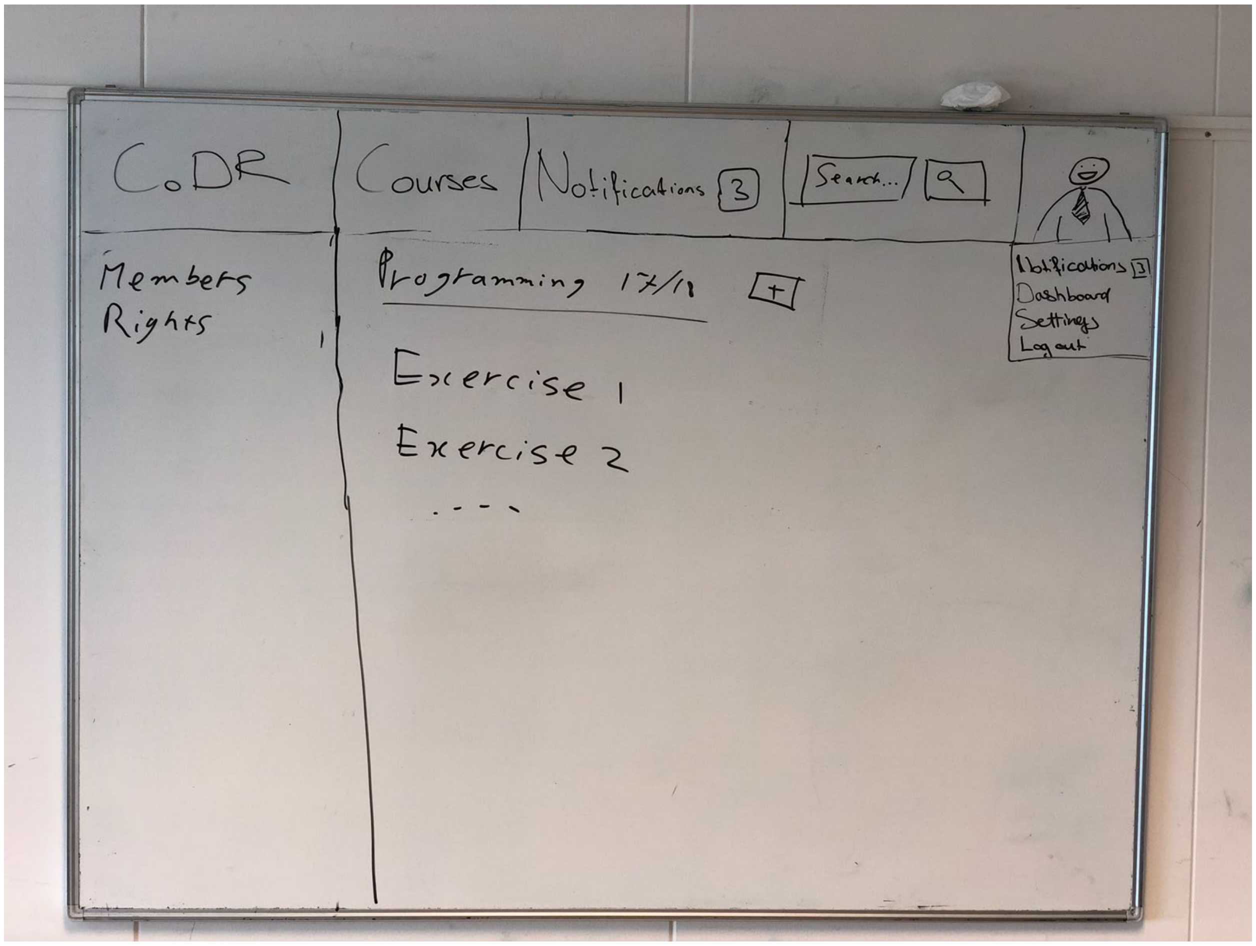
