Education Tool 6000

Paul O’Beirne

X10203800

Project completed in partial fulfillment of the Cloud Application Development module at the National College of Ireland.

# Introduction

The purpose of this project is to design and develop an Intelligent Tutoring Tool that could help a first year student build competence in a first year module. The source material chosen by me will reflect material covered over the first year by students taking an introduction to programing and Object Oriented Programming module. The project has development requirements such as it is required to be developed using Rails 4.1.5 be deployed on a cloud service provider. The application must also be able perform an evaluation of the users progress and be able to automatically generate of suitable questions to help scaffold the learner process of the identified domain.

# Pedagogic Strategy

The pedagogic strategy that I will be employing will revolve mainly around providing a framework in which the users can test their knowledge on the domain subject while being assessed continually. The users will be provided with an opportunity to assess their current level at any time through taking an assessment quiz or alternatively can be assessed through ordinary quiz’s based on results.

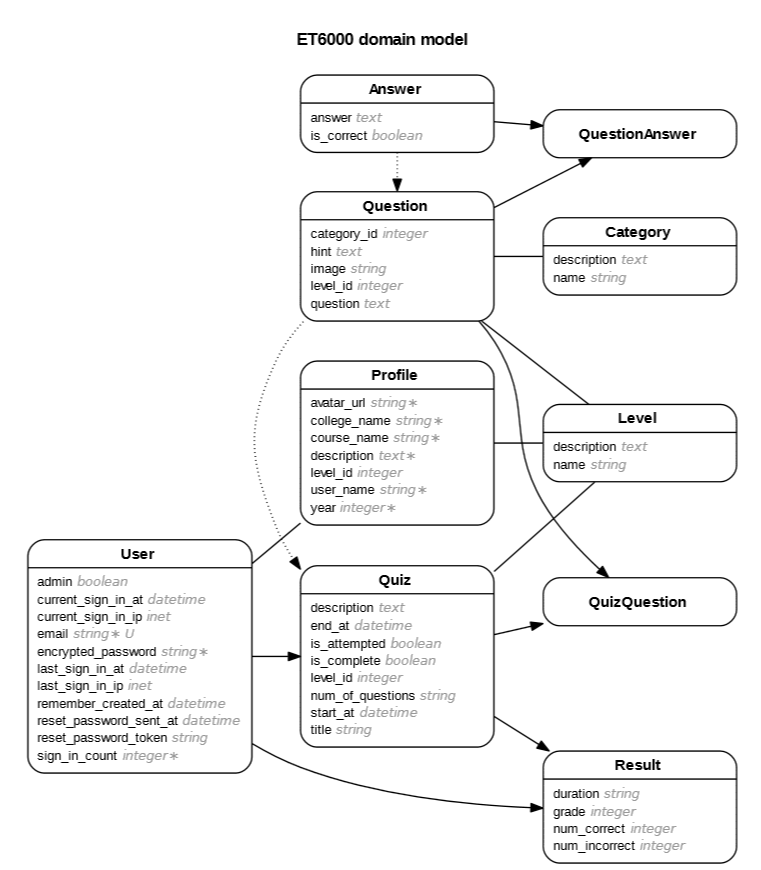
Upon completion of a quiz the user’s results are graded and an evaluation function will determine the user’s level of ability. The next time the user chooses a quiz to take a new set of questions will be generated according to the users level settings.

The user will be provided with a results overview to show progress in and additional statistical information regarding the quizzes taken.

# Development Architecture

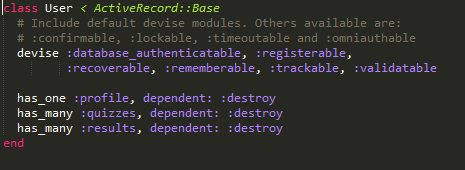
**Domain model**

This is a diagram based on my application's Active Record models. The diagram gives an overview of how your models are related.



**Models**

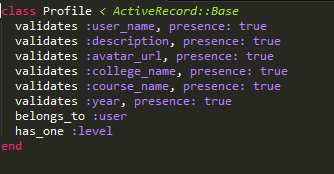
User



Associations:

* Each user has one profile and it will be destroyed upon the user being destroyed (one to one relationship)
* Each user has many quizzes and they will be destroyed upon the user being destroyed (one to many relationship).
* Each user has many results and they will be destroyed upon the user being destroyed (one to many relationship).

Profile

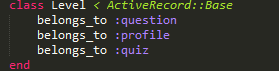


Validations: Required on all fields stated doesn’t allow null values to be passed.

Associations:

* Each profile belongs to a user sets up a one-to-one connection with another model
* Each profile has one level (one to one relationship)

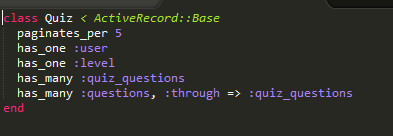
Level



Associations:

* Each sets up a one-to-one connection with another model

Quiz

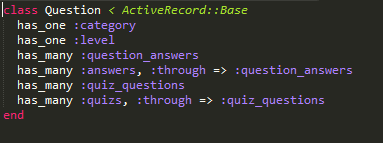


Paginate parameter limits 5 records per page.

Associations:

* Each Quiz has one user (one to one relationship)
* Each Quiz has one level (one to one relationship)
* Each Quiz has many questions, set up a many-to-many connection with another model (quiz questions).

Question



Associations:

* Each Question has one category (one to one relationship)
* Each Question has one level (one to one relationship)
* Each Question has many answers, set up a many-to-many connection with another model (question answers).
* Each Question has many quizzes, set up a many-to-many connection with another model (quiz questions).

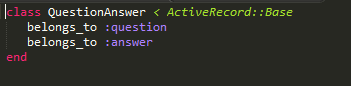
Category



Associations:

* Sets up a one-to-one connection with another model

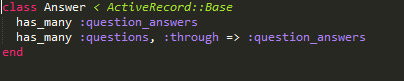
Question Answer



Associations:

* Each sets up a one-to-one connection with another model

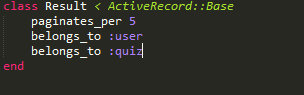
Answer



Associations:

* Each Answer has many questions, set up a many-to-many connection with another model (question answers).

Result



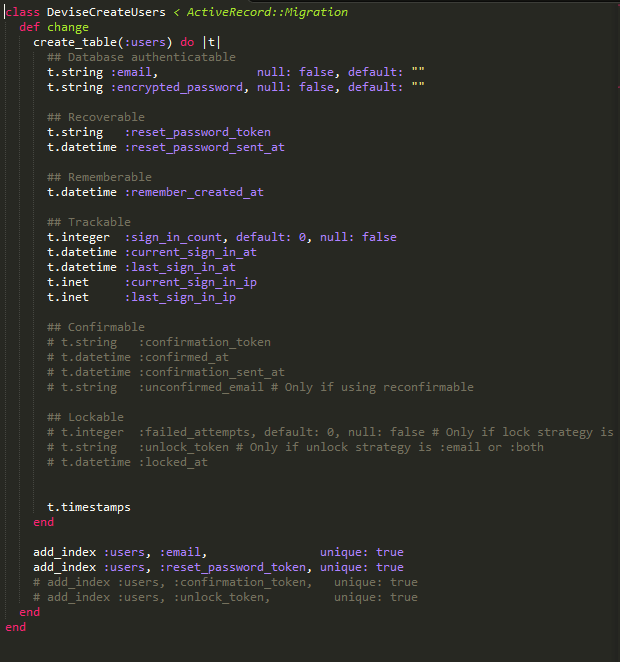
Paginate parameter limits 5 records per page.

Associations:

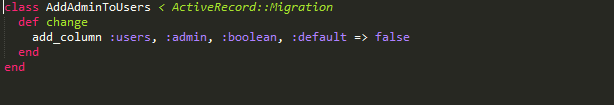
* Each sets up a one-to-one connection with another model

## Migrations

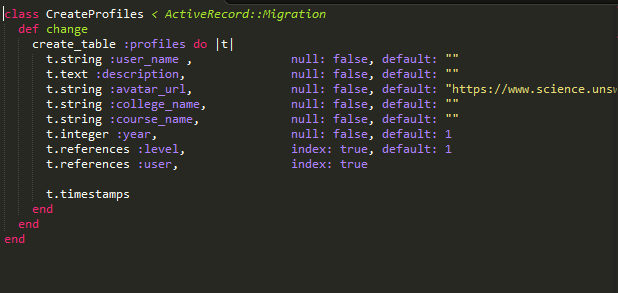
Devise Create Users



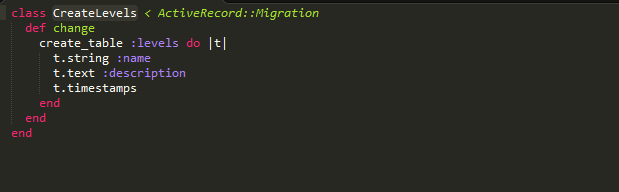
Add Admin To User



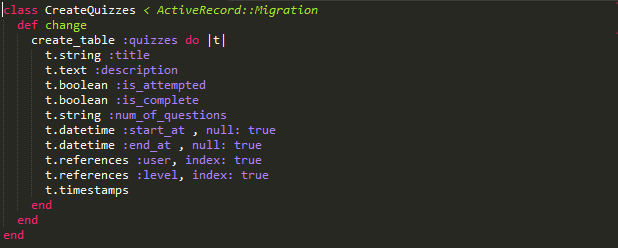
Create Profiles (Modified null and default values)



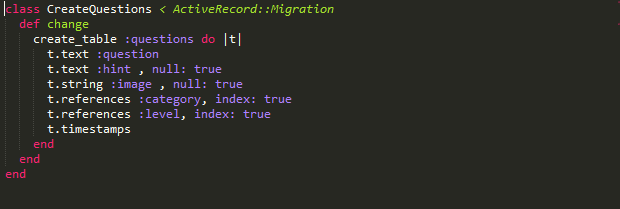
Create Levels



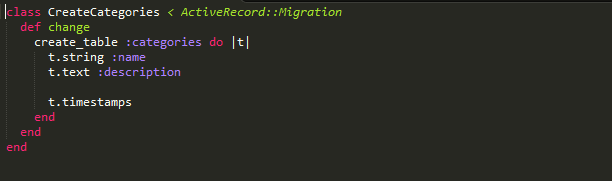
Create Quizzes



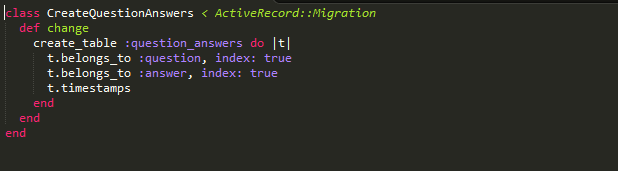
Create Questions



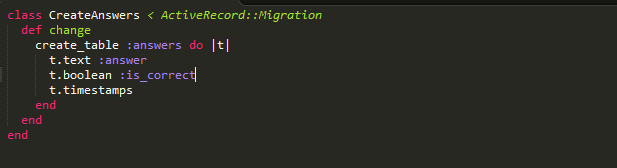
Create Categories



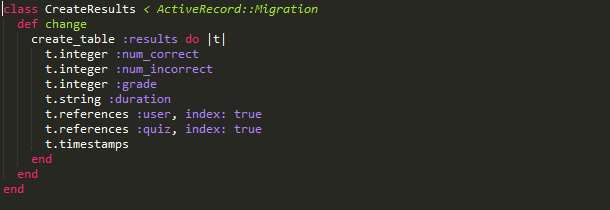
Create Question Answers



Create Answers



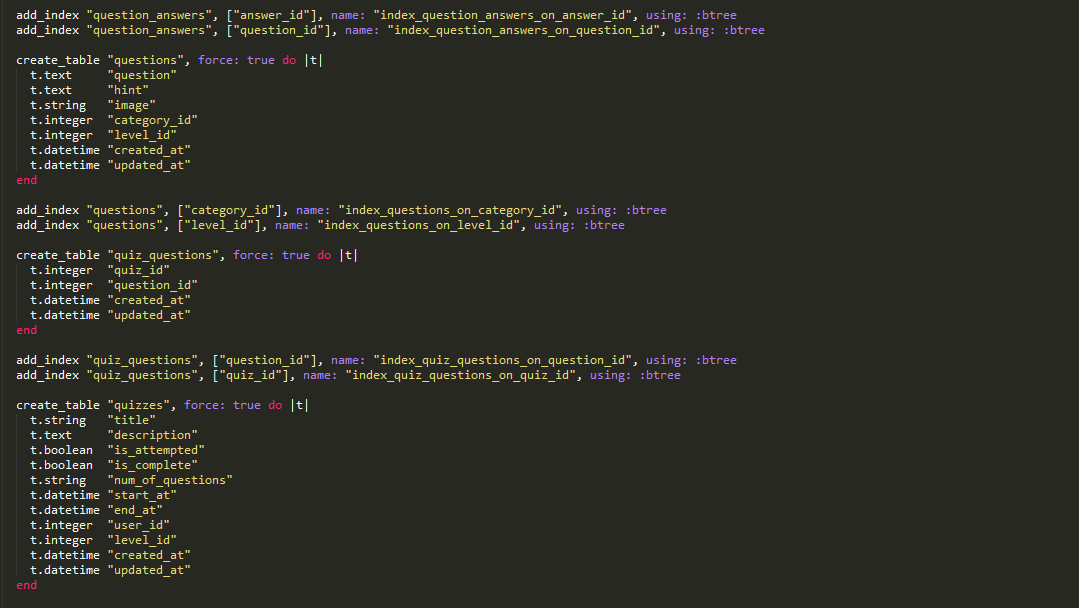
Create Results

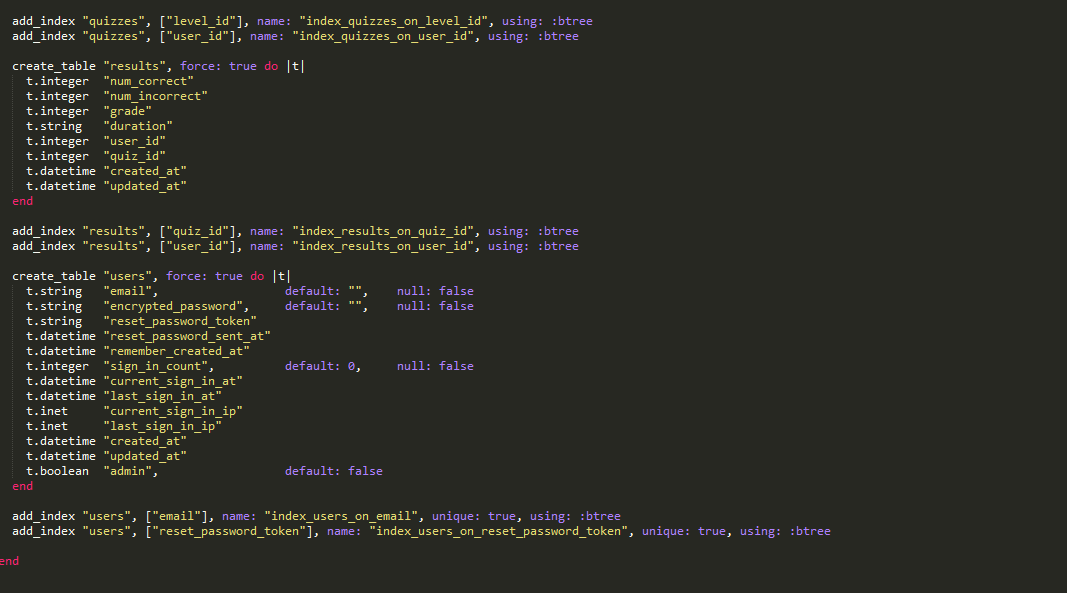


**Schema**

Generated when rake dB: migrate is run



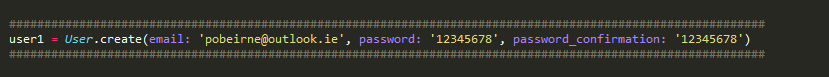




## Seeding

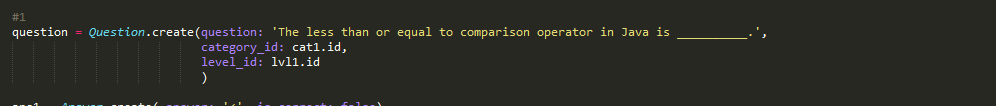
I have configure the seeding file to create one user account and setup the level and category tables. All the questions, answers and the relationships between each used in this application are also created from the seed file.

