

Report on the Development of a WebSite for the National Skeptics Association using Ruby-on-Rails (v3.2)

Thomas Dowling¹
(x12113441)

¹National College of Ireland.

Web Application Frameworks
M.Sc in Web Technologies

Lecturer: Jonathan McCarthy

January 2, 2014

Home

Commodities

Gems

Profile

Discussions

Users XML View

Comments

Calculator

RubyGems/Git

EDIT PROFILE / LOG OUT / LOGGED IN AS THOMASGDOWLING@GMAIL.COM (ADMINISTRATOR) /

Your Cart


Thomasgdowling

	Unit Price	
1 xSlipper Orchid	€49	€49
1 xAgile Web Develop...	€48	€48
1 xCoffeeScript	€64	€64
1 xRails Test Prescr...	€65	€65
Total Price		€226

Empty cart

Checkout

National Skeptics Association



Marie Curie

Contents

1	Introduction	1
2	Architecture	2
3	General Implementation Methods	2
3.1	Programs and Frameworks	2
3.2	External Gems	3
3.3	Gems developed in this Work	3
3.4	Ajax	5
3.5	Append to and Sort an XML file when a New User Registers	5
3.6	Use XMLHttpRequest to View an XML File Dynamically	6
3.7	Validation	7
3.8	Mailers	8
3.9	Unit Testing and Test-Driven Development	8
3.10	Paperclip and Image Upload	9
3.11	Search	9
3.12	Will_paginate	9
3.13	Devise	9
3.14	Gravatars	9
3.15	Before Filtering	10
4	Specific Implementation Components	10
4.1	Shopping Cart/Commodities Page	10
4.2	Check-out	11
4.3	Dynamic Users XML File	11
4.4	Static Users XML File	11
4.5	Profile Pages	11
4.6	Discussion Pages and User Comments	12
4.7	Comments Page	12
4.8	Authentication Logic	12
5	GitHub and RubyGems	13
6	Browser	13
7	Discussion	13
8	Declaration	13
9	References	14

1 Introduction

The goal of the project was to build a web site for a fictional society, the National Skeptics Association, with the following requirements.

It was desired that the site would have log-in facilities where three types of user were envisaged: An unregistered user, a registered user and an administrator. An unregistered user should be encouraged to register, but should be able to browser the site with restricted privileges even while not registered.

A registered user should be able to log-in with an email address and password, be able to update that password, and be able to easily generate a new password from a lost-password request.

It was envisaged that the site would have full e-commerce functionality that allows a user to purchase products from a Commodities page. The Commodities page should be searchable.

In addition, a shopping cart should be visible on every page that allows a user to easily add and delete items, to at all times see the quantity, price and total cost of all items purchased, and to proceed to checkout. A key requirement is that the shopping card be user-friendly. The cart should not be visible unless an item has been added, but a registered user should be aware when the cart is empty.

When a registered user makes a purchase from the Checkout page, she/he should receive an email message detailing the items purchased

The name of a logged-in user should appear on the nav-bar, together with the status of that user (administrator or non-administrator).

A registered user's Gravatar should appear on the nav-bar.

When a new user registers, her/his details should be appended to an XML file that may be viewed dynamically from a web page by a user with administration status (and only by such a user).

In addition, an administrator should be able to view a formatted version of the 'raw' XML file.

All registered users should have a Profile page which the user may edit. In addition, a registered user should be able to upload an image to the Profile page, and edit that image if desired. A registered user should only have access to his/her Profile page.

It was also a requirement that a user with administration status could create Discussion pages, upload images to the Discussion pages, and edit the pages.

In addition, registered users may leave comments on the Discussion pages which will be visible to all users as soon as posted. A registered user should be able to edit his/her own comments **but not those of another user**. An administrator should be able to view, search, edit and delete all comments.

It was desired that the database be protected from invalid and incomplete entries.

This report outlines how the above requirements were implemented.

