Quick Quiz Grader

CSM Paone

Dagny Stahl, Grace Jung, Adam Sandstedt

https://mastergo.mines.edu/quickgrader

Fall 2020

1 Introduction

The current Canvas grading infrastructure is lacking and requires solutions such as Quick Quiz Grader (QQG). The team developing this application worked for CSM Paone to create a better solution for grading Canvas administered quizzes. While this project was under the direction of Dr. Paone, Quick Quiz Grader is a tool for any TAs and professors at Mines wanting an easier way to grade Canvas quizzes.

Currently, quizzes can be graded through SpeedGrader, which is a native Canvas tool. The issue with SpeedGrader is that the user can see a single student's quiz submission on a page and must load the next page to grade the next student's submission. Rather than grading an entire quiz for one student, it's generally more efficient to grade one question at a time. In SpeedGrader, this method creates a lot of overhead because the next page must load in between grading each student's submission to a question. A more optimal user flow is to see all the submissions for a question on a single page, so the grader can scroll through quickly to grade without any downtime caused by changing pages. This is the basic principle behind Quick Quiz Grader.

Quick Quiz Grader is a web tool that communicates with Canvas to retrieve quiz data and display student submissions in a per-question format. It also includes the ability to create a rubric for each question. Then, when the user goes to grade a question, they can select which rubric items apply to each submission. Grading information can be saved to the database in between sessions. After grading is completed, grades and comments are calculated based on the applied rubric items applied. The grades and comments can be previewed and then posted to Canvas.

2 Requirements

2.1 Functional Requirements

As was stated above, the overall goal of the application was to allow faster grading than using Speed Grader or grading manually. More specifically, the application needs to:

- display pages of courses, quizzes, and quiz questions where user selects what to grade
- vertically display all the responses to one question at a time
- allow the user to build rubrics for each question that are then applied to all quiz submissions for grading
- have rubric items that are selectable for each submission
- calculate a grade is based on what rubric items are selected
- allow for easy re-grading if rubric items are modified after responses have been graded generate comments based on the rubric item descriptions that were applied to that submission
- save rubrics and selected rubric items for each submission between sessions and be accessible to different graders
- have a login page where users input a use code and an access token

- only allow approved graders have access to the tool
- allow for a rubric to be positive or negative

2.2 Non-Functional Requirements

The application must have a clear, concise, and nice-looking user interface with an intuitive flow. The application must transfer grades to and from Canvas securely, and handle tokens securely. The end-goal is to deploy on campus so that it can be used without the individual users needing to build it from source code. Overall the tool must be more efficient than the existing speedgrader. For the sake of any future teams, the code needs to be clean and easy to build locally. Thorough documentation will also ease future development and start-up cost.

3 System Architecture

Two Field Session teams have previously worked on this project. The codebase that was handed off to us used the .NET framework, but based on our experience of trying to get the outdated codebase functional, our team started the project from scratch. We decided to start over for 3 main reasons: (1) We wanted to update to ASP.NET Core 3.1 to be compatible on Linux, Windows, and MacOS, (2) there wasn't much code in the project and we could not get it functioning, and (3) we wanted to ensure the software design was that of a typical .NET application. We choose to stick with .NET and not move to something like Angular because it would be helpful to have a similar structure if we needed to reference the old code. A large change from the original project is that ASP.NET Core uses the Razor Pages programming model, which we fully embraced and will elaborate more on in section 4.2. Figure 1 depicts the web page hierarchy of the application.

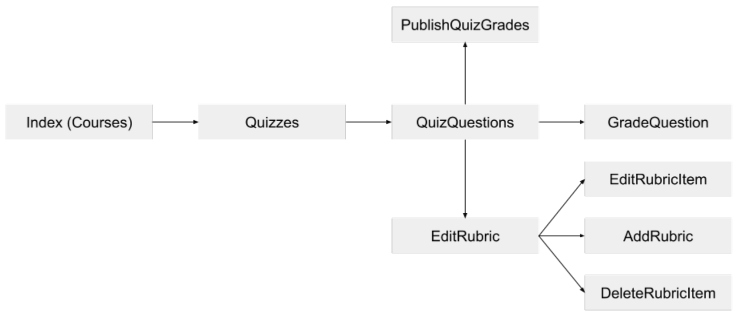


Figure 1: Web Page Hierarchy

Our application relies heavily on API calls to gather data. Figures 2 and 3 illustrate a simple end-to-end flow of our application, including what calls are made to Canvas and our database.