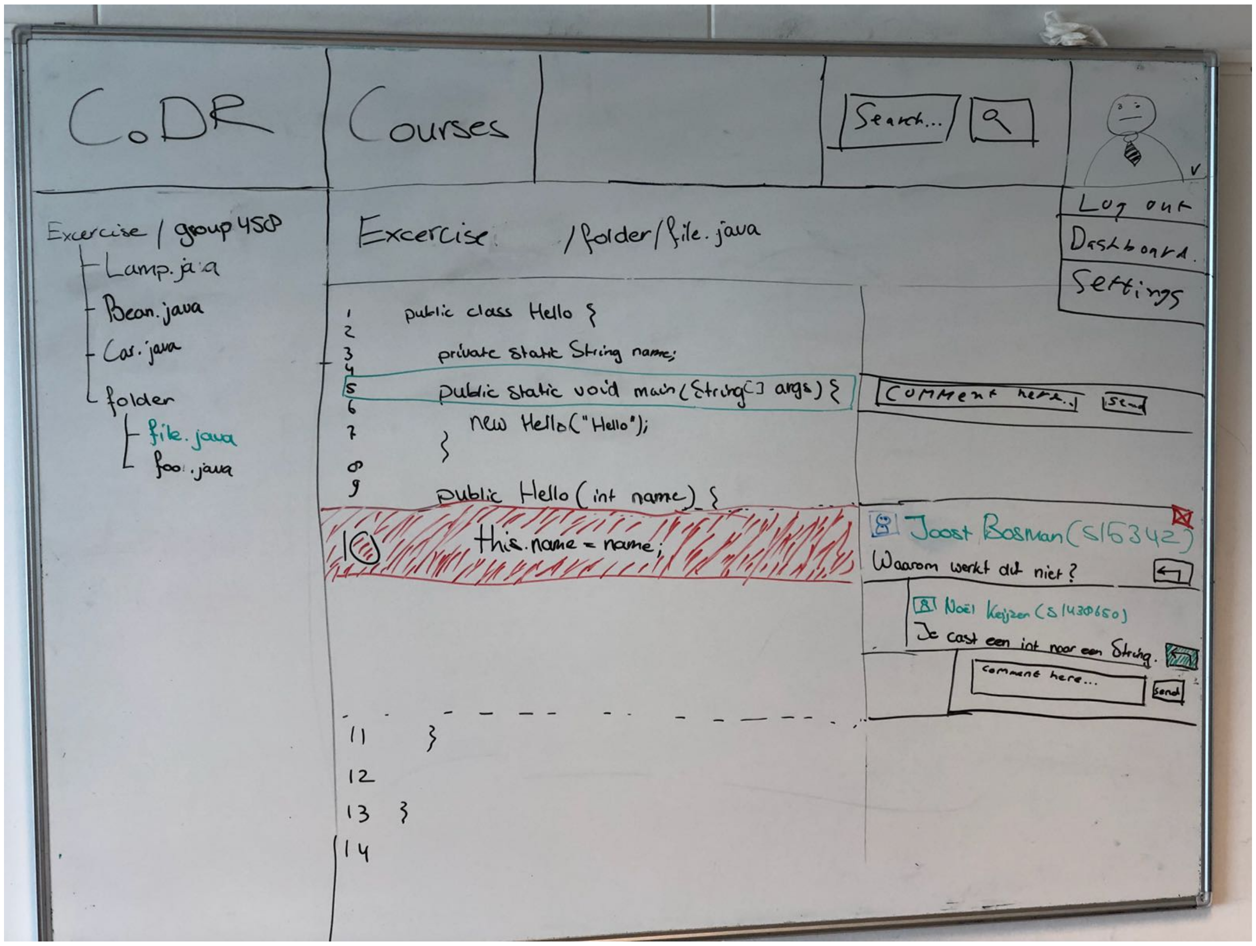