2 Architecture

The architecture of the site, as viewed from the point of view of a user browsing to the home page, is shown in [Fig. 1](#). It is largely self-explanatory, but a couple of points require comment.

An unregistered user will have access to all pages except the Profile, Comments, Shopping Cart, Checkout, Dynamic XML and Static XML pages. In addition, an unregistered user may not leave comments (but may read the comments of other users), and may not purchase products (but may search the database of Commodities).

Upon registration, a user will have access to an automatically generated Profile page (which may be edited), may purchase commodities, and may leave comments. A registered user without administrator status will not have access to the Comments page, the Dynamic XML Page, or view the static ('raw') XML file.

An administrator has access to all pages.

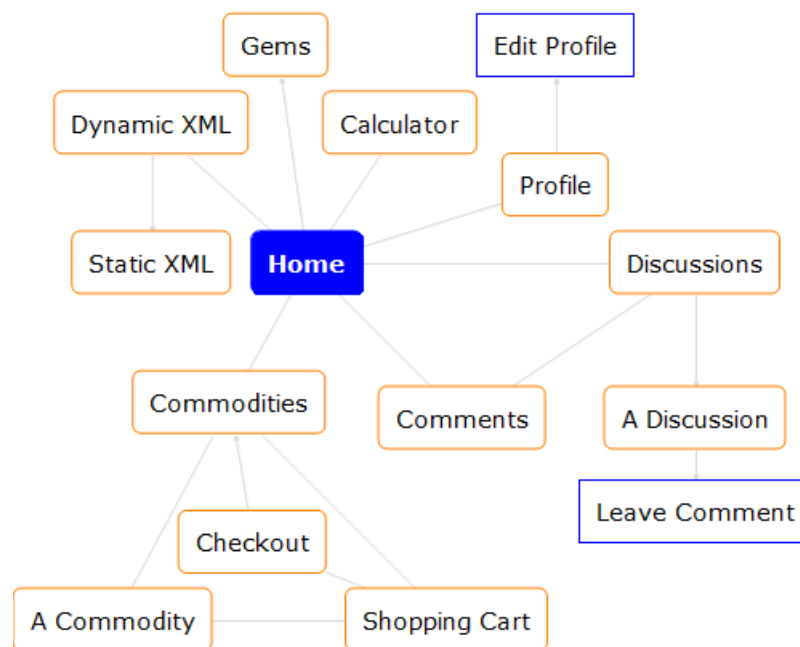


Figure 1: Site Architecture as Viewed From Home Page

3 General Implementation Methods

3.1 Programs and Frameworks

The website was developed using Ruby(v 1.9.3p0) as the primary programming language and Ruby-on-Rails (v 3.2.14) as framework.

A 'skeleton front-end' was created using the [Zurb Foundation](#) CSS framework.

JavaScript was used primarily to create dynamic content integrated with an XMLHttpRequest (to dynamically display a list of users from an XML file).

jQuery was used in Ajax requests (see below, [Section 3.4](#))

jQuery-UI was used to add advanced functionality features, such as making the shopping cart HTML div draggable.

The [Mailcatcher](#) gem was used to 'catch' emails (issued when the user purchased a product, say) in development mode.

The [Paperclip](#) gem was used to allow users to upload photographs to a profile page, and to allow administrators to upload images to the Discussion forums.

The application was built on a Dell PC with Ubuntu 12.04 as operating system.

3.2 External Gems

The following gems were used

[annotate](#) Annotates Rails models based on the database schema

[will-paginate](#) Advanced pagination functionality for Rails.

[faker](#) Allows the generation of 'fake' data

[foundation-rails](#) Installs the Zurb-Foundation CSS framework

[jquery-ui-rails](#) Makes advanced jQuery features available

[devise](#) Advanced user authentication functionality

[nokogiri](#) Generate and sort XML files

[gravatar-image-tag](#) Displays a user's gravatar

[paperclip](#) Allows user to easily upload images. [ImageMagick](#) must be installed for full functionality

[rspec-rails](#) Advanced test-driven development

[mail](#) Great mailer functionality becomes available

[mailcatcher](#) Allows emails to be 'caught' in development mode. May be accessed via port 1080 (localhost:1080).