User a new user account.

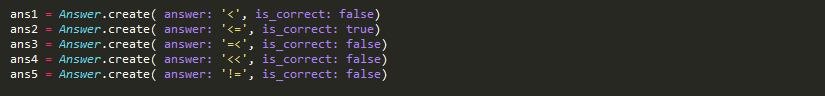


## Level each level kind is created.

## Category each category is created this is a sample one.

Question a new question is created and assigned a category and level.

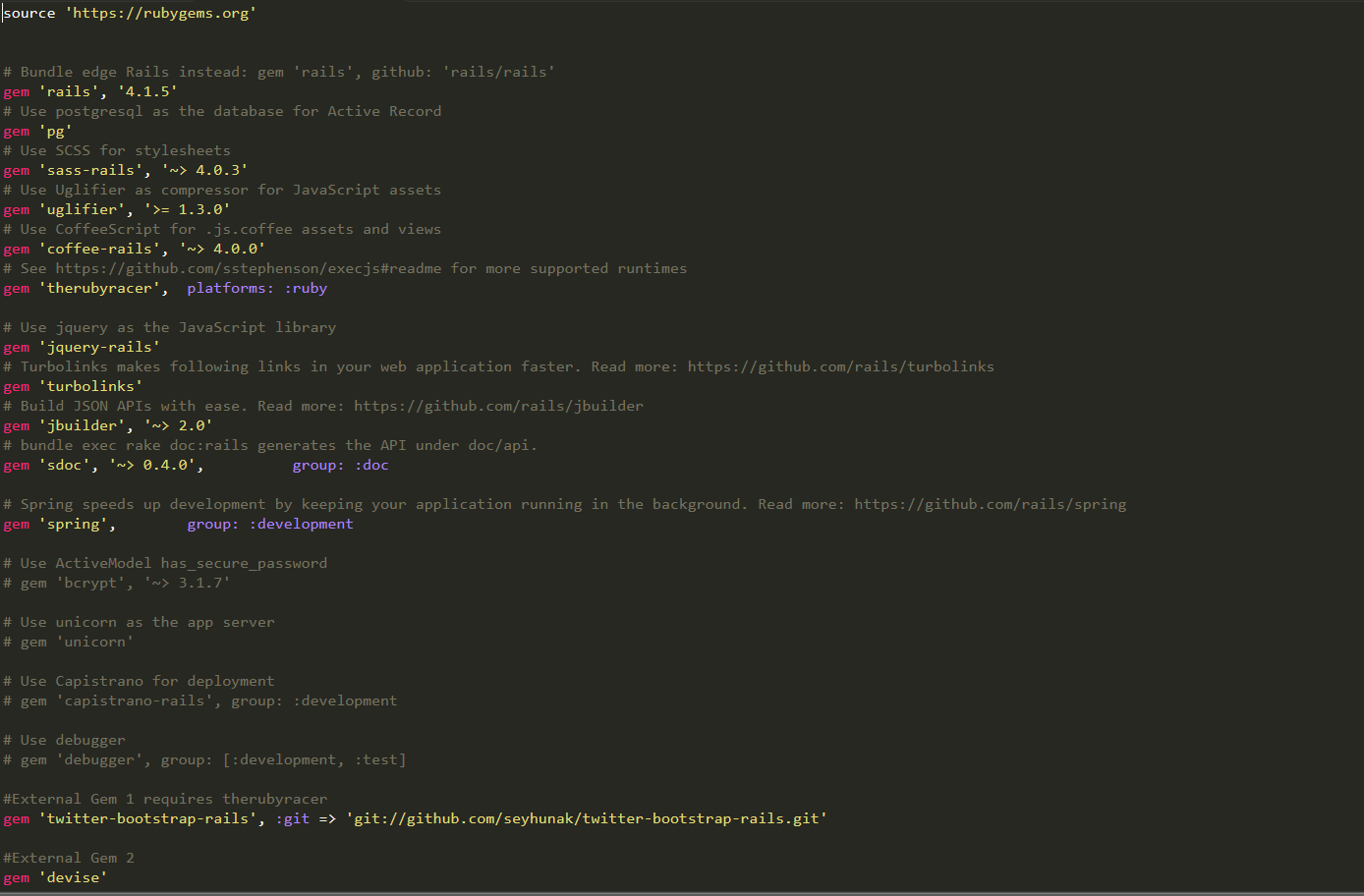
Answer multiples answers are created.



Question Answers many to many relationships created between question and answers tables.



## Gem File





## Custom Gems

**Gem 1:** gradeeval

**Source:** <https://github.com/pobeirne/gradereval>

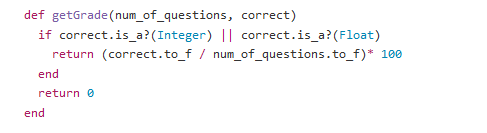
**Overview**

This gem provides the application with some basic functionality that can be used to grade the user’s answers and evaluation of the user’s level of performance.

**Grade method**

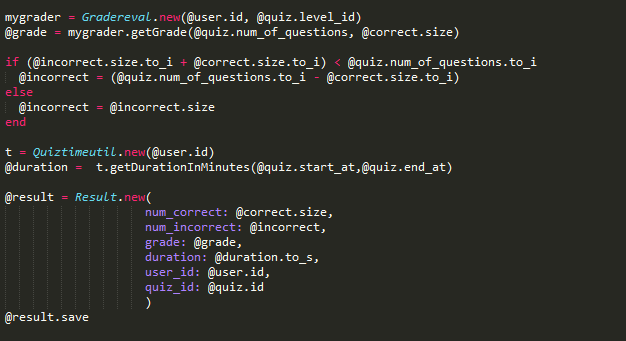
The grading method simply take the users correct answers and the amount of questions in the exam and returns a percentage to the application. The result is saved into a quiz results record.

**Grade method Code**



**Example usage**

**Quiz Controller: action: finish**



**Evaluation method**

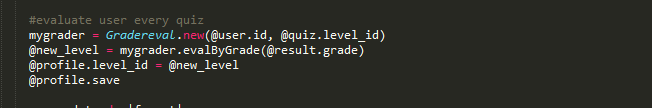
The evaluation method takes the users grade and determines if the user’s level should raise, fall or remain the same. If the users level changes it will be saved into the user’s profile. Upon each quiz being created it will look at the user’s level and select questions according to that level.

**Evaluation method code**

|  |  |
| --- | --- |
|  |  |

**Example Usage**

**Quiz Controller: action: finish**



**Gem 2:** quiztimeutil

**Source:** <https://github.com/pobeirne/quiztimeutil>

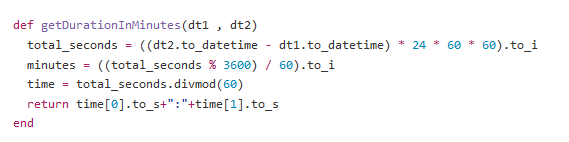
**Overview**

This gem provides the application with some basic functionality that can be used to determine the users duration of time for each quiz taken and the users average duration per quiz.

**Duration method**

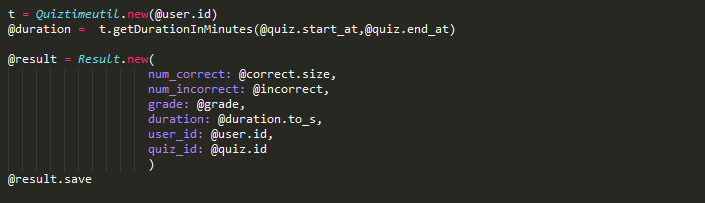
The Duration method takes in two date-times as parameters dt1 would be the date-time the quiz was started by the user and dt2 would be the end date of the quiz. The method would calculate the duration in minutes and the returned value would be saved to a particular quiz’s results record.

**Duration method code**

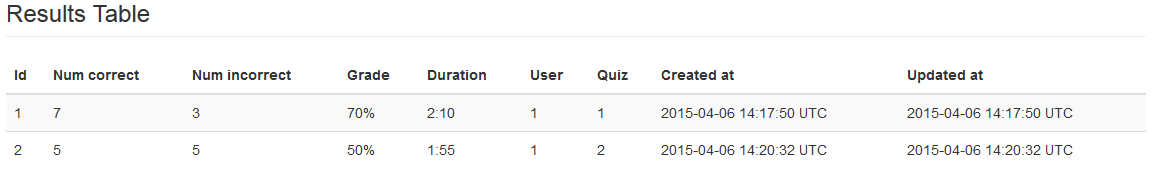


**Example usage**

**Quiz Controller: action: finish**

****

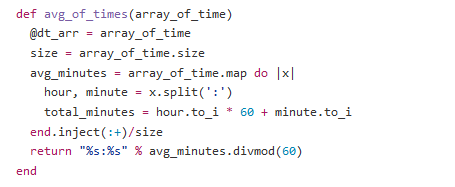
**Result view**



**Average duration method**

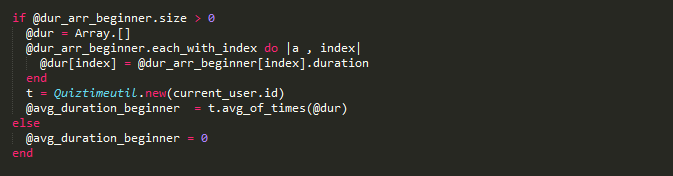
The average Duration method takes an array of the users result records durations and returns the average in minutes.

**Average duration method code**

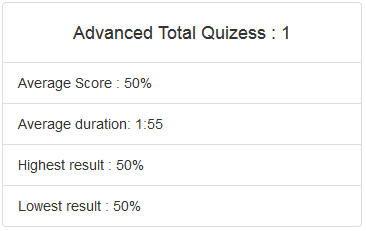


**Example usage**

**Overview Controller: action: result**

****

**Result view**



## External Gems

**Gem :** Devise

**Source:** <https://github.com/plataformatec/devise>

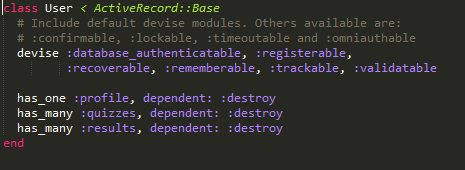
**Overview**

Devise provides an advanced authentication engine for models, controllers and views. Devise is a authentication solution that can be used to determines if users are who they say they are, commonly through a username and password combination. Authorization determines which actions an authenticated user can perform.

**Integration**

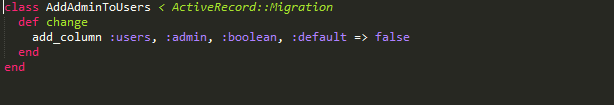
After the initial installation and configuration was complete, I generated the devise views and models. The model was altered to include associations with other tables see below:

User Model



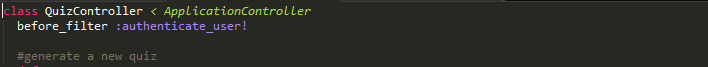
And a migration was created to add an extra field for administrative use see below:

Add Admin To Users



Devise comes with controller and view helpers that can be used to verify a user is authentic, logged in and has access to controllers or views. I added a helper before\_filter :authenticate\_user! To each of the controllers that required a user to be valid see below:

before\_filter :authenticate\_user!

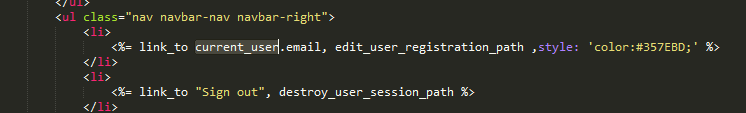


And in the main applications layout I used user\_signed\_in? as a condition to determine which navigation links the user can view. I also referenced the current\_user helper to display the users email in the navigation bar when logged in.

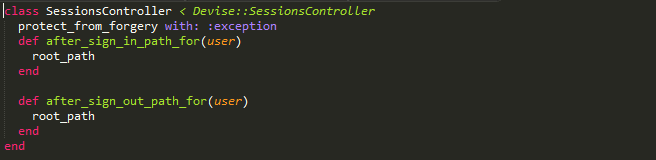
user\_signed\_in?



