Table of Contents

[Overview 2](#_Toc16607358)

[GitHub Classroom Instructions 2](#_Toc16607359)

[Instructor Instructions 2](#_Toc16607360)

[GitHub Classroom Setup 2](#_Toc16607361)

[Downloading the Class Roster 5](#_Toc16607362)

[Student Instructions 6](#_Toc16607363)

[Aces 7](#_Toc16607364)

[Student Use 7](#_Toc16607365)

[Instructor Use 7](#_Toc16607366)

[Creating a Class 8](#_Toc16607367)

[Deleting a Class 8](#_Toc16607368)

[Running the Program 8](#_Toc16607369)

[Ratings (colors) 8](#_Toc16607370)

[Code Explanation 9](#_Toc16607371)

[Controllers 9](#_Toc16607372)

[Services 9](#_Toc16607373)

[Models and ViewModels 11](#_Toc16607374)

[Database 11](#_Toc16607375)

[Unit Test Files 12](#_Toc16607376)

[To do 12](#_Toc16607377)

# Overview

Aces is a web application designed to make grading easier, catch cheaters using the student's git commit history, and to allow students to test their code on the server that will grade it. The students get a unit test containing a git commit function that does a git commit every time the student tests their code. Aces uses the number and frequency of commits to determine how likely it is a student cheated.

Aces makes grading easier by pulling each student’s code from their git repository, compiling it, and then grading it. The grader will see a list of all of the students assignments with the students names, grades, and a rating of how likely they cheated. The grader may also look at the reasons a student was given a certain rating.

Students may test their code by uploading it to the Aces student page, along with their unit test. The top priority for continuing the Aces project is security. Some of the things that need to be done are:

* Make it so code the students upload is not able to damage the server or anything on it
* Change how the instructor logs in to GitHub, so that it is no longer plain text
* Make the professor screen inaccessible to the students
* Save assignment data in .net core Session data
* Validate input
* Handle problems with student's code