[figaro](#) Rails configuration using ENV and a YAML file (used in conjunction with the [mail](#) gem)

3.3 Gems developed in this Work

Three gems were created in this project, and published on RubyGems. All may be included in the Gem file of a Rails application and installed using the `bundle install` command.

[JavaScript Calculator\(jscal\)](#)

This gem was created using a Rails engine (`Rails::Engine`) method and allows a JavaScript calculator (written by [Tom Dowling](#)) to be incorporated into a Rails application. (The engine method allows the incorporation of ‘mini-apps’ into gems).

[jscalc](#) has so far been downloaded more than 1300 times from [RubyGems](#). In the present work, a clash with [Zurb-Foundation](#) CSS (which added unwanted classes to the buttons) was solved by incorporating the calculator into a web page with a separate controller and a separate layout file (`jscalc.html.erb`). One attraction of [jscalc](#) is that it accepts JavaScript functions, such as `Math.random()`, as input.

A version of [jscalc](#) in a basic Rails application is also available at <http://tgd-jscalc.herokuapp.com/>

Binary to Decimal Converter (`binToDec`)

This gem provides the functionality to convert a binary number to a decimal. If the argument provided is not binary, an exception is thrown. It has been unit-tested.

When the gem is installed, the following functionality becomes available.

```
Converter.binToDec(arg)
```

The program uses the following code to test if the input is binary.

```
arg1.to_s.split(//)
  .map { |i| i.to_i }
  .find_all { |value| value > 0 }
  .inject(:*)
```

If the test passes the input is converted to binary with the following function.

```
arg1.to_s
  .split(//)
  .map { |i| i.to_i }
  .inject(0) { |accumulator, value| (accumulator + value) * 2 }

return result/2
```

`binToDec` has been downloaded more than 750 times from [RubyGems](#).

Cost Price to Formatted Selling Price (`priceMarkUp`)

This gem allows the selling price to be calculated from the cost price based on a markup.

The default markup is 0.25.

In addition, the number of decimal places in the output may be specified. When the gem is installed, the following functionality becomes available.

```
Dowstore.priceMarkup(price, markup, decimal_places)
```

Examples of usage are as follows

```
Dowstore.priceMarkup(100) => "125"
```

```
Dowstore.priceMarkup(100, 2) => "300"
```

```
Dowstore.priceMarkup(100, 2, 3) => "300.000"
```

Extensive use of [priceMarkUp](#) has been made during the course of this work. It has been unit-tested.

3.4 Ajax

Two methods were used to incorporate Ajax requests.

The first method used the jQuery `$.ajax()` method to set up the XMLHttpRequest and is based on the method learned in lectures. It was used primarily to ‘Ajax-enable’ nav-bar links (so that when a user clicks on such a link the whole page is not renewed via a new HTTP request, but the relevant content is ‘injected’ into the current page using an Ajax request).

The second method uses the ‘Rails way’ as outlined, for example, in *Agile Web Development with Rails* (Ruby et al., 2012, pp 136 - 139).

Briefly the method used was as follows. Say a nav-bar contains a link to a page rendered from the index method of a controller, and it is desired to Ajax-enable that link. In the example below this is the index action of the `comments_controller`.

