

UNIVERSITY OF HERTFORDSHIRE
Faculty of Science, Technology and the Creative Arts

Modular BSc Honours in Information Technology

6COM0285 – Information Technology Project

Final Report
April 2012

SKYSCRAPER CUSTOMER MANAGEMENT

J.C.D. FEIST

Supervised by: Maria Schilstra

Abstract

Skyscraper is an online customer management service for small to medium sized businesses who need a quick, simple and easy way to maintain customer details and interaction history. Many customer relationship management (CRM) solutions available today are overly complex in nature; usually requiring software installation and integration over a network. Skyscraper aims to solve the problems associated with complex CRMs by offering a lightweight online web service alternative, that is quick to register and easy to use.

Skyscraper was created with the Ruby on Rails web framework, and was successful in the majority of objectives it set out to achieve.

Acknowledgements

Before starting the project I thought it would be vitally important for me to become familiar with Ruby on Rails in order to have enough understanding to properly dig into the application. To that end, I decided to use the book “*Agile Web Development With Ruby on Rails*” with its tutorial ecommerce application to get to grips with the framework. In doing so, I was able to learn about the basics of Rails, including authentication, AJAX, ATOM feeds, pagination, limiting access, testing, and so on. The end product of this tutorial application that I created is hosted on Github:

<https://github.com/defaye/depot>

On Github you may find the the source code is viewable, and the entire version history can be tracked through commits.

<u>1. Introduction</u>	1
<u>1.1 The Purpose of the Project</u>	1
<u>1.2 The Objectives</u>	1
<u>1.3 The Agile Web Development Methodology</u>	2
<u>1.4 The Ruby on Rails Framework</u>	3
<u>1.5 Report Structure</u>	4
<u>2. Getting Started</u>	5
<u>2.1 Setting Up the Development Environment</u>	5
<u>2.2 Version Control and Back-Up Strategy</u>	6
<u>2.3 Designing the Application</u>	9
<u>3. Developing the Application</u>	11
<u>3.1 Creating the Layout</u>	11
<u>3.2 Authentication</u>	17
<u>3.3 Contacts</u>	20
<u>3.4 Deploying the Early Prototype</u>	26
<u>3.5 Creating a Search for Contacts</u>	28
<u>3.6 Notes</u>	32
<u>3.7 Sub-Users</u>	36
<u>3.8 Drop-Down Menu for User Accounts</u>	41
<u>3.9 Recently Viewed and Activity History</u>	43
<u>3.10 Parties</u>	49
<u>3.11 Reviewing the Prototype</u>	56
<u>4. Conclusion</u>	69
<u>4.1 Discussion & Evaluation</u>	69
<u>4.2 Future Work</u>	73
<u>References</u>	74
<u>Bibliography</u>	76
<u>Appendices</u>	78

1. Introduction

1.1 The Purpose of the Project

The purpose of the application - (named *Skyscraper*) is to save details and correspondence of customers that is easy to use and easy to accommodate into the workflow of a business, serving as a lightweight customer relations management system to maintain information on customers. The inspiration for *Skyscraper* comes from existing solutions like Highrise¹, and Salesforce², which address the same problem in the same manner - a web service as opposed to software that is installed. It is in standing on the shoulders of giants, that I set out to create another simple CRM solution.

1.2 The Objectives

1.2.1 Objectives

The Primary objectives of the project are to:

- Create a Customer Relationship Management (CRM) System that:
 - Stores information on each users' own customers
 - Is quick and easy to search on customers on a range of criteria
 - Can store additional information on customer history and interaction
 - Can import and export customers in a variety of ways
 - Has customisable characteristics on what can be stored

The Secondary objectives which would enhance the functionality of the Web Application:

- Styling of the Web Site for aesthetic appeal
- To create an Application Programming Interface (API) that:
 - Can be queried to serve as an alternative to using the Web User Interface (UI)

¹ <http://highrisehq.com> - Highrise, by 37 signals, a contending CRM solution

² <http://salesforce.com> - official website of one of the world's leading CRM solutions

1.2.2 Preordained Primary Specification Features to Implement

- Account (i.e. Business) Registration
- Individual Users of Account (i.e. Users associated with Business Account)
- User profiles associated with Account (to be customisable)
- Client Portal page including:
 - Activity history on the Account (with all associated users)
- Create, Read, Update, Delete (CRUD) functions on:
 - Customers or Parties (i.e. Businesses) (will be collectively referred to as entities)
 - Users associated with Account
 - Account details itself
- To Show: all entities or individual entities (on own page)
- Search function for an individual entity
- Ability to add notes to an entity
- Can add custom properties to an entity type (outside of standard details)
- Function to import & export entities
- Administration back-end for the service

1.2.3 Secondary Specification Features to Implement

- Styling of the Web Site for aesthetic appeal with Cascading Style Sheets
- Implementation of an Application Programming Interface (API)
 - Implementation of an example Web Service Client to query the API

1.3 The Agile Web Development Methodology

The design methodology that my project will follow will be in accordance with agile web development in the spirit of the Agile Manifesto. (Beck, Kent; et al, 2009)

Based on iterative and incremental development, agile web development stresses a strategy which relies on the flexibility with which to respond to change, tending to have cycles of planning, design, implementation, testing and evaluation, rather than following any atomic order. (A. Cockburn, Unraveling incremental development, 1993) Cockburn argues that, incremental development is a key ingredient of the success of a project.

One of the co-signatories of the Agile Manifesto, states (in relation to software design) all of the effort is in the design, which requires creativity and intellect. He goes on to state that creative processes are not easily planned, and so predictability of an overall plan is an impossible target. He concludes that the key to controlling an unpredictable process is in iteration. (Fowler, 2005a)

At the heart of iteration and the key to tackling an unpredictable specification therefore is to frequently produce working versions of the final system that have a subset of the required features. (Fowler, 2005b)

The earlier it is discovered that a change must be made, the less expensive it will be to implement as there is less code to re-work; a great strength of the Agile Web Development process.

Choices that will be made in the planning and development of the project will be taking the principles of the agile web development to heart. Although there will not be a relationship involved with any particular end user or customer, this will become a self-reflective process of requirements elicitation in the same regard.

1.4 The Ruby on Rails Framework

The choosing of a programming language is therefore crucial to the success of this project, as it is one of the design decisions that cannot be changed later on in development. I have chosen to use the Ruby programming language on the Rails framework for a few reasons:

1. Ruby on Rails is used to build interactive and engaging web applications and Ruby is considered to be exceptionally popular amongst programming languages on the web. (Softweb Solutions, 2010)
2. Ruby on Rails reduces development time by using convention over configuration, increasing the simplicity of coding for dynamic websites by having established methods of doing things, such that you can substantially cut down the amount of configuration required to set up repeated tasks. (Ladd; et al, 2006)
3. Ruby on Rails follows the *Don't Repeat Yourself* (DRY) principle - that every piece of knowledge in a system should be expressed in just one place. (Ruby; et al, 2011)
4. The Rails framework is considered to be a leader in the implementation of the Model View Controller architectural pattern (by decoupling models and views, reduces the complexity in architectural design and increases the flexibility and maintainability of code by helping to divide the whole application into manageable layers.

1.5 Report Structure

The main chapters are divided by the Introduction (giving an overview of the project); Getting Started (all about technology choices, development environment setup, including the initial design phase); Developing the Application (the main bulk of the report detailing the tasks involved in building Skyscraper, and the Conclusion (bringing an overall closure to the report and the project as a whole).

Throughout the report I have used footnotes to clarify things that have needed clarification without pulling away from the current context. I have also used footnotes for making references to documentation online or to a website of particular interest. I have decided not to consider these types of source to be listed as an official reference for these reasons.

I have also shied away from referencing the appendices as I would prefer to encourage the reader to cross-reference from the digital copy from which I have cited resource paths on many occasions in the footnotes to follow, to find the source code in context. In addition, the reader is encouraged to visit sources listed in footnotes if particularly emphasised.

2. Getting Started

2.1 Setting Up the Development Environment

Setting up the development environment meant preparing my computer to be Ruby on Rails ready and able to do the job well. This meant having a system that is fast at running code in Ruby.

2.1.1 Linux

I chose to use Linux (on Ubuntu 11.10) to be the development operating system as Windows is known to have performance issues with Ruby (Pack, *Installing Ruby*, 2010). In light of this it became important to re-configure my personal computer to dual-boot Ubuntu alongside Windows.

2.1.2 Ruby Version Manager (RVM)

Why Use RVM

Setting up Rails is highly recommended through RVM as it well supported and maintained. (Wayne E. Seguin, *Ruby Version Manager (RVM)*, 2011) Alternative methods of installing Ruby and Rails are often outdated and are prone to having problems - such as installing through apt-get, “the command-line tool for handling packages” (Moura, *apt-get*, 2010). By using RVM projects can use different versions of Ruby and different versions of Rails without any conflicts.

Using RVM to Setup Ruby on Rails

Setting up Rails involved installing RVM from apt-get (details on setting up RVM can be obtained from a good tutorial online) (Bigg, *Ubuntu, Ruby, RVM, Rails and You*, 2010). Then installing a version of Ruby through RVM, setting it as the default Ruby interpreter, and finally installing Rails with the following commands into bash “the shell, or command language interpreter, for the GNU operating system”. (Bourne, *What is Bash?*, 2009):

```
rvm install 1.9.2.  
rvm --default use 1.9.2  
gem install rails -v 3.2.1
```

2.2 Version Control and Back-Up Strategy

2.2.1 Why

“No cautious, creative person starts a project nowadays without a back-up strategy. Because data is ephemeral and can be lost easily—through an errant code change or a

catastrophic disk crash, say—it is wise to maintain a living archive of all work.” (Loeliger, 2009, p.1) Indeed, it would be crazy not to use a back-up strategy, and this is why I have chosen to use one that both fits Rails development and has proven to be effective at its job.

2.2.2 Git

Why Use Git

Git is a particularly powerful, low overhead version control system that runs locally by default. A popular alternative such as sub-version (otherwise known as SVN) requires a central “data repository to store the information and all its history” (Collins-Sussman; et al. *Version Control with Subversion*, 2011) for almost every operation, which can slow down development.

Git was invented by Linus Torvalds to support the development of the Linux Kernel, but has since proven valuable as a mainstream version control system.

Setting Up Git

Git can be installed through bash by issuing the command

```
sudo apt-get install git-core
```

Git can then be initialised in any directory using `git init`. It is then wise to set the author name and email address as a global setting, so this persists in all initialisations of a local git repository with the commands:

```
git config --global user.name "Jonathan Feist"  
git config --global user.email j.feist1@herts.ac.uk
```

Using Git

Throughout the course of development, at regular meaningful intervals, git will be used to make a commit of the current state of the application. This will take a meaningful message for the name of the commit as well as preserving the history and integrity of the source tree.

The following commands will generate a fresh rails application named *skyscraper* within the directory `~/www/skyscraper`, where we will initialise git, add the all files and folders within to be staged for the first commit and finally issuing the first commit with the message “Rails application” like so:

```
cd ~
mkdir www
cd www
rails new skyscraper
git init
git add .
```

The command `git status` can be issued here to see which files and folders are currently staged for commit, and finally:

```
git commit -m “Rails application”
```

Once done, we can review our handiwork with the command `git log`, to review the last commit, and `git status` to see that there are no longer any files or folders staged for commit.

2.2.3 Github

Why Use Github

Github works seamlessly with Git, and what’s more - functions well as a remote back-up for Git and is kind-of like a central repository with respect to sub-version - in the sense that it can provide the means for collaboration on a project across the seven seas (the internet).

This makes it the perfect companion of Git, and together provides a viable version control and back-up strategy. The ultimate back-up solution well-suited for Rails development in my personal opinion. I could back this up by saying that this is a trend very much endorsed by the development team behind Rails. (Hansson, *Rails is moving from SVN to Git*, 2008)

Setting Up Github

Setting up Github requires a registered account. With Git installed, an SSH Key must be generated for your email address with a passphrase. Once generated, the contents of the file to which the key is saved must be copy-pasted into your SSH Keys for your Github account. This then enables pushing local git commits to Github, usually asking for the password for the SSH

key for the first time in one terminal session prior to establishing a connection and permitting transfer.

Provided the email set in the global configuration for Git is the same as the email registered with Github, the only thing left to do is authorize with Github by issuing the command: `ssh -T git@github.com` and confirming the response dialog. (Github, 2012a)

Using Github

Using Github is straightforward. First, create a repository via Github (in the case of this project named *Skyscraper*). Then (with my username *defaye*), adding a remote repository known to the Git project by issuing the relative command to the details above:

```
git remote add origin git@github.com:defaye/Skyscraper.git
```

Finally, from the Git project we can then push to Github with the command:

```
git push origin master
```

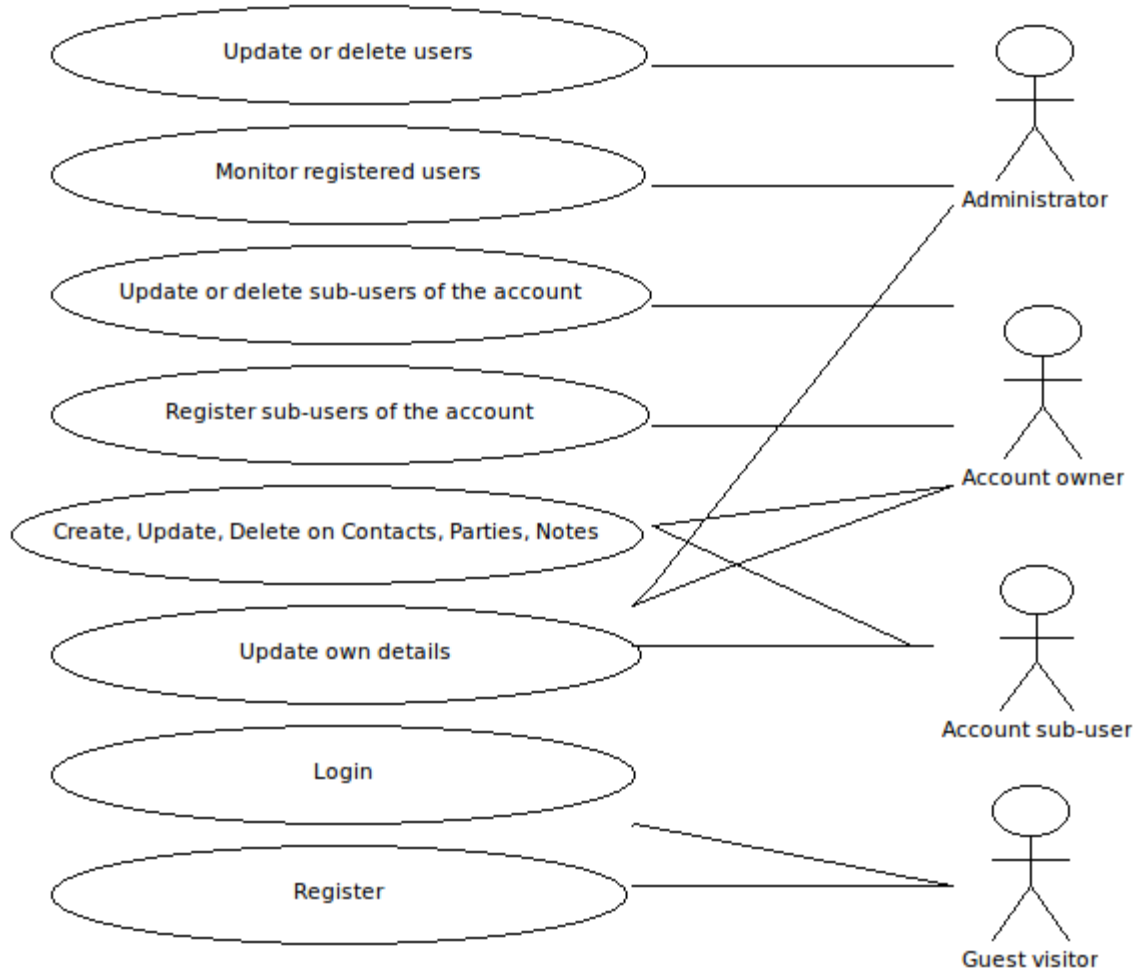
Once done, the state of development of the project can be accessed online from <https://github.com/defaye/Skyscraper>, which gives a traversable file and folder structure of the application in real-time, as well as showing the history of commits, with a graphical representation of differences to the files modified from previous commits, and a host of other functionality. (Github, 2012b)

The reverse process of downloading the master branch (the main branch worked on by default) and therefore, in obtaining a copy of the repository on to another machine is by initializing Git into some local directory, adding the remote repository as above and issuing the command: `git pull origin master`

2.3 Designing the Application

2.3.1 Use Case

The system has 4 distinct users and how they will use the system:



The Use Case Explained

- **Administrator**, self-explanatory, a user with elevated privileges
 - Uses the system monitor registered users
 - Has permission to update or delete any user at will
- **Account owner**, i.e. a representative of a business
 - Became a member via registration form
 - The user is considered to have a main account meaning:
 - the user represents a business
 - can register sub-users of the account

- can update or delete sub-users of the account at will
 - Can create, update and delete contacts, parties and notes
 - Can update the user's own details
- **Account sub-user**, i.e. an employee or colleague in relation to the account owner
 - Was registered by an account owner, and so belongs to that account
 - Can create, update and delete contacts, parties and notes
 - Can update the user's own details
- **Guest visitor**, i.e. a potential customer
 - Can not access any part of the internal system, however
 - Can register an account to become a main account holder, or
 - Can login to an existing account

3. Developing the Application

3.1 Creating the Layout

3.1.1 Templating Languages

Rails uses embedded ruby templates - ERB by default. Embedded ruby makes it possible to embed ruby code within an XHTML web document i.e. the view.

However, instead of using ERB as standard, I opted to use HAML³. HAML is indented markup; child nodes of an element are indented one point deeper than the parent node. To close the tag off requires only that the indentation return the same number indentation level as the parent node.

Comparison of the two templating languages

ERB	HAML
<pre><div id='foo'> <% Time.now %> <div class='bar'><%= Time.now %></div> </div> <foo>Bar</foo></pre>	<pre>#foo - Time.now .bar= Time.now %foo Bar</pre>

As you can see, HAML is a lot more beautiful, cleaner and semantically meaningful than ERB. XML tags are declared with the percent symbol, unevaluated ruby with a hyphen and evaluated and substituted ruby with an equals symbol - there's even shortcuts to divs with IDs by a number sign and a period for a div with a class - which matches up to CSS⁴ perfectly.

Sometimes compound statements can be used on the same line as well. This dramatically speeds up and simplifies the writing of views, with a bonus point to convention over configuration.

³ <http://haml-lang.com/> - the official website about HAML

⁴ CSS also uses a number sign to denote an ID and a period sign to denote a class.

Switching to HAML

Switching to HAML required a few things:

- Updating the Gemfile⁵ with `haml-rails` ruby gem⁶
- Running `bundle install`⁷ in console⁸ to fetch and install the gem
- Converting⁹ the existing application layout from ERB to HAML
- Renaming the existing layout filename extension from `.erb` to `.haml`
- Restarting¹⁰ the rails server to run the new gem in order to parse HAML
 - By terminating the existing process and running `rails server` to restart

With this done, writing views in Rails became a lot easier and faster to implement with the more semantic markup of HAML. All subsequent views generated by Rails would now be in HAML.

3.1.2 Implementing the Layout

The basic layout and structure of the application after login was essentially as follows:

```
+-----+
|  Header  |
+-----+
+-----++-----+
|  Side   || Main content area  |
+-----++-----+
```

This was achieved through the use of layouts in rails in combination with CSS scripting. The layouts in rails are kept within the directory `app/views/layouts` with the default layout being `application.html.haml`.

The layout would need a few things to function properly in rails:

- `stylesheet_link_tag` declaration in the head tag for any embedded stylesheets
- `javascript_include_tag` declaration in the head tag for any embedded javascript
- a `yield` declaration within a section of the body to which page content is loaded

Within the layout prescribed above, the `yield` declaration would go somewhere within the main content area, the rest of the design complexity has been omitted for brevity.

⁵ where gem dependencies are declared such that gem `'haml-rails'` will install HAML for Rails

⁶ <http://docs.rubygems.org/read/chapter/1> - a gem is a packaged ruby application or library

⁷ http://gembundler.com/bundle_install.html - the bundler gem ensures all dependencies are met

⁸ console, otherwise known as the terminal window or Bash, the command line prompt in Linux

⁹ <http://html2haml.herokuapp.com/> - using a website that automates the conversion of ERB to HAML

¹⁰ by terminating the existing server process with `Ctrl^C` and entering `rails server` to restart

Having made the transition from ERB to HAML, let us compare what the two might have looked like so we can clearly see the benefit of using HAML, which as you might agree, is a lot more intuitive and easy to read:

ERB	HAML
<pre><!DOCTYPE html> <html> <head> <%= stylesheet_link_tag "application"%> <%= javascript_include_tag "application"%> </head> <body class="sky"> <div id="header"></div> <div id="container"> <div id="left"></div> <div id="main"> <%= yield %> </div> </div> </body> </html></pre>	<pre>!!! %html %head = stylesheet_link_tag "application" = javascript_include_tag "application" %body.sky #header #container #left #main= yield</pre>

3.1.3 Implementing the Appearance

SASS

With the overall structure of the layout complete, it needed to be styled with CSS¹¹ - without any style, the layout would be nothing, and that includes the structural behaviour which can be achieved through the use of CSS as well.

Just one example of this being the `float`¹² property (set to `float left`) - needed on at least the `#left` element, in order to have the `#main` content area sit beside it properly.

I opted for another markup gem to help facilitate the implementation of stylesheets using SASS¹³. With indentation in place of brackets¹⁴ and newlines instead of semicolons to delimit a statement - and a lot of other tricks such as nesting (*see on the following page*) made writing CSS a process that became a lot more enjoyable.

¹¹ Cascading Style Sheets - a language used to describe the presentation semantics of a web document

¹² floating pushes an element to the left or right of its container, allowing other elements to wrap around it

¹³ <http://sass-lang.com/> - *Syntactically Awesome Stylesheets* - the official website

¹⁴ Using indentation to for logical code blocks instead of braces → ‘{ ... }’

SASS - server-side abstraction ¹⁵	CSS - client browser output ¹⁶
<pre>li font: family: serif weight: bold size: 12px</pre>	<pre>li { font-family: serif; font-weight: bold; font-size: 12px; }</pre>

The Static Width Layout

The original layout had static widths with the main content area having a width set at about 580 pixels, with a 200 pixel width side bar for navigation - optimised to cater for computer screens on a resolution¹⁷ of 800*600 as it would not exceed a width of 800 pixels.

This was suitable for most of the project progression. However I found it necessary to use a resizable content area to cater for displays exceeding the minimum resolution, not just for making use of space, but also for extending the layout for right side bar on some pages.

The Problem With Static Width Content

Another side bar for additional content would need a minimum width of 200 pixels. Together with the existing width, this would exceed the minimum resolution by roughly 200 over 800.

The solution might be to reduce the width of the main content area to about 400 pixels to accommodate. However this would significantly impair the viewable content area and look ridiculous on higher resolutions, having an enormous amount of unused space.

The chosen solution was to have an expandable and contractible content area - down to 400 pixels and up to 1200 pixels. This would improve the usability of the website at both ends of the resolution spectrum.

¹⁵ the code which is interpreted by the server before it served to the client web browser

¹⁶ the resulting output that has been generated by SASS code

¹⁷ the dimensions measured in pixels of a computer display

Implementing the Dynamic Width Content Area

Using *jsfiddle*¹⁸ to test HTML and CSS based on a tutorial¹⁹ adaptation, and looking at the embedded result²⁰ of the fiddle²¹, the current implementation had a flaw in that, the left side bar would depart from the main content on a resolution exceeding a width of 1200 pixels²².

To summarise - the layout used a left, center and right div²³ within a container div. To achieve the effect of having a contractible center content, the container was padded²⁴ on both sides with enough space for the side panels to take up that space.