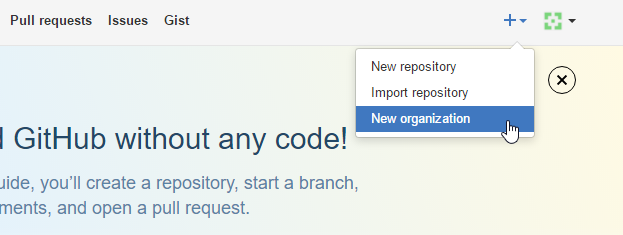
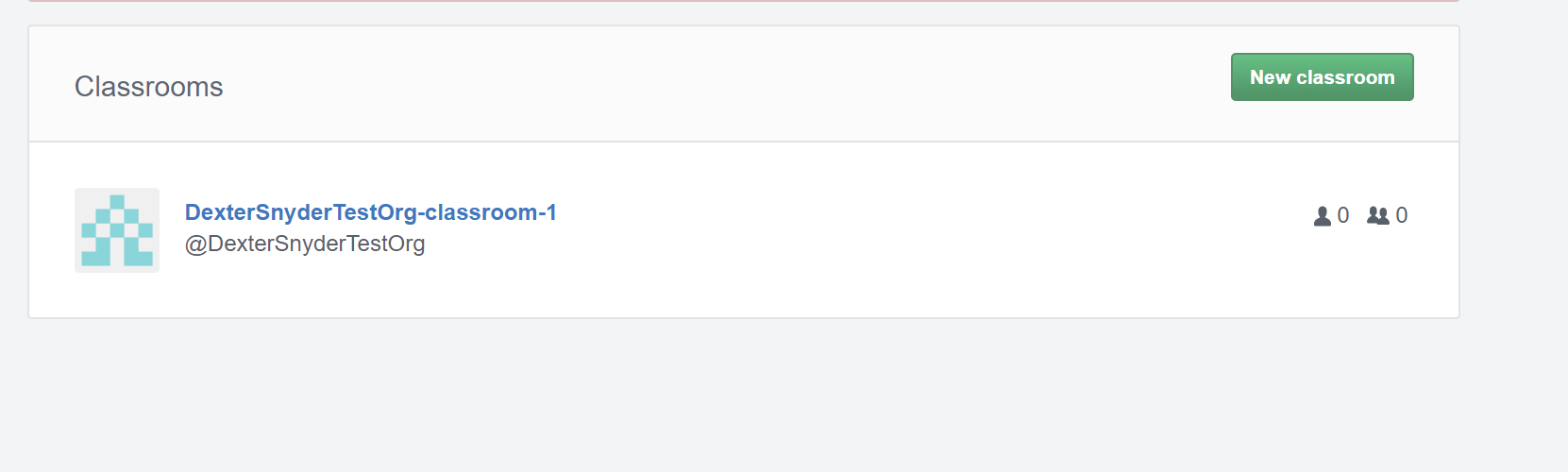
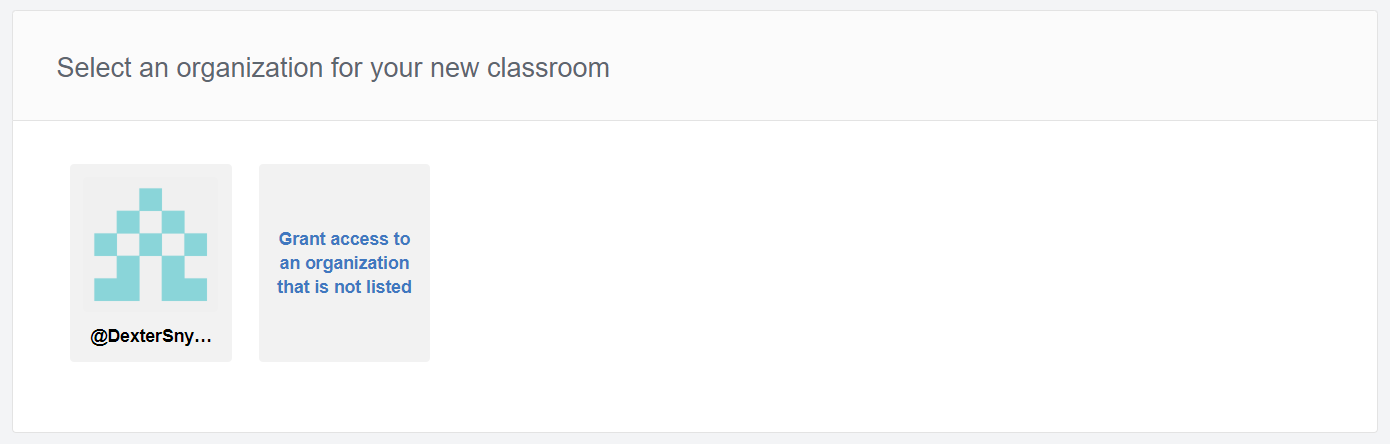
