

CPW AutoGrader

CSCI 370 Field Session 2016

June 20, 2016

Client:

Christopher Painter-Wakefield

Authors:

Michael Bartlett

Harry Krantz

Eric Olson

Chris Rice

Caleb Willkomm

Table of Contents

<i>Introduction</i>	<i>2</i>
<i>Requirements</i>	<i>2</i>
<i>System Architecture</i>	<i>3</i>
<i>Technical Design</i>	<i>3</i>
<i>Testing Tool:</i>	<i>3</i>
<i>Progress Bar:</i>	<i>4</i>
<i>Improved user experience:</i>	<i>5</i>
<i>Admin Interface:</i>	<i>6</i>
<i>Decisions</i>	<i>8</i>
<i>Framework: Ruby on Rails</i>	<i>8</i>
<i>Usage of ACE Code editor:</i>	<i>8</i>
<i>Code Saving/Uploading:</i>	<i>9</i>
<i>Inclusion of Admin GUI:</i>	<i>9</i>
<i>Login required immediately upon accessing to site:</i>	<i>9</i>
<i>Single-page responsive layout:</i>	<i>9</i>
<i>Visual Feedback of Test Results:</i>	<i>9</i>
<i>Results</i>	<i>10</i>
<i>Appendix</i>	<i>11</i>
<i>Deployment Guide</i>	<i>11</i>
<i>Known Issues</i>	<i>12</i>

Introduction

Our client was Christopher Painter-Wakefield, hereafter referred to as CPW, from the Colorado School of Mines. He tasked us with creating a website for use in the Data Structures class he teaches. CPW had been using a bare bones prototype website which lacked many components and functionality that he desired. The prototype website gave students a problem description for which they would have to write code in an external editor and then upload a file to test its functionality. Any edits or corrections to the code required the file to be uploaded again to test. Then the students would upload the same file to the Blackboard website for grading, where either CPW or a teaching assistant would have to download the file and upload it to the first website and test its functionality again to determine each student's grades.

In creating the new website, CPW wished to have a single location where students could read the problem description, write the code, test it, and submit it for grading. He requested that the grading process be simplified by having the website calculate and record the students' grades, which could then be automatically sent to Blackboard or be easily downloaded as a spreadsheet or csv file.

Requirements

The project has two main parts: grading software that executes tests on submitted code and a web interface that allows students to interact with the grading software.

The requirements for the grading software are as follows:

- Ability to execute tests on submitted code
- Results generated for each test
- Runs tests in a way that can't modify system
- Connects to web interface

The requirements for the web interface are as follows:

- Student login
- Grader/Instructor login
- Ability to select lab to work on
- Text editor to edit code in browser
- Ability to save partially completed code for later
- Description of lab requirements
- Ability for instructor to create/modify/delete labs

Non-Functional Requirements:

- Single, responsive web page that holds all assignments/descriptions
- Identifies users through their multipass account
- Passes the user's code into a chroot jail for execution
- Creates a database/stores users results on each assignment

System Architecture

A high level view of the system is shown in figure 1. The basis for all operations is the Rails server which is running under Apache. When a user logs in to the website the Rails server will redirect the user to the Mines Multipass server which upon a successful login will redirect the user back to the website and send corresponding info about the user to the Rails server. The Rails server manages a Postgres database which stores all the data for the website including users, grades, labs, and assignments. The Testing Tool is also running on the server but runs separately from Rails. When a user submits their code to be tested the Rails server executes the command to run the Testing Tool which upon completion returns the results of the test to the Rails Server.

