

SE31520: Enhancing the CS-Alumni Application

Department of Computer Science, University of Aberystwyth

Monday, December 4, 2012

Contents

Introduction	3
CSA Application	3
Technologies	3
Architecture & Design Diagrams	4
REST API	5
Changes & Rationale	5
RESTful Client	7
Technologies	7
Architecture & Design Diagrams	7
RESTful Interoperability	9
Rationale	10
Resultant System	10
Testing	12
Strategy	12
RESTful Client Testing	12
Results	13
Evaluation	13
Approach	13
Problems & Compromises	13
Areas of Expansion	14
Output & Assessment	14
Acknowledgements	15

Introduction

Many applications written today offer more than simply one interface to access their data. There are a multitude of ways in which systems communicate with customers and other systems directly, in previous years technologies such as SOAP have been prevalent, but with the aid of more mobile technologies, lighter-weight and easier to implement protocols have found favour, such as REST.

The Computer Science Alumni (CSA) application prototype, provided and authored by Chris Loftus, provides features relating to the broadcast of information relevant to Computer Science Alumni who have registered as users to the system in either the form of email, twitter or both. This information can be sent to two mailing lists, one for general information and one specifically for job news.

This document discusses the creation of a RESTful desktop client that may communicate with the CSA application entirely through the REST protocol and any necessary adjustments made to the provided CSA prototype in order to facilitate such transactions. The design decisions, compromises and implementations will be discussed, as will be an evaluation of the two resultant systems.

CSA Application

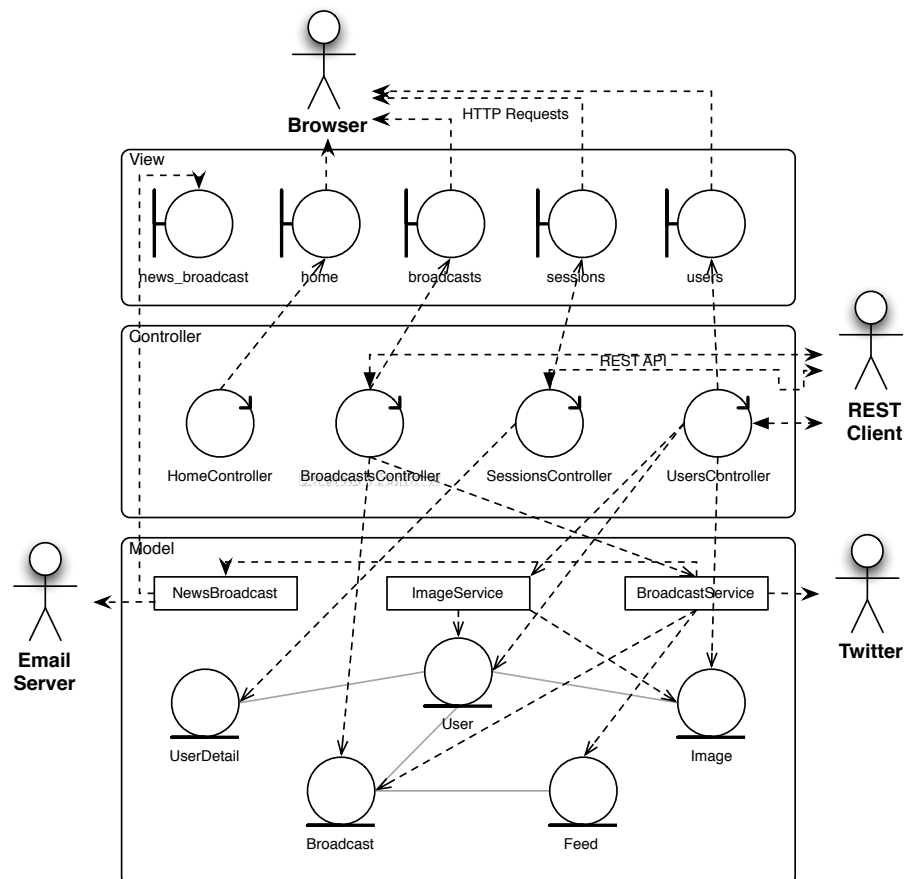
Technologies

The CSA application prototype is built upon Ruby on Rails (henceforth rails), no extra gems were required in either the development or production environments, further to those mandated by the original prototype. The existing gems present were utilised and are listed below with their purpose:

1. **sass-rails** - SASS compiler
2. **coffee-rails** - CoffeeScript compiler
3. **uglifier** - JavaScript obfuscation
4. **jquery-ui-rails** - jQuery UI bindings
5. **jquery-rails** - jQuery bindings
6. **simple-navigation** - Assists web-based front-ends
7. **will_paginate** - Assists a collection with paginating its results
8. **paperclip** - Storing file attachments to objects
9. **bartt-ssl_requirement** - Mandates SSL encryption
10. **mongrel** - Web Server
11. **oauth** - Authentication schema used by Twitter
12. **jbuilder** - Formats JSON objects into desired output

