Sandbox code overview

1.7.2011. Helsingin yliopisto. Tietojen käsittelytieteen laitos.

Patrik Marjanen, Jarno Mynttinen, Martti Rannanjärvi, Katri Rantanen

This program is a program that lets course instructors create a skeleton program with tests that compile. It is then possible for a student to use the Palikka netbeans plugin, or any other editor/ ide, to finish the skeleton program, so that the tests pass. When the student has a working program, which passes the tests made by the instructor, he can submit his exercise (either by using the palikka plugin or by submitting via the webinterface) and receive points for them.

The usual flow of the program

- An instructor logs in to the program, and creates a course by inserting a name in the new course textfield.
- He then clones the git address supplied to him by the program.
- He creates a skeleten program, with tests, that compiles.
- He then pushes his “changes” into the git repository. A folder in the git repository, equals an exercise in the Course.
- He goes to the webinterface and refreshes the course.
- He sees the exercises which he has added to the course.
- The instructor is now happy :).
- A student opens up his netbeans ide and polls for all the active courses.
- He then finishes the exercises which are due in 5 minutes.
- He submits his exercises, and the program says that his submission is correct.
- The instructor decides to (finally, after a lot of students have harrassed him about the points not showing in gdocs) upload the accumulated points into gdocs.
- He goes to the courses page and sees a whopping 532 points not uploaded. He presses the upload to gdocs button and waits for a while (praying that gdocs works). And notices that on the new pageload, there are 0 points not uploaded to gdocs.

Tools

Our program is written in Ruby on Rails version 3.0.7. (http://rubyonrails.org/).
Testing is done with Rspec (http://relishapp.com/rspec).

Git repository

We use git (http://git-scm.com) as a means for an instructor to be able to easily collaborate with other instructors in creating exercises for a course.

NetBeans plugin

Our program is used by the netbeans plugin created by the palikka team. They use a set of json outputs (courses, exercises), with which they get the information about courses and exercises and where they can upload/download exercises/courses.

All student answers

Server stores all student answers. All student submissions are stored in the ExerciseReturns table as a blob.

Testrunner

Test runner will take a project to compile, then it will run the tests.

Testrunner has a default timeout (1 min).

Tests have a Exercise annotation. The Exercise annotation is an annotation that specifies that this is a test that must pass in order to be able to check that assignment/test as passed. The same annotation can be before multiple testmethods, ie. an exercise can be dependent of many tests.

Students' points database
After running tests (testrunner) will store results of tests in the Points table with a true / false depending whether the test has passed or not.

Temporary points database
When an Exercise is returned by the student, all the tests in that exercise are run. For an exercise (ie. exercise "2.5") a row in the Points table is created. If all tests for an exercise have passed, the Point tables tests_passed will be true, otherwise false. All points (where tests_passed is true) that have not been uploaded to Google Docs also has an entry in a PointsUploadQueue. When an instructor decides to upload the points to Google Docs, the point reference in PointsUploadQueue will be destroyed (if successfull upload, otherwise they will be kept in the table).

Google docs
Server will store gained points to Google docs. In order to access Google docs, we make use of library (google-spreadsheet-ruby) which supply us with easy access to editing existing document. In order to access documents in Google docs, we make use of a Google account (username: pajaohtu, password: qwerty1234567). While creating new course on web site, the system will also create new spreadsheet for the course, using "Model" spreadsheet as a model. Instructors will have to manually create new sheets for the document.

Controllers

- application_controller.rb
    - Main application settings, has a before_filter for authentication checking and a before_filter for changing the default_url_options.
- exercise_returns_controller.rb
    - Controller that manages the exercises that the student returns.
- sessions_controller.rb
    - Manages the login of the program.
- test_suite_runs_controller.rb
    - When an exercise is submitted a TestSuiteRun is created, each TestSuiteRun has a number of TestCaseRuns, which are individual tests. This controller also supplies the Palikka team with a json output of the created TestCaseRuns.
- courses_controller.rb
    - Controls the action of a course (creation, destruction, creating a new one etc.). It has a non-standard method called refresh. This method is called when refreshing the contents of a course (updating exercises of a course) after doing a git push.
- exercises_controller.rb
    - This controller controls the exercises which have been pushed into the courses git repository. This controller also supplies the palikka team with a json output of exercises for a specific course. This controllera also gives out the zip files of each exercise.
- points_controller.rb
    - Lists all points in all courses for all students. Also has a the method upload_to_gdcos which is called when an instructor pushes the upload to gdocs button in the points view.
- test_case_runs_controller.rb
    - Manages TestCaseRuns.
- users_controller.rb
    - Manages Users.

Helpers

- sessions_helper.rb
    - Has helpers that help the sessions_controller in the signing in/out of the user.
- points_helper.rb
    - Has two view helpers (course_name and queue_status).

Models

- course.rb
  - The Course model manages all the database functionality of the Course table. It includes “Rails.application.routes.url_helpers” and “GitBackend”. GitBackend is a module in the lib directory that gives the Course model the functionality to create, destroy and refresh the git repository.
- exercise.rb
  - This model represents one Exercise in a Course. Exercises are not added via a web-interface, but are added to the program as folders in the courses git repository (where the folder name is going to be the exercise name). Exercises are updated when calling the refresh hook (either in the url or a refresh link in the courses page).
- exercise_point.rb
  - This model represents the exercise “numbers” that are possible for a student to get, when he/she submits an exercise. This model belongs to Exercise and has_many Points.
- exercise_return.rb
  - An exercise_return belongs to a exercise. An exercise_return is a returned exercise. The table has an exercise field which contains the students returned answer as a binary file.
- test_suite_run.rb
  - This model represents a suite of TestCaseRuns. It belongs to ExerciseReturn. There is a method run_tests which call the TestRunner to run the tests in the suite, and then after that check all the points from the created TestCaseRuns.
- test_case_run.rb
  - This model represents a single test method in a java testfile. It has information about if a certain test succeeds or fails, and what the error message for a failed test is.
- point.rb
  - If TestCaseRuns for a certain exercise_point (number) have all passed a success row is added to Points, if the TestCaseRuns have not all succeeded a fail row is added to the Points table. If a Points tests_pass field is true, an entry to PointsUploadQueue is also added.
- points_upload_queue.rb
  - Stores a reference for all successfull entries in the Points table, that have not been uploaded to google docs. When an instructor decides to upload the points to google docs, the reference to the row in the Points table is removed (on successfull upload). If the upload failes, the reference is kept, so that the upload can be retried an other time.
- user.rb
  - This model represents a single user of the program. This model has no user interface, so if you would like to add a user, you have to do it manually into the

db. This model also has the logic for encrypting the user password.

Lib

- gdocs.rb
  - Includes all methods and functionality used when interacting with google docs. It is possible to disable the gdocs functionality by setting the disabled variable to true in the initialize block.
- git_backend.rb
  - Has all the functionality of managing Git repositories.
- system_commands.rb
  - Helper method that raises an exception when the system! method fails.
- test_runner.rb
  - Has all the functionality required to run submitted exercises.

Backends

- git
- sql
- google docs

Views

- json
- zip
- html

References

http://ruby.railstutorial.org/ruby-on-rails-tutorial-book
http://net.tutsplus.com/tutorials/ruby/how-to-build-an-unobtrusive-login-system-in-rails/
http://api.rubyonrails.org/
http://today.java.net/pub/a/today/2008/04/10/source-code-analysis-using-java-6-compiler-apis.html
http://www.tutorialspoint.com/ruby-on-rails/rails-file-uploading.htm
http://guides.rubyonrails.org/form_helpers.html