

The logo for Pfaff. Corp features the text "Pfaff. Corp" in a bold, black, serif font. A red, curved swoosh underline starts under the "Pfaff." and extends under the "Corp", ending with a small upward curve. A small "TM" trademark symbol is positioned to the upper right of the "Corp" text.

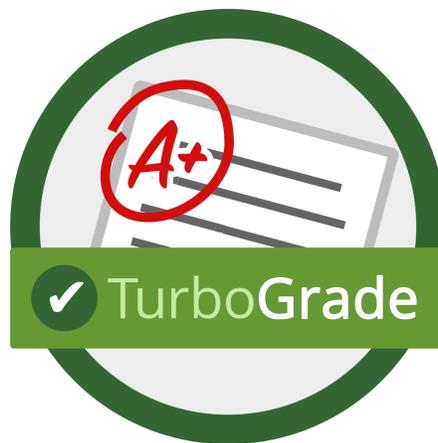
Pfaff. CorpTM

Leaders of design decisions.

CS 205: Software Engineering

May 11th, 2017

TurboGradeTM



Project Synthesis

Wassim Gharbi, Darren Norton, Sam Cutrone

I . Problem Definition	3
II. Client Request and Major Features	3
III . Rubric Use	4
IV . Grading Scheme	5
V . Implemented Features	7
Implemented requirements:	8
Additional features:	8
VI . User Stories	9
VII . Process View	13
VIII . Use cases	14
Class Management	14
Section Management	17
Assignment Management	20
Grading Engine	23
Overview/Submission Report	27
Student Deliverable / Gradebook	29
Data Archiving	31
IX . Logic View	33
X . Final Database Design	36
XI . Final User Interface Design	39
XII. Physical View	45
XIII Installation Process	48
XIV. Data Archival & Git Synchronization	49
XV. Testing Plan	54
XVI. Reflections	54
XVII. User Manual	55

I . Problem Definition

As the Computer Science department is growing, grading labs for introductory classes (CS105 and CS150) is becoming less and less efficient. The current approach for grading lab submissions relies on repetitive tasks and unsuitable software causing an inefficient workflow.

Professors/TAs download submissions from Moodle, individually look at each of the students' folders and keep going back and forth between multiple windows:

- the student's folder in the Finder
- the student's code in BlueJ/Processing
- a text document that contains comments on the student's work
- an Excel sheet that has the final grades for each submission as well as the comments given to the student.

In addition to the grading process being inefficient and inconvenient for the professors, the current approach relies on out-of-context text comments delivered to the student on moodle. This leads to suboptimal feedback to the student which causes students to either ignore the comments (impairing the learning process) or ask for clarification from the professor.

II. Client Request and Major Features

I want a more convenient way of grading student submissions for the CS105 and CS150 labs that, at the same time, gives the students clear and visually meaningful feedback on their code. I want to be

able to:

- Grade students' submissions
- Grade the assignments based on a rubric
- Grade the submission anonymously
- Use the software on multiple computers (clients) and resume grading wherever he stopped on the last computer he used
- Use the software on multiple screen sizes (laptop, desktop, etc.)
- Export grades to an Excel sheet
- Export individual submissions to a deliverable (PDF or text)
- See meaningful statistics about student progress
- Reuse previous comments through auto-complete

As part of the technical project requirements, the software must:

- Be able to archive grading progress to a Git repository
- Work both online and offline
- Work on Windows, Mac, Linux (Platform agnostic)
- Have a simple/friendly installation process
- Have an "uninstall" process
- Have an intuitive user interface

III . Rubric Use

In their current form, rubrics, especially in technical fields, are primarily point based, assigning values in a set range to task accomplished to a certain amount of proficiency. In addition, a pass/fail-style assignment of a quantity of points can be present in certain portions of rubrics, if the important aspect of an assignment was the binary capability or inclusion of some element.

To offer the greatest amount of assistance to the user, TurboGrade's rubric layout falls incredibly close to this model of a rubric. With every comment made, the user is given the option to associate that comment with a given rubric element, and assign a corresponding score. This rubric layout also offers a level of accountability to both the professor, and the student being graded, as every point is tied to the rubric. This design promotes accurate point allocation, and clear task definition, to ultimately result in a grade most aligned with the professor's assignment desires.

For most rubrics regarding lab grading, there is a combination of these two evaluation styles, with several levels of segmentation existing, depicting the varying levels of specificity to point assignment. With TurboGrade, as rubrics are created, they can be filled with a hierarchical assignment of points, beginning with general parent categories, such as functionality or style, and specifying within those categories further. For consistency of grading, parent categories can not be assigned point values to. Instead, each of these parent categories will automatically be assigned a point value equal to the sum of the point values of every one of their child categories. If a category has no subcategories within it, a maximum point value can be assigned to that category, enabling it to receive a score ranging from 0 to that maximum value, during the grading process. In addition, given the subjectivity of certain aspects gradable by a professor, TurboGrade offers a second template for grading categories. As opposed to a category with a score composed of a total of subcategories, certain categories can be seen as using an "essay-style" grading template. In this second method of grading, a category can be assigned a point from 0 to the maximum value, with each score corresponding to a level of correctness in the category, and providing several attributes as a reasoning for why that score was given.

IV . Grading Scheme

Currently, the grading scheme divides criteria into a series of categories (ex: Design, Correctness, etc.). Maximum point values are attributed to every sub-category which summed up give the total score for each category. Accompanying the final grades on each category, several comments are made, either pertaining to particular categories mentioned, or generally relating to the submission as a whole. The proposed grading scheme incorporates much of the same core ideas, but with a greater level of flexibility to account for differences in grading between professors and assignments (ex: lab versus projects).

During the grading process, when a submission is being overlooked, the user has the option to leave comments tagged to particular lines of the assignment. When leaving a comment, in addition to the text-based message, the user can also leave a point adjustment value (either a penalty such as -1, -2, etc or an extra credit such as +1, +2 etc) paired with the comment. These comments (and their corresponding point values, if present) are to be associated with one of the scorable portions of a rubric that were defined by the user, at the creation of the assignment.

However, it is clear that the user might not have a specific comment for every category, or may have not been able to fully define a category's score through the commenting process. Similarly, certain grades may be dependent on the entirety of the assignment, or may be more subjective and not be tied to any particular tasks. To accommodate this pattern of grading, the user will be able to modify any of the grades, with respect to each category of the rubric. Any category that utilized an "essay-style" grading process will be unable to have been filled during the grading process, and will be left blank when the user arrives to the final screen. These categories, like any others, have the capacity to be altered after the grading process is done. Following the commenting process, the user will be provided with a breakdown of what grades would be assigned, based on the scores associated with each of the comments. This final "Submission Report/Overview" screen (*Figure 1.*) would present a final rubric, partially or fully filled, containing any grades, and the comments that contributed to those grades. The user can then fill in any remaining grades, or alter any grades already defined. In addition, the user will be allowed to add additional general comments, allowing for a final grade before being exported as the student deliverable PDF.

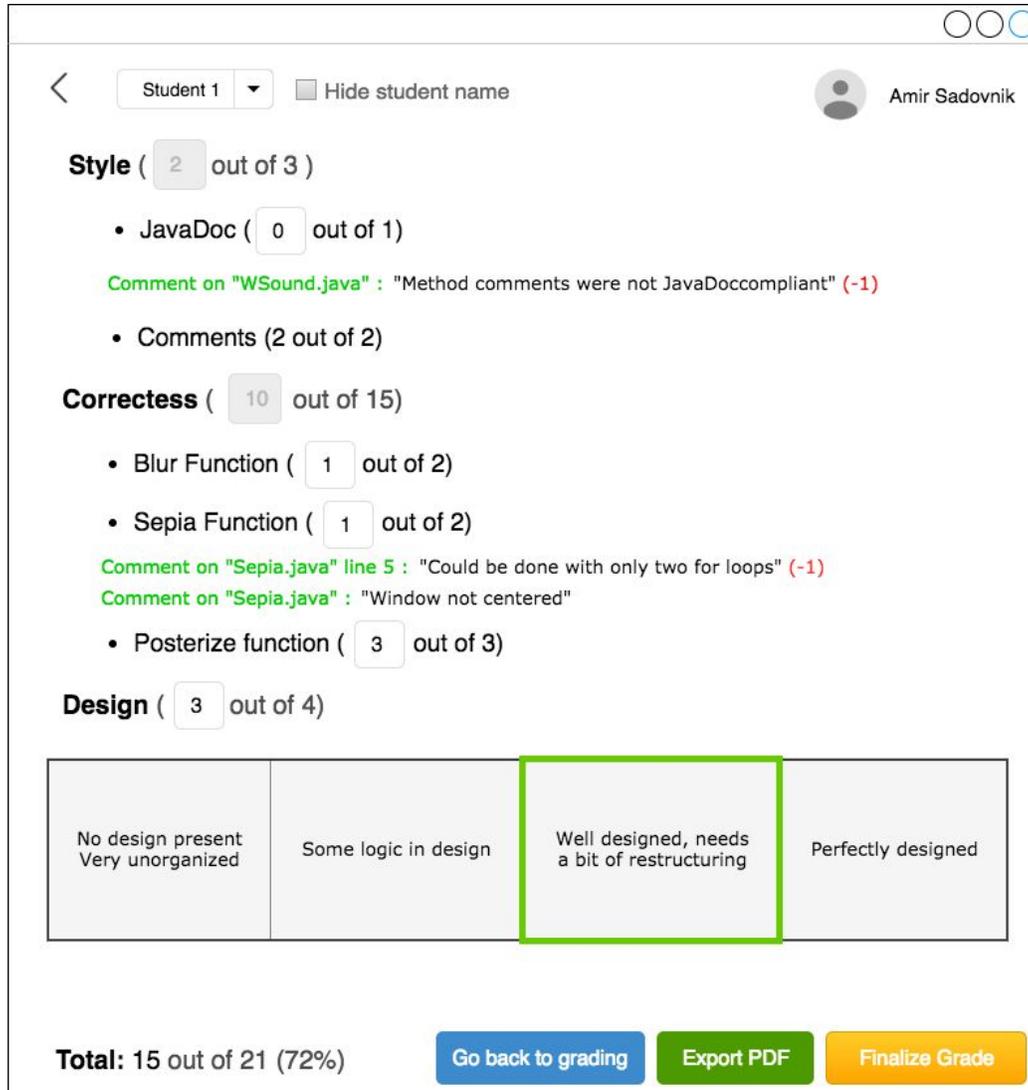


Figure 1. Wireframe showing the “Submission Report/Overview” once a submission is graded

V . Implemented Features

At the current stage, our software is an almost finished product, encompassing multiple features that make it easier for the professor to grade assignments.

Implemented requirements:

- Grade student submissions ✓
- Grade the assignments based on a rubric ✓
- Grade the submission anonymously ✓
- Use the software on multiple computers (clients) and resume grading wherever he stopped on the last computer he used

Note: We implemented synchronization, however the user doesn't have the choice of favoring local vs remote changes yet

- Use the software on multiple screen sizes (laptop, desktop, etc.) ✓
- Export grades to an Excel sheet ✓
- Export individual submissions to a deliverable (PDF or text) ✓
- See meaningful statistics about student progress

Note: We implemented simple statistics such as the mean and mode, however we are still missing charts

- Reuse previous comments through auto-complete ✓
- Be able to archive grading progress to a Git repository ✓
- Work both online and offline ✓
- Work on Windows, Mac, Linux (Platform agnostic)

Note: We have found some compatibility problems with Git and Processing on Windows)

- Have a simple/friendly installation process ✓
- Have an “uninstall” process ✓
- Have an intuitive user interface ✓

Additional features:

- Ability to run Process Sketches from within the software
- Compile errors shown inside the grading interface
- In-UI Interactive Rubric Maker
- Automatically adds students (no need for CSV)
- Grading Progress
- Organization by semester/year

- Works on mobile
- Reports are in PDF format

Our project development timeline was as follows:

Week 9	<ul style="list-style-type: none"> + Formal Design Proposal - Beginning of development + Grading Engine (display of student code, adding comments, linking comments to portion of code) - at this stage the grading is not saved + Start work on deliverable PDF + Integration Testing
Week 10	<ul style="list-style-type: none"> + Begin implementing database (minimum tables for a single submission) + Rubric creation form + Grading Engine (linking comments to a rubric, attributing scores to a rubric, point adjustment with comments through deductions and extra credits) + Integration Testing
Week 11	<ul style="list-style-type: none"> + Finish implementing database (class, section, student and submission tables) + Implement class and section creation + Develop statistics + Work on synchronization between different clients + Integration Testing
Week 12	<ul style="list-style-type: none"> + Refine user interface + Improve deliverables (PDF and Excel files) + Perform System Testing and Acceptance Testing

VI . User Stories

Class View (optional)

- As a professor, I want to see a listing of classes I am teaching
- As a professor, I want to create a new class if it does not exist on the software
- As a professor, I can delete a class
- As a professor, I can edit the name of a class
- As a professor, if I click on a class, I want to move to the Section View

Section View (optional)

- As a professor, I want to add a section to a class
- As a professor, I want to add students to a section
- As a professor, I want to remove students from a section
- As a professor, I can edit the name of a section
- As a professor, I can delete a section
- As a professor, if I click on a section, I want to move to the Assignment View

Assignment View (optional)

- As a professor, if I have assigned a Lab/Project to a section, I want to be able to see it in the assignment view
- As a professor, if I click on an existing Lab/Project, I want to move to the Submission List
- As a professor, if I want to assign a new Lab/Project to a section, I want to be able to move to the Assignment Creation Form.
- As a professor, I can hide existing projects from my view, and reopen them at a later point.
- As a professor I can remove an assigned project from a section while keeping it assigned to other sections of the course

Assignment/Rubric Creation Form (critical)

- As a professor, I want to be able to create a Lab/Project and give it a name.
- As a professor, I can assign a Lab/Project to a section and point it to the location of a folder that contains student submissions (from Moodle).
- As a professor, I can create a rubric by choosing from previous criteria or by adding new ones.
- As a professor, I want to be able to rearrange grading criteria in the rubric, create hierarchies and assign a maximum score to each criterion.
- As a professor, I want to be able to dynamically edit the project's name and grading rubric.

Submission List (optional)

- As a professor, I want to list all submissions on a specific Lab/Project assigned to a section
- As a professor, I want to see the status of each submission (graded or not) and the current grade attributed to the submission
- As a professor, if I click on a student's submission, I want to go to the Grading View to start grading the submission.

- As a professor, if I click the “Start Grading” button, I want to go to the Grading View and start grading an anonymous batch of submissions.
- As a professor, I want to see if a registered student has not submitted a Lab/Project.
- As a professor, I want to continue grading or edit previous work on a submission that I previously worked on.

Grading View (critical)

- As a professor, I want to browse/move to other files within a submission.
- As a professor, I want to be able to comment in context (link a comment to a code block).
- As a professor, I want to comment quickly using auto-completed comments.
- As a professor, I want link a comment to a specific rubric category.
- As a professor, I want to be able to link a point adjustment (penalty/extra credit) to the comment, which will be used to estimate the total grade for the submission.
- As a professor, I can see the comments I left on each file.
- As a professor, given that I am batch grading, I want to view my progress in completing the batch.
- As a professor, I want to be able pause then resume grading the submission I was working on.

Submission Report/Overview (critical)

- As a professor, I want to look at an overview of comments that I made on the most recent submission
- As a professor, I want to look at the points I have added/subtracted with each comment for each rubric criterion
- As a professor, I want to get an adjustable estimate of the grade assigned to each criterion based on the points added/subtracted with each comment
- As a professor, I want to override/amend points in a rubric criterion.
- As a professor, after reviewing/adjusting grades, I want to mark the submission as completed.
- As a professor, after reviewing/adjusting grades, I want to be able to return to the Grading View and add comments if grading is not completed
- As a professor, given I have marked a submission as “completed” and given I am grading an anonymous batch of submissions, I should be redirected to the Grading View to grade a new anonymous submission.

- As a professor, given I have marked a submission as “completed” and given I am grading/editing a single submission, I should be redirected back to the Submission List.

Statistics View (required)

- As a professor, I want to see statistics about how a class or section is doing
- As a professor, I want to see statistics concerning a Lab/Project assignment.
- As a professor, I want to look at an individual student’s progress (in terms of grade).
- As a professor, I want to look at a student’s progress in a certain grading criterion (completeness, commenting, style, etc..)

