VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

David Lundy

A Teaching Tool for the IEC 61850 Substation Configuration Language

Moodle Integration for Energy Technology ICT

Information Technology

2015

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	David Lundy
Title	A Teaching Tool for IEC 61850 Substation Configuration Language
Year	2015
Language	English
Pages	66 Pages
Name of Supervisor	Smail Menani

This thesis work was made in cooperation with Dr. Smail Menani and Vaasan Ammattikorkeakoulu. The main purpose of this thesis was to create a teaching tool that would provide students of the Energy Technology ICT course with online tasks that could demonstrate an understanding of IEC 61850 Substation Configuration Language.

The development of the thesis work involved research into the integration interfaces allowing an external web application to provide content to the Moodle Learning Management System, and into OAuth security signing. It also provided an opportunity to research and apply new developments in standalone web application technology on the JVM using Spring Boot.

The result of the thesis project is a stand-alone web application that acts as an LTI 1.1 Tool Provider for Moodle and other learning management systems that enables instructors to provide tangible learning experiences for students learning the IEC 61850 Substation Configuration Language.

Keywords IEC61850, Moodle, LTI, Teaching Tools

CONTENTS

LIST OF FIGURES AND TABLES

ABBREVIATIONS

AOP	Aspect Oriented Programming
API	Application Programming Interface
BSD	Berkeley Software Distribution
CSS	Cascading Style Sheets
GPL	GNU Public License
HTML	Hypertext Transport Markup Language
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IED	Intelligent Electronic Device
JAR	Java Archive
JPA	Java Persistence API
JPQL	Java Persistence Query Language
JSON	Javascript Object Notation
JVM	Java Virtual Machine
LGPL	Lesser GNU Public License
LMS	Learning Management System
MPL	Mozilla Public License
MVC	Model View Controller
NIO	Nonblocking Input Output

OAUTH	Open Authentication
POM	Project Object Model
RDBMS	Relational Database Management System
SCL	Substation Configuration Language
SQL	Structured Query Language
URL	Uniform Resource Locator
WAR	Web Archive
XML	Extensible Markup Language
XSD	XML Schema Definition

1 INTRODUCTION

Energy Technology ICT is a Professional Basic Studies course in the Bachelor of Engineering – Information Technology degree at Vaasa University of Applied Sciences.

IEC 61850 is a standard for Substation Automation design that focuses on modeling the information available from different primary equipment and from the substation automation functions, specifying the communication between the IEDs of the substation automation, and on defining a configuration language used to exchange configuration information. (Zhang and Gunter 2011, 8)

The primary objective of the Energy Technology ICT course is to provide students with an understanding of data communication in the field of power distribution and transmission networks, with focus on the IEC 61850 Standard and the Substation Configuration Language (VAMK Curricula, 2014). Such understanding is essential to providing value to local industries which focus on the energy business and developing marketable skill growth for students entering professional life.

However, reaching this objective from a pedagogical standpoint has proved challenging with regard to providing the students with tasks which can measure their understanding of theoretical concepts as taught from the textbook and other lecture driven sources.

This thesis work has the goal of providing such tools to the Energy Technology ICT course to aid the course instructors in challenging students to learn with hands on application of their studies and measure their aptitude in a concrete and specific manner.

2 RELEVANT TECHNOLOGIES

The project solution was written in Java, leveraging Spring Projects to provide dependency injection, embedded http server, dynamic interface driven persistence support, and an MVC web application pattern on the Tool Producer server code. Application data is persisted to MySQL RDBMS. The jQuery library was used to supplement the client side Javascript code which integrates the jQuery.xmleditor and CKEditor components and communicates user input back to the server.

2.1 Java 7

Java SE 7 was a major release of the Java language and class libary in 2011. The primary language features introduced were the new diamond operator for instantiation of generic types, using strings in switch statements, automatic resource disposal in try statements, the new file system API (NIO 2.0), and support for dynamic language implementation on the JVM via the new java.lang.invoke package.

2.2 Spring Boot

Spring Boot is a project that enables building stand alone web applications that do not carry a dependency on an application server or an external servlet container. Web applications built with Spring Boot carry an embedded version of Tomcat or Jetty and produce a single runnable java archive (JAR). Spring Boot does also provide configuration options for generating a web application archive (WAR) that may be deployed to a lightweight servlet container or application server.

A key innovative feature of Spring Boot is that of automatic detection of dependencies and default configurations by convention which minimizes the amount of configuration required to build and deploy a web application. When the provided default configurations are unsuitable for use, override is possible by either providing an implementation of a base class or interface and registering with the application context or via specification in a configuration file.

Spring Boot was first released with 1.0.0.RELEASE in March of 2014, and at the time of this project was on version 1.2.2.RELEASE.

2.3 Spring Data JPA

Spring Data is a sub-project of Spring Data, which is a parent project containing sub-projects which enable easier use of databases and data access technologies.

The Spring Data JPA project simplifies the implementation of application data layers by eliminating the need for repetitive boilerplate code to implement simple queries and functionality such as pagination and auditing. By inspecting classes annotated with JPA annotations Spring Data JPA can provide full default repository implementations complete with a DSL specific to your entity classes for query generation by simply declaring an interface of the desired type.

Spring Data JPA version 1.0.0.RELEASE was first introduced in July of 2011, with the current version at the time of this project being 1.7.2.RELEASE.

2.4 Spring Framework

Spring Framework is an open source programming model for building enterprise applications. Key features are dependency injection, transaction management, support for aspect oriented programming (AOP), an implementation of request based Model View Controller pattern, web service and webservice client support, and integrated support for object relational mappers. An overview of the Spring Framework runtime is shown below in Figure 1.

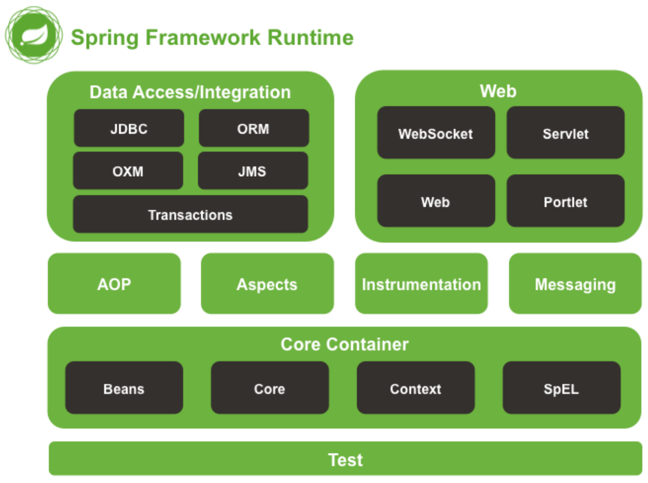


Figure 1. Spring Framework Overview (Pivotal, 2014).

One advantage of building applications with Spring Framework is that a full Java Enterprise Edition capable application server is not necessary as a deployment and runtime dependency. Spring Framework was initially introduced in 2005 and at the time of this project is currently on version 4.1.5.RELEASE.

2.5 Flyway

Flyway is a database migration tool that can be embedded in an application archive. The tool is configured with the destination schema, and creates its own version table in the schema in which it stores a list of changes to the database schema and or data and a checksum of each script that is run. The schema migrator works by reserving a directory in the project structure for sql files named according to a specified convention.

On every application startup, the Flyway tool will query the database version table and compare the entries and checksums with the files in the reserved migration directory. Should there be migrations that have not yet been applied to the database, Flyway will execute the missing migrations to bring the database up to current state.

The benefit of such a tool is that the application and database states can be synchonously updated in deployment without consideration for executing database scripts in downtime, and deployment of applications can be a much more streamlined and automated process.

2.6 Freemarker Templating Engine

Freemarker is a Java class library which focuses on merging Java objects with a defined template file to produce a text output. While Freemarker can be used to generate any format of text output given a suitable text template and object, the primary focus of the library is to provide a practical and efficient engine for generating HTML pages in an MVC architecture. The current version of the Freemarker library is 2.3.20 and it is provided under the BSD license.

2.7 Moodle

Moodle is a configurable open source all-in-one learning platform that is used by tens of thousands of learning environments around the world. While it is backed by Moodle HQ, an Australian company with a dedicated team of developers financially supported by Moodle Partners (Moodle 2015.), the project is developed in an open source model with hundreds of international contributors. Moodle is implemented in PHP language and at the time of this project was on Moodle version 2.8.

2.8 LTI 1.1 Specification

Learning Tools Interoperability (LTI) is a specification backed by IMS Global Learning Consortium as a ”standard way of integrating rich learning applications with platforms like learning management systems, portals, or other educational environments.” (IMS Global 2015: Developers)

The LTI specification is implemented by both the learning management system, called the Tool Consumer, and the external tool, which is called the Tool Producer. Competing implementations of Learning Management Systems include Blackboard, Moodle, Desire2Learn, Sakai, Coursera and Instructure.

LTI 1.0 was initially released in 2010 providing basic access to external tools from learning management systems, with an update following in 2012 to LTI 1.1 which provided basic support for returning a grade back from the external tool to Moodle.

An LTI launch diagram shown below in Figure 2 shows the launch process. The user visiting Moodle in their browser clicks on a link to a resource configured to launch a Tool Provider. Moodle prepares a set of launch parameters and securely signs them using OAuth and then sends them back to the users browser, where Javascript code in the Moodle page the user is on will POST those parameters to the Tool Provider, which will, if the parameters are validated, deliver the content to the users browser to be embedded in the Moodle page.

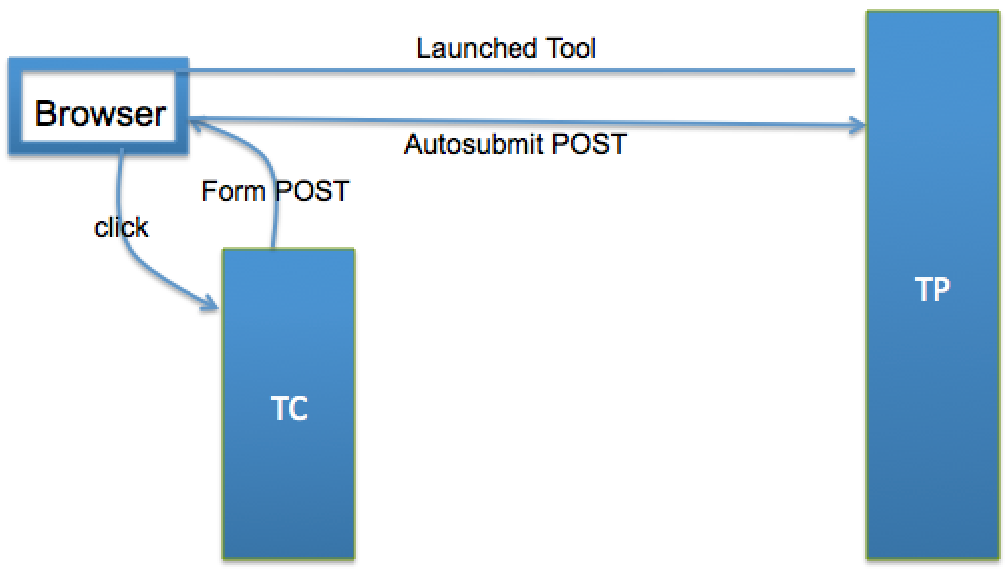


Figure 2. LTI Communication (IMSGlobal, 2015)

The LTI 1.2 specification update provides for a smoother transition to LTI v2.0, with some existing optional parameters being upgraded to recommended status, and profiles for tool consumers which will allow dynamic discovery of functionality by Tool Providers.

LTI 1.2 technically supports LTI Outcomes Management version 2.0, which extends the basic outcomes, i.e. grades, but is being documented seperately by the specification which means that a Tool Consumer or Learning Management System can be certified as LTI 1.2 compliant without actually implementing the basic outcomes extensions.

The 1.2 specification was finalized along with a revision to the v2 specification in early January of 2015, following the public draft of the LTI Outcomes Management v2.0 on December 14th of 2014.

2.9 OAuth 1.0

OAuth is an open standard for authorization which specifies a manner of accessing secured resources or assets without sharing login information. OAuth 1.0 has several authetnication mechanisms which may be used to secure access, including a *two-legged* mechanism which generates an additional prompt for users to log in. This approach is used frequently on internet sites which provide the option to log in with your Facebook or other social account credentials.

Another approach, shown in Figure 3, is a *zero-legged* or *signed fetch* mechanism where an application that has already authenticated the user builds a request and signs it using a cryptographic hash, which it submits to a service which will authenticate the request and provide access to the restricted resources.

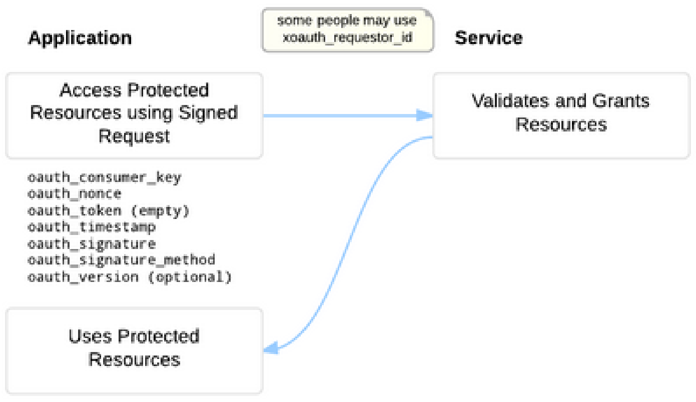


Figure 3. Zero Leg OAuth Process.

2.10 jQuery

jQuery is a powerful open source Javascript library which provides a useful and necessary abstraction layer over the inconsistent Javascript and CSS rendering engines of the various web browsers. It also provides straightforward implementations of document object model manipulation and asynchronous javascript data exchanges with servers.

2.11 CKEditor

CKEditor is a highly configurable open source *What You See Is What You Get* Javascript text editor component that can be embedded in an html page to provide word processor like features in line with page content.

It is provided under the GPL, LGPL, and MPL open source licenses, and at the time of this project was on version 4.4.7.

2.12 jQuery.XMLEditor

jQuery.XMLEditor is an open source web browser based XML editor. It is a Javascript component which provides a graphical tool for creating or modifying XML documents from inside the web browser. In the right hand pane of the tool a list of current options for next elements is provided given the current location in the structure, which is represented visually in the main pane to the left of the schema pane.

In the default configuration of the component there is a mode selector at the top of the menu bar that allows switching between the visual XML representation view and a view populated by a rich text editor which shows the raw XML structure that has been built in the visual representation view. This feature has been disabled due to an incompatibility with the integration of this 3rd party component and the CKEditor used on the same page.

It is provided under the Apache 2.0 license and is released unversioned from the project Github repository.

2.13 XMLUnit

XMLUnit is a Java class library which allows semantic comparison of XML data structures. A semantic comparison evaluates the meaning of the structure instead of only the exact duplication. This is important because whitespace and element ordering are often irrelevant when comparing structures. While XMLUnit is developed with unit testing in mind, we will be using it at runtime to compare the XML structures created by the students with the XML ’key’ structures created by the course professor to determine whether they are correct or incorrect.

The XMLUnit class library is provided under the Apache 2.0 license and is currently nearly v2.0 release. However, in this project we are using version 1.6.

2.14 JsDiffLib

JsDiffLib is a Javascript library which shows a user friendly difference between two texts, and contains a built in beautification function that can display HTML or XML structures in a nicely formatted structure instead of one long string.

It is released unversioned on Github (https://github.com/cemerick/jsdifflib) under the BSD License.

3 PROCESSES AND TOOLS

3.1 Development Process

Due to the Moodle integration issues being the largest unknown of the project work the adopted development process was executed in a way to get up and running as quick as possible with Moodle. This approach allowed us to take a more iterative a feature focused development plan once that integration was completed.

3.1.1 Risk Management

From the beginning of this project it was recognized that there are quite many components that needed to be integrated smoothly to achieve the final project deliverable. To manage this risk the functional specification of the application has defined a core feature set which must be completed to deliver the project, with remaining *nice to have* features that may be planned for future work.

3.1.2 Architectural Philosophy

The application structure was designed in a flattened 3 tier structure.

As with a traditional 3-layer application structure we have a domain, service, and presentation layer. The domain layer contains the objects which represent the model our system is concerned with manipulating and encapsulated business logic. The service layer contains the repository interfaces which manage persisting objects in the domain which require it and scheduled services which perform periodic tasks in the system. The presentation layer contains our Spring MVC web application, which depends on the service layer for persistence concerns and on the domain layer to execute business logic.

Unlike a traditional 3-layer application, a hard division does not exist between layers and all three tiers exist in the same root package. In larger systems with many developers working on the same code base this approach is worth the extra time investment to ensure that the application does not become ”dependency entangled” as it grows and thus become hard to maintain and extend. With that risk being minimal for this project we avoid that extra configuration and project management overhead and ”flatten” the tiers.

We do, however, respect the logical boundaries of the tiers; i.e. domain classes may not reference service classes or presentation classes, and service classes may not reference presentation layer classes. All dependencies must point down through the tiers and codependencies are not allowed across tier boundaries.

3.1.3 Code Organization

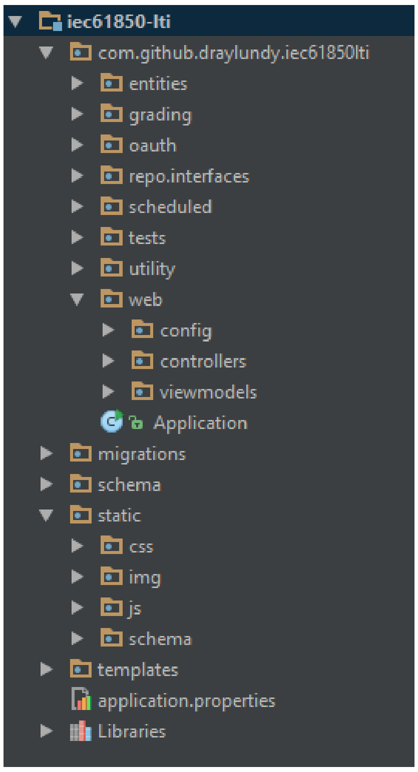
As previously discussed in Architectural Philosophy, the project was organized into a flattened 3-layer model, discussed in detail below, with leaf level package names in *italics*, and an image of the code layout later in the discussion.

Our domain layer consists of the *entity* classes which must be persisted to the application database and encapsulate domain logic, *grading* business logic, classes containing *oauth* related logic, and *utility* classes which provide common validation logic and enumerations useful for strong-typing string based comparisons.

The service layer consists of the *repo.interfaces* package which contains the repository interfaces that Spring Data JPA will provide default implementations for and the *scheduled* package containing the services which periodically clean the Nonce table in the database and report available grades back to Moodle. It also includes one class in the *oauth* package which validates OAuth requests.