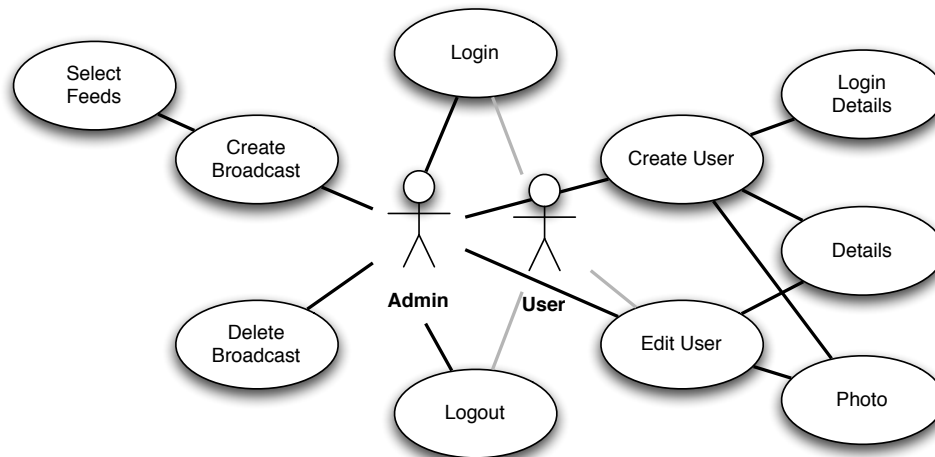
Architecture & Design Diagrams

The architecture of the system has not been altered fundamentally, instead it has been built upon to enable the REST client to communicate whilst having minimal impact on the existing web front-end provided. Due to the nature in which rails generates objects, a strong MVC structure is adhered to throughout. The separation of models and views and the delegatory nature of the controllers ensures strict adherence.



The primary difference is the introduction of the REST Client actor and its bi-directional data flow amongst the broadcast, session and user controllers. All other relations, use of existing services and methods have remained unaltered.

There are two specific roles to which the use cases accommodate, standard users and an administrator as defined below:



REST API

Compared to the provided prototype, the only inclusions to the API are the methods relating to the retrieval and assignment of user images via GET and POST. In other to refactor the pre-existing methods, they were modified to handle JSON input and output in all cases where not present, except in the new methods in controllers, which are specifically for HTML web forms.

```

/CompSci/Year3/SE31520_DIBA/Assignment/csa $ rake routes
search_users GET    /users/search(.:format)  users#search
users        GET    /users(.:format)         users#index
              POST   /users(.:format)         users#create
new_user     GET    /users/new(.:format)     users#new
edit_user    GET    /users/:id/edit(.:format) users#edit
user         GET    /users/:id(.:format)     users#show
              PUT    /users/:id(.:format)     users#update
              DELETE /users/:id(.:format)     users#destroy
broadcasts   GET    /broadcasts(.:format)    broadcasts#index
              POST   /broadcasts(.:format)    broadcasts#create
new_broadcast GET    /broadcasts/new(.:format) broadcasts#new
broadcast    GET    /broadcasts/:id(.:format) broadcasts#show
              DELETE /broadcasts/:id(.:format) broadcasts#destroy
session      POST   /session(.:format)       sessions#create
new_session  GET    /session/new(.:format)   sessions#new
              DELETE /session(.:format)       sessions#destroy
home         GET    /home(.:format)          home#index
              GET    /users/:id/photo(.:format) users#get_photo
              POST   /users/:id/photo(.:format) users#update
root         /
/CompSci/Year3/SE31520_DIBA/Assignment/csa $ █

```

This output of `rake routes` shows the conformity between the presented application and the original prototype.

Changes & Rationale

1. ActiveRecord Pagination

ActiveResource provides pagination of collections when requested, specifically the Broadcast and User controller indexes contained listings of all the entities contained within them. In order to replicate the functionality that the web front-end provides in the REST API, the parameter of page is required as a component of a REST request. From this parameter, the current offset is determined by the defined items per page (in the example of these controllers, it is hardcoded as 6).

The logic proceeds to ascertain whether, after the current subset of records, there are any more to retrieve later. If this is the case, the HTTP 'Link' header is set as the resource to which the next subset may be acquired. Once there are no more records to return, the HTTP 'Link' header is returned empty as a signal thereof.

The use of the 'Link' header in this is conventional[4] and can be seen in RESTful APIs such as those provided by GitHub and Facebook[5]. It is important to note that in those examples, the header has a `rel` attribute explicitly denoting the relation, or indeed a dictionary within the record set itself entitled `'link'` which contains an array of related meta and contextual data.

For the purposes of this server & client correspondence, a simpler mechanism as provided was deemed appropriate and specification satisfying, hence its inclusion.

2. REST Format

Throughout the prototype, all responders respond to only HTML or JSON, code relating to XML has been altered accordingly. This is so that the client may focus purely on interacting with the server using JSON, this representation was chosen due to its prevalence in modern internet based applications[?], the ease of which Ruby can serialise a dictionary object into a JSON object and due to the platform of choice (with discussions noted in support later in this document) for the RESTful client.

3. Format Specific Responses & HTTP Status Codes

In conjunction with deciding on the data format for exchanges with the client, the correct HTTP status codes for given tasks[7], including whether to return body content or not, were implemented throughout the controllers. By adhering to the specification as such, clients may confidently ascertain the status of their requests.

4. Session Controller

The SessionController has been modified such that it now handles successful and unsuccessful attempts at logging in when attempting authorisation from a client expecting a JSON (HTTP Content-Type `'application/json'`) response.

If successful, the authenticated user's ID is returned for the application creating the session to store along with a HTTP status code OK (200) to inform the client of the success. If unsuccessful, a suitable HTTP status code (401) is returned with no content[3].

Once authenticated, the SessionController is not interacted with again. Given the stateless nature of REST[3], maintaining server-side structures seems to be counter-intuitive. Certainly, gems and solutions do exist (such as `devise`) but, ultimately, appear to be far in excess of what is required in this application. A discussion of alternative methods is presented in **Areas of Expansion**.

