Tonight - An iOS app for event aggregation

Final Report for CS39440 Major Project

Author: Ryan Clarke (ryc@aber.ac.uk)
Supervisor: Dr. Richard Jensen (rkj@aber.ac.uk)

8th April 2014
Version: 1.0 (Draft)

This report was submitted as partial fulfilment of a BSc degree in Computer Science (G401)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature ....

Date ....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature ....

Date ....

Acknowledgements

I would like to thank my dissertation tutor Richard Jensen for helping throughout the project.

I would also like to thank my family for their continued support throughout my time at university!

Last but not least I would like to thank all of my friends, mainly the `Harvington household' and the `Orchard Banter House' for the support and telling me when my app `looked rubbish' and the usual `that dont work'. Cheers guys!

Abstract

This report describes a project for helping users discover events around them. When visiting a new area trying to find out what's going on can be difficult even for the most adept regular. Tonight tries to solve that problem by providing a mobile application that allows you to discover events that are going on around you. It does this by presenting the user with events that may be of interest to them via a `My Feed' section. A user also has the ability to explore events by selecting a city/area then drilling down to a particular venue or category. The users is then able to view all of the details on a particular event, with the option to follow the event. By following an event the system can learn about the user and present them with suggestions of events, based on what similar users are also following. To do this I developed a Ruby on Rails server application that provided a REST based API to access the information that a user needed, as part of this I also developed a data mining module to be able to pull in information from various data sources. Using this in conjunction with an iOS application to present the data and add in functionality to the user on a mobile device.

CONTENTS

1 Background & Objectives **1**
1.1 Background . . . 1
1.1.1 Current Solutions . . . 2
1.2 Analysis . . . 4
1.2.1 Primary objectives . . . 4
1.2.2 Secondary objectives . . . 5
1.2.3 Non-Functional requirements . . . 6
1.2.4 Hardware requirements . . . 6
1.3 iOS and Ruby on Rails . . . 6
1.3.1 iOS Development . . . 6
1.3.2 Server Development . . . 7
1.3.3 API Interoperability . . . 8
1.4 Process . . . 8
1.4.1 Server application . . . 9
1.4.2 iOS Application . . . 9
1.4.3 GitHub . . . 9
1.4.4 Trello . . . 10

2 Design **11**
2.1 Database . . . 11
2.2 Server Application . . . 12
2.2.1 Development tools . . . 12
2.2.2 OAuth . . . 12
2.2.3 Routes . . . 12
2.2.4 Data mining element . . . 13
2.2.5 Class diagram . . . 14
2.3 iOS Application . . . 14
2.3.1 Development tools . . . 14
2.3.2 Wireframes . . . 14
2.3.3 Class diagrams . . . 15

3 Implementation **19**
3.1 Personalised feed . . . 19
3.2 Registration and logging in . . . 19
3.3 Storing passwords . . . 20
3.4 Search functionality . . . 20
3.5 Adding in the following functionality . . . 21
3.6 Retrieving events from the jobs . . . 21
3.7 Interoperability . . . 22
3.8 OAuth . . . 23
3.9 Storing user credentials . . . 23
3.10 Evaluation . . . 24

4 Testing **25**
4.1 Overall Approach to Testing . . . 25

4.2 Automated Testing . . . 25
4.2.1 Stress Testing . . . 26
4.3 Integration Testing . . . 26

5 Evaluation **28**
5.1 Testing . . . 28
5.2 Requirements . . . 28
5.3 Tools used . . . 29
5.4 iOS App . . . 30
5.5 Server Application . . . 30
5.6 User expectations . . . 31
5.7 Reflections . . . 31

Appendices **32**

A Third-Party Code and Libraries **33**
1.1 Server side libraries . . . 33
1.2 Mobile application libraries . . . 33

B Test Results **34**
2.1 RSpec Results . . . 34

Annotated Bibliography **39**

LIST OF FIGURES

1.1 Line Up home screen . . . 2
1.2 Spotnight home screen . . . 3
1.3 Songkick home screen . . . 4
1.4 Image of Google Analytics for popular events site . . . 7
1.5 Trello . . . 10

2.1 Overview design . . . 11
2.6 Basic wireframes and design choices for the mobile application . . . 15
2.7 Individual event wireframe . . . 15
2.8 List of events wireframe . . . 15
2.2 Entity Relationship Diagram . . . 16
2.3 Class Diagram for API integration . . . 17
2.4 Overview of the data mining aspect . . . 17
2.5 Controllers Class Diagram . . . 18
2.9 Class diagram for the iOS data classes . . . 18

4.1 Stess testing of the iOS Application . . . 26
4.2 Memory usage during operation . . . 26
4.3 CPU usage during operation . . . 26

LIST OF TABLES

1.1 Data retrieved on events and venues from Facebook API 8
1.2 Event data 8
1.3 Venue data 8

2.1 API Routes 13

Chapter 1

Background & Objectives

1.1 Background

The UK music events and festival industry was estimated at 3.5 billion in 2010 and expected to raise to 4.2 billion in 2015. With roughly 530,000 full time equivalent jobs employed by 25,000 employers. [8] The industry itself is very diverse with a huge range of events going on every night, from intimate acoustic sets to all night raves, many people find themselves going to the same venue over and over again not aware of other events happening around them. Currently the promotion of events is limited to the users of certain social media sites, or purpose built websites for promotion of events. The tools that a promoter currently has at their disposal is limited and sporadic at the best, for an example The Rainbow Venues [27] use a mixture of their own site, social media (Facebook/Twitter) and the ticket masters. This brings to the core problem for a potential customer to discover the venue, they must be aware of the venue first to be able to find the information that they want. Even with use of social media the reach of a promoted event will only reach those that know of the venue or by those that 'share' the promotion to their friends.

With the cost of living going down people are looking further afield for evening entertainment, by not being from around the area they are at an immediate dis-advantage. With the increase of popularity of smart phones many people will pick up their smart phones, and immediately search for an application to help them find events on the go. The current selection of applications are very limited to the style of events and the area they are based, which means a new application for each time they go away this is not ideal and will quickly fill up a smart phone.

At this time there is a wide selection of similar applications available to download from the Apple AppStore. Each of these applications differ slightly be it the way the information is presented or the key functionality they provide. Many of the applications are only suited towards a particular city or specific headlining artists, by doing this they are somewhat limiting the scope of their audience. Many of the solutions out there also currently only utilise data that's being input by employees, this means that only a representative of all the events happening are selected and presented by the application.

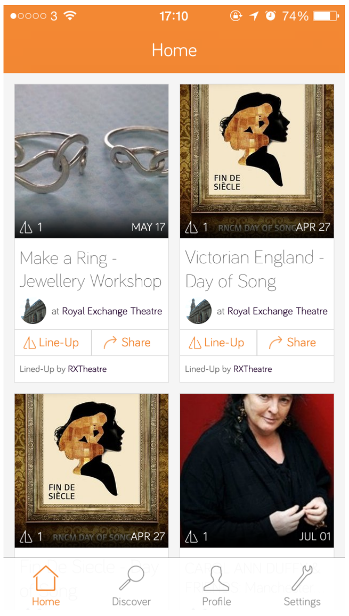
1.1.1 Current Solutions

Here are some of the current applications that are available to download from the Apple AppStore. All of the applications are free downloads, along with free registration to use the features. All of the applications as standard provide a feed of events (in a relevant order), and the ability to filter them in various ways.

1.1.1.1 Line Up

Line Up [15] shows a wide variety of events in the Manchester area, it gives you the ability to add events to their `Line Up' which is essentially a list of events that they are planning to go to or going to. It also gives you the opportunity to share the event via popular social network sites, increasing there own reach and allowing the users friends to see what they are attending/interested in. To discover events you are able to browse events by type of place, People, and all events. The application also allows you to follow other users that use the application, allowing a user to see what other users are attending. When viewing an individual event you can see a title, description, dates of the event, and the location.

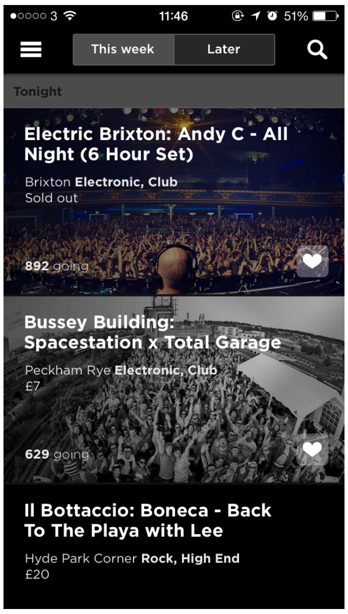
Figure 1.1: Image showing the home screen for the Line Up application


17:10
74%
Home
MAY 17
APR 27
Make a Ring - Jewellery Workshop
at Royal Exchange Theatre
Victorian England - Day of Song
at Royal Exchange Theatre
Line-Up
Share
Lined-Up by RXTheatre
FIN DE SIÈCLE
JUL 01
Home
Discover
Profile
Settings

1.1.1.2 Spotnight

Spotnight [25] shows selected events happening in the London area, it gives you the ability to purchase tickets for the events available through a 3rd party service. You can view the events that are happening either this week or later on, however you are able to apply filters for specific areas, style of music, and genres. The application allows you to like events, however this does not seem to have any particular effect to the ordering of the events or any other aesthetic/functional item of the application. When viewing an individual event, you are able to view the location, description, price, images, people attending, and venue contact details. You are also able to share particular events through social networking sites; Facebook and Twitter.