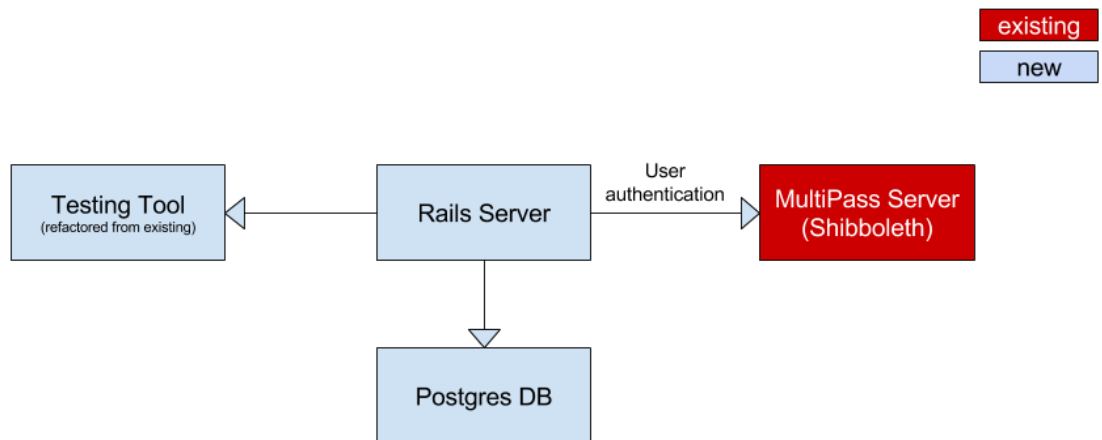


Figure 1

Technical Design

Testing Tool:

We decided that the testing tool was a large enough piece of the project that it didn't really need to be linked in with the Rails framework. We decided to make the testing tool a standalone command line application. By not having to access the testing tool through the web interface made it much easier to debug and to develop.

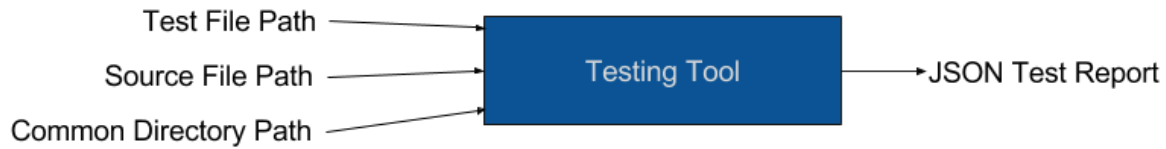


Figure 2: High level view of testing tool

The testing tool uses chroot to create a secure testing environment for the user's code. This ensures they cannot delete anything or mess with the server at all. Also, the code is not run as a super-user so they should not be able to execute any code that could break the server.

Progress Bar:

The progress bar is the primary method for the program to tell the user what their code did. If the code executed successfully, the progress bar will display the results of the unit tests, along with inputs, outputs, and expected values in a popover as shown in figure 3. If the test results are too large for the popover, you can click the bar and it will show a modal view with the full test results as in figure 4. Additionally, any compiler, runtime, and timeout errors will show up on the progress bar as in figure 5 and more detailed error reports are also accessible by clicking on the bar as in figure 6.

