National College of Ireland
BSc in Computing
2015/2016

Damien Fenton
11104589
damien.fenton@student.ncirl.ie

Tripency

Technical Report

Table of Contents

Executive Summary

Tripency is a system that allows tourists to quickly match attractions in a particular location to their interests and disinterests. The methodology of Tripency is to apply the same technology and algorithmic logic used to match couples in dating websites to match travellers with attractions that best suit their interests. The benefit of this system is that users are able to quickly and efficiently build their travel dynamic itinerary rather than relying on “out of the box” itineraries found on other travel websites or print media.

Tripency will enjoy full functionality that one would expect from a modern website including social media connectivity and the ability for users to view their itineraries on-the-go through the use of a small mobile app, even without internet connection. Tripency also has a strong community aspect to the website whereby user generated content such as review and rating are encouraged, and are an important part of matching users to their desired attractions. Users will also have the ability to update attraction details so that the information on the website can remain current and up-to-date.

The backend will be written in Java, using the Spring framework, a PostGreSQL database, and the Hibernate library providing ORM support. The front end will comprise of HTML/CSS pages with an angularJS client server providing a sleek modern single page user experience. The mobile app will also be written in Java and initially deployed to the Android platform.

1 Introduction

1.1 Objectives

The objective of the project is to develop a web application called “Tripency”. The name “Tripency” is a combination of the word “trip” and “efficiency”. The primary functionality of Tripency will be to assist users with efficiently planning their holidays by dynamically matching their interests (and disinterests) with attractions, cities and other locations. The user will then be provided with a tailored list of attractions from which they can build their itinerary. The initial scope of the web application will be to primarily cater for city-break holidays instead of beach and sports holidays due to the much wider variety of attractions, museums and monuments typically found in towns and cities.

In addition to this tailored itinerary builder, the web application will also provide user profile management functionality typically found in modern websites. This includes the ability to save and retrieve itineraries. Also, similar to Amazon and other online retailers, the web application will automatically suggest other possible destinations based on those saved user itineraries.

There will also be a community aspect to the web application which will be leveraged by the system to determine how a user’s interests matches the various attractions. Users will have the ability to write text-based reviews on each attraction, but will also have the ability to rate those attractions on a numeric scale against that person’s interests. For example, a user might rate London’s Tate Gallery very high for “Art”, and this would be taken into account when another user specifies “Art” as an interest for building their itinerary. The web application will also adopt the “wiki” philosophy so that users can update attraction details (such as description, opening hours, prices) themselves without relying on Tripency administrators. To further increase the community aspect, the web application will utilize social media APIs such as Facebook and Twitter so users can post/tweet about their itinerary directly from Tripency.

Finally, given the nature of the industry Tripency operates in, it would be important to allow the users to view their finished itinerary offline so that it is available to them during the actual holiday especially if internet access is not readily available or if it incurs high roaming charges. This will be accomplished with a small app that will supplement the web application. Although 90% of the project will be focused on the web application itself, the Tripency app will be developed to provide users with that important offline functionality. One alternative would have been for Tripency to be entirely app based, but an observation of the travel industry determined a strong online presence is very important, e.g. Tripadvisor, Expedia, hotels.com, etc…

To summarize, the Tripency system will provide the following main functionality:

- Tailored itinerary builder
- User profile and preferences management
- Community management through user reviews/rating, user content and social media connectivity
- Offline mobile itinerary storage via a small supplementary app

1.2 Background

The core idea of dynamically generating a travel itinerary based on user preferences originated from a third year project. That project was developed using basic Java servlets (see Technical Approach below), and did not include any additional functionality outlined above such as user management, user content and social media connectivity. Upon presenting the finished project, the core idea was received well but the primary critical feedback was that the user required internet access to view his itinerary during the actual holiday.

The main alternative for this project is a training management system which would be used by Human Resource departments and training companies to track and monitor their training. The developer has considerable experience in this area and believes most solutions on the market for this industry do not fully cater for the complexities and realities of administrating the training function within a large human resources department.

However, Tripency was ultimately chosen for this project because it was always the developer's intention to expand that core idea to a full enterprise level application while taking into account the feedback received. The developer believed the third year project did not allow the core idea of dynamically generating tailored itineraries to sufficiently reach its potential. Also, the third year project utilized out-dated and relatively simplistic technology. Tripency also provided for obvious uses of social media unlike the training management application. Therefore, Tripency will be redeveloped from the ground up using modern technologies, a vastly redesigned user interface together with considerably more functionality.

1.3 Technologies

Tripency will be written in Java using the following technologies:

- The Spring Framework will provide RESTful services to handle requests from both the web application and the app. The client / server data communication will be done using the JSON data structure. Spring will also handle core system non-functional requirements such as exception handling and logging.
- Data persistence will be provided by a PostGreSQL database.
- Database connectivity will be handled by utilizing the Hibernate library to provide ORM support.
- The view technology will be angularjs. This will provide a modern and seamless user experience. The developer had considered using standard JSPs but decided for angularjs because it afforded him the opportunity to learn a new technology, using a newer technology in the project, and provide a “single page” UX.
- The unit tests will be done using the JUnit testing library.
- Tomcat will be used as the web server.
- The supplementary app will target the Android platform. Apple devices will not be supported in this project, although that remains an opportunity for future expansion. Developing an app for two completely different platforms using different languages will not be possible during the timeframe involved.

2 System

2.1 Requirements

2.1.1 Requirement Gathering Techniques

The initial scoping and requirements gathering process involved face-to-face interviews with both consumers of travel websites and members of the travel industry.

First Iteration

The first iteration of these requirements specified a fairly “locked down” website where all attraction details were administered and edited by a central team of administrators. Also, the system did not provide a mobile app that allows users to view their itineraries on locally stored mobile devices without internet connection.

Strong feedback was provided that clearly indicated an app that allowed users to view their details without internet connection was a big requirements gap due to the nature of the travel industry. Users also liked the idea of user editable content, so users with knowledge of a particular city could update details about the attractions in that city.

Second Iteration

The second iteration of these requirements included a more emphasis on user generated content. This included the ability to add reviews, authenticated users could edit attractions, and user’s themselves could provide weighting for an attraction against the various attraction types. The requirements has also included a small mobile app that allows users to locally store their itinerary information on their mobile device.

Upon demonstration this second iteration, users felt the system was more user focused. Members of the travel industry asked for the ability for unregistered users to try a small part of the system as a trail to entice users to register.

Third Iteration

The second iteration of these requirements facilitated the ability for unregistered users to create itineraries for a small selection of cities. Once the user registered and signed in, the system full selection of cities would become available.

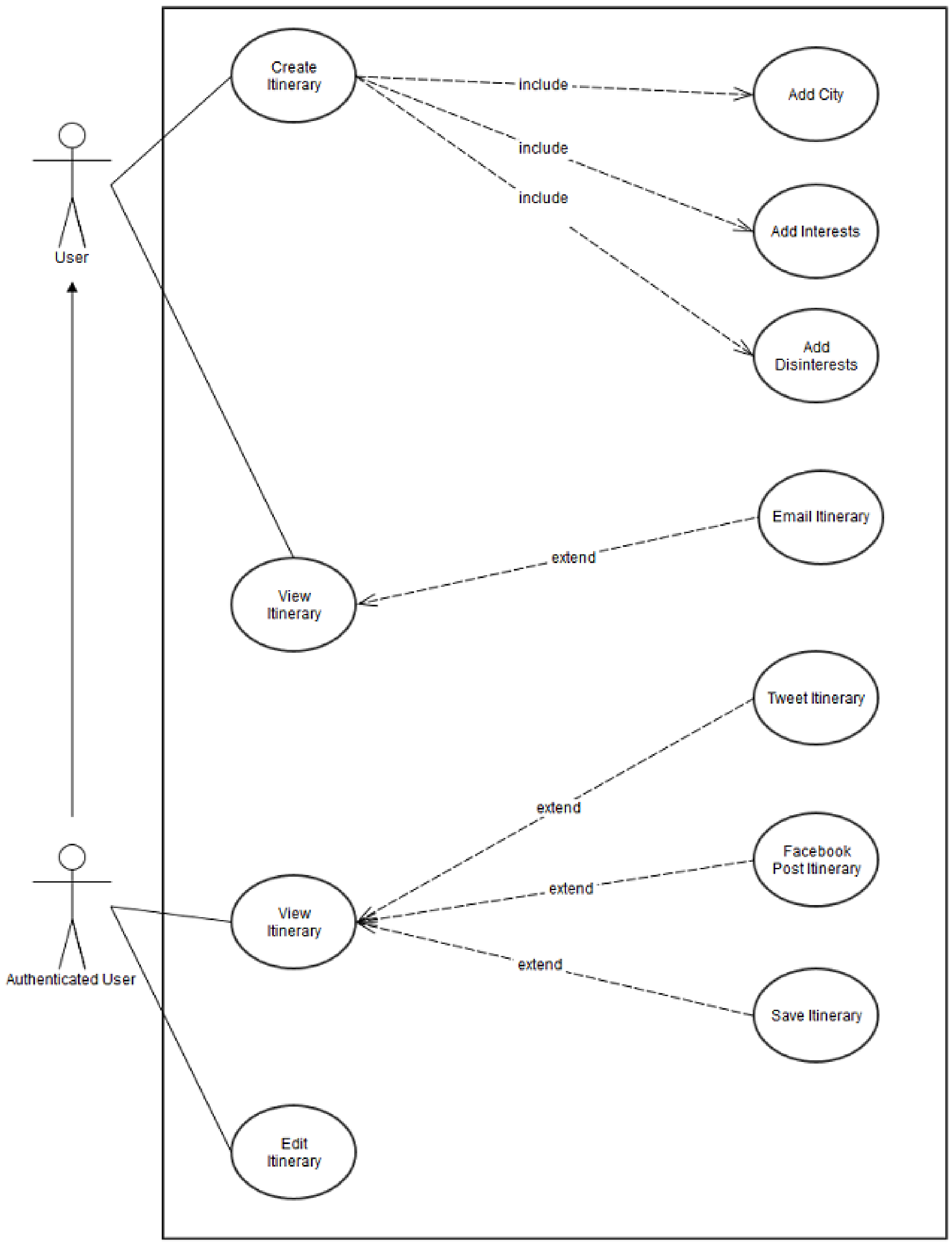
2.1.2 Glossary of Domain Terminology

This section summarises the various domain and system terminology that are referenced in the requirements:

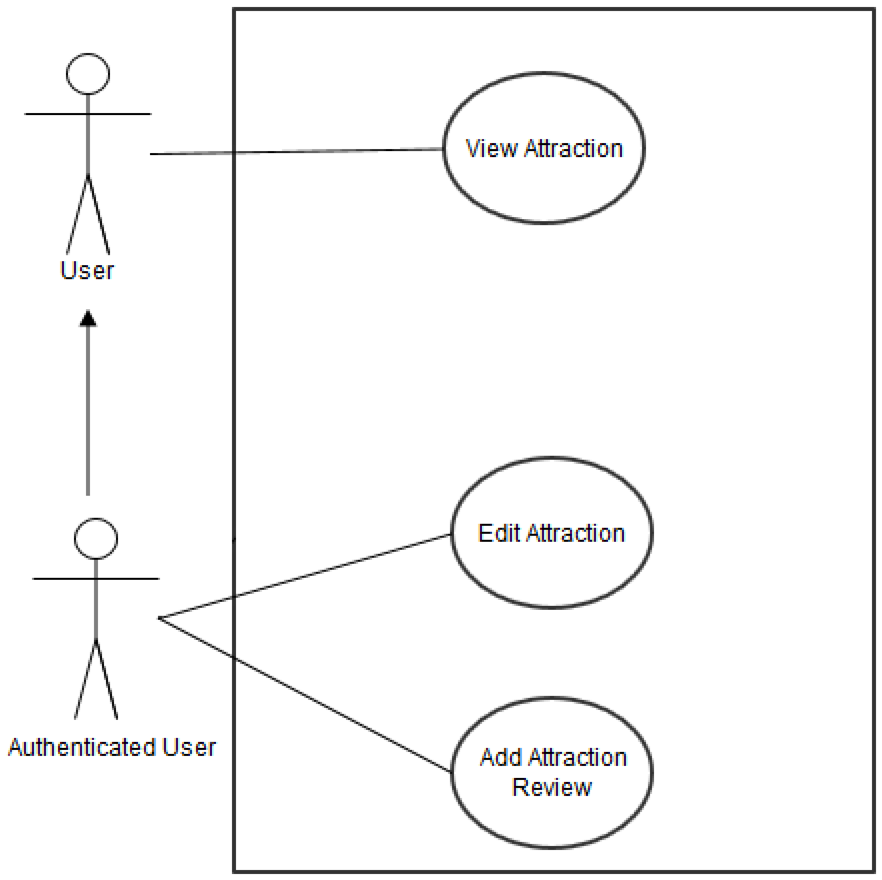
Term	Description
Destination / Location	A city a user might visit, e.g. London, Paris, etc…
Attraction	A place of interest in a city, e.g. a district (e.g. Notting Hill), a structure (e.g. The Eiffel Tower, London Eye), a building (e.g. a museum, a famous retailer), a geographic feature (e.g. Arthur's Seat in Edinburgh, Central Park)
Itinerary	A collection of attractions generated as per the user’s interests
Attraction Type	Categories of attractions, e.g. art, history, views, outdoor activities, must-sees, hidden gems
Interest Categories	The level of interest or disinterest a user can assign to attraction types, e.g. very disinterested, slightly interested, very interested.
Attraction Type Rating	The rating (out of 5) an attraction has against a particular attraction type. For example, The Tower of London would have a rating of 4.9 for the “history” attraction type

2.1.3 Use Case Diagrams

2.1.3.1 Itinerary Management



2.1.3.2 Attraction Management



2.1.4 Functional requirements - Itinerary Management

2.1.4.1 INTR-001 : Initialisation of Itinerary Creation

Requirement ID	INTR -001
Scope	The scope of this requirement is to allow the user to navigate from the Home Page to the page which begins the itinerary creation process.
Description	This requirement describes how the user begins the itinerary creation process
Precondition	The user must be currently viewing the home page. The user does not need to be logged in.
Activation	This requirement starts when the user is currently viewing the home page
Main Flow	1. The user clicks the "Create Itinerary" button on the navigation bar from the home page 2. The first page of the itinerary creation process is displayed to the user
Alternate Flow	1. As the navigation bar will be visible on all pages, the user may also begin the itinerary process by clicking the "Create Itinerary" button from any page 2. The first page of the itinerary creation process is displayed to the user
Exceptional Flow	None
User Interface Impacts	The facilitation of this requirement requires a button on the navigation bar with the display value of 'Create Itinerary'.
Termination	The use case terminates when the user has successfully navigated to the first page of the itinerary creation process
Post Condition	The user is currently viewing the first page of the itinerary creation process

2.1.4.2 INTR-002 : City Selection

Requirement ID	INTR-002

Scope	The scope of this requirement is to allow the user to select a destination city for their itinerary.
Description	This requirement describes how the user can specify a destination city for their itinerary. The selection of cities available to the user is dependent on the user's logged in status
Precondition	INTR-001 must have seen successfully completed and be in post condition
Activation	This use case starts when the user is currently viewing the first page of the itinerary creation process
Main Flow	If the user is logged in: 1. The system will display a full list of available cities to the user 2. The user selects their desired destination from this list of cities 3. The user clicks the 'Next' button 4. The system displays the second page of the itinerary creation process
Alternate Flow	If the user is NOT logged in: 1. The system will only display a partial list of cities which is available to unauthenticated/unregistered used 2. The user selects their desired destination from this list of cities 3. The user clicks the 'Next' button 4. The system displays the second page of the itinerary creation process
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • A drop down list, listing the cities available to the user grouped by country • A button with a display value of 'Generate Itinerary.
Termination	The use case terminates when the user has successfully navigated to the second page of the itinerary creation process
Post Condition	The user is currently viewing the second page of the itinerary creation process having selected their desired destination city. The system will persist this selected destination for the remainder of the itinerary creation process.