5. Searching Users

The search method for users uses the gem `jBuilder` as an intermediary step between a request from a JSON representation and the HTTP response body. This proves to be an issue as it is not REST compliant. Instead, the value returned by this function is a list of strings which are HTML nodes to be injected into the DOM. As discussed later, moving this rendering to the client would be a better option in order to achieve strict adherence to the REST protocols[3]. In order to counteract this whilst maintaining the AJAX functionality, in lieu of refactoring the web front end a new parameter, `unformatted`, is looked for in the API request. If presented and its value is `true`, `jBuilder` is ignored and the raw JSON returned.

6. Broadcast Feeds

In order to send broadcast data to clients, the format output for XML was disregarded and JSON put in its place, further to this the default renderer for JSON objects appears to neglect references to other

models, specifically feeds. In order to work around this, the render is given the JSON representation of the broadcasts and specifically told to include feeds in the serialisation.

In doing so, the JSON data returned contains all the feed data related to, and embedded within, each broadcast.

7. User Images

A new method in the UsersController, `get_photo`, permits an authenticated administrator to request the image file linked to the User object directly. This path takes the form `GET: /users/:id/photo` where `:id` is the unique ID of the required user. This is done using the provided gem Paperclip, and the MIME-type returned is ascertained at the time of query execution given the file format stored for the user at the time.

If it is the case that the user does not have a custom image associated with their account, then the default placeholder image is returned instead.

Conversely, sending a POST, multi-part encoding request to the same path with an accompanying image file will use the existing `UserController::update` method to assign the new file as the user's new image, provided it is deemed valid by the Image model.

8. **Twitter API** Due to a recent change in the way in which Twitter's API handles authentication and calls, the OAuth endpoint URL was updated to `http://api.twitter.com` in `twitter.rb` and subsequent calls have an API version identifier. As Twitter integration is limited in this prototype to only posting statuses, the post path was updated from `/statuses/update.json` to `/1/statuses/update.json` (where 1 is the targeted API version) in `broadcast_service.rb`.

RESTful Client

Technologies

The client is written as a desktop application in Python 2.7, targeting Mac OS X specifically (tested only on 10.8). wxPython¹ is required to run the application from source, as it is the GUI toolkit used throughout the client. For managing HTTP requests and responses, the Python library `requests`² is also required in the Python environment.

Full instructions on where to obtain and install these prerequisites are noted in the accompanying `README.txt`

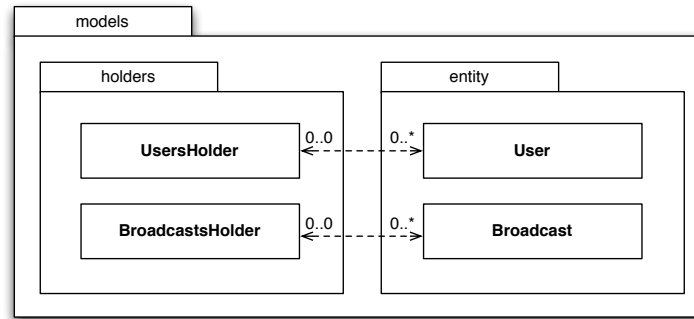
Architecture & Design Diagrams

The RESTful client gives access to all the functionality that the CSA application affords an administrator using the web front-end. It does not, however, cover functionality specifically of general users. Although it is the case that a general user may login and use the application, their experience will be limited due to lack of authorisation to receive data & perform tasks on the server.

MVC principles are used throughout the client with the three main packages `models`, `gui` & `remote` serving as the model, view & controller packages respectively. The modelling of objects locally has been kept deliberately lightweight due to the fact that the data of interest is modelled remotely. However, logic relating to the creation of new entities in the client is handled by their respective local models with which the remote package constructs the data to be sent to the server.

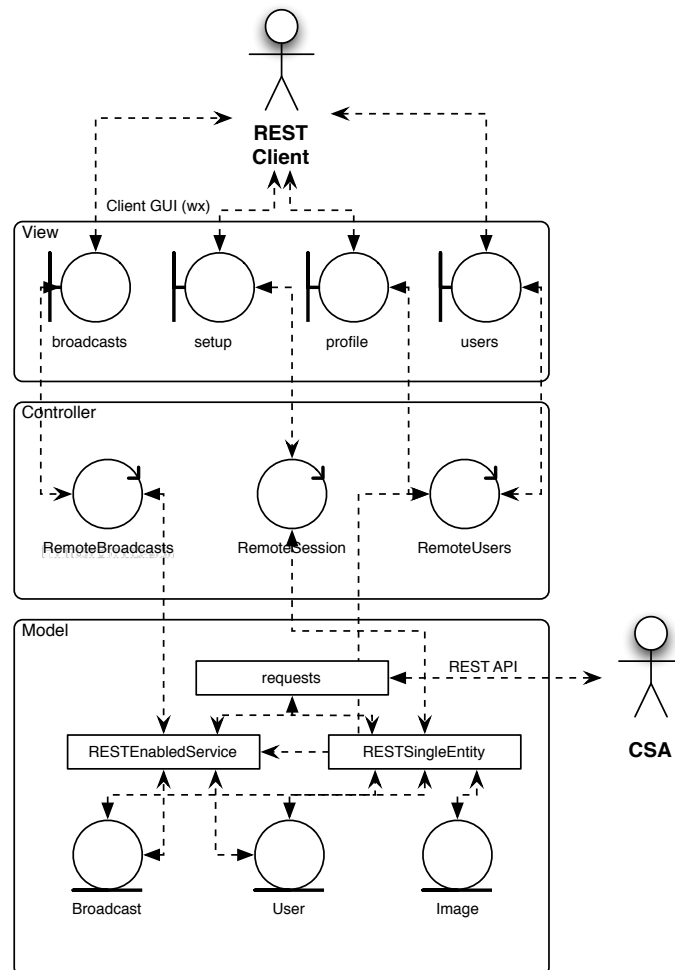
¹wxPython 2.9 - wx GUI bindings for Python

²requests - a Python library for managing HTTP requests in place of `urllib2`, which it itself uses internally.



