Software Requirements Document

2IPE0 Software/Web Engineering Project Great Graders

A. Agaronian
1017525

C.W.S. Freyer
1036039

C. Gutiérrez Bierbooms
1028054

T.P.H. Hoeijmakers
0996802

T.K.H.G. Jansen
1003562

T. Kafoe
1194252

I. Makantasis
1002480

K. Mankevic
1036163

R.D. Sinx
1001972

I.M. Smits
1010890

K.R. Vlaswinkel
1016271

Great Graders

Quartile 4 – Group 5

Date
July 3, 2019

Version
1.0

Managers
A.S. Brouwers
G. Walravens

Supervisor
dr. S. Roubtsov

Customer
B. Corbijn, G. Vaessen

Abstract

This document is the Software Requirements Document (SRD) for Grade Calculation, developed by Great Graders. Grade Calculation is a Learning Tools Interoperability (LTI) plugin for the Learning Management System (LMS) Canvas. The requirements for this SRD correspond with the requirements listed in the User Requirements Document (URD) for Grade Calculation [1].
Grade Calculation is developed as part of the Software Engineering Project (2IPE0) at the Technical University of Eindhoven. Moreover, this document complies with the ESA software standards [2].

Contents

Document Status Sheet

General

Document title:	Software Requirements Document	
Document identifier:	SRD/1.0	
Authors:	A. Agaronian	1017525
	C.W.S. Freyer	1036039
	C. Gutiérrez Bierbooms	1028054
	T.P.H. Hoeijmakers	0996802
	T.K.H.G. Jansen	1003562
	T. Kafoe	1194252
	I. Makantasis	1002480
	K. Mankevic	1036163
	R.D. Sinx	1001972
	I.M. Smits	1010890
	K.R. Vlaswinkel	1016271
Document status:	Done	

Document History

Version	Date	Authors	Reason
0.1	09-05-2019	All	Setting up framework document
0.2	13-06-2019	All	First draft
0.3	27-06-2019	All	Implemented feedback and added missing sections
1.0	01-07-2019	K.R. Vlaswinkel	Implement feedback

Document Change Record

Version	Date	Section	Reason
0.1	09-05-2019	All	Setting up framework document
0.2	13-06-2019	All	First draft
0.3	27-06-2019	All	Second draft
1.0	01-07-2019	Performance	Implement feedback

Chapter 1

Introduction

1.1 Purpose

This Software Requirements Document (SRD) contains the software requirements for the Grade Calculation plugin. The requirements in this document are a translation of the user requirements listed in Chapter 3 of the User Requirements Document (URD) for Grade Calculation [1]. The URD formulated the desired functionality of the plugin, whereas the SRD describes how this functionality is to be implemented. These requirements are developed in accordance with B. Corbijn and G. Vaessen, the customers and commissioners of the project.

1.2 Scope

Great Graders is a team of Bachelor students working on a Software Engineering Project for the TU/e, B. Corbijn and G. Vaessen. B. Corbijn and G. Vaessen are representatives of Drieam B.V. (further referred to as Drieam), a company that provides support to European universities and other educational institutions when operating Canvas.
Canvas is a Learning Management System (LMS) which makes the teaching and learning processes easier. It is an easy and convenient platform for educators to organize courses for students. Additionally, it offers a vast number of functionalities for submitting assessments and providing feedback on these submissions using the SpeedGrader.
The goal of Great Graders is to develop a Learning Tools Interoperability (LTI) plugin for Canvas that streamlines the grading process for educational institutes from grading individual assignments in Canvas to publishing final grades in the Student Information Systems (SIS).

1.3 List of Definitions

1.3.1 Definitions

Canvas	An open-source Learning Management System (LMS) developed by Instructure [3].
MoSCoW	A prioritization technique used in software development to reach a common understanding with stakeholders on the importance of each requirement.
Submission	The work that the student submits.
Assessment	Any work a student can be graded on, e.g. Quiz, Written Exam, Report.
Calculation method	A rule used to compute a partial score.
Marks	The judgment of the quality assigned by the teacher to a submission of a student on a specific assessment as defined by the mark type.
Score	Either a final score or a partial score.
Partial scores	The result of applying a calculation method on a set of partial scores and/or marks.
Final score	The result of applying a final score structure.
Partial grade	A partial score on which a grading scheme has been applied.
Final grade	A final score on which a grading scheme has been applied. This is usually the end result of the course.
Final score structure	A collection of calculation methods which takes marks as input and outputs a final score.
Grading scheme	The conversion process from a score to a grade by applying a grading standard, e.g. into a letter grade or into a grade in the range 0-10.
Description of final score structure	A brief outline explaining how the final score is calculated added by the teacher.
Score type	A data item that determines the calculation range of the score (e.g. 'complete_incomplete', 'percent', 'points').
Mark type	A data item that determines the calculation range of the mark (e.g. 'complete_incomplete', 'percent', 'points').
Final grade type	The type of the final grade. (e.g. letter grade (US), 1-10 rounding to the nearest integer, GPA scale).
Calculation range	A range of all values that a mark or score can take.
Contents of an assessment	Information about the assessment containing assessment name, calculation range, mark type, SIS ID, and the students' corresponding marks.

+ and - system	The system used in Dutch primary schools where a student can get a $n+$ or $n-$ grade, where n is an integer between 0 and 10 (e.g. 8+ or 8-, where these grades correspond to 8.25 and 7.75 respectively).
Letter system following British standards	British Letters grade system as defined by the Grade Point Average system [4].
Letter system following American standards	Letter grade system used as default by Canvas [5].
CSV file	Comma-separated values file, a file type used to store tabular data.
LTI Launch	An LTI launch is used to load an LTI plugin from an LMS, such as Canvas. When the Canvas page is loaded, a hidden HTML form is submitted as a POST request to the LTI plugin. From this request the page is able to retrieve the necessary information to store the user's data and page context. A signature is included to verify the authenticity of the launch data.

1.3.2 Abbreviations and Acronyms

URD	User Requirement Document
SRD	Software Requirement Document
TU/e	Technical University of Eindhoven
LMS	Learning Management System
LTI	Learning Tools Interoperability
SIS	Student Information System
API	Application Programming Interface

1.4 List of References

[1] *Grade Calculation, User requirements document*. Eindhoven University of Technology, 2019.

[2] *E. B. for Software Standardisation and Control*. ESA software engineering standards, 1991.

[3] Ellis, Ryann K., *Field Guide to Learning Management*. ASTD Learning Circuits, 2009.

[4] `https://www.heacademy.ac.uk/system/files/resources/Guide%20on%20grade%20point%20average%20for%20students_0.pdf`.

[5] `https://community.canvaslms.com/docs/DOC-13067-4152206341`.

[6] *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*, European Union, Apr. 2016.

[7] D. Haughey, *Moscow method*, `https://www.projectsmart.co.uk/moscow-method.php`, [Online; accessed 26-April-2019]. [Online]. Available: `https://www.projectsmart.co.uk/moscow-method.php`.

1.5 Overview

The remainder of this document consists of three chapters.

Chapter 2 is a general description of Grade Calculation. In Section 2.1, the context of Grade Calculation in relation to other current projects is detailed. In Section 2.2 details about the context of Grade Calculation in relation to past and future projects are presented. Section 2.3 consists of a general description of the function and purpose of Grade Calculation. Next, Section 2.4 contains the general overview of the operational environment. In Section 2.5, the relations between Grade Calculation and other systems are described. Section 2.6 describes the general constraints that Grade Calculation must comply with. Lastly, Section 2.7 is a description of the logical model, including an environment model, a class diagram, a data model, and sequence diagrams.

Chapter 3 includes a detailed list of software requirements. These are divided into two subsections; functional, and non-functional requirements. In this section, the implementation of the requirements that were agreed upon with B. Corbijn, the customer, is defined.

Chapter 4 includes a traceability matrix, to ensure that all user requirements from the URD [1] are included as a software requirement in the SRD. This matrix maps each user requirement to its matching software requirement(s) and vice-versa.

Lastly, in the Appendices, UI mockups for both the Student and the Teacher Interface, as well as transition diagrams between views are provided.

Chapter 2

General Description

2.1 Relation to Current Projects

Grade Calculation extends the functionality of Canvas, an LMS that allows teachers and students to share content and interact throughout the duration of a course. Since Grade Calculation will be replacing and extending the grading functionality of Canvas, the current grading functionality of Canvas is closely related to Grade Calculation. Thus, Grade Calculation will be tightly integrated with Canvas. Yet, Grade Calculation will allow for more flexibility and provide a wider range of grading options required by educational institutions, such as mark adjustment.

SISs, such as Osiris, are also strongly co-related to Grade Calculation. Canvas currently offers functionality to export marks and grades as a CSV file, which could potentially be imported into a SIS. The SIS can then either import the grades directly or the final score can be calculated externally and entered into the SIS. However, both options create unnecessary inconveniences for the administration. Firstly, Canvas does not export the grades in a format that is accepted by most SISs and thus importing the grades into a SIS cannot be done directly. Secondly, calculating the grades externally and manually entering them into OSIRIS is a highly error prone process. Therefore, Grade Calculation allows for a centralized and flexible means to calculate the grades that can be directly exported to a SIS. Moreover, Grade Calculation extends the functionality of Canvas by providing mark adjustments to normalize marks and allows for an easy way to create multiple attempts for assessments, which are both key factors missing in the current design of Canvas.

Blackboard, another LMS, has more extensive grading functionality compared to Canvas. It supports calculated columns, including averages, minimum/maximum, totals, and weights. However, despite offering greater grading functionality, it requires switching to a completely different LMS, which might not have the same required functionality in other areas of use. Therefore, it is unlikely that educational institutions would shift to a different LMS just for the convenience of a slightly better grading functionality. Especially if the functionality of Canvas could be extended directly using the Grade Calculation plugin.

2.2 Relation of Predecessor and Successor Projects

There are no formal predecessor projects for Grade Calculation. However, Grade Calculation will extend the current Canvas workflow that is embedded into Canvas. Therefore, Grade

Calculation will rely on some of the current Canvas workflow elements and adjust the provided functionality based on the feedback of potential clients. Thus, the Canvas workflow will serve as a basis and as a data entry to get Canvas assignments in Grade Calculation.

After the development of Grade Calculation, Drieam will create their own product that will be based on Grade Calculation. Therefore depending on the final product, functionalities of Grade Calculation might be extended, and existing functionalities are likely to be rewritten. Grade Calculation will serve as a proof of concept that Drieam will use as a base for their plugin. Drieam is already in contact with universities that could be potential customers for their plugin, and thus Grade Calculation has also complied with the requirements that these potential customers expect from such a plugin. Complying with these requirements will help in making Grade Calculation a proper base for the successor project of Drieam.

2.3 Function and purpose

Nowadays, many universities use Canvas as their LMS. Canvas offers functionality for submitting assessments, grading them, publishing grades and providing feedback. As this information is already present in Canvas, it is reasonable to want to centralize all the information of the course to the students by providing them with their marks and grades. However, Canvas does not have sufficient capabilities to calculate partial or final grades. At many universities this creates an inconvenient situation for the administration who need to execute certain tasks such as awarding ECTS credits. Grade Calculation will offer a more extensive alternative for the built-in grading workflow of Canvas and will provide a way to facilitate an efficient grading process. It will overcome the existing challenges in Canvas by the implementation of the following functions:

- Define a final score structure: Grade Calculation allows for defining complex score structures that are evaluated recursively.
- Final grade calculation: Grade Calculation can calculate, in a flexible way, the final grade of a course based on marks obtained during the course and a grading scheme of choice.
- Importing marks: Teachers can import marks in Grade Calculation. These marks can be obtained from assignments in Canvas, or imported from an external data source. Optionally a final score structure and/or grading scheme can be applied to these marks resulting in grades that can be published to students.
- Retrieval of data: Grade Calculation provides an endpoint so that the course results can be retrieved, with the possibility of being submitted to the SIS in a confined manner. This would allow for reducing manual labour in the registration process, to a minimum (i.e. if SIS supports automatic grade registration from a data source, it can be connected to Grade Calculation so that no manual work needs to be done).

2.4 Environment

Grade Calculation is going to be used in a web-based environment. Since the tool runs inside the Canvas environment it is imperative that it supports all the browsers currently supported by Canvas. The plugin itself will run on an external server and will be embedded into Canvas by means of a frame. The plugin should therefore be developed such that it can be used correctly from within a frame. Furthermore, since the plugin does not run on the same server as Canvas, the external server does need to communicate with Canvas through its API.

Considering the fact that Canvas is developed using the Ruby on Rails framework and React, and the fact that Drieam works with these languages, Grade Calculation uses the same framework in order to support maintainability and to simplify integration.

Regarding data management, PostgreSQL is used due to the fact that Drieam's platform makes use of this database. In addition, PostgreSQL provides various features that fit Grade Calculation's requirements.

2.5 Relation to Other Systems

The main system related to Grade Calculation is Canvas. In fact, Grade Calculation depends on Canvas in its entirety, as it is a plugin made specifically for Canvas. Grade Calculation is not standalone and will not function if Canvas is offline or otherwise unavailable.

2.5.1 Canvas

Grade Calculation is an LTI plugin for Canvas that streamlines the grading workflow for educational institutes. Currently there are no other plugins that attempt to replace the grading workflow in Canvas. However, Canvas itself offers a grading environment that also includes the calculation of grades, importing, and exporting grades. Yet, the grading environment of Canvas is limited to a handful of actions. The limited possibilities of the grade process capabilities in Canvas gave rise to this project. Grade Calculation will replace the built-in grade display and calculation of Canvas itself. However, it is important to note that Canvas could potentially extend their capabilities regarding grading workflow, which might decrease the necessity of Grade Calculation. However, no plans for such an extension have been announced, and it is unlikely that all functionality will be added to Canvas in the near future.

2.6 General Constraints

2.6.1 Security and privacy

Grade Calculation should highly value the constraints of security and privacy. Since the plugin will run on a separate server, steps need to be taken to ensure that the data is secured. In order to prevent data from being exposed to third parties information needs to be transferred to the server in a secure way. It is therefore clear that only HTTPS connections should be used between Canvas and the server, such that only encrypted data-communication is allowed. This will also guarantee that information is not altered during transmission.

Users of Grade Calculation should not have access to information they are not authorized to view. Therefore, Grade Calculation needs to inherit the permission scheme of Canvas and only allow for data retrieval that adheres to the permissions of the user. The infamous SQL injection could harm these permissions and possibly enable users to view data they are not authorized to view. Moreover, SQL injection could harm integrity of data. Therefore, input sanitation should be performed to prevent SQL injection and other such injection attacks.

Since Grade Calculation will be used by organizations, Grade Calculation will also need to comply with the European General Data Protection Regulations (GDPR) [6]. This will create some additional requirements for the software. In the case of Grade Calculation the most relevant article in the GDPR is Art 25. (Data protection by design and by default). An implication of this article is the limited time frame that user data can be stored after graduation. Therefore, it should be possible to remove the data after this time period ended. The other rights of the data subjected in Ch. 2 of the GDPR are not strictly required, because either these are to be implemented by the educational institution or because they are not required due to the basis of processing.

Furthermore, only data that is strictly required for the proper functioning of Grade Calculation should be stored. Information that can be fetched from the Canvas API should only be stored if this is needed for the proper functioning of Grade Calculation. Thus, if it does not cause too much overhead, data is only fetched from the Canvas API and not stored by Grade Calculation.

Art 25. of the GDPR is taken into account and as such, the specified technical and organizational techniques are used to protect data, for example pseudonymisation.

2.6.2 Usability

On the web, users are used to navigating over sites and web apps without needing additional explanation. It is therefore clear that Grade Calculation should work intuitively in order for Grade Calculation to be successful. Grade Calculation will accomplish this by trying to give an acquainted feel to the users of Grade Calculation. The plugin will comply with the style of Canvas by using native Canvas UI components, or UI components that provide a similar look and feel, to increase usability. For components not available from Canvas, existing design libraries such as Ant Design will be used to again make the user feel familiar with the interactions of the plugin.

A main task of a Teacher user of Grade Calculation is to define score structures. Multiple options can be used to define these score structures. By providing clear input components, the Teacher should know with what elements they can interact with. Furthermore, text labels will be placed appropriately so that it is more clear what the results of an interaction is. For some teachers it might not be clear at first glance how all options can be used. Thus, a help box on the page to define a score structure is available to provide some more textual information about how to enter a final score structure. Furthermore, a user manual is available for more in depth information on all the options and how they can be used.

2.6.3 Environment

Grade Calculation is used in a web-based environment. As Grade Calculation is a Canvas LTI plugin, it should support all the browsers which are supported by Canvas. For Grade Calculation the supported browsers are: Google Chrome version 72/73, Mozilla Firefox version 65/66, Safari version 11/12 for MacOS, Microsoft Internet Explorer version 11, and Microsoft Edge version 42/44.

Grade Calculation is developed using Ruby, React, the Ruby on Rails framework, and the Drieam framework. This is due to the fact that Canvas is developed using Ruby, React, and Ruby on Rails, which simplifies integrating the LTI to Canvas and maintaining Grade Calculation with respect to updates of Canvas. Additionally, the Drieam framework helps accelerate the creation of the LTI plugin as Drieam designed it to be used in a wide variety of LTI plugins.

Data management in Grade Calculation is done using PostgreSQL. PostgreSQL is a relational database management system to store information. It was chosen as it provided Great Graders with all the functionality needed to meet Grade Calculation requirements. Moreover, PostgreSQL is used within Drieam's platform.

2.6.4 Language

Canvas supports a variety of languages and is internationally used. As English is the default language of Canvas and is the standard international language, it is important that Grade Calculation is implemented in English. Thus, Grade Calculation supports the English language and can be extended at a later stage to include different languages. The front-end uses `i18n-react` which also simplifies new language extensions desired in the future.

2.6.5 Performance

A Canvas instance can have up to 50 thousand users. As Grade Calculation deals with the marks, scores, and grades of users, Grade Calculation will also deal with a large part of these 50 thousand users. This means Grade Calculation should be able to support a vast amount of users and be able to process the data belonging to these users.

It is expected that the final score calculation will calculate the grades for a single student within 2 seconds. Importing marks from a CSV file is expected to take at most 60 seconds, while exporting grades is expected to complete within 5 minutes. Changing the visibility of marks or whether they are muted will be visible to the student within 30 seconds. These estimations are within the bounds formulated in the URD [1].

With regards to response times, it is expected that the front-end assets are compiled within 300,000 milliseconds and that for all subsequent actions the web interface will react within 200 milliseconds per action. These statistics are all based on the maximum response times allowed while enabling a user-friendly and usable product.

Grade Calculation uses PostgreSQL as its relational database. As Grade Calculation needs to support a vast amount of users all their data will have to be processed as well. This means the PostgreSQL database should be able to handle a vast amount of data and the queries that are run on it.

2.6.6 Reliability

It is important to guarantee the functioning of Grade Calculation. If teaching institutes are to use Grade Calculation, they will require a reliable tool that has a high uptime. When Grade Calculation would be down, it would significantly harm the grading administration, as by design many stakeholders will rely on it.

Moreover, it is important that grades of students are correct and up to date. If a data-source provides updated information it is the task of Grade Calculation to ensure that the updated information is used in its process. For example, if a teacher is editing a final score structure for one course, Grade Calculation should adjust the final score structure for only that course.

2.7 Model Description

2.7.1 Environment Model

In *Figure* 2.1, one can see the Grade Calculation in its environment. The area in grey defines the scope of Grade Calculation. The remainder is part of the existing environment.

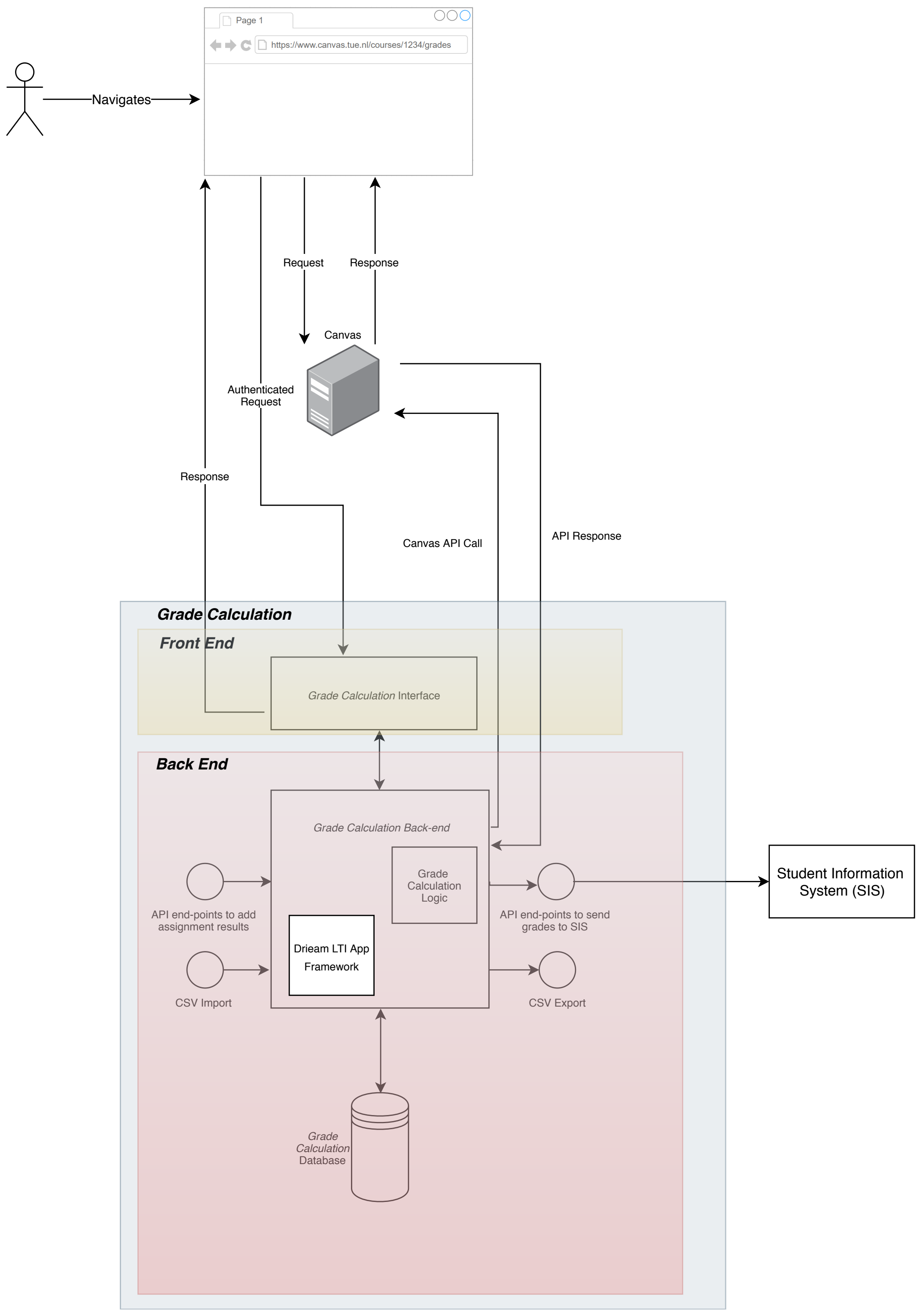

Page 1
https://www.canvas.tue.nl/courses/1234/grades
Navigates
Request
Response
Canvas
Authenticated Request
Response
Canvas API Call
API Response
Grade Calculation
Front End
Grade Calculation Interface
Back End
Grade Calculation Back-end
Grade Calculation Logic
API end-points to add assignment results
Drieam LTI App Framework
CSV Import
API end-points to send grades to SIS
CSV Export
Student Information System (SIS)
Grade Calculation Database

Figure 2.1: The Grade Calculation plugin in the environment

2.7.1.1 Canvas

Grade Calculation is a plugin within Canvas. A user navigates on the Canvas website and requests to the Canvas server will be sent accordingly. The Canvas server will handle these requests by sending the appropriate responses. The plugin will be hosted on an external server and loaded in a frame. Therefore, some requests resulting from the Canvas responses will then be sent to Grade Calculation in order to load the plugin into the frame. The responses from the Canvas server allow for making authenticated requests towards Grade Calculation so that Grade Calculation can provide user-specific data.

2.7.1.2 Student Information Systems (SIS)

SISs are software designed web-based applications that introduce a useful and structured information flow environment for students, teachers, and the administration of a teaching institute. A SIS will be able to obtain grades for the students from Grade Calculation.

2.7.1.3 Grade Calculation Interface

The front-end of Grade Calculation takes care of the proper visualization of data in the Grade Calculation Interface, which is embedded into the Canvas website. The front-end handles the user interaction by sending the appropriate requests to the back-end.

2.7.1.4 Drieam LTI Framework

The back-end of Grade Calculation will rely on the Drieam LTI framework. This framework is provided by Drieam. The Drieam LTI framework allows for easy coupling with Canvas, facilitating processes such as authentication and data retrieval.

2.7.1.5 Grade Calculation back-end

The Grade Calculation back-end implements all server side functionality to be used by the front-end. It serves as an extra control layer between the front-end and the back-end database, and it is responsible for authentication (using the Drieam LTI framework), providing the front-end with the correct data to be displayed, importing data from Canvas, and reading from and writing to the database.

2.7.1.5.1 Grade Calculation Logic The Grade Calculation Logic is one of the major modules of the Grade Calculation Back-end which implements all functionality with regard to the calculation of the final grade. It receives the necessary data to perform calculations and returns the calculated grades for each student.

2.7.1.6 Grade Calculation's Database

The back-end database stores all necessary information that is needed for the full functionality of Grade Calculation. It is a stand-alone database without any dependency on an external database. The database will contain information about the courses, the students, the assessments, the marks, and the grades.

2.7.1.7 API endpoints

The back-end exposes two API endpoints: one that allows other systems to input marks into Grade Calculation, and one that allows for obtaining partial and final grades from Grade Calculation.

2.7.1.8 CSV import and export

Apart from the endpoints, the back-end has the ability to import CSV files with marks and export CSV files with grades.

2.7.2 Class Diagram

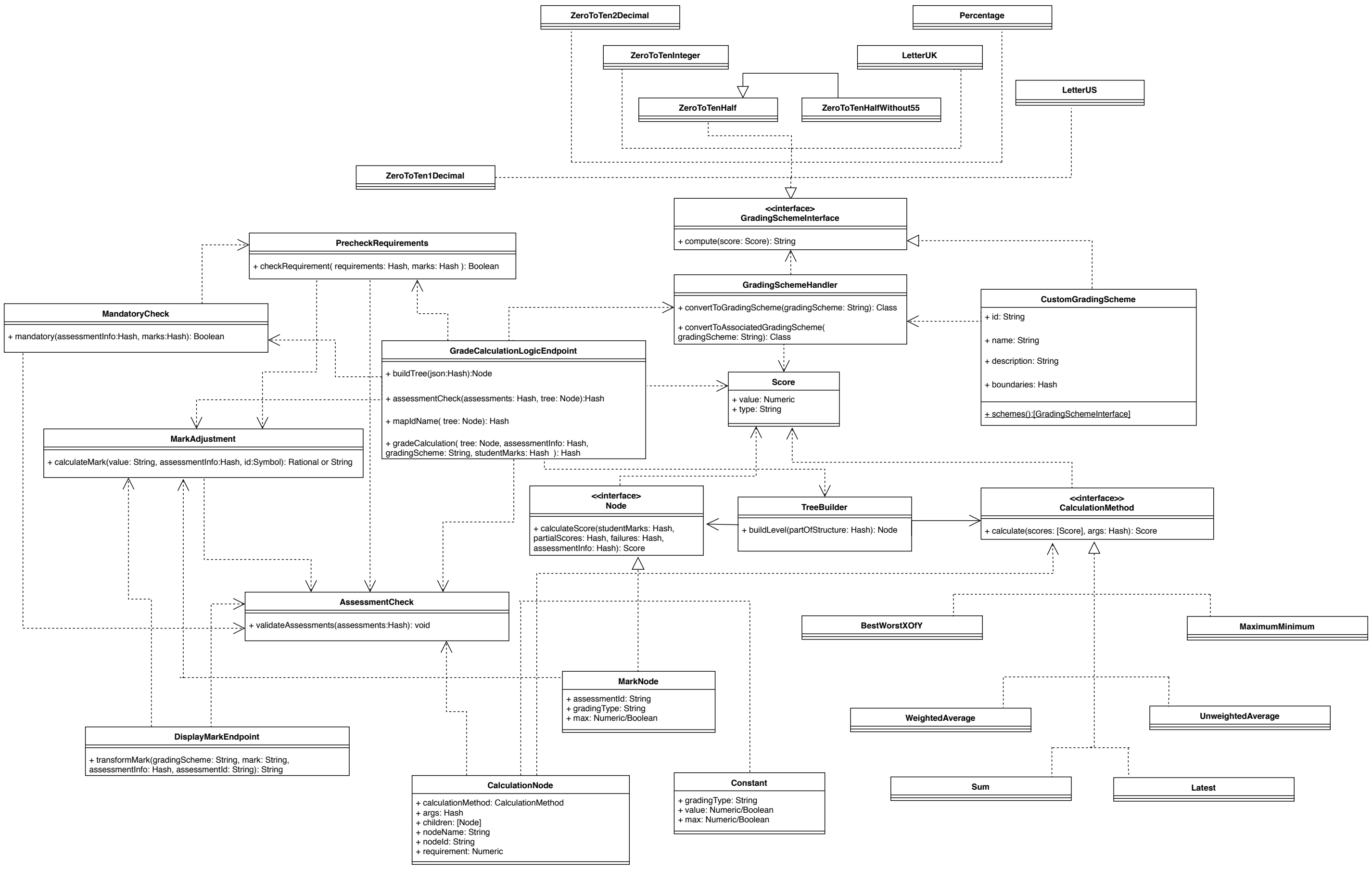

ZeroToTen2Decimal
Percentage
ZeroToTenInteger
LetterUK
LetterUS
ZeroToTenHalf
ZeroToTenHalfWithout55
ZeroToTen1Decimal
<<interface>> GradingSchemeInterface
+ compute(score: Score): String
PrecheckRequirements
+ checkRequirement(requirements: Hash, marks: Hash): Boolean
GradingSchemeHandler
+ convertToGradingScheme(gradingScheme: String): Class
+ convertToAssociatedGradingScheme(gradingScheme: String): Class
CustomGradingScheme
+ id: String
+ name: String
+ description: String
+ boundaries: Hash
+ schemes():[GradingSchemeInterface]
MandatoryCheck
+ mandatory(assessmentInfo:Hash, marks:Hash): Boolean
GradeCalculationLogicEndpoint
+ buildTree(json:Hash):Node
+ assessmentCheck(assessments: Hash, tree: Node):Hash
+ mapIdName(tree: Node): Hash
+ gradeCalculation(tree: Node, assessmentInfo: Hash, gradingScheme: String, studentMarks: Hash): Hash
Score
+ value: Numeric
+ type: String
MarkAdjustment
+ calculateMark(value: String, assessmentInfo:Hash, id:Symbol): Rational or String
<<interface>> Node
+ calculateScore(studentMarks: Hash, partialScores: Hash, failures: Hash, assessmentInfo: Hash): Score
TreeBuilder
+ buildLevel(partOfStructure: Hash): Node
<<interface>> CalculationMethod
+ calculate(scores: [Score], args: Hash): Score
AssessmentCheck
+ validateAssessments(assessments:Hash): void
BestWorstXOfY
MaximumMinimum
MarkNode
+ assessmentId: String
+ gradingType: String
+ max: Numeric/Boolean
WeightedAverage
UnweightedAverage
DisplayMarkEndpoint
+ transformMark(gradingScheme: String, mark: String, assessmentInfo: Hash, assessmentId: String): String
CalculationNode
+ calculationMethod: CalculationMethod
+ args: Hash
+ children: [Node]
+ nodeName: String
+ nodeId: String
+ requirement: Numeric
Constant
+ gradingType: String
+ value: Numeric/Boolean
+ max: Numeric/Boolean
Sum
Latest

Figure 2.2: UML Grade Calculation Logic Class diagram

Sections 2.7.2.1 to 2.7.2.5 refer to the class diagram for the Grade Calculation logic module found in Figure 2.2.

2.7.2.1 Tree construction

2.7.2.1.1 TreeBuilder The TreeBuilder class offers a single functionality: its purpose is to turn a valid input hash into the tree structure that Grade Calculation uses to calculate student

grades.
This class has the following method:

- *buildLevel(partOfStructure:* `Hash`): `Node`
  Takes in a valid hash according to specifications, and uses it to recursively construct the tree structure. It returns the root Node of the tree.

2.7.2.1.2 Node

The Node is an interface that is used to define the methods of the other tree node classes.
This interface has the following method:

- *calculateScore(studentMarks:* `Hash`, *partialScores:* `Hash`, *failures:* `Hash`, *assessmentInfo:* `Hash`): `Score`
  Every Node should in some way, using its attributes and/or parameters, be able to produce a Score object.

2.7.2.1.3 CalculationNode

The CalculationNode class represents the calculation of a partial score within the final score structure.
An instance of this class has the following attributes:

- **calculationMethod:** CalculationMethod
- **args:** hash containing parameters for the calculation
- **children:** array of children nodes of this calculation node
- **nodeName:** string
- **nodeId:** unique string within the final score structure
- **requirement:** number greater than or equal to 0

An instance of this class has the following method:

- *calculateScore(studentMarks:* `Hash`, *partialScores:* `Hash`, *failures:* `Hash`, *assessmentInfo:* `Hash`): `Score`
  Stores the calculation result in *partialScores* using the **nodeId** as a key, and returns the calculation result. Additionally, before returning it checks if the calculation result meets the **requirement**, and if not, adds a failure description to *failures* using **nodeId** as a key.

2.7.2.1.4 MarkNode

The MarkNode class represents an assessment mark within the final score structure.
An instance of this class has the following attributes:

- **assessmentId:** a unique string or symbol for an assessment
- **gradingType:** a string that defines what kind of assessment the class represents, either ‘percentage’ or ‘points’
- **max:** the maximum mark for this assessment

An instance of this class has the following method:

- *calculateScore(studentMarks:* `Hash`, *partialScores:* `Hash`, *failures:* `Hash`, *assessmentInfo:* `Hash`): `Score`
  Looks up the **assessmentId** in the *studentMarks* hash, calls the MarkAdjustment class with the found value, and returns the result.

