CGS backend

Software Design Document

|  |  |
| --- | --- |
| Document Owner: | Micha Shlain |
| Document File: | CGS\_backend\_SDD.docx |
| Document Location: | [link](http://turtle.timetoknow.local:8080/svn/dev/t2kdev/trunk/cgs/doc/design/architecture) |

# Revisions

|  |  |  |  |
| --- | --- | --- | --- |
| Revision | Author | Description | Date |
| 0 | Micha Shlain | First Draft | 10/11/2012 |
| 1 | Ophir Barnea | High level approach | 10/17/2012 |
| 2 | Ophir Barnea | Tuneup to requirments | 11/17/2012 |

# Table of Contents

[Revisions 1](#_Toc337984884)

[Table of Contents 2](#_Toc337984885)

[1 Solution, Product Component or Feature Overview 4](#_Toc337984886)

[1.1 Definitions & Abbreviations 4](#_Toc337984887)

[1.2 Feature Description 4](#_Toc337984888)

[1.3 QA Requirements for the Feature 4](#_Toc337984889)

[1.4 Functionality Overview 4](#_Toc337984890)

[1.4.1 Behavior/Appearance 5](#_Toc337984891)

[1.4.2 User Experience and Compliance Requirements 5](#_Toc337984892)

[1.4.3 Configurability 5](#_Toc337984893)

[1.4.4 Usage and Procedures 5](#_Toc337984894)

[1.4.5 Error Handling 5](#_Toc337984895)

[1.4.6 Database Changes 5](#_Toc337984896)

[1.4.7 Internalization/Localization 5](#_Toc337984897)

[1.4.8 Performance 5](#_Toc337984898)

[1.4.9 Compatibility & Migration 5](#_Toc337984899)

[1.4.10 Supportability 6](#_Toc337984900)

[1.4.11 Portability and Platform-Specific Requirements 6](#_Toc337984901)

[1.4.12 Security 6](#_Toc337984902)

[1.4.13 Installer Changes 6](#_Toc337984903)

[1.5 Testing & Exit Criteria 7](#_Toc337984904)

[1.6 External Dependencies 7](#_Toc337984905)

[1.7 Assumptions 7](#_Toc337984906)

[2 Architecture 7](#_Toc337984907)

[2.1 Architectural Block Diagram 7](#_Toc337984908)

[3 Solution, Product Component or Feature Design 8](#_Toc337984909)

[3.1 Design Overview 10](#_Toc337984910)

[3.1.1 Class Diagrams 10](#_Toc337984911)

[3.2 [Application Name – e.g. CGS frontend, CGS backend, LMS frontend, LMS backend, CMS, LMS Admin etc.] 10](#_Toc337984912)

[3.2.1 Class Diagrams 10](#_Toc337984913)

[3.2.2 [Use Case 1] **Error! Bookmark not defined.**](#_Toc337984914)

[3.2.3 [Use Case 2] **Error! Bookmark not defined.**](#_Toc337984915)

[3.3 [Application Name – e.g. CGS frontend, CGS backend, LMS frontend, LMS backend, CMS, LMS Admin etc.] 11](#_Toc337984916)

[3.3.1 Class Diagrams 11](#_Toc337984917)

[3.3.2 [Use Case 1] 11](#_Toc337984918)

[3.3.3 [Use Case 2] 11](#_Toc337984919)

[4 Traceability to Requirements 11](#_Toc337984920)

[5 Open Issues and Future Enhancements 12](#_Toc337984921)

[6 Risks 12](#_Toc337984922)

[6.1 Impact 12](#_Toc337984923)

[6.2 External Risks 12](#_Toc337984924)

[7 References 12](#_Toc337984925)

# Solution, Product Component or Feature Overview

## Definitions & Abbreviations

|  |  |
| --- | --- |
| CGS | Content Generation Studio |
| TOC | Table of Content |
| LDAP | Lightweight directory Access Protocol |
| Active Directory | Microsoft implementation of the LDAP protocol |
| SpEL | Spring Expression Language |
| REST | Representational state transfer |

## Feature Description

The CGS backend is a RESTful web-server which provides a set of API’s to support the Content edting process in collaboration with the CGS client.