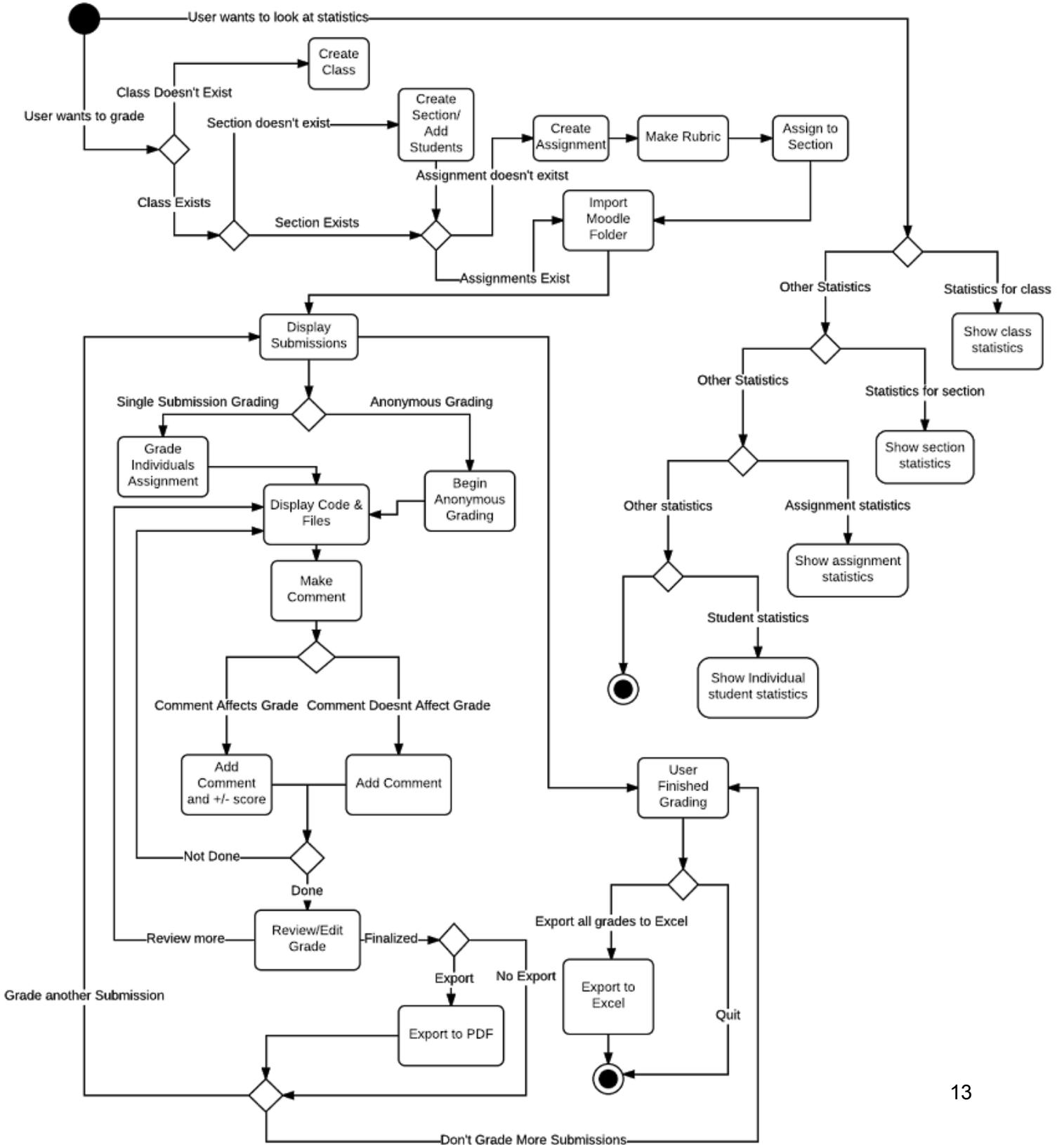
Deliverable (critical)

- As a student, I want to obtain a grade breakdown for my submission in the form of a PDF.
- As a student, I want my report to contain my score for each criterion
- As a student, I want my report to contain my professor’s comments in context of my code.
- As a student, it would be nice if I my report would have statistics about my submission (compared to the class, compared to my previous submissions).
- As a professor, I want to give myself the option to provide statistics to the student or not.

Archiving (medium)

- As a grader, I want to be able to archive my progress.
- As a grader, I want to be able to reload previous archives.
- As a grader, I want to create a save point for my grading progress.
- As a grader, I want to keep my archive consistent across multiple machines.

VII . Process View



VIII . Use cases

Class Management

Use Case Name	Create Class
Successful End Condition	Class is created
Failed End Condition	Class not created
Primary Actors	Professor
Secondary Actors	Class DB
Main Flow	<ol style="list-style-type: none">1. Professor enters the desired class name.2. Class DB checks to see if the professor has a class with the same name.3. If the class-name entered is unique, add the record,

	otherwise class creation fails.
--	---------------------------------

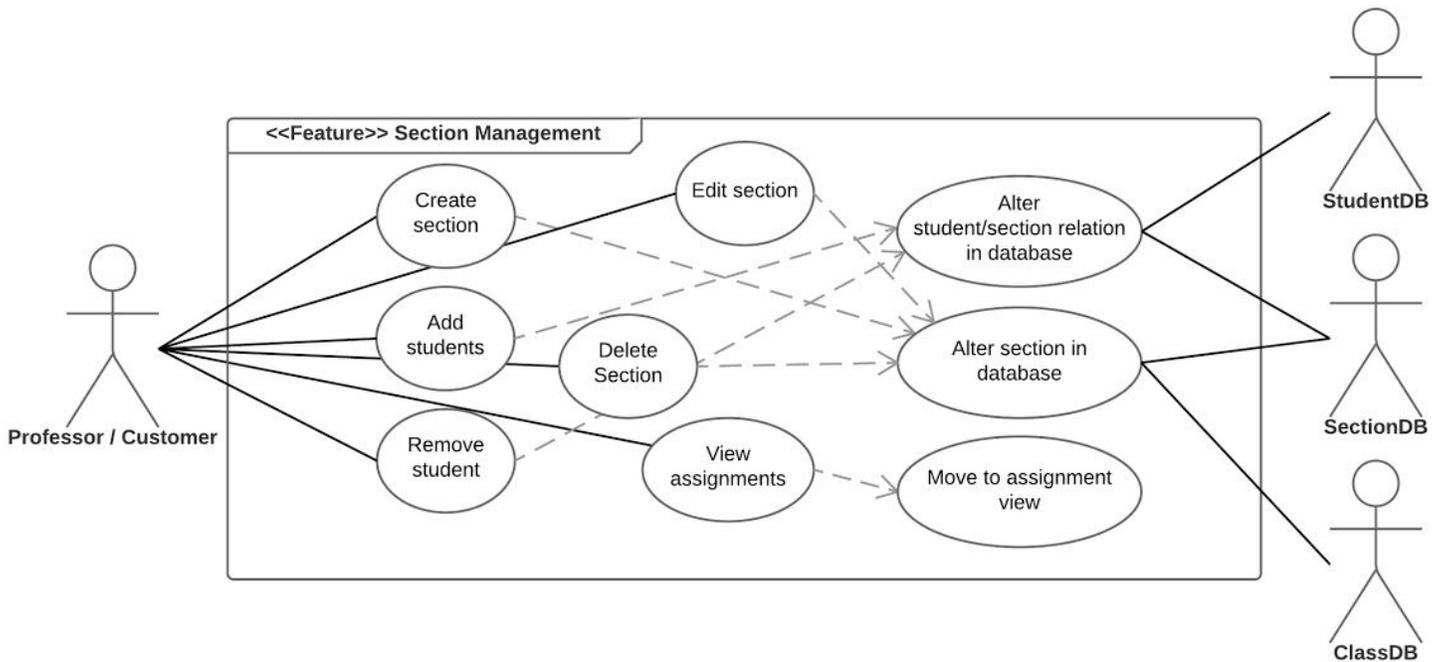
Use Case Name	View Classes
Preconditions	A class exists
Successful End Condition	The classes are viewable
Failed End Condition	The classes are not viewable
Primary Actors	Professor
Secondary Actors	Class DB
Main Flow	<ol style="list-style-type: none"> 1. Professor is directed to Class View. 2. List of classes selected from the Class DB. 3. Display the contents of the list.

Use Case Name	Delete a Class
Preconditions	The class exists.
Successful End Condition	The class is deleted from the professor's view.
Failed End Condition	The class is not deleted from the professor's view.
Primary Actors	Professor
Secondary Actors	Class DB
Main Flow	<ol style="list-style-type: none"> 1. Professor selects the class. 2. Professor wants to delete the class. 3. Professor is shown a warning message. 4. Professor confirms deletion. 5. Class DB removes the link between the Professor and the Class.

Use Case Name	Edit Class Name
Preconditions	The class exists.
Successful End Condition	The class's name changes.
Failed End Condition	The class's name doesn't change.
Primary Actors	Professor
Secondary Actors	Class DB
Main Flow	<ol style="list-style-type: none"> 1. Professor selects the class. 2. Professor wants to change the name of the class. 3. Class DB changes the name of the class.

Use Case Name	Move to Section View
Preconditions	Class exists
Successful End Condition	Moved to Section View
Failed End Condition	Stayed on Class View
Primary Actors	Professor
Main Flow	<ol style="list-style-type: none"> 1. Professor clicks on a class. 2. ViewManager loads the list of sections. 3. The list of sections is now displayed.

Section Management



Use Case Name	Create Section
Preconditions	A class is selected.
Successful End Condition	A section is added.
Failed End Condition	A section is not added.
Primary Actors	Professor
Secondary Actors	ClassDB, SectionDB
Main Flow	<ol style="list-style-type: none"> 1. Professor enters the desired section name. 2. SectionDB checks to see if the professor has a section with the same name. 3. If the section-name entered is unique, add the record, otherwise section creation fails. 4. SectionDB adds the section information. 5. SectionDB links the class and section.

Use Case Name	Add Students to Section
Preconditions	A section is selected, CSV file (optional).
Successful End Condition	A student is added to the section.
Failed End Condition	A student is not added to the section.
Primary Actors	Professor
Secondary Actors	SectionDB, StudentDB
Main Flow	<ol style="list-style-type: none"> 1. Parse student information from csv, or from manual input. 2. Add the students to the section. 3. StudentDB adds the student information. 4. SectionDB links the students and the section.

Use Case Name	Remove Student From Section
Preconditions	A section is selected, a student is selected.
Successful End Condition	A student is removed from the Professor's view.
Failed End Condition	A student is not removed.
Primary Actors	Professor
Secondary Actors	SectionDB, StudentDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to remove the student. 2. Professor is shown a warning message. 3. Professor confirms deletion. 4. SectionDB removes the link between the section and the student.

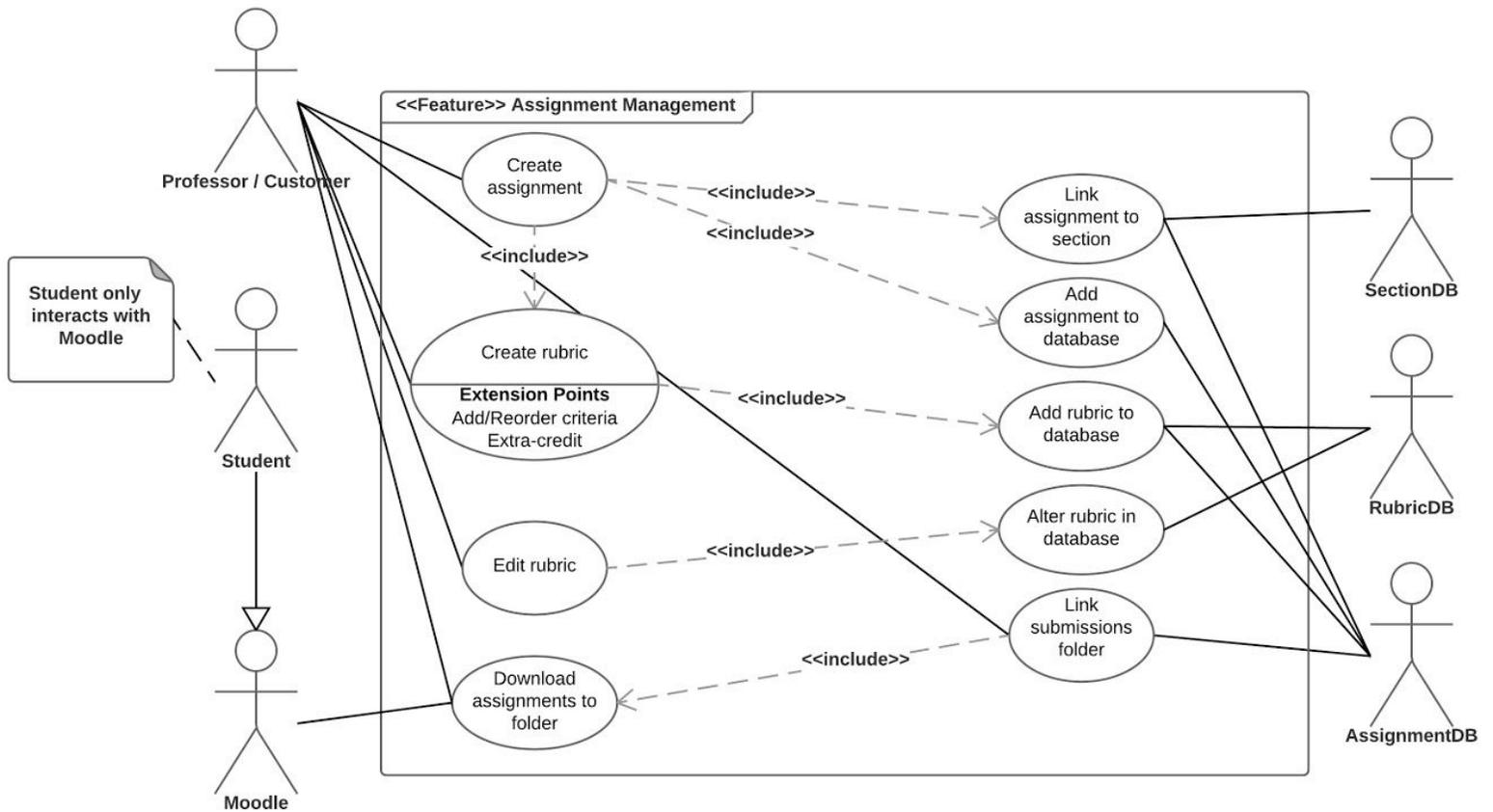
Use Case Name	Edit Section Name
Preconditions	The section exists.
Successful End Condition	The section's name is changed.
Failed End Condition	The section's name is not changed.

Primary Actors	Professor
Secondary Actors	Section DB
Main Flow	<ol style="list-style-type: none"> 1. Professor enters the desired section name. 2. Professor confirms information. 3. Section DB modifies the section's name.

Use Case Name	Delete Section
Preconditions	The section exists.
Successful End Condition	The section is removed from the Professor's view.
Failed End Condition	The section is not removed.
Primary Actors	Professor
Secondary Actors	SectionDB, ClassDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to remove the section. 2. Professor is shown a warning message. 3. Professor confirms deletion. 4. Class DB removes the link between section and class.

Use Case Name	Move To Assignment View
Preconditions	The assignment exists.
Successful End Condition	Moved to Assignment View.
Failed End Condition	Stay on Section View
Primary Actors	Professor
Main Flow	<ol style="list-style-type: none"> 1. Professor clicks on the assignment. 2. ViewManager loads the assignment information. 3. ViewManager displays the assignment information.

Assignment Management



Use Case Name	Create Assignment
Successful End Condition	An assignment is created.
Failed End Condition	An assignment is not created.
Primary Actors	Professor
Secondary Actors	AssignmentDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to create an assignment. 2. Professor selects an existing rubric or creates one. 3. Professor writes the objective and name of assignment. 4. AssignmentDB stores the relevant information.

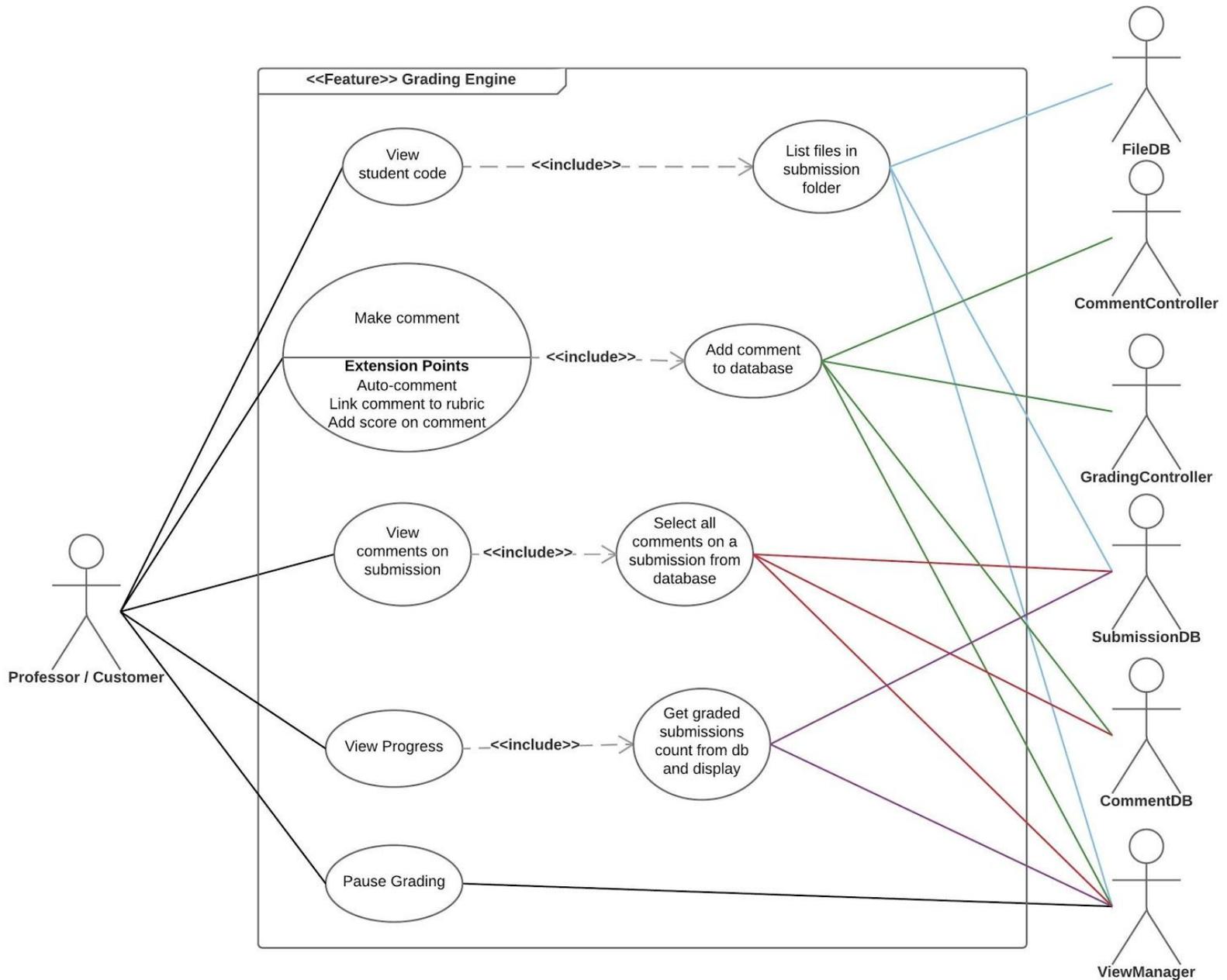
Use Case Name	Point Assignment to Folder
Preconditions	Assignment exists and is linked to section. Professor downloaded assignments from Moodle to a local folder.
Successful End Condition	The assignment is associated with a folder of submissions.
Failed End Condition	The assignment is not associated with a folder.
Primary Actors	Professor
Secondary Actors	AssignmentDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to point a folder to an assignment. 2. Professor drags the folder into drag/drop widget, or browses using a file browser. 3. AssignmentDB stores the location for the given assignment.