2.1.4.3 INTR-003 : Interested Attraction Types Selection

Requirement ID	INTR-003
Scope	The scope of this requirement is to allow the user to select the attraction types that are desirable and of interest to the user.
Description	This requirement describes how the user can specify the attraction types that the user is interested in.
Precondition	INTR-002 must have seen successfully completed and be in post condition
Activation	This use case starts when the user is currently viewing the second page of the itinerary creation process
Main Flow	1. The system will display a list of attraction types. 2. The system will display three interest categories to the user: (1) Slight Interest, (2) Medium Interest, (3) Significant Interest 3. For one or many attraction types, the user assigns the user's desired interest category. The user must do this for at least one attraction type. An attraction type may be linked to only one interest category. 4. The user clicks the 'Next' button 5. The system displays the third page of the itinerary creation process.
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • Three selectable elements, each representing one of the three interest category. • A button with a display value of 'Next'.
Termination	The use case terminates when the user has successfully navigated to the third page of the itinerary creation process
Post Condition	The user is currently viewing the third page of the itinerary creation process having selected their desired interest category per attraction type. The system will persist this selection for the remainder of the itinerary creation process.

2.1.4.4 INTR-004 : Disinterested Attraction Types Selection

Requirement ID	INTR-004

Scope	The scope of this requirement is to allow the user to select the types of attractions that are undesirable and of little or no interest to the user.
Description	This requirement describes how the user can specify the types of attractions that the user is not interested in.
Precondition	INTR-003 must have seen successfully completed and be in post condition
Activation	This use case starts when the user is currently viewing the third page of the itinerary creation process
Main Flow	The facilitation of this requirement requires: 1. One selectable element, representing disinterest. 2. A button with a display value of 'Next'.
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • One selectable element, representing disinterest. • A button with a display value of 'Next'.
Termination	The use case terminates when the user has successfully navigated to the fourth page of the itinerary creation process
Post Condition	The user is currently viewing the fourth page of the itinerary creation process having selected their undesired interest category per attraction type (where applicable). The system will persist this selection for the remainder of the itinerary creation process.

2.1.4.5 INTR-005 : View New Itinerary

Requirement ID	INTR-005
Scope	The scope of this requirement is to allow the user to view, modify and save the new system generated Itinerary, with attractions in the itinerary determined with respect to the user's interest and disinterest selections.
Description	This requirement describes how the system will display a list of attractions ("the itinerary"). The system will select such attractions by weighting attractions' attraction type rating against the interest/disinterest selection chosen by the user. This weighting will be

	displayed as a percentage. The system will display the attractions in descending order based on this percentage.
Precondition	INTR-004 must have seen successfully completed and be in post condition
Activation	This use case starts when the user is currently viewing the forth page of the itinerary creation process
Main Flow	1. The system will display a list of attractions matched against the user's specified interests/disinterests. 2. The system will provide the option to the user of removing individual attractions from their itinerary on an ad hoc basis. 3. If the user is logged in, the user clicks the 'Save' button. 4. The system will persist the itinerary against the user's account.
Alternate Flow	1. The system will display a list of attractions matched against the user's specified interests/disinterests. 2. The system will provide the option to the user of removing individual attractions from their itinerary on an ad hoc basis. 3. If the user is not logged in, the system will display: – a message prompting the user to log in to allow the itinerary to be saved. – A button to allow the itinerary to be emailed to an email address See requirement EMAIL-001 below for further details of the email functionality for unauthenticated users
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • A collection of elements, each representing an attraction • A button with a display value of 'Save'. • A button with a display value of 'Email'. • A text box for the email address • A button with a display value of 'Log In' (where applicable).
Termination	The use case terminates when the user is viewing the new itinerary
Post Condition	The user is currently viewing the "itinerary" page

2.1.4.6 INTR-006 : View Existing Itinerary

Requirement ID	INTR-006

Scope	The scope of this requirement is to allow the user to view, modify and save a previously saved itinerary which was saved against the user's account.
Description	This requirement describes how the system will allow the user to view, modify and save a previously saved itinerary which was saved against the user's account.
Precondition	The user must be authenticated and logged in
Activation	This use case starts when the user is authenticated and logged in
Main Flow	1. The user clicks the 'Itineraries' link on the navigation bar 2. The system will display a list of previously saved Itineraries for that user 3. The user clicks on an itinerary 4. The system will display that itinerary and its attractions 5. The system will provide the option to the user of removing individual attractions from their itinerary on an ad hoc basis; and any such changes are persisted by the user through the use of a 'Save Changes' button
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • A collection of elements, each representing an attraction • A button with a display value of 'Save'.
Termination	The use case terminates when the user is currently viewing the itinerary page
Post Condition	The user is currently viewing the itinerary page

2.1.5 Functional requirements – *Attraction Management*

2.1.5.1 ATTR-001 : User Reviews

Requirement ID	ATTR-001
Scope	The scope of this requirement is to allow the user to submit a review for an attraction

Description	This requirement describes how the system will facilitate the submission of an attraction review from the user
Precondition	The user must be successfully logged in.
Activation	This use case starts when the user is currently logged in
Main Flow	1. The user navigates to an attraction via one of their itineraries or the attraction location. 2. The user clicks the 'Add Review' button 3. The system displays the 'Add Review' page 4. The user adds the following information: • The review title • The review body • A rating scale for zero or many attraction types for the attraction being reviewed. 5. The user clicks the 'Submit Review' button 6. The system persists the user review and redirects back to the attraction's page.
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • A single-line text box for the review's title • A multi-line text box for the review's body • A rating sliding scale for each attraction type
Termination	The use case terminates when the user has saved the review
Post Condition	The user is currently viewing attraction page containing the newly submitted review.

2.1.5.2 ATTR-002 : Edit Attraction Details

Requirement ID	ATTR-002
Scope	The scope of this requirement is to allow the user to edit attractions attraction details.
Description	This requirement describes how the system will facilitate the submission of user generated content for attractions and attraction details.
Precondition	The user must be successfully logged in.

Activation	This use case starts when the user is currently logged in
Main Flow	1. The user navigates to an attraction via one of their itineraries or the attraction location. 2. The user clicks the 'Edit Attraction' button 3. The system displays the 'Edit Attraction' page displaying all the fields for the selected attraction in editable UI elements 4. The user alters the details. 5. The user clicks the 'Save Changes' button 6. The system persists the changes and redirects back to the attraction's page.
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • A button on the attraction page with the display value of "Edit Attraction" • A collection of editable elements that allows the user to alter the value of the attraction's details. • A button on the "Edit Attraction" page with the display value of "Save Changes"
Termination	The use case terminates when the user has saved the attraction
Post Condition	The user is currently viewing attraction page containing the newly submitted details.

2.1.6 Functional requirements – *Social Media Connectivity*

2.1.6.1 SOCM-001 : Facebook Connectivity

Requirement ID	SOCM-001
Scope	The scope of this requirement is to allow the user to share a link to their itinerary by posting to their Facebook account through the system.
Description	This requirement describes how the system will facilitate the posting to the user's Facebook account of a link to the user's itinerary
Precondition	The user must be successfully logged in and viewing one of their itineraries.

Activation	This use case starts when the user is currently viewing an itinerary.
Main Flow	1. The user clicks the Facebook branded 'Share' button 2. The system will ask the user to sign into their Facebook account (where applicable). 3. The system will generate a default text for the Facebook post which can be edited by the user. This will include a URL to the itinerary which is viewable by all users with that URL. A user does not need to be logged in to view that URL. 4. The user clicks the 'Post to Facebook' button 5. The system will utilise the Facebook API to post a link to the user's itinerary on the user's Facebook account
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • The Facebook branded 'Share' button • A multi-line text box for the Facebook post text • A 'Post To Facebook' button to actually send the post
Termination	The use case terminates when the user has successfully posted the link on their Facebook account.
Post Condition	The user is currently viewing their itinerary page having successfully posted the link on their Facebook account.

2.1.6.2 SOCM-002 : Twitter Connectivity

Requirement ID	SOCM-002
Scope	The scope of this requirement is to allow the user to Tweet a link to their itinerary by Tweeting from their Twitter account through the system.
Description	This requirement describes how the system will facilitate the tweeting from the user's Twitter account of a link to the user's itinerary
Precondition	The user must be successfully logged in and viewing one of their itineraries.
Activation	This use case starts when the user is currently viewing an itinerary.
Main Flow	1. The user clicks the Twitter branded 'Tweet' button

	2. The system will ask the user to sign into their Twitter account (where applicable). 3. The system will generate a default text for the tweet which can be edited by the user. This will include a URL to the itinerary which is viewable by all users with that URL. A user does not need to be logged in to view that URL. 4. The user clicks the 'Tweet' button 5. The system will utilise the Twitter API to tweet a link to the user's itinerary on the user's Twitter account
Alternate Flow	None
Exceptional Flow	None
User Interface Impact	The facilitation of this requirement requires: • The Facebook branded 'Tweet' bottom • A multi-line text box for the tweet text • A 'Tweet' button to actually send the tweet
Termination	The use case terminates when the user has successfully tweeted the link using their Twitter account.
Post Condition	The user is currently viewing their itinerary page having successfully posted the link on their Twitter account.

2.1.7 Functional requirements – *Email*

2.1.7.1 EMAIL-001 : Itinerary Email

Requirement ID	EMAIL-001
Scope	The scope of this requirement is to allow the user to send a copy of their generated itinerary to their email account.
Description	This requirement describes how the system will facilitate the sending of the user's itinerary to their email account
Precondition	The user must viewing one of their itineraries.
Activation	This use case starts when the user is currently viewing an itinerary.
Main Flow	If the user is registered and logged in: 1. The user clicks the 'Email' button

	2. The system will send a copy of the itinerary to the user's email account 3. The system will notify the user that the email has been sent successfully.
Alternate Flow	If the user is not registered or logged in: 1. The user clicks the 'Email' button 2. The system will display a text box that allows the user to enter an email address. 3. The system will send a copy of the itinerary to the email address specified above. 4. The system will notify the user that the email has been sent successfully.
Exceptional Flow	If the email sending operating fails, the user will be notified of the reason.
User Interface Impact	The facilitation of this requirement requires: • The button with the display value of 'Email Itinerary' • A text box for an unauthenticated user to enter their email address. • An element that displays the email success or failure message to the user
Termination	The use case terminates when the user has clicked the email button and received either a success or failure notification
Post Condition	The user is currently viewing their itinerary page

2.1.8 User requirements

2.1.8.1 USER-001 : User Registration

Requirement ID	USER-001
Scope	The scope of this requirement is to allow a user register an account.
Description	This requirement describes how the system will facilitate registration of a user account
Precondition	The user must be unregistered.
Activation	This use case starts when the user is currently viewing the website

Main Flow	If the user is registered and logged in: 1. The user clicks ‘Register Account’ button 2. The system navigates to the register account page 3. The user enters their firstname, surname, username, password and email address 4. The user clicks the ‘Create Account’ button 5. The system creates the new user account, authenticates the user, and redirects to the home page.
Alternate Flow	None
Exceptional Flow	• If the chosen username or email address is not unique, the system will prevent registration until this is rectified. • The system will prevent registration if the specified password does not meet the following security guidelines: – Must contain at least 8 characters – Must contain a combination of upper and lower case characters – Must contain at least one numeric character – Must contain at least one special character – Must not contain the username string
User Interface Impact	The facilitation of this requirement requires: • A button with the display value of ‘Register Account’ • A user registration form with text boxes for firstname, surname, username, password and email address. • A button with the display value of ‘Create Account’
Termination	The use case terminates when the user has successfully created a new account and is logged in
Post Condition	The user’s account is authenticated and logged in.

2.1.8.2 USER-002 : User Log In

Requirement ID	USER-002
Scope	The scope of this requirement is to allow a user to log in
Description	This requirement describes how the system will authenticate a user’s account through a log in functionality
Precondition	The user must be registered.

Activation	This use case starts when an unauthenticated user is currently viewing the website
Main Flow	If the user is registered and logged in: 1. The user clicks ‘Log In’ button 2. The system displays the log in page 3. The user enters their username and password 4. The user clicks the “OK” button 5. The system will authenticate the user’s details
Alternate Flow	None
Exceptional Flow	• If the username and password combination does not match that of any registered account, the system will prevent authentication
User Interface Impact	The facilitation of this requirement requires: • A button with the display value of ‘Log In’ • A log in form with text boxes for username and password • A button with the display value of ‘OK’
Termination	The use case terminates when the user has successfully created a new account and is logged in
Post Condition	The user’s account is authenticated and logged in.

2.1.9 Mobile App requirements

2.1.9.1

The app must be supported by the Android platform. The Windows and Apple platforms are not in scope.

2.1.9.2

The app must provide log in functionality to authenticate the user’s account

2.1.9.3

The app must display a list of itineraries linked to user’s account. This information must be storable locally so it is retrievable with no internet access.

2.1.9.4

The app must display a list of attractions liked to a particular itinerary. This information must be storable locally so it is retrievable with no internet access.

2.1.9.5

The app must facilitate the user writing a review of an attraction which includes a rating system for that attraction against attraction types. This review must be persisted to the system’s online database through the use of a ‘Save Review’ button. Where no internet access is available, this review information must be stored locally until it can be synced with the system’s online database.

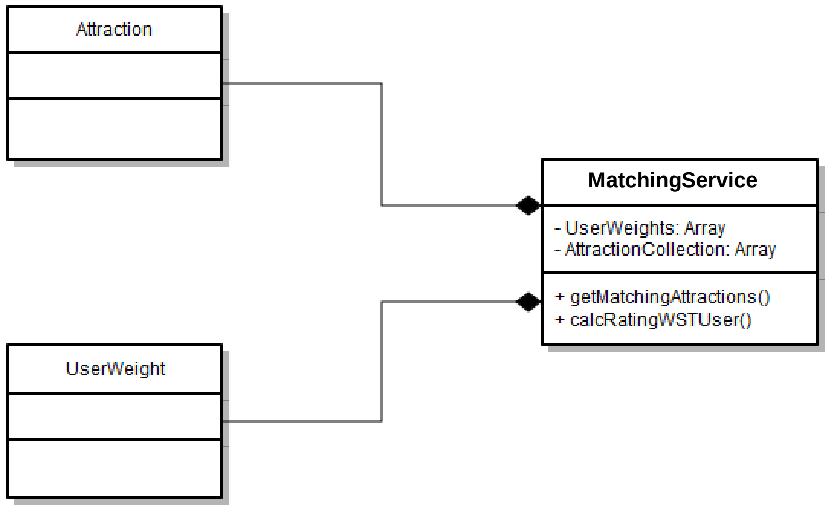
2.2 Design and Architecture

2.2.1 Overview

The server architecture of the system will be a Java Spring web application. This will be divided into three distinct logical layers. The REST controller will provide JSON data to the client. The main benefit with the server returning JSON instead of HTML/JSP pages is that the same REST controller can be also called from the Android app. The data access layer will use the Hibernate library to provide ORM support for database access and manipulation. Between these two layers will sit the business logic layer, which contains the main logic of the system. The client will be written in angularJS in order to provide a single page user experience, and this will receive JSON data from the server. This angularJS client server will also manage the different views (pages) presented to the user.

The main challenge with developing this application is the method used to match attractions with user’s interests and disinterests. After some research, the solution was to use the Euclidean distance matrix algorithm. This is an algorithm that represents the spacing of a set of points in a 3D space. In this domain, these points represent (1) the user’s interests and disinterests rating against a particular attraction, and (2) that attraction’s overall Attraction Type Rating. The spacing between the two points as returned by this algorithm determines the attraction’s suitability for the user.

This is implemented in Java with a MatchingService class. This contains a method called getMatchingAttractions that takes as parameters an array of Attractions, and an array of the user’s interests/disinterest weights. These are passed to a private method called calcRatingWRTUser. This method iterates through each combination of attraction and interest/disinterests weight, and passes them through the Euclidean distance matrix algorithm. The getMatchingAttractions method receives the returns of this calculation then returns a collection of Attractions sorted in descending order of the Euclidean distance matrix value.