The server supplies persistency services for the book elements and manages security, revisions , assets , permissions , validations and packaging.

The CGS backend, is also responsible for managing the publisher accounts which is the contextual data which the editor can work with in order to edit the course.

Since the course contents might reside on local editors device (eg PC), the server provides a control API to manage the content-element state to support multi users on a shared course editing.

## QA Requirements for the Feature

<Describe any requirements needed by QA to test this feature (e.g. backdoors, special log entries etc.).>

## Functionality Overview

The server provides the following services to the CGS client:

* Security: The Content author must be permitted to access (read/write) the book contents.

The server provides an authentication service to an LDAP server and validates each client REST requestssss.

* Persistency: Course contents and Publisher accounts are stored in a designated database. The server support a full CRUD operations for the course elemets.
* Versioning : The server is responsibe to manage the versions logics of the course and lessons.
* Locking: To alow concurrent authoring on shared course elements, the server provides a locking mechanism and permissions.
* Validations: Validates the books contents against a schema in order to provide data integrity and structure correctness.
* Assets : The server allows the client to manage assets (store and retrieve) in reference to a book.
* Packaging: On request , the server wraps the book contents to a well known format and publishes it to a catalogue (currently to the local file system as zip file).

### Behavior/Appearance

<Include the details about behavior/appearance and screen mockups (if applicable).>

### User Experience and Compliance Requirements

<Describe in this section any design-time or run-time user experience requirements including 508 compliance, RTL support, Keyboard shortcuts, tool tips etc.>

### Configurability

For runtime purposes, the server has a set of configurations files with a single API. The configurations give the server information regaring DB & LDAP access etc.

### Usage and Procedures

<Describe any special procedures needed to run the component.>

### Error Handling

The server modules are designed to handle errors. Error messages are logged to a local file appender (log4j) and also submitted to a logSys server.

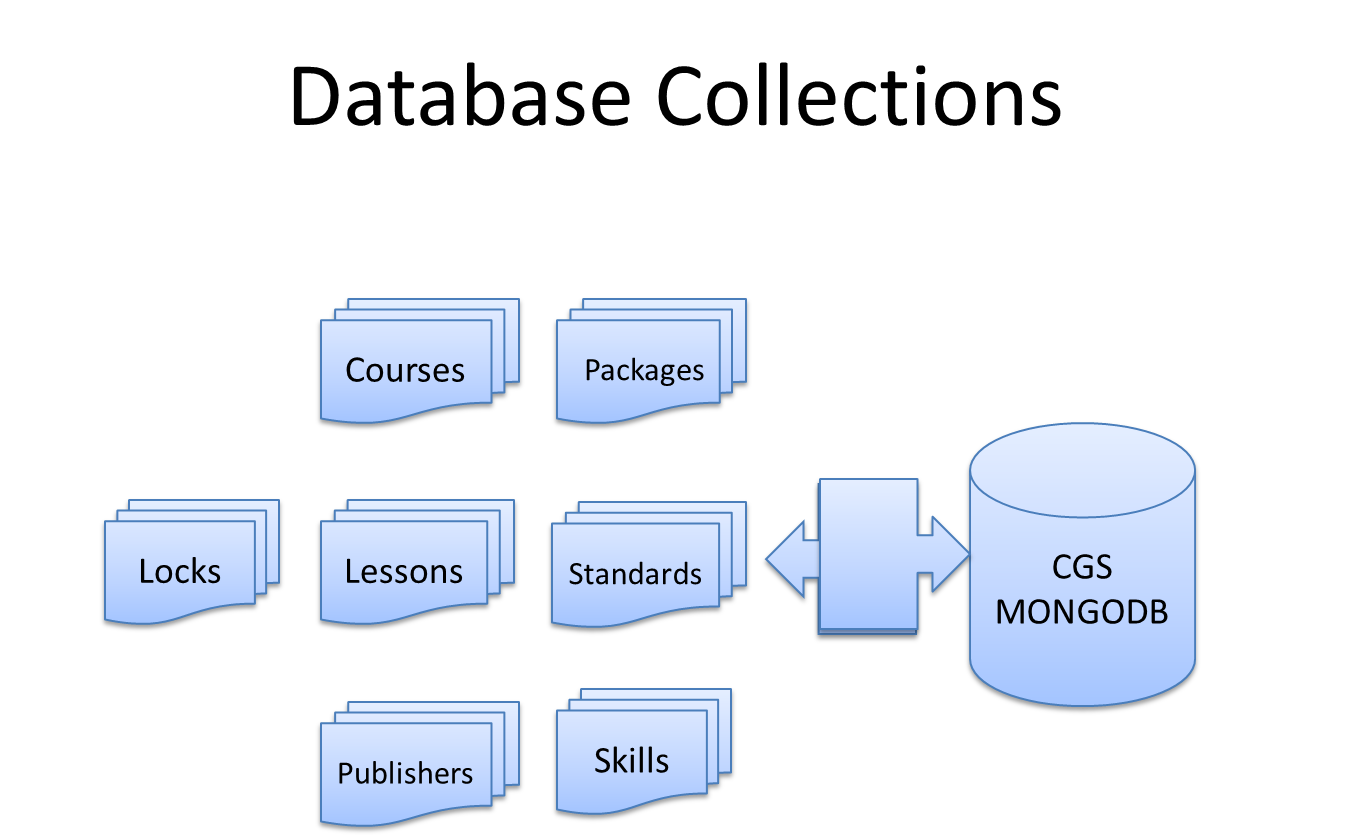
Bussiness errors like resourseNotFound or locks confilicts will be returned as HTTP will relevant status (like 404,423 etc..)