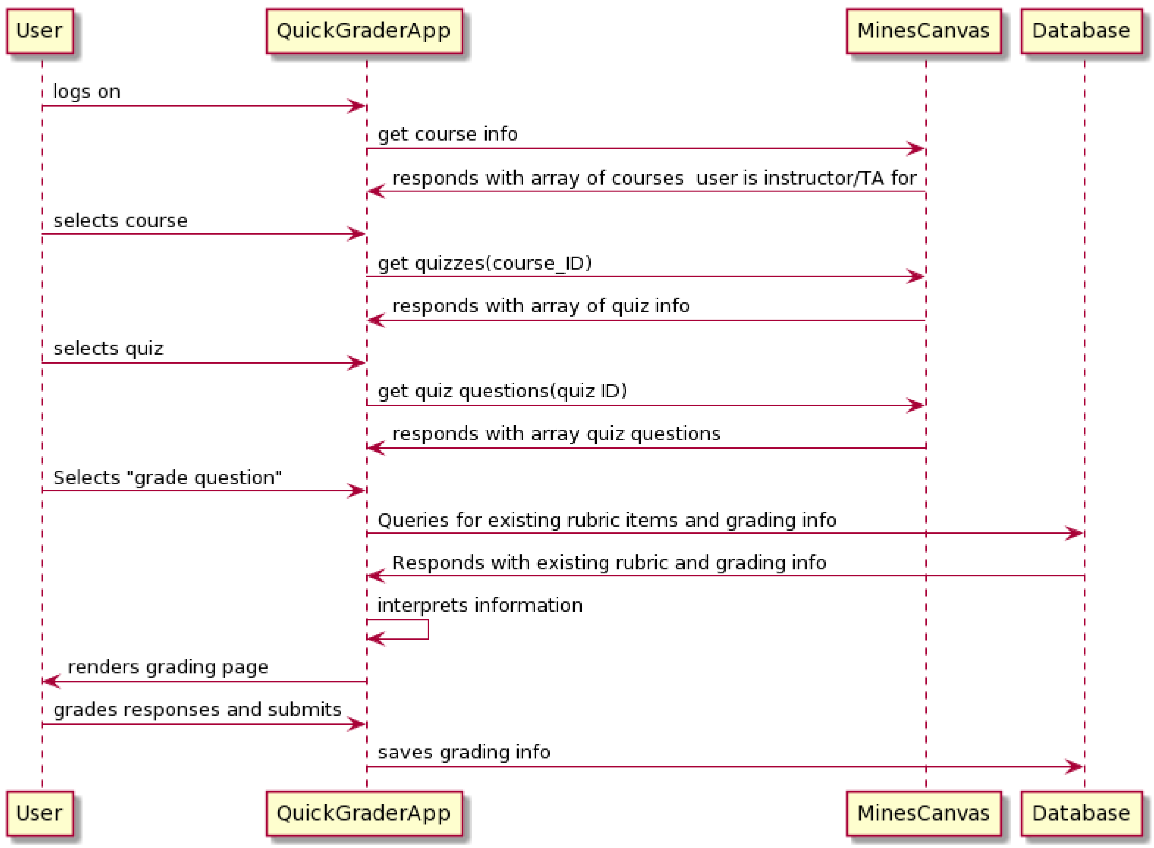

User
QuickGraderApp
MinesCanvas
Database
logs on
get course info
responds with array of courses user is instructor/TA for
selects course
get quizzes(course_ID)
responds with array of quiz info
selects quiz
get quiz questions(quiz ID)
responds with array quiz questions
Selects "grade question"
Queries for existing rubric items and grading info
Responds with existing rubric and grading info
interprets information
renders grading page
grades responses and submits
saves grading info
User
QuickGraderApp
MinesCanvas
Database

Figure 2: Application Grading Sequence Diagram

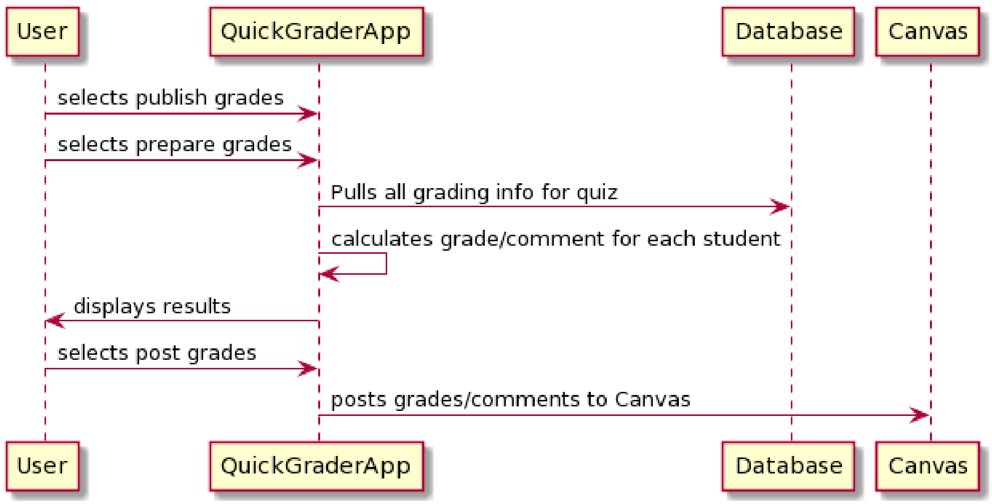


Figure 3: Application Grade Posting Sequence Diagram

The application is deployed natively on Mastergo, but the database is hosted on Megasort. Both are Computer Science Department servers, but Megasort requires the user to be on the Mines network to access it. This provides more security for the database. Mastergo has access to Megasort, so users do not need to be on a VPN to access and connect to the app. All external calls are made to the Mines Canvas Instance, which is hosted by ITS. This setup is depicted in figure 4 and the arrows represent the external calls being made by the application. The database schema is shown in figure 5.

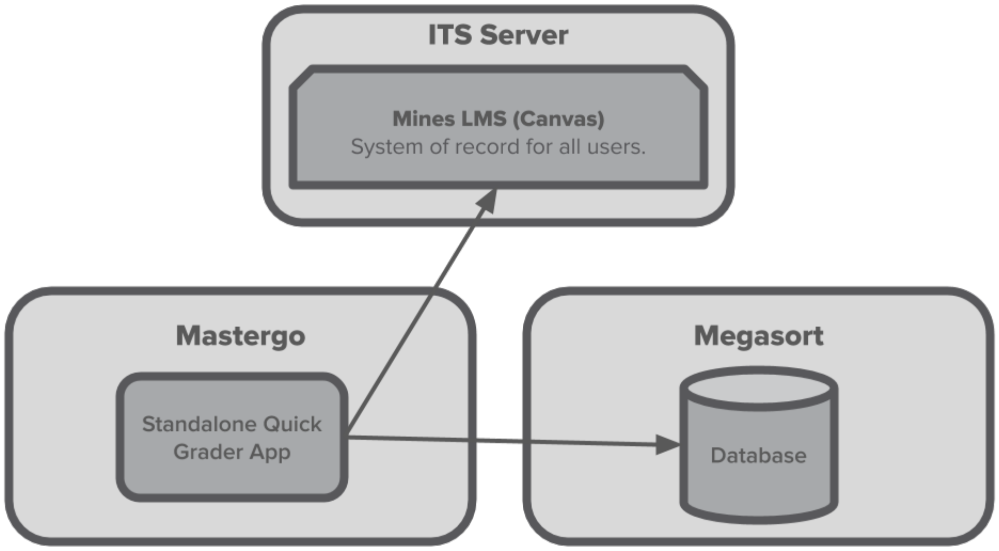


Figure 4: Application Deployment Setup

4 Technical Design

4.1 Modifying Rubric Items

One client criteria was that rubric items can be easily modified after grading. Let's say there was an item worth +0.5 points, but afterwards the user wanted it to be changed to +1 point. The user should be able to modify the rubric and then simply recalculate the grades without having to individually change each grade, and we accomplished this via a specific database design. The database has two tables that pertain to rubric items: GradedRubricItem and RubricItem as seen in the database schema in figure 5.

