Canvas LMS Testing Framework Report

Don’t Test Me

Kenneth Dela Cruz, Kyle Glick, Sam Word

Table of Contents

Chapter 1

I. Canvas LMS

Language(s):

Ruby, Javascript (Coffeescript), HTML+CSS

Framework(s) and Tools:

Ruby on Rails, Node.js, PostgreSQL

About:

Canvas is a well-established (circa 2010, used in many colleges, universities, and K-12 schools), open-source LMS by Instructure Inc. It is released under the AGPLv3 license for use by anyone interested in learning more about or using learning management systems.

Why we picked it:

- ➢ It has extensive documentation on Github
- ➢ The Github repository is being actively maintained and still has active development
- ➢ Because Canvas uses Ruby on Rails, it allows us to write tests and script in Ruby
- ➢ Canvas provides a script to automatically set up the application from scratch (though, it did not work in our experience on Ubuntu 16.04)
- ➢ It is a massive web application with a clear MVC architecture and extensive test suite written in Ruby.
- ➢ It cleverly combines Node.js and Ruby on Rails, meaning each team member has something to gain from the project.

II. Installation Report

Problems and difficulties were encountered when attempting to build Canvas on an Ubuntu image. On their GitHub page, Canvas provided a Quick-Start Guide which included a set of instructions for using their quick-start script and another set without using the script. Our first hurdle came from attempting to use their automated script. The script failed at some point during its execution, and being unaware of the details of the script, we chose to set up Canvas manually. Most of the issues we encountered while manually installing Canvas were a result of our inexperience with installing Ruby on Rails projects and their dependencies. For instance, two of us installed an incorrect version of Ruby, causing problems later on in the process, and we all encountered issues with permissions in PostgreSQL. We often resolved these issues by searching online for a way to undo our mistakes and trying again.

Unfortunately not all of the obstacles we encountered were as trivial as installing the wrong Ruby version. In fact, the most difficult issue we encountered surfaced while running a built in Rake task to install all of the required assets. Canvas uses a custom fork of jQuery 1.7.2 with a very, very small change. When Canvas’s asset management tool, Yarn, tried to retrieve the code from that jQuery repository, it complained about an Invalid Tar Header coming from the source of the repository. We were originally unable to get this version of jQuery to run, so we tested various jQuery releases, eventually finding that version 2.2.4 would compile. However, when running the application, nothing would load on the page. It turned out that jQuery wasn’t being defined using version 2.2.4. After struggling with this issue for awhile, we decided to write an Issue on Canvas’s Github page. While writing the issue, we reset the jQuery version used by Yarn to get a screenshot of the error to find that the error no longer appeared. Surprised, once we finished compiling the assets again and ran the application, everything functioned as intended. We’re still not entirely sure what fixed this issue, but we believe that adding several asset folders and files, a step only mentioned in the Production Start Guide, may have resolved our issue.

On our team Github wiki, we provided step-by-step instructions for settings up Canvas. Many of the steps were taken from Canvas’s Quick Start Guide, though a few additions were made that were required to get the application running. Some of these were manual edits that we believe are Ubuntu-only issues, such as providing access to the PostgreSQL server, while others were steps in the Canvas Production Start Guide that were not in the Quick Start Guide.

III. Custom Installation Guide

This is a guide on setting up Canvas LMS by Instructure Inc. in an Ubuntu environment based on our experiences. The official development and production start guides can be found on the Canvas GitHub:

- Official Quick Start Guide
- Official Production Start Guide

Set Up Git

```
sudo apt-get install git
git clone https://github.com/instructure/canvas-lms.git canvas
cd canvas
```

Install Curl

```
sudo apt install curl
```

Install Node.js and PostgreSQL

```
curl -sL https://deb.nodesource.com/setup_6.x | sudo bash -
sudo apt-get install nodejs postgresql
```

Install Ruby and Ruby Dependencies

```
sudo apt-get install software-properties-common
sudo add-apt-repository ppa:brightbox/ruby-ng
sudo apt-get update
sudo apt-get install ruby2.4 ruby2.4-dev zlib1g-dev libxml2-dev
libsqlite3-dev libpq-dev libxmlsec1-dev curl make g++
```

Create PostgreSQL Superuser

```
sudo -u postgres createuser $USER
sudo -u postgres psql -c "ALTER USER $USER WITH SUPERUSER" postgres
```

Configure Gem Installation Directory

Note: You will have to export the GEM_HOME directory for each new bash session. If you don't want to do this, the Quick Start guide explains how to permanently link to the gem installation directory.

```
mkdir ~/.gems
export GEM_HOME=~/.gems
```