Server Errors (of type 500 ) will be exposed to the client with the context description.

### Database Changes

The CGS server uses MongoDB for pesistance of business documents .

DB schema is ‘CGS’ and it holds the following collections (like tables it relational db)



### Internalization/Localization

<Mention any special support needed for internationalization/localization.  
 e.g. encoding issues, date/time/number/currency formatting or style, LTR and RTL language special needs, support formatting of multiple languages in the same web page etc.>

### Performance

Backend internal services should be with a high perfomace to achieve a comfort user experience. The server handles all requests in a synchronized manner; access to resources (DB , file server) are indexed.

The server impact on the client experience , is bases on the contents size requested by the client.

### Compatibility & Migration

#### Backward Compatibility

No backward compatability will be developed.

#### Platform Compatibility

* OS: Windows 32/64 bit, Linux (Red Hat distribution) 32/64 bit

#### Migration/Upgrade Requirements

N/A

### Supportability

The server logs all loggable information into a designated log file (Log4j) and also sends log messages to a SysLog server (udp).

### Portability and Platform-Specific Requirements

### Security

#### Overview

The security component will be based on the [Spring Security framework](http://static.springsource.org/spring-security/site/). This framework provides an easy integration with LDAP services and we will use the T2K Active Directory for authentication and authorization.

#### Authentication

Most of the authentication details will be handled by Spring Security framework and only a small number of changes will be made by us.

A custom *CGSUserDetails* will be used to decorate the basic *UserDetails* provided by the framework to extract extra information from the Active Directory (information such as: first name, last name, e-mail address).

In order to populate the custom *CGSUserDetails* Object we will decorate another existing framework class called *LdapUserDetailsMapper.*  *CGSUserDetailsMapper* decorator will populate the additional information we decided to have in the *CGSUserDetails* object.  
  
A logged in user will only search the Active directory for his information. This removes the need for a “Special” user of the Active Directory with permissions to search the entire Active Directory.

#### Authorization

To provide simple and elegant approach for authorization Spring Security framework provides a *PreAuthorize* annotation that enables a simple use of SpEL for specifying condition for which a user can access resources in the system (resources can be assets, REST APIs and more).  
  