Use Case Name	Create Rubric
Successful End Condition	A rubric is created.
Failed End Condition	A rubric is not created.
Primary Actors	Professor
Secondary Actors	RubricDB, AssignmentDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to create a rubric. 2. Professor starts typing criteria. 3. If a criterion already exists, it will be displayed to the professor to be selected. 4. Otherwise the professor creates a new criterion. 5. Professor positions criterion within the hierarchy (is it a category ex. Completeness? Does it have a parent category?) 6. RubricDB stores the criterion 7. Professor enters the maximum total pts for each rubric category. 8. Professor adds extra-credit (optional) 9. Professor confirms the rubric as complete. 10. RubricDB stores the rubric information. 11. Rubric gets linked to assignment in AssignmentDB

Use Case Name	Edit Rubric
Successful End Condition	A rubric is edited.
Failed End Condition	A rubric stays the same.
Primary Actors	Professor
Secondary Actors	RubricDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to edit a rubric. 2. Professor edits / rearranges criteria. 3. Professor creates new criteria (optional) 4. Professor adds/removes extra-credit (optional) 5. RubricDB stores the criterion 6. Professor enters the grading total for the rubric and individual criteria. 7. Professor confirms the rubric as complete. 8. RubricDB stores the rubric information.

Use Case Name	Move to Grading View
Preconditions	A submission is selected.
Successful End Condition	Moved to Grading View.
Failed End Condition	Not Moved to Grading View
Primary Actors	Professor
Secondary Actors	View Manager. File DB, Submission DB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to start grading. 2. Professor selects anonymous, or individual grading. 3. File DB and Submission DB load submission and file data into ViewManager. 4. ViewManager moves to Grading View.

Grading Engine



Use Case Name	View files (code) in student submission
Preconditions	Submission exists. Files exist.
Successful End Condition	Display a file in full view and a list of other files.
Failed End Condition	No files to display
Primary Actors	Professor
Secondary Actors	ViewManager, FileDB, SubmissionDB
Main Flow	<ol style="list-style-type: none"> 1. List all files inside submission folder. 2. Professor wants to open a file in submission. 3. Professor clicks on file from sidebar. 4. ViewManager moves to the new file.

Use Case Name	Make Comment In File
Preconditions	The file exists.
Successful End Condition	A comment is made.
Failed End Condition	A comment is not made.
Primary Actors	Professor
Secondary Actors	Comment DB, CommentController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to comment on a specific area of text. 2. Professor highlights the text from start to end. 3. ViewManager opens a textbox. 4. Professor types the comment. 5. Comment DB and Comment Controller store it.

Use Case Name	Auto Complete a Comment
Preconditions	The file exists, a comment is being typed.
Successful End Condition	A comment is made.
Failed End Condition	A comment is not made.
Primary Actors	Professor

Secondary Actors	CommentDB, CommentController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to comment on a specific area of text. 2. Professor highlights the text from start to end. 3. ViewManager opens a textbox. 4. Professor types the comment. include::Make Comment 5. CommentController searches for auto completions. 6. Professor selects auto completion and finishes the comment. 7. Comment DB and Comment Controller store it.

Use Case Name	Link a Comment to Rubric
Preconditions	Comment exists, or is being typed, file exists, rubric exists.
Successful End Condition	Comment is linked to Rubric.
Failed End Conditions	Comment is not linked to Rubric.
Primary Actors	Professor
Secondary Actors	CommentDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to link a comment on a specific area of text to a rubric. 2. Professor selects the comment. 3. ViewManager displays rubric criteria. 4. Professor selects rubric criterion. 5. Comment DB links comment to rubric.

Use Case Name	Add a Score to a Comment with Rubric
Preconditions	Comment linked with rubric exists.
Successful End Condition	Comment linked with grade.
Failed End Condition	Comment not linked with grade.
Primary Actors	Professor
Secondary Actors	GradingController, CommentDB

Main Flow	<ol style="list-style-type: none"> 1. Professor wants to add a grade value to a comment. 2. Professor selects a comment. 3. ViewManager displays a field to add a value as a grade. 4. Professor enters score. 5. GradingController links the comment with the score. 6. CommentDB stores the link between comment and score.
-----------	---

Use Case Name	View All Comments on Submission
Preconditions	Comments exist.
Successful End Condition	Comments are viewed in bulk.
Failed End Condition	Comments are not viewed.
Primary Actors	Professor
Secondary Actors	ViewManager, CommentController, SubmissionDB
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to view all of the comments on a submission. 2. CommentController passes all comments to ViewManager to display. 3. ViewManager displays all comments.

Use Case Name	View Progress in Batch
Preconditions	Grading has started.
Successful End Condition	Professor sees a progress bar.
Failed End Condition	Professor doesn't see a progress bar.
Primary Actors	Professor
Secondary Actors	ViewManager, ClassController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to see how far in the batch they have gone. 2. ClassController counts the number of unfinished and finished submissions in the project. 3. ViewManager displays the information.

Use Case Name	Pause Grading
Preconditions	Grading session has started.
Successful End Condition	All progress saved.
Failed End Condition	All progress could not be saved.
Primary Actors	Professor
Secondary Actors	GradingController, CommentController, ViewManager
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to take a break from grading. 2. Controllers send their contents to the DB to be stored. 3. ViewManager moves display away from the grading view.

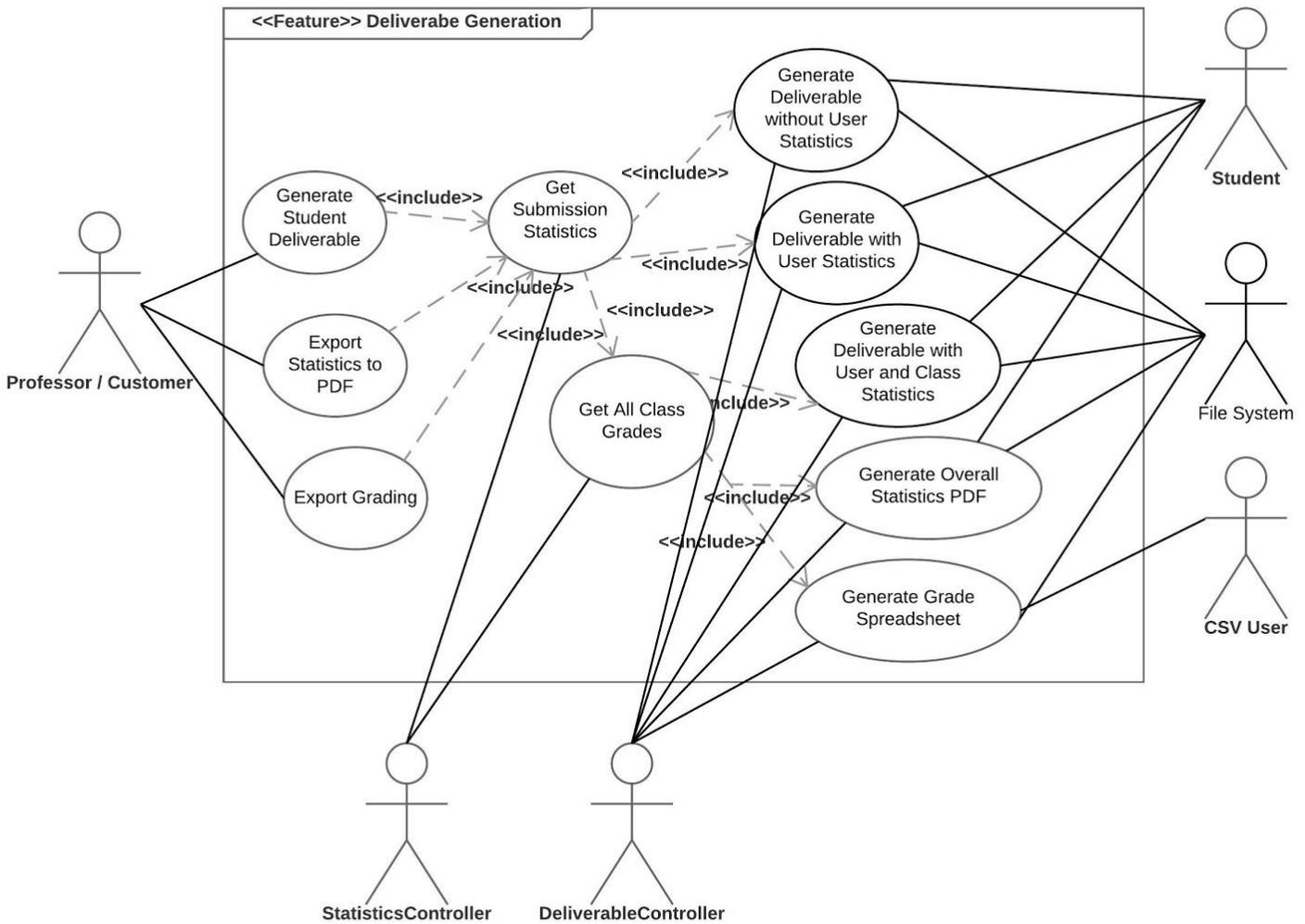
Overview/Submission Report

Use Case Name	View Statistics on Comments, Rubric and Grades
Preconditions	Some grades have been made.
Successful End Condition	Professor can view statistics regarding grades, comments and rubric.
Failed End Condition	Professor can't view statistics.
Primary Actors	Professor
Secondary Actors	ViewManager, StatisticsController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to view statistics regarding comments and rubrics. 2. StatisticsController calculates a set of statistics. 3. ViewManager displays the statistics.

Use Case Name	Amend/Override Grade for Rubric Criterion
Preconditions	Some grades have been made.
Successful End Condition	The grade is amended.
Failed End Condition	The grade is not amended.
Primary Actors	Professor
Secondary Actors	ViewManager, StatisticsController, GradingController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to see an estimated grade and amend it if necessary. 2. StatisticsController calculates a set of statistics. 3. ViewManager displays that set of statistics. 4. Professor amends a grade. 5. GradingController stores the grade. 6. Update the StatisticsController.

Use Case Name	Mark a Submission as Graded (Complete)
Preconditions	A submission was in overview.
Successful End Condition	The submission is marked as complete.
Failed End Condition	The submission is not marked complete, and professor is redirected back to grading view.
Primary Actors	Professor
Secondary Actors	ViewManager, GradingController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to mark a submission completely graded. 2. GradingController marks the submission as graded. 3. ViewManager congratulates the Professor for grading another assignment. 4. Depending on the mode, the ViewManager switches back to the GradingView or submission list.

Student Deliverable / Gradebook



Use Case Name	Generate Deliverable
---------------	----------------------

Preconditions	The submission was marked as complete.
Successful End Condition	The deliverable was generated.
Failed End Condition	The deliverable was not generated.
Primary Actors	Professor
Secondary Actors	ViewManager, StatisticsController, DeliverableController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to generated a deliverable for a submission. 2. StatisticsController loads submission information. 3. DeliverableController formats comments with context to add to the preview. 4. Professor customizes information in the deliverable from a list of options. 5. ViewManager displays a preview of the deliverable. 6. Professor confirms to print the deliverable. 7. DeliverableController formats the information into an exportable pdf and saves it in a specified location.

Use Case Name	Export to Spreadsheet
Preconditions	A submission has been marked complete.
Successful End Condition	The spreadsheet is generated.
Failed End Condition	The spreadsheet is not generated.
Primary Actors	Professor
Secondary Actors	ViewManager, StatisticsController, DeliverableController
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to export grading data to a spreadsheet/csv. 2. StatisticsController loads submission information. 3. ViewManager displays a preview of the spreadsheet. 4. Professor confirms to export the spreadsheet. 5. DeliverableController formats the information into an exportable csv/spreadsheet compatible format and saves it in a specified location.

Data Archiving

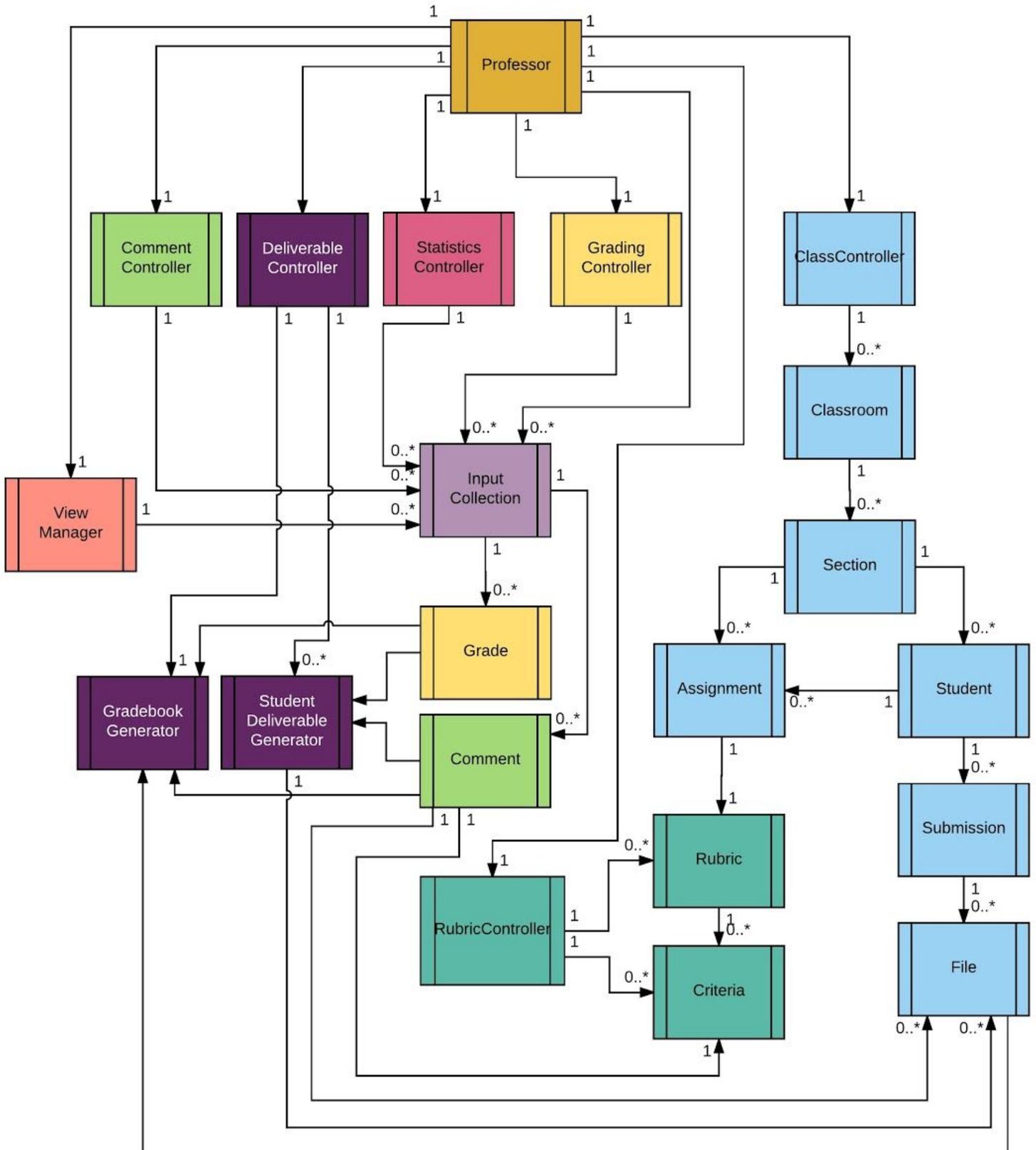
Use Case Name	Commit Archive
Preconditions	User has progressed in grading and has a setup archive.
Successful End Condition	SQL Dump is saved.
Failed End Condition	SQL Dump is not saved.
Primary Actors	Professor
Secondary Actors	GitModule, Database
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to commit progress. 2. Database creates a dump of the database. 3. Data folder and dump are sent to GitModule for Git commit. 4. Git commit is completed.