The presentation layer consists of the *web* package which contains our Spring MVC web application consisting of configuration classes, controllers, view models which will be used to transfer data to the view rendering engine, and the Application main which bootstraps the application and the Spring context.

This code structure layout is shown visually in Figure 4, presented on the following page.



```
▼ iec61850-lti
  ▼ com.github.draylundy.iec61850lti
    ▶ entities
    ▶ grading
    ▶ oauth
    ▶ repo.interfaces
    ▶ scheduled
    ▶ tests
    ▶ utility
    ▼ web
      ▶ config
      ▶ controllers
      ▶ viewmodels
      Application
  ▶ migrations
  ▶ schema
  ▼ static
    ▶ css
    ▶ img
    ▶ js
    ▶ schema
  ▶ templates
    application.properties
  ▶ Libraries
```

Figure 4. Code Organization.

Also included in the presentation layer are the *static* and *templates* directories in the root namespace. Spring Boot MVC applications can serve static content from several locations, but to provide the flexibility of also being able to build as a WAR instead of only as a runnable JAR, it was necessary to use the *static* folder in the root package. By the same token the project's dynamic templates exist in the *templates* directory. These templates are combined with the view models on the server side to generate the views that will be injected back into the Moodle IFrame.

3.1.4 Convention Over Configuration

Spring Boot was chosen as the platform for the ease and quickness with which one can get up and running quickly with a basic web application where the behavior is configured with sane defaults. In aspects where we wish to change those configurations we may do so but where we do not wish to change those sane defaults we save quite a bit of man hours that would have been wasted on manually setting up these configurations.

3.2 Tools

3.2.1 Moodle 2.8 Development Installation

It was essential that we be able to test the development of our LTI Tool Provider with an LTI Tool Consumer, and since Moodle is the target LMS system it made sense to install that latest version of Moodle on the development machine.

This also enables the creation of a guide for user administrators guide to setting up the Tool Provider and configuring access that can be selected by the course professor or system administrator. In addition, black box acceptance testing was conducted from the use cases described using the Moodle web interface only. While access directly to the tool Provider was possible in the development environment, this is removed for Production use.

3.2.2 jQuery.XMLEditor xsd2json

jQuery.XMLEditor xsd2json is a utility that builds a Javascript Object Notation (JSON) representation of an XML Schema (XSD). This utility was required to create JSON representations of the Substation Configuration Language schema which provide the structure samples for the jQuery.XMLEditor component.

While support exists to transform these on demand, the Substation Configuration Language schema is not constantly changing so it made little sense to leverage this feature when we could generate the JSON representations once and serve them as static resources in the Tool Provider web application.

3.2.3 Git / Github

Git is a distributed version control system which enables workflows that preserve development effort and helps manage the complexity of changes to a single file by different feature set implementations or bug fixes that must be executed in parallel.

Git was used as the version control system for this solution, with GitHub public hosting being used as the repository provider. Interactions with the public repository were conducted through the git plugins provided in the IntelliJ Idea development IDE.

3.2.4 Fiddler

Fiddler is a free web debugging proxy that allow intercepting and monitoring HTTP traffic on the local machine, and composing HTTP POSTs at a low level outside the application for quick iterative testing of the interface without requiring changes to the application code and repeated functional testing.

The Fiddler debugging proxy was also vital in the project for debugging communication level errors between Moodle and the Tool Provider undergoing implementation, and exposing the raw HTTP POST and GET requests between the systems to provide insight to the API when the documentation was not completely clear.

3.2.5 IntelliJ Idea 14

IntelliJ Idea is the most efficient and sophisticated integrated development environment for building solutions on the JVM. It supports Java, Scala, Clojure, PHP, Python, Ruby and Groovy out of the box, along with integrated support for Enterprise frameworks, version control, database tools, cloud service management, Java EE application servers, and many productivity boosting features like advanced refactoring and smart code completion with automated code analysis.

While a commercial product, it is free to open source projects and student developers who register with an academic e-mail address. The current release is version 14.1.

3.3 MySQL Workbench

MySQL Workbench is a graphical database design and administration tool. In this project it was used to both view the structure of data in the Moodle database and explore possibilities for the structure of the solution database.

One very efficient workflow for creating the structure of the database is to do so via the visual tools and then test for viability. Once the structure is acceptable you can use the Workbench tooling to generate ’Create Table’ scripts that can be placed into migration scripts that Flyway can use to recreate the database structure from a blank schema.

3.4 Apache Maven

Apache Maven is a declarative project management and build tool that is used to manage dependencies and build solution artifacts. Maven can be installed standalone as a command line build tool but in this case is used as a plugin which comes preinstalled and configured with the IntelliJ development environment.

Instead of defining each step in the build process procedurally, we declare the what goals to execute in each phase of the structured build process which greatly reduces the boilerplate code and configuration of previous build systems such as Ant.

The core strength of Maven is in the Central Repository which contains versioned JARs of publicly available class libraries. Defining a dependency on a specific JAR in the Project Object Model (POM) file will cause Maven to retrieve that dependency from the repository and place it in the classpath to be referenced by code and then later the build process.

3.5 Brackets

Brackets is an open source code editor that is itself written in the languages of the web; HTML, CSS, and Javascript. It is developed by the Adobe Systems Incorporated, and has a growing eco-system of plugins, code analysis and refactoring functionality, and build tools that are useful for working with Javascript applications, which is an area where most Java integrated development environments are still quite weak.

3.6 GulpJs

GulpJs is a streaming build system used to create build pipelines for front end web applications. In this project the primary use was to replace the Ruby build process for the jQuery.XMLEditor and provide quick integration with the browser to speed development time when rewriting parts of the jQuery.XMLEditor component. An example of a ”build pipeline” is ”linting” the Javascript code, combining multiple files of code into a single file, and then minimizing it, obfusciating it if required, and then copying it to a distribution directory. ”Linting” is a process which analyzes Javascript code for errors and bad practices which might create unwanted behavior in the program.

4 SYSTEM DESCRIPTION AND DESIGN

4.1 General Description

The IEC61850 LTI Tool Producer is a stand-alone web application that is access from inside Moodle as an external tool resource instance.

Each ”resource” in Moodle is a link on the course page that must be created and configured as an External Tool by the course or system administrator. When this resource link is clicked by a course administrator or student the IEC61850 LTI tool web application will be embedded into an IFrame in the course site.

Course administrators accessing this link will be able to create problem sets composed of a description, a starting XML structure i.e. the question, and an answer key XML structure.

Alternatively, if a manual grading option has been specified the course administrator may access a list of completed problem sets ordered by student for which they may enter a grade for each student – problem set. Each problem / answer combination is shown as a ”diff” with color coded markings for deviations between the answer and problem set. Yellow highlights are for elements which are the same, but have slightly differing answers. Red highlights are for elements which exist in one structure but not the other.

Students accessing the link will be able to follow the descriptions entered by the course administrator, which is read only for students, and create their own XML answer structures that will be compared to the grading key entered by the course administrator.

Should manual grading be selected by the course administrator, these answer sets must await grading. If automatic grading is selected by the course administrator, the XML answer structures the student creates will be more strictly compared with the answer key with either a pass or fail grade given.

The student may make a number of attempts on the same problem set equal to the amount set by the course administrator when configuring the resource link.

In both the automatic and manual grading methods, a scheduled service posts the grades back to the Moodle grade book for the student – resource.

4.2 Security

As an LTI 1.1 external tool, the IEC61850 LTI Tool Provider depends on Moodle for authentication and authorization. Users who log into Moodle are assigned a role, and that role is passed in the POST parameters from the Tool Consumer (Moodle) to the Tool Provider (IEC61850 LTI).

This approach makes the POST parameters the primary attack vector for malicious intent toward the IEC61850 LTI Tool provider. This intent is secured by a cryptographic hash which is generated by a fingerprint of all parameter values in the POST request and a shared secret between the Tool Consumer and Tool Provider. This shared secret must be securely configured on both ends by system administration as it is never transmitted between the two systems.

Each request is also accompanied with a Nonce, which is a generated hash string unique to each request. The Tool Provider keeps track of all Nonce values used in requests and prohibits the use of one nonce value more than once. This is combined with the request timestamp to validate incoming requests against replay attacks. If the request timestamp is more than 5 minutes old, it is rejected. If the request timestamp is less than 5 minutes old, the Tool Provider checks to ensure that the Nonce value hasn’t been used in the last 5 minutes, which is how long it keeps Nonce values stored in the application database.

Tampering with the Nonce value or the timestamp field alters the post parameters and thus the cryptographic hash value sent to sign the request to the Tool Provider, which will also invalidate the request.

Security for the Grade POSTS back to the Moodle system follows a ”Plain Old XML” pattern and the messages are signed using OAuth body signing to ensure message integrity. The body of the message is XML that follows the schema for the service operation being requested and the message is signed using the oauth_consumer_key and oauth_consumer_secret that was used to execute the launch. (IMSGlobal, 2014)

4.3 Functional Specification

The following listed functionality describes the requirements that must be met for the IEC61850 LTI Tool Provider to meet the needs targeted in this project.

The requirements changed slightly throughout the project life cycle as it was demonstrated to the end users. Automated grading was a *nice to have* feature that became a *must have* functionality late in the project life cycle, but was completed before the thesis completion.

Other *nice to have* functionality was not completed due to time constraints that arose due to difficulties integrating front end components.

4.3.1 Must Have Functionality

- Professor Role : Ability to Create Problem Sets
- Professor Role : Ability to Manually Grade Problem Sets
- Professor Role : Ability to set Problem Sets to Automated Grading
- Professor Role : Ability to view grading of Problem Sets
- Professor Role : Ability to set attempt limits on Problem Sets
- Professor Role : Ability to specify the number of Problems in a Problem Set
- Student Role : Must be able to submit answers to Problem Sets
- System : Must be able to report grades back to Moodle

4.3.2 Need to Have Funtionality

- Must be able to validate an LTI 1.1 specification OAuth signed POST
- Must be able to construct a valid LTI 1.1 specification OAuth webservice request to the webservice exposed for reporting grades.
- Must be able to automatically grade student attempts on a pass / fail basis.

- Must be able to validate and expire Nonce values in the system.
- Must provide ability to display and edit XML structures in a visual manner.

4.3.3 Nice to Have Functionality

- Could also allow professors or course administrators to pre-select existing problems to allow for more convenient course management.
- Could also allow copying of the ”problem” XML structure to the ”answer key” XML structure which would allow more convenient modification of the answer key instead of having to repeat actions.

4.4 Flow Chart

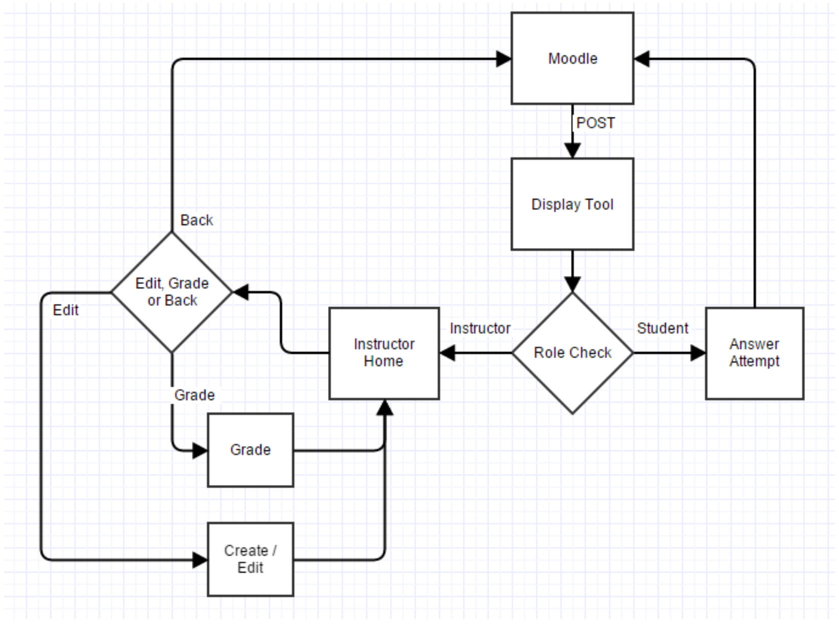


Figure 5. User Interface Flow.

4.5 Sequence Diagram

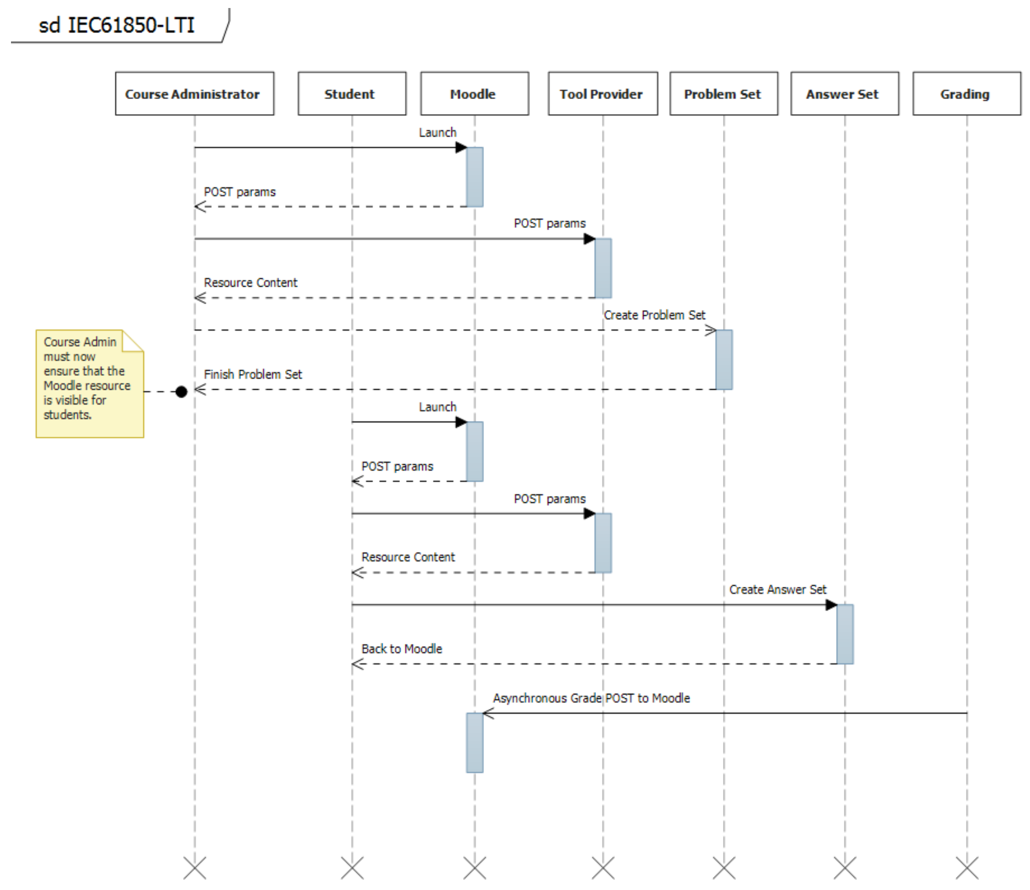


Figure 6. User Sequence Diagram.

4.6 Domain Model

4.6.1 Consumer

A Consumer is the entity which represents the Tool Consumer, in this case the organization running the Moodle instance. It has a key, which identifies the Consumer name, and a SharedSecret which is used to OAuth sign requests between the Tool Consumer and the Tool Provider.

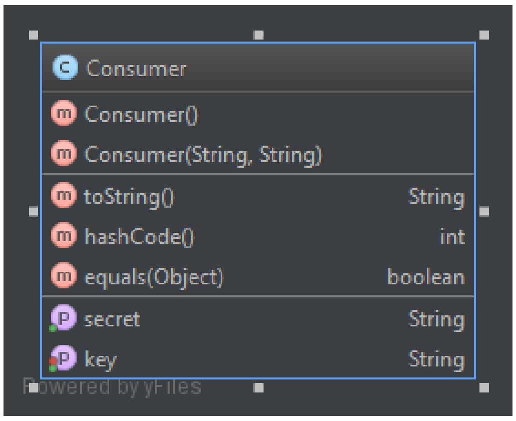


Figure 7. Consumer UML Diagram.

4.6.2 Nonce

As mentioned in the Security discussion the system must store Nonce values sent in requests from Moodle after successfully validating them. This entity class represents those Nonce values and the timestamp of their arrival.

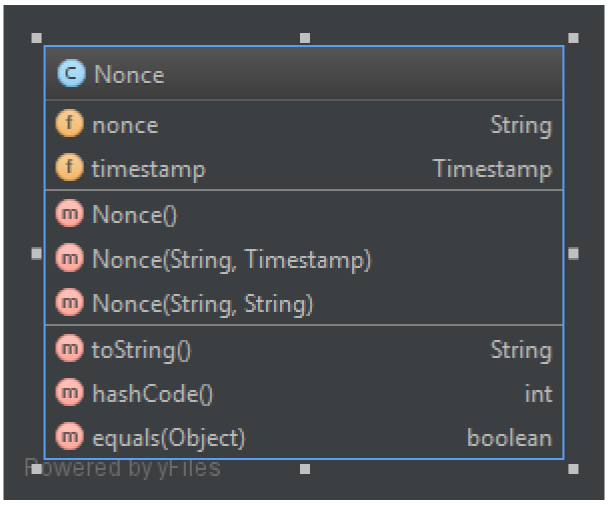
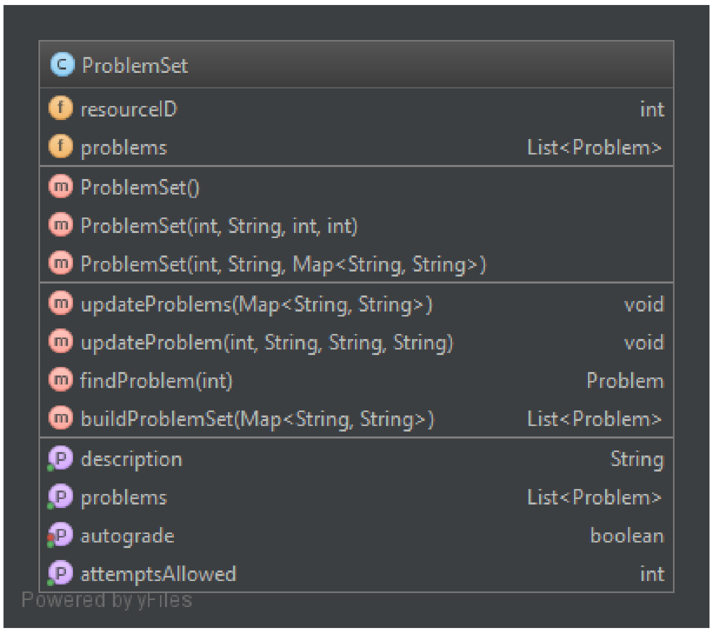


Figure 8. Nonce UML Diagram.

