Grading Ecosystem - Architecture



Contents

[Introduction 4](#_Toc357978222)

[High-Level Design 5](#_Toc357978223)

[General Overview 5](#_Toc357978224)

[Data Model 8](#_Toc357978225)

[Web Services 10](#_Toc357978226)

[Technologies 12](#_Toc357978227)

[Data Model 13](#_Toc357978228)

[Web Services 13](#_Toc357978229)

[Grading 13](#_Toc357978230)

[Grading Units 13](#_Toc357978231)

[Low-level Design 13](#_Toc357978232)

[Server 13](#_Toc357978233)

[Logical View 13](#_Toc357978234)

[Development View 13](#_Toc357978235)

[Process View 13](#_Toc357978236)

[Physical View 13](#_Toc357978237)

[Scenarios 13](#_Toc357978238)

[Database model 13](#_Toc357978239)

[Admin Web Application 13](#_Toc357978240)

[Integrations 14](#_Toc357978241)

[Spoj0 Integration 14](#_Toc357978242)

[Arena Maycamp Integration 14](#_Toc357978243)

[Eclipse Plug-In 14](#_Toc357978244)

[Security 14](#_Toc357978245)

[Deployment 14](#_Toc357978246)

[Future Considerations 14](#_Toc357978247)

[Visual Studio Add-On 14](#_Toc357978248)

[Advanced Grader 14](#_Toc357978249)

[References 14](#_Toc357978250)

|  |  |  |
| --- | --- | --- |
| **Author** | **Date Changed** | **Note** |
| Martin Toshev | 01.06.2013 | Initial draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Introduction

The purpose of this document is to describe in details the architecture of a system called **Grading Ecosystem** that provides a number of enhancements and new features to a the set of existing grading systems used for both university and school educational purposes in the field of Computer Science. These enhancements and features include:

* unified data model for the various artifacts used by the system (contests, problems);
* easier submission of problems using an administrative web application;
* easier administration and maintenance;
* integration of different grading systems.

Furthermore there are several quality attributes (ordered by priority) considered by the architecture:

* performance – the system should be able to undertake a load of a relative large number of simultaneous users (e.g. 400-500 hundred) that perform intensive tasks;
* security – the system should minimize security concerns regarding the particular user (whether this is a teacher, contestant or administrator);
* scalability – this system should be able to scale both horizontally and vertically, although this is not a strict requirement – it is a necessary in the case when performance should be preserved in a growing community of users of a single deployment. Load balancing is considered;
* availability – since it is important that the system should be highly available (especially when doing an upgrade or there is a peak load during, for example, a contest) - load balancing and failover are considered;
* fault tolerance – the system should perform an adequate level of error handling so that the system does not go down in case of a user-triggered or system fault;
* portability – the system should not be coupled to a particular system

The system is pluggable and allows for integration with various third-party clients and multiple grading systems. In essence – it is a middleware for grading systems.

# High-Level Design

## General Overview

The following diagram gives an overview of the system:

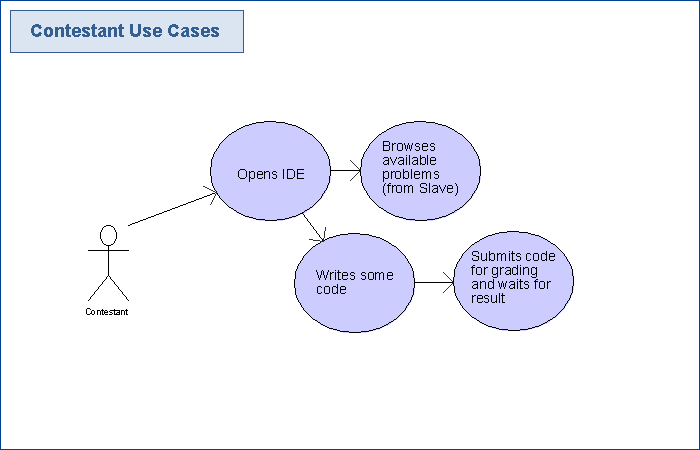


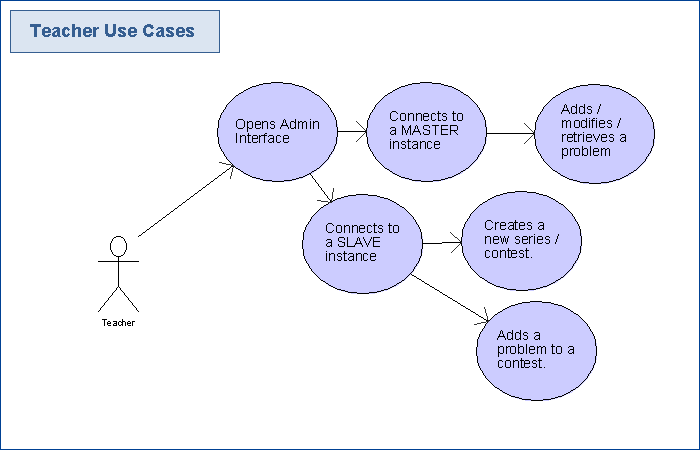
At the core of the system there are several server instances that provide the integration between the various clients and the graders:

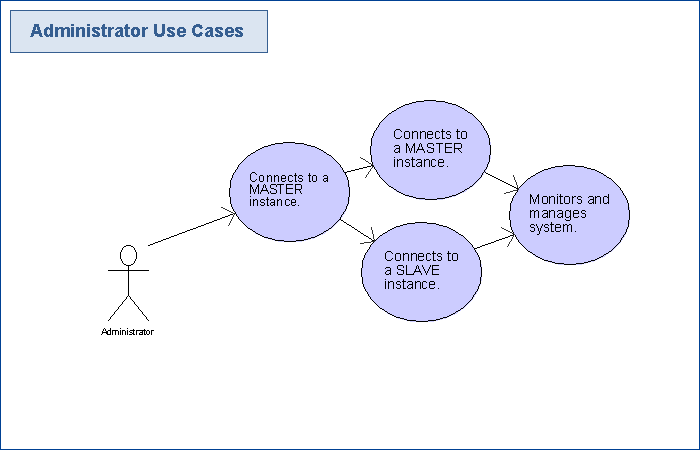
* MASTER server – this instance provides a set of RESTful web services for CRUD operations on problems (part of the Grading Units Façade). This instance is intended to serve as a central repository for problem data – many SLAVE instances could link to a single MASTER instance in order to retrieve problems. Problems are submitted to the MASTER instance only from users with role ‘’teacher’ (see the section on security for more details) from the administrative interface. Users with role ‘admin’ are able to manage the MASTER instance from the administrative interface. A designated Mongo database is used to store the problem data. The MASTER instances may retrieve and convert problem data from the various graders.

* SLAVE server - this instance provides a set of RESTful web services for retrieving CRUD operations on a series of contests (here a series could be a particular course, for example), a particular contest in a series, or a particular problem in a contest and for submission of problem solutions in the form of source code. This instance makes use of both web services facades. The SLAVE instance retrieves available problems from the MASTER instance and adds them to a particular contest. Third party applications targeted for contestants (such as the various IDE integration provided to contestants) connect to a SLAVE instance for problem retrieval and submission. Users with role ‘teacher’ are able to create series, contests and add problems to contests via the administrative interface. The SLAVE instances submit solutions to the graders.

There could be various third party clients for viewing problems and submitting solutions (e.g. an Eclipse Plug-In and a Visual Studio Add-On). There is an administrative web application that can be used by teachers and administrators to manage the system. The following diagrams display the typical use cases for the different types of users:







We will further describe in details the low-level details of the particular components using the 4+1 architectural view model in a subsequent section.