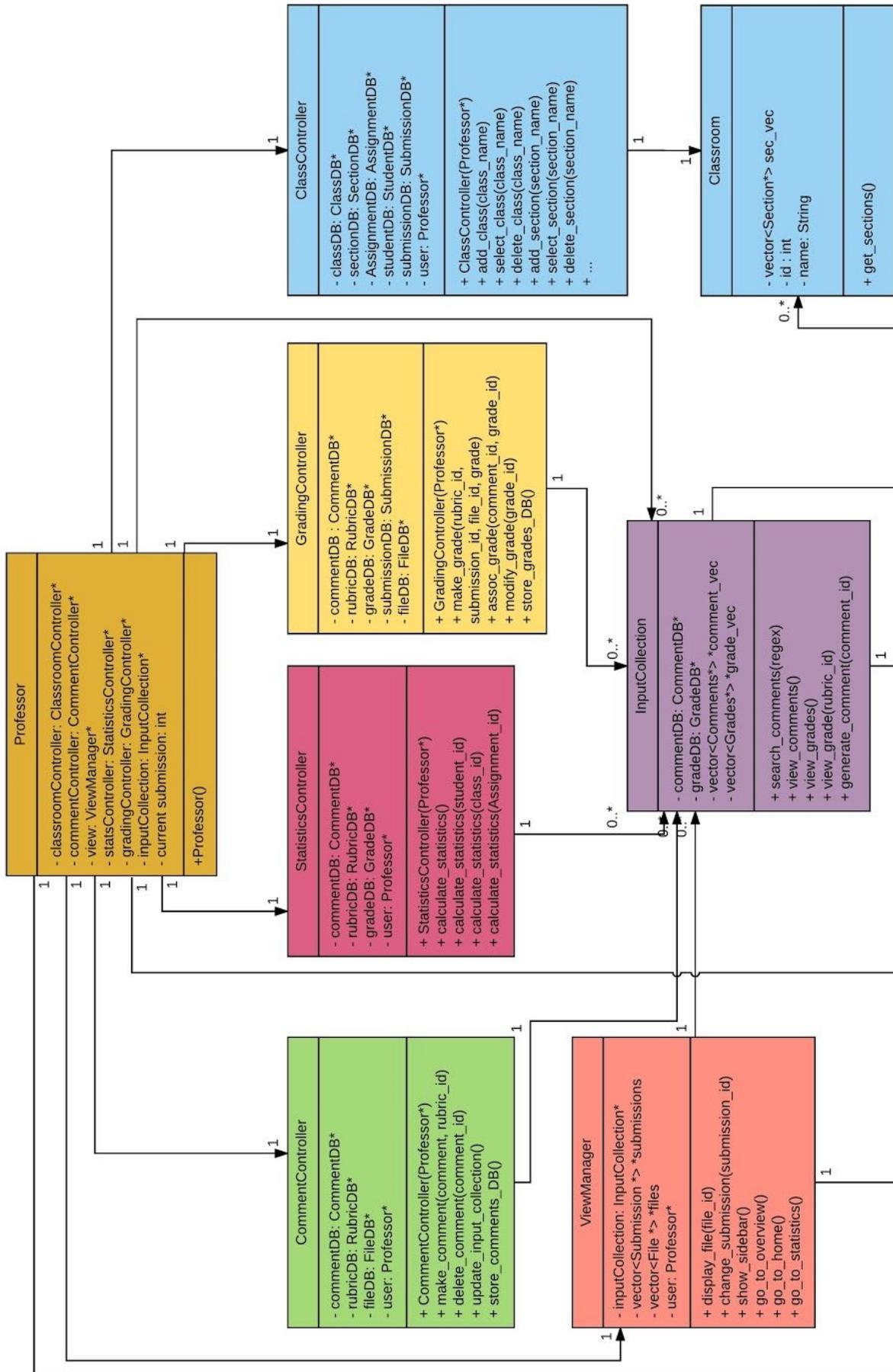
Use Case Name	Pull From Archive
Preconditions	User has a setup archive and is online.
Successful End Condition	Database is up to date with archive.
Failed End Condition	Database is not up to date with archive.
Primary Actors	Professor
Secondary Actors	GitModule, Database
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to pull from archive. 2. GitModule receives request. 3. GitModule pulls the latest dump from origin. 4. GitModule performs combination algorithm with current database and origin dump.

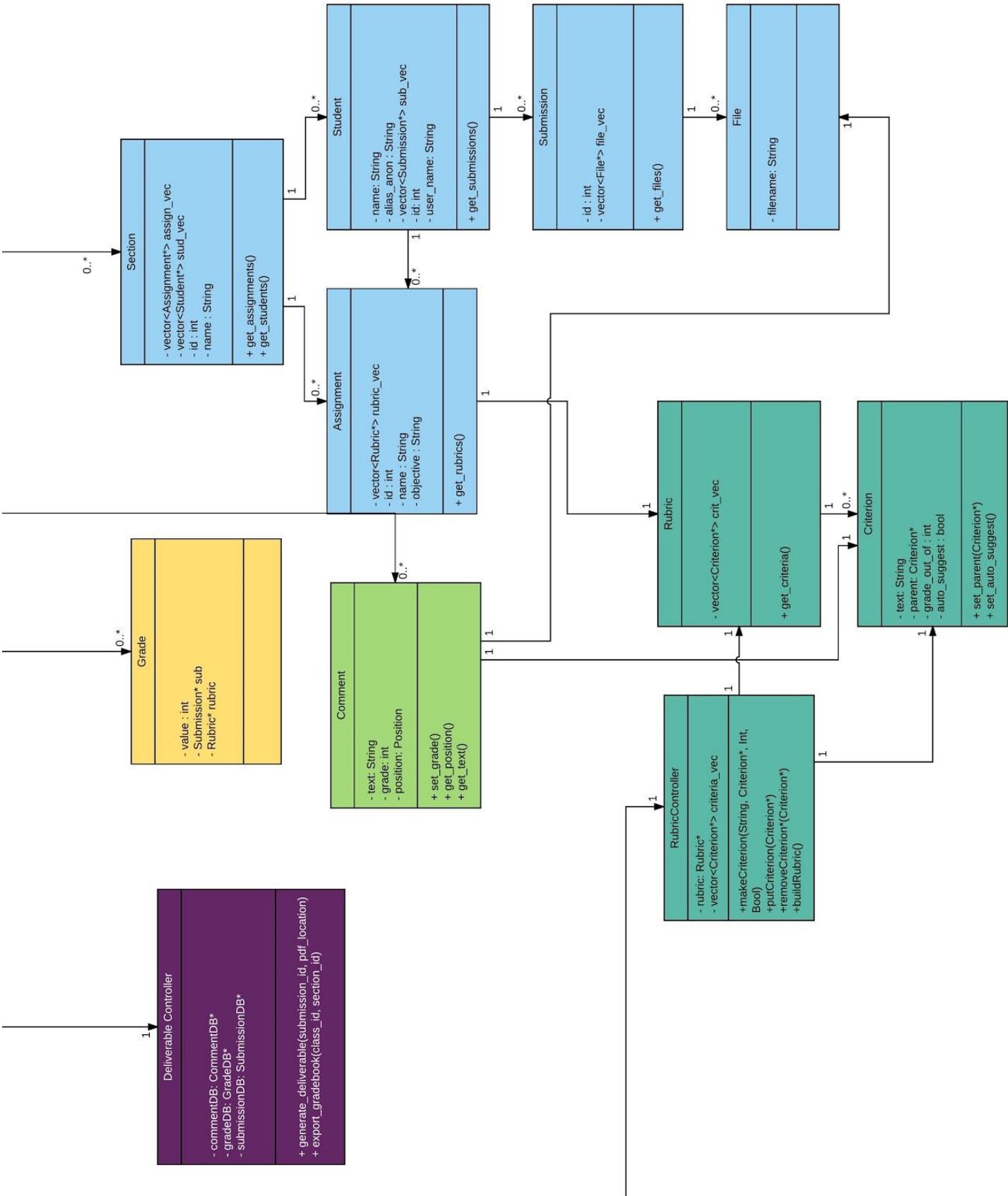
	<ol style="list-style-type: none"> 5. Database is now up to date with origin. 6. Git pull is completed.
--	---

Use Case Name	Push to Archive
Preconditions	User has a setup archive and is online.
Successful End Condition	Origin updates its dump with current dump.
Failed End Condition	Origin does not update.
Primary Actors	Professor
Secondary Actors	GitModule, Database
Main Flow	<ol style="list-style-type: none"> 1. Professor wants to push archive. 2. GitModule receives request. 3. GitModule pulls the latest dump from origin. 4. GitModule performs combination algorithm. 5. GitModule uses Git commit on merged SQL dump. 6. Git push is completed.

IX . Logic View







X . Final Database Design

Table: Course

Course_ID (PRIMARY_KEY)	name	semester
INTEGER	VARCHAR(50)	VARCHAR(50)

Table: Section

Section_ID (PRIMARY_KEY)	Course_ID (FOREIGN KEY)	name
INTEGER	INTEGER	VARCHAR(50)

Table: Student

Student_ID (PRIMARY_KEY)	Section_ID (FOREIGN KEY)	name
INTEGER	INTEGER	VARCHAR(50)

Table: Assignment

Assignment_ID (PRIMARY_KEY)	name	objective	full_grade
INTEGER	VARCHAR(50)	TEXT	ENUM (yes/no)

Table: Assignment_Section (links an assignment to multiple sections)

AS_ID (PRIMARY_KEY)	Assignment_ID (FOREIGN KEY)	Section_ID (FOREIGN KEY)
INTEGER	INTEGER	INTEGER

Table: Submission

Submission_ID (PRIMARY_KEY)	Student_ID (FOREIGN KEY)	Assignment_ID (FOREIGN KEY)	General_Comment	status
INTEGER	INTEGER	INTEGER	TEXT	ENUM("not started", "in progress", "graded")

Table: Comment

Comment_ID (PRIMARY_KEY)	Filename	Rubric_ID (FOREIGN_KEY)	Text	grade	start_pos	end_pos
INTEGER	VARCHAR(50)	INTEGER	TEXT	INT	INT	INT

Table: Rubric

Rubric_ID PRIMARY_KEY	Rubric_Text	Assignment_ID FOREIGN_KEY	parent FOREIGN_KEY	grade_out_of	extra_credit
INTEGER	TEXT	INTEGER	INTEGER	INTEGER	ENUM (y/n)

Table: Grades

Grade_ID (PRIMARY_KEY)	Rubric_ID (FOREIGN_KEY)	Submission_ID (FOREIGN KEY)	score
INTEGER	INTEGER	INTEGER	INTEGER

XI . Final User Interface Design

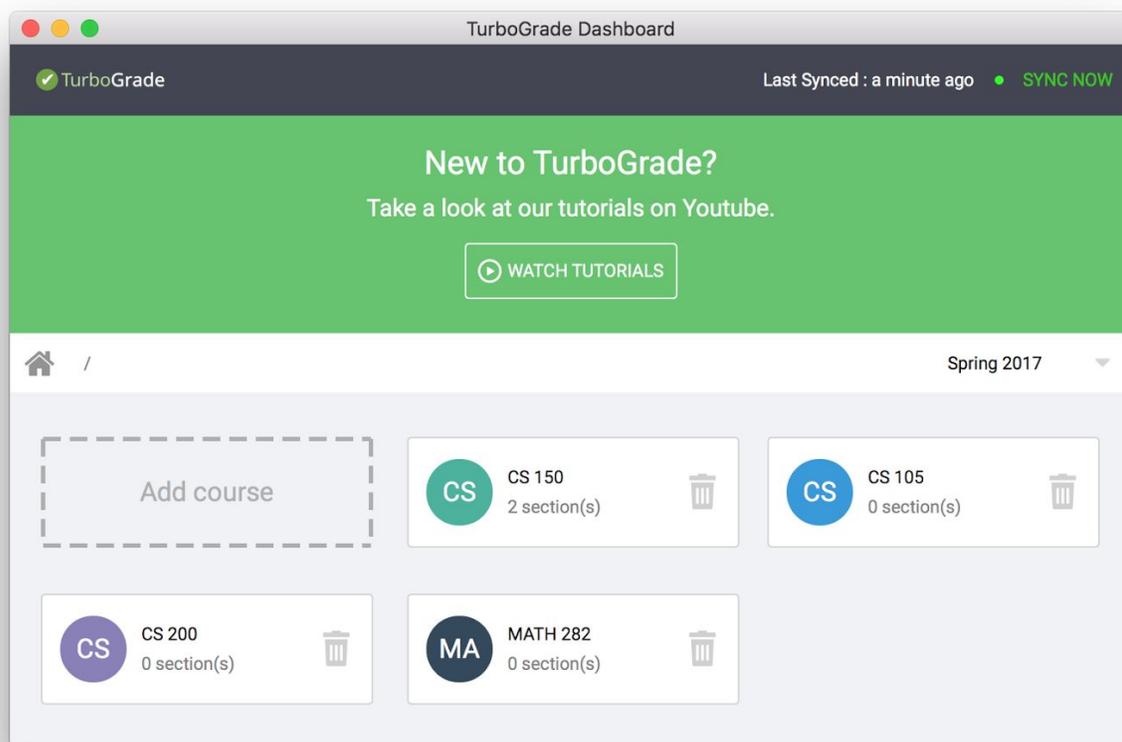


Figure 2a. The Dashboard showing the “Course View”

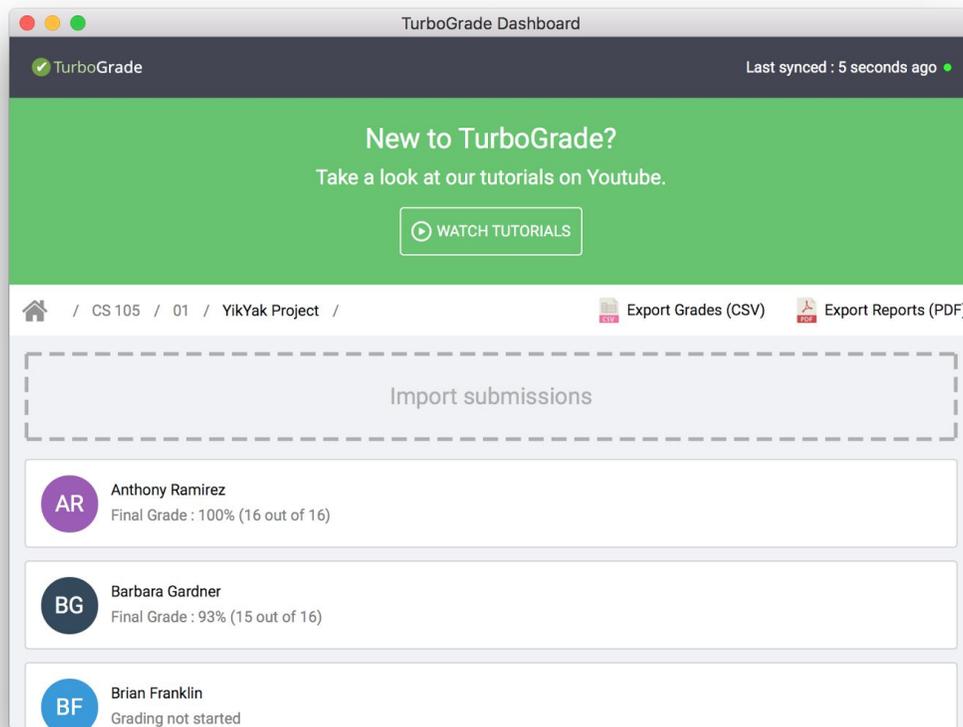


Figure 2a. The Dashboard showing the “Submissions View” for a specific section and assignment

