# COGS

## Central Online Grading System

Final Document

## Team

Forrest Scott – Computer Engineering
Daniel Riechers – Computer Engineering
Zachary Lones – Computer Engineering
Daniel McDonough – Computer Engineering
Kathryn Widen – Computer Engineering

## Client and Advisor

Dr. Thomas Daniels

Team Dec15-21

# Table of Contents

# Project Design

## Problem Statement

Grading code submitted through blackboard is tedious and time consuming. Blackboard does not display .c files, meaning that TAs who are grading student code must download and compile each individual submission. With a large amount of students in a course, this becomes very time consuming.

## Solution

Create a website where students can submit assignment code. This website will provide a streamlined single page grading process to instructors that will automate parts of the grading process so that Instructors are only responsible for grading student code. The website will include interfaces for students to customize their code arguments and view their compile and run results. Lastly, the website will be equipped with MOSS modules that will aid Instructors in cheating detection.

## Definitions

COGS: Abbreviation for Central Online Grading System
Server: Hosts grader, web interface, and database
Grader: Compiles and runs student code, is not a human. Grader does not assign scores, it generates reports
Report: The output of the compiler and code. This includes student source code, student comments, and student inputs to be used for execution
Student: Person who submits code to COG server, generally a student in the course
Instructor: Person who uses grader report to assign a score to submissions, generally TAs
Professor: Person with slightly more power than Instructor, generally the professor
Submission: A student's single attempt at an assignment
ISO: An image file used to record data onto compact discs.
Kickstart: A service for automating the installation and configuration of Redhat ® systems
CSV file: Comma Separated Values file which allows data to be saved in a structured format
RPM: Redhat ® Package Manager. RPM files are used to package and deliver compiled software

## Overview

COGS is split into two major components, the front end and the back end. The front end is the web server which houses the user interface of COGS. The front end is what all users will interact with, but these interactions will be different for different users. User authentication is handled by Shibboleth, which is the standard ISU authentication scheme. For students, this is where they will view their assignments and submit code. For student code submissions, students will be given a report that will mimic what they would see if they are compiling their code normally, this is done in order to make the system seem more familiar and friendly. For Instructors, the main feature of COGS is the single page grading. Instructors will be given all information required for grading a student submission on one page. This page will include the student's source code and executables, student comments, and tools to assign a grade and give feedback to the student. There are other functionalities for instructors as well, such as MOSS cheating detection, and managing students. For professors, this is where they will create assignments and manage different aspects of the course, such as sections, students, and instructors. The back end of COGS is responsible for safely compiling student code and generating reports.

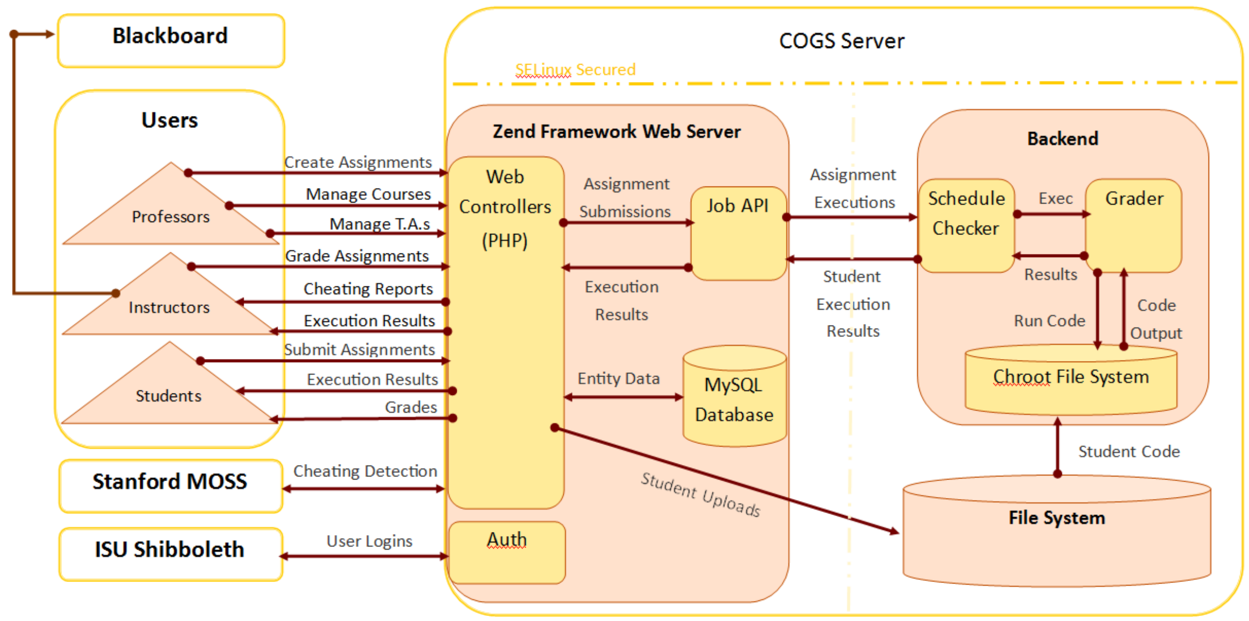


*Figure4.1 – System Overview Diagram*

## System Description

- Front End: The system will be accessible through a webpage hosted on ISU servers. The front end will have four distinct users:
    - Student Front End - Any user that is identified as a student will use COGS to upload assignments.