4.6.3 ProblemSet

A ProblemSet, shown in Figure 9, is a collection of Problem entities that represent the task for the Moodle resource that is used to access the Tool Provider.



```
ProblemSet
resourceID                                   int
problems                           List<Problem>
ProblemSet()
ProblemSet(int, String, int, int)
ProblemSet(int, String, Map<String, String>)
updateProblems(Map<String, String>)         void
updateProblem(int, String, String, String)  void
findProblem(int)                         Problem
buildProblemSet(Map<String, String>) List<Problem>
description                               String
problems                           List<Problem>
autograde                                boolean
attemptsAllowed                              int
```

Figure 9. ProblemSet UML Diagram.

The primary key identifier of a ProblemSet is the resourceID, which is the resource identifier sent from Moodle that corresponds to the activity used to access the Tool Provider. It has data fields for a description, the number of attempts allowed, and a flag specifying whether the Problems in the set will be graded by the automated system or manually by the instructor.

The ProblemSet class also encapsulates functionality to find a specific Problem in the set by the Problem id, functionality to update all Problems in the set or individual Problems, and the ability to build a collection of Problems if the ProblemSet entity is not initialized with the constructor that takes in a raw set.

4.6.4 Problem

A Problem represents a single task that the student must attempt. There may be many Problems in a ProblemSet or only one, as shown by the UML Diagram in Figure 10 on the next page.

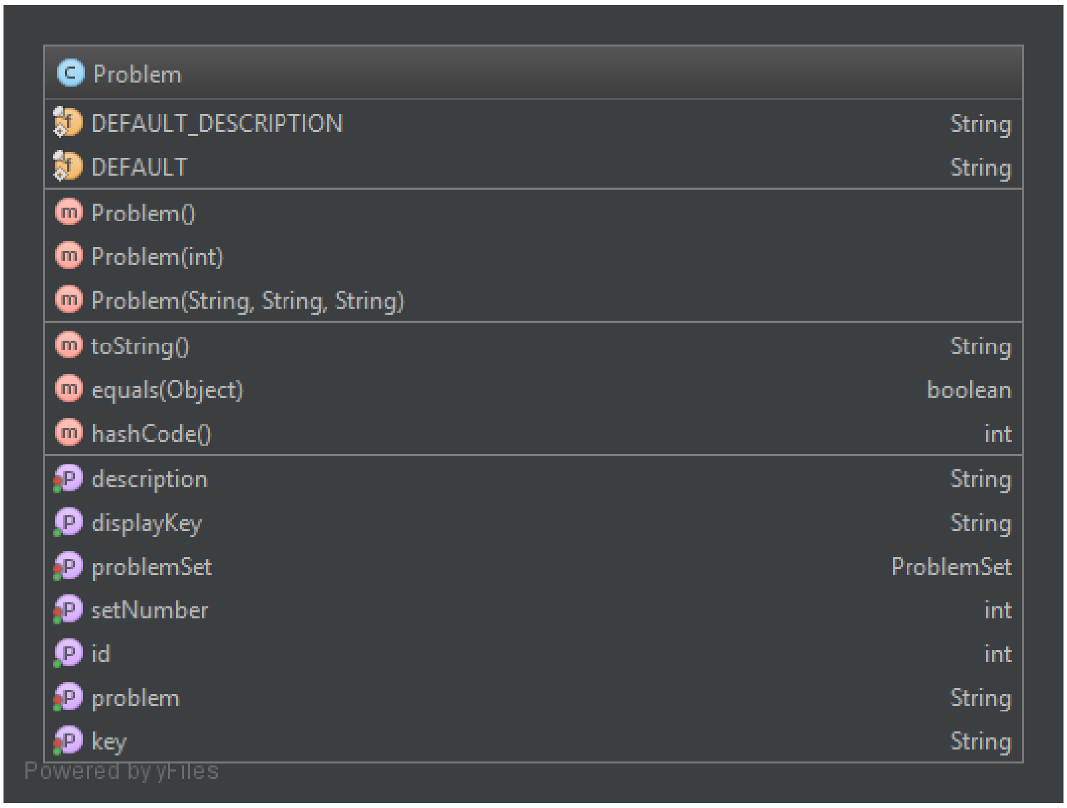


Figure 10. Problem UML Diagram.

A Problem entity is keyed by a database generated Id, and contains a foreign key which references the ProblemSet parent for the Problem. Each Problem is also given a SetNumber which defines the ordering of Problems in the Set. A Problem also contains the description, the problem definition, and answer key which make up the task. A default description and problem are also provided to initialize the entity in a way that ensures we do not have the possibility of NULL values in the database.

4.6.5 AnswerSet

An AnswerSet represents a student attempt at a ProblemSet. It is keyed by an incrementing database generated identifier, and contains foreign keys linking to the related Consumer to which the student belongs, the ProblemSet for which it is an attempt to solve, how many attempts have been made at the ProblemSet, identifying information for the student, and fields used by the Grade Reporting service.

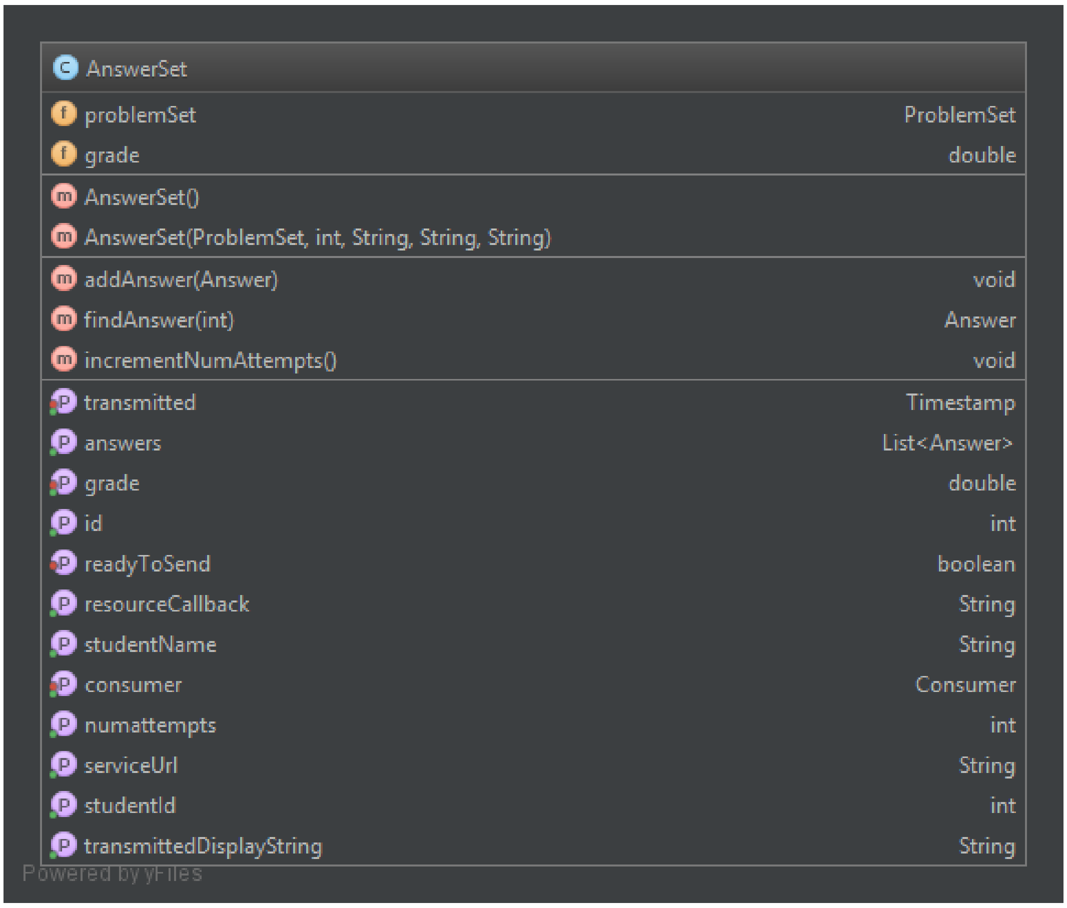


Figure 11. AnswerSet UML Diagram.

The fields used by the Grade Reporting service are the readytoSend flag, the serviceUrl, which specifies the webservice provided in the POST parameters when the student submitted his answers, the resource callback, which is the SourcedId needed by Moodle to accept a ReplaceResult grade request, and the grade itself.

The AnswerSet class also encapsulates functionality to set the grade with validation which also sets the readyToSend flag, to create a human readable and consistently formatted display string for the transmission date, and the ability to add Answers to the set and increment the number of attempts at the ProblemSet.

4.6.6 Answer

An Answer is an individual attempt at a Problem which may be correct or incorrect. Answers are keyed by an incrementing database generated identifier, and contains a foreign key to the Problem to which it is an attempt, shown here in Figure 12.

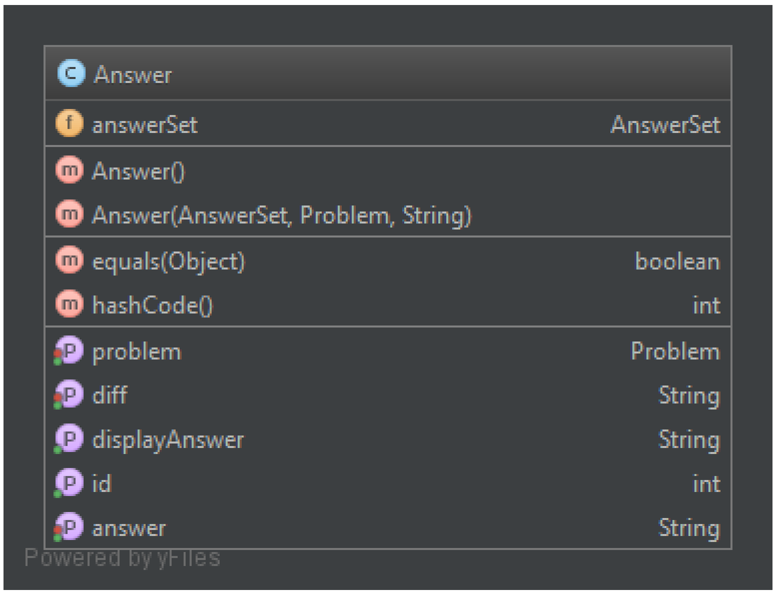


Figure 12. Answer UML Diagram.

An Answer also contains a field for holding a difference between the Problem key and the provided answer, and encapsulates functionality for providing a formatted version of the answer for display in the user interface.

4.6.7 OAuthMetadata

OAuthMetadata is a class which encapsulates some metadata about the Moodle user and their Moodle session which is used in view generation to provide a personalized greeting and allow views to access the special redirect url that Moodle can capture to end the Tool Provider session and return to the Moodle course view instead of the resource view with Tool Provider content. A UML diagram is provided below in Figure 13.

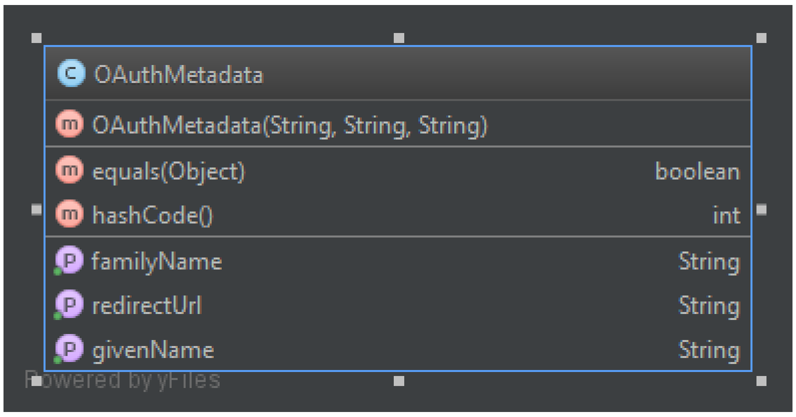
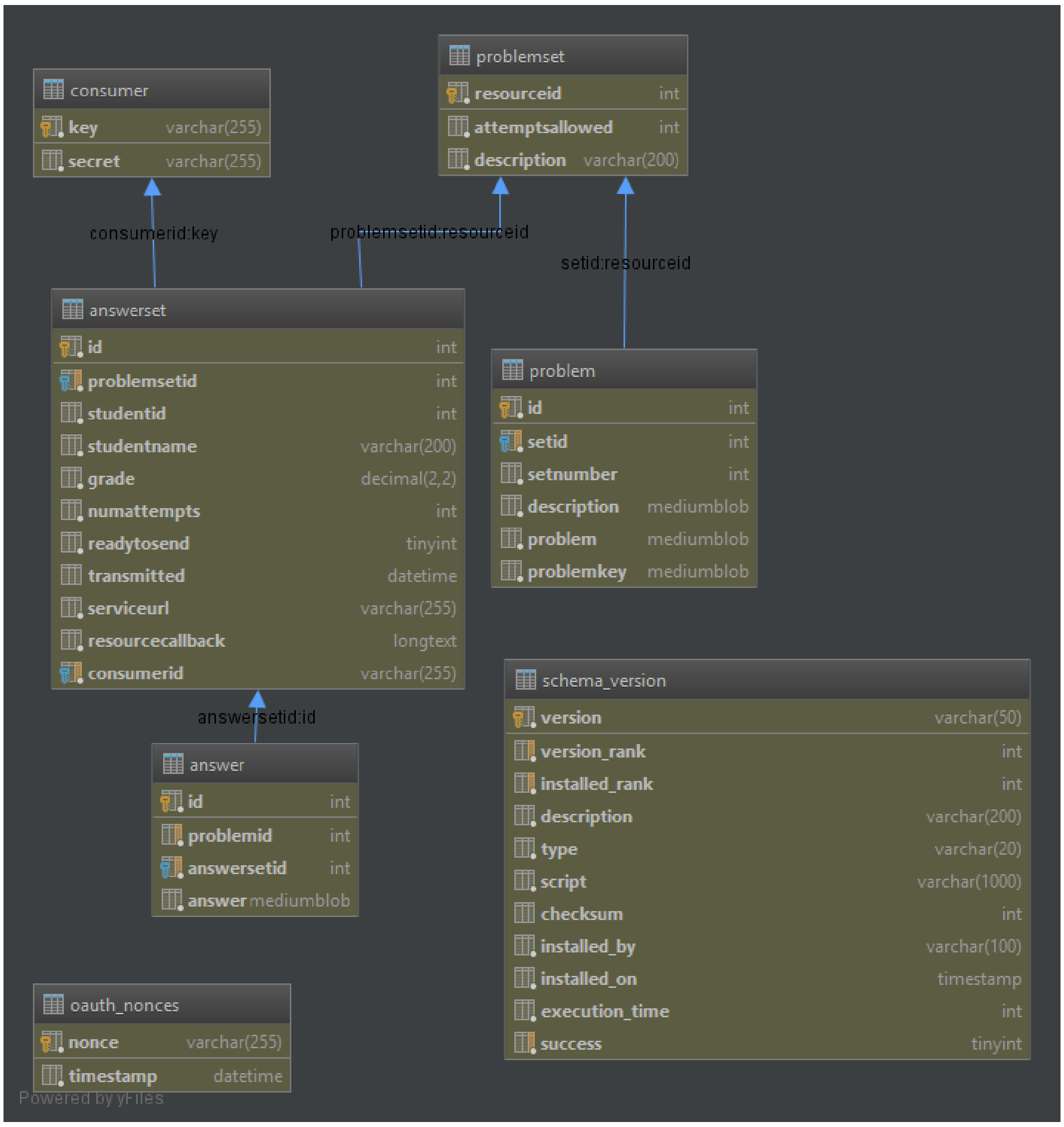


Figure 13. OAuthMetadata UML Diagram.

4.7 Database Structure

The database structure in Figure 14 shows graphically the relationships between entities described in the domain discussion. In addition, we can see the structure of the schema_version table which is used by Flyway to manage the database migrations for changes to the IEC61850 LTI Tool Provider.



consumer	
key	varchar(255)
secret	varchar(255)

problemset	
resourceid	int
attemptsallowed	int
description	varchar(200)

answerset	
id	int
problemsetid	int
studentid	int
studentname	varchar(200)
grade	decimal(2,2)
numattempts	int
readytosend	tinyint
transmitted	datetime
serviceurl	varchar(255)
resourcecallback	longtext
consumerid	varchar(255)

problem	
id	int
setid	int
setnumber	int
description	mediumblob
problem	mediumblob
problemkey	mediumblob

answer	
id	int
problemid	int
answersetid	int
answer	mediumblob

schema_version	
version	varchar(50)
version_rank	int
installed_rank	int
description	varchar(200)
type	varchar(20)
script	varchar(1000)
checksum	int
installed_by	varchar(100)
installed_on	timestamp
execution_time	int
success	tinyint

oauth_nonces	
nonce	varchar(255)
timestamp	datetime

Figure 14. Database Schema.

The database structure can be built from scratch on an empty schema by merely starting up the application. The migration scripts, shown below in Figure 15, will be executed in numerical order until the schema is complete.