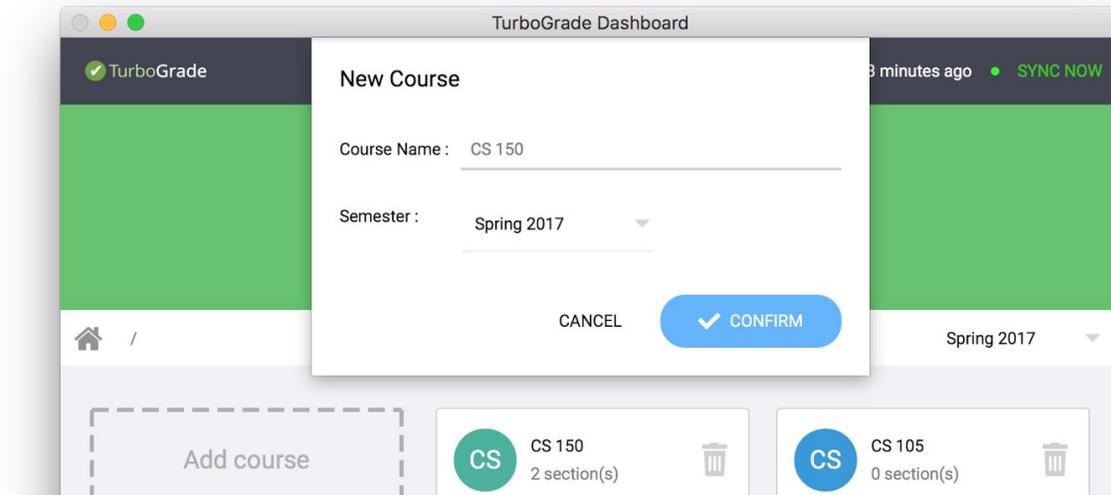


Figure 3. “Add Course Dialog” (similar view for sections, assignments and students)

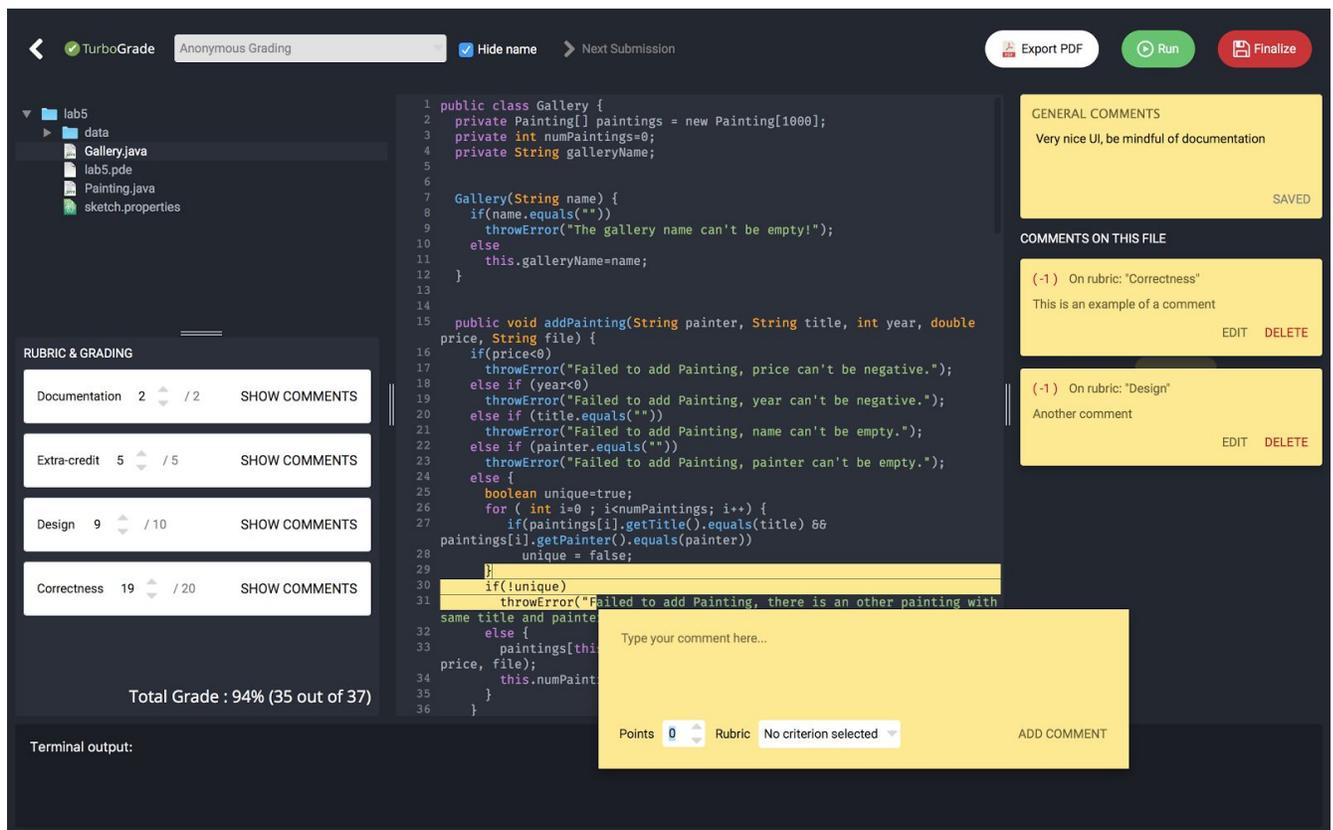


Figure 4. “Grading View” Implementation

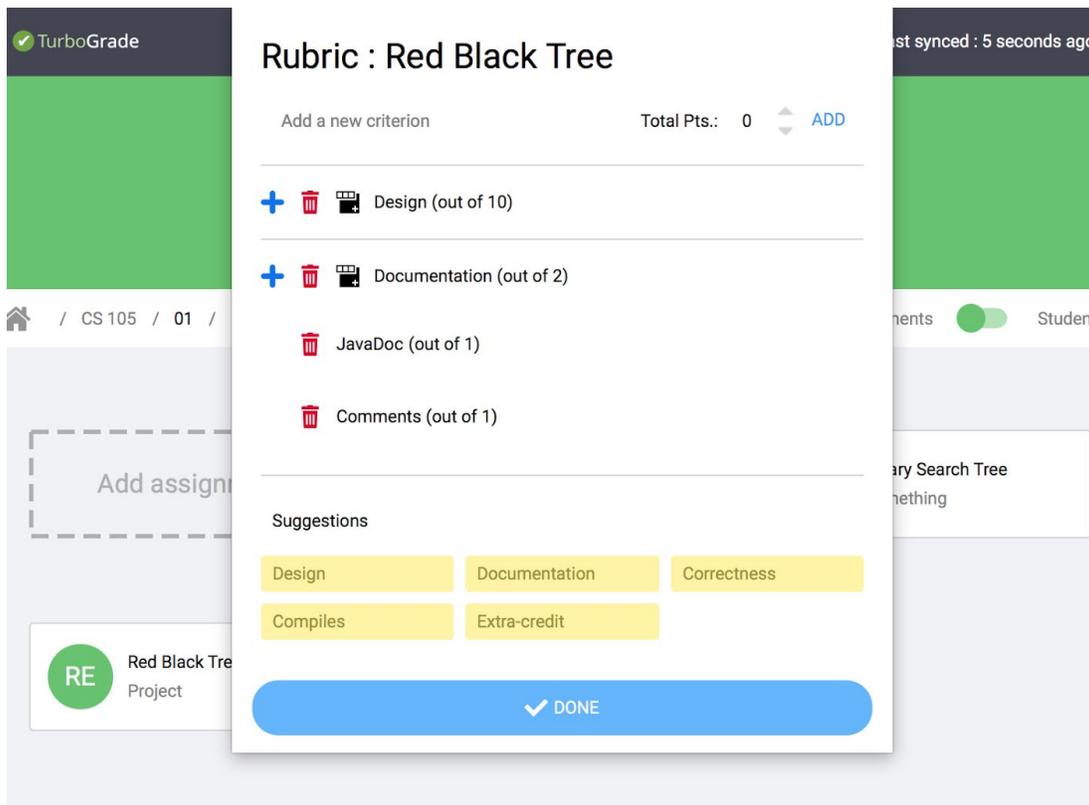


Figure 5. Rubric Maker



Student
Wassim Gharbi

TOTAL GRADE

87 %

Assignment: YikYak Project

Objective:

This project will help you work with raw data

GENERAL COMMENTS

You did well on this assignment, beware of commenting and documentation.

Grade Breakdown

Design 8 /10

Correctness 5 /20

Visualizations : 5 out of 5

Documentation 1 /1

○ Design 9 / 10

Remarks:

On "lab5/Gallery.java" for "Design"

```
1  throwError("Failed to add Painting, price can't be negative.");
2  else if (year<0)
3  throwError("Failed to add Painting, year can't be negative.");
4  else if (title.equals(""))
5  throwError("Failed to add Painting, name can't be empty.");
6  else if (painter.equals(""))
7  throwError("Failed to add Painting, painter can't be empty.");
8  else {
9  boolean unique=true;
```

-1 Input control should be in painting

○ Documentation 2 / 2

Remarks:

On "lab5/Painting.java" for "Comments"

```
1  public Painting(String painter, String title, int year, double price, String file)
   {
```

Figure 6. Student Deliverable PDF

XII. Physical View

There are two basic end systems in the TurboGrade package: the client device and the archiving machine. In our document, we depict the common case that the archiving machine as external to the machine running the TurboGrade software, although this is not necessary.

Given that the package is meant to act as both a standalone offline client and a synchronized online component, most of the pieces of the package lie within the machine running TurboGrade. The TurboGrade machine will have the student submissions, a unique identifier, its own local database and archive, and finally, the TurboGrade executable.

Most notably, the archive machine holds the unique identifier tracker responsible for providing uniqueness across all clients accessing the archive machine. This comes into play in section XIV Data Archival, but in short, by controlling the access to this archive machine, we ensure uniqueness. Other than this file, the archive machine will hold only the archived state of the TurboGrade package.

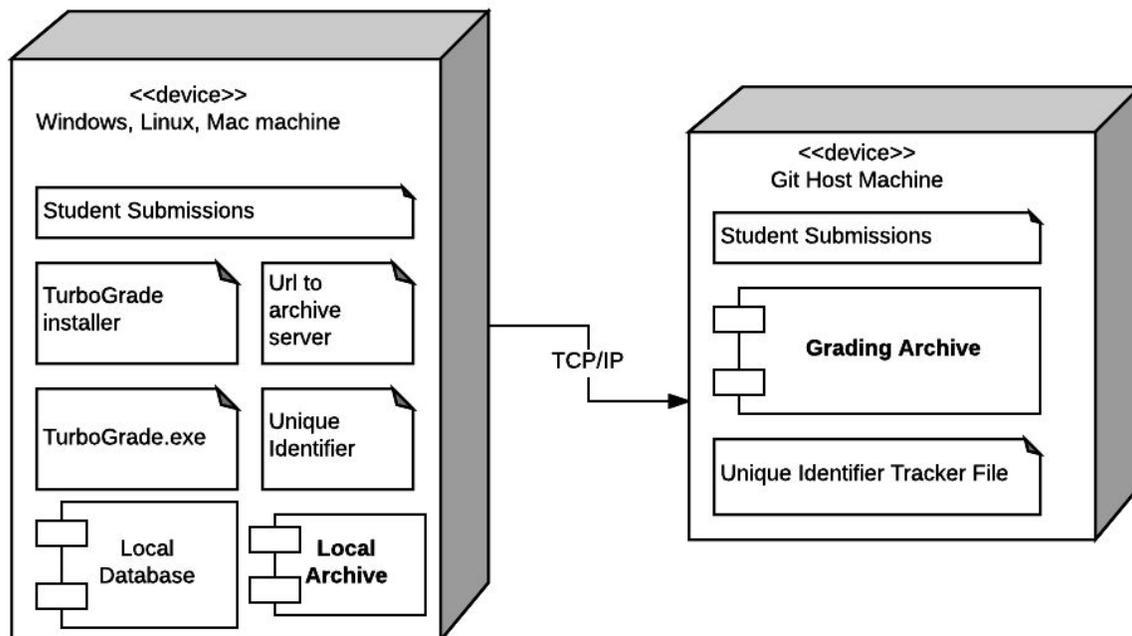


Figure 7. Physical view of the nodes for TurboGrade

Another aspect of the physical view would be how TurboGrade is meant to add students. Currently, TurboGrade uses a system that requires a Moodle Submission as per project specifications. This directory is another component in the physical view. Once a set of moodle submissions have been imported, they get copied to the /sync/data folder. During this process, any students who are not in the system are created, and their submissions added accordingly.

To summarize the meat of the program, see Figure 8 below where we illustrate the heart of the software. In essence, this software is meant to help instructors provide feedback and grades on submissions, according to a rubric, and finally generate deliverables to the student and the professor. By adding the coupling and links between the pieces of the heart of the software, the development of the software becomes transparent in regards to where the resources are located and how they have to come together.

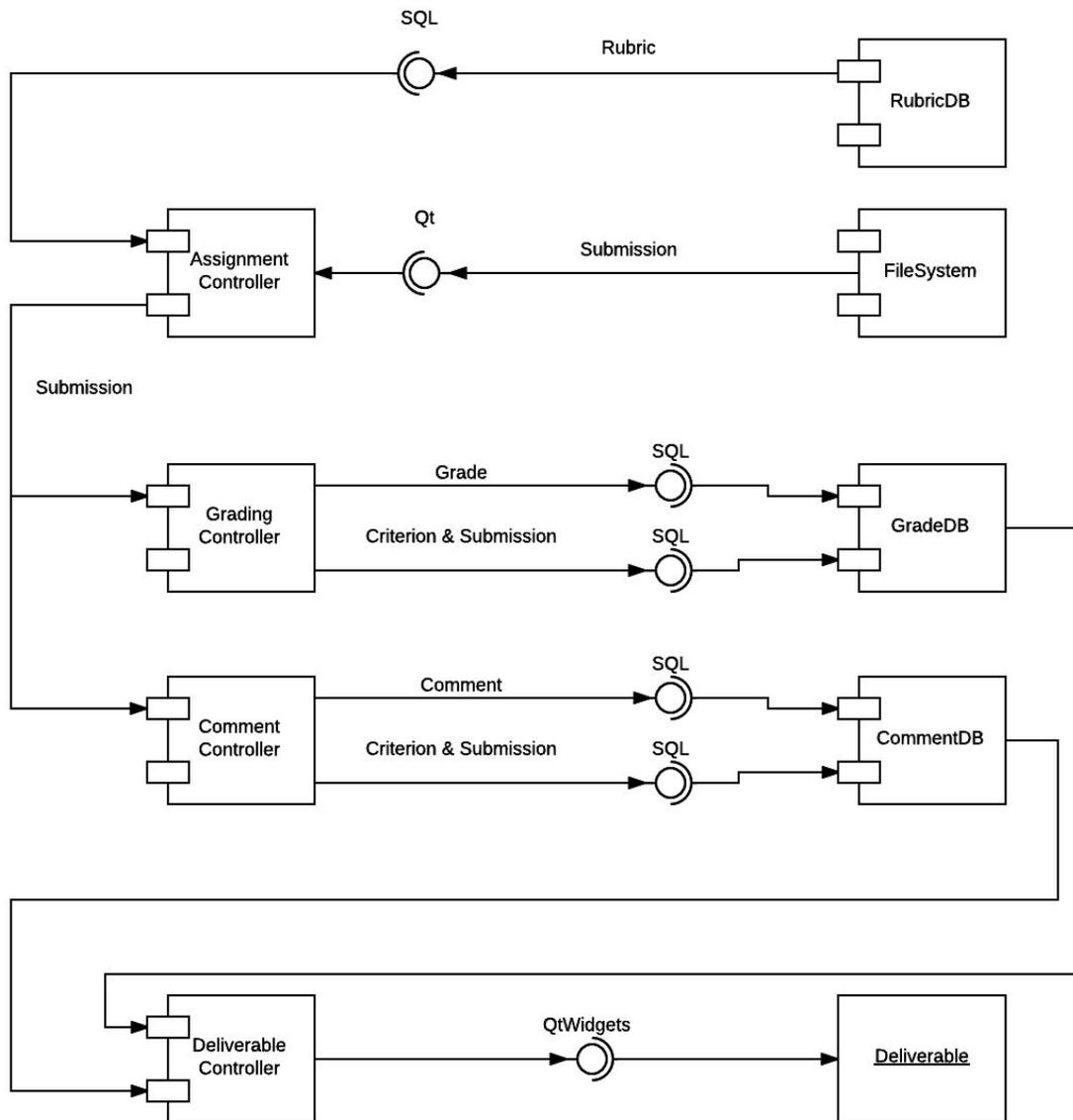


Figure 8. The development view of the software.

XIII Installation Process

The installation of TurboGrade is simple and lightweight. TurboGrade relies on the use of Qt's Libraries, SQLite and Git. We use the Qt Installer Framework to create the installer file for every supported platform. The Qt Installer Framework compiles all the required libraries and files into one executable installer. *Figure 9* shows a breakdown of what the user can expect to be installed in their system on a Windows platform.