        i. View assignment details
        ii. Submit/Resubmit assignments
        iii. View report from grader
    - Instructor Front End - The instructor front end will be viewable by those with instructor privileges. The instructor will be the heaviest users of COGS, and so the system should be set up to make their workflow the smoothest.
        i. Add/remove students
        ii. View/grade submissions that they are authorized to see
        iii. Apply MOSS cheating detection to student submissions that they are authorized to see
        iv. Generate CSV file with student grades
    - Professor Front End – The professor front end will have all the same functionality as an instructor user with added privileges:
        i. Add/remove Instructors that they are authorized to see
        ii. Add/remove/edit Courses that they are authorized to see
        iii. Add/remove/edit Assignments that they are authorized to see
    - Admin Front End - The admin front end will have all the same functionality as a professor user with added privileges:
        i. Add/remove/edit Professors
        ii. Add/remove Admins
        iii. Manage individual Professor privilege
        iv. Add/remove/edit all Courses
        v. Add/remove/edit all Assignments
        vi. Add/remove/edit all Submissions
- Back End: The back end system is responsible for safely compiling submissions and generating reports. The front end delegates the tasks of compiling and running student code to the backend through and http interface

The following diagram shows the process used by our front end web server to generate web pages for the user.

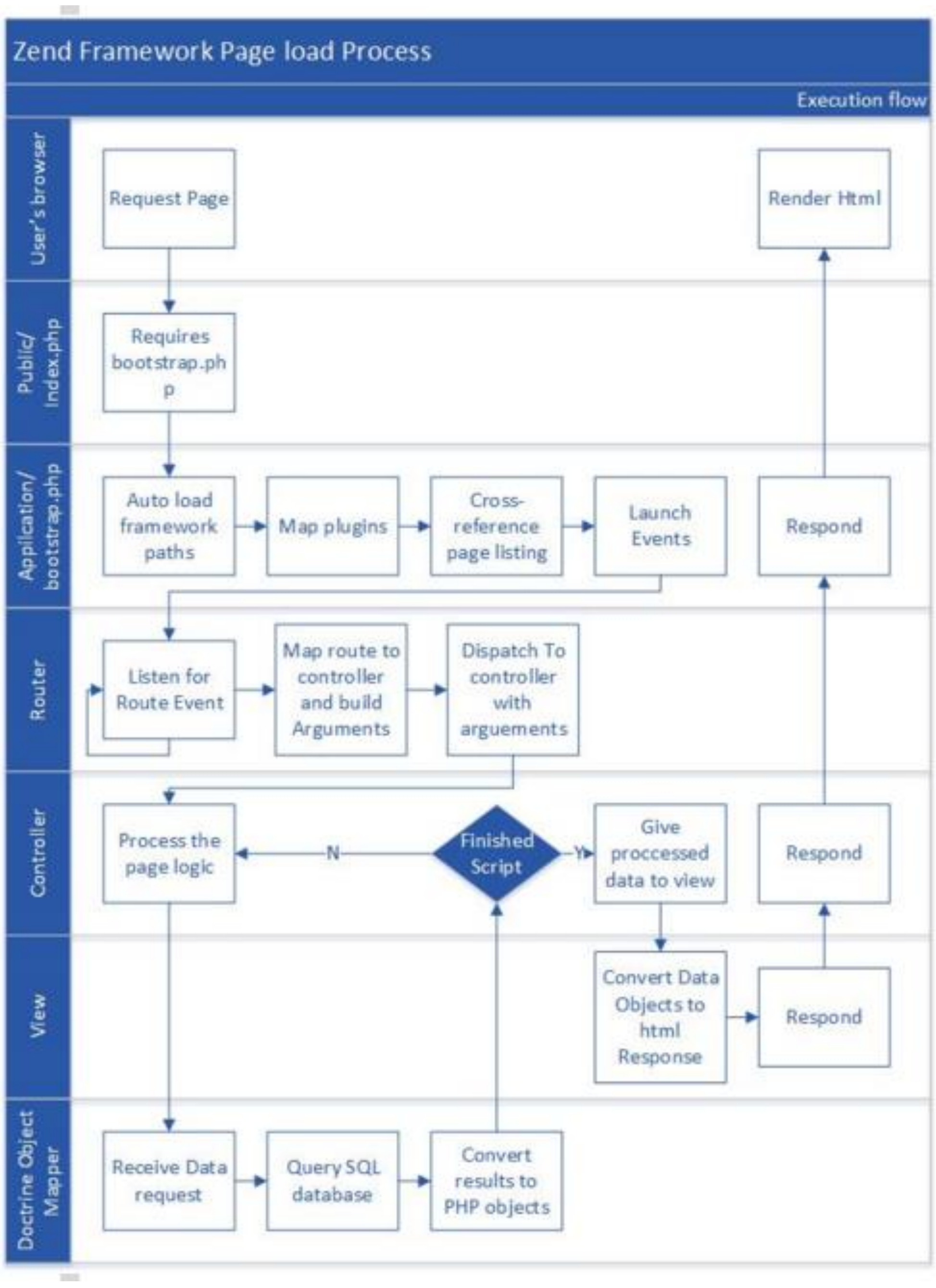

Zend Framework Page load Process
Execution flow
User's browser
Public/ Index.php
Application/ bootstrap.php
Router
Controller
View
Doctrine Object Mapper
Request Page
Requires bootstrap.ph p
Auto load framework paths
Map plugins
Cross-reference page listing
Launch Events
Listen for Route Event
Map route to controller and build Arguments
Dispatch To controller with arguements
Process the page logic
N
Finished Script
Y
Give proccessed data to view
Convert Data Objects to html Response
Respond
Respond
Respond
Render Html
Receive Data request
Query SQL database
Convert results to PHP objects

*Figure 6.1 – Front End Execution Flow Diagram*

## System Breakdown

System Deliverables:

- ISO containing all necessary COTS packages and dependencies
    - Install from ISO automated with Kickstart
    - All system configuration automated with Kickstart
- COGS System:
    - Secure encapsulated environment for running student code
    - Compiler
        1. Support for gcc
        2. Support for clang
    - Mandatory Access Control Policy
        1. Transparent to administrators
        2. Tightly secured environment for running encapsulated code

Software Deliverables:

- System ISO
    - The install process is entirely automated except for network, disk, and password setup.
    - The ISO installs and configures all software to a working state.
    - All software dependencies are added to the ISO as RPMs.
    - Dependencies with no RPM packages are built from source during the install phase.
    - COGS software executables and web content are added to an overlay tarball.