2.7.2.1.5 Constant

The Constant class represents a user-defined constant in the final score structure. An instance of this class always returns the same value regardless of the marks of the student.
An instance of this class has the following attributes:

- **gradingType:** a string that defines what kind of constant the class represents, 'percentage' is converted to 'points' for internal consistency
- **value:** the value of this constant
- **max:** the maximum of this constant, used for turning the **value** into a fraction

An instance of this class has the following method:

- *calculateScore(studentMarks:* `Hash`, *partialScores:* `Hash`, *failures:* `Hash`, *assessmentInfo:* `Hash`): `Score`
  Returns a Score that represents the value of this constant.

2.7.2.2 Calculation Methods

2.7.2.2.1 CalculationMethod

The CalculationMethod class is an interface that is used to unify the implementation of the other calculation method classes.
This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Every calculation method has a method *calculate* that accepts two arguments: *scores*: `[Score]` and *args*: `Hash`. Each calculation method requires different arguments, which are stored in the *args* hash. This hash may be empty for some calculation methods, but should not be nil.

2.7.2.2.2 WeightedAverage

The WeightedAverage class computes the weighted average of a set of scores.
For this class the *args* hash contains the following key/value pairs:

- **weights:** `[Hash]`
  One weight per score. Weights are fractions as a hash of the form { *numerator*: `int`, *denominator*: `int` }

This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Calculates the weighted average of a set of scores using the specified set of weights for each score. These weights are found in the *args* hash.

2.7.2.2.3 UnweightedAverage

The UnweightedAverage class calculates the unweighted average of a set of scores.
For this class the *args* hash does not contain key/value pairs.
This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Calculates the unweighted average of a set of scores.

2.7.2.2.4 BestWorstXOfY

The BestWorstXofY class is used to compute the unweighted average of the best or worst x scores of the received y scores, with the option of setting certain scores to always count.
For this class the *args* hash contains the following key/value pairs:

- **x:** `int`
  how many assignments should count
- **always_counts:** `[boolean]`
  One boolean per score. A boolean is set to true if the corresponding score should always be included in the average.
- **best_worst:** `boolean`
  A boolean set to 'true' if best x out of y is desired, and to 'false' if worst x out of y is desired.

This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Calculates the unweighted average of the best or worst x scores of the received y scores, including those scores set to always count.

2.7.2.2.5 MaximumMinimum

The MaximumMinimum class that takes either the minimum or the maximum of a list of scores.
For this class the *args* hash contains the following key/value pairs:

- **maximum:** `boolean`
  A boolean set to true if maximum score is desired and to false if minimum.

This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Finds either the maximum or minimum score in scores depending on the *maximum* value in the *args* hash.

2.7.2.2.6 Latest

The Latest class takes the latest score of a set of scores.
For this class the *args* hash contains the following key/value pairs:

- **dates:** `[date]`
  Array of dates with one date per score.

This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Finds the latest available score in *scores* by checking the 'dates' array in the *args* hash

2.7.2.2.7 Sum

The Sum class takes the minimum between 1 (equivalent to 100%) and the sum of a set of scores.
For this class the *args* hash does not contain key/value pairs.
This class has the following method:

- *calculate(scores:* `[Score]`, *args:* `Hash`): `Score`
  Calculates the sum of a set of scores, which is capped at 1.

2.7.2.3 Grading schemes

2.7.2.3.1 GradingScheme

GradingScheme is an interface that is implemented by all grading scheme classes, except for GradingSchemeHandler, allowing for a modular use of the grading schemes. This class has the following method:

- *compute(score:* `Score`): `string`
  Every grading scheme should convert a Score object to some string.

2.7.2.3.2 GradingSchemeHandler The GradingSchemeHandler class has the function of transforming a string input to the corresponding grading scheme class.
This class has the following methods:

- *convertToGradingScheme(gradingScheme:* `string`): `Class`
  Converts a string input to the corresponding class.
- *convertToAssociatedGradingScheme(gradingScheme:* `string`): `Class`
  Given a string input that corresponds to the grading scheme for the final grade, returns the corresponding partial grade grading scheme.

2.7.2.3.3 ZeroToTenInteger The ZeroToTenInteger class is a class that converts a score input to an integer between 0 and 10.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to an integer grade between 0 and 10.

2.7.2.3.4 ZeroToTen1Decimal The ZeroToTen1Decimal class is a class that converts a score input to a number between 0 and 10 accurate to one decimal.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to a number grade between 0 and 10 accurate to one decimal.

2.7.2.3.5 ZeroToTen2Decimal The ZeroToTen2Decimal class is a class that converts a score input to a number between 1 and 10 accurate to two decimals.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to a number grade between 0 and 10 accurate to two decimals.

2.7.2.3.6 ZeroToTenHalf The ZeroToTenHalf class is a class that converts a score input to a number between 0 and 10, rounded to the nearest half.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to a number grade between 0 and 10, rounded to the nearest half.

2.7.2.3.7 ZeroToTenHalfWithout55 The ZeroToTenHalfWithout55 class is a class that converts a score input to a number between 0 and 10, rounded to the nearest half, with the exclusion of 5.5.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to an number grade between 0 and 10, rounded to the nearest half, with the exclusion of 5.5.

2.7.2.3.8 LetterUK The LetterUK class is a class that converts a score input to a letter following the UK scheme.
This class has the following method:

- *compute(score:* `Score`): `string`
  Converts a score to a letter grade following the UK scheme.

2.7.2.3.9 LetterUS The LetterUS class is a class that converts a score input to a letter following the US scheme.
This class has the following methods:

- *compute(score:* `Score`): `string`
  Converts a score to a letter grade following the US scheme.

2.7.2.3.10 Percentage The Percentage class is a class that converts a score input to a number between 0 and 100 accurate to two decimals.
This class has the following methods:

- *compute(score:* `Score`): `string`
  Converts a score to a number grade between 0 and 100 accurate to two decimals.

2.7.2.3.11 CustomGradingScheme The CustomGradingScheme class can be used to define custom grading schemes on certain boundary conditions. It will convert a score into a string using these boundary conditions.
This class has the following methods:

- *compute(score:* `Score`): `string`
  Converts a score to a grade based on the boundary conditions.

2.7.2.4 Data Types

2.7.2.4.1 Score The Score class is a data type that is used internally for passing typed values between classes.
An instance of this class has the following attributes:

- **type:** the type of the score, either 'points' or 'excused'
- **value:** the value of the score, as a fraction between 0 and 1, inclusive.

2.7.2.5 Miscellaneous

2.7.2.5.1 GradeCalculationLogicEndpoint The GradeCalculationLogicEndpoint class is a singleton that provides access to the overarching functionality in the Grade Calculation logic module. It is essentially a facade for the Grade Calculation logic module.
This class has the following methods:

- *buildTree(json:* `Hash`): `Node`
  This method is for building a tree from a Hash, it directly references to *TreeBuilder.buildLevel*.
- *assessmentCheck(assessments:* `Hash`, *tree:* `Node`): `Hash`
  This method checks if all assessments in the given *tree* are also found in *assessments* with the same details. It returns a hash with found inconsistencies.
- *mapIdName(tree*: `Node`): `Hash`
  This method iterates through the tree, and returns a hash, mapping all nodeIds to nodeNames for all CalculationNodes.
- *gradeCalculation(tree*: `Node`, *markRequirements*: `Hash`, *gradingScheme*: `string`, *studentMarks*: `Hash`): `Hash`
  This method calculates the partial and final grades of a student, using the marks of the student in *studentMarks*, the final score structure *tree*, the grading scheme *gradingScheme*, and the mark requirements *markRequirements*.

2.7.2.5.2 PrecheckRequirements The PrecheckRequirements class has a single function: to check if a set of requirements is met by a set of marks.
This class has the following methods:

- *checkRequirement(requirements*: `Hash`, *marks*: `Hash`): `Boolean`
  This method checks if all marks in *marks* meet their associated requirements specified in *requirements*, returning true if they do, returning false if any mark does not meet its requirement.

2.7.2.5.3 DisplayMarkEndpoint The DisplayMarkEndpoint class has a single function: to convert a mark for display to the user according to the grading scheme.
This class has the following method:

- *convertMark(gradingScheme: String, mark: String, assessmentInfo: Hash, assessmentId: String)*: `String`
  This method converts the mark according to the grading scheme. If the mark is 'complete', 'incomplete', or 'excused', then it returns the achieved mark, otherwise it alters the mark.

2.7.2.5.4 AssessmentCheck The AssessmentCheck class has a single function: it checks if a given assessments hash is valid. This class has the following method:

- *validateAssessments(assessments: Hash)*: `void`
  This method validates if a given assessments hash is valid. All keys should be assessment IDs, and the values should be hashes containing information about the associated assessment. It checks those values as well, and throws an error if any information is incorrect.

2.7.2.5.5 MarkAdjustment The MarkAdjustment class has a single function: to adjust the value of a mark. This class has the following method:

- *calculateMark(value: String, assessmentInfo: Hash)*: `Rational or String`
  This method converts an original value for a mark into an adjusted value that accounts for factors such as no shows and mark adjustment.

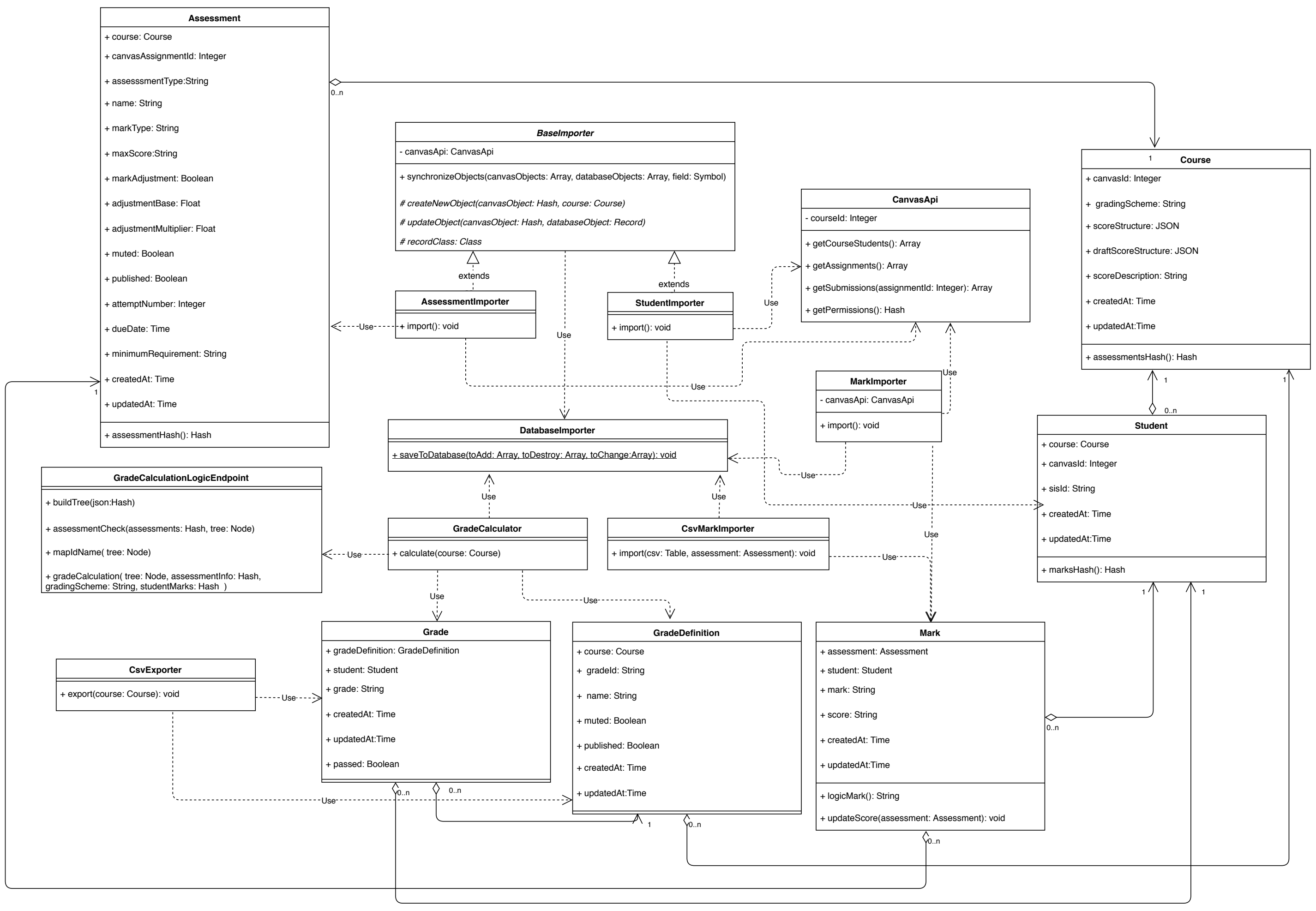


Figure 2.3: UML Back-end Class diagram

Sections 2.7.2.6 to 2.7.2.9 refer to the class diagram for the general back-end of Grade Calculation found in Figure 2.3.

2.7.2.6 Database models

2.7.2.6.1 Course Course is the class that corresponds to the database object. A course represents to a single Canvas course and stores course-wide attributes. An instance of this class has the following attributes:

- **canvasId:** an integer that defines the ID of this course in Canvas
- **gradingScheme:** a string that defines the grading scheme used in this course
- **scoreStructure:** a JSON structure that defines the score structure for this course
- **draftScoreStructure:** a JSON structure that defines the draft score structure for this course which corresponds to the saving of progress by a teacher.
- **scoreDescription:** a string that defines the score description shown to the student for this course
- **createdAt:** the time at which the course was initially created in Grade Calculation

- **updatedAt:** the time at which one of the course's attributes was last changed

This class has the following methods:

- *assessmentsHash()*: `Hash`
  This method returns a hash of assessments as input into the Grade Calculation logic endpoint. For each assessment, the key will be the assessments's ID and the value will be the value of the assessment's *assessmentHash* method.

2.7.2.6.2 Assessment

Assessment is the class that corresponds to the database object. An assessment corresponds to either a single Canvas assignment or to an assessment created by a teacher. An instance of this class has the following attributes:

- **course:** a reference to the course that this assessment belongs to
- **canvasAssignmentId:** an integer that defines the ID of this assignment in Canvas if the *assessmentType* is "canvas"
- **assessmentType:** a string that defines the type of this assessment, which is either "canvas" or "import", which corresponds to a Canvas assignment or a teacher-created assessment respectively
- **name:** a string that defines the name of the assessment
- **markType:** a string that defines the mark type of the assessment, which is either "points", "percentage", or "complete_incomplete".
- **maxScore:** a string that is either "complete" for "complete_incomplete" assessments or a numeric float for all other mark types.
- **markAdjustment:** a boolean that defines whether mark adjustment is enabled for all marks of this assessment.
- **adjustmentBase:** a float that defines the mark adjustment base that is either null if the *markAdjustment* is false or a positive number if it is true
- **adjustmentMultiplier:** a float that defines the mark adjustment multiplier that is either null if the *markAdjustment* is false or a positive number if it is true
- **muted:** a boolean that defines whether this assessment's marks are muted for the student. For Canvas assignments, this value is taken from Canvas and cannot be changed by the teacher.
- **published:** a boolean that defines whether this assessment is published for the student. For Canvas assignments, this value is taken from Canvas and cannot be changed by the teacher.
- **attemptNumber:** an integer that defines which attempt this is. The value will be null if this is not part of a multiple attempt or if it is the first attempt, while it will be 1 if this is the second attempt, 2 if it is the third attempt, etc.
- **dueDate:** the time at which this assessment is due. For Canvas assignments, this value is taken from Canvas and cannot be changed by the teacher.
- **minimumRequirement:** a string that defines the minimum requirement for this assessment, which is either null, a number, or "complete", "incomplete", "no_show", "excused".
- **createdAt:** the time at which the assessment was initially created in Grade Calculation
- **updatedAt:** the time at which one of the assessment's attributes was last changed

This class has the following methods:

- *assessmentHash()*: `Hash`
  This method returns a hash of the values that should be passed as input into the Grade Calculation logic module. The keys are defined by the Grade Calculation logic module and are "grading_type", "minimum_requirement", "mark_adjustment", "adjustment_base", "adjustment_multiplier", "max", "repeated_attempt" and "repeated_attempt_assessment_id".

2.7.2.6.3 GradeDefinition

GradeDefinition is the class that corresponds to the database object. A grade definition corresponds to either the final grade or a partial grade defined in the score structure. An instance of this class has the following attributes:

- **course:** a reference to the course that this grade definition belongs to
- **gradeId:** a string that corresponds with the "node_id" of the partial grade. It can also be either "final" for the final grade or "mandatory" for the mandatory assessments grade.
- **name:** a string that defines the name of this grade definition, which is either the "node_name", "Final Grade", or "Mandatory Assessments"
- **muted:** a boolean that defines whether this definition's grades are muted for the student
- **published:** a boolean that defines whether this grade definition is published for the student
- **createdAt:** the time at which the grade definition was initially created in Grade Calculation
- **updatedAt:** the time at which one of the grade definition's attributes was last changed

2.7.2.6.4 Student

Student is the class that corresponds to the database object. A student corresponds to either a single Canvas user with the role student. An instance of this class has the following attributes:

- **course:** a reference to the course that this student belongs to
- **canvasId:** an integer that defines the ID of this user in Canvas
- **sisId:** a string that defines the SIS ID of this student, as returned by Canvas
- **createdAt:** the time at which the student was initially created in Grade Calculation
- **updatedAt:** the time at which one of the student's attributes was last changed

This class has the following methods:

- *marksHash()*: `Hash`
  This method returns a hash of the assessment ID's to the marks of this student, as defined by the Grade Calculation logic module.

2.7.2.6.5 Mark

Mark is the class that corresponds to the database object. A mark is the result of a single assessment for a single student. An instance of this class has the following attributes:

- **assessment:** a reference to the assessment that this mark is for
- **student:** a reference to the student that this mark is for
- **mark:** a string that defines the value of such mark
- **score:** a string that defines the value of such mark after applying a grading scheme.

- **createdAt:** the time at which the mark was created
- **updatedAt:** the time at which one of the mark's attributes was last changed

This class has the following methods:

- *logicMark()*: `String`
  This method returns the score for input into the Grade Calculation logic module. It will either be "excused" , "complete","incomplete", "no_show" or be a number between 0 and the assessment's maxScore.
- **updateScore(assessment:Assessment)**
  This method updates the score value whenever the mark value or the assessment's markAdjustment is changed.

2.7.2.6.6 Grade

Grade is the class that corresponds to the database object. A grade is the output of the Grade Calculation logic module of a single grade for a single student. An instance of this class has the following attributes:

- **gradeDefinition:** a reference to the grade definition that this grade is for
- **student:** a reference to the student that this grade is for
- **grade:** a string that defines the returned value of the grade, according to the grading scheme
- **passed:** a boolean that defines whether this grade is seen as a pass, which is only set for the final grade
- **createdAt:** the time at which the grade was created
- **updatedAt:** the time at which one of the grade's attributes was last changed

2.7.2.7 Database object updaters

2.7.2.7.1 DatabaseImporter

DatabaseImporter is a utility class that is used for saving changes to the database in a uniform way. This class has the following static method:

- *saveToDatabase(toAdd: Array, toDestroy: Array, toChange: Array)*: `void`
  This method will start a database transaction and then insert all records in toAdd to the database, delete all records in toDestroy from the database and update all records in toChange in the database. Finally, it will end the transaction and commit all changes to the database.

2.7.2.7.2 BaseImporter

BaseImporter is an abstract class that can be extended from to synchronize Canvas objects to Grade Calculation database objects. An instance of this class has the following attributes:

- **canvasApi:** a reference to the Canvas API that can be used by the importer, which also contains a reference to the current course

This class has the following methods:

- *synchronizeObjects(canvasObjects: Array, databaseObjects: Array, field: Symbol)*: `void`
  This method will use the abstract methods and find the toAdd, toDestroy and toChange parameters for the *DatabaseImporter* using the field argument.

This class has the following abstract methods:

- *createNewObject(canvasObject: Hash, course: Course)*: `Record`
  This method will be called when a new database record needs to be created. The canvasObject and course can be used to construct the database record.
- *updateObject(canvasObject: Hash, databaseObject: Record)*: `Record`
  This method will be called when a database record could be updated. The databaseObject should be updated based on the information found in canvasObject and returned.
- *recordClass()*: `Class`
  This method should return the class of the database record that is used in this importer, e.g. *Student* for *StudentImporter*.

2.7.2.7.3 AssessmentImporter

AssessmentImporter is a class that is used to import Canvas assessments into the Grade Calculation database. This class has the following methods:

- *import()*: `void`
  This method will retrieve all Canvas and database assessments and then use *synchronizeObjects* of its parent class to synchronize the two.

2.7.2.7.4 StudentImporter

StudentImporter is a class that is used to import Canvas users with the student role into the Grade Calculation database. This class has the following methods:

- *import()*: `void`
  This method will retrieve all Canvas and database students and then use *synchronizeObjects* of its parent class to synchronize the two.

2.7.2.7.5 MarkImporter

MarkImporter is a class that is used to import Canvas marks into the Grade Calculation database. An instance of this class has the following attributes:

- **canvasApi:** a reference to the Canvas API that can be used by the importer to get all students which also contains a reference to the current course

This class has the following methods:

- *import()*: `void`
  This method will retrieve all Canvas and database marks and then construct the difference and pass it to *DatabaseImporter*'s *saveToDatabase*.

2.7.2.7.6 GradeCalculator

GradeCalculator is a class that is used to calculate all grades of a course and save them to the database. This class has the following methods:

- *calculate(course: Course)*: `void`
  This method will call the Grade Calculation logic module's endpoint to calculate the grades of every student and then save them to the database.

2.7.2.8 CSV import and export

2.7.2.8.1 CsvMarkImporter

MarkImporter is a class that is used to import marks into the Grade Calculation database from a CSV file. This class has the following methods:

- *import(csv: Table, assessment: Assessment)*: `void`
  This method will take in a parsed CSV file and an assessment and update or create the marks for every student in the file.

2.7.2.8.2 CsvExporter CsvExporter is a class that is used to export grades into a CSV file. This class has the following methods:

- *export(course: Course)*: `String`
  This method will export the grades of all students of a course to a CSV string.

2.7.2.9 Miscellaneous

2.7.2.9.1 CanvasApi CanvasApi is a utility class that is used for connecting to the Canvas API. An instance of this class has the following attributes:

- **courseId:** an integer that defines the current course ID such that the Canvas API is always called for the correct course

This class has the following methods:

- *getCourseStudents()*: `Array`
  This method will call the Canvas API ”List users in course" endpoint with the enrollment type filtered to student and return all pages in a single array.
- *getAssignments()*: `Array`
  This method will call the Canvas API ”List assignments" endpoint and return all pages in a single array.
- *getSubmissions(assignmentId: Integer)*: `Array`
  This method will call the Canvas API ”List assignment submissions" endpoint and return all pages in a single array.
- *getPermissions()*: `Hash`
  This method will call the Canvas API ”Permissions" endpoint and return the value as a hash.

2.7.3 Data Model

The data model for Grade Calculation is represented as an Entity Relationship diagram presented below in Figure 2.4. It describes which data is being stored and how that data is structured in the databases required by Grade Calculation. The databases provided by Drieam are not components of Grade Calculation, therefore they are not included in this diagram. Following the figure, a description of each entities in the data model with a corresponding explanation of their attributes is provided.

2.7.3.1 Database Diagram

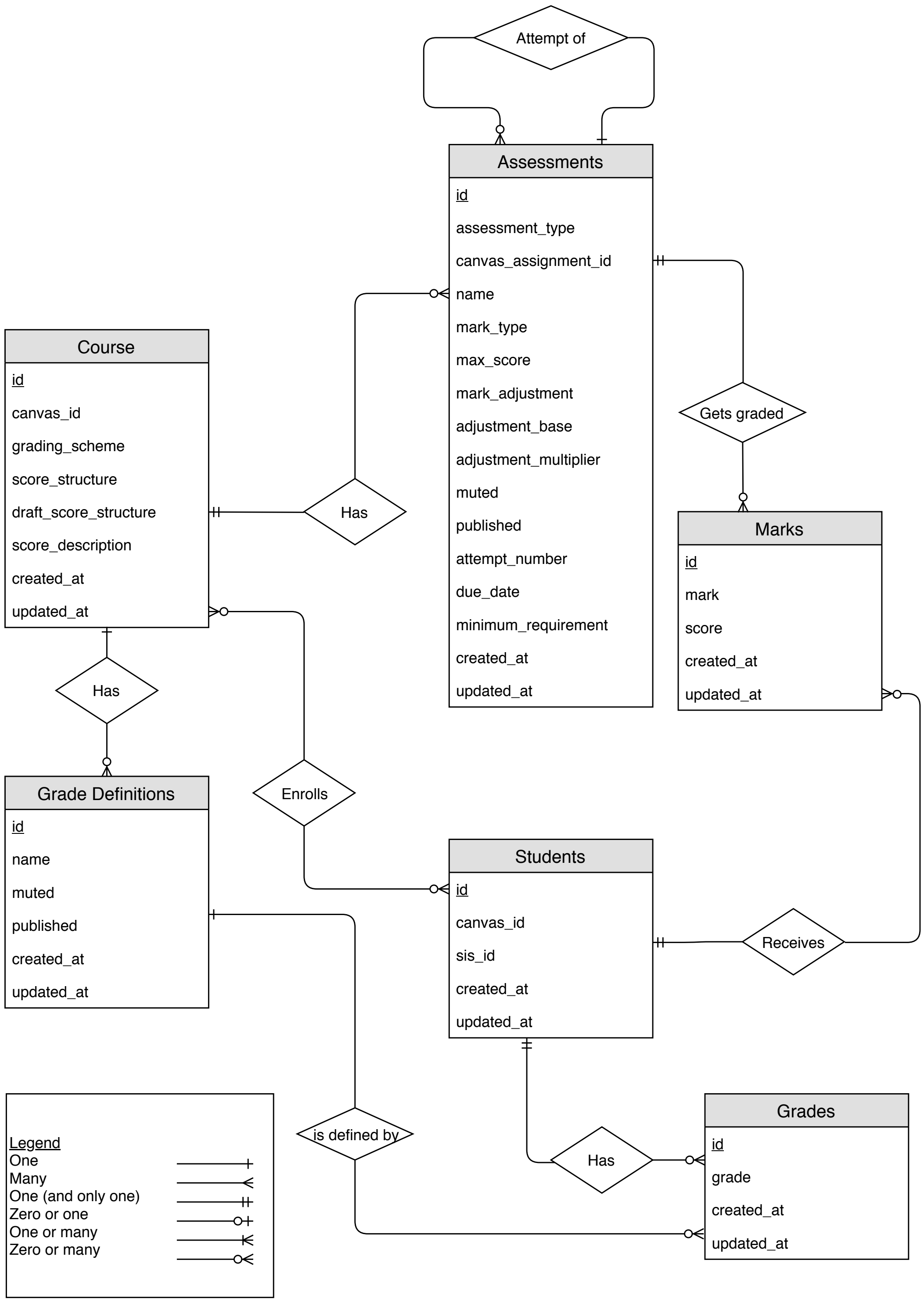


Figure 2.4: ER Data model

2.7.3.1.1 Course

A course entity represents a course that is provided in the educational institute. Courses have the following properties:

- `id`: a sequential number which is unique per course, representing the id of the course on Grade Calculation
- `canvas_id`: a number unique per course, representing the id of the course on Canvas.
- `grading_scheme`: a string, representing the grading scheme of choice to be applied in all scores within that course.
- `score_structure`: a JSON object, representing the final score calculation structure.
- `draft_score_structure`: a JSON array, representing a final score calculation structure that has not yet been finalized.
- `score_description`: A string that is a textual description of the score structure
- `created_at`: a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the course was created.
- `updated_at`: a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the course was implemented.

2.7.3.1.2 Assessment

An assessment represents an exam, an assignment, a quiz, etc. within a course. Assessments have the following properties:

- `id`: a sequential number which is unique per assessment, representing the id of the assessment on Grade Calculation
- `course_id`: an integer, that will link an assessment to a specific course
- `assessment_type`: a string, whose value is either "canvas" or "import". It represents the type of assessment depending on from which source marks will be imported.
- `canvas_assignment_id`: a number, unique per assessment, representing the id of the assignment on Canvas.
- `name`: a string, representing the name this assessment is assigned, either from Canvas or it is created when creating an imported assessment manually.
- `mark_type`: a string, whose value is either "points", "complete_incomplete" or "percentage". It represents the type of marks the assessment expects.
- `max_score`: will be "complete" if mark_type is "complete_incomplete", otherwise it will contain a number, representing the maximum score that can be obtained for the assessment. This will be equal to the upper-bound of the mark if grade_type is equal to "points" or "percentage".
- `mark_adjustment`: a boolean which is set to be true when all marks for this assessment need to be adjusted
- `adjustment_base`: a positive number representing the amount of points that are always added to a student's mark in the mark adjustment process
- `adjustment_multiplier`: a positive number which represents the number by which the mark of the student will be multiplied in the mark adjustment process.
- `muted`: A boolean which is set to be true when all marks for this assessment are not visible to students, or is set to be false otherwise.
- `published`: A boolean which is set to be true when the assessment itself and its associated marks are not visible to students, or is set to be false otherwise.

- `attempt_assessment_id`: a unique number, that will link an assessment that is an xth ($x \in \mathbb{N}^+$) attempt of the original assessment.
- `attempt_number`: an integer representing the x ($x \in \mathbb{N}^+$) in the xth attempt.
- `due_date`: is a timestamp without time zone, representing the last moment at which an assessment can be submitted.
- `minimum_requirement`: A string, representing the minimum score needed to avoid failing the course.
- `created_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the assessment was created.
- `updated_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the assessment was implemented.

2.7.3.1.3 Marks

A mark represents the result a student is given for an assessment.

- `id`: a sequential number which is unique per mark, representing the id of the mark on Grade Calculation
- `assessment_id`: an integer, that will link a mark to a specific assessment.
- `student_id`: an integer, that will link a mark to a specific student.
- `mark`: can be empty, contain a number or contain one of “complete", “incomplete", “no_show", or “excused" representing the result the student obtained for that assessment.
- `score`: can be empty, contain a number or contain one of “complete", “incomplete", “no_show", or “excused" representing the result the student obtained for that assessment as a display value, which can be mark adjusted based on the assessment information
- `created_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the mark was created.
- `updated_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the mark was implemented.

2.7.3.1.4 Grades

- `id`: a sequential number which is unique per grade, representing the id of the grade on Grade Calculation
- `student_id`: an integer, that will link a grade to a specific student.
- `grade_definition_id` an integer, linking a grade to the corresponding grade definition.
- `grade`: a string representing the actual grade the student has obtained
- `passed`: a boolean representing whether this grade is a pass, only set if this grade is for the final grade or for the mandatory assessments
- `created_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the grade was created.
- `updated_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the grade was implemented.

2.7.3.1.5 Students

- `id`: a sequential number which is unique per student, representing the id of the student on Grade Calculation
- `canvas_id`: a number, unique per student within a course, representing the ID of the student on Canvas.
- `course_id`: an integer, that will link a student to a specific course.
- `sis_id`: A unique string within a course, representing the student ID used in the Student Information System.
- `created_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the student profile was created.
- `updated_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the student profile was implemented.

2.7.3.1.6 Grade Definitions

- `id`: a sequential number which is unique per grade definition, representing the id of the grade definition on Grade Calculation
- `course_id`: an integer, that will link a grade definition to a specific course.
- `grade_id`: an integer, which is unique per node in the final score structure per course, representing the id of the grade used in Grade Calculation.
- `name`: a string representing how the grade definition is addressed to. The name will be inherited from the node_name.
- `muted`: A boolean which is set to be true when all marks for this grade definition are not visible to students, or is set to be false otherwise.
- `published`: A boolean which is set to be true when the grade definition itself and its associated grades are not visible to students, or is set to be false otherwise.
- `created_at`: is a time stamp without a time zone, created automatically by Ruby sends the time the grade definition was created.
- `updated_at`: is a time stamp without a time zone, created automatically by Ruby on Rails, which represents the time the last change in the grade definition was implemented.

2.7.3.2 JSON Structure

A specification was created in order to allow for consistent storing and retrieving of score structures as JSON objects.
This is a recursive structure that consists of a root node specification, possibly with nested children nodes.

2.7.3.2.1 Calculation Node A calculation node represents a set of assessments with an associated calculation.
A calculation node requires the following fields:

- **node_type:** ='calculation', specifies that this node is a calculation node
- **node_name:** the string name of this node, used for display purposes

- **node_id:** the unique id of this node, used for keeping track of partial scores
- **calculation_type:** a string, specifying exactly what calculation is associated with this node
- **args:** an object of arguments to be passed to the calculation
- **children:** an array of children node specifications, allowing for the recursive tree structure
- **requirement:** the minimum requirement on this partial score, may be nil

2.7.3.2.2 Mark Node

A mark node represents a single assessment.
A mark node requires the following fields:

- **node_type:** ='mark', specifies that this node is a mark node
- **assessment_id:** specifies what assessment this node represents
- **grading_type:** specifies what type of grade the assessment is, either 'points' or 'percentage'
- **max:** specifies the maximum mark that can be obtained for the associated assessment

2.7.3.2.3 Constant Node

A constant node represents a constant fraction, such as $5/10$.
A constant node requires the following fields:

- **node_type:** ='constant', specifies that this node is a constant node
- **grading_type:** specifies whether this constant is a percentage or points type
- **value:** the value of the constant
- **max:** the maximum of the constant. This is needed because a value of 5 out of 10 is different from 5 out of 20.

2.7.4 Sequence Diagrams

2.7.4.1 Student Interface

2.7.4.1.1 Student opens Grade Calculation Student Interface

Students can open Grade Calculation. To do this the student navigates in Canvas to the Grade Calculation tab. After selecting this tab, the student user is able to see the grades view if the database can be reached, otherwise the student receives an error page.

Goal: Student opens the Grade Calculation Student Interface in Canvas.
Precondition:
The actor is enrolled as a Student for the specific course.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
A final score structure has been created for computing the final score.
A grading scheme has been selected for computing the final grade.

