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

LIST OF FIGURES

LIST OF TABLES

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. [6] 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 [17] 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.

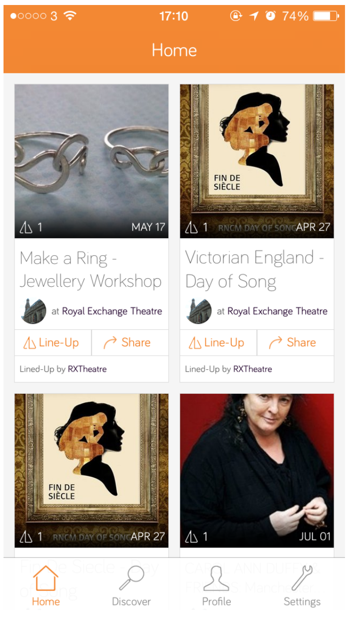
1.1.1 Current Solutions

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 [9] 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.

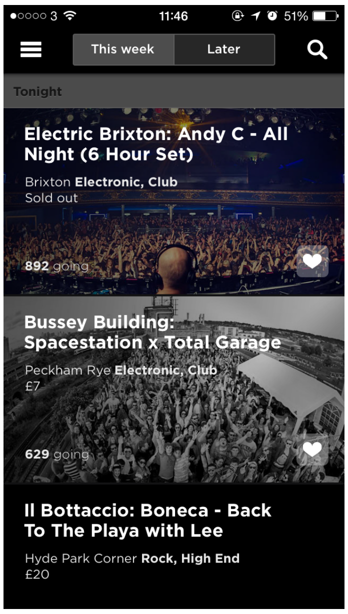
Figure 1.1: Image showing the home screen for the Line Up application



1.1.1.2 Spotnight

Spotnight [15] 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



Electric Brixton: Andy C - All Night (6 Hour Set)
Brixton **Electronic, Club**
Sold out
892 going

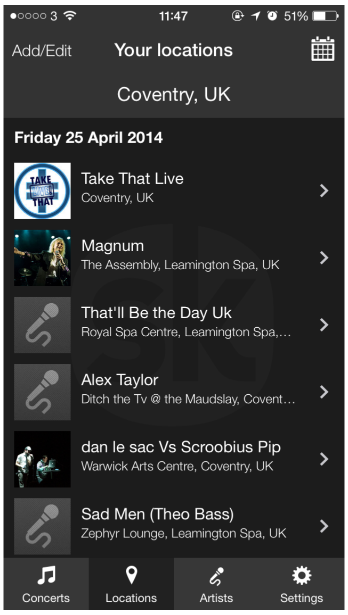
Bussey Building: Spacestation x Total Garage
Peckham Rye **Electronic, Club**
£7
629 going

Il Bottaccio: Boneca - Back To The Playa with Lee
Hyde Park Corner **Rock, High End**
£20