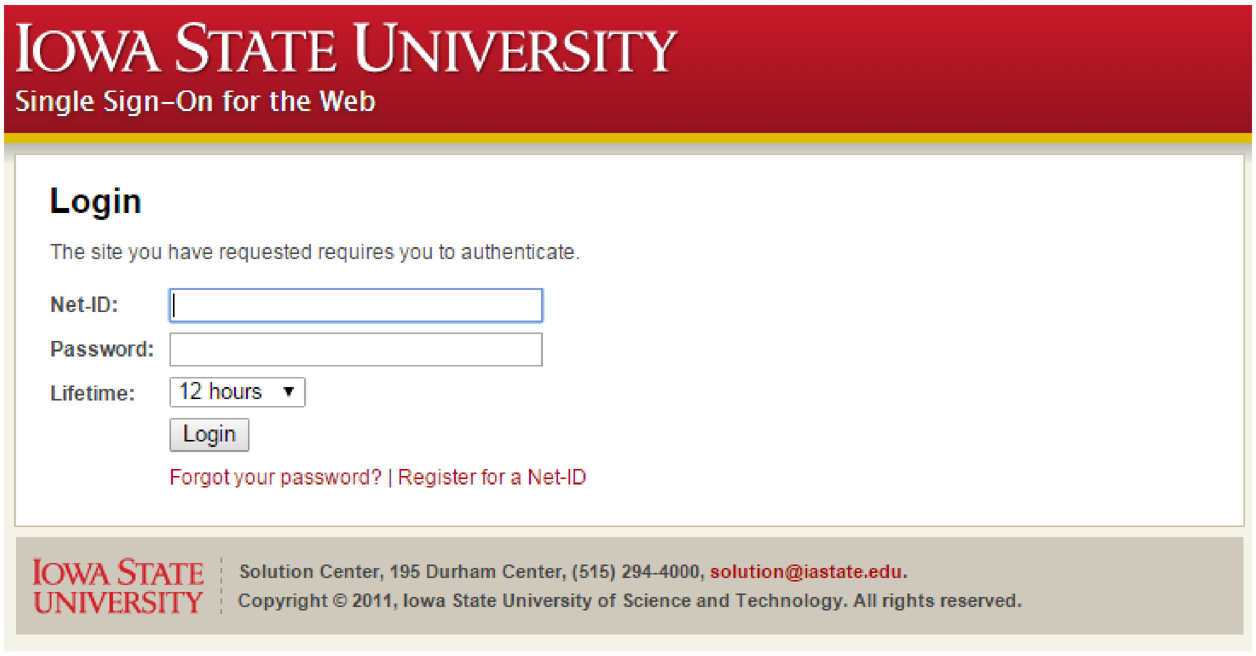
## Testing

Our back end was designed through test driven development, meaning that unit tests were created before coding was started. This ensures that the code reached full coverage of the requirements and gave us a way to test while coding. Our security policy has been thoroughly manually tested. The front end components have been individually tested. We had further plans to test the front end in alpha and beta phases. The alpha test would have a small amount of users looking for bugs and flaws in the system that we would fix and then immediately have beta testing with a large amount of users. Beta testing would find any remaining bugs as well as perform load tests on the system. Due to time constraints, we were not able to meet necessary deadlines to include outside testing. In lieu of this, we have conducted bug hunts ourselves. We realize this is not enough testing that we can confidently say that the system is fully functional and working as intended, so one of our group member has agreed to make sure COGS is up and running in the following semester as an independent study.

# Operation Manual

## Authentication

In order to make it so that students, Instructors, and Professors do not need to create and remember the information for a new account, as well as so that our website is familiar to users and friendly, we use Shibboleth to authenticate users for COGS. Users will log in with their usual netID and password. The login page is shown below.


IOWA STATE UNIVERSITY
Single Sign-On for the Web

Login

The site you have requested requires you to authenticate.