Postcondition: All published assessments and assessment sets included in the final score structure are displayed and for those which are not muted, the mark, or grade for this specific student are displayed.
Summary: The actor can view their own marks, partial grades, and final grade in the Grade Calculation Student Interface in Canvas.
Priority: Must have

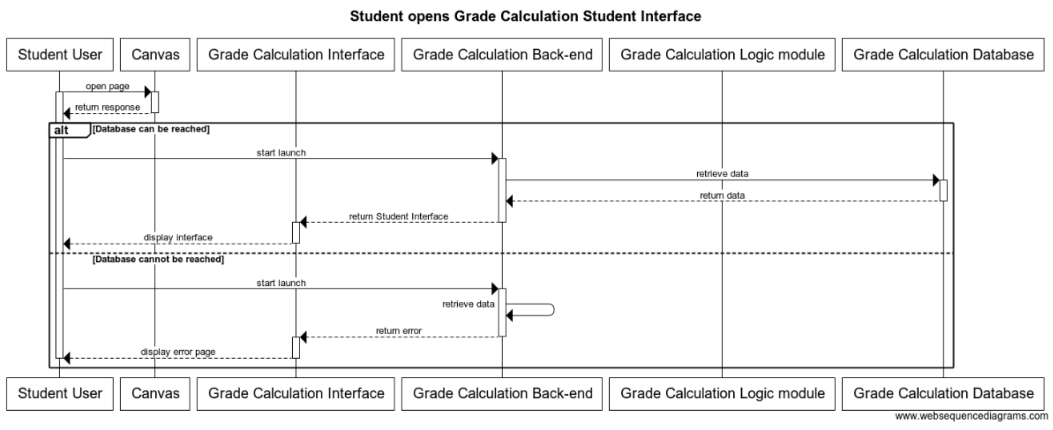

Student opens Grade Calculation Student Interface
Student User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
alt
[Database can be reached]
start launch
retrieve data
return data
return Student Interface
display interface
[Database cannot be reached]
start launch
retrieve data
return error
display error page
www.websequencediagrams.com

Figure 2.5: Student opens Grade Calculation Student Interface

2.7.4.1.2 Student exports marks and grades using a CSV file

Students can export their marks, partial grades, and final grades in Grade Calculation. To do this they will need to navigate in Canvas to the Grade Calculation tab. If the database can be reached, the student user is able to select the export option, otherwise the student receives an error page.

Goal: Student exports their marks, partial grades, and the final grade using a CSV file.
Precondition:
The actor is enrolled as a Student in the specific course.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The marks, partial grades, and final grade of the actor are stored in a CSV file.
Summary: The actor can export their own marks, partial grades, and final grade such that they are saved in a CSV file.
Priority: Won't have

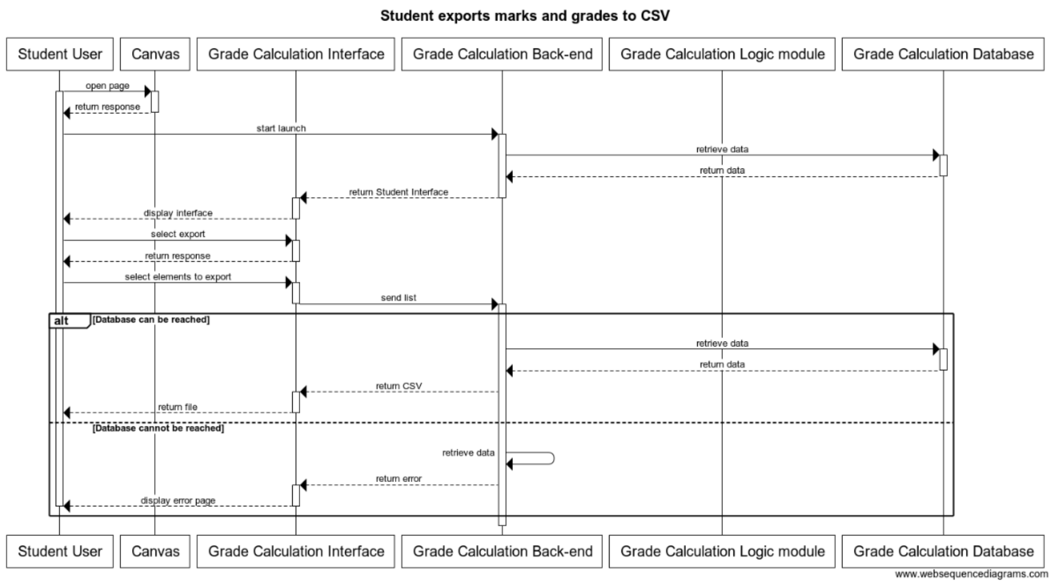

Student exports marks and grades to CSV
Student User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Student Interface
display interface
select export
return response
select elements to export
send list
alt
[Database can be reached]
retrieve data
return data
return CSV
return file
[Database cannot be reached]
retrieve data
return error
display error page
www.websequencediagrams.com

Figure 2.6: Student exports marks and grades to CSV

2.7.4.2 Teacher interface

2.7.4.2.1 Teacher opens Grade Calculation Teacher Interface

Teachers can open Grade Calculation. To do this they will need to navigate to the Grade Calculation tab in Canvas. If the database is accessible, the teacher user is able to see the Teacher Interface, otherwise the teacher receives an error page.

Goal: Teacher opens the Grade Calculation Teacher Interface in Canvas.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: If a final score structure exists, it is displayed, otherwise an empty score structure is displayed.
Summary: The actor requests the page of Grade Calculation. The page is displayed in Teacher interface, which contains all functionality of the plugin.
Priority: Must have

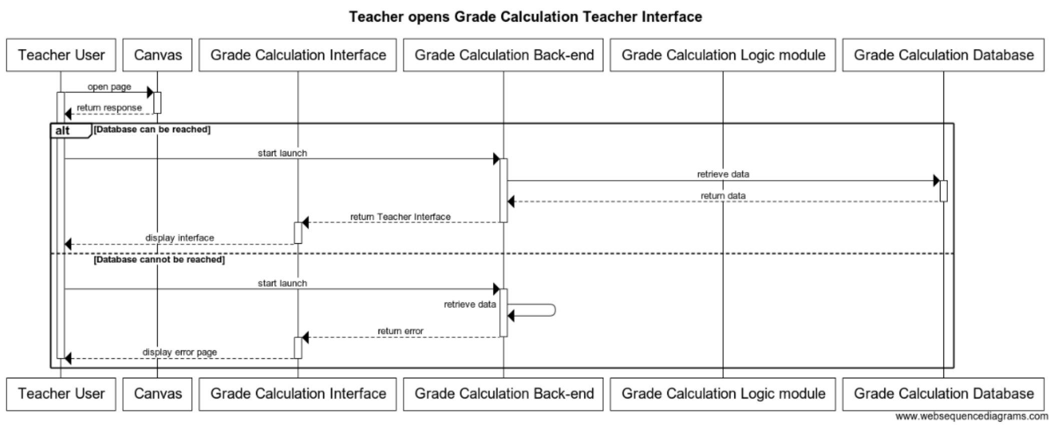

Teacher opens Grade Calculation Teacher Interface
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
alt
[Database can be reached]
start launch
retrieve data
return data
return Teacher Interface
display interface
[Database cannot be reached]
start launch
retrieve data
return error
display error page
www.websequencediagrams.com

Figure 2.7: Teacher opens Grade Calculation Teacher Interface

2.7.4.2.2 Teacher defines or edits a grading scheme

Teachers defines a new grading scheme or edits an existing one in Grade Calculation. To do so, they need to select the create grading scheme option and build the desired grading scheme.

Goal: Teacher defines a new grading scheme or edits an existing one.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The database is accessible.
Postcondition: The grading scheme is saved into the database.
Summary: The Teacher defines the grading scheme that is used for the calculation of the final grade.
Priority: Won't have

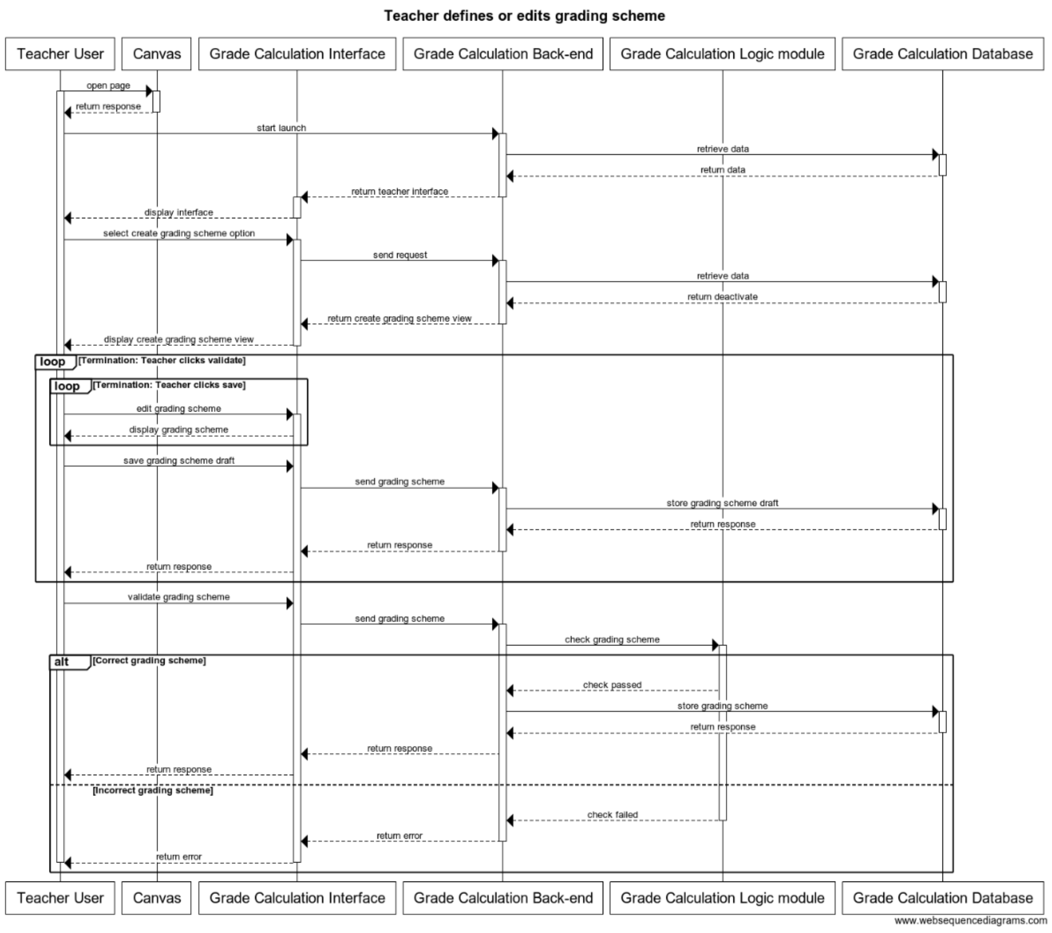

Teacher defines or edits grading scheme
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return teacher interface
display interface
select create grading scheme option
send request
retrieve data
return deactivate
return create grading scheme view
display create grading scheme view
loop [Termination: Teacher clicks validate]
loop [Termination: Teacher clicks save]
edit grading scheme
display grading scheme
save grading scheme draft
send grading scheme
store grading scheme draft
return response
return response
return response
validate grading scheme
send grading scheme
check grading scheme
alt [Correct grading scheme]
check passed
store grading scheme
return response
return response
return response
[Incorrect grading scheme]
check failed
return error
return error
www.websequencediagrams.com

Figure 2.8: Teacher defines or edits grading scheme

2.7.4.2.3 Teacher defines or edits final score structure and selects a grading scheme

Teachers defines a new final score structure or edits an existing one. Moreover the teacher must select a grading scheme, since a final score structure cannot be validated without it. This is done using the “Grading Structure" view of the Teacher Interface to build the desired structure and select the desired grading scheme.

Goal: Teacher defines a new final score structure or edits an existing one and selects a grading scheme. All of which will be used to determine the final grade.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
All assessments are defined.
The database is accessible.
Postcondition: The final score structure and selected grading scheme are saved in the database
Summary: The final score structure and grading scheme are defined by the teacher.
Priority: Must have

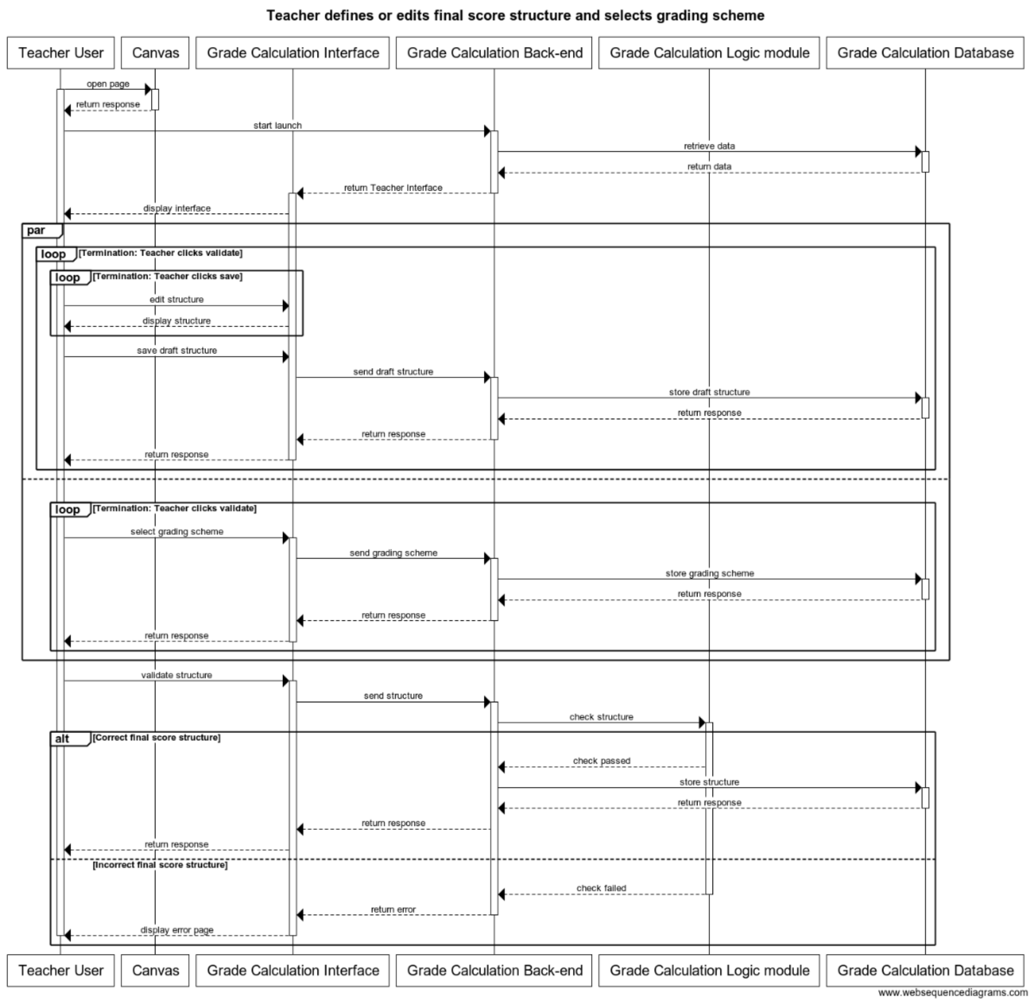


Figure 2.9: Teacher defines or edits final score structure and selects grading scheme

2.7.4.2.4 Teacher views audit logs

Teachers views audit logs in Grade Calculation which displays the changes that occurred within the plugin.

Goal: Teacher views audit logs.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The final score structure that the actor wishes to edit is modified.
Summary: The actor is able to view the audit logs of the plugin. This way the actor can investigate potential problems.
Priority: Should have

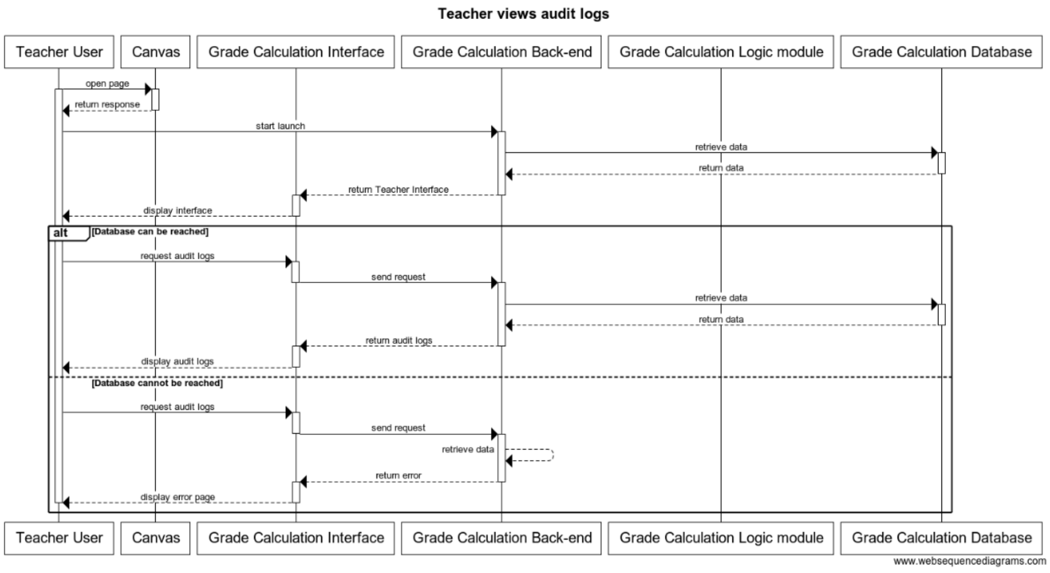


Figure 2.10: Teacher views audit logs

2.7.4.2.5 Teacher filters audit logs

Teachers wants to view the audit logs to which a specific filter applies.

Goal: Teacher filters audit logs.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The actor has navigated to the audit logs view.
The specific course has Grade Calculation enabled.
Postcondition: The audit logs to which the filter applies are displayed.
Summary: The actor is able to view the filtered audit logs of the plugin. This way the actor can investigate potential problems.
Priority: Won't have

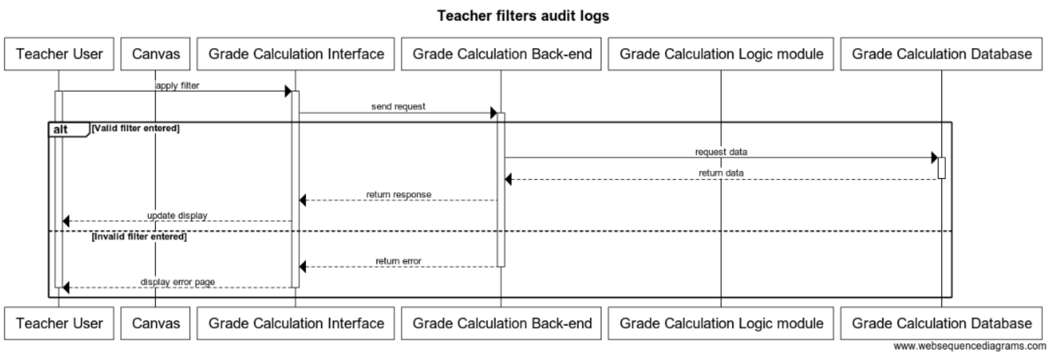


Figure 2.11: Teacher views audit logs

2.7.4.2.6 Teacher computes the final grade

Teachers wants to compute the final grade using the final score structure and grading scheme defined in Grade Calculation. To do this, the teacher must manually select the option to compute final grades.

Goal: Teacher computes the final grade for each Student.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged in Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
A final score structure has been created and saved.
A grading scheme has been selected.
The database is accessible.
Postcondition: The final grade for each student is computed and stored.
Summary: When the teacher desires to compute the final grade (e.g. the term ended), the final grade must be computed. The defined final score structure is used to calculate the final score, which is then converted to the final grade using the grading scheme.
Priority: Must have

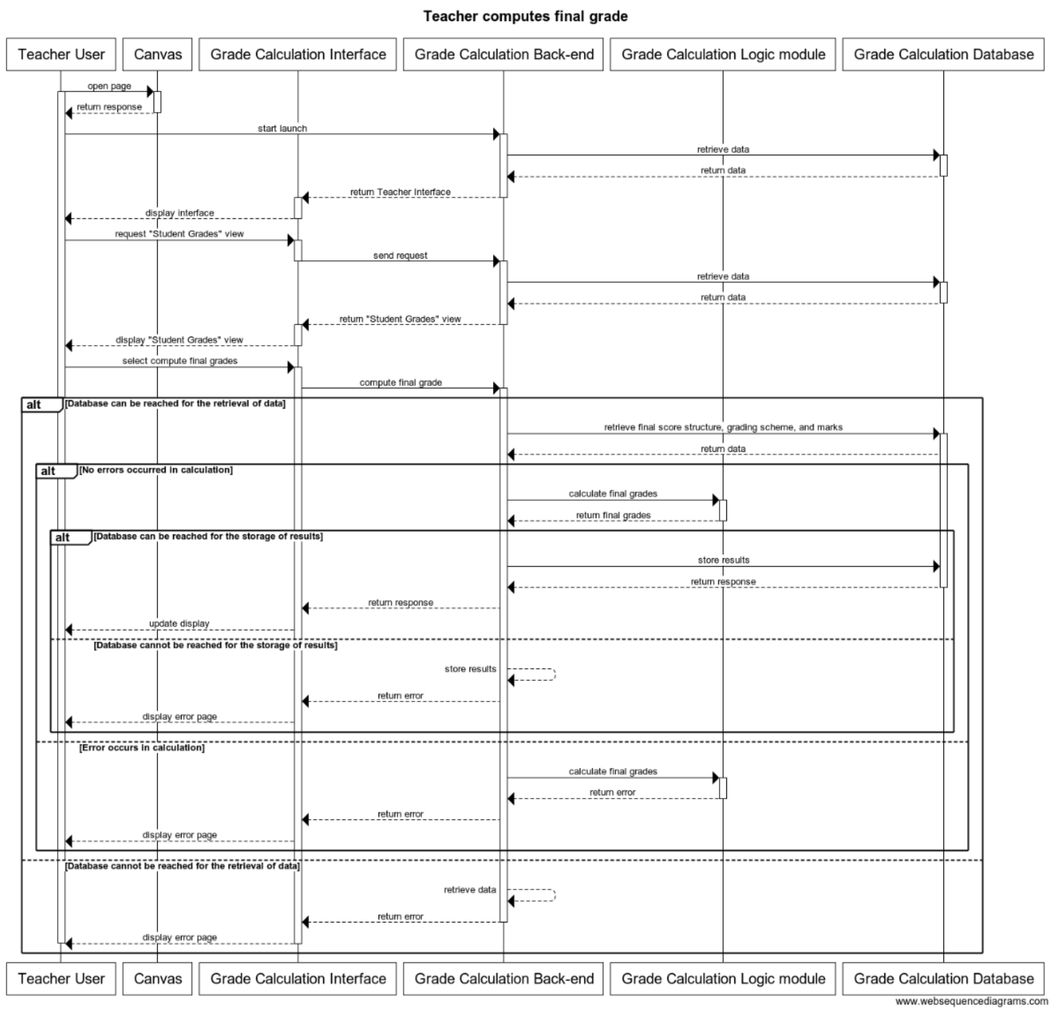

Teacher computes final grade
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Teacher Interface
display interface
request "Student Grades" view
send request
retrieve data
return data
return "Student Grades" view
display "Student Grades" view
select compute final grades
compute final grade
alt [Database can be reached for the retrieval of data]
retrieve final score structure, grading scheme, and marks
return data
alt [No errors occurred in calculation]
calculate final grades
return final grades
alt [Database can be reached for the storage of results]
store results
return response
return response
update display
[Database cannot be reached for the storage of results]
store results
return error
display error page
[Error occurs in calculation]
calculate final grades
return error
return error
display error page
[Database cannot be reached for the retrieval of data]
retrieve data
return error
display error page
www.websequencediagrams.com

Figure 2.12: Teacher computes final grade

2.7.4.2.7 Teacher mutes grade

Teachers wants hide a specific grade to the students. To do so the teacher has to select the mute options of that specific grade.

Goal: Teacher mutes the grade.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The specific grade is unmuted
Postcondition: A grade is muted.
Summary: The actor mutes a grade. This grade is no longer visible to the Students.
Priority: Should have

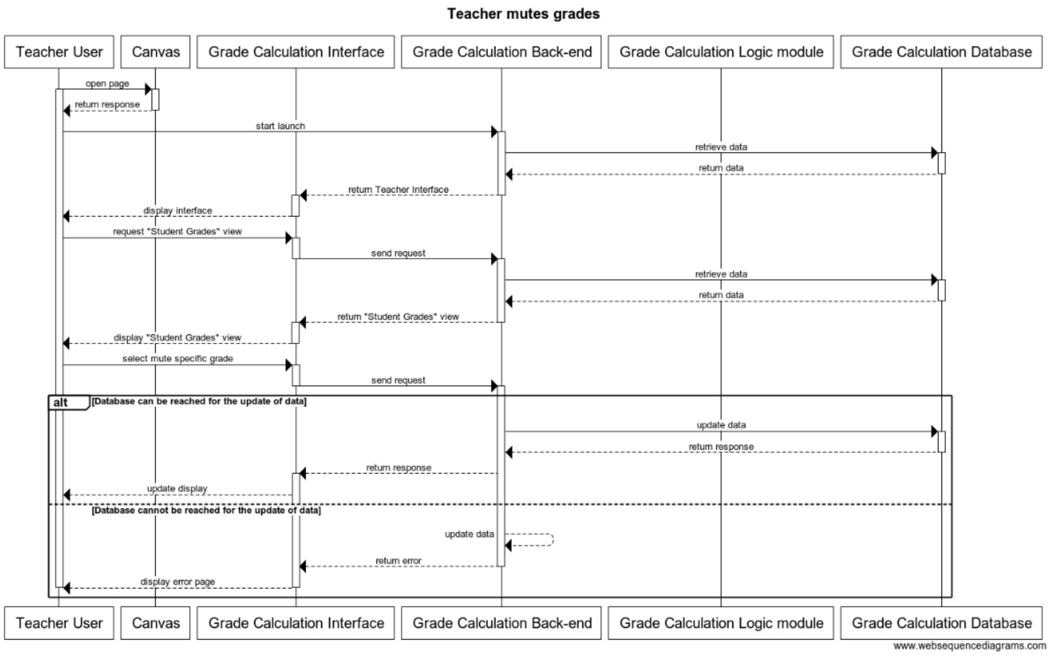

Teacher mutes grades
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Teacher Interface
display interface
request "Student Grades" view
send request
retrieve data
return data
return "Student Grades" view
display "Student Grades" view
select mute specific grade
send request
alt
[Database can be reached for the update of data]
update data
return response
return response
update display
[Database cannot be reached for the update of data]
update data
return error
display error page
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.13: Teacher mutes grade

2.7.4.2.8 Teacher unmutes grade

Teachers wants make a specific grade visible to students. To do so the teacher has to select the unmute options of that specific grade.

Goal: Teacher unmutes the grade.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The specific grade is muted
Postcondition: A grade is unmuted.
Summary: The actor unmutes a grade. This grade becomes visible for the Students.
Priority: Should have

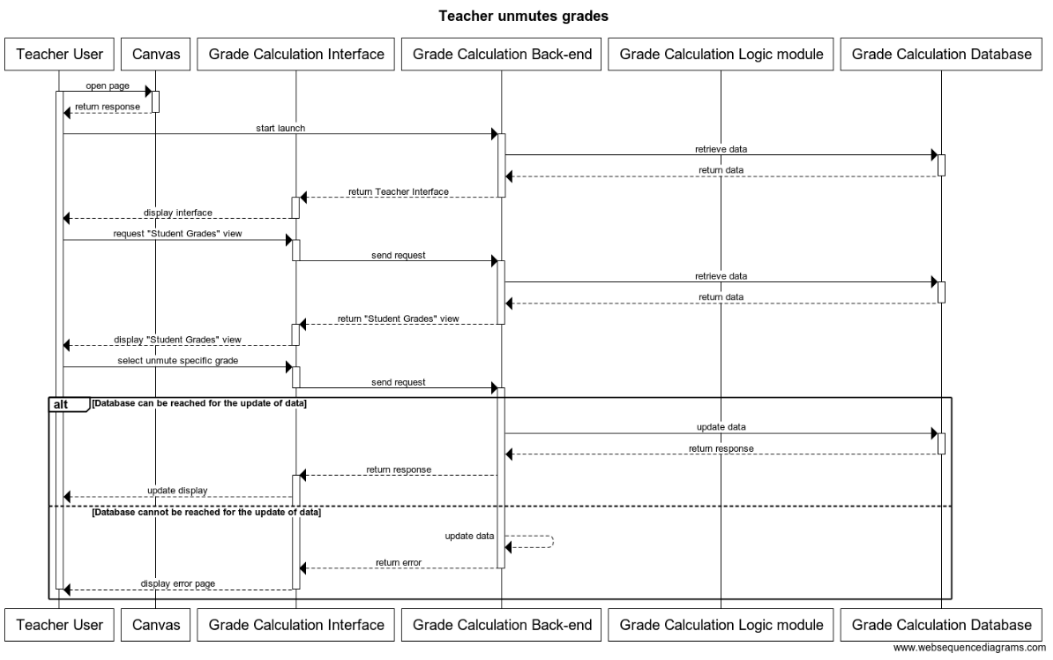

Teacher unmutes grades
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Teacher Interface
display interface
request "Student Grades" view
send request
retrieve data
return data
return "Student Grades" view
display "Student Grades" view
select unmute specific grade
send request
alt
[Database can be reached for the update of data]
update data
return response
return response
update display
[Database cannot be reached for the update of data]
update data
return error
display error page
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.14: Teacher unmutes grade

2.7.4.2.9 Teacher filters in the set of marks and grades

Teachers defines a filter condition such that only marks and grades that fulfil this condition are shown to the Teacher.

Goal: Teacher can apply filters to the set of marks, partial grades, and final grades.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged in Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The database is accessible.
Postcondition: Filtered items are returned to the actor, such that the actor can view them.
Summary: The actor applies filters to the items available. The filters get applied and the items get returned to the actor.
Priority: Could have

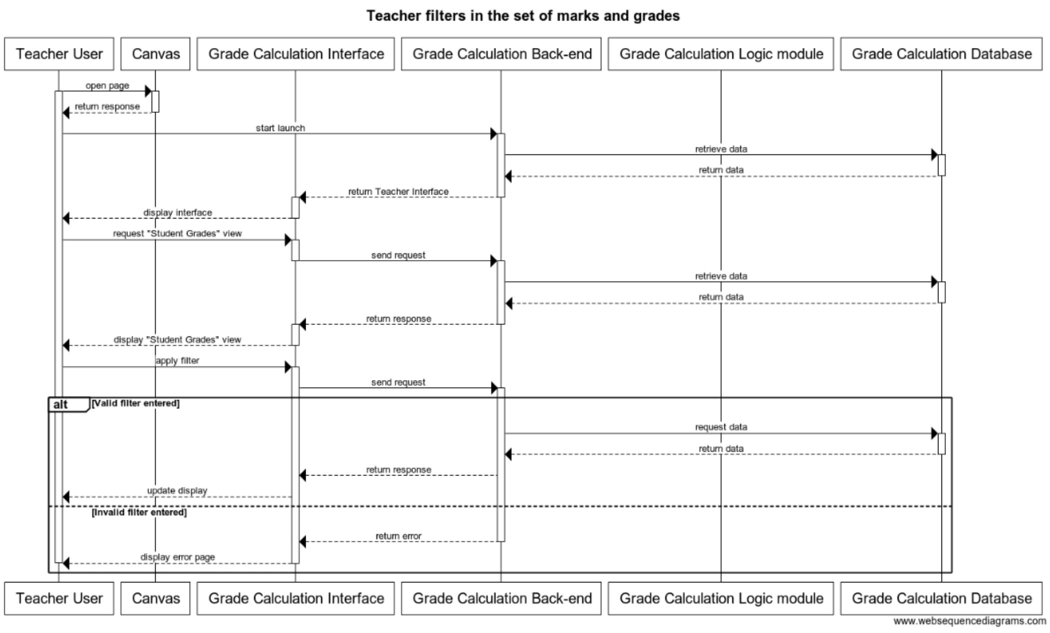


Figure 2.15: Teacher filters in the set of marks and grades

2.7.4.2.10 Teacher imports marks using a CSV

Teachers wants to import the marks for a specific assessment, to do so the teacher uploads a CSV file containing the marks for this assessment. If marks have already been imported for this assessment, uploading new marks for this assessment will updated the stored marks.