1. The `remote: true` parameter is passed to the link as shown:

```
<%= link_to "Comments", comments_path, remote: true %>
```

2. The line ‘`format.js`’ is added to the `index` method of the `comments_controller` so that it reads as follows.

```
respond_to do |format|
  format.html # index.html.erb
  format.js
  format.json { render json: @comments }
```

3. In views, a partial is made (in the same directory) of `index.html.erb` (called `_index.html.erb`)
4. Also in the same directory, a file called `index.js.erb` is created and jQuery is used to render the partial. Typically, the `.js.erb` file contained code something like the following (where `#yieldId` is the id of a div (in `application.html.erb`) containing the `<%= yield %>` command:

```
$('#yieldId').html('<%=j render partial: 'comments/index' %>');
```

The `j` is important, as it invokes the `j()` helper method which converts a Ruby string to a form compatible with JavaScript (Ruby et al., 2012, p 138).

This method was used quite a lot to add Ajax functionality to the application. As well as nav-bar links, it was used for example to Ajax-enable the addition of items to the shopping cart (so that when an item is placed in the cart by a user a separate HTTP request is not sent).

3.5 Append to and Sort an XML file when a New User Registers

Two functions, `builder()` and `myAdd()` (within a module called `TGD`), were written that generate relevant XML tags (`builder`), add content to those tags, append the populated tags to a well-formed XML file, and sort it alphabetically such that the updated file is also well-formed (`myAdd`).

When called from the `create` action of a relevant controller, these functions may be used to update and sort an XML file every time a new user registers.

An abridged version of the code is shown in [Fig. 2](#) The unabridged version may be found in `lib/xmlfunctions.rb` in the *skeptics* Rails application.

In the present work, `TGD.myAdd()` was called from the `create` action of the

registrations_controller, and public/xmlNewUsers.xml is updated when a new user registers.

Functionality provided by the [nokogiri](#) gem was incorporated into both functions to help generate and insert the XML tags, and to sort the XML file.

The XML file to be updated must exist, and must contain a root directory called `root`. The reason for this is that `myAdd()` removes the closing tag from the open XML file (`f.seek(-8, IO::SEEK_END)`) before appending the new contents **together with a new closing root tag**. The file need minimally contain `<root> </root>`, but of course it may also be populated with XML data and contain DTD meta-data in the Head section.

```
module TGD
  def self.builder (my_username, my_id, ...)
    Nokogiri::XML::Builder.new do |xml|
      xml.myUsers do |xml|
        xml.email my_username;
        xml.userID my_id;
        ...
      end
    end
  end
end

def self.myadd(email,my2id, ... filename)
#xml file must have a root-directory called root
  f = File.open(filename, "r+")
  f.seek(-8, IO::SEEK_END)
  f << builder(email,my2id ...)
  .to_xml(:save_with => Nokogiri::XML::Node::SaveOptions::NO_DECLARATION)
  f<< "</root>"
  f.close
  f = File.open(filename)
  doc = Nokogiri::XML(f) do |config|
    config.options = Nokogiri::XML::ParseOptions::NOBLANKS
  end
  node1 = doc.at_xpath('//root')
  sorted = node1.children.sort_by{ |n| n.text }
  sorted.each{ |n| node1 << n }
  file = File.open(filename,'w')
  file.puts doc.to_xml
  file.close
end
end
```

Figure 2: Append to and Sort an XML file (xmlfunctions.rb, abridged)

3.6 Use XMLHttpRequest to View an XML File Dynamically

In order to display an XML file on a web-page dynamically, a set of functions (`/app/assets/javascripts/tgdxmlHTTP.js`) were written which, at their core, use an XMLHttpRequest to retrieve data from an XML file. In addition, JavaScript was used to create a series of buttons that allow a user to dynamically browse the contents of the file. A user may advance one entry, go back one

Users XML File

Attribute	Value
Username	adam@gmail.com
User Id	75
Created At	27 Dec. 2013
Updated At	27 Dec. 2013
Sign-in Count	1
Current Sign At	27 Dec. 2013
Last Sign At	27 Dec. 2013
Current Sign-in IP	127.0.0.1
Last Sign-in IP	127.0.0.1

<<

<< - 5

<<

>>

+ 5 >>

>>

Figure 3: Dynamic XML File Functionality in a Web Page

entry, go forward or back five entries, to to the beginning of the file, or go to the end of the file (Fig. 3). In the present work, this functionality was used to allow browsing of the newUsers.xml file discussed in the previous section (Section 3.5). When a new user registers, the relevant data are added to /public/xmlNewUsers.xml, and the (alphabetically sorted) entry may immediately be viewed on a web page. This functionality has been restricted to a user with administration status.