current\_user

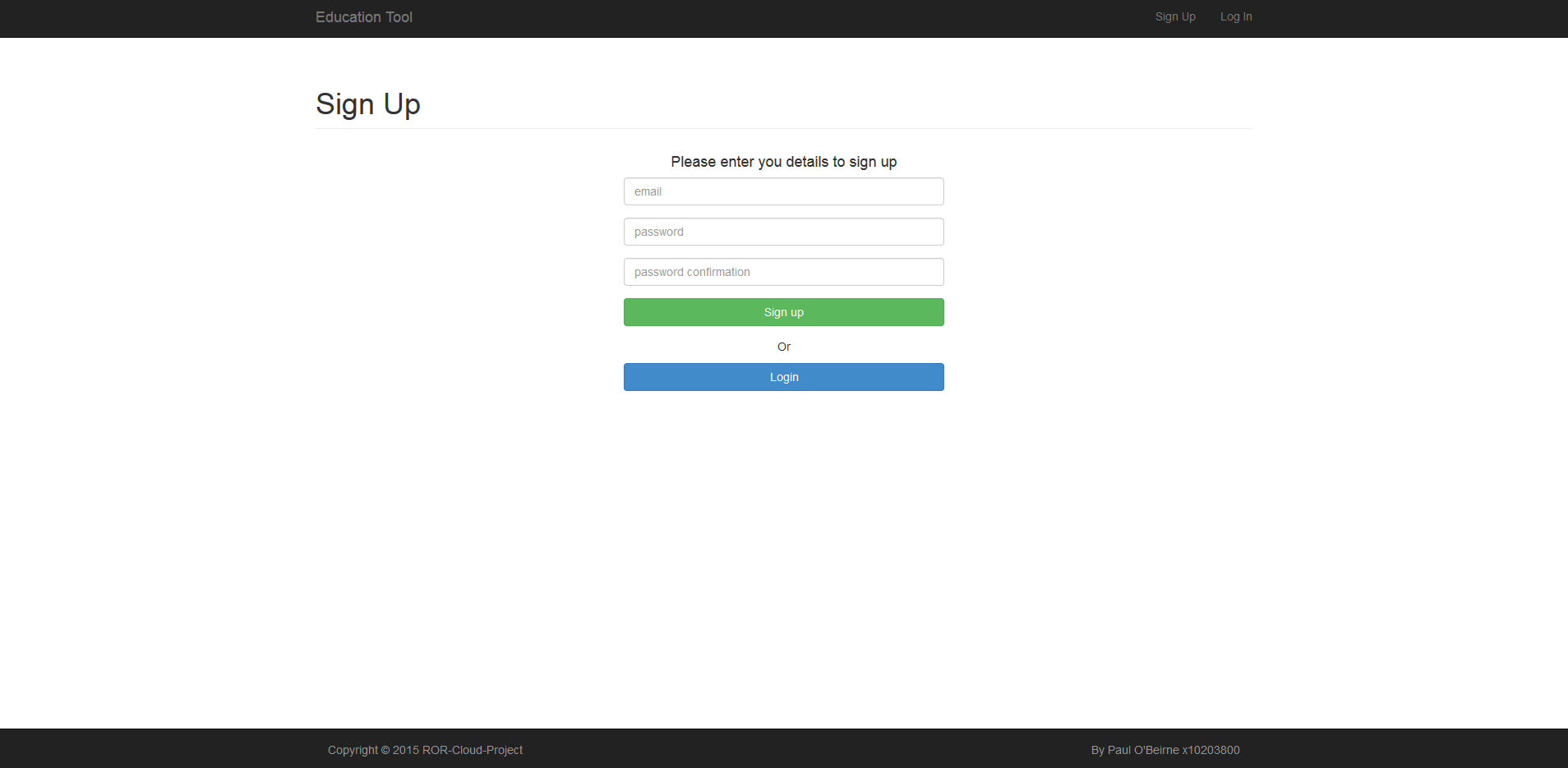


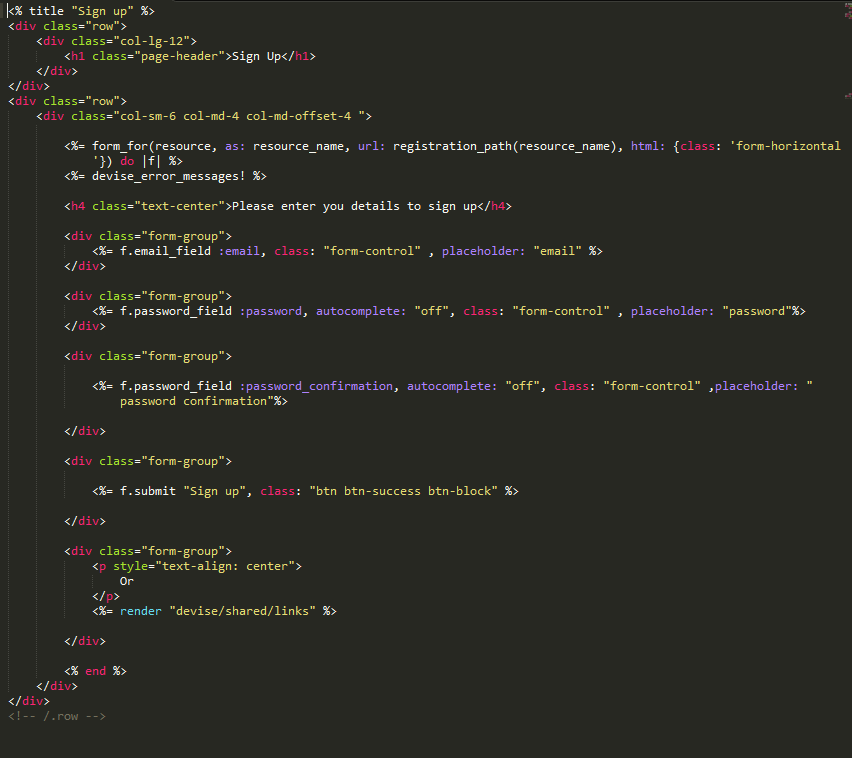
To handle what actions the application should take upon logging in or out I created a Session controller to override some of the default Devise Session controller methods and added in some custom methods to redirect to different paths of my choosing.



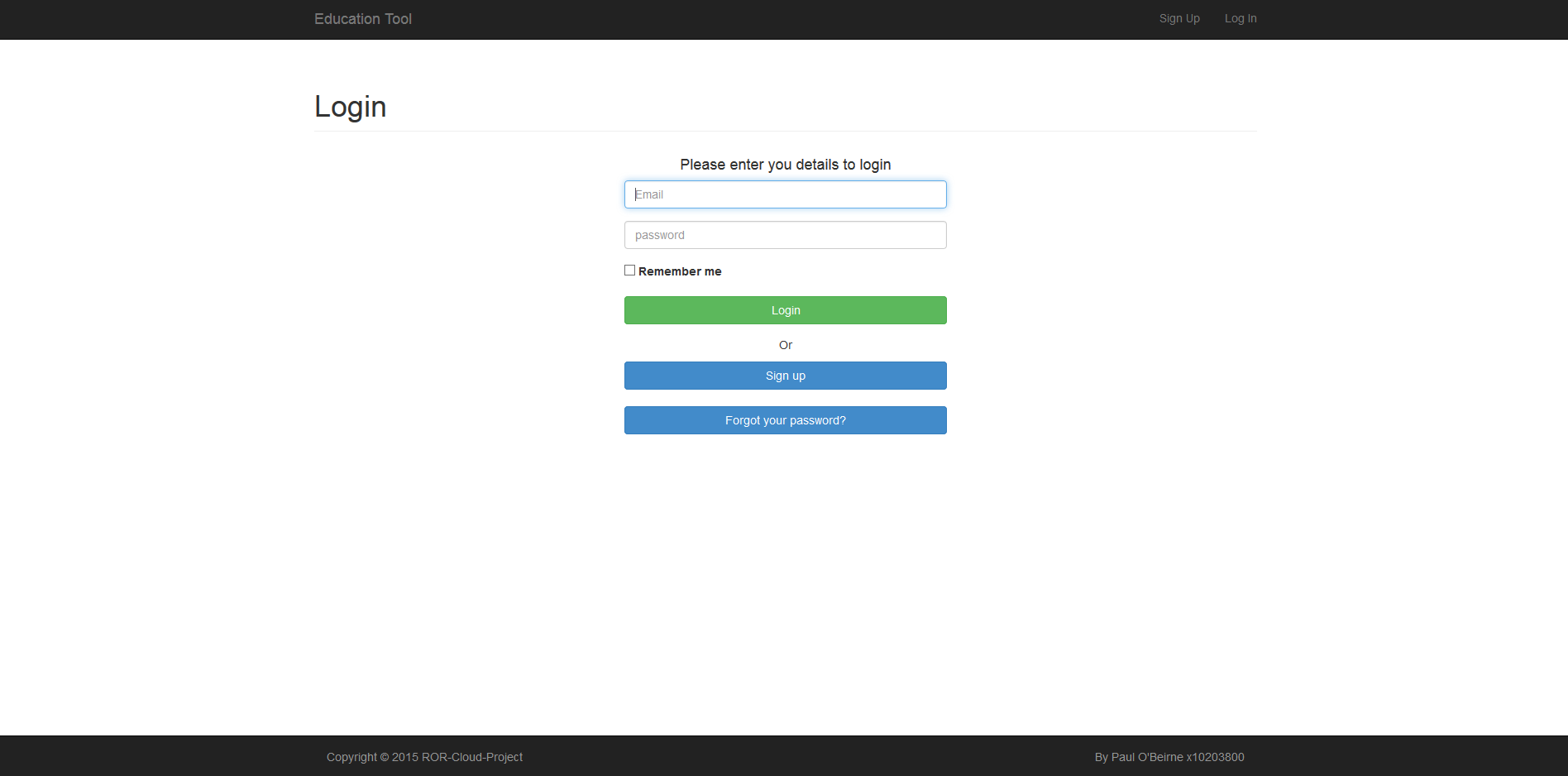
For the generated views devise created I customized them using twitter bootstrap to keep with the look and feel of the overall application see below for examples:

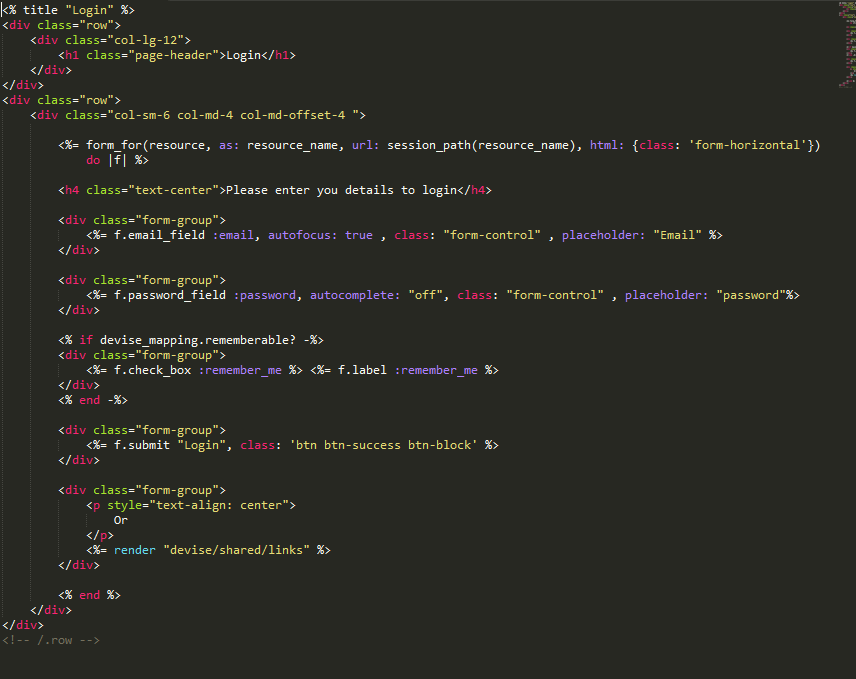
Sign up page

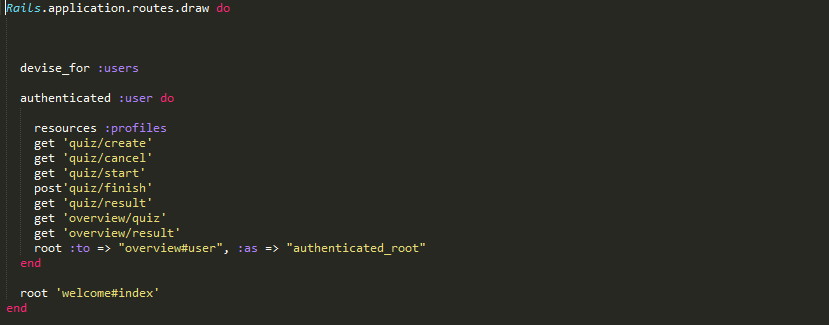




Login Page





Additional the Routes con-fig was setup to restrict unauthenticated users from being able to access certain paths.

**Gem :** kaminari

**Source :** https://github.com/amatsuda/kaminari

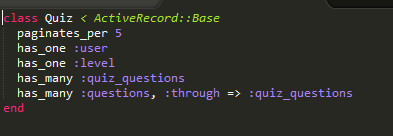
**Overview**

A Scope & Engine based, clean, powerful, customizable and sophisticated paginator for modern web app frameworks and ORMs

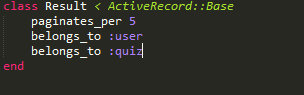
**Integration**

I want to display to the user a results and quiz table to allow the user to keep track with all the quiz’s take and results of each quiz. I installed the gem successfully and it generated a set of partial views. To this I needed to make some modifications and start by configuring the default per\_page value for the Quiz and Result models.

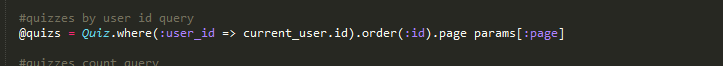
Quiz



Result



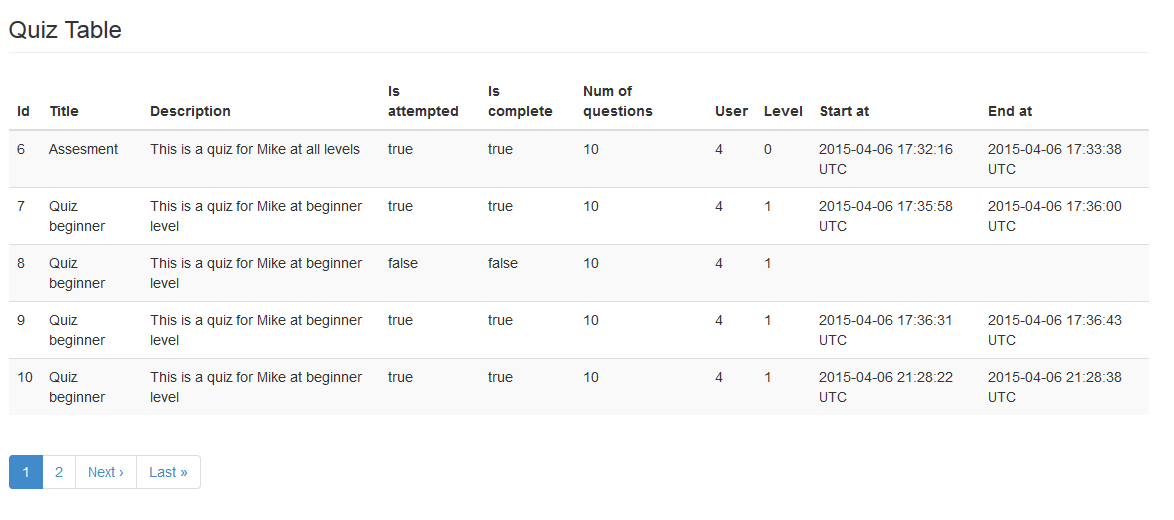
The next step was to implement it in the controller actions that where required.



And finally in the view I Just call the paginate helper:



This would produce the following result once there was sufficient records in the database for the user.



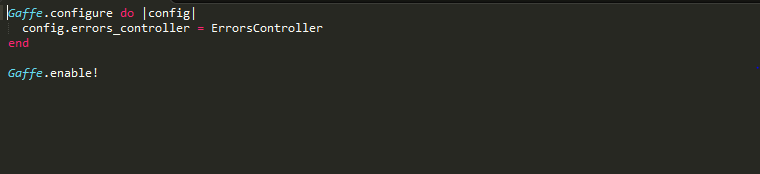
## Error Handling

The custom error handling pages used in my application are built upon a ruby gem called Gaffe. Gaffe makes having customized error pages in Rails applications an easy thing. To use this gem I simply added a reference in the gem-file and installed it through bundler. Once installed I followed the instructions outlined on the gems repository and created an initializer file with the required code to enable the gem.

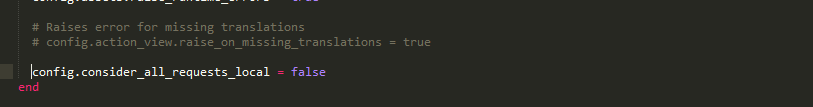
The instruction manual outlined how to further customize the gem features by specifying a new controller that will render error pages. A slight modification to the initializer file to tell it when an error is raised, the show action of Errors-controller will be called. Having a custom controller allowed me to make sure all users could see the error pages and if required customize the response.

The default error view rendered correctly but I wanted them to render inside my application layout so I specified inside the error controller that it should use my layout rather than a default one. I then created 9 error views in the errors view folder and customized using bootstrap and displayed the variables provided by the error controller. In the development environment rails prefers to render its own debug-friendly errors so to see the gaffes rendered errors I had to edit the development environment con-fig to make Rails use “exceptions\_app” in development.