Goal: Teacher imports marks using a CSV file.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The marks of the assessment are uploaded to the database.
Summary: The actor imports the marks of a single assessment for all students to the database.
Priority: Should have

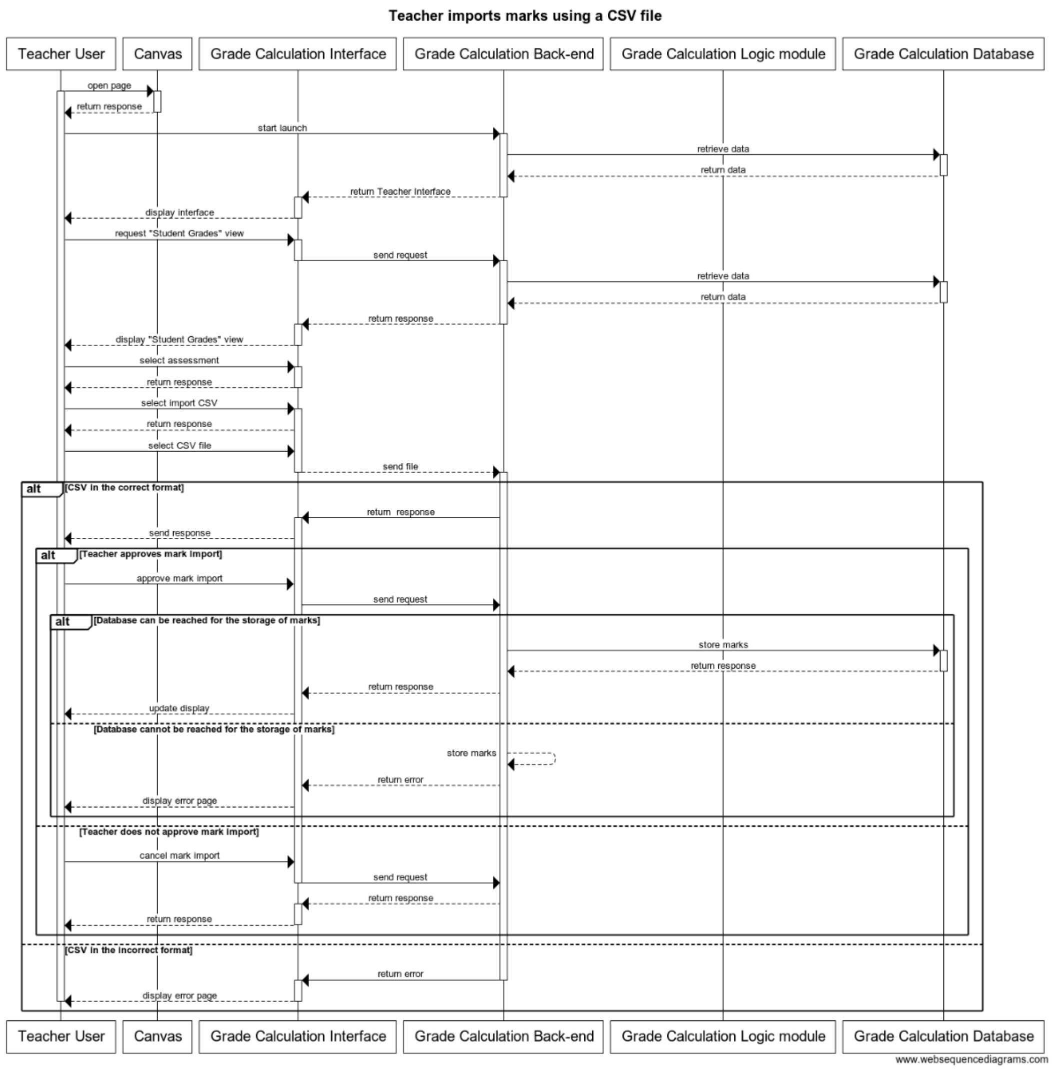
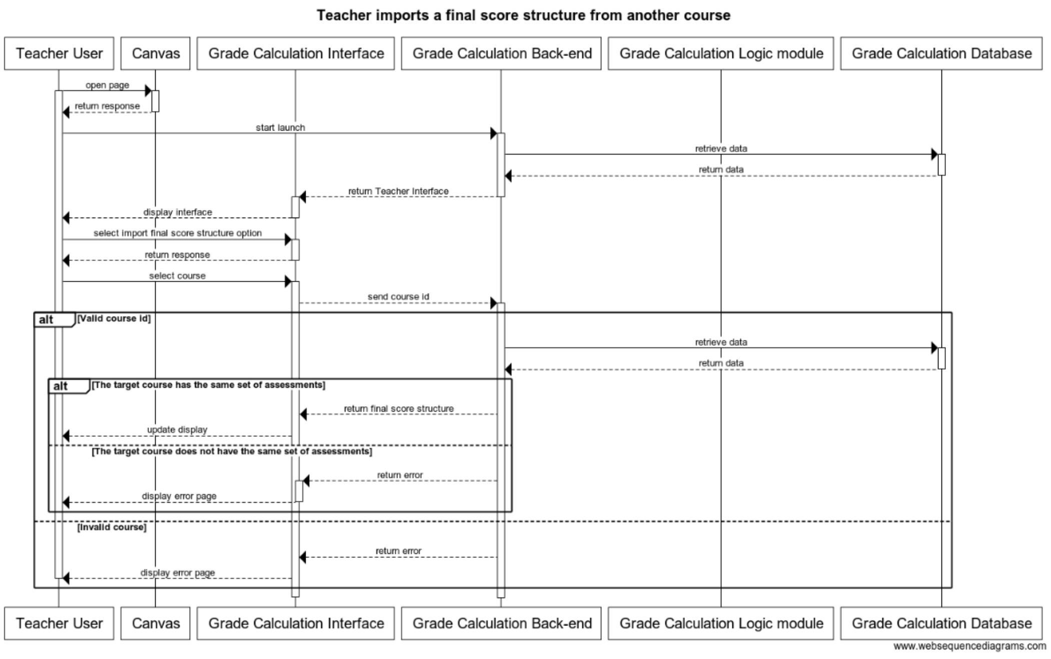


Figure 2.16: Teacher imports marks using a CSV file

2.7.4.2.11 Teacher imports a final score structure from another course

If the Teacher desires to re-use a final score structure from a different course, the teacher must select the import score structure option and specify from which course the desired structure to be imported from.

Goal: Teacher can import a final score structure from another course.
Precondition:
The actor is enrolled as a Teacher for the specific courses.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The course the actor wishes to copy the final score structure from has a final score structure defined.
Postcondition: The final score structure is accessible in the target course.
Summary: The actor imported the final score structure from another course into the current course, and the final score structure is defined for such current course.
Priority: Could have



Teacher imports a final score structure from another course
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Teacher Interface
display interface
select import final score structure option
return response
select course
send course id
alt
[Valid course id]
retrieve data
return data
alt
[The target course has the same set of assessments]
return final score structure
update display
[The target course does not have the same set of assessments]
return error
display error page
[Invalid course]
return error
display error page
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.17: Teacher imports a final score structure from another course

2.7.4.2.12 Teacher exports to CSV file

Teacher desires to export a set of marks, partial and/or final grades into a CSV file.

Goal: Teacher exports marks, partial grades and/or final grades with the corresponding SIS IDs to a CSV file.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The final grades, partial grades, and/or marks with the corresponding SIS IDs are stored in a CSV file.
Summary: The actor can export final grades, partial grades, and/or marks with the corresponding SIS IDs such that they are saved in a CSV file.
Priority: Should have

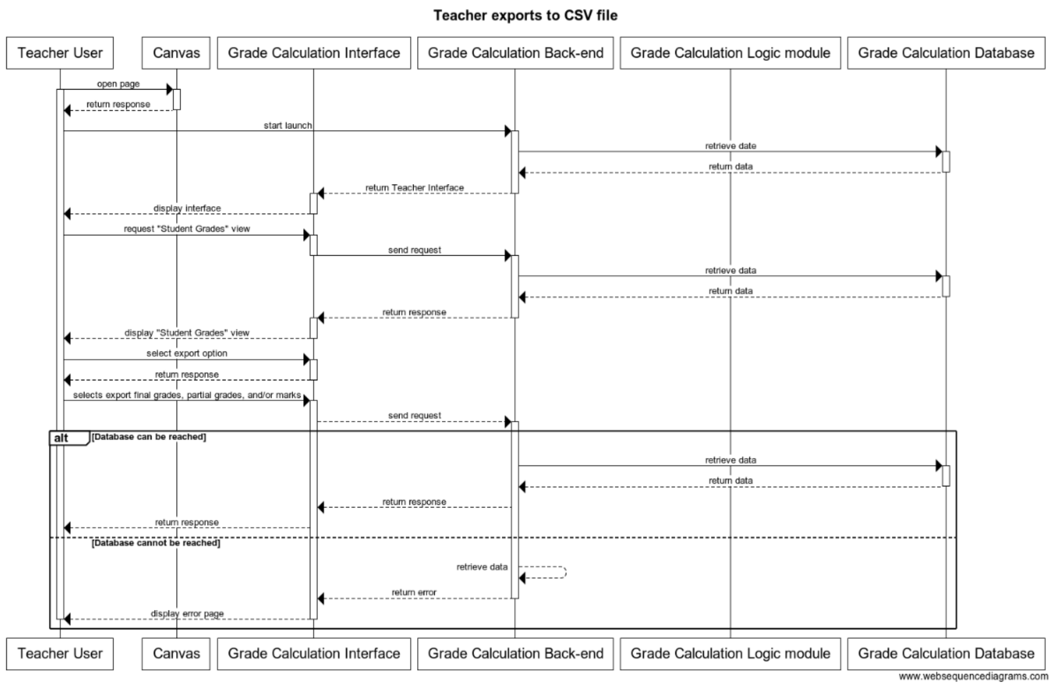

Teacher exports to CSV file
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve date
return data
return Teacher Interface
display interface
request "Student Grades" view
send request
retrieve data
return data
return response
display "Student Grades" view
select export option
return response
selects export final grades, partial grades, and/or marks
send request
alt
[Database can be reached]
retrieve data
return data
return response
return response
[Database cannot be reached]
retrieve data
return error
display error page
Teacher User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.18: Teacher exports to CSV file

2.7.4.2.13 Teacher performs a test calculation

The teacher can perform a test calculation in Grade Calculation. To do this they need to navigate to the Grade Calculation tab in Canvas. After this action they can enter test marks, and trigger the calculation.

Goal: The Teacher can insert test marks to evaluate the outcome of a final score structure.
Precondition:
The actor is enrolled as a Teacher for the specific course.
The actor has the permissions specified in Section 2.4.1 of the URD [1].
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
The final score structure the actor wants to do a test calculation for is defined.
The database is accessible.
Postcondition: The actor obtains a final grade for the final score structure using the test marks as input.
Summary: The actor can perform a test calculation using input marks and a predefined final score structure and grading scheme.
Priority: Could have

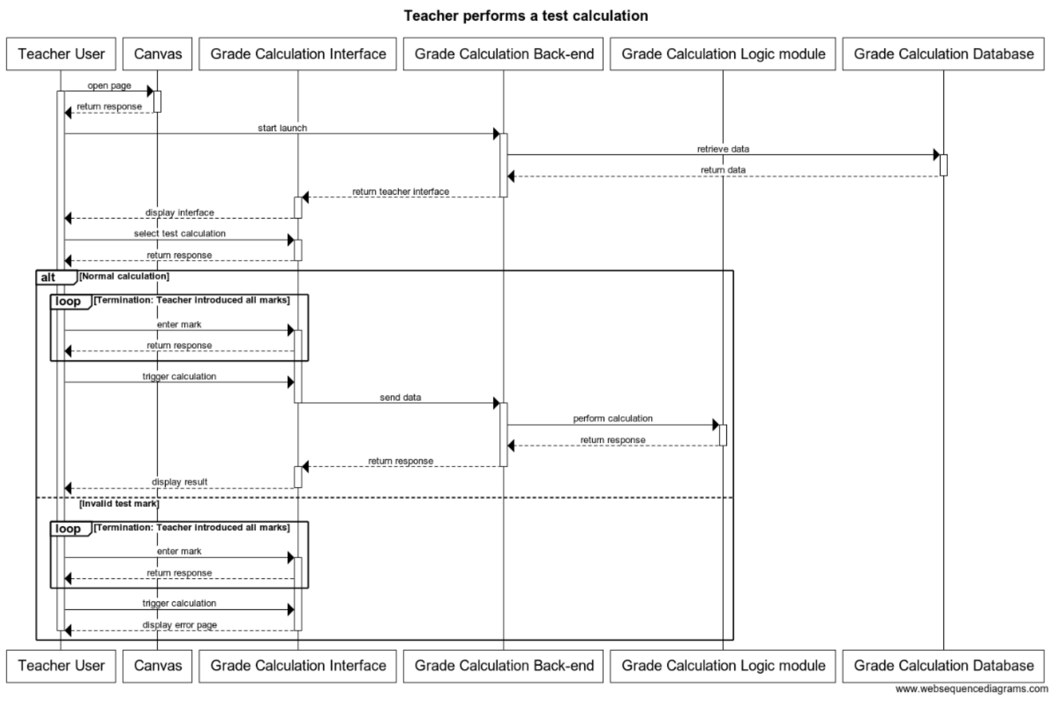


Figure 2.19: Teacher makes a test calculation

2.7.4.3 Administrator interface

2.7.4.3.1 Administrator opens Grade Calculation Teacher Interface for any course

Administrators can open Grade Calculation. To do this they will need to navigate to the Grade Calculation tab in Canvas. If the database is accessible, the user is able to see the Teacher Interface, otherwise the teacher receives an error page.

Goal: Administrator opens the Grade Calculation Teacher Interface in a specific course in Canvas.
Precondition: The actor is an Administrator.
The actor is logged into Canvas.
The course the actor wants to access exists.
The specific course has Grade Calculation enabled.
Postcondition: The Teacher interface of Grade Calculation is shown for the accessed course.
Summary: The actor requests the Grade Calculation Teacher Interface for the accessed course. Grade Calculation will show the Teacher interface for this course.
Priority: Could have

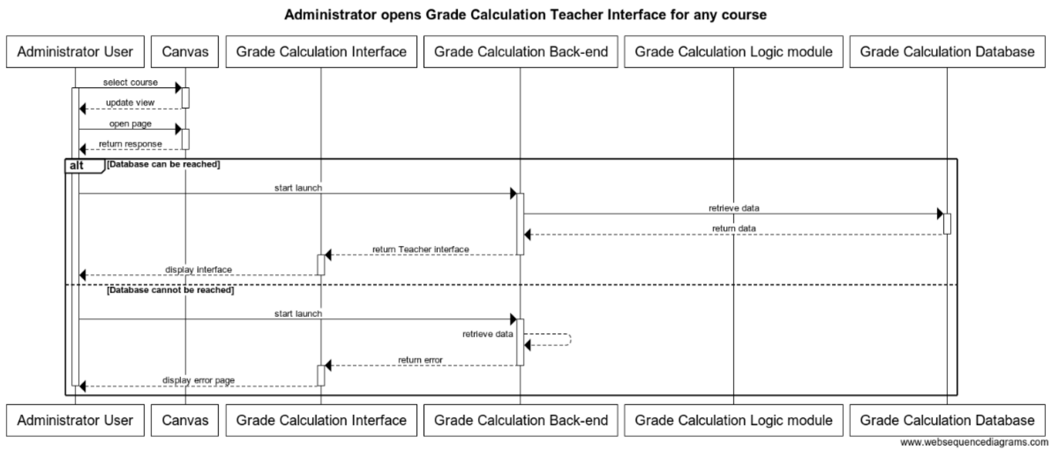

Administrator opens Grade Calculation Teacher Interface for any course
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
select course
update view
open page
return response
alt
[Database can be reached]
start launch
retrieve data
return data
return Teacher interface
display interface
[Database cannot be reached]
start launch
retrieve data
return error
display error page
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.20: Administrator opens Grade Calculationfor any course

2.7.4.3.2 Administrator defines or edits the final score structure and selects grading scheme

Administrator defines a new final score structure or edits an existing one in Grade Calculation. Moreover, since a final score structure cannot be validated without a grading scheme, the administrator must select a grading scheme. To do this they need to use the “Grading Structure" view in the Teacher Interface to build the desired structure, and select their desired grading scheme.

Goal: Administrator defines a new final score structure or edits an existing one and selects a grading scheme, all of which will be used to determine the final grade.
Precondition:
The actor is an Administrator.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
All assessments are defined.
The database is accessible.
Postcondition: The final score structure and the selected grading scheme are saved into the database.
Summary: The actor defines the final score structure that will be used to calculate the final score whenever necessary.
Priority: Could have

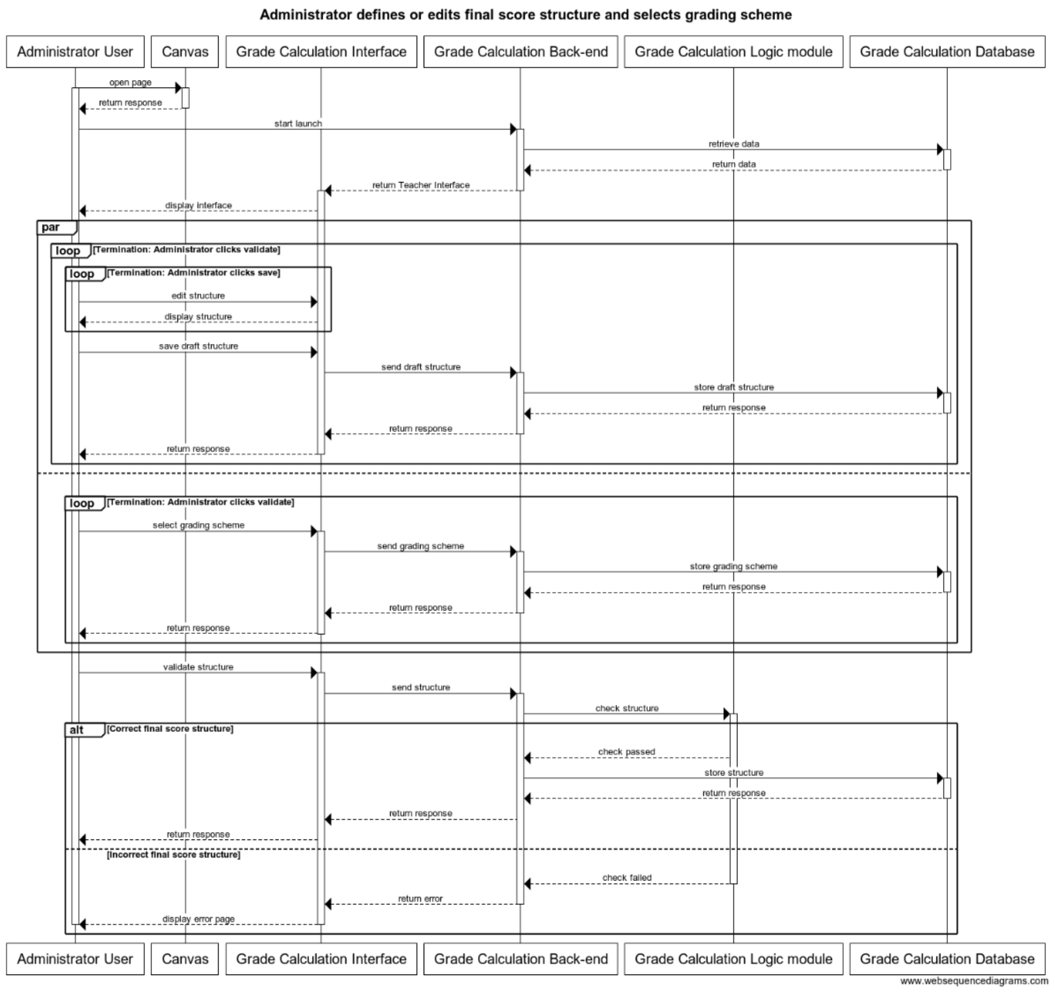


Figure 2.21: Administrator defines or edits final score structure and selects grading scheme

2.7.4.3.3 Administrator views audit logs

Administrators views audit logs in Grade Calculation which displays the changes made within the plugin.

Goal: Administrator views audit logs.
Precondition:
The actor is an Administrator.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The audit logs are displayed.
Summary: The actor is able to view the audit logs of the plugin. This way the actor can investigate potential problems.
Priority: Could have

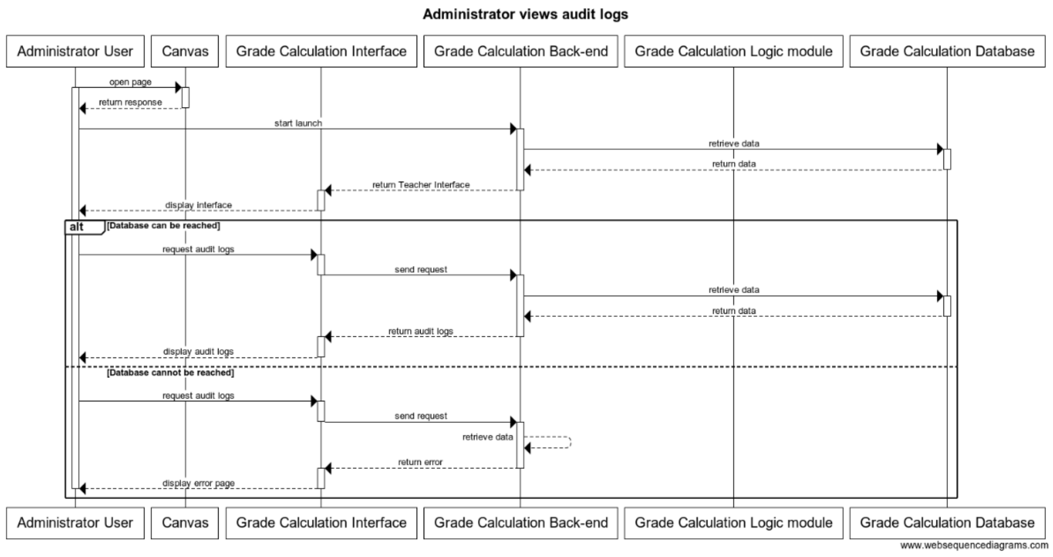

Administrator views audit logs
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve data
return data
return Teacher Interface
display interface
alt
[Database can be reached]
request audit logs
send request
retrieve data
return data
return audit logs
display audit logs
[Database cannot be reached]
request audit logs
send request
retrieve data
return error
display error page
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.22: Administrator views audit logs

2.7.4.3.4 Administrator filters audit logs

Administrator wants to view the audit logs to which a specific filter applies.

Goal: Administrator filters audit logs.
Precondition:
The actor is enrolled as a Administrator for the specific course.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The actor has navigated to the audit logs view.
The specific course has Grade Calculation enabled.
Postcondition: Only the audit logs for which the filter applies are displayed.
Summary: The actor is able to view the filtered audit logs of the plugin. This way the actor can investigate potential problems.
Priority: Won't have

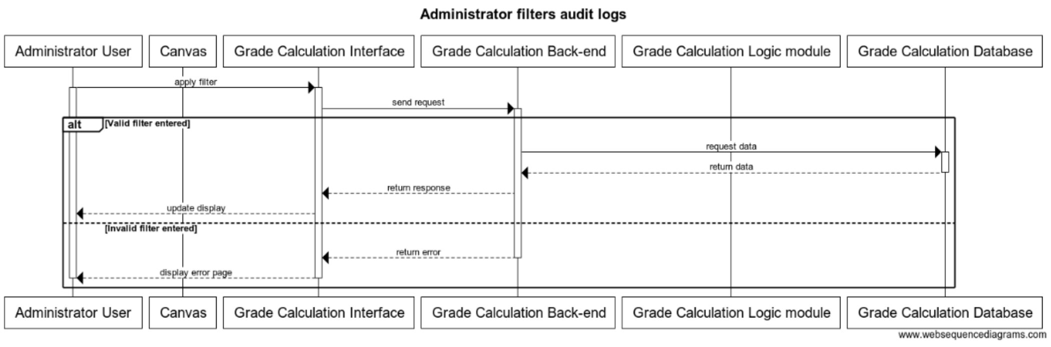


Figure 2.23: Administrator filters audit logs

2.7.4.3.5 Administrator exports to CSV

Administrator desires to export a set of marks, partial and/or final grades into a CSV file.

Goal: Administrator exports marks, partial grades and/or final grades with the corresponding SIS IDs to a CSV file.
Precondition:
The actor is enrolled as a Administrator for the specific course.
The actor is logged into Canvas.
The actor has navigated to the specific course.
The specific course has Grade Calculation enabled.
Postcondition: The final grades, partial grades, and/or marks with the corresponding SIS IDs are stored in a CSV file.
Summary: The actor can export final grades, partial grades, and/or marks with the corresponding SIS IDs such that they are saved in a CSV file.
Priority: Could have

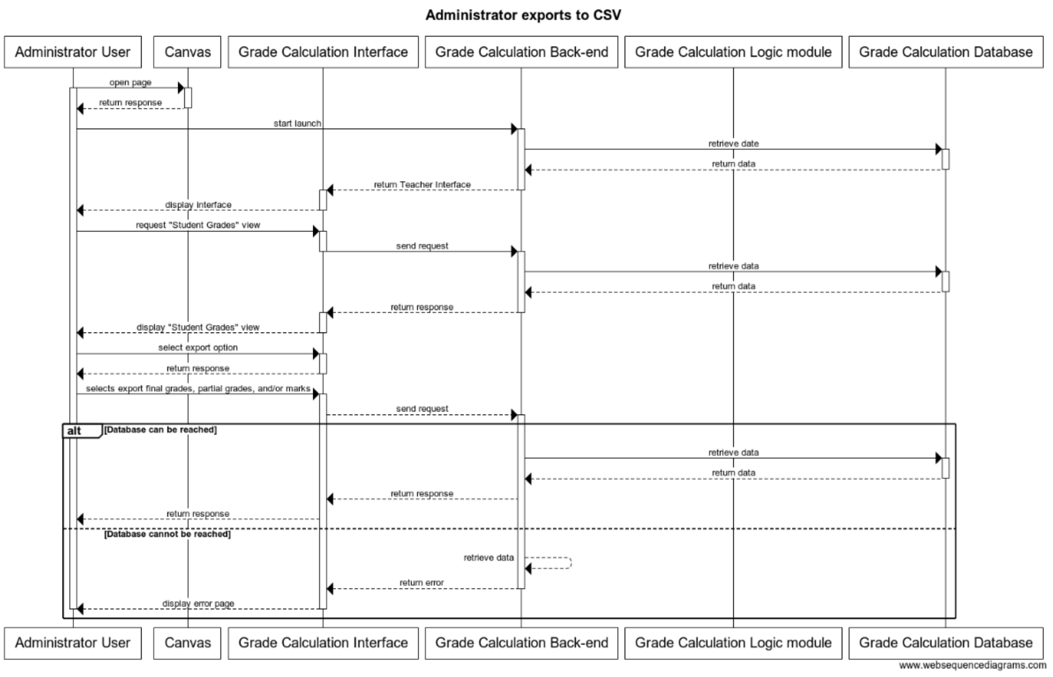

Administrator exports to CSV
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
open page
return response
start launch
retrieve date
return data
return Teacher Interface
display interface
request "Student Grades" view
send request
retrieve data
return data
return response
display "Student Grades" view
select export option
return response
selects export final grades, partial grades, and/or marks
send request
alt
[Database can be reached]
retrieve data
return data
return response
return response
[Database cannot be reached]
retrieve data
return error
display error page
Administrator User
Canvas
Grade Calculation Interface
Grade Calculation Back-end
Grade Calculation Logic module
Grade Calculation Database
www.websequencediagrams.com

Figure 2.24: Administrator exports to CSV

Chapter 3

Specific Requirements

In this chapter, all software requirements of the product to be developed are specifically stated. The software requirements are based on tequirements as stated in the URD [1]. For prioritizing requirements, the MoSCoW model is used [7]. The capital letters in MoSCoW stand for:

Priority	Abbreviation	Explanation
Must have	M	These requirements will be implemented by the end of this project.
Should have	S	These requirements should ideally but not essentially be implemented during this project.
Could have	C	These requirements will be implemented if there is still time available after having implemented the 'Must have' and 'Should have' requirements.
Won't have	W	These requirements will not be implemented during this project and could be implemented in a later project.

3.1 Functional Requirements

3.1.1 Student Interface

3.1.1.1 Attributes

SR - 1 **Must have**
course: `object`
An object describing all variables set by a teacher or administrator including the grading scheme used, the description of the final score structure, the final score structure itself, all assessments, and the grade definitions.

SR - 2 **Must have**
marks: `array`

SR - 2 **Must have**
An array of marks obtained by the student. Every element is an object which holds the assessment ID, the date of change, and the mark and score the student received.

SR - 3 **Must have**
grades: `array`
An array of the grades obtained by the student. Every element is an object which holds the assessment ID, the date of change, the grade the student received, and whether this grade is seen as a pass.

3.1.1.2 Operations

SR - 4 **Must have**
convert_json_data_source(): array
Transforms a JSON object into an array which can be displayed in the student interface.
Input

- $final\_score\_structure$: `JSON object`
- $course$: `object`
- $marks$: `array`
- $grades$: `array`

Precondition

- $final\_score\_structure \neq NULL$
- $final\_score\_structure$ is defined as described in 2.7.3.2.
- $course \neq NULL$
- $marks \neq NULL$
- $grades \neq NULL$

Postcondition

- The transformed JSON object is returned.

3.1.2 Teacher Interface - Grading Structure

3.1.2.1 Attributes

SR - 5 **Must have**
grading_schemes: `array`
An array that contains all the different grading schemes the teacher can select.

SR - 6 **Must have**
course: `object`
An object describing all variables previously saved by a teacher or administrator including the grading scheme used, the description of the final score structure, the final score structure itself, all assessments, and the grade definitions.

3.1.2.2 Operations

SR - 7 **Could have**

import_final_score_structure(): void

The teacher imports the final score structure from another course.

Input

- $course\_id$: `integer`

Precondition

- $course\_id$ exists in the database.
- $course\_id \neq NULL$.

Postcondition

- A final score structure from the course selected by the teacher is loaded to the current course's "Grading Structure" view.
- The result is displayed to the teacher.
- If the source course has different assessments than the target course, then the teacher is notified.

SR - 8 **Must have**

select_grading_scheme(): void

The teacher selects the grading scheme they desire.

Input

- $grading\_scheme\_name$: `string`

Precondition

- $grading\_scheme\_name \neq NULL$

Postcondition

- The teacher selects the grading scheme they desire and the choice is saved to the database.
- The result is displayed to the teacher.

SR - 9 **Should have**

edit_description(): void

Teacher can edit the description of the final score structure.

Input

- $description$: `string`

Postcondition

- The teacher edits the description of the final score structure and all edits are saved to the database.
- The result is displayed to the teacher.

SR - 10 **Must have**

add_assessment(): void

The teacher adds an assessment to an assessment set in the final score structure.

SR - 10 **Must have**

Input

- $assessment\_set\_child\_array$: `array`
- $minimum\_requirement$:`string, float or nil`
- $assessment\_name$: `string`
- $grading\_type$: `string`
- max: `float`
- due_date: `Time`
- $weight$: `float or nil`

Postcondition

- The chosen assessment is now a child of the assessment set.
- The result is displayed to the teacher.

SR - 11 **Should have**

delete_assessment(): void

The teacher deletes an assessment from the final score structure.

Postcondition

- The chosen assessment is deleted from the final score structure.
- The result is displayed to the teacher.

SR - 12 **Should have**

edit_assessment(): void

The teacher edits an assessment in the final score structure.

Input

- $weight$: `integer`
- $min\_requirement$: `string`
- $name$: `string`

Postcondition

- The variables of the assessment are overwritten by the input.
- The result is displayed to the teacher.

SR - 13 **Could have**

add_constant(): void

The teacher adds a constant to an assessment set in the final score structure.

Input

- $name$: `string`
- $value$: `number`

Postcondition

- The constant is now a child of the assessment set.
- The result is displayed to the teacher.

SR - 14 **Must have**

add_assessment_set(): void

SR - 14 **Must have**

The teacher adds an assessment set to the final score structure.

Input

- $weight$: `float`
- $min\_requirement$: `float`
- $name$: `string`
- $calculation\_type$: `string`

Postcondition

- The assessment set with the input variables is displayed in the final score structure.

SR - 15 **Should have**

delete_assessment_set(): void

The teacher deletes an assessment set from the final score structure.

Precondition

- The children array of the assessment set must be empty.

Postcondition

- The chosen assessment set and all of its children are deleted from the final score structure.
- The result is displayed to the teacher.

SR - 16 **Should have**

edit_assessment_set(): void

The teacher edits an assessment set in the final score structure.

Input

- $weight$: `integer`
- $min\_requirement$: `string`
- $name$: `string`
- $calculation\_type$: `string`

Postcondition

- The variables of the assessment set are overwritten by the input.
- The result is displayed to the teacher.

SR - 17 **Must have**

add_multiple_attempts(): void

The teacher creates a new attempt for an assessment in the final score structure.

Input

- $assessment$: `MarkNode`
- $calculationMethod$: `string`

Postcondition

- An assessment set with the given calculation method is created in place of the original assessment. The original assessment, along with its new attempt, are added to this assessment set.
- The result is displayed to the teacher.

SR - 18 **Could have**

add_condition_component(): void

The teacher creates a new condition block in the final score structure.

Input

- $condition$: `string`
- $choice1$: `array`
- $choice2$: `array`

Precondition

- The two choice arrays are defined.

Postcondition

- The condition block is added to the assessment set.
- The result is displayed to the teacher.

SR - 19 **Must have**

save(): void

The selected grading scheme, final score structure, and the description of the final score structure are saved to the database.

Input

- $grading\_scheme\_name$: `string`
- $translated\_final\_score\_structure$: `array`
- $description$: `string`

Postcondition

- The selected grading scheme, final score structure, and the description of the final score structure are saved to the database.

SR - 20 **Must have**

submit(): void

The final score structure is validated and if the validation tests pass the final score structure is submitted to the database.

Input

- $grading\_scheme\_name$: `string`
- $translated\_final\_score\_structure$: `array`
- $description$: `string`

Postcondition

- If all validation tests are passed, the final score structure, the selected grading scheme and the description is saved to the database. The result is then displayed to the teacher.
- If at least one of the validation tests fail, an error is thrown.

SR - 21 **Could have**

warning(): void

A warning is shown to the Teacher if they define the final score structure before defining assignments in Canvas.

SR - 21 **Could have**

Precondition

- No assignments defined in Canvas.

Postcondition

- The warning is shown to the teacher.

SR - 22 **Could have**

notifyTeacher(): void

The teacher is notified to manually recompute the final and partial grades, if a change in the marks, final score structure, or the grading scheme is made.

Precondition

- A change is made in the marks, final score structure, or the grading scheme.

Postcondition

- The pop-up is shown to the teacher stating that they should recompute the final and partial grades.

3.1.3 Teacher Interface - Student Grades

3.1.3.1 Attributes

SR - 23 **Must have**

course: `object`

An object describing all variables previously saved by a teacher or administrator including the grading scheme used, the description of the final score structure, the final score structure itself, all assessments, and the grade definitions.

SR - 24 **Must have**

users: `array`

An array which contains elements holding the user's name and the marks and grades that correspond to this user.

3.1.3.2 Operations

SR - 25 **Should have**

mute(): void

The teacher mutes an assessment so that the students can no longer see the result.

SR - 25 **Should have**

Input

- $assessment\_id$: string

Precondition

- The assessment is unmuted.

Postcondition

- The assessment is now muted.
- The result is displayed to the teacher.

SR - 26 **Should have**

unmute(): void

The teacher unmutes an assessment so that the students can see the result.

Input

- $assessment\_id$: string

Precondition

- The assessment is unmuted.

Postcondition

- The assessment is unmuted.
- The result is displayed to the teacher.

SR - 27 **Should have**

mute_set(): void

The teacher mutes an assessment set so that the students can no longer see the result.

Input

- $assessment\_id$: string

Precondition

- The assessment set is unmuted.

Postcondition

- The assessment set is now muted.
- The result is displayed to the teacher.

SR - 28 **Should have**

unmute_set(): void

The teacher unmutes an assessment set so that the students can see the result.

Input

- $assessment\_id$: string

Precondition

- The assessment set is unmuted.

Postcondition

- The assessment set is unmuted.
- The result is displayed to the teacher.

SR - 29 **Could have**

filter(): void

The teacher applies a filter condition to the student grades table to only show students that fall within this filter condition.

Input

- $filter$: string

Postcondition

- Only the students and the corresponding grades are displayed for the students that fall within the filter condition.

SR - 30 **Should have**

mark_adjustment(): void

The teacher applies a mark adjustment to an assessment's marks using a formula they defined.

Input

- $base$: number
- $multiplier$: number

Postcondition

- The adjusted marks are shown to the teacher in place of the original marks.