To avoid repetition of SpEL coding in every class, custom annotation will be created to replace the use of *PreAuthorize.* The new annotation will be *AllowedForContentDeveloper* and *AllowedForAdmin* and will already containn the SpEL conditions that are required for each role.

To provide distinction between a content developer and an admin we will add new authorities to the Active directory (these authorities will be loaded into the *CGSUserDetails* object at login).

#### Login page

Spring Security Framework contains a customizable login screen that a user will be automatically redirected to when trying to access any resource that needs authentication.

We will customize this page to have the required UI look.

### Installer Changes

<Describe changes to installation scripts.>

## Testing & Exit Criteria

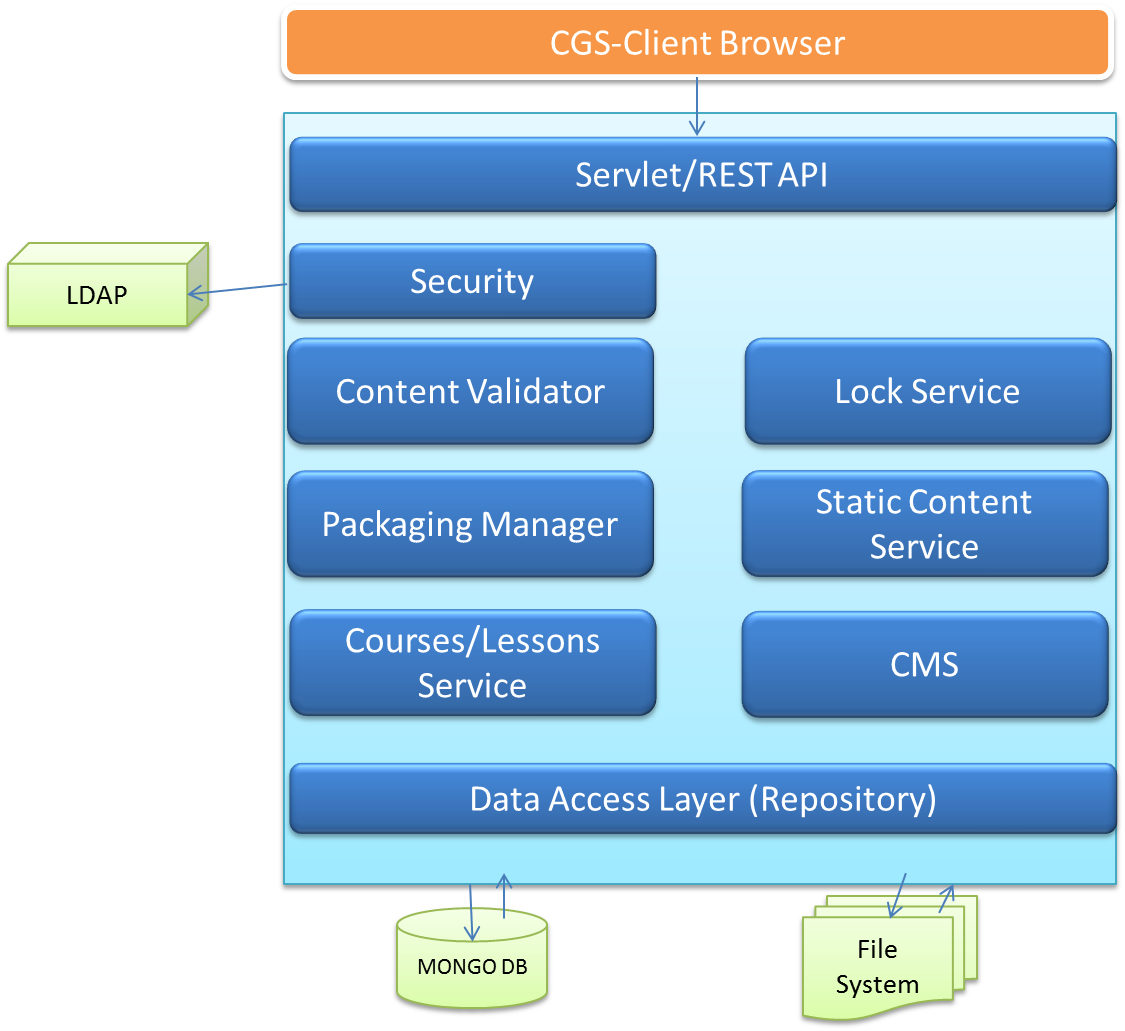
<Describe briefly the Unit test plan for the component. If the component may requires any special kind of testing then list that. Also list the high level exception scenarios for the component and exit criteria.>