The `UsersHolder` and `BroadcastsHolder` objects are those which are interacted with directly in the application, building the resources for the views and having the ability to delegate functions such as deletion and creation to the models and their controllers directly. They are the basis for both the `UserList` and `BroadcastList` GUIs.

`app.context.ContextHolder` is a single, static object within the scope of the application instance, it is propagated upon a successful login attempt and provides all requests with the same `HTTPBasicAuth` object for authorisation and contextual information about the current user.



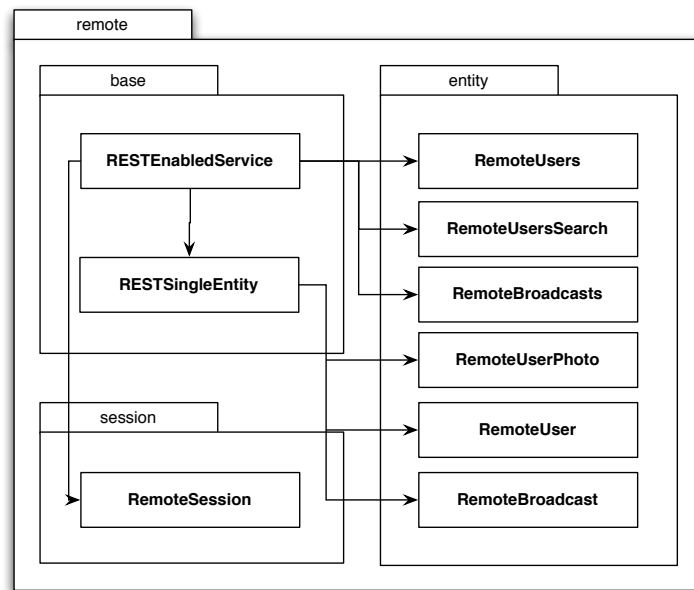
RESTful Interoperability

1. REST Format

The client uses JSON exclusively for RESTful transactions with the CSA application, this is preferable when working with Python due to the strong similarity between the JSON structure and the native Python dictionary (`dict`), and the native JSON libraries of the platform.

2. REST Services & Entities

The client is able to communicate with the CSA application through the use of entities and collections which are defined in the `remote` package. Two classes, `RESTEnabledService` and `RESTSingleEntity`, reside in `remote.base` and serve to be the basis of modelling any remote entity whose state is obtainable through the CSA REST API.



This hierarchy demonstrates the simple inheritance model used and how server entities are have CRUD operations applied to them.

3. CRUD

These operations are tied to all remote entities through the `RESTEnabledService` class which, through `requests`, is able to determine the appropriate HTTP method, assign data, files and authorisation automatically. The result of this is that entities inheriting from the service all obtain the ability to create, read, update and delete resources.

4. Security & Authentication

All authentication is done through HTTP basic authentication. The library around which the remote requests are modelled, `requests`, provides native support for such authentication through the `requests.auth.HTTPBasicAuth` class. See **Problems** later in the document for a discussion on SSL & encryption.

5. Resource Pagination

The client checks any incoming remote entity which inherits from `RESTEnabledService` for the aforementioned HTTP Link header, if present it automatically requests the next page and appends the results to its internal structure until the Link header does not point to a further resource.

Rationale

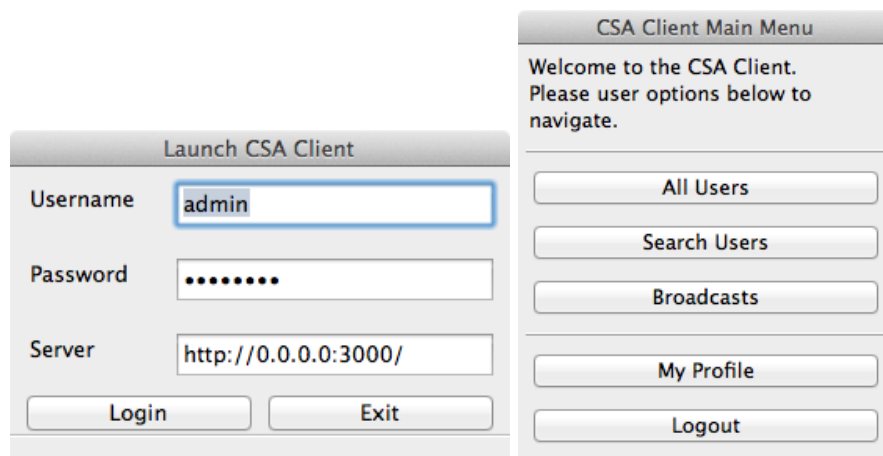
Python was chosen as the platform to be targeted due to the speed in which a prototype may be created, the wxPython package provides near-native performance. Both Python (≥ 2.7) and wx are available on a multitude of platforms, the only other dependency being a platform-agnostic Python library (`requests`), therefore with the correct environment set up, application portability is easily achieved.

A key consideration was the simplicity of the implementation and not binding it to local definitions of models to the highest extent possible, as such the User and Broadcast models are essentially wrappers around a JSON which extended logic for construction and formatting. This allows a high degree of malleability, should the server API change in the future.