This was not working and needed further adjustment. The end result was to use the container div to apply the minimum, maximum width rule; 800 to 1200 pixels. The center div took the its content area minimum, maximum widths; 400 to 800 pixels.

Then, a right and left margin was applied to the center div with the same widths (200 pixels) as the side panels, in order for it to appear in between them when they overlap.

To achieve an overlap of the side panels in the right positions, no float was applied to the center, with a float to the left and right divs to their respective sides. Together with their widths defined as 200 pixels, positioned them correctly without departing from the center.

In applying the width rules in this way meant that all children of the container would be at a width of at least 800 pixels, (meeting the minimum requirement) extending up to 1200 pixels at most, with the internal center able to contract down to 400 pixels with 400 pixels either side, and up to 800 + 400 when a full width.

¹⁸ <http://jsfiddle.net/defaye/DhaHP/4/> - jsfiddle: an online editor for snippets built from HTML, CSS and JavaScript which may be shared with others

¹⁹ <http://www.alistapart.com/articles/holygrail/> (Levine, 2006)

²⁰ <http://jsfiddle.net/defaye/DhaHP/4/embedded/result/> - the embedded result of the above

²¹ fiddle: a term used to describe a creation on JSFIDDLE

²² <http://www.whatismyscreenresolution.com/> - a useful tool to find out your current display resolution

²³ though it has no particular meaning, a <div> element is used to style the children it contains. (W3C, 2012)

²⁴ having additional space on the internal sides of an element

3.1.4 Cross-Browser Compatibility

There were a few particular recurring CSS3 styles that were widely used within the application - gradient-backgrounds, box-shadows, and border-radii. Of each of these recent additions to CSS, are different implementations among browsers.

To overcome this meant simply finding what each implementation was, and repeating the same style several times for each implementation for a given rule. What could have been a laborious task became very simple with the use of SASS mixins.

SASS Mixins

A mixin is a block of code that can be reused by either calling it from within the same style, or by first importing the style from what the mixin has been declared and then calling it.

Together with the ability to use variables within SASS made code reuse possible within stylesheets, as this meant a pseudo-function could be created to apply a style, and within the block applying the style in each browser's preferred syntax.

To illustrate just how valuable code reuse is with SASS, an example of a mixin which applies the gradient background compatible with many different browser standards, can be seen at: <https://gist.github.com/2333344>

The mixin can be called to apply a gradient-background (e.g. from grey to black) using `@include gradient-background(#ccc,#000)` within a particular rule.

Within one line, when the server process is executed, it looks up and compiles the rules for each browser's specific syntax, profoundly demonstrating the power of mixins.

3.2 Authentication

After having designed the layout for two distinct sections - (a layout for users without a session²⁵ and those with an active session) it was onto creating the users capable of logging into the system. For this, I used the *Authlogic* gem.

Looking up an example for *authlogic*²⁶ required certain attributes for the User model in order to work. So I had to ensure that these were met when creating the scaffold²⁷ for the users.

3.2.1 Creating the Users

After having noted these dependencies, I used a Rails script to generate a scaffold for the model, view and controller, accepting several arguments. The first of which being the name of the model, followed by each attribute (and its associated data-type which may be omitted if it is of type *string*, as this is otherwise assumed) with the following command:

```
rails g scaffold User username email crypted_password password_salt  
persistence_token
```

The command `rake db:migrate` would then create the `users` database with these attributes.

Worthy of note here are some specifics about what each of these attributes are for:

- The attribute `username` and `email` are self explanatory.
- The attribute `crypted_password` is the encrypted plain text password into SHA-512 (by default), which is considered to be one of the strongest encryption algorithms available. (Andrew H, 2007)
- The attribute `password_salt` is a way to further obfuscate the encryption by hashing the literal password with a salt to make the encryption more complex, for the purpose of impeding the use of table assisted dictionary attack²⁸.
- The attribute `persistence_token` is used for remembering a user's session.

This single command as of Rails 3.2 has quite a few omissions from previous versions, allowing `rails generate` to be shortened to `rails g`, like `rails server` to `rails s`, among most other calls of this kind, as well as `username:string` to just `username` (with string the default), makes it easier to quickly set up a scaffold in the framework.

After having generated the user scaffold, it would require a few tweaks to work as intended:

- In the User model, the *authlogic* code would be extended into the class with the

²⁵ a user who is not logged into the system

²⁶ https://github.com/binarylogic/authlogic_example

²⁷ Rails terminology to broadly define the automatic generation of the model, view and controller

²⁸ an exhaustive list-based attack which literally attempts to guess the password. (Shirey, 2007)

declaration `acts_as_authentic`. This would set up the authenticated class.

- Altering the form to ask for the username, email, password and password confirmation, removing references to the crypted password, salt and persistence token.

These changes meant that a guest could register an account via accessing the path `users/new`, however they could not yet login.

3.2.2 Creating the User Sessions

The user session would require a few things:

- A session model, to manage sessions
- A login form, to accept a username and password
- A session controller, to interact between the two

First creating the model with: `rails g model user_session`

Then extending the model by `authlogic` - altering the class declaration to:

```
class UserSession < Authlogic::Session::Base
```

Then removing the migration²⁹ that has been generated, as the session would be cross-referenced against the User model.

Then the login view and controller could be generated in one go with the following rails command to generate the user session controller with fields for the username and password for the login form: `rails g scaffold_controller user_sessions username password`

There would then be a few more tweaks:

- Removing redundant actions in `user_sessions_controller.rb` in `app/controllers` (leaving only `new`, `create` and `destroy`)
- Removing generated files that are redundant in the session views in `app/views/user_sessions` (i.e. all but the form partial³⁰ and `new.html.haml` (the login authentication page))
- Changing the field type from text field to password field in the login form

3.2.3 Logging In and Logging Out

One rather unpretty problem was having URLs for login and logout that look like `user_sessions/new` and `user_sessions/destroy`.

This was solved by editing the routes in `config/routes.rb`, the purpose of which is to have the paths for `login` and `logout` as they are.

²⁹ a generated class that creates the relevant database tables against the model it relates to

³⁰ a template that is rendered in another page, the most common example being a form

Fortunately this was simple in Rails with the lines:

```
match 'login' => 'user_sessions#new', :as => :login
match 'logout' => 'user_sessions#destroy', :as => :logout
```

3.2.4 Controlling Access to the System

To put a barrier between the logged in user and the guest, authlogic uses `:require_user` and `:require_no_user` to distinguish between the two. It is declared in a controller with `before_filter` which takes an optional `:only` parameter to specify the actions permitted.

So within the sessions controller:

```
before_filter :require_no_user, only: [:new, :create]
before_filter :require_user, only: :destroy
```

3.3 Contacts

Taking forward what was learned from the previous chapter, contacts would be made using a scaffold `rails g scaffold contact first_name last_name email title company` creating the model, views and controller ready to be edited. Then, migrating the new contacts table with `rake db:migrate`.

3.3.1 Associations

The first thing required of the newly created contacts would be to set up the relation between a contact and a user. A user can have many contacts and a contact belongs to one user as:

In `app/models/contact.rb`:
`belongs_to :user`

In `app/models/user.rb`:
`has_many :contacts`

This specifies a one-to-many association from user to contacts understood by ActiveRecord.³¹

3.3.2 Migrations

One problem was that now the relation had been made, there is no actual foreign key reference to the user in the contact, which should have been defined as `user_id:integer` in the scaffold script. To overcome this, a migration script can be used to add another attribute to the contacts table:

```
rails g migration AddUserIdToContacts user_id:integer
```

This migration configures a script in `db/migrate` that would run an SQL query on the database:

```
ALTER TABLE contacts ADD COLUMN user_id INTEGER via ActiveRecord as:  
add_column :contacts, :user_id, :integer
```

An alternative to running this migration script would have been to run the command `rake db:rollback` to undo the last migrate, and then, editing the create table script generated by the scaffold to include `t.integer :user_id` as one of the column declarations.

Whichever method to make the change is undertaken, the last step would be again to migrate the changes to the database with `rake db:migrate`.

3.3.3 Users Can Only Access Their Own Contacts

³¹ http://api.rubyonrails.org/classes/ActiveRecord/Associations/ClassMethods.html#method-i-belongs_to

Another problem is that with the generated controller code, using the index action as an example; the current behaviour is to retrieve all contacts with: `@contacts = Contact.all`

I want to show only the contacts belonging to the user which would need to know the ID of the user making the request, and thankfully, we have just implemented the foreign key reference earlier by adding the `user_id` column to the `contacts` table.

In order to obtain the current user's ID, I looked up how to persist a session in the documentation for authlogic³² and implemented the recommended helper methods into the `app/controllers/application_controller.rb`³³ in order to call them from the contacts controller³⁴.

By doing this, I was then able to use the current User object through the `current_user`³⁵ helper method to find all the contacts belonging to that user like so:

`@users = Contacts.where("user_id = ?", current_user.id)` however, this could be further simplified to: `@users = current_user.contacts` that has been enabled by the previous associations being set up.

In much the same way, the other actions in the contacts controller could be updated with `current_user.contacts` to prevent any users from accessing contacts to which they do not own as the SQL would not find a contact from an ID if they did not belong to that user as well.

3.3.4 Validating the Contact's Details

Details entered into the registration form for a user was largely taken care of by authlogic, that would recognise the username, email and password fields having suitable validations to prevent a magnitude of incorrect data being saved. But since contacts does not extend authlogic, validation for its attributes would need to be considered.

Of the five attributes (*first* and *last* name, *email* address, *title* and *company*) I decided that the only validation required at this stage in development was to ensure that the first and last name were present, and that the email address - if specified, was in the format of an actual email address (not caring whether or not it is genuine).

With help from the Rails documentation³⁶ I could use the `validates_presence_of` method to ensure that the first and last name were both present on the contact object, and not blank. For the email address format, I could have used the `validates_format_of`³⁷

³² <http://bit.ly/authlogic-sessions> authlogic documentation URL on persisting sessions

³³ the parent controller of all controllers, methods declared here will be accessible throughout the application

³⁴ location: `app/controllers/contacts_controller.rb`

³⁵ location: `app/controllers/application_controller.rb`

³⁶ <http://api.rubyonrails.org/classes/ActiveModel/Validations/HelperMethods.html>

³⁷ http://bit.ly/validates_format_of

method with a regex³⁸ for an email address, however I decided to use a gem³⁹ called *validates_email_format_of*⁴⁰. Then, adding another line to the `contact`⁴¹ model to allow no email address, but to check its format if it has been specified with:

```
validates :email,  
  email_format: { message: 'does not look like an email  
address.' },  
  allow_blank: true
```

The added message here specifies what the error should say if the validation failed and sent the user back to the contact form for correcting.

3.3.5 Formatting the Appearance of the Contacts Listing

Partial Templating

The current formatting of the contact listing was unacceptable and needed to be redone. This problem would be tackled by designing a template for one person that could be repeated to represent many people as well. This is where the use of rendering a partial template⁴² came in handy.

By creating a partial template named `_contact.html.haml` in the `app/views/contacts` directory, I could design how a contact would appear.

In the `index`⁴³ view, I could replace most of the existing markup with a single call to `render @contacts` that would (for each contact in the instance variable `@contacts`) render the `_contact.html.haml` template passing it a single `contact` object to use. In the template, data from the contact could be pulled from that `contact` object.

This partial template could then be reused in the `show`⁴⁴ view as well, greatly improving the appearance of contacts in the application.

Adding Gravatar Avatars

In addition to the partial template, and having an email address attribute that can be used, I decided to give contacts their own avatars, based on their email address using the web service *Gravatar* (<http://en.gravatar.com/>).

Through this service, people can register an account and have a profile that can be used by other web services. The highlight of Gravatar is the avatar service that it provides.

³⁸ regular expression: “a way to describe a set of strings based on common characteristics shared by each string in the set” (Oracle, n.d.)

³⁹ following the same install process; adding to the `gemfile`, installing with `bundler` and restarting the server

⁴⁰ https://github.com/alexandunae/validates_email_format_of

⁴¹ location: `app/models/contact.rb`

⁴² http://guides.rubyonrails.org/layouts_and_rendering.html

⁴³ location: `app/views/contacts/index.html.haml`

⁴⁴ location: `app/views/contacts/show.html.haml`

In order to use an image avatar from their web service, the source URL of the image must be an MD5 hash⁴⁵ of the person's email address that you are looking for, appended by `"http://www.gravatar.com/avatar/"`⁴⁶.

In addition to this, Gravatar provides a default image by passing in a query string⁴⁷ `"d="` to either some defaults provided by the service, or to specify another URL to link to the default. I would opt to use the service default `"d=mm"`, for a silhouette image of a person.

To implement the method to format an email as the image source request, I put the method within the contact model⁴⁸. As this would be a class method, it would be invoked by `Contact.gravatar(email)`, with the email as the contact's email address to be hashed.

To have a class method, it would be declared as `self.gravatar()` within the class, enabling it to be called from the class itself.

Within the method, it would return the string representation of the URL:

```
"http://www.gravatar.com/avatar/#{Digest::MD5.hexdigest(email)}.jpg?d=mm&s=60"
```

I used interpolation to sew the MD5 hash of the email into the string.

One problem was that, if a user had entered an email address with any uppercase characters, it would be impossible to source, as the web service stores email address in lower-case only. Fixing this involved using the `downcase` method to ensure the email was all lowercase characters.

Later iterations of the application would involve using avatars for users as well, and using the Contact class to call the method was no longer suitable. It then occurred to me that this should be a helper method, accessible to all views, so it was placed in the application helper⁴⁹ so that it could now be called from a view simply by `gravatar(email, size)`, later including another query parameter `size` to have various sizes of avatars in the application.

Paginating the Contact Listing With "Will Paginate"

To have the effect of paginations, I used a gem called `will_paginate`⁵⁰. Following examples from documentation, the only changes necessary were in the index view⁵¹ and the controller⁵², adding the `paginate` method to the end of the contacts query:

```
@contacts = main_account.contacts.search(params[:search]).paginate \
```

⁴⁵ <https://en.gravatar.com/site/implement/hash/>

⁴⁶ <https://en.gravatar.com/site/implement/images/>

⁴⁷ request data to be passed to an application (Berners-Lee, 1994)

⁴⁸ location: `app/models/contact.rb`

⁴⁹ location: `app/helpers/application_helper.rb`

⁵⁰ https://github.com/mislav/will_paginate

⁵¹ location: `app/views/contacts/index.html.haml`

⁵² location: `app/controllers/contacts_controller.rb`

```
page: params[:page], order: 'last_name, first_name asc', per_page:
15
```

This method would have a maximum of 15 results per page, in ascending order - primarily on last name, then by first name. The backslash after `paginate` tells the interpreter to continue parsing the statement on the next line. Then, adding the `will_paginate` method to the bottom of the index view as: `will_paginate @contacts`

Modifying the Time Format

I wanted to display the creation date of a contact on their summary view⁵³ in the format “*Tuesday, April 3*”. To do this, I took advice from a *RailsCasts* episode⁵⁴ on formatting time, to use my own format that would be loaded into the application’s environment at runtime by adding a date format to an initialiser script⁵⁵ with the statement:

```
Time::DATE_FORMATS[:day_month] = "%A, %B %e"
```

With reference from Ruby documentation⁵⁶, I formatted `:day_month` to be in the format desired.

I could then use the `created_at` attribute of a contact with the chained methods, `to_datetime.to_formatted_s`⁵⁷ with the new format to have the desired effect:

```
contact.updated_at.to_datetime.to_formatted_s(:day_month)
```

As the initialiser scripts are loaded at runtime into the environment, the `:day_month` format was not accessible until I restarted the Rails server.

⁵³ location: `app/views/contacts/_contact.html.haml`

⁵⁴ <http://railscasts.com/episodes/31-formatting-time>

⁵⁵ location: `config/initializers/time_formats.rb`

⁵⁶ <http://www.ruby-doc.org/core-1.9.2/Time.html>

⁵⁷ http://api.rubyonrails.org/classes/DateTime.html#method-i-to_datetime

3.4 Deploying the Early Prototype

With an initial login, user registration, layout and contacts that can be added and viewed, I felt that it was appropriate to test putting the application online. For that I would use Heroku⁵⁸.

3.4.1 Heroku

Heroku provides a platform as a service (PAAS) for building, deploying, and running cloud apps using Ruby. (<http://www.heroku.com/business>) This enables developers to focus on the development of the application itself and worry less about the specifics of deployment, as Heroku takes care of the details.

Apart from having a free service that helps greatly in testing an application online, Heroku supports `git push`, enabling it to work seamlessly with Git. This also meant that, without problems, deploying the application online would be a few steps away from `git push heroku master`.

3.4.2 Setting up Heroku

On Linux, I downloaded the Heroku client application for interfacing with Heroku by downloading and executing the installer using the command:

```
wget -qO- https://toolbelt.heroku.com/install.sh | sh
```

After having created an account, I could then login with my credentials to create an SSH key specific for my machine with the command: `heroku login` (<https://toolbelt.heroku.com/linux>)

From the application directory, I then ran `heroku create --stack cedar`, to initialise an application on Heroku. Then, from my account, renamed the application to *skyscraper*, and added the remote repository to Git, by:

```
git remote add heroku git@heroku.com:skyscraper.git
```

All that was left was to *push* to Heroku.

⁵⁸ <http://www.heroku.com>

3.4.3 Pushing to Heroku

After having tried to push to Heroku by `git push heroku master`, it failed because it needed to be setup for the database running on PostgreSQL⁵⁹.

To overcome this problem, it meant changing the Gemfile⁶⁰ and re-bundling⁶¹. However, in doing this, I would no longer be able to develop the application and test it using SQLite3, and would have to setup PostgreSQL to run locally as well.

The benefits of developing in the same database as the production database was obvious, in terms of testing and ensuring compatibility - however much of this was outweighed by the platform-as-a-service (Heroku), and time constraints in the project.

I came to the decision to *branch*⁶² off from the master branch (by issuing the command `git branch naruto` (naming it *naruto*)), so that I could maintain a separate version of the application that was specifically tailored to be pushed to Heroku.

After having branched, I could switch into that branch by `git checkout naruto`. Then, treating it as a regular Git repository, making changes and committing them as per usual.

Naruto would use the 'pg' gem for PostgreSQL instead of the 'sqlite3' gem. Heroku takes care of generating the `config/database.yml` file (which stores the settings to connect to the database) automatically. (JD, 2012)

After having done this, I could then push to Heroku with a slight modification to the push command to push from one branch into another, i.e. from *naruto* into Heroku's master branch, by `git push heroku naruto:master`.

Once on Heroku, I would need to migrate the database with the command:

```
heroku run rake db:migrate, to initialise the tables for the first time.
```

From then on, at any time that I would want to deploy changes in the master branch to Heroku, I would checkout *naruto*, and merge the changes from the master branch into it by issuing the command: `git merge master` (from *naruto*) and then pushing it to Heroku.

⁵⁹ <http://www.postgresql.org/>

⁶⁰ location: Gemfile (in root directory)

⁶¹ running the command `bundle install`

⁶² branching is a way to develop in isolation from another branch that can be merged again at any time

3.5 Creating a Search for Contacts

3.5.1 Implementing the Search

Model, View and Controller

The idea behind the search was that a form would be put into the main layout⁶³ (in order for it to persist on every page) - when used, it would pass the search argument in parameters from the form to the contacts controller⁶⁴ index action.

Within the index action, the `current_user.contacts` statement (performing an SQL search for all contacts belonging to the current user) would be used as the basis of the search, and, using a class method declared in `Contact`⁶⁵, perform the search with the search argument passed up from the form via the controller.

```
@contacts = main_account.contacts.search(params[:search])
```

Contact Class Method for Performing the Search

Within the class, a method would be made that would operate on the results of the SQL query from the controller (obtaining all contacts belonging to the user), and performing another search on those results based on the search argument given.

The method would be designed initially such that, if it was passed a search argument, it would perform the search, or otherwise just return all results.

Future Versions of the Method

Later versions of the method would check to see if there was any search argument, then checking to see if the search contained more than one pattern, and searching on *first* and *last* name, or otherwise searching on either the *names* or *email* address. If there was no search argument in the first place, it would return all contacts by default.

Using Squeel for SQL Instead of the ActiveRecord Default

I decided to use another gem for creating the SQL statements using Squeel⁶⁶. I will demonstrate its use in comparison with the Rails standard when demonstrating the first search query example later on.

3.5.2 Single String Pattern Search Algorithm

Initial problems with the search algorithm were due to the way the names are stored in the database (split into first and last name). When the search was used with a whole name e.g.

⁶³ location: `app/views/layouts/application.html.haml`

⁶⁴ location: `app/controllers/contacts_controller.rb`

⁶⁵ location: `app/models/contact.rb`

⁶⁶ <https://github.com/ernie/squeel>

“%jonathan feist%”, no result could be found as the match would search on the entire input against the first or last name. Such that:

Squeel:

```
where{(first_name.like "%#{search}%") |  
  (last_name.like "%#{search}%")}
```

ActiveRecord:

```
where ["first_name LIKE ? OR last_name  
LIKE ?", "%#{search}%", "%#{search}%"]
```

A search on %feist% would return the contact by my name because the search criteria would match on my last name, however a search on my whole name would find nobody because it would not match against either the first or last name.

3.5.3 Multiple String Pattern Search Algorithm

This was resolved by splitting the search on any spaces and if the search was comprised of more than one element in the search array, then checking to see if those parts matched in both the first and last name. In essence the search function would look like:

```
search = "%#{search}%" .split .join(' % ') .split  
where{ (first_name.like_any search) & (last_name.like_any search) }
```

This takes the search value e.g. “jonathan feist”, and interpolates it as “%jonathan feist%”. Then splits on any spaces into an array [“%jonathan”, “feist%”]. Then rejoining, glueing together the elements into a string to “%jonathan% feist%” and finally splitting it again to [“%jonathan%”, “feist%”].

This would produce an anomaly however. Due to the nature of the search finding anything that relates to any part of any word in the set it would allow a search like ‘*S Lightman*’ to retrieve *Astrid Farnsworth*, *Sharon Wallowski* and *James Wilson*, because the ‘S’ prevails in the first and last name.

3.5.4 Removing Single Character Patterns

This problem was resolved by deleting any element from the array with a length of one character, as a single character wildcard would be highly likely to return invalid results.

```
search = search.split.delete_if { |s| s.length == 1 }.join(' ')
search = "%#{search}%" .split.join('% ').split
```

With two characters (or more), the chances of false positive results is significantly less likely, and so decidedly permitted (that only single character patterns should be removed).

3.5.5 Refactoring the Search Algorithm

In deliberating over whether or not the search algorithm was as efficient as possible, (that would potentially be a major concern if the application were to be used by thousands of customers (*in my dreams*)), then a task such as this had its merits.

Refactored Search Algorithm

The final part of the search code was refactored to one line where `map` iterates over each element to wrap it in wildcards ready for SQL:

```
search = search.split.delete_if{|s| s.length == 1}.map{|s| s
= "%#{s}%"}
```

However, this version could further be optimised.

The `map`⁶⁷ function returns a new array, taking up additional CPU cycles, which could be changed to `map!`⁶⁸ function changes the original array, saving CPU cycles.

One last change was to use the function `reject!` instead of `delete_if`, however in the documentation⁶⁹, it would appear that they are equivalent functions anyway. But I found that exclamation mark notation to signify that a mutation⁷⁰ is occurring gave greater syntactic clarity.