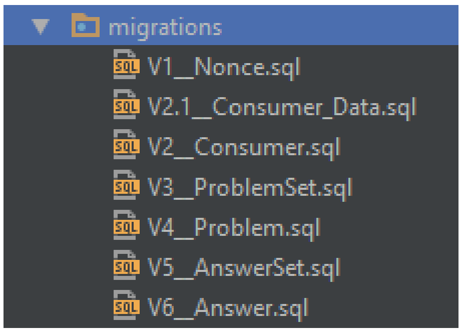
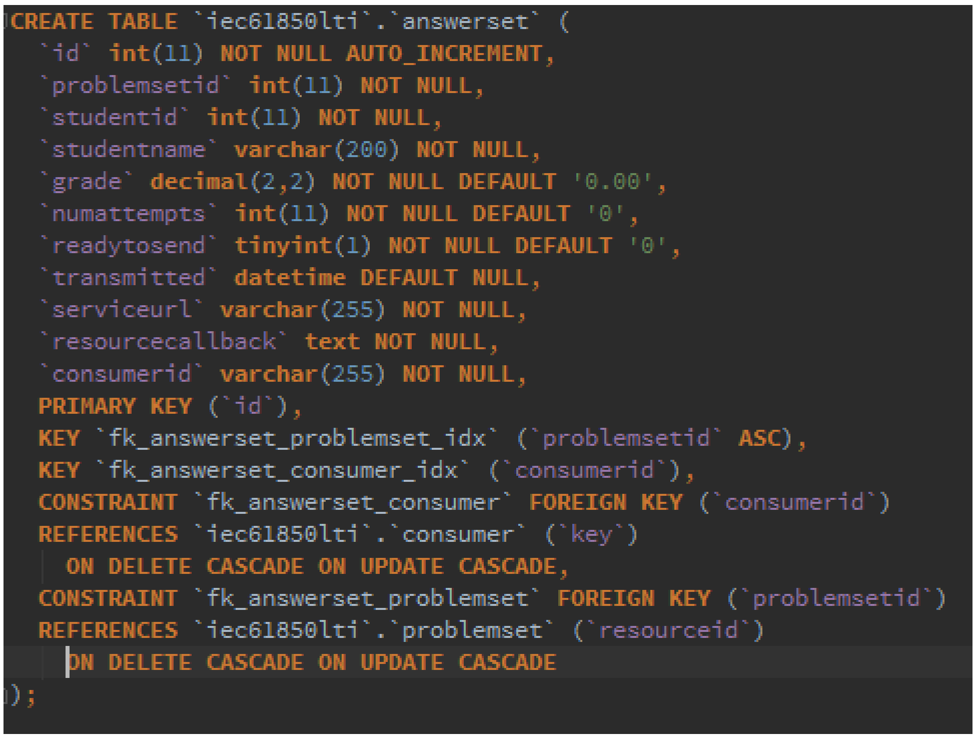


Figure 15. Flyway Database Migration Script Conventions

A sample SQL file is provided below in Figure 16, for the AnswerSet entity, which not only creates the answerset table in the iec61850lti schema but also configures the primary key and foreign key relationships.



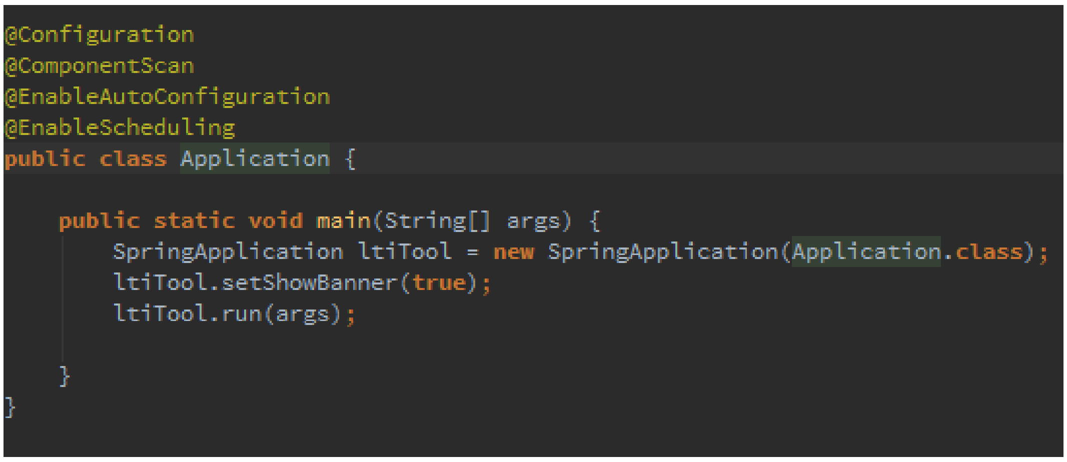
```
CREATE TABLE `iec61850lti`.`answerset` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `problemsetid` int(11) NOT NULL,
  `studentid` int(11) NOT NULL,
  `studentname` varchar(200) NOT NULL,
  `grade` decimal(2,2) NOT NULL DEFAULT '0.00',
  `numattempts` int(11) NOT NULL DEFAULT '0',
  `readytosend` tinyint(1) NOT NULL DEFAULT '0',
  `transmitted` datetime DEFAULT NULL,
  `serviceurl` varchar(255) NOT NULL,
  `resourcecallback` text NOT NULL,
  `consumerid` varchar(255) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `fk_answerset_problemset_idx` (`problemsetid` ASC),
  KEY `fk_answerset_consumer_idx` (`consumerid`),
  CONSTRAINT `fk_answerset_consumer` FOREIGN KEY (`consumerid`)
  REFERENCES `iec61850lti`.`consumer` (`key`)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_answerset_problemset` FOREIGN KEY (`problemsetid`)
  REFERENCES `iec61850lti`.`problemset` (`resourceid`)
    ON DELETE CASCADE ON UPDATE CASCADE
);
```

Figure 16. Sample Migration – AnswerSet.

5 IMPLEMENTATION

5.1 Spring Boot Web Application

The foundation of the IEC61850 LTI Tool Provider is a Spring Boot MVC application with an embedded Tomcat servlet container. The artifact produced by the Java compiler is a runnable Java archive or JAR, which creates and runs a SpringApplication on boot. The code of the Application class is shown below in Code Snippet 1.


```
@Configuration
@ComponentScan
@EnableAutoConfiguration
@EnableScheduling
public class Application {

    public static void main(String[] args) {
        SpringApplication ltiTool = new SpringApplication(Application.class);
        ltiTool.setShowBanner(true);
        ltiTool.run(args);

    }
}
```

Code Snippet 1. Application Main.

The @Configuration attribute designates the Application class as the primary source for configuration in the application.

@ComponentScan instructs the Spring Boot application to search the package containing the Application class and all sub-packages for any application component annotated with the @Component, @Service, @Repository, or @Controller annotations. These classes will be registered as Spring Beans and are eligible for autowiring throughout the application.

The @EnableAutoConfiguration instructs the Spring Application that it should examine the Java archives present in the classpath and create default configurations for those libraries which it can support with a sane default configuration.

For instance, by including the spring-boot-starter-freemarker dependency in our pom.xml file, Spring Boot automatically configures the Freemarker template engine as the view resolver in our MVC web application.

@EnableScheduling simply allows the Spring Boot application to instantiate its own scheduler class which may trigger executions of scheduled services which have been annotated with @Scheduled and live in packages under the root package.

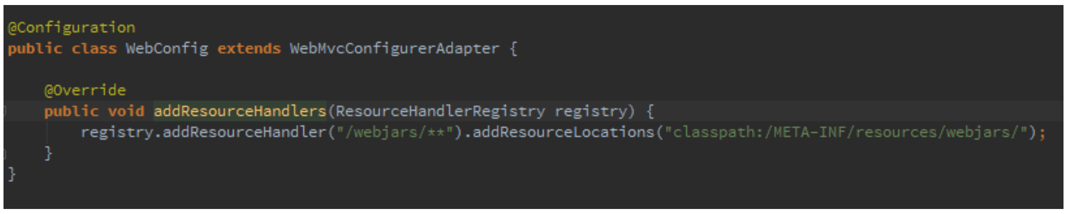
5.2 Configuration

There are multiple methods of configuration in a Spring Boot application. While the old method of XML configuration is possible, it is going out of style in favor of Java, annotations, and property file based configuration. The IEC61850 LTI Tool Provider relies on all three methods to configure for proper use.

5.2.1 Java Configuration

For some of the Javascript dependencies on the front end it was desired that we avoid a front end package manager like Bower or the Node Package Manager as it didn’t seem to be worth the additional effort in project setup and management. An alternative approach was to leverage WebJars, which packages Javascript dependencies as Java archive dependencies and allows them to be managed with Maven.

However, this is not supported out of the box with Spring Boot, so it was necessary to implement a WebMvcConfigurerAdapter. This class, shown below in Code Snippet 2, is annotated with the @Configuration annotation which designates the Java code in the class as a configuration to be picked up by the Spring Boot application at startup.



```java
@Configuration
public class WebConfig extends WebMvcConfigurerAdapter {

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/resources/webjars/");
    }
}
```

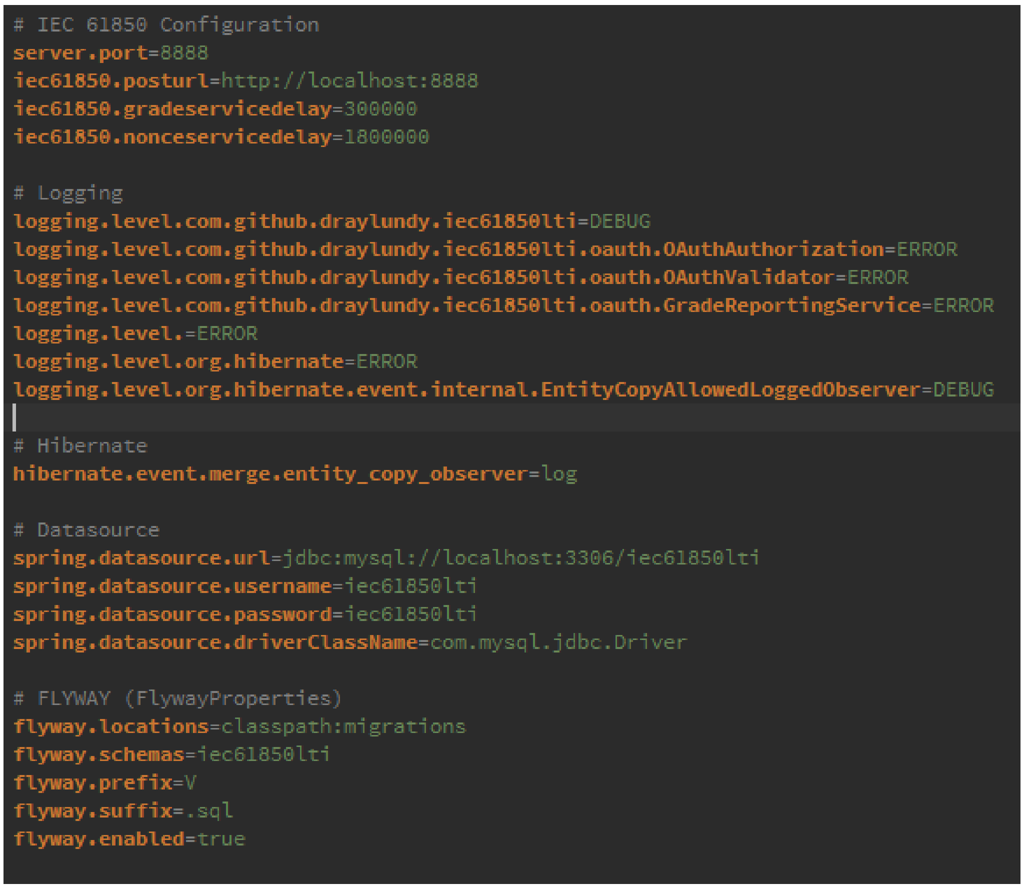
Code Snippet 2. WebMvcConfigurerAdapter.

This configuration class specifies that we want to add a location handler which maps all the WebJars into the /META-INF/resources/webjars folder where we can reference them from our HTML includes.

5.2.2 Application.Properties

The default application.properties configuration file is supplied in the classpath root and is packaged with the runnable JAR. This default properties file may be overriden by an application.properties configuration file supplied in the same directory as the runnable JAR or in an application.properties file in a config directory which lives in the same parent directory as the runnable JAR, with this last option taking the highest priority.

A sample configuration file from the development environement is shown below in Code Snippet 3, which specifies server ports and post urls, delays for the scheduled services, debug levels for logging, the datasource connection information, and the Flyway database schema migration configuration.



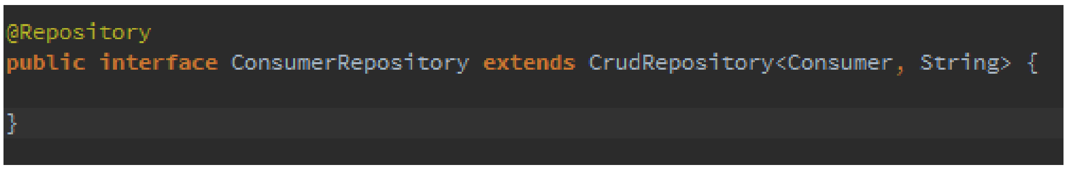
```
# IEC 61850 Configuration
server.port=8888
iec61850.posturl=http://localhost:8888
iec61850.gradeservicedelay=300000
iec61850.nonceservicedelay=1800000

# Logging
logging.level.com.github.draylundy.iec61850lti=DEBUG
logging.level.com.github.draylundy.iec61850lti.oauth.OAuthAuthorization=ERROR
logging.level.com.github.draylundy.iec61850lti.oauth.OAuthValidator=ERROR
logging.level.com.github.draylundy.iec61850lti.oauth.GradeReportingService=ERROR
logging.level.=ERROR
logging.level.org.hibernate=ERROR
logging.level.org.hibernate.event.internal.EntityCopyAllowedLoggedObserver=DEBUG

# Hibernate
hibernate.event.merge.entity_copy_observer=log

# Datasource
spring.datasource.url=jdbc:mysql://localhost:3306/iec61850lti
spring.datasource.username=iec61850lti
spring.datasource.password=iec61850lti
spring.datasource.driverClassName=com.mysql.jdbc.Driver

# FLYWAY (FlywayProperties)
flyway.locations=classpath:migrations
flyway.schemas=iec61850lti
flyway.prefix=V
flyway.suffix=.sql
flyway.enabled=true
```

Code Snippet 3. Sample application.configuration file.

5.3 Persistence Repositories

Persistence in the IEC61850 LTI Tool Provider leverages Spring Data JPA to generate repository implementations from defined interfaces. We can then @Autowire these interfaces where needed in our scheduled services and controller and Spring will inject these dependencies on instantiation of these classes.

For entities where we only need the ability to FindById and do basic CRUD operations on the Entity type, we can specify a naked interface and have these method provided for us. For example, the ConsumerRepository shown below in Code Snippet 4, where we must only extend the CrudRepository<T,K> interface, where T is the Entity type and K is the defined primary key type.

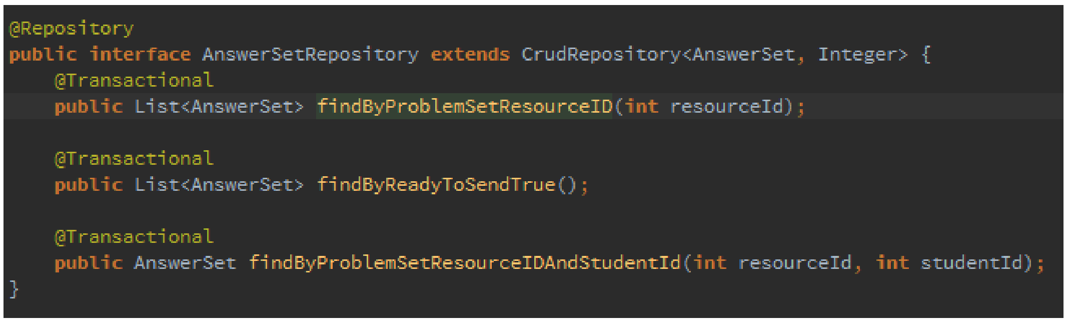


```
@Repository
public interface ConsumerRepository extends CrudRepository<Consumer, String> {

}
```

Code Snippet 4. No Implementation ConsumerRepository.

For more demanding queries it is possible to provide declarations of query methods in the interface according to the correct convention and Spring Data JPA will generate the correct query syntax. For example, in the AnswerSetRepository we need the functionality to search by AnswerSets which are ready to send and by both the Student Id and the Resource Id. This approach is shown below in Code Snippet 5.

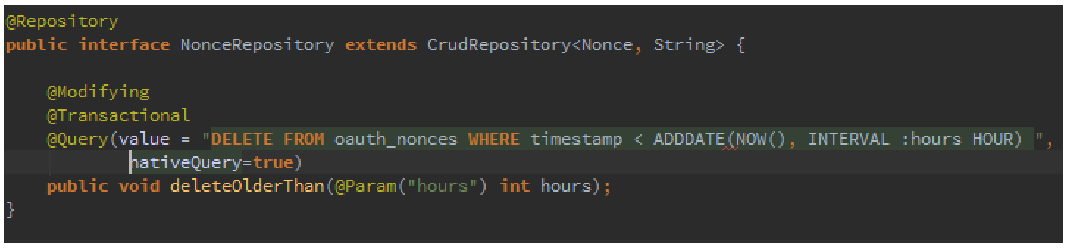
The method name is specified as findByProblemSetResourceIDAndStudentId. This will create a query that walks the object graph from ProblemSet to AnswerSet where the ResourceId on the ProblemSet and the StudentId on the AnswerSet are equal to the provided input values.



```
@Repository
public interface AnswerSetRepository extends CrudRepository<AnswerSet, Integer> {
    @Transactional
    public List<AnswerSet> findByProblemSetResourceID(int resourceId);

    @Transactional
    public List<AnswerSet> findByReadyToSendTrue();

    @Transactional
    public AnswerSet findByProblemSetResourceIDAndStudentId(int resourceId, int studentId);
}
```

Code Snippet 5. Convention based query methods – AnswerSetRepository.

For even more demanding use cases where there is no support for this convention based query definition it is possible to define the query via annotations, which can be in either JPQL or in raw native SQL. This approach is visibile in Code Snippet 6, which shows our NonceRepository which batch deletes all Nonces greater than a specific time limit, which does not have support in the convention based approach.



```
@Repository
public interface NonceRepository extends CrudRepository<Nonce, String> {

    @Modifying
    @Transactional
    @Query(value = "DELETE FROM oauth_nonces WHERE timestamp < ADDDATE(NOW(), INTERVAL :hours HOUR) ",
            nativeQuery=true)
    public void deleteOlderThan(@Param("hours") int hours);
}
```

Code Snippet 6. Native Query Definition – NonceRepository.

By leveraging this convention based approach and only writing our own data access queries when necessary we greatly improve our development time and the time required to implement the infrastructure. This was essential because we required much of our infrastructure to be functional before we could begin the process of validating POST parameters from Moodle, and the quicker we could get to this point the better.

5.4 Spring MVC Controller

The Spring MVC Controller handles requests dispatched from the DispatcherServlet and FrontController according to the annotation mappings on the Controller. This dispatch pattern is shown here in Figure 17.

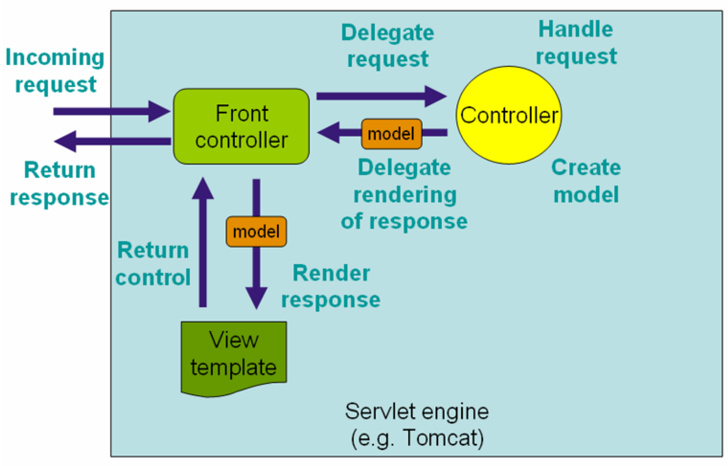


Figure 17. Spring MVC Controller Function.