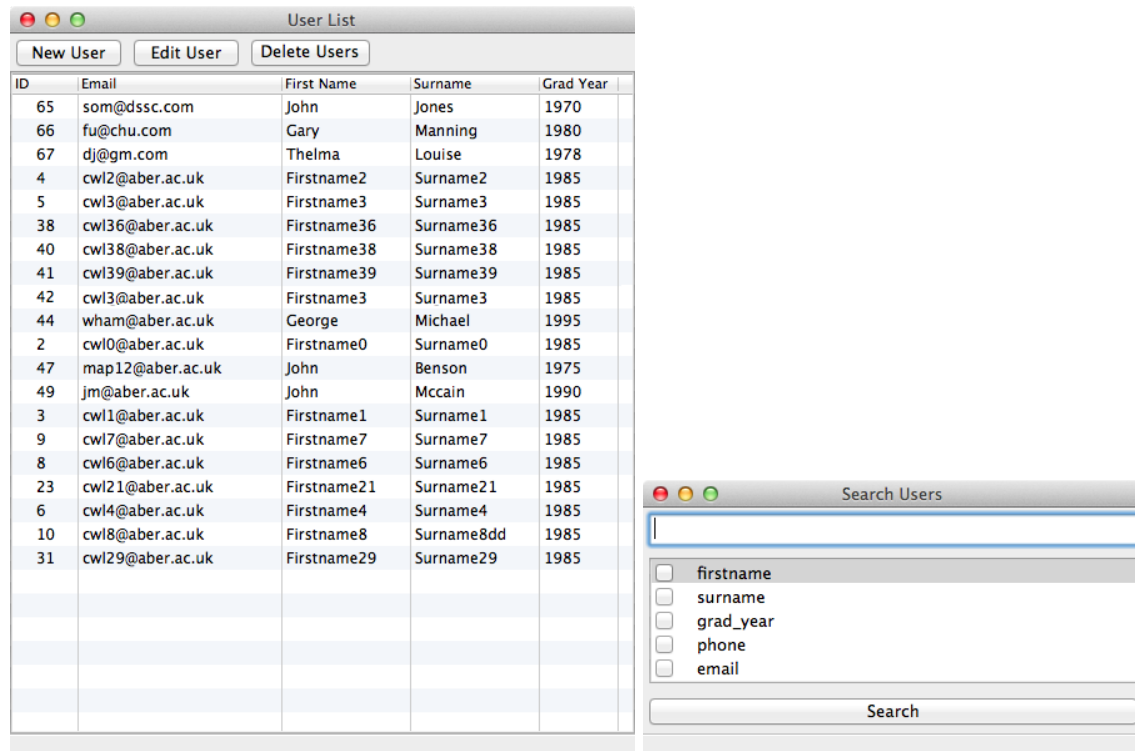
Resultant System

The resultant system performs all the tasks available to an administrator using the native CSA web interface, it correctly does so without jeopardising or changing the core functionality of said application also. No features are missing, and all have been implemented using REST principles.

This section contains screenshots from the resultant system running, which may also be viewed in the accompanying screencast.



User login prompt with pre-populated values of username, password and CSA server address. Once validated, the main menu is presented with the application functionality exposed.

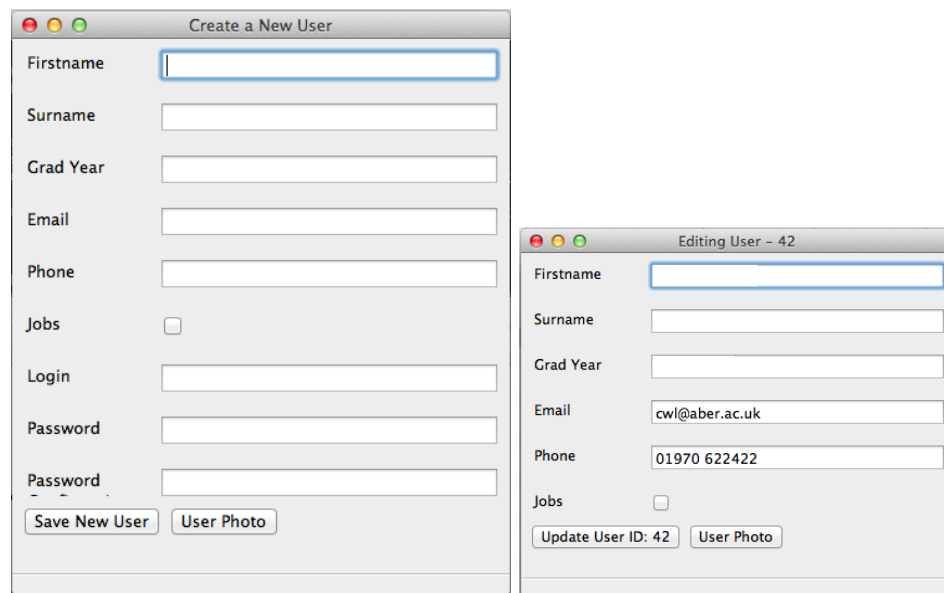


The image shows two windows from the CS-Alumni Application. The 'User List' window displays a table of users with columns for ID, Email, First Name, Surname, and Grad Year. The 'Search Users' window allows filtering by various fields.