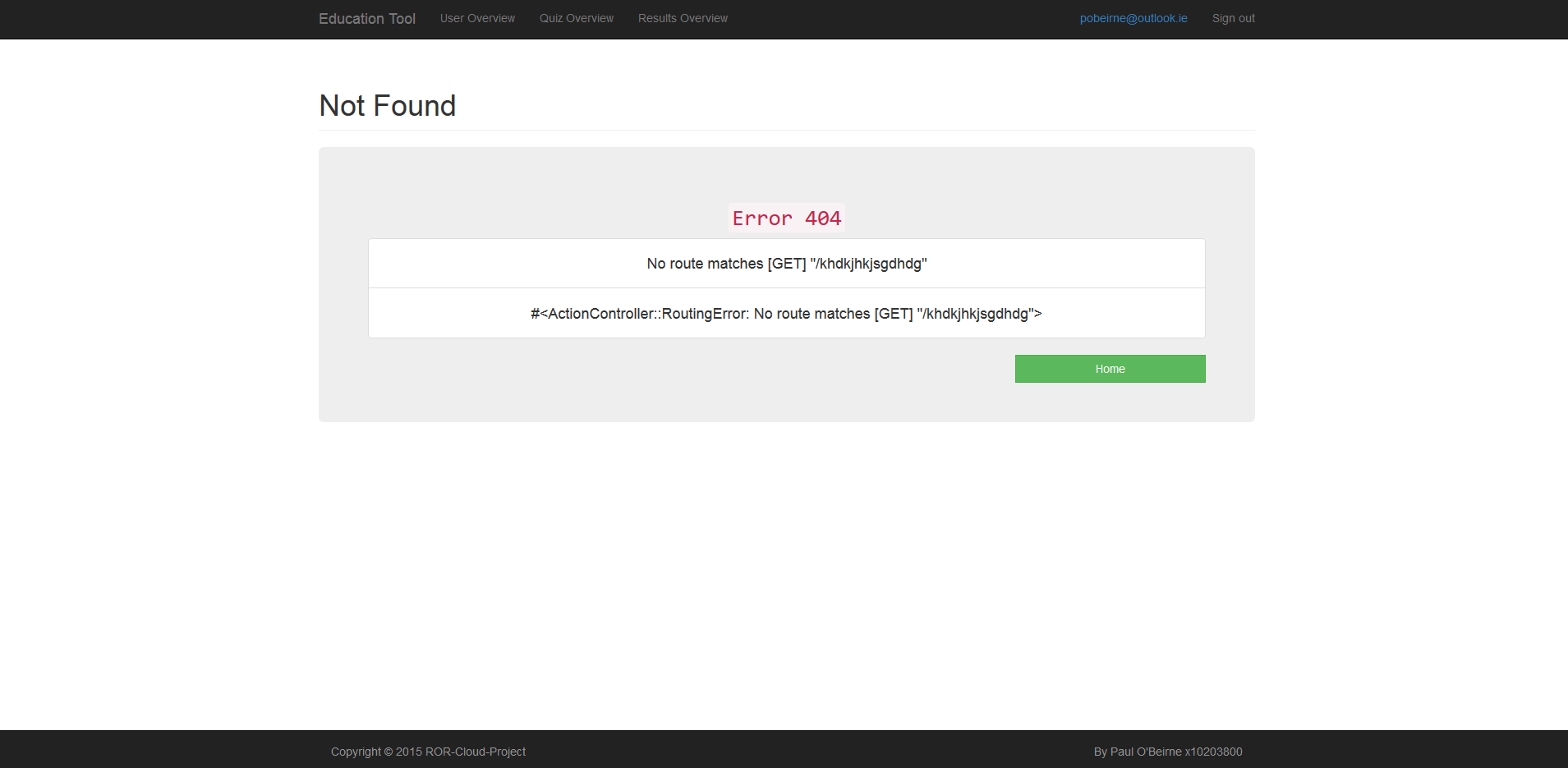
Gaffe Initializer file



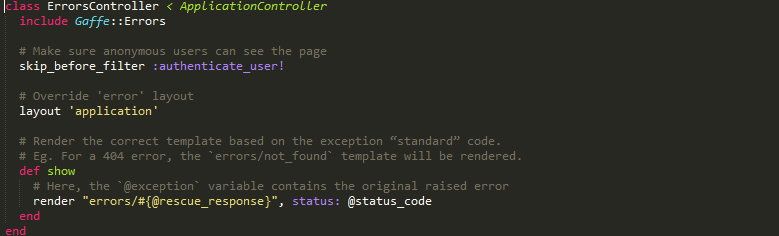
Development environment con-fig



Not found view



Errors Controller



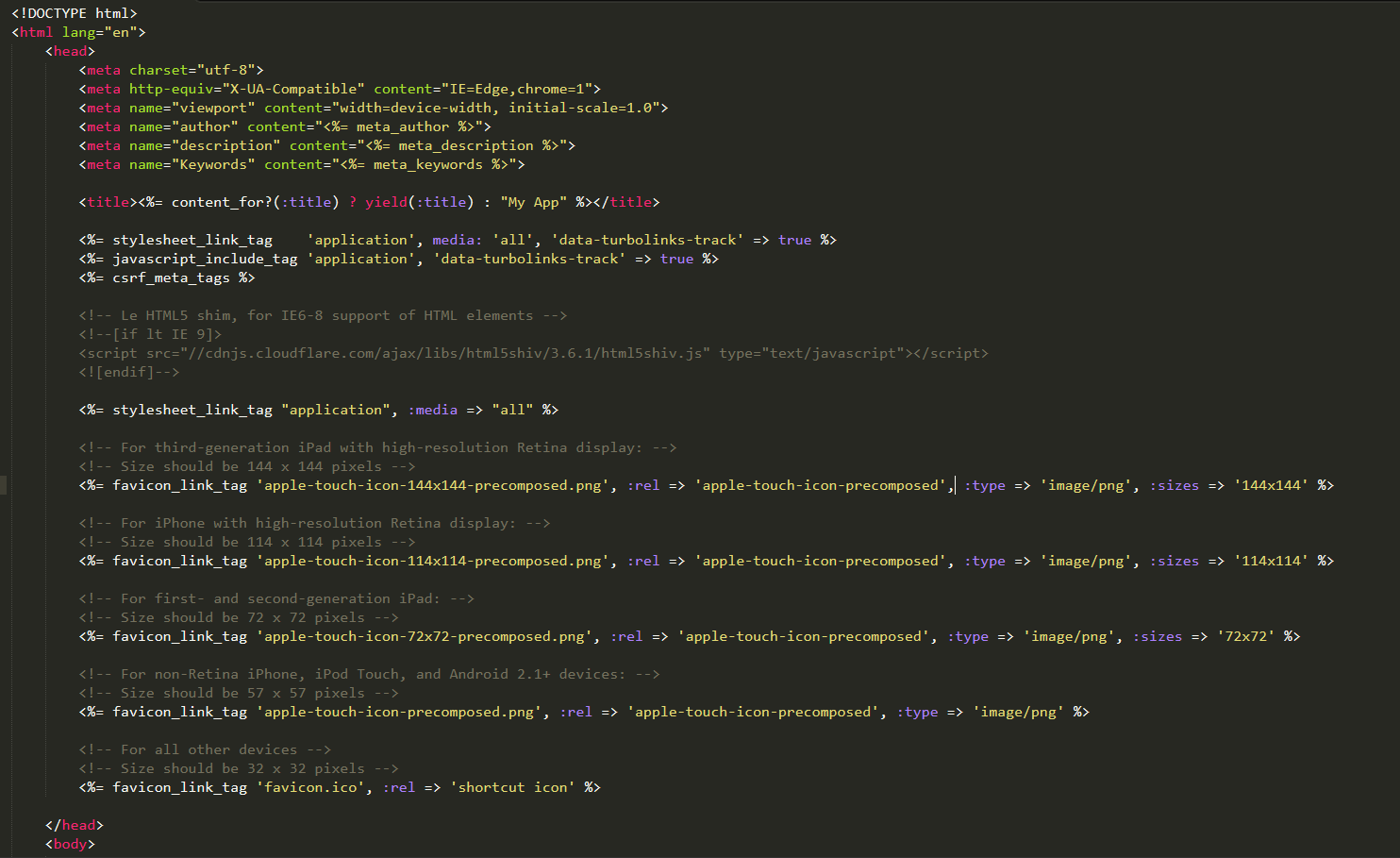
Not Found view code

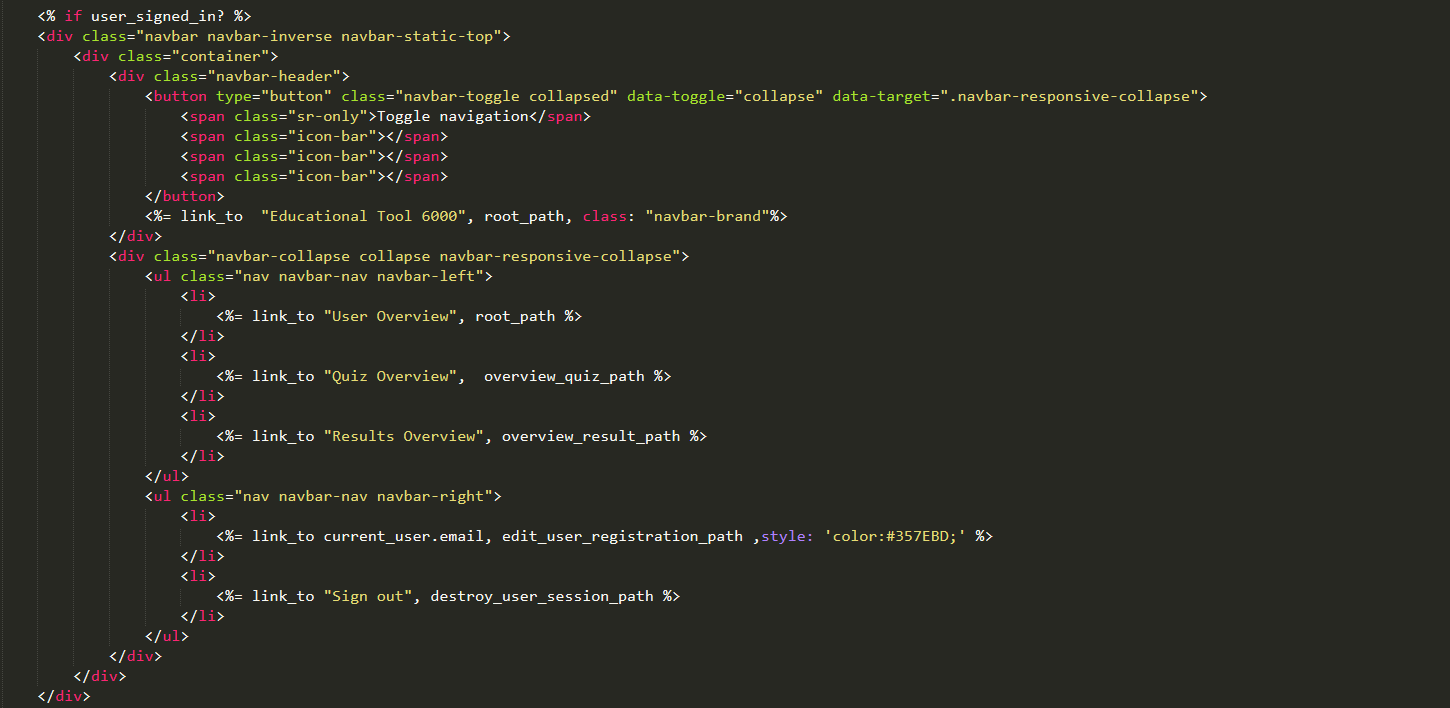


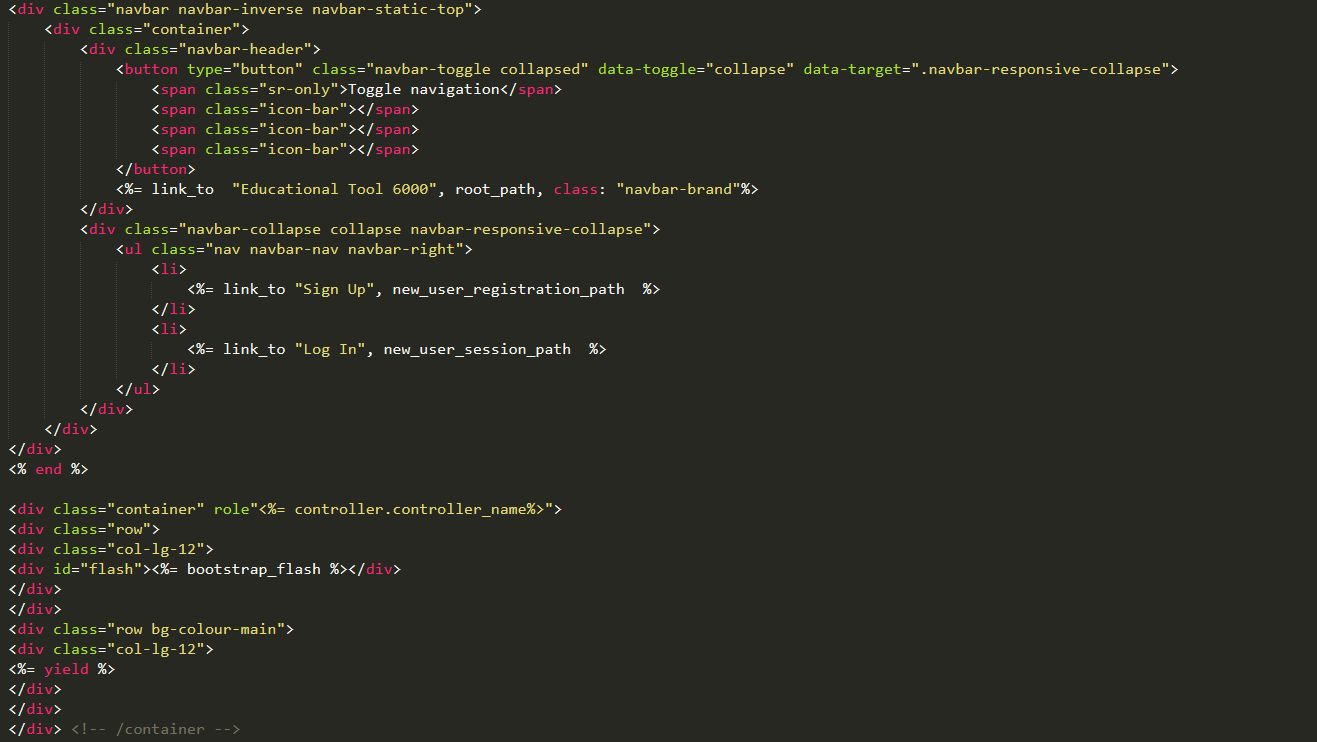
## User Interface Design

The application layout and client interfaces used twitter bootstrap components and features. The application layout contained the navigation menu, footer and content placeholder in which other views content would be rendered into. With the installed gem twitter-bootstrap-rails I was able to further make use of the framework by convert existing generated scaffolding to bootstrap themed layouts.