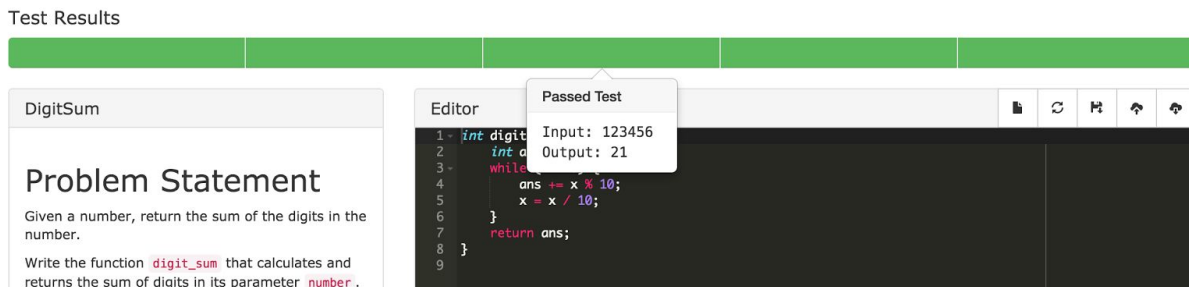


Figure 3

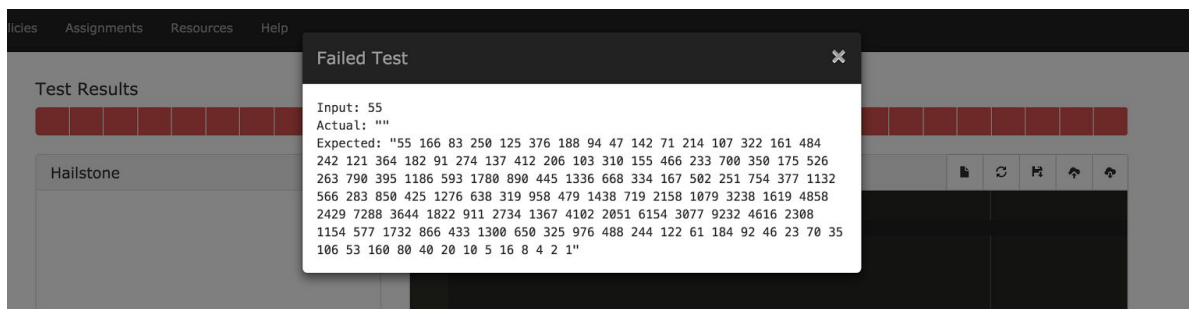


Figure 4

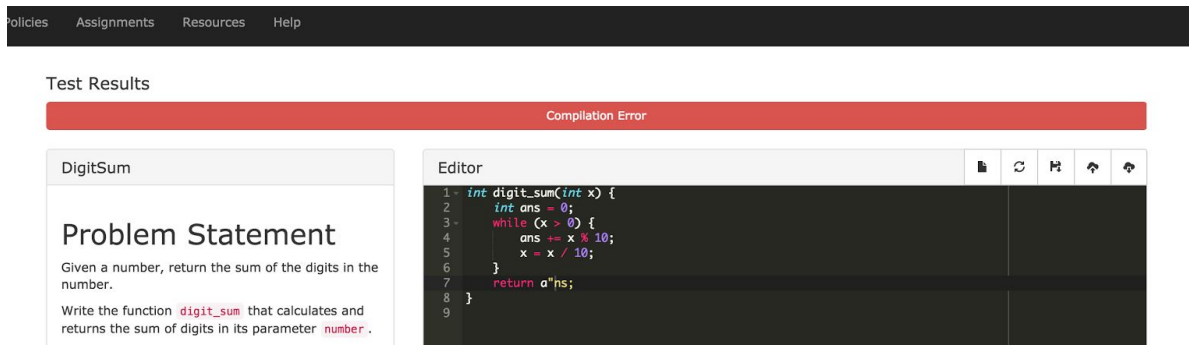


Figure 5

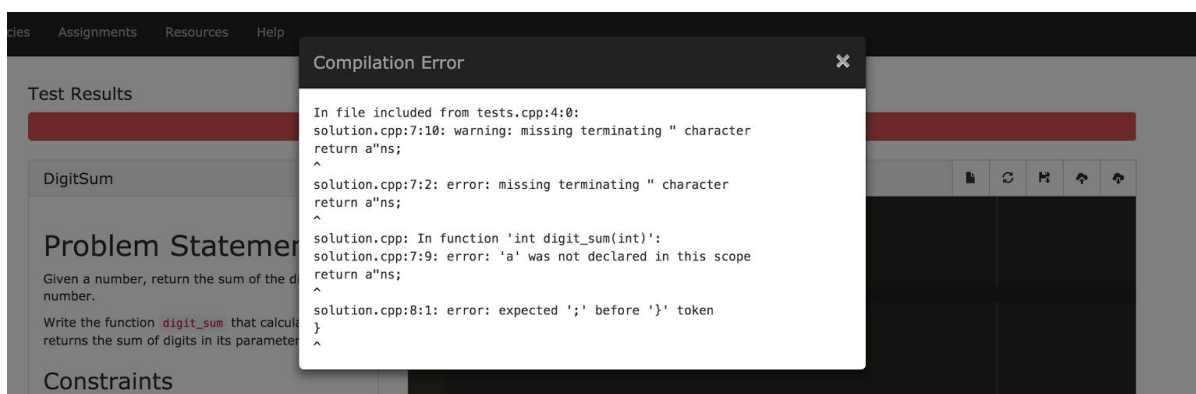


Figure 6

Improved user experience:

We designed the website with a modern clean appearance made possible with Bootstrap, shown in figure 7. At the top the page is the navigation bar from which the students can access info about the class. The assignment page, shown below, features the progress bar on top, the description panel on the left and the code editor in its own panel on the right.

The description for each problem features full html and allows custom formatting and image support. If the description is too long to fit in the panel it will automatically add scrolling. At the bottom of this panel are the "Test" and "Submit" buttons. The "Test" button will run the functionality tests on the code and update the progress bar. The "Submit" button will test the code and then record the student's grade. The database will only be updated if the new score is better than the student's existing.

The code editor includes syntax highlighting to make reading and writing code easier. At the upper right hand corner of the editor panel are several buttons for editing functions. These include buttons(from left to right) to reset the code to the original template, refresh the code to what has been saved, save the code, upload code from a file, and download the current code.

The screenshot shows the AutoGrader web interface. At the top is a navigation bar with the AutoGrader logo and links for Admin, Policies, Assignments, Resources, and Help. The user's name, Harrison Krantz, is in the top right. Below the navigation bar is a 'Test Results' section with 'Edit' and 'View Grades' buttons. The main content area is split into two panels. The left panel, titled 'DigitSum', contains a 'Problem Statement' asking for a function to sum digits, 'Constraints' stating the number is non-negative, and 'Examples' showing inputs 81 and 1075 with their respective outputs and calculations. At the bottom of this panel are 'Test' and 'Submit' buttons. The right panel is a code editor titled 'Editor' showing a C++ function signature: `1 int digit_sum(int number) {`, `2 // fill in code here`, `3 }`, and `4`.

Figure 7


Admin Interface:

The admin interface includes extensive control over the website. From the admin menu grades can be viewed and exported as a csv (figure 8). Individual assignments can be created, edited, and deleted from the admin menu (figure 9-10). Labs can also be created, edited, and deleted. The labs and assignments exist independently so they can be managed easily; assignments can be added or removed from labs without actually deleting and recreating them. Labs also feature due dates and the ability to hide them from the students view. In addition to the full admin privileges there is a teaching assistant user role which includes the ability to view grades but no other admin privileges.

Auto Grader Admin

Dashboard

Home



hkrantz@mines.edu

Log out

NAVIGATION

Assignments

Grades

Labs

Statics

Users

Dashboard / Grades

List

+ Add new

Export













Add filter

Selected items

Filter

Refresh

Export found Grades

<input type="checkbox"/>	Id	Score	User	Assignment	Created at	Updated at	
<input type="checkbox"/>	4	0	Caleb Willkomm	DigitSum	June 20, 2016 13:02	June 20, 2016 13:02	  
<input type="checkbox"/>	3	10	Michael Bartlett	DigitSum	June 20, 2016 12:04	June 20, 2016 12:04	  
<input type="checkbox"/>	2	10	Christopher Rice	DigitSum	June 20, 2016 06:36	June 20, 2016 06:36	  
<input type="checkbox"/>	1	10	Eric Olson	DigitSum	June 18, 2016 18:43	June 18, 2016 18:43	  