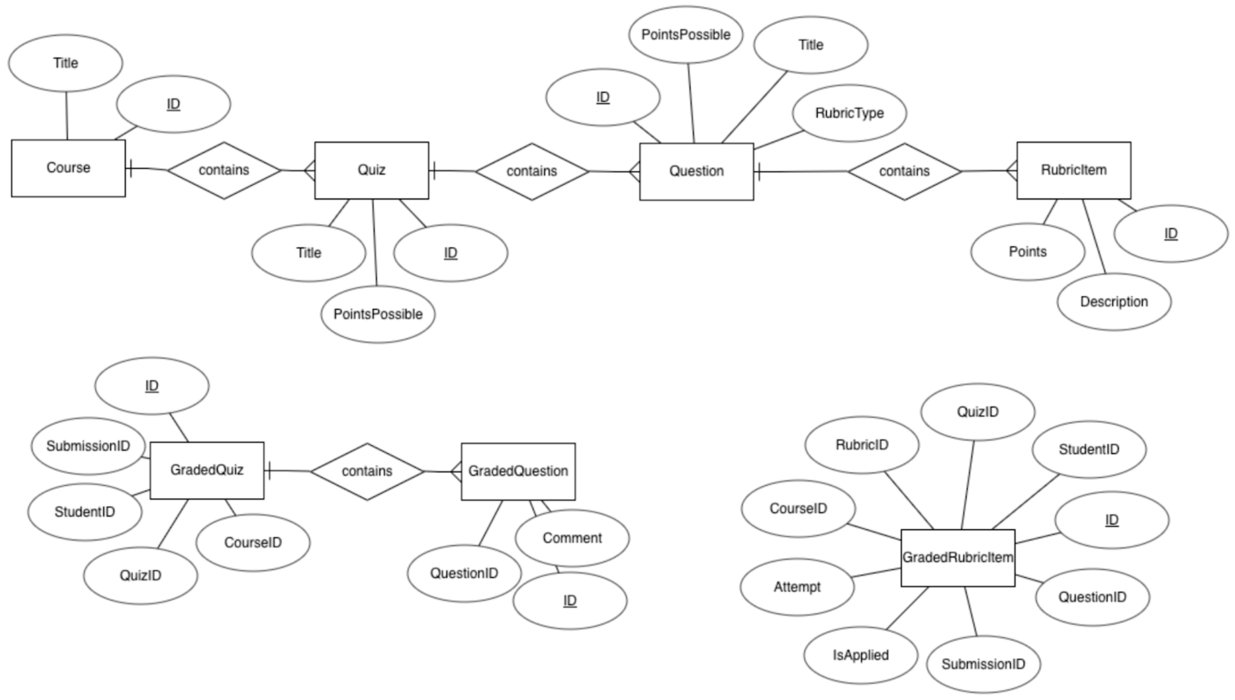


Figure 5: Database Schema

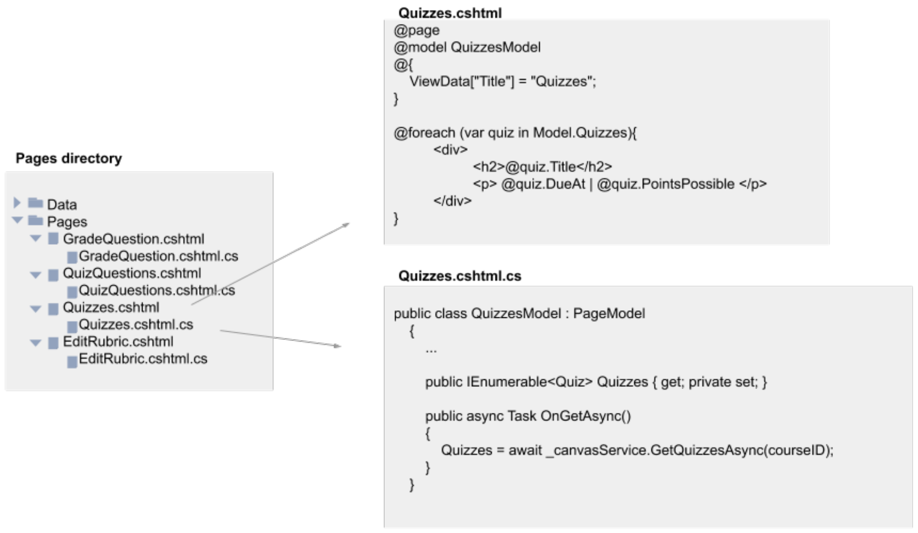
By splitting these tables, we track rubric item information in RubricItems and information about what rubric items have been applied to student submissions in GradedRubricItems. This way, if the user edits a rubric item, the raw data in RubricItem is updated, but the grading information stored in GradedRubricItems stays the same.

When the user decides to prepare grades we reference the applied rubric items in GradedRubricItems, and then pull the point and description info from the actual RubricItem table. This ensures that everything is kept up to date with any rubric modifications. The same logic applies if the user deletes an item. We will search for it, find nothing, and do nothing to the grade.

4.2 Razor Pages

As mentioned earlier, ASP.NET Core uses the Razor Pages programming model for the front-end. The benefit of this model is that it is page-focused. The alternative to Razor Pages would be using a Model-View-Controller format, which is popular, but requires information about a page to be in several locations. For a small application like ours, where each page has just a few tasks, it's very convenient to have each page and its behavior contained when building the UI.

Each url endpoint in our application corresponds to an HTML file in the Pages directory of our code. Each of these files contains a C# file within it. The C# file contains all the logic we need for our HTML. This includes a model for the page and any handlers we need to make API calls. This page essentially defines the behavior and gathers the information that we want to display in our HTML. Figure 6 shows a simplified example of our pages directory and how the two files that makeup a page look. It's best practice to do as much of the logic in the C# file but as demonstrated in the top right file, we can include C# within our HTML if needed. In this example, this page displays the list of quizzes. The bottom right C# file makes an API call to gather the Quiz objects, then the HTML file uses C# to loop through these objects to display each of them and their attributes.



```
Data
Pages
  GradeQuestion.cshtml
    GradeQuestion.cshtml.cs
  QuizQuestions.cshtml
    QuizQuestions.cshtml.cs
  Quizzes.cshtml
    Quizzes.cshtml.cs
  EditRubric.cshtml
    EditRubric.cshtml.cs
```

```
@page
@model QuizzesModel
@{
   ViewData["Title"] = "Quizzes";
}

@foreach (var quiz in Model.Quizzes){
        <div>
                <h2>@quiz.Title</h2>
                <p> @quiz.DueAt | @quiz.PointsPossible </p>
        </div>
}
```

```
public class QuizzesModel : PageModel
   {
      ...

      public IEnumerable<Quiz> Quizzes { get; private set; }

      public async Task OnGetAsync()
      {
         Quizzes = await _canvasService.GetQuizzesAsync(courseID);
      }
   }
```

Figure 6: Razor Page Hierarchy

4.3 Access Token Authentication

Although we attempted to redirect the login to ITS, so users could just use their multipass credentials, doing so turned out to be a bit more complicated than we thought. We also didn't have the time to go through ITS's security screen. So, we opted to keep the access token login so we could produce a functioning product. See figure 7 for the flow of verifying access tokens.