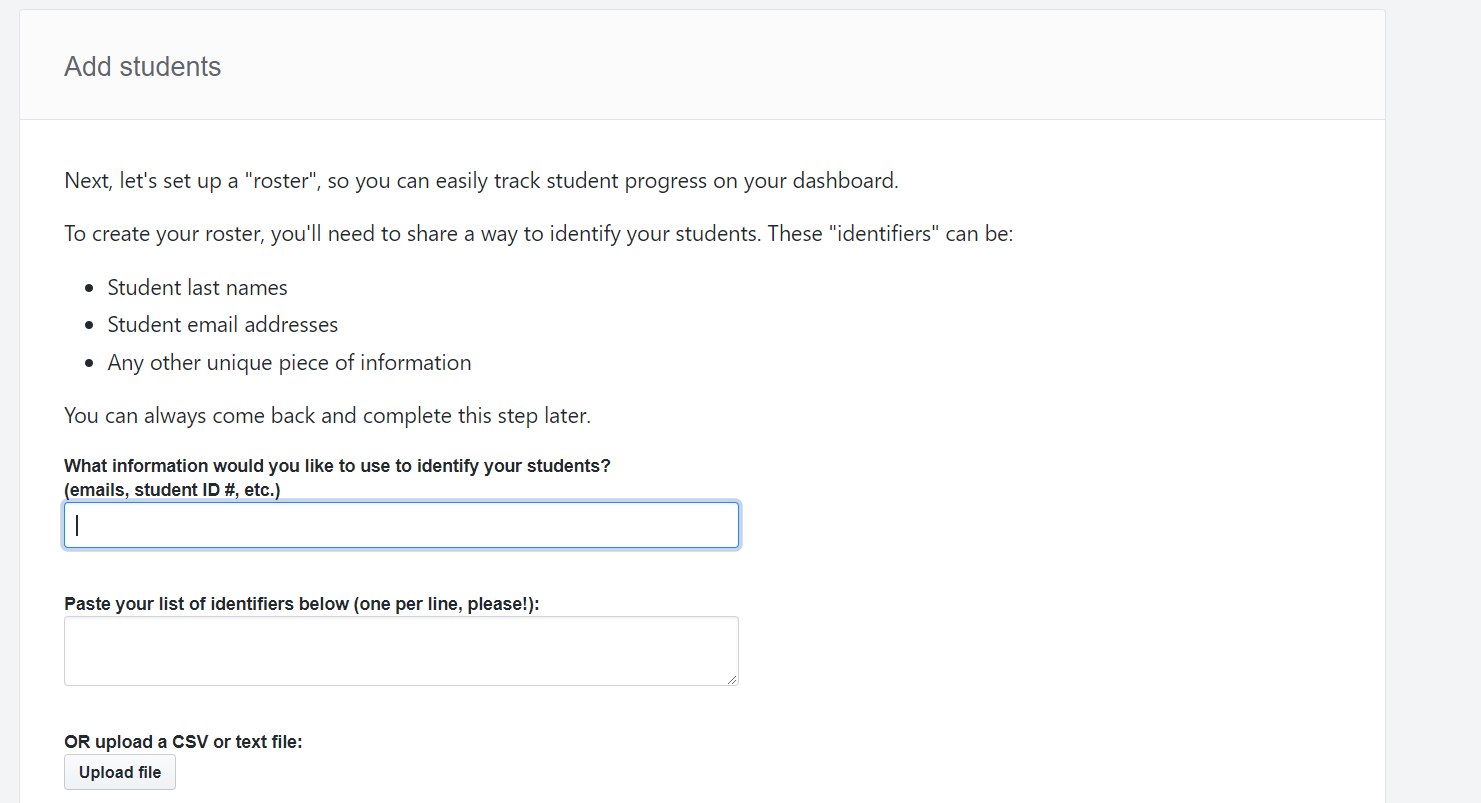
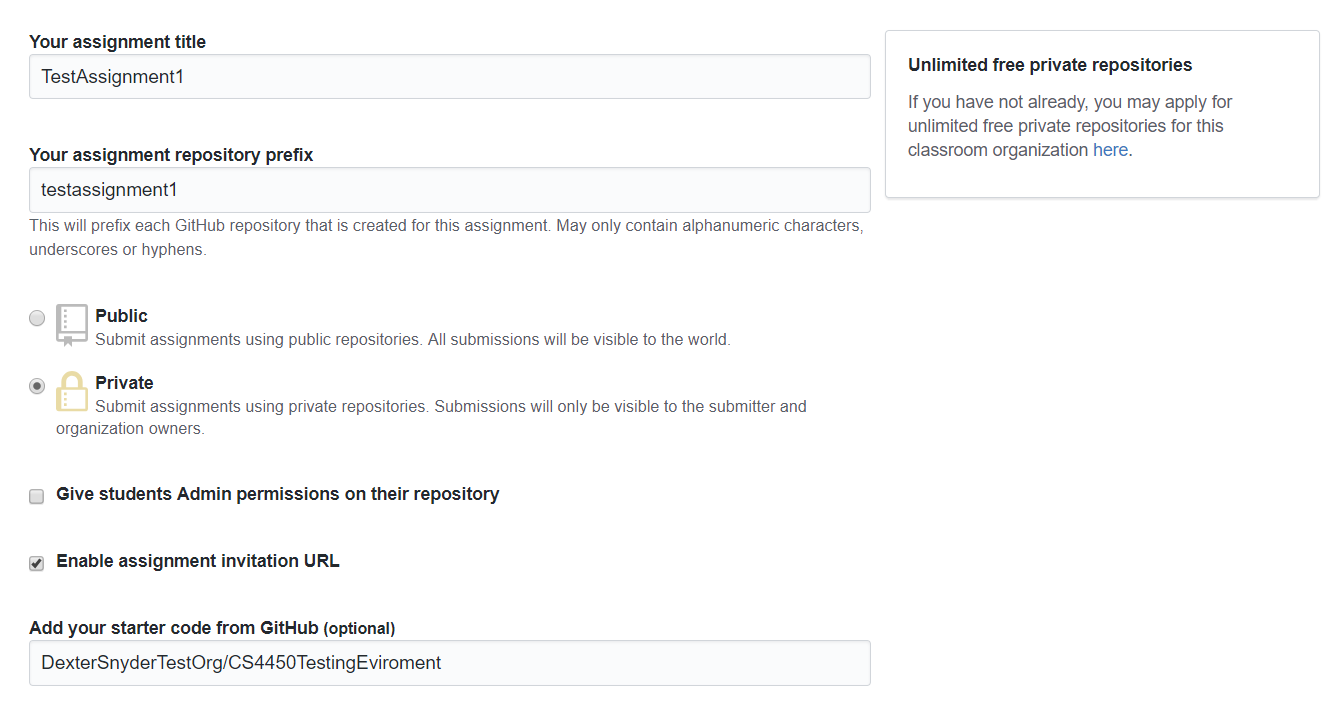
Some lower priority things that could be done are:

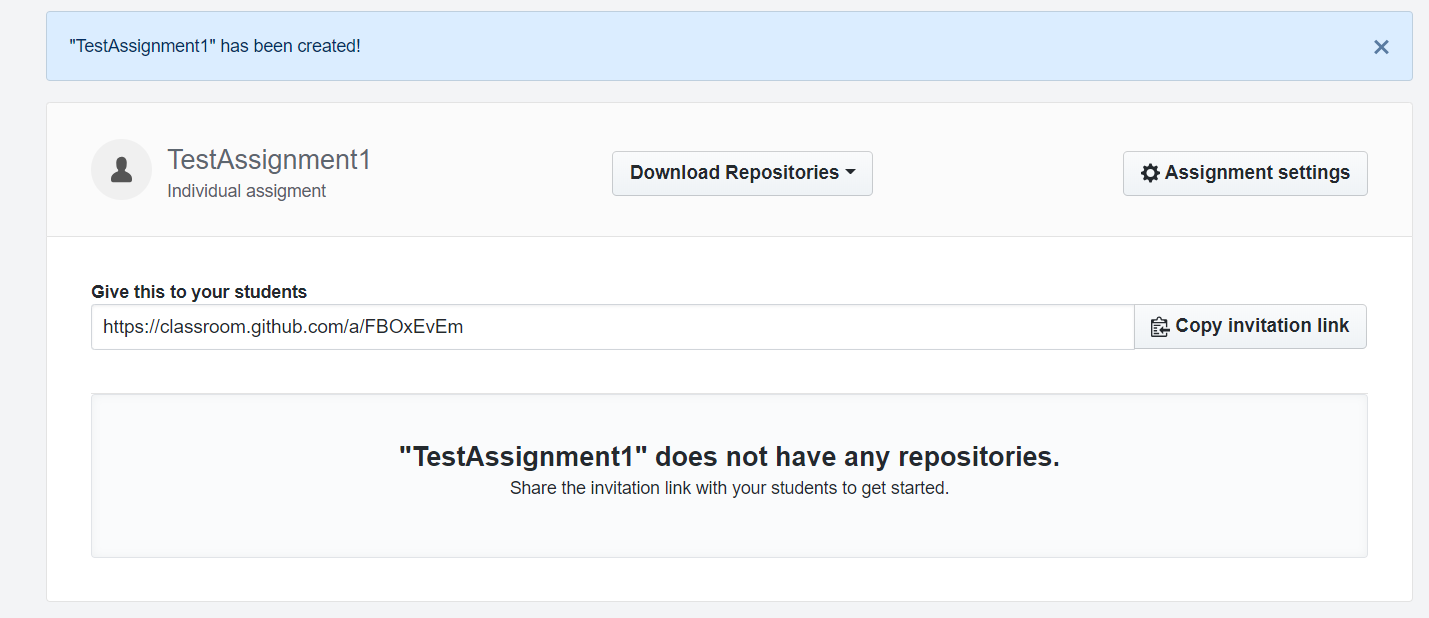
* add more anti-cheating features (e.g watermarking)
* Compile and run student's code asynchronously