2.2.2 Client Side (Front End)

The client side will utilise the angularjs library. Angularjs is a structural Javascript framework created and maintained mostly by Google to allow web developers to accomplish two primary goals: (1) easily create single page applications; and (2) generate powerful dynamic client side applications using data binding and the model view controller pattern (MVC). Historically, a web application's view (i.e. HTML pages) were generated entirely by the server which returned to the client a fully created web page. All the data binding, including some DOM manipulation such as hiding certain elements, was done on the template by the server before sending the response back to the client. A good example of this is the relationship between Spring MVC and Java Server Pages (JSPs). In this case, a Java object representing domain information was passed to the JSP, which in turn replaced placeholders in the JSP with the data from the Java object. This was accomplished using the JSTL tag library. The server then returned the entire generated page to the client which was then rendered by the browser. Spring MVC worked very well with JSPs but there are a number of drawbacks to this approach:

- The server almost always returned HTML which may be suitable to web browsers but also meant the same server could not be used for other media which as mobile phone apps even if both the app and the website used the same data

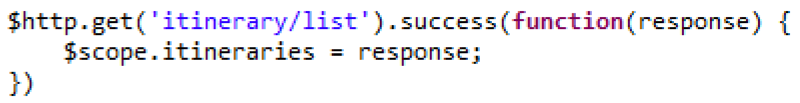
- The server returned a full web page which was fully rendered by the browser after each request. Therefore, the experiences a full page refresh (and a consequent “flicker” of the screen) even if only a small section of the page actually required updated data from the server. A web application’s navigation bar is a good example of this. After the initial loading of the web application, the navigation bar does not need to be re-rendered every time the user moves to a new area of the site

To overcome these shortcomings, angularjs was chosen as the client side technology for Tripency.

There are a number of core components of angularjs with are utilized by Tripenecy:

Scope

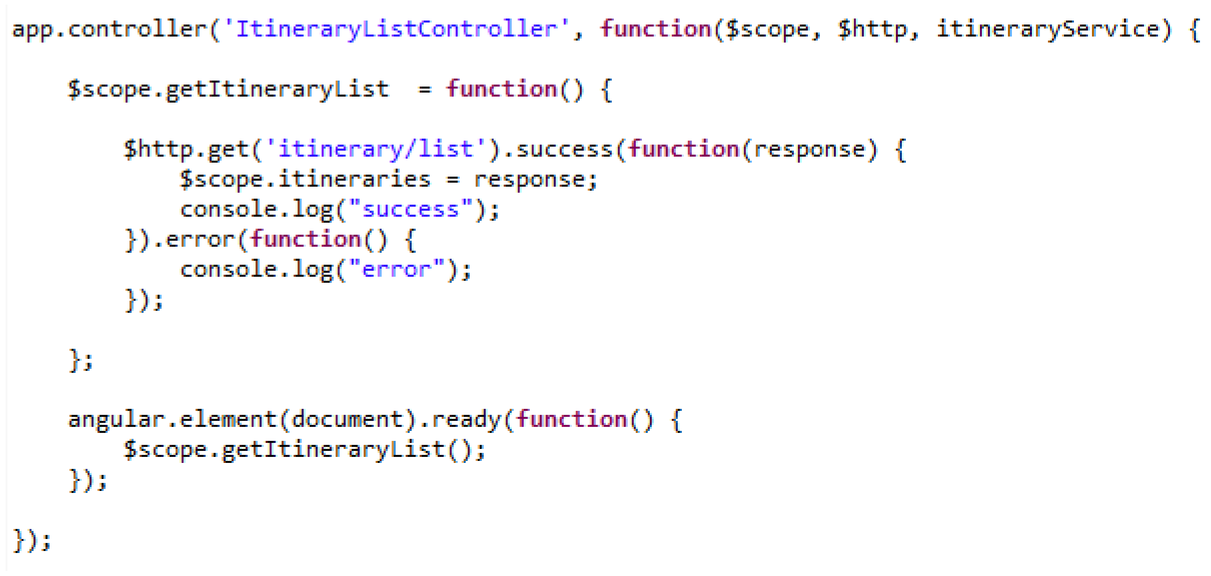
In the world of angularjs, scope is data, also known as the model of the MVC pattern. Scope is represented as javascript variables, objects or arrays. This data is retrievable from the server by means of an AJAX rest call. The response of this call is returned as a JSON string of data which can then be set to the value of a scope variable. For example, the following code snippet calls the ‘itinerary/list’ rest endpoint using the HTTP GET command, and its response is set to a scope variable called ‘itineraries’ that represents a list of itineraries.



```
$http.get('itinerary/list').success(function(response) {
    $scope.itineraries = response;
})
```

Controllers

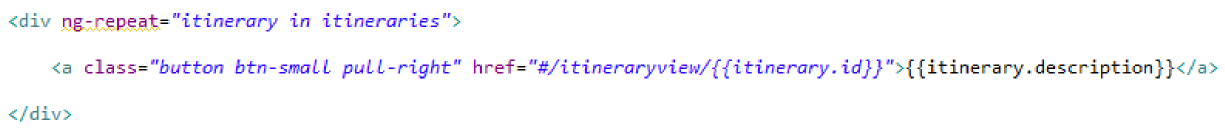
An angularjs controller represents the controller of the MVC pattern. The controller is responsible for setting the initial value of the scope and adding behaviour to the scope such as sending or receiving data to or from the server. Controllers are organised by logical domain functionality and can have various functions tasked with performing such functionality. The code snippet above is part of the function called ‘getItineraryList’ in the controller called ‘ItineraryListController’ as shown below:



```
app.controller('ItineraryListController', function($scope, $http, itineraryService) {

    $scope.getItineraryList  = function() {

        $http.get('itinerary/list').success(function(response) {
            $scope.itineraries = response;
            console.log("success");
        }).error(function() {
            console.log("error");
        });

    };

    angular.element(document).ready(function() {
        $scope.getItineraryList();
    });

});
```

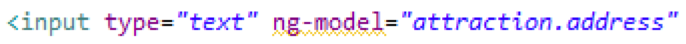
Views

A view in an angularjs web application is a HTML page embedded with angularjs directives and expressions. This markup is mostly used to respond to an event (such as the angularjs directive ‘ng-click’ for a click event), or to display data from the scope on the page. In the code snipped below, the ‘’ng-repeat’ directive is used to iterate over the list of itineraries returned by the controller function above. Individual items of scope data can be rendered by including them within two sets of curly braces, such as ‘{{itinerary.description}}’ below which obviously displays the description text of the itinerary.



```
<div ng-repeat="itinerary in itineraries">

    <a class="button btn-small pull-right" href="#/itineraryview/{{itinerary.id}}">{{itinerary.description}}</a>

</div>
```

Also, an alternative method of data binding is the use of the ng-model directive which is often used to bind HTML form elements to the scope, such as the attraction’s address in the textbox in the code snipped below:

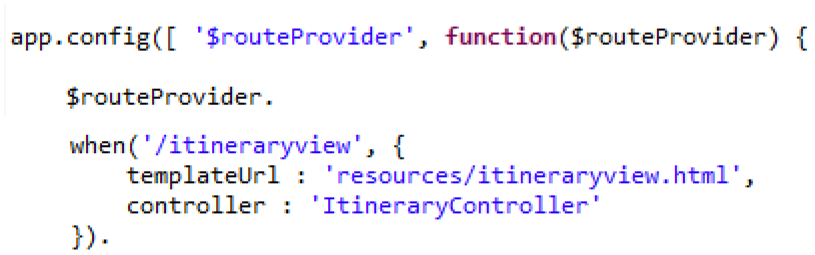


```
<input type="text" ng-model="attraction.address"
```

A major benefit of data binding in angularjs is that any changes made to the page’s DOM (such as a user entering text in an input box) is automatically and immediately reflected in the underlying scope. This means the developer does not need to write code to manually read the text box’s value and assign it to a Javascript variable. Angularjs takes care of this ‘plumbing’ code inherently.

__RouteProvider__

The routeProvider is a very important component of angularjs that links a URL to a particular view template and controller. This is achieved by routing a controller and template to a specific URL request. For example in the code snippet below, when the user navigates to the ‘/itineraryview’ URL, the routeProvider retrieves the ‘itineraryview.html’ file from the server and instantiates the ‘ItineraryController’ controller. That controller then initiziates the scope and data-binds it to the html page.



```
app.config([ '$routeProvider', function($routeProvider) {

    $routeProvider.

    when('/itineraryview', {
        templateUrl : 'resources/itineraryview.html',
        controller : 'ItineraryController'
    }).
```

,

__Services__

Finally, it is important to note that controllers are instantiated newly upon each request. Therefore, if scope data is to persist between different controllers, an angularjs service must be used which can then be injected into the controller. For example, the code snippet below is an angularjs service for attractions with a getter and setter function for an attraction.

```
app.service('attractionService', function() {


    var attraction = {};


    this.setAttraction = function(a) {

        attraction = {};
        attraction = a.slice();


    };

    this.getAttraction= function() {

        return attraction;

    };
```

This service is then injected to the attraction controller in the controller's function definition, as shown below.

```
app.controller('AttractionController', function($scope, $http, $routeParams, $route, attractionService) {
```

Please note that the attractionService object does not need to be initialised by the developer before it is injected into the controller. Angularjs automatically recognised that the attractionService is required by the controller and automatically creates the object. This is known as Dependency Injection which is another benefit of using angularjs.

Disadvantages

As outlined above, there are many advantages to using angularjs and it was indeed the correct choice for Tripency, Although the benefits outweigh the negatives, there are a few disadvantages to using angularjs in a web application. These are:

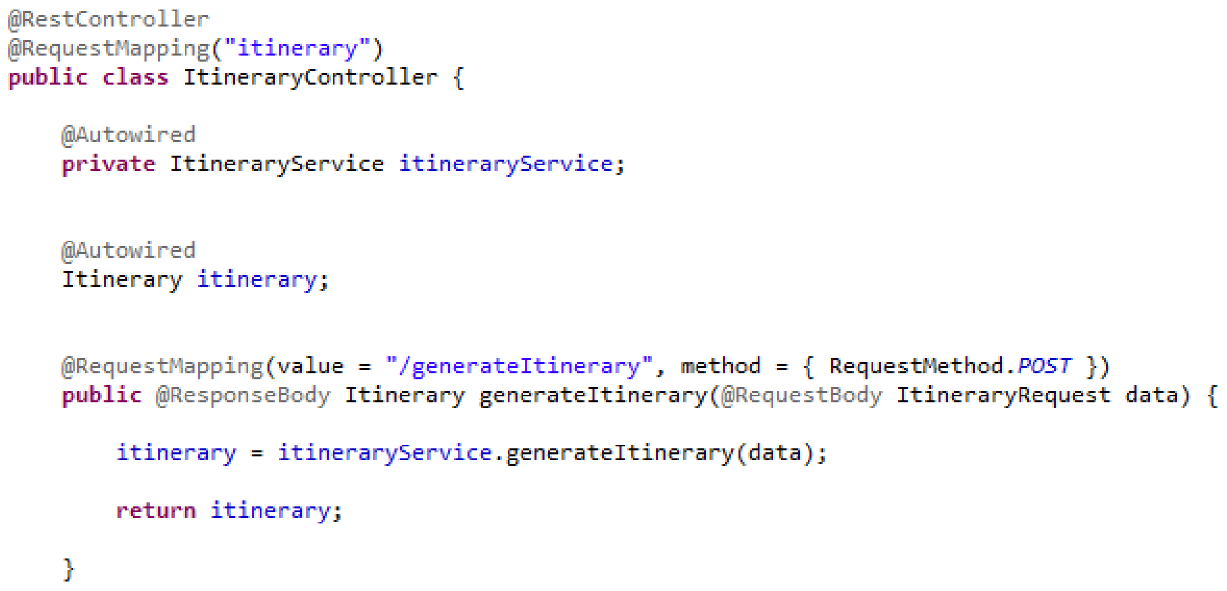
- Not Secure: Angularjs is entirely a Javascript framework and therefore runs on the client side, therefore the application is not secure. Angularjs must work in conjunction with server side authentication and authorisation in order to secure the application
- Degree of difficulty: Developing web applications using angularjs is more difficult and complex than the more traditional alternatives such as JSPs

2.2.3 Server Side

Tripency's server, which ultimately is responsible for user management and returning JSON domain data to the client, is a rest server written in Spring. Hibernate is a common complimentary ORM library to Spring, and that is also used by Tripency to provide database object modelling.

Rest Controller Layer:

The entry point to the server for the client is provided by means of rest API endpoints. One of the primary benefits of using the Spring framework is that the developer can easily add rest functionality to any method of any class through the use of annotation based programming. This is illustrated nicely in the code snipped below:



```
@RestController
@RequestMapping("itinerary")
public class ItineraryController {

    @Autowired
    private ItineraryService itineraryService;


    @Autowired
    Itinerary itinerary;


    @RequestMapping(value = "/generateItinerary", method = { RequestMethod.POST })
    public @ResponseBody Itinerary generateItinerary(@RequestBody ItineraryRequest data) {

        itinerary = itineraryService.generateItinerary(data);

        return itinerary;

    }
```

This is a typical Java class called 'ItineraryController' with a method called 'generateItinerary'. That method takes an ItineraryRequest object as a parameter and returns an Itinerary object. This is basic Java behaviour which, of course, does not provide any inherent HTTP functionality. However, we can use Spring's annotations to quickly and easily convert this basic Java class to a rest controller. The annotations used in the snippet above are described as follows:

Class Level Annotations

Annotation	**Usage**
@RestController	A high-level annotation that specifies that the class is to be used as a rest controller
@RequestMapping(“itinerary”)	This specifies the first part of the rest API endpoint. All endpoints in this class must be prefixed with this mapping, and it is used to map endpoints to logical domain groups. For example, this is the ‘itinerary’ part of the ‘itinerary/generateItinerary’ endpoint. It is usually a noun representing a domain object.
@Autowired	As with angularjs, Spring also provides dependency injection support. This is accomplished using the @Autowired annotation. As can be seen from the code snippet above, the classes members are not instantiated by traditional Java means, but rather the Spring framework creates the appropriate autowired objects when the parent object is created, in this case the ItineraryController object.

Method Level Annotations

Annotation	**Usage**
@RequestMapping(value = "/generateItinerary", method = { RequestMethod.POST })	This specifies the second part of the rest API endpoint that targets a specific method. For example, this is the ‘generateItinerary’ part of the ‘itinerary/generateItinerary’ endpoint. It is usually a verb representing an action that can be performed on a domain object. The HTTP action type can also be specified, e.g. POST, GET, PUT.
@ResponseBody	This indicates the type of object to be returned to the client via HTTP
@RequestBody	This indicates the type of object passed to the server by the client

A major benefit of the @ResponseBody and the @RequestBody annotations is that Spring will automatically map and convert a JSON string received from the client to the specified Java domain object, and likewise, will convert the object to JSON when sending the response to the client. Spring does this by using the Jackson JSON library. The developer simply has to ensure that the JSON’s structure matches the Java object’s structure. This is highly advantageous because the developer need not concern himself with manually converting JSON, and this was one of the primary reasons Spring was chosen as the server side technology.

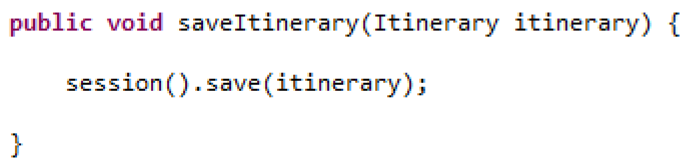
Service Layer

In order to main good separation of concerns, it is good (if not mandatory) programming practice not to have the controller interacting directly with the business logic. Therefore, the application exposes a number of services to the controllers which contain the business logic. This is why the service layer is often also referred to as the business logic layer. This layer contains a number of service objects, each handling a distinct domain object, for example the attractionService handling the business logic for the Attraction object. The objects in the service layer are standard Java classes exposed to the controllers through an interface. An interface is basically a contract that a class agrees to by implementing all the methods in the interface. It is good programming practice to use interfaces between the controller and service layer because it promotes easily extensibility if required during future development.