Final Search Algorithm

The result of these changes would produce:

```
search = search.split
search.reject! { |s| s.length == 1 }
search.map! { |s| "%#{s}%" }
```

⁶⁷ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-map>

⁶⁸ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-map-21>

⁶⁹ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-reject-21>

⁷⁰ changing the original argument passed to a function, rather than returning an altered copy

3.5.6 Benchmarking the Competing Algorithms

To benchmark the two algorithms, I would use `irb` (interactive Ruby) a command line Ruby interpreter, with the `Benchmark` class and a function to benchmark the two search functions as:

```
require 'benchmark'

Benchmark.bm do|b|
  b.report("1st ") do
    search = "jimmy a b smith"
    1_000_000.times do
      search = search.split
      search.reject! { |s| s.length == 1 }
      search.map! { |s| "%#{s}%" }
    end
  end

  b.report("2nd ") do
    search = "jimmy a b smith"
    1_000_000.times do
      search.split.delete_if{ |s| s.length == 1 }.map{ |s| s
= "%#{s}%" }
    end
  end
end
```

The results would show that the first (*1st*) algorithm would be noticeably faster:

	user	system	total	real
1st	1.210000	0.000000	1.210000	(1.215855)
2nd	2.290000	0.000000	2.290000	(2.292804)

3.6 Notes

The main purpose of a note was that it should be associated to a contact. This would also be visually represented by displaying all notes belonging to a contact on their profile page (i.e. the contact show view⁷¹).

3.6.1 Implementing the Notes Scaffold

I would begin by creating the notes scaffold with the command:

```
rails g scaffold note contact_id:integer user_id:integer body:text
```

This would create the Model, Views and Controller (MVC) as a base to work from.

3.6.2 Associations

Through the scaffold generation script, you can see that I made reference to two foreign keys (to the contacts and users tables primary keys). This is because the note belongs to a user through a contact - which would be represented via association:

```
app/models/contact.rb → has_many :notes, dependent: :destroy
```

```
app/models/user.rb → has_many :notes, through: :contacts
```

The `dependent: :destroy` declaration would have the effect of an `ON DELETE CASCADE` in SQL, such that if the contact was deleted, notes relating to that contact would also be deleted. Also note that in User model and its association to contacts, it also has the same declaration. This would mean that if a user account was deleted, so too would the contacts associated to them, which would also delete all notes associated to those contacts.

3.6.3 Working Notes Into the Contact View

The task was to have notes that could be both posted and displayed on the same page as the summary details of a contact. This meant integrating the notes form on the page as well as displaying all notes belonging to that user.

Through the use of partial templates, displaying both the note form⁷² and the note partial⁷³ was possible by creating an empty `Note` object as `@note` in the show action of the contact controller⁷⁴ to be used by the form.

In rails, an object is used to initialise forms as it needs to be able to extract details from an object if it is an edit request. In order to have *one* form for both types of request (*new* and *edit*), an empty object is used for new requests.

⁷¹ location: app/views/contacts/show.html.haml

⁷² location: app/views/notes/_form.html.haml

⁷³ location: app/views/notes/_note.html.haml

⁷⁴ location: app/controllers/contacts_controller.rb

To finish the form, it was then rendered to the page by specifying the path to the form:

```
render 'notes/form'
```

To show all the notes associated to the contact being viewed, another instance variable `@notes` was declared as `@notes = @contact.notes.order('created_at desc')` which, through the association, obtained all the notes belonging to the contact, in descending order of creation time.

But how would the form obtain the contact and user ID to be correctly allocated, without having to push this responsibility on to the user itself?

3.6.4 Assigning the Correct User and Contact ID to the Note

To assign the correct user ID, the `current_user` method would be used in the create action of the notes controller⁷⁵ after initialization of a note object from the form parameters. This would ensure that any `user_id` specified in parameters would be ignored, in case of the wrong ID being specified.

Specifying the contact ID would be a little more complicated, however it was achieved by:

- Creating a hidden field within the form for the `contact_id`
 - Assigning the value of the field to value of the `:id` in parameters, which is derived from the RESTful⁷⁶ URL of the contact being shown, e.g. “`domain/contacts/1`” would point to the contact by `id = 1`, and have this `:id` stored in parameters:

```
= f.hidden_field :contact_id, value: params[:id]
```

The alternative to this method would have been to assign the `@note` object's `contact_id` to `params[:id]` within the controller⁷⁷. I felt the job was better delegated to the view, as it could all be assigned within one statement.

⁷⁵ location: `app/controllers/notes_controller.rb`

⁷⁶ Representational State Transfer (Tilkov, 2007)

⁷⁷ location: `app/controllers/contacts_controller.rb`

Updating a Note

One problem that arose from this was on update. As the contact ID was being derived from parameters, editing a note from the RESTful URL “domain/notes/15/edit” meant that it was losing ownership on update if the ID of the note differed from the ID of the contact (almost always going to be true).

To overcome this, the update action of the notes controller would only update the body attribute of the record. In other words, it would only update the text of the note, and leave the user and contact details alone, as they should never need to be edited anyway.

This meant a changing the `update_attributes` method on the object to `update_attribute` on `:body`, with the value of the body element in the parameters array such that:

```
@note.update_attribute(:body, params[:note][:body])
```

Preventing Dirty Notes

With the contact ID being assigned by parameter and being held as a value in a hidden field, this meant that there was a loophole that could enable a malicious to pass dirty notes onto another user’s notes by changing the ID to a random number that did not belong to their own account.

I could not find any way around having a hidden field attribute, so the only way to prevent this kind of mal-use was to check if the contact ID belonged to the user creating the note.

To do this, I defined a custom `validate` method `belongs_to_user` within the Note model:

```
validate :belongs_to_user, on: :create
private
def belongs_to_user
  errors.add :base, 'Invalid request.' \
    unless Contact.find_by_id(self.contact_id).user_id == self.user_id
end
```

First of all, the validation was made to occur only when a note is created for the first time. Having passed the validation thereafter not being permitted to change the user or contact ID, there was no reason to force validation on update as well.

This method would then prevent a note being saved and return an error unless a contact was found who could be cross-referenced by the user ID in the request.

Subsequent Version of the Validation Method

With the addition of sub-users later on, the method was updated to take into account that a note may belong to a sub-user. This meant updating the validation method with the ID to cross-reference as either the main account holder’s ID if the user making the request was a sub-user.

```
private
def belongs_to_user
```

```
id = User.find_by_id(self.user_id).user_id || self.user_id
errors.add :base, 'Invalid request.' \
  unless Contact.find_by_id(self.contact_id).user_id == id
end
```

If it is a sub-user, the main account owner's ID is assigned to `id` (which can be deduced from whether or not they have a foreign key reference to another user) or else it is assumed that they are the account owner and their user ID is used.

3.7 Sub-Users

Sub-users would provide a way to have work colleagues have access to the same account under a different profile. To make it work, I decided to have a recursive association on the user model, such that it would reference the account owner's ID if they were registered under a sub-user's account.

3.7.1 Recursive Association

To implement the recursive relation, I simply declared: `has_many :users` within the user model⁷⁸. At the same time, I edited the `create_users` migration⁷⁹ to include `t.integer :user_id`, so that there was a column within the users table to reference to the ID of the parent user if it was a sub-user.

3.7.2 Re-Seeding the Database

Instead of creating a migration to add in the additional column, I edited the original create table migration directly to make this change. The reason that a migration is useful is when you want the database to persist. However, all the data in my database is tangible at the development stage, and all the records in it can be re-seeded after migration, so there is no real loss of time or effort.

The process to incorporating the changes to the database would involve:

- Deleting the old sqlite3 database⁸⁰
- Running `rake db:migrate` from command line
- Seeding the database with sample data by running: `rake db:seed`

3.7.3 Formatting the User Views

Now that sub-users would be implemented, the users would eventually be given access to see who is registered with access to the same account and who the account owner is.

The account owner would have administration rights over sub-users, with the ability to register a sub-user (the only way possible within the scope of the project), edit a sub-user, or delete a sub-user.

This meant that all the user views would now come into play as the index view would show all users pertaining to the account, as well as the other views for editing, creating and showing a user that would already be used by a single user.

Displaying Users

To display users, I would copy the style of a contact's summary using a partial template again⁸¹, with changes to make it unique to a user.

⁷⁸ location: `app/models/user.rb`

⁷⁹ location: `db/migrate/20120229164200_create_users.rb`

⁸⁰ location: `db/development.sqlite3`

⁸¹ location: `app/views/users/_user.html.haml`

In the partial, I would put the gravatar method to use again, as users register with an email address as well, and it adds some personalisation.

I would also use conditional statements within the view to distinguish users on a main account, so that it would only give the options to edit or delete users to the main account holder on sub-users.

I would also go on to prevent users being wrongfully viewed, edited or deleted with changes to the controller, as the code in the view by no means provides the integrity against violation.

3.7.4 Checking for the Main Account

With sub-users potentially able to roam within the account owner's realm, some old functions would no longer work as intended, namely the use of `current_user` within controller queries to obtain results related to an account.

The assignment query `@contacts = current_user.contacts` from a sub-user would fail as contacts (should) belong to the main account holder.

The reason the function would no longer suffice is because ownership of contacts and notes currently works on the premise that they belong to user currently logged in, and this is no longer the case.

To overcome this, (as alluded to earlier), I decided that all contacts created should only belong to the account owner. Notes can (and should) belong to whoever created them, however this would not be a problem as notes belong to contacts - themselves to the main account⁸².

3.7.5 Required Methods

Two new methods would be needed to take the application forward:

- A method to check to see if the current user is the account owner
- A method to access the account owner of the sub-user

These methods would be used throughout the controllers (mainly) and occasionally within the views. The most suitable place to declare these methods would be within the parent controller⁸³ as this would then allow the methods to be accessed from all other controllers and therefore from within all views as well.

The methods were defined as below:

```
helper_method :main_account?, :main_account
```

```
def main_account?  
  current_user && current_user.user_id.nil?  
end
```

```
def main_account  
  if main_account?
```

⁸² the details of which were defined in the association discussed in the Notes chapter

⁸³ location: `app/controllers/application_controller.rb`

```

    return current_user
  else
    User.find(current_user.user_id)
  end
end
end

```

The method `helper_method` specifies the methods declared in the controller that may be shared in the view. So having been declared in the application controller, these methods would then be accessible to all views.

The `main_account?` method simply returns a boolean (`true` or `false`) if the `user_id` attribute of the user currently logged in is `nil` (otherwise known as `null`, or no value). This would identify if the user is an account owner, as sub-users will always have a foreign key reference to the parent.

The `main_account` method then checks to see if the user currently logged in is the owner, and if so returns that user. If not, it will return the account owner from that current user's foreign key.

3.7.6 Putting the Methods to Use

The majority of changes would be throughout the existing controllers, to reference from the `main_account` instead of the `current_user`, for example, the query to list all contacts would now be `@contacts = main_account.contacts`.

Without going into the details, these methods could be used interchangeably to allow sub-users to have access to data pertaining to the main account.

3.7.7 Sub-User Registration

Now that most of the groundwork had been laid for sub-users to use the application, one of the only things left to implement (and also to control) would be the registration of a sub-user's account.

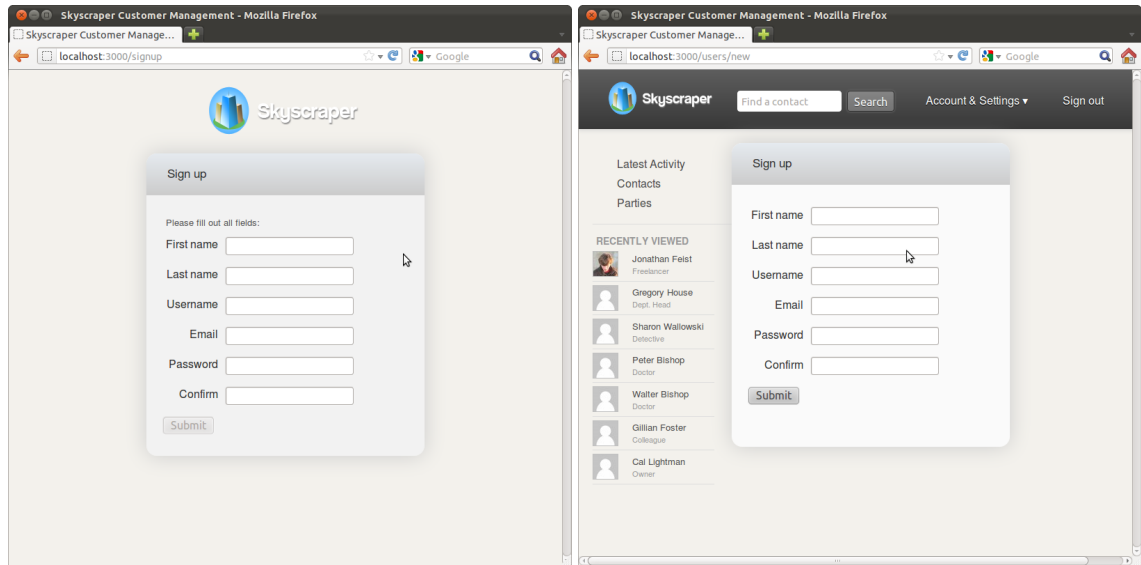
Resolve Layout

The `resolve_layout` method, was used only in the users controller⁸⁴, to select a layout based on whether or not the user was an administrator. This was to allow an administrator to create users without the layout being changed to the layout for the login screen.

As main account users would be extended this luxury, the layout picker method would need a minor change to check whether or not the user is a `main_account?` instead (as an administrator would never be a sub-user and so the check for `admin` becomes redundant).

This would mean that main account holders can create sub-users, and their screen would remain within the application layout, as opposed to the layout for unregistered users as below:

⁸⁴ location: `app/controllers/users_controller.rb`



Creating the Sub-User

The final changes would be within the new and create actions of the users controller.

In the new action, the existing check `current_user_session.nil?` returns true if there is no user logged in, and thus permitting registration. For administrators and main accounts, it would be edited to append a conditional OR statement to check for the main account:

```
if current_user_session.nil? || main_account?
```

If neither were true, a message would be sent to the notice page upon redirection to the portal, saying 'Must be logged into main account', thereby preventing additional registrations from registered sub-users.

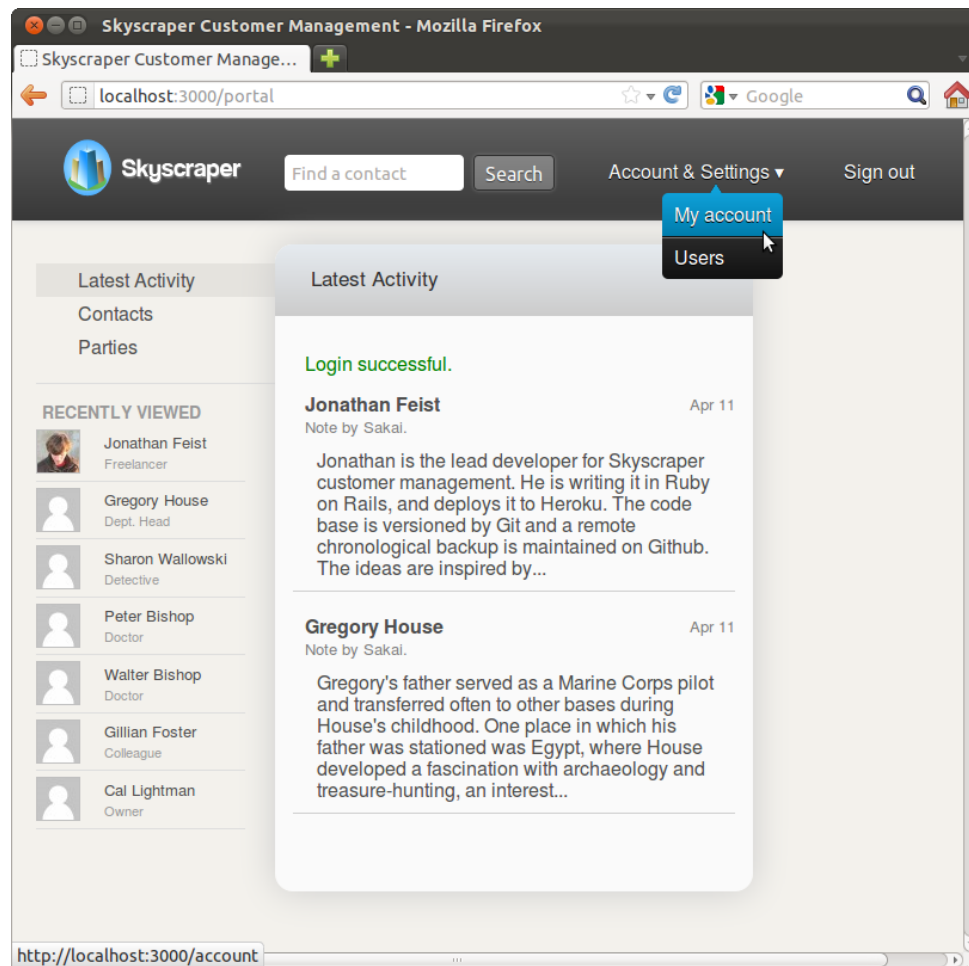
In the create action, it would then simply assign the `user_id` attribute of the new user object to the account owner's ID if they were on a main account:

```
@user.user_id = current_user.id if main_account?
```

3.8 Drop-Down Menu for User Accounts

The need for a dropdown was immediately apparent after the implementation of sub-users. What used to be an administrator-only privilege (to even see information on any user other than one's self) meant a link only to a user's account also needed to include a link to all user accounts (restricted to seeing only users associated to the account for regular users).

To keep maintain having the account link towards the top of the layout, it now seemed appropriate to have a drop-down menu with a link to the user's own account, and a link to all users associated to the account.



3.8.1 Re-Inventing the Wheel

The prospect of spending time on a two-link drop-down menu seemed ridiculous when there must be many solutions widely available on the web already. Indeed, in a professional environment, there would come a time when a decision has to be made on whether or not to reinvent something that has already been solved once before.

Robins pitches the balance rather well, suggesting that students should copy and learn from existing code to help understand how it works to a greater degree than working professionals. Whereas working professionals often need to be able to pick the right tool for the job, making alterations where it might only partially solve the problem. (Robins, 2012)

To get a working drop-down menu, I used a tutorial⁸⁵ to learn how a suitable solution had been implemented, and then re-modelled it to suit my application.

3.8.2 Implementing the Menu

These are the processes I went through in order to reuse the Catalin's solution:

1. Converting the CSS to SASS⁸⁶
2. Importing existing mixins to apply CSS3 styles by `@import mixins.css.sass`
3. Removing repetitive styles with mixin includes such as `@include border-radius(5px)`
4. Saving the style in its own file within `app/assets/stylesheets`⁸⁷
5. Adding the navigation links for the Account and Users as list items to the layout
6. Assigning the UL element the `#menu` ID, to match the style in the stylesheet
7. Loading the application in the browser and modifying the style until it suited the layout

At step 4, I decided to implement a partial template for the top menu navigation. At the same time I create another partial template for the left navigation as well, and placed each partial within its own view directory in `apps/views/navigation` as `_top_menu.html.haml` and `_side_menu.html.haml` respectively.

I would then load the partials by using `render 'navigation/side_menu'` and `render 'navigation/side_menu'` in the areas in which the menus would appear within the layout before. By doing this, I would be able to simplify the layout a bit, as well as allow me to focus on the structure of the top menu in isolation from the layout.

3.9 Recently Viewed and Activity History

In making contacts easy to find, I decided to have a recent activity history to display all the recently viewed contacts and recently created notes.

Contacts recently viewed would be accessible through the layout on the left side bar, underneath the links, with recently created notes accessible from the entry point page, renamed from *'Home'* to the more adequately named *'Latest Activity'*.

⁸⁵ <http://www.red-team-design.com/css3-dropdown-menu> Rosu, Catalin. (2011)

⁸⁶ <http://css2sass.herokuapp.com/>

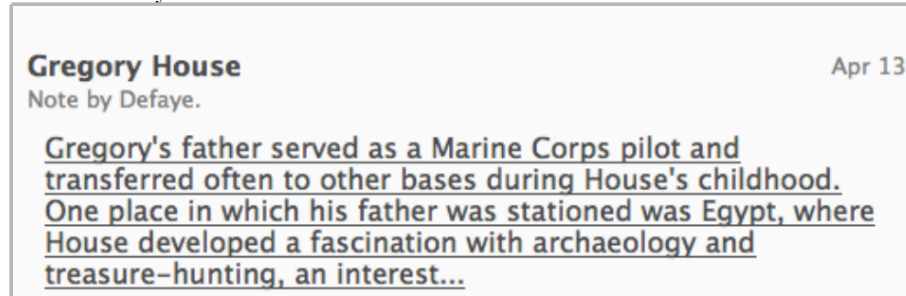
⁸⁷ was later placed in: `app/assets/stylesheets/optionals/dropdown.css.sass`

3.9.1 Implementing the Notes Activity

Putting together a feed of notes created for the main page would be relatively simple. As it would essentially be similar to the index action for notes, but limit the number of notes returned to 10 (as it only needed the most *recent* notes).

The activity for notes would also rely on the original notes partial template. This would need to be adjusted for use in the activity feed, as I decided to truncate the note text for brevity, and also intended to make the text a clickable link to follow through to the actual note.

Recent Activity Note



Controller

When the application layout was first devised, I created a controller named *sky*⁸⁸ with one action for the index with its relative view⁸⁹. It would serve as the entry point page for logging in, and now also as the *recent activity* history for notes.

As instance variables⁹⁰ are accessible to the view, I simply had to run a query from within the index action of this controller to retrieve all notes belonging to the main account, ordered by when they were created, and limited to 10:

```
def index
  @notes = main_account.notes.order('updated_at DESC').limit(10)
end
```

When checking the development log⁹¹ and loading the portal, by clicking on the logo or the link for *Latest Activity*, it would produce the query in SQL:

```
SELECT "notes".* FROM "notes" INNER JOIN "contacts"
ON "notes"."contact_id" = "contacts"."id" WHERE "contacts"."user_id" =
1 ORDER BY updated_at DESC LIMIT 10
```

View

⁸⁸ location: app/controllers/sky_controller.rb

⁸⁹ location: app/views/sky/index.html.haml

⁹⁰ in Ruby, instance variables are appended by an *at* sign → @

⁹¹ location: log/development.log

In the index view, I could render the results of the query from the controller by the statement:

```
= render @notes
```

The only problem with this was that it would format in the same style as notes on the contact page. I wanted to truncate the text and make it clickable as a link to the actual note.

I perhaps should have just made another partial template by creating one in the sky view as `_notes_activity.html.haml` and rendering it by:

```
= render 'sky/notes_activity', object: @notes
```

As well as passing it the `@notes` object (so that it would have access to this object from the partial (as it was not related to the sky controller), this would work.

Indeed, this may be something to change when refactoring the application, as it would reduce the complexity of the note partial template, that would be made more complex by what I actually did.

I differentiated between notes on the activity feed and notes on the contact page and viewing a note in isolation by checking to see if the parameter ID had been set, as this is something both the contact and note pages had in common (e.g. `skyscraper/contacts/1` and `skyscraper/notes/12`), whereas the activity page had no parameter set (e.g. `skyscraper/portal`).

Thus, in the partial template, I set a variable `actual_note` to the parameter ID if present. I could then alter the view⁹² accordingly, based on whether or not the partial was loaded from the portal page, and had allowed me to have the truncated, clickable text like so:

```
link_to truncate(note.body, length: 256, separator: ' '), note
```

3.9.2 Implementing the Recently Viewed Contacts History

To implement a section on the side bar to show the most recently viewed contacts, I would need to be able to store data on when those contacts were looked at.

I decided that I should create a new table in the database for activities, that would save the foreign keys of the users and contacts table. To do this I used Rails command line to generate the model and migrate (and create) the new table.

```
rails g model activity user_id:integer contact_id:integer
rake db:migrate
```

The table would be *descendingly* ordered by the auto-incremented primary key ID. This would mean the latest activity would have the highest ID value, the oldest activity, the lowest value - which would come in handy when implementing the method to update the activity.

I then set up the associations to the relative models so that it expressed users and contacts having many activities and that activities belong to a user and contact.

I wanted to list a maximum of 10 contacts, ordered by the most recently viewed. If a contact already exists in the list and is viewed again, they get pushed to the top. In addition to this, the list should never exceed 10 contacts, as the table would become filled with extraneous records.