Install Bundler

```
gem install bundler -v 1.15.3
```

Install Yarn

```
curl -sS https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee
/etc/apt/sources.list.d/yarn.list
sudo apt-get update && sudo apt-get install yarn
```

Install Python (for Yarn)

```
sudo apt-get install python
```

Install Gems

```
$GEM_HOME/bin/bundle install
```

Install CoffeeScript

```
sudo npm install -g coffee-script@1.6.2
```

Set Up Config YML Files

```
for config in amazon_s3 delayed_jobs domain file_store outgoing_mail
security external_migration; do cp -v config/$config.yml.example
config/$config.yml; done
```

Set Up Database Config YML File

```
cp config/database.yml.example config/database.yml
```

Create Tables and Populate Data

Note: The email and password you enter here will be your login credentials later.

```
$GEM_HOME/bin/bundle exec rake db:initial_setup
```

Create Test Database Admin

```
psql -c 'CREATE USER canvas' -d postgres
psql -c 'ALTER USER canvas CREATEDB' -d postgres
```

Give PostgreSQL Permissions

```
sudo gedit /etc/postgresql/9.5/main/pg_hba.conf
```

Change `peer` to `trust` where it's used.

Create Test Database

```
createdb -U canvas canvas_test
psql -c 'GRANT ALL PRIVILEGES ON DATABASE canvas_test TO canvas' -d
canvas_test
psql -c 'GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO canvas' -d
canvas_test
psql -c 'GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO canvas'
-d canvas_test
psql -c "ALTER USER canvas WITH PASSWORD 'password'" -d canvas_test
```

Configure Test Database Credentials

```
gedit config/database.yml
```

Add `password: password` inside of the test environment YML `test:`

Set Up Test Database Tables

```
RAILS_ENV=test $GEM_HOME/bin/bundle exec rake db:test:reset
```

Create Asset Files and Directories

```
mkdir -p log tmp/pids public/assets app/stylesheets/brandable_css_brands
touch app/stylesheets/_brandable_variables_defaults_autogenerated.scss
touch Gemfile.lock
sudo chown -R canvasuser config/environment.rb log tmp public/assets
app/stylesheets/_brandable_variables_defaults_autogenerated.scss
app/stylesheets/brandable_css_brands Gemfile.lock config.ru
```

Compile Assets

```
yarn install
$GEM_HOME/bin/bundle exec rake canvas:compile_assets
$GEM_HOME/bin/bundle exec rake brand_configs:generate_and_upload_all
sudo chown -R canvasuser public/dist/brandable_css
```

Run Server

```
$GEM_HOME/bin/bundle exec rails server
```

Chapter 2

I. Test Report

The testing process
Our test suite is testing methods from various model files in the Canvas LMS system.

Testing schedule
08/31/2017 - Picking an HFOSS project
09/12/2017 - Deliverable 1: H/FOSS Project Evaluation
10/03/2017 - Deliverable 2: Initial Test Plan (originally 09/26/2017)
10/31/2017 - Deliverable 3: Design Test Framework (originally 10/24/2017)
11/14/2017 - Deliverable 4: Complete Test Framework
11/21/2017 - Deliverable 5: Fault Testing

Test recording procedures
Results of the tests will be recorded in an HTML document and opened in a Web Browser at the end of the suite. The individual test reports will include the test number, requirement being tested, method tested, test inputs, expected outcome, test outcome, and whether or not the test passed or failed.

Hardware and software requirements
At the moment, we are unsure of the specific hardware and software requirements. We should have a better idea of the hardware and software requirements once we start implementing these test cases. What we are certain however, that running this test suite will require the following software components:

- Ruby version 2.4.1
- Bash Terminal
- *Canvas LMS Code (TBD)*
- *Canvas LMS Dependencies (TBD)*

Constraints
Our team is not familiar with the Canvas LMS system. This makes it difficult for us to find and confirm test cases and requirements. Various requirements are not specifically documented for us to use. Additionally, these tests must be written outside of the Canvas application with as few dependencies as possible.