SR - 31 **Should have**

search(): void

The teacher applies a search condition to the student grades table to only show students that fall within this search condition.

Input

- $search$: string

Postcondition

- Only the students and the corresponding grades are displayed for the students that fall within the search condition.

SR - 32 **Could have**

test_calculation(): void

Teacher enters marks for assessments to do a test calculation to check the final score structure.

input

- $marks$: `array`
- $translated\_final\_score\_structure$: `array`

Precondition

- $translated\_final\_score\_structure \neq NULL$
- $translated\_final\_score\_structure$ is defined as described in 2.7.3.2.

Postcondition

- The calculated final score is displayed to the teacher.

SR - 33 **Should have**
display_logs(): void
The teacher views audit logs.
input
- $course\_id$: `integer`

Precondition
- $course\_id$ exists in the database.
- $course\_id \neq NULL$.

Postcondition
- Audit logs of the course are retrieved.
- The result is displayed to the teacher.

3.1.3.3 DataSource-JSON Conversion

SR - 34 **Must have**
convert_data_source_json(): JSON object
Transforms the array which is used for display into a JSON Object.
Input
- $translated\_final\_score\_structure$: `array`

Precondition
- $translated\_final\_score\_structure \neq NULL$
- $translated\_final\_score\_structure$ is defined as described in 2.7.3.2.

Postcondition
- The transformed final score structure object is returned as a JSON object.

3.1.4 Administrator Interface - Grading Structure

3.1.4.1 Attributes

SR - 35 **Could have**
grading_schemes: `array`
An array that contains all the different grading schemes the administrator can select.

SR - 36 **Could have**
course: `object`
An object describing all variables previously saved by a teacher or administrator including the grading scheme used, the description of the final score structure, the final score structure itself, all assessment, and the grade definitions.

3.1.4.2 Operations

SR - 37 **Could have**

select_grading_scheme(): void

The administrator selects the grading scheme they desire.

Input

- $grading\_scheme\_name$: `string`

Precondition

- $grading\_scheme\_name \neq NULL$

Postcondition

- The administrator selects the grading scheme they desire and the choice is saved to the database.
- The result is displayed to the administrator.

SR - 38 **Could have**

save(): void

The selected grading scheme, final score structure, and the description of the final score structure are saved to the database.

Input

- $grading\_scheme\_name$: `string`
- $translated\_final\_score\_structure$: `array`
- $description$: `string`

Postcondition

- The selected grading scheme, final score structure, and the description of the final score structure are saved to the database.

SR - 39 **Could have**

submit(): void

The final score structure is validated and if the validation tests pass the final score structure is submitted to the database.

Input

- $grading\_scheme\_name$: `string`
- $translated\_final\_score\_structure$: `array`
- $description$: `string`

Postcondition

- If all validation tests are passed, the final score structure, the selected grading scheme and the description is saved to the database. The result is displayed to the administrator.
- If at least one of the validation tests fail, an error is thrown.

SR - 40 **Could have**

add_assessment(): void

The administrator adds an assessment to an assessment set in the final score structure.

SR - 40 **Could have**

Input

- $assessment\_set\_child\_array$: `array`
- $assessment$: `MarkNode`
- $weight$: `string`

Postcondition

- The chosen assessment is now a child of the assessment set.
- The result is displayed to the administrator.

SR - 41 **Could have**

delete_assessment(): void

The administrator deletes an assessment from the final score structure.

Postcondition

- The chosen assessment and all of its children are deleted from the final score structure.
- The result is displayed to the administrator.

SR - 42 **Could have**

edit_assessment(): void

The administrator edits an assessment in the final score structure.

Input

- $weight$: `integer`
- $min\_requirement$: `string`
- $name$: `string`

Postcondition

- The variables of the assessment are overwritten by the input.
- The result is displayed to the administrator.

SR - 43 **Could have**

add_constant(): void

The administrator adds a constant to an assessment set in the final score structure.

Input

- $name$: `string`
- $value$: `number`

Postcondition

- The constant is now a child of the assessment set.
- The result is displayed to the administrator.

SR - 44 **Could have**

add_assessment_set(): void

The administrator adds an assessment set to the final score structure.

SR - 44 **Could have**

Input

- $weight$: `float`
- $min\_requirement$: `float`
- $name$: `string`
- $calculation\_type$: `string`

Postcondition

- The assessment set with the input variables is displayed in the final score structure.
- The result is displayed to the administrator.

SR - 45 **Could have**

delete_assessment_set(): void

The administrator deletes an assessment set from the final score structure.

Postcondition

- The chosen assessment set and all of its children are deleted from the final score structure.
- The result is displayed to the administrator.

SR - 46 **Could have**

edit_assessment_set(): void

The administrator edits an assessment set in the final score structure.

Input

- $weight$: `integer`
- $min\_requirement$: `string`
- $name$: `string`
- $calculation\_type$: `string`

Postcondition

- The variables of the assessment set are overwritten by the input.
- The result is displayed to the administrator.

SR - 47 **Could have**

add_multiple_attempts(): void

The administrator creates a new attempt for an assessment in the final score structure.

Input

- $assessment$: `MarkNode`

Postcondition

- An assessment set of *maximum* calculation method is created in place of the original assessment. The original assessment, along with its new attempt, are added to this assessment set.
- The result is displayed to the administrator.

SR - 48 **Could have**

add_condition_component(): void

The Administrator creates a new condition block in the final score structure.

Input

- $condition$: `string`
- $choice1$: `array`
- $choice2$: `array`

Precondition

- The two choice arrays are defined.

Postcondition

- The condition block is added to the assessment set.
- The result is displayed to the teacher.

3.1.5 Administrator Interface - Student Grades

3.1.5.1 Attributes

SR - 49 **Could have**

course: `object`

An object describing all variables previously saved by the teacher or administrator including grading scheme used, description of the final score structure, the final score structure itself, all assessment, and grade definitions.

SR - 50 **Could have**

users: `array`

An array which contains elements holding the user's name and the marks and grades that correspond to this user.

3.1.5.2 Operations

SR - 51 **Could have**

display_logs(): void

The administrator views audit logs.

input

- $course\_id$: `integer`

Precondition

- $course\_id$ exists in the database.
- $course\_id \neq NULL$.

Postcondition

- Audit logs of the course are retrieved.
- The result is displayed to the administrator.

3.1.5.3 DataSource-JSON Conversion

SR - 52 **Could have**
convert_data_source_json(): JSON object
Transforms the array which is used for display into a JSON Object.
Input
- $translated\_final\_score\_structure$: `array`

Precondition
- $translated\_final\_score\_structure \neq NULL$
- $translated\_final\_score\_structure$ is defined as described in 2.7.3.2.

Postcondition
- The transformed final score structure object is returned as a JSON object.

3.1.6 Course

3.1.6.1 Attributes

SR - 53 **Must have**
grading_scheme: `string`
The representation of the grading scheme used in the course.

SR - 54 **Must have**
score_structure: `json`
The representation of the final score structure used in the course. Matches the specification described in 2.7.3.2

SR - 55 **Should have**
score_structure_description: `string`
The description of the score structure added by the teacher

3.1.7 Assessment

3.1.7.1 Attributes

SR - 56 **Must have**
course: `Course`
A reference to the course this assessment belongs to

SR - 57 **Must have**
name: `string`
The name of the assessment

SR - 58 **Must have**
mark_type: `string`
The representation type of the type of mark expected, either 'points', 'percent', or 'complete_incomplete'

SR - 59 **Must have**
required: `string or float`
The minimum required mark for this assessment to pass the course, depending on the mark_type

SR - 60 **Should have**
mark_adjustment: `boolean`
Determines whether this assessment's marks are adjusted using the adjustment_base and adjustment_multiplier

SR - 61 **Should have**
adjustment_base: `float`
Determines n in $n + m \cdot \frac{\text{points obtained}}{\text{total number of points}}$ when the assessment's marks are mark adjusted

SR - 62 **Should have**
adjustment_multiplier: `float`
Determines m in $n + m \cdot \frac{\text{points obtained}}{\text{total number of points}}$ when the assessment's marks are mark adjusted

3.1.8 Student

3.1.8.1 Attributes

SR - 63 **Should have**
name: `string`
The name of this student in Canvas

SR - 64 **Should have**
student_id: `string`
The SIS ID of this student

SR - 65 **Must have**
course: `Course`
A reference to the course this student belongs to

3.1.9 Mark

3.1.9.1 Attributes

SR - 66 **Must have**
max: `String`
The maximum value that this mark can have

SR - 67 **Must have**
assessment_id: `String`
The assessment this mark belongs to

SR - 68 **Must have**
student_id: `String`
The student this mark belongs to

SR - 69 **Must have**
value: `String`
The value a student obtained for an assessment. This can be 'complete', 'incomplete', 'no_show', 'excused', or a string containing the numeric value achieved.

SR - 70 **Must have**
grading_type: `String`
The type of the mark,which can be 'points', 'percentage' or 'complete_incomplete'.

3.1.9.2 Operations

SR - 71 **Must have**
transform(): `score`
Takes a mark and converts it into a score
Postcondition

- the parameters of the mark are the parameters of the returned score object

SR - 72 **Should have**
adjust(): `number or string`
Adjusts the value of a mark ($\frac{score}{max\_score} \cdot multiplier + base$)

SR - 72 **Should have**

Input

- $score$: `float`
- max_score: `float`
- $base$: `float`
- $multiplier$: `float`

Postcondition

- The adjusted mark is returned

3.1.10 Grade

3.1.10.1 Attributes

SR - 73 **Must have**

student: `Student`

A reference to the student which the grade belongs to

SR - 74 **Must have**

grade_id: `string`

The unique ID of the grade in the score structure, final if this grade represents the final grade, or mandatory if this grade represents the mandatory assessments

SR - 75 **Must have**

grade: `string`

Represents the grade the student has received, which can be ‘excused’, ‘pass’, ‘fail’, ‘N/A’ or a string returned by the grading scheme

SR - 76 **Should have**

muted: `boolean`

Determines whether the grade is muted

SR - 77 **Should have**

updated_at: `date`

Determines when the grade was last updated

3.1.10.2 Operations

SR - 78 **Should have**

convert_fail(): `void`

If the grade is ’fail’, replaces it with *constant* instead.

SR - 78 **Should have**
Input
- *constant* : `string`

Precondition
- For each student the final grade is calculated and is set to "fail" if not all minimum requirements are met.

Postcondition
- For all students of the course, each course grade that is already in the database, is updated with the possibly changed grade

3.1.11 Score

3.1.11.1 Attributes

SR - 79 **Must have**
value: `rational or nil`
The value of the score, can be numeric or 'excused'

SR - 80 **Must have**
type: `string`
The representation type of the value, either 'points' or 'excused'

3.1.12 Constant

3.1.12.1 Attributes

SR - 81 **Could have**
value: `numeric`
The value of the constant

SR - 82 **Could have**
max: `numeric`
The maximum value of the constant, used for turning the constant into a fraction

3.1.12.2 Operations

SR - 83 **Could have**
transform(): `score`
Returns the value of the constant as a Score object

SR - 83 **Could have**

Postcondition

- the parameters of the constant are the parameters of the returned score.

3.1.13 Calculation Method

3.1.13.1 Unweighted average

3.1.13.1.1 Operations

SR - 84 **Must have**

calculate(): `score`

Takes an array of scores and computes the unweighted average

Input $scores$: `array of score objects`

Postcondition

- the unweighted average is returned as a score object

3.1.13.2 Weighted average

3.1.13.2.1 Operations

SR - 85 **Must have**

calculate(): `score`

Takes an array of scores and an array of weights and computes the weighted average

Input

- $scores$: `array of score objects`
- $weights$: `array of weights`

Precondition

- all elements of $scores$ are of type `score` with type points
- all $weights$ input as percentages are transformed into fractions.
- all elements of $weights$ add up to one
- $scores$ and $weights$ are of the same length

Postcondition

- the weighted average is returned as a `score object`

3.1.13.3 Best x of y

3.1.13.3.1 Operations

SR - 86 **Should have**

calculate(): `score`

SR - 86 **Should have**

Takes an array of scores, an array of booleans and an integer x and computes the unweighted average of the best given scores, always including the scores i for which boolean i is true

Input

- $scores$: `array of score objects`
- $always\_counts$: `array of booleans`
- x: `int`

Precondition

- x is smaller than or equal to size of the array of scores
- The sizes of the arrays are equal.
- The number of "true" valued booleans is smaller than or equal to x

Postcondition

- The best x of y is returned as a score object

3.1.13.4 Worst x of y

3.1.13.4.1 Operations

SR - 87 **Should have**

calculate(): `score`

Takes an array of scores, an array of booleans and an integer x, and computes the unweighted average of the worst given scores, always including the scores i for which boolean i is true

Input

- $scores$: `array of score objects`
- $always\_counts$: `array of booleans`
- x : `int`

Precondition

- x is smaller than the array size of scores
- The sizes of the arrays are equal.
- The number of "true" valued booleans is smaller than or equal to x

Postcondition

- The worst x of y is returned as a score object

3.1.13.5 Sum

3.1.13.5.1 Operations

SR - 88 **Should have**

calculate(): `score`

Takes an array of scores, computes the sum of score values, and takes the minimum of that sum and 1

Input

- $scores$: `array of score objects`

Postcondition

- The result is returned as a score object

3.1.13.6 Subtraction

3.1.13.6.1 Operations

SR - 89 **Should have**
calculate(): `score`
Takes an array of scores, computes the subtraction of the first score value and the rest of the scores values
Input
- $scores$: `array of score objects`

Postcondition
- The result is returned as a score object

3.1.13.7 Maximum

3.1.13.7.1 Operations

SR - 90 **Should have**
calculate(): `score`
Takes an array of scores, returns the maximum score value
Input
- $scores$: `array of score objects`

Postcondition
- The result is returned as a score object

3.1.13.8 Minimum

3.1.13.8.1 Operations

SR - 91 **Should have**
calculate(): `score`
Takes an array of scores, returns the minimum score value
Input
- $scores$: `array of score objects`

Postcondition
- The result is returned as a score object

3.1.14 Condition

3.1.14.1 Operations

SR - 92 **Could have**
calculate(): `score`
Takes two scores and a condition, and returns the first score if the condition holds, otherwise it returns the second score

SR - 92 **Could have**

Input

- $score1$: `score object`
- $score2$: `score object`
- $condition$: `statement that evaluates to true or false`

Postcondition

- The result is returned as a score object

3.1.15 Grading schemes

3.1.15.1 Grading scheme handler

3.1.15.2 Operations

SR - 93 **Must have**

apply(): `string`

Takes a score and a grading scheme, and returns the grade obtained by applying the grading scheme to the score

Input

- $score$: `score object`
- $grading\_scheme$: `grading scheme object`

Postcondition The result is returned as a string

3.1.15.3 0-10 rounding to nearest integer

3.1.15.3.1 Operations

SR - 94 **Must have**

compute(): `string`

Takes a score and returns the grade obtained by applying the 1 to 10 integer grading scheme to the score

Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.4 0-10 rounding to 1 decimal

3.1.15.4.1 Operations

SR - 95 **Must have**

compute(): `string`

Takes a score and returns the grade obtained by applying the grading scheme which rounds a score to one decimal

Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.5 0-10 rounding to 2 decimals

3.1.15.5.1 Operations

SR - 96 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which rounds a score to two decimals
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.6 0-10 rounding to nearest half

3.1.15.6.1 Operations

SR - 97 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which rounds a score to the nearest half
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.7 0-10 rounding to nearest quarter

3.1.15.7.1 Operations

SR - 98 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which rounds a score to the nearest quarter
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.8 UK letter grade

3.1.15.8.1 Operations

SR - 99 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which transforms the score to an UK letter grade
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.9 US letter grade

3.1.15.9.1 Operations

SR - 100 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which transforms the score to an US letter grade
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.15.10 Percentage

3.1.15.10.1 Operations

SR - 101 **Must have**
compute(): `string`
Takes a score and returns the grade obtained by applying the grading scheme which transforms the score to an percentage
Input

- $score$: `score object`

Postcondition The result is returned as a string

3.1.16 Grade Calculator

3.1.16.1 Operations

SR - 102 **Must have**
calculate(): `void`
Calculates the grades by the defined $course$ score structure, for each student in $course$, using the marks in the database and stores them in the database.

SR - 102 **Must have**

Input

- $course$: `Course entry`

Precondition

- All marks used in the score structure, are in the database for each student of the course.
- The score structure of the $course$ has been defined and matches the specifications listed in 2.7.3.2.
- The grading scheme of the $course$ has been defined.

Postcondition

- For each student the partial grades are calculated
- For each student the final grade is calculated and is set to "fail" if not all minimum requirements are met.
- For all students of the $course$, each $course$ grade that is not in the database, is added to the database
- For all students of the $course$, each $course$ grade that is already in the database, is updated with the possibly changed grade
- For all students of the $course$, each $course$ grade in the database that is not a grade anymore, is removed from the database

---

SR - 103 **Could have**

count(): `void`

Counts the amount of marks with 'no_show' value, the amount with 'incomplete' value and the amount equal to the sum of these two amounts.

Input

- $course$: `Course entry`
- $no\_show\_constant$ `float`
- $incomplete\_constant$ `float`
- $no\_show\_plus\_incomplete\_constant$ `float`

Precondition

- All marks used in the score structure, are in the database for each student of the course.
- The final grade of all students has already been calculated

Postcondition

- For each student the final grade is set to "fail" if one of these three amounts is lower than a specified constant

3.1.17 Import

3.1.17.1 Canvas API

3.1.17.1.1 Attributes

SR - 104 **Must have**

course_id: `int`

The Canvas course ID for which this instance will be used

3.1.17.1.2 Operations

SR - 105 **Must have**
get_course_users(): `array`
Returns the users of the course that the Canvas API provides, parses the JSON data, and returns the data as a array
Precondition

- A connection with Canvas is instantiated

Postcondition

- An array containing hashes with user information of Canvas (id, name, created_at, sortable_name, short_name) is returned

SR - 106 **Must have**
get_course_assignments(): `array`
Returns all the assignments of the course that the Canvas API provides, parses the JSON data, and returns the array containing a hash for each assignment
Precondition

- A connection with Canvas is instantiated

Postcondition

- An array containing for each assignment a hash with the provided assignment information of Canvas is returned

SR - 107 **Must have**
get_submissions(): `array`
Returns all the submissions of a specific assignment that the Canvas API provides, parses the JSON data, and returns in an array a hash for each submission
Input

- $assignment\_id$: int

Precondition

- A connection with Canvas is instantiated

Postcondition

- An array containing for each assignment a hash with the provided assignment information of Canvas is returned

3.1.17.2 Assessment importer

3.1.17.2.1 Attributes

SR - 108 **Must have**
canvas_api: `CanvasApi`
Object that can make API requests to Canvas

3.1.17.2.2 Operations

SR - 109 **Must have**

import(): `void`

Imports all Canvas assignments of the course into the assessments table of the Grade Calculation database

Precondition

Postcondition

- Canvas assignments that are not assessments in the database are added
- Canvas assignments that are already assessments in the database are updated with the imported information
- Canvas assessments that are no longer a Canvas assignment are removed from the database

3.1.17.3 CSV marks importer

3.1.17.3.1 Operations

SR - 110 **Should have**

import(): `void`

The teacher imports a CSV file with marks for an assessment it in the database

Input

- csv: `CSV object`
- assessment: `assessment database entry`

Precondition

- The file uploaded is of CSV format
- The CSV has 2 columns
- The first header name of CSV is called 'StudentID'
- The second header name of csv is equal to the value of `assessment.name`
- All students in csv are also in the database

Postcondition

- Assessment marks for students that do not have a mark for $assessment$ in the database but have a mark in csv, are added to the database.
- Assessment marks of students that already have a mark for $assessment$ and are in csv, are updated with the mark in csv.

SR - 111 **Should have**

import_multiple(): `void`

The teacher imports a CSV with marks for multiple assessments in the database

SR - 111 **Should have**

Input

- csv : `CSV object`
- $assessments$: `array of assessment database entries`

Precondition

- The file uploaded is of CSV format
- csv has $n + 1$ columns, where n is the number of assessments in $assessments$.
- The first header is called ‘StudentID’ and the rest are the names of the assessments.
- `assessments[`i`].name` equals the name of the $i + 1^{th}$ header
- All students in the CSV are also in the database

Postcondition

- For each assessment in $assessments$: Assesment marks for students that do not have a mark for assessment in the database but have a mark in csv, are added to the database.
- For each assessment in $assessments$: Assessment marks of students that already have a mark for the assessment and are in csv, are updated with the mark in the csv.

3.1.17.4 Mark importer

SR - 112 **Must have**

canvas_api: `CanvasApi`

Object that can make API requests to Canvas

SR - 113 **Must have**

import(): `void`

Imports assignment marks of the Canvas course into the database

Precondition

- For each mark that is added, the corresponding Canvas assignment is already added as assessment into the database

Postcondition

- The Canvas assignment marks are inserted/updated/removed from the database.
- For all students of the course, each assignment mark that is not in the database, is added to the database
- For all students of the course, each assignment mark that is already in the database, is updated with the possibly changed mark
- For all students of the course, each assignment mark in the database that is not a mark anymore, is removed from the database

3.1.17.5 Student importer

SR - 114 **Must have**

canvas_api: `CanvasApi`

SR - 114 **Must have**
Object that can make API requests to Canvas

SR - 115 **Must have**
import(): `void`
Imports students of the launched Canvas course, into the database
Postcondition
- All students of the Canvas course that are not in the database, are added to the database.
- All students of the Canvas course that are already in the database, are updated with the new student information.
- All students in the database that are not in Canvas anymore, are removed from the database.

3.1.18 Export

3.1.18.1 CSV exporter

3.1.18.1.1 Operations

SR - 116 **Should have**
export(): `csv object`
A teacher exports a CSV containing all students with the corresponding marks and scores for each assessment, as well as the partial and final grades
Precondition
- The teacher selects category of grades for CSV export
- The teacher clicks 'Export' button

Postcondition
- A CSV containing all students and their corresponding marks for each assessment is returned.
- The CSV has 4 columns with student ID, SIS ID, a grade name and the corresponding grade.
- The teacher chooses the directory where the resulting CSV file should be saved.

3.1.19 Controllers

3.1.19.1 Assessment Controller

3.1.19.1.1 Operations

SR - 117 **Must have**
import(): `void`
Imports assignment information of the course, from Canvas, into the database
Input `POST` request
Precondition

SR - 117 **Must have**

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition
- A request to the ASSESSMENT IMPORTER is made

SR - 118 **Must have**
index(): `void`
For each assessment of the course the information is rendered as JSON
Input `GET` request
Precondition
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher
- The assignments of the course are imported

Postcondition
- A JSON response containing the information for each assessment is returned.

SR - 119 **Must have**
show(): `void`
For a requested assessment of the course the information is rendered as JSON
Input `GET` request containing assessment id of the requested assessment
Precondition
- The requested assignment of the course is imported in the database
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition
- A JSON response containing the information of the requested assessment is returned.

SR - 120 **Should have**
create(): `void`
Creates an assessment from a CSV and adds it to the database
Input
- `POST` request containing assessment information is made
- CSV file
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Precondition

SR - 120 **Should have**

- The CSV file is properly formatted (see 110)
- The POST request contains all assessment information parameters
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- The assessment with the provided assessment information is made

---

SR - 121 **Must have**

update(): `void`

Updates information of an assessment

Input

- `POST` request containing assessment information that the user wishes to change

Precondition

- The `POST` request contains all assessment information parameters of which the user wishes to change
- The `POST` request can only contain the parameters: name, mark_type, max_score, published, muted, normalized, normalized_base, normalized_multiplier, minimum_requirement, attempt_number and attempt_assessment_id
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- The assessment with the provided assessment information is updated

---

3.1.19.2 Audit Log Controller

3.1.19.2.1 Operations

SR - 122 **Could have**

index(): `void`

Each audit log related to the course is rendered as JSON

Input `GET` request

Precondition

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- A JSON response containing the audit logs of the course is returned.

3.1.19.3 Course controller

3.1.19.3.1 Operations

SR - 123 **Must have**

SR - 123 **Must have**
show(): `void`
For the currently launched course, the information is rendered as JSON
Input `GET` request containing the current session
Precondition

- The course of the current session is in the database
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- A JSON response containing information for the currently launched course is returned.

SR - 124 **Must have**
update(): `void`
Updates grading_scheme, score_description, score_structure and draft_score_structure of the course
Input

- `POST` request containing course information that the user wishes to change

Precondition

- The course that the user wishes to update is in the database
- The `POST` request contains all course information parameters which the user wishes to change
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- The course is updated with the passed course information parameters
- If changes to either the grading_scheme or final_score_structure are made stored
- Updates are saved in the audit logs

3.1.19.4 Grade Controller

3.1.19.4.1 Operations

SR - 125 **Must have**
calculate(): `void`
Computes the grades using the information stored in the database
Input `POST` request
Precondition

SR - 125 **Must have**

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher
- All the information needed to compute the grades is present in the database. (Marks, assessments, course, students, grade definitions

Postcondition

- All the grades stored in the database

SR - 126 **Could have**

notifies(): `void`

Notifies the front-end if marks have been changed after calculating grades has happened. Gets called automatically if marks are changed.

Input `GET` request

Postcondition

- Front-end is notified if grades have been calculated before the marks where changed.

3.1.19.5 Grade Definition Controller

3.1.19.5.1 Operations

SR - 127 **Must have**

index(): `void`

For each grade definition of the course, get the information and renders it as JSON

Input `GET` request

Precondition

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher
- The grade definitions of the course are imported

Postcondition

- A JSON response containing information of each grade definition is returned

SR - 128 **Must have**

show(): `void`

For a requested grade definition the information is rendered as JSON

Input `GET` request containing the ID of the grade definition

Precondition

- The requested grade definition is in the database
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- A JSON response containing information for the requested grade definition is returned.

SR - 129 **Must have**
update(): `void`
Updates muted and published attributes of a grading definition
Input
- `POST` request containing the ID of the grade definition and the grade definition information that the user wishes to change

Precondition
- The grade definition that the user wishes to update is in the database
- The `POST` request contains all grade definition parameters which the user wishes to change
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition
- The grade definition is updated with the passed grade definition parameters

3.1.19.6 Grading Scheme Controller

3.1.19.6.1 Operations

SR - 130 **Must have**
index(): `void`
For each available grading scheme in Grade Calculation, get the information and renders it as JSON
Input `GET` request
Precondition
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition
- A JSON response containing information of each grading scheme is returned

SR - 131 **Should have**
show(): `void`
For a requested grading scheme the information is rendered as JSON
Input `GET` request containing the ID of the grading scheme
Precondition
- The requested grading scheme is in the database or is a hardcoded grading scheme
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition
- A JSON response containing information for the requested grading scheme is returned.

3.1.19.7 Marks Controller

3.1.19.7.1 Operations

SR - 132 **Must have**
import(): `void`
Imports mark information of the course, from Canvas, into the database
Input `POST` request
Precondition

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- A request to the MARK IMPORTER is made

3.1.20 REST API Controller

3.1.20.1 Operations

SR - 133 **Should have**
import(): `void`
Imports the provided content of an assignment, into the database
Input `POST` request
Precondition

- The user that makes the request is authenticated
- The `POST` request contains at least the following parameters:
  - Name
  - Assessment type
  - Mark Type
  - Course ID

Postcondition

- The assignment is stored as an assessment in the Grade Calculation database.

SR - 134 **Could have**
index(): `void`
A JSON containing the the SIS IDs and the corresponding marks, final grades and partial grades for each student within the Course.
Input `GET` request
Precondition

- The `GET` request contains a Course ID.
- The user that makes the request is authenticated.

Postcondition

- A JSON containing for each student in the course (with provided Course ID) the student ID their corresponding marks, partial grades and final grade.

3.1.20.2 Student View Controller

3.1.20.2.1 Operations

SR - 135 **Must have**
index(): `void`
All information needed for the student interface is retrieved and returned as JSON
Input `GET` request
Precondition

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Student
- All the information needed for the student interface has been imported into the database

Postcondition

- A JSON response containing all information needed for the student view of the student that made the request, is returned

---

3.1.20.3 Student Controller

3.1.20.3.1 Operations

SR - 136 **Must have**
import(): `void`
Imports student information of the course, from Canvas, into the database
Input `POST` request
Precondition

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- A request to the STUDENT IMPORTER is made

3.1.20.4 Front-end Controller

3.1.20.4.1 Operations

SR - 137 **Must have**
main_app(): `void`
Checks the role of the current user and shows the front-end view accordingly
Input `GET` request
Precondition

- The user is logged into Canvas
- The user is authorized to access Grade Calculation

Postcondition

- Shows the appropriate front-end view, either the Teacher or Student view.

3.1.20.5 Proxy Controller

3.1.20.5.1 Operations

SR - 138 **Must have**

add_grades(): `void`

Gets a Canvas response and adds the grades to each student listed in the response. It then returns the response added with the grades as JSON

Input `GET` request

Precondition

- The user is authorized to access Grade Calculation
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- returns the grades of all students in the course as JSON

SR - 139 **Must have**

add_marks(): `void`

Gets a Canvas response and adds the marks to each student listed in the response. It then returns the response added with the marks as JSON

Input `GET` request

Precondition

- The user is authorized to access Grade Calculation
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- returns the marks of all students in the course as JSON

3.1.20.6 CSV import controller

SR - 140 **Must have**

import(): `void`

Takes a CSV and passes this on to the function which processes the CSV file

Input `POST` request

Precondition

- The user is authorized to access Grade Calculation
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- The marks read from the CSV are shown
- A confirmation button of the read marks is shown
- Upon confirmation:
  - The function which processes the CSV is called

3.1.20.7 CSV export controller

SR - 141 **Must have**
export(): `void`
takes the request for export and passes this on to the function which exports the CSV file
Input `GET` request
Precondition

- The user is authorized to access Grade Calculation
- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Teacher

Postcondition

- Function which exports the grades of the database to CSV is called

3.1.21 Score Structure Validator

3.1.21.1 Operations

SR - 142 **Must have**
validate_each(): `void`
Checks if the score structure provided is valid
Input The new score structure to be saved
Precondition

- Score structure to be validated is not nil

Postcondition

- An error is returned if the score structure is invalid

3.1.22 Authorization

3.1.22.1 Operations

SR - 143 **Must have**
authorize(): `void`
Checks the role of a user
Precondition

- User has a canvas account

Postcondition

- If the Canvas role is Administrator or Instructor and the permission listed in [1] section 2.4.1 hold then the user is authorized to the Teacher view
- If Canvas role is Learner then the user is authorized to the Student view
- Otherwise an error page is shown.

3.1.23 Routes

3.1.23.1 Attributes

SR - 144 **Must have**
import students route: `route`
<host>/api/v1/students/import
The path that needs to be called to import the students from Canvas to Grade Calculation

SR - 145 **Must have**
import assessments route: `route`
<host>/api/v1/assessments/import
The path that needs to be called to import the assessments from Canvas to Grade Calculation

SR - 146 **Must have**
import marks route: `route`
<host>/api/v1/marks/import
The path that needs to be called to import the marks from Canvas to Grade Calculation

SR - 147 **Must have**
grade definition route: `route`
<host>/api/v1/grade-definitions
<host>/api/v1/grade-definitions/"id"
The path that needs to be called to show the information about the grade-definitions or to show a single grade-definitions

SR - 148 **Must have**
course route: `route`
<host>/api/v1/course
The path that needs to be called to show the information about the course

SR - 149 **Must have**
student-view route: `route`
<host>/api/v1/student
The path that needs to be called to show the information required for the student-view

SR - 150 **Must have**
grading schemes route: `route`
<host>/api/v1/grading-schemes
<host>/api/v1/grading-schemes/"id"

SR - 150 **Must have**
The path that needs to be called to show the information about the grading schemes or to show a single grading scheme

3.1.24 Audit logs

3.1.24.1 Attributes

SR - 151 **Should have**
Action: `enum`
Represents the type of modification that a user of Grade Calculation can do: { change, add, remove, recalculate }

3.1.24.2 Operations

SR - 152 **Should have**
createUserAuditLog(): `void`
This function is called when a single modification to one or more database entries of the Grade Calculation database is done, in order to store which user did what kind of modification. Both the old and new entries are stored in the Audit log database
Input
- action: `Action`
- entries_to_be_changed: `Array database entry`

Precondition
- `entries_to_be_changed` contains at least one database entry
- A user initiates a request to make changes in the database

Postcondition
- `entries_to_be_changed` is copied to a new array called `changed_entries`
- The modification in the Grade Calculation database is done to `changed_entries`
- `entries_to_be_changed` is overwritten by `changed_entries`
- A user audit log containing: "action, User that initiates action, `entry_to_be_changed`, `changed_entry`, Current Time" is stored as immutable entry in the "User Log" table of the Audit Log database.

SR - 153 **Should have**
restoreUserAuditLog(): `void`
This function is called to restore the state of one user audit log
Input
- audit_log: `entry of User Log Table`

Precondition

SR - 153 **Should have**

- The user that initiates the request is logged in into Canvas
- The user that initiates the request is a Administrator

Postcondition

- The modified entries are reverted to the entries before modification in the Grade Calculation database.

3.1.25 Notification

3.1.25.1 Operations

SR - 154 **Should have**

notifyStudent(): `void`

This function is called to notify a student of changes

Input

- $student$: `string`
- $change$: `string`

Precondition

- The change relevant for the student has been made in the Grade Calculation database

Postcondition

- The student gets notified that a change has been made to the final score structure, the grading description, or a grade.

SR - 155 **Should have**

notifyUnmuted(): void

Sends a notification to the student stating that their final grade has been unmuted.

Input

- $student$: `string`

Precondition

- The final grade of $student$ has been unmuted

Postcondition

- The student gets a notification stating their final grade has been unmuted.