Application Layout

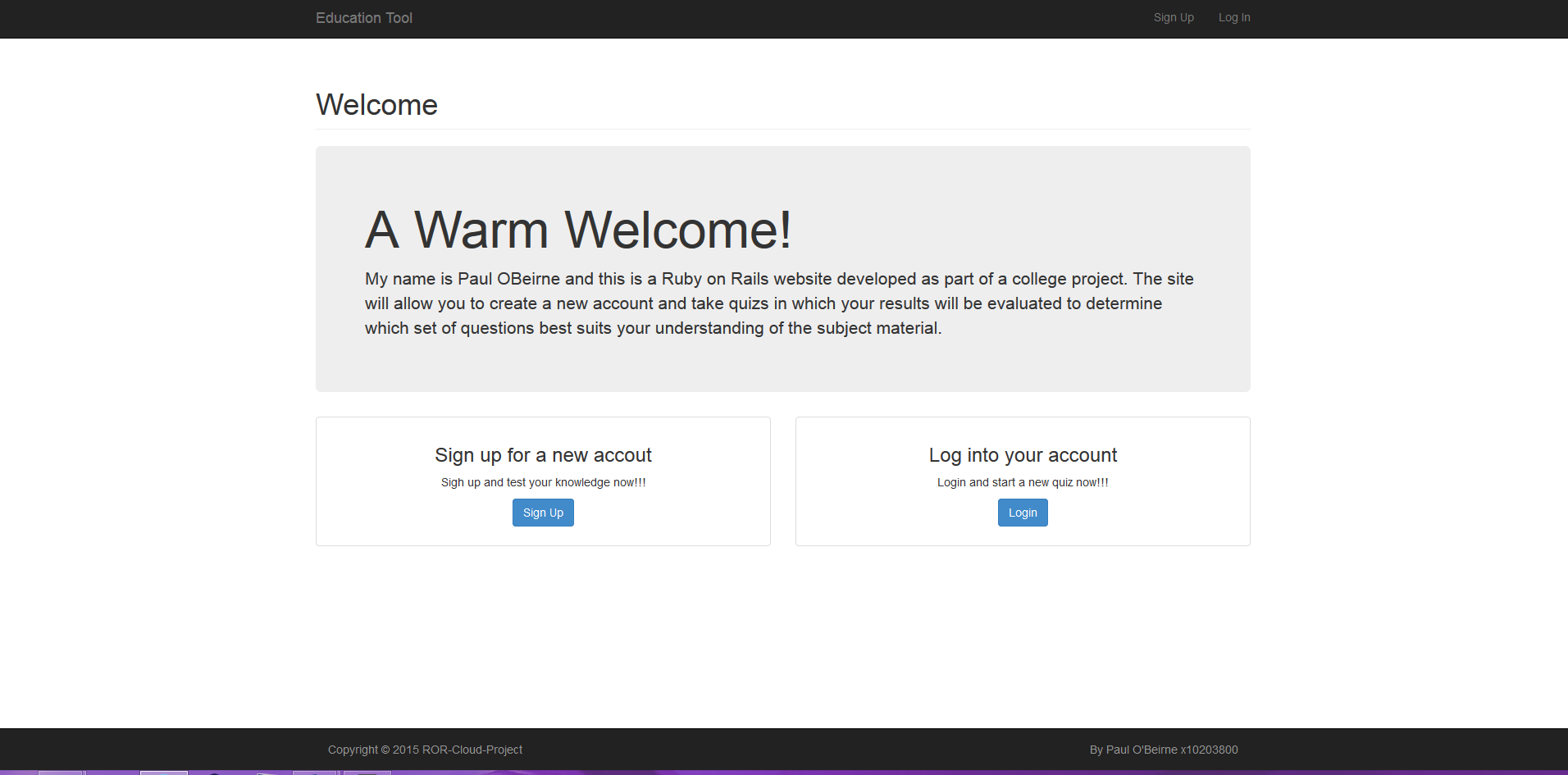




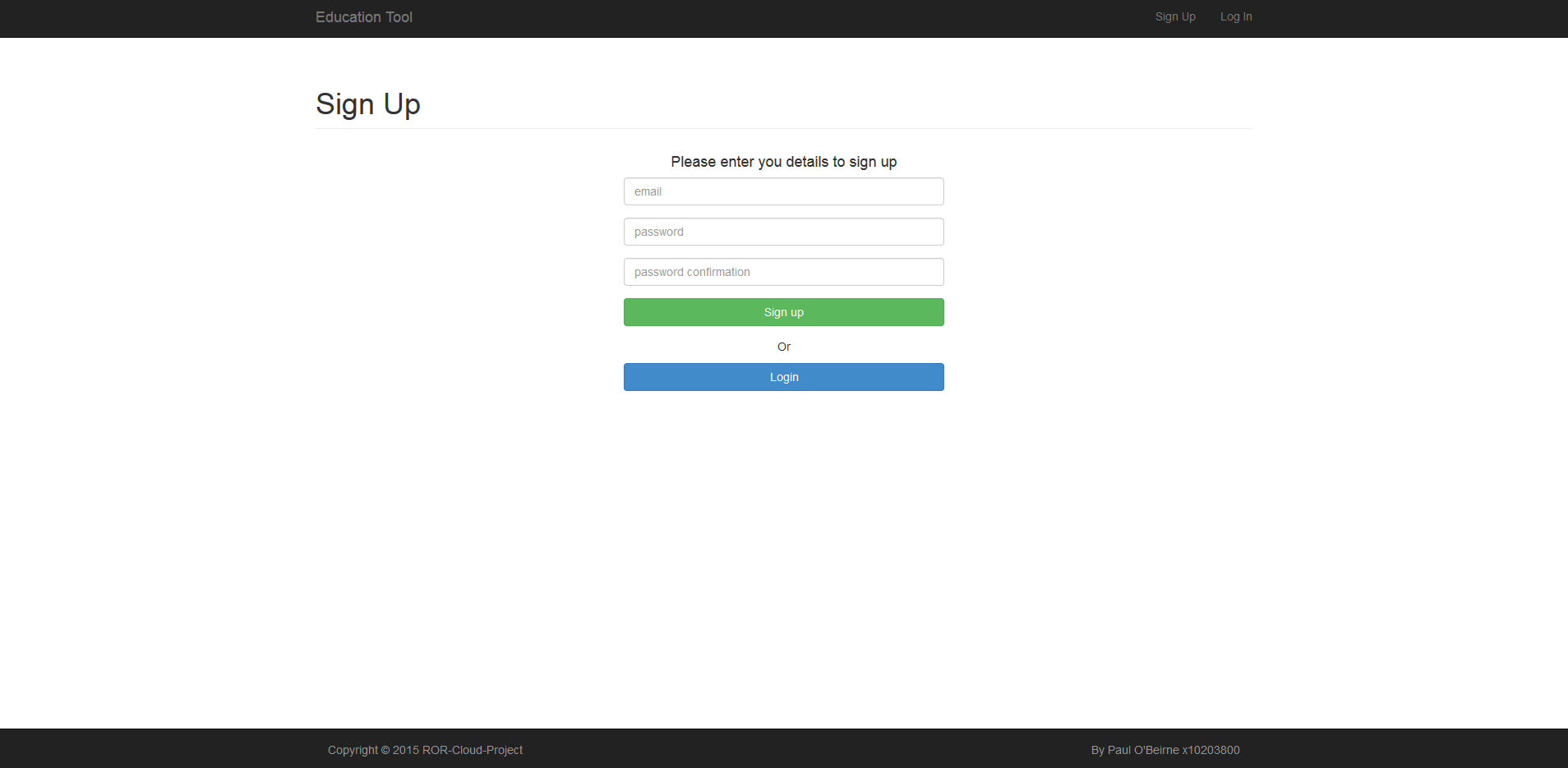




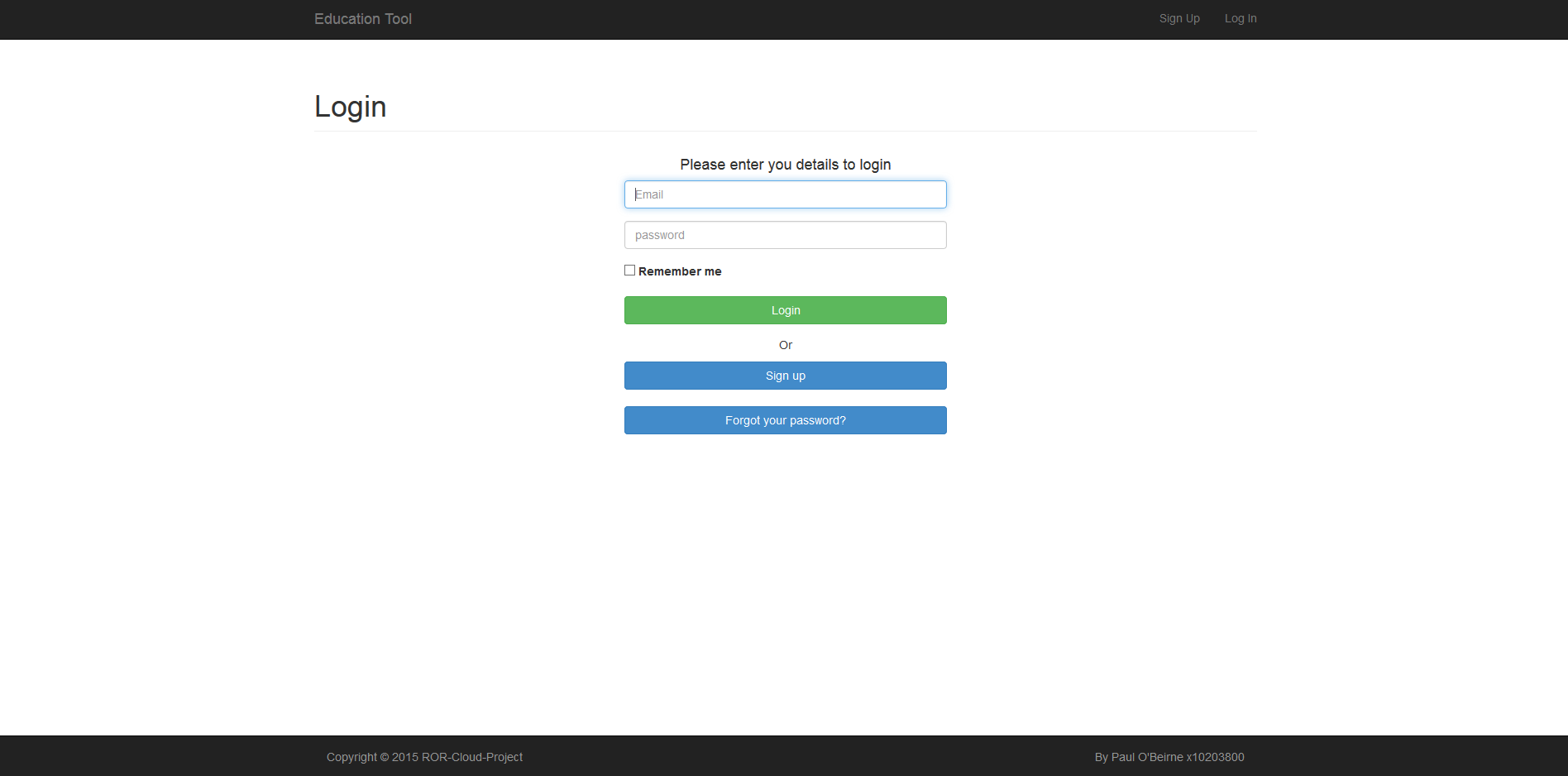
Welcome page



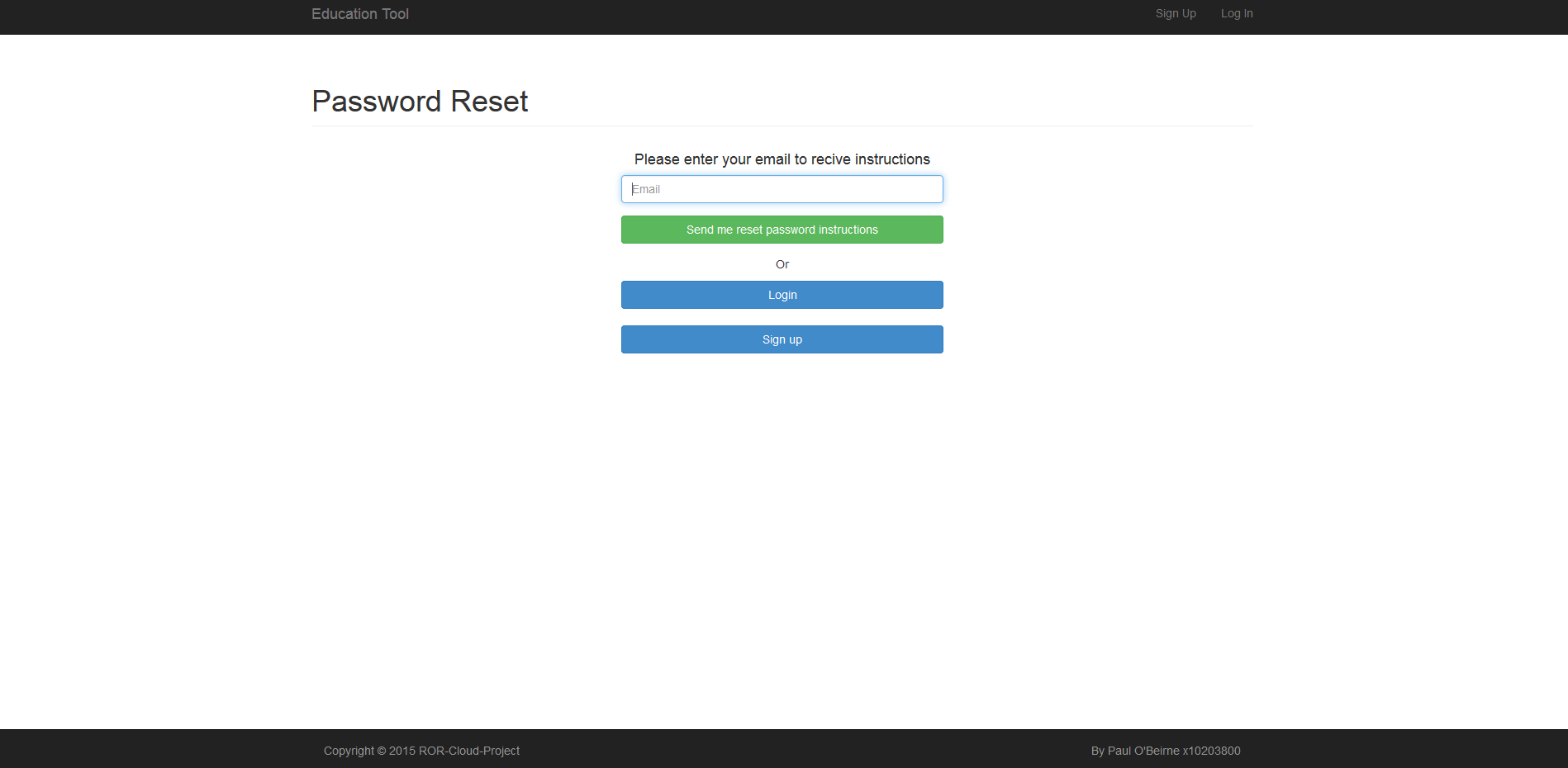
Sign up page



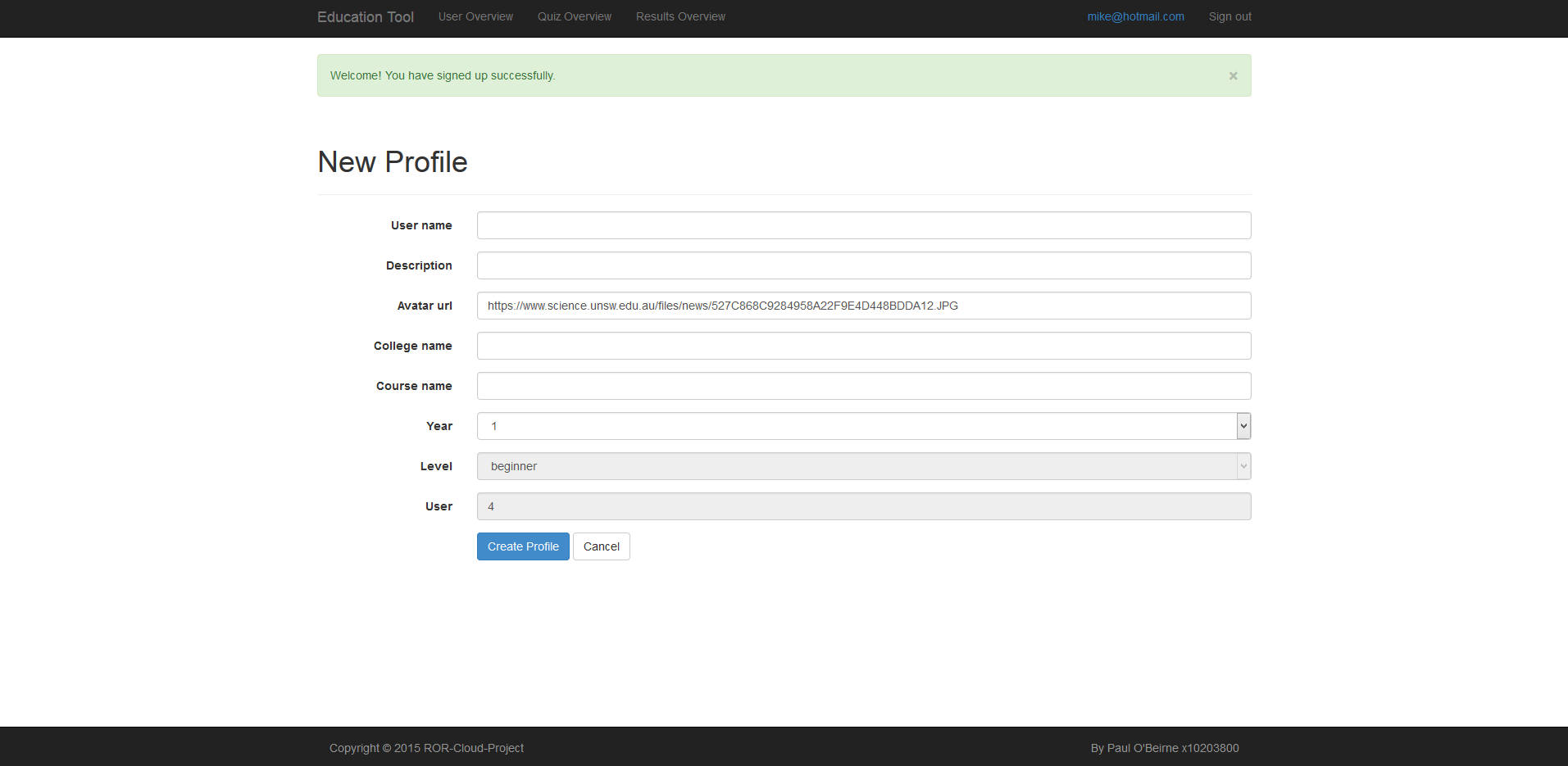