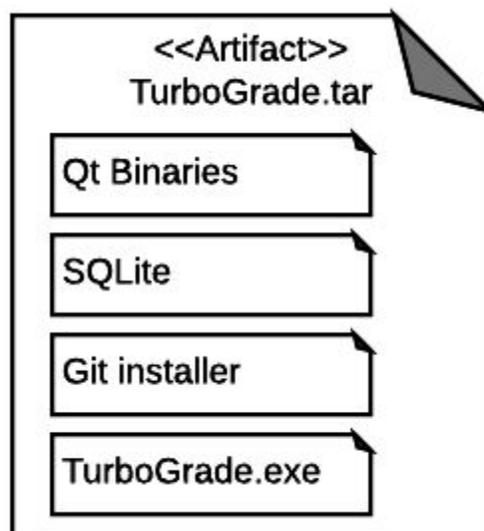


Figure 9. Breakdown of TurboGrade.tar

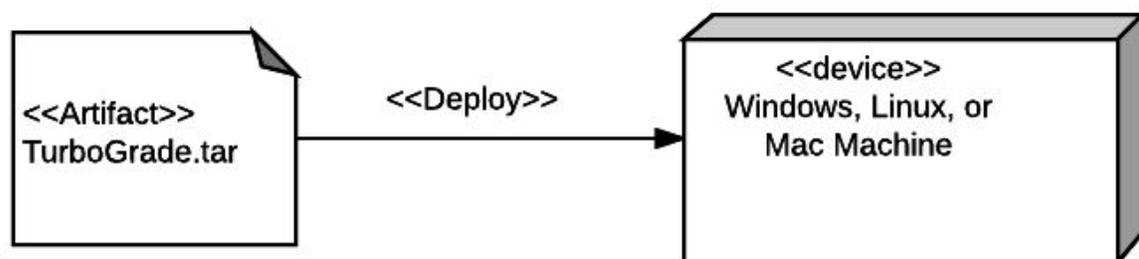


Figure 10. Deployment of TurboGrade

The installation process is detailed in the user manual.

XIV. Data Archival & Git Synchronization

As a grader, it might be nice to be able to offload the work to teaching assistants, work on different machines, or take the work into an offline setting, all the while maintaining an organized archive of grading. In the TurboGrade product, we hope to provide support for doing so, and have begun investigating and designing a system for doing so. First, let's look at the many complications that arise with the implementation of data archival:

- Interface Complication
 - How can we provide a smooth interface between the user, the TurboGrade program, and a git repository?
 - Provide easy to use buttons on the UI
 - Can we find a way to create momento's of the system and reconcile those properly against multiple versions?
 - Use SQL Dumps to represent the user's state
- Offline Synchronization
 - How can we do our best to make sure that when the user returns to using the software while connected to the internet, that the user's data will get pushed to repository correctly?
 - Allow the user to store their progress locally, and then push it when they go online.
- Conflicting Changes
 - If the user is using multiple machines or multiple users are interacting with the same set of submissions to grade, how can we minimize conflict as to maintain data integrity?
 - Provide a unique identifier to each machine that has pulled from the archiving system.

Hoping to cover these bases, we designed a data archival system that attaches to the already existing logical structure. To start, we decided to utilize the version control utilities provided by Git. With Git, we are confident that by providing the user access to repositories through our interface, we can actively maintain data integrity and minimize problems from the above complications, with minimal effort.

The user can interact with the archiving system through 3 major actions: **commit**, **push**, and **pull**.

- **Commit**: The current database state is saved via an sql dump, and committed using Git's traditional commit command.
- **Pull**: The sql dump on the origin repo is pulled using Git's pull command, and unpacked into the current database's state.
- **Push**: The most recent commit and origin dumps are combined, and the merged dump becomes the dump of the origin

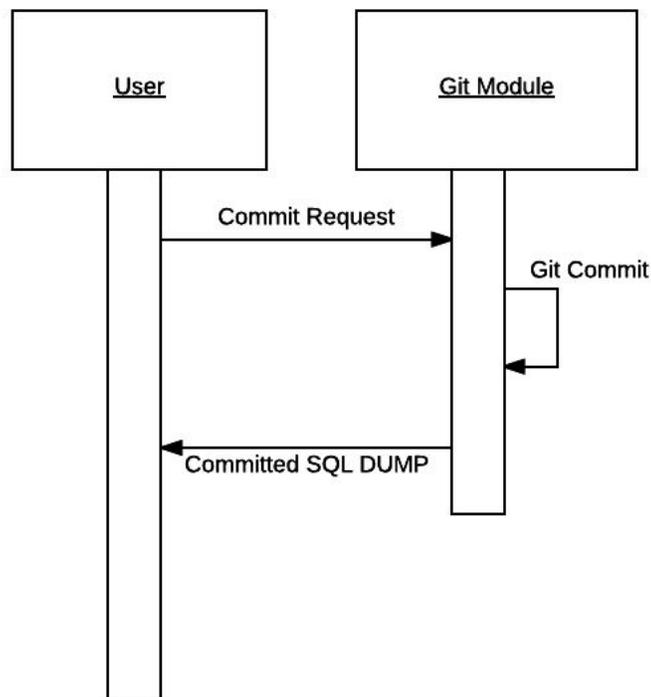


Figure 11. Timing Sequence for committing to archive.

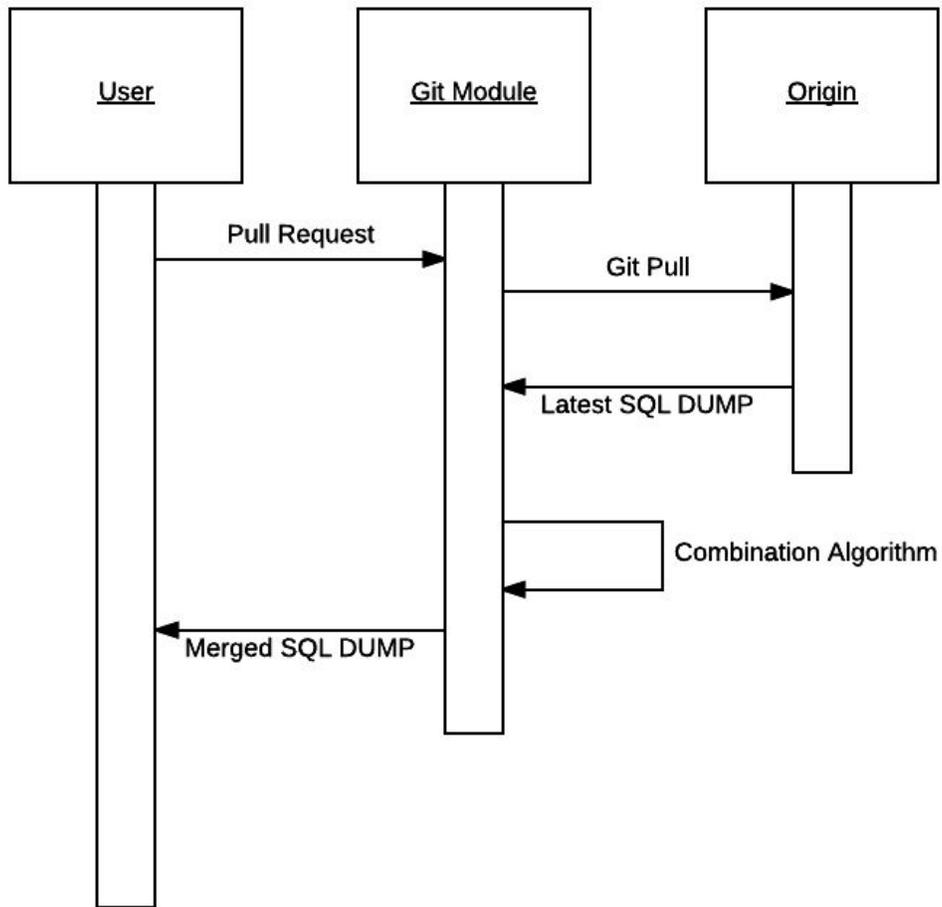


Figure 12. Timing Sequence for pulling archive.

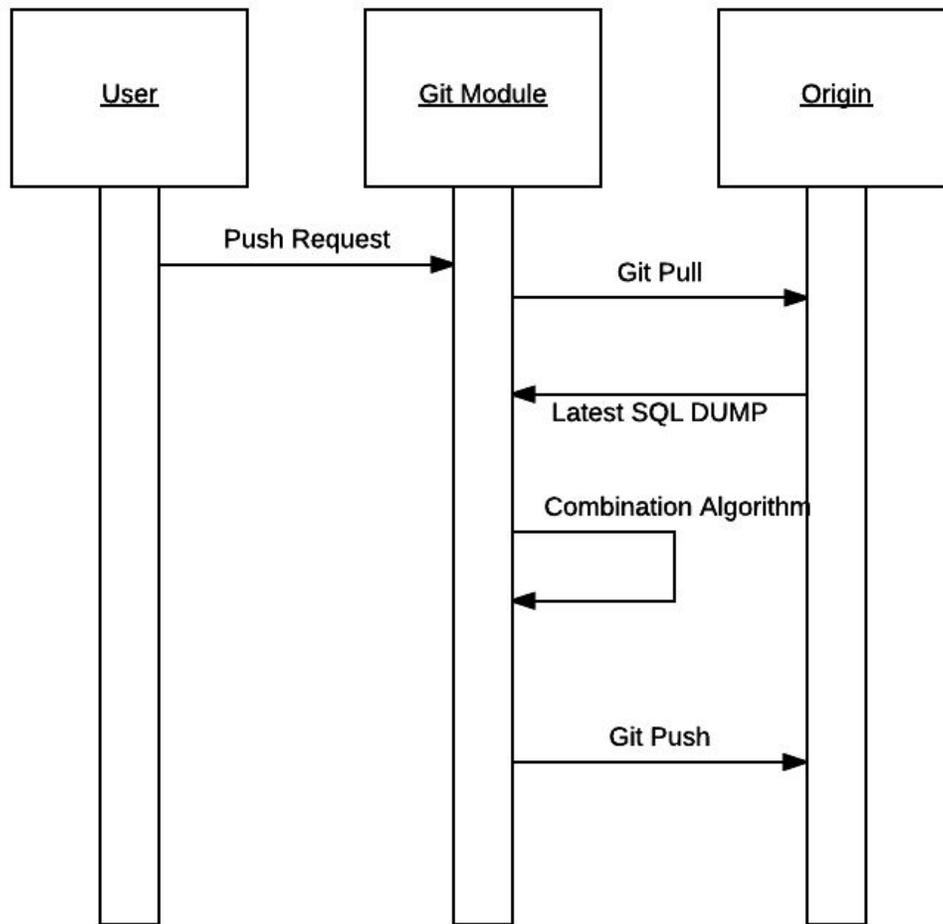


Figure 13. Timing Sequence for pushing to archive.

In its current state, TurboGrade is capable of archiving data across multiple systems. Any changes the user makes in TurboGrade are able to be committed via Git. This includes all of the classroom components and any grading process on submissions. All it requires is that the user be connected to the internet and able to access the Git repository they provided.

Conflicts in grading in TurboGrade is a void in our domain. We have no way to prioritize local changes or global changes upon conflict. This gets the user at least the option to begin grading on one machine, and later continue on another. For small colleges where grading is not expedited to multiple graders, this would not be an issue. We would have liked to handle this by creating global uniqueness through UUID keys in the database and having the option to decide what changes to keep through the form of a DIFF.

In the manner of long-term archiving, TurboGrade also has the potential to be used as a form of grade storage, however, we have not implemented such a mechanism. Long-term grade storage would be

an important feature to increase the longevity of the product's lifetime. While, we do not have the feature directly, the users have the tools to implement a system themselves by manipulating the Git URL. They could grade assignments on one repository, and when finished, change the URL to the permanent storage repository and push the changes there. This subsystem is quite primitive, but given the time constraints on the project, this is what we managed to implement.

XV. Testing Plan

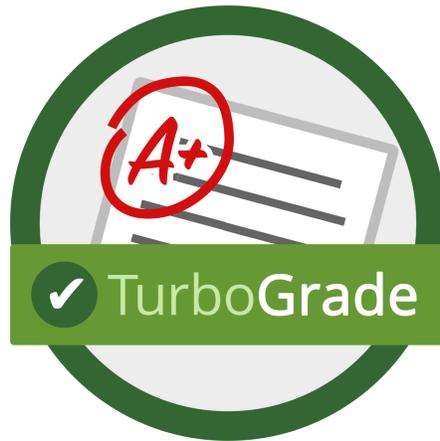
We have fully tested the software through the four common software testing mechanisms : Unit Testing, Integration Testing, System Testing and finally Acceptance Testing as follows :

1. We performed Unit Testing on every new feature each of the group members implements.
2. As we merged and combined the various components/features, we tested the combined units as a whole through Integration Testing (this occurred once a week at the same time we merged our branches).
3. We performed a preliminary Acceptance Test with professor Sadovnik during our demo meeting.
4. Prior to code-freeze we performed a thorough System Test to make sure the software works as intended and complied with the client's requirements

XVI. Reflections

Through our work on TurboGrade, we have learned that creating a detailed and well-thought design helps a lot in the implementation. By creating a strong foundation for the project, we did not have to rethink our vision or go back and forth between design and implementation. Since the beginning we had a solid plan, a good database design and a well-thought engine structure. This made the development process very easy which let us focus on improving our user interface and interactivity as well as create additional features that we thought would be helpful for the professor.

TurboGradeTM



User Manual

Introduction	3
Getting started	4
Installing TurboGrade™	4
Configuring TurboGrade™	6
Uninstalling TurboGrade™	7
Dashboard Overview	8
Grading UI Overview	9
Managing a class	11
Managing courses	11
Managing semesters	13
Managing students	14
Managing assignments	15
Anatomy of an assignment	15
Rubric Maker	16
Assignments for multiple sections	17
Grading a submission	17
Student Deliverables & Gradebook	18
User Interface	18
Example Student Report	19
CSV Gradebook	21
Archiving & Git Synchronization	22

I. Introduction

Welcome to **TurboGrade™**, the easy to use grading software for Computer Science courses. **TurboGrade™** is a powerful tool that will help you grade assignments easily and intuitively. Using the built-in **TurboGrade™** features, you can achieve a lot. Here is an overview of the features that **TurboGrade™** offers you:

1. **Class management:** Classes are organized by year and semester. You can create sections, assign labs and exercises effortlessly.
2. **Moodle Integration:** Full support of importing student submissions in the format delivered by Moodle.
3. **Visual Rubric Creator:** An integrated rubric creation delivers a “WYSIWYG” experience when creating rubrics, making it intuitive to assign points to different criteria. The Rubric Creator is also flexible, allowing you to grade by penalties / credit points as well as sub-criteria / grading guides.
4. **Integrated Grading Environment:** An IDE-like interface provides everything needed for grading in one comprehensive window: running Processing sketches, capturing terminal output, looking at student code (with full syntax highlighting), adding comments, adjusting grades, all within the same interface.
5. **Automatic comment suggestion:** The software is able to memorize previous comments and suggest them (as well as the penalty accompanying them) just by typing the first few letters.
6. **Synchronization to the cloud:** Submissions and grading progress are automatically backed-up in a configurable remote git server. This allows for use from multiple clients on multiple platforms.
7. **Configurable Interface:** The user interface is adjustable to your needs, you can resize and hide portions of the interface, use the preset skins or change the font size to your likings.

... as well as multiple other features that make grading easier and more efficient.

II. Getting started

A. Installing TurboGrade™

Installing TurboGrade™ is very easy: Locate the appropriate installer for your platform, launch it and follow the easy installation steps.