ID	Email	First Name	Surname	Grad Year
65	som@dssc.com	John	Jones	1970
66	fu@chu.com	Gary	Manning	1980
67	dj@gm.com	Thelma	Louise	1978
4	cwl2@aber.ac.uk	Firstname2	Surname2	1985
5	cwl3@aber.ac.uk	Firstname3	Surname3	1985
38	cwl36@aber.ac.uk	Firstname36	Surname36	1985
40	cwl38@aber.ac.uk	Firstname38	Surname38	1985
41	cwl39@aber.ac.uk	Firstname39	Surname39	1985
42	cwl3@aber.ac.uk	Firstname3	Surname3	1985
44	wham@aber.ac.uk	George	Michael	1995
2	cwl0@aber.ac.uk	Firstname0	Surname0	1985
47	map12@aber.ac.uk	John	Benson	1975
49	jm@aber.ac.uk	John	Mccain	1990
3	cwl1@aber.ac.uk	Firstname1	Surname1	1985
9	cwl7@aber.ac.uk	Firstname7	Surname7	1985
8	cwl6@aber.ac.uk	Firstname6	Surname6	1985
23	cwl21@aber.ac.uk	Firstname21	Surname21	1985
6	cwl4@aber.ac.uk	Firstname4	Surname4	1985
10	cwl8@aber.ac.uk	Firstname8	Surname8dd	1985
31	cwl29@aber.ac.uk	Firstname29	Surname29	1985

The 'Search Users' window has a search bar and checkboxes for the following fields: ☐ firstname, ☐ surname, ☐ grad_year, ☐ phone, and ☐ email. A 'Search' button is at the bottom.

Navigating users can be performed by viewing all or searching for a specific subset.



The image shows two windows for user management. The 'Create a New User' window has fields for Firstname, Surname, Grad Year, Email, Phone, Jobs (checkbox), Login, Password, and Password (confirm). The 'Editing User - 42' window shows the same fields pre-filled with data for user ID 42.

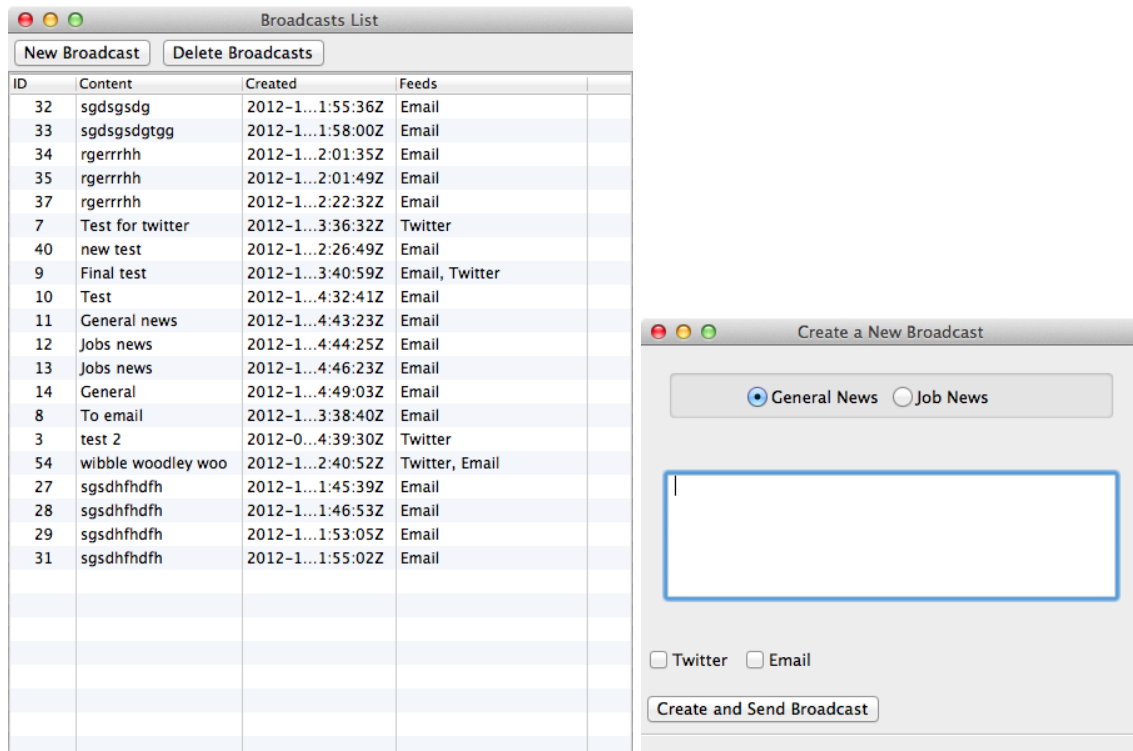
Create a New User

Firstname:
Surname:
Grad Year:
Email:
Phone:
Jobs: ☐
Login:
Password:
Password:
Buttons: Save New User, User Photo

Editing User - 42

Firstname:
Surname:
Grad Year:
Email: cwl@aber.ac.uk
Phone: 01970 622422
Jobs: ☐
Buttons: Update User ID: 42, User Photo

New user creation and the editing of an existing user.



The list of saved broadcasts and a dialog to create a new one.

Testing

Strategy

The testing strategy includes the formation of unit and functional tests on the CSA application, with real world use of the RESTful client being portrayed in the accompanying screencast.

CSA application testing will focus around unit testing application models using rails' ActiveSupport::TestCase and controllers using ActionController::TestCase. The scope of the tests has been determined through course materials and books such as Programming Ruby 1.9[1].

This testing aims to establish whether the behaviours encountered during prototyping and use may be formalised, repeated at will and provide a basis for future regression testing.

The majority of testing was performed manually, as per the methodology chosen, and was focused on the interworking between client and server. Evidence of this is presented as the successful use of the system within the screencast.

RESTful Client Testing

Limited unit or functional testing was performed on the client, instead as the methodology followed was akin to prototype-driven-development, suitability and robustness was tested iteratively whilst implementing features.