## Data Model

Before we can give a more detailed overview on the system internals the data model used by the application will be outlined. We have reused the CORE (Contests Repository) model outlined in [1]. Since the outlined data is presented using JSON format (JavaScript Object Notation) we will be using the same format. This format will also serve as a basis for describing later the schema-less structure of the data as persisted in a Mongo database.

We distinguish between the following data types:

**Series** – defines a grouping of contests   
Example:

{

"type": "standard",

"title": "Design and analyses of algorithms - 2013",

"about": " Lectors :... , Hours :... etc .",

"notes": "( Some sensitive information )"

"contest\_order" : [ "homework 1"]

"parent": "Courses 2013"

}

**Contest** – defines a programming contest that has a set of problems  
Example:

{

"type": "standard",

"title": "Homework 1",

"start\_time": "2012 -06 -28 T09 :00:00 Z",

"duration": 300,

"about": "",

"grading\_style": "acm",

"problem\_order ": [" fish ", " honey ", " swim "],

"problem\_scores": [120 , 150 , 80]

}

**Problem** – defines a particular problem  
Example:

{

"type": "standard",

"title": "hw1 - problem -A",

"description": "... problem description ...",

"time\_limit": 4.5,

"memory\_limit": 64,

"origin": "Taken from Contest -X",

"categories": ["Dynamic Programming"],

"authors":["author1", "author2"],

"test\_weights": [5, 10, 10, 10, 15] ,

"checker": "diff"

}

**Problem** **Category** – defines the programing category of a problem  
Example:

{

"name": "Spanning Trees"

"description": "Problems on spanning trees"

"parent": "Graph Theory"

}

**Grader Instance** – defines a grader instance  
Example:

{

"type": "spoj0"

"name": "spoj0-1"

"URL": "http://judge.openfmi.net:9080/spoj0/"

}

**Submission** – defines a problem submission from a contestant  
Example:

{

"handle": "martin",

"grader\_type": "spoj0",

"grader\_handle": "martin\_spoj0"

"series": "Design and analyses of algorithms - 2013"

"contest": "Homework 1"

"language" : "Java"

"problem": "hw1 - problem -A"

"source": "... source code ...",

"status": "ok",

"results": ""

}

**Grader User** – defines a user handle for a particular grader  
Example:

{

"grader\_type" : "spoj0"

"handle" : "martin\_spoj0"

"pass": "skajdUOdsa77sadKL"

}

**User** – defines a user of the system  
Example:

{

"handle" : "martin"

"name" : "Martin Toshev"

"pass" : "cX6kajsdU76"

"role" : "contestant"

"details" : ""

"permissions" : "rw"

}

## Web Services

The following section defines the web service interfaces used by the application (RESTful web services for grading units and problem grading).

For the purpose of simplicity request/response payloads are omitted (the format follows the data model presented in the previous section) – typically JSON/XML is used as the format of sending the REST requests.

Services available from MASTER instance:

* Retrieve available top level problem category IDs:

**GET http://<server\_host>:<server\_port>/categories**

* Retrieve available child category IDs:

**GET http://<server\_host>:<server\_port>/categories/{id}/categories**

* Retrieve available category:

**GET http://<server\_host>:<server\_port>/categories/{id}**

* Create a category:

**PUT http://<server\_host>:<server\_port>/categories**

* Edit or create a category:

**POST http://<server\_host>:<server\_port>/categories/{id}**

* Delete a category:

**DELETE http://<server\_host>:<server\_port>/categories/{id}**

* Retrieve available problem IDs:

**GET http://<server\_host>:<server\_port>/problems**

Additional request parameters:

**type** – problem type used to filters the retrieved problems   
**categories** – comma-separated list of categories used to filter the problems  
**authors** – comma-separated list of authors used to filter the problems

* Retrieve a particular problem:

**GET http://<server\_host>:<server\_port>/problems/{id}**

* Create a problem:

**PUT http://<server\_host>:<server\_port>/problems**

* Edit or create a problem:

**POST http://<server\_host>:<server\_port>/problems**

* Delete a problem:

**DELETE http://<server\_host>:<server\_port>/problems**

Services available from SLAVE instance:

* Retrieve top-level series IDs:

**GET http://<server\_host>:<server\_port>/series**

* Retrieve child series IDs:

**GET http://<server\_host>:<server\_port>/series/{id}/<series>**

* Retrieve a particular series:

**GET http://<server\_host>:<server\_port>/series/{id}**

* Create series:

**PUT http://<server\_host>:<server\_port>/series**

* Edit or create series:

**POST http://<server\_host>:<server\_port>/series**

* Delete series:

**POST http://<server\_host>:<server\_port>/series**

* Retrieve contest IDs from a series:

**GET http://<server\_host>:<server\_port>/series/{ id}/contests**

* Retrieve a particular contest from a series:

**GET http://<server\_host>:<server\_port>/series/{id}/contests/{id}**

* Create a contest in a series:

**POST http://<server\_host>:<server\_port>/series/{id}/contests**

* Edit or create a contest in a series:

**PUT http://<server\_host>:<server\_port>/series/{id}/contests**

* Delete a contest from a series:

**DELETE http://<server\_host>:<server\_port>/series/{id}/contests**

* Retrieve problem IDs from a contest:

**GET http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems**

* Retrieve a problem from a contest:

**GET http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems/{id}**

* Create a problem in a contest:

**DELETE http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems**

* Edit or create a problem from a contest:

**DELETE http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems**

* Delete a problem from a contest:

**DELETE http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems**

* Retrieve submission IDs for a problem:

**GET http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems/{id}/submissions**

* Retrieve a submission for a problem:

**GET http://<server\_host>:<server\_port>/series/{id}/contests/{id}/problems/{id}/submissions/{id}**

* Submit a problem for grading:

**POST http://<server\_host>:<server\_port>/submissions**

## 

## Technologies

The following technologies are used in the project:

* Java SE 7.0 – for the overall implementation of the server application, admin web application and the Eclipseintegration;
* Maven – for building the various projects;
* Spring – a DI (dependency injection) container used to provide runtime configuration management for the various projects;
* Apache CXF – web service framework for building SOAP/RESTful web service (provides integration with Spring);
* Mongo – a NoSQL database (document store) that stored data in the form of JSON documents. All server data (including problem data) is stored in Mongo.
* GIT – a VCS (version-control system) used to store the documentation and source code of the system (a GitHub (see [2]) public repository is used for the purpose).

Additionally the following technologies are used for the various clients:

* Eclipse PDE – for developing of the Eclipse integration;
* Tycho – provides Maven integration for OSGi bundles (used to build the Eclipse-related projects from Maven);
* Visual Studio Add-in API – for developing the Visual Studio integration;
* Apache Velocity – for problem templates in the admin interface;
* iText – for generating PDF documents from the HTML pages of the problems.

## Web Services

### Grading

The web services used to perform the grading of solutions targeted for a particular grader are SOAP services defined as follows:

### Grading Units

The web services used to create series

# Low-level Design

## Server

### Logical View

### Development View

### Process View

### Physical View

### Scenarios

### Database model

## Admin Web Application

See document **Grading Ecosystem - Admin Web Application**.

# Integrations

## Spoj0 Integration

See document **Grading Ecosystem - Spoj0 Integration**.

## Arena Maycamp Integration

See document **Grading Ecosystem - Arena Maycamp Integration**.

## Eclipse Plug-In

See document **Grading Ecosystem - Eclipse Plug-In Integration**

# Security

// TODO

# Deployment

This section describes the deployment of the system.

// TODO – describe deployment specifics and installer

# Future Considerations

## Visual Studio Add-On

// TODO

## Advanced Grader

// TODO

# References

[1] CORE – a multi-purpose programming contest repository system, Chaushev, Manev, Sredkov  
<http://www.math.bas.bg/smb/2012_PK/tom_2012/pdf/219-224.pdf>

[2] Grading Ecosystem Project Repository  
<https://github.com/martinfmi/grading_ecosystem>