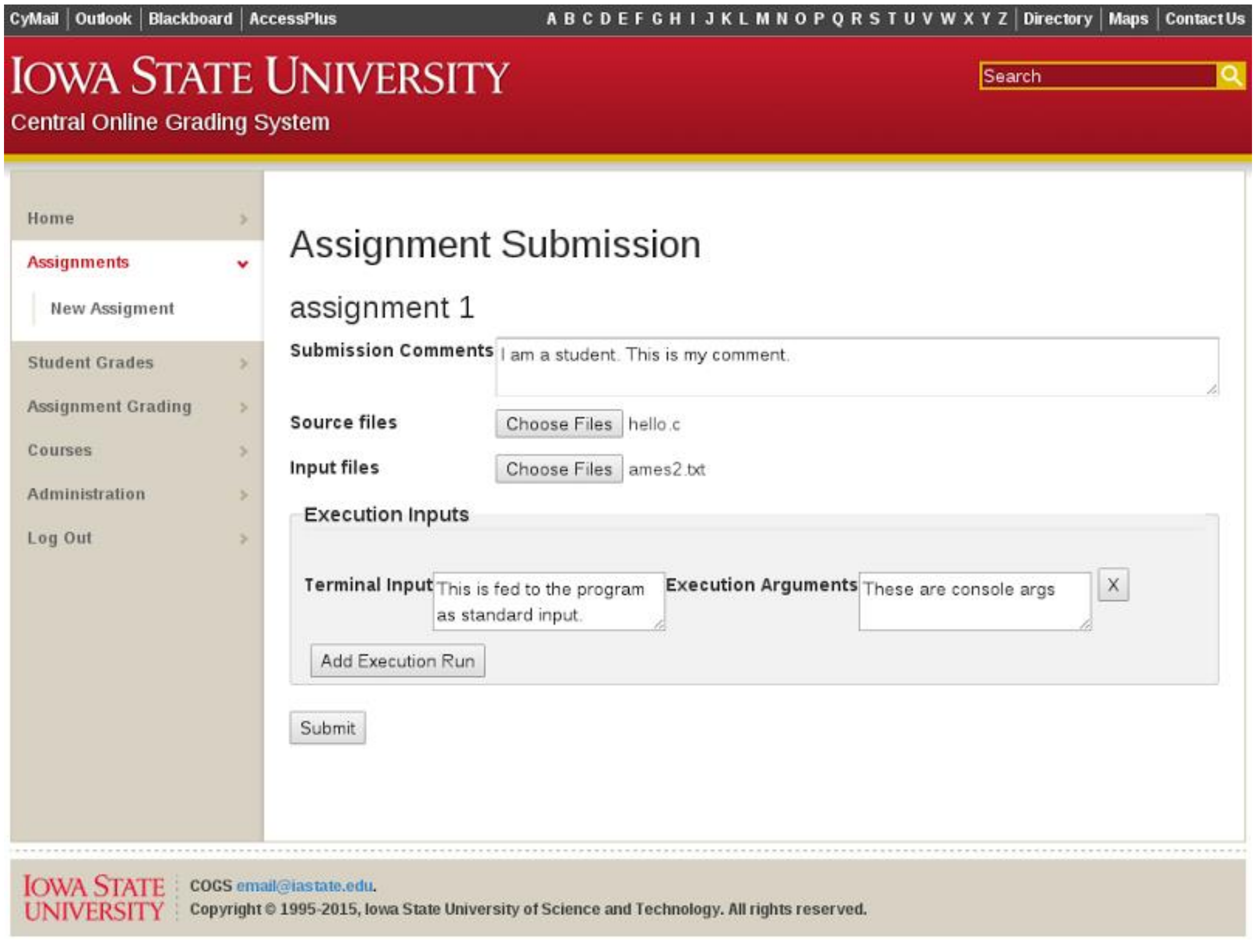
Net-ID:
Password:
Lifetime: 12 hours
Login
Forgot your password? | Register for a Net-ID

IOWA STATE UNIVERSITY
Solution Center, 195 Durham Center, (515) 294-4000, solution@iastate.edu.
Copyright © 2011, Iowa State University of Science and Technology. All rights reserved.

*Figure 8.1 – Authentication page*

## Submitting assignments

Students' main use of COGS will be to submit their code for assignments. After logging in, students will select the course and then the assignment that they wish to submit for, at this point the student will be shown the page below. On this page, there are straightforward places where the student can upload their source and input files, include terminal input and execution arguments, as well as submit any comments they would like the Instructor to see. When they hit the submit button, they will be shown a report of their submission. This report will include any compilation errors and warnings as if they had compiled the code themselves, they will also be shown the output of their execution. The student is able to submit as many times as they like until the assignment due date and only their latest submission will be graded.


CyMail | Outlook | Blackboard | AccessPlus
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z | Directory | Maps | Contact Us
IOWA STATE UNIVERSITY
Central Online Grading System
Search
Home
Assignments
New Assigment
Student Grades
Assignment Grading
Courses
Administration
Log Out

Assignment Submission

assignment 1

Submission Comments: I am a student. This is my comment.

Source files: Choose Files hello.c

Input files: Choose Files ames2.txt

Execution Inputs

Terminal Input: This is fed to the program as standard input.