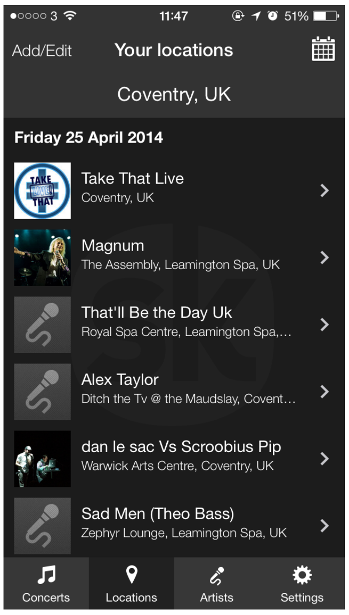
Figure 1.2: Image showing the home screen for the Spotnight application


•○○○○ 3
11:46
51%
This week
Later
Tonight
Electric Brixton: Andy C - All Night (6 Hour Set)
Brixton Electronic, Club
Sold out
892 going
Bussey Building: Spacestation x Total Garage
Peckham Rye Electronic, Club
£7
629 going
Il Bottaccio: Boneca - Back To The Playa with Lee
Hyde Park Corner Rock, High End
£20

1.1.1.3 Songkick

Songlick [23] offers a selection of artist specific events, Songkick allows you to see events happening in your area, and by artist. From here you are presented with the details of the event including the line up, location and an option to purchase tickets via a 3rd party service. Songkick also allows you to track an event and mark an event as attending, to use these functions you are required to sign up however casual use of the application does not require signing up. The application also allows you to follow artists and will suggest events that are happening near you within the artists you are following.

Figure 1.3: Image showing the home screen for the Songkick application



Coventry, UK

Friday 25 April 2014

Take That Live
Coventry, UK

Magnum
The Assembly, Leamington Spa, UK

That'll Be the Day Uk
Royal Spa Centre, Leamington Spa,...

Alex Taylor
Ditch the Tv @ the Maudslay, Covent...

dan le sac Vs Scroobius Pip
Warwick Arts Centre, Coventry, UK

Sad Men (Theo Bass)
Zephyr Lounge, Leamington Spa, UK

1.2 Analysis

With the main problem being users not knowing an area there is a prevalent set of key features that is required by the application for it to help solve the issue at hand. The most prominent being be able to grab the users current location and to be able to filter out the events that are happening near to their current location. However many people may not want to travel to the area only to find out that nothing that interests them is happening, so ideally the events should be categorised by area and then a user should be able to browse the areas they are visiting. Bearing this in mind, as the user is visiting a potentially new area they may require directions or some sort of map indicating the location of the event.

Another issue is that many of the previous systems require human interaction to find out about events and manually input these into to the system. Whilst this allows for a human based data validation it can be slow and time consuming to conduct, and result in only key events being selected and not provide a large enough range for their users. By use of data mining techniques we are able to pull in a must larger variety of events and therefore cater for a wider audience, thus allowing the user to actually discover new events that they may not have thought about. This relates to the first issue, by pulling in the details automatically from various API's we are able to half the work required by us and the promotions companies. Making the use of this service much more attractive to both parties, ultimately widening the stakeholders and potential audience.

1.2.1 Primary objectives

There a number of primary objectives for this task, these are listed below

Personalised feed
The application must produce a feed that's personalised to the currently logged in user

Registration/Authentication
The system must provide the ability to create a new user, and to login a previously registered user. They should also stay logged in until the have logged out of the account.

Ability to follow events
The application should provide functionality to follow an event to help facilitate the personalised feed.

Categorise the events
The application should allow the events to be presented in multiple ways including categories and areas.

Show individual events
The application must provide all of the relevant details of each event

Connect to various API's
The server must be able to pull in information from a number of different API's and

Application - Server interoperability
The application and server need to be able to pass information to each using some form of interoperability.

Interoperability authentication
The implementation of the interoperability must utilise some form of authentication for access.

1.2.2 Secondary objectives

These objectives are not critical to the running of the application but provides extra functionality to the service.

Location awareness
The application should be of the location of the user, and only use the nearest events.

Notifications
The user should receive notifications then events have been updated based off the data mining application.

User profile
The application should show a users profile, showing the events they are following and their personal details.

Link to purchase the events
The application should provide some sort of back link to enable a user to purchase tickets through a 3rd party service.

Ability to administer data from an administration panel
The server application must give the ability to administer the data stored from an administration panel.

Ability to administer jobs from an administration panel
The server application must be able to administer data mining jobs from an administration panel

1.2.3 Non-Functional requirements

Application needs to training to be used
The user should not need any training to use the application

1.2.4 Hardware requirements

Application must run on > iPhone 4
The iOS application must run on all iPhones that run the latest version of the iOS firmware.

Server should run on cloud computing provider
The server element should run on a cloud computer to allow for high demand with ease.

1.3 iOS and Ruby on Rails

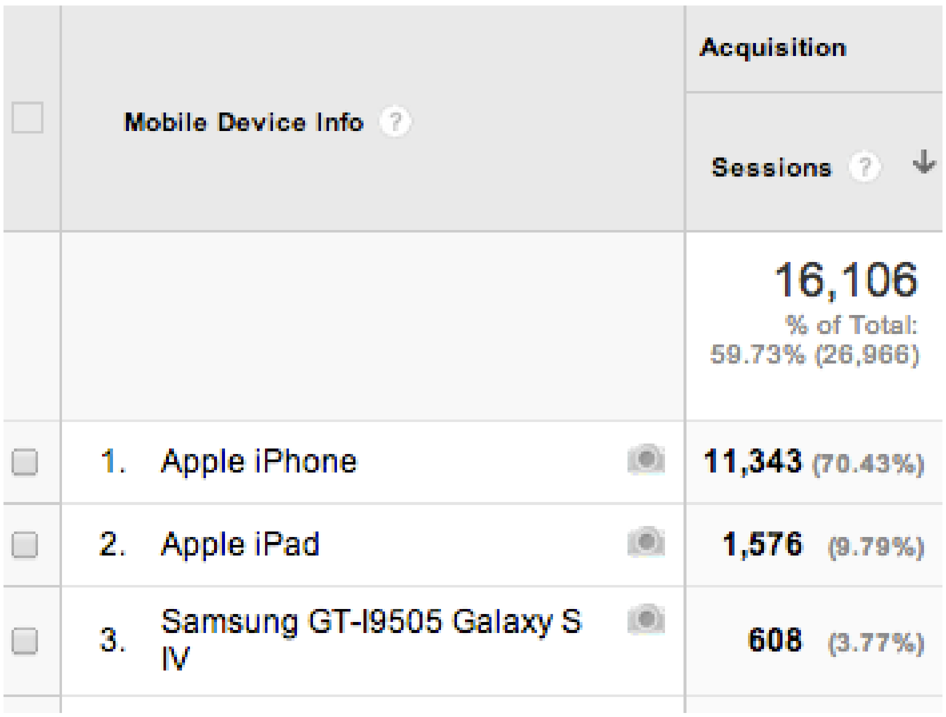
1.3.1 iOS Development

The application itself will be based on the iOS platform for Apple iPhones, being programmed using Apples Objective C language. This decision was made based on some Google Analytics information, which was provided by a popular events company in Birmingham. Figure 1.4 shows us the statistics that's been produced by Google Analytics showing that over 70% of their traffic was coming from Apple iPhones. The programmer also had readily available access to an iPhone to assist with the development process underlined.

To develop iOS applications you are required to use Apples own IDE XCode 5 packaged into this is the iOS simulator, this allows me to test applications developed locally on the machine without needing to make a payment to the developers network. The simulator allows for all parts of the application to be tested, however I will need to manually set up locations to test the location awareness of the application. I will also be utilising a package manager called CocoaPods [3] which allows me to pull in various libraries with ease and make sure I'm using the latest versions with future updates. To get used to the new IDE and we will utilise the resources from Ray Wenderlichs sites [20], Ray gives many resources including sample projects that can be analysed and tutorials.

iOS has 2 main design patterns that are usually followed, the first being model view controller (MVC) this is implemented by having 3 main sections of code; a place where the

Figure 1.4: This image shows within a month their main traffic is through Apple iPhones, with secondary traffic of Android being a much smaller percentage.



	Mobile Device Info	Acquisition
		Sessions
		16,106 % of Total: 59.73% (26,966)
1.	Apple iPhone	11,343 (70.43%)
2.	Apple iPad	1,576 (9.79%)
3.	Samsung GT-I9505 Galaxy S IV	608 (3.77%)

data is defined/stored(model), a place where any business logic is performed(controller), and a place where the data is outputted(view). Typically you can find multiple instances of the MVC pattern inside of one project. However it also utilises a delegation & target/action delegation is used to pass data between the multiple MVC's you will find inside a project, and target/action which is used by iOS to link buttons with methods to be called upon that button being selected.

1.3.2 Server Development

The project will also require a server side element to be able to mine the data and provide the data to the application, this will be written in Ruby using the Ruby on Rails framework. Using the framework gives access to lots of functionality not built into the core of Ruby and provides a production ready environment. Due to the explosive nature of applications, it's been decided to use cloud computing to be able to effectively manage the demand of resources used by the application. For this it was decided the best cloud platform is Heroku [9] this was mainly due to the ease of deployment, which involved a git push to their remote server, Heroku also provided a free service to be used during development and for small scale applications.

1.3.3 API Interoperability

To get the data to be used throughout the system, it will need to communicate to a number of API's and store the outputted data to to be processed and outputted through the server element of the project. The main API to communicate with is the Facebook Graph API [5], the key event and venue details that have been provided from Facebook have been listed in Table 1.1.

Table 1.1: Data retrieved on events and venues from Facebook API

Table 1.2: Event data

Property Name	Type
id	numeric string
cover	CoverPhoto
description	string
end_time	datetime
is_date_only	bool
location	string
name	string
owner	User|Page|Group
parent_group	Group
privacy	string
start_time	datetime
ticket_uri	string
timezone	string
updated_time	datetime
venue	Page

Table 1.3: Venue data

Property Name	Type
id	string
about[1]	string
checkins	int32
company_overview[1]	string
cover[1]	CoverPhoto
current_location	string
description[1]	string
general_info[1]	string
hometown	string
likes	int32
link	string
location	object
country	string
city	string
longitude	float
zip	string
state	string
street	string
latitude	float
name[1]	string
parking[1]	object
street	int32
lot	int32
phone[1]	string
username	string
website[1]	string

1.4 Process

Mostly the methodology followed was a cut down version of agile, optimised for a single developer. This decision was made to allow the programmer to add in extra functionality as seen fit throughout the project, and to allow the design to flex with these requirements. To develop the bulk of the application feature driven development (FDD) was used as there were 3 main logical aspects to the project those being; Interoperability, iOS Application, and Data mining. Due to the nature of each of these sub sections, there will a slightly different sub-methodology for each. Each feature relies on one already being present bar the data mining feature, this is key to the order of features. The first feature developed was the data mining, which received information and input it into a database, the interoperability feature of the server element (REST API) and then the iOS application itself to use the server element of the project. Agile was also chosen as whilst we know the scope of the project currently, looking to the future it may be required to add in extra functionality based off the reception received. Inside of each feature was a set of iterations, these were primarily individual requirements for each feature.