## External Dependencies

The CGS server is a web server, deployed on Tomcat 7 runs with Java7. It uses a MongoDB ver2.2 for the documents persistence (lessons and courses).

# Architecture

## Architectural Block Diagram



# Solution, Product Component or Feature Design

This section describes the CGS main modules. it provides an overview on the functionalities and on the selected techonologies.

## WS-REST API

The CGS Client communicates with the server using a WS RESTful protocol

The CGS REST API exposes all the needed services to the client; as a web controller layer, it receives the requests and dispatches them to the relevant services for further handling.

CGS uses Spring-MVC as the REST controller and provides the following services groups:

Courses API: service for get , create and update of a course.

Lessons API: service for get , create and update of a lesson.

CMS API: service for upload and retrieve assets.

Locks API: service for acquire and release locks on book elements

Static Content API: manages access to publisher account data like standards and skills

## Locks Service

The Lock service is responsible for controlling the locks on content elements by the users.

When the user request a lock on an object (lesson or course), the lock service retrieves the current state of the object from the database and checks if lock is valid and updates as needed.

Only one user can maintain a lock on an item.

When a user wants to release a lock on an item, the permission manager checks that the user is the owner of the lock and releases it.

## VersioningManager

The VersioningManager is responsible to manipulate the courses and lessons versions.

The versions can be modified by request or automatically.

The manager holds the logics of controlling the content versions, revisions and editions.

(For example, on lesson save - the manager checks the last modifier - if different user, it will increase the lesson version)

## Content Validator

The validation service maintains a validation on the course/lesson components. This operation assures that the authoring process is valid in term of data and formation.

The validation process comes to avoid problems on the packaging stage and therefore to alert the user of issues on the fly (when saving course or lesson)

The validator will use a json schema.

## Packaging Manager

This service is responsible for creating a complete course package for the catalogue.

The process collects all course resources , checks for validation , integrations and cleans up unnessasary fields from the contens. Finally all contents are packed to a zip file .

## CourseService

This is the main service to access course contents; provides API to get courses , save and update.

it uses mongoDB API to achieve access to the data.

## LessonService

This is the main service to access lessons contents; provides API to get lessons , save and update.

it uses mongoDB API to achieve access to the data

## CMS AssetsService

This is the main service to access assets; provides API to save and retrieve assets

it uses file system repository API to achieve access to the data.

## Static Content Service

When a user uses the CGS client , all the relevant information regarding the publisher contextual data should be available. On login , the client request for the publisher account , using the user details, the service finds the mapped account object and returns to the client .

Currently, there is only one generic default account in the system for all users.

# Design Overview

Contents objects are reflected as decorators for both Course and Lesson.

The JSON contents are wrapped by the CGS objects to supply a strict access to some fields on the JSON documents.

The ContentItem interface assumes that the content has pre-defined fields, like header , toc etc..