5.4.1 Autowired Dependencies

Each request will result in the creation of a new ToolController instance, each of which will be injected with a specific set of dependencies that will be scoped to the user session. The ToolController requires an OAuthValidator, which validates incoming requests, an Autograder, which grades student AnswerSet submissions if automated grading is specified, and repository interfaces for ProblemSet, AnswerSet, and Consumer.

5.4.2 Method Implementations

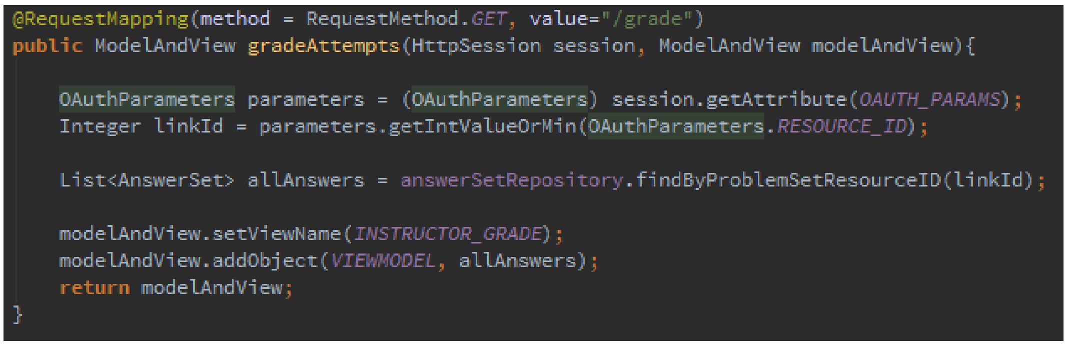
All instances of the ToolController share a factory injected instance of an slf4j Logger which reports errors and security violations into the Tool Provider application logs.

The following route mappings shown in Table 1 are specified in the ToolController, which handles all use cases for the IEC61850 LTI Tool Provider.

Table 1 : ToolController Route Mappings

Path	HTTP	Method	Returns	Consumes
displayTool	POST	displayTool	ModelAndView	x-www-form-urlencoded
edit	GET	editProblemSet	ModelAndView	
grade	GET	gradeAttempts	ModelAndView	
submitGrade	POST	submitGrade	ResponseBody	
attemptAnswer	POST	answerAttempt	ResponseBody	
saveset	POST	saveSet	ModelAndView	

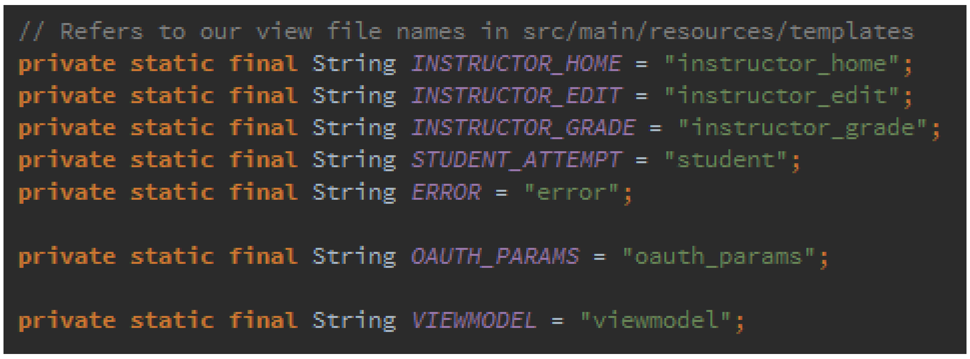
Route-method mappings which return a ModelAndView have a ModelAndView instance injected into the controller method to which we set the view name, which resolves the view template, and we add an Object with a String key that can be referenced by the view template. Example from the grade functionality shown below in Code Snippet 7.



```
@RequestMapping(method = RequestMethod.GET, value="/grade")
public ModelAndView gradeAttempts(HttpSession session, ModelAndView modelAndView){

    OAuthParameters parameters = (OAuthParameters) session.getAttribute(OAUTH_PARAMS);
    Integer linkId = parameters.getIntValueOrMin(OAuthParameters.RESOURCE_ID);

    List<AnswerSet> allAnswers = answerSetRepository.findByProblemSetResourceID(linkId);

    modelAndView.setViewName(INSTRUCTOR_GRADE);
    modelAndView.addObject(VIEWMODEL, allAnswers);
    return modelAndView;
}
```

Code Snippet 7. Grade Student Attempts Controller Method.

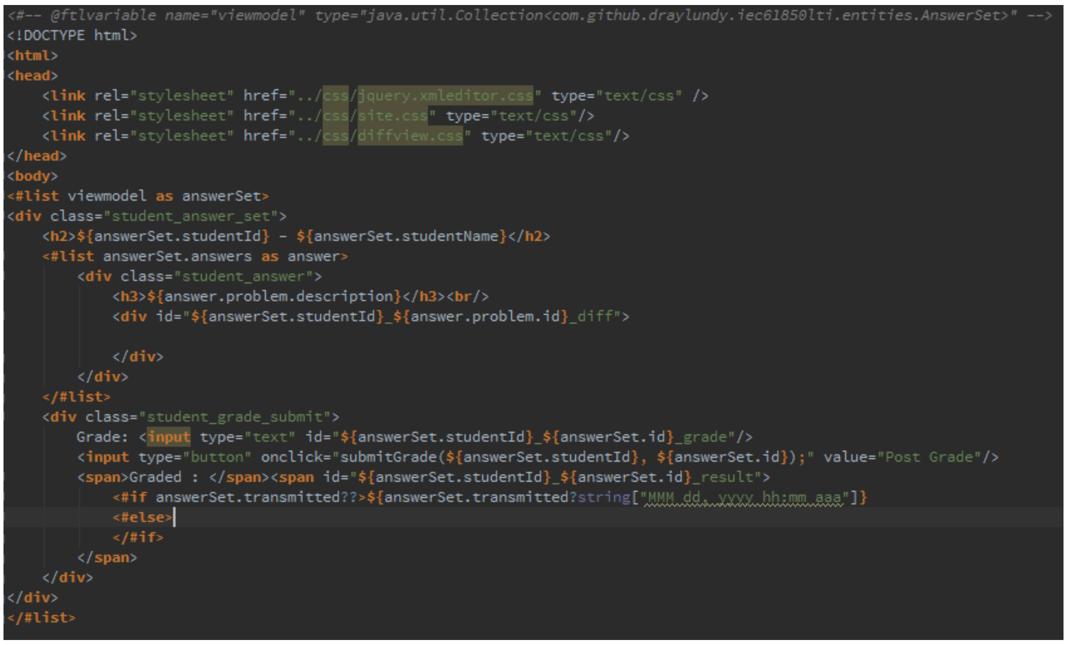
In this example the ModelAndView object is populated with a List of AnswerSet entities with the VIEWMODEL key, and the view name is set to the INSTRUCTOR_GRADE key. These keys are included in the controller as static final String values, shown here in Code Snippet 8, which allow us to reference these throughout the controller in a much more robust way, i.e. a change to the name of a template requires a change in only one place, not in several.



```
// Refers to our view file names in src/main/resources/templates
private static final String INSTRUCTOR_HOME = "instructor_home";
private static final String INSTRUCTOR_EDIT = "instructor_edit";
private static final String INSTRUCTOR_GRADE = "instructor_grade";
private static final String STUDENT_ATTEMPT = "student";
private static final String ERROR = "error";

private static final String OAUTH_PARAMS = "oauth_params";

private static final String VIEWMODEL = "viewmodel";
```

Code Snippet 8. View Template Definitions.

When the Template Engine, in our case Freemarker, recieves the template name and the model collection, it resolves the *instructor_grade* template, shown below in Code Snippet 9, and injects the model into the locations specified in the template. In this case, we iterate through the AnswerSet entities and display a list of Answers with the Student Id and Student Name as a header. At the end of each AnswerSet we create an input to which we attach a Javascript method *submitGrade* to the onClick handler.



```
<#-- @ftlvariable name="viewmodel" type="java.util.Collection<com.github.draylundy.iec61850lti.entities.AnswerSet>" -->
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="../css/jquery.xmleditor.css" type="text/css" />
    <link rel="stylesheet" href="../css/site.css" type="text/css"/>
    <link rel="stylesheet" href="../css/diffview.css" type="text/css"/>
</head>
<body>
<#list viewmodel as answerSet>
<div class="student_answer_set">
    <h2>${answerSet.studentId} - ${answerSet.studentName}</h2>
    <#list answerSet.answers as answer>
        <div class="student_answer">
            <h3>${answer.problem.description}</h3><br/>
            <div id="${answerSet.studentId}_${answer.problem.id}_diff">

            </div>
        </div>
    </#list>
    <div class="student_grade_submit">
        Grade: <input type="text" id="${answerSet.studentId}_${answerSet.id}_grade"/>
        <input type="button" onclick="submitGrade(${answerSet.studentId}, ${answerSet.id});" value="Post Grade"/>
        <span>Graded : </span><span id="${answerSet.studentId}_${answerSet.id}_result">
            <#if answerSet.transmitted??>${answerSet.transmitted?string["MMM dd, yyyy hh:mm aaa"]}
            <#else>
            </#if>
        </span>
    </div>
</div>
</#list>
```

Code Snippet 9. Grade Student Attempts View Template.

5.5 OAuth Validation

The first technical challenge to get something very simple working was being able to accept POST parameters from Moodle through our ToolController methods and correctly validate them. To accomplish this task efficiently Fiddler was used to capture raw POST parameters coming from Moodle and these parameters were hard-coded into the application so that a normal HTTP GET request would have these parameters injected and redirected into the HTTP POST method on our MVC controller.

5.5.1 Moodle Request Validation

Moodle request validation has multiple layers that must all validate. First we must verify that there are a UserId, Roles, and Resource Id present in the POST parameters. The UserId and Roles information is required to ensure that the user has been authenticated and authorized in Moodle, and the Resource Id is required as it makes up the unique identifier for Problem Sets and Answer Sets in the IEC61850-LTI Tool Provider Domain.

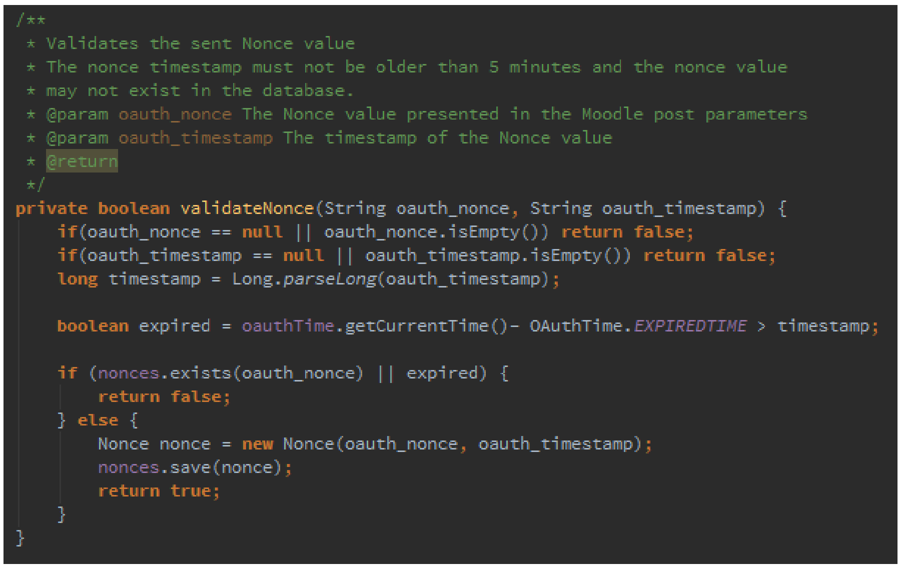
The POST parameters must also contain a valid Tool Consumer Id that has been registered with the Tool Provider, an OAuth generated Nonce value, and a timestamp.

Once verified that these values are present we must not only validate the Nonce, but the POST parameters as a whole.

5.5.2 Nonce Validation

When validating the Nonce we *fail-fast*, or invalidate the Nonce if it or the accompanying timestamp does not exist. Then we validate the timestamp, with the limit being chosen as 5 minutes. This is a window that seeks to minimize differences between the system clock on the Moodle server and the system clock on the IEC61850 LTI Tool Provider server.

If the Nonce already exists in the database or if it has been expired, we invalidate the Nonce. If it has not expired and does not already exist, we validate the nonce and save it to the Nonces table in the database as an original request. This approach is demonstrated in Code Snippet 10.



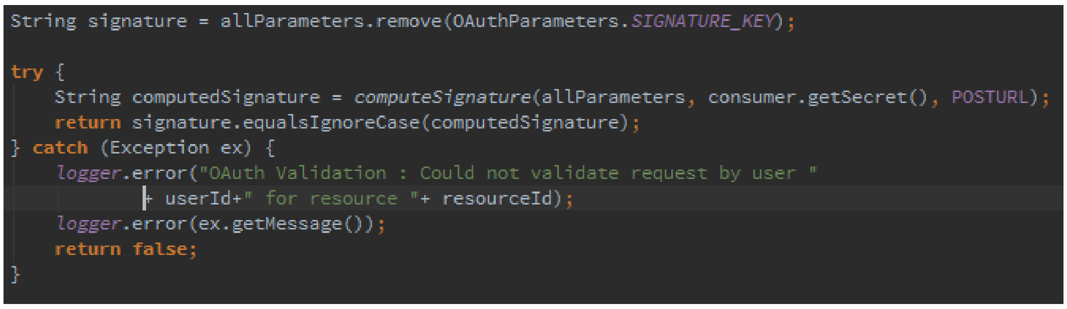
```java
/**
 * Validates the sent Nonce value
 * The nonce timestamp must not be older than 5 minutes and the nonce value
 * may not exist in the database.
 * @param oauth_nonce The Nonce value presented in the Moodle post parameters
 * @param oauth_timestamp The timestamp of the Nonce value
 * @return
 */
private boolean validateNonce(String oauth_nonce, String oauth_timestamp) {
    if(oauth_nonce == null || oauth_nonce.isEmpty()) return false;
    if(oauth_timestamp == null || oauth_timestamp.isEmpty()) return false;
    long timestamp = Long.parseLong(oauth_timestamp);

    boolean expired = oauthTime.getCurrentTime()- OAuthTime.EXPIREDTIME > timestamp;

    if (nonces.exists(oauth_nonce) || expired) {
        return false;
    } else {
        Nonce nonce = new Nonce(oauth_nonce, oauth_timestamp);
        nonces.save(nonce);
        return true;
    }
}
```

Code Snippet 10. Nonce Validation.

5.5.3 POST Parameter Validation

After ensuring that the Nonce and timestamp for the request are valid, we must then validate signature of the POST parameters (Code Snippet 11).

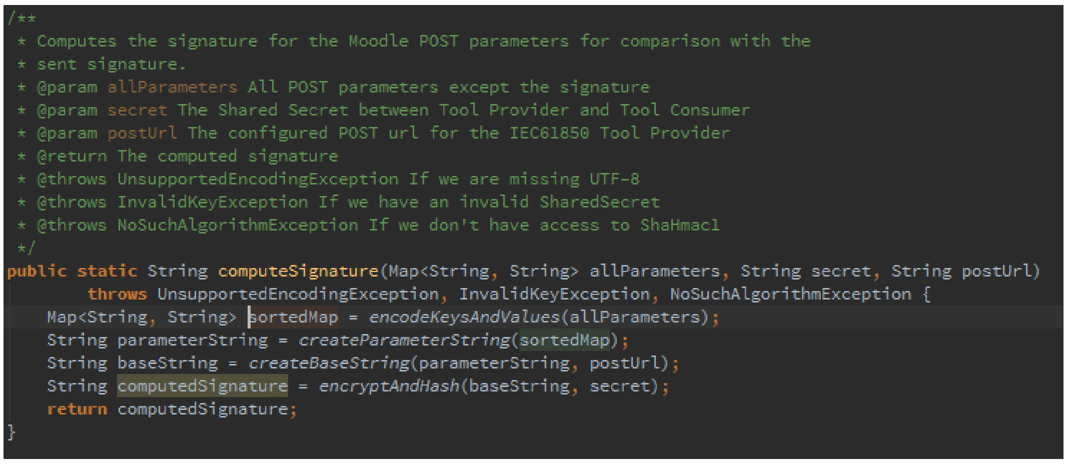
First we must remove the OAuth signature key from the POST parameter set, as it cannot be used to generate itself on the Moodle end. The remaining parameters are those which were used to sign the request.



```java
String signature = allParameters.remove(OAuthParameters.SIGNATURE_KEY);

try {
    String computedSignature = computeSignature(allParameters, consumer.getSecret(), POSTURL);
    return signature.equalsIgnoreCase(computedSignature);
} catch (Exception ex) {
    logger.error("OAuth Validation : Could not validate request by user "
            + userId+" for resource "+ resourceId);
    logger.error(ex.getMessage());
    return false;
}
```

Code Snippet 11. POST Parameter Validation.

Then we must compute the signature for those parameters and compare it with the computed signature that Moodle sent along with the request, with the root method shown in Code Snippet 12 at the top of the next page.