⁹² location: `app/views/notes/_note.html.haml`

To populate the activities table, I would use the show action of the contacts controller⁹³ to invoke a method that would update the user's activities whenever a contact was viewed.

```
Activity.update_activity(current_user.id, @contact.id)
```

⁹³ location: app/controllers/contacts_controller.rb

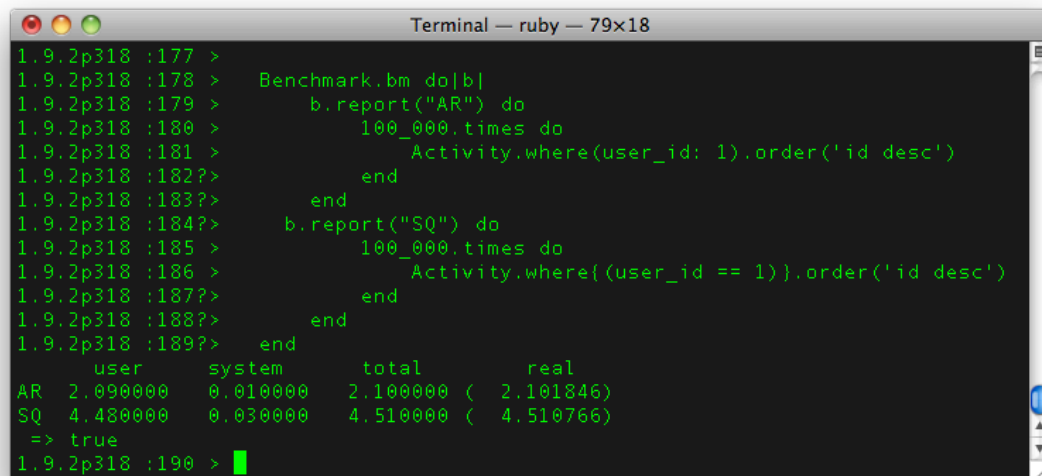
The method would accept the contact and user IDs to add a record in the table, which would be defined within the Activity model⁹⁴. The `update_activity` method implements the following logic:

- Check for any activity, otherwise go to `create(3)` (as an empty list has no conflicts)
 1. Check for and delete any conflicting contact in the list and go to `create(3)`.
Otherwise go to check the list size(2)
 2. Check if the list has reached the maximum size and delete the oldest activity (i.e. the record with the lowest ID value), and/or go to `create(3)`
 3. Create a new activity record from the user and contact IDs

3.9.3 Problems

Just one minor implementation detail that would make a difference in performance on testing was in the use of the Squeel gem for SQL. For some queries, using the default ActiveRecord SQL syntax ran twice as fast as its Squeel counterpart.

Using the benchmarking library through Rails console, I ran a simple comparison of a query that was being used in the activity update method, demonstrating the speed increase when using the native ActiveRecord syntax when running comparative queries over 100,000 times:



```
Terminal — ruby — 79x18
1.9.2p318 :177 >
1.9.2p318 :178 > Benchmark.bm do|b|
1.9.2p318 :179 >   b.report("AR") do
1.9.2p318 :180 >     100_000.times do
1.9.2p318 :181 >       Activity.where(user_id: 1).order('id desc')
1.9.2p318 :182?>     end
1.9.2p318 :183?>   end
1.9.2p318 :184?>   b.report("SQ") do
1.9.2p318 :185 >     100_000.times do
1.9.2p318 :186 >       Activity.where{user_id == 1}.order('id desc')
1.9.2p318 :187?>     end
1.9.2p318 :188?>   end
1.9.2p318 :189?> end
   user      system      total      real
AR  2.090000    0.010000    2.100000 ( 2.101846)
SQ  4.480000    0.030000    4.510000 ( 4.510766)
=> true
1.9.2p318 :190 > █
```

The main problems would come in understanding how to best handle the SQL queries to implement the logic in the most efficient and simple way. It took several revisions of the method to come to the most appropriate solution.

Query Methods Versus Array Methods

One important lesson taken away from the implementation of the latest contacts viewed history, was how queries work in Rails. Quite often I was trying to use methods only available to

⁹⁴ location: `app/models/activity.rb`

`ActiveRecord::Relations`⁹⁵ (a relational database object as opposed to a standard array object (all things being objects in Ruby) when they had changed to an `Array`).

For example, on a line in which I was checking for any activity history:

```
history = where(user_id: user).reverse
```

I was writing an SQL query, but the last method on the chain⁹⁶ (`reverse`)⁹⁷ was outputting an `Array` object. The statement would be fine now, but the result was no longer an `ActiveRecord::Relation` object, and so could no longer use query methods⁹⁸.

The statement would fetch all records from the activities table pertaining to the current user, and order them from highest to lowest ID value (the most recent at the head of the array).

I then wanted to check for the conflicting contact's existence using the result from the previous statement in part, to reuse code and save CPU cycles, with:

```
previous = history.where(contact_id: contact)
```

But would result in an error, the reason for which described several paragraphs above:

```
NoMethodError: undefined method 'where' for #<Array:0x000001031a3408>
```

In fact, I already had a working version implemented on the first iteration, however I was not satisfied with the arguments that I was passing from one statement to the next. I wanted to write *good* code, not just in-efficient, *slap-dash* unprofessional, *student* code.

I already had one solution; to use the query method `order('id desc')` instead of `reverse`, to get the desired highest to lowest ID row ranking. But it seemed a little un-intuitive to be using this query method to order on the primary key when there must be something as intuitively named as *reverse* already? Of course. The query method `reverse_order`⁹⁹. With the mystery solved on why Ruby would throw *no method* exceptions, the statements were becoming cleaner and more efficient.

⁹⁵ <http://api.rubyonrails.org/classes/ActiveRecord/Relation.html>

⁹⁶ chained methods being delimited by periods

⁹⁷ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-reverse>

⁹⁸ <http://api.rubyonrails.org/classes/ActiveRecord/QueryMethods.html>

⁹⁹ http://api.rubyonrails.org/classes/ActiveRecord/QueryMethods.html#method-i-reverse_order

Variable Verbosity

Another valuable improvement in the activity method was improving the argument values to the destroy methods that would remove either the oldest activity or an existing activity for the same contact.

As stated before, the code had been working from the most early revisions. However, one thing that drastically improved was in passing an ID of a record to be destroyed rather than passing an entire `ActiveRecord::Relation` object.

For example, in checking for the presence of an existing contact, the following statement could be used: `previous = history.where(contact_id: contact)`. But this would fetch the relational object with all its associated attributes and values, the destroy method did not need so much verbosity.

I then chained the method with `select(:id)`, that would still return a relational object but with less attributes, skimpier on transactional bandwidth and giving more precise instruction to the destroy method - bringing me closer to my goal.

```
previous = history.where(contact_id: 1).select(:id)
...
SELECT "activities"."id" FROM "activities"
WHERE "activities"."user_id" = 1 AND "activities"."contact_id" = 1
ORDER BY "activities"."id" DESC => [#<Activity id: 22>]
```

At this point I realised that these statements could be further dissected from a relational object into an integer by using methods for arrays such as `map`¹⁰⁰, to extract out a value into an array: `previous = history.where(contact_id: 1).select(:id).map(&:id) giving => [22]`

All that was left to do was chain the method `first`, to extract the first element within the array, enabling me to pass in `previous` with the activity ID to be destroyed. Finally, with regards to the method to destroy the oldest activity entry when the list had got to 10 contacts in size, the solution was comparatively easier.

As all the activity records pertaining to the current user were already obtained by the query that set the result into `history`, this could then be used by chaining the method `last`¹⁰¹, to obtain the oldest record (as the result was ordered at the start) and then chaining `id` - an attribute accessor method, that would return the `id` of the relation e.g. `=> 1`.

3.10 Parties

Parties was the fancy word of choice to represent companies and groups. Implementing parties was not without its own set of problems. However, much of the preliminary setup was straightforward.

¹⁰⁰ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-map-21>

¹⁰¹ <http://www.ruby-doc.org/core-1.9.3/Array.html#method-i-last>

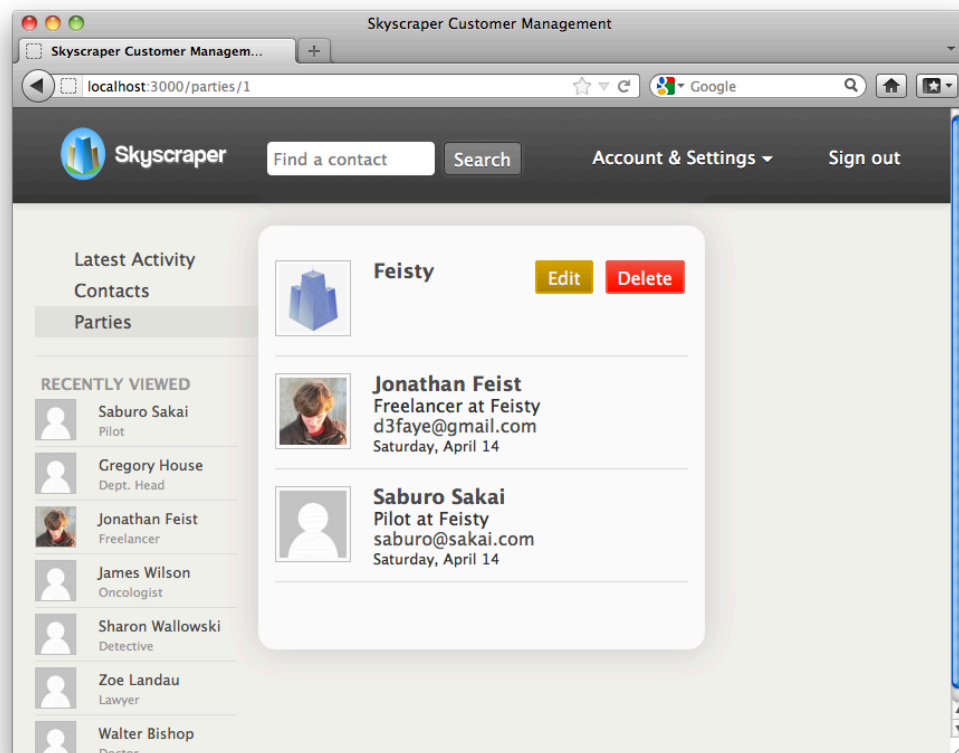
3.10.1 Scaffolding

Parties had the associations that, it belongs to a user and has many contacts. As well as having the need to have pages about parties, I used Rails scaffolding¹⁰² to implement this:

```
rails g scaffold Party name user_id:integer
```

I then set up the `belongs_to` and `has_many` associations in the related models with the *ON DELETE CASCADE*-like relational declaration (`:dependent => :destroy`) with a user.

I also went about setting up some partial template for parties, and set up the show page to render both the party partial template followed by rendering the users associated to that party below:



After having set up the cosmetics of how parties would appear (that would also help me to visually test the implementation) I moved immediately on to solving the first obstacle.

3.10.2 Linking Contacts to Parties by Name

In order to have parties as their own entity within the application, and in order to have the party name listed on a contact's details, I needed to have a method to facilitate the connections between contacts *at creation time* based on their party.

To paraphrase - in order to glue people together by their company name, I needed to make something they share unique (the company). I would do this by converting a name represented

¹⁰² going by reasonable assumption that you can now navigate yourself to relevant files with little assistance

by an ID invisible to the user, but vital in creating the associations.

I would do this by creating a method to mediate between what the user writes and creating a new party record with a unique ID reference number if it did not already exist, and assigning that number back to the contact details, all in one fell-swoop. The method would:

1. Take the name of the party from a contact's details
2. Find the party by name and either return the existing ID or create a new party
3. Return the party ID to be saved in lieu of the company name

The method would be instrumental because:

- It would create a party if it was referenced for the first time, which would mean that users would never have an error thrown back at them about non-existent parties
- It would allow users to associate contacts to parties by their shared company name, because it would never create a new party from the same name, avoiding clashes

Implementation

To do this, I would first define the method taking two parameters, the party name and the main user account ID. The method would search for and return the ID of the party whose name was a match, otherwise creating a new party by that name and returning its ID.

```
1 def self.update_party(party_name,user)
2   party_id = User.find_by_id(user).parties.where \
3     {(lower(name) == lower(party_name))}.select{id}.map(&:id).first
5   party_id = Party.create(name: party_name, user_id: user).id if
party_id.nil?
6   return party_id
7 end
```

The query on line 2 selects all parties of the user by condition that follows onto the next line (separated by backslash tells the parser to continue on the next line as if it were one statement).

The query is refined with the **where** clause to conditionally select the name of the party equal to the party name argument (using downcasing to make it case insensitive).

The following is an example of lines 2-3 prior to the result being mapped (with the user id = 2 and the party name being assigned from the match 'feisty'):

```
SELECT "users".* FROM "users" WHERE "users"."id" = 2 LIMIT 1
SELECT "parties"."id" FROM "parties" WHERE "parties"."user_id" = 2 AND
lower("parties"."name") = lower('feisty')
```

The remaining chained methods would alter the query to select by ID and then as the query is terminated by a chained array method (**map**), the query is formed and executed.

The resulting ID column from the record is mapped out to a new array (which should only be an array with a single element which is extracted by **first**).

Line 5 has a conditional statement to create a new party from the arguments if the assignment of the previous query did not assign a matching ID, assigning the ID from the new party to `party_id`, which is then returned on line 6 as the result of the method.

Refactoring the Method

I found that a single line method would do the same thing as the method defined earlier using the `find_or_create_by`¹⁰³ method.

This would allow me to have a method as simply:

```
1 def self.update_party(party_name,user)
2
3   User.find_by_id(user).parties.find_or_create_by_name(party_name).id
4 end
```

However, the problem with this method is that it was case sensitive, and would fail if trying to change a contact's party to the party *Feisty* if it was given as *feisty*. This was not user friendly and not helpful.

This forced me to rethink and revert back to the old logic, making optimisations there. I began by using ActiveRecord queries instead of relying on Squeel by adjusting the existing *where* clause and using the query method `pluck`¹⁰⁴ to select out the ID to an array, instead of `select{id}.map(&:id)` on line 3:

```
1 def self.update_party(party_name,user)
2   party_id = User.find_by_id(user).parties.where \
3     ('lower(name) = ?', party_name.downcase).pluck(:id).first
4   party_id = Party.create(name: party_name, user_id: user).id if
5     party_id.nil?
6   return party_id
7 end
```

I then realised that the query (example on the previous page) could also be optimised by forming the query directly from the parties table itself, by changing the way it was referenced. Line 2 would become:

```
2 party_id = where(user_id: user).where \
```

The omission of the class (*where* instead of *Party.where*) is allowed, as *self.* is implied by the method declaration *self.update_party* making it a class method. This would optimise the SQL to just *one* query instead of two as:

```
SELECT id FROM "parties" WHERE "parties"."user_id" = 2 AND (lower(name)
= 'feisty')
```

I then went about further simplifying the conditional logic of the method so that the final method

¹⁰³ <http://api.rubyonrails.org/classes/ActiveRecord/Base.html>

¹⁰⁴ <http://api.rubyonrails.org/classes/ActiveRecord/Calculations.html#method-i-pluck>

would be cut down significantly in comparison to the previous revisions:

```
1 def self.update_party(party_name,user)
2   where(user_id: user).where \
3     ('lower(name) = ?', party_name.downcase).pluck(:id).first ||
4     create(name: party_name, user_id: user).id
5 end
```

As all functions in Ruby return something, the default return is either the result of the last statement processed or `nil`. In this case, `return` has been omitted as the result comes out of single (*large*) statement, using the OR operator (the double bar) to return the first statement (from left-to-right) that equates to `true`.

In Ruby, everything equates to `true` unless it is either `false` or `nil`, and both statements either side of the operator will return either an ID number or `nil`). In addition, the use of `return` is tended to be used to forcibly return from a function early, or for the purpose of making the code more declarative).

Hooking Up the Method in the Controller

The next phase of implementation was putting the method to use by having it accept a party name value from the contact form and return the ID of that party to be saved in the user's attribute for the `party_id` (the entire purpose of this endeavour).

I would begin by altering the contact form¹⁰⁵ so that it could print out the party name by its ID if it had been set, using a helper method¹⁰⁶ to so. This would mean that the form would not have a numeric value in the *Company* field that would confuse the user.

The helper method returns the name of the party from its ID:

```
module ContactsHelper
  def party_name(id)
    Party.where(user_id: main_account.id).where(id: id).pluck(:name).first
  end
end
```

This method could then be put to use in the form:

Before:

```
21 .field
22   = f.label :company
23   = f.text_field :company
```

After:

```
21 .field
22   = f.label :party_id, 'Company'
23   = f.text_field :party_id, value: party_name(@contact.party_id)
```

¹⁰⁵ location: `app/views/contacts/_form.html.haml`

¹⁰⁶ location: `app/helpers/contacts_helper.rb`

Finally, the `update_party` method could be put to use in the create and update actions of the contacts controller, to take the value of the name specified from within the form and assign the ID to the contact's `party_id` foreign key:

```
54 def create
55   @contact = Contact.new(params[:contact])
57   party_name = params[:contact][:party_id]
58   @contact.party_id = Party.update_party(party_name,main_account.id)
73 def update
74   @contact = main_account.contacts.find(params[:id]
76   party_name = params[:contact][:party_id]
77   params[:contact][:party_id] =
Party.update_party(party_name,main_account.id)
```

Both of these methods have been summarised to show the relevant statements. One problem that I had with updates caused great confusion as I was trying to assign the `party_id` in the same way as in the `create` action.

However, whereas the `create` action attempts to save the contact with `@contact.save` - the `update` action attempts to update the `@contact` from the parameters with:

```
@contact.update_attributes(params[:contact])
```

I ended up having a peculiar error of `party_id = 0`, where the name had persisted right through to it being assigned to the `party_id` column. Rails handled this by converting it to `0`, which made troubleshooting a little more convoluted for what seemed to be a blundersome error.

3.10.3 Preventing Parties With the Same Name Being Created

A method had been created to convert party names into new parties if it did not exist before, but what about parties being created on their own?

As parties are all stored in the same table and the likelihood unrelated customers of the service having conflicting party names meant that using existing rails validation methods to check for the uniqueness¹⁰⁷ of an attribute would not suffice, as it would return a false positive¹⁰⁸ where a name has not been used on one account, but used on another.

I defined a custom validation method `unique_name` to check for uniqueness only within the scope of each individual account:

```
validate :unique_name
def unique_name
return if self.id && find_by_id(self.id).name.downcase == self.name.downcase
errors.add :base, 'A party with this name already exists.' \
```

¹⁰⁷ the validation method *validates_uniqueness_of* on its own

¹⁰⁸ an error that has been falsely perceived as an error

```
if where(user_id: self.id).where('lower(name) = ?', self.name.downcase).first
  end
```

This method came in a few iterations in which the first line of the method solved a problem in which updates would always return an error as the name would be taken by itself. This first line used the `return` (as previously discussed in its use) would forcibly exit the method if the name was the same as its former name.

This entire method had was made redundant as I soon discovered in the documentation for `validates_uniqueness_of`¹⁰⁹ that there was convention for handling this particular scenario.

I could use the method with a `scope` argument to add additional columns to which the uniqueness pertains, as well as telling Rails whether or not the check is case sensitive with:

```
validates_uniqueness_of :name, scope: :user_id, case_sensitive: false
```

In the end, this made a lot of relatively complex code into literally a few lines in both the creation of a method to run the `update_party` check as well as the uniqueness check, having learned a lot in simplifying, optimising and writing code by convention.

¹⁰⁹ http://api.rubyonrails.org/classes/ActiveRecord/Validations/ClassMethods.html#method-i-validates_uniqueness_of

3.11 Reviewing the Prototype

It had come to a time when development needed to stop so that I could take in what changes or adaptations were necessary to the prototype.

After having a quick review of the system, I had identified several things which needed to be fixed, enhanced or added to the existing prototype:

3.11.1 Enhancements

No Search Results

When using the search functionality of the website and finding no contacts, the result would be an empty page. I decided that this should be changed so that whenever a search is being made:

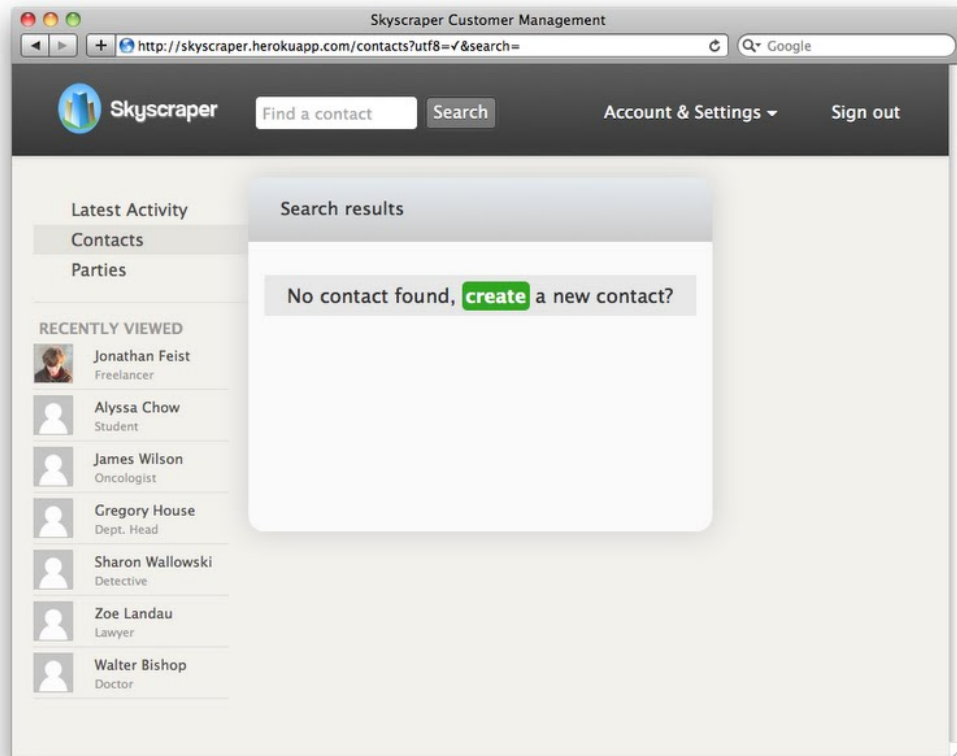
- There is distinction to the user that it is indeed a search and not listing all contacts
- There would be a suitable message if there was no result

The distinction was made by using a conditional statement checking for the presence of `params[:search]` in the index view (this would signify that a search request has been made). From this, the *All Contacts* header and button would change to *Search Results*.

I decided that a suitable message would then inform the user simply: *No contact found, create a new contact?* With the word *create* redirecting the user to the new contact form when clicked.

I had to check for the presence of an actual result using a conditional statement checking for the presence of the first result in the array using `@contacts.first`.

Additionally, this message would be neutrally highlighted with the create link being highlighted in the same green colour scheme as the regular creation buttons. The result would be an improvement on there being a completely blank content for no result and better for usability.



Preventing Page Re-Alignment

One minor niggling issue that at first seemed to be a styling fault was little more than the page content being shifted whenever the vertical page content exceeding the available window space. This occurred most often on the contacts page (as it fetches over 15 results by default), with all other pages quite often fitting within the available window space. This was resolved by adding the CSS rule to the `body: overflow-y: scroll` to forcibly show a scrollbar at all times, and prevent the page appearing to shift left a centimeter when many contacts were viewed.

3.11.2 Fixing Bugs

Preventing Blank Notes

In preventing users from maliciously saving notes to the wrong account (by editing the DOM)¹¹⁰, I had created an exception-friendly redirect that was not reliant on the note path being correct. The side-effect of this was that users who push ‘add to notes’ legitimately in error (without any note content) would fail the validation that a note must not be blank¹¹¹.

The redirect would be inconvenient in such circumstances, as it would take the user away from the contact that they were viewing, rather than the page being redisplayed. This meant changing the redirection on failure to save a note back to its original statement:

```
redirect_to contact_path(@note.contact_id), notice: 'Invalid note.'
```

The knock-on effect of this resolved genuine user errors (getting redirected back to the contact to which they posted the form from) but still gave a nasty exception on dirty notes that needed to be resolved in a different way.