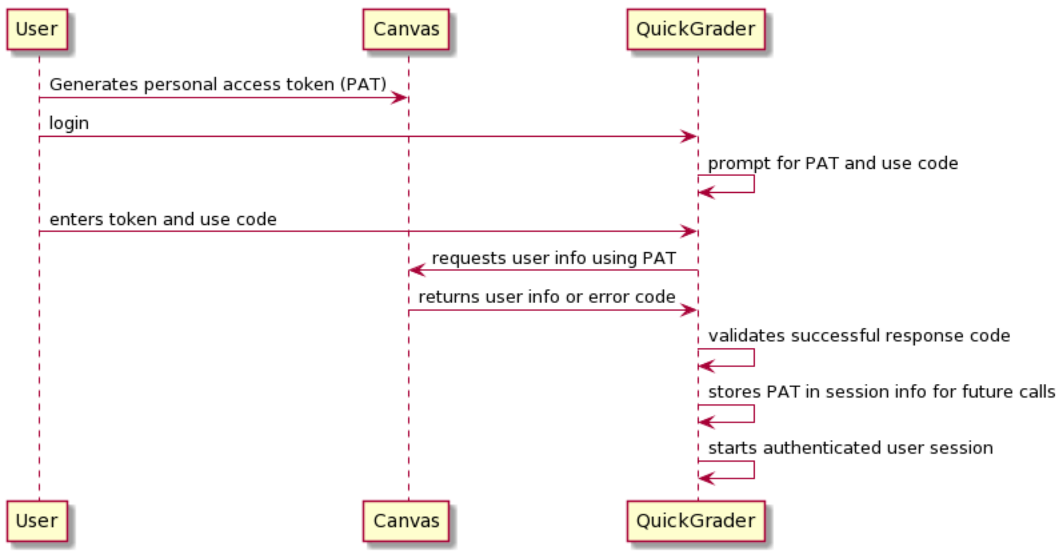

User
Canvas
QuickGrader
Generates personal access token (PAT)
login
prompt for PAT and use code
enters token and use code
requests user info using PAT
returns user info or error code
validates successful response code
stores PAT in session info for future calls
starts authenticated user session
User
Canvas
QuickGrader

Figure 7: User Authentication Sequence

Once we verify the access token, it needs to be stored so we can authenticate API calls to Canvas on behalf of the user. We chose to store the token in session data. Session data is stored for the length of the session, so after the user log outs, or the session times out, any session data lost. This is superior to storing tokens in a database, which would be preserved forever.

5 Quality Assurance

5.1 Unit Tests

Our team completed unit testing for things like authenticating tokens, and authorizing access to certain pages. Services were not tested on live servers, but rather given manufactured cases to run on, ensuring that testing is isolated and data is protected. Every service implements an interface, which allowed us to swap out services to return fake values for other services. See an example unit test in Appendix 7.3.

5.2 Integration Testing

As part of our testing, we did a first pass at vetting the API calls by running them in Postman before implementing. This way we could see what JSON values were returned and make sure they aligned with our object classes. Then we performed integration testing with Canvas. Since we could not test against active courses, all tests were run against a sandbox canvas course with manufactured data. We tried to create data with as much variation as possible to mimic a real course that's not going to have "perfect" data. This included:

- creating quizzes where fields were left blank
- using all the question types
- multiple attempt quizzes

- incomplete submissions
- quizzes with a lot of questions
- changing quizzes after grading had started
- resubmitting attempts after grading

5.3 User Testing

Our client was mainly concerned with the UI portion and user flow. As far as design and user testing, we met with our client whenever we added a new page or feature to the application so he could give feedback and reject anything early if it did not match his requirements. These meetings occurred every one to two weeks, so we could assess our user interface often.

5.4 Code Review

Since we had not built a .NET application before, we recruited a software engineer familiar with the framework to review our code. The goal of this was to ensure we met .NET coding and security standards. Feedback we got included:

- keep .cshtml files short and implement the logic in the corresponding .cs file
- pass around the auth token via session data
- override connection string to database in user secrets
- organize app into multiple projects, and not just one mega-project
- use an in-memory database to test, or a JSON database to start or un-block teamates
- split up Entity Core Frameworks (database tables) to be more modular
- pass routing information via OnGet parameters
- utilize OnPost to push data from user on pages
- pass objects to API functions, not every parameter of the object

We were able to implement all these suggestions in the final product.

6 Results

6.1 Unimplemented Features

We were able to implement all essential features required to make Quick Quiz Grader usable end-to-end. However, there was some functionality that we did not have time to complete. The tool works for grading fill-in-the-blank and essay style questions, which was the main goal of the project, but we did not implement grading for other question types (ie. multiple choice, true/false). Additionally, we planned on allowing the user to login with their multipass so it was integrated like other Mines websites, however that would have required an in-depth, lengthy security review by ITS that we did not have time for this semester. Lastly, a grader can use the “save” button to store progress to the database, but it would be more ideal if data was saved automatically as they graded to make the app more reactive. This would require modifying the front-end to utilize JavaScript, as we cannot accomplish this in HTML.

6.2 Lessons Learned

We learned a few lessons from this semester. One being that we should have focused on the most important features first. We spent a lot of time on configuring dev keys for Canvas with ITS, which we did not end up implementing. Instead, we should have been concerned with getting the product working in any matter, and use an access token login from the start.
We also learned that interfaces and temp classes were a great way to unblock teammates and keep forward momentum. For example, lets say one person was ready to work on the front-end portion of rendering grading information pulled from the database. We would write an interface for that service and whip up a fake temporary class for them to use that pulled manufactured data from a JSON file. Then another teammate could work on the real class that would actually pull from the database. When it was done, we could easily swap in the real class and integrate the work. However, because the interfaces are defined before the real functionality is fully thought-out, the closer the starter interface is to the end result makes integration easier. It is worth it to take more time to plan these out with teammates first, to avoid large changes later.