Figure A.2: The overview of a Course page with different courses



```
public class Hello {
    private static String name;
    public static void main(String[] args) {
        new Hello("Hello");
    }

    public Hello(int name) {
        this.name = name;
    }

}
```

Waarom werkt dit niet?

Je cast een int naar een String.

Figure A.3: The overview of a file page with a file and comments

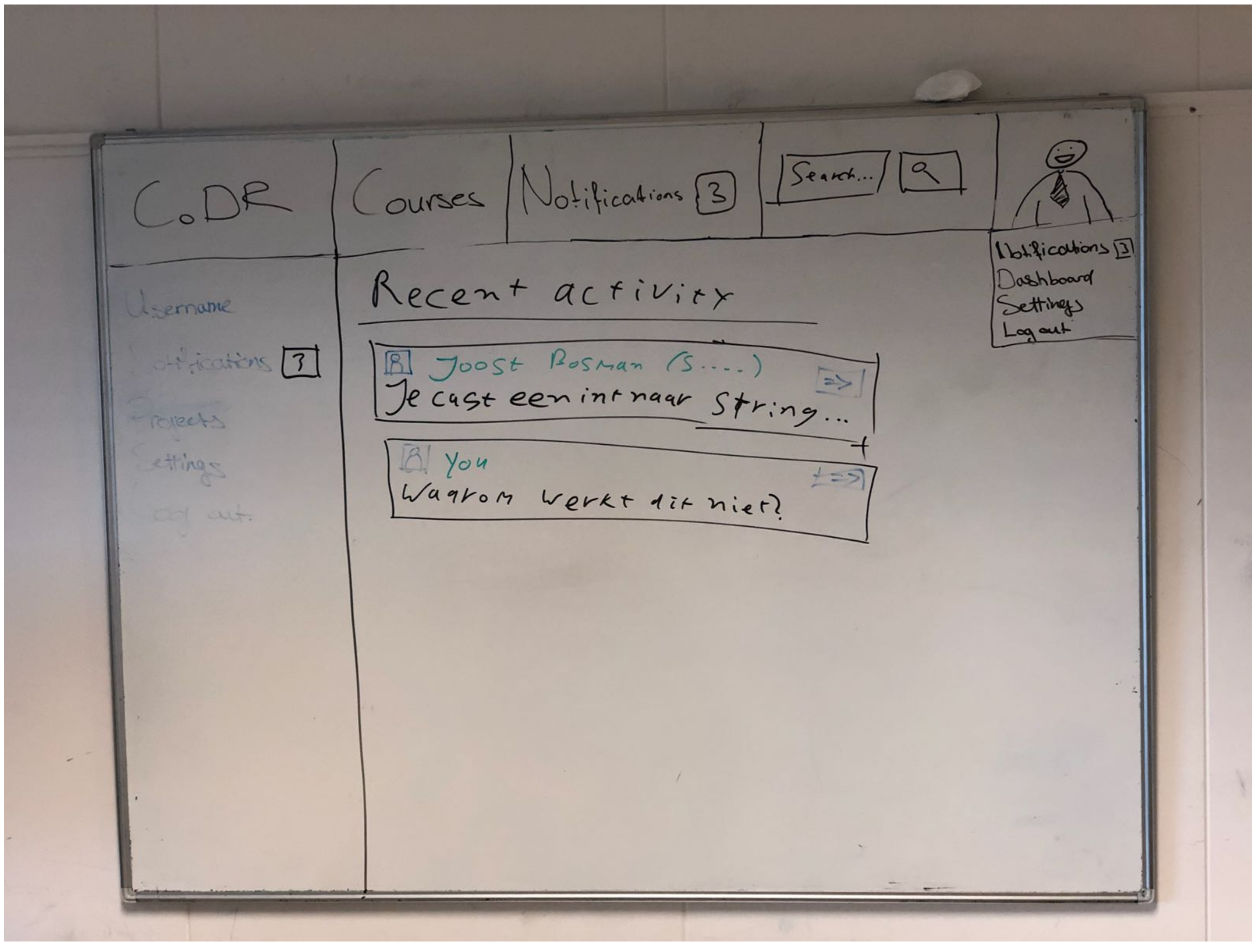


Figure A.4: The overview of a profile page with comments

Appendix B

Class diagram database

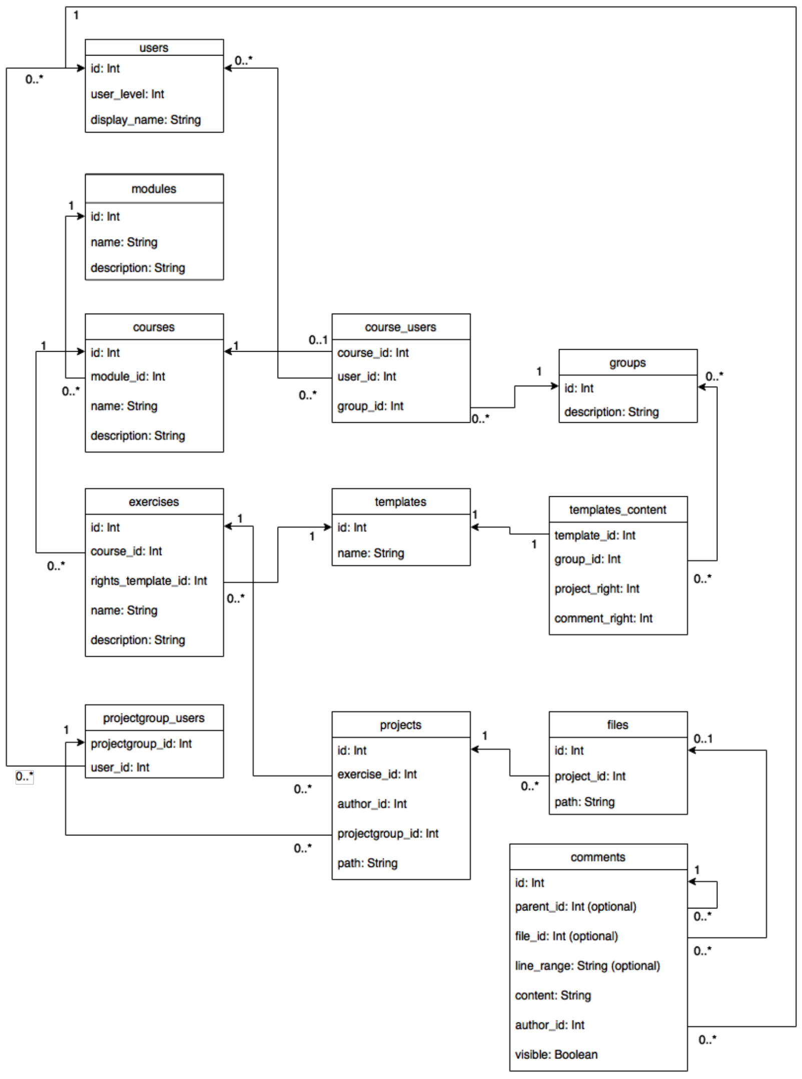


Figure B.1: Class diagram of the database

Appendix C

API specification

Overview

Version information

Version : 0.0.1

URI scheme

Host : localhost:10010
BasePath : /api
Schemes : HTTP, HTTPS

Consumes

- `application/json`
- `multipart/form-data`

Produces

- `application/json`

Definitions

CommentResponseHead

Name	Schema
author_id *optional*	integer (int64)
author_name *optional*	string
can_delete *optional*	boolean
can_edit *optional*	boolean
can_toggle_visibility *optional*	boolean
children *optional*	< CommentResponseTail > array
content *optional*	string
course_id *optional*	integer (int64)
deleted *optional*	boolean
exercise_id *optional*	integer (int64)
file_id *optional*	integer (int64)
id *optional*	integer (int64)
line_range *optional*	string
module_id *optional*	integer (int64)
parent_id *optional*	integer (int64)
project_id *optional*	integer (int64)
timestamp *optional*	integer (int64)
visible *optional*	boolean

CommentResponseList

Type : < CommentResponseHead > array

CommentResponseTail

Name	Schema
author_id *optional*	integer (int64)
author_name *optional*	string
can_delete *optional*	boolean
can_edit *optional*	boolean
can_toggle_visibility *optional*	boolean
children *optional*	< object > array
content *optional*	string
course_id *optional*	integer (int64)
deleted *optional*	boolean
exercise_id *optional*	integer (int64)
file_id *optional*	integer (int64)
id *optional*	integer (int64)
line_range *optional*	string