Execution Arguments: These are console args

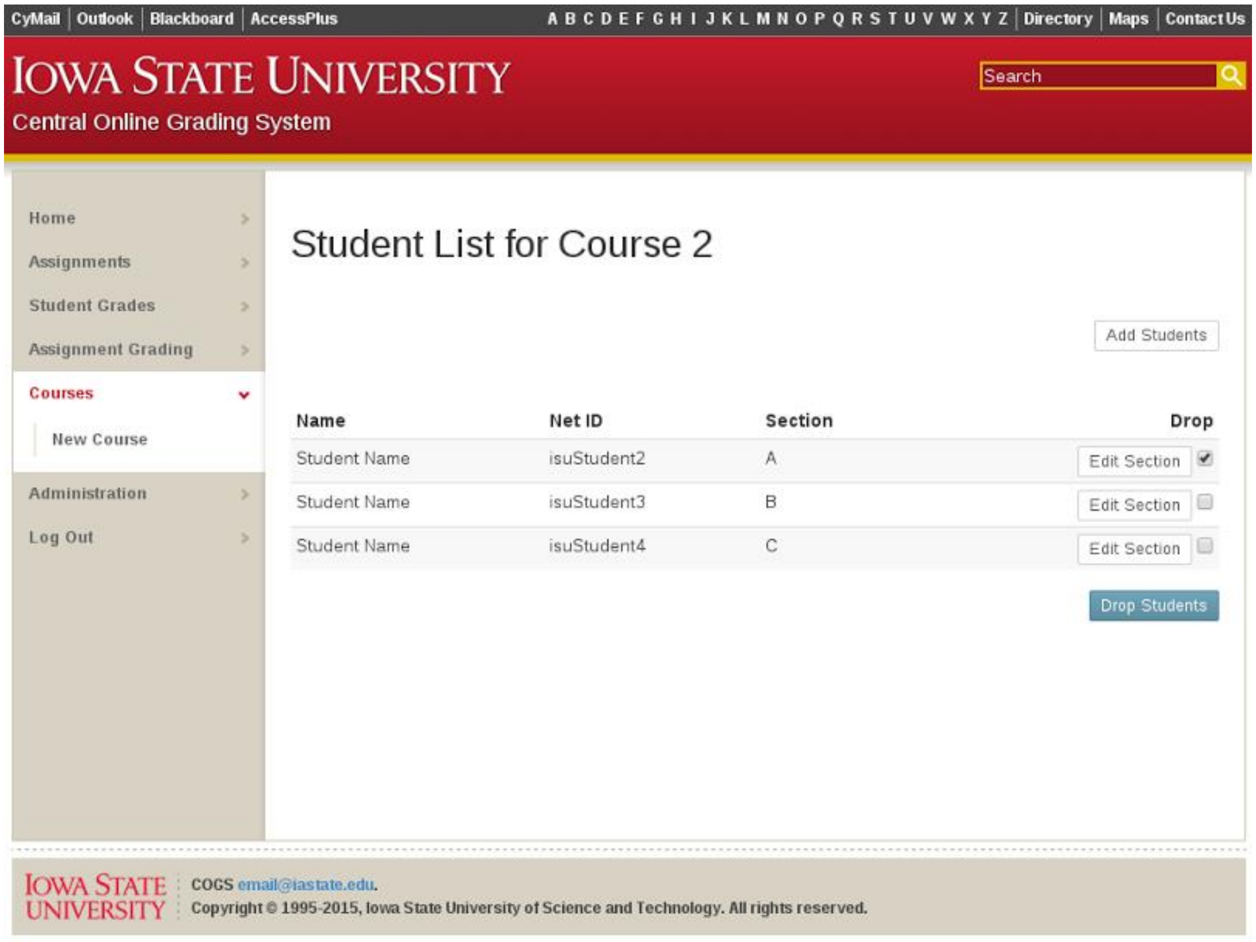
X
Add Execution Run
Submit

COGS email@iastate.edu.
Copyright © 1995-2015, Iowa State University of Science and Technology. All rights reserved.

*Figure 9.1 – Assignment Submission page*

## Add/remove/edit Users

Higher level users will be able to add/remove/edit other users to courses/sections etc. (an instructor can add students, a professor can add instructors and students, admins can add any users) Shown below is what an Instructor would see when adding/removing/editing students in sections. Clicking the checkbox next to a student and then clicking "drop" will remove the student. The Instructor can add a student by clicking add and then entering the student's netID. The Instructor can edit the student's section by clicking the Edit Section button.


CyMail | Outlook | Blackboard | AccessPlus
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z | Directory | Maps | Contact Us
IOWA STATE UNIVERSITY
Central Online Grading System
Search
Home
Assignments
Student Grades
Assignment Grading
Courses
New Course
Administration
Log Out
Add Students
Drop Students