# GitHub Classroom Instructions

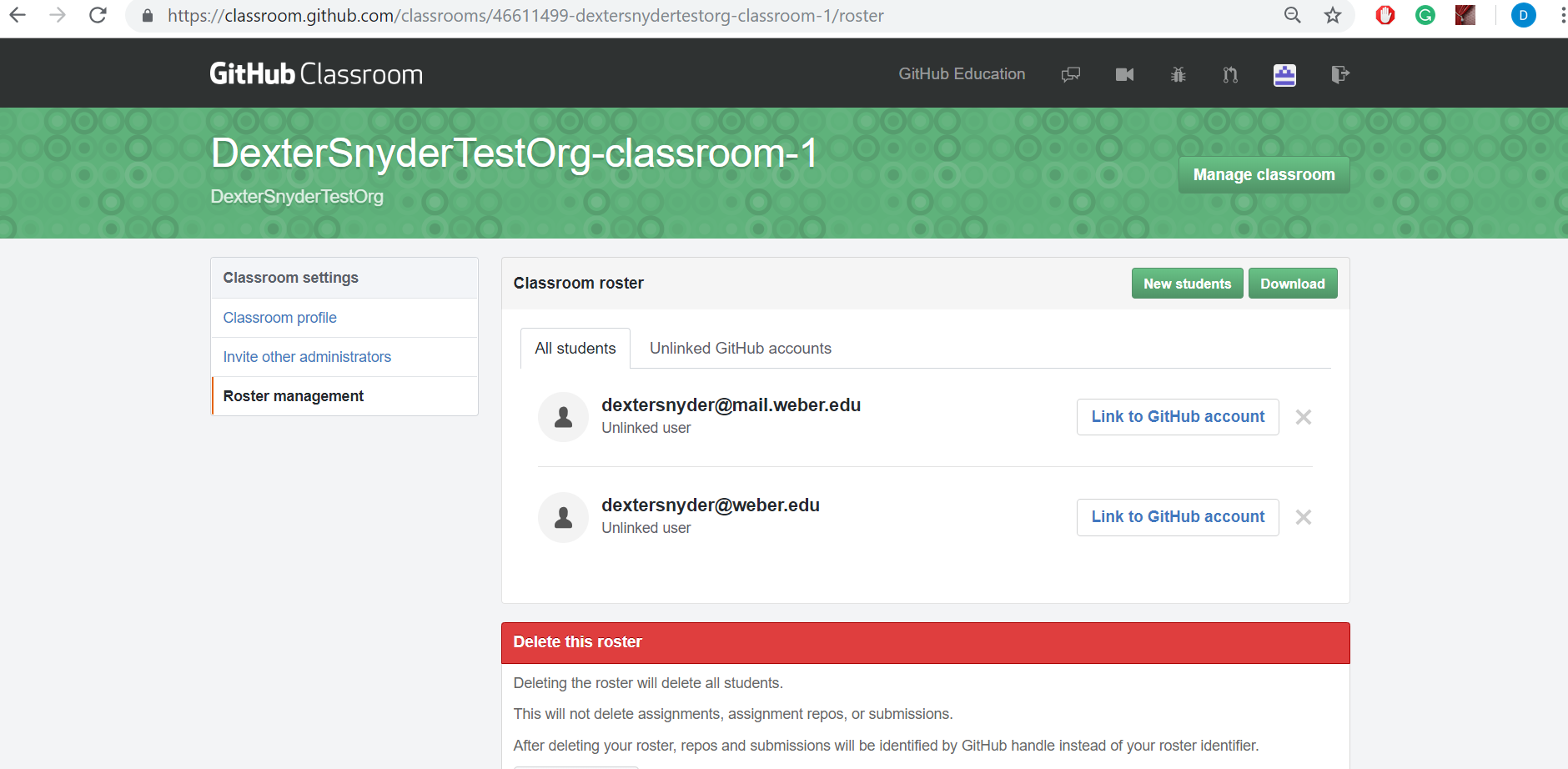
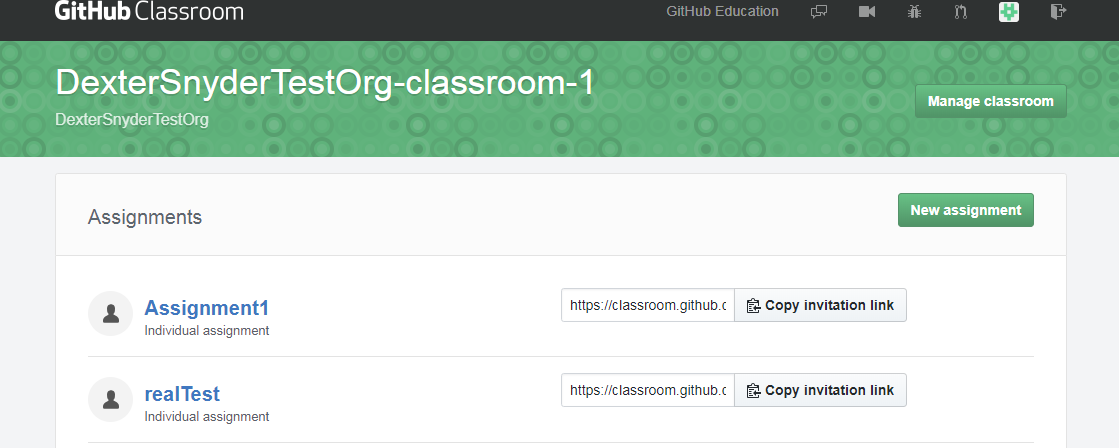
## Instructor Instructions