3.7 Validation

To protect the database from invalid date entry, the Orders, Commodities and Discussion models were validated using Rails validation. For example:

```
validates :name,  
          :category,  
          :description,  
          :price,  
          :presence => true  
  
validates :price,  
          :numericality => { :greater_than_or_equal_to => 1 }
```

3.8 Mailers

The functionality provided by the [mail](#) gem was used to set up a mailer (mailers/order_notifier.rb) that sends a formatted email to a registered user when commodities are purchased via the Checkout page. The formatted email contains the price and quantity of each product.

In addition, the [Devise](#) mailer has been set up so that a user receives an email when the ‘Forgot Password?’ link is clicked on the log-in menu. Clicking on the link within the email brings the user to the [Devise](#) Change Password page, and allows the password to be changed.

For development purposes, the emails are ‘caught’ by [Mailcatcher](#) but it is easy to change the setting so that emails are sent to (say) gmail (three relatively simple changes in config/environment.rb).

[Mailcatcher](#) may be accessed on port 1080 (localhost:1080)

3.9 Unit Testing and Test-Driven Development

Unit tests were written to test the functionality of both the [binToDec](#) and [priceMarkup](#) gems.

For example (test_binToDec in the [jscale](#) gems directory)

```
# Test if an valid binary number given an incorrect value
def test_binary_false
  assert_not_equal 9,
    Converter.binToDec(1000)
end

# Test if an invalid binary number raises an exception
def test_non_binary_true

  assert_raise RuntimeError do
    Converter.binToDec(2222)
  end
end
```

In addition, [RSpec](#) was installed and some preliminary TDD tests were written. A mistake made here is that [RSpec](#) should have been installed at the beginning of the project, rather than towards the end.

An example of an [RSpec](#) test for the commodities_controller is the following (spec/controller-commodities_controller_spec.rb):

```
describe CommoditiesController do

  describe "GET 'index'" do
    it "returns http success" do
      get 'index'
      response.should be_success
    end

    it 'should have a non-blank body' do
      get 'index'
      response.body.should_not =~ /<body>\s*</body>/
    end
  end

  ...
end
```

3.10 Paperclip and Image Upload

[Paperclip](#), in conjunction with [ImageMagick](#), was used to provide two pieces of image-upload functionality.

Users may upload a image to their Profile page, or edit that image, by browsing from a Paperclip-provided link on the Devise Sign-Up and Edit-User pages. The uploaded image appears correctly formatted on the Profile page.

An administrator may upload or edit images on the Discussion pages. The uploaded image appears on the relevant Discussions Page and as a thumbnail on the Index page of the Discussions Model.

It was observed that [Paperclip](#) is a lot easier to use (almost flawless) on an Ubuntu OS, compared with Windows (where, in the experience of this user, it is very difficult to get to work properly).

3.11 Search

Search functionality was incorporated into the Commodities/index page, Comments/index page and Discussions/index page.

On the Commodities/index page a user may choose, by radio-button selection, whether to search by name, by category or by description, or by all of the above criteria (default).

3.12 Will-paginate

[Will-pagination](#) was used for pagination of tables and for pagination of search results on the Comments/index, Discussions/index and Commodities/index pages.

[Digg pagination](#) (CSS) was used for formatting. In the opinion of this user, [Will-pagination](#) is one of the very best gems.

3.13 Devise

The [Devise](#) gem was used for user authentication. An [administrator role](#) was added to the basic installation.

A user may log-in using an email address. The user is asked to confirm address.

An administrator may log in (and gain additional privileges).