1.4.1 Server application

The development of the server application will be following the test driven development (TDD) methodology, utilising the red, green, re-factor ideology. This was chosen as it will allow for flexibility in the overall design of the application, and will also ensure tests are being written for the functionality of the software. By following TDD it will also ensure that the code is the most efficient, as no more code is written except for what's required to pass the various tests.

1.4.2 iOS Application

The development of the iOS application was a lot more conventional, a feature was programmed with relevant unit testing implemented to ensure the functionality produced the correct output. A form of TDD was used as sometimes a test failed so it was required to go back and fix the code to ensure the test passed. After each feature was added, device testing was undertaken ensuring it all ties together well and works on the device.

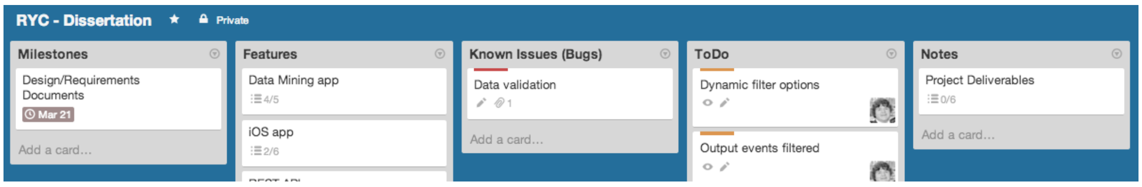
1.4.3 GitHub

Git will be used for the version control system (VCS), Git is a decentralised system allowing the repository to be stored on multiple machines. Git works off a `branching' and `merging ' workflow, so for each new feature you create a branch off the currently stable repository add in the feature and then if it passes the relevant QA and tests then it will be merged into the stable branch. Allowing a stable branch of code to be accessible whilst adding in new features. As its de-centralised the repository will be stored on my laptop and the pushed to GitHub to be stored remotely in case anything was to happen to the files stored inside the repository. GitHub also offer an issues system which is a ToDo list especially for programming by allowing certain tasks such as bugs and feature it can be sued to help track the issues with the project.

1.4.4 Trello

To help with the process of development Trello will be used. Trello is a project management application, by defining titles for tasks you can swap cards around to be able to see a quick overview of where you are. You are able to see and add more details into each individual card, however the title must be as descriptive as it can be to show a concise overview on the main screen. Trello will be used to keep track of each requirement that needs to be implemented and at what stage the documentation is at.

Figure 1.5: This images shows how Trello was set up, with a selection of cards used to help keep track of the project.

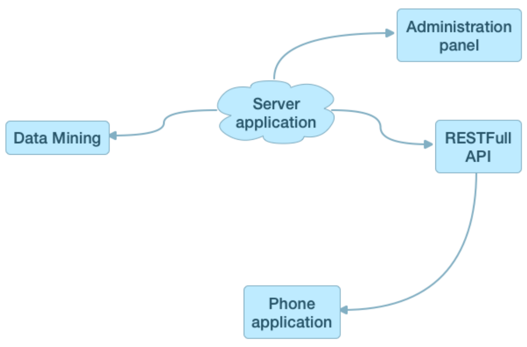


Chapter 2

Design

There are 2 main parts of the project the first being the server element and the second the mobile application itself. The mobile application will be connecting to the server applications through a RESTFull API to retrieve all of the data that's required. The server element will be composed of 3 main parts the RESTFull API, Administration panel, and the data mining part.

Figure 2.1: This diagram shows the main areas of the project and how the interact with each other.



2.1 Database

To full fill the requirement of utilising a cloud server solution the database will be using PostgresSQL which the server element will interface with it using the ActiveRecord gem bundled with Ruby on Rails. Because the server element was designed using TDD and each the overall design, was moulded during the process Figure 2.2 shows the resultant entity relationship diagram at the end of the project. There is also a set of definite fields that was worked from the Facebook API schema listed in Table 1.1, schema.org [22] was also used to help build the list of relevant information on an event.

2.2 Server Application

The server application, will be composed of 3 components these will be the Data mining module, the RESTFull API for interfacing with the mobile application, and the administration panel. These components whilst being separate, will all interact with the same database and will reside within the same code base to be uploaded and ran on the cloud application platform (CAP).

2.2.1 Development tools

The server application will be written in Ruby utilising the Ruby on Rails (RoR) framework, there were other options to use such as; Node, Clojure, Java, Python. However the RoR framework provided great native support for developing API's and database integration. Rails also gives us the ability to run code using the rails environment via the command line by use of rake tasks. The CAP will call the task hourly using a cron job to pull in new events and venues, these jobs will be specified by the administration panel and then ran by the rake task.

To develop the server side code the programmer will be using the Sublime Text 2 text editor [12] with the following packages installed; RSpec [7], Ruby Test [16], and Rails Developer Snippets [26]. These packages will assist the programmer by allowing them to run tests within the editor and provide key snippets to be used by them. The use of a text editor with the packages noted installed allowed a cleaner interface for the programmer to deal with and was a tool they where familiar with.

As stated above, the programmer will be following a test driven development approach as such it will be required to use some form of test framework. For this the programmer will be using RSpec for RoR gem [21], this gem provides the user with the RSpec suite correctly optimised and set-up to be used within a RoR environment. The programmer will also be using FactoryGirl [28] a higher grain of control over mock models, and WebMock [2] to be able to mock API connections with test data.

2.2.2 OAuth

The application will use the OAuth standard for authentication to the application, OAuth is an open protocol that offers `secure client delegation'. By delegating a different user token for each user the server can serve user specific data to each user, whilst ensuring statelessness. The use of OAuth will also restrict server resources to defined applications, allowing a higher level of control to the applications that utilise the API.

2.2.3 Routes

Ruby on Rails allows for a series of URL routes to be defined, to ensure the project follows REST principles the API the interface is required to be uniform, for this all requests will be responded in JSON and will be in the format similar to snippet 1. The API URI structure should also be representational of the data being served and how the data is structured in the database. Table 2.1 shows the endpoints and accessors for the data that's available through the API. The returned data will also be paginated using the gem api-pagination [4] the pagination URIs will be formed as part of the header information of the response to keep the returned body easy to parse.

Example Code 1 Example JSON output for REST requests

```
{
`code' => `201',
`errors' => `',
`body' => [ ]
}
```

Table 2.1: API Routes

Verb	URI Pattern	Controller#Action
GET	/api/v1/events(.:format)	api/v1/events#index
GET	/api/v1/events/:id(.:format)	api/v1/events#eventByID
GET	/api/v1/venues(.:format)	api/v1/venues#index
GET	/api/v1/venues/:id(.:format)	api/v1/venues#show
GET	/api/v1/venues/:id/events(.:format)	api/v1/events#eventsByVenue
POST	/api/v1/register(.:format)	api/v1/user#register
GET	/api/v1/user(.:format)	api/v1/user#index
POST	/api/v1/user(.:format)	api/v1/user#update
GET	/api/v1/user/feed(.:format)	api/v1/user#feed
GET	/api/v1/user/following(.:format)	api/v1/following#index
POST	/api/v1/user/follow(.:format)	api/v1/following#followEvent
POST	/api/v1/user/unfollow(.:format)	api/v1/following#unfollowEvent
GET	/api/v1/categories(.:format)	api/v1/categories#index
GET	/api/v1/categories/:id(.:format)	api/v1/categories#show
GET	/api/v1/categories/:id/events(.:format)	api/v1/events#eventsByCategory
GET	/api/v1/cities(.:format)	api/v1/cities#index
GET	/api/v1/cities/:id(.:format)	api/v1/cities#show
GET	/api/v1/cities/:id/events(.:format)	api/v1/events#eventsByCity
GET	/api/v1/cities/:city_id/venues(.:format)	api/v1/venues#venuesByCity
GET	/api/v1/cities/:city_id/categories(.:format)	api/v1/categories#catsByCity

2.2.4 Data mining element

The data mining element is broken up into a 2 different parts these being the interfaces to the external data sources, and the second being the job scheduler. Figure 2.4 shows how these parts will work together to input data from the various API's that's needed. To allow for any number of API's to be added to the system, a base class called `Resources' will be needed with any inherited functions needed Figure 2.3 shows the class diagram for this part of the project. Then another class that inherited from `Resources' will be used to define each individual API connection. By doing this the API is easy to extend simply by working as a interpreter to the different names for data fields and layout for each API response. The scheduler RAKE task will then read in the jobs defined by the administration panel and pull in the relevant information and insert this into the database.

2.2.5 Class diagram

Following on from the MVC design pattern and the fact the API needs to be representational, each set of data (table) is represented by a controller. However in some cases there are filters such as being able to get the events via the venue, since this will return a list of events rather than venues the method to do this is inside the events controller. Figure 2.5 shows the structure of the controllers, and the methods inside of them.