### GitHub Classroom Setup

1. Go to github.com and sign in
2. Create a new organization
3. Enter the information and choose the free plan (note that all classrooms you create will be under the organization name you choose)
4. Create a new repository for the starter code of the assignment you what to add. (Make sure to add \*.opendb to the gitignore file, or Visual Studio will break the auto commit)
5. Go to classroom.github.com, sign in, and authorize GitHub
6. Create a new classroom
7. Select the organization you created
8. Choose the identifiers students will use to identify themselves when linking their GitHub account.
9. Create an assignment
10. Click create an individual assignment
    1. Enter an assignment name & assignment repository prefix. (It is recommended to make these exactly the same, because in the Aces program when it asks for the assignment name it is really asking for the assignment repository prefix.)
    2. Add the starter code repository you created in step 4
11. Now that you have created an assignment it will show you the URL to give to the students



### Downloading the Class Roster

1. Go to manage classroom.
2. Go to roster management then download roster.

## Student Instructions

Students must have or create a GitHub account.

* Students can create an account by following the instructions below:
  + First go to github.com
  + Next, on the top right corner of the page, click on Sign Up or they can click on Sign Up for GitHub in the center of the homepage
  + Then, create a username and password
  + Finally, after that, choose the free account option for the subscription
* This must be done in order for students to submit their assignments and receive a grade for their work.
* Once students’ accounts are set up, the professor will email the students an invitation to join their GitHub Classroom.
  + Students will click on the link which will take them to a GitHub Classroom page where they will locate their name and add themselves to the classroom.
  + Once they are added to the classroom the students will then be able to clone the repos to their assignments when the professor sends out the links to clone them.
* Students will then clone the repository, and complete the work on their local machines.
* In order to submit the assignment, the student will complete the following steps
  + Git add
  + Git commit
  + Git push

# Aces

## Student Use

1. Run the program. The homepage is the students page.
2. Upload the students files.
   1. Click browse under “Assignment Code”, and upload all the student assignment files (the cpp and optional header file).
   2. Click browse under “Unit Test Code”, and upload the students unit test.
   3. Note: you should also be able to paste the code into text boxes below the browse buttons
3. Click “Upload and Run”. The output will be displayed in the output box

## Instructor Use

To go to the Professor screen, add *ProfScreen to the URL when you run the program. (e.g. https:*/localhost:5001/ProfScreen). The reason for this is so that only those with the URL can access this page.

### Creating a Class

1. Click the create class button.
2. Enter the organization and class name (this is the organization and classroom you created in github classroom).
3. Upload the class roster (see the “Downloading the Class Roster” section under the Github Classroom instructions.
4. Click create class.

### Deleting a Class

1. Choose the class you want to delete from the drop-down box next to the delete class button.
2. Click delete class.
3. After you are taken to another screen saying the class has been deleted, click the browsers back button.

