School Engine

Relationship Managers LLD

Document Number 4

Table of Contents

[Introduction 3](#_Toc126074323)

[Purpose 3](#_Toc126074324)

[Intended audience 3](#_Toc126074325)

[Environment Manager 4](#_Toc126074326)

[Answers Manager 4](#_Toc126074327)

[Announcements Manager 4](#_Toc126074328)

[Announcements related functions 4](#_Toc126074329)

[Answer related functions 5](#_Toc126074330)

[Users manager 6](#_Toc126074331)

[For references 7](#_Toc126074332)

[Courses Manager 8](#_Toc126074333)

[Tag Manager 9](#_Toc126074334)

[Tag related functions 9](#_Toc126074335)

[Relations related functions 9](#_Toc126074336)

[Todos Manager 10](#_Toc126074337)

[Grades Manager 11](#_Toc126074338)

# Introduction

## Purpose

The purpose of this file is to explain the logic behind the classes that are part of the relationship managers group. These classes will take care of validating the data before it is sent to the database, check the user permissions, and generate the actual response that the API will send back as a response (containing the HTTP code and the Json formatted body of the request).

## Intended audience

# General Structure

The purpose of these classes is to manage the conditions, process the information and manage the relationships for the system to work.

## Constructor

The constructor of each class will use multiple cql\_managers that have been presented in the precious document. It will take as parameters multiple pointers to such constructors and will store them as private variables.

## General structure for a function

The general idea behind a function is to process the data and to check if all conditions are met (mode details in the next chapters about all of these processes), and if they are, add, remove, get, or update data. The return value of the functions is going to be a pair between a Drogon::HttpStatusCode (example k200OK) and a JSON value. Although this not necessarily the most elegant way to do this, considering that these functions are going to be called in the API Managers for the purpose of processing the data but also for the final result, it would be a good idea to return the information in a way that does not require more processing. Leaving the return values as HttpStatusCode and JSON value would give us the ability for the API Managers to give the information to the caller without any more parsing.

The parameters that the functions are going to get will be the school id, a token that the user is assigned, and other information that is needed depending on the case.

In order to make the process of readding the logic behind the functions easier, the algorithms applied are going to be written in a pseudocode way. Keep in mind that the implementation in the bellow described functions is shortened in order to keep track of the most important aspects. The error checking is not described into the functions, nether is the returned value.

## Returned values

|  |  |  |
| --- | --- | --- |
| Case | HTTP\_CODE | JSON |
| Everything went ok for get | 200OK | The expected json value |
| Everything went of for create | 201Created | The newly created object in json format |
| Everything went ok for update & delete | 200OK | null |
| There was an error when reading the information | 500InternalServerError | Specific description for the error in a “error” field of a json |
| Fields are incorrect or given parameters is not found (i.e., the token) | 400BadRequest | Specific description for the error in a “error” field of a json |
| Action is not allowed because of the user type | 403Forbiden | Specific description for the error in a “error” field of a json |
| Some parameters that are given, could not be found in the database | 404NotFound | Specific description for the error in a “error” field of a json |

# Announcements Manager

## Constructor

The constructor of this class will take as parameters pointes of the following classes and will store them to private variables accordingly:

* AnnouncementsCqlManager
* AnnouncementsByTagCqlManager
* TagsCqlManager
* TokensCqlManager
* UsersCqlManager
* AnswersByAnnouncementOrQuestionCqlManager
* AnswersCqlManager
* FilesCqlManager
* TagsByUserCqlManager

## Announcements related functions

The functions are going to be split into 4 categories: functions that would manage the announcement, functions that would manage the relations between the tags and announcements, functions that would manage the files and functions that would manage the announcement’s answers.

### Create announcement

* Get the user by token.
* Check if the user is a teacher or an admin.
* Create the announcement and put it into the database.
* Return the newly created announcement in JSON format.

### Get announcements

* Get the user by token.
* If the user is not an admin:
  + Get all the user’s tags.
  + Get all the announcements ids related to the received tags.
  + Read the announcements by the above ids and store them.
* Else:
  + Get all the announcements ids from the school.
  + Read the announcements by the above ids and store them.
* Return the above stored announcements in JSON format.

### Delete announcement

* Get the user by token.
* Get the announcement to be deleted.
* Check if the user is either admin or has created the announcement.
* Delete the announcement from the database.
* Delete the relations between tags and announcements that contain the deleted announcement id.
* Delete the answers that are related to the announcement.
* Delete the relations between the answers and the announcement.
* Delete the files that are related (not the actual file, just it’s entry in the database).
* Return the path to the announcement in our folder structure. This is going to be used by the API Manager to delete the file/folder.