Data Access Objects Layer

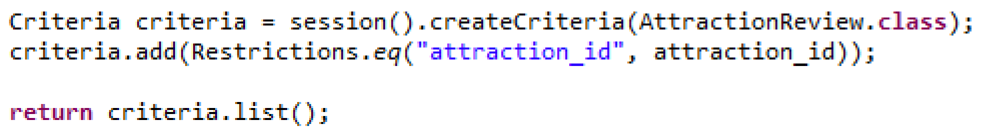
As it is good programming practice for the controller not to directly interact with the business logic, it is also good practice for the service layer not to connect directly to the database. Therefore, the DAO layer is used to provide database manipulation and query functionality. As with the service layer, the objects in the DAO layer handle database connectivity for specific domain objects, for example the itineraryDao is used to write and read data related to the Itinerary object. Each of Tripency's domain DAO object extends a primary ApplicationDao class through the use of inheritance. Tripency has been developed in this manner so that the individual DAO objects do not have to concern themselves with the mechanics of how to connect to the database (such as database hostname, password, username, port) and what Spring objects it must use as part of the Spring framework. Instead, all this configuration is handling by the ApplicationDao and the domain DAO object need only concern themselves with domain functionality. This is also done to promote code reuse and mitigate code duplication. Any code that would have been duplicated in the individual domain DAO objects instead resides in the ApplicationDao.

Java connects to databases using the Java Database Connectivity Library (JDBC). Using standard JDBC, the developer must write SQL and pass that SQL as a string to a JDBC object. The disadvantage of this is that any changes to the domain object (such as adding a new member) also requires an update to the SQL. To mitigate this, Tripency uses the Hibernate library for which there is good interoperability with Spring. Hibernate is an Object Relational Mapping (ORM) library which automatically generates SQL based on domain objects. This means that the developer can simply pass an object to the DAO layer, call a hibernate method, and the Hibernate library will automatically generate the SQL and execute it against the database. For example, in the code snippet below, we see a method used to save an itinerary object to the itinerary table in the database. Instead of writing SQL, we just pass the itinerary into the save method of Hibernate session object, and the new itinerary is committed to the database. To accomplish this rather seamless functionality, the members of the class must match the column names in the table, but this is a small price to pay for the efficiently of avoiding manually writing dynamic SQL.



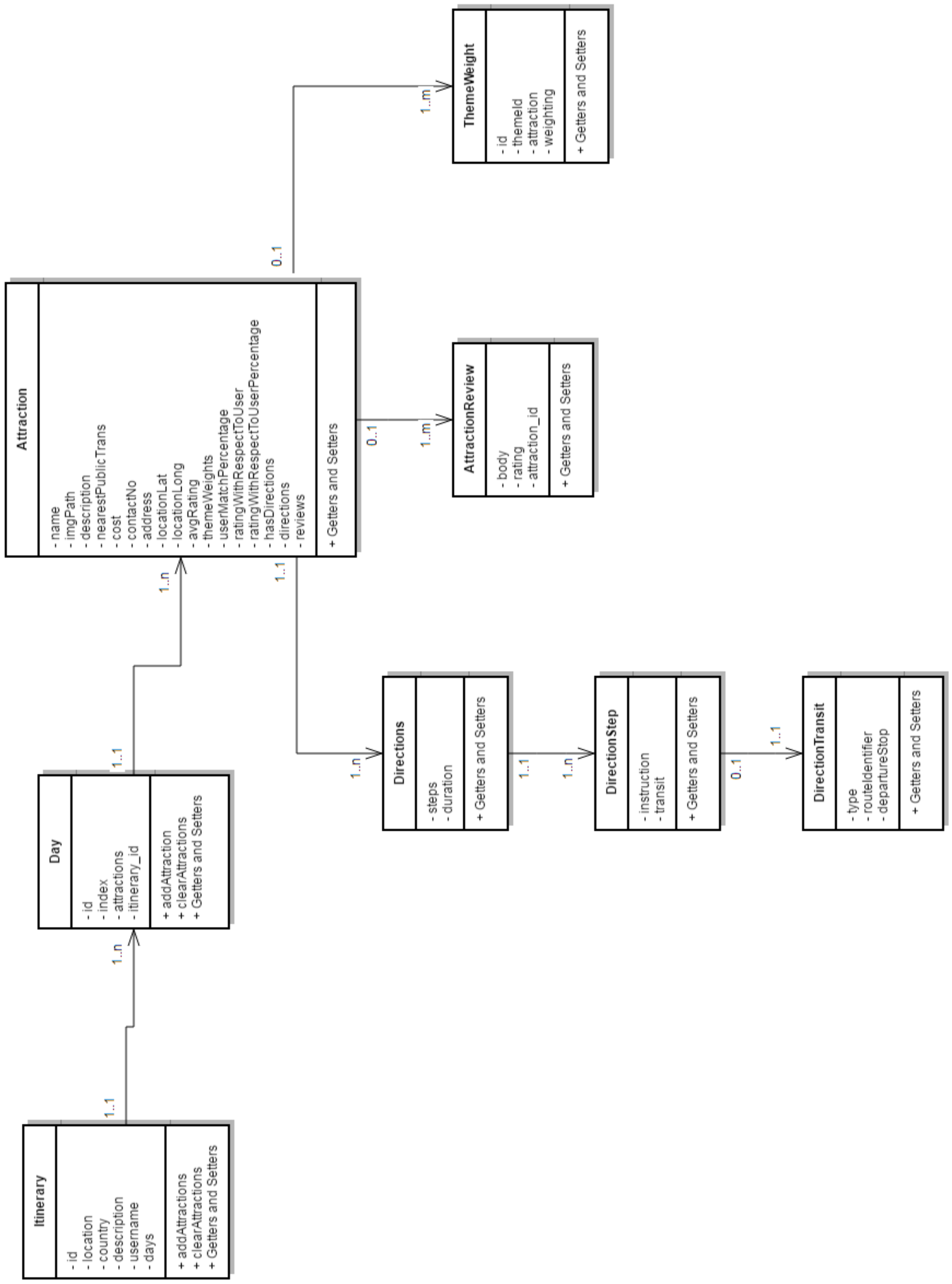
```
public void saveItinerary(Itinerary itinerary) {

    session().save(itinerary);

}
```

Similarly, we can use Hibernate to query a database. In the code snippet below, Tripency uses Hibernate to query all AttractionReview objects where the attraction_id is equal to a variable. This code returns a list of AttractionReview objects. The same functionality could be 10 times the number of lines of code if the developer had to manually create the SQL and then map each row in the result set to a new AttractionReview object.



```
Criteria criteria = session().createCriteria(AttractionReview.class);
criteria.add(Restrictions.eq("attraction_id", attraction_id));

return criteria.list();
```

__Domain Objects Design__



The preceding diagram is a class diagram of the domain objects used by Tripency. Also, as the Hibernate IRM model is used to link domain objects with the database, the database schema also follows this design. The following is a brief explanation of each class, its responsibility, and its relationship to other classes

Itinerary

This is the primary class of Tripency which ultimately contains all data about the itinerary in its members. The Itinerary class contains a list of Day objects under a 1 to MANY relationship. The addAttractions method includes an algorithm to determine what attraction should be added what day based on the recommended duration of that attraction, the time it takes to go to that attraction, and the current time allocated to that day. If a new attraction is deemed to take too long to fit into an existing day, a new day is created and added to that instead.

Day

The Day class holds a list of attractions to be visited in a single day. It also includes an addAttraction method should attraction management need to take place on a Day level

Attraction

Attraction class contains all information about a particular attraction. In addition to members for typical attraction details (cost, address, name, description, etc), it also have members for data used by the Euclidean distance matrix algorithm in determining an attraction's relevancy to a user's preferences. These are: userMatchPercentage, ratingWithRespectToUser, ratingWithRespectToUserPercentage. The Attraction class contains lists of three other classes: AttractionReview, Directions, ThemeWeight.

AttractionReview

The AttractionReview class contains details about a single review of an Attraction. The Attraction class can have zero or many AttractionReview objects.

ThemeWeight

The ThemeWeight classes contains an theme id and its corresponding weighting for that attractions, e.g. 1 and 4.5, where 1 is the id for the 'History' theme and 4.5 is the weight out of 5 for the attraction for that theme. The Attraction object contains list of these ThemeWeights and is used for the MatchService class to determine what attractions best suit the user’s preferences.

Directions

The Directions class contains all a list the individual steps for the route to the attraction. It also contains the duration it takes to get to the Attraction. The Attraction class can have 1 or many Direction objects (there can be different routes to an attraction).

DirectionStep

The DirectionStep class contains a single string indicating a particular step within a Direction, for example “Turn left at Main St”. The Directions class can have 1 or many DirectionsStep objects.

DirectionTransit

The DirectionTransit class contains details about public transport relevant to a DirectionStep. It contains the following members:

- type: Bus, Subway, Tram
- routeIdentifier: Bus number (e.g. 16) or subway line (e.g. Victoria)
- departureStop Bus stop number (e.g. 107) or subway station (e.g. Picadilly)

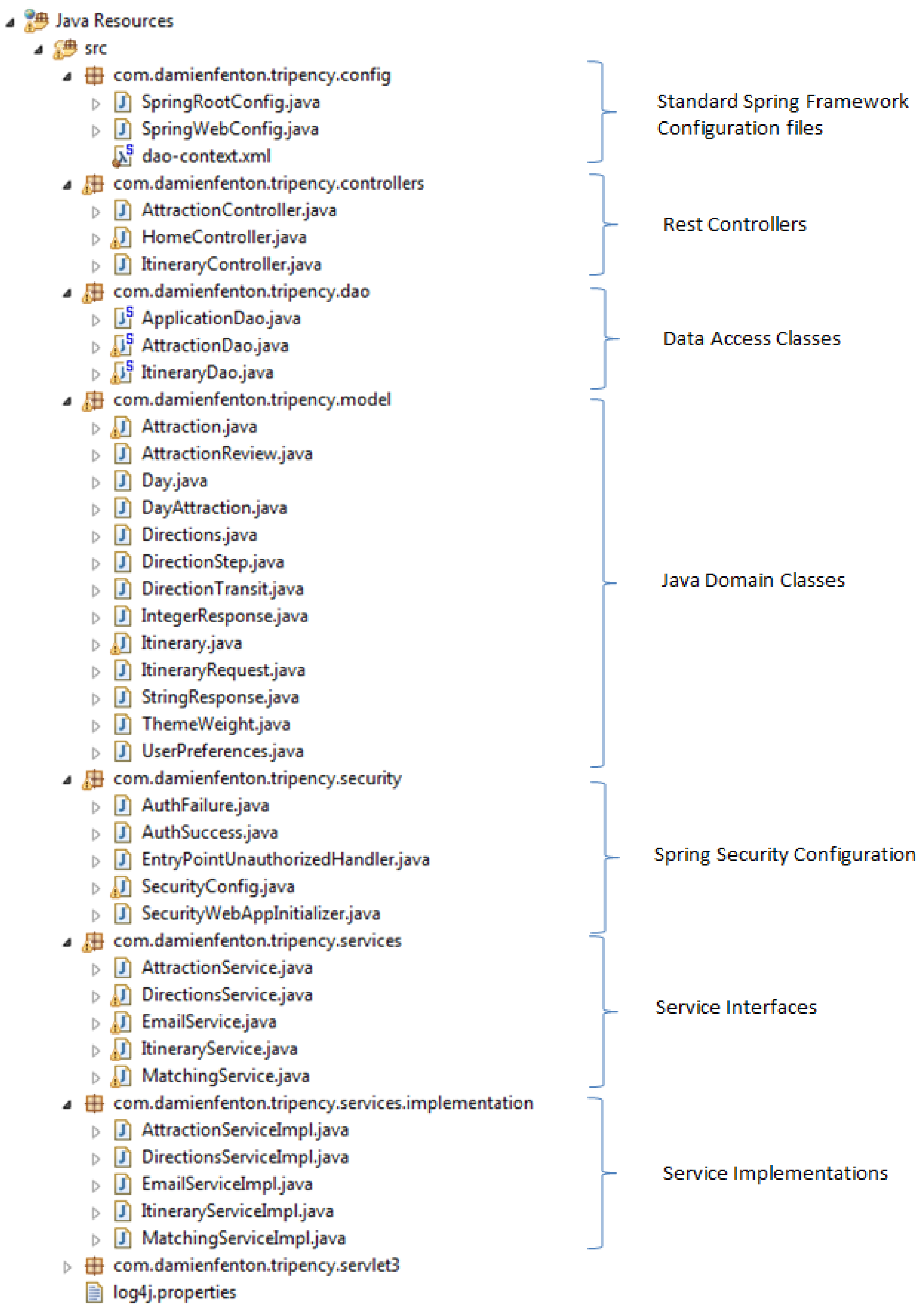
<u>Android App Design</u>

As mentioned in the requirements, the Tripency web application will be supported by a small app that consumes the rest API and displays the user’s itinerary details. This is stored locally using an android SQLite database. The Android’s only domain objects is the Itinerary class, Attraction class and AttractionReview class. This allows the user to write an AttractionReview on their phone and upload it for an Attraction. The design of these three domain objects matches exactly the class definitions for the main Tripency web application. The application uses the ‘com.loopj.android.http’ library to expose HTTP functionality to the app.

2.3 *Implementation*

2.3.1 Project Structure

The project, as written using the Eclipse IDE, is outlined in the following page:

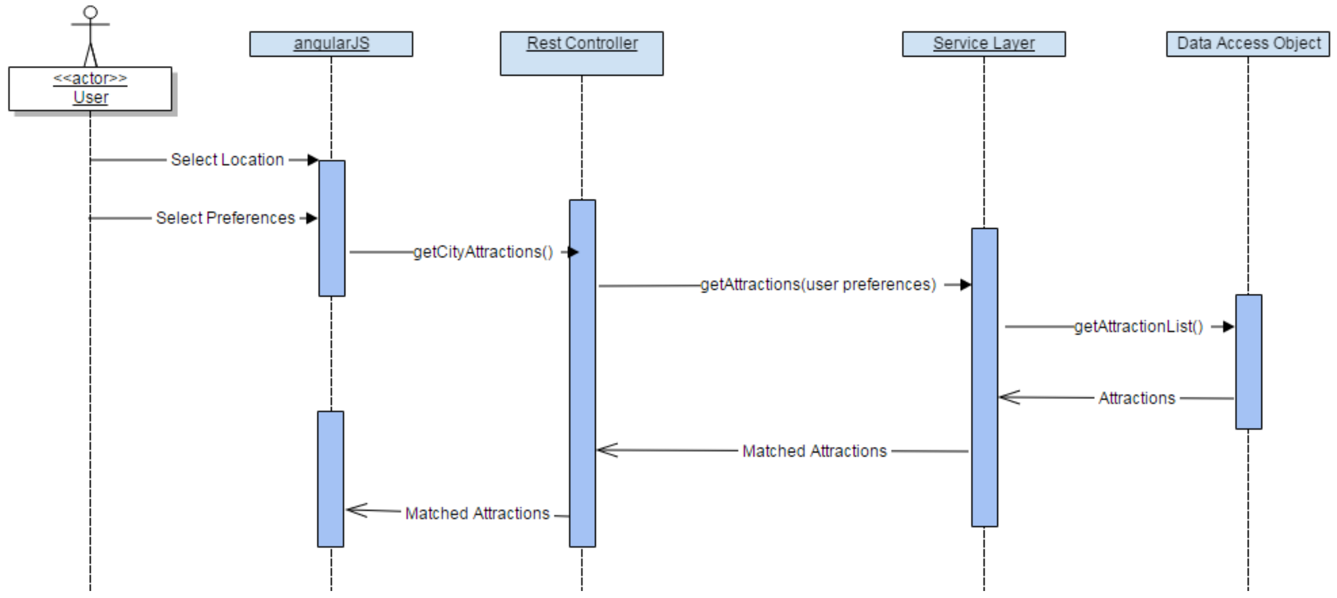


```
Java Resources
  src
    com.damienfenton.tripency.config
      SpringRootConfig.java
      SpringWebConfig.java
      dao-context.xml
    com.damienfenton.tripency.controllers
      AttractionController.java
      HomeController.java
      ItineraryController.java
    com.damienfenton.tripency.dao
      ApplicationDao.java
      AttractionDao.java
      ItineraryDao.java
    com.damienfenton.tripency.model
      Attraction.java
      AttractionReview.java
      Day.java
      DayAttraction.java
      Directions.java
      DirectionStep.java
      DirectionTransit.java
      IntegerResponse.java
      Itinerary.java
      ItineraryRequest.java
      StringResponse.java
      ThemeWeight.java
      UserPreferences.java
    com.damienfenton.tripency.security
      AuthFailure.java
      AuthSuccess.java
      EntryPointUnauthorizedHandler.java
      SecurityConfig.java
      SecurityWebAppInitializer.java
    com.damienfenton.tripency.services
      AttractionService.java
      DirectionsService.java
      EmailService.java
      ItineraryService.java
      MatchingService.java
    com.damienfenton.tripency.services.implementation
      AttractionServiceImpl.java
      DirectionsServiceImpl.java
      EmailServiceImpl.java
      ItineraryServiceImpl.java
      MatchingServiceImpl.java
    com.damienfenton.tripency.servlet3
    log4j.properties
```