### Running the Program

1. Choose the class from the drop-down.
2. Enter the assignment name (remember this is actually the assignment repository prefix. If you go to the assignment in GitHub classroom you can find this in the URL.)
3. Enter the security key the instructor unit test uses to grade the assignment.
4. Enter the GitHub username & password for the GitHub classroom account.
5. Upload the instructor unit test (not the student version).
6. Click the “Check” button.
7. On the assignments screen it will show a rating showing how likely it is the student cheated (the section “Ratings (colors)” details what each color means) and the score they received. If the score is 0/0 the assignment did not compile.
8. Click on details next a student to see a list of reasons why they got that rating.
9. Click “Return to Assignments” to go back to the assignments screen (the browsers back button does not work)

## Ratings (colors)

There are 3 levels of results, Green (lowest), Yellow, and Red (highest).

**Red**

These are students that have a high likelihood of having cheated

Rules

* 3 yellow marks
* Less than 2 commits
* 2 different authors once the students commits begin
  + Right now, ACES looks for the author “Default”. This would be good to change, but will require a fair bit of refactoring

**Yellow**

These students have a medium likelihood of having cheated

Rules

* Less than 5 commits
* The average time between commits is lower than 2 standard deviation below the average

**Green**

These students have a low likelihood of having cheated. A green rating means that no yellow or red ratings were given, or that they were forcefully overwritten (rare/ not implemented).

## Code Explanation

### Controllers

**AssignmentController.cs**

The AssignmentController takes the info from the AssignmentService class and handles displaying it to the assignments page of the ProfScreen.

**ProfScreenController.cs**

This controller allows the user, the instructor, to put in all the info related to their GitHub classroom, store it to a database, or pull from a database. Once they have put in all the info it starts the program to clone, compile, run, grade the students’repos, and then displays that info. The ProfScreenController will delete the instructor unit test file that was uploaded when done.

**StudentPageController.cs**

This controller uploads the submitted files, the student code and the student unit tests, or allows the student to enter their code in a textbox and creates files from it. It then compiles and runs the code and displays the output. It doesn’t access the database or any of the other functions like the ProfScreenController does. It just allows the students to compile and run their code in the same environment as their professor.

### Services

**Analyzer.cs**

The analyzer class in this file handles checking the student’s repo for cheating. It checks the code for things like low number of commits, change in author, and assigns a color to represent the chance the student cheated. Uses the GitInterface class.

**AssignmentService.cs**

The assignment services class takes the current user log in info and passes the students and classroom info to the analyzer class. This class then takes what the analyzer has reported, formats it correctly, and puts that info in a list, assignments. This is the class that allows the program to access the assignments.

**ClassRoom.cs**

The classroom class holds the GitHub classroom info. It also takes the list of students from the csv file and stores them. It also calls the gitinterface class to start automatically cloning the students GitHub repos.

**GitCommit.cs**

This GitCommit class is used for testing purposes and is not used in the normal running of the program. This class simulates a student’s commits.

**GitInterface.cs**

The GitInterface class is uses the Linux terminal to automatically clone all the students’ repo in the classroom to the student repo folder. It also grabs the git info like author and number of commits for use later in the program.

**Score.cs**

This simple class is used to store the number of correct and incorrect tests and formats the display with an overridden ToString method.

**Student.cs**

This class hold all the student’s info like name, rating, GitHubUserName, and so on. It also handles setting and getting of all the student’s info.

**SystemInterface.cs**

This class takes the student repos and compiles and runs their code. It uses the g++ compiler and deletes the repos when done. The program pipes the info from the terminal and will display all output from standard out and standard error. It then takes that output and calculate the number of correct and incorrect tests.

**UserInfo.cs**

This class handles the GitHub user of the user of the program, the instructor, and test to make sure they are able to log in based on the info they game the program.

### Models and ViewModels

**Classroom.cs**

Instance of a classroom holds classId, orgName, and className.

**IClassroomRepository**

Interface class for ClassroomRepository.

**ClassroomRepository.cs**

Code to retrieve an instance of classroom from the database.

**AppDbContext**

Not necessarily a model but holds information on what classes will be interfacing with the database. In this case classrooms and students.

**Student.cs**

Holds and instance of a student. Contains id, classId, githubUrsName, githubEmail, name.