1.1.1.3 Songkick

Songlick [14] 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

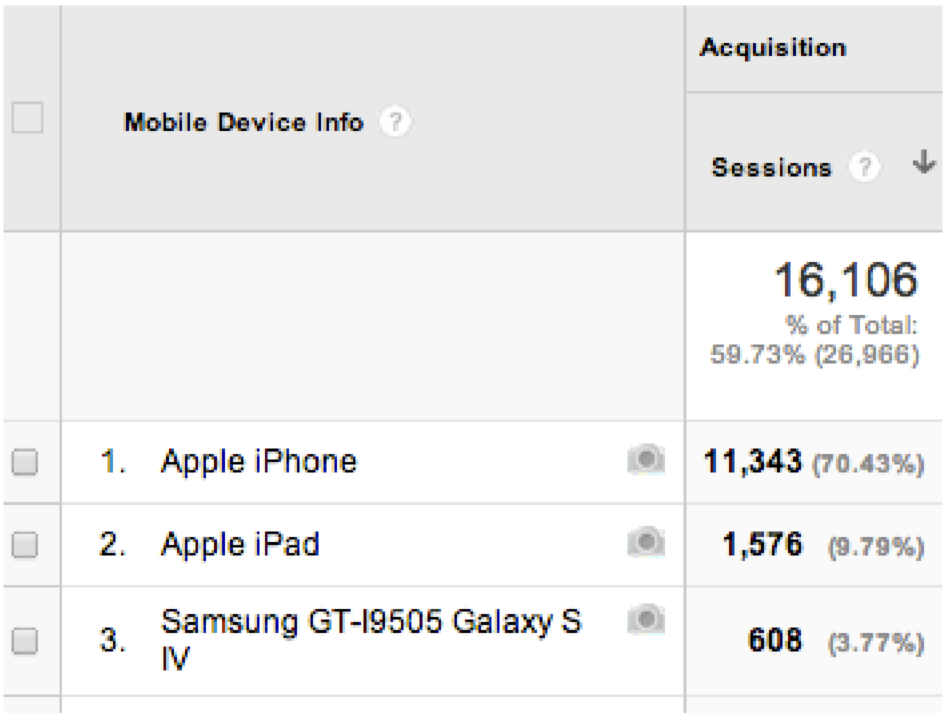
1.3.1 iOS Development

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 [2] 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 [11], 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 [7] 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 [3], 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.

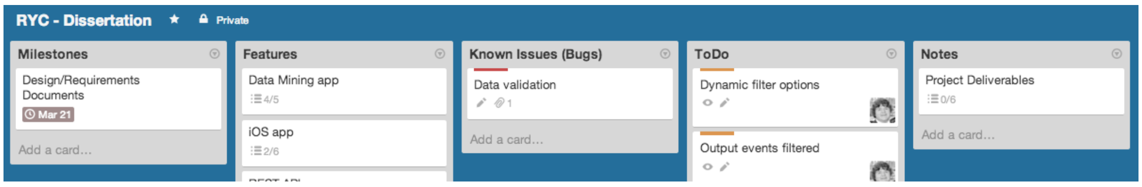
1.4.3 GitHub

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.



RYC - Dissertation ★ Private

Milestones
- Design/Requirements Documents — Mar 21

Features
- Data Mining app — 4/5
- iOS app — 2/6

Known Issues (Bugs)
- Data validation — 1

ToDo
- Dynamic filter options
- Output events filtered

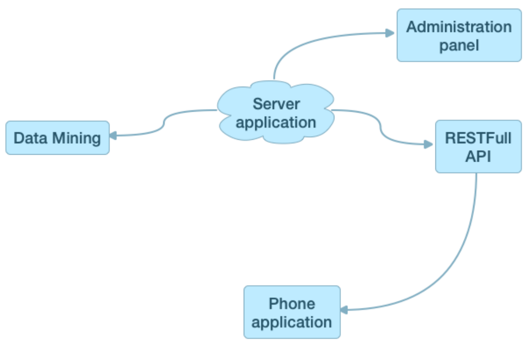
Notes
- Project Deliverables — 0/6

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 [13] 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 [8] with the following packages installed; RSpec [5], Ruby Test [10], and Rails Developer Snippets [16]. 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 [12], 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 [18] a higher grain of control over mock models, and WebMock [1] 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 [**?**] 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