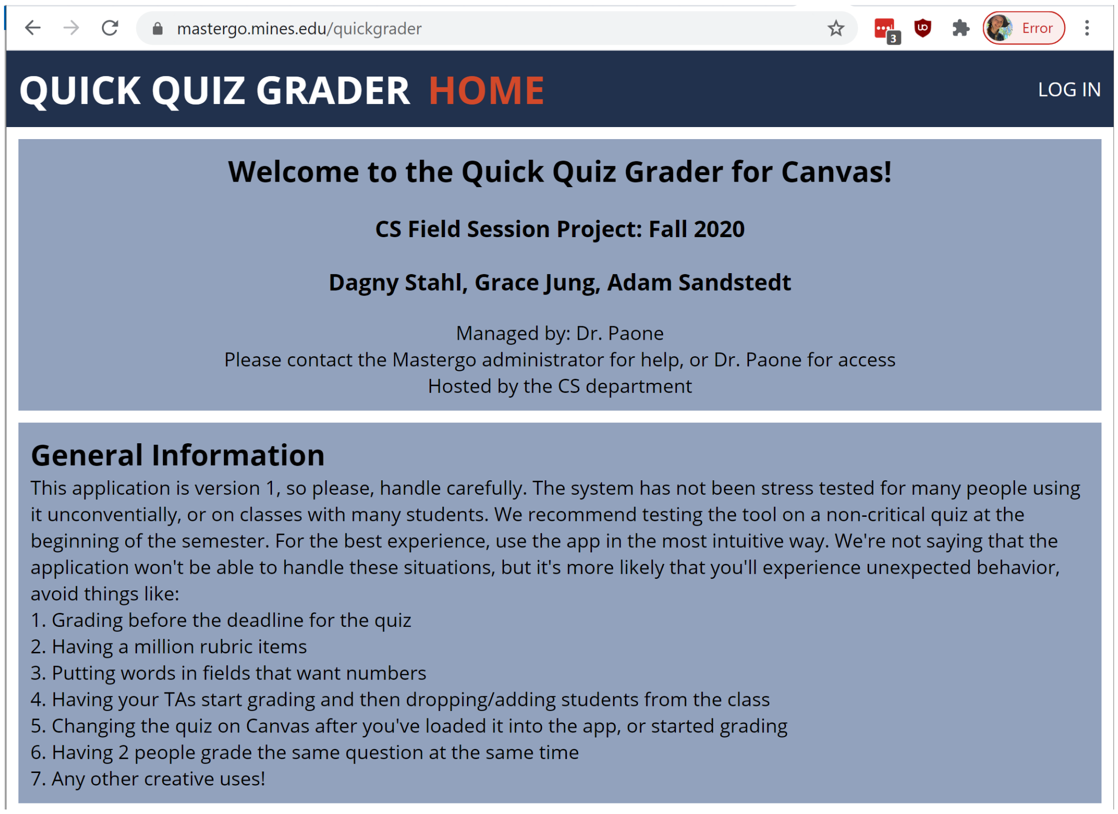
A lesson we knew going into the project was to utilize git to keep teammates on individual branches to accomplish specific goals. Branches can be merged as features are completed. This was very beneficial when we had different database versions on separate branches.

6.3 Achievements

There was quite a lot we achieved to create a deployable version of Quick Quiz Grader that we feel confident in. The user interface and flow of the application is much more intuitive than the previous version. The user is able to login with their access token and see all courses they have a grading role in, then choose a quiz and question to begin grading. The grader can create a positive or negative rubric to add to or subtract from the point value, respectively. When grading, you can hide students’ names to grade anonymously. Rubric and grading information is stored in a database hosted by the CS department. After grading, the user can preview grades and comments that will be posted to Canvas so they can double check and feel confident about the grades they’re submitting. The finished version of Quick Quiz Grader is available at https://mastergo.mines.edu/quickgrader.

7 Appendix

7.1 Application Page Images



Welcome to the Quick Quiz Grader for Canvas!

CS Field Session Project: Fall 2020

Dagny Stahl, Grace Jung, Adam Sandstedt

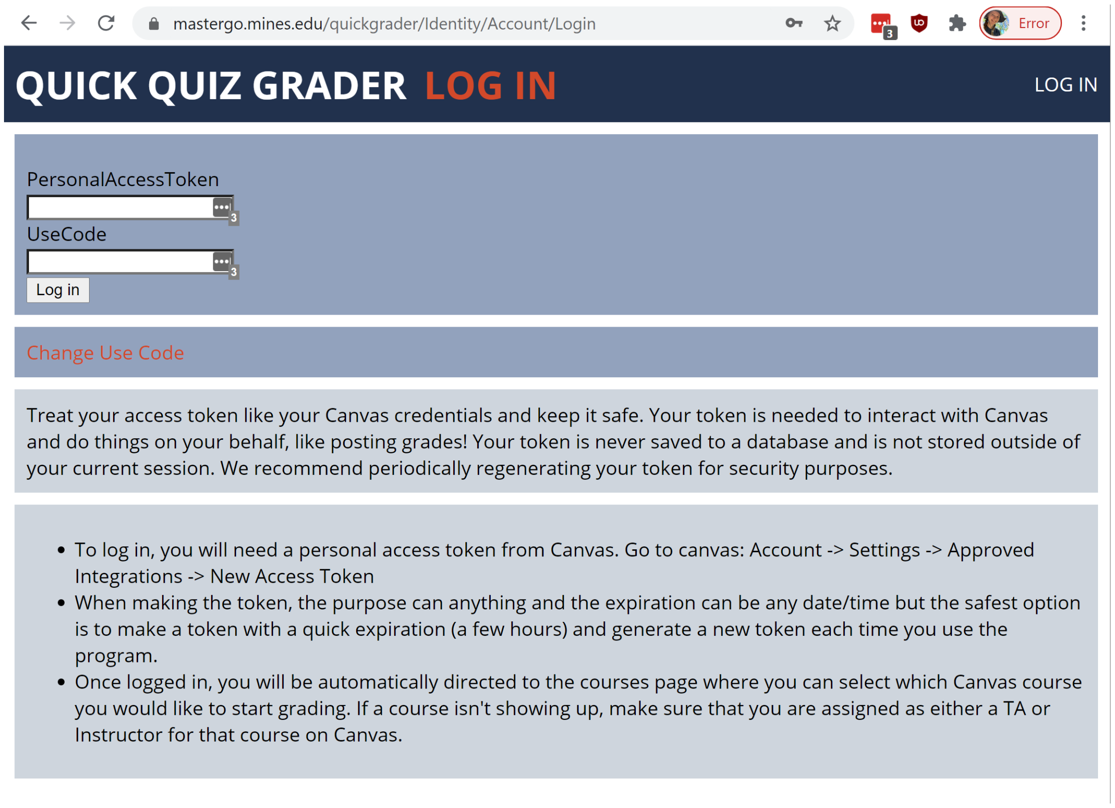
Managed by: Dr. Paone
Please contact the Mastergo administrator for help, or Dr. Paone for access
Hosted by the CS department