### Add tag to announcement

* Get the user by token.
* Get the announcement (just checking if it exists).
* Get the tag (just checking if it exists).
* Check if the user adding is the admin or the user that created the announcement.
* Add the announcement – tag relationship.
* Return 200OK and null JSON.

### Get announcement tags

* Get the user by token.
* Check if the announcement exists.
* Check if the user adding is the admin or the user that created the announcement.
* Get all the tags id that are related to this announcement.
* Get the actual tag objects from the ids above.
* Return 200OK and the array of tags (containing the id, name, and colour).

### Remove tag from announcement

* Get the user by token.
* Check if the announcement exists.
* Check if the tag exists.
* Check if the user adding is the admin or the user that created the announcement.
* Remove the relation.
* Return 200OK and null JSON.

### Create announcement file

* Get the user by token.
* Check if the announcement exists.
* Check if the user is an admin or created the announcement.
* Create the file for the database.
* Add the file to the database.
* Update the announcement’s file field to also have this entry.
* Return 201Created and the file’s JSON.

### Delete announcement file

* Get the user by token.
* Check if the announcement exists.
* Check if the user is an admin or created the announcement.
* Check if the file is in the announcement.
* Remove the file id from the announcement and update it in the database.
* Remove the file from the database.
* Return 200OK and a JSON containing either the path to the file (if this is not the last file that the announcement has) or the path to the announcement (if there are no more files).

### Has permission to get file

* Get the user by token.
* Check if the announcement exists.
* Check if the user has access to the announcement.
* Check if the file is in the announcement.
* Return 200OK and the file’s path in JSON format.

### Create answer

* Get the user by token.
* Check if the announcement exists.
* Check if the user has access to the announcement.
* Check if the announcement allows answers.
* Create the answer and put it in the database.
* Create the relation between the answer and the announcement.
* Return 201Created and the answer’s JSON.

### Delete answer

* Get the user by token.
* Check if the announcement exists.
* Check if the answer exists.
* Check if the user is an admin or the answer’s creator.
* Delete the answer from the database.
* Delete the answer – announcement relation.
* Return 200Ok and null JSON.

# Courses Manager

## Constructor

The constructor of this class will take as parameters pointes of the following classes and will store them to private variables accordingly:

* UsersCqlManager,
* TokensCqlManager,
* FilesCqlManager,
* GradesCqlManager,
* CoursesCqlManager,
* UsersByCourseCqlManager,
* CoursesByUserCqlManager,
* LecturesCqlManager,
* TagsCqlManager,
* UsersByTagCqlManager,
* QuestionsCqlManager,
* AnswersCqlManager,
* AnswersByAnnouncementOrQuestionCqlManager,
* QuestionsByCourseCqlManager

## create\_course

* Get the user from the token.
* Check if the user is an admin or a teacher.
* Build the course object.
* Add the course to the database.
* If the user is a teacher, assign him to the course.
* Return 201 and the id of the course.

## get\_course

## get\_all\_user\_courses

## ger\_courses\_users

## update\_course

## delete\_course

## set\_course\_thumbnail

## get\_course\_thumbnail

## delete\_course\_thumbnail

## create\_course\_file

## get\_course\_files

## update\_course\_files

## delete\_course\_file

## has\_permission\_to\_get\_file

## add\_users

## remove\_users

## create\_question

## get\_questions\_by\_course

## delete\_question

## create\_answer

## get\_answers

## delete\_answer

# Users manager

User cql manager

School manager

Tokens cql manager

User references cql manager

Courses by user cql manager

Users by course cql manager

Announcement manager

For POST:

* Only the admins can create users
* Get the user form the admin token
* Check that he is an admin
* Check if the school exists
* Check if the information is valid. (todo, add function to get all users from the database, must have a generated password)
* Add the user to the database

For GET:

* Only admins and related teachers and the actual user can get the student’s information
* Get the user from the token
* Check if this user is the one that we want to return or is a related teacher or admin. (TODO: to check if the teacher is related, check all the courses that the user is in and check all other users that are in that course)
* Return the user’s information depending on circumstances:
  + If we only want the user’s name and surname, return them + the id
  + If we want more information return all of the user’s data (not the password and the changed password field) plus the user’s references

For PUT:

Only the actual user can change his password/name/surname

* Get the user from the token
* Check if the target user is the one that changes
* Change the data
  + If it is the password that the user changes, also change the “changed\_password” field.