2.3.2 Itinerary Generation

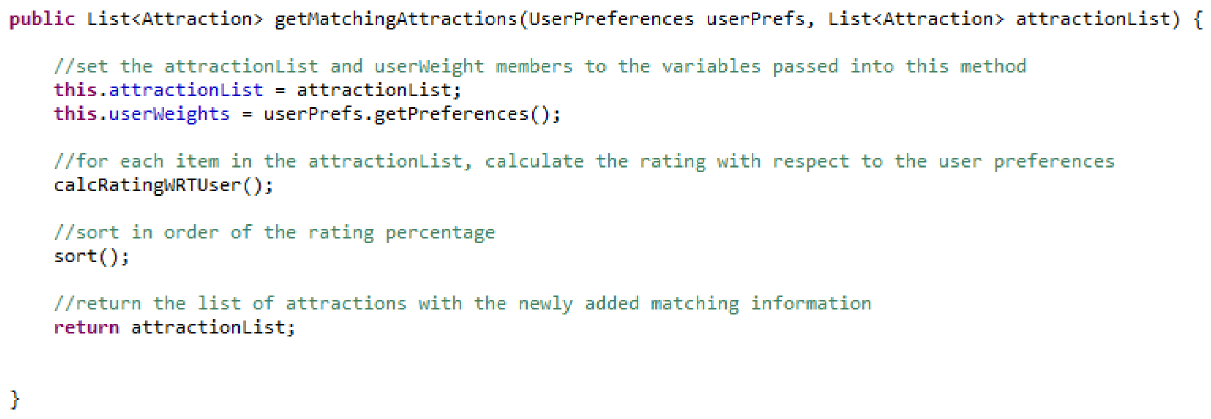
The most complex and lengthy task performed by the application is the actual end to end process of generating an itinerary from the category selection to the saving of the itinerary. This involves the following separate sub processes:

Get Attractions With Respect To User Preferences



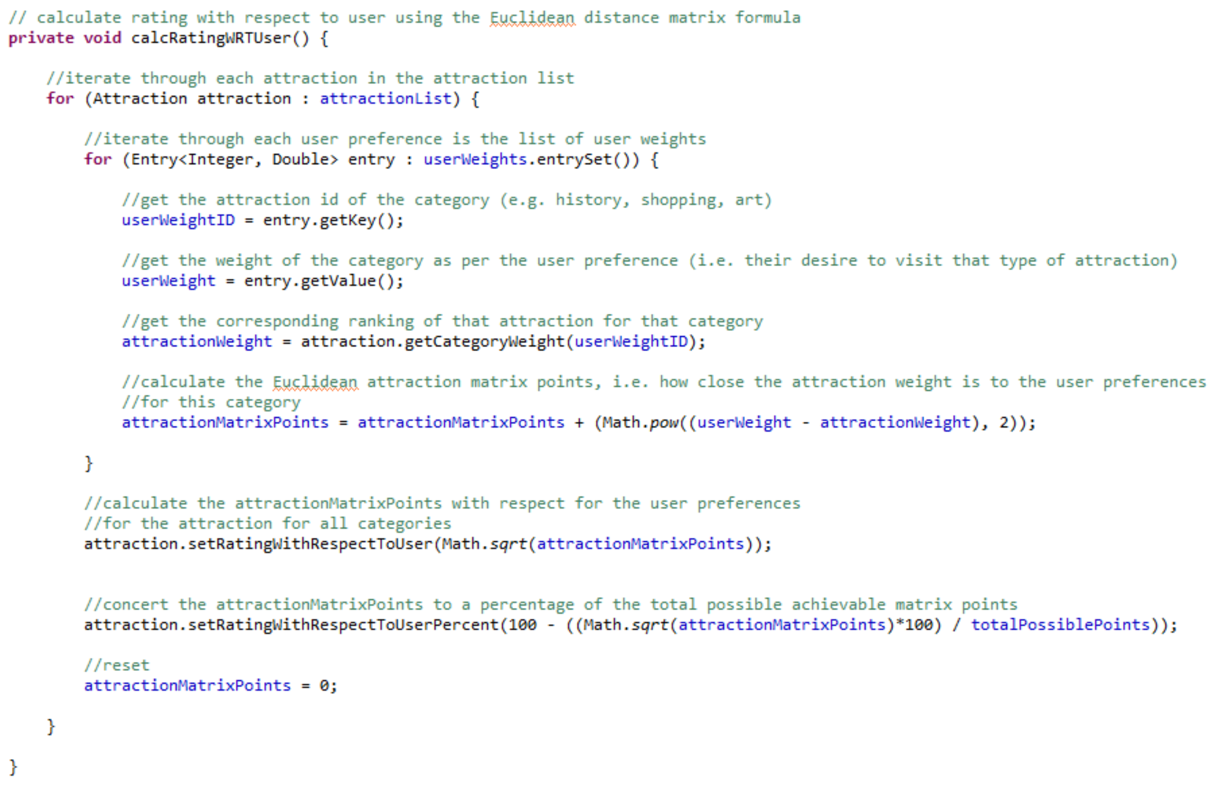
1. The user selects the location city they wish to visit. This selection is stored by angularjs.
2. The application displays a list of categories from which the user defines their preferences.
3. The angularjs PreferenceController sends this location and preference information to the 'attraction/getCityAttractions' rest endpoint
4. The rest controller forwards this request to the itineraryService in the Service layer
5. The service layer calls the getAttractionList method in the DAO layer to return from the database all the attractions for the selected location.
6. The itineraryService passes both the attraction list and user prefernces to the 'getMatchingAttractions' method of the MatchingService. The MatchingService is

an object also in the service layer that is responsible for implementing the Euclidean distance matching algorithm.



```java
public List<Attraction> getMatchingAttractions(UserPreferences userPrefs, List<Attraction> attractionList) {

    //set the attractionList and userWeight members to the variables passed into this method
    this.attractionList = attractionList;
    this.userWeights = userPrefs.getPreferences();

    //for each item in the attractionList, calculate the rating with respect to the user preferences
    calcRatingWRTUser();

    //sort in order of the rating percentage
    sort();

    //return the list of attractions with the newly added matching information
    return attractionList;


}
```

The above ‘getMatchingAttractions’ method is the public entry method for the class. The actual Euclidean calculation is performed by the private calcRatingWRTUser method, as shown below:

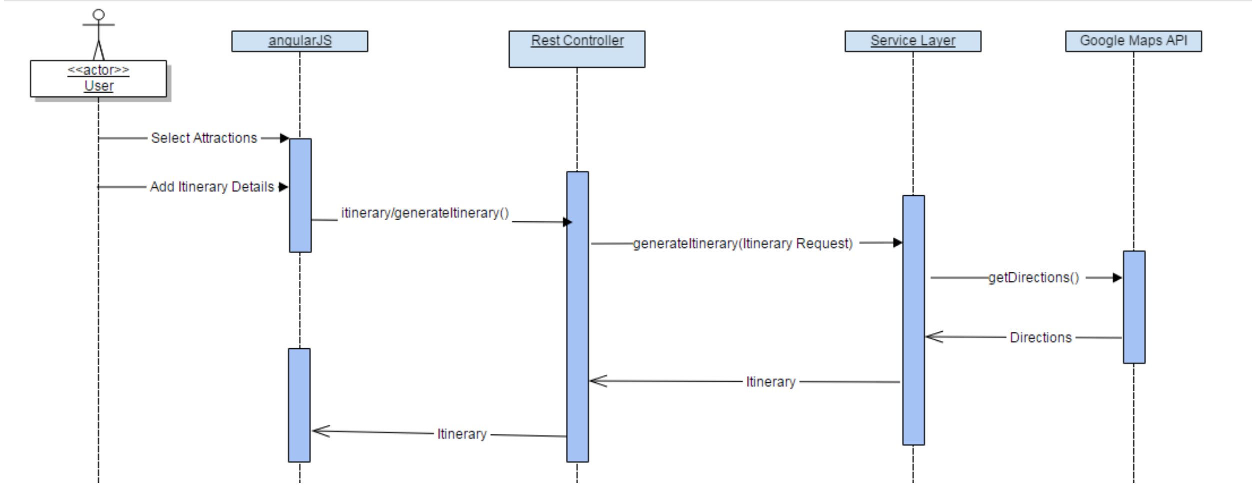


```java
// calculate rating with respect to user using the Euclidean distance matrix formula
private void calcRatingWRTUser() {

    //iterate through each attraction in the attraction list
    for (Attraction attraction : attractionList) {

        //iterate through each user preference is the list of user weights
        for (Entry<Integer, Double> entry : userWeights.entrySet()) {

            //get the attraction id of the category (e.g. history, shopping, art)
            userWeightID = entry.getKey();

            //get the weight of the category as per the user preference (i.e. their desire to visit that type of attraction)
            userWeight = entry.getValue();

            //get the corresponding ranking of that attraction for that category
            attractionWeight = attraction.getCategoryWeight(userWeightID);

            //calculate the Euclidean attraction matrix points, i.e. how close the attraction weight is to the user preferences
            //for this category
            attractionMatrixPoints = attractionMatrixPoints + (Math.pow((userWeight - attractionWeight), 2));

        }

        //calculate the attractionMatrixPoints with respect for the user preferences
        //for the attraction for all categories
        attraction.setRatingWithRespectToUser(Math.sqrt(attractionMatrixPoints));


        //concert the attractionMatrixPoints to a percentage of the total possible achievable matrix points
        attraction.setRatingWithRespectToUserPercent(100 - ((Math.sqrt(attractionMatrixPoints)*100) / totalPossiblePoints));

        //reset
        attractionMatrixPoints = 0;

    }

}
```

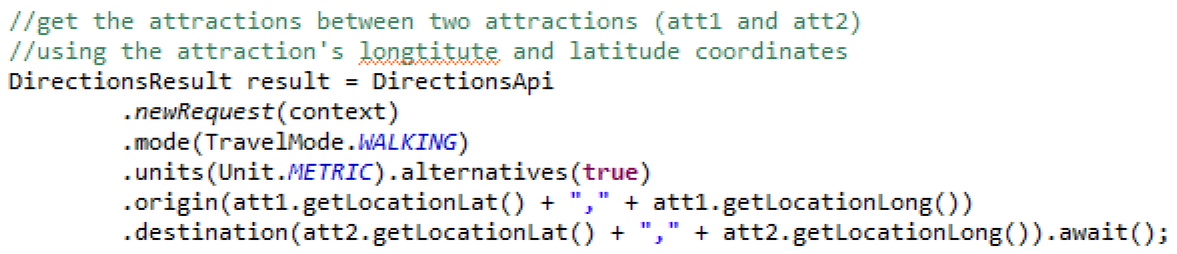
7. This list of matched attractions is returned to the rest controller

8. The rest controller in turn converts the list to a JSON string and returns it to the angularjs controller.
9. The angularjs controller sets its scope to the JSON data and displays the list of attractions in the attractionselection.html page.

User Finalises Itinerary



1. The user selects the attractions to appear on the itinerary based on Tripency's ssuggestions in the order they want to visit.
2. The user also adds additional information such as a description of the itinerary
3. The angularjs controller sends this data to the server's rest controller.
4. The rest controller passes this to the service layer as a ItineraryRequest object
5. The service layer calls a Google Maps API to get the distances and directions information between each attraction while maintaining the order as specified by the user, as shown below:

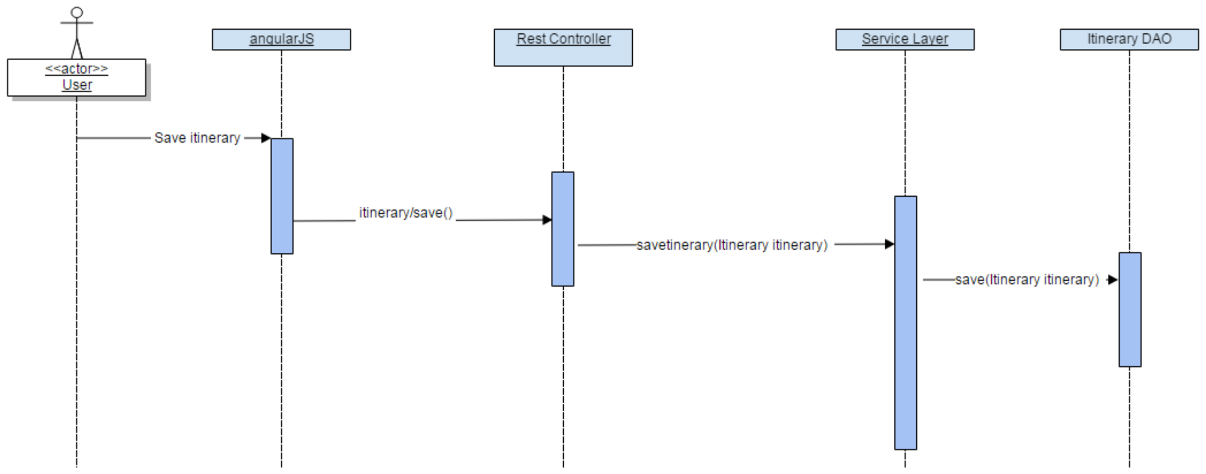

```
//get the attractions between two attractions (att1 and att2)
//using the attraction's longtitute and latitude coordinates
DirectionsResult result = DirectionsApi
        .newRequest(context)
        .mode(TravelMode.WALKING)
        .units(Unit.METRIC).alternatives(true)
        .origin(att1.getLocationLat() + "," + att1.getLocationLong())
        .destination(att2.getLocationLat() + "," + att2.getLocationLong()).await();
```

6. The information returned by the Google API is then mapped to the Directions, DirectionStep and DirectionTransit classes as shown in the class diagram above.
7. The direction information is then added to the attraction.
8. The attractions are then added to an Itinerary object using that object's 'addAttractions' method. This method keeps track of the time spent in each day (including travel time) and allocates the attractions to days accordingly

```java
//iterate through the list of attractions
for (int i = 0; i < items.size(); i++) {

    //get the current attraction
    Attraction currentAttraction = items.get(i);

    //add that attractions recommended duration to the day's durations
    dayDuration = dayDuration + currentAttraction.getDuration();

    //add that attraction to the day
    day.addAttraction(currentAttraction);

    //if there is another attraction in the list
    if (i + 1 < items.size()) {

        //get that next attraction
        Attraction nextAttraction = items.get(i + 1);

        //get the time it takes to travel from the current attraction to the next attraction
        long nextAttractionTravelTime = currentAttraction.getDirections().getDuration();

        //if the time it takes to travel to the next attraction, plus the recommended duration for that attraction
        //will exceed the remaining time left in the day
        //that means no more attractions can be added to the current day
        //so we add the day to the itinerary's list of days and then create a new Day and reset the dayDuration
        if ((dayDuration + nextAttractionTravelTime + nextAttraction.getDuration()) > theshold){

            days.add(day);
            dayCount = dayCount + 1;
            day = new Day(dayCount);
            dayDuration = 0;

        }

    //if the current attraction is the last attraction in the list
    } else if (i == (items.size() - 1)){

        //add the day to the itinerary's list of days
        days.add(day);

    }

}
```

9. The itinerary object is then returned from the service layer to the rest controller.
10. The rest controller returns the itinerary to the angularjs client as a JSON string
11. The angualjs controller sets its scope to the itinerary JSON data and displays the newly generated itinerary to the user in the itineraryview.html page.

User Saves Itinerary



1. Saving the itinerary (i.e persisting the itinerary to the database) simply involves passing the itinerary and the save command along the chain from the angularjs controller to the itineraryDao.
2. The Hiberate save method is used in the saveItinerary method in the DAO. As discussed earlier, this automatically maps the itinerary object's data to SQL, and then executes that SQL against the database.

```
public void saveItinerary(Itinerary itinerary) {

    session().save(itinerary);

}
```

2.3.3 Add Attraction Review

Saving a review to an attraction basically follows the same sequence as the "User Saves Itinerary" sequence above except the object being passed through the layers is an AttractionReview object.