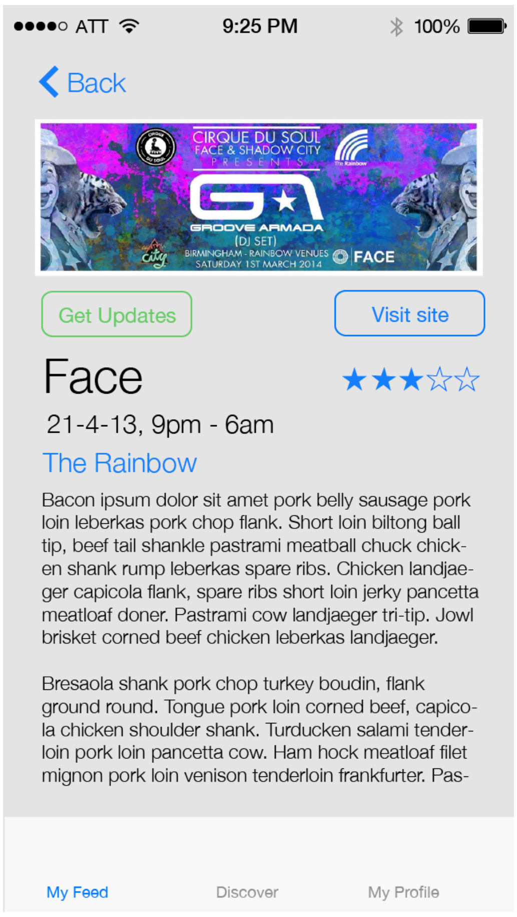
To develop in iOS you are required to use Apples own IDE XCode, version 5 is the latest and supports the latest version of iOS. XCode 5 includes a number of tools bundled with the IDE including the iOS Simulator and GIT Integration. Cocoa pods will also be used for package management and pulling in various 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

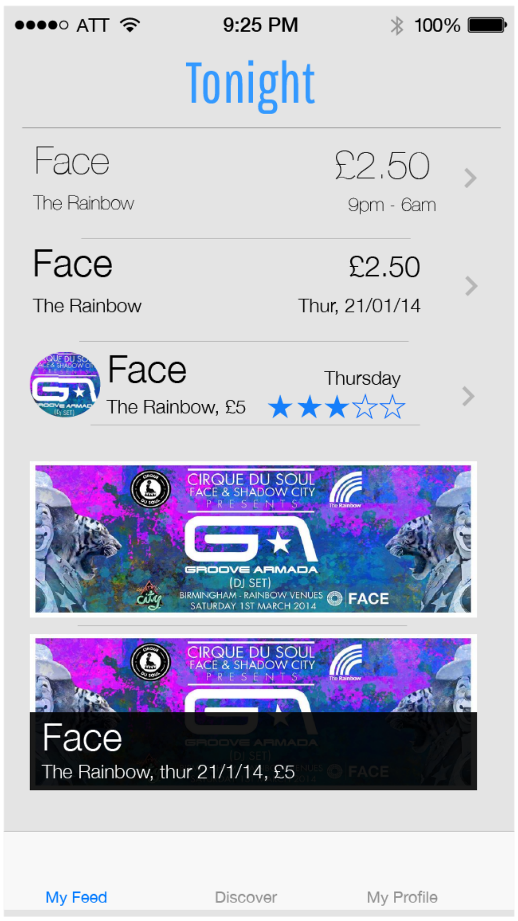


Figure 2.8: List of events wireframe

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.