Using the `rescue_from`¹¹² method in the application controller (so that all controllers would inherit this capability, notes included), I could capture all exceptions and redirect them in one statement:

```
rescue_from Exception, with: lambda { redirect_to :portal, notice: 'Invalid Request.' }
```

This statement would capture the `NoMethodError` exception and redirect it back to the main page of the website, with the *Invalid Request* notice. It needed to have a method specified as to how it would handle the exception, so I used `lambda` to define an anonymous function without the need to define a method of its own.

¹¹⁰ Document Object Model

¹¹¹ this validation was achieved using `validates_presence_of:body`

¹¹² <http://api.rubyonrails.org/classes/ActiveSupport/Rescuable/ClassMethods.html>

Re-Routing Bad Paths

As a consequence of coming to a solution on the previous issue, I was then lead on to solve another related issue with routing going wrong when given a path to a resource that does not exist. For instance, giving a route to a resource

E.g. `http://application/route-that-does-not-exist`

Rails would return: Routing Error: No route matches [GET] `"/route-that-does-not-exist"`

TVD (2011) suggested using a routing rule: `match '*a' => 'errors#routing'` that is a wildcard that matches on any path, and is put after the match rules for all other resources, in order of least priority.

The order is important, and is demonstrated by giving it a higher priority than the portal URL and refreshing the main page, ending up with a browser exception such as *'Firefox has detected that the server is redirecting the request for this address in a way that will never complete.'*

The rule would point to a new *errors* controller with the action *routing*. In that action, I defined a method call to `render_404`. The call to this method would be placed in the application controller. The contents of the method would be similar to the lambda expressed before, that I would change to point to the same method.

I would also use TVD's suggestion to output exception information to the log whenever they occur, so that the valuable information the developer should see and the user should not is accessible from the log file.

```
def render_404(exception = nil)
  logger.info "Exception, redirecting: #{exception.message}" if
exception
  redirect_to :portal, notice: 'Invalid Request.'
end
```

In testing this with the dirty notes problem (by trying to save a note to a contact that belongs to another user), the re-direction was working, but not by virtue of catching an exception.

This was because the validation method was failing validation and trying to return to the notes form with the errors from `errors.add :base, 'Invalid request.'`. This was simply failing the `if @note.save` condition in the notes controller - then trying to redirect to the contact with the wrong ID. Rails would then make a fresh GET request for the contact resource e.g. `"http://application/contacts/1"` to which the user does not have ownership, and in the query finding no contacts thereby assigning `nil` to the `@contact` variable¹¹³.

¹¹³ as is the custom of the `#find_by*` query method, as opposed to the `#find` method, which would return an exception `ActiveRecord::RecordNotFound` for no result found (<http://api.rubyonrails.org/classes/ActiveRecord/RecordNotFound.html>)

It would be redirected one last time to the contacts index. All without giving any notice, but quite an adequate trail¹¹⁴ for trying to deceive the system.

As great as it is to punish the user by forcing them to wait some additional milliseconds for trying to grief another user of the service, I felt it necessary to patch this by altering the the `belongs_to_user` validation method in the notes model.

This was resolved by simply forcing the validation method to raise an exception¹¹⁵ that would trigger the `rescue_from Exception` clause in the application controller, redirecting back to the portal page with the notice: *Invalid Request*. as well as outputting a suitable log message.

The effect of which in testing the previous issue with dirty notes could be visibly seen¹¹⁶ to be effective by seeing both the database transaction rollback¹¹⁷ and the exception being output to the log.

Diff of the changes to the custom validation method in the notes model `app/models/note.rb`

		<code>def belongs_to_user</code>
14	14	<code>id = User.find_by_id(self.user_id).user_id self.user_id</code>
15		<code>- errors.add :base, 'Invalid request.' \</code>
	15	<code>+ raise Exception.new("Invalid Note Request: User: #{id}.") \</code>
16	16	<code>unless Contact.find_by_id(self.contact_id).user_id == id</code>
17	17	<code>end</code>

¹¹⁴ the log file for this request can be viewed at <https://gist.github.com/2409441>

¹¹⁵ <http://api.rubyonrails.org/classes/ActiveSupport/BasicObject.html#method-i-raise>

¹¹⁶ the log file for this request raising an exception can be viewed at <https://gist.github.com/2409516>

¹¹⁷ a database transaction that is rolled back is undone, so that changes do not take effect.

Recursive Delete on Sub-Users

Recursively deleting associated users of a main account would be very useful for an administrator who is closing an overall business account, as it leads to less human error in judgement.

The problem was that at the moment, deleting users was a privilege of the admin only and it would be better to extend this to the account owner to relinquish some administrator responsibility.

This was achieved by adding adjusting the destroy action of the users controller with some conditional statements as well as using the `:dependent => :destroy` option on the user association in the user model `has_many :users`

The adjusted conditional statement prior to permitting user deletion:

```
if !same_user?(@user) && sub_user?(@user) && main_account? || admin?  
  @user.destroy
```

This would check that the user making the request is from an owner account or from an administrator that is not trying to self-destruct and is only trying to destroy sub-users of the account.

Existing Notes and Last User Edits

The problem that would come up from testing the deletion of users was that any last user edits to a note¹¹⁸ or indeed notes belonging to a sub-user who was being deleted would either be destroyed (by the `ON DELETE CASCADE` behaviour) or cause exceptions from non-existent parent records pertaining to their foreign key references (with last user edits).

This was resolved by creating a function to be called `before_destroy`¹¹⁹, such that, the method specified is invoked before the actual record of the user is deleted. This is the moment in which all note edits can be erased and notes belonging to a sub-user can be passed on to the account owner, rather than being deleted.

¹¹⁸ the implementation of last user edits has been omitted for brevity, the purpose of last user edits was to show any other user who had made the last update to a note

¹¹⁹ <http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html>

The actual method implemented to perform this task had drastically changed over time. To start off with, it was passing the user ID and its owner ID to a class method in the note model to perform the note operations.

```
before_destroy { Note.relinquish!(self.id, self.user_id) if  
self.user_id }
```

Then it was entering a broker method that would invoke the two separate processes to operate on the notes in a recursive fashion, but running multiple SQL queries to fetch the first matching note, updating that note and then either fetching the next or terminating and returning to the broker method (self.relinquish!), there - it would do the same thing for the remaining recursive method to update the notes matching on the second criteria.

```
def self.relinquish!(user,main)  
  null_last_user_id!(user,main)  
  pass_user_id!(user,main)  
end  
  
def self.null_last_user_id!(user,main)  
  if note = where(last_user_id: user).first  
    note.update_attributes(last_user_id: nil)  
    null_last_user_id!(user,main)  
  else  
    return  
  end  
end  
  
def self.pass_user_id!(user,main)  
  if note = where(user_id: user).first  
    note.update_attributes(user_id: main)  
    pass_user_id!(user,main)  
  else  
    return  
  end  
end  
end
```

Simplifying the Method and Optimising the Queries

This was later improved to a single function that would fetch all queries being operated on and make separate queries to update them that would only require an $N+1$ amount of queries (where N equates to the number of records to update) compared to $N*2$ in the recursive invocation style.

The second style as below with $N+1$ number of queries to update, iterates over each result making separate queries to update, compared to the former style of query to both fetch and update:

```
def self.relinquish!(user,main)
  where(last_user_id: user).each{ |n| n.update_attributes(last_user_id: nil)
}
  where(user_id: user).each{ |n| n.update_attributes(user_id: main) }
end
```

The method was then further simplified by being extracted out of the note model and into the user model. There it would become a part of its own block function of `before_destroy`:

```
before_destroy do |record|
  if record.user_id
    Note.where(last_user_id: record.id).each do |n|
      n.update_attributes(last_user_id: nil)
    end
    Note.where(user_id: record.id).each do |n|
      n.update_attributes(user_id: record.user_id)
    end
  end
end
```

Optimising the Query One Last Time

Mulling over the SQL, I came across a query method `update_all`¹²⁰. This method would further optimise the queries to just ONE query per operation!

```
before_destroy do |record|
  if record.user_id
    Note.where(last_user_id: record.id).update_all(last_user_id: nil)
    Note.where(user_id: record.id).update_all(user_id: record.user_id)
  end
end
```

The resulting SQL would be one to update the `last_user_id` of any matching notes to `nil` and one to set the `user_id` of any matching notes to the account owner's ID e.g:

```
UPDATE "notes" SET "last_user_id" = NULL WHERE "notes"."last_user_id" = 7
UPDATE "notes" SET "user_id" = 2 WHERE "notes"."user_id" = 7
```

¹²⁰ http://apidock.com/rails/ActiveRecord/Relation/update_all

Listing All Notes In Latest Activity

The current notes listing worked by having Rails automate a query through the associations (notes belong to users through contacts) where:

```
@notes = main_account.notes.order('created_at DESC')
```

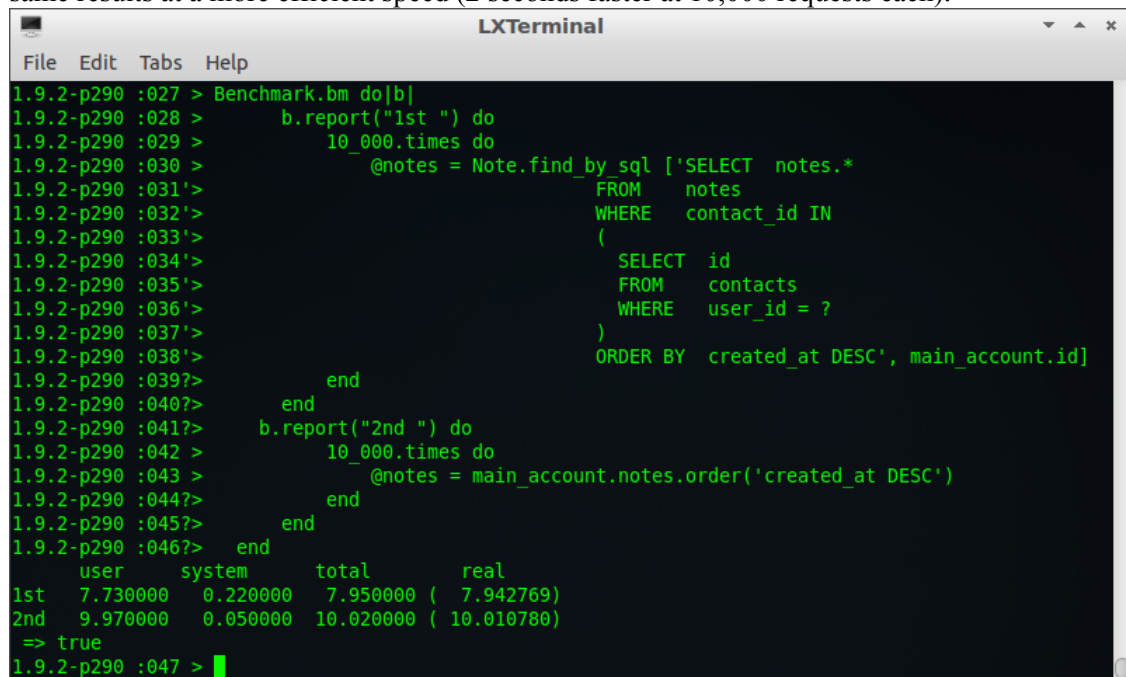
This was working fine, but I was not satisfied with the query it was building:

```
SELECT "notes".* FROM "notes" INNER JOIN "contacts" ON "notes"."contact_id" = "contacts"."id" WHERE "contacts"."user_id" = 2 ORDER BY created_at DESC
```

So, using a method `find_by_sql`¹²¹ I wrote my own version of the query to knock out the middlemen (to reduce the pressure on Rails to figure out the appropriate query to run):

```
@notes = Note.find_by_sql ['SELECT  notes.*
                           FROM    notes
                           WHERE   contact_id IN
                           (
                             SELECT id
                             FROM    contacts
                             WHERE   user_id = ?
                           )
                           ORDER BY  created_at DESC', main_account.id]
```

I ran this under benchmarking tests to compare it against the previous version and it gave the same results at a more efficient speed (2 seconds faster at 10,000 requests each):



```
LXTerminal
File Edit Tabs Help
1.9.2-p290 :027 > Benchmark.bm do|b|
1.9.2-p290 :028 >   b.report("1st ") do
1.9.2-p290 :029 >     10_000.times do
1.9.2-p290 :030 >       @notes = Note.find_by_sql ['SELECT  notes.*
1.9.2-p290 :031 >                                FROM    notes
1.9.2-p290 :032 >                                WHERE   contact_id IN
1.9.2-p290 :033 >                                (
1.9.2-p290 :034 >                                  SELECT id
1.9.2-p290 :035 >                                  FROM    contacts
1.9.2-p290 :036 >                                  WHERE   user_id = ?
1.9.2-p290 :037 >                                )
1.9.2-p290 :038 >                                ORDER BY  created_at DESC', main_account.id]
1.9.2-p290 :039 >     end
1.9.2-p290 :040 >   end
1.9.2-p290 :041 >   b.report("2nd ") do
1.9.2-p290 :042 >     10_000.times do
1.9.2-p290 :043 >       @notes = main_account.notes.order('created_at DESC')
1.9.2-p290 :044 >     end
1.9.2-p290 :045 >   end
1.9.2-p290 :046 > end
   user      system    total      real
1st  7.730000    0.220000    7.950000 ( 7.942769)
2nd  9.970000    0.050000   10.020000 ( 10.010780)
=> true
1.9.2-p290 :047 > █
```

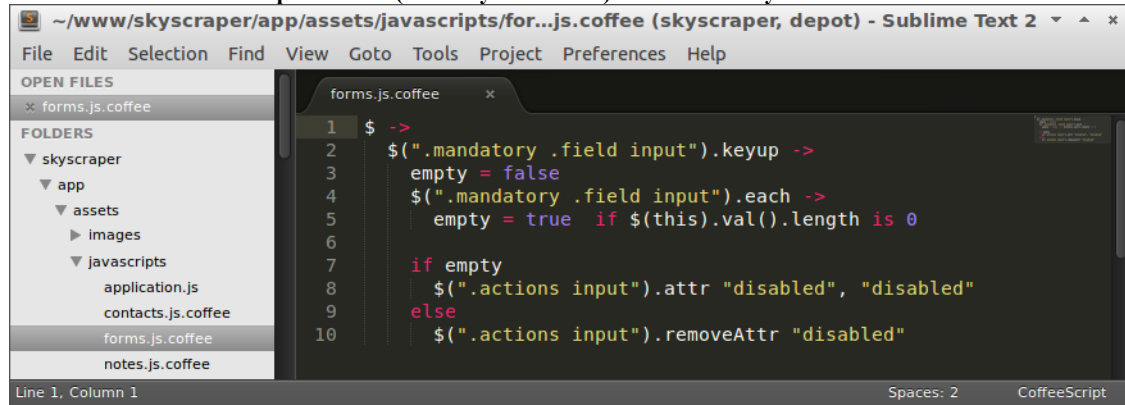
3.11.3 Javascript Enhancements

Preventing Eager Form Submission

¹²¹ http://apidock.com/rails/ActiveRecord/Base/find_by_sql/class

This enhancement focused on preventing early submission of a form before mandatory fields had been filled in, that would be used in multiple areas of the application for different types of forms. I put it to use on the users form and the party creation form.

Screenshot of the Javascript function (from my text editor) to conditionally disable form submission



The Javascript functions are deployed within the `app/assets/javascripts` directory. In this directory the `application.js` file which is included in the layout imports all other Javascript files within the same directory (as well as jQuery that each Javascript enhancement used).

Rails 3.2.1 comes preconfigured with CoffeeScript¹²², which is a Javascript pre-processor that helps to take a lot of the complexity out of native Javascript code. Never the less, this does not make it any easier without a seasoned understanding of the native code.

I designed the function to apply a function to all inputs of the a form immediately preceded in the DOM hierarchy by divs with a class="field" and their parent divs with a class="mandatory".

The function would be fired by `keyup`, that is triggered when a keystroke has finished. The benefit of which, as opposed to an `onchange` event, is that `onchange` fires upon leaving an input whereas `keyup` fires upon immediate change has occurred.

In many cases this would leave the user with a disabled submit button after the final input, having to click the cursor out in order for it to trigger. Within the same function, a boolean variable `empty` is set to false that is conditionally checked later on. Each input is checked for emptiness, to which the boolean `empty` is set to true if so.

At the end of the sequence of statements within the function triggered by the `keyup` event, the boolean is checked for truth. If it is empty, the submit button is disabled, and if it is not empty, the disabled attribute that is applied is removed, allowing it to be clicked again.

A couple more adjustments involved adding the `mandatory` parent class to any form fields and applying an initial disablement of the submit button through HTML on those forms to prevent them being posted before any keystrokes had been triggered in any inputs.

¹²² <http://coffeescript.org/>

The effect of this functionality would reduce the number of forms being submitted without their mandatory fields being filled with some data, that would also help to reduce overall usage service's bandwidth.

Highlighting Links By the Current Page

This was a rather minor enhancement to notice from a user's standpoint but would greatly improve navigational awareness of a user within the application, as it would highlight any link on the left side bar to which section the user was currently browsing.

Current Link Highlight Function within `app/assets/javascripts/sky.js.coffee`:

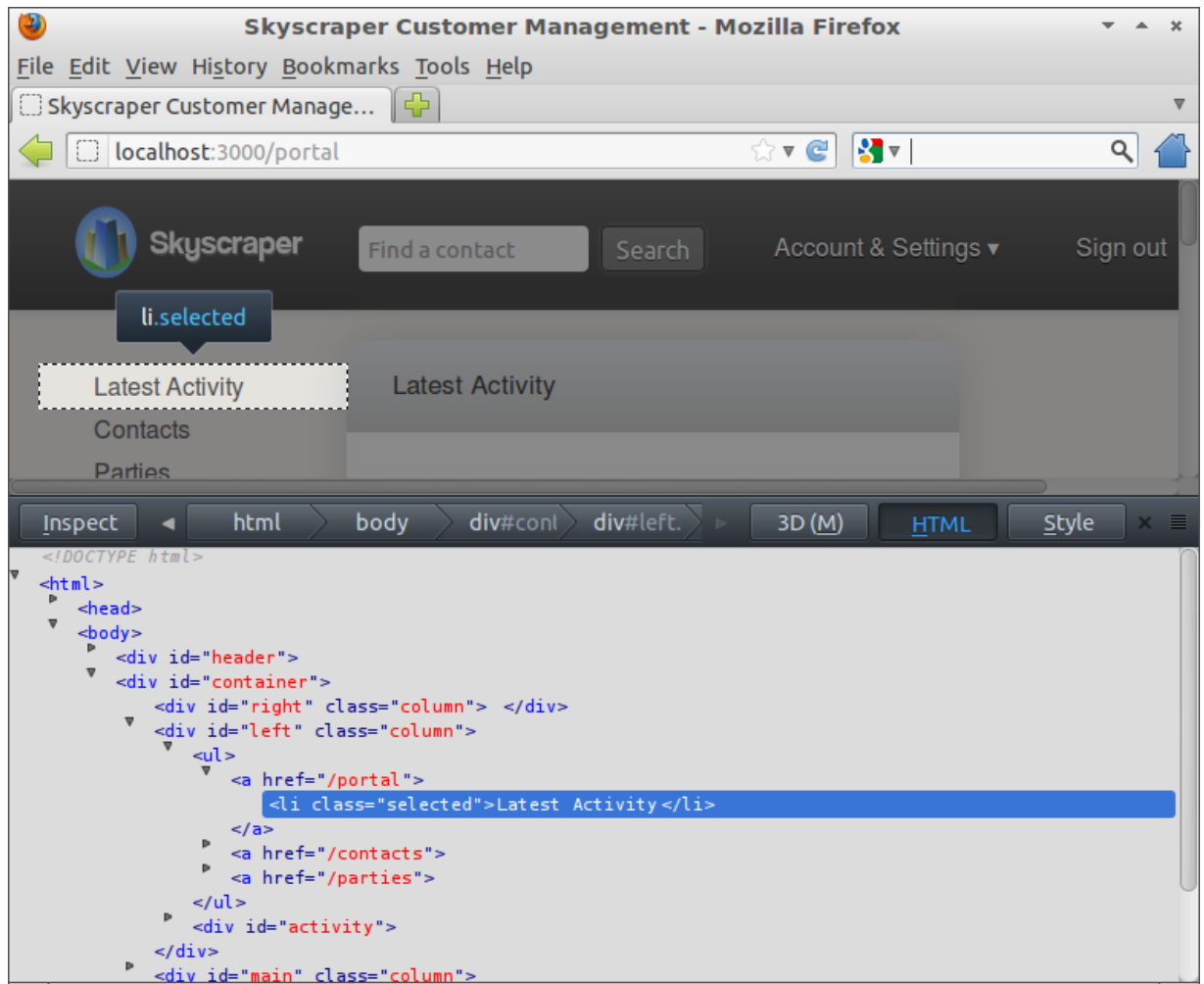
```
$ ->
  pathname = window.location.pathname.split("/")
  $(".selected").removeClass "selected"
  $('#left > ul > a[href$="/" + pathname[1] + '"] > li').addClass "selected"
```

This function would coincide with the addition of a new CSS class “selected” to apply a grey background colour to the link to which the function would select as the page currently being browsed.

It would be triggered when the page was loaded and considered *ready* by the DOM, to which it would obtain the path of the current resource from `window.location.pathname` (that would be equal to “./portal” when on the “http://application/portal” page).

This path would be split into an array on the forward slash (into say [“.”, “portal”]) that would then come in handy for the jQuery selection that would build up the correct path to select the list item for the “selected” class to be applied to. The clever part of this method was that it could correctly identify the node within the DOM from the actual hyperlink matching to the path itself.

The following screenshot shows the path selected: `#left > ul > a[href$="/portal"] > li`



URL Hover Highlight Script

This script¹²³ would work in conjunction with the previous script to lightly highlight any URL on the left navigation except the link to which section the user was currently browsing.

This was achieved by selecting the correct node within the DOM and applying a function that would be triggered by the entry and exit of a mouse cursor over the links.

¹²³ location: app/assets/javascripts/sky.js.coffee

For each list element within the navigation, the function it would apply or remove a background colour so long as it did not have the class “selected” applied. This would have the effect sought after to highlight only the links that are not being browsed.

```
$ ->
$( "#left > ul > a" ).children( "li" ).each ->
  unless $(this).hasClass( "selected" )
    $(this).mouseover ->
      $(this).css "background-Color", "#FAF8F3"
    $(this).mouseout ->
      $(this).css "background-Color", ""
```

This function was similarly used on the recently viewed contacts sidebar as well.

4. Conclusion

4.1 Discussion & Evaluation

Ruby on Rails is rapidly changing, and with it comes new ways of solving old problems. For a newcomer such as myself, the change in concepts can be overwhelming at first. Joel Spolsky (2002) _the co-founder of Stackoverflow_ coins it as “Fire and Motion”. We must always react to changes, but be wary of becoming overwhelmed.

The coarse grained features of a standard Rails application support CRUD¹²⁴ functions very well. But I can attest to the opinion Wayne M (2011) has, that Rails is deceptively simple. I set out in my deliverables that I would put aesthetics secondary in priority to the overall functionality, but I found that a web application in its early stages of development (that I consider Skyscraper to be in still) heavily relies on the styling to be done well, to compensate for the lack of fuller features. But this was also a personal source of obsession, as I felt that the application should have a professional appearance or none at all. Making sure that things were displaying correctly and beautifully therefore rose up on my list of priorities and I continually worked and reworked the styling and structure.

Testing of the application had been done on-the-fly through Rails console (the Rails interpreter equivalent to Ruby’s IRB (Interactive Ruby), that loads the current environment of the application). Through the console, I could evaluate queries and statements, inspect objects, control inputs and see the outcomes directly and dynamically. I could also use the console to benchmark statements to test the efficiency of code.