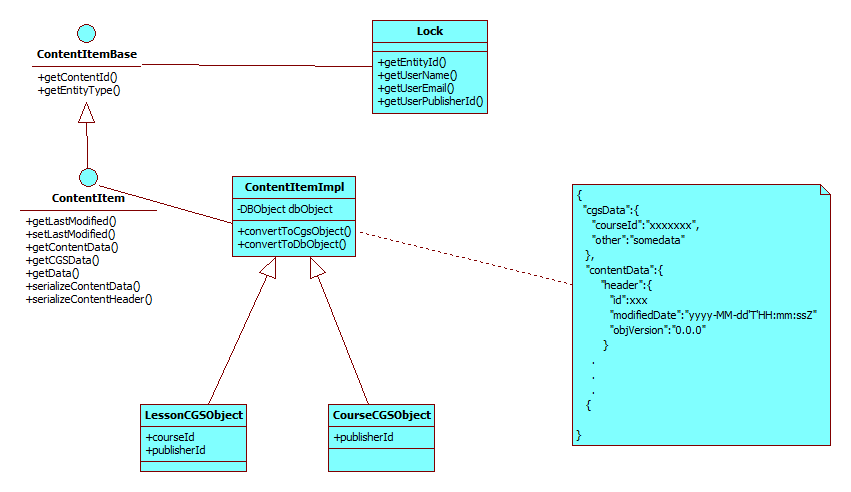
The CGS objects implemets interface to read/write those values.

On Saving to Mongo DB, the wapper reflected on the saved object (DBObject).

When client retrieves contents , only the Content field (Document) is exposed.

This way, the server can manage the contents, add references for lookups and tags for modification time.

### Class Diagram (Content Model)



## CGS backend

### getPublisherCoursesProperties

#### Description

When an editor wants to open a book, he first checks all available books that are related to the publisher.

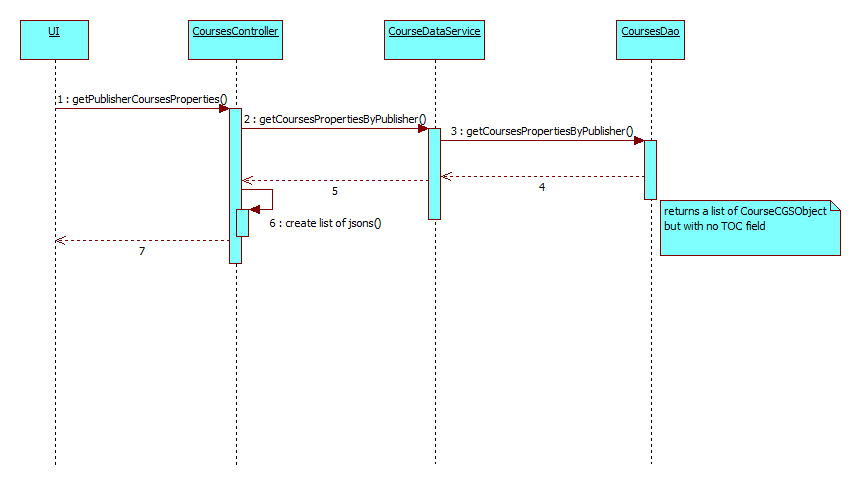
The client will receive a list of course properties.

Course properties is a the course document without the “toc” field.

The API uses the booksService to retrieve all the courseProperties by publisher ID.

Returns a list of course JSONs.

#### [Use case 1] Sequence Diagrams



### getCourse

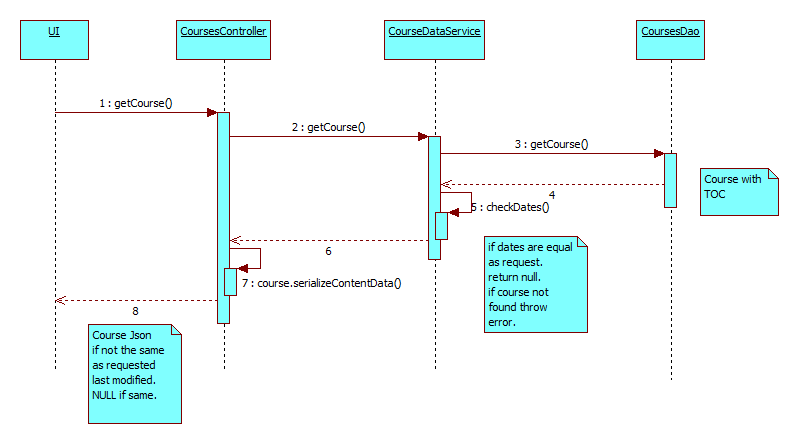
#### Description

This use case describes a situation when the user wants to get a complete course document.