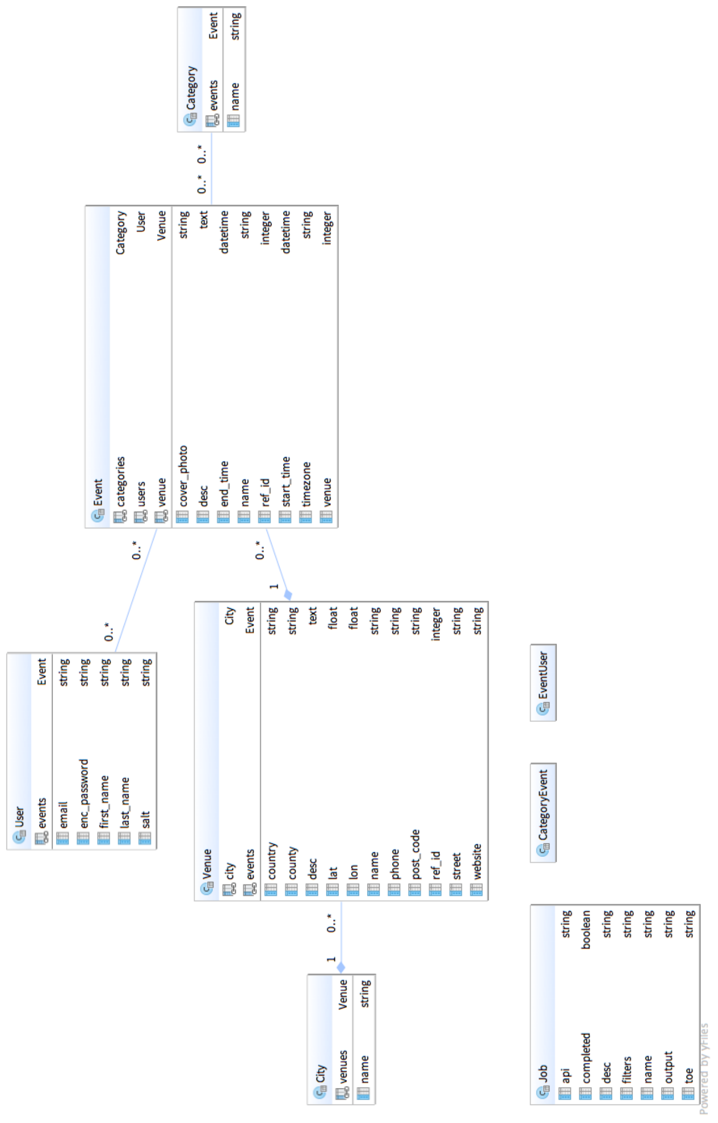


Figure 2.3: This diagram describes the functions and inheritance used for the library that connects to the APIs