Due to the time constraints of the project, I had not taken testing much farther than this, however I had continuously ran my own (live) integration testing by posing as a customer and using the system to its limits. Without doing so I would not have discovered security holes in the hidden form field `contact_id` for notes submission that _without coming up with a custom validation against_ would have left customers potentially dumbfounded by notes to which were not created by any user associated to their account (having been passed from one to another).

¹²⁴ Create, Read, Update, Delete

The extent of this problem could have lead to customers being given a `NoMethodFound` exception in the rare cases in which this malicious use of the application in tandem with the deletion of the user from which it was sent (i.e. the user was then deleted by its account owner), would have left Rails without an ID to reference for the note's username - giving an exception.

Again, without having done integration testing first hand, I would not have discovered routing exceptions from requests for paths to resources that do not exist, which was subsequently dealt with by using a wildcard at the lowest priority on path name matching to reroute to the 404 method - the same method that would be responsible for all exceptions and routing errors that would redirect the user to the main page.

If I had time to write tests for the application, I would have done so in the same style that I learned through the *Depot*¹²⁵ application, in which I learned how to write *fixtures* that resemble *things* in the system to be used for testing (such as items or users); *functional* tests - that assert the system follows prescribed behaviour and *unit* tests - to assert the measurable accuracy of the application in its methods. The automated equivalent to my own live *integration* testing can be seen within the `test/integration/user_stories_test.rb` file of the *Depot* application on Github (URL accessible via footnote 124). In it, I learned how to write an integration test that assumes the role of a customer making an order within the online store.

So, whilst the application was not tested in automation, the means by which I had been testing was through active and thorough use of the application after every change, and by systematically testing other loose ends of the system to which a change may have had effect (such as through associations of models). The drawbacks of this are very clear _not every aspect of the application can be foreseen to be adversely affected by change_ however knowing what and how to test is a task and skill unto itself.

One reason for the omission of prescribed testing is due to the fact that, as a beginner in the Rails web framework and the Ruby programming language, I was continually bombarded by unexpected results, behaviour and failure of my methods. I still consider *Skyscraper* to be in a very raw but ready phase, and may be at a pivotal moment to begin writing automated tests for. For the sole reason that everything in the application was drastically changing a lot of the time, and allocating time to writing testing procedures was not the wisest thing to do *in my opinion*, simply because I was still getting to grips with the framework and had not truly gotten onto the significantly challenging problems.

I believe that the time I spent actively testing the system myself including the need to repeat certain steps at some points (such as re-registration of a sub-user and its notes for the purpose of testing the `on_destroy` user function), was both as valuable as having written automated tests to do the same thing, but would have resulted in a greater loss of time if I had wrote tests for the system at this stage in the game.

One of the earliest difficulties of the application came before it began. Using the Rails framework and building an application out of it needed sufficient time and energy spent on learning the groundwork. Fortunately I had written an application by example to get to grip with using Rails as well as Distributed Version Control Systems (DVCS) with Git as the

¹²⁵ testing of the depot application (<https://github.com/defaye/Depot/tree/master/test>)

recommended solution for version control with the project. Both went very well and proved to be invaluable in giving me the edge necessary to begin my project in earnest.

In hindsight I felt that I perhaps spent a little too long learning by example and not enough time in doing exploratory coding myself. But I believe it has had both benefits and drawbacks. Time not spent on the actual project meant that I reduced the time I could spend on its full development. However, sticking to the tutorial long enough may have also taught me some useful conventions on dealing with the MVC in Rails.

If I had not spent enough time on learning by example, I may have had written a lot more spaghetti code¹²⁶. In fact, the majority of my functions would lead to many opportunities for refactorization and discovery of better suited methods and conventions for dealing with the problems that I faced in which many examples had come up in this report.

As I wrote more and more queries through Rails, I became less attached to using the Squeel gem. At one point in development I benchmarked an equivalent query in ActiveRecord against Squeel, and found it quite inferior in performance. This should have come as no surprise probably just adding another layer of abstraction to ActiveRecord itself, but it was also a wake-up call not to treat gems as *treasure* or believe that they are “magic” (Netshade, 2011) or as good as they are touted to be. In the end I phased out the use of Squeel for all but the most difficult queries to write, and only after I had spent much time testing alternatives, and learning new methods and conventions along the way.

I am pleased with what I have created and believe that it is a project that has many loose ends that can be attended to and expanded upon. I am most pleased with the number of methods that I had written having genuinely and literally evolved from very poor efficiency to very high efficiency (especially with the user on `_destroy` method for notes).

I am also pleased that my application is seemingly very stable; I do not know of any ways to break it from within the interface, as any user.

It is secure. Accounts are in isolation from one another (and protected from strange bugs that the most curious of customers might find with regards to dirty notes). Account owners and administrators alike cannot delete themselves. Account owners can manage their own sub-users. Deletion of users is secure, and cascades without any traces left behind. Deleting sub-users who created their own notes are not lost but passed to the owner. Perhaps I could go on...My point is I feel that I have covered a great many areas of vulnerabilities and have thought meticulously of the knock-on effects of the methods I implement. For example, the `on_destroy` method only triggers if the user is a sub-user, as deleting an account owner would delete everything anyway.

I have to confess that I did not find much use from creating a page flow diagram or taking an initial guess at the application data. My reason is that the design of the application revolved around one minor and one major layout, with the content always yielding within the center of the layout. For this reason I came to the decision to axe spending time on a page flow diagram as it would only tell me what I already knew - I had envisioned the layout at an early stage

¹²⁶ highly coupled code as opposed to loosely coupled code, in which program components work with greater interdependence, exposing less of their inner workings and have more defined purpose

and this became a process of abstracting an idea into boxes in HTML with CSS. If I were to conceptualise this at a level in that detail, I might as well do it on the fly. The same goes for the application data, within the Rails application, it was painfully easy to build up in iterations whenever new application data was needed; I could either add a migration to adjust an existing table, or destroy the database, edit the existing create table statements, migrate and re-seed the database with my dummy data. See Appendix 41 to see how I had implemented the seeds, run with the command `rake db:seed`.

One regret however is not considering planning the database with an ERD. However, the only major issue in considering the database structure only came to the surface when considering the implementation detail of having a range of criteria for contacts and users. See Appendix 53 - Database Schema Redesign for the overall plan that did not make it to the end product.

Due to having spent over a month learning about Rails without having written a single line of code towards my actual project, I did not achieve some of the objectives in my proposal. I cover these in more detail in the next chapter, however it is important to say that this had a profound impact on me and was a valuable lesson to learn.

4.2 Future Work

The application has come a long way, and yet did not achieve all of the deliverables I had set out to. Of those deliverables, those that had not been implemented:

- Import and export of contacts
- Customisable attributes for contacts - i.e. specifying your own details
- Expanding on the basic attributes of contacts to have:
 - A flexible number of extra attributes, more than one email for example
- An application programming interface (API) to query the application remotely:
 - An example API client to connect to the API interface

Although not all deliverables were attempted, I had been in the planning stages of taking the application much further in terms of expanding the basic attributes of a contact. By using nested attributes to help with having more than one occurrence of an attribute. I.e. it would be possible to have as many email addresses saved to a contact as the user desires, by using nested forms, AJAX, and specialised database tables for those criteria.

It would have involved some reorganisation of the database schema and I would have set it up such that users, contacts and parties inherit attributes from a details table. The details table would itself contain foreign key references to tables dedicated to storing specific criteria such as emails, addresses, phone numbers, websites, and so on.

These changes would have enriched the current application into a truly flexible and useful customer address book.

Other features that could have been extended upon were notes to have comments, as well as tasks, as miniature to-do lists for users where they can set tasks within time frames and check them out as they are completed or overdue.

Some other basic functionality that would have been required to tout the application as a self-standing service would have been to verify accounts by email, including the registration of sub-users via invitation rather than being declared by an account owner.

Skyscraper has many possibilities to be taken further. But it seems likely that I may away the experience gleaned from this project to more professionally crafting its incarnation. For my first Rails application, a clean sheet would do wonders for a fresh iteration of a new prototype.

References

- Beck, Kent; et al. (2001). *Manifesto for Agile Software Development*. Available: <http://agilemanifesto.org/>. Last accessed January 22, 2012.
- Berners-Lee, T; et al. (1994). *Uniform Resource Locators (URL)*. Available: <http://tools.ietf.org/html/rfc1738>. Last accessed 9th April
- Bigg, Ryan. (2010). *Ubuntu, Ruby, RVM, Rails and You*. Available: <http://ryanbigg.com/2010/12/ubuntu-ruby-rvm-rails-and-you/>. Last accessed 5th April 2012.
- Bourne, Stephen. (2009). *What is Bash?*. Available: http://www.gnu.org/software/bash/manual/html_node/What-is-Bash_003f.html#What-is-Bash_003f. Last accessed 5th April 2012.
- Cockburn, Alistair. (1993). *Unraveling incremental development*. Available: <http://alistair.cockburn.us/Unraveling+incremental+development>. Last accessed 22 January 2012.
- Collins-Sussman, B., Fitzpatrick, B.W. & Pilato, C.M. (2011). *Version Control With Subversion*. Available: <http://svnbook.red-bean.com/en/1.7/svn.intro.whatis.html#svn.intro.history>. Last accessed 5th April 2012.
- Fowler, Martin. (2005a). *The Unpredictability Of Requirements*. <http://martinfowler.com/articles/newMethodology.html#TheUnpredictabilityOfRequirements> Last accessed 22 January 2012.
- Fowler, Martin. (2005b). *Controlling an Unpredictable Process – Iterations*. <http://martinfowler.com/articles/newMethodology.html#ControllingAnUnpredictableProcess-Iterations> Last accessed 22 January 2012.
- Github. (2012a). *Set Up Git*. Available: <http://help.github.com/linux-set-up-git/>. Last accessed 4th April 2012.
- Github. (2012b). *Create A Repo*. Available: <http://help.github.com/create-a-repo/>. Last accessed 4th April 2012.
- H, Andrew. (2007). *What is the strongest algorithm?*. Available: <http://www.kellermansoftware.com/t-articlestrongesthash.aspx>. Last accessed 7th April 2012.
- Hansson. (2008). *Rails is moving from SVN to Git*. Available: <http://weblog.rubyonrails.org/2008/4/2/rails-is-moving-from-svn-to-git/>. Last accessed 5th April 2012.
- Netshade. (2011). *Ruby on Rails downsides and caveats*. Available: <http://programmers.stackexchange.com/a/105102/51406>. Last accessed 19th April 2012.
- JD. notifications-support@heroku.com. [Heroku Support] Re: This is a really simple question. Does Heroku require the config/database.yml file to be p... (ticket #48588). 11th April 2012.
- Ladd, Seth; et al. (2006). *Convention over configuration*. Available: <http://>

static.springsource.org/spring/docs/2.0.x/reference/mvc.html. Last accessed 22nd January 2012.

Levine, Matthew. (2006). *In Search of the Holy Grail*. Available: <http://www.alistapart.com/articles/holygrail/>. Last accessed 1st April 2012.

Loeliger, J, 2009. *Version Control With Git*. 1st ed. Sebastopol: O'Reilly

Moura, Ruda. (2010). *apt-get*. Available: <http://linux.die.net/man/8/apt-get>. Last accessed 5th April 2012.

Oracle. (n.d.). *What are regular expressions?*. Available: <http://docs.oracle.com/javase/tutorial/essential/regex/intro.html>. Last accessed 8th April 2012.

Pack, Roger D. (2010). *Installing Ruby*. Available: http://en.wikibooks.org/wiki/Ruby_Programming/Installing_Ruby#Windows_is_slow. Last accessed 5th April 2012.

Robins, S. (2012). *Is the phrase "never reinvent the wheel" suitable for students?*. Available: <http://programmers.stackexchange.com/a/143882/51406>. Last accessed 12th April 2012.

Rosu, Catalin. (2011). *CSS3 dropdown menu*. Available: <http://www.red-team-design.com/css3-dropdown-menu>. Last accessed 12th April 2012.

Ruby, Sam; et al (2011). *Agile Web Development with Rails*. 4th ed. Dallas: Pragmatic Programmers. p.xvii-xix.

Seguin, Wayne E. (2011). *Ruby Version Manager (RVM)*. Available: <https://rvm.beginrescueend.com/>. Last accessed 5th April 2012.

Shirey, R. (2007). *Internet Security Glossary, Version 2*. Available: <http://tools.ietf.org/html/rfc4949>. Last accessed 7th April 2012.

Softweb Solutions. (2010). *Effective Reasons for the Popularity of Ruby on Rails Development*. Available: <http://ruby-on-rails-web-development.blogspot.com/2010/12/3-effective-reasons-for-popularity-of.html>. Last accessed 22 January 2012.

Spolsky, Joel. (2002). *Fire And Motion*. Available: <http://www.joelonsoftware.com/articles/fog0000000339.html>. Last accessed 19th April 2012.

Tilkov, Stefan. (2007). *A Brief Introduction to REST*. Available: <http://www.infoq.com/articles/rest-introduction>. Last accessed 11th April 2012.

TVD. (2011). *Rails 3.0 Rescue From Routing Error Solution*. Available: <http://techoctave.com/c7/posts/36-rails-3-0-rescue-from-routing-error-solution>. Last accessed 17th April 2012.

W3C. (2012). *The div Element*. Available: <http://www.w3.org/TR/html5/the-div-element.html#the-div-element>. Last accessed 7th April 2012.

Wayne M. (2011). *Ruby on Rails downsides and caveats*. Available: <http://programmers.stackexchange.com/a/104424/51406>. Last accessed 16th April 2012.

Bibliography

Chacon, Scott. (2009). What a Branch Is. In: Parkes, D.; Taylor, T.; Berry, L. *Pro Git*. New York: Apress. p38-39.

Kyrnin, Jennifer. (n.d.). *What does !important mean in CSS?*. Available: <http://webdesign.about.com/od/css/f/blcssfaqimportn.htm>. Last accessed 11th April 2012.

Pate. (2008). *Benchmarking Makes it Better*. Available: <http://on-ruby.blogspot.co.uk/2008/12/benchmarking-makes-it-better.html>. Last accessed 9th April 2012.

Wald, Ragan. (2007). *Closures and Higher-Order Functions*. Available: <http://weblog.raganwald.com/2007/01/closures-and-higher-order-functions.html>. Last accessed 17th April 2012.

Appendices

Appendix 1 - User Model

```
class User < ActiveRecord::Base
  acts_as_authentic
  #acts_as :detail
  has_many :contacts, dependent: :destroy
  has_many :parties, dependent: :destroy
  has_many :activities, dependent: :destroy
  has_many :notes, through: :contacts
  has_many :users, dependent: :destroy
  validates_presence_of :first_name, :last_name, :username, :email
  before_destroy do |record|
    if record.user_id
      Note.where(last_user_id: record.id).update_all(last_user_id: nil)
      Note.where(user_id: record.id).update_all(user_id: record.user_id)
    end
  end
end
```

Appendix 2 - Party Model

```
class Party < ActiveRecord::Base
  #acts_as :detail
  has_many :contacts, dependent: :destroy
  belongs_to :user
  validates_uniqueness_of :name, scope: :user_id, case_sensitive: false
  validates_presence_of :name
  auto_strip_attributes :name, squish: true

  private
  def self.update_party(party_name,user)
    where(user_id: user).where('lower(name) = ?', party_name.downcase).pluck(:id).first ||
    create(name: party_name, user_id: user).id
  end
end
```

Appendix 3 - Note Model

```
class Note < ActiveRecord::Base
  belongs_to :contact
  validates_presence_of :body
  validate :belongs_to_user, on: :create

  private
  def belongs_to_user
    id = User.find_by_id(self.user_id).user_id || self.user_id
    raise Exception.new("Invalid Note Request: User: #{id}.") \
    unless Contact.find_by_id(self.contact_id).user_id == id
  end
end
```

Appendix 4 - Activity Model

```
class Activity < ActiveRecord::Base
  belongs_to :user
  belongs_to :contact

  private
```

```

        def self.update_activity(user, contact)
          if history = where(user_id: user).reverse_order
            if previous = history.where(contact_id: contact).select(:id).map(&:id).first
              destroy(previous)
            elsif history.count > 9
              destroy(history.last.id)
            end
          end
          new(user_id: user, contact_id: contact).save
        end
      end
    end
  end
end

```

Appendix 5 - Contact Model

```

class Contact < ActiveRecord::Base
  #acts_as :detail
  belongs_to :user
  belongs_to :party
  has_many :notes, dependent: :destroy
  has_many :activities, dependent: :destroy
  validates_presence_of :first_name, :last_name
  validates :email,
    email_format: { message: 'does not look like an email address.' },
    allow_blank: true

  def name
    "#{first_name} #{last_name}"
  end

  def self.search(search)
    if search
      search = search.split
      search.reject!{ |s| s.length == 1 }
      search.map!{ |s| "%#{s}%" }
      if search.length > 1
        where{(first_name.like_any search)&(last_name.like_any search)}
      else
        where{(first_name.like_any search)|(last_name.like_any search)|(email.like_any search)}
      end
    else
      find(:all)
      self.order('last_name, first_name ASC')
    end
  end
end

```

Appendix 6 - Application Controller

```
class ApplicationController < ActionController::Base
  protect_from_forgery
  rescue_from Exception, with: :render_404

  helper_method :current_user, :current_user_session, :admin?, :resolve_layout,
    :main_account?, :main_account, :redirect_back_or_default, :store_location

  private

    def current_user_session
      return @current_user_session if defined?(@current_user_session)
      @current_user_session = UserSession.find
    end

    def current_user
      return @current_user if defined?(@current_user)
      @current_user = current_user_session && current_user_session.record
    end

    def require_user
      unless current_user
        store_location
        flash[:notice] = "You must be logged in to access this page"
        redirect_to :login
        return false
      end
    end

    def require_no_user
      if current_user
        store_location
        flash[:notice] = "You must be logged out to access this page"
        redirect_to portal_url
        return false
      end
    end

    def store_location
      session[:return_to] = request.url
    end

    def redirect_back_or_default(default)
      redirect_to(session[:return_to] || default)
      session[:return_to] = nil
    end

    def admin?
      current_user && current_user.id == 1
    end

    def main_account?
      current_user && current_user.user_id.nil?
    end

    def main_account
      if main_account?
        return current_user
      else
        User.find(current_user.user_id)
      end
    end

    def resolve_layout
      if main_account?
        return "application"
      end
    end
end
```



```

        end
        case action_name
        when "new", "create"
          "splash"
        else
          "application"
        end
      end
    end

    def render_404(exception = nil)
      logger.info "Exception, redirecting: #{exception.message}" if exception
      redirect_to :portal, notice: 'Invalid Request.'
    end
  end
end

```

Appendix 7 - Contacts Controller

```

class ContactsController < ApplicationController
  before_filter :require_user
  layout "application"

  # GET /contacts
  # GET /contacts.json
  def index
    @contacts = main_account.contacts.search(params[:search]).paginate(page: params[:page],
per_page: 15)
    @parties = main_account.parties.all.inject({}) {|h, obj| h[obj.id] = obj.name; h }

    respond_to do |format|
      format.html # index.html.haml
      format.json { render json: main_account.contacts }
    end
  end

  # GET /contacts/1
  # GET /contacts/1.json
  def show
    @contact = main_account.contacts.find_by_id(params[:id])
    return redirect_to contacts_url unless @contact
    @party = @contact.party || nil

    Activity.update_activity(current_user.id, @contact.id)

    @notes = @contact.notes.order('created_at desc')
    @note = Note.new

    respond_to do |format|
      format.html # show.html.haml
      format.json { render json: @contact }
    end
  end

  # GET /contacts/new
  # GET /contacts/new.json
  def new
    @contact = Contact.new

    respond_to do |format|
      format.html # new.html.haml
      format.json { render json: @contact }
    end
  end
end

```

```

end

# GET /contacts/1/edit
def edit
  @contact = main_account.contacts.find(params[:id])
end

# POST /contacts
# POST /contacts.json
def create
  @contact = Contact.new(params[:contact])
  @contact.user_id = main_account.id
  party_name = params[:contact][:party_id]
  @contact.party_id = Party.update_party(party_name,main_account.id)

  respond_to do |format|
    if @contact.save
      format.html { redirect_to @contact, notice: 'Contact added.' }
      format.json { render json: @contact, status: :created, location: @contact }
    else
      format.html { render action: "new" }
      format.json { render json: @contact.errors, status: :unprocessable_entity }
    end
  end
end

# PUT /contacts/1
# PUT /contacts/1.json
def update
  @contact = main_account.contacts.find(params[:id])
  params[:contact][:user_id] = main_account.id
  party_name = params[:contact][:party_id]
  params[:contact][:party_id] = Party.update_party(party_name,main_account.id)

  respond_to do |format|
    if @contact.update_attributes(params[:contact])
      format.html { redirect_to @contact, notice: 'Contact updated.' }
      format.json { head :no_content }
    else
      format.html { render action: "edit" }
      format.json { render json: @contact.errors, status: :unprocessable_entity }
    end
  end
end

# DELETE /contacts/1
# DELETE /contacts/1.json
def destroy
  @contact = main_account.contacts.find(params[:id])
  @contact.destroy

  respond_to do |format|
    format.html { redirect_to contacts_url }
    format.json { head :no_content }
  end
end

# def export
#   require 'csv'

```

```

# render format.csv {
#   contacts = main_account.contacts
#   columns = ['id','first_name','last_name','email','title','company'].to_csv
#   filename = "contacts-#{Date.today.to_s(:db)}"

#   self.response.headers["Content-Type"] ||= 'text/csv'
#   self.response.headers["Content-Disposition"] = "attachment; filename=#{@filename}"
#   self.response.headers["Content-Transfer-Encoding"] = "binary"

#   self.response_body = Enumerator.new do |y|
#     contacts.each_with_index do |c, i|
#       if i == 0
#         y << @columns
#       end
#       y << [c.id, c.first_name, c.last_name, c.email, c.title, party_name(c.party_id)].to_csv
#     end
#   end
# }
# end

# def export
#   require 'csv'
#   contacts = main_account.contacts
#   filename = "contacts_#{DateTime.now.to_i}.csv"
#   csv_data = CSV.generate do |csv|
#     csv << ['id','first_name','last_name','email','title','company']
#     contacts.each do |c|
#       csv << [ c.id, c.first_name, c.last_name, c.email, c.title, party_name(c.party_id)]
#     end
#   end
#   send_data contacts.export(params[:url_type]), \
#     :type => 'text/csv; charset=iso-8859-1; header=present', \
#     :disposition => "attachment; filename=#{filename}.csv"
# end
end

```

Appendix 8 - Notes Controller

```

class NotesController < ApplicationController
  before_filter :require_user
  layout "application"

  # GET /notes
  # GET /notes.json
  def index
    @notes = main_account.notes.order('created_at DESC')

    respond_to do |format|
      format.html # index.html.html
      format.json { render json: @notes }
    end
  end

  # GET /notes/1/edit
  def edit
    @note = main_account.notes.find(params[:id])
  end
end

```

```

# POST /notes
# POST /notes.json
def create
  @note = Note.new(params[:note])
  @note.user_id = current_user.id

  respond_to do |format|
    if @note.save
      format.html { redirect_to contact_path(@note.contact_id), notice: 'Note saved.' }
      format.json { render json: @note, status: :created, location: @note }
    else
      format.html { redirect_to contact_path(@note.contact_id), notice: 'Invalid note.' }
      format.json { render json: @note.errors, status: :unprocessable_entity }
    end
  end
end

# PUT /notes/1
# PUT /notes/1.json
def update
  @note = main_account.notes.find(params[:id])
  @note.last_user_id = current_user.id

  respond_to do |format|
    if @note.update_attribute(:body, params[:note][:body])
      format.html { redirect_to contact_path(@note.contact_id), notice: 'The note was
updated.' }
      format.json { head :no_content }
    else
      format.html { render action: "edit" }
      format.json { render json: @note.errors, status: :unprocessable_entity }
    end
  end
end

# GET /notes/1
# GET /notes/1.json
def show
  @note = main_account.notes.find(params[:id])
  @contact = Contact.find_by_id(@note.contact_id)

  respond_to do |format|
    format.html
    format.json { render json: @note }
  end
end

# DELETE /notes/1
# DELETE /notes/1.json
def destroy
  @note = main_account.notes.find(params[:id])
  @note.destroy

  respond_to do |format|
    format.html { redirect_to contact_path(@note.contact_id) }
    format.json { head :no_content }
  end
end
end
end