Student List for Course 2

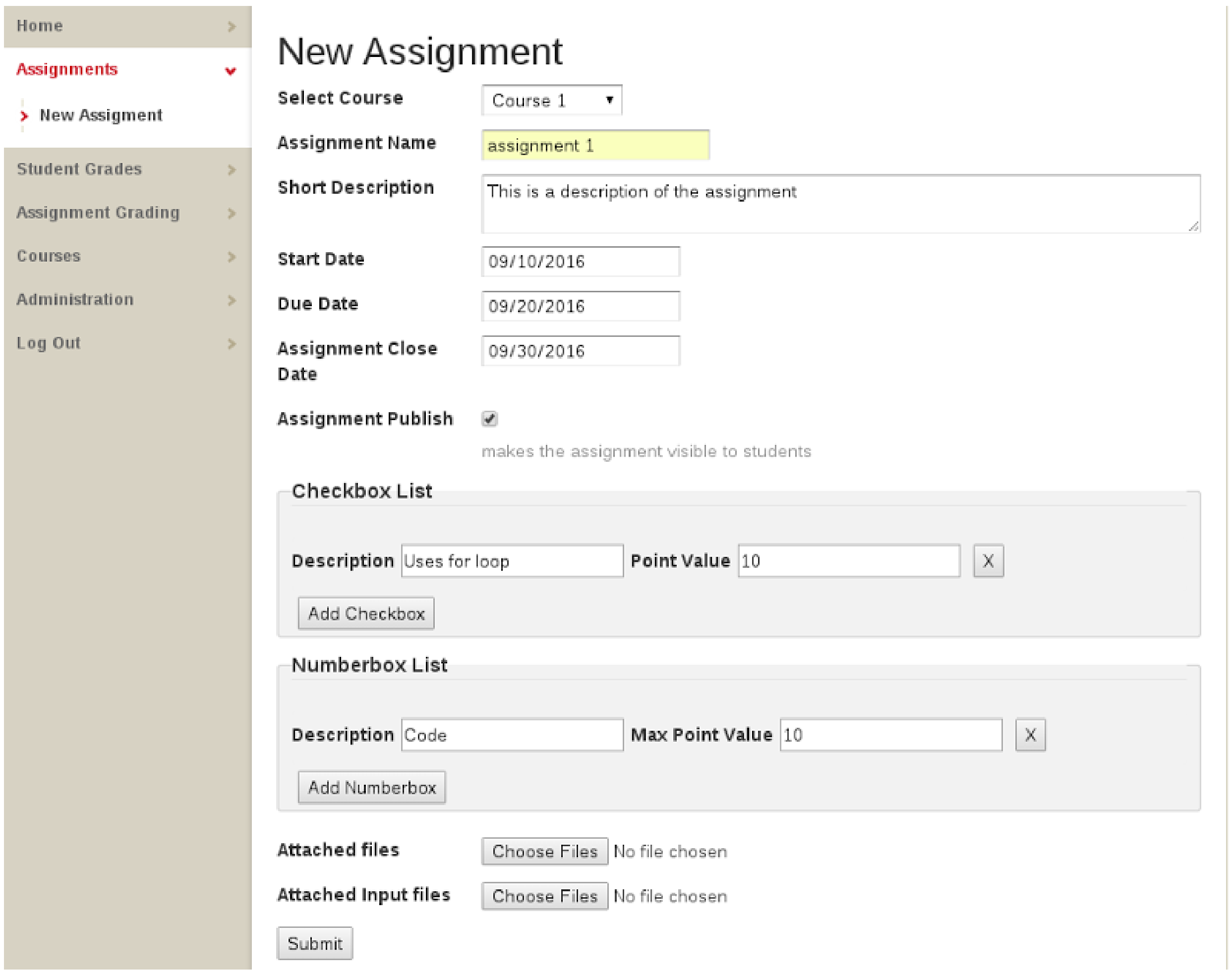
| Name | Net ID | Section | Drop |
|---|---|---|---|
| Student Name | isuStudent2 | A | Edit Section ✓ |
| Student Name | isuStudent3 | B | Edit Section ☐ |
| Student Name | isuStudent4 | C | Edit Section ☐ |

COGS email@iastate.edu.
Copyright © 1995-2015, Iowa State University of Science and Technology. All rights reserved.

*Figure 10.1 – Add/Remove/Edit Students page*

## Creating Assignments

Professors' main use of COGS will be creating assignments that the students will have to complete. Shown below is the straightforward form a professor will need to complete in order create a new assignment. The professor will select the course they are creating the assignment for from a dropdown box which will display the courses they are a professor for. They then fill out the assignment name and assignment description and assignment start, due, and last submission dates. Students will not be able to submit code for an assignment before the start date, their submissions will be counted late if it is after the due date, and submissions will not be accepted after the last submission date. The professor can also fill out any checkboxes and numberboxes they wish to be included, these will be shown on the instructor grading page and are intended to make the grading process as fast and as easy as possible. In the example shown below, there is a checkbox for whether the student uses a for loop in their code or not. When grading, the checkbox can be checked to give the student 10 points for using a for loop, or not checked to give a student 0 points. The numberbox shown in the example is for code quality and a grader can assign up to 10 points. The professor can also attach files, such as an assignment pdf, and also attach input files that will automatically be added to the student's input files for assignment creation.



New Assignment

| | |
|---|---|
| Select Course | Course 1 |
| Assignment Name | assignment 1 |
| Short Description | This is a description of the assignment |
| Start Date | 09/10/2016 |
| Due Date | 09/20/2016 |
| Assignment Close Date | 09/30/2016 |
| Assignment Publish | ✓ makes the assignment visible to students |