module_id *optional*	integer (int64)
parent_id *optional*	integer (int64)
project_id *optional*	integer (int64)
timestamp *optional*	integer (int64)
visible *optional*	boolean

CourseResponse

Name	Schema
description *optional*	string
id *optional*	integer (int64)
name *optional*	string

CourseResponseGet

Name	Schema
can_edit *required*	boolean
description *required*	string
exercises *optional*	< Exercise > array
id *optional*	integer (int64)
name *required*	string

CourseUpdate

Name	Schema
description *optional*	string
name *optional*	string

CurrentUser

Name	Schema
display_name *optional*	string
id *optional*	integer (int64)
user_level *optional*	integer (int64)

ErrorResponse

Name	Schema
message *optional*	string

Exercise

Name	Schema
description *required*	string
id *optional*	integer (int64)
name *required*	string
projects *optional*	< Project > array
rights_template_id *required*	integer (int64)

File

Name	Schema
comments *required*	< CommentResponseHead > array
content *required*	< Line > array
id *required*	integer (int64)
path *required*	string

Group

Name	Schema
Users *optional*	< User > array
id *optional*	integer (int64)
module_id *optional*	integer (int64)

GroupList

Type : < Group > array

Line

Type : string

Module

Name	Schema
can_edit *optional*	boolean
can_view *optional*	boolean
description *optional*	string
id *optional*	integer (int64)
name *optional*	string

ModuleCourse

Name	Schema
description *required*	string
id *optional*	integer (int64)
name *required*	string

ModuleList

Name	Schema
can_edit *optional*	boolean
modules *optional*	< Module > array

ModuleResponse

Name	Schema
can_edit *optional*	boolean
courses *optional*	< ModuleCourse > array
description *optional*	string
groups *optional*	< Group > array
id *optional*	integer (int64)
name *optional*	string

ModuleUpdate

Name	Schema
description *optional*	string
name *optional*	string
users *optional*	< users > array

users

Name	Schema
group_id *optional*	integer (int64)
user_id *optional*	integer (int64)