General Information

This application is version 1, so please, handle carefully. The system has not been stress tested for many people using it unconventially, or on classes with many students. We recommend testing the tool on a non-critical quiz at the beginning of the semester. For the best experience, use the app in the most intuitive way. We're not saying that the application won't be able to handle these situations, but it's more likely that you'll experience unexpected behavior, avoid things like:

1. Grading before the deadline for the quiz
2. Having a million rubric items
3. Putting words in fields that want numbers
4. Having your TAs start grading and then dropping/adding students from the class
5. Changing the quiz on Canvas after you've loaded it into the app, or started grading
6. Having 2 people grade the same question at the same time
7. Any other creative uses!

Figure 8: Home Page


mastergo.mines.edu/quickgrader/Identity/Account/Login

QUICK QUIZ GRADER LOG IN

LOG IN

PersonalAccessToken

UseCode

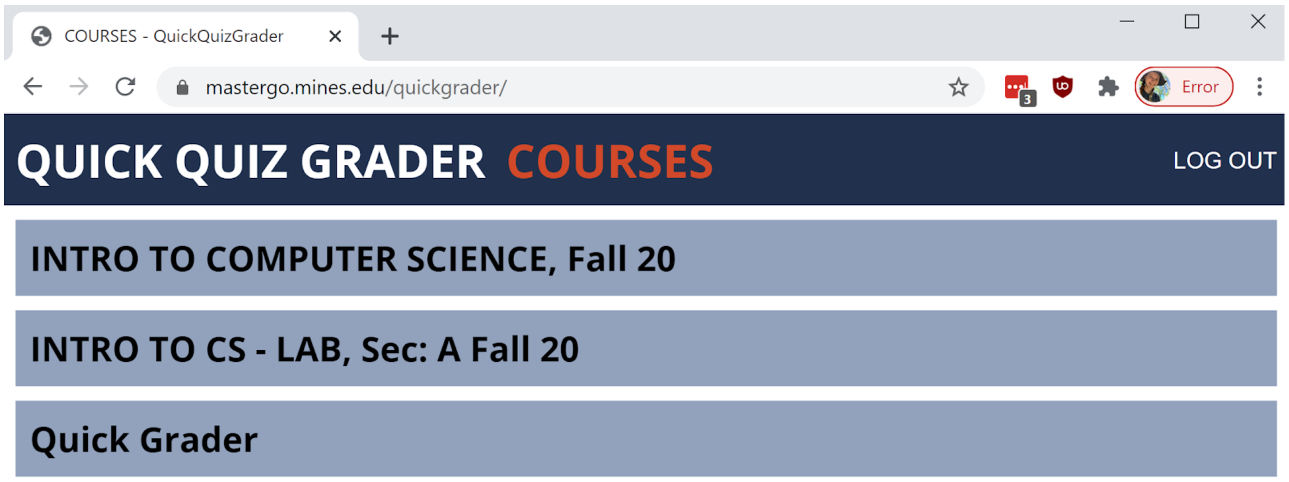
Log in

Change Use Code

Treat your access token like your Canvas credentials and keep it safe. Your token is needed to interact with Canvas and do things on your behalf, like posting grades! Your token is never saved to a database and is not stored outside of your current session. We recommend periodically regenerating your token for security purposes.

- To log in, you will need a personal access token from Canvas. Go to canvas: Account -> Settings -> Approved Integrations -> New Access Token
- When making the token, the purpose can anything and the expiration can be any date/time but the safest option is to make a token with a quick expiration (a few hours) and generate a new token each time you use the program.
- Once logged in, you will be automatically directed to the courses page where you can select which Canvas course you would like to start grading. If a course isn't showing up, make sure that you are assigned as either a TA or Instructor for that course on Canvas.