Login Page



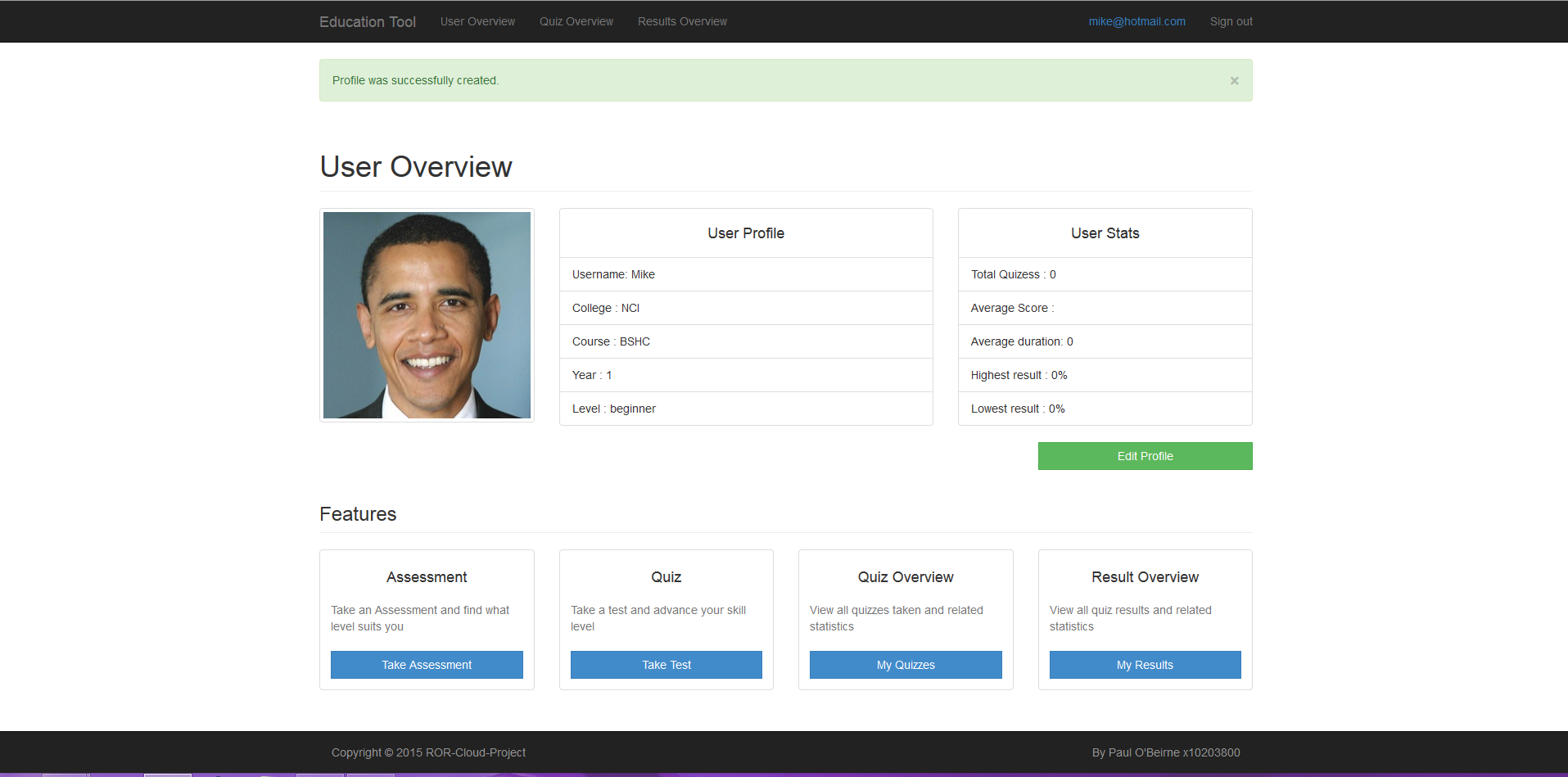
Password Reset page



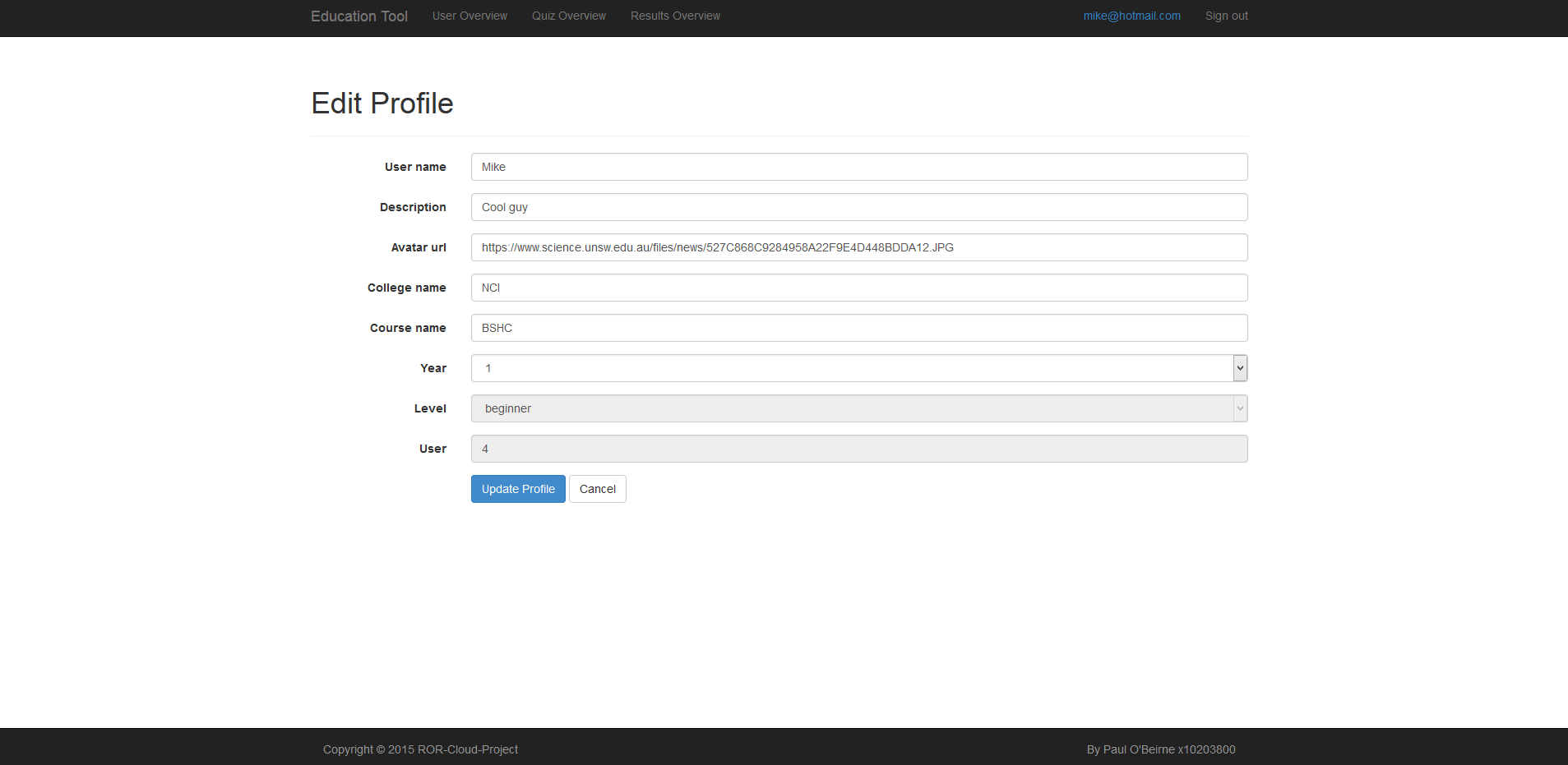
On sign up redirected to profile page



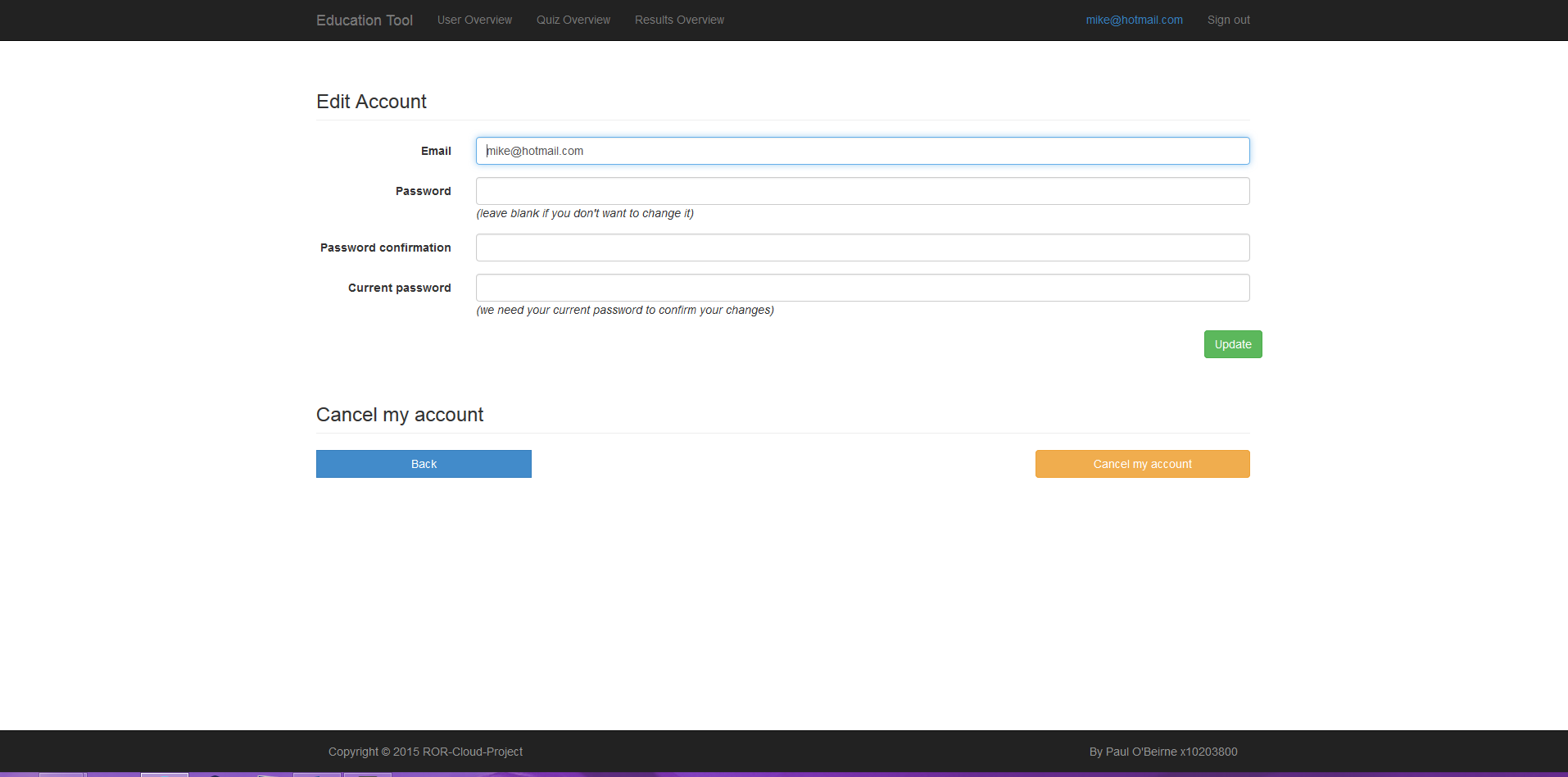
Once profile is saved the user is redirected to the user overview page