Project

Name	Schema
files *optional*	< File > array
id *optional*	integer (int64)
projectgroup_id *required*	integer (int64)

RightGroup

Name	Schema
description *required*	string
id *required*	integer (int64)

Rights

Name	Schema
comment_rights *optional*	comment_rights
description *optional*	string
id *optional*	integer (int64)
project_rights *optional*	project_rights

comment_rights

Name	Schema
member *optional*	member
other *optional*	other
owner *optional*	owner

member

Name	Schema
can_delete *required*	boolean
can_edit *required*	boolean
can_toggle_visibility *required*	boolean
can_view *required*	boolean

other

Name	Schema
can_delete *required*	boolean
can_edit *required*	boolean
can_toggle_visibility *required*	boolean
can_view *required*	boolean

owner

Name	Schema
can_delete *required*	boolean
can_edit *required*	boolean
can_toggle_visibility *required*	boolean
can_view *required*	boolean

project_rights

Name	Schema
member *optional*	member
other *optional*	other

member

Name	Schema
can_create_comment *required*	boolean
can_create_visible_comment *required*	boolean
can_delete *required*	boolean
can_edit *required*	boolean
can_view *required*	boolean

other

Name	Schema
can_create_comment *required*	boolean
can_create_visible_comment *required*	boolean
can_delete *required*	boolean
can_edit *required*	boolean
can_view *required*	boolean

Template

Name	Schema
id *optional*	integer (int64)
name *optional*	string
rights *optional*	< Rights > array

TemplateList

Type : < Template > array

User

Name	Schema
display_name *optional*	string
id *optional*	integer (int64)

UserCommentResponse

Name	Schema
author_id *optional*	integer (int64)
can_delete *optional*	boolean
can_edit *optional*	boolean

Name	Schema
can_toggle_visibility *optional*	boolean
content *optional*	string
course_id *optional*	integer (int64)
deleted *optional*	boolean
exercise_id *optional*	integer (int64)
file_id *optional*	integer (int64)
id *optional*	integer (int64)
line_range *optional*	string
module_id *optional*	integer (int64)
parent_id *optional*	integer (int64)
project_id *optional*	integer (int64)
timestamp *optional*	integer (int64)
visible *optional*	boolean

UserCommentResponseList

Type : < UserCommentResponse > array

UserList

Type : < User > array

Paths

POST /groups

Description

Creates a new group.

Parameters

Type	Name	Description	Schema
Body	**body** *optional*	ID of the module for which the group should be created.	body

body

Name	Schema
module_id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
201	Success, new group has been created.	Response 201
401	Error, user is not allowed to access this page.	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

Response 201

Name	Schema
id *optional*	integer (int64)

GET /groups

Description

Returns a list of all groups.

Responses

HTTP Code	Description	Schema
200	Success	GroupList
401	Error, user is not allowed to access this page.	No Content
500	Internal server error	No Content

GET /groups/{group_id}

Description

Returns the details of the specified group.

Parameters

Type	Name	Description	Schema
Path	**group_id** *required*	ID of the group to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Group
401	Error, user is not allowed to access this page.	No Content
404	Error, group could not be found.	No Content
500	Internal server error	No Content

PUT /groups/{group_id}

Description

Updates the specified group.

Parameters

Type	Name	Description	Schema
Path	**group_id** *required*	ID of the group to update.	integer (int64)
Body	**body** *required*	Data for the group to update.	< body > array

body

Name	Schema
id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Group
400	Error, invalid input.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Error, group could not be found.	No Content
500	Internal server error	No Content

DELETE /groups/{group_id}

Description

Deletes the specified group.

Parameters

Type	Name	Description	Schema
Path	**group_id** *required*	ID of the group to delete.	integer (int64)

Responses

HTTP Code	Description	Schema
204	Success.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Error, group could not be found.	No Content
500	Internal server error	No Content

POST /modules

Description

Creates a new module.

Parameters

Type	Name	Description	Schema
Body	**body** *required*	JSON representation of the module to create.	body

body

Name	Schema
description *optional*	string
name *optional*	string

Responses

HTTP Code	Description	Schema
201	Success, new module has been created.	Response 201
401	Error, user is not allowed to access this page.	No Content
405	Error, invalid input.	No Content
500	Internal server error	No Content

Response 201

Name	Schema
description *optional*	string
id *optional*	integer (int64)
name *optional*	string

GET /modules

Description

Returns a list of all modules.