Figure 9: Login Page


COURSES - QuickQuizGrader

mastergo.mines.edu/quickgrader/

QUICK QUIZ GRADER COURSES

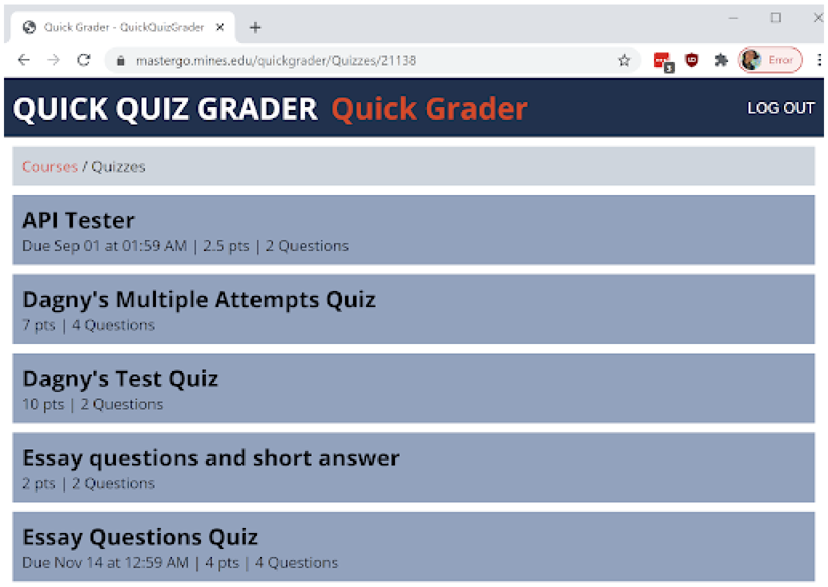
LOG OUT

INTRO TO COMPUTER SCIENCE, Fall 20

INTRO TO CS - LAB, Sec: A Fall 20

Quick Grader

Figure 10: Courses Page


Quick Grader - QuickQuizGrader

mastergo.mines.edu/quickgrader/Quizzes/21138

QUICK QUIZ GRADER Quick Grader

LOG OUT

Courses / Quizzes

API Tester
Due Sep 01 at 01:59 AM | 2.5 pts | 2 Questions

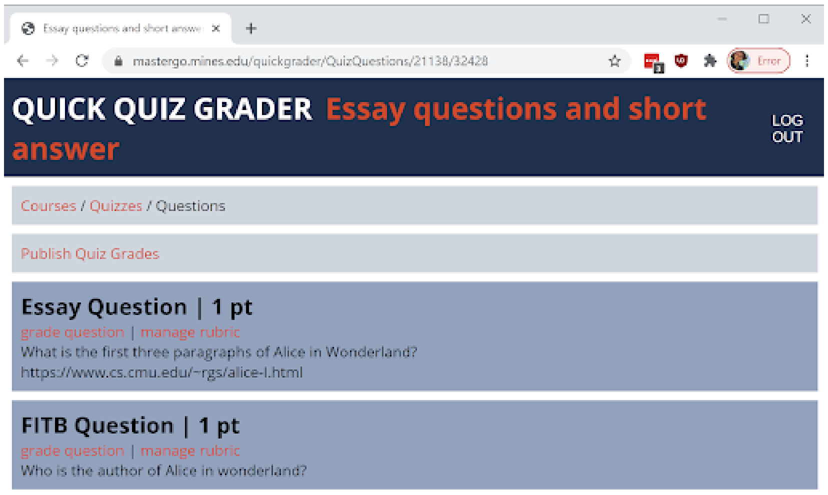
Dagny's Multiple Attempts Quiz
7 pts | 4 Questions

Dagny's Test Quiz
10 pts | 2 Questions

Essay questions and short answer
2 pts | 2 Questions

Essay Questions Quiz
Due Nov 14 at 12:59 AM | 4 pts | 4 Questions

Figure 11: Quizzes Page


Essay questions and short answe

mastergo.mines.edu/quickgrader/QuizQuestions/21138/32428

QUICK QUIZ GRADER Essay questions and short answer

LOG OUT

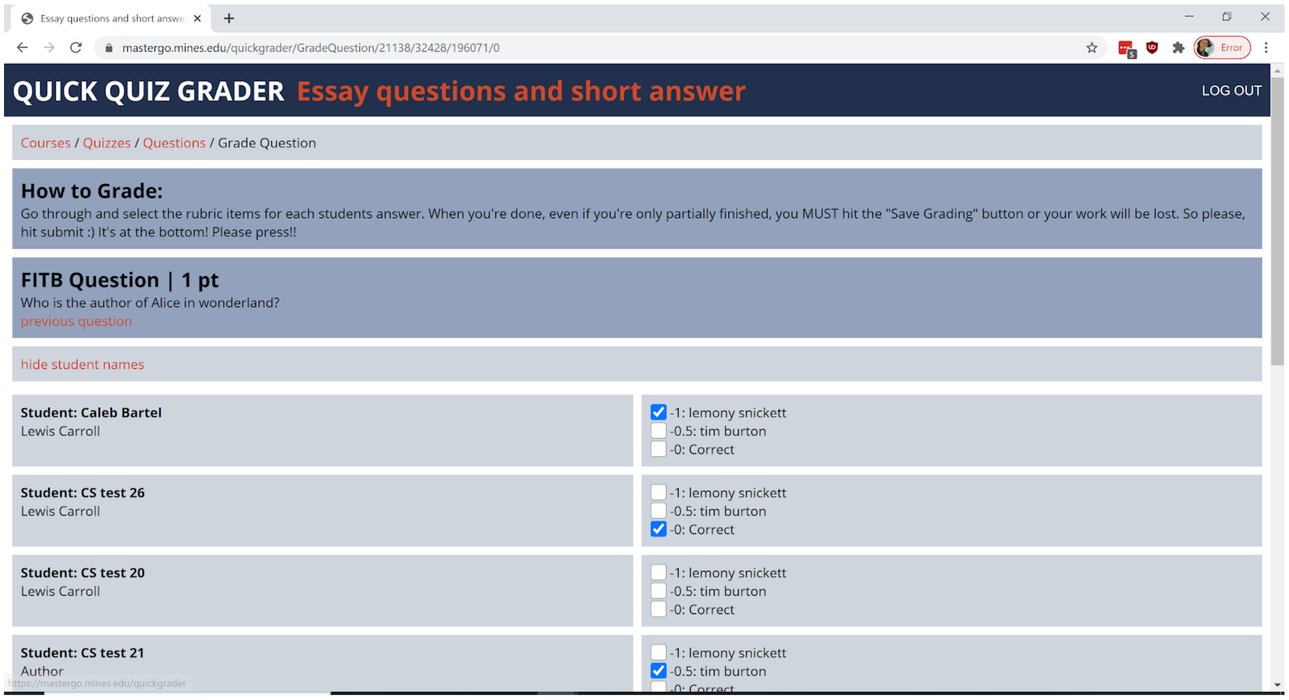
Courses / Quizzes / Questions

Publish Quiz Grades

Essay Question | 1 pt
grade question | manage rubric
What is the first three paragraphs of Alice in Wonderland?
https://www.cs.cmu.edu/~rgs/alice-I.html

FITB Question | 1 pt
grade question | manage rubric
Who is the author of Alice in wonderland?