A user may change her/his password via a link sent to the user's email account

A user may update her/his profile

A `registrations_controller` was created in order to allow data to be appended to an XML file when a new user registered (called from the `def/create` action of the `registrations_controller`).

A `users_controller` was added allow users to create a profile (`def/show`) and to add authentication functionality

3.14 Gravatars

The [Gravatar Image Tag](#) was installed a Ruby gem and was used to display the Gravatar of a registered user on the 'breadcrumbs' nav-bar. If a registered user does not have a Gravatar, a default image is displayed.

3.15 Before Filtering

The `before_filter` method was used to restrict access to controller actions.

For example, to allow only a registered user to access his/her Profile page but not the Profile page of any other registered user, and in addition to allow an administrator to access (via the nav-bar) the Profile page of any user, the following code was added to the `users_controller`.

```
before_filter :validate_user

def validate_user
  redirect_to root_path unless current_user
    and current_user.id.to_s == params[:id]
    or current_user.admin
end
```

4 Specific Implementation Components

4.1 Shopping Cart/Commodities Page

A lot of effort was put into integrating the Shopping Cart and Commodities Page and these will be taken together.

Only a registered user may purchase items. An unregistered user may, however, view and search the list of commodities

The shopping cart is only visible when a registered user adds an item to it. When the Cart is empty, a div with the message ‘Your Cart is empty’ is displayed. (This is done using `hidden_div_if` and `hidden_div_unless` methods in `application_helper.rb`).

A user may add items by clicking on the ‘Add to Cart’ button **or by clicking on the image of the product**.

The Shopping Cart (rendered as a partial) is visible on every page as long as it is not empty. This includes the Checkout page.

When a user adds an item to the Cart, the price of the item, the quantity and total price are immediately displayed in the Cart, and these are dynamically updated every time a user adds a product.

Adding a product to the cart, either by clicking on the button or the image, are ‘Ajax-enabled’. That is, only the Cart portion of the page is updated and **a separate HTTP request is NOT sent for every item purchased**.

When an item is added to the Cart, a green background fades in and then fades out in that line-item, indicating to the user that an item has been successfully added.

A user may delete individual items from the Cart (by hovering over the line item), or delete the whole Cart.

The Cart may be made smaller by clicking on the ‘Smaller’ button. When this is done, only the total price is displayed, and the user is given the options of going to Checkout or making the Cart larger again.

Making the Cart smaller is done with an Ajax request. When the ‘Smaller’ button is clicked a new partial is rendered showing a smaller cart. (Unfortunately, making the Cart larger has not been Ajax-enabled).

The Cart is draggable (done with jQuery-UI).

The ‘humanized’ name of the registered user appears at the top of the Cart

Only an administrator may Edit and Destroy items on the Commodities Page.

4.2 Check-out

When the user click the ‘Check Out’ button on the Cart, he/she is taken to the Checkout page and invited to fill in details such as first name, last name email address, credit card type. The user must also confirm the email address and Rails validation is used to ensure that the two emails match (`validates :email_confirmation` in the Orders Model).

When the User successfully fills in the form and clicks ‘Purchase’, he or she receives a formatted email message (send to the address supplied) showing details of the items purchased.

The Cart is emptied

The user is redirected to the Commodities Page and receives a message ‘Thank you for your order’.

4.3 Dynamic Users XML File

The dynamic display of the Users XML file has been discussed in [Section 3.6](#). Here it need only be added that this functionality is restricted to a user with administrator status.

4.4 Static Users XML File

By clicking on a link in on the Dynamic Users XML view page, an administrator may view (in a separate page) a formatted version of the ‘raw’ XML file(`xmlNewUsers.xml`). Formatting was done using an XML-transform (see `/public/xmlNewUsers.xsl`).

4.5 Profile Pages

When a user registers, a Profile page is automatically created.