II. Test Cases

Test Case 1: Course#score_to_grade

Model: *Course*

Method: *#score_to_grade*

Inputs: *grade* - int / float

Requirements: Grade is correctly calculated when using the default Grading Standard

	Description	**Expected Result**
STEP 1:	Create a course with no grading standard	
STEP 2:	Retrieve default grading standard from *GradingStandard* model	
STEP 3:	Check values of default grading standard for accuracy	[["A", 0.94], ["A-", 0.90], ["B+", 0.87], ["B", 0.84], ["B-", 0.80], ["C+", 0.77], ["C", 0.74], ["C-", 0.70], ["D+", 0.67], ["D", 0.64], ["D-", 0.61], ["F", 0.0]]
STEP 4:	Compare scores from each grade range using *#score_to_grade(grade)*	Grade returned for each score matches default grading standard

Test Case 2: Course#score_to_grade

Model: *Course*

Method: *#score_to_grade*

Inputs: *grade* - int / float

Requirements: Grade is correctly calculated when using a custom Grading Standard

	Description	**Expected Result**
STEP 1:	Create a course with a custom grading standard	
STEP 2:	Compare scores from each grade range using *#score_to_grade(grade)*	Grade returned for each score matches custom grading standard

Test Case 3: LatePolicy#missing_points_deducted

Model: *LatePolicy*

Method: *#missing_points_deducted*

Inputs: *points_possible* - int
grading_type - string

Requirements: Calculates missing points to deduct from late submission using a “pass-fail” grading type

	Description	**Expected Result**
STEP 1:	Set *missing_submission_deduction* to a valid value	
STEP 2:	Call #missing_points_deducted with *points_possible = 100*, *grading_type = “pass_fail”*	Should return *points_possible* as a float

Test Case 4: LatePolicy#points_for_missing

Model: *LatePolicy*

Method: *#points_for_missing*

Inputs: *points_possible* - string
grading_type - string

Requirements: Deducts missing points

	Description	**Expected Result**
STEP 1:	Set *missing_submission_deduction* to a valid value	
STEP 2:	Call *#points_for_missing* with *points_possible = 100*, *grading_type = “string”*	Should return *points_possible* as a float with missing points deducted from it

Test Case 5: Setting#set

Model: *Setting*

Method: *#set*

Inputs: *name* - string
value - string

Requirements: Sets the value of a setting

	Description	**Expected Result**
STEP 1:	Call *#set* with *name = “setting1”*, *value = “true”*	
STEP 2:	Call *#get* with *name = “setting1”*, *value = “true”*	Should return true

III. Experiences with Test Report

We found this deliverable to be fairly straightforward since we began looking for methods to test several weeks ago. One of the biggest challenges of creating this Test Report was determining the requirements to test for each method. Since we are not a part of Canvas’s development team, and most of their methods have no documentation, we had to make up requirements to test. Ultimately, we created our requirements to test based off of the model specs in Canvas to test these methods.

Something interesting we noticed, was that unlike Java, Ruby does not specify the types of inputs that each model method can take. For example, the missing_points_deducted method in late_policy.rb has two parameters: points_possible and grading_type. Inside this method, the method #to_f is called on points_possible. Without looking at the LatePolicy model spec, we would expect points_possible to be a string that needs to be parsed to a float. However, the specs to test missing_points_deducted calls the method with points_possible as an int. This made us realize that due to the nature of languages like Ruby and Python, it’s possible to call a method with unexpected data types, and possibly get unexpected results. This is something we likely wouldn’t have to test if Canvas was coded in Java.

Glossary of Terms

JavaScript:
A high-level object-oriented programming language developed by Brendan Eich and released in 1995. It is used primarily in World Wide Web content to make interactive pages and online applications.

jQuery:
A JavaScript library built originally by John Resig and released in 2006. It’s primary purpose is to make manipulating HTML documents easier and provide improved interactivity with document elements.

PostgreSQL:
A popular Database Management System (DBMS) released in 1996.

Rake (Ruby Make):
A build utility for Ruby, similar to the “make” command.

Rake Task:
A Ruby script built to run using Rake syntax.

Ruby:
A dynamic, general-purpose programming language developed in the mid-1990s by Yukihiro Matsumoto. It was designed with ease-of-use and flexibility in mind. It can be used for functional, object-oriented, and imperative programming.

Ruby on Rails (Rails):

A server-side MVC web application framework in Ruby developed by David Heinemeier Hansson in 2005. It uses HTML, CSS, and JavaScript for user interfacing and JSON and XML for data transfer. It also provides a built-in test suite and a variety of external libraries for developers.

Gem:

An external Ruby on Rails package source. They’re compiled and installed by defining the gem source in the Gemfile and running bundle install.

Spec (Specification):

The name given to Ruby on Rails automated tests using the built-in Gem, RSpec.

Yarn:

A Dependency Management System (DMS) used to compile and control packages and assets.