2.3 iOS Application

2.3.1 Development tools

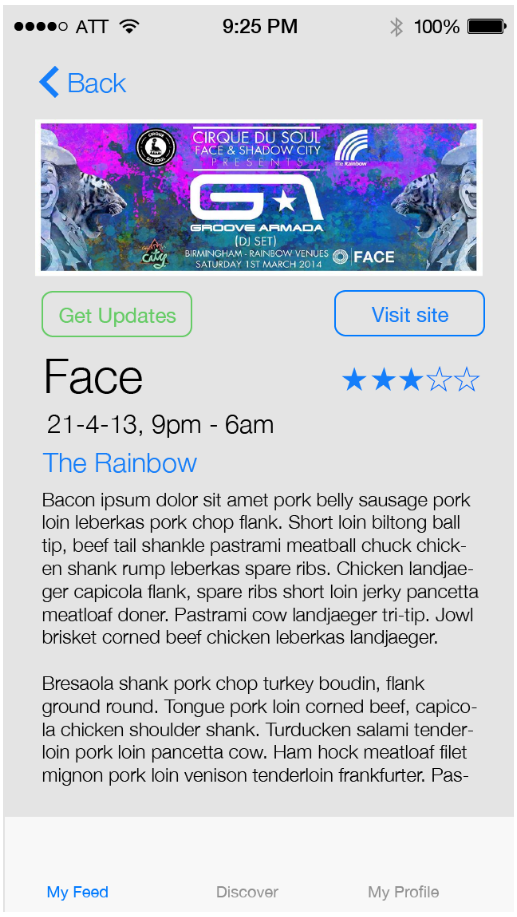
To develop in iOS you are required to use Apples own IDE XCode, version 5 is the latest and supports the latest version of iOS. XCode 5 includes a number of tools bundled with the IDE including the iOS Simulator and GIT Integration. Cocoa pods will also be used for package management and pulling in various libraries that will be used throughout the project. XCode 5 itself is a purpose built IDE for iOS applications and everything that was required came as standard and was used as a one stop solution for the development of the mobile application.

2.3.2 Wireframes

The main application will consist of 2 main views these being a list of events with some filter applied and the event details themselves. Figure 2.6 depicts a rough outline of where elements will be positioned and the possible interactions a user can undertake. Figure 2.8 shows the various different ways I could present the data using the iOS' table view to show list events happening, the list of events screen will be re-usable and able to show a list of events with a wide range of filters applied to it. The filters applied will be dependant on how the user access the list view be it through the `Discover' tab or the `My Feed' tab.

The wireframes show the 3 key different screens, the first being the `My Feed' tab this tab is used to show the users personalised feed, this will utilise the list view for the events and then when an event is selected it will show the user the selected event using the individual event view. The second is the `Discover' tab this will be used to be able to select events from a particular city, it will then give you the option to select a particular category or venue to see relevant events to the selector chosen. The third is the `My Profile' tab this will be where you can see your information potentially change it and view the events you are following, this will also present you with the option to `unfollow' these events as well.

Figure 2.6: Basic wireframes and design choices for the mobile application



Bacon ipsum dolor sit amet pork belly sausage pork loin leberkas pork chop flank. Short loin biltong ball tip, beef tail shankle pastrami meatball chuck chicken shank rump leberkas spare ribs. Chicken landjaeger capicola flank, spare ribs short loin jerky pancetta meatloaf doner. Pastrami cow landjaeger tri-tip. Jowl brisket corned beef chicken leberkas landjaeger.

Bresaola shank pork chop turkey boudin, flank ground round. Tongue pork loin corned beef, capicola chicken shoulder shank. Turducken salami tenderloin pork loin pancetta cow. Ham hock meatloaf filet mignon pork loin venison tenderloin frankfurter. Pas-

Figure 2.7: Individual event wireframe

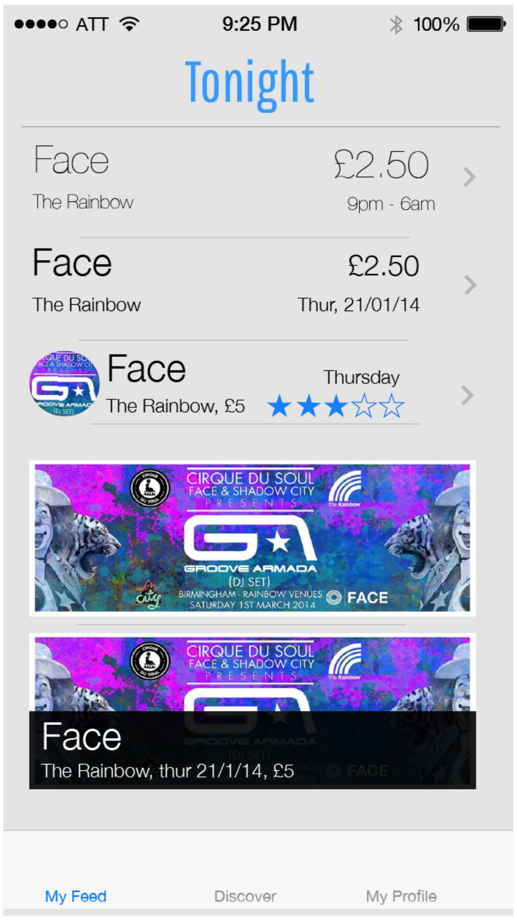


Figure 2.8: List of events wireframe

2.3.3 Class diagrams

The majority of the application design was based on the delegation design pattern where actions and data are linked with the UI elements, and so the mobile application is a series of controllers where it retrieves information and outputs this into the UI. There will be a different controller for each screen that can be viewed as part of the application. Including this I will be using data classes to store and use the data pulled in from the server. Figure 2.9 shows the 2 classes I will be using to keep the data retrieved from the API, these classes are simple data classes that have some functionality applied to them and allow for scope as the project develops.

Figure 2.2: This diagram depicts the various entities, and their interactions with each other.

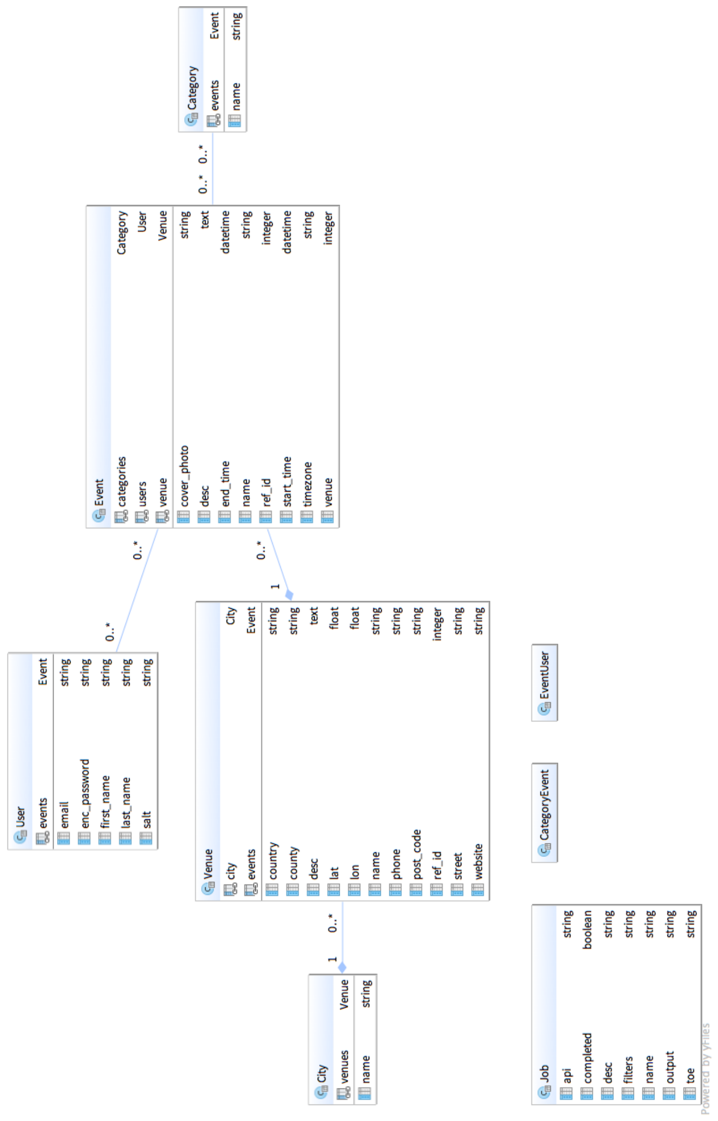


Figure 2.3: This diagram describes the functions and inheritance used for the library that connects to the APIs

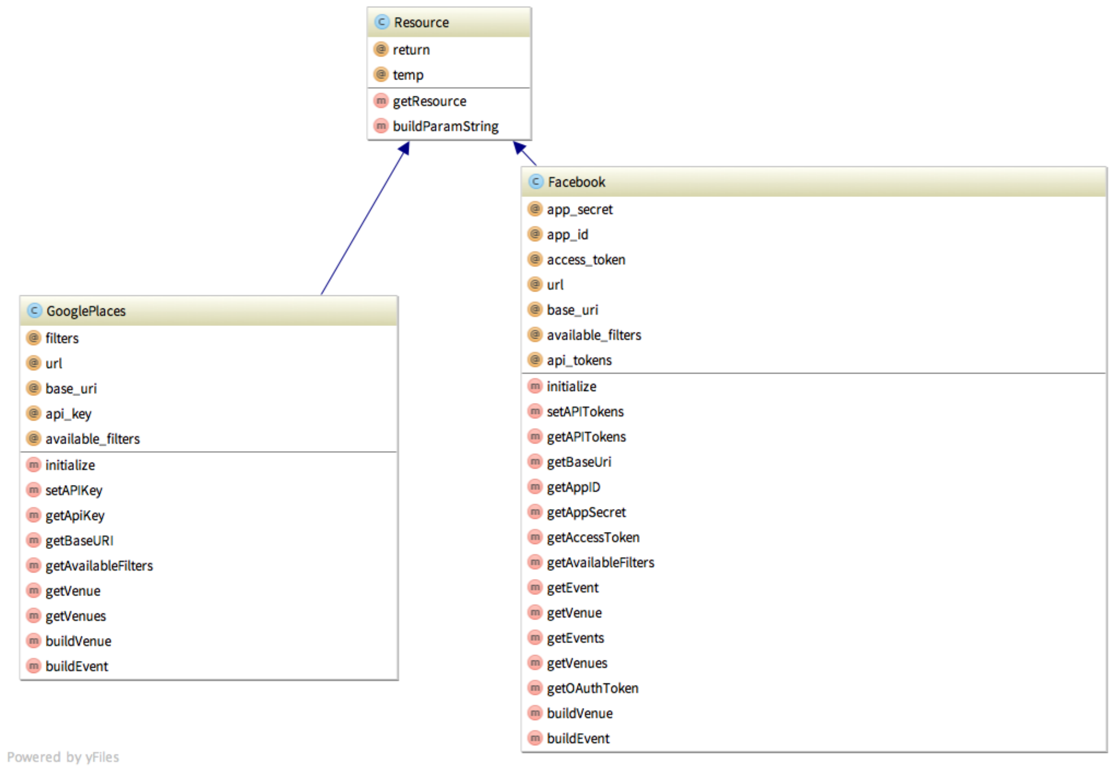


Figure 2.4: This diagram shows an overview of how the different data mining module is composed.