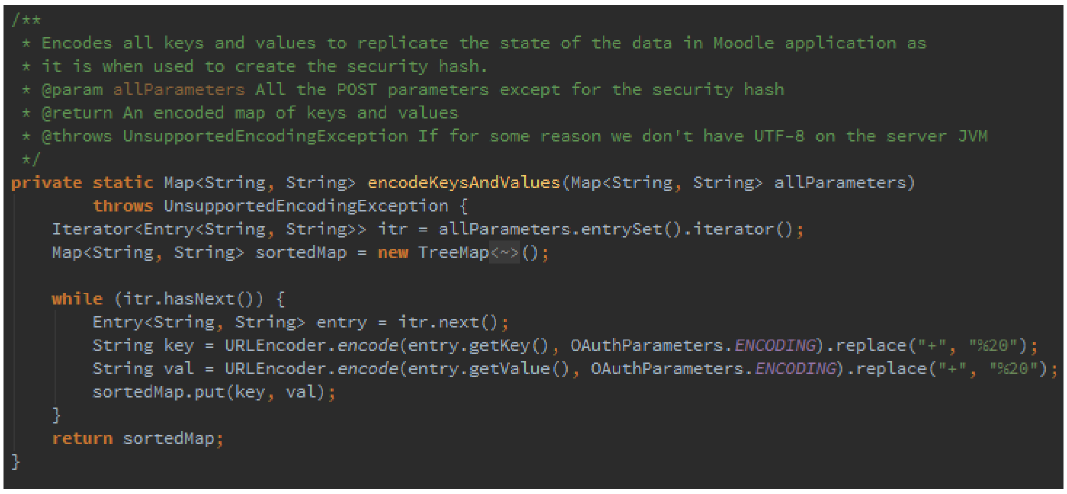


```
/**
 * Computes the signature for the Moodle POST parameters for comparison with the
 * sent signature.
 * @param allParameters All POST parameters except the signature
 * @param secret The Shared Secret between Tool Provider and Tool Consumer
 * @param postUrl The configured POST url for the IEC61850 Tool Provider
 * @return The computed signature
 * @throws UnsupportedEncodingException If we are missing UTF-8
 * @throws InvalidKeyException If we have an invalid SharedSecret
 * @throws NoSuchAlgorithmException If we don't have access to ShaHmac1
 */
public static String computeSignature(Map<String, String> allParameters, String secret, String postUrl)
        throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException {
    Map<String, String> sortedMap = encodeKeysAndValues(allParameters);
    String parameterString = createParameterString(sortedMap);
    String baseString = createBaseString(parameterString, postUrl);
    String computedSignature = encryptAndHash(baseString, secret);
    return computedSignature;
}
```

Code Snippet 12. Compute OAuth Signature.

This process is broken down into 4 steps:

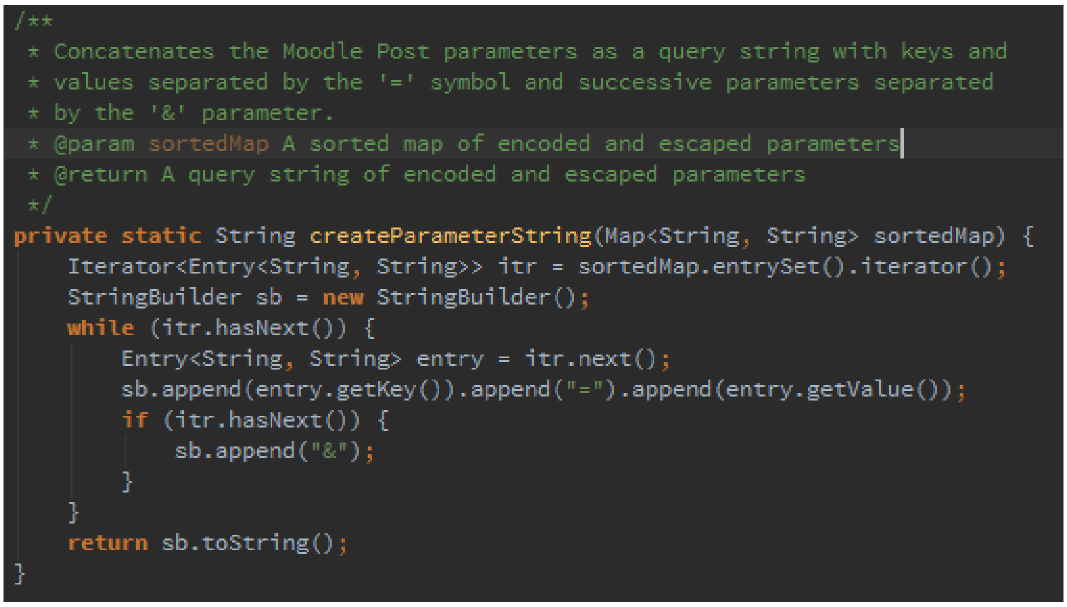
1. Encoding and escaping the keys and values in the parameter map, shown here in Code Snippet 13.



```
/**
 * Encodes all keys and values to replicate the state of the data in Moodle application as
 * it is when used to create the security hash.
 * @param allParameters All the POST parameters except for the security hash
 * @return An encoded map of keys and values
 * @throws UnsupportedEncodingException If for some reason we don't have UTF-8 on the server JVM
 */
private static Map<String, String> encodeKeysAndValues(Map<String, String> allParameters)
        throws UnsupportedEncodingException {
    Iterator<Entry<String, String>> itr = allParameters.entrySet().iterator();
    Map<String, String> sortedMap = new TreeMap<~>();

    while (itr.hasNext()) {
        Entry<String, String> entry = itr.next();
        String key = URLEncoder.encode(entry.getKey(), OAuthParameters.ENCODING).replace("+", "%20");
        String val = URLEncoder.encode(entry.getValue(), OAuthParameters.ENCODING).replace("+", "%20");
        sortedMap.put(key, val);
    }
    return sortedMap;
}
```

Code Snippet 13. Encoding Parameter Keys and Values.

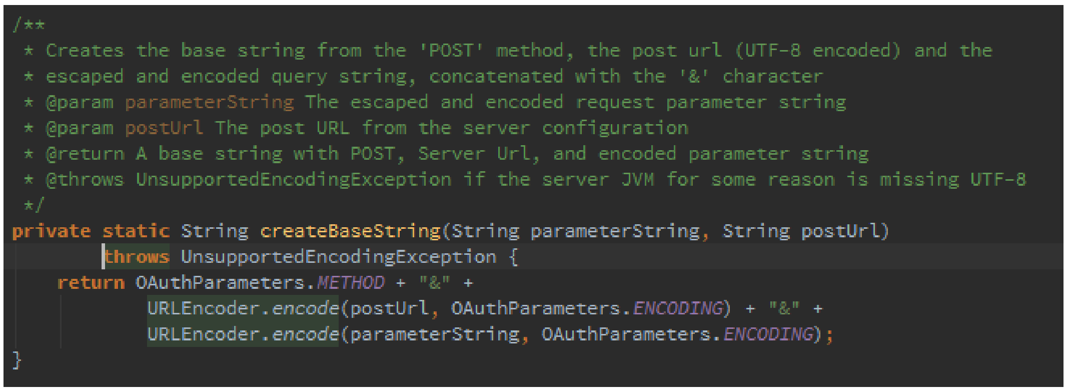
2. Concatenating the keys and values in the parameter map into a parameter string, shown in Code Snippet 14 on the next page.



```java
/**
 * Concatenates the Moodle Post parameters as a query string with keys and
 * values separated by the '=' symbol and successive parameters separated
 * by the '&' parameter.
 * @param sortedMap A sorted map of encoded and escaped parameters
 * @return A query string of encoded and escaped parameters
 */
private static String createParameterString(Map<String, String> sortedMap) {
    Iterator<Entry<String, String>> itr = sortedMap.entrySet().iterator();
    StringBuilder sb = new StringBuilder();
    while (itr.hasNext()) {
        Entry<String, String> entry = itr.next();
        sb.append(entry.getKey()).append("=").append(entry.getValue());
        if (itr.hasNext()) {
            sb.append("&");
        }
    }
    return sb.toString();
}
```

Code Snippet 14. Parameter String Concatenation.

3. Concatenating the POST method with the POST url and the parameter string, Code Snippet 15, below.

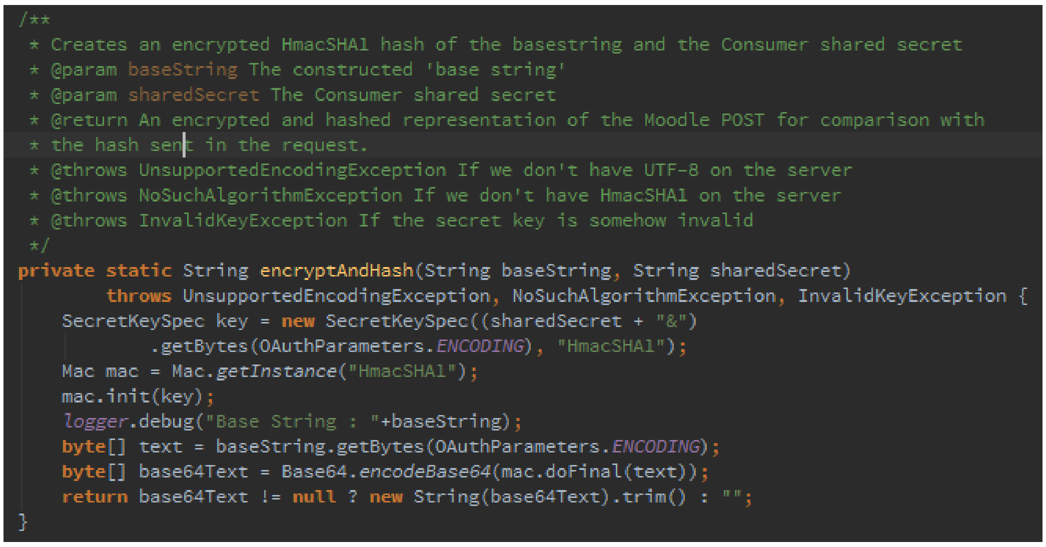


```java
/**
 * Creates the base string from the 'POST' method, the post url (UTF-8 encoded) and the
 * escaped and encoded query string, concatenated with the '&' character
 * @param parameterString The escaped and encoded request parameter string
 * @param postUrl The post URL from the server configuration
 * @return A base string with POST, Server Url, and encoded parameter string
 * @throws UnsupportedEncodingException if the server JVM for some reason is missing UTF-8
 */
private static String createBaseString(String parameterString, String postUrl)
        throws UnsupportedEncodingException {
    return OAuthParameters.METHOD + "&" +
            URLEncoder.encode(postUrl, OAuthParameters.ENCODING) + "&" +
            URLEncoder.encode(parameterString, OAuthParameters.ENCODING);
}
```

Code Snippet 15. Base String Concatenation.

4. And finally encrypting and hashing the result with the Shared Secret, Code Snippet 16, next page.

Once the POST parameters and the Nonce have been validated then and only then will the IEC61850 LTI Tool Provider serve content to the Moodle instance.

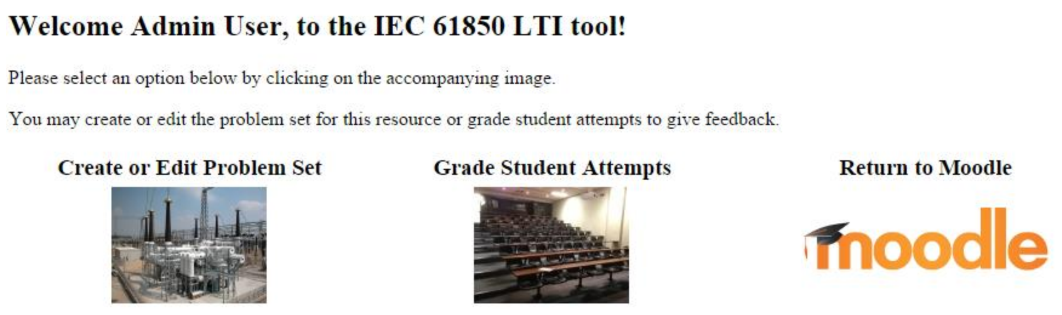


```
/**
 * Creates an encrypted HmacSHA1 hash of the basestring and the Consumer shared secret
 * @param baseString The constructed 'base string'
 * @param sharedSecret The Consumer shared secret
 * @return An encrypted and hashed representation of the Moodle POST for comparison with
 * the hash sent in the request.
 * @throws UnsupportedEncodingException If we don't have UTF-8 on the server
 * @throws NoSuchAlgorithmException If we don't have HmacSHA1 on the server
 * @throws InvalidKeyException If the secret key is somehow invalid
 */
private static String encryptAndHash(String baseString, String sharedSecret)
        throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException {
    SecretKeySpec key = new SecretKeySpec((sharedSecret + "&")
            .getBytes(OAuthParameters.ENCODING), "HmacSHA1");
    Mac mac = Mac.getInstance("HmacSHA1");
    mac.init(key);
    logger.debug("Base String : "+baseString);
    byte[] text = baseString.getBytes(OAuthParameters.ENCODING);
    byte[] base64Text = Base64.encodeBase64(mac.doFinal(text));
    return base64Text != null ? new String(base64Text).trim() : "";
}
```

Code Snippet 16. Encrypt and Hash the Base String.

5.6 User Interface

After the request has been validated, course administrators are welcomed with a home view providing the options available. In the test environment the user is named ”Admin User”, so the course administrator is welcomed by name as shown in Figure 18.



Welcome Admin User, to the IEC 61850 LTI tool!

Please select an option below by clicking on the accompanying image.

You may create or edit the problem set for this resource or grade student attempts to give feedback.

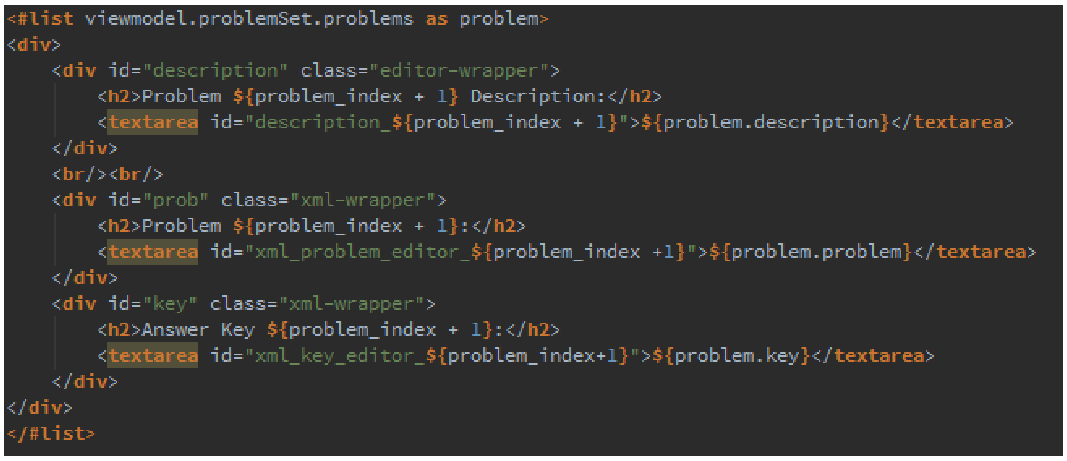
Figure 18. User Interface - Instructor Home.

When choosing to Create or Edit the Problem Set, the course administrator will be redirected to the *instructor edit* view which contains the Problem Set for this Moodle resource link. This Problem Set may contain one or multiple Problems, which are blank in the case that this is the first access to this resource, or the previously saved Problem Set instances if this is a return visit to this resource.

Each Problem is represented by three components, one description component which may contain rich text and links to images uploaded to the Moodle course, and two XML editor fields for creating the problem definition and answer key to the Problem.

Each instance of the description rich text editor and visual XML editor components are HTML TextAreas that hold the underlying data, which is hidden and replaced with the respective Javascript components; CKEditor and jQuery.XMLEditor (discussed below).

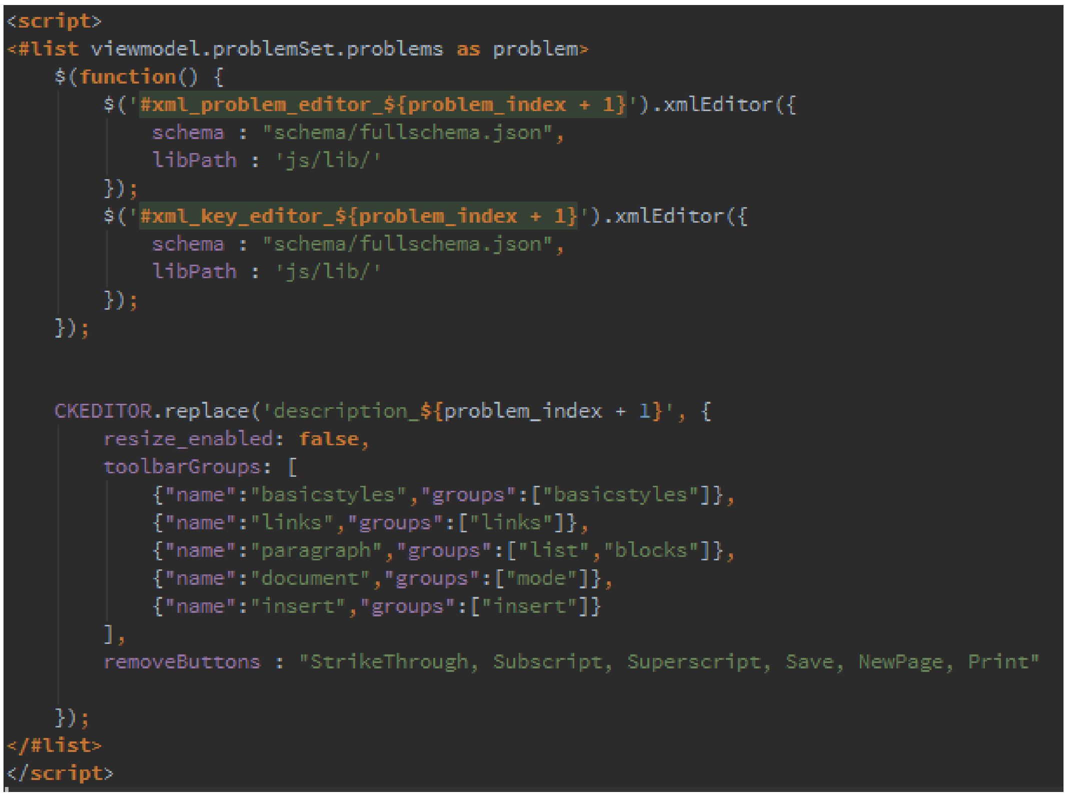
We create these dynamically in the Freemarker template based on the quantity and/or existence of Problems in the ProblemSetViewModel returned from the controller logic, shown here in Code Snippet 17.



```
<#list viewmodel.problemSet.problems as problem>
<div>
    <div id="description" class="editor-wrapper">
        <h2>Problem ${problem_index + 1} Description:</h2>
        <textarea id="description_${problem_index + 1}">${problem.description}</textarea>
    </div>
    <br/><br/>
    <div id="prob" class="xml-wrapper">
        <h2>Problem ${problem_index + 1}:</h2>
        <textarea id="xml_problem_editor_${problem_index +1}">${problem.problem}</textarea>
    </div>
    <div id="key" class="xml-wrapper">
        <h2>Answer Key ${problem_index + 1}:</h2>
        <textarea id="xml_key_editor_${problem_index+1}">${problem.key}</textarea>
    </div>
</div>
</#list>
```

Code Snippet 17. Edit ProblemSet View Template.

After the page successfully loads a script embedded in the page (Code Snippet 18, below) creates the necessary components for each ProblemSet along with configuring options for the CKEditor description component to reduce clutter in the toolbar.