Chapter 4

Requirements Traceability Matrix

4.1 URD to SRD

URF	SRF	Priority
URF1.1	**SR**-137, **SR**-147, **SR**-148, **SR**-150	Must have
URF1.2	**SR**-24, **SR**-50 **SR**-66, **SR**-67, **SR**-68, **SR**-69, **SR**-139	Must have
URF1.3	**SR**-24, **SR**-50, **SR**-73, **SR**-74, **SR**-75, **SR**-138, **SR**-147	Must have
URF1.4	**SR**-24, **SR**-50 **SR**-73, **SR**-74, **SR**-75, **SR**-138, **SR**-147	Must have
URF1.5	**SR**-5, **SR**-6, **SR**-8, **SR**-19, **SR**-20, **SR**-53, **SR**-130, **SR**-150	Must have
URF1.6	**SR**-5, **SR**-6, **SR**-8, **SR**-19, **SR**-20, **SR**-53, **SR**-130, **SR**-150	Must have
URF1.7	*Won't be implemented*	Won't have
URF1.8	*Won't be implemented*	Won't have
URF1.9	**SR**-5, **SR**-6, **SR**-23, **SR**-53, **SR**-131, **SR**-123, **SR**-150	Should have
URF1.10	**SR**-6, **SR**-9, **SR**-10, **SR**-11, **SR**-12, **SR**-14, **SR**-15, **SR**-16, **SR**-17, **SR**-18, **SR**-19, **SR**-20, **SR**-21, **SR**-34, **SR**-54, **SR**-142	Must have
URF1.11	**SR**-6, **SR**-9, **SR**-10, **SR**-11, **SR**-12, **SR**-14, **SR**-15, **SR**-16, **SR**-17, **SR**-18, **SR**-19, **SR**-20, **SR**-21, **SR**-34, **SR**-54, **SR**-124, **SR**-142	Should have
URF1.12	**SR**-6, **SR**-10, **SR**-11, **SR**-12, **SR**-14, **SR**-15, **SR**-16, **SR**-17, **SR**-18, **SR**-34, **SR**-54, **SR**-123	Should have
URF1.13	**SR**-102, **SR**-125	Must have
URF1.14	**SR**-22, **SR**-126	Could have
URF1.15	**SR**-3, **SR**-26, **SR**-28, **SR**-74, **SR**-75, **SR**-76, **SR**-121, **SR**-129	Should have
URF1.16	**SR**-3, **SR**-25, **SR**-27, **SR**-74, **SR**-75, **SR**-76, **SR**-121, **SR**-129	Should have
URF1.17	**SR**-3, **SR**-26, **SR**-28, **SR**-74, **SR**-75, **SR**-76, **SR**-121, **SR**-129	Could have
URF1.18	**SR**-3, **SR**-25, **SR**-27, **SR**-74, **SR**-75, **SR**-76, **SR**-121, **SR**-129	Could have
URF1.19	**SR**-10, **SR**-14, **SR**-56, **SR**-57, **SR**-117, **SR**-118, **SR**-119	Must have
URF1.20	**SR**-9, **SR**-19, **SR**-20, **SR**-55	Should have
URF1.21	**SR**-32, **SR**-102, **SR**-125	Could have

URF	SRF	Priority
URF1.22	**SR**-21, **SR**-142	Could have
URF1.23	**SR**-31	Could have
URF1.24	**SR**-31	Could have
URF1.25	**SR**-31	Could have
URF1.26	**SR**-31, **SR**-63, **SR**-114	Should have
URF1.27	**SR**-31, **SR**-64, **SR**-114, **SR**-136	Should have
URF1.28	**SR**-31, **SR**-63, **SR**-114, **SR**-136	Should have
URF1.29	**SR**-31, **SR**-64, **SR**-114, **SR**-136	Should have
URF1.30	**SR**-31, **SR**-63, **SR**-114, **SR**-136	Should have
URF1.31	**SR**-31, **SR**-64, **SR**-114, **SR**-136	Should have
URF1.32	**SR**-18, **SR**-92	Could have
URF1.33	**SR**-18, **SR**-92	Could have
URF1.34	**SR**-143	Must have
URF1.35	*Won't be implemented*	Won't have
URF1.36	**SR**-1, **SR**-2, **SR**-3, **SR**-4, **SR**-65, **SR**-135, **SR**-136, **SR**-137, **SR**-149	Must have
URF1.37	**SR**-1, **SR**-2, **SR**-68 **SR**-69, **SR**-135, **SR**-137, **SR**-149	Must have
URF1.38	**SR**-1, **SR**-3, **SR**-73, **SR**-75, **SR**-127, **SR**-128, **SR**-135, **SR**-137, **SR**-149	Should have
URF1.39	**SR**-1, **SR**-3, **SR**-73, **SR**-75, **SR**-127, **SR**-128, **SR**-135, **SR**-137, **SR**-149	Must have
URF1.40	**SR**-1, **SR**-53, **SR**-135, **SR**-105, **SR**-131, **SR**-137, **SR**-144, **SR**-149	Should have
URF1.41	**SR**-1, **SR**-55, **SR**-105, **SR**-123, **SR**-137, **SR**-135, **SR**-144, **SR**-149	Should have
URF1.42	**SR**-1,**SR**-3, **SR**-77, **SR**-154	Should have
URF1.43	**SR**-1,**SR**-3, **SR**-155,	Should have
URF1.44	*Won't be implemented*	Won't have
URF1.45	**SR**-36, **SR**-49, **SR**-137, **SR**-143	Could have
URF1.46	**SR**-68, **SR**-69, **SR**-73, **SR**-75, **SR**-138, **SR**-139, **SR**-143	Could have
URF1.47	**SR**-36, **SR**-47, **SR**-52, **SR**-54, **SR**-123, **SR**-143,	Could have
URF1.48	**SR**-36, **SR**-38, **SR**-39, **SR**-40, **SR**-41, **SR**-44, **SR**-45, **SR**-46, **SR**-47, **SR**-52, **SR**-54, **SR**-124, **SR**-142	Could have
URF1.49	**SR**-36, **SR**-38, **SR**-39, **SR**-40, **SR**-41, **SR**-44, **SR**-45, **SR**-46, **SR**-47, **SR**-52, **SR**-54, **SR**-124, **SR**-142,	Could have
URF1.50	**SR**-35, **SR**-36, **SR**-53, **SR**-123	Could have
URF1.51	**SR**-35, **SR**-36, **SR**-37, **SR**-38, **SR**-39, **SR**-53, **SR**-124	Could have
URF1.52	**SR**-35, **SR**-36, **SR**-37, **SR**-38, **SR**-39, **SR**-53, **SR**-124	Could have
URF1.53	*Won't be implemented*	Won't have
URF1.54	*Won't be implemented*	Won't have

URF	SRF	Priority
URF2.1	**SR**-58	Must have
URF2.2	**SR**-59	Should have
URF2.3	**SR**-58, **SR**-70	Must have
URF2.4	**SR**-58, **SR**-70	Must have
URF2.5	**SR**-69	Must have
URF2.6	**SR**-69	Must have
URF2.7	**SR**-69	Must have
URF2.8	**SR**-69	Must have
URF2.9	**SR**-2, **SR**-66, **SR**-69, **SR**-119, **SR**-121	Should have
URF2.10	**SR**-30, **SR**-60, **SR**-61, **SR**-62, **SR**-66, **SR**-69, **SR**-72, **SR**-121	Should have
URF2.11	**SR**-79, **SR**-80	Must have
URF2.12	**SR**-79, **SR**-80	Must have
URF2.13	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-150	Must have
URF2.14	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71,**SR**-84	Must have
URF2.15	**SR**-14, **SR**-44,**SR**-84	Must have
URF2.16	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71, **SR**-85	Must have
URF2.17	**SR**-14, **SR**-44, **SR**-85	Must have
URF2.18	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71, **SR**-88	Should have
URF2.19	**SR**-14, **SR**-44, **SR**-88	Should have
URF2.20	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71, **SR**-90	Should have
URF2.21	**SR**-14, **SR**-44,**SR**-90	Should have
URF2.22	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71, **SR**-91	Should have
URF2.23	**SR**-14, **SR**-44, **SR**-91	Should have
URF2.24	**SR**-14, **SR**-10, **SR**-40, **SR**-44, **SR**-71,**SR**-86	Should have
URF2.25	**SR**-14, **SR**-44, **SR**-86	Should have
URF2.26	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-71, **SR**-87	Should have
URF2.27	**SR**-14, **SR**-44, **SR**-87	Should have
URF2.28	**SR**-10, **SR**-13, **SR**-14, **SR**-40, **SR**-43, **SR**-44, **SR**-71, **SR**-81, **SR**-82, **SR**-83, **SR**-91	Could have
URF2.29	**SR**-13, **SR**-14, **SR**-43, **SR**-44, **SR**-81, **SR**-82, **SR**-83, **SR**-91	Could have
URF2.30	**SR**-10, **SR**-13, **SR**-14, **SR**-40, **SR**-43, **SR**-44, **SR**-71, **SR**-81, **SR**-82, **SR**-83, **SR**-90	Could have
URF2.31	**SR**-13, **SR**-14, **SR**-43, **SR**-44, **SR**-81, **SR**-82, **SR**-83, **SR**-90	Could have
URF2.32	**SR**-10, **SR**-13, **SR**-14, **SR**-40, **SR**-43, **SR**-44, **SR**-71, **SR**-81, **SR**-82, **SR**-83, **SR**-89	Could have
URF2.33	**SR**-13, **SR**-14, **SR**-43, **SR**-44, **SR**-81, **SR**-82, **SR**-83, **SR**-89	Could have
URF2.34	**SR**-10, **SR**-13, **SR**-14, **SR**-40, **SR**-43, **SR**-44, **SR**-71, **SR**-81, **SR**-82, **SR**-83, **SR**-88	Could have

URF	SRF	Priority
URF2.35	**SR**-14, **SR**-13, **SR**-43, **SR**-44, **SR**-81,**SR**-82,**SR**-83, **SR**-88	Could have
URF2.36	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-85	Should have
URF2.37	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-85	Could have
URF2.38	**SR**-18, **SR**-48, **SR**-92	Could have
URF2.39	**SR**-10, **SR**-14, **SR**-18, **SR**-40, **SR**-44, **SR**-48, **SR**-71, **SR**-92	Could have
URF2.40	*Won't be implemented*	Won't have
URF2.41	*Won't be implemented*	Won't have
URF2.42	*Won't be implemented*	Won't have
URF2.43	*Won't be implemented*	Won't have
URF2.44	**SR**-79, **SR**-80	Must have
URF2.45	**SR**-79, **SR**-80	Must have
URF2.46	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-150	Must have
URF2.47	**SR**-14, **SR**-44, **SR**-84	Must have
URF2.48	**SR**-14, **SR**-44, **SR**-85	Must have
URF2.49	**SR**-14, **SR**-44, **SR**-88	Should have
URF2.50	**SR**-14, **SR**-44, **SR**-90	Should have
URF2.51	**SR**-14, **SR**-44, **SR**-91	Should have
URF2.52	**SR**-14, **SR**-44, **SR**-86	Should have
URF2.53	**SR**-14, **SR**-44, **SR**-87	Should have
URF2.54	**SR**-13, **SR**-14, **SR**-44, **SR**-43, **SR**-83, **SR**-91	Could have
URF2.55	**SR**-13, **SR**-14, **SR**-44, **SR**-43, **SR**-83, **SR**-90	Could have
URF2.56	**SR**-13, **SR**-43, **SR**-83, **SR**-89	Could have
URF2.57	**SR**-13, **SR**-43, **SR**-83, **SR**-88	Could have
URF2.58	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-85	Should have
URF2.59	**SR**-10, **SR**-14, **SR**-40, **SR**-44, **SR**-85	Could have
URF2.60	**SR**-18, **SR**-48, **SR**-92	Could have
URF2.61	**SR**-18, **SR**-48, **SR**-92	Could have
URF2.62	*Won't be implemented*	Won't have
URF2.63	*Won't be implemented*	Won't have
URF2.64	*Won't be implemented*	Won't have
URF2.65	*Won't be implemented*	Won't have
URF2.66	**SR**-73, **SR**-74, **SR**-75	Must have
URF2.67	**SR**-102,**SR**-148	Should have
URF2.68	**SR**-102,**SR**-148	Should have
URF2.69	**SR**-102, **SR**-139	Should have
URF2.70	**SR**-75	Must have

URF	**SRF**	**Priority**
URF2.71	**SR**-75	Must have
URF2.72	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-94	Could have
URF2.73	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-95	Could have
URF2.74	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-96	Must have
URF2.75	**SR**-8, **SR**-37,**SR**-75, **SR**-93, **SR**-101	Must have
URF2.76	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-97	Could have
URF2.77	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-98	Could have
URF2.78	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-99	Could have
URF2.79	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-100	Could have
URF2.80	*Won't be implemented*	Won't have
URF2.81	**SR**-73, **SR**-74	Must have
URF2.82	**SR**-93, **SR**-102, **SR**-148	Should have
URF2.83	**SR**-93, **SR**-102, **SR**-148	Should have
URF2.84	**SR**-74, **SR**-102, **SR**-138	Should have
URF2.85	**SR**-75	Must have
URF2.86	**SR**-75	Must have
URF2.87	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-94	Must have
URF2.88	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-95	Must have
URF2.89	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-97	Could have
URF2.90	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-98	Could have
URF2.91	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-99	Could have
URF2.92	**SR**-8, **SR**-37, **SR**-75, **SR**-93, **SR**-100	Could have
URF2.93	*Won't be implemented*	Won't have
URF2.94	**SR**-16, **SR**-17, **SR**-47, **SR**-71, **SR**-90	Must have
URF2.95	**SR**-16, **SR**-17, **SR**-47, **SR**-71, **SR**-90	Should have
URF2.96	**SR**-17, **SR**-47, **SR**-71, **SR**-90	Should have
URF2.97	**SR**-16, **SR**-17, **SR**-47, **SR**-71, **SR**-84	Should have
URF2.98	**SR**-14, **SR**-17, **SR**-47, **SR**-69, **SR**-71, **SR**-90	Must have
URF2.99	**SR**-12, **SR**-42, **SR**-59, **SR**-124	Should have
URF2.100	**SR**-78, **SR**-102, **SR**-138	Should have
URF2.101	**SR**-10, **SR**-12, **SR**-40, **SR**-42, **SR**-59, **SR**-102	Should have
URF2.102	**SR**-14, **SR**-16, **SR**-44, **SR**-46, **SR**-59, **SR**-102	Should have
URF2.103	**SR**-10, **SR**-12, **SR**-40, **SR**-42, **SR**-59, **SR**-102	Could have
URF2.104	**SR**-10, **SR**-40, **SR**-42, **SR**-59, **SR**-102	Could have
URF2.105	**SR**-102, **SR**-103	Could have
URF2.106	**SR**-102, **SR**-103	Could have

URF	SRF	Priority
URF2.107	**SR**-102, **SR**-103	Could have
URF3.1	**SR**-104, **SR**-108, **SR**-109, **SR**-112, **SR**-106, **SR**-107, **SR**-113, **SR**-117, **SR**-132, **SR**-145, **SR**-146	Must have
URF3.2	**SR**-104, **SR**-106, **SR**-107, **SR**-108, **SR**-109, **SR**-113, **SR**-117, **SR**-132, **SR**-145, **SR**-146	Must have
URF3.3	**SR**-109	Should have
URF3.4	**SR**-109	Should have
URF3.5	**SR**-109	Should have
URF3.6	**SR**-109	Should have
URF3.7	**SR**-104 , **SR**-106, **SR**-107, **SR**-108, **SR**-109, **SR**-113, **SR**-117, **SR**-132, **SR**-145, **SR**-146	Could have
URF3.8	**SR**-110, **SR**-111, **SR**-120, **SR**-140	Should have
URF3.9	**SR**-110, **SR**-111	Should have
URF3.10	**SR**-110, **SR**-111	Should have
URF3.11	**SR**-111	Could have
URF3.12	**SR**-133	Should have
URF3.13	**SR**-140	Could have
URF3.14	**SR**-140	Could have
URF3.15	**SR**-64, **SR**-73, **SR**-75, **SR**-116, **SR**-141, **SR**-143,	Should have
URF3.16	**SR**-64, **SR**-73, **SR**-75, **SR**-116, **SR**-141, **SR**-143,	Should have
URF3.17	**SR**-64, **SR**-68, **SR**-69, **SR**-116, **SR**-143, **SR**-141	Should have
URF3.18	**SR**-64, **SR**-73, **SR**-75, **SR**-116, **SR**-143, **SR**-141	Could have
URF3.19	**SR**-64, **SR**-73, **SR**-75,**SR**-116, **SR**-141, **SR**-143,	Could have
URF3.20	**SR**-64, **SR**-68, **SR**-69, **SR**-116, **SR**-141, **SR**-143	Could have
URF3.21	**SR**-134	Could have
URF3.22	**SR**-134	Could have
URF3.23	**SR**-134	Could have
URF3.24	**SR**-116	Should have
URF3.25	**SR**-110, **SR**-111	Should have
URF3.26	*Won't be implemented*	Won't have
URF3.27	*Won't be implemented*	Won't have
URF4.1	**SR**-151, **SR**-152	Should have
URF4.2	**SR**-124, **SR**-151, **SR**-152,	Should have
URF4.3	**SR**-124, **SR**-151, **SR**-152,	Should have
URF4.4	**SR**-152	Should have
URF4.5	**SR**-152	Should have
URF4.6	**SR**-152	Should have
URF4.7	**SR**-152	Should have

URF	**SRF**	**Priority**
URF4.8	*Won't be implemented*	Won't have
URF4.9	**SR**-153	Won't have
URF4.10	**SR**-152	Should have
URF4.11	**SR**-152	Should have
URF4.12	**SR**-122, **SR**-33	Should have
URF4.13	**SR**-51, **SR**-122	Could have
URF4.14	*Won't be implemented*	Won't have
URF4.15	*Won't be implemented*	Won't have
URF4.16	*Won't be implemented*	Won't have
URF5.1	**SR**-110, **SR**-111, **SR**-132	Should have
URF5.2	*Won't be implemented*	Won't have
URF5.3	*Won't be implemented*	Won't have
URF5.4	*Won't be implemented*	Won't have
URF5.5	*Won't be implemented*	Won't have
URF5.6	*Won't be implemented*	Won't have
URF5.7	*Won't be implemented*	Won't have
URF5.8	*Won't be implemented*	Won't have
URF5.9	*Won't be implemented*	Won't have
URF5.10	*Won't be implemented*	Won't have
URF5.11	*Won't be implemented*	Won't have
URF5.12	**SR**-7, **SR**-124, **SR**-142	Could have
URF5.13	*Won't be implemented*	Won't have
URF6.1	*This URF has been moved to URC3.15*	Must have
URF6.2	*This URF has been moved to URC3.16*	Must have
URF6.3	**SR**-7, **SR**-124	Could have
URF6.4	*Won't be implemented*	Won't have
URF6.5	*Won't be implemented*	Won't have
URF6.6	*Won't be implemented*	Won't have
URF6.7	*Won't be implemented*	Won't have
URF6.8	*Won't be implemented*	Won't have
URF6.9	*Won't be implemented*	Won't have

4.2 SRD to URD

SRF	URF	Priority
SR-1	URF1.36, URF1.37, URF1.38, URF1.39, URF1.40, URF1.41, URf1.42, URF1.43	Must have
SR-2	URF1.36, URF1.37, URF2.9	Must have
SR-3	URF1.15, URF1.16, URF1.17, URF1.18, URF1.36, URF1.38, URF1.39, URF1.42, URF1.43	Must have
SR-4	URF1.36	Must have
SR-5	URF1.5, URF1.6, URF1.9	Must have
SR-6	URF1.5, URF1.6, URF1.9, URF1.10, URF1.11, URF1.12	Must have
SR-7	URF5.12, URF6.3	Could have
SR-8	URF1.5, URF1.6, URF2.13, URF2.46, URF2.72, URF2.73, URF2.74, URF2.75, URF2.76, URF2.77, URf2.78, URf2.79, URF2.87, URF2.88, URF2.89, URF2.90, URF2.91, URF2.92	Must have
SR-9	URF1.10, URF1.11, URF1.20	Should have
SR-10	URF1.10, URF1.11, URF1.12, URF1.19, URF2.14, URF2.16, URF2.18, URF2.20, URF2.22, URF2.24, URF2.26, URF2.28, URF2.30, URF2.32, URF2.34, URF2.36, URF2.37, URF2.39, URF2.58, URF2.59, URF2.101, URF2.103, URF2.104	Must have
SR-11	URF1.10, URF1.11, URF1.12	Should have
SR-12	URF1.10, URF1.11, URF1.12, URF2.99, URF2.101, URF2.103, URF2.104	Should have
SR-13	URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.54, URF2.55, URF2.56, URF2.57	Could have
SR-14	URF1.10, URF1.11, URF1.12, URF1.19, URF2.14, URF2.15, URF2.16, URF2.17, URF2.18, URF2.19, URF2.20, URF2.21, URF2.22, URF2.23, URF2.24, URF2.25, URF2.26, URF2.27, URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.36, URF2.37, URF2.39, URF2.47, URF2.48, URF2.49, URF2.50, URF2.51, URF2.52, URF2.53, URF2.54, URF2.55, URF2.58, URF2.59, URF2.98, URF2.102	Must have
SR-15	URF1.10, URF1.11, URF1.12	Should have
SR-16	URF1.10, URF1.11, URF1.12, URF2.94, URF2.95, URF2.97, URF2.102	Should have
SR-17	URF1.10, URF1.11, URF1.12, URF2.94, URF2.95, URF2.96, URF2.97, URF2.98	Must have
SR-18	URF1.10, URF1.11, URF1.12, URF1.32, URF1.33, URF2.38, URF2.39, URF2.60, URF2.61	Could have
SR-19	URF1.5, URF1.6, URF1.10, URF1.11, URF1.20	Must have
SR-20	URF1.5, URF1.6, URF1.10, URF1.11, URF1.20	Must have
SR-21	URF1.10, URF1.11, URF1.22	Could have
SR-22	URF1.14	Could have
SR-23	URF1.9	Must have

SRF	URF	Priority
SR-24	URF1.2, URF1.3, URF1.4	Must have
SR-25	URF1.16, URF1.18	Should have
SR-26	URF1.15, URF1.17	Should have
SR-27	URF1.16, URF1.18	Should have
SR-28	URF1.15, URF1.17	Should have
SR-29	URF1.23, URF1.24, URF1.25	Could have
SR-30	URF2.10	Should have
SR-31	URF1.26, URF1.27, URF1.28, URF1.29, URF1.30, URF1.31	Should have
SR-32	URF1.21	Could have
SR-33	URF4.12	Should have
SR-34	URF1.10, URF.1.11, URF1.12	
SR-35	URF1.50, URF1.51,URF1.52	Could have
SR-36	URF1.45, URF1.47, URF1.48, URF1.49, URF1.50, URF1.51, URF1.52	Could have
SR-37	URF1.51, URF1.52, URF2.13, URF2.46, URF2.72, URF2.73, URF2.74, URF2.75, URF2.76, URF2.77, URF2.78, URF2.79, URF2.87, URF2.88, URF2.89, URF2.90, URF2.91, URF2.92	Could have
SR-38	URF1.48, URF1.49, URF1.51, URF1.52	Could have
SR-39	URF1.48, URF1.49, URF1.51, URF1.52	Could have
SR-40	URF1.48, URF1.49, URF2.14, URF2.16, URF2.18, URF2.20, URF2.22, URF2.24, URF2.26, URF2.28, URF2.30, URF2.32, URF2.34, URF2.36, URF2.37, URF2.39, URF2.58, URF2.59, URF2.101, URF2.103, URF2.104	Could have
SR-41	URF1.48, URF1.49	Could have
SR-42	URF1.48, URF1.49, URF2.99, URF2.101, URF2.103, URF2.104	Could have
SR-43	URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.54, URF2.55, URF2.56, URF2.57	Could have
SR-44	URF1.48, URF1.49, URF2.14, URF2.15, URF2.16, URF2.17, URF2.18,URF2.19, URF2.20, URF2.21, URF2.22, URF2.23, URF2.24, URF2.25, URF2.26, URF2.27, URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.36, URF2.37, URF2.39, URF2.47, URF2.48, URF2.49, URF2.50, URF2.51, URF2.52, URF2.53, URF2.54, URF2.55, URF2.58, URF2.59, URF2.102	Could have
SR-45	URF1.48, URF1.49	Could have
SR-46	URF1.48, URF1.49, URF2.102	Could have
SR-47	URF1.47, URF1.48, URF1.49, URF2.94, URF2.95, URF2.96, URF2.97, URF2.98	Could have
SR-48	URF2.38, URF2.39, URF2.60, URF2.61	Could have
SR-49	URF1.45	Could have
SR-50	URF1.2, URF1.3, URF1.4	Could have

SRF	URF	Priority
SR-51	URF4.13	Could have
SR-52	URF1.47, URF1.48, URF1.49	Could have
SR-53	URF1.5, URF1.6, URF1.9, URF1.40, URF1.50, URF1.51, URF1.52	Must have
SR-54	URF1.10, URF1.11, URF1.12, URF1.47, URF1.48, URF1.49	Must have
SR-55	URF1.20, URF1.41	Should have
SR-56	URF1.19	Must have
SR-57	URF1.19	Must have
SR-58	URF2.1, URF2.3, URF2.4	Must have
SR-59	URF2.2, URF2.99, URF2.101, URF2.102, URF2.103, URF2.104	Must have
SR-60	URF2.10	Should have
SR-61	URF2.10	Should have
SR-62	URF2.10	Should have
SR-63	URF1.26, URF1.28, URF1.30	Should have
SR-64	URF1.27, URF1.29, URF1.31, URF3.15, URF3.16, URF3.17, URF3.18, URF3.19, URF3.20	Should have
SR-65	URF1.36	Must have
SR-66	URF1.2, URF2.9, URF2.10	Must have
SR-67	URF1.2	Must have
SR-69	URF1.2, URF1.37, URF1.46, URF2.5, URF2.6, URF2.7, URF2.8, URF2.9, URF2.10, URF2.98, URF3.17, URF3.20	Must have
SR-70	URF2.3, URF2.4	Must have
SR-71	URF2.14, URF2.16, URF2.18, URF2.20, URF2.22, URF2.24, URF2.26, URF2.28, URF2.30, URF2.32;URF 2.34, URF2.39, URF2.94, URF2.95, URF2.96, URF2.97, URF2.98	Must have
SR-72	URF2.10	Should have
SR-73	URF1.3, URF1.4, URF1.38, URF1.39, URF1.46, URF2.66, URF2.81, URF3.15, URF3.16, URF3.18, URF3.19	Must have
SR-75	URF1.3, URF1.4, URF1.15, URF1.16, URF1.17, URF1.18, URF1.38, URF1.39, URF1.46, URF2.13, URF2.46, URF2.66, URF2.70, URF2.71, URF2.72,URF2.73, URF2.74, URF2.75, URF2.76, URF2.77, URF2.78, URF2.79, URF2.85, URF2.86, URF2.87, URF2.88, URF2.89, URF2.90, URF2.91, URF2.92, URF3.15, URF3.16, URF3.18, URF3.19	Must have
SR-76	URF1.15, URF1.16, URF1.17, URF1.18	Should have
SR-77	URF1.42	Should have
SR-78	URF2.100	Should have
SR-79	URF2.11, URF2.12, URF2.44, URF2.45	Must have
SR-80	URF2.11, URF2.12, URF2.44, URF2.45	Must have
SR-81	URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.54, URF2.55, URF2.56, URF2.57	Could have

SRF	URF	Priority
SR-82	URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.54, URF2.55, URF2.56, URF2.57	Could have
SR-83	URF2.28, URF2.29, URF2.30, URF2.31, URF2.32, URF2.33, URF2.34, URF2.35, URF2.54, URF2.55, URF2.56, URF2.57	Could have
SR-84	URF2.14, URF2.15 URF2.47, URF2.97	Must have
SR-85	URF2.16, URF2.17, URF2.36, URF2.37, URF2.48, URF2.58, URF2.59	Must have
SR-86	URF2.24, URF 2.25, URF2.52	Should have
SR-87	URF2.26, URF2.27, URF2.53	Should have
SR-88	URF2.18, URF2.19, URF2.34, URF2.35, URF2.49, URF 2.57	Should have
SR-89	URF2.32, URF3.33, URF2.56	Should have
SR-90	URF2.20, URF2.21, URF2.30, URF2.31, URF2.50, URF2.55, URF2.94, URF2.95, URF2.96, URF2.98	Should have
SR-91	URF2.22, URF2.23, URF2.28, URF2.29, URF2.51, URF2.54	Should have
SR-92	URF1.32, URF1.33, URF2.38, URF2.39, URF2.60, URF2.61	Could have
SR-93	URF2.13, URF2.46, URF2.72, URF2.73, URF2.74, URF2.75, URF2.76, URF2.77, URF2.78, URF2.79, URF2.82, URF2.83, URF2.87, URF2.88, URF2.89, URF2.90, URF2.91, URF2.92	Must have
SR-94	URF2.72, URF2.87	Must have
SR-95	URF2.73, URF2.88	Must have
SR-96	URF2.74	Must have
SR-97	URF2.76, URF2.89	Must have
SR-98	URF2.77, URF2.90	Must have
SR-99	URF2.78, URF2.91	Must have
SR-100	URF2.79, URF2.92	Must have
SR-101	URF2.75	Must have
SR-102	URF1.13, URF1.21, URF2.67, URF2.68, URF2.69, URF2.82, URF2.83, URF2.84, URF2.100, URF2.101, URF2.102, URF2.103, URF2.104, URF2.105, URF2.106, URF2.107	Must have
SR-103	URF2.105, URF2.106, URF2.107	Could have
SR-104	URF3.1, URF3.2, URF3.7	Must have
SR-105	URF1.40, URF1.41	Must have
SR-106	URF3.1, URF3.2, URF3.7	Must have
SR-107	URF3.1, URF3.2, URF3.7	Must have
SR-108	URF3.1, URF3.2, URF3.7	Must have
SR-109	URF3.1, URF3.2, URF3.3, URF3.4, URF 3.5, URF3.6, URF3.7	Must have
SR-110	URF3.8, URF3.9, URF3.10, URF3.25, URF5.1	Should have
SR-111	URF3.8, URF3.9, URF3.10, URF3.11, URF3.25, URF5.1	Should have
SR-112	URF3.1	Must have

SRF	URF	Priority
SR-113	URF3.1, URF3.2, URF3.7	Must have
SR-114	URF1.26, URF1.27, URF1.28, URF1.29, URF1.30, URF1.31	Must have
SR-115	URF1.26, URF1.27, URF1.28, URF1.29, URF1.30, URF1.31	Must have
SR-116	URF3.15, URF3.16, URF3.17, URF3.24	Should have
SR-117	URF1.19, URF3.1, URF3.2, URF3.7	Must have
SR-118	URF1.19	Must have
SR-119	URF1.19, URF2.9	Must have
SR-120	URF3.8, URF5.2	Should have
SR-121	URF1.15, URF1.16, URF1.17, URF1.18, URF2.9, URF2.10	Must have
SR-122	URF4.12, URF4.13	Could have
SR-123	URF1.9, URF1.12, URF1.41, URF1.47, URF1.50	Must have
SR-124	URF1.11, URF1.48, URF1.49, URF1.50, URF1.52, URF2.99, URF4.2, URF4.3, URF5.12, URF6.3	Must have
SR-125	URF1.13, URF1.21	Must have
SR-126	URF1.14	Could have
SR-127	URF1.38, URF1.39	Must have
SR-128	URF1.38, URF1.39	Must have
SR-129	URF1.15, URF1.16, URF1.17, URF1.18	Must have
SR-130	URF1.5, URF1.6	Must have
SR-131	URF1.9, URF1.40	Should have
SR-132	URF3.1, URF3.2, URF3.7, URF5.1	Must have
SR-133	URF3.12	Should have
SR-134	URF3.21, URF3.22, URF3.23	Could have
SR-135	URF1.36, URF1.37, URF1.38, URF1.39, URF1.40, URF1.41	Must have
SR-136	URF1.26, URF1.27, URF.128, URF1.29, URF1.30, URF1.31, URF1.36	Must have
SR-137	URF1.1, URF1.36, URF1.37, URF1.38, URF1.39, URF1.41, URF1.45	Must have
SR-138	URF1.3, URF1.4, URF1.46, URF2.84, URF2.100	Must have
SR-139	URF1.2, URF1.46, URF2.69	Must have
SR-140	URF3.8, URF3.13, URF3.14	Must have
SR-141	URF3.15, URF3.16, URF3.17, URF3.18, URF3.19, URF3.20	Must have
SR-142	URF1.10, URF1.11, URF1.22, URF1.48, URF1.49	Must have
SR-143	URF1.34, URF1.45, URF1.46, URF1.47, URF3.15, URF3.16, URF3.17, URF3.18, URF3.19, URF3.20	Must have
SR-144	URF1.40, URF1.41	Must have
SR-145	URF3.1, URF3.2, URF3.7	Must have
SR-146	URF3.1, URF3.2, URF3.7	Must have

SRF	URF	Priority
SR-147	URF1.1, URF1.3, URF1.4	Must have
SR-148	URF1.1, URF2.67, URF2.68, URF2.82, URF2.83	Must have
SR-149	URF1.36, URF1.37, URF1.38, URF1.39, URF1.40, URF1.41	Must have
SR-150	URF 1.1, URF1.5, URF1.6, URF1.9, URF2.13, URF2.46, URF4.10, URF4.11	Must have
SR-151	URF4.1, URF4.2, URF4.3	Should have
SR-152	URF4.1, URF4.2, URF4.3, URF4.4, URF4.5, URF4.6, URF4.7	Should have
SR-153	URF4.9	Should have
SR-154	URF1.42, URF1.43	Should have
SR-155	URF1.43	

Appendix A

User Interface - Student

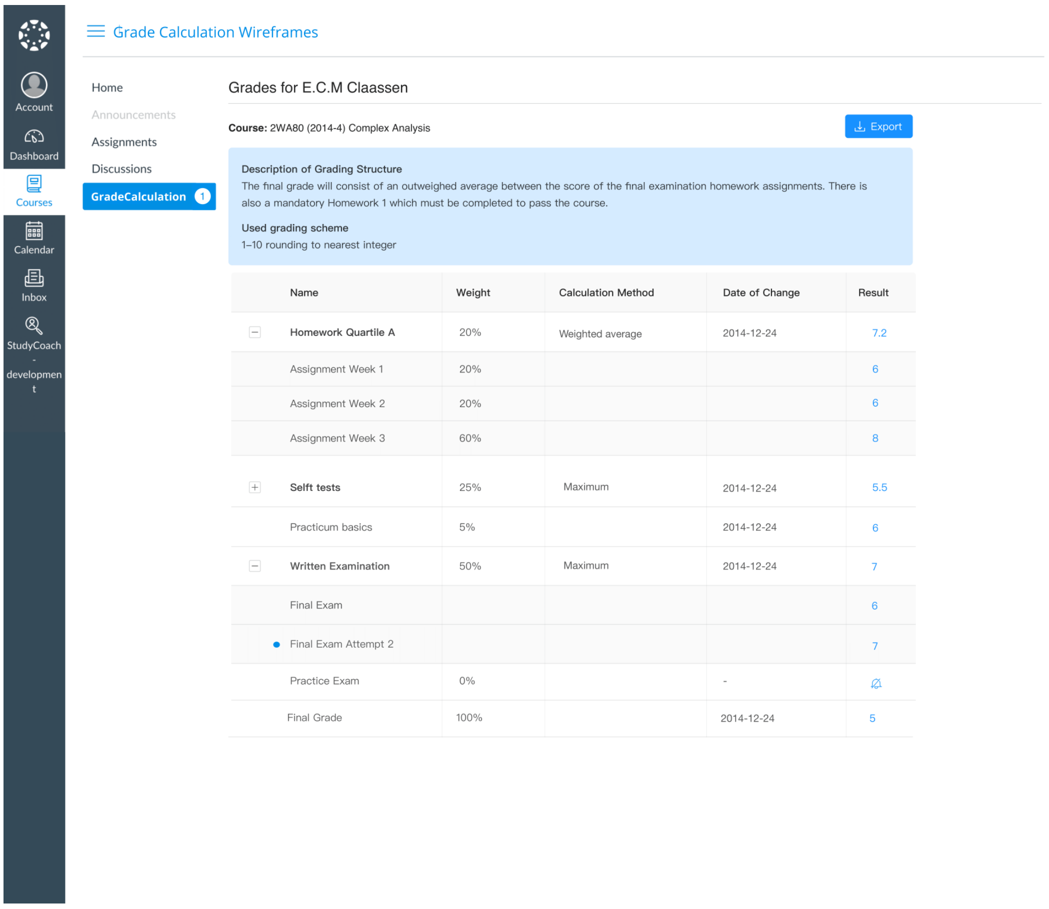
Given the fact that Grade Calculation is not a separate application but an LTI plugin embedded into Canvas, the designs are confined to one page with multiple tabs. Furthermore, Canvas handles all authentication and login matters. Thus, these aspects are not considered in the designs. Transitions between the different views can be found in Appendix C.
It is also important to note that in the context of the user interface, "Grading Structure" refers to both the grading scheme and the final score structure together. This term is not defined in the definitions due to the fact that it is only used to enhance the user experience. Within the documentation produced for Grade Calculation by Great Graders, the grading scheme and the final score structure will always be separate entities.