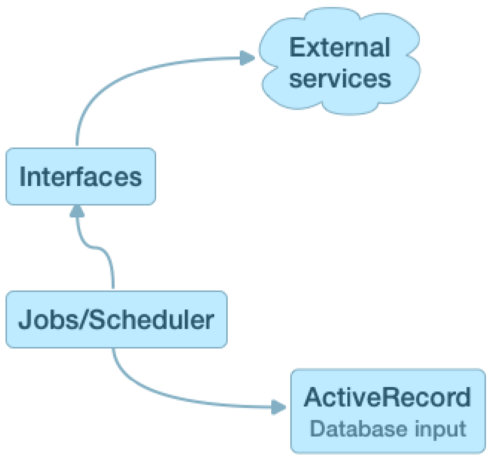


Figure 2.5: This diagram shows the classes along with the methods and class variables

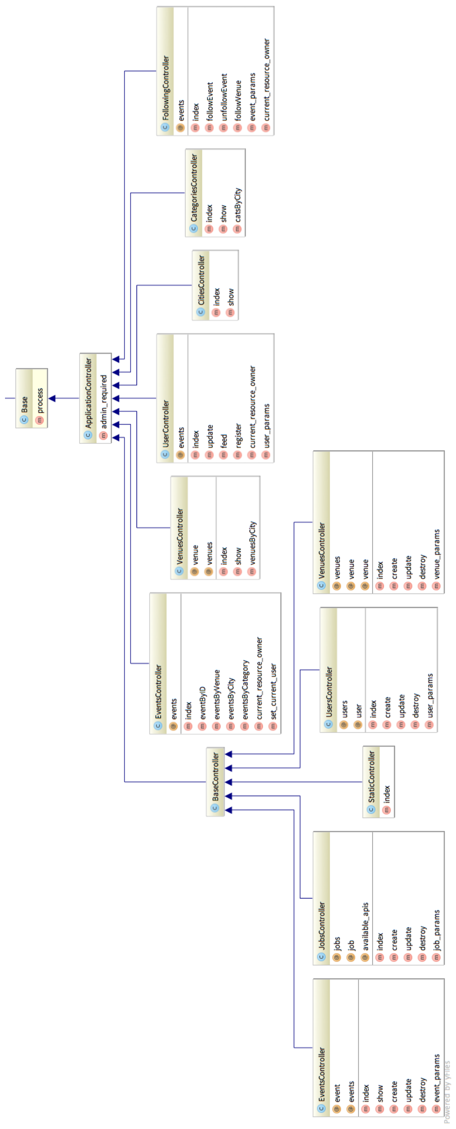
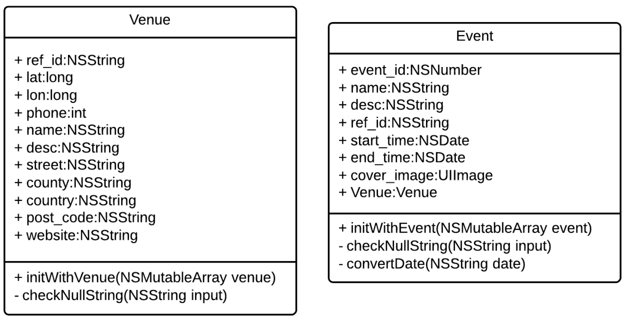


Figure 2.9: Class diagram for the iOS data classes



Chapter 3

Implementation

3.1 Personalised feed

The personalised feed is a feed of events that a user has either followed or suggested events based off what other users have followed. The main premise of this function written on the server side and then served by the API the mobile application connects to. To be able to get suggestions the application will retrieve a list of the currently logged in users followed events and then a search of the database for other users that are following one or more of those events. The users followed list is then retrieved which is then parsed, the parsing itself involves grouping the events together and counting how many times these appear then the up to the top 20 events are used to produce the personalised feed. As the mobile application utilised the feed this algorithm is used solely on the server element of the project as data transfer would be high, it gives centralisation to the algorithm and if it needed to be updated it can be with ease, it allows us to use SQL statements to easily filter the data and to cut the processing power needed. This system also does not limit the user to a particular genre or style of event, whereas a tag based suggestion system may do this ultimately allowing the user to see a wider selection of events.

However this algorithm requires some data to be input by the user and other users to function effectively. Other methods could include a user selecting tags that are of interest to them with the suggestion system working from that data to see what other people have viewed, however this may limit the suggestions to specific genre/style. A newly on boarded user will have nothing in their feed, therefore a list of events available in their current area should be presented to them when their suggestions are blank. The system will also have to have some training data with example users and events that are similar to each other, so that the initial users can make use of the functionality.

3.2 Registration and logging in

When a user was to first open the application they are presented with a login screen, where they are able to input their credentials if they have already registered or it gives them the ability to register a new account by selecting a different screen. The login utilises the OAuth library to authenticate the user and to respond with a access token to access the resources inside the API. However if a user was to input incorrect details they will be presented with an error, all off the validation is conducted on the server side application. This was done as if there was a large security bug or vulnerability in any part of the authentication process it can be patched quickly without the user needing to do anything, such as updating the application of which through Apple can take up to 2 weeks to process an update. If for example the user was to input incorrect credentials then an error message would be returned by the server stating the error, this is then relayed to the view so the user can see where the problem lies.

During the registration process a user must input their name, email and password, as this is a mobile based application users don't necessarily want to have to type things out multiple times. Therefore they will only type it the once within the `My Profile' view the user will then be presented with the opportunity to update these details if they mistyped/misspelled anything.

3.3 Storing passwords

The storing of passwords is a topic of constant debate of how best to do it, with considerations to the most secure method and how the data is transferred. The web API supports both HTTP and HTTPS connections, to ensure that no passwords are transmitted over a non-secure connection. However if an attacker was to somehow gain access to the users database table they will be able to see the passwords. To combat this the system will utilise BCrypt to encrypt the password, along with the it will also use a salt to harden the encryption. BCrypt was chosen as it's computationally expensive meaning it takes a while to convert a string into the encrypted version, this will slow down the use of rainbow tables. Also each individual user has their own salt, by doing this the attacker is required to generate a new rainbow table for each user slowing down the process even more. By taking these precautions to slow down the rate an attacker could get the passwords, we could warn users to change their passwords after patching the vulnerability with minimal impact to their services.

3.4 Search functionality

The search itself was implemented on the device using the built in search bar and search display controller that comes as a pre-set element in XCode 5. The filter is applied after every new character is input to the search field, and to conduct the search it uses the NSPredicate function that will look for a specific string or similar inside of a NSArray. Figure 2 shows how you use the NSPredicate function, first you define how the the filter should operate and then apply it to a NSArray to create a new NSArray. The search function I have implemented searches the event names for a name that contains the input string. This function could be improved by use of multiple conditions to allow more items to be searched, however the description can be very long and a user is already able to filter by venue within the discover area of the application.

The only issue with this method is that it doesn't search everything that's available on the server, only what's currently retrieved to the device. In the future to make this function work more effectively it will need to conduct a request to the server during the search process to retrieve it's results. However this method can be quite heavy as it will need to retrieve the event details along with images to show the results in the standard format resulting in a less responsive and slower user experience.

Example Code 2 Example of the use of NSPredicate in Objective C

```
NSPredicate *resultPredicate = [NSPredicate predicateWithFormat:@"name contains[c] %@", searchText];
searchResults = [sortedArray filteredArrayUsingPredicate:resultPredicate];
```

3.5 Adding in the following functionality

To add in the following functionality initially the route `user/following' was used, this route simply outputted all of the event ID's a user was following. When it came to using this feed on the mobile application it quickly became messy, as the mobile application had to 2 retrieve a minimum of two datasets whenever it needed to retrieve event information. Issues where also raised when passing data between the different views and having to pass the relevant information from the following array and persisting this data between the views. These 2 datasets where collated into 2 separate arrays and then used in tandem to show if the user was following that event. To help combat this I decided to add in a node inside the returned event information storing a boolean as to whether or not the current user is following that event, then the mobile application will not have to use the 2 different arrays.

To do this I created a function inside the event model that returned a boolean value dependant on whether the user is following the event or not. Doorkeeper allowed us to use the variable `current_resource_owner' this would hold the current users information so initially the idea was to use this inside the model to retrieve the current users information however this variable could only be accessed via the controllers. To combat this issue, an accessor was created inside the model, this allowed the model to store a bit of information not defined inside the database but defined by the controller. So whenever an event was accessed by a controller the current user was set using the `before_filter' function. Then to retrieve the following information the `following' function had to be called for each record, this was done by using the `:methods => :following' option of the `render' command making the code to output events to be`render json: @events, :methods => :following'.

3.6 Retrieving events from the jobs

One of the main parts of the server application is the way it retrieves the events from the various API's. The requirements state it has to be able to do this from multiple API's and to input the data to the same database that's to be used by the API. The main issue being is that each API uses different verbs to describe the data used and different methods to connecting to their API's to retrieve the data. So with that being said, the first step was to find the common ground between the major API's to connect to being Facebook and Google Places. The most basic similarity was they all connect over HTTP and use JSON to respond to requests, this was key as it meant I only had to use the one extra library to connect to the external API's. This was done using the library `httparty' as gave much better error handling over the standard http library bundled with ruby. A function to add in an GET parameters to append to the URL, this was mainly used to define the API key used to access that particular resource.

A class for each API was then created defining functions that were needed to access the different resources these are named the same and resembles what the function does such as getEvents got the events from that API with certain filters applied to them. Inside of the class a class variable was defined inside of which holds the various filters that can be applied to a request, giving the ability to filter specific events from the system such as events within a particular area. These functions helped with the issue of different API's connect differently, however the response data had to be parsed to match the database structure, so the `buildEvent' function was defined that simply took in the response hash and outputted it in the correct format for the database to use.