The clients sends a request with the course Id (and publisherId), the service looks for the Course in the data store and returns it.

In case the user also supplies a ‘lastModified’ value, the server will check if the given value is the same as the stored copy value. If they are different , the service will return the stored instance, otherwise will return NULL which means that the client has a synchronized copy of the course.

#### [Use case 2] Sequence Diagrams



### saveCourse

#### Description

in this case, the user wants to save his current course, the client Posts the course document and the server stores the document in the database.

The save process can only accure if the user is the owner of the course lock, Therefore the server will check that the user is the lock owner. If the course is new , the server will accept empty lock.

In any other case, the server checks that the user is the actual locker.

When conficts of locks accures , the server sends an HTTP LOCKED(423, "Locked").

Saving a book also changes the ‘lastModified’ indicator in the course header.

The course header is returned to the client in case of success HTTP 200 OK.

A header json sample :

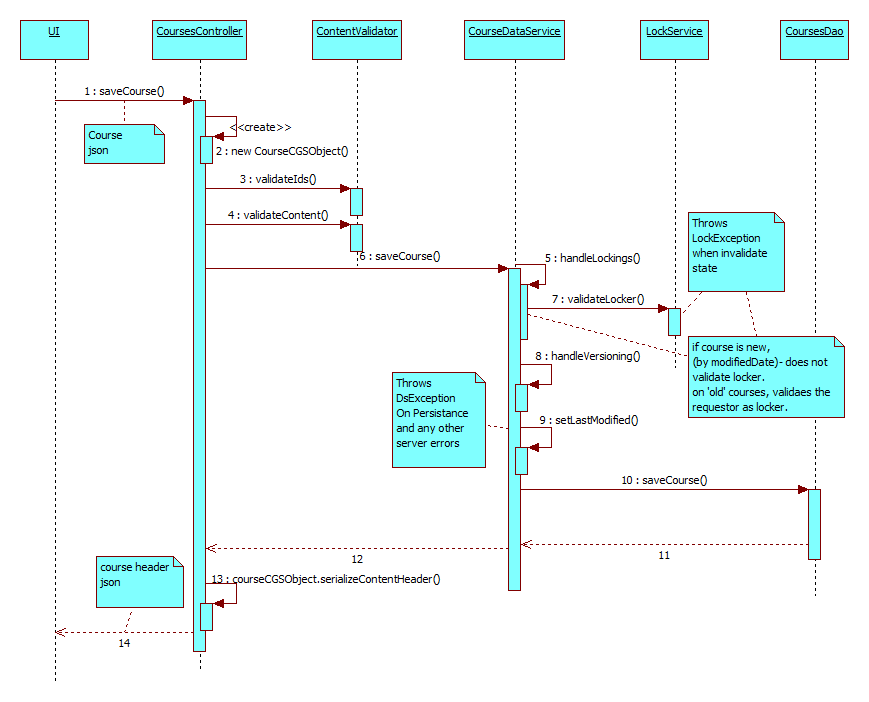
{ "id" : "69bb10b0-26a6-11e2-81c1-0800200c9a61"

, "lastModified" : { "$date" : "2012-11-13T14:00:40.279Z"} ,

"objVersion" : "0.0.0"

}

#### [Use case 3] Sequence Diagrams



### LockCourse

#### Description

When the the editor wants to edit a course content , the client send a lock acquire request,