Responses

HTTP Code	Description	Schema
200	Success	ModuleList

HTTP Code	Description	Schema
500	Internal server error	No Content

GET /modules/{module_id}

Description

Returns the details of the specified module.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	ModuleResponse
404	Error, module could not be found.	No Content

PUT /modules/{module_id}

Description

Updates the specified module.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module to update.	integer (int64)
Body	**body** *optional*	Data for the module to update.	ModuleUpdate

Responses

HTTP Code	Description	Schema
200	Success	ModuleResponse
401	Error, user is not allowed to access this page.	No Content
404	Error, module could not be found.	No Content

HTTP Code	Description	Schema
405	Incorrect input	No Content
500	Internal server error	No Content

DELETE /modules/{module_id}

Description

Deletes the specified module.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module to delete.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success.	No Content
401	Not allowed.	No Content
404	Error, module could not be found.	No Content
500	Internal server error	No Content

POST /modules/{module_id}/courses

Description

Creates a new course for the given module.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Body	**body** *required*	JSON representation of the course to create.	CourseUpdate

Responses

HTTP Code	Description	Schema
201	Success, new course has been created.	CourseResponse
401	Error, user is not allowed to access this page.	No Content
405	Error, invalid input	No Content
500	Internal server error	No Content

GET /modules/{module_id}/courses

Description

Returns the courses of the specified module.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	< Response 200 > array
401	Error, user is not allowed to access this page.	No Content
404	Error, module could not be found.	No Content
500	Internal server error	No Content

Response 200

Name	Schema
can_edit *optional*	boolean
description *required*	string
id *optional*	integer (int64)
module_id *optional*	integer (int64)
name *required*	string

GET /modules/{module_id}/courses/{course_id}

Description

Returns the specified course for the specified module.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	CourseResponseGet
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	ErrorResponse
500	Internal server error	No Content

PUT /modules/{module_id}/courses/{course_id}

Description

Updates the specified course for the specified module.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module from which to update a course.	integer (int64)
Body	**body** *required*	Data for the course to update.	CourseUpdate

Responses

HTTP Code	Description	Schema
200	Success	CourseResponse
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

DELETE /modules/{module_id}/courses/{course_id}

Description

Deletes the specified course from the specified module.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to delete.	integer (int64)
Path	**module_id** *required*	ID of the module from which to delete a course.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	No Content
500	Internal server error	No Content

POST /modules/{module_id}/courses/{course_id}/exercises

Description

Adds a new exercise to the specified course.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Body	**body** *required*	JSON representation of the Exercise to create.	body

body

Name	Schema
description *required*	string
name *required*	string
rights_template_id *required*	integer (int64)

Responses

HTTP Code	Description	Schema
201	Success, new course has been created.	Response 201
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	No Content
405	Error, invalid input.	No Content
500	Internal server error	No Content

Response 201

Name	Schema
course_id *required*	integer (int64)
description *required*	string
id *required*	integer (int64)
name *required*	string
rights_template_id *required*	integer (int64)

GET /modules/{module_id}/courses/{course_id}/exercises

Description

Returns a list of exercises for the specified course for the specified module.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	< Response 200 > array
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	No Content
500	Internal server error	No Content

Response 200

Name	Schema
course_id *optional*	integer (int64)
description *required*	string
id *optional*	integer (int64)
name *required*	string
rights_template_id *required*	integer (int64)

GET /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}

Description

Returns the specified exercise.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Response 200
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise could not be found.	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

Response 200

Name	Schema
can_edit *optional*	boolean
description *required*	string
id *optional*	integer (int64)
name *required*	string
projects *optional*	projects
rights_template_id *required*	integer (int64)

projects

Name	Schema
author_id *optional*	integer (int64)
can_delete *optional*	boolean
exercise_id *optional*	integer (int64)
id *optional*	integer (int64)
projectgroup *optional*	projectgroup
projectgroup_id *optional*	integer (int64)

projectgroup

Name	Schema
id *optional*	integer (int64)
module_id *optional*	integer (int64)
users *optional*	< users > array

users

Name	Schema
display_name *optional*	string
id *optional*	integer (int64)

PUT /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}

Description

Updates the specified exercise.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module from which to update a course.	integer (int64)
Body	**body** *required*	Data for the Exercise to update.	body

body

Name	Schema
description *optional*	string
name *optional*	string
rights_template_id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Response 200
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise could not be found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

Response 200

Name	Schema
description *optional*	string
id *optional*	integer (int64)
name *optional*	string
rights_template_id *optional*	integer (int64)

DELETE /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}

Description

Deletes the specified exercise.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module from which to delete a course.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise could not be found.	No Content
500	Internal server error	No Content

POST /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects

Description

Adds a new Project to the specified exercise.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Body	**body** *required*	JSON representation of the Project to create.	body

body

Name	Schema
projectgroup_id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
201	Success, new course has been created.	Response 201
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course could not be found.	No Content
405	Error, invalid input.	No Content
500	Internal server error	No Content