4 grades

Figure 8

Auto Grader Admin

Dashboard

Home



hkrantz@mines.edu

Log out

NAVIGATION

Assignments

Grades

Labs

Statics

Users

Dashboard / Assignments

List

+ Add new

Export









































Add filter

Selected items

Filter

Refresh

Export found Assignments

<input type="checkbox"/>	Id	Description	Name	Lab	Created at	Updated at	...
<input type="checkbox"/>	14	-	Encryption	Lab 2	June 20, 2016 11:09	June 20, 2016 12:06	...    
<input type="checkbox"/>	13	-	MemberCheck	Lab 2	June 20, 2016 11:09	June 20, 2016 12:06	...    
<input type="checkbox"/>	12	-	Tourney	Lab 2	June 20, 2016 11:09	June 20, 2016 12:06	...    
<input type="checkbox"/>	11	-	Nesting	Lab 2	June 20, 2016 11:08	June 20, 2016 12:06	...    
<input type="checkbox"/>	8	-	ContinuedFractions	Lab 1	June 20, 2016 10:43	June 20, 2016 10:43	...    
<input type="checkbox"/>	7	-	CommonCount	Lab 1	June 20, 2016 10:42	June 20, 2016 10:42	...    
<input type="checkbox"/>	6	-	Hailstone	Lab 1	June 20, 2016 10:42	June 20, 2016 10:42	...    
<input type="checkbox"/>	5	-	CGRatio	Lab 1	June 20, 2016 10:42	June 20, 2016 10:42	...    
<input type="checkbox"/>	4	-	CountAppearances	Lab 1	June 20, 2016 10:42	June 20, 2016 10:42	...    
<input type="checkbox"/>	1	<h2>Problem Statem...	DigitSum	Demo Lab	June 17, 2016 13:29	June 20, 2016 10:50	...    