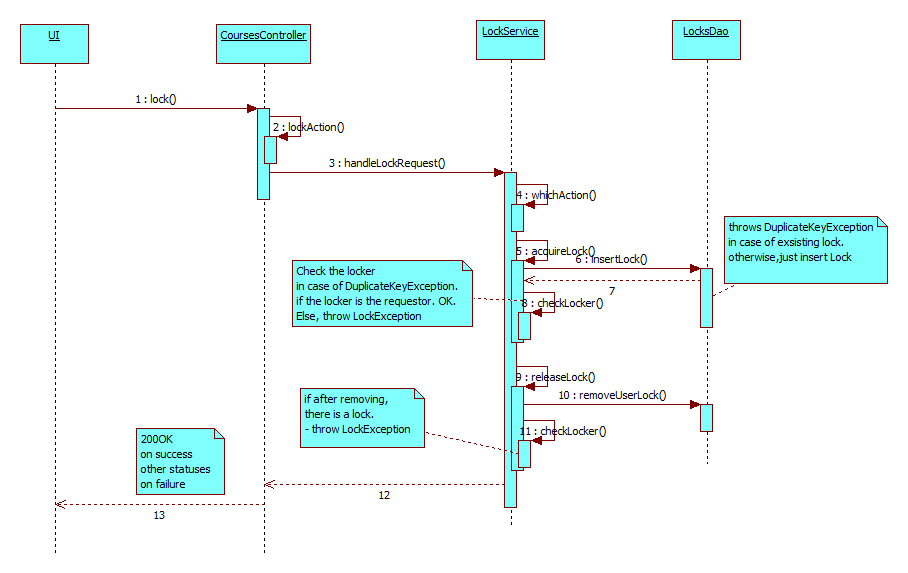
The sever checks if the request is valid , and performs a lock on the given course.

When the editor ends to work on the course, the client sends a lock release request.

The server checks validity of the request (eg. is the requestor owns the lock) and performs a release.

If conflicts accures , the server sends an HTTP LOCKED(423, "Locked") else 200-OK.

#### [Use case 3] Sequence Diagrams



### getCourseLock

#### Description

Returns the lock object of the course by Id.

If there is a lock on the given course, the method will return a json lock document, with the lock details (like owner, email, and date) . if there is no lock on the course , returns NULL.

Lock doc sample

{

"entityId": "69bb10b0-26a6-11e2-81c1-0800200c9a99",

"entityType": "COURSE",

"userName": "ophir.barnea",

"userEmail": "firstName.Last@timetoknow.com",

"userPublisherId": 1,

"lockDate": 1352968092852

}

### getLesson

#### Description

This use case describes a situation when the user wants to get a complete lesson document.

The clients sends a request with the lesson Id (anpublisherId), the service looks for the Course in the data store and returns it.

In case the user also supplies a ‘lastModified’ value, the server will check if the given value is the same as the stored copy value. If they are different , the service will return the stored instance, otherwise will return NULL which means that the client has a synchronized copy of the course.

### getLessonsProperties

#### Description

A method to get all lessons that are different from the instances on the client side.

The API receives a list of attributes were each is a lessonId and LastModified (and version).

The server checks if the client lesson lastModified value is different than the one in the server.

Each different lesson is added to the returned list.

NOTE: the lessons documents dcontain the ‘learningActivities’ field.

#### [Use case 1] Description

#### 

## [Application Name – e.g. CGS frontend, CGS backend, LMS frontend, LMS backend, CMS, LMS Admin etc.]

### Class Diagrams

### [Use Case 1]

#### [Use case 1] Description

#### [Use case 1] Sequence Diagrams

### [Use Case 2]

#### [Use case 2] Description

#### [Use case 2] Sequence Diagrams

# Traceability to Requirements

| # | Requirement Description | Reference to PRD | Release |
| --- | --- | --- | --- |
| 1 |  |  |  |
| 2 |  |  |  |
| 3 |  |  |  |

# Open Issues and Future Enhancements

<List any open issues and also list any possible future enhancements.>

# Risks

## Impact

<What is the impact of this change to external components or teams: RSP, LMS, ADMIN, CMS, CGS etc.>

## External Risks

<Any external risks which may prevent this feature from working properly.>

# References