Response 201

Name	Schema
author_id *optional*	integer (int64)
exercise_id *optional*	integer (int64)
id *optional*	integer (int64)
projectgroup_id *optional*	integer (int64)

GET /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects

Description

Returns all projects for an exercise

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	< Response 200 > array
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise or project could not be found.	No Content
500	Internal server error	No Content

Response 200

Name	Schema
author_id *optional*	integer (int64)
exercise_id *optional*	integer (int64)
id *optional*	integer (int64)
projectgroup_id *optional*	integer (int64)

GET /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}

Description

Returns the specified project.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module to fetch.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Response 200
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise or project could not be found.	No Content
500	Internal server error	No Content

Response 200

Name	Schema
author_id *optional*	integer (int64)
can_create_comment *optional*	boolean
can_create_visible_comment *optional*	boolean
can_delete *optional*	boolean
can_edit *optional*	boolean
can_view *optional*	boolean
exercise_id *optional*	integer (int64)
files *optional*	< files > array
id *optional*	integer (int64)
project_groups *optional*	< project_groups > array

files

Name	Schema
id *optional*	integer (int64)
path *optional*	string

project_groups

Name	Schema
id *optional*	integer (int64)
users *optional*	< users > array

users

Name	Schema
display_name *optional*	string
id *optional*	integer (int64)

PUT /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}

Description

Updates the specified project.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module from which to update a course.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)
Body	**body** *required*	Data for the project to update.	body

body

Name	Schema
projectgroup_id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Response 200
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise or project could not be found	No Content
500	Internal server error	No Content

Response 200

Name	Schema
author_id *optional*	integer (int64)
exercise_id *optional*	integer (int64)
id *optional*	integer (int64)
projectgroup_id *optional*	integer (int64)

DELETE /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}

Description

Deletes the specified project.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course to fetch.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module from which to delete a course.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Error, module or course or exercise or project could not be found.	No Content
500	Internal server error	No Content

POST /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files

Description

Saves a file to the database.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)
FormData	**files** *optional*	files to post	file

Responses

HTTP Code	Description	Schema
200	Success	No Content

HTTP Code	Description	Schema
401	Not allowed	No Content
405	Invalid input	No Content
500	Internal server error	No Content

Consumes

- `multipart/form-data`

GET /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files/{file_id}

Description

Returns the file with the specified id.

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**file_id** *required*	ID of file to fetch	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	Response 200
401	Not allowed	No Content
404	Id not found	No Content
500	Internal server error	No Content

Response 200

Name	Schema
content *optional*	< string > array
id *optional*	integer (int64)
path *optional*	string
project_id *optional*	integer (int64)

POST /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files/{file_id}/comments

Description

Creates a comment

Parameters

Type	Name	Description	Schema
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**file_id** *required*	ID of file to fetch comments for	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)
Body	**comment** *required*		comment

comment

Name	Schema
content *optional*	string
line_range *optional*	string

Name	Schema
parent_id *optional*	integer (int64)

Responses

HTTP Code	Description	Schema
201	Success, new comment has been created.	No Content
401	Error, user is not allowed to access this page.	No Content
404	Not found.	No Content
405	Error, invalid input.	No Content
500	Internal server error	No Content

PUT /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files/{file_id}/comments/{comment_id}

Description

Updates the specified comment.

Parameters

Type	Name	Description	Schema
Path	**comment_id** *required*	ID of comment	integer (int64)
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**file_id** *required*	ID of file to fetch comments for	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)
Body	**body** *optional*	body of the request	body

body

Name	Schema
content *optional*	string
visible *optional*	boolean

Responses

HTTP Code	Description	Schema
200	Success	No Content
401	Not allowed	No Content
404	Id not found	No Content
500	Internal server error	No Content

DELETE /modules/{module_id}/courses/{course_id}/exercises/{exercise_id}/projects/{project_id}/files/{file_id}/comments/{comment_id}

Description

Deletes the specified comment.

Parameters

Type	Name	Description	Schema
Path	**comment_id** *required*	ID of comment	integer (int64)
Path	**course_id** *required*	ID of the course from which to fetch exercises.	integer (int64)
Path	**exercise_id** *required*	ID of the exercise to fetch.	integer (int64)
Path	**file_id** *required*	ID of file to fetch comments for	integer (int64)
Path	**module_id** *required*	ID of the module from which to fetch courses.	integer (int64)
Path	**project_id** *required*	ID of the project to fetch.	integer (int64)

Responses

HTTP Code	Description	Schema
204	Item deleted	No Content
401	Not allowed	No Content
404	Id not found	No Content
500	Internal server error	No Content

GET /modules/{module_id}/group

Description

Returns the groups for this module for the current user.