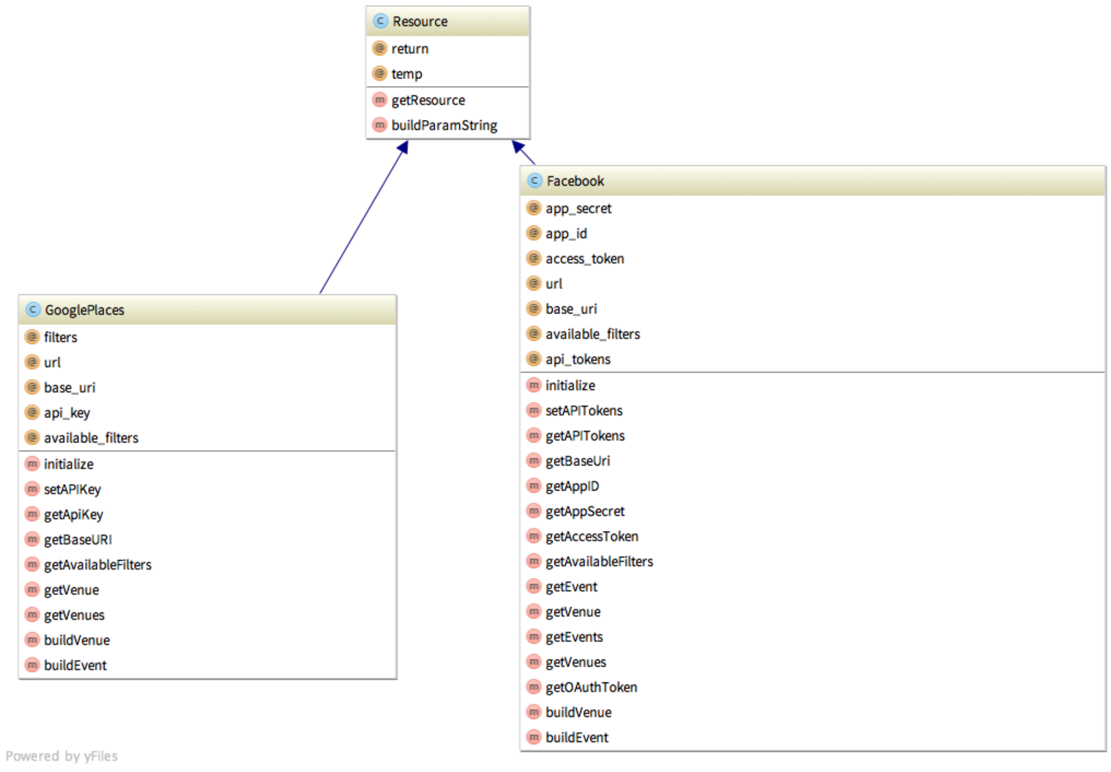


Figure 2.4: This diagram shows an overview of how the different data mining module is composed.

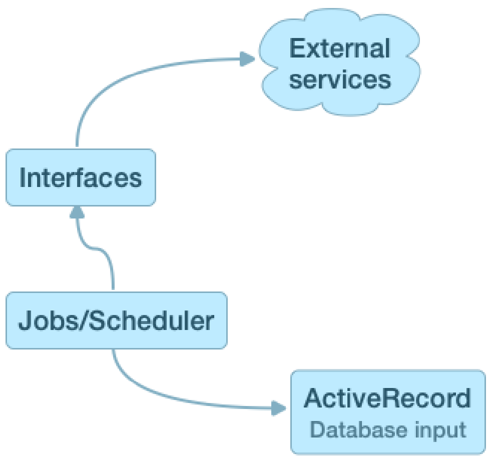


Figure 2.5: This diagram shows the classes along with the methods and class variables

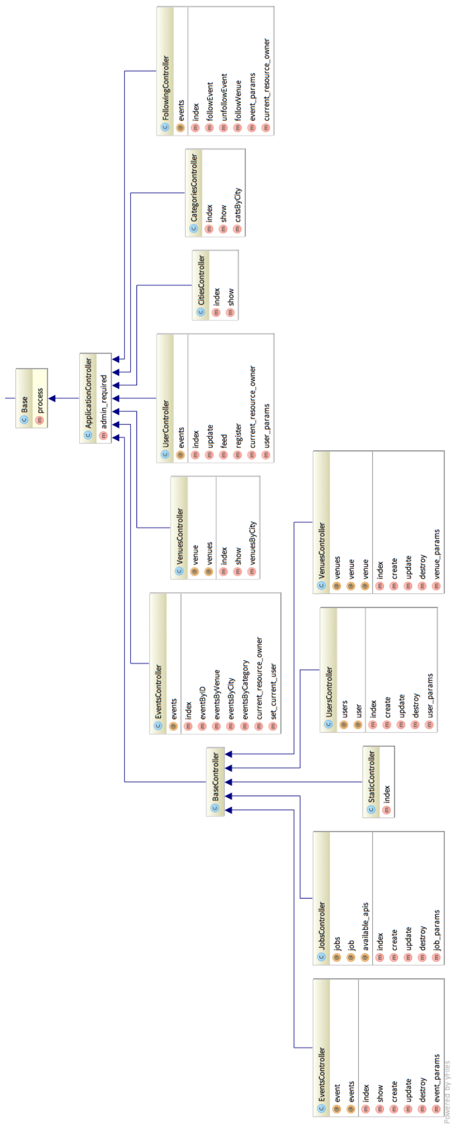
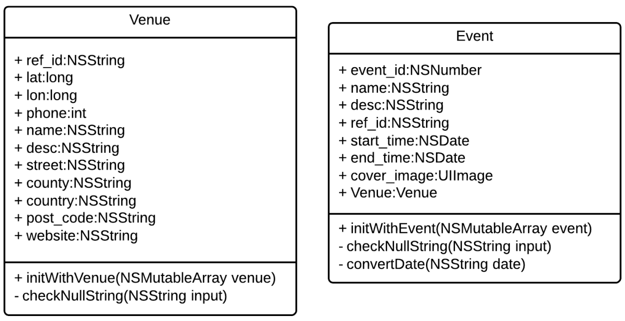


Figure 2.9: Class diagram for the iOS data classes



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 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.