To create a new interface for a different API it's as simple as using one of the previous API interfaces and replacing the code inside the functions to resemble the correct way of connecting to the new API and it's response format. Once done and saved in the folder where the interfaces are, the administration panel will recognise a new interface and allow you to set up jobs to do retrieve information from the new API.

The job rake task simply loops through each job applies the filters, inputs the data retrieved assuming it's not already in the database. It uses the ID field from the original data source uses to check if it's currently their, however this needed to be refined as there may be clashes however this is a very unlikely situation and will only result in a missed event. The rake task itself can be ran multiple times and will only add in new events that have been found, however in the future the system needs to be able to recognise changes to the data as well as new data added to be able ensure it's up to date.

3.7 Interoperability

The method of which to transmit the data was quite a large part of the project, deciding whether to use SOAP or REST required some research into what the API was transferring and how the data was organised. SOAP itself is much heavier standard which is not as lenient as REST requests can be. SOAP requests are also generally requests for outputs from functions on the server such as getting the current stock rates for IBM. However REST is designed for more resource orientated architectures where supplying resources to be used in functions is the main game, bearing this in mind it makes sense to use a RESTFull approach as I want to be getting data and then manipulating it for my view on the mobile application. REST requests are accessed by a specific URI for each resource and can be very logical to what is needed to be retrieved.

A truly RESTFull API uses heavy linking inside of it's responses to link to other resources required, as this was mainly designed to link in with a mobile application the linking has been sacrificed for faster processing and data transfer times. However the URI structure to access the different resources, is well defined and logical so this isn't much of an issue.

To respond to the requests I decided to use JSON as this is a much looser notation meaning I get more flexibility on the how the requests and responses are retrieved/sent. Ruby on Rails also has a lot of built-in functionality for parsing and creating JSON strings. As the API could potentially be digested by a wide range of other systems as well it's crucial that the outputting format is readable by many other systems, and JSON offers the best cross language compatibility.

3.8 OAuth

To help with the authentication process of the application OAuth was used to authenticate users and applications to the API. To get an access token to use the API a user is required to login with their credentials along with a client secret and ID, by using the client secret and ID the system is able to differentiate different code bases utilising the API ultimately giving more fidelity to any restrictions to the use of the API. OAuth was implemented using the Doorkeeper gem, as OAuth is used for authentication use of the gem is great as it helps to eliminate code based vulnerabilities. Doorkeeper is also an actively worked on open source project so it gets regular updates and bug fixes applied to it.

To set up the Doorkeeper gem a function was needed inside the User model to authorise that simply returned the user details dependant on if the username/password combination was found inside the database. Once authorised an API access token was issues and this was used in conjunction with all requests to be able to get a response, if this was incorrect then a HTTP 401 code was served denying access to the resource. By use of a different authorisation code for each registered user the system is able to serve data dependant on the current user, allowing the URI /user/feed to be used with a different response for each user. This helped to follow the REST principles by ensuring the server didn't have to keep a specific state stored for each logged in user.

3.9 Storing user credentials

The mobile application must be able to store the credentials of a user, to be able to retrieve a new access token when the current one expires. iOS has use of what's called property list (PList) which are similar to an XML file where you can parse in the information wherever you need it around the application, however this PList could potentially be read in by other applications. Or if someone was to get physical access of the device they could read your username/password, and then potentially use that to access other devices. For this reason the system does not utilise PLists to store the data, iOS has a pre built key chain where you are able to store any sensitive information into the application. This information is only accessible by the application so it's a lot more secure than a file that's stored inside the application bundle, the data that resides inside the key chain is also encrypted to ensure it's privacy. To implement this feature a library was used called UICKeyChainStore [14], this was used as again it's a open source project and any bugs or issues can be found quickly which will assist with the overall security of the project.

3.10 Evaluation

Overall the implementation part of the project went well, the odd issue was hit however after analysing the issue and looking at possible workarounds that ended up being better ways of doing it the issues were worked out. Looking back at the proposed requirements of the system, most of the requirements where hit allowing the user to download the application register a new account and view events that are happening around them. The system also retrieved events from external sources and allowed a user to favourite them to allow a personalised feed, with event suggestions. In total the finished product completed the brief of allowing a user to explore events that's happening around them and in new cities.

One requirement that wasn't hit was having the system 100% location aware, whilst reviewing the data that's retrieved from the API's exact location for events was difficult to retrieve whilst many listed street addresses little longitude/latitudinal information was to hand, meaning the system had to make further requests to a different API to retrieve this information. This was something I didn't have time to implement meaning I had to sacrifice that requirement however the user is still able to view events by city so it doesn't remove the whole purpose of the application

Chapter 4

Testing

This section describes the approach to testing undertaken throughout the project, along with some results from the tests that where undertaken.

4.1 Overall Approach to Testing

To develop the server side element of the project the programmer followed a test driven approach utilising the red-green-refactor principle. By doing this it enabled the code to be as efficient as possible whilst still undertaking the task that block of code was expected to do. This method was chosen as the majority of the code required on the server side application didn't utilise a GUI and benefited from a much more pragmatic approach to testing. As part of the server element had to pull in information from external sources `WebMock' was used to mock HTTP connections to the external API's, this was used to serve test data to the code and to be able to check any transformations on the data.

To develop the mobile application a more behaviour driven approach was used, by writing the code and conducting a manual test seemed the more logical way. As most of the code was by use of target-action concept, meaning that the bulk of code was methods to be ran when a particular button is selected in the GUI. This also helped with the requirement testing, by ensuring that the requirements where in place and functioning on the mobile device itself.

4.2 Automated Testing

To conduct the automated tests on the server element of the project I used RSpec which is a testing environment for Ruby. RSpec has a version that's designed for the use within Ruby on Rails of which is the version used, RSpec gives the programmer the ability to test all aspects of the code including models, controllers and views. The programmer first wrote the tests that where needed for a particular feature or element of the program, and then implemented the function. However sometimes the test needed to be reviewed to ensure that it was testing the code correctly as, occasional odd results would appear after implementing a feature.

4.2.1 Stress Testing

Stress testing the server side element is pretty irrelevant due to the nature of the cloud based server it's running on we almost have unlimited resources to utilise if the server comes under significant stress. Heroku gives you the ability to add and remove physical resources such as CPU power and RAM whenever the demand is so high it's required. With the current scale of the application there is plenty of leeway to supply to the potential demand. If this demand became more persistent then it would be an adequate time to ensure the code is running as leanly as possible.

XCode 5 gives some great tools to be able to visualise how well the application is running on the mobile device, these include being able to see the current CPU and RAM usage the application is currently utilising. This data has been collated and used as part of the stress testing, currently the application only loads a certain number of events and so this is the maximum it will need to handle. However this number could go up as the server application builds a larger dataset of events, for this reason the resources used need to be as low as possible. Figures 4.1 shows the resources that's in use throughout the use of the application, as you can see this is a low number. The stress test was tested on a mid-level iOS device (iPhone 5), however it has been running smoothly on the lowest specification device available (iPhone 4).

Figure 4.1: Stess testing of the iOS Application

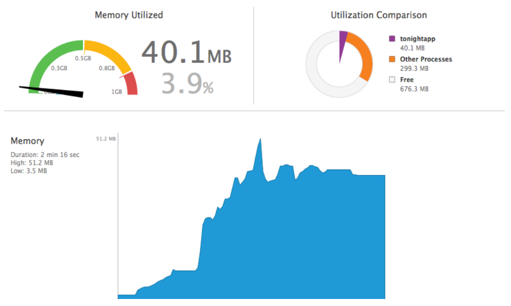


Figure 4.2: Memory usage during operation

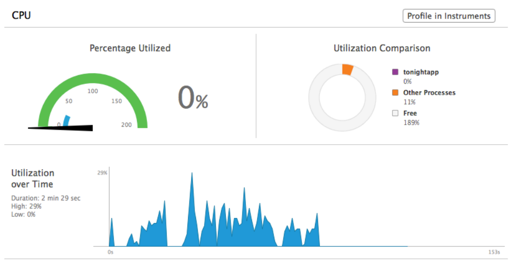


Figure 4.3: CPU usage during operation

4.3 Integration Testing

As the project was developed in the various different features, and tested inside of these features a big bang approach was taken for the integration testing. To do this I first built the sections and then using the development server option of Rails I set up a job and ran the rake task to pull in the event data for the job that was specified. This all worked without any issues, it pulled the data into the correct place and followed the right rules set by the job definition stored in the database.

Integration testing between the mobile application and server application was completed by checking that the application receives the events and parses these correctly into the views. Due to the RSpec tests used to test the server application we know that the API is outputting data as expected and that the functions work as expected, therefore it's just calling these URI's and parsing the output from them.

Chapter 5

Evaluation

Overall the project went smoothly without hitting any major issues along the way. I found that the server side application took up most of time, even knowing the main project was to produce a mobile application at the end of the project. However it was key that the server infrastructure was there to enable the mobile application to work as expected.

5.1 Testing

I found that test driven development worked really well for the server side application, being able to write the tests first allowed me to understand what the inputs and outputs where to be for each method. This ultimately gave me a much better understanding of how the inner logic would work, it also allowed me to easily drill down functions to find where code could be reused and abstracted into a separate function. However I did think I could follow the TDD principles much stronger throughout the course of the project, initially I found myself writing the tests first but as I was implementing a function to pass a test I would find some code that would be reused. I would then quickly pop this into a function without writing a test, ending up in having to retrograde the tests for these quick functions I implemented.

However when it came to the development of the iOS application I found that TDD became much more difficult because of the target action concept. I found that it was much easier to compile and run the application on the device and then test the newly implemented functionality, to ensure the target was linked together with the action. Throughout the development of the iOS application I also found myself conducting user interface testing, by moving around the elements and seeing what looks good and functions well where.

5.2 Requirements

As for fulfilling the requirements I managed to fulfil 13 out of 14 main requirements, however I did end up making the personalised feed a lot simpler than envisaged. This is mainly because the original idea of how the personalised feed was way over complicated and a similar output could be achieved by other methods, this worked nicely as I was able to concentrate my efforts on implementing the other requirements ensuring the resultant project was stable with good quality code.