Parameters

Type	Name	Description	Schema
Path	**module_id** *required*	ID of the module from which to fetch groups.	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	< Response 200 > array
500	Internal server error	No Content

Response 200

Name	Schema
group_id *optional*	integer (int64)
users *optional*	< User > array

POST /right_groups

Description

Creates a new right_groups.

Parameters

Type	Name	Description	Schema
Body	**body** *required*	Description of the new right's group.	body

body

Name	Schema
description *optional*	string

Responses

HTTP Code	Description	Schema
201	Success	RightGroup
401	Not allowed	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

GET /right_groups

Description

Returns a list of all right_groups.

Responses

HTTP Code	Description	Schema
200	Success	RightGroup
500	Internal server error	No Content

PUT /right_groups/{right_group_id}

Description

Updates the specified rights group.

Parameters

Type	Name	Description	Schema
Path	**right_group_id** *required*	ID of rights group	integer (int64)
Body	**body** *required*	Description of the new right’s group.	body

body

Name	Schema
description *optional*	string

Responses

HTTP Code	Description	Schema
200	Right group updated	No Content
401	Not allowed	No Content
404	Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

DELETE /right_groups/{right_group_id}

Description

Deletes the specified rights group.

Parameters

Type	Name	Description	Schema
Path	**right_group_id** *required*	ID of the right group to delete	integer (int64)

Responses

HTTP Code	Description	Schema
204	Item deleted	No Content
401	Not allowed	No Content
404	Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

POST /templates

Description

Creates a new template.

Parameters

Type	Name	Description	Schema
Body	**body** *required*	Rights for the template	Template

Responses

HTTP Code	Description	Schema
201	Success, new template has been created.	No Content
401	Error, user is not allowed to access this page.	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

GET /templates

Description

Returns a list of all rights templates.

Responses

HTTP Code	Description	Schema
200	Success	TemplateList
500	Internal server error	No Content

PUT /templates/{template_id}

Description

Updates the specified template.

Parameters

Type	Name	Description	Schema
Path	**template_id** *required*	ID of template	integer (int64)
Body	**body** *optional*	body of the request	Rights

Responses

HTTP Code	Description	Schema
200	Group rights updated	No Content
201	new group rights created	No Content
401	Not allowed	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

DELETE /templates/{template_id}

Description

Deletes the specified rights template.

Parameters

Type	Name	Description	Schema
Path	**template_id** *required*	ID of the template to delete	integer (int64)

Responses

HTTP Code	Description	Schema
204	Item deleted	No Content
401	Not allowed	No Content
404	Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

GET /users

Description

Returns a list of all user accounts.

Responses

HTTP Code	Description	Schema
200	Success	UserList
401	Not allowed	No Content
500	Internal server error	No Content

GET /users/current

Description

Returns information for the currently logged in user.

Responses

HTTP Code	Description	Schema
200	Success	CurrentUser
401	Not allowed	No Content

GET /users/current/comments

Description

Return all comments that the current user can see.

Responses

HTTP Code	Description	Schema
200	Success	CommentResponseList
401	Not allowed	No Content

GET /users/{user_id}

Description

Returns the details of the specified user account.

Parameters

Type	Name	Description	Schema
Path	**user_id** *required*	ID of customer to fetch account for	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	CurrentUser
401	Not allowed	No Content
404	User_Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

PUT /users/{user_id}

Description

Updates the specified user account.

Parameters

Type	Name	Description	Schema
Path	**user_id** *required*	ID of customer to fetch account for	integer (int64)
Body	**body** *required*	body of the request	User

Responses

HTTP Code	Description	Schema
200	Success	User
401	Not allowed	No Content
404	Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

DELETE /users/{user_id}

Description

Deletes the specified user account.

Parameters

Type	Name	Description	Schema
Path	**user_id** *required*	ID of customer to fetch account for	integer (int64)

Responses

HTTP Code	Description	Schema
204	Item deleted	No Content
401	Not allowed	No Content
404	Id not found	No Content
405	Incorrect input	No Content
500	Internal server error	No Content

GET /users/{user_id}/comments

Description

Returns all comments for the specified user.

Parameters

Type	Name	Description	Schema
Path	**user_id** *required*	ID of customer to fetch account for	integer (int64)

Responses

HTTP Code	Description	Schema
200	Success	UserCommentResponseList
405	Incorrect input	No Content
500	Internal server error	No Content