A user may upload an image to his/her Profile Page

A user may edit his/her Profile page (including the image)

The Profile page displays the image uploaded by the user together with user details. For example:

```
E-mail  thomasgdowling@gmail.com
Human Name  Thomasgdowling
User Id  51
User Created At  25 Dec. 2013
User Updated At  01 Jan. 2014
Sign-in Count  28
Current Sign in On  01 Jan. 2014
Last Sign-in On  31 Dec. 2013
Current IP  127.0.0.1
Last Sign-in IP  127.0.0.1
```

Please Leave Your Comments

Comment	User Name	Time		
What about evolution? Is that theory falsifiable? Popper himself called it a metaphysical research program	thomasgdwling	2 days ago	Edit Comment	Destroy

Figure 4: An example of a User Comment

4.6 Discussion Pages and User Comments

Discussion Pages are created by an administrator, but a registered user may leave comments

Only a registered user may leave a comment

When a user leaves a comment, it is immediately visible on the relevant **Discussions page** (not the Comments page) and appears at the top of the paginated list. The length of time since the comment was posted is recorded (`time_ago_in_words(comment.created_at)`), as is the (humanized) username.

A user may, at any time, destroy or edit his or her comments, **but not the comments of another user**.

An administrator may edit and delete all comments (and may administer comments on a separate Comments page).

A paginated list of all Discussions are shown on a separate Index page.

An example of a user comment is shown in [Fig. 4](#)

4.7 Comments Page

A separate Comments page is provided to allow a user with administrator status to view all comments, to search the database of Comments, and to Edit or Delete any comment.

4.8 Authentication Logic

In order to show certain functionality to a user depending on the registration status of that user, extensive use was made of the following methods (provided by [Devise](#))

```
<% if current_user and current_user.admin %>
<% if current_user.present?%>
<% if current_user.admin? %>
```

For example, certain nav-bar items become visible when the user logs in and logs out. In addition, the visibility of 'Edit' and 'Destroy' buttons were selectively manipulated using the above methods.

5 GitHub and RubyGems

The code for the skeptics web application is available from GitHub at <https://github.com/tomGdow/skeptics>

The jscalculator gem is available at <https://rubygems.org/gems/jscalculator>

The binToDec gem is available at <https://rubygems.org/gems/binToDec>

The priceMarkup gem is available at <https://rubygems.org/gems/priceMarkup>

6 Browser

The application is best viewed in a Firefox browser.

7 Discussion

A dynamic, functional web-site for the fictional National Skeptics Association was successfully created using Ruby-on-Rails which incorporated most of the desired functionality. The following observations and criticisms may be made

Rails is a lot easier to use on an Ubuntu operating system, compared to Windows. This is particularly evident with the [Paperclip](#) gem, which works almost flawlessly with Ubuntu.

RSpec should have been installed earlier in order to avail of the advanced TDD functionality it offers.

An Index page showing a list of all users where an administrator can edit and delete user details was not implemented.

The [Devise](#) gem is an excellent method for providing authorization functionality.

There was a ‘clash’ between the [jscalculator](#) gem developed during the course of this work and [Zurb Foundation](#) CSS, mainly because Zurb was adding extra classes to the buttons, and was adding undesired formatting to the input box. There seems no easy way around this as it does not seem to be possible to protect a button from this type of class ‘injection’. However, this problem needs to be sorted, and the solution posted to RubyGems.

Will-pagination is an excellent gem

A method needs to be implemented that allows searching of the XML file from a web page.

In addition, a method needs to be implemented that allows ‘purging’ of all data except the root directory from the XML file. In this way, an administrator could record in an XML file the details of users who register in a given time period (and browse the XML data dynamically).

8 Declaration

I hereby certify that this material, which I now submit for assessment of the program of study leading to the award of Master of Science in Web Technologies is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Thomas Dowling (x12113441)

9 References

Ruby, S., Thomas, D. and Heinemeier Hasson, D. (2012) *Agile Web Development with Rails*. 4th edn The Pragmatic Programmers.