Figure 12: Questions Page



Essay questions and short answer
mastergo.mines.edu/quickgrader/GradeQuestion/21138/32428/196071/0
Error

QUICK QUIZ GRADER Essay questions and short answer LOG OUT

Courses / Quizzes / Questions / Grade Question

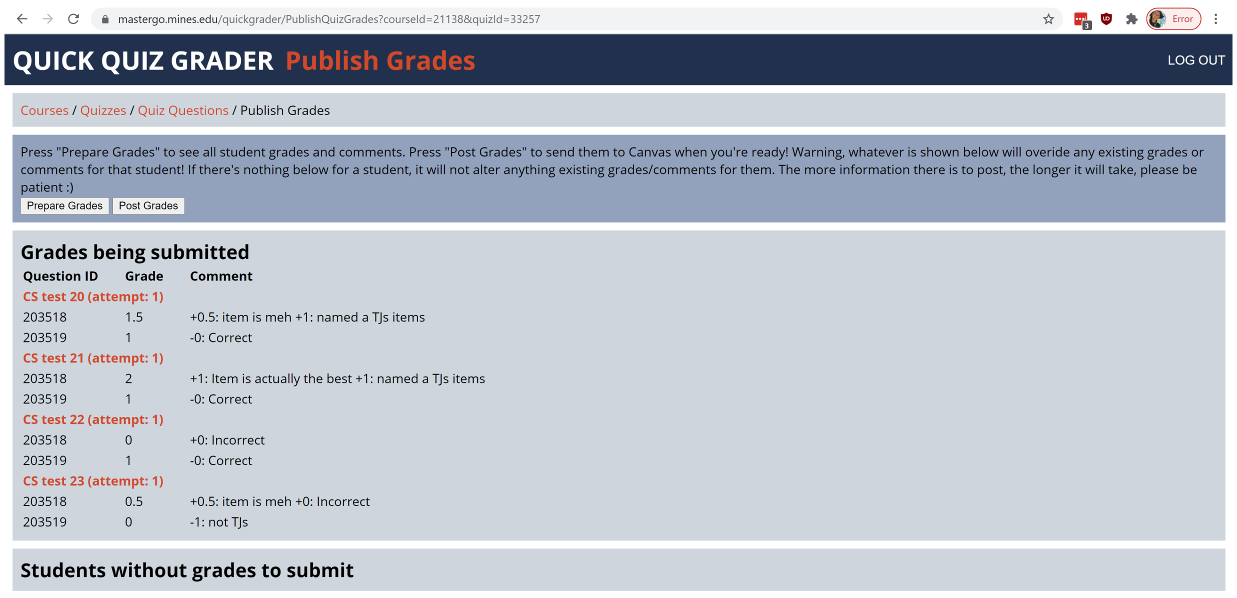
How to Grade:
Go through and select the rubric items for each students answer. When you're done, even if you're only partially finished, you MUST hit the "Save Grading" button or your work will be lost. So please, hit submit :) It's at the bottom! Please press!!

FITB Question | 1 pt
Who is the author of Alice in wonderland?
previous question

hide student names

Student: Caleb Bartel Lewis Carroll	✓ -1: lemony snickett ☐ -0.5: tim burton ☐ -0: Correct
Student: CS test 26 Lewis Carroll	☐ -1: lemony snickett ☐ -0.5: tim burton ✓ -0: Correct
Student: CS test 20 Lewis Carroll	☐ -1: lemony snickett ☐ -0.5: tim burton ☐ -0: Correct
Student: CS test 21 Author	☐ -1: lemony snickett ✓ -0.5: tim burton ☐ -0: Correct

Figure 13: Grading Page



mastergo.mines.edu/quickgrader/PublishQuizGrades?courseId=21138&quizId=33257
Error

QUICK QUIZ GRADER Publish Grades LOG OUT

Courses / Quizzes / Quiz Questions / Publish Grades

Press "Prepare Grades" to see all student grades and comments. Press "Post Grades" to send them to Canvas when you're ready! Warning, whatever is shown below will overide any existing grades or comments for that student! If there's nothing below for a student, it will not alter anything existing grades/comments for them. The more information there is to post, the longer it will take, please be patient :)

Prepare Grades | Post Grades

Grades being submitted

Question ID	Grade	Comment
CS test 20 (attempt: 1)		
203518	1.5	+0.5: item is meh +1: named a TJs items
203519	1	-0: Correct
CS test 21 (attempt: 1)		
203518	2	+1: Item is actually the best +1: named a TJs items
203519	1	-0: Correct
CS test 22 (attempt: 1)		
203518	0	+0: Incorrect
203519	1	-0: Correct
CS test 23 (attempt: 1)		
203518	0.5	+0.5: item is meh +0: Incorrect
203519	0	-1: not TJs

Students without grades to submit

Figure 14: Grade Posting Page

7.2 Start Up Instructions

Part 1 has instructions on how to build the program from the source code locally. If you already have it running or if someone else is running it for you, skip to Part 2

Part 1

1. Clone the project to your computer
2. Switch to branch FieldSessionF20
3. Open Visual Studio and choose "open a local project, solution, or file". Then open Canvas-Speed-Grader+.sln
4. There are several options to run. Make sure to select QuickGraderCoreF20 when you hit build
5. In order to connect to the database, you will need to add the DB connection string to your user secrets to overide the existing string in appsettings.json. This will allow you to connect to the live database. The connection string in appsettings.json is to a local DB we used during development that you do not have. Anything in our user secrets file will overide whats in appsettings.
   - If you're using VS on Windows, simply right click on the "QuickGraderCoreF20" project and choose "manage user secrets" then put the following json in that file: { "ConnectionStrings": { "postgresConnectionString": "Host=megasort;Username=enterusername;Password=enterPW" }}
   - On Mac, open the command line, navigate to the QuickQuizGrader folder and run the following command with the correct information filled in: dotnet user-secrets set "ConnectionStrings:postgresConnectionString" "Host=megasort;Username=enterusername;Password=enterPW;Database=canvasquickgrader"

Part 2

1. To log in, you will need a personal access token from Canvas. Go to canvas: Account - Settings - Approved Integrations - New Access Token
2. When making the token, the purpose can anything and the expiration can be any date/time but the ultra safest option is to make a token with a quick expiration (a few hours) and generate a new token each time you use the program.
3. Once logged in, you will be automatically directed to the courses page where you can select which Canvas course you would like to start grading. If a course isn't showing up, make sure that you are assigned as either a TA or Instructor for that course on Canvas.

7.3 Example Unit Test

```
//proves that authorized web routes will redirect you to login
       [Fact]
       public async Task Get_SecurePageRedirectsAnUnauthenticatedUser()
       {
           // Arrange
           var client = _factory.CreateClient(
               new WebApplicationFactoryClientOptions
               {
                   AllowAutoRedirect = false
```

```
        });
    // Act
    var response = await client.GetAsync("/Quizzes");
    // Assert
    Assert.Equal(HttpStatusCode.Redirect, response.StatusCode);
    Assert.StartsWith("http://localhost/Identity/Account/Login",
        response.Headers.Location.OriginalString);
}
```