2.3.4 Editing an Attraction

Editing an attraction basically follows the same sequence as the "User Saves Itinerary" sequence above except the object being passed through the layers is an Attraction object.

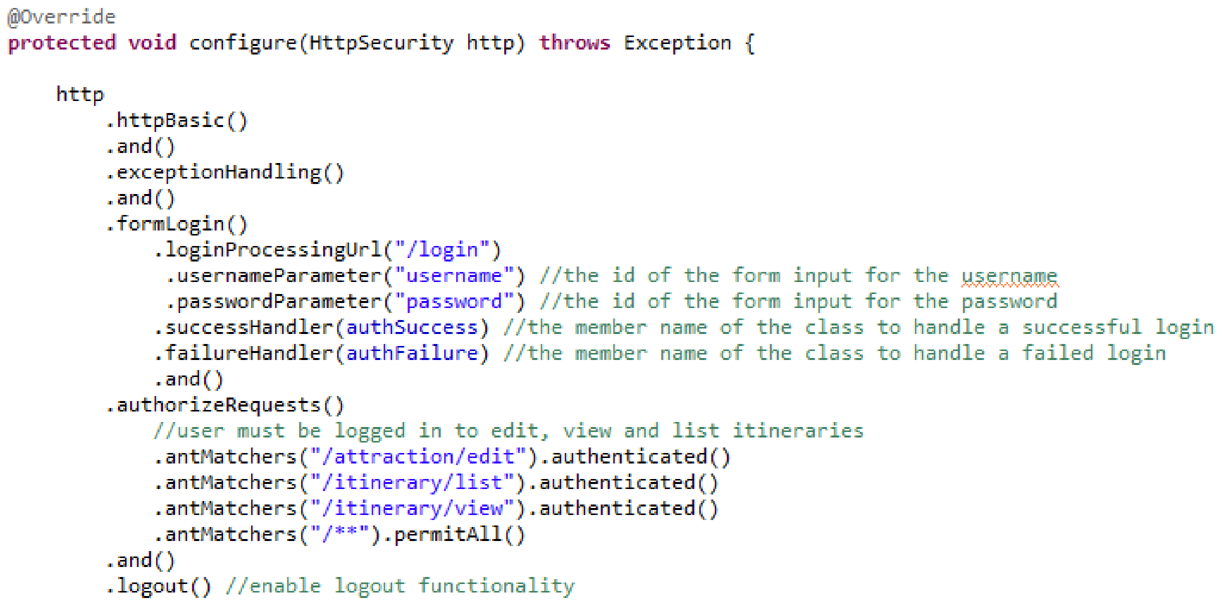
2.3.5 Email Functionality

Email functionality, i.e. emailing the itinerary to a user's email account, is providing using a combination of two technologies: (1) Spring's JavaMail library; and (2) Velocity templates. The JaveMail library provides an object called the MimeMessagePreparator which is responsible for creating the actual email. The Velocity template engine actually works very similarly to JSPs. There is a template file written in HTML markup (in this case, a .vm file), and a data model in the form of a map. Our user's itinerary is added to this map, and Velocity's 'mergeTemplateIntoString' method adds the model's data to the template to return a generated email. That email is then sent. This is shown below:

```java
public class EmailService {

    private void sendItineraryEmail(Itinerary itinerary, User user) {

        MimeMessagePreparator preparator = new MimeMessagePreparator() {

            public void prepare(MimeMessage mimeMessage) throws Exception {

                //create message object
                MimeMessageHelper message = new MimeMessageHelper(mimeMessage);

                //set the To address using the user object's email address
                message.setTo(user.getEmailAddress());

                //set the From address
                message.setFrom("tripency_test_email@gmai.com");

                //create a new map object and add the itinerary to it
                //the itinerary was passed into this method as a parameter
                Map model = new HashMap();
                model.put("itinerary", itinerary);

                //generate the body of the email using the velocity template engine.
                //The path to the email template and the model containing the itinerary are passed as parameters
                String emailBody = VelocityEngineUtils.mergeTemplateIntoString(velocityEngine, "itinerary-email.vm", model);

                //add the email body to the message
                message.setText(emailBody, true);

            }
        };

        //send the message
        this.mailSender.send(preparator);

    }
```

2.3.6 Security

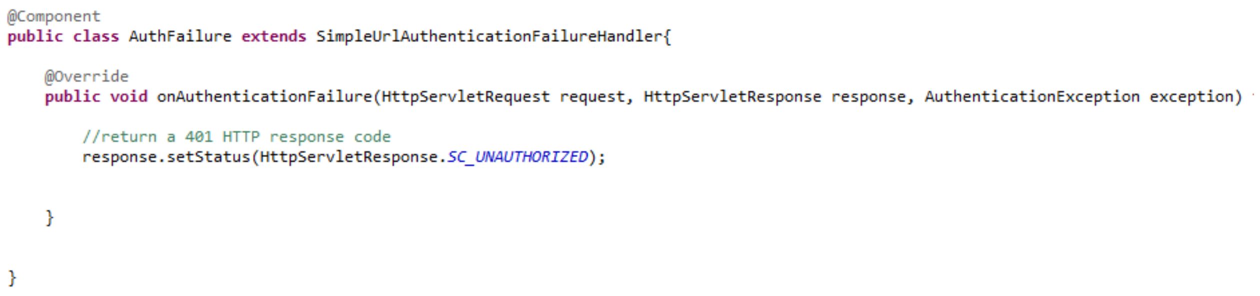
User authentication and authorisation is provided by Spring Security. Enabling Spring Security for a Spring based server is a relatively straight forward affair. The developer simply needs to create a class called SecurityWebAppInitializer which must extend Spring's AbstractSecurityWebApplicationInitializer class. Once that has been completed, the actual configuration of the security setting can be done using a configuration class, as shown below for Tripency.



```
@Override
protected void configure(HttpSecurity http) throws Exception {

    http
        .httpBasic()
        .and()
        .exceptionHandling()
        .and()
        .formLogin()
            .loginProcessingUrl("/login")
             .usernameParameter("username") //the id of the form input for the username
             .passwordParameter("password") //the id of the form input for the password
            .successHandler(authSuccess) //the member name of the class to handle a successful login
            .failureHandler(authFailure) //the member name of the class to handle a failed login
            .and()
        .authorizeRequests()
            //user must be logged in to edit, view and list itineraries
            .antMatchers("/attraction/edit").authenticated()
            .antMatchers("/itinerary/list").authenticated()
            .antMatchers("/itinerary/view").authenticated()
            .antMatchers("/**").permitAll()
        .and()
        .logout() //enable logout functionality
```

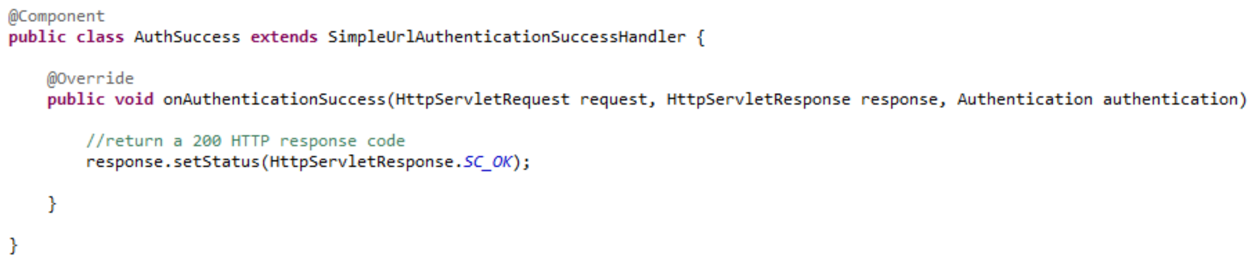
The authSuccess and authFailure members refer to objects that have been autowired into the configuration class that handles successful or failed login attempts respectfully. Typically, these classes return a HTTP response code, as shown below:

A failed login attempt is routed through the AuthFailure class:



```
@Component
public class AuthFailure extends SimpleUrlAuthenticationFailureHandler{

    @Override
    public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response, AuthenticationException exception)

        //return a 401 HTTP response code
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);


    }

}
```

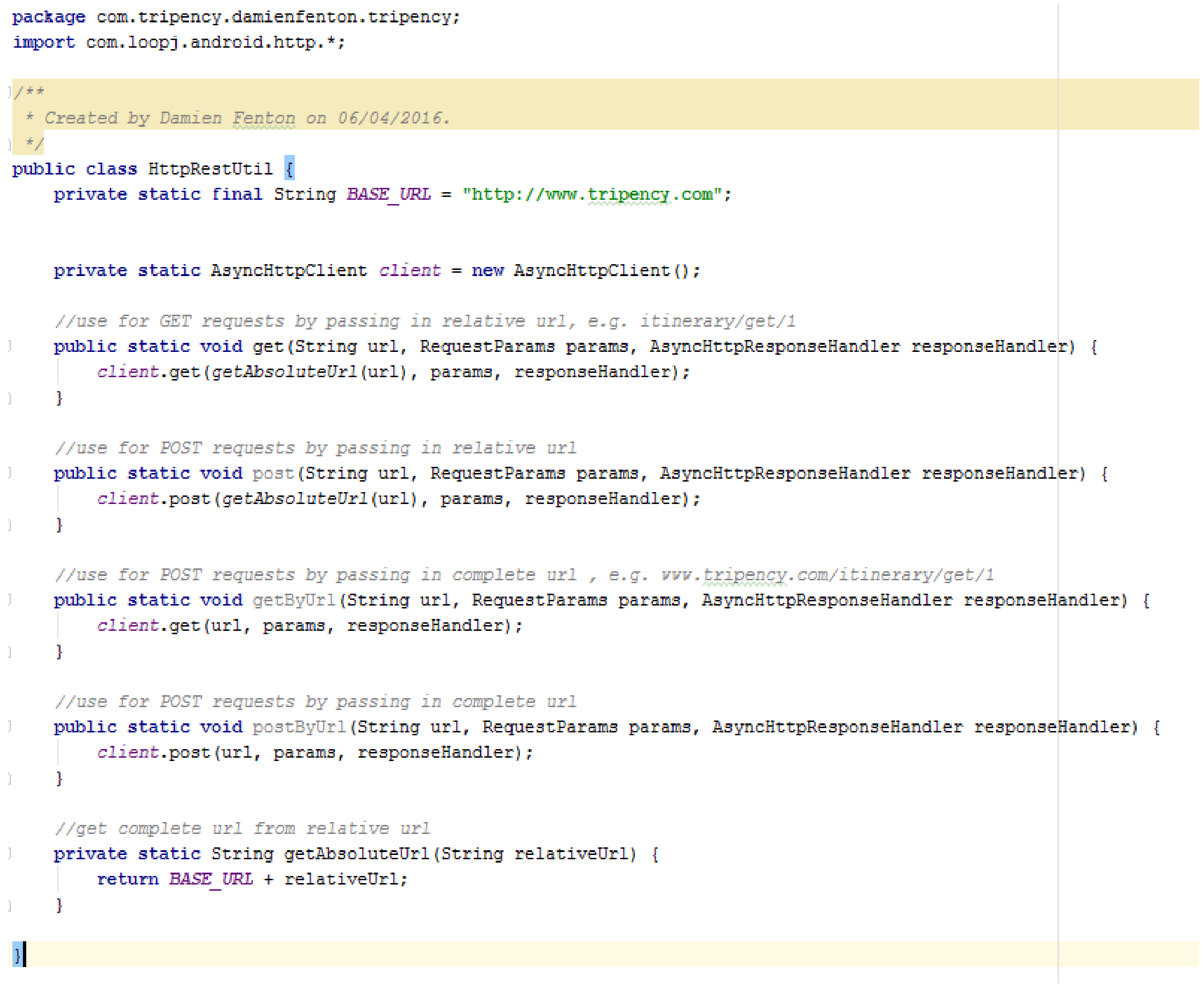
A successful login attempt is routed through the AuthSuccess class:



```java
@Component
public class AuthSuccess extends SimpleUrlAuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication)

        //return a 200 HTTP response code
        response.setStatus(HttpServletResponse.SC_OK);

    }

}
```

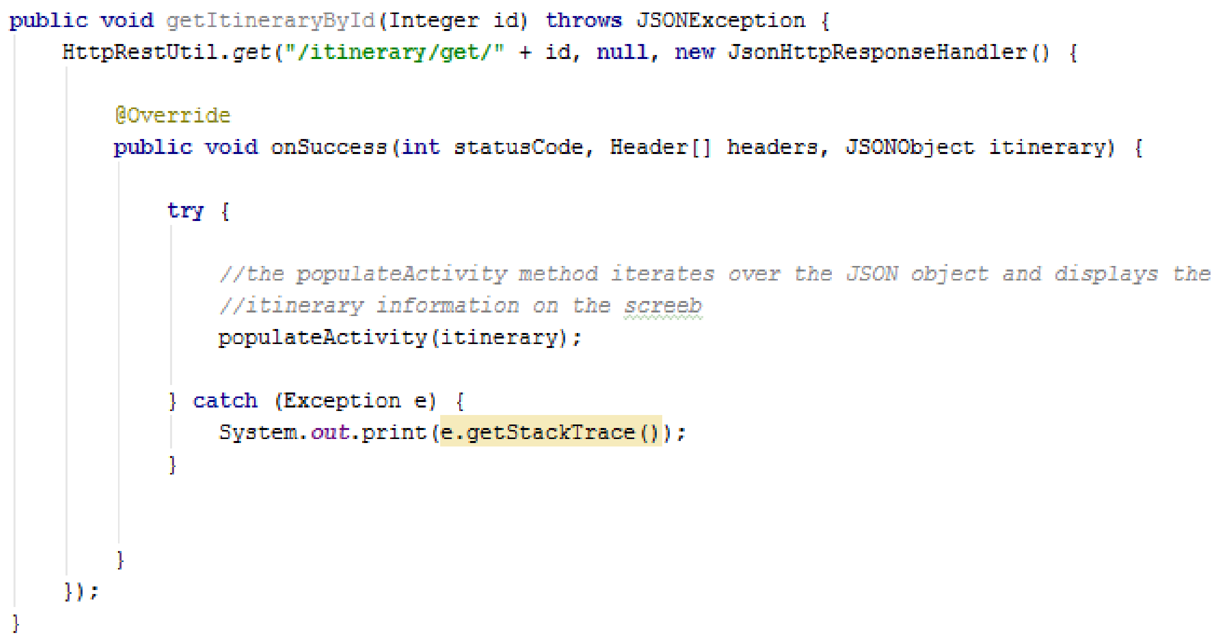
2.3.7 Consume Itinerary API from Android App

1. Consuming the rest API from an Android app is arguably simpler because of the extra complexities the angularjs framework adds, which is obviously not relevant for Android. First, I created a generate HTTP Rest utility class that can all handle HTTP requests to Tripency's server.



```java
package com.tripency.damienfenton.tripency;
import com.loopj.android.http.*;

/**
 * Created by Damien Fenton on 06/04/2016.
 */
public class HttpRestUtil {
    private static final String BASE_URL = "http://www.tripency.com";


    private static AsyncHttpClient client = new AsyncHttpClient();

    //use for GET requests by passing in relative url, e.g. itinerary/get/1
    public static void get(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.get(getAbsoluteUrl(url), params, responseHandler);
    }

    //use for POST requests by passing in relative url
    public static void post(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.post(getAbsoluteUrl(url), params, responseHandler);
    }

    //use for POST requests by passing in complete url , e.g. www.tripency.com/itinerary/get/1
    public static void getByUrl(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.get(url, params, responseHandler);
    }

    //use for POST requests by passing in complete url
    public static void postByUrl(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.post(url, params, responseHandler);
    }

    //get complete url from relative url
    private static String getAbsoluteUrl(String relativeUrl) {
        return BASE_URL + relativeUrl;
    }

}
```

2. Secondly, I am consuming this service from a TripencyRestClient class that has various methods for getting itineraries and posting attraction reviews. For example, the code snippet below calls the ‘get’ method of the utility class, and passes the returned JSON object to a populateActivity method which shows the information on screen.



```
public void getItineraryById(Integer id) throws JSONException {
    HttpRestUtil.get("/itinerary/get/" + id, null, new JsonHttpResponseHandler() {

        @Override
        public void onSuccess(int statusCode, Header[] headers, JSONObject itinerary) {

            try {

                //the populateActivity method iterates over the JSON object and displays the
                //itinerary information on the screeb
                populateActivity(itinerary);

            } catch (Exception e) {
                System.out.print(e.getStackTrace());
            }


        }
    });
}
```

2.3.8 Social Media Connectivity

The ability to tweet using Twitter and share using Facebook is provided by simply using the API’s provided by those two organisations.