```

Appendix 9 - Parties Controller

```
class PartiesController < ApplicationController
  before_filter :require_user
  layout "application"
  # GET /parties
  # GET /parties.json
  def index
    @parties = main_account.parties.paginate \
      page: params[:page], order: 'name, created_at asc', per_page: 15

    respond_to do |format|
      format.html # index.html.erb
      format.json { render json: @parties }
    end
  end

  # GET /parties/1
  # GET /parties/1.json
  def show
    @party = main_account.parties.find(params[:id])
    @contacts = main_account.contacts.where(party_id: params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.json { render json: @party }
    end
  end

  # GET /parties/new
  # GET /parties/new.json
  def new
    @party = Party.new

    respond_to do |format|
      format.html # new.html.erb
      format.json { render json: @party }
    end
  end

  # GET /parties/1/edit
  def edit
    @party = main_account.parties.find(params[:id])
  end

  # POST /parties
  # POST /parties.json
  def create
    @party = Party.new(params[:party])
    @party.user_id = main_account.id

    respond_to do |format|
      if @party.save
        format.html { redirect_to @party, notice: 'Party added.' }
        format.json { render json: @party, status: :created, location: @party }
      else
        format.html { render action: "new" }
      end
    end
  end
end
```

```

        format.json { render json: @party.errors, status: :unprocessable_entity }
      end
    end
  end

  # PUT /parties/1
  # PUT /parties/1.json
  def update
    @party = Party.find(params[:id])

    respond_to do |format|
      if @party.update_attributes(params[:party])
        format.html { redirect_to @party, notice: 'Party updated.' }
        format.json { head :no_content }
      else
        format.html { render action: "edit" }
        format.json { render json: @party.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /parties/1
  # DELETE /parties/1.json
  def destroy
    @party = Party.find(params[:id])
    @party.destroy

    respond_to do |format|
      format.html { redirect_to parties_url }
      format.json { head :no_content }
    end
  end
end
end

```

Appendix 10 - Sky Controller

```

class SkyController < ApplicationController
  before_filter :require_user
  def index
    @notes = Note.find_by_sql ['SELECT notes.*
                                FROM notes
                                WHERE contact_id IN
                                (
                                  SELECT id
                                  FROM contacts
                                  WHERE user_id = ?
                                )
                                ORDER BY created_at DESC', main_account.id]
  end
end

```

Appendix 11 - Users Controller

```

class UsersController < ApplicationController
  before_filter :require_user, only: [:show, :edit, :update]
  layout :resolve_layout
end

```

```

# GET /users
# GET /users.json
def index
  if admin?
    @users = User.all order: 'created_at ASC'
  else
    owner_id = main_account.id
    @users = User.where{(user_id == owner_id) | (id == owner_id)}.order('created_at ASC')
  end

  respond_to do |format|
    format.html # index.html.haml
    format.json { render json: @users }
  end
end

# GET /users/1
# GET /users/1.json
def show
  if admin?
    @user = User.find(params[:id])
  else
    @user = User.find(current_user.id)
  end
  respond_to do |format|
    format.html # show.html.haml
    format.json { render json: @user }
  end
end

# GET /users/new
# GET /users/new.json
def new
  if current_user_session.nil? || main_account?
    @user = User.new

    respond_to do |format|
      format.html # new.html.haml
      format.json { render json: @user }
    end
  else
    flash[:notice] = 'Must be logged into main account'
    redirect_to portal_url
  end
end

# GET /users/1/edit
def edit
  if sub_user?(User.find_by_id(params[:id])) || admin?
    @user = User.find(params[:id])
  else
    @user = User.find(current_user.id)
  end
end

# POST /users
# POST /users.json
def create
  @user = User.new(params[:user])
  @user.user_id = current_user.id if main_account?
end

```

```

        respond_to do |format|
          if @user.save
            format.html { redirect_to portal_url, notice: 'Registration successful.' }
            format.json { render json: @user, status: :created, location: @user }
          else
            format.html { render action: "new" }
            format.json { render json: @user.errors, status: :unprocessable_entity }
          end
        end
      end

      # PUT /users/1
      # PUT /users/1.json
      def update
        @user = User.find(params[:id])
        if same_user?(@user) || sub_user?(@user) || admin?
          respond_to do |format|
            if @user.update_attributes(params[:user])
              format.html { redirect_to portal_url, notice: 'Account successfully updated.' }
              format.json { head :no_content }
            else
              format.html { render action: "edit" }
              format.json { render json: @user.errors, status: :unprocessable_entity }
            end
          end
        else
          redirect_to portal_url
        end
      end

    end

    # DELETE /users/1
    # DELETE /users/1.json
    def destroy
      @user = User.find(params[:id])
      respond_to do |format|
        if !same_user?(@user) && sub_user?(@user) && main_account? || admin?
          @user.destroy

          format.html { redirect_to users_url, notice: 'User deleted.' }
          format.json { head :no_content }
        else
          format.html { redirect_to users_url, notice: 'This user cannot be deleted.' }
          format.json { head :no_content }
        end
      end
    end

    private

    def sub_user?(user)
      user.user_id == current_user.id
    end

    def same_user?(user)
      user.id == current_user.id
    end
  end
end

```

Appendix 12 - Application Helper


```

module ApplicationHelper
  def gravatar(email,size)
    "http://www.gravatar.com/avatar/#{Digest::MD5.hexdigest(email.downcase)}.jpg?s=#{size}&d=mm"
  end

  def last_viewed_contacts
    ids = current_user.activities.pluck(:contact_id).reverse
    contacts = Contact.find_all_by_id(ids).index_by(&:id)
    contacts = ids.collect {|id| contacts[id] }
  end
end

```

Appendix 13 - Contacts Helper

```

module ContactsHelper
  def party_name(id)
    Party.where(user_id: main_account.id).where(id: id).pluck(:name).first
  end
end

```

Appendix 14 - Activities Partial

```

#activity
.header RECENTLY VIEWED
%table
  - last_viewed_contacts.each do |contact|
    %tr.activity
    %td.avatar= link_to image_tag(gravatar(contact.email, 35)), contact
    %td
    = link_to(contact) do
      .name= contact.name
      .title= contact.title
    end
  end

```

Appendix 15 - Contact Brief Partial

```

.contact
  .delete
    = button_to 'Delete', @contact, confirm: 'Really delete this contact?', method: :delete
  .edit
    = button_to 'Edit', edit_contact_path(@contact), method: :get
  .gravatar
    = link_to image_tag(gravatar(@contact.email, 70)), @contact
  .name= link_to @contact.name, @contact
  .details
    = @contact.title
    = 'at' unless @contact.title.blank? || @contact.party_id.nil?
    = 'from' if @contact.title.blank? && @contact.party_id
    = link_to @party.name, @party if @party

```

Appendix 16 - Contact Partial

```

.contact
  .gravatar
    = link_to image_tag(gravatar(contact.email, 60)), contact
  .name= link_to contact.name, contact
  .details
    = contact.title
    = 'at' unless contact.title.blank? || contact.party_id.nil?

```

```

= 'from' if contact.title.blank? && contact.party_id
= @parties[contact.party_id] if contact.party_id
%br= mail_to contact.email
.updated_at= contact.updated_at.to_datetime.to_formatted_s(:day_month)

```

Appendix 17 - Contact Index

```

#top
  .title
    - unless params[:search]
      All Contacts
      .create= button_to 'Add Contact', new_contact_path, method: :get
      / .export= button_to 'Export', :export, method: :get
    - else
      Search results
#bottom
  %p#notice
    = notice
  = render @contacts
  - unless @contacts.first
    #result
    No contact found,
    = link_to 'create', new_contact_path
    a new contact?
  %br= will_paginate @contacts

```

Appendix 18 - Contact Show

```

#content
  %p#notice= notice
  -if @note.errors.any?
    #error_explanation
    %h2= "#{pluralize(@note.errors.count, "error")} prevented the note being saved:"
    %ul
      - @note.errors.full_messages.each do |msg|
        %li= msg
  = render 'brief'
  = render 'notes/form'
  = render @notes

```

Appendix 19 - Contact Form Partial

```

= form_for @contact do |f|
  -if @contact.errors.any?
    #error_explanation
    %h2= "#{pluralize(@contact.errors.count, "error")} prevented this contact being saved:"
    %ul
      - @contact.errors.full_messages.each do |msg|
        %li= msg

    .field
      = f.label :first_name
      = f.text_field :first_name
    .field
      = f.label :last_name
      = f.text_field :last_name
    .field

```

```

        = f.label :email
        = f.text_field :email
    .field
        = f.label :title
        = f.text_field :title
    .field
        = f.label :party_id, 'Company'
        = f.text_field :party_id, value: party_name(@contact.party_id)
    .actions
        = f.submit 'Save'

```

Appendix 20 - Application Layout

```

!!!
%html
  %head
    %title Skyscraper Customer Management
    = stylesheet_link_tag "application", :media => "all"
    = javascript_include_tag "application"
    = yield :head
    = csrf_meta_tags
  %body
    #header
    #logo
    = link_to image_tag("logo_s.png"), portal_path
    #search
    = form_tag contacts_path, method: :get do
    = text_field_tag :search, params[:search], placeholder: 'Find a contact'
    = submit_tag "Search", name: nil
    #top_menu
    = render 'navigation/top_menu'
    #container
    #right.column
    = yield :right
    #left.column
    = render 'navigation/side_menu'
    = render 'activities/activities'
    #main.column
    = yield
    #footer

```

Appendix 21 - Splash Layout

```

!!!
%html
  %head
    %title Skyscraper Customer Management
    = stylesheet_link_tag "application", :media => "all"
    = javascript_include_tag "application"
    = yield :head
    = csrf_meta_tags
  %body
    #centered
    #logo
    = link_to image_tag('logo.png'), :login
    #splash
    = yield

```

```
#footer
```

Appendix 22 - Side Navigation

```
%ul
  = link_to(portal_path) do
    %li Latest Activity
  = link_to(contacts_path) do
    %li Contacts
  = link_to(parties_path) do
    %li Parties
```

Appendix 23 - Top Navigation

```
- content_for :head do
  = stylesheet_link_tag 'optionals/dropdown'
%ul#menu
  %li
    = link_to 'Account & Settings ▾', '#'
    %ul
    %li
      - if admin?
      = link_to 'My account', edit_user_path(current_user)
      - else
      = link_to 'My account', :account
    %li= link_to 'Users', users_path
  %li= link_to 'Sign out', :logout
```

Appendix 24 - Note Form Partial

```
#note_form
  = form_for @note do |f|
    -if @note.errors.any?
      #error_explanation
      %h2= "#{pluralize(@note.errors.count, "error")} prevented this note being saved:"
      %ul
      - @note.errors.full_messages.each do |msg|
        %li= msg

      .field
      - unless @contact.nil?
      = f.label :body, "Add a note about #{@contact.first_name}"
      .help
      You might want to save a note about a conversation you have just had with this contact.
      Notes are great when it comes to keeping the history of your communications with
    contacts.
    = f.text_area :body, rows: 4
    - else
    = f.text_area :body, rows: 8
    = f.hidden_field :contact_id, value: params[:id]
    .actions
    - unless @contact.nil?
    = f.submit 'Add this note'
    - else
    = f.submit 'Update'
  .bottom
```

Appendix 25 - Note Partial

```
.note
.updated_at= note.updated_at.to_datetime.to_formatted_s(:day_month)
.options
  .edit_text= link_to 'Edit', edit_note_path(note), method: :get
  .cancel_text= link_to 'Delete', note, confirm: 'Are you sure?', method: :delete
.by_user= "Note by #{User.find_by_id(note.user_id).username}."
- if note.last_user_id && note.last_user_id != note.user_id
  .last_edit= "Last edited by #{User.find_by_id(note.last_user_id).username}."
.body= note.body
```

Appendix 26 - Note Edit

```
- content_for :head do
  = stylesheet_link_tag 'notes'
#top
  .title
    Edit
    = link_to "Note", @note
    by
    = User.find_by_id(@note.user_id).username
    .back= button_to 'Back', contact_path(@note.contact_id), method: :get
#bottom
  %p#notice
    = notice
  = render 'form'
```

Appendix 27 - Note Index

```
#top
  .title
    All Notes
#bottom
  %p#notice
    = notice

  = render @notes
```

Appendix 28 - Note Show

```
#top
  .title
    = link_to "Note", @note
    about
    = link_to "#{@contact.first_name} #{@contact.last_name}", @contact
    by
    = User.find_by_id(@note.user_id).username
#bottom
  %p#notice
    = notice
  = render @note

  .back= button_to 'Back', @contact, method: :get
```

Appendix 29 - Party Contact Partial

```

- @contacts.each do |contact|
  .contact
    .delete
    = button_to 'Delete', contact, confirm: 'Really delete this contact?', method: :delete
    .edit
    = button_to 'Edit', edit_contact_path(contact), method: :get
    .gravatar
    = link_to image_tag(gravatar(contact.email, 60)), contact
    .name= link_to contact.name, contact
    .details
    = contact.title
    %br= mail_to contact.email
    .updated_at= contact.updated_at.to_datetime.to_formatted_s(:day_month)

```

Appendix 30 - Party Partial

```

.party
- if params[:id]
  .delete
  = button_to 'Delete', party, confirm: 'Delete party and all associated contacts?',
method: :delete
  .edit
  = button_to 'Edit', edit_party_path(party), method: :get
  .image
  = link_to image_tag('party_60.jpg'), party
  .name= link_to party.name, party
  .details

```

Appendix 31 - Party Index

```

#top
  .title
    All Parties
    .create= button_to 'Add Party', new_party_path, method: :get
#bottom
  %p#notice
    = notice
  #parties
    = render @parties

  %br= will_paginate @parties

```

Appendix 32 - Party Show

```

#content
  %p#notice= notice

  = render @party
  = render 'parties/contact', object: @contacts

```

Appendix 33 - Sky Notes Activity Partial

```

- @notes.each do |note|
  .note
    - contact = Contact.find_by_id(note.contact_id)
    .focus= link_to "#{contact.first_name} #{contact.last_name}", contact
    .month_day= note.updated_at.to_datetime.to_formatted_s(:month_day)

```

```

.by_user= "Note by #{User.find(note.user_id).username}."
- if note.last_user_id && note.last_user_id != note.user_id
.last_edit= "Last edited by #{User.find_by_id(note.last_user_id).username}."
.body
.hover= link_to truncate(note.body, length: 256, separator: ' '), note

```

Appendix 34 - Sky Index

```

#top
.title Latest Activity
#bottom
%p#notice
  = notice
#notes_activity
  = render 'sky/notes_activity', object: @notes

```

Appendix 35 - User Session Form Partial

```

= form_for @user_session do |f|
  -if @user_session.errors.any?
    #error_explanation
    %h2= "#{pluralize(@user_session.errors.count, "error")} prevented authentication:"
    %ul
      - @user_session.errors.full_messages.each do |msg|
        %li= msg
  .mandatory
    .field
      = f.label :username
      = f.text_field :username
    .field
      = f.label :password
      = f.password_field :password
  .actions
    = f.submit 'Login'
    or
= button_to 'Sign up', :signup

```

Appendix 36 - User Form Partial

```

= form_for @user do |f|
  -if @user.errors.any?
    #error_explanation
    %h2= "#{pluralize(@user.errors.count, "error")} prevented registration:"
    %ul
      - @user.errors.full_messages.each do |msg|
        %li= msg
  .mandatory
    .please= current_user_session ? nil : 'Please fill out all fields:'
    .field
      = f.label :first_name
      = f.text_field :first_name
    .field
      = f.label :last_name
      = f.text_field :last_name
    .field
      = f.label :username
      = f.text_field :username

```

```

        .field
        = f.label :email
        = f.text_field :email
        .field
        = f.label :password
        = f.password_field :password
        .field
        = f.label 'Confirm'
        = f.password_field :password_confirmation
    .actions
        = f.submit 'Submit', disabled: current_user_session ? nil : 'disabled'

```

Appendix 37 - User _User_ Partial

```

.user
- if current_user.id != user.id && main_account?
    .delete
    = button_to 'Delete', user, confirm: 'Really delete this user?', method: :delete
    .edit
    = button_to 'Edit', edit_user_path(user), method: :get
.gravatar
    = image_tag(gravatar(user.email, 60))
.name= "#{user.first_name} #{user.last_name}"
- if main_account?
    .username
    Username:
    = user.username
.email= mail_to user.email
.role= "Account owner" if user.user_id.nil? unless user.id == 1

```

Appendix 38 - User Show

```

#top
    .title
        Account Details
        .edit= button_to 'Edit', edit_user_path(@user), method: :get
#bottom
    %p#notice
        = notice
    #show
        .user
        .gravatar
        = image_tag gravatar(@user.email,60)
        .name
        = "#{@user.first_name} #{@user.last_name}"
        %br= mail_to @user.email
        .details
        Username:
        = @user.username
        .description
        Edit your details to change your name, email, username or password.

```

Appendix 39 - Time Formats


```
Time::DATE_FORMATS[:day_month] = "%A, %B %e"  
Time::DATE_FORMATS[:month_day] = "%b %e"
```

Appendix 40 - Database Schema

```
ActiveRecord::Schema.define(:version => 20120403120512) do
```

```
  create_table "activities", :force => true do |t|  
    t.integer "user_id",      :null => false  
    t.integer "contact_id",   :null => false  
  end
```

```
  create_table "addresses", :force => true do |t|  
    t.string "address"  
    t.string "city"  
    t.string "state"  
    t.string "postcode"  
    t.string "country"  
    t.string "type",          :null => false  
    t.integer "detail_id",    :null => false  
    t.datetime "created_at",  :null => false  
    t.datetime "updated_at",  :null => false  
  end
```

```
  create_table "contacts", :force => true do |t|  
    t.string "first_name", :null => false  
    t.string "last_name",  :null => false  
    t.string "email",      :null => false  
    t.string "title"  
    t.integer "party_id"  
    t.integer "user_id",   :null => false  
    t.datetime "created_at", :null => false  
    t.datetime "updated_at", :null => false  
  end
```

```
  create_table "details", :force => true do |t|  
    t.integer "as_detail_id"  
    t.string "as_detail_type"  
    t.datetime "created_at", :null => false  
    t.datetime "updated_at", :null => false  
  end
```

```
  create_table "emails", :force => true do |t|  
    t.string "address", :null => false  
    t.string "type",    :null => false  
    t.integer "detail_id", :null => false  
    t.datetime "created_at", :null => false  
    t.datetime "updated_at", :null => false  
  end
```

```
  create_table "ims", :force => true do |t|  
    t.string "address", :null => false  
    t.string "service", :null => false  
    t.string "type",    :null => false  
    t.integer "detail_id", :null => false  
    t.datetime "created_at", :null => false  
    t.datetime "updated_at", :null => false  
  end
```

```

create_table "notes", :force => true do |t|
  t.integer "contact_id", :null => false
  t.integer "user_id", :null => false
  t.text "body", :null => false
  t.integer "last_user_id"
  t.datetime "created_at", :null => false
  t.datetime "updated_at", :null => false
end

create_table "parties", :force => true do |t|
  t.string "name", :null => false
  t.integer "user_id", :null => false
  t.datetime "created_at", :null => false
  t.datetime "updated_at", :null => false
end

create_table "phones", :force => true do |t|
  t.string "number", :null => false
  t.string "type", :null => false
  t.integer "detail_id", :null => false
  t.datetime "created_at", :null => false
  t.datetime "updated_at", :null => false
end

create_table "users", :force => true do |t|
  t.string "first_name", :null => false
  t.string "last_name", :null => false
  t.string "username", :null => false
  t.string "email", :null => false
  t.string "crypteed_password", :null => false
  t.string "password_salt", :null => false
  t.string "persistence_token", :null => false
  t.integer "login_count", :default => 0, :null => false
  t.integer "failed_login_count", :default => 0, :null => false
  t.datetime "last_request_at"
  t.datetime "current_login_at"
  t.datetime "last_login_at"
  t.string "current_login_ip"
  t.string "last_login_ip"
  t.integer "user_id"
  t.datetime "created_at", :null => false
  t.datetime "updated_at", :null => false
end

create_table "websites", :force => true do |t|
  t.string "url", :null => false
  t.string "type", :null => false
  t.integer "detail_id", :null => false
  t.datetime "created_at", :null => false
  t.datetime "updated_at", :null => false
end
end

```

Appendix 41 - Seeds

```

User.delete_all
User.create(first_name: 'Jonathan', last_name: 'Feist', username: 'Defaye',

```

```

email: 'd3faye@gmail.com', password: 'secr3t', password_confirmation: 'secr3t')
User.create(first_name: 'Saburo', last_name: 'Sakai', username: 'Sakai',
email: 'saburo@sakai.com', password: 'password', password_confirmation: 'password')
User.create(first_name: 'Tetsuo', last_name: 'Iwamoto', username: 'Tetsuo',
email: 'tetsuo@sakai.com', password: 'password', password_confirmation: 'password', user_id: 2)
User.create(first_name: 'Hiroyoshi', last_name: 'Nishizawa', username: 'Hiroyoshi',
email: 'Hiroyoshi@sakai.com', password: 'password', password_confirmation: 'password', user_id:
2)

Party.delete_all
Party.create(name: 'Feisty', user_id: 1)
Party.create(name: 'Feisty', user_id: 2)
Party.create(name: 'PPTH', user_id: 1)
Party.create(name: 'PPTH', user_id: 2)
Party.create(name: 'Lightman Group', user_id: 1)
Party.create(name: 'Lightman Group', user_id: 2)
Party.create(name: 'Fringe', user_id: 1)
Party.create(name: 'Fringe', user_id: 2)

Contact.delete_all
Contact.create(
  first_name: 'Jonathan',
  last_name: 'Feist',
  email: 'd3faye@gmail.com',
  title: 'Freelancer',
  party_id: 1,
  user_id: 1
)

Contact.create(
  first_name: 'Jonathan',
  last_name: 'Feist',
  email: 'd3faye@gmail.com',
  title: 'Freelancer',
  party_id: 2,
  user_id: 2
)

Contact.create(first_name: 'Cal', last_name: 'Lightman', email: 'cal@lightmangroup.com',
title: 'Owner', party_id: 5, user_id: 1)
Contact.create(first_name: 'Gillian', last_name: 'Foster', email: 'gillian@lightmangroup.com',
title: 'Colleague', party_id: 5, user_id: 1)
Contact.create(first_name: 'Walter', last_name: 'Bishop', email: 'w.bishop@dfringe.com',
title: 'Doctor', party_id: 7, user_id: 1)
Contact.create(first_name: 'Emily', last_name: 'Lightman', email: 'emily@lightmangroup.com',
title: 'Family', party_id: 5, user_id: 1)
Contact.create(first_name: 'Zoe', last_name: 'Landau', email: 'zoe@lightmangroup.com',
title: 'Lawyer', party_id: 5, user_id: 1)
Contact.create(first_name: 'Sharon', last_name: 'Wallowski', email: 'sharon@lightmangroup.com',
title: 'Detective', party_id: 5, user_id: 1)
Contact.create(first_name: 'Karl', last_name: 'Dupree', email: 'karl@lightmangroup.com',
title: 'Agent', party_id: 5, user_id: 1)
Contact.create(first_name: 'Peter', last_name: 'Bishop', email: 'p.bishop@dfringe.com',
title: 'Doctor', party_id: 7, user_id: 1)
Contact.create(first_name: 'Gregory', last_name: 'House', email: 'house@house.com', title: 'Dept.
Head', party_id: 3, user_id: 1)
Contact.create(first_name: 'Lisa', last_name: 'Cuddy', email: 'cuddy@house.com', title: 'Dean',
party_id: 3, user_id: 1)
Contact.create(first_name: 'James', last_name: 'Wilson', email: 'wilson@house.com',
title: 'Oncologist', party_id: 3, user_id: 1)