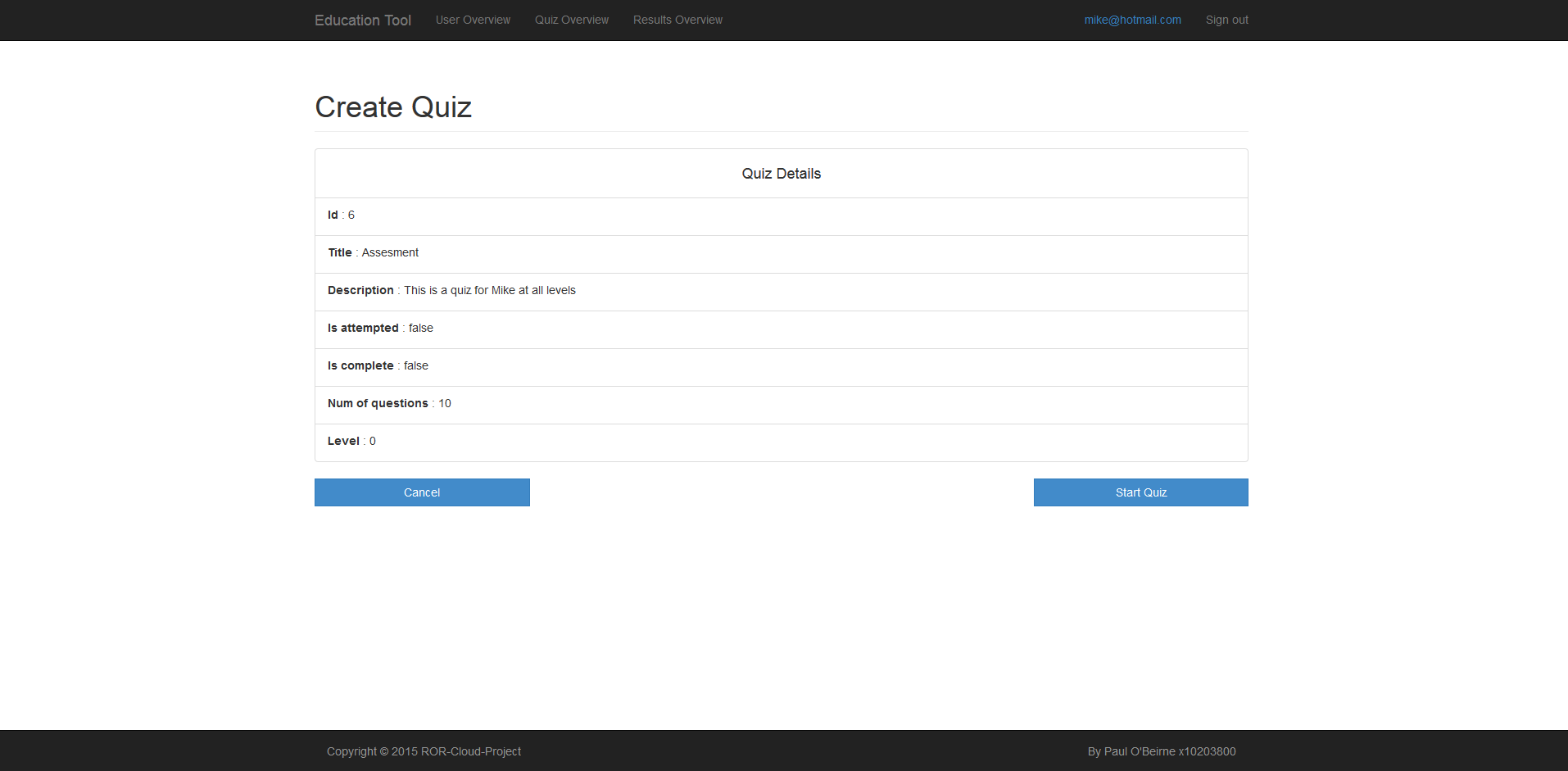
Edit profile page



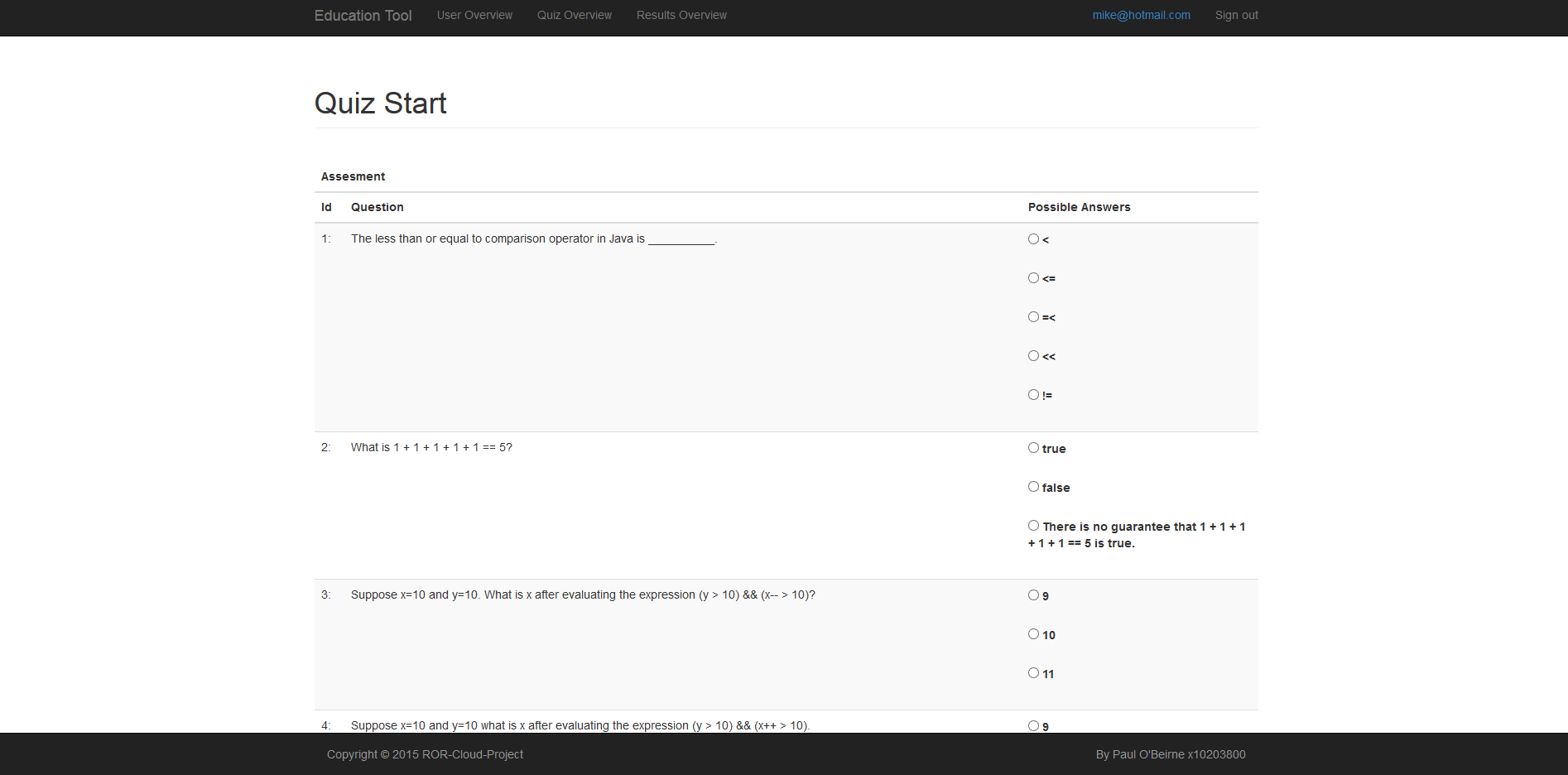
Edit Account page

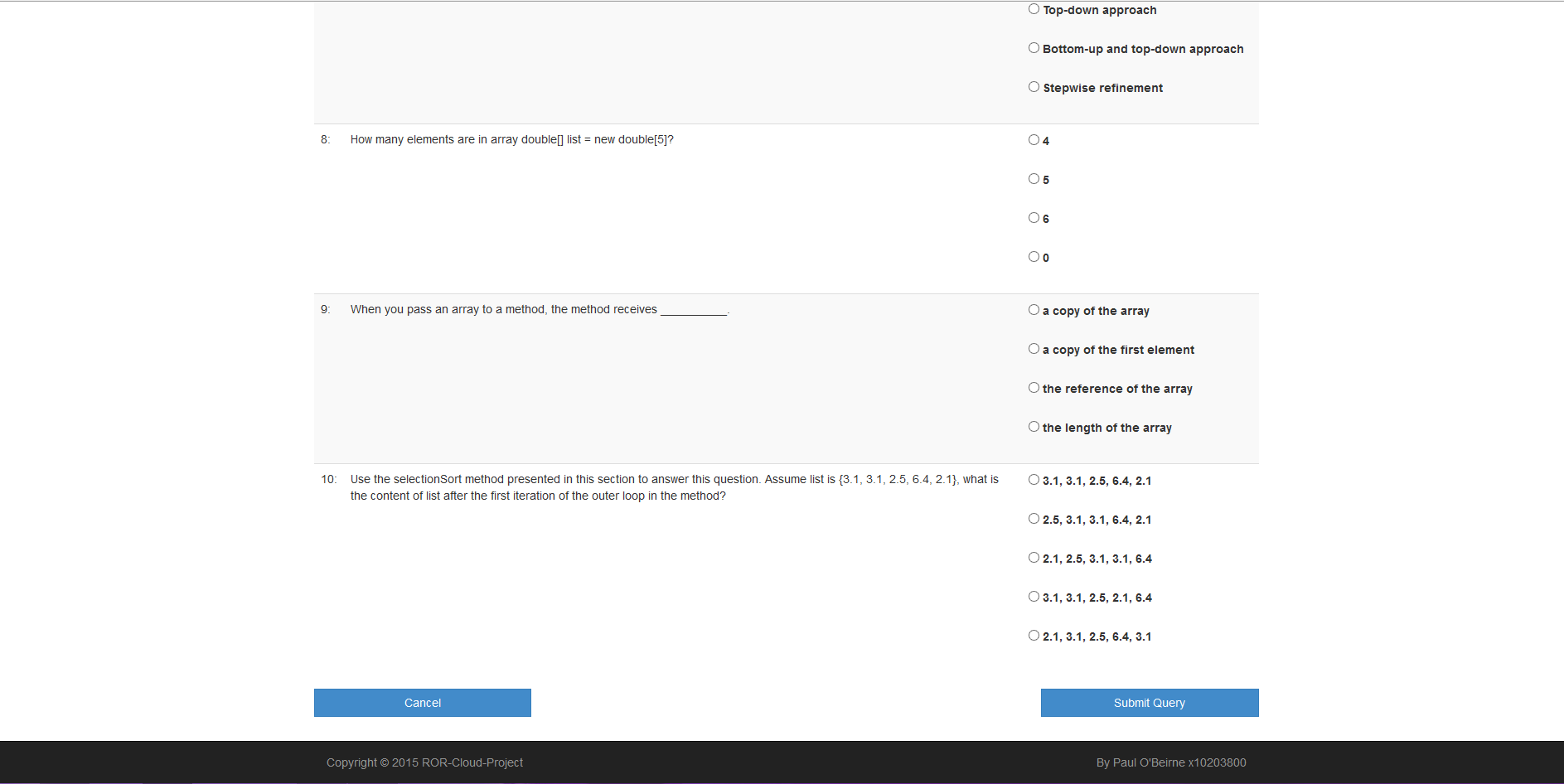


Quiz Create Page

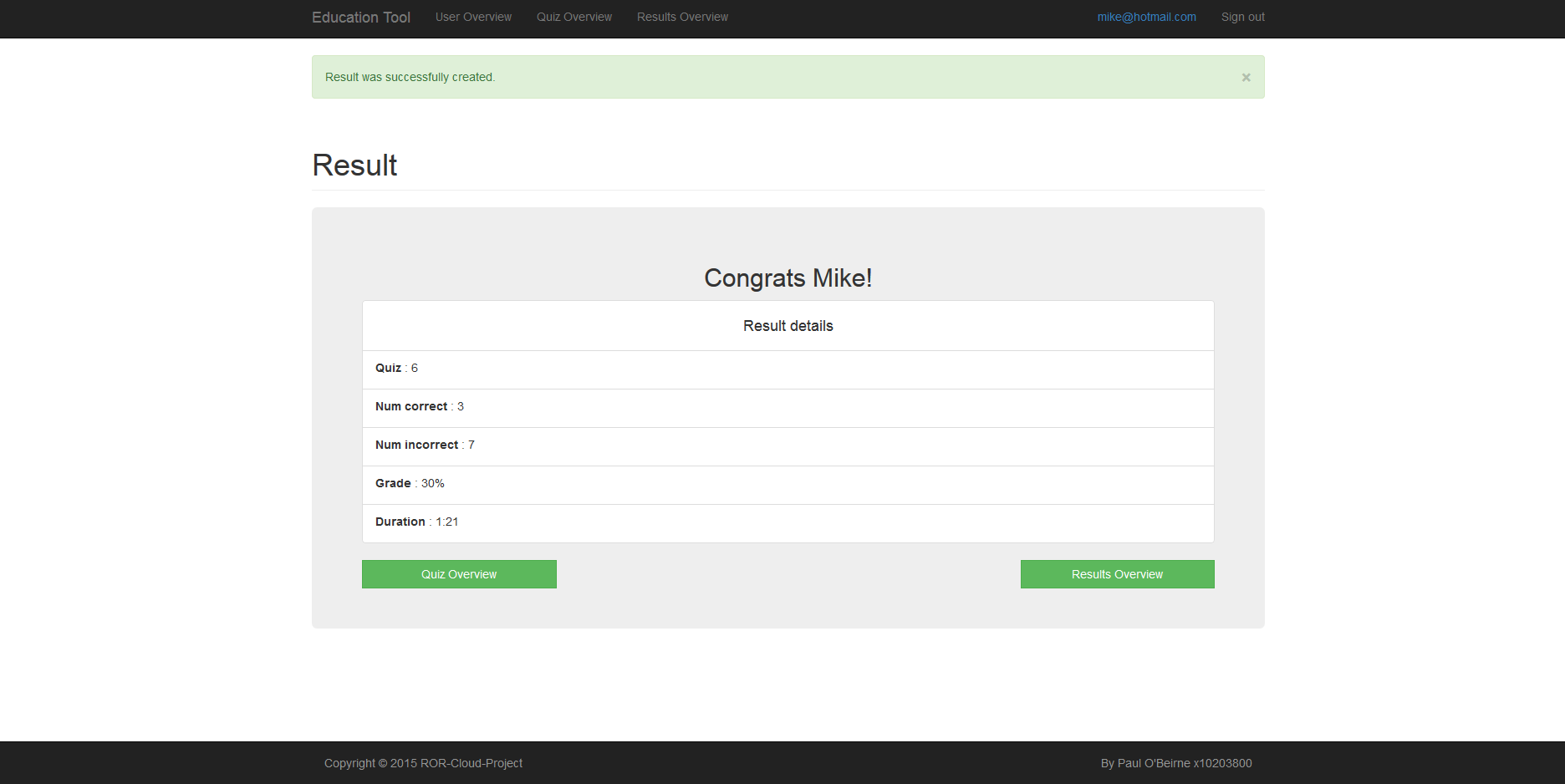


Quiz start page

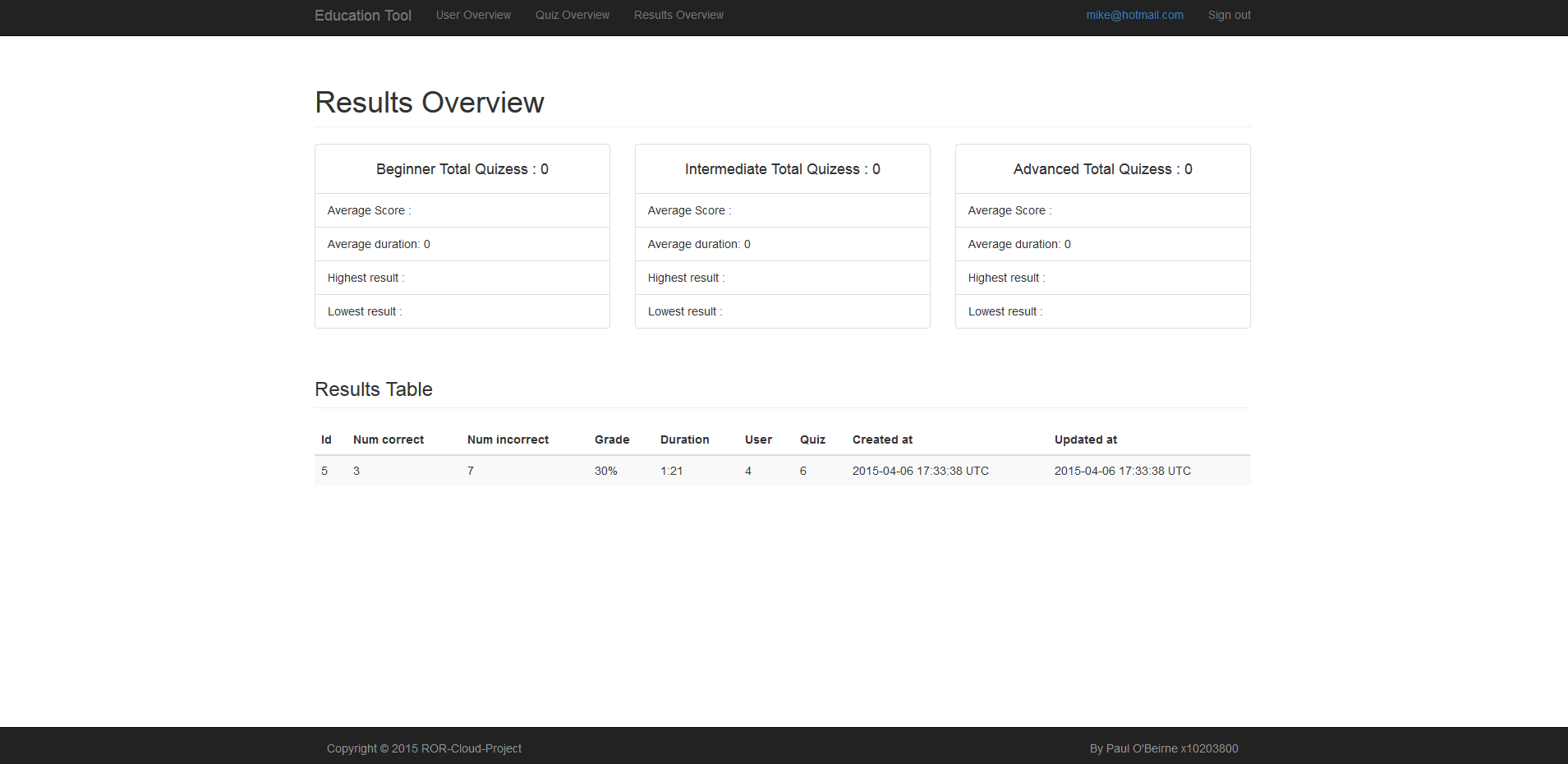




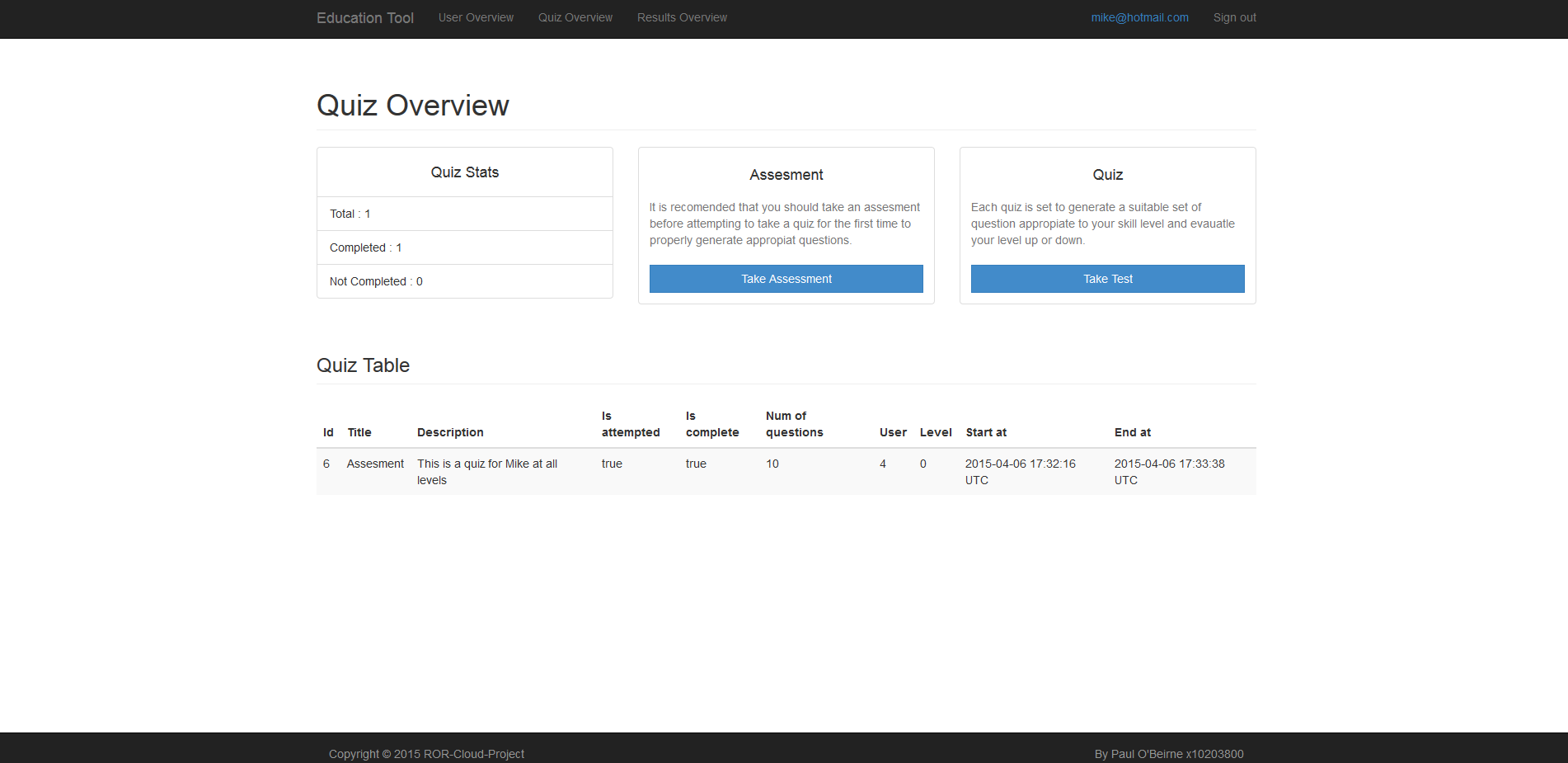
Quiz result page



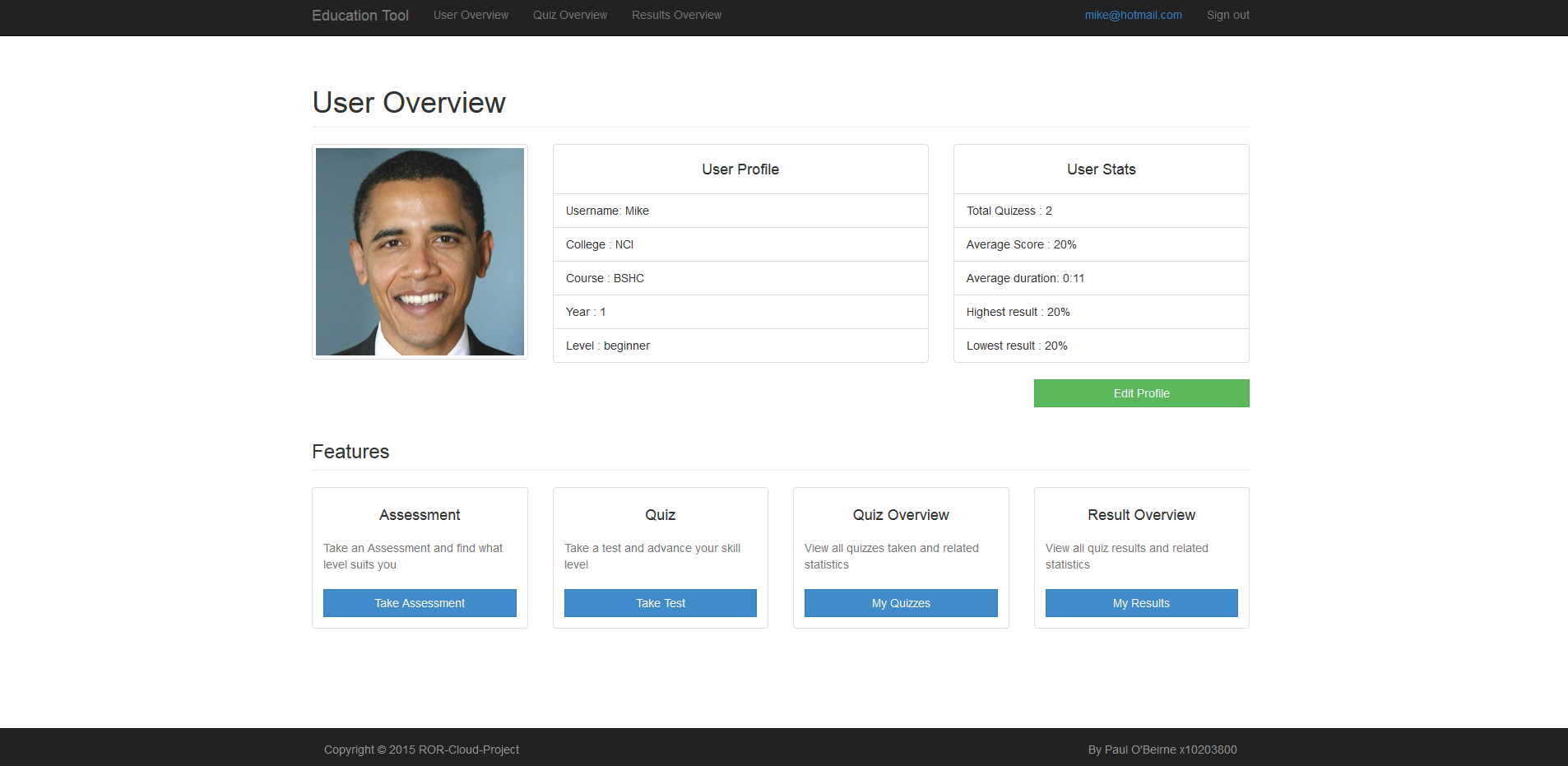
Results Overview Page



Quiz Overview Page



Updated Profile page



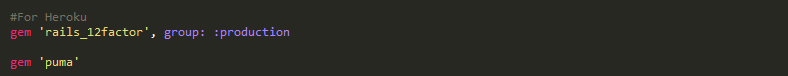
# Deployment

Application Link: http://nameless-chamber-7595.herokuapp.com/

Heroku is a cloud-based development platform as a service (PaaS) provider. I chose to deploy on Heroku as it had good support and instructions for Rails and PostgreSQL setup. Heroku has a basic tutorial on their website that allowed me to be up and deployed (including PostgreSQL database) within 10 minutes of starting the tutorial. Another reason was my familiarity with using GIT and Heroku from the previous project. Git is a powerful decentralized revision control system, and is the means for deploying apps to Heroku.

Once I created the initial git repositiory locally and deployed it to heroku I could work with the knowledge that each succesive change to the application would be pushed to Heroku and reduce any loss I might face. This was important in the beginning faze of the develoment process as I was using the virtual OS to devolpe under. I ran into an error sortly after which caused the virtual image to be corrupted.

**Gems used for Heroku**



**Gem: rails\_12factor**

Source:https://github.com/heroku/rails\_12factor

Description: Rails gets a lot right when it comes to twelve-factor apps, but it could still be better. The two biggest areas right now are that in production logs should be directed to stdout and dev/prod parity while delivering assets. This gem enables serving assets in production and setting your logger to standard out, both of which are required to run a Rails 4 application on a twelve-factor provider. The gem also makes the appropriate changes for Rails 3 apps.

**Gem: puma**

Source: <https://github.com/puma/puma>

Description: Web applications that process concurrent requests make more efficient use of dyno resources than those that only process one request at a time. Puma is a webserver that competes with Unicorn and allows you to process concurrent requests. Puma uses threads, in addition to worker processes, to make more use of available CPU. You can only utilize threads in Puma if your entire code-base is thread safe. Otherwise, you can still use Puma, but must only scale out through worker processes.

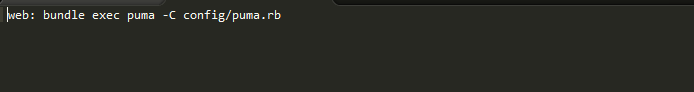
**Specify Ruby version in app:**

Rails 4 requires Ruby 1.9.3 or above. Heroku has a recent version of Ruby installed, however you can specify an exact version by using the ruby DSL in your Gemfile. At the end of Gemfile add:



**Procfile**

By default, the web process runs rails server, which uses Webrick. This is fine for testing, but for production apps you’ll want to switch to a more robust webserver. On Cedar, we recommend Puma as the webserver. Specify the webserver explicitly in the Procfile.



# Conclusion

I was a little apprehensive about how successfully I would be able to complete my application project to a decent standard which may have made me less enthusiastic about starting the project. The first few days of development was fraught with mistakes and head scratching but I was able to implement what is was trying to do in parts.

I made quite a few errors in the beginning when issuing commands to destroy scaffolding which in turn removed models, controllers and migration’s I was working on. I also forgot while using Aptana studio that there was limited undoing and redoing available to the user. Another crucial issue that I ran into was the default creation of a rails application sets the database to sqlite3 which was not optimal for deployment with Heroku.