10 assignments

Figure 9

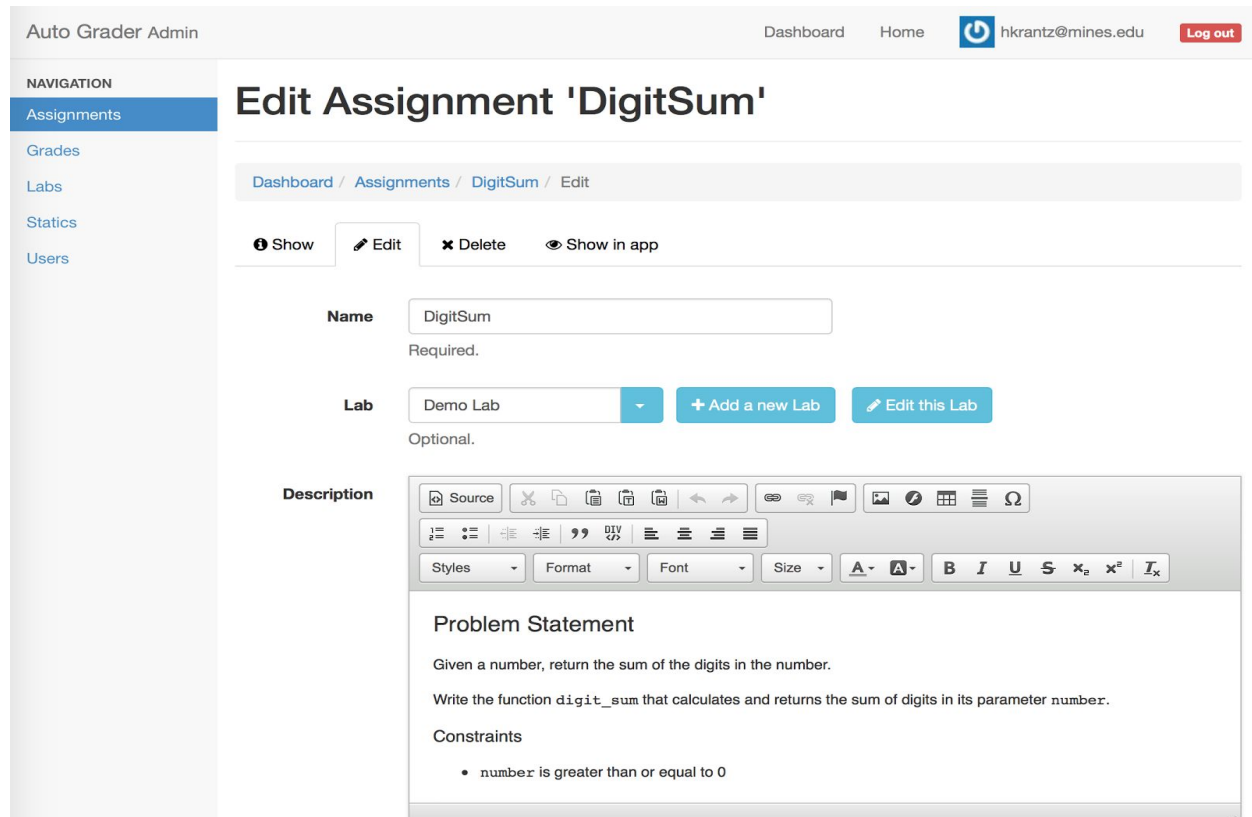


Figure 10

Decisions

Framework: Ruby on Rails

Our first major decision was the choice of what framework to build the website with. Our client gave us free reign over what to use and from the many different frameworks out there, our team chose to use Ruby on Rails. We did this because three of our five members already had experience with Ruby. In addition there exists a lot of open source code for Ruby and Rails that we were able to pull from to complete the website.

Usage of ACE Code editor:

We chose to include the ACE code editor in the website because we needed a code editor for students to use and ACE is an extensive, open source and highly customizable resource. It features syntax highlighting for C++ and customizable visual themes. Ace also proved to be easy to integrate into the website without additional changes.

Code Saving/Uploading:

Code saving, uploading and downloading are features we felt that the students would very much appreciate. Since all of our members had taken this class before, we know how useful it would have been to have all our code stored in one easy to access place and thus chose to include it.

Inclusion of Admin GUI:

An admin interface is something that would save so much time, it was impossible to overlook. This would allow CPW and TA's to have easy access to all the grades of their students and allow for easy creation of new assignments. This will be a massive improvement from the old system which required access to the server.

Login required immediately upon accessing to site:

A login system linked to the School of Mines account system was included out of necessity. Our website allows students to run arbitrary code on a computer inside the school's network and this is one of our security measures to prevent malicious attacks. This also creates unique accounts for the students in the class and enables grades to be easily recorded for the students.

Single-page responsive layout:

Having a responsive layout was a requirement from our client but this also forced us to maintain a simple design layout and improved the overall feel of the website. As a feature of using Bootstrap the website is mobile friendly and one could even write code on a smartphone, if you were feeling masochistic enough.

Visual Feedback of Test Results:

This was a remnant of CPW's old site that we expanded upon into the fully featured progress bar. This allows students to immediately see the results of their code and the input and output of each individual test. The progress bar is an important feature of the website and we decided to spend a lot of time with it in order to include all the features we felt were necessary such as test feedback, error handling, and full test reports.

Results

We were able to make a fully functioning website inline with the requirements given to us by CPW. Students can write and save code for each assignment, can test that code on the website with immediate feedback, and then submit that code for grading. CPW and TA's can easily access grades and have an interface for creating new assignments and modifying old ones. This website works across all modern browsers and also functions on smartphones and tablets.

However, there were some features that we wished to have but were unable due to time restraints. Some of these were: easter eggs/fanfare upon successful execution of all tests, integrating with the School of Mines grade storage service, and modifying some parameters for assignments has to be done through a file system rather than through the website. As far as improvements that could be added to this website: the above mentioned features could be implemented, expanding the website to cover several courses, and creating error highlighting and basic autocompletion inside the code editor. Throughout the project we learned several things: how to make a Ruby application, website development on PC can be difficult due to Ruby and Rails assuming a unix working environment, website styling using bootstrap, and creating unit tests in C++ is difficult to do.