A.1 Student View

When a student logs into Canvas and enters a course page, then along with the provided tabs of Canvas, the tab called Grade Calculation is also shown for courses for which have Grade Calculation enabled. Selecting this tab provides the student access to the overview of their individual grades for the specific course. At the top of the page the name of the student is shown. Below that, the name of the course along with when the course is being taught, are displayed.

After that, a highlighted box is shown. This box contains the description of the "Grading Structure" as well as the used grading scheme. In more detail, in the subsection called "Description of the Grading Structure", the student can find an explanation of how the final grade is calculated and which components it consists of. Moreover, in the section "Used grading scheme", the way the final grade is represented is shown.

The main focus of the Student Interface is the table in the center. For every assessment, the names of the assessments, their weight, the calculation method, the date of change, and finally the result the student attained for this assessment are shown. More precisely, in this table the student can find information not only for individual assessments, but also for sets of assessments that are defined in the final score structure. In addition, since the set of assessments are collapsible menus, they give the option to either show only information for the set as a whole or, if expanded, to also show information for each of the individual assessments it contains.



Grades for E.C.M Claassen

Course: 2WA80 (2014-4) Complex Analysis

Description of Grading Structure
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Used grading scheme
1–10 rounding to nearest integer

Name	Weight	Calculation Method	Date of Change	Result
⊟ **Homework Quartile A**	20%	Weighted average	2014-12-24	7.2
Assignment Week 1	20%			6
Assignment Week 2	20%			6
Assignment Week 3	60%			8
⊞ **Selft tests**	25%	Maximum	2014-12-24	5.5
Practicum basics	5%		2014-12-24	6
⊟ **Written Examination**	50%	Maximum	2014-12-24	7
Final Exam				6
● Final Exam Attempt 2				7
Practice Exam	0%		-	🔕
Final Grade	100%		2014-12-24	5

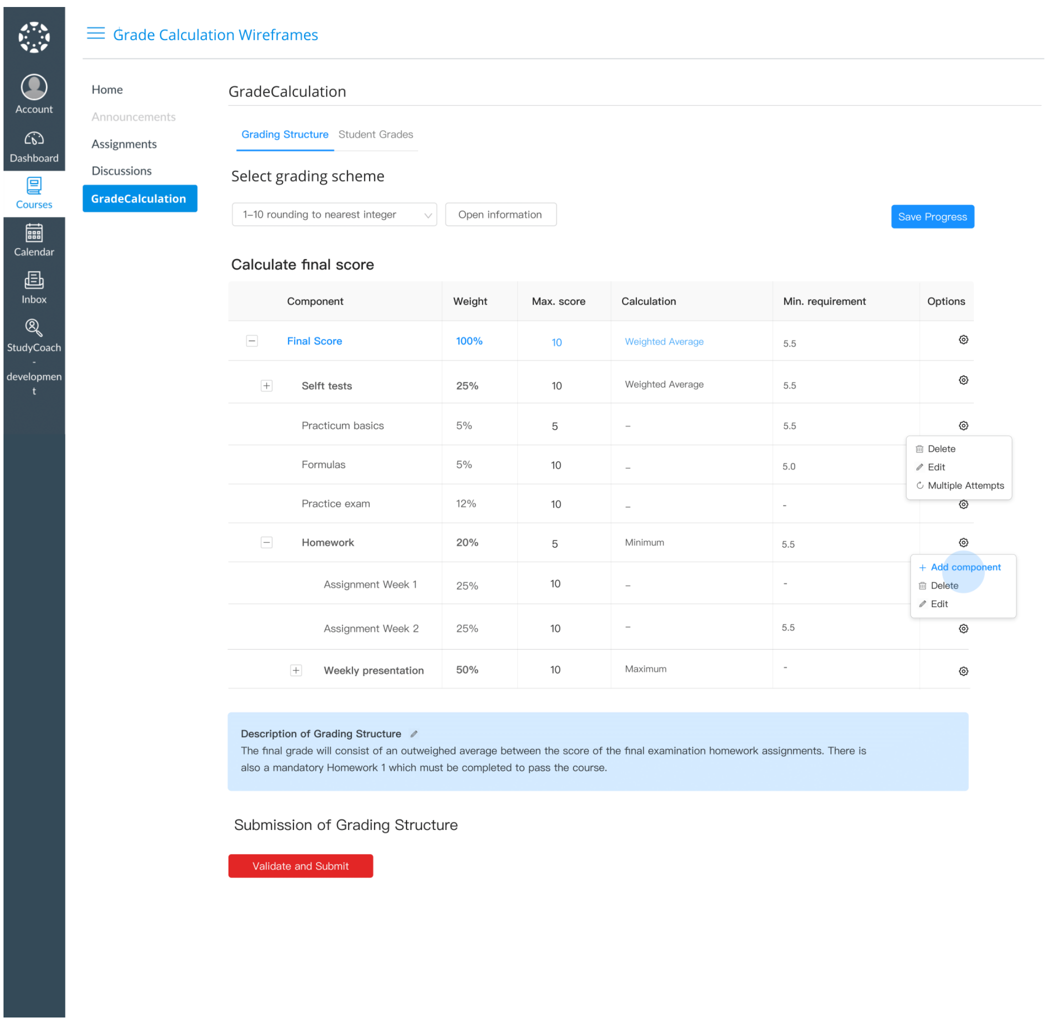
Figure A.1: Student Interface: Overview page

Finally, in the top right corner of the Student View there is an Export button. Clicking this button allows the student to obtain a CSV file containing their marks, partial grades, and final grades as shown in the table.

Appendix B

User Interface - Teacher

B.1 “Grading Structure" view



GradeCalculation

Select grading scheme

1–10 rounding to nearest integer

Calculate final score

Component	Weight	Max. score	Calculation	Min. requirement	Options
Final Score	100%	10	Weighted Average	5.5	
Selft tests	25%	10	Weighted Average	5.5	
Practicum basics	5%	5	–	5.5	
Formulas	5%	10	–	5.0	
Practice exam	12%	10	–	-	
Homework	20%	5	Minimum	5.5	
Assignment Week 1	25%	10	–	-	
Assignment Week 2	25%	10	–	5.5	
Weekly presentation	50%	10	Maximum	-	

Description of Grading Structure
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Submission of Grading Structure

Figure B.1: “Grading Structure" view: Overview Page

When a teacher with the permissions defined in Section 2.4.1 of the URD [1] logs onto Canvas and enters a course page, then along with the provided tabs of Canvas, the tab called Grade Calculation is also shown. Selecting this tab provides the teacher access to the Teacher interface for the specific course. If a teacher does not have the permission defined in the Section 2.4.1 of the URD, an error page is shown. From now on, it is assumed that the teacher has the necessary permission.

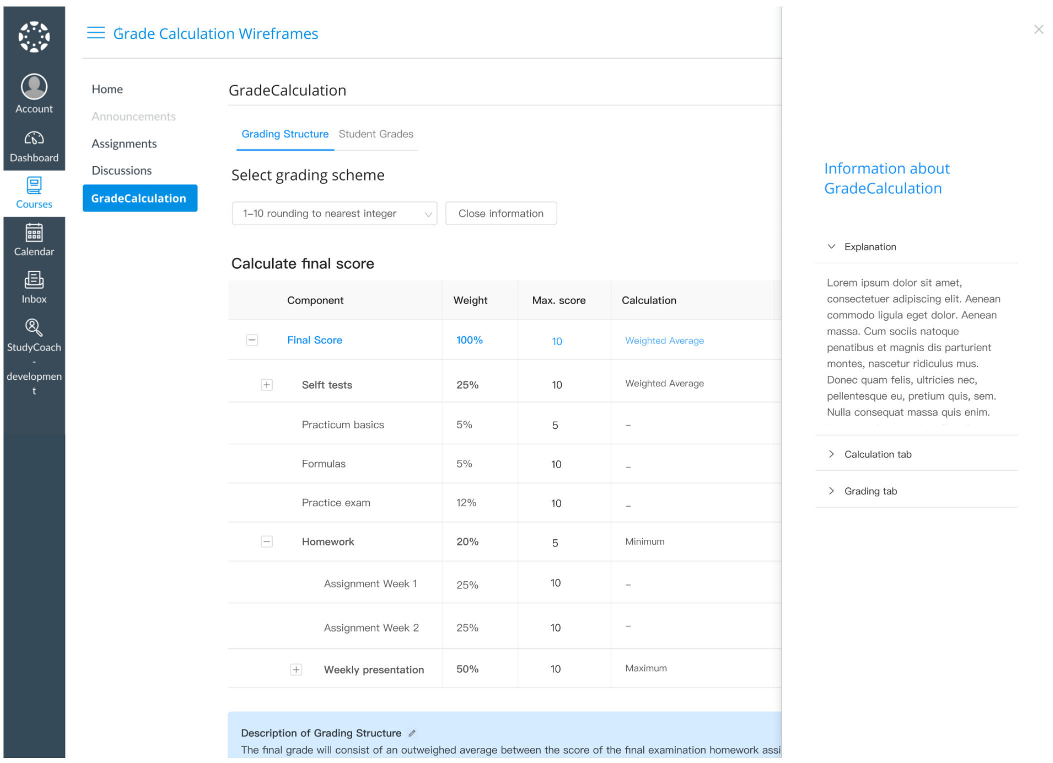
After obtaining access to the Teacher Interface, the “Grading Structure" tab is shown to the teacher. At the top of the page, the teacher can select, via a drop-down menu, the grading scheme of their choosing. For example, in the picture above, the scheme selected is the “1-10 rounding to nearest integer", which is the default grading scheme selected by Grade Calculation. Two more items that can also be found at the top of the page are the “Open Information" and “Save Progress" buttons. By clicking the “Open Information" button a information page is displayed, appearing from the right-side of the screen. This page contains information to help the teacher interact with Grade Calculation. Secondly, by clicking on the “Save progress" button, the teacher can save their progress when creating the grading structure.

At the center of the screen there is a table that displays the components of the final score structure and their definitions. The teacher or administrator can see the different sets of assessments as well as the individual assessments that the sets are composed of. This table contains six columns, namely “Component", “Weight", “Max. score", “Calculation", “Min. requirement", and “Options". The “Component" column holds the name of the assessment or assessment set. The “Weight" column contains a percentage which denotes the extent to which the assessment or assessment set contributes to the final score. These weights are only shown when the calculation method is “Weighted average". In this case, it is imperative to know the extent to which the children of an assessment set influence the score of the assessment set. For all other calculation methods, the weight is not necessary in order to gain a full understanding of the composition of the assessment set. For example, if the calculation method is “Unweighted average" then trivially all children of the assessment set will have an equal weight which sums up to 100.

In third place, the column “Max. score" holds the maximum score that one could attain for this assessment. For example, if the assessment has a grading type of “percentage", then the maximum score would be 100. Then, the “Calculation" column is used to display the calculation method for determining the score for the assessment set. As individual assessments cannot have a calculation method on their own, this column is always left empty for them. Furthermore, the “Min. requirement" column displays a value, within the calculation range of the assessment. This value represents the minimum score that the student must attain in order to pass the course. If no minimum requirement is necessary for assessment, this column is left empty. Lastly, the “Options" column contains a button, which when selected, shows a menu. Both an individual assessment and a set of assessments have this option button, however, the contents in the menu differ between the two types. For an assessment sets, the menu allows one to add a new component, to delete the whole set (including its children), and to edit the information of the assessment set in the other columns. For an individual assessments, the available options are to edit the the information of the assessment in the other columns, to delete the assessment, and to create multiple attempts. Please see Appendix B.1.5 for further clarification of the create multiple attempts option.

Below the table, there is the “Description of the Grading Structure". By clicking on the edit button next to the title, the teacher can describe how the final grade will be calculated. The information entered will also be shown to the students in the “Student Interface". Finally, at the bottom of the page there is a button called “Validate and Submit". If the grading structure is incomplete, clicking this button will produce a warning, mentioning that the grading structure is invalid and what needs to be changed. However, if the final score structure and grading scheme are valid, clicking this button will make the changes made in the grading structure official by saving it to the database.

B.1.1 Information Button



Component	Weight	Max. score	Calculation
Final Score	100%	10	Weighted Average
Selft tests	25%	10	Weighted Average
Practicum basics	5%	5	–
Formulas	5%	10	–
Practice exam	12%	10	–
Homework	20%	5	Minimum
Assignment Week 1	25%	10	–
Assignment Week 2	25%	10	–
Weekly presentation	50%	10	Maximum

Description of Grading Structure
The final grade will consist of an outweighed average between the score of the final examination homework assi

Information about GradeCalculation

Explanation

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim.

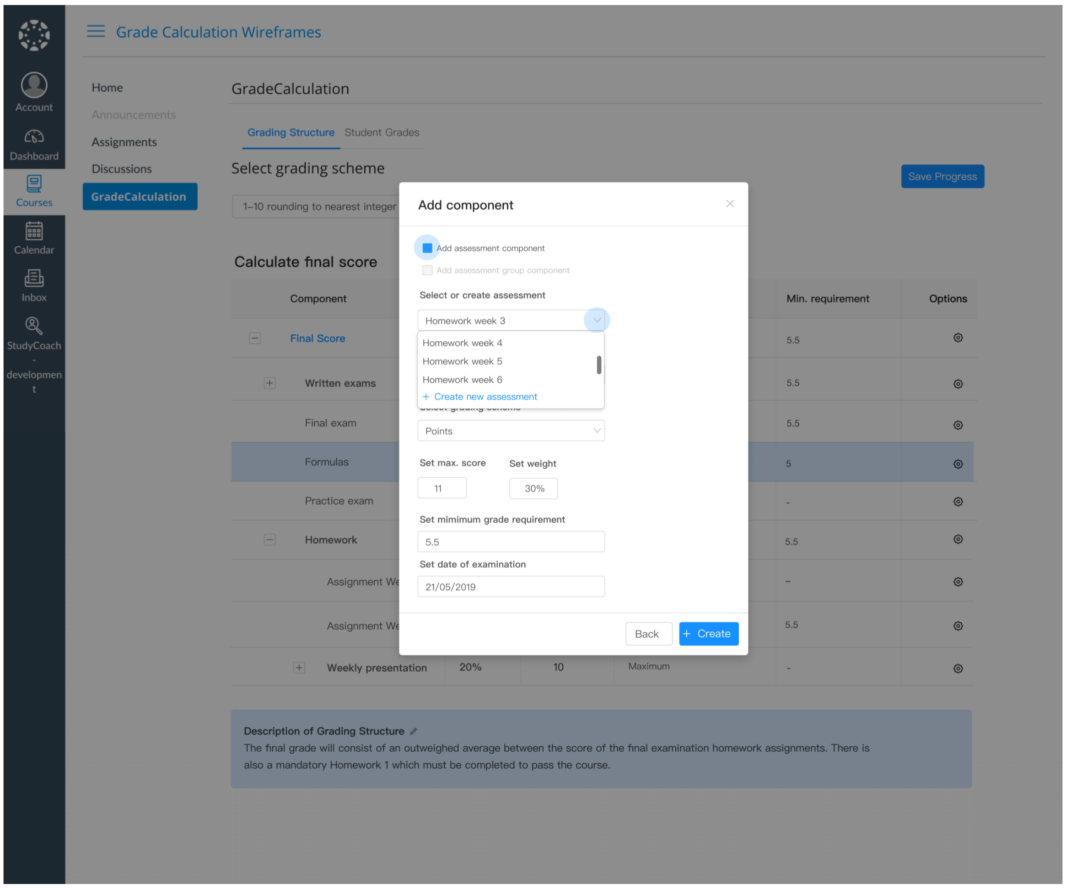
Calculation tab

Grading tab

Figure B.2: “Grading Structure" view: Open Information

Clicking the “Open Information" button at the top of the page will make an information page appear from the right side of the screen. In that page, an explanation of “Grading Structure" view and the “Student Grades" view will be given. For example, the teacher will be able to find information along with tips and hints on how to set up the grading structure for their course.

B.1.2 Create Assessment



GradeCalculation

Select grading scheme: 1–10 rounding to nearest integer

Calculate final score

Component				Min. requirement	Options
Final Score				5.5	⚙
Written exams				5.5	⚙
Final exam				5.5	⚙
Formulas				5	⚙
Practice exam				-	⚙
Homework				5.5	⚙
Assignment We				–	⚙
Assignment We				5.5	⚙
Weekly presentation	20%	10	Maximum	-	⚙

Add component

[✓ Add assessment component]
Add assessment group component

Select or create assessment: Homework week 3
Homework week 4
Homework week 5
Homework week 6
+ Create new assessment

Select grading scheme: Points

Set max. score: 11
Set weight: 30%

Set mimimum grade requirement: 5.5

Set date of examination: 21/05/2019

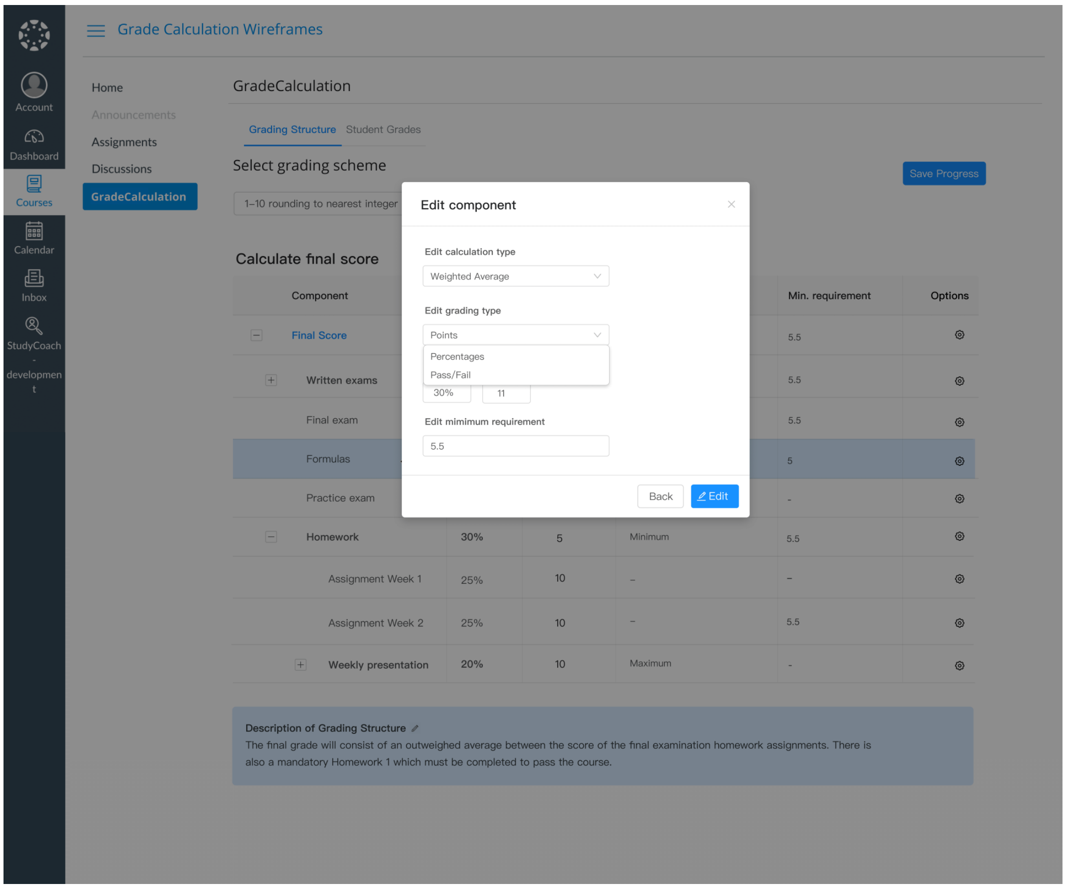
Description of Grading Structure
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Figure B.3: “Grading Structure" view: Create Assessment

This pop-up shows up when the “Add Component" item is selected in the option menu of a set of assessments. At the top of this pop-up, the teacher first has to select whether the component to be added is going to be an individual assessment or a set of assessments. The rest of the choices depend on the initial selection from the teacher. In the picture above an individual assessment was selected as the component to be added. The teacher then has to either select one of the already defined Canvas assessments for this course, or create a new assessment. If the teacher selects a Canvas assessment then they would only need to input the weight of the assessment, should it need one. Otherwise, the teacher would also have to define how the mark is represented as well as set the maximum score attainable for this assessment and input its due date. When all required inputs are filled, the teacher can finalize the process by clicking the button “Create" to complete the creation process of an individual assessment.

B.1.3 Edit Assessment

In case a change has to be made, the teacher can select the “Edit" option inside the option menu for each assessment or assessment set. As a result of selecting the “Edit" option, a new pop-up appears. Its contents are similar to the “Add Assessment" pop-up with the main omission being the selection of an assessment or assessment set.



GradeCalculation

Select grading scheme

1–10 rounding to nearest integer

Calculate final score

Component				Min. requirement	Options
Final Score				5.5	⚙
Written exams				5.5	⚙
Final exam				5.5	⚙
Formulas				5	⚙
Practice exam				-	⚙
Homework	30%	5	Minimum	5.5	⚙
Assignment Week 1	25%	10	–	–	⚙
Assignment Week 2	25%	10	–	5.5	⚙
Weekly presentation	20%	10	Maximum	-	⚙

Edit component

Edit calculation type: Weighted Average

Edit grading type: Points

Percentages

Pass/Fail

30% 11

Edit mimimum requirement: 5.5

Back Edit

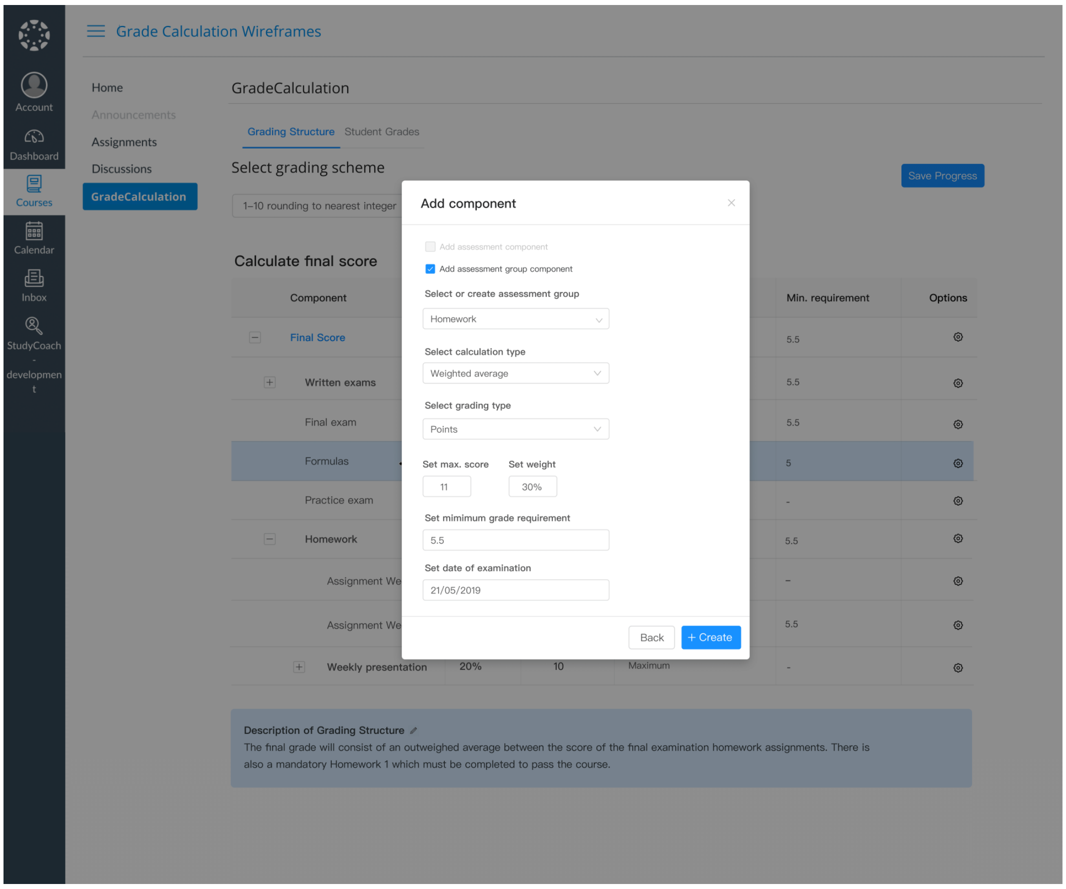
Description of Grading Structure

The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Figure B.4: “Grading Structure" view: Edit Assessment

In the case that the “Edit" option is selected for an individual assessment there are two different cases to consider. If the individual assessment is a Canvas defined assessment, only the weight can be changed. However, if the individual assessment was created by the teacher, its name, weight, and representation of the mark along with the maximum score and the minimum requirement for this assessment can be changed. In the case that the “Edit" option is selected for a set of assessments then its name, weight, calculation method, and representation of the mark along with the maximum score and the minimum requirement for this assessment set can be edited.

B.1.4 Create Assessment Set



GradeCalculation

Select grading scheme: 1–10 rounding to nearest integer

Calculate final score

Component				Min. requirement	Options
Final Score				5.5	⚙
Written exams				5.5	⚙
Final exam				5.5	⚙
Formulas				5	⚙
Practice exam				-	⚙
Homework				5.5	⚙
Assignment We				-	⚙
Assignment We				5.5	⚙
Weekly presentation	20%	10	Maximum	-	⚙

Add component

- [ ] Add assessment component
- [x] Add assessment group component

Select or create assessment group: Homework

Select calculation type: Weighted average

Select grading type: Points

Set max. score: 11

Set weight: 30%

Set mimimum grade requirement: 5.5

Set date of examination: 21/05/2019

Description of Grading Structure

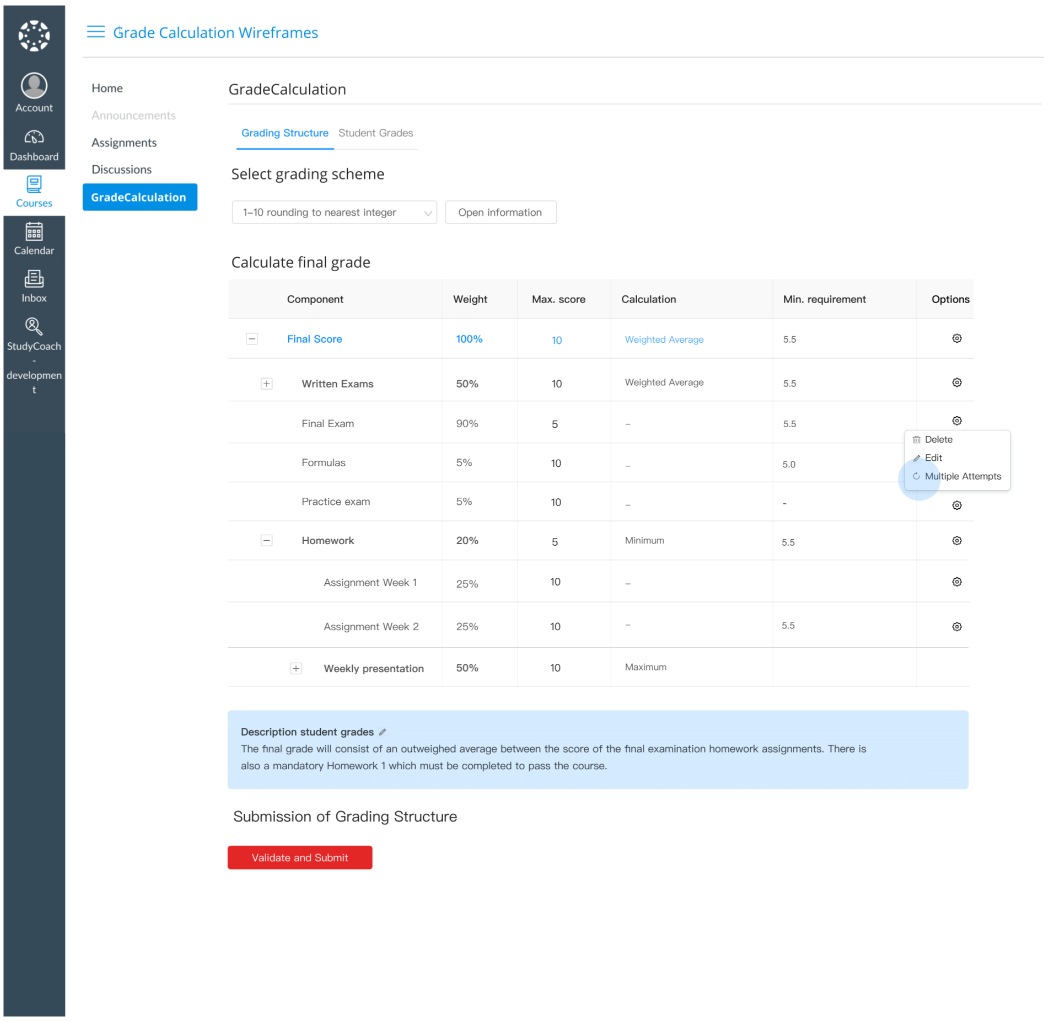
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Figure B.5: “Grading Structure" view: Create Assessment Set

The option “Add assessment set component" is the second option that appears in the “Add Component" pop-up. When selected, a different set of inputs appear. Most of them are similar to the “Add assessment component", such as name, maximum score, and weight. However, there is also a new one that appears only for the assessment sets. This new option is the calculation method used to calculate the score for the assessment set. These calculation methods are the following; weighted average, unweighted average, best/worst x out of y assessments, maximum, minimum, sum and latest counts, as defined in the URD [1]. When all required inputs are filled, the teacher can finalize the process by clicking the button “Create" to complete the creation process of a set of assessments.

GREAT GRADERS

B.1.5 Multiple Attempts Overview



GradeCalculation

Select grading scheme

1–10 rounding to nearest integer

Calculate final grade

Component	Weight	Max. score	Calculation	Min. requirement	Options
Final Score	100%	10	Weighted Average	5.5	⚙
Written Exams	50%	10	Weighted Average	5.5	⚙
Final Exam	90%	5	–	5.5	⚙
Formulas	5%	10	–	5.0	
Practice exam	5%	10	–	-	⚙
Homework	20%	5	Minimum	5.5	⚙
Assignment Week 1	25%	10	–		⚙
Assignment Week 2	25%	10	–	5.5	⚙
Weekly presentation	50%	10	Maximum		

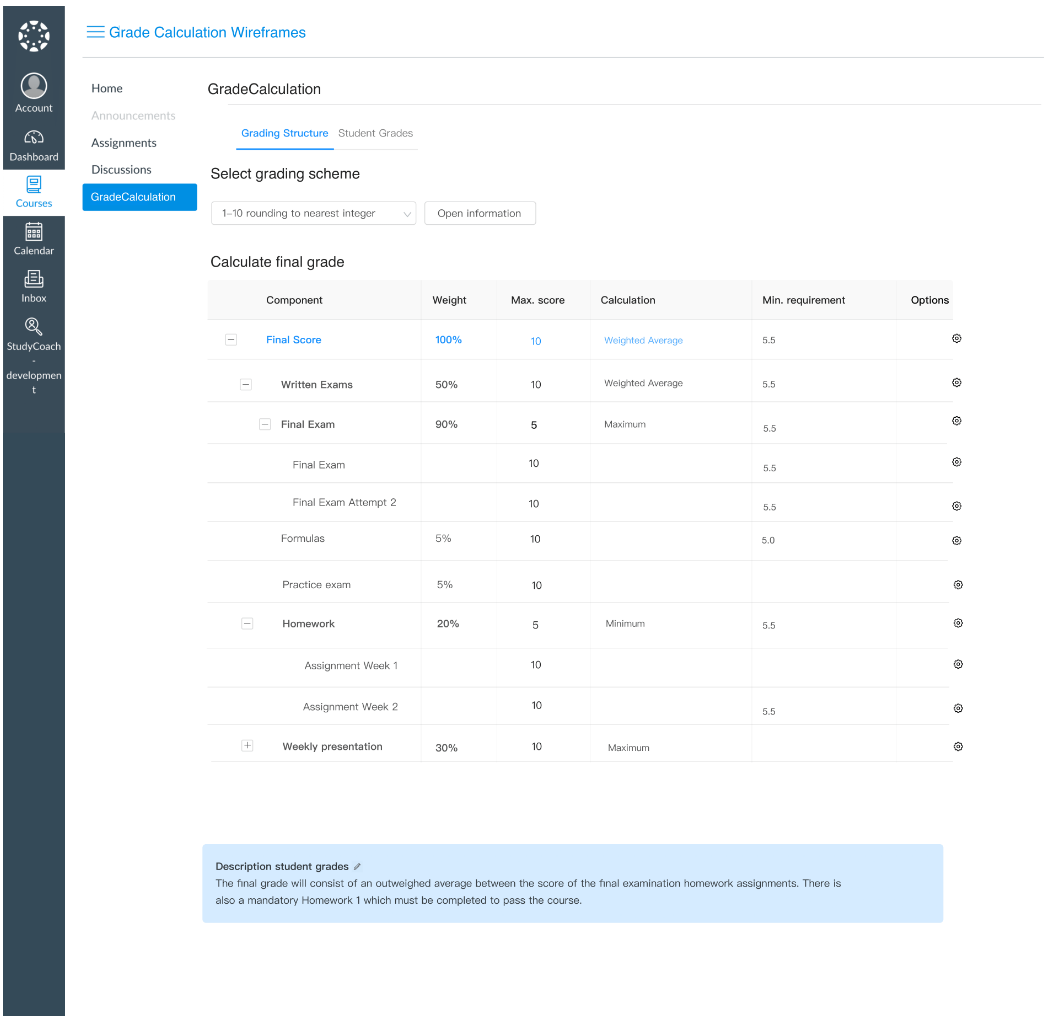
Description student grades
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Submission of Grading Structure

Figure B.6: “Grading Structure" view: Multiple Attempts Overview

When the teacher clicks on the options menu of an individual assessment, the third option in the option menu is “Multiple attempts". When this option is selected for an assessment, a second attempt for this assessment is created and both the original assessment and its second attempt are placed in a new assessment set. This assessment set will replace the original assessment, assuming the original assessment's weight, grading type, max. score, and minimum requirement. The calculation of the assessment set will be set, by default, to “Maximum". An example for the assessment “Final Exam" can be seen in figure B.7 below. Here the second attempt for “Final Exam" was created and added with the original “Final Exam" assessment to the assessment set “Final Exam". As the original “Final Exam" assessment had a weight of 90%, with a maximum score of 10 and a minimum requirement of 5.5, the new assessment set assumed these values. The calculation method of the assessment set is then set to “Maximum" and the original assessment and its second attempt then drop their weight as they are now part of an assessment set with calculation method “Maximum" and thus they do not need a weight. With these changes, the “Final Exam" assessment now has two attempts where the highest mark the student attains will count for 90% of the partial score for the assessment “Written Exam".