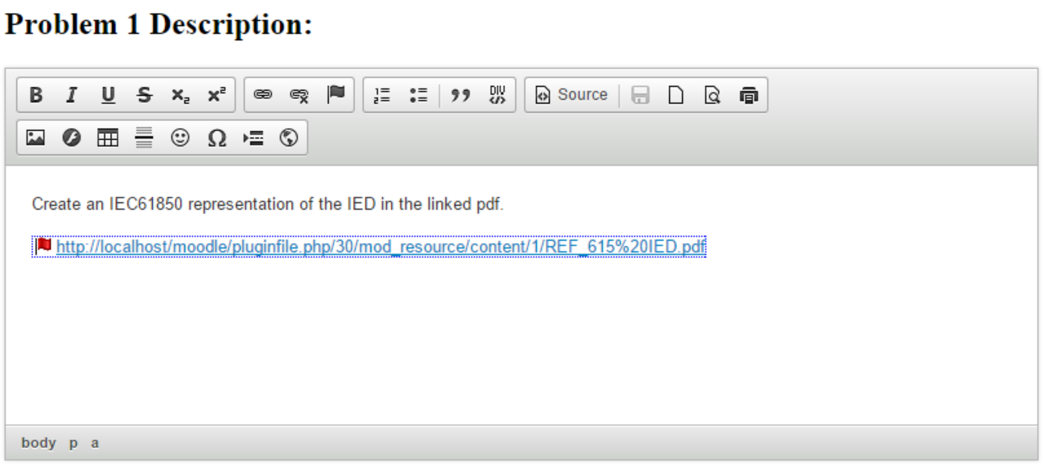


```
<script>
<#list viewmodel.problemSet.problems as problem>
    $(function() {
        $('#xml_problem_editor_${problem_index + 1}').xmlEditor({
            schema : "schema/fullschema.json",
            libPath : 'js/lib/'
        });
        $('#xml_key_editor_${problem_index + 1}').xmlEditor({
            schema : "schema/fullschema.json",
            libPath : 'js/lib/'
        });
    });


    CKEDITOR.replace('description_${problem_index + 1}', {
        resize_enabled: false,
        toolbarGroups: [
            {"name":"basicstyles","groups":["basicstyles"]},
            {"name":"links","groups":["links"]},
            {"name":"paragraph","groups":["list","blocks"]},
            {"name":"document","groups":["mode"]},
            {"name":"insert","groups":["insert"]}
        ],
        removeButtons : "StrikeThrough, Subscript, Superscript, Save, NewPage, Print"

    });
</#list>
</script>
```

Code Snippet 18. Edit ProblemSet View Template Scripts.

5.6.1 Description Component UI (CKEditor Rich Text Editor)

The Description component for each Problem allows the course administrator or instructor the ability to provide any instructions or external resources required for the student to be able to complete the Problem assignment. In the example shown below in Figure 19, a short instruction is given along with a link to a IED spec sheet.


Problem 1 Description:

Create an IEC61850 representation of the IED in the linked pdf.

http://localhost/moodle/pluginfile.php/30/mod_resource/content/1/REF_615%20IED.pdf

Figure 19. User Interface - Description Component.

5.6.2 Problem and Key Components (jQuery.XMLEditor)

Both the Problem and Answer Key components provide the ability to create an XML structure by point and click process with select controls and text fields for restricted value and free text entries. When an element in the main view is selected, the element listing on the right hand side of the main pane changes to reflect the available child elements or attributes for the current selected element in the XML structure.

When no element is selected, the available elements are the top level elements in the SCL structure as defined by the IEC61850 standard. In the example shown in Figure 20, below, a Substation is being configured with a Voltage Level of 10 MV.

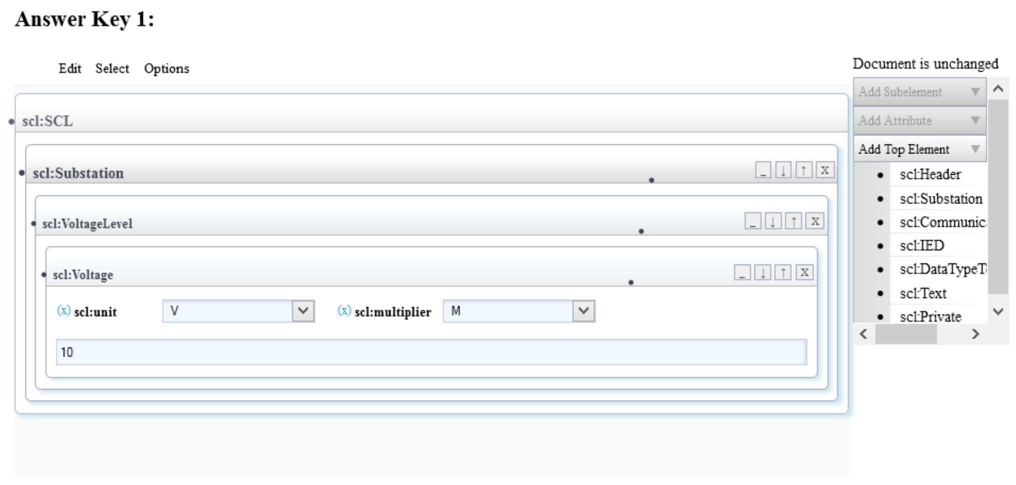


Figure 20. User Interface - Visual XML Editor Component.

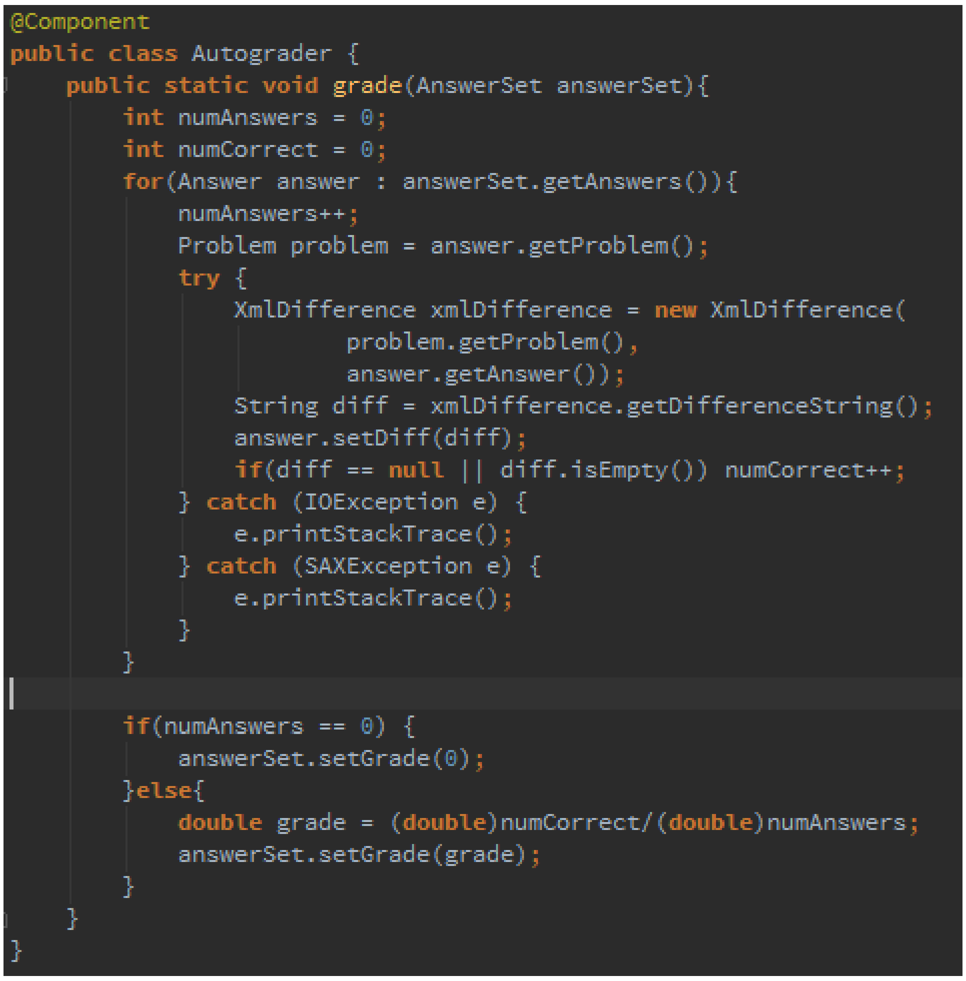
5.7 Grading Options

The grading option for a Moodle resource instance can be specified by passing a custom parameter to the IEC61850 LTI Tool Provider with a key value of *auto_grade* and a value of either TRUE or FALSE, case insensitive.

5.7.1 Automatic Pass / Fail Grading

Automated grading is implemented for a required use case, and leverages the XMLUnit testing tool to semantically compare XML structures.

When automatic grading is selected each Problem Key and Answer are semantically compared, with each comparison being graded either Pass, if no semantic differences are found between the two XML structures, or Fail, if semantic differences are found. The implementation of the Autograder which uses the XMLDifference engine provided by the XMLUnit library is shown below in Code Snippet 19.



```java
@Component
public class Autograder {
    public static void grade(AnswerSet answerSet){
        int numAnswers = 0;
        int numCorrect = 0;
        for(Answer answer : answerSet.getAnswers()){
            numAnswers++;
            Problem problem = answer.getProblem();
            try {
                XmlDifference xmlDifference = new XmlDifference(
                        problem.getProblem(),
                        answer.getAnswer());
                String diff = xmlDifference.getDifferenceString();
                answer.setDiff(diff);
                if(diff == null || diff.isEmpty()) numCorrect++;
            } catch (IOException e) {
                e.printStackTrace();
            } catch (SAXException e) {
                e.printStackTrace();
            }
        }

        if(numAnswers == 0) {
            answerSet.setGrade(0);
        }else{
            double grade = (double)numCorrect/(double)numAnswers;
            answerSet.setGrade(grade);
        }
    }
}
```

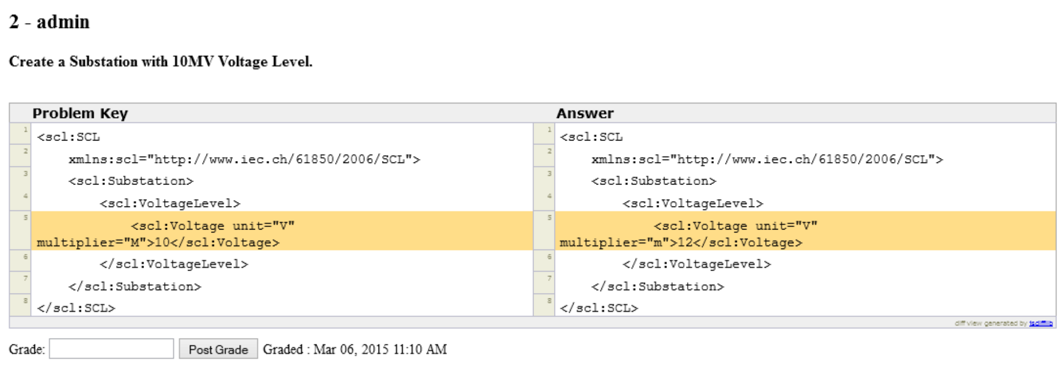
Code Snippet 19. Autograder Implementation.

The final grade for the Problem Set will be the proportion of Problems with a Pass grade divided by the total number of Problems in the Problem Set, and this grade will be assigned to the Answer Set, as well as the difference string that marked the Answer as ”Fail” should it exist to the Answer entity. These will be persisted to the database for the Grade Reporting service to report back to Moodle.

5.7.2 Manual Grading

If the automatic grading POST parameter is missing or is provided with the value of FALSE, the course administrator may select the Grade Student Attempts option to view the AnswerSets that correspond to each ProblemSet for each Student, and submit a grade for each set.

An example of manual grading is shown below in Figure 21. The Problem description instructed the student to create a Substation with a 10MV Voltage Level. However, the student, in this case a student named ”admin” has made an error and created a Substation with a 12mV Voltage Level.



2 - admin

Create a Substation with 10MV Voltage Level.

Problem Key

```
<scl:SCL
    xmlns:scl="http://www.iec.ch/61850/2006/SCL">
    <scl:Substation>
        <scl:VoltageLevel>
            <scl:Voltage unit="V"
multiplier="M">10</scl:Voltage>
        </scl:VoltageLevel>
    </scl:Substation>
</scl:SCL>
```

Answer

```
<scl:SCL
    xmlns:scl="http://www.iec.ch/61850/2006/SCL">
    <scl:Substation>
        <scl:VoltageLevel>
            <scl:Voltage unit="V"
multiplier="m">12</scl:Voltage>
        </scl:VoltageLevel>
    </scl:Substation>
</scl:SCL>
```

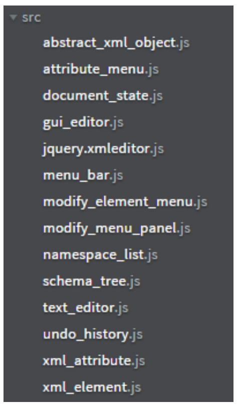
diff view generated by jsdifflib

Grade: Post Grade Graded : Mar 06, 2015 11:10 AM

Figure 21. User Interface - Manual Diff Grading.

5.8 jQuery.XMLEditor Modifications

Unfortunately the jQuery.XMLEditor component was not suitable for our use straight out of the box. When placing more than one instance of the component on the page major usability issues arose, specifically that interaction with one instance of the component would modify all instances of the component. Resolving this issue required some extensive modifications to the library.



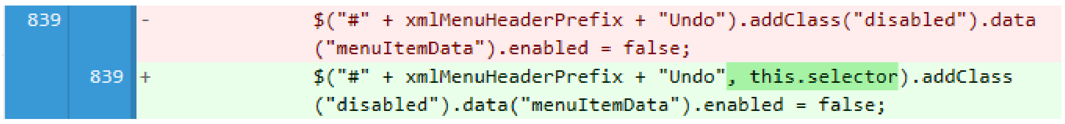
```
src
    abstract_xml_object.js
    attribute_menu.js
    document_state.js
    gui_editor.js
    jquery.xmleditor.js
    menu_bar.js
    modify_element_menu.js
    modify_menu_panel.js
    namespace_list.js
    schema_tree.js
    text_editor.js
    undo_history.js
    xml_attribute.js
    xml_element.js
```

Figure 22. jQuery.XMLEditor Code Organization.

As shown above in Figure 22, the jQuery.XMLEditor library is composed of multiple files that are concatenated at build time into one single Javascript file which can be minified and included in the script references for the HTML page. However, there is not one root object that is created, but rather several root objects that work together to make up the component. These sub-components have no direct dependency on each other. Instead, they use jQuery selectors to target elements on each other, which is the root cause of the observed issue with multiple components.

To resolve this issue, we initialize each component with an id attribute, and we have modified all of the jQuery selectors to take a specificity argument referencing that id which limits the selector to the scope of the object with that id.

A diff from a source code commit between the original file and the modified file in Code Snippet 20 shows the extent of the modifications to the component, where each this.selector was a selector that targeted the id of the component instance. This had to be done throughout the component, which resulted in several hundred edits.



```
-            $("#" + xmlMenuHeaderPrefix + "Undo").addClass("disabled").data
             ("menuItemData").enabled = false;
+            $("#" + xmlMenuHeaderPrefix + "Undo", this.selector).addClass
             ("disabled").data("menuItemData").enabled = false;
```

Code Snippet 20. Sample Modification - jQuery.XMLEditor.

The first attempt was a re-architecting of the component to bring all the Javascript code for the component under a single root scope, but this proved to be more work than anticipated and was eventually scrapped in favor of the jQuery selector which constrained the selection scope.

5.9 Grade Reporting Service

The grade reporting service must be configured when the resource is created in Moodle, the Tool Consumer. Specifically, in the site administrative configuration web-services must be enabled, and then in the resource link grade return from Tool Providers must also be enabled.

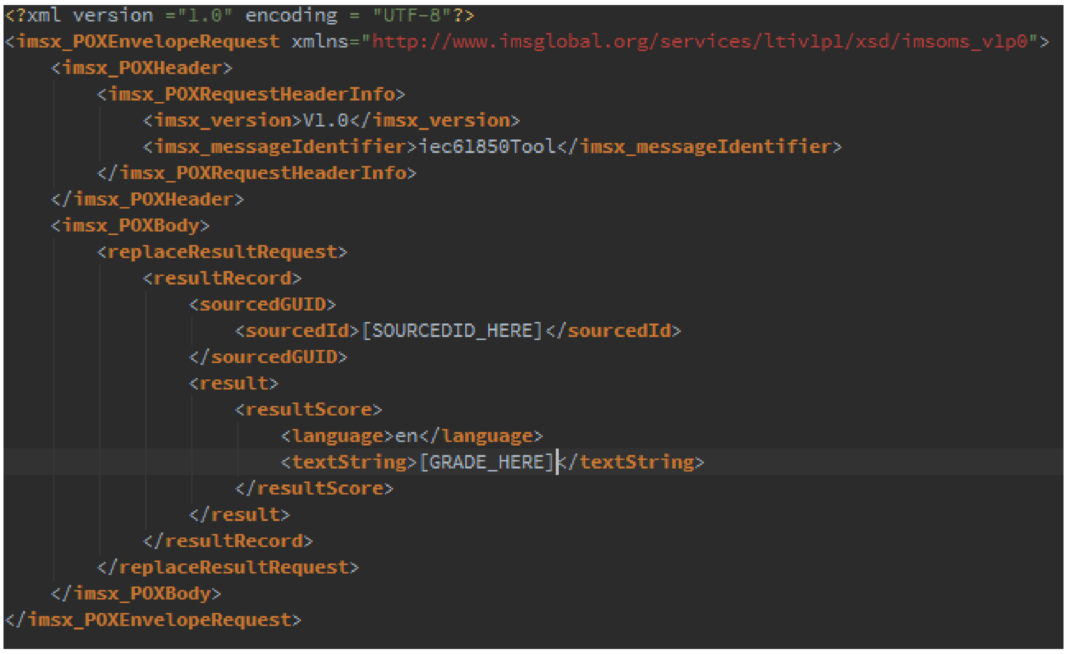
5.9.1 Grade Report Format

The format of a webservice call is a POST of application/xml data to a URL that is specified by the POST parameters from Moodle during the student answer attempt.

Also required in the XML body of the request is the sourcedId that is also included in the POST parameters from Moodle during the student answer attempt.

This sourcedId is a JSON structure which contains the instanceid, which identifies the resource, the userid, which identifies the user, the typeid, which identifies the type of user, the launch id, which specifies the launch attempt for which the grade is valid, and a security hash which must be returned with the grade POST.