Checkbox List

Description: Uses for loop — Point Value: 10 — X

Add Checkbox

Numberbox List

Description: Code — Max Point Value: 10 — X

Add Numberbox

Attached files: Choose Files No file chosen

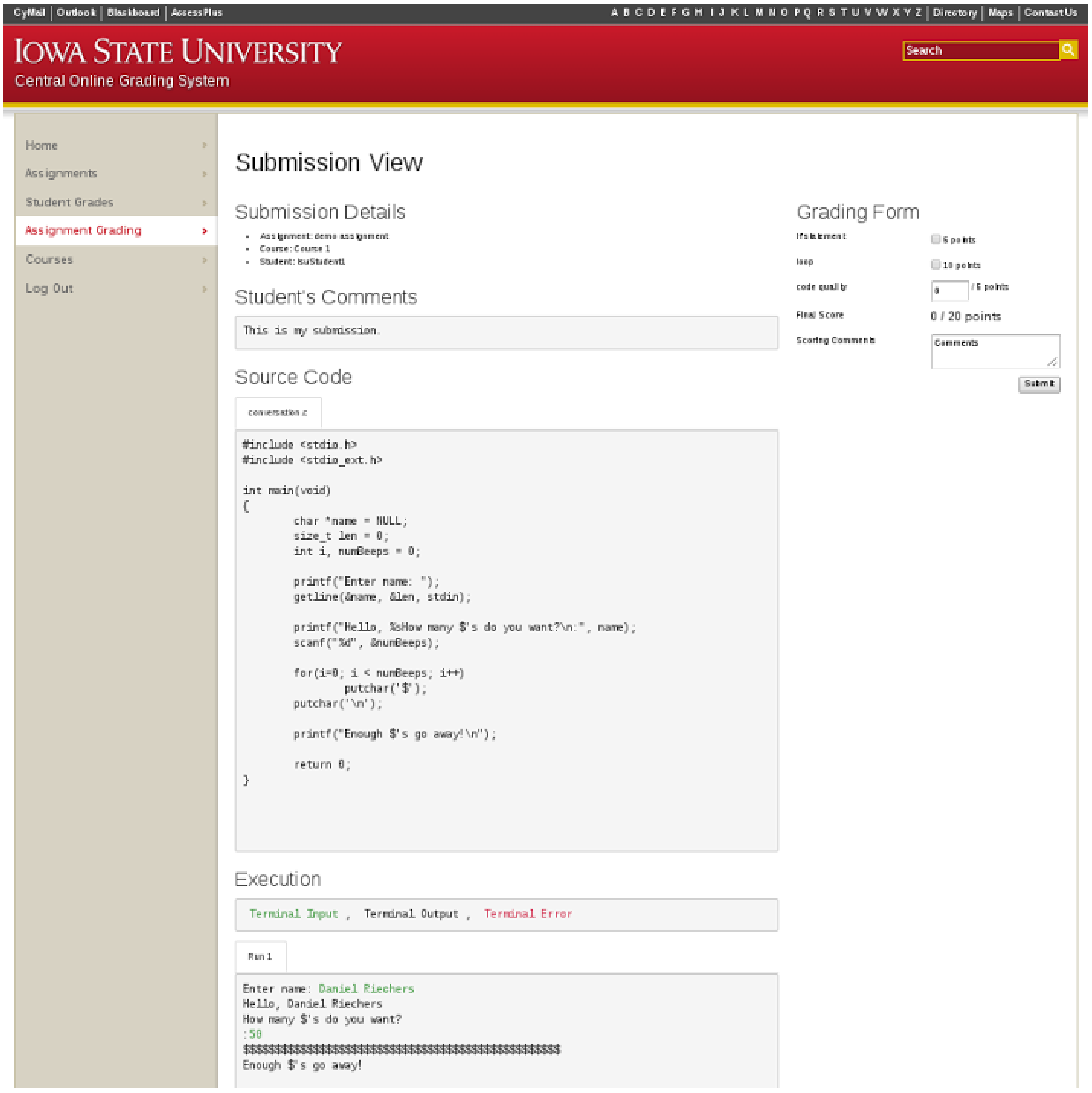
Attached Input files: Choose Files No file chosen

Submit

*Figure 11.1 – Assignment Creation page*

## Grading Submissions

Shown below is the screen Instructors will see when grading code. This page displays student source code, comments, and the execution results of any inputs they provided. With all necessary information on the page, the Instructor can check any checkboxes and assign numbers to any numberboxes that were created with the assignment, provide feedback to the student in the comment section, and then simply click next to start grading the next student's submission.



- Assignment: demo assignment
- Course: Course 1
- Student: isuStudent1

This is my submission.

```
#include <stdio.h>
#include <stdio_ext.h>

int main(void)
{
        char *name = NULL;
        size_t len = 0;
        int i, numBeeps = 0;

        printf("Enter name: ");
        getline(&name, &len, stdin);

        printf("Hello, %sHow many $'s do you want?\n:", name);
        scanf("%d", &numBeeps);

        for(i=0; i < numBeeps; i++)
                putchar('$');
        putchar('\n');

        printf("Enough $'s go away!\n");

        return 0;
}
```

| | |
|---|---|
| If statement | ☐ 5 points |
| loop | ☐ 10 points |
| code quality | 0 / 5 points |
| Final Score | 0 / 20 points |
| Scoring Comments | Comments |

Terminal Input , Terminal Output , Terminal Error

```
Enter name: Daniel Riechers
Hello, Daniel Riechers
How many $'s do you want?
:50
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
Enough $'s go away!
```

*Figure 12.1 – Submission Grading page*

# Alternative Versions

Our main goal has always been to streamline the process of grading student code by creating a website that would achieve this goal. The process of website creation has a standard general design that has been proven to work and be efficient. In addition to this, our team had a member who has considerable experience with web design, and as a group we spent a large amount of time creating a project plan that effectively achieved our goals, made sense for our project, and that we knew would work. Due to this, the main functionality and design of COGS has remained pretty much the same throughout the project.

However, not all of our team members had experience with web design, and we did have some differences in ideas while creating our initial project plan. With a clear goal that seemed easily achievable, the members of our group with no web design experience got caught up in the idea of creating this amazing website that would do it all. There were some functionalities that we wanted to add to our website that while they would have been nice to have, where not required functionalities to achieve our goal. Our group member who had previous experience with web design insisted that we only include these in our project plan as “stretch goals”. He was right. The rest of the team had underestimated how long it would take to learn website design and create our main functionalities, and because of this our “stretch goals” were not able to be implemented. Listed below are the stretch goals that did not make it into the project.

## Automated Program Analysis

Our client for the project is a professor for Cpr E 185, an introduction to programming course that teaches the basics of programming in C. He employs the mindset that programming should include a creative aspect and assigns homework such as “spend an hour making a program that uses arrays.” This makes it clear why the main goal of COGS is to streamline the process of humans grading the code. However, we had ideas of this website being used by many professors for different courses, and so we wanted to include some form of static analysis that would be able to either fully automate the grading process of code or make it so that only minimal human effort was required. This was the first stretch goal that we decided would not be implemented. Implementation of a fully, or even mostly, automated grader would have been difficult and very time consuming.

## Unit Testing

Keeping with our goal main goal of streamlining the grading process, our assignment creation dictates what our grading page will look like for that assignment. Professors can include checkboxes and number boxes specific for that assignment. For example, if the assignment requires students to use a “for loop”, a checkbox can be included on the grading page for whether the student uses a “for loop” or not. We had also wanted to include the option for Professors to specify unit tests that would be applied to student code and results would be shown on the grading page. This stretch goal was not implemented mainly due to time constraints.