2.4 Testing

2.4.1 User Acceptance Testing

User Test Cases

The UAT was performed using the following sample user base that was chosen as a sample of the demographic that would use the application:

User A

A digital marketing professional working in the travel industry. Strong knowledge of the travel industry and strong computer skills.

User B

A professional not working within the travel industry but with a keen interest to travel Also strong computer skills

User C

A retired person with a keen interest to travel. Also strong computer skills

User D

A retired person with a keen interest to travel. Weak computer skills

User E

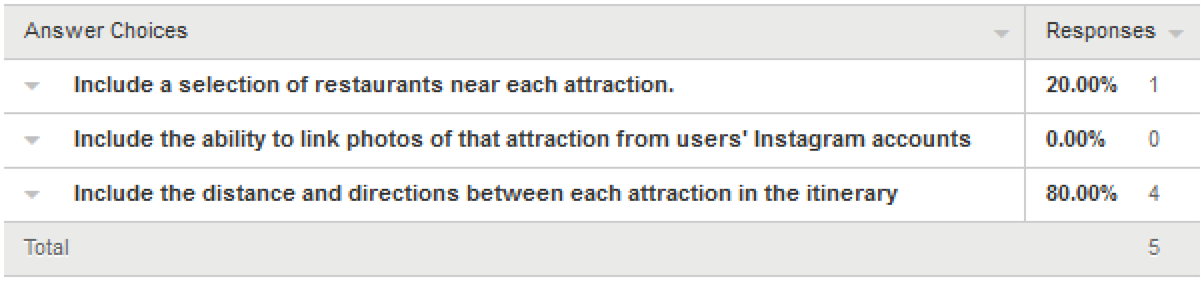
A professional who rarely travels. Good computer skills.

User Test Results 1

I performed two rounds of UAT. The first round was during the project to ensure I was on the right track. This gave me time to correct any challenges that may have been presented during UAT and also allowed me to survey a possible new functionality not in the original requirements.

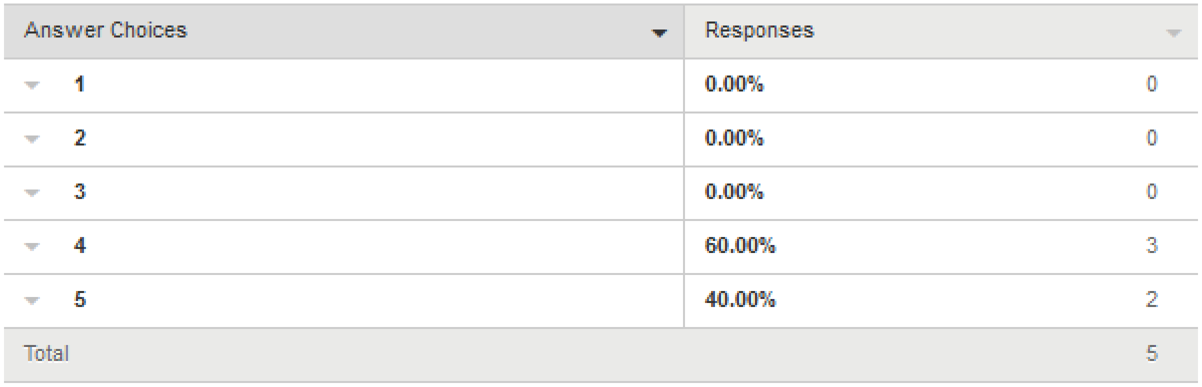
The results are as follows (screenshots from SurveryMonkey):

Question: What feature would you most like to see added to the itinerary in Tripency



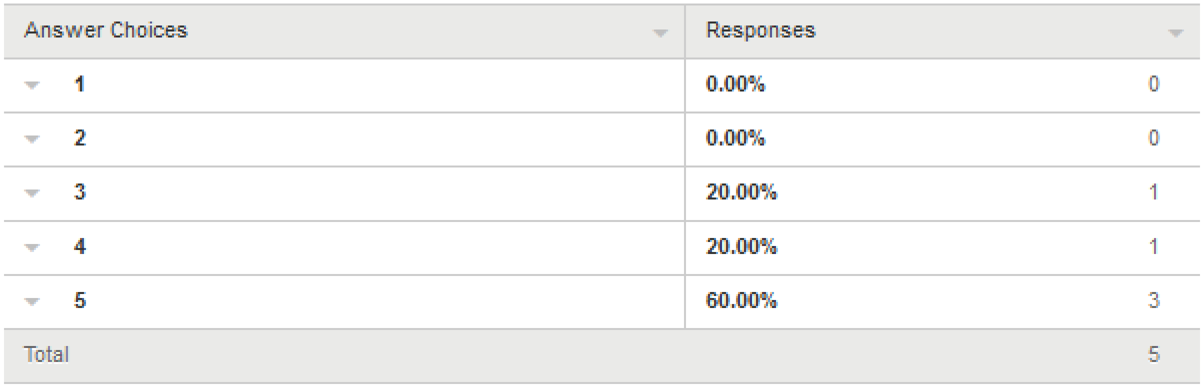
Answer Choices	Responses	
Include a selection of restaurants near each attraction.	20.00%	1
Include the ability to link photos of that attraction from users' Instagram accounts	0.00%	0
Include the distance and directions between each attraction in the itinerary	80.00%	4
Total		5

Question: On a scale of 1 - 5, how easy was it to select your desired categories (5 being the highest)?



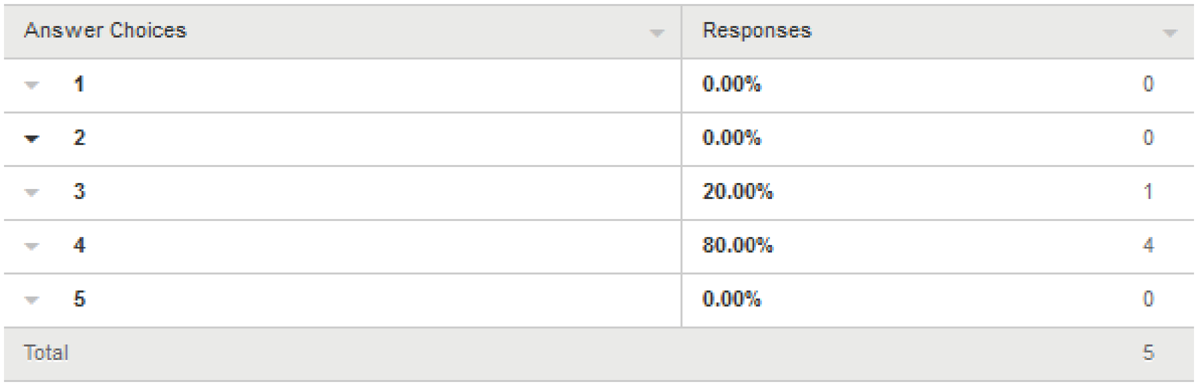
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	60.00%	3
5	40.00%	2
Total		5

Question: On a scale of 1 - 5, how easy was it to order the list of attractions on your itinerary (5 being the highest)?



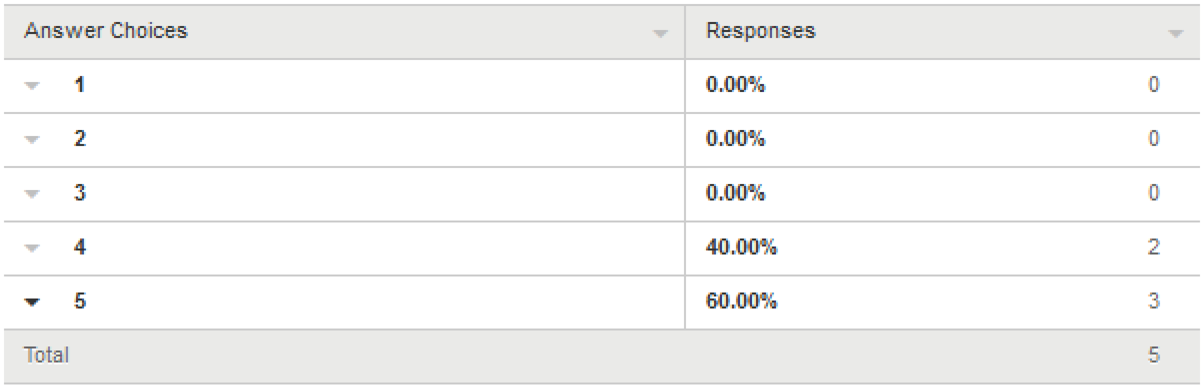
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	20.00%	1
4	20.00%	1
5	60.00%	3
Total		5

Question: On a scale of 1 - 5, how easy was it to complete the generation of the itinerary and save it (5 being the highest)?



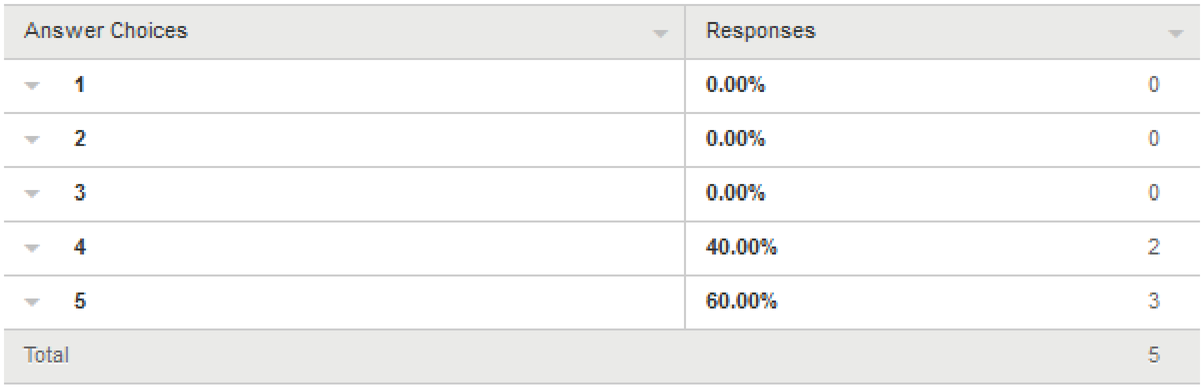
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	20.00%	1
4	80.00%	4
5	0.00%	0
Total		5

Question: On a scale of 1 - 5, how easy was it to edit an attraction (5 being the highest)?



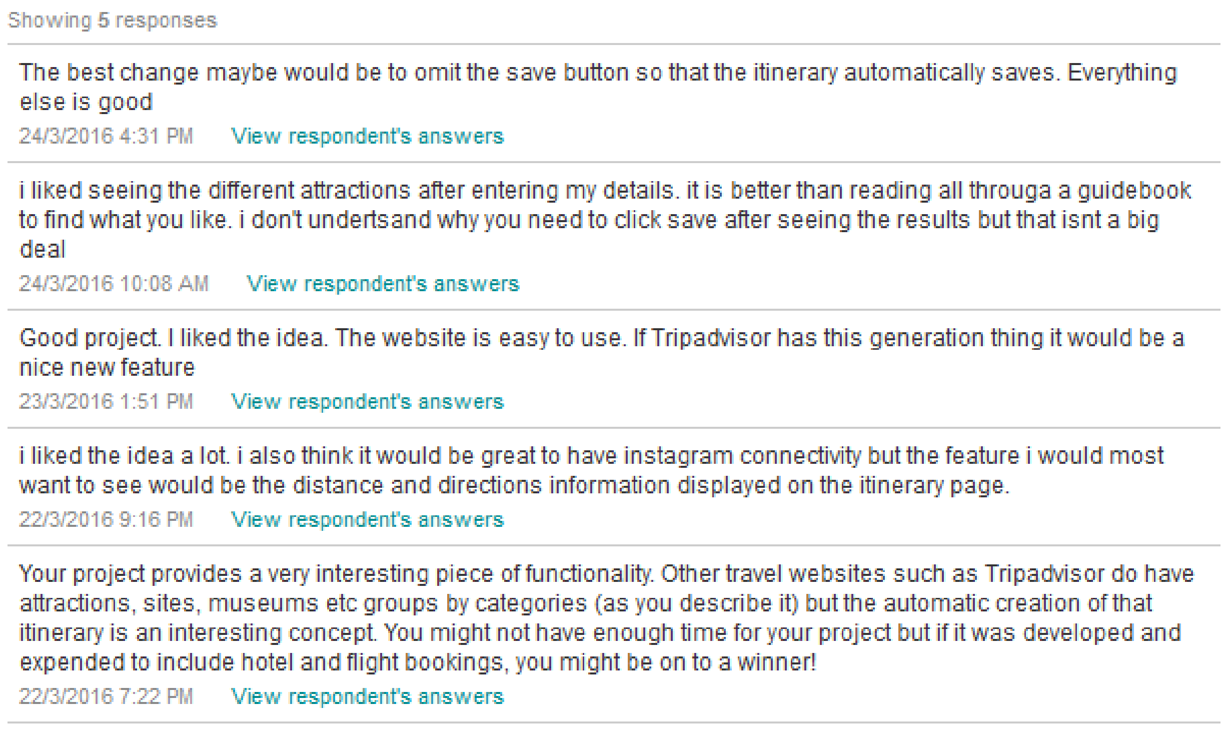
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	40.00%	2
5	60.00%	3
Total		5

Question: On a scale of 1 - 5, how would you rate your experience using Tripency (5 being the highest)?



Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	40.00%	2
5	60.00%	3
Total		5

Question: Please provide any additional comments, feedback or criticisms.


Showing 5 responses

The best change maybe would be to omit the save button so that the itinerary automatically saves. Everything else is good
24/3/2016 4:31 PM View respondent's answers

i liked seeing the different attractions after entering my details. it is better than reading all througa a guidebook to find what you like. i don't undertsand why you need to click save after seeing the results but that isnt a big deal
24/3/2016 10:08 AM View respondent's answers

Good project. I liked the idea. The website is easy to use. If Tripadvisor has this generation thing it would be a nice new feature
23/3/2016 1:51 PM View respondent's answers

i liked the idea a lot. i also think it would be great to have instagram connectivity but the feature i would most want to see would be the distance and directions information displayed on the itinerary page.
22/3/2016 9:16 PM View respondent's answers

Your project provides a very interesting piece of functionality. Other travel websites such as Tripadvisor do have attractions, sites, museums etc groups by categories (as you describe it) but the automatic creation of that itinerary is an interesting concept. You might not have enough time for your project but if it was developed and expended to include hotel and flight bookings, you might be on to a winner!
22/3/2016 7:22 PM View respondent's answers

Analysis:

The feedback is overall positive which is encouraging. All the usability questions scored at least a 3 with the average being 4.36. I am therefore happy with that.