When changing user type or name/surname

* Get the user from the token
* Check if it the admin that changes the type/name/surname
* Change the data

For DELETE:

* Get the user from the token
* Check if the user deleting is an admin
* Delete the user
  + Delete all of his tokens (todo, add function to get all of user’s tokens and then delete them)
  + Get all of the tags that he is related to and delete them from the tags by user and users by tag tables
  + Delete all of the entries from user references table
  + Delete all of the entries in the todos table and todos by user table
  + Delete all of the entries in the courses by user and users by course tables
  + Delete all the announcements that the user posted and it’s related questions (if exists)
  + Delete all the questions that the user posted and it’s related questions (If exists)

## For references

For POST:

* Get the user from the token
* Check if the user adding is the target user
* Add the reference

For GET:

* Get the user from the token
* Check if the getting user is the target user or teacher/admin
* Return the reference

For DELETE:

* Get the user from the token
* Check if the user deleting is target user
* Delete the reference

# Courses Manager

User cql manager

Token cql manager

File cql manager

Grades cql manager

Course cql manager

Users by course cql manager

Courses by user cql manager

For POST:

* Get the user from the token
* Check to see if the user is an admin or a teacher
* Get all the related users from the tokens and individual users (students)
* Get the information and build the course
* Add the users into the users by course and courses by user tables
* Add the information into the courses table

For GET:

* Get the user from the token
* Get all of this user’s courses and check if the target course is in this list
* Get the courses data
* Get the files
* Return this data

For PUT:

* Get the user from the token
* Check to see if the user is a maintainer (TODO: add maintainers list of uuids to the table)
* Check to see if the data is valid
* Change the data

For DELETE:

* Get the user from the token
* Check to see if the user deleting is an admin
* Delete the relations from users by course table
* Delete the relations from courses by user table
* Delete the related files of the course
* Delete the grades from this course
* Delete the actual course

# Tag Manager

Tag cql manager

Token\_cql\_manager

User cql manager

Users by tag cql manager

Tags by user cql manager

## Tag related functions

For POST:

* Get the user from the token
* Check to see if the user is admin or teacher
* Check the data
* Add the tag

For GET (want to return all the tags):

* Get the user from the token
* Check if the user in an admin or a teacher
* Return the list of tags

For PUT:

* Get the user from the token
* Check if the user is admin or teacher
* Check the data
* Change the tag

For DELETE:

* Get the user from the token
* Check if the user is admin or teacher
* Delete the relations in tags by users and users by tag tables
* Delete the tag

## Relations related functions

For POST:

* Get the user from the token
* Check if the user is an admin or a teacher
* Check if the user and the tag exist
* Add the relation in users by tag and tags by user tables

FOR GET:

* Get the user from the token
* Check if the user in an admin or a teacher
* Check if the tag exists
* Return the list of users in short form

For DELETE:

* Get the user from the token
* Check if the user is an admin or a teacher
* Delete the relations in tags by users and users by tag tables

# Todos Manager

Users cql manager

Token cql manager

Totos by user cql manager

Todos cql manager

For POST:

* Get the user from the token
* Parse and check that the data is valid
* Create the todo
* Add the relation in the todos by user table

For GET:

* Get the user from the token
* Get all the todos that the user has

For PUT:

* Get the user from the token
* Check if the user exists
* Check if the todo is related to the user
* Get the todo and change it’s fields

For DELETE:

* Get the user from the token
* Delete the todo
* Delete the relation in the todos by user manager

# Grades Manager

Users cql manager

Token cql manager

Grades cql manager

Courses by user cql manager

Users by course cql manager

Course cql manager

TODO: add SELECT \* FROM schools.grades WHERE school = 1 and evaluated\_id = now() AND course\_id = now() ALLOW FILTERING ; option in the grades cql manager

For POST:

* Get the user from the token
* Check to see if the user is a teacher
* Check to see if the course exists
* Check to see if the teacher has access to this data (user courses by user cql manager)
* Check to see if the evaluated user exists
* Add the grade

For GET:

* Get the user from the token
* For teacher:
  + Get the user from the token
  + Get the courses that this teacher is administering
  + Get the user for each course
  + Get each grade for the users in that course that are students and calculate the final grade
  + Return the information
* For student
  + Get the courses that this user is assigned to
  + Get every grade for each individual course and store them separately
  + Calculate the grade for the whole course
  + Return the information

For POT:

* Get the user from the token
* Check if the user is the one who assigned the grade
* Modify the grade

For DELETE:

* Get the user from the token
* Check if the user is the one who assigned the grade
* Delete the grade

# Files manager

User cql manager

Tag cql manager

File cql maanger

## File related

For GET:

* Get the user from the token
* Return the file (consider the file format)

## JSON related

For GET:

* Get the user from the token
* Return
  + If the object is a folder return its contents
  + If the object is a file return the path to that file

For