**IStudentRepository.cs**

Interface class for StudentRepository.

**StudentRepository.cs**

Code to retrieve an instance of Student from the database.

**ProfScreenViewModel.cs**

Holds information that is passed to and from the ProfScreen pages.

**StudentPageModel.cs**

Holds information that is passed to and from the StudentPage pages.

# Database

**Tables**

Students: table to hold students that belong to a classroom. Keeps record of their github username and email.

Classroom: holds the classId, ClassName and orgName for a class.

**Explanation**

For our database we use sqlite which is just part of the project files (AcesDatabase.db). We created the database with a code first approach using Entity Core Framework. If changes or additions to the database need to be made it is important to make the appropriate updates to the code then update the database using migrations.

# Unit Test Files

For every assignment there must be two unit tests the instructor version that is used when running Aces, and a student unit test.

ACES\_Cpp contains a cpp function that will automatically do a git commit each time it is run. In order to work, this must be run each time the student runs the program. The best way to do this is to include it in a main method that the student doesn’t modify.

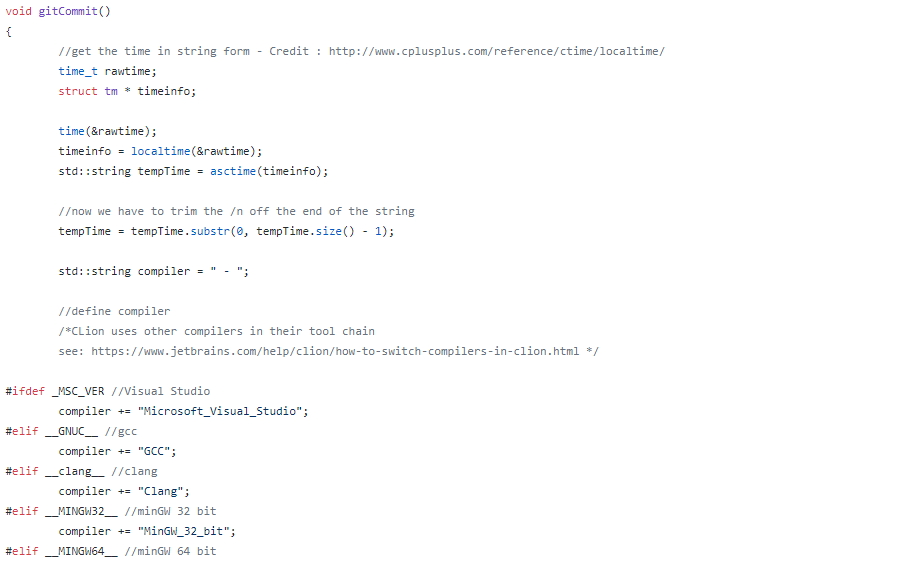
There are 3 files that should be in a student assignment:

* .Cpp file where the students do their work
* .H file (optional) for the students to work in
* Unit test file. This should contain the main() method, the unit test for the program, and the git commit function

In addition to the files given to the students, the instructor will need a unit test file with several modifications

* It should not contain the git commit function
* A successful unit test should output the string “Passed [security code]” where the security code is a user defined string
* It should contain a main() method

When ACES is run, the student unit test is deleted and the instructor version is copied in its place.



# To do

* Save assignment data in .net core Session data
  + Currently the assignment data is stored in a static variable inside the AssiginmentServices Class. This should be deleted and the data should instead be stored in session data, so that the program can handle multiple instructors using it concurrently. This should also fix the problem of not being able to click the back button on the assignment details page.
* Validate input
  + The program only validates the login. It should check that all other input (assignment name, security key, etc.) is valid
* Secure login
  + Use OAuth to login with a token or find another way to login without using the instructors plain text password to login.
* Handle problems with student's code
  + Aces needs to be able to handle any problems with the student’s code that would prevent the program from finishing (e.g. infinite loops).
* Compile and run student's code asynchronously
  + To make the program more efficient, and so it does not wait for too long waiting on one students code that has problems.
* Permissions for student uploads
  + Anything the students upload should not have the ability to damage or modify the server, or anything on it.
* Make it so students can’t access the Professor Screen
  + The students need the URL to access the Professor screen, but if the get that URL they can add & delete classes. Maybe add a login page to prevent students from getting access to it.