Appendix

Deployment Guide

1. Clone project from git repository or source

(<https://github.com/eric-olson/AutoGrader.git>)

2. Install ruby and rails

- a. The guide we used is here: <https://gorails.com/setup/ubuntu/16.04>
- b. Skip the git and MySQL sections

3. Set up configuration files (examples are provided in same folder)

- a. server/config/database.yml: Input database user/pass
- b. server/config/secrets.yml: run `rake secret` and copy the result into the production field
- c. server/config/test_tool.yml: Update fields with appropriate paths
 - i. Use `which ruby` to find ruby path

4. Set up passenger with apache

- a. https://www.phusionpassenger.com/library/walkthroughs/deploy/ruby/ownserver/apache/oss/trusty/install_passenger.html
- b. The guide above handles most of the installation, although you will need to use the appropriate Ubuntu version when installing from APT (16.04 is 'xenial' instead of 'trusty')
- c. There is an example grader.conf file in the root directory of the project repository. This should be copied to `/etc/apache2/sites-available/` and the server name should be updated along with the SSL certificate file locations

5. Set up mod_shib

- a. This is done in the file located at `/etc/apache2/conf-available/shib.conf`
- b. There is an example shib.conf file located in the root directory of the project
- c. Once the file is copied/created, run `sudo a2enconf shib`
- d. Ensure that shibboleth config is correct for mines IDP, especially the attribute-map.xml file in `/etc/shibboleth/`. The application needs to have access to the displayName, mail, and uid attributes.

6. Rails project setup

- a. Run `bundle install --deployment` in server directory
- b. Run `rake db:migrate RAILS_ENV="production"` in server directory

7. Test Tool Setup

- a. Copy 'server/config/test_tool_default.yml' to 'server/config/test_tool.yml' and update the configuration values for the environment
 - i. **home_path**: The top level directory of AutoGrader (The directory where server, testing_tool, problems, ... reside)
 - ii. **testing_tool_script**: Either 'testing_tool/test_tool.rb' or 'testing_tool/test_tool_mock.rb' for normal operation or debugging, respectively.
 - iii. **ruby_executable**: This is the executable that the server uses to run the testing tool. It must be the one with sudo access or else the testing tool will not be able to enter the secure environment. Usually this will be the executable in '/usr/bin/' (more about that in the authentication instructions)
 - iv. **assignments_relative_path**: The path relative to home_path where all of the assignments are stored. In our repo the directory is named 'problems'
 - v. **users_relative_path**: The path relative to home_path where all of the user code will be stored. In our repo the directory is named 'users'
 - vi. **common_relative_path**: The path relative to home_path where the test environment files are. In our repo the directory is named 'common'
- b. Installing Stanford CPP Library:
 - i. Download the library
 - ii. Copy the contents of the 'include' directory in the library folder to '/usr/local/include/StanfordCPPLib'
 - iii. Copy 'lib/StanfordCPPLib.a' in the library folder to '/usr/local/lib/StanfordCPPLib', making sure that the file is compiled for your version of linux/ubuntu.
- c. Installing Google Test
 - i. Run 'sudo apt-get install libgtest-dev cmake'
 - ii. Build gtest from source:
 1. cd /usr/src/gtest
 2. sudo cmake .
 3. sudo make
 4. mkdir /usr/local/lib/gtest
 5. sudo cp *.a /usr/local/lib/gtest
- d. Authenticating the ruby executable:

- i. You must provide the ruby executable specified in 'server/config/test_tool.yml' with superuser access. The ruby executable must be owned by root or else it will not have superuser access.
- ii. Do this with: `sudo setcap cap_sys_chroot+ep <ruby_executable>`
 1. For example: `sudo setcap cap_sys_chroot+ep /usr/bin/ruby2.3`

8. Add admin user

- a. ``rails c production`` from server directory
- b. ``User.all``, find ID
- c. ``User.find(1).update(role:"admin")`` where 1 is the User ID found earlier

Known Issues

- When connecting to the website via the secure.mines.edu browser portal a warning is shown stating "The certificate was not issued by a trusted certificate authority". You can still login to the website but attempting to test code results in a server error.