B.1.6 Multiple Attempt Added



GradeCalculation

Select grading scheme

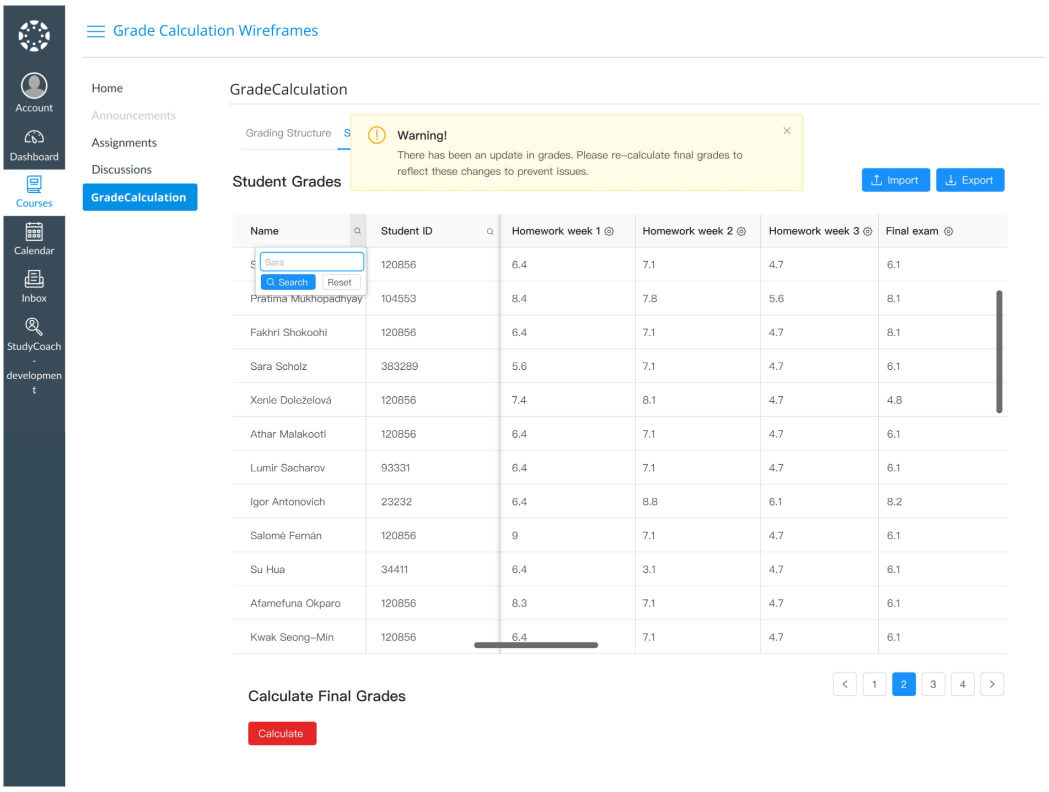
Calculate final grade

Component	Weight	Max. score	Calculation	Min. requirement	Options
⊟ Final Score	100%	10	Weighted Average	5.5	⚙
⊟ Written Exams	50%	10	Weighted Average	5.5	⚙
⊟ Final Exam	90%	5	Maximum	5.5	⚙
Final Exam		10		5.5	⚙
Final Exam Attempt 2		10		5.5	⚙
Formulas	5%	10		5.0	⚙
Practice exam	5%	10			⚙
⊟ Homework	20%	5	Minimum	5.5	⚙
Assignment Week 1		10			⚙
Assignment Week 2		10		5.5	⚙
⊞ Weekly presentation	30%	10	Maximum		⚙

Description student grades
The final grade will consist of an outweighed average between the score of the final examination homework assignments. There is also a mandatory Homework 1 which must be completed to pass the course.

Figure B.7: “Grading Structure" view: Multiple Attempt Added

B.2 Student Grades view



GradeCalculation

Warning!
There has been an update in grades. Please re–calculate final grades to reflect these changes to prevent issues.

Student Grades

Name	Student ID	Homework week 1	Homework week 2	Homework week 3	Final exam
S	120856	6.4	7.1	4.7	6.1
Pratima Mukhopadhyay	104553	8.4	7.8	5.6	8.1
Fakhri Shokoohi	120856	6.4	7.1	4.7	8.1
Sara Scholz	383289	5.6	7.1	4.7	6.1
Xenie Doleželová	120856	7.4	8.1	4.7	4.8
Athar Malakooti	120856	6.4	7.1	4.7	6.1
Lumir Sacharov	93331	6.4	7.1	4.7	6.1
Igor Antonovich	23232	6.4	8.8	6.1	8.2
Salomé Fernán	120856	9	7.1	4.7	6.1
Su Hua	34411	6.4	3.1	4.7	6.1
Afamefuna Okparo	120856	8.3	7.1	4.7	6.1
Kwak Seong–Min	120856	6.4	7.1	4.7	6.1

Calculate Final Grades

Figure B.8: Student Grades view: Overview page

The purpose of this view is for the teacher to have access to and see the grades and marks for all of their students in the selected course. At the top right corner, there are two buttons namely “Import" and “Export". Clicking on the “Import" button enables the teacher to upload an assessment's marks using a CSV file. On the other hand, clicking the “Export" button creates a pop-up where the teacher can select whether they want to export marks, partial grades, and/or final grades. Based on their selection, a CSV file will be downloaded to the teacher's computer containing the relevant information.

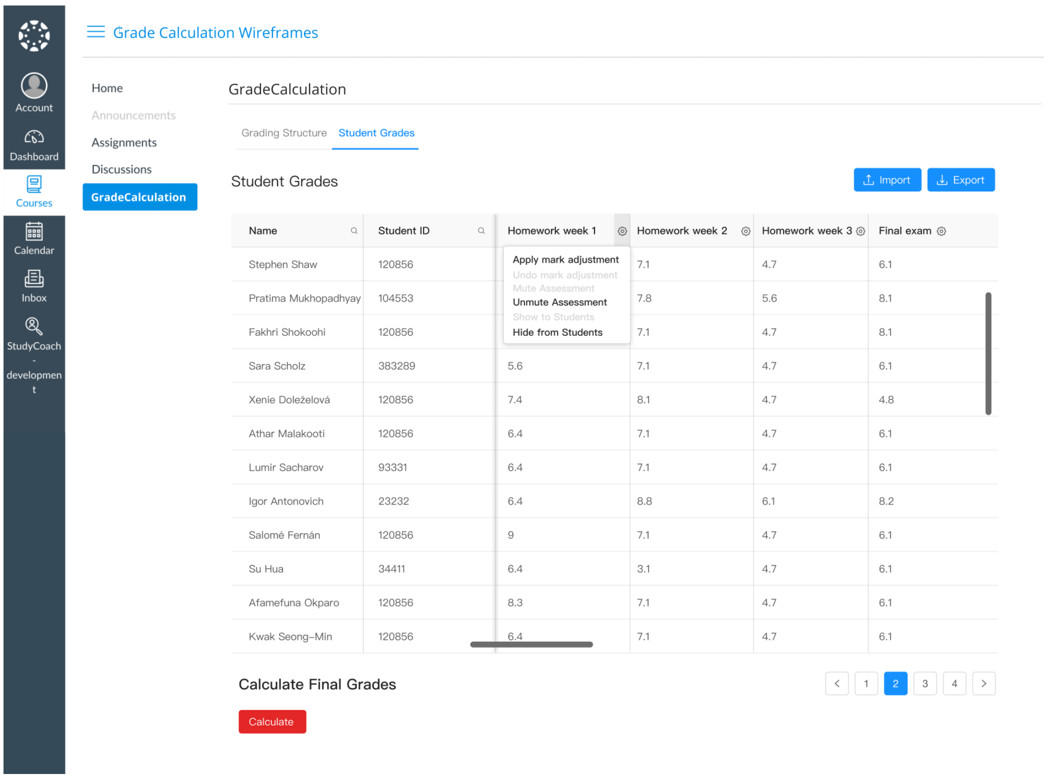
The main focus of this tab is the table at the center. The first two columns, “Name" and “Student ID" are visible at all times, while the rest of the columns depend on the number of assessments imported by the teacher and the construction of the final score structure. For each imported assessment and each assessment set a column is added where the corresponding cell is filled with the student's grade or mark.

As this table can grow extremely large, the teacher is given the option to search the data of the table. The teacher is able search based on either the Student Name or the Student ID. This can be done by pressing the “Search" icon that is next to the header of the columns, “Name" and “Student ID", respectively. In future version of Grade Calculation, Great Graders would also recommend adding a filter option for each column.

As can be seen in the figure above, after the first two columns the search is replaced by an option button. This options button entails a plethora of options available for the teacher to use such as mute and unmute, show and hide, and mark adjustment. A further explanation of these features is given in the figure below.

At the bottom of the page, there is a red button, called “Calculate". When the grading structure and marks are finalized, pressing this button will calculate the partial grades for each assessment set as well as calculate the final grade for each student based on the marks they attained for each assessments. These partial grades and final grades will be displayed to the teacher by adding columns for the grades in the table of the “Student Grades" view.

B.2.1 Settings Button



GradeCalculation

Student Grades

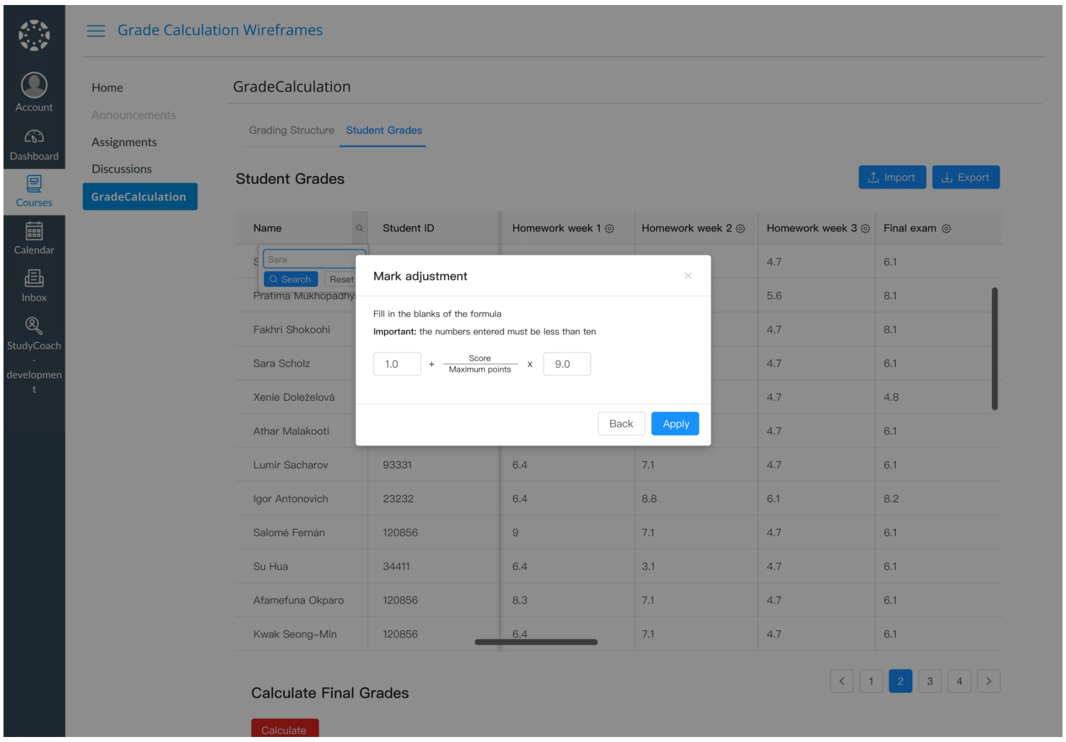
Name	Student ID	Homework week 1	Homework week 2	Homework week 3	Final exam
Stephen Shaw	120856		7.1	4.7	6.1
Pratima Mukhopadhyay	104553		7.8	5.6	8.1
Fakhri Shokoohi	120856		7.1	4.7	8.1
Sara Scholz	383289	5.6	7.1	4.7	6.1
Xenie Doleželová	120856	7.4	8.1	4.7	4.8
Athar Malakooti	120856	6.4	7.1	4.7	6.1
Lumir Sacharov	93331	6.4	7.1	4.7	6.1
Igor Antonovich	23232	6.4	8.8	6.1	8.2
Salomé Fernán	120856	9	7.1	4.7	6.1
Su Hua	34411	6.4	3.1	4.7	6.1
Afamefuna Okparo	120856	8.3	7.1	4.7	6.1
Kwak Seong-Min	120856	6.4	7.1	4.7	6.1

Calculate Final Grades

Figure B.9: Student Grades: Settings Button View

In the figure above, a view of the options menu is shown. This menu is shown only after the options button next to the name of each assessment is clicked by the teacher. The items that appear in this menu are “Apply mark adjustment", “Undo a mark adjustment", “Mute assessment", “Unmute assessment", “Show assessment", and finally, “Hide assessment". Although it may seem that muting and unmuting is identical to showing and hiding an assessment there is an important different between them. When muting an assignment, the student is still able to see that the assessment exists however, is unable to see the grade or mark they have attained for this assessment. On the other hand, when an assessment is hidden, the entire assessment row is removed from the “Student Interface" and thus the existence of the assessment is not shown to the student. Lastly, the mark adjustment is used to adjust the marks based on the difficultly of the assessment. This is widely practice in the Netherlands, which is where the development of Grade Calculation is located. Thus, this option was a major focus during development and is further elaborated in Appendix B.2.2.

B.2.2 Mark Adjustment



GradeCalculation

Student Grades

Mark adjustment

Fill in the blanks of the formula

Important: the numbers entered must be less than ten

$1.0 + \frac{\text{Score}}{\text{Maximum points}} \times 9.0$

Name	Student ID	Homework week 1	Homework week 2	Homework week 3	Final exam
S				4.7	6.1
Pratima Mukhopadhy				5.6	8.1
Fakhri Shokoohi				4.7	8.1
Sara Scholz				4.7	6.1
Xenie Doleželová				4.7	4.8
Athar Malakooti				4.7	6.1
Lumir Sacharov	93331	6.4	7.1	4.7	6.1
Igor Antonovich	23232	6.4	8.8	6.1	8.2
Salomé Fernán	120856	9	7.1	4.7	6.1
Su Hua	34411	6.4	3.1	4.7	6.1
Afamefuna Okparo	120856	8.3	7.1	4.7	6.1
Kwak Seong–Min	120856	6.4	7.1	4.7	6.1

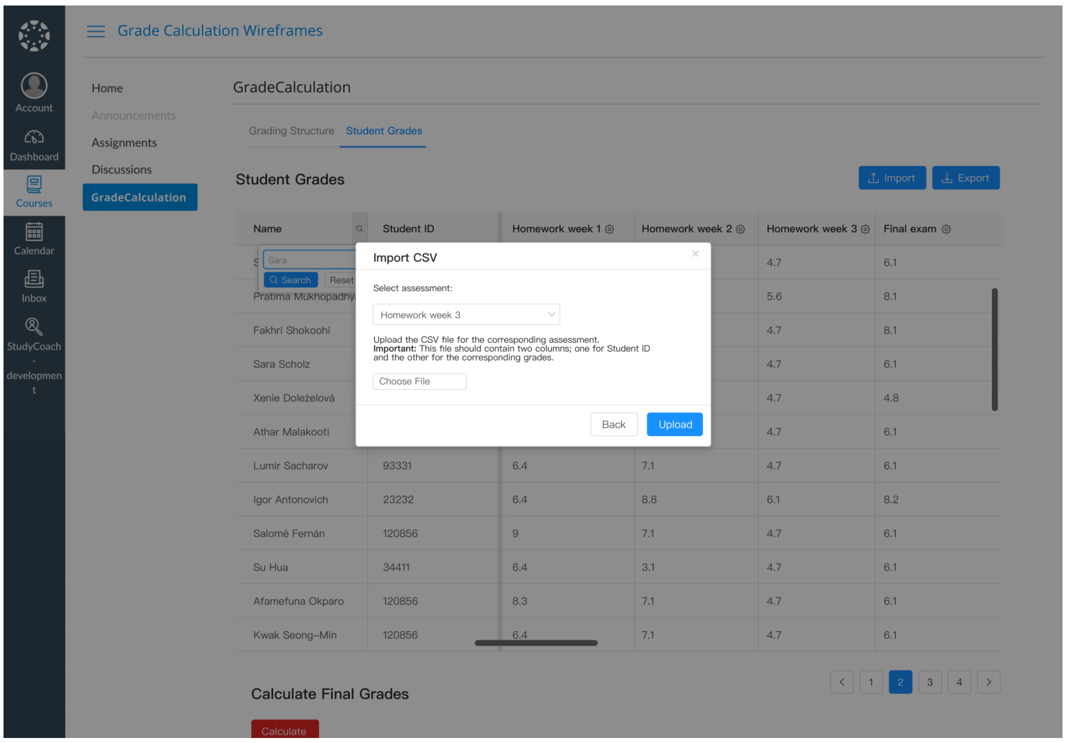
< 1 2 3 4 >

Calculate Final Grades

Figure B.10: Student Grades: Mark Adjustment

If the mark adjustment is selected for an assessment then a pop-up window appears prompting the user to input the mark adjustment formula they desire. The pop-up has two input fields which construct the formula for adjustment when they are filled in. An example of such a formula can be seen in the figure above. Then, when the “Apply" button is clicked the marks of the corresponding assessment are changed based on the inputted formula.

B.2.3 Import and Export Buttons



Student Grades

Name	Student ID	Homework week 1	Homework week 2	Homework week 3	Final exam
S				4.7	6.1
Pratima Mukhopadhy				5.6	8.1
Fakhri Shokoohi				4.7	8.1
Sara Scholz				4.7	6.1
Xenie Doleželová				4.7	4.8
Athar Malakooti				4.7	6.1
Lumir Sacharov	93331	6.4	7.1	4.7	6.1
Igor Antonovich	23232	6.4	8.8	6.1	8.2
Salomé Fernán	120856	9	7.1	4.7	6.1
Su Hua	34411	6.4	3.1	4.7	6.1
Afamefuna Okparo	120856	8.3	7.1	4.7	6.1
Kwak Seong–Min	120856	6.4	7.1	4.7	6.1

Import CSV

Select assessment: Homework week 3

Upload the CSV file for the corresponding assessment.
Important: This file should contain two columns; one for Student ID and the other for the corresponding grades.

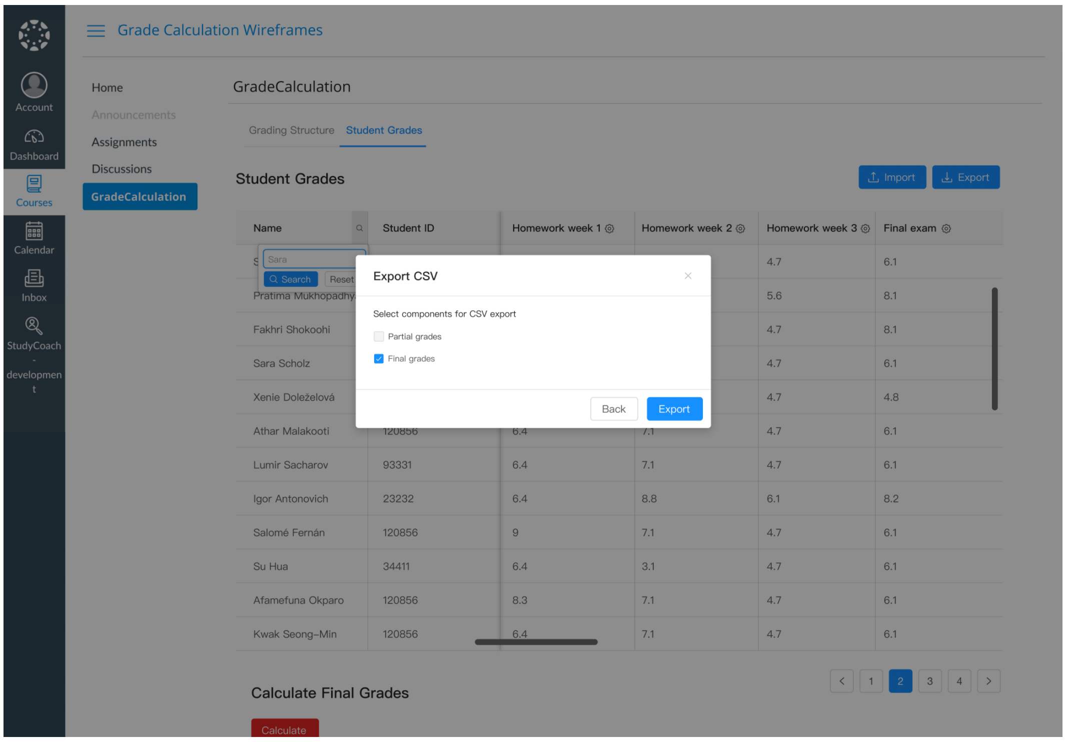
Choose File

Back Upload

Calculate Final Grades

Figure B.11: Student Grades: Import Button View

Clicking on the “Import" button on the top right-hand side of the “Student Grades" tab, will produce the pop-up shown above. On the top of the pop-up, a drop-down menu can be seen. In this drop-down menu the teacher can select the assessment for which the marks will be imported for. After the assessment has been chosen, the teacher can click on the “Choose File" button. Then a local import window will be shown to the teacher. In this local window, the teacher can select a CSV file which contains the marks for the chosen assessment.



GradeCalculation

Student Grades

Name	Student ID	Homework week 1	Homework week 2	Homework week 3	Final exam
S				4.7	6.1
Pratima Mukhopadhy				5.6	8.1
Fakhri Shokoohi				4.7	8.1
Sara Scholz				4.7	6.1
Xenie Doleželová				4.7	4.8
Athar Malakooti	120856	6.4	7.1	4.7	6.1
Lumir Sacharov	93331	6.4	7.1	4.7	6.1
Igor Antonovich	23232	6.4	8.8	6.1	8.2
Salomé Fernán	120856	9	7.1	4.7	6.1
Su Hua	34411	6.4	3.1	4.7	6.1
Afamefuna Okparo	120856	8.3	7.1	4.7	6.1
Kwak Seong–Min	120856	6.4	7.1	4.7	6.1

Export CSV

Select components for CSV export

- [ ] Partial grades
- [✓] Final grades

Calculate Final Grades

Figure B.12: Student Grades: Export Button View

When the “Export" button is clicked in the“Student Grades" view, the pop-up shown in the figure above will appear. In this pop-up the teacher would first have to select whether they would like to export marks, partial grades, and/or the final grades of the students. Based on their selection, a CSV file will be downloaded to the teacher’s computer containing the relevant information.

Appendix C

Transitions

The previous section discusses user interfaces in great detail. In turn, this section introduces transition diagrams using the Petri Nets representation. These diagrams schematically model the behaviour of the application, making use of places and transitions in order to reflect various views every interface has and the actions that are possible on these views, respectively.

C.1 Application Launch

Grade Calculation application is a plugin which is launched directly from the LMS, Canvas. Upon application launch, Drieam Framework performs user authentication, which is used by the Grade Calculation plugin. During the authentication, the user's role is verified as being either a Teacher (with the permissions stated in [1] section 2.4.1) or a Student, since these are the main two roles. Based on the role, the user is then redirected to the corresponding interface. The figure C.1 shows the corresponding transition to the respective interface.

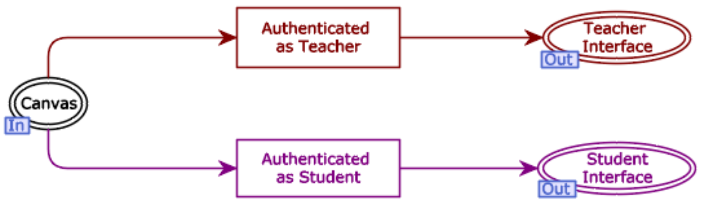


Figure C.1: Transition Diagram of Application Launch

Both Teacher and Student interfaces allow for multiple transitions, which are discussed in the sections below.

C.2 Student Interface

The Student Interface displays the marks and grades for a particular student. Here the student can examine their grades by expanding the tree-view table. Additionally, there is a possibility for the student to export their marks and grades. For this purpose, the application has a separate Export View, implemented as a pop-up dialog. The figure C.2 below captures these actions in a transition diagram.

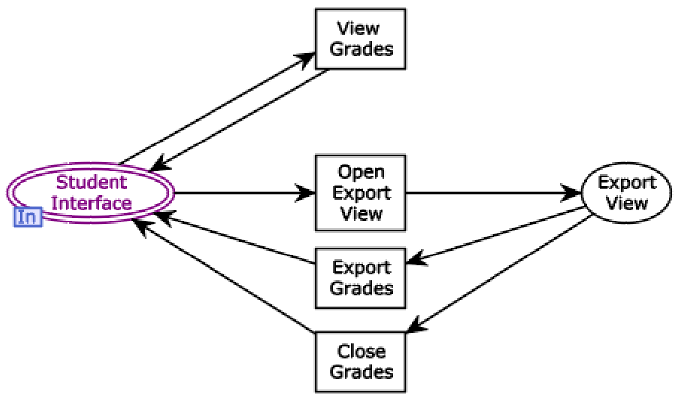


Figure C.2: Transition Diagram of Student Interface

C.3 Teacher Interface

The Teacher Interface has two views: “Grading Structure" view and“Student Grades" view. They can be reached by clicking the appropriate tabs in the interface (see figure C.3).

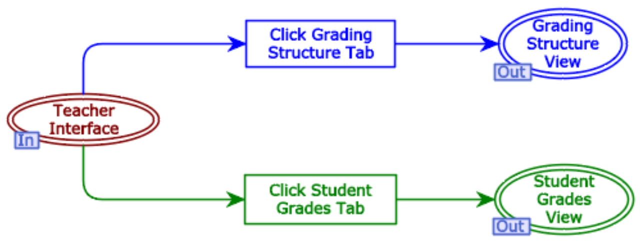


Figure C.3: Transition Diagram of Teacher Interface

The figure C.4 below shows the transition diagram for the “Grading Structure" view. It is possible to access the Information View from this page. Besides this, the “Grading Structure" view allows a teacher to select a grading scheme, edit the final score structure, edit description of grading structure, and save the progress. These are self-transitions because they do not require separate views and can be implemented on the “Grading Structure" view itself (e.g. by using a drop-down menu, a button, or an editable text field). Finally, there is a possibility to validate and submit the final score structure once it has been completed.

GREAT GRADERS

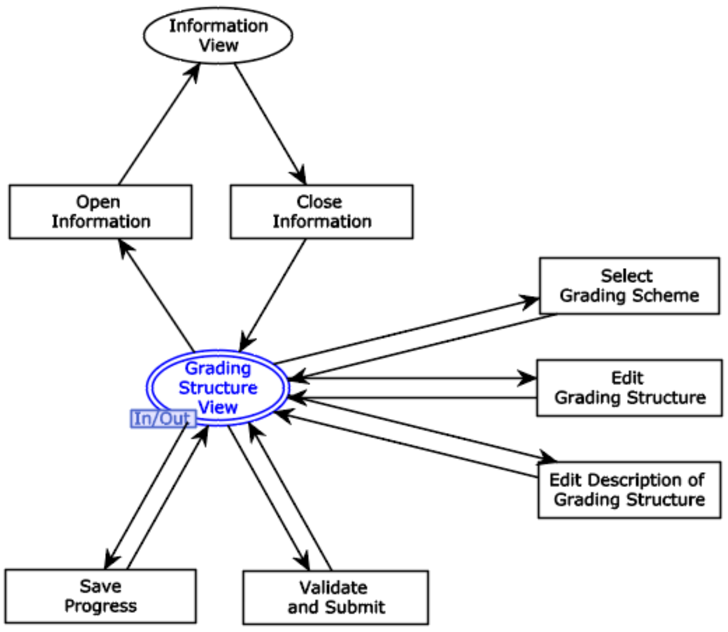


Figure C.4: Transition Diagram of “Grading Structure" view

The last figure C.5 captures the behavior of the“Student Grades" view. The main purpose of this view is to display the grades of all students to a teacher of the specific course.
When the teacher navigates to the“Student Grades" view, the application detects if there has been an update in the grades. If this is the case, the application will display a warning, notifying that the grades have to be recalculated. In the diagram, this is the state called “Student Grades view with Warning". The teacher can either close this warning, working in Outdated“Student Grades" view (without the warning), or initiate the recalculation of grades, getting to an up-to-date“Student Grades" view (without the warning). The three places mentioned above allow for the same transitions, however only the“Student Grades" view is elaborately explained in the diagram for simplicity.
The grades on the“Student Grades" view are displayed in columns per assessment. Furthermore, the view allows for the manipulation of marks of a specific assessment. Adjusting the marks is possible via setting the “Mark adjustment", which is implemented as a pop-up dialogue, and thus is shown separately as the Mark Adjustment View. Furthermore, it is possible to mute and unmute any assessments, as well as hide and show the final grades. Finally, there is a possibility for the teacher to search the grades by the student name or the student ID.
Moreover, the“Student Grades" view supports importing and exporting of the“Student Grades". Both of these actions require separate views, which are displayed as Import Grades View and Export Grades View in the diagram.

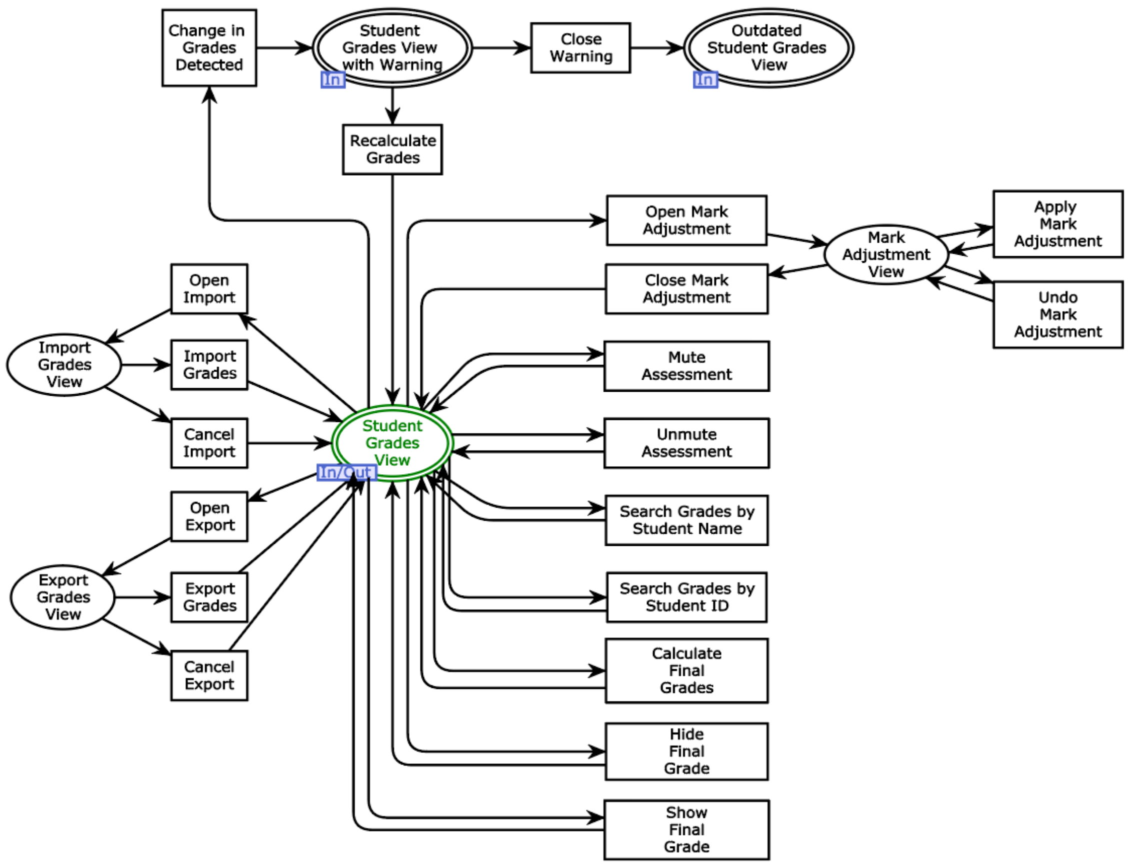


Figure C.5: Transition Diagram of“Student Grades" view