Results

The ongoing testing and interactive debugging helped significantly in the refinement of specifications relating to the API and the suitability of models in both the CSA application and the client. Results of the unit and functional tests accompanies as an appendix to this document.

Evaluation

Approach

Initially, much reading was conducted around REST including most prominently REST API Design Rulebook[3] and building rails applications[1, 2].

Further to this, investigations into languages and suitability eventually led to Python due to its nature for fast prototyping and similarities with Ruby in modern web application development. From this point, research into GUI toolkits and learning their relative APIs was also conducted.

The CSA application was developed in RubyMine and the client in Sublime Text 2 simultaneously based on the designs, using an iterative feature-based prototyping development methodology.

Problems & Compromises

1. REST Pagination

Pragmatically speaking, although this implementation herein works well in this example, it is tied directly to the small size of both the pagination subset and the entire dataset size. Such an automatic approach would not be suitable on large incoming datasets and, furthermore, the server API should present an option to return all data in one request via HTTP streaming. However, based on the size of this dataset and the resources of the server, this may not be a wise choice. Given the theoretical nature of these issues within this context, the pagination implementation is sufficient.

2. WEBrick and SSL

The assignment specification[?] does not mandate nor mention the use of SSL in the prototype and, as such, the exercise was purely academic and exploratory, thus the screencast demonstration uses authentication but no encryption. The exercise was successful to get apache through MAMP to redirect data through web browsers, however `requests` would not communicate with the server through the mechanism, claiming there to be something spurious about the target server. As such, this functionality was removed (a specific parameter, *verify*, in the `requests` methods).

3. requests vs. urllib2

Originally, the client worked solely through a bespoke library wrapping around `urllib2`, Python's library for dealing with HTTP requests and responses. I encountered issues whereby things were taking far too much effort and obfuscated logic to achieve (HTTP headers, binary data) and found `requests` to be "for humans"[8]. Using this as the basis for the client services saved a lot of time.

4. Client Testing

Given the nature of the methodology used and a focus on iterative feature and prototype introduction on the author's part, there is a distinct lack of formal, reusable testing harnesses available for the client. In hindsight, this should have been more of a priority in lieu of manually testing through `python`³ and debuggers such as `pudb`⁴.

³iPython - Interactive Python Shell - <http://ipython.org>

⁴PUDB - Python port of udb - <http://pypi.python.org/pypi/pudb>

Areas of Expansion

The security mechanism employed in both the CSA prototype and the client is, as previously discussed, simply HTTP basic authorisation. Although sufficient for the purposes of this exercise, there are more secure methods for performing the authorisation. One example of which would be the handling of roles per user within the CSA application itself and, based on these roles, the allocation of unique API keys to authorised users.

Using this key, both the server and client could use the same algorithm (for example, SHA1 of the query string including the API key) to construct a secret API key to append onto the query which both the server and client could validate upon. There are notable security and performance benefits to such an approach.

A further area of scrutiny outside the scope of this exercise would be the way in the MVC model is violated with respect to controller and view separation. The aforementioned search users function responds to a request for JSON formatting through generating HTML via the jBuilder gem to be implanted into the DOM. Ideally, this should simply return a JSON object, avoiding jBuilder entirely, and have the view rendered client-side in some suitable technology such as Backbone.js⁵. The solution of the parameter unformatted accomplishes the goal of enabling the client through CSA refactoring, but is not a particularly RESTful attribute. The underlying issue here is poor original design as the refactoring required to alleviate the issue is relatively large.

Output & Assessment

Overall, the aim of the project was realised successfully through the production of a usable, stable and robust client with no features omitted. A higher emphasis on testing would have been more appropriate as the general approach was manual testing through debuggers on client and server, this is reflected in the simplicity of the tests provided, even if broad in scope.

For a server application, rails makes a lot of sense. Its direction towards REST services and the ease with which such services may be constructed make it very suitable for such a task. Learning rails, however, is not without its difficulties. Certainly, it seems that a lot of the functionality and file / module include appears by 'magic', that is to say that there is no clear reason why any given helper is available anywhere else, nor how a method from somewhere else is apparently globally defined anywhere. This is, of course, down to the model with which rails constructs its environment and is very simple to use, if you can think high-level enough to do so. The truth for some would be that it is unintuitive when trying to introduce your own components into such an environment. This is clearly the reason that the rails generate tools exist. Programmatically speaking, it is unclear how good a decision this is, when the use of a platform almost mandates the use of a suitable IDE and such generation routines, one must question the performance impact and underlying structure which such tools mask.

Python development was a very different affair, using only a text editor and API documentation, a clean, small and usable application was developed. The technological choices for this client are deemed to be appropriate as their intended benefits were fully utilised in the clients' construction and performed exactly as planned.

Ideally, more emphasis on the server component would have been beneficial in order to create a truly RESTful API, as noted in **Problems**, however the design and design goals of minimal impact to the existing system were duly met with all existing functionality remaining in place.