The requirement I didn't implement was sending user notifications when an event was updated, this was primarily because the developer account we were using as part of the Universities development tools didn't support notifications and I was required to purchase a licence to implement this. Also I would have to add in some of version control to where the data was stored to be able read in these changes to the event and distribute the notifications whenever an event was updated.

In terms of implementing the other requirements undertaking the modular approach to each section, really helped me to concentrate on the section at hand. By just concentrating on the current feature I was able to ensure the code worked and was efficient at the task it was undertaking, from here I still had a good understanding of how the resultant project was going to fit together as each feature was a logical group of functionality. The overall roadmap helped with this as there was a logical roadmap, almost a waterfall of activities that one part of the system relied on another for the whole project to work. By understanding this I was really able to ensure that everything would work together, and function as required.

5.3 Tools used

I was well averse to using Sublime Text as a code editor and had used it for many other projects, however never for ruby development. The plugins I installed to assist me with the running the tests and snippets allowed me to code as I would inside any other editor. However I did still find myself going to terminal to run commands such as installing new gems and using the generator to generate models/controllers, this wasn't that much of an issue as I had used the terminal before. Some IDEs such as RubyMine already has a GUI to be able to generate code to be used inside the rails project, however I feel that they don't always give you the control you need that a raw text editor does.

Heroku allowed me to easily deploy the server application to the server simply by using a git push command to the remote Heroku server, once the new code was copied I could then remotely run things like db:migrate command to update the database and run other rake commands directly on the server from my own terminal. This gave me great control over what was happening on the production server, with other providers you don't always get this control.

XCode 5 I found to be a great IDE, it's purpose built for developing for iOS and has some great tools to write the code. The main one I found myself working on was the story board editor being able to drag and drop elements into the view currently being edited was great. Then it was as simple as dragging and dropping a connector to the code to link an element with code, this simplified the process ten fold. With Android you had to name an element and then use the findElementByID function to refer to it which can quickly become messy. XCode also houses a simulator to run your applications inside of, the benefits off a simulator over an emulator are again so much better. Emulators tend to be buggy and can take a long time to boot, whereas the iOS simulator was quick and always ran allowing a much quicker rate of development.

5.4 iOS App

The development of the iOS application I found to be a lot more tedious than the server application this I think this was due to not knowing any objective C and only learning it from the online courses put on by Harvard. However after looking into some sample projects on the ray wenderlich site I understood how the applications were structured and how to interact with the elements positioned on the storyboard. I found that Objective C is a very readable language in the way the methods are set out, for example the login method was ` [loginUser:``username'' withPassword:``Password'']' this method clearly defines what's required as inputs and what the function does.

The use of cocoa pods was a nice addition as well, a very similar system to the way Ruby Gems are installed you have a Pod file which defines the libraries that you wish to use and run the command pod install. Once installed you can use the import command to import the library into the location you wish to use it, I opted to use a cocoa pod over the httpRequest library to interface with the API as it gave much better error handling and also transformed the response it an NSDictionary which was a much easier format to deal with.

I found that developing the GUI for the mobile application to be a lot harder than initially thought, making sure everything was in a place and presenting the information in a readable format was difficult. The sheer amount of data that was required to be displayed was a lot and needed to be presented in a format that a user would find appealing yet informative, however the wireframes where reminiscent of the how the application ended up looking like which was good. However I did remove the star rating system, as this proved to be quite difficult to implement and wasn't needed for the recommendation system I was implementing.

Another improvement would be to download the images to be used as part of the mining process and scale and process these images to load a lot quicker onto the mobile application, these images could also be served by the API itself as base 64 encoded images or as image files dependant on business factors. The images currently being outputted are ones stored on the Facebook severs and are in a larger format than needed for the application itself.

5.5 Server Application

The development of the server part of the application overall went well, there was the odd bug with the code but I found following the TDD methodology everything slotted in nicely and just worked well when it came to the integration of each module. Using the Ruby on Rails really helped to accelerate my workflow allowing a lot of helper functions that in other languages I would of have needed to of have written doubling the work load. I also found that the use of Ruby Gems assisted me a lot with the development of the server application in the sense that the use cases I needed code for where already used and people had open sourced solutions for them.

However at times I did find I was having to constantly re-run the RSpec tests after every change to ensure they still passed which was a little troublesome and could sometimes take a long time to prepare dependant on the complexity of the tests that where being ran. I did attempt to speed this up by always having a test server running using spork, but the test cases had to still remove the information from the database and re-add the test data every time a test was ran. However this was really not irrelevant to the benefits that I was getting from the TDD approach, overall I found I had a much more usable system than that was originally conceived during the design phase.

I also found that when it came to standardising the output of the API ensuring that the requests where the same format I found this to be difficult as I had to change every function that returned an API response. This could of have been utilised with a before_filter or similar for all API responses, however I didn't start off doing it this way so I ended up doing it the long way.

5.6 User expectations

Overall the people that used my application in passing found that the screens where all laid it in a logical manner, and that it was easy to navigate around the application. They also noted that buttons where represented correctly, and they where also self explanatory on what they did. The feed was quite clearly a feed of events that you are following and suggested events, and the overall looks were aesthetically pleasing. The search function is great as it's you don't need to enter a full search string it just searches after every key press.

However inside the profile page when clicking on the events being followed table it's expected that when you click on an event you are following it should unfollow the event. Although it currently does not and is somewhat illogical to not do this.

5.7 Reflections

If I was to do this project again I would work on ensuring that I wrote tests for the iOS application and making sure that the tests where passable from the offset would of made the the development of the project a lot easier. Also having solid wireframes of how the views where to be set out, this would of have made the overall design of the application a lot more use friendly, and eased the development process as I found I was spending more time worrying about the aesthetics of the application than the functionality of it.

I would also make a concious effort to learn any programming languages I hadn't used to a higher level than I did within this project. As I found myself stumbling through the development of the iOS application, as I had never written in objective C before. That being said there where some great resources on the internet and I definitely utilised these, and managed to absorb enough knowledge to be able develop the mobile application.

Finally I wouldn't follow TDD for a whole project, it's a great process that worked really well for my server side element, however as stated it didn't work as well when used with the iOS application. Development process are really a mixture between what the project is and personal choice unto how you want to develop the code, however a formal QA process is necessary to ensure code quality and test cases passing.

Appendices

Appendix A

Third-Party Code and Libraries

1.1 Server side libraries

Ruby on Rails is the base framework I used for my server application, RoR comes packaged with a host of other libraries all used as well. [18]. The library is released using the MIT License [19]. This library was used without modification.

The packages Kaminari and api-pagination where used to add in pagination to the project [18]. Both libraries is released using the MIT License [1]. This library was used without modification. [4]. This library was used without modification.

The packages RSpec, FactoryGirl, WebMock, spork-rails, guard-spork, and childprocess where all used as part of the testing setup. [18]. All libraries where released under the MIT LIcense. [**?**]. This library was used without modification. [28]. This library was used without modification. [2]. This library was used without modification. [24]. This library was used without modification. [10]. This library was used without modification.

The httparty library was used to connect to external APIs [18]. This library where released under the MIT LIcense. [11]. This library was used without modification.

1.2 Mobile application libraries

UICKeyChainStore was used to store personal data using the iOS Key Chain [18]. This library where released under the MIT LIcense. [14]. This library was used without modification.

UniRest was used to retrieve data from the API over HTTP [18]. This library where released under the MIT LIcense. [17]. This library was used without modification.

MBProgressHUD was used to display a progress bar when undertaking certain tasks [18]. This library where released under the MIT LIcense. [13]. This library was used without modification.