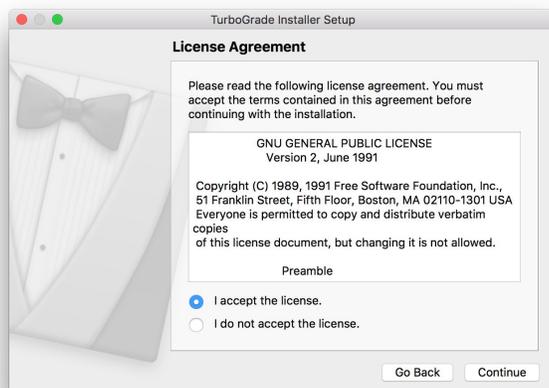
1. Locate the Installer binary



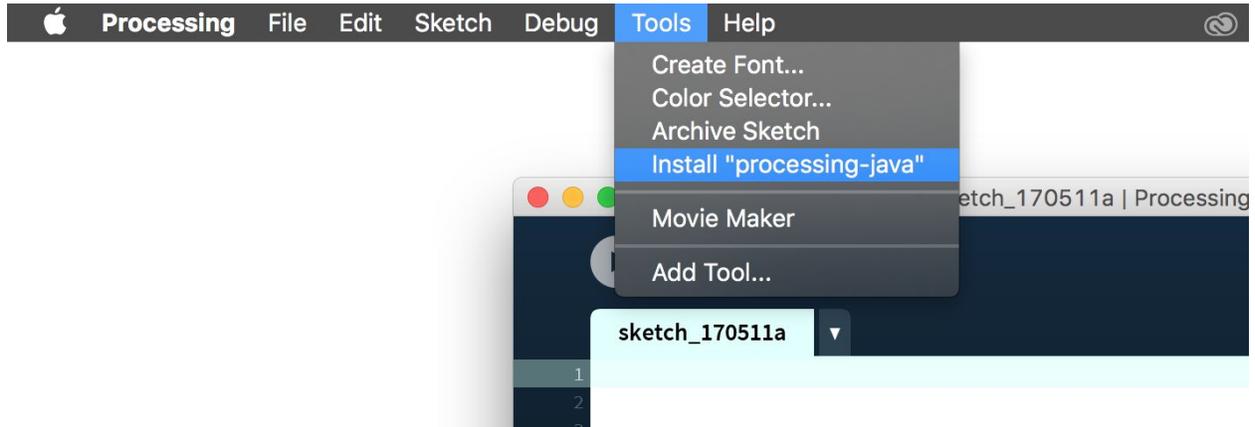
2. Run the Installer



3. Read the License Agreement and proceed with the installation

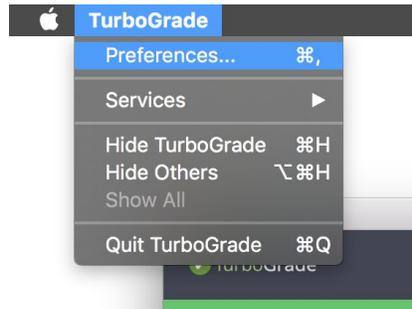


4. Install “git” and “Processing” (the application uses them internally)
5. To **activate the RUN feature** (running processing sketches from TurboGrade™), please don’t forget to activate Processing-Java from Processing > Tools > Install “processing-java”:

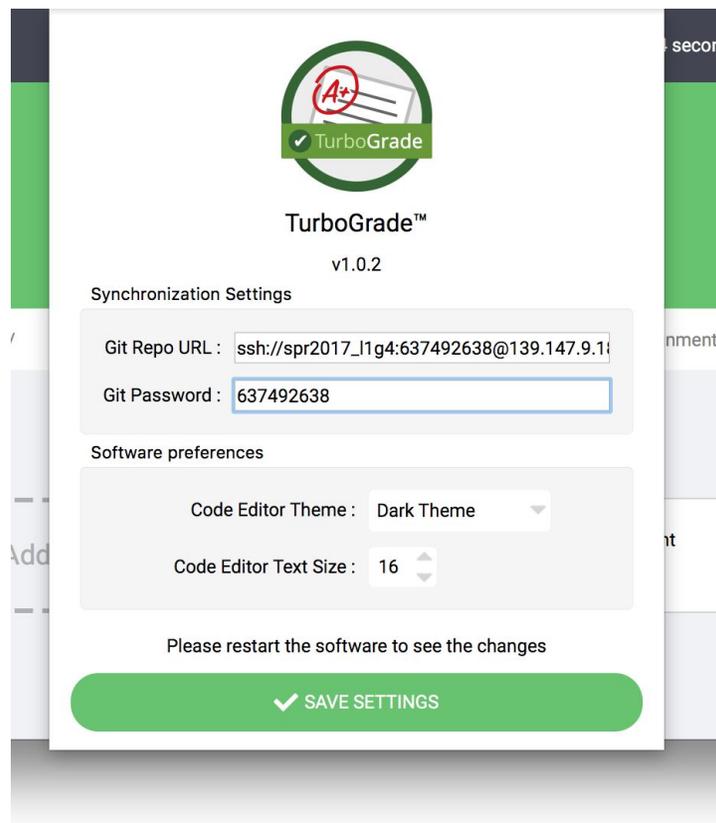


B. Configuring TurboGrade™

TurboGrade™ provides various configurable options easily accessible by clicking on the “Preferences” menu in the menu bar.



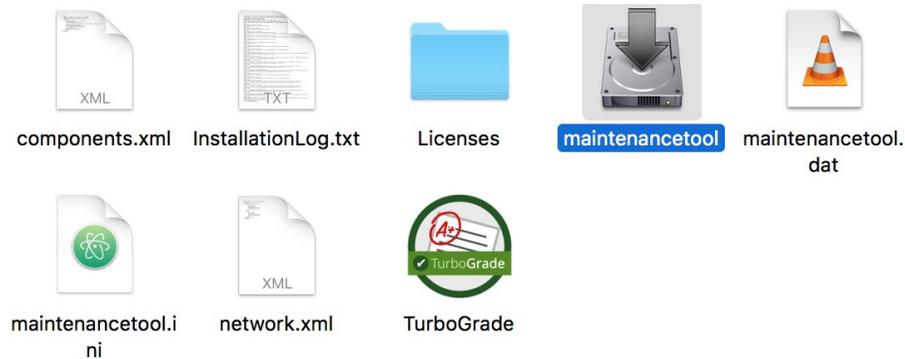
The Preferences dialog provides you with various customization options such as skins, font-size and Git synchronization settings:



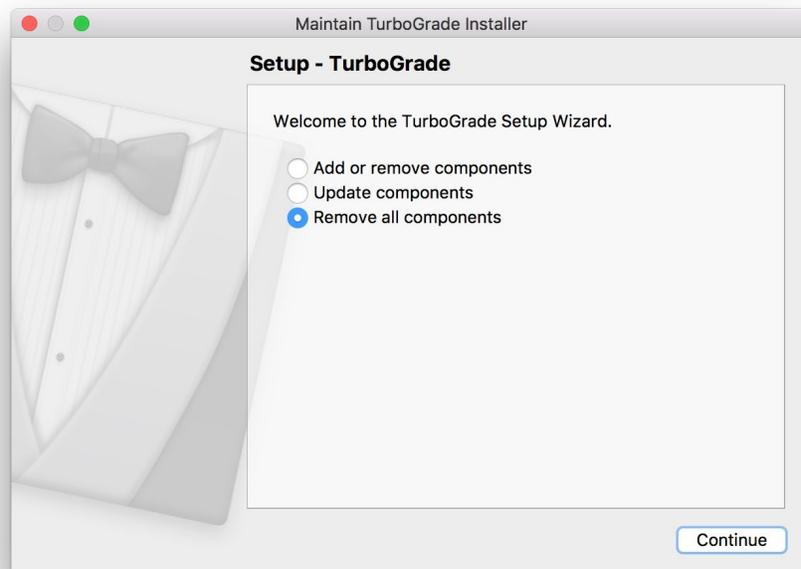
C. Uninstalling TurboGrade™

To remove TurboGrade™, please follow these simple steps:

1. Locate the Maintenance Tool in the install folder specified in step A



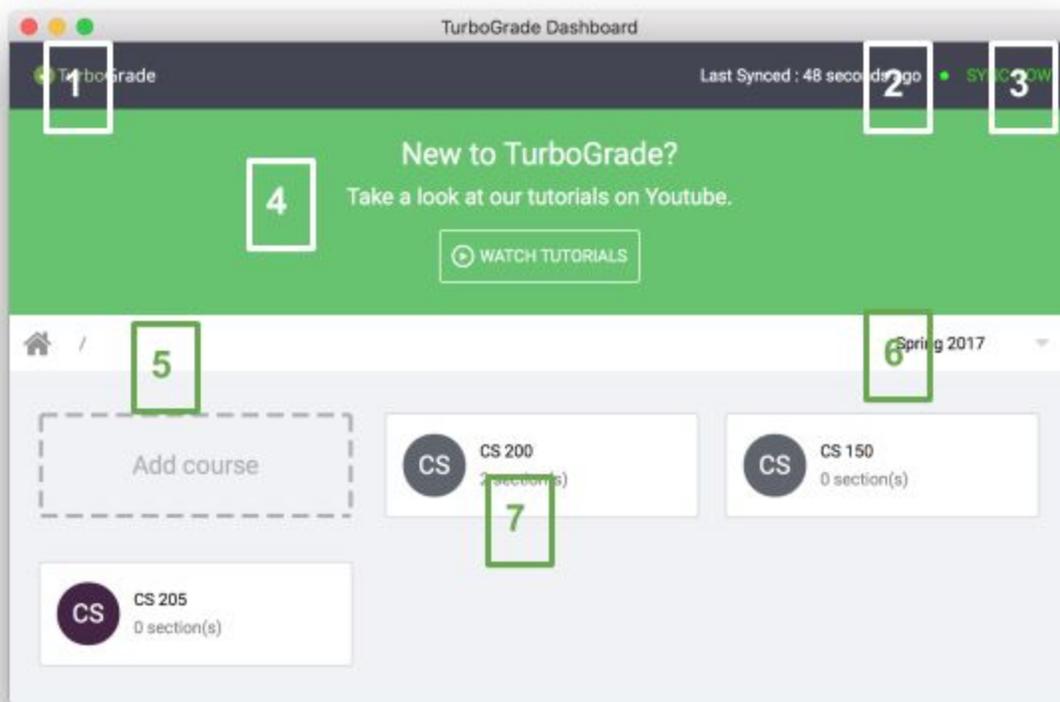
2. In the Maintenance Tool, Choose “Remove all components”



3. Proceed with the uninstallation as directed by the Maintenance Tool

D. Dashboard Overview

The **TurboGrade™** dashboard is the first screen you see when you open the software. It is also the main user interface for **TurboGrade™**, it provides a comprehensive look at your classes, sections, students and assignments. The **TurboGrade™** dashboard is very easy to understand:



Elements on the dashboard from top to bottom, left to right:

1. The **TurboGrade™** logo always takes you to the course view
2. **Synchronization status** (updated every 15 seconds)
3. **"SYNC NOW"** button: forces a git synchronization (saves to the cloud)
4. **TurboGrade™** Tutorials, takes you to the Youtube page to watch tutorials
5. **Breadcrumb trail**: shows you the course/section/assignment you are browsing
6. **Additional controls** (such as semester drop down, assignment/student switch, etc.)
7. **List view**: lists the content you are looking at (ex. List of courses)

E. Grading UI Overview

The screenshot displays the Pfaff Corp TurboGrade grading interface. The top bar features a back button, TurboGrade status, Anonymous Grading selector, Hide name checkbox, and Next Submission button. On the right, there are Export PDF, Run, and Finalize buttons. The left sidebar shows a file explorer for 'lab5' and a RUBRIC & GRADING panel with four rows: Documentation (2/2), Extra-credit (5/5), Design (9/10), and Correctness (19/20). A Total Grade of 94% (35 out of 37) is shown. The central code editor displays Java code for a Gallery class. The right sidebar contains GENERAL COMMENTS and COMMENTS ON THIS FILE. A comment input box is open over the code editor, showing a Points field set to 0 and a Rubric dropdown set to No criterion selected.

The Integrated Grading Environment offers you multiple sophisticated tools in a simple interface. The top bar contains multiple controls:

1. **Progress bar** (far top) shows you how many submissions are left to be graded in a section
2. **Back button** gets you back to the dashboard to browse submissions and manage courses
3. **Submission selector** (defaults to hidden student names) lets you choose another student to grade
4. **Next submission button** (only activated when the current grade is finalized) takes you to a randomly selected submission that wasn't graded yet

5. **Export PDF** lets you save the current submission's student report in a folder that you specify.
6. **Run**, automatically detects the language the submission is written in and runs it (either through java in the console or in Processing).
7. **Finalize** locks the grade and changes the submission's graded status to "finalized"

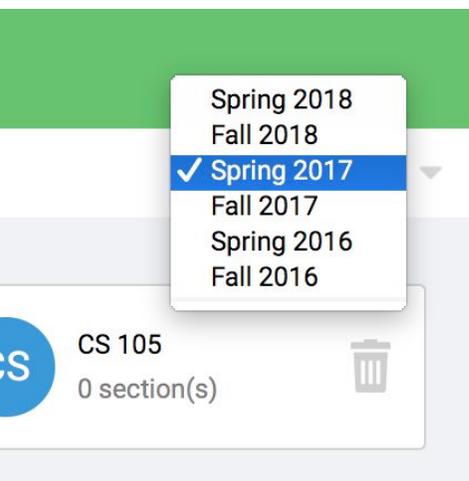
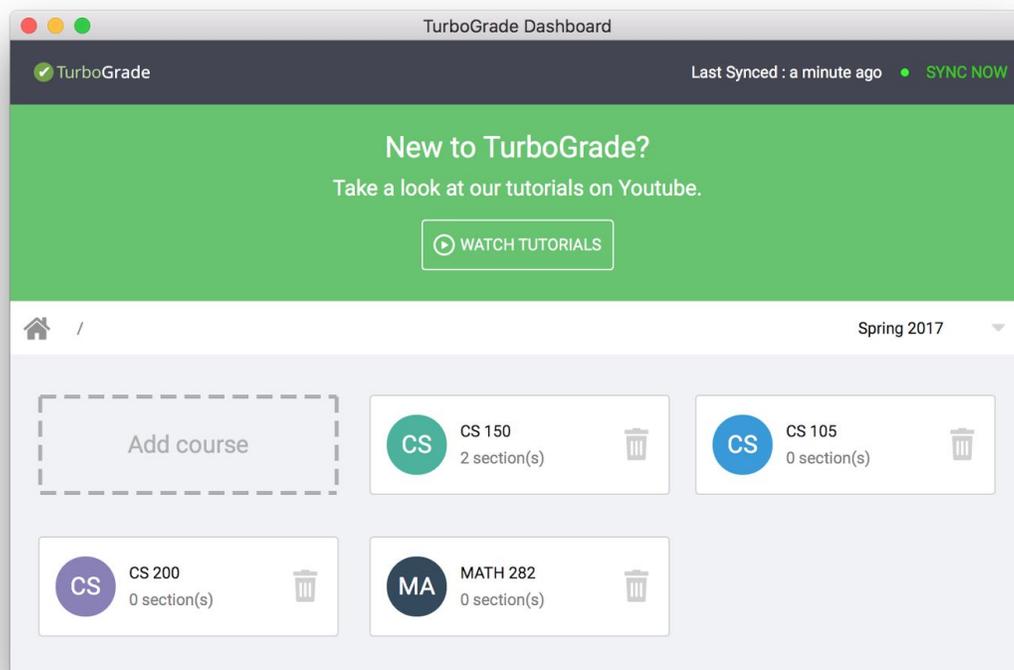
The Interface is made up of various "widgets", each of them equally important in the grading process:

1. **File tree view** (top left), shows you the files that belong to the current submission and allows you to switch between them
2. **Grading view** (middle left), shows you the rubric and lets you adjust grades for each criterion (also shows comments associated with a criterion when they exist)
3. **Code view** (center), shows you the code for the selected file with syntax highlighting. Selecting code in this view opens up a prompt to add a comment.
4. **Comment view** (right), lists all comments on the current file as well as general comments on the submission as a whole. Hovering the mouse over a comment highlights it in the code.
5. **Terminal** (bottom) displays output from Processing/Java when the program is ran (also displays compile errors and any standard error output).

III. Managing a class

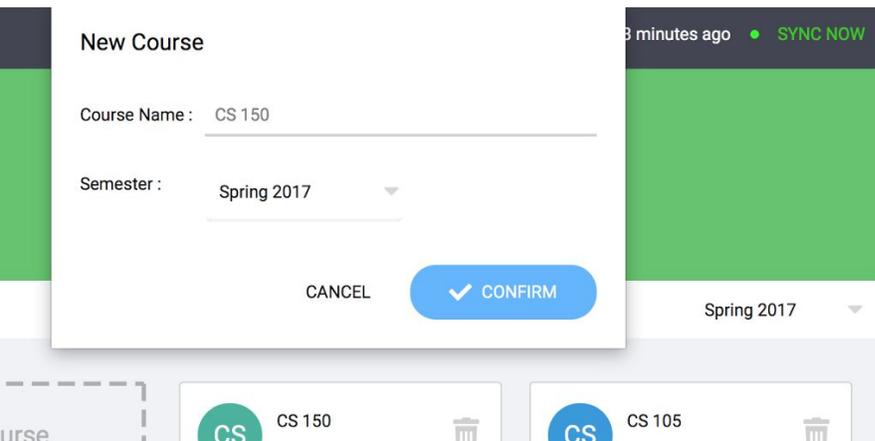
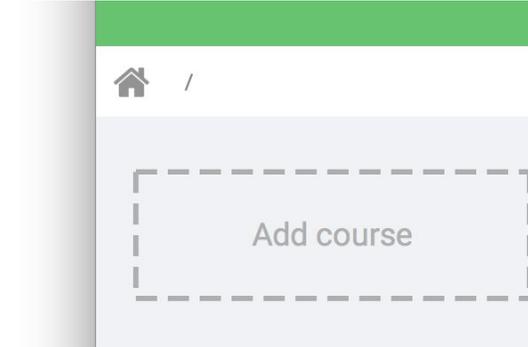
A. Managing courses

On the Dashboard, the first screen that comes up is the “Course View”. This screen displays all the previously created courses (that are also synced to the git repository).



To the right of the breadcrumb trail, a semester dropdown allows you to choose the semester/year for which courses will be displayed. This defaults to the current semester (based on the current month and year).