3.3 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];
```

3.4 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'.

3.5 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.

3.6 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.

3.7 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.

3.8 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

Detailed descriptions of every test case are definitely not what is required here. What is important is to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Have you tested your system on □real users□? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

4.1 Overall Approach to Testing

4.2 Automated Testing

4.2.1 Unit Tests

4.2.2 User Interface Testing

4.2.3 Stress Testing

4.2.4 Other types of testing

4.3 Integration Testing

4.4 User Testing

Chapter 5

Evaluation

Examiners expect to find in your dissertation a section addressing such questions as:

- Were the requirements correctly identified?
- Were the design decisions correct?
- Could a more suitable set of tools have been chosen?
- How well did the software meet the needs of those who were expecting to use it?
- How well were any other project aims achieved?
- If you were starting again, what would you do differently?

Such material is regarded as an important part of the dissertation; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things and room for improvement with any project. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

Review the discussion on the Evaluation section from the lectures. A recording is available on Blackboard.

Appendices

Appendix A

Third-Party Code and Libraries

If you have made use of any third party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. The key requirement is that we understand what is your original work and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

As an example, you might include a definition such as:

Apache POI library □ The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client□s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation [**?**]. The library is released using the Apache License [**?**]. This library was used without modification.

Appendix B

Code samples

2.1 Random Number Generator

The Bayes Durham Shuffle ensures that the psuedo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs [**?**].

```
#define IM1 2147483563
#define IM2 2147483399
#define AM (1.0/IM1)
#define IMM1 (IM1-1)
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+IMM1/NTAB)
#define EPS 1.2e-7
#define RNMX (1.0 - EPS)

double ran2(long *idum)
{
  /*--------------------------------------------------*/
  /* Minimum Standard Random Number Generator         */
  /* Taken from Numerical recipies in C               */
  /* Based on Park and Miller with Bays Durham Shuffle */
  /* Coupled Schrage methods for extra periodicity     */
  /* Always call with negative number to initialise    */
  /*--------------------------------------------------*/

  int j;
  long k;
  static long idum2=123456789;
  static long iy=0;
  static long iv[NTAB];
  double temp;

  if (*idum <=0)
  {
    if (-(*idum) < 1)
    {
      *idum = 1;
    }else
    {
      *idum = -(*idum);
    }
    idum2=(*idum);
    for (j=NTAB+7;j>=0;j--)
    {
      k = (*idum)/IQ1;
      *idum = IA1 *(*idum-k*IQ1) - IR1*k;
      if (*idum < 0)
      {
        *idum += IM1;
      }
      if (j < NTAB)
      {
        iv[j] = *idum;
      }
    }
    iy = iv[0];
  }
  k = (*idum)/IQ1;
  *idum = IA1*(*idum-k*IQ1) - IR1*k;
  if (*idum < 0)
  {
    *idum += IM1;
  }
  k = (idum2)/IQ2;
  idum2 = IA2*(idum2-k*IQ2) - IR2*k;
  if (idum2 < 0)
  {
    idum2 += IM2;
  }
  j = iy/NDIV;
  iy=iv[j] - idum2;
  iv[j] = *idum;
  if (iy < 1)
  {
    iy += IMM1;
  }
```

```
  if ((temp=AM*iy) > RNMX)
  {
    return RNMX;
  }else
  {
    return temp;
  }
}
```

Annotated Bibliography

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

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

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

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

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

[5] 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

[6] 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.

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

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

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

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

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

[10] 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

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

[12] 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

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

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

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

[16] 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

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

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

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

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