⁵Backbone.js - <http://backbonejs.org>

Acknowledgements

1. <http://guides.rubyonrails.org/testing.html> - Ruby on Rails Testing Guide
2. http://guides.rubyonrails.org/layouts_and_rendering.html - Building header-only responses
3. <http://garrickvanburen.com/archive/passing-sessions-and-referers-in-rails-functional-tests/> - Passing session data to tests
4. <http://www.blog.pythonlibrary.org/2008/05/18/a-wxpython-sizers-tutorial/> - Introduction to wxPython
5. <http://lanyrd.com/2011/frozen-rails/sfgzx/> - Implementing a RESTful API with Ruby
6. <http://accentuate.me/blog/?p=12> - The Exhausting Guide to The Rails Router
7. <http://www.houseofding.com/2008/11/generate-a-self-signed-ssl-certificate-for-local-development-on-a-mac/> - generating SSL certificates on a Mac

References

- [1] THOMAS, DAVE Programming Ruby 1.9, The Pragmatic Programmers' Guide. 2010, *The Pragmatic Programmers, LLC*.
- [2] CARNEIRO, CLOVES AND AL BARAZI JR, RIDA Beginning Rails 3. 2010, *Apress*
- [3] MASSE, MARK REST API Design Rulebook. 2011, *O'Reilly*
- [4] KLABNIK, STEVE *Some People Understand REST and HTTP* - <http://blog.steveklabnik.com/posts/2011-08-07-some-people-understand-rest-and-http> .
- [5] GITHUB API GitHub API Documentation - <http://developer.github.com/v3/>.
- [6] W3C W3C Linked Resources - <http://www.w3.org/Protocols/9707-link-header.html>
- [7] W3C W3C HTTP Status Codes - <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
- [8] REITZ, KENNETH Requests API Documentation - <http://docs.python-requests.org/en/latest/api/>
- [9] CALCOTE, JOHN RESTful Authentication - <http://jcalcote.wordpress.com/2009/08/10/restful-authentication/>

UNIT TESTS: PASSED WITHOUT ISSUE

```
csa — bash — 80x24
/CompSci/Year3/SE31520_DIBA/Assignment/csa $ rake test:units
Run options:
# Running tests:
.....
Finished tests in 1.562982s, 12.1563 tests/s, 18.5543 assertions/s.
19 tests, 29 assertions, 0 failures, 0 errors, 0 skips
/CompSci/Year3/SE31520_DIBA/Assignment/csa $
```

FUNCTIONAL TESTS: IMPLEMENTATION UNFINISHED, FEW PASSES (TIME CONSTRAINTS)

```
csa — bash — 172x84
FFFFFFFFF
Finished tests in 0.395798s, 27.7920 tests/s, 20.2123 assertions/s.
1) Failure:
test_should_create_broadcast(BroadcastsControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/broadcasts_controller_test.rb:21]:
"Broadcast.count" didn't change by 1.
<2> expected but was
<1>.
2) Failure:
test_should_get_broadcast(BroadcastsControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/broadcasts_controller_test.rb:17]:
Expected response to be a <:success>, but was <302>
3) Error:
test_should_get_broadcasts(BroadcastsControllerTest):
NoMethodError: undefined method `asset_response' for #<BroadcastsControllerTest:0x007f9f485e1688>
/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/broadcasts_controller_test.rb:12:in `block in <class:BroadcastsControllerTest>'
4) Error:
test_send_news(NewsBroadcastTest):
ArgumentError: wrong number of arguments (0 for 3)
/CompSci/Year3/SE31520_DIBA/Assignment/csa/app/mailers/news_broadcast.rb:2:in `send_news'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/abstract_controller/base.rb:167:in `process_action'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/abstract_controller/base.rb:121:in `process'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/abstract_controller/rendering.rb:45:in `render'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_mailer/base.rb:457:in `process'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_mailer/base.rb:452:in `initialize'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_mailer/base.rb:439:in `new'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_mailer/base.rb:439:in `method_missing'
/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/news_broadcast_test.rb:6:in `block in <class:NewsBroadcastTest>'
5) Failure:
test_should_create_user(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:20]:
"User.count" didn't change by 1.
<3> expected but was
<2>.
6) Failure:
test_should_destroy_user(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:43]:
"User.count" didn't change by -2.
<8> expected but was
<2>.
7) Failure:
test_should_get_edit(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:34]:
Expected response to be a <:success>, but was <302>
8) Failure:
test_should_get_index(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:10]:
Expected response to be a <:success>, but was <302>
9) Failure:
test_should_get_new(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:16]:
Expected response to be a <:success>, but was <302>
10) Failure:
test_should_show_user(UsersControllerTest) [ /Dropbox/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:29]:
Expected response to be a <:success>, but was <302>
11) Error:
test_should_update_user(UsersControllerTest):
ActionController::RoutingError: No route matches {:action=>"show", :controller=>"users", :id=>nil}
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:532:in `raise_routing_error'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:528:in `rescue in generate'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:520:in `generate'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:561:in `generate'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:586:in `url_for'
/rvm/gems/ruby-1.9.3-p194/gems/bartt-ssl_requirement-1.4.2/lib/url_for.rb:44:in `url_for_with_non_ssl_host'
/rvm/gems/ruby-1.9.3-p194/gems/bartt-ssl_requirement-1.4.2/lib/url_for.rb:33:in `url_for_with_secure_option'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/url_for.rb:148:in `url_for'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/routing/route_set.rb:213:in `user_path'
/rvm/gems/ruby-1.9.3-p194/gems/actionpack-3.2.8/lib/action_dispatch/testing/assertions/routing.rb:174:in `method_missing'
/CompSci/Year3/SE31520_DIBA/Assignment/csa/test/functional/users_controller_test.rb:39:in `block in <class:UsersControllerTest>'
11 tests, 8 assertions, 8 failures, 3 errors, 0 skips
rake aborted!
Command failed with status (11): [ /rvm/rubies/ruby-1.9.3-p194/bin/ ]
Tasks: TOP => test:functionals
(See full trace by running task with --trace)
/CompSci/Year3/SE31520_DIBA/Assignment/csa $
```