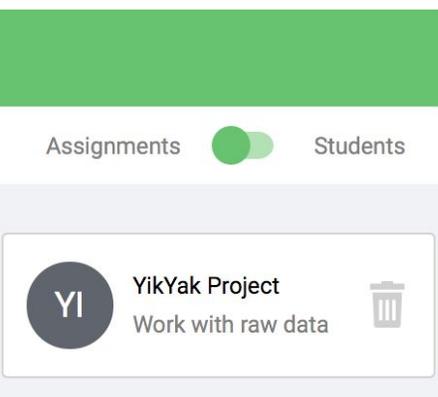
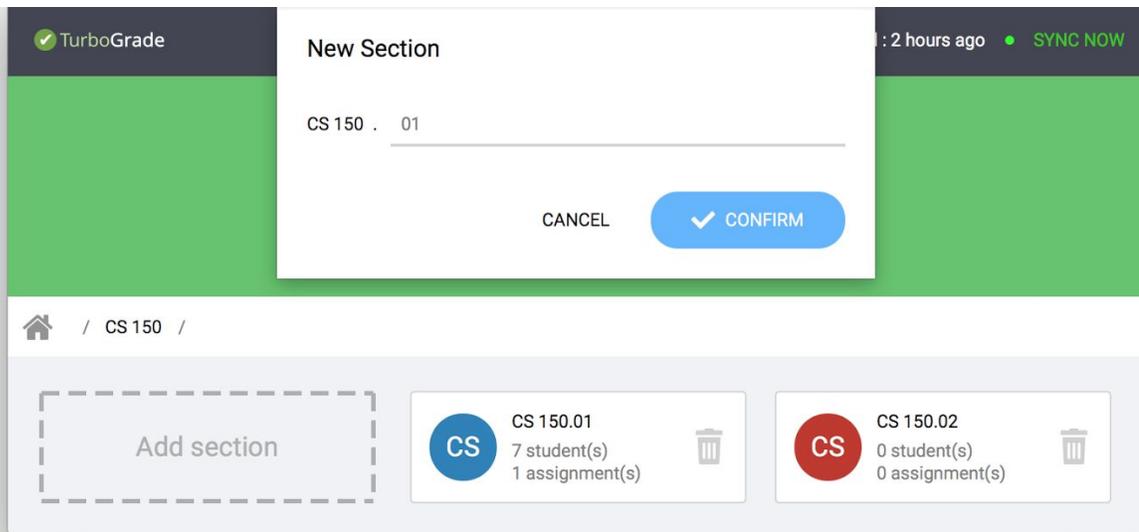
To add course, click on the “Add course” button, which will open the dialog shown below, allowing you to specify the new course’s name and semester (the current semester will be selected automatically).



After filling out the course name and semester, clicking the “Confirm” button will create the course and display it on the dashboard.

B. Managing semesters

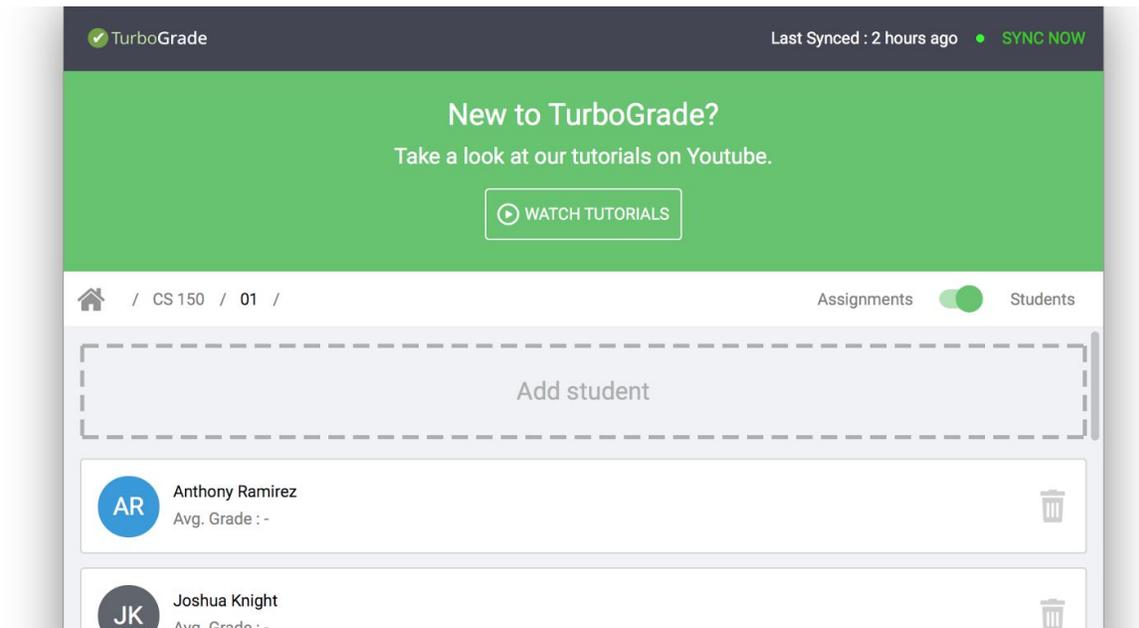
Managing submissions is very similar to managing courses. Using the same buttons you can create, delete and manage sections within a course :



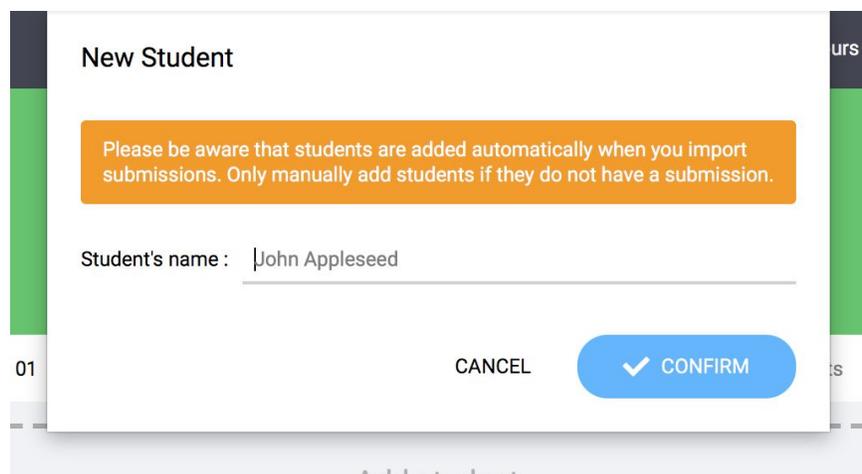
Within the semester view, you can switch between the student view and the assignments view to either look at the list of assignments linked to this section or the list of students that belong to this section.

C. Managing students

The students view list all students within a section and gives you insight on their grade in the course:



Students are not supposed to be manually created in the software. Importing the first assignment submissions to a section will automatically detect and create students if they don't exist. Subsequent assignments will use the existing student objects (matching by name). Only manage students manually if a student registered late to the class, dropped out or did not submit the first assignment.



IV. Managing assignments

A. Anatomy of an assignment

Before you start grading submissions, you should create an assignment in the section view. An assignment should be given a title, and objective, and an associated rubric, which you will create in the software.

Assign an existing lab/project

Existing assignment : No assignment selected ▾

Add new assignment

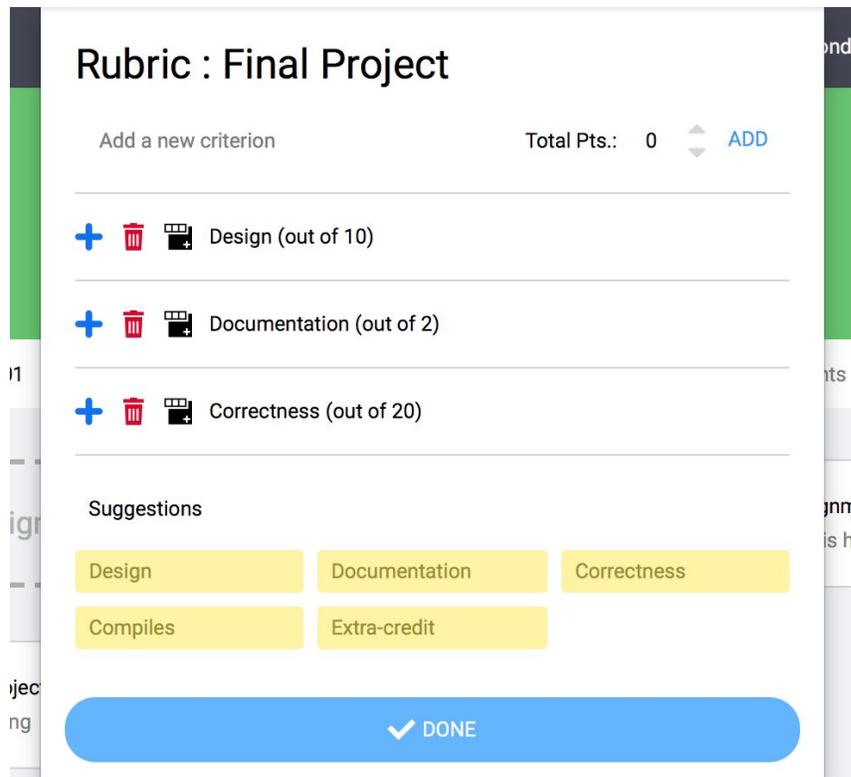
Assignment Name : Binary Search Tree

Assignment Objective : This project will help you get familiar with data structures

Initialize submissions with full grades (grade by penalties)

CANCEL

B. Rubric Maker



The screenshot shows a web interface for creating a rubric. The title is "Rubric : Final Project". At the top right, it says "Total Pts.: 0" with a double-headed arrow and an "ADD" button. Below this is a list of criteria:

- Design (out of 10)
- Documentation (out of 2)
- Correctness (out of 20)

Each criterion has a blue plus icon, a red trash icon, and a black icon representing a document or checklist. Below the criteria is a "Suggestions" section with five yellow buttons: "Design", "Documentation", "Correctness", "Compiles", and "Extra-credit". At the bottom is a large blue button with a white checkmark and the text "DONE".

Rubrics are customizable through the rubric maker and allow you to have a transparent grading policy with the students. They also allow you to speed up the grading process by associating grade-adjusted comments with rubric criteria.

TurboGrade™ views rubrics as collection of criteria, each potentially having sub-criteria. We have a few criteria suggestions to help you get started. In general, you can create a rubric criterion by naming it, typing the point allocation, and hitting ADD.

To add a sub-criterion, simply hit the + underneath a pre-existing criterion. The creation process for a sub-criterion is similar to a plain criterion. It should be noted that once a sub-criterion is added to a criterion, the software will grade the parent rubric by summing up all of the parent's sub-criteria.

In addition, to accommodate for a variety of grading styles, graders can choose for each assignment to be based on an additive or negative point system, where:

Additive point systems have all assignments start with a score of 0 in every category, where accomplishments by students are rewarded with points, totalling to the highest possible score, while

Negative point systems assume a full score for each student at the beginning of every grading session. When the student doesn't meet the standards of the grader, he/she may deduct points for insufficient work.

C. Assignments for multiple sections

Once you have created an assignment, we provide the option to import that rubric over to another section. To do this, simply add an assignment from the corresponding section and select the assignment from the "Existing assignment" drop down menu.

V. Grading a submission

A. Commenting

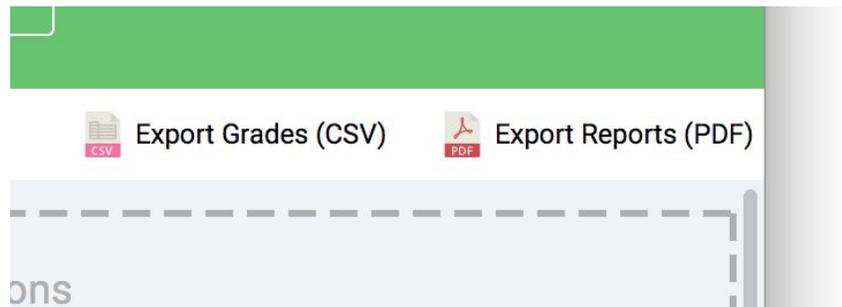
TurboGradeTM is designed for the commenting process to contribute directly to the final score given to the student's submission. In order to write a comment, simply highlight a portion of code you'd like to comment on, and type in your comment in the appearing window. When each comment is created, the user will be prompted to associate a point value, as well as a rubric category to the comment. As each comment is added, any point addition/deduction that has been assigned to it will be automatically factored into the the grade view on the left hand side of the the code grading user interface.

B. Manual Alteration

Independent of the commenting process, grades for any category can be changed at any time. By selecting the up or down arrow on any present criterion, that criterion's score can be set from 0 to the maximum allotted score for that category. This allows for a bit more flexibility for the user, where any additional alteration to a student's grade, that may not necessarily align with a line of code.

VI. Student Deliverables & Gradebook

A. User Interface



There are two ways to export the student's output PDF, which contains all relevant information for a submission, including all comments, how they alter the student's grade, as well as the final grades the student received. From the grading page of any student's submission, the user can select "Export PDF" option, in the top right hand corner of the screen. Once a destination is selected, a PDF version of the student's results will be saved to that directory. Alternatively, from any assignment's dashboard screen, all reports from that assignment can be exported to any directory. This is meant to save the user some time, so they can focus on grading, and then get all of their results after the fact.

B. Example Student Report



Student
Wassim Gharbi

TOTAL GRADE

87 %

Assignment: YikYak Project

Objective:

This project will help you work with raw data

GENERAL COMMENTS

You did well on this assignment, beware of commenting and documentation.

Grade Breakdown

Design 8 /10

Correctness 5 /20

— Visualizations : 5 out of 5

Documentation 1 /1

TOTAL GRADE : 14 /16 (87%)

○ Design 9 / 10

Remarks:

On "lab5/Gallery.java" for "Design"

```
1   throwError("Failed to add Painting, price can't be negative.");
2   else if (year<0)
3       throwError("Failed to add Painting, year can't be negative.");
4   else if (title.equals(""))
5       throwError("Failed to add Painting, name can't be empty.");
6   else if (painter.equals(""))
7       throwError("Failed to add Painting, painter can't be empty.");
8   else {
9       boolean unique=true;
```

-1 Input control should be in painting

○ Documentation 2 / 2

Remarks:

On "lab5/Painting.java" for "Comments"

```
1   public Painting(String painter, String title, int year, double price, String file)
   {
```

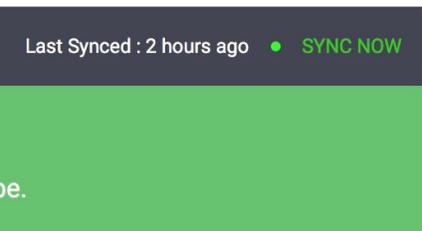
C. CSV Gradebook

	A	B	C	D	E	F	G	H
1	Student	Design (10)	Documentation - <u>JavaDoc</u> (1)	Documentation - Comments (1)	Documentation (2)	Correctness (20)	Total	Instructor Comments
2	Anthony Ramirez	10	1	1	2	20	32	
3	Barbara Gardner	10	1	1	2	20	32	
4	Joshua Knight	10	1	1	2	20	32	
5	Lao Cai	10	1	1	2	20	32	
6	Paul Castillo	10	1	1	2	20	32	
7	Thomas Day	10	1	1	2	20	32	
8	Wassim Gharbi	10	1	1	2	20	32	
9	William Kim	10	1	0	1	20	31	Poor comments

In addition to the reports, which are primarily meant for the students use, TurboGrade™ gives the professor the ability to keep a record as well. So we’ve built virtual gradebooks, that can be exported to any directory from the assignment dashboard. By selecting “Export CSV”, and providing a directory to save in, TurboGrade™ will export a CSV file of all students’ grades for that assignment, including the points earned per criterion of the assignment’s rubric, in addition to list all comments given to the student. This allows for a fast and easy way to keep track of an entire classes performance on an assignment.

VII. Archiving & Git Synchronization

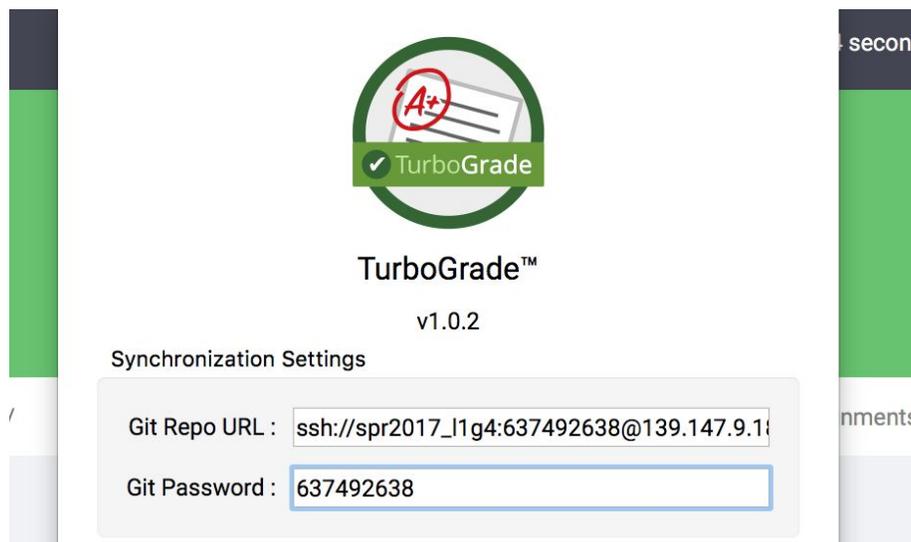
By default, the software links to a repository hosted at <http://github.com/turbograde/sync>.



Last Synced : 2 hours ago • **SYNC NOW**

Every time the software is ran, it performs a synchronization cycle by pulling the database changes as well as student submissions from the Cloud. At the end of every session (or when “SYNC NOW” is clicked), the software pushes local changes and backs up the database and the submission files to the git repository.

Clients that are setup to use the same Git repository will be synchronized. The software favors local changes over remote changes by default. This will be changed in future releases allowing the user to choose which version to keep through the user interface.




TurboGradeTM
v1.0.2

Synchronization Settings

Git Repo URL :

Git Password :

The Git repository and the password can be changed in the configuration panel. However, we currently only support synchronization through HTTPS and HTTP (no official multi-platform SSH support yet, although SSH works on Mac OS clients).