Appendix B

Test Results

2.1 RSpec Results

```
...FF...F..*....*.......*..............*.........*.*.F.............*.......*..******..............*.*...........*

Pending:
  API::V1::UserController.feed Implement the feed
    # Not yet implemented
    # ./spec/controllers/api/v1/user_controller_spec.rb:45
  Admin::EventsController.update redirects to index after update
    # Not yet implemented
    # ./spec/controllers/admin/events_controller_spec.rb:46
  Admin::EventsController.create shows flash message when an error occurs
    # Not yet implemented
    # ./spec/controllers/admin/events_controller_spec.rb:31
  Admin::UsersHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/
        tonight_server/spec/helpers/admin/users_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/admin/users_helper_spec.rb:14
  Admin::JobsController.create shows flash message when an error occurs
    # Not yet implemented
    # ./spec/controllers/admin/jobs_controller_spec.rb:35
  Admin::UsersController.update redirects to the index page
    # Not yet implemented
    # ./spec/controllers/admin/users_controller_spec.rb:50
  Admin::VenuesController.update redirects to the index page
    # Not yet implemented
    # ./spec/controllers/admin/venues_controller_spec.rb:46
  Admin::VenuesController.create shows flash message when an error occurs
    # Not yet implemented
    # ./spec/controllers/admin/venues_controller_spec.rb:31
  Venue add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/tonight_server/spec/
        models/venue_spec.rb
    # Not yet implemented
    # ./spec/models/venue_spec.rb:4
  FollowingControllerHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/
        tonight_server/spec/helpers/following_controller_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/following_controller_helper_spec.rb:14
  Admin::EventsHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/
        tonight_server/spec/helpers/admin/events_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/admin/events_helper_spec.rb:14
  Admin::VenuesHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/
        tonight_server/spec/helpers/admin/venues_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/admin/venues_helper_spec.rb:14
  StaticHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/tonight_server/
        spec/helpers/static_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/static_helper_spec.rb:14
  Admin::JobsHelper add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/
        tonight_server/spec/helpers/admin/jobs_helper_spec.rb
    # Not yet implemented
    # ./spec/helpers/admin/jobs_helper_spec.rb:14
  API::V1::FollowingController Implement the following venue stuff
    # Not yet implemented
    # ./spec/controllers/api/v1/following_controller_spec.rb:34
  API::V1::FollowingController.index responds with events logged in user follows
    # Not yet implemented
    # ./spec/controllers/api/v1/following_controller_spec.rb:20
  City add some examples to (or delete) /Users/user/Dropbox/dissertation/code/server_software/tonight_server/spec/
        models/city_spec.rb
    # Not yet implemented
    # ./spec/models/city_spec.rb:4

Failures:

  1) Facebook gets an event from it's ID
     Failure/Error: stub_request(:get, "https://graph.facebook.com/204504813084059/?access_token=accesstoken").
          to_return(:body => event);
     NameError:
       undefined local variable or method `event' for #<RSpec::ExampleGroups::Facebook_2:0x007f8e3f0532a8>
     # ./spec/facebook_spec.rb:49:in `block (2 levels) in <top (required)>'

  2) Facebook searches for various venues
     Failure/Error: expect(@facebook.getVenues(params)).to eq(venuesJSON);

       expected: {"data"=>[{"category"=>"Arts/entertainment/nightlife", "category_list"=>[{"id"=>"211155112228091", "
            name"=>"Event Venue"}, {"id"=>"191478144212980", "name"=>"Night Club"}, {"id"=>"179943432047564", "name
            "=>"Concert Venue"}], "name"=>"The Rainbow Venues", "id"=>"224699987546240"}], "paging"=>{"next"=>"https
            ://graph.facebook.com/search?type=place&center=52.41649271,—4.07785633&distance=500&limit=5000&offset
            =5000&__after_id=134964199922380"}}
            got: [{"category"=>"Arts/entertainment/nightlife", "category_list"=>[{"id"=>"211155112228091", "name"=>"
                 Event Venue"}, {"id"=>"191478144212980", "name"=>"Night Club"}, {"id"=>"179943432047564", "name"=>"
                 Concert Venue"}], "name"=>"The Rainbow Venues", "id"=>"224699987546240"}]

       (compared using ==)

       Diff:
      @@ —1,3 +1,8 @@
      —"data" => [{"category"=>"Arts/entertainment/nightlife", "category_list"=>[{"id"=>"211155112228091", "name"=>"
            Event Venue"}, {"id"=>"191478144212980", "name"=>"Night Club"}, {"id"=>"179943432047564", "name"=>"
            Concert Venue"}], "name"=>"The Rainbow Venues", "id"=>"224699987546240"}],
      —"paging" => {"next"=>"https://graph.facebook.com/search?type=place&center=52.41649271,—4.07785633&distance
            =500&limit=5000&offset=5000&__after_id=134964199922380"},
      +[{"category"=>"Arts/entertainment/nightlife",
      +  "category_list"=>
      +   [{"id"=>"211155112228091", "name"=>"Event Venue"},
      +    {"id"=>"191478144212980", "name"=>"Night Club"},
      +    {"id"=>"179943432047564", "name"=>"Concert Venue"}],
      +  "name"=>"The Rainbow Venues",
      +  "id"=>"224699987546240"}]

     # ./spec/facebook_spec.rb:76:in `block (2 levels) in <top (required)>'

  3) Facebook gets a list of events from a venue
     Failure/Error: @facebook.getEvents("204504813084059").should eq(eventsJSON);

       expected: {"data"=>[{"name"=>"Godskitchen Presents Clash Of The Gods", "start_time"=>"2014—04—26T22
            :00:00+0100", "end_time"=>"2014—04—27T06:00:00+0100", "timezone"=>"Europe/London", "location"=>"The
            Rainbow Venues", "id"=>"204504813084059"}], "paging"=>{"cursors"=>{"after"=>"MzI5Mzc5MDYzODYwOTM5", "
            before"=>"MjA0NTA0ODEzMDg0MDU5"}, "next"=>"https://graph.facebook.com/224699987546240/events?limit=25&
            after=MzI5Mzc5MDYzODYwOTM5"}}
            got: [{"name"=>"Godskitchen Presents Clash Of The Gods", "start_time"=>"2014—04—26T22:00:00+0100", "
                 end_time"=>"2014—04—27T06:00:00+0100", "timezone"=>"Europe/London", "location"=>"The Rainbow Venues
                 ", "id"=>"204504813084059"}]

       (compared using ==)

       Diff:
      @@ —1,3 +1,7 @@
      —"data" => [{"name"=>"Godskitchen Presents Clash Of The Gods", "start_time"=>"2014—04—26T22:00:00+0100", "
            end_time"=>"2014—04—27T06:00:00+0100", "timezone"=>"Europe/London", "location"=>"The Rainbow Venues", "id
            "=>"204504813084059"}],
      —"paging" => {"cursors"=>{"after"=>"MzI5Mzc5MDYzODYwOTM5", "before"=>"MjA0NTA0ODEzMDg0MDU5"}, "next"=>"https://
            graph.facebook.com/224699987546240/events?limit=25&after=MzI5Mzc5MDYzODYwOTM5"},
      +[{"name"=>"Godskitchen Presents Clash Of The Gods",
      +  "start_time"=>"2014—04—26T22:00:00+0100",
      +  "end_time"=>"2014—04—27T06:00:00+0100",
      +  "timezone"=>"Europe/London",
      +  "location"=>"The Rainbow Venues",
      +  "id"=>"204504813084059"}]

     # ./spec/facebook_spec.rb:67:in `block (2 levels) in <top (required)>'

  4) Admin::UsersController.update updates a record
     Failure/Error: expect(@user[:updated_at]).to_not eq(prev_updated);

       expected: value != 2014—05—09 08:05:21.088558000 +0000
            got: 2014—05—09 08:05:21.088558000 +0000

       (compared using ==)

       Diff:

     # ./spec/controllers/admin/users_controller_spec.rb:56:in `block (3 levels) in <top (required)>'

Finished in 6.21 seconds
112 examples, 4 failures, 17 pending

Failed examples:

rspec ./spec/facebook_spec.rb:44 # Facebook gets an event from it's ID
rspec ./spec/facebook_spec.rb:70 # Facebook searches for various venues
rspec ./spec/facebook_spec.rb:62 # Facebook gets a list of events from a venue
rspec ./spec/controllers/admin/users_controller_spec.rb:52 # Admin::UsersController.update updates a record


Randomized with seed 47516
```

Annotated Bibliography

[1] Akira Matsuda, ``Kaminari,'' https://github.com/amatsuda/kaminari, 2014.

The cloud server being used to deploy to a production ready state

[2] Bartosz Blimke, ``WebMock,'' https://github.com/bblimke/webmock, 2014.

The cloud server being used to deploy to a production ready state

[3] Cocoa Pods, ``CocoaPods.org - The Dependency Manager for Objective C. ,'' http://cocoapods.org/, 2014.

[4] David Celis, ``Rails API Pagination,'' https://github.com/davidcelis/api-pagination, 2014.

[5] Facebook, ``Facebook Graph API Documentation,'' https://developers.facebook.com/docs/graph-api, 2014.

[6] GitHub, ``GitHub - Hosted GIT Server,'' http://www.github.com/, 2014.

[7] Greg Williams, ``RSpec Package for Sublime Text 2/3,'' https://github.com/SublimeText/RSpec, 2014.

The cloud server being used to deploy to a production ready state

[8] HBAA, ``Opportunities for Growth in the UK Events Industy,'' http://www.hbaa.org.uk/sites/default/files/Opportunities%20for%20Growth%20in%20the%20UK%20Events%20Industry.pdf, 2014.

[9] Heroku, ``Heroku - Cloud Application Platform,'' https://www.heroku.com/, 2014.

The cloud server being used to deploy to a production ready state

[10] Jari Bakken, ``childprocess,'' https://github.com/jarib/childprocess, 2014.

The cloud server being used to deploy to a production ready state

[11] John Nunemaker, ``httparty,'' https://github.com/jnunemaker/httparty, 2014.

The cloud server being used to deploy to a production ready state

[12] Jon Skinner, ``Sublime Text 2,'' http://www.sublimetext.com/2, 2014.

The cloud server being used to deploy to a production ready state

[13] Jonathan George, ``MBProgressHUD,'' https://github.com/jdg/MBProgressHUD, 2014.

[14] kishikawa katsumi, ``UICKeyChainStore,'' https://github.com/kishikawakatsumi/UICKeyChainStore, 2014.

[15] Line Up, ``Line Up - Events in Manchester,'' http://lineupnow.com/, 2014.

[16] Maciej Gajek, ``Sublime Text 2 Ruby Tests,'' https://github.com/maltize/sublime-text-2-ruby-tests, 2014.

The cloud server being used to deploy to a production ready state

[17] Mashape, ``UniRest,'' https://github.com/Mashape/unirest-obj-c, 2014.

[18] MIT, ``MIT Licence,'' http://opensource.org/licenses/MIT, 2014.

The cloud server being used to deploy to a production ready state

[19] Rails, ``Ruby on Rails,'' https://github.com/rails/rails, 2014.

The cloud server being used to deploy to a production ready state

[20] Ray Wenderlich, ``Ray Wenderlich | Tutorials for iPhone / iOS Developers and GamersRay Wenderlich | Tutorials for iPhone / iOS Developers and Gamers,'' www.raywenderlich.com, 2014.

[21] RSpec, ``RSpec Package for Ruby on Rails,'' https://github.com/rspec/rspec-rails, 2014.

The cloud server being used to deploy to a production ready state

[22] schema.org, ``Internationally recognised schema for events,'' http://schema.org/Event, 2014.

[23] Songkick, ``SongKick - Concerts, tour dates, and festivals for your favorite artists,'' http://www.songkick.com/, 2014.

[24] sporkrb, ``spork-rails,'' https://github.com/sporkrb/spork-rails, 2014.

The cloud server being used to deploy to a production ready state

[25] Spotnight, ``Spotnight -Nights out in London,'' http://spotnightapp.com/, 2014.

[26] Stefano Schiavi, ``railsdev-sublime-snippets,'' https://github.com/j10io/railsdev-sublime-snippets, 2014.

The cloud server being used to deploy to a production ready state

[27] The Rainbow Venues, ``The Rainbow Venues,'' http://www.therainbowvenues.co.uk/, 2014.

[28] thoughtbot, inc., ``FactoryGirl,'' https://github.com/thoughtbot/factory_girl, 2014.

The cloud server being used to deploy to a production ready state

[29] Trello, ``Trello - Project Management,'' http://www.trello.com/, 2014.