A sample XML structure for the POST is shown in Code Snippet 21 below.

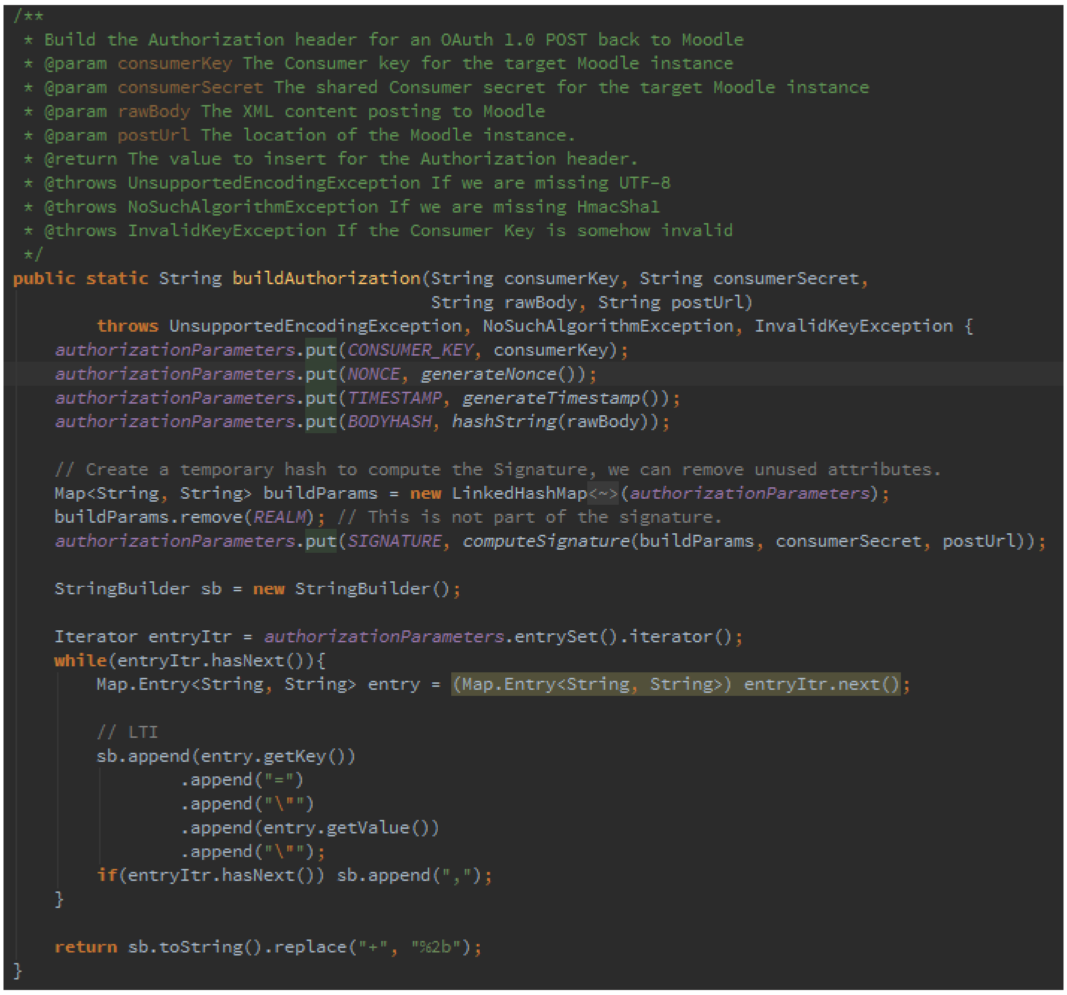


```xml
<?xml version ="1.0" encoding = "UTF-8"?>
<imsx_POXEnvelopeRequest xmlns="http://www.imsglobal.org/services/ltiv1p1/xsd/imsoms_v1p0">
    <imsx_POXHeader>
        <imsx_POXRequestHeaderInfo>
            <imsx_version>V1.0</imsx_version>
            <imsx_messageIdentifier>iec61850Tool</imsx_messageIdentifier>
        </imsx_POXRequestHeaderInfo>
    </imsx_POXHeader>
    <imsx_POXBody>
        <replaceResultRequest>
            <resultRecord>
                <sourcedGUID>
                    <sourcedId>[SOURCEDID_HERE]</sourcedId>
                </sourcedGUID>
                <result>
                    <resultScore>
                        <language>en</language>
                        <textString>[GRADE_HERE]</textString>
                    </resultScore>
                </result>
            </resultRecord>
        </replaceResultRequest>
    </imsx_POXBody>
</imsx_POXEnvelopeRequest>
```

Code Snippet 21. ReplaceResultRequest Grade POST Format.

5.9.2 Building the Authorization String

The Authorization string is a POST parameter that provides Moodle with an OAuth signature of the request body with the Consumer information and service location.

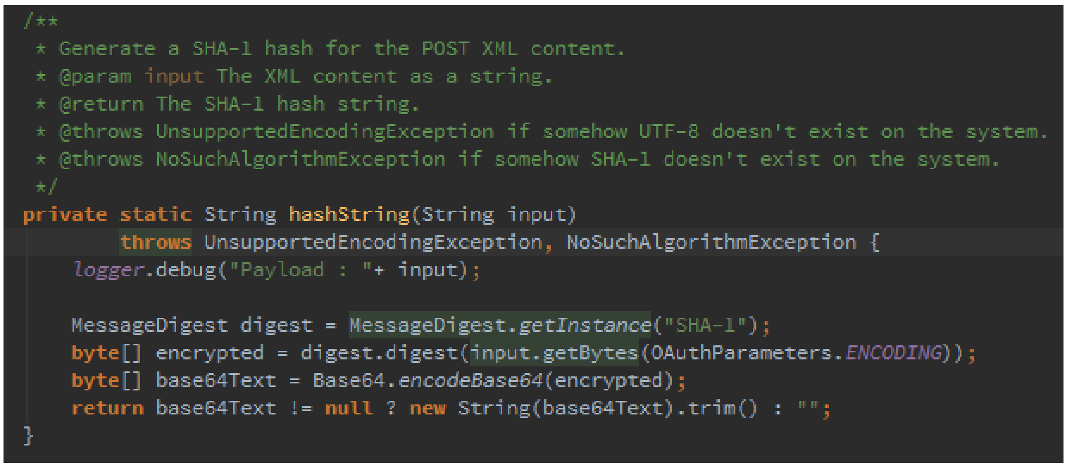


```java
/**
 * Build the Authorization header for an OAuth 1.0 POST back to Moodle
 * @param consumerKey The Consumer key for the target Moodle instance
 * @param consumerSecret The shared Consumer secret for the target Moodle instance
 * @param rawBody The XML content posting to Moodle
 * @param postUrl The location of the Moodle instance.
 * @return The value to insert for the Authorization header.
 * @throws UnsupportedEncodingException If we are missing UTF-8
 * @throws NoSuchAlgorithmException If we are missing HmacSha1
 * @throws InvalidKeyException If the Consumer Key is somehow invalid
 */
public static String buildAuthorization(String consumerKey, String consumerSecret,
                                        String rawBody, String postUrl)
        throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException {
    authorizationParameters.put(CONSUMER_KEY, consumerKey);
    authorizationParameters.put(NONCE, generateNonce());
    authorizationParameters.put(TIMESTAMP, generateTimestamp());
    authorizationParameters.put(BODYHASH, hashString(rawBody));

    // Create a temporary hash to compute the Signature, we can remove unused attributes.
    Map<String, String> buildParams = new LinkedHashMap<~>(authorizationParameters);
    buildParams.remove(REALM); // This is not part of the signature.
    authorizationParameters.put(SIGNATURE, computeSignature(buildParams, consumerSecret, postUrl));

    StringBuilder sb = new StringBuilder();

    Iterator entryItr = authorizationParameters.entrySet().iterator();
    while(entryItr.hasNext()){
        Map.Entry<String, String> entry = (Map.Entry<String, String>) entryItr.next();

        // LTI
        sb.append(entry.getKey())
                .append("=")
                .append("\"")
                .append(entry.getValue())
                .append("\"");
        if(entryItr.hasNext()) sb.append(",");
    }

    return sb.toString().replace("+", "%2b");
}
```

Code Snippet 22. Build Grade POST Authorization String.

As shown in Code Snippet 22, we must populate the oauth_nonce, oauth_timestamp, oauth_body_hash, and oauth_signature parameters. The nonce value is simply 32 random characters generated with the Java SecureRandom generator, and the timestamp is simply the number of seconds since the Unix Epoch.

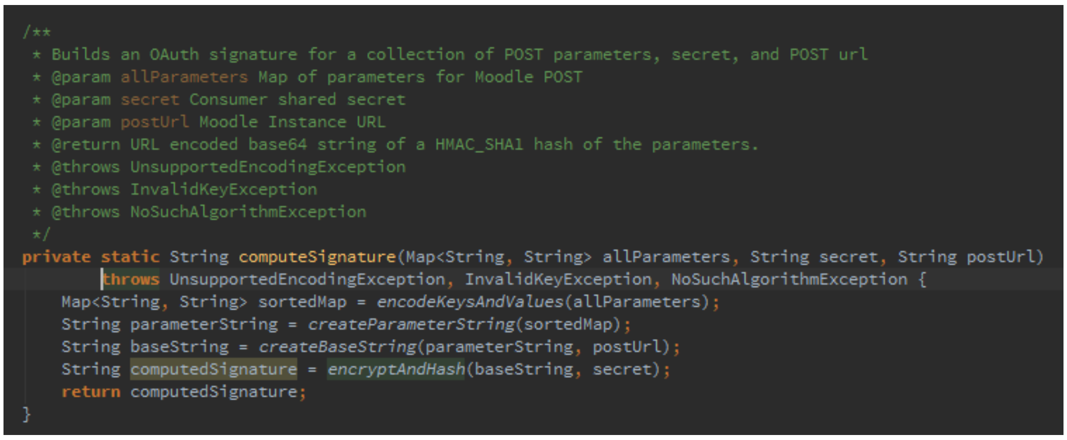
The oauth_body_hash is slightly more complex, being a base64 encoded representation of an SHA-1 encrypted output of the XML body content. The implementation is shown below in Code Snippet 23.



```java
/**
 * Generate a SHA-1 hash for the POST XML content.
 * @param input The XML content as a string.
 * @return The SHA-1 hash string.
 * @throws UnsupportedEncodingException if somehow UTF-8 doesn't exist on the system.
 * @throws NoSuchAlgorithmException if somehow SHA-1 doesn't exist on the system.
 */
private static String hashString(String input)
        throws UnsupportedEncodingException, NoSuchAlgorithmException {
    logger.debug("Payload : "+ input);

    MessageDigest digest = MessageDigest.getInstance("SHA-1");
    byte[] encrypted = digest.digest(input.getBytes(OAuthParameters.ENCODING));
    byte[] base64Text = Base64.encodeBase64(encrypted);
    return base64Text != null ? new String(base64Text).trim() : "";
}
```

Code Snippet 23. Building the Body Hash for the Grade POST.

Then we must build the oauth_signature for the POST parameters, below in Snippet 24, which now includes the oauth_body_hash.



```java
/**
 * Builds an OAuth signature for a collection of POST parameters, secret, and POST url
 * @param allParameters Map of parameters for Moodle POST
 * @param secret Consumer shared secret
 * @param postUrl Moodle Instance URL
 * @return URL encoded base64 string of a HMAC_SHA1 hash of the parameters.
 * @throws UnsupportedEncodingException
 * @throws InvalidKeyException
 * @throws NoSuchAlgorithmException
 */
private static String computeSignature(Map<String, String> allParameters, String secret, String postUrl)
        throws UnsupportedEncodingException, InvalidKeyException, NoSuchAlgorithmException {
    Map<String, String> sortedMap = encodeKeysAndValues(allParameters);
    String parameterString = createParameterString(sortedMap);
    String baseString = createBaseString(parameterString, postUrl);
    String computedSignature = encryptAndHash(baseString, secret);
    return computedSignature;
}
```

Code Snippet 24. Signing the Grade POST.

Which is a familiar four part process very similar to building the signature for comparison when authorizing POST request parameters from Moodle. Those steps are encoding the keys and values, concatenating the parameter key and value map into a parameter string, concatenating the POST method and service URL of the Moodle webservice with the parameter string, and finally encrypting and hashing that result with the SharedSecret.

6 CONCLUSIONS

6.1 Overall Result

I think the implementation of the IEC61850 LTI tool is a successful project that can aid educators in teaching the specification to students entering the energy field. One of my personal goals for this project was to learn new technologies that are modernizing development on the JVM and on the front end of web applications with Javascript. I believe I have been successful in this regard as well.

Exposure to Spring Boot has yielded some valuable insight about the future of web applications which contain their own bare bones application server. While the official Java EE stack does not appear to be going in this direction, many of the trends in DevOps are moving toward self contained application servers deployed in virtual environments that can be spooled up quickly to respond to heavy loads. The next generation of web application technology from Microsoft, ASP.Net vNext is also trending in this direction.

In addition to adding some experience on the server side of modern web applications, the modifications required to the jQuery.XMLEditor allowed me to explore modern build tools on the client end such as Gulp, which was a new and valuable experience as well.

6.2 LTI Specification

The LTI specification has become widely implemented by LMS providers but the feature set is quite limited. One significant impact of the limited feature set of LTI 1.1 is the grading support, which only allows the return of a single floating point value between 0 and 1. At this point it is not yet possible to return more rich grading based on models defined in Moodle or significant feedback from manual grading.

The LTI 2.0 specification promises to resolve this issue by providing a free text field that may be returned via call to the LMS grade reporting web service, but this is still a very limited feature set that could be improved upon and not all Learning Management Systems implement the LTI 2.0 specification yet.

Unfortunately the LTI 2.0 specification also brings with it added complexity in the integration by requiring the development of a Tool Provider proxy, and the feature set additions aren’t significant enough to warrant taking on the added complexity of the system design.

The shortcomings of the specification combined with the scarcity of libraries provided in the LMS ecosystem to execute these integrations and even less current and up-to-date documentation leave LTI as a poor solution.. but unfortunately one of the only available solutions.

6.3 Challenges

6.3.1 LTI Implementation

The primary use case of OAuth is that of *two-legged* authentication which requires the retrieval of a token which can be approved and signed by the Consumer and returned to the Producer to authorize access to protected resources. This is the model used by public APIs like Twitter, Facebook, DropBox, Google, and Tumblr.

The model of *zero-legged* authentication used by the LTI specification is not so widely used as it requires some implicit level of trust established between the Producer and Consumer. The primary challenge this provides is that existing library support for *zero-legged* OAuth is very sparse and where existing is often tightly coupled to other implementation concerns such as Servlets or Spring Security Filters.

This required that for both accepting the OAuth signed POST parameters from Moodle to initiate the Tool instance and returning the OAuth signed grade report back to Moodle that we implement directly from the OAuth specification which was quite tricky to get correct. The initial estimate on the time investment to implement the signed fetch process ended up being less than half of the actual time required to successfully implement it.

With the documentation also being quite sparse, the most successful strategy ended up being the implementation of a large quantity of logging in the Moodle implementation recording the requests and responses as they were processed on Moodle end and resolving errors until requests could be processed.

6.3.2 jQuery.XMLEditor

The jQuery.XMLEditor component provided some difficulty during the implementation of the thesis work. Initial studies of the component appeared to show a mature implementation ready to be used in this project, however, unexpected challenges arose when attempting to integrate this component into the project.

The specification for the front end requires two instances of the jQuery.XMLEditor component on the Create / Edit Problem Set page. When placing multiple instances of the component, it was observed that changes and selections to one component interacted with both components and broke integration with the underlying CKEditor components.

Upon investigation, it was found that quite much of the functionality inside jQuery.XMLEditor was implemented using jQuery selectors that had no qualifier unique to the component. With investigation showing that no competing component providing the same or even similar feature set existed, it was resolved that the best course of action was to modify the source of the component to bring it into a usable state.

Modifications were made under object oriented design principles where child components instances are injected into parent component instances and related components are interacted with via subcomponent interface, not via jQuery selector.

However, this approach has also proven to have some downsides, as the integration with the underlying CKEditor is not resolved.

With the specification for the page also using a standalone CKEditor, there are some conflicts with how jQuery.XMLEditor handles the child instances of CKEditor that it manages, namely that it can only track via instance number and not via a unique identifier. As an immediate workaround, the CKEditor functionality has been hidden from view and disabled.

6.4 Future Work

6.4.1 Continue Rewrite of jQuery.XMLEditor

The jQuery.XMLEditor component was chosen for this project implementation largely on the fact that it is the only open source component with the feature set required to meet our needs. It does so, but is very limited due to some design flaws as previously discussed. The implementation is also based on the jQuery Widget technology which is quite deprecated in modern web development.

With HTML5 technologies coming to life and wide support on the horizon it might make sense to re-implement the component as an HTML5 WebComponent or alternatively a ReactJs component for use in modern single page application development. Such development should be undertaken with the goal of delivering a self-contained component that does not play in the global namespace of the browser DOM.

6.4.2 Extract 0-Legged OAuth signing library

One of the early difficulties was that there doesn’t exist a good library that focuses on OAuth signing. There are many that focus on providing OAuth security in the token based 2-legged model, such as is needed to authenticate using Facebook or Google, and Spring Security also offers a built in solution. However, none support 0-Legged OAuth signing in a straightforward manner that doesn’t also attempt to intrude and become tightly bound to the application which is not a trade-off worth making for many applications.

A library which provides straightforward 0-Legged OAuth signing and verification might be quite valuable to the open source LMS community.

7 REFERENCES

IMS Global Learning Consortium. IMS Global Learning Tools Interoperability Implementation Guide. Accessed 15.12.2014. http://www.imsglobal.org/lti/ltiv1p2pd/ltiIMGv1p2pd.html

IMS Global Learning Consortium. Developer Tutorials. Accessed 23.04.2015. http://developers.imsglobal.org/tutorials.html

IMS Global Learning Consortium. Developers. Accessed 23.04.2015. http://developers.imsglobal.org/

Moodle. 2015 About Moodle. Accessed 22.04.2015 https://docs.moodle.org/28/en/About_Moodle

Pivotal. Spring Framework Reference. 2014. Accessed 21.04.2015 http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html

VAMK, University of Applied Sciences, Curricula – Energy Technology ICT. Accessed 23.04.2015. http://www.puv.fi/opsweb/?lang=en&code=IITA0503

Zang J. and Gunter C. 2011. IEC 61850 – Communication Networks and Systems in Substations : An Overview of Computer Science. University of Illinois at Urbana-Champaign. PDF. Accessed 23.04.2015. http://seclab.web.cs.illinois.edu/wp-content/uploads/2011/03/iec61850-intro.pdf