## Automatic Blackboard Integration

In the beginning, our plan was to have grades that were inputted in COGS be automatically sent to blackboard. However, our client was worried about that a system error potentially overwriting or wiping student grade data which made us rethink how blackboard integration would work. We decided to have COGS generate a CSV file that could be uploaded to blackboard after it was checked for errors.

## Cheating Detection

In programming classes with lots of students, there is reason to be concerned about cheating. Students will often work together on assignments whether permitted or not, but often times students will go so far as to copy and paste code from other students or from online. There are programs that have already been written that will analyze similarities between code submissions in order to catch cheating from copying code. These programs are smart enough to catch cheating even if student change variable names, etc. We had a stretch goal of creating our own cheating detection algorithm and there was talk of it being able to catch cheating from students copying code from online. While it would have been neat to have created our own program, the same functionality could be achieved by using the programs that are already in existence and are known to work, and work well. It made the most sense to make use of this, and so we have implemented use of Stanford MOSS to catch cheating.

# Things Deemed Noteworthy

## Looking forward

There is a large step between completing a software project and making sure that that software is implemented correctly and is fully functional. In addition to this, a lot of our components took longer to implement than we had planned and we did not get to conduct as much testing as we would have liked to. In order to make sure that our goal of having COGS vastly improve student and instructor experience in programming courses, Daniel Riechers will be conducting work as a system administrator next semester to ensure that COGS is up and running.

## Security

One of the many things done by COGS is compiling and running arbitrary, student written code. This presents a very obvious attack vector to a regular user, submitting malicious code. This was the main focus of our security efforts. Other security concerns include: Cross site scripting, SQL Injection, General user permissions, and Buffer Overruns.

A few steps were taken to mitigate the problem of malicious student code. First to prevent students from having full access to the file system, their code is run a chroot jail. This jail contains only the bare minimum files required to run the student code: Runtime libraries, and student / instructor provided input files. All files in the chroot jail have hard coded permissions and the student code is not allowed to change permissions of files in the jail. The next key security feature to prevent malicious student code is a targeted selinux policy. This security policy restricts the calls the student's code is capable of making to the system. For instance, the student code cannot open network sockets, fork new processes, break out of chroot jails, change file permissions, etc. The policy has a whitelist of only the necessary system calls for an intro to C course.

The other security issues were mostly handled by best practice coding. All SQL queries are processed by Doctrine 2, which automatically sanitizes them to prevent things like dumping all the user data in the frontend database. All web forms use CSRF tokens to prevent cross site scripting. During vulnerable phases of the compile and run process, the system switches to a low privilege user. To prevent buffer overruns all the input is validated so it can only run in a defined way on the system, and to further mitigate buffer overruns we compile with canary values turned on. Canary values are put on the end of buffers and checked periodically to ensure the buffer has not been overrun and kills the program if it has.

## Binary Injection

In order to generate a "conversational style" view of student code input/output, buffering needed to be turned off on the student code child process. Disabling buffering on the parent does not disable buffering on the child due to the buffering parameters being reloaded to their default values with the standard c library. When originally looking at this problem, the solution seems to be to have code that runs on the child process to disable buffering, but our child process is the student's code, which we treat as a black box and can't edit.

To solve this problem we injected binary code so that during compilation, their main() is renamed to studentmain() and a new main() function is injected that turns off buffering and then calls studentmain(). This results in the conversation style communication that is needed without editing the student's source code.

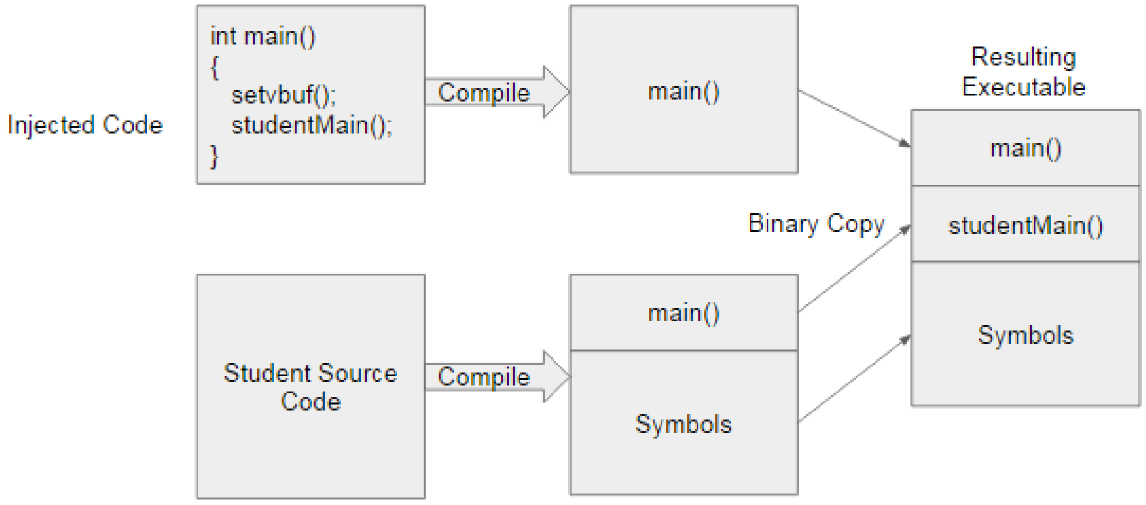


*Figure 16.1 – Binary Injection Diagram*