After the first week of trying to get the feel for it I decided to start from the beginning and document what I had learnt to date. I guess the most confusing part was in essence it was all rather too simple to get up and running if you use the right commands from the very beginning. The second time round I had a new application up and running and stored in a GIT repository within the space of 10 minutes.

I took what parts I learnt from my failures and was able to implement them successfully into the new application seamlessly. The installation of gems was relatively straight forward and there was sufficient resources online if help was required. Creating the custom gems was a breeze nothing to complicated involved in the whole process. I did initially just create a two classes in the library folder and get the functionality implemented and working with the project before converting the classes to gems. I deployed both gems to GitHub and when installing the gems in my application specified the repositories which is pretty cool.

The whole interface design was also simple with the installation of a twitter bootstrap gem. The gem was able to convert generated scaffolding views into a bootstrapped themed view. Which gave me some solutions and useful examples for converting some of the devise views.

Working with data was easy enough bar figuring out some nil errors appearing and requiring some escape clauses. Querying data was somewhat difficult at first but after a while you start to see how similarly structured it is to other querying languages and SQL statements.

The main issue I had with rails was in attempting to go outside the scaffolding to create custom controllers and views. Especially custom forms that don’t have and model associated with them. I tried to create some custom form model to handle verification but didn’t succeed in the end. Another issues would be around the need sometimes to pass more than one model back to the view which I take it is not 100% proper but can be done. The lack in the ability to create a custom view model really didn’t help me some of my views having multiple models.

My overall view on the Rails framework is that once you have some prior knowledge of how to setup your application properly you can rapidly develop a full-fledged application in a short time I really did enjoy using it at the end and will more than likely try keep learning it after college.

**References**

Active Record Associations — Ruby on Rails Guides. 2015. Active Record Associations — Ruby on Rails Guides. [ONLINE] Available at: <http://guides.rubyonrails.org/association_basics.html>. [Accessed 07 April 2015].

Getting Started with Rails 4.x on Heroku | Heroku Dev Center. 2015. Getting Started with Rails 4.x on Heroku | Heroku Dev Center. [ONLINE] Available at: <https://devcenter.heroku.com/articles/getting-started-with-rails4>. [Accessed 07 April 2015].

plataformatec/devise · GitHub. 2015. plataformatec/devise · GitHub. [ONLINE] Available at: <https://github.com/plataformatec/devise>. [Accessed 07 April 2015].

seyhunak/twitter-bootstrap-rails · GitHub. 2015. seyhunak/twitter-bootstrap-rails · GitHub. [ONLINE] Available at: <https://github.com/seyhunak/twitter-bootstrap-rails>. [Accessed 07 April 2015].

puma/puma · GitHub. 2015. puma/puma · GitHub. [ONLINE] Available at: <https://github.com/puma/puma>. [Accessed 07 April 2015].

heroku/rails\_12factor · GitHub. 2015. heroku/rails\_12factor · GitHub. [ONLINE] Available at: <https://github.com/heroku/rails_12factor>. [Accessed 07 April 2015].

mirego/gaffe · GitHub. 2015. mirego/gaffe · GitHub. [ONLINE] Available at: <https://github.com/mirego/gaffe>. [Accessed 07 April 2015].

amatsuda/kaminari · GitHub. 2015. amatsuda/kaminari · GitHub. [ONLINE] Available at: <https://github.com/amatsuda/kaminari>. [Accessed 07 April 2015].