```

```

Contact.create(first_name: 'Eric', last_name: 'Foreman', email: 'foreman@house.com',
title: 'Neurologist', party_id: 3, user_id: 1)
Contact.create(first_name: 'Olivia', last_name: 'Dunham', email: 'dunham@df fringe.com',
title: 'Agent', party_id: 7, user_id: 1)
Contact.create(first_name: 'Allison', last_name: 'Cameron', email: 'cameron@house.com',
title: 'Immunologist', party_id: 3, user_id: 1)
Contact.create(first_name: 'Robert', last_name: 'Chase', email: 'chase@house.com',
title: 'Surgeon', party_id: 3, user_id: 1)
Contact.create(first_name: 'Eli', last_name: 'Loker', email: 'eli@lightmangroup.com',
title: 'Employee', party_id: 5, user_id: 1)
Contact.create(first_name: 'Ria', last_name: 'Torres', email: 'ria@lightmangroup.com',
title: 'Employee', party_id: 5, user_id: 1)
Contact.create(first_name: 'Lawrence', last_name: 'Kutner', email: 'kutner@house.com',
title: 'Sports Medicine', party_id: 3, user_id: 1)
Contact.create(first_name: 'Chris', last_name: 'Taub', email: 'taub@house.com', title: 'Plastic
Surgeon', party_id: 3, user_id: 1)
Contact.create(first_name: 'Remy', last_name: 'Hadley', email: 'hadly@house.com',
title: 'Internist', party_id: 3, user_id: 1)
Contact.create(first_name: 'Astrid', last_name: 'Farnsworth', email: 'farnsworth@df fringe.com',
title: 'Junior Agent', party_id: 7, user_id: 1)

Contact.create(first_name: 'Cal', last_name: 'Lightman', email: 'cal@lightmangroup.com',
title: 'Owner', party_id: 6, user_id: 2)
Contact.create(first_name: 'Gillian', last_name: 'Foster', email: 'gillian@lightmangroup.com',
title: 'Colleague', party_id: 6, user_id: 2)
Contact.create(first_name: 'Walter', last_name: 'Bishop', email: 'w.bishop@df fringe.com',
title: 'Doctor', party_id: 8, user_id: 2)
Contact.create(first_name: 'Emily', last_name: 'Lightman', email: 'emily@lightmangroup.com',
title: 'Family', party_id: 6, user_id: 2)
Contact.create(first_name: 'Zoe', last_name: 'Landau', email: 'zoe@lightmangroup.com',
title: 'Lawyer', party_id: 6, user_id: 2)
Contact.create(first_name: 'Sharon', last_name: 'Wallowski', email: 'sharon@lightmangroup.com',
title: 'Detective', party_id: 6, user_id: 2)
Contact.create(first_name: 'Karl', last_name: 'Dupree', email: 'karl@lightmangroup.com',
title: 'Agent', party_id: 6, user_id: 2)
Contact.create(first_name: 'Peter', last_name: 'Bishop', email: 'p.bishop@df fringe.com',
title: 'Doctor', party_id: 8, user_id: 2)
Contact.create(first_name: 'Gregory', last_name: 'House', email: 'house@house.com', title: 'Dept.
Head', party_id: 4, user_id: 2)
Contact.create(first_name: 'Lisa', last_name: 'Cuddy', email: 'cuddy@house.com', title: 'Dean',
party_id: 4, user_id: 2)
Contact.create(first_name: 'James', last_name: 'Wilson', email: 'wilson@house.com',
title: 'Oncologist', party_id: 4, user_id: 2)
Contact.create(first_name: 'Eric', last_name: 'Foreman', email: 'foreman@house.com',
title: 'Neurologist', party_id: 4, user_id: 2)
Contact.create(first_name: 'Olivia', last_name: 'Dunham', email: 'dunham@df fringe.com',
title: 'Agent', party_id: 8, user_id: 2)
Contact.create(first_name: 'Allison', last_name: 'Cameron', email: 'cameron@house.com',
title: 'Immunologist', party_id: 4, user_id: 2)
Contact.create(first_name: 'Robert', last_name: 'Chase', email: 'chase@house.com',
title: 'Surgeon', party_id: 4, user_id: 2)
Contact.create(first_name: 'Eli', last_name: 'Loker', email: 'eli@lightmangroup.com',
title: 'Employee', party_id: 6, user_id: 2)
Contact.create(first_name: 'Ria', last_name: 'Torres', email: 'ria@lightmangroup.com',
title: 'Employee', party_id: 6, user_id: 2)
Contact.create(first_name: 'Lawrence', last_name: 'Kutner', email: 'kutner@house.com',
title: 'Sports Medicine', party_id: 4, user_id: 2)
Contact.create(first_name: 'Chris', last_name: 'Taub', email: 'taub@house.com', title: 'Plastic
Surgeon', party_id: 4, user_id: 2)
Contact.create(first_name: 'Remy', last_name: 'Hadley', email: 'hadly@house.com',

```

```

title: 'Internist', party_id: 4, user_id: 2)
Contact.create(first_name: 'Astrid', last_name: 'Farnsworth', email: 'farnsworth@df fringe.com',
title: 'Junior Agent', party_id: 8, user_id: 2)

Note.delete_all
Note.create(contact_id: 1, user_id: 1, body: \
  %{Jonathan is the lead developer for Skyscraper customer management. He is writing it in Ruby
  on Rails, and deploys it to Heroku. The code base is versioned by Git and a remote chronological
  backup is maintained on Github. The ideas are inspired by Highrise however the implementation is
  written entirely on his own. He also likes to listen to Shamisen music when writing code.})

Note.create(contact_id: 11, user_id: 1, body: \
  %{Gregory's father served as a Marine Corps pilot and transferred often to other bases during
  House's childhood. One place in which his father was stationed was Egypt, where House developed
  a fascination with archaeology and treasure-hunting, an interest which led him to keep his
  treasure-hunting tools well into his adulthood.})

Note.create(contact_id: 2, user_id: 3, body: \
  %{Jonathan is the lead developer for Skyscraper customer management. He is writing it in Ruby
  on Rails, and deploys it to Heroku. The code base is versioned by Git and a remote chronological
  backup is maintained on Github. The ideas are inspired by Highrise however the implementation is
  written entirely on his own. He also likes to listen to Shamisen music when writing code.})

Note.create(contact_id: 32, user_id: 2, body: \
  %{Gregory's father served as a Marine Corps pilot and transferred often to other bases during
  House's childhood. One place in which his father was stationed was Egypt, where House developed
  a fascination with archaeology and treasure-hunting, an interest which led him to keep his
  treasure-hunting tools well into his adulthood.})

Activity.delete_all
Activity.update_activity(1,5)
Activity.update_activity(1,7)
Activity.update_activity(1,8)
Activity.update_activity(1,11)
Activity.update_activity(1,13)
Activity.update_activity(1,1)

Activity.update_activity(2,24)
Activity.update_activity(2,25)
Activity.update_activity(2,26)
Activity.update_activity(2,31)
Activity.update_activity(2,29)
Activity.update_activity(2,2)

Activity.update_activity(3,26)
Activity.update_activity(3,31)
Activity.update_activity(3,29)
Activity.update_activity(3,2)

```

Appendix 42 - Forms Javascript

```

$ ->
  $(".mandatory .field input").keyup ->
    empty = false
    $(".mandatory .field input").each ->
      empty = true if $(this).val().length is 0

```

```

    if empty
    $(".actions input").attr "disabled", "disabled"
    else
    $(".actions input").removeAttr "disabled"

```

Appendix 43 - Sky Javascript

```

$ ->
  pathname = window.location.pathname.split("/")
  $(".selected").removeClass "selected"
  $('#left > ul > a[href$="/" + pathname[1] + "'] > li').addClass "selected"

$ ->
  $('#left > ul > a').children("li").each ->
    unless $(this).hasClass("selected")
    $(this).mouseover ->
      $(this).css "background-Color", "#FAF8F3"
    $(this).mouseout ->
      $(this).css "background-Color", ""

$ ->
  $(".activity").each ->
    $(this).mouseover ->
      $(this).css "background-Color", "#FAF8F3"
    $(this).mouseout ->
      $(this).css "background-Color", ""

```

Appendix 44 - Application CSS

```

// *= require_self
// *= require_directory .
@import mixins.css.sass

// structure
body
  margin: 0
  padding: 0
  overflow-y: scroll
#header
  min-width: 800px
#container
  margin: 0 auto
  min-width: 800px
  max-width: 1200px
  padding: 0 20px
  .column
    position: relative
#main
  min-width: 400px
  max-width: 800px
  margin-left: 200px
  margin-right: 200px
#left
  float: left
  width: 200px
#right
  float: right
  width: 185px
#footer

```

```

    clear: both
* html #left
    left: 200px
// end structure

body
    color: #333
    background-color: #F3F1EC
    text-align: left
    font-family: "Lucida Grande", "Lucida Sans Unicode", helvetica, arial, verdana, sans-serif
    font-size: 15px
    img
        border: 0
#header
    @include gradient-background(#5E5E5E,#383838)
    @include box-shadow(0px,2px,5px,#D7D7D7)
    margin-bottom: 20px
    height: 85px
    border-bottom: 1px #D7D7D7
    #logo img
        float: left
        margin-left: 20px
#menu
    margin-top: 30px
    float: right
    margin-right: 20px
    a
        color: white
#container
    @include default-links
#main
    margin-bottom: 30px
    min-height: 300px
    border: none
    padding-bottom: 30px
    @include border-radius(15px)
    @include box-shadow(0,0,30px,#CCC)
    @include default-links
    background-color: #FAFAFA
#top
    @include border-radius-top(15px)
    @include gradient-background(#E6EBF0,#CCC)
    .title
        padding: 20px 15px 10px 30px
        font-size: 17px
        height: 30px
        .create, .edit, .delete, .back
            margin-top: -7px
#bottom
    padding: 15px
    @include border-radius-bottom(15px)
#content
    padding: 15px
#left
    border: none
    ul
        font-size: 16px
        list-style: none
        line-height: 28px
        .selected

```

```

        background-color: #E5E3DE
        li
        white-space: nowrap
        margin-left: -40px
        padding-left: 35px
#right

form
  .please
    font-size: smaller
    color: #555
    margin: 0 0 10px 13px
  label
    width: 5.5em
    float: left
    text-align: right
    padding-top: 0.2em
    margin-right: 0.1em
    display: block
    white-space: nowrap
  input
    &[type=text], &[type=password]
    height: 20px
    font-size: 16px
    width: 180px
  select, textarea
    margin-left: 0.5em
  input
    margin-left: 0.5em
    &[type=submit]
    font-size: 16px
  .submit
    margin-left: 4em
  br
    display: none

#centered
  margin: 0 auto
  width: 400px
  #logo
    text-align: center
  #splash
    width: 400px
    min-height: 300px
    @include box-shadow(0,0,30px,#CCC)
    @include border-radius(15px)
    border: none
    background-color: #F2F2F2

```

Appendix 45 - Buttons CSS

```

@import mixins.css.sass
.button
  float: left
  font-size: 15px
  height: 30px

.create

```



```

    @include button(3px,#40E038,#33B300,#FAFAFA)
    input
        border-style: 1px solid #13AD13

.edit
    @include button(3px,#D1A11D,#AD8310,#F2F2F2)

.delete
    @include button(3px,#EB5149,red,#F2F2F2)

.back
    @include button(3px,silver,#CCC,#FAFAFA)
    input
        border-style: 1px solid #13AD13

.export
    @include button(3px,#D1A11D,#AD8310,#F2F2F2)

```

Appendix 46 - Contacts CSS

```

@import mixins.css.sass

.contact
    height: 85px
    border-bottom: 1px solid #DEDEDE
    margin-bottom: 15px
    .gravatar
        float: left
        margin-right: 20px
        border: 1px solid silver
        padding: 3px 3px 0px 3px
    .name
        display: table-row
        height: 20px
        font:
            size: 18px
            weight: bold
    .details
        display: table-row
        font-size: 16px
        .updated_at
            font-size: smaller

```

Appendix 47 - Mixins CSS

```

@mixin box-shadow($x-off,$y-off,$blurspread,$color)
    -webkit-box-shadow: $x-off $y-off $blurspread $color
    -moz-box-shadow: $x-off $y-off $blurspread $color
    box-shadow: $x-off $y-off $blurspread $color

@mixin border-radius($radius)
    -webkit-border-radius: $radius
    -moz-border-radius: $radius
    border-radius: $radius

@mixin all-border-radius($top, $right, $bottom, $left)
    -webkit-border-radius: $top $right $bottom $left
    -moz-border-radius: $top $right $bottom $left
    border-radius: $top $right $bottom $left

```

```

@mixin border-radius-top($radius)
  -webkit-border-top-left-radius: $radius
  -webkit-border-top-right-radius: $radius
  -moz-border-radius-topleft: $radius
  -moz-border-radius-topright: $radius
  border-top-left-radius: $radius
  border-top-right-radius: $radius

@mixin border-radius-bottom($radius)
  -webkit-border-bottom-right-radius: $radius
  -webkit-border-bottom-left-radius: $radius
  -moz-border-radius-bottomright: $radius
  -moz-border-radius-bottomleft: $radius
  border-bottom-right-radius: $radius
  border-bottom-left-radius: $radius

@mixin gradient-background($color1,$color2)
  background: $color2
  background-image: -ms-linear-gradient(top, $color1 0%, $color2 100%) /* IE10 */
  background-image: -moz-linear-gradient(top, $color1 0%, $color2 100%) /* Mozilla Firefox */
  background-image: -o-linear-gradient(top, $color1 0%, $color2 100%) /* Opera */
  background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0, $color1),
  color-stop(1, $color2)) /* Webkit (Safari/Chrome 10) */
  background-image: -webkit-linear-gradient(top, $color1 0%, $color2 100%) /* Webkit (Chrome 11+) */
  /*
  background-image: linear-gradient(top, $color1 0%, $color2 100%) /* Proposed W3C Markup */
  filter: progid:DXImageTransform.Microsoft.gradient(GradientType=0,startColorstr='#{$color1}',
  endColorstr='#{$color2}') /* IE6 & IE7 */
  -ms-
  filter: "progid:DXImageTransform.Microsoft.gradient(GradientType=0,startColorstr='#{$color1}',
  endColorstr='#{$color2}')" /* IE8 */

@mixin button($radius,$color1,$color2,$font-color)
  float: right
  margin-right: 3px
  input
    @include border-radius($radius)
    @include gradient-background($color1,$color2)
    border: 2px solid $color1
    color: $font-color
    min-width: 50px
    min-height: 30px
    font-size: 16px

@mixin default-links
  a:link, a:visited
    color: #4F4F4F
    text-decoration: none
  .hover a:hover
    text-decoration: underline

```

Appendix 48 - Notes CSS

```

.note
  border-bottom: 1px solid #cfcfcf
  margin-bottom: 20px
  padding: 0 0 10px 10px
  .updated_at

```

```

        font-size: 15px
        font-weight: bold
    .options
        float: right
        font-size: smaller
    .by_user, .last_edit
        display: inline
        font-size: smaller
        color: grey
    .body
        padding: 10px 10px 0 10px
    .focus
        float: left
        font-weight: bold
    .month_day
        text-align: right
        font-size: smaller
        color: grey

#note_form
    form
        label
            width: 100%
            text-align: left
            padding-top: 5px
            margin-left: 10px
            font-weight: bold
        .help
            font-size: smaller
            color: grey
            margin: 0 10px 0 5px
        textarea
            color: #333
            resize: vertical
            margin-left: 0.5em
            width: 90%
            font-size: 16px
            font-family: "Lucida Grande", "Lucida Sans Unicode", helvetica, arial, verdana, sans-
serif
        .actions
            float: right
            margin: -5px 20px 0 0
        .bottom
            width: 100%
            height: 30px

```

Appendix 49 - Parties CSS

```
@import mixins.css.sass
```

```

#parties
    // a
    //  &:link, &:visited
    //  color: #265080
    //  text-decoration: none
    .party
        height: 85px
        border-bottom: 1px solid #DEDEDE
        margin-bottom: 15px

```

```

.image
  float: left
  margin-right: 20px
  border: 1px solid silver
  padding: 3px 3px 0px 3px
.name
  font:
    size: 18px
    weight: bold
.details
  font-size: 16px
  .updated_at
    font-size: smaller

.edit
  @include button(3px,#D1A11D,#AD8310,#F2F2F2)
.delete
  @include button(3px,#EB5149,red,#F2F2F2)

```

Appendix 50 - Scaffolds CSS

```

@import mixins.css.sass

body
  font-size: 16px
  line-height: 18px
.edit_text
  display: inline
  a
    color: #666 !important
    &:visited
      color: #666 !important
    &:hover
      color: white !important
      background-color: #D1A11D
      @include border-radius(4px)
.cancel_text
  display: inline
  a
    color: #8C2D2D !important
    &:visited
      color: #8C2D2D !important
    &:hover
      color: white !important
      background-color: #8C2D2D
      @include border-radius(4px)

p, ol, ul, td
  font-size: 16px
  line-height: 18px

pre
  background-color: #eee
  padding: 10px
  font-size: 16px

div
  &.field, &.actions
    margin-bottom: 15px

```

```

#notice
  margin-left: 10px
  color: green

.field_with_errors
  input[type=text], input[type=password]
    height: 20px
    font-size: 16px

#error_explanation
  width: auto
  border: 2px solid #c00
  padding: 7px
  padding-bottom: 0
  margin-bottom: 20px
  background-color: #f0f0f0
  @include border-radius(6px)
  h2
    text-align: left
    font-weight: bold
    padding: 5px 5px 5px 15px
    font-size: 13px
    margin: -7px
    margin-bottom: 0px
    @include gradient-background(#c00,red)
    color: white
  ul li
    font-size: 12px
    list-style: square

```

Appendix 51 - Sky CSS

```

@import optionals/dropdown.css.sass
@import mixins.css.sass

#activity
  border-top: 1px solid #dedede
  padding: 15px 0 0 0

  .header
    font:
      size: 14px
      weight: bold
    color: #999
    padding-left: 5px

  table
    border-collapse: collapse
    tr
      td
        border-bottom: 1px solid #dedede
        padding: 5px 15px 2px 0

  .activity
    .name
      font-size: small
      padding-left: 5px

```

```

        height: auto
        width: 100%
        .avatar
        img
        border: 1px solid silver
        .title
        font-size: 11px
        padding-left: 5px
        color: #999

#notes_activity
    //latest activity feed wrapper

#search
    float: left
    padding-top: 30px
    input
        float: left
        &[type=text]
        @include border-radius(4px)
        height: 30px
        width: 140px
        border: none
        background: #fff
        font-size: 15px
        padding: 0 5px 0 5px
        &[type=submit]
        height: 30px
        font-size: 16px
        color: #FAFAFA
        text-shadow: 0 1px #404040
        border: 1px solid #2E2E2E
        @include border-radius(4px)
        @include gradient-background(#858585,#757575)
        @include box-shadow(0,0,3px,#7A7A7A)

#top_menu
    margin-top: 30px
    float: right
    margin-right: 20px
    a
        color: white

#result
    padding: 10px
    background-color: #E8E8E8
    font-size: large
    text-align: center
    margin-bottom: 15px
    a
        font-weight: bold
        &:hover
        @include border-radius(5px)
        color: white
        padding: 3px
        background-color: #32A623

```

Appendix 52 - Users CSS

```

@import mixins.css.sass

$default-border: 1px solid silver

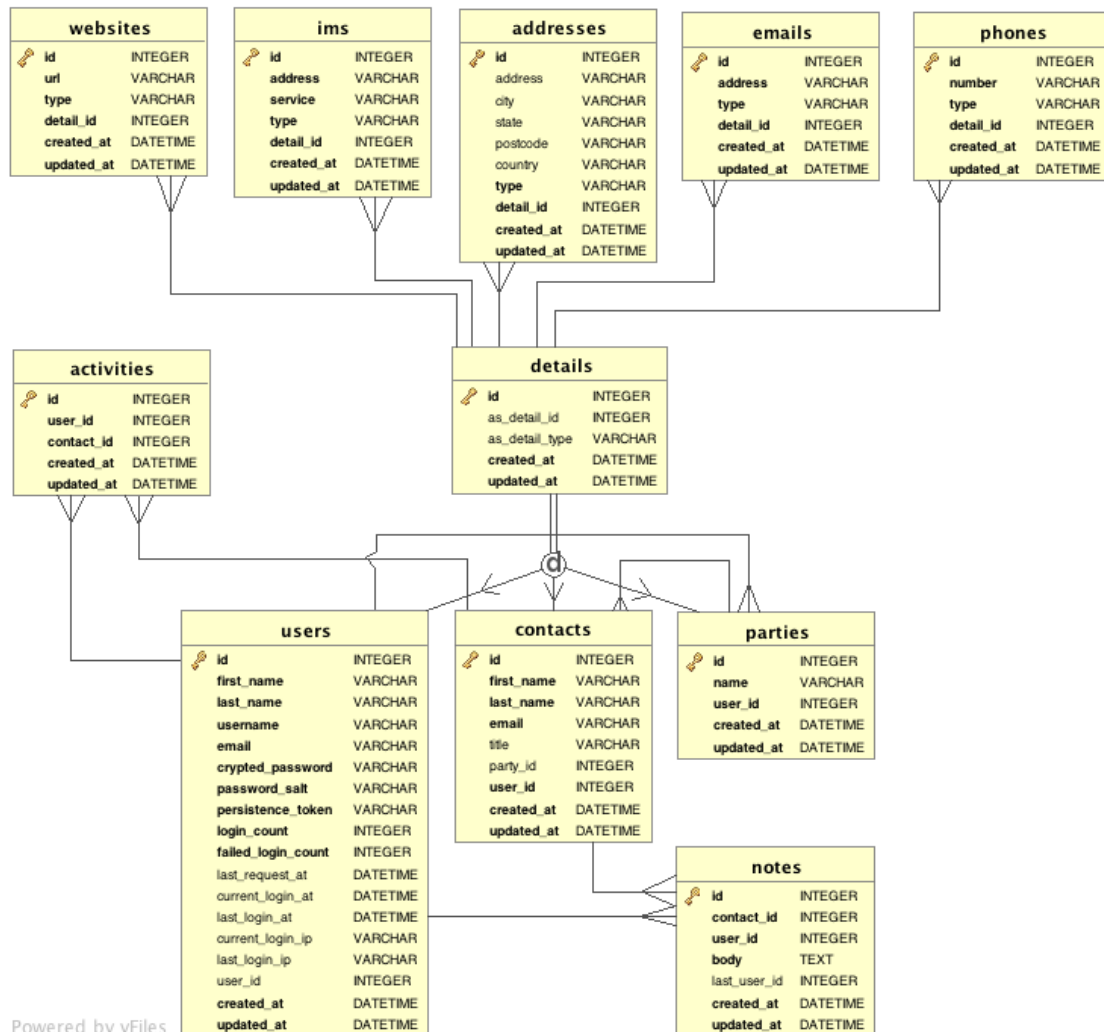
.user
  width: auto
  color: #444
  height: 75px
  border-bottom: $default-border
  margin-bottom: 7px
  .gravatar
    float: left
    margin-right: 20px
    border: $default-border
    padding: 3px 3px 0px 3px
  .name
    display: table-row
    font:
      size: 16px
      weight: bold
  .email
    display: table-row
    font-size: smaller
    a
      color: grey
  .role
    display: table-row
    float: right
    color: grey
    font-size: smaller
    margin-right: 40px
  .username
    display: table-row
    font:
      size: smaller
      weight: bold

.description
  border: 1px solid #E8E9EB
  padding: 10px
  margin: 0 10px 15px 10px
  background: #E8E9EB
  font-size: small
  color: #666
  @include box-shadow(0,0,15px,#E8E9EB)
  @include border-radius(10px)

#show
  color: #666
  .details
    padding-top: 5px

```

Appendix 53 - Database Schema Redesign



Powered by yFiles