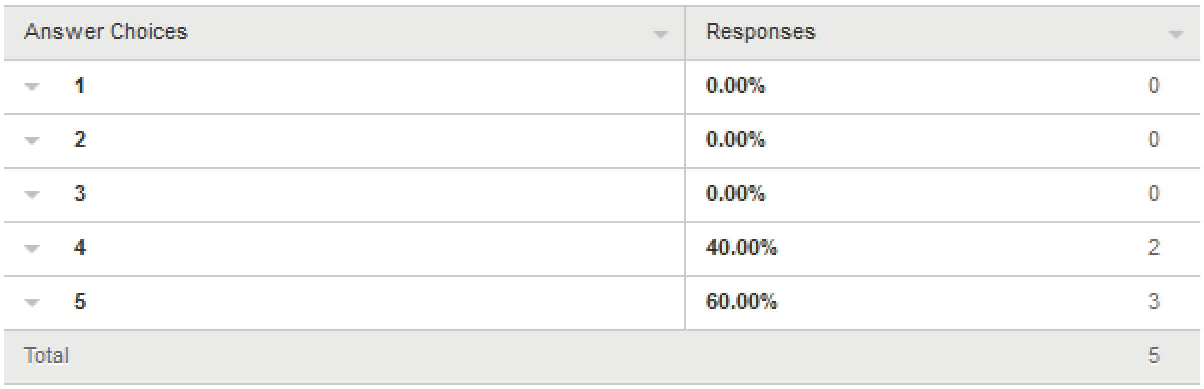
Two of the respondents mentioned the necessity of saving the itinerary automatically rather than requiring the user to click the save button. While I understand this from a user viewpoint, I don't believe that should be implemented because the user may change the itinerary but then decide they made a mistake and do wish the changes to be persisted. If I implemented an autosave feature, such a mistake would be automatically saved. The ability to undo saved changes is not in scope for this project. However, I view it as a minor criticism, the overall score was positive so I am willing to accept it as a minor change request for future development.

Finally, there was overwhelming desire to add distance and directions to the itinerary. It should be noted that the user with perhaps the most insights into the industry, User A, asked for the selection of restaurants options. However, as all out four respondents choose the distance/directions options, I implemented that functionality instead. I couldn't ignore the result.

User Test Results 2

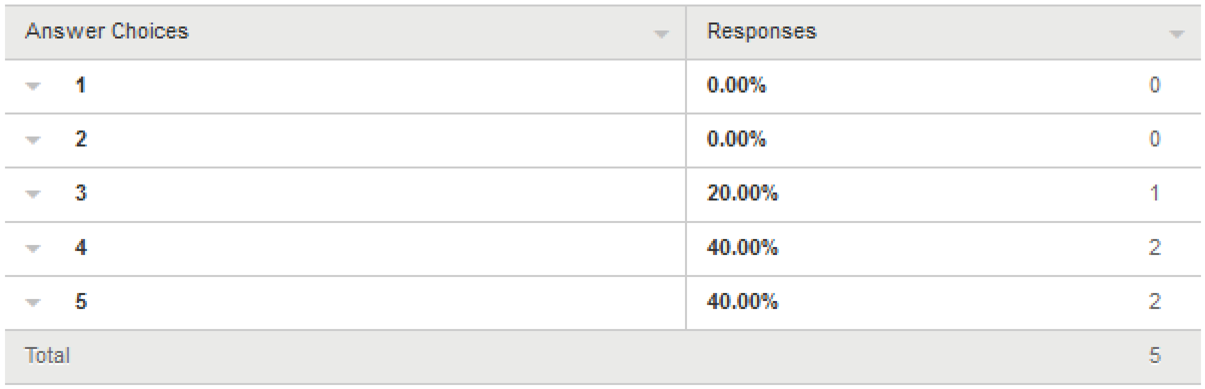
The second round of UAT was performed using the same selection of users in order to gauge any trends from the first round of UAT. The UI was redesigned and the requested directions/distance was added. This second round of UAT was performed close to the project deadline and is intended to ensure the final project meets the required functionality from a user viewpoint. The results are:

Question: On a scale of 1 - 5, how would you rate the new directions/distance information added to the itinerary (5 = highest)?



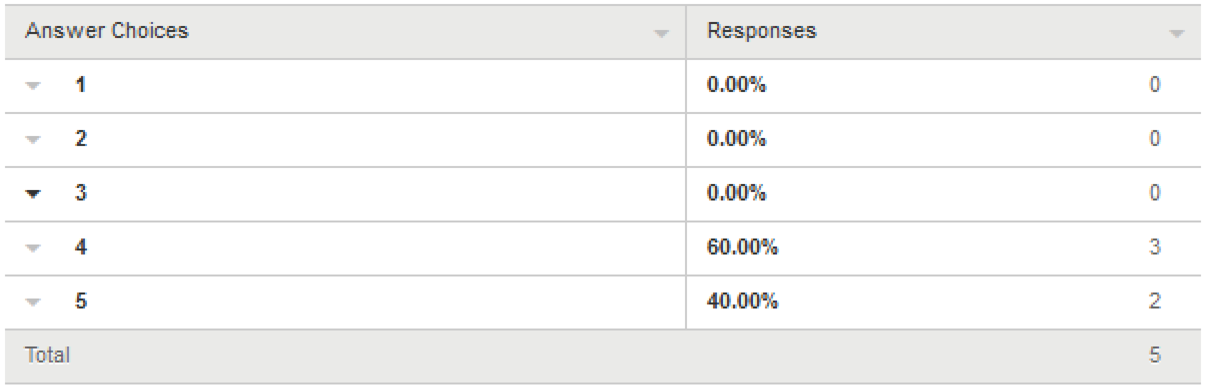
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	40.00%	2
5	60.00%	3
Total		5

Question: On a scale of 1 - 5, how would you rate the changes to the UI since you last used the website (5 = highest)



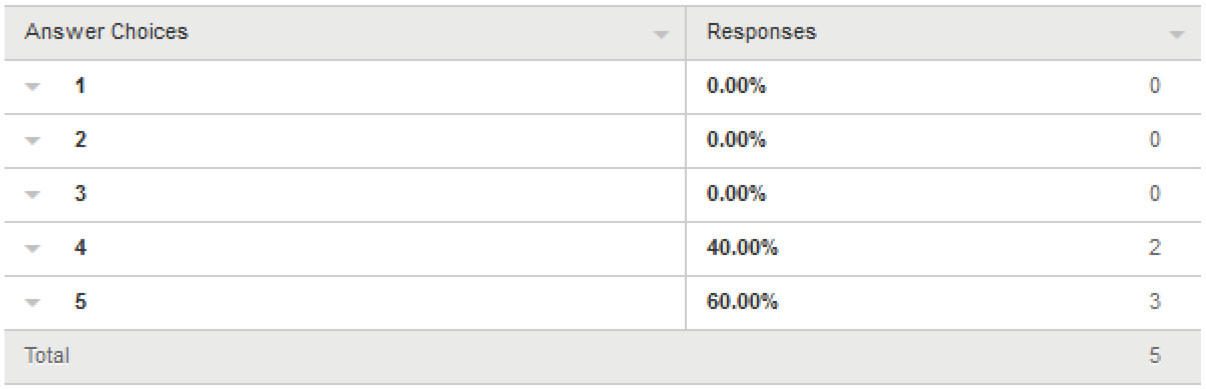
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	20.00%	1
4	40.00%	2
5	40.00%	2
Total		5

Question: On a scale of 1 - 5, how would you rate your overall experience with Tripency (5 = highest)?



Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	60.00%	3
5	40.00%	2
Total		5

Question: On a scale of 1 - 5, would you consider using Tripency when planning your next holiday?



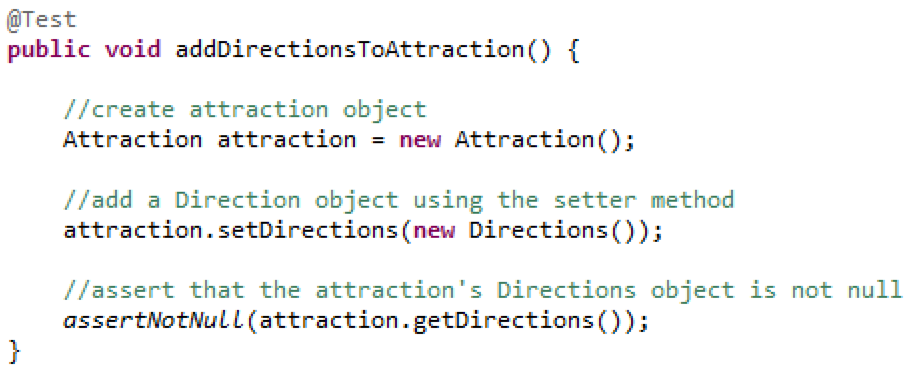
Answer Choices	Responses	
1	0.00%	0
2	0.00%	0
3	0.00%	0
4	40.00%	2
5	60.00%	3
Total		5

Analysis

Overall I am very happy with the second round of UAT and the results have shown I have not introduced any defects that would negatively affect the user experience. Also, the distance and directions details have been met with a warm response following user's suggestion that they would like to see them added to the itinerary generation process.

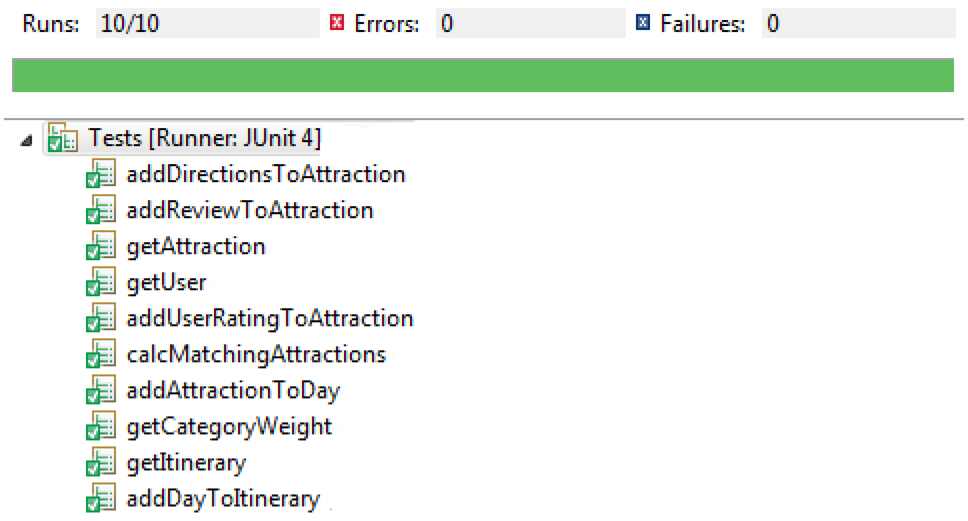
2.4.2 Unit Test

Unit testing was performed using JUnit. This is a very common unit testing framework whereby the developer asserts certain outcomes and the test passes if the code being tested provides those outcomes. For example, the following test asserts that a Directions object has been successfully added to the Attraction object



```
@Test
public void addDirectionsToAttraction() {

    //create attraction object
    Attraction attraction = new Attraction();

    //add a Direction object using the setter method
    attraction.setDirections(new Directions());

    //assert that the attraction's Directions object is not null
    assertNotNull(attraction.getDirections());
}
```

jUnit tests are run directly in the Eclipse IDE and the results are displayed, as shown below with all tests passing:



```
Runs: 10/10    Errors: 0    Failures: 0

Tests [Runner: JUnit 4]
    addDirectionsToAttraction
    addReviewToAttraction
    getAttraction
    getUser
    addUserRatingToAttraction
    calcMatchingAttractions
    addAttractionToDay
    getCategoryWeight
    getItinerary
    addDayToItinerary
```

3 Appendix

3.1 Project Proposal

1 Objectives

The objective of the project is to develop a web application called “Tripency”. The name “Tripency” is a combination of the word “trip” and “efficiency”. The primary functionality of Tripency will be to assist users with efficiently planning their holidays by dynamically matching their interests (and disinterests) with attractions, cities and other locations. The user will then be provided with a tailored list of attractions from which they can build their itinerary. The initial scope of the web application will be to primarily cater for city-break holidays instead of beach and sports holidays due to the much wider variety of attractions, museums and monuments typically found in towns and cities.

In addition to this tailored itinerary builder, the web application will also provide user profile management functionality typically found in modern websites. This includes the ability to save and retrieve itineraries, and also for the site to “remember” the user’s interests so that building subsequent holidays takes less time. Also, similar to Amazon and other online retailers, the web application will automatically suggest other possible destinations based on those saved user interests.

There will also be a community aspect to the web application which will be leveraged by the system to determine how a user’s interests matches the various attractions. Users will have the ability to write text-based reviews on each attraction, but will also have the ability to rate those attractions on a numeric scale against that person’s interests. For example, a user might rate London’s Tate Gallery very high for “Art”, and this would be taken into account when another user specifies “Art” as an interest for building their itinerary. The web application will also adopt the “wiki” philosophy so that users can update attraction details (such as description, opening hours, prices) themselves without relying on Tripency administrators. To further increase the community aspect, the web application will utilize social media APIs such as Facebook and Twitter so users can post/tweet about their itinerary directly from Tripency.

Finally, given the nature of the industry Tripency operates in, it would be important to allow the users to view their finished itinerary offline so that it is available to them during the actual holiday especially if internet access is not readily available or if it incurs high roaming charges. This will be accomplished with a small app that will supplement the web application. Although 90% of the project will be focused on the web application itself, the Tripency app will be developed to provide users with that important offline functionality. One alternative would have been for Tripency to be entirely app based, but an observation of the travel industry determined a strong online presence is very important, e.g. Tripadvisor, Expedia, hotels.com, etc…

To summarize, the Tripency system will provide the following main functionality:

- Tailored itinerary builder
- User profile and preferences management
- Community management through user reviews/rating, user content and social media connectivity
- Offline mobile itinerary storage via a small supplementary app

2 Background

The core idea of dynamically generating a travel itinerary based on user preferences originated from a third year project. That project was developed using basic Java servlets (see Technical Approach below), and did not include any additional functionality outlined above such as user management, user content and social media connectivity. Upon presenting the finished project, the core idea was received well but the primary critical feedback was that the user required internet access to view his itinerary during the actual holiday.

The main alternative for this project is a training management system which would be used by Human Resource departments and training companies to track and monitor their training. The developer has considerable experience in this area and believes most solutions on the market for this industry do not fully cater for the complexities and realities of administrating the training function within a large human resources department.

However, Tripency was ultimately chosen for this project because it was always the developer’s intention to expand that core idea to a full enterprise level application while taking into account the feedback received. The developer believed the third year project did not allow the core idea of dynamically generating tailored itineraries to sufficiently reach its potential. Also, the third year project utilized out-dated and relatively simplistic technology. Tripency also provided for obvious uses of social media unlike the training management application. Therefore, Tripency will be redeveloped from the ground up using modern technologies, a vastly redesigned user interface together with considerably more functionality.

3 Technical Approach

The approach to the development of Tripency is as follows:

1. Research of existing travel and travel orientated websites to gauge the design, features and UX prevalent in the industry. It is important that the website is extremely intuitive for users and instantly identifiable as a travel website while also maintaining its own brand identity.
2. Requirements document and use cases (user stories) will be written.
3. Major UI and UX design decisions will be made following industry research and requirements. This will form Tripency’s branding including logo and color scheme design. The website’s main subsections and user journey through the website will be extrapolated from the use cases. Substantial CSS and HTML work will be done at this point.
4. At this stage, user feedback from contacts in the travel industry will be sought to evaluate the website’s overall design. Adjustments to be made based on this feedback.
5. Component and Class diagrams will be created that will outline the relationships between the various Java classes. This stage will help identify any design patterns that can be leveraged, especially by the business layer of the application.
6. Angularjs and server side application development will begin using the Test Driven Development (TTD) approach. Also, a sprint based approach will be used

to organize and manage development. The content and performance of each spring will be documented in the monthly project journals. The sprints will be two weeks long and therefore each project journal will report on two completed sprints together with the expected content of next month's sprints.

4 Special resources required

The only resources required will be the Eclipse IDE for development and the MySQL database. The domain www.tripency.com has already been purchased by the developer for use with this project. The developer is currently reviewing hosting options, and an update will be provided in the project journal with regards to this.

5 Technical Details

Tripency will be written in Java using the following technologies:

- The Spring Framework will provide RESTful services to handle requests from both the web application and the app. The client / server data communication will be done using the JSON data structure. Spring will also handle core system non-functional requirements such as exception handling and logging.
- Data persistence will be provided by a MySQL database.
- Database connectivity will be handled by utilizing the Hibernate library to provide ORM support.
- The view technology will be angularjs. This will provide a modern and seamless user experience. The developer had considered using standard JSPs but decided for angularjs because it afforded him the opportunity to learn a new technology, using a newer technology in the project, and provide a "single page" UX.
- Tomcat will be used as the web server.
- The supplementary app will target the Android platform. Apple devices will not be supported in this project, although that remains an opportunity for future expansion. Developing an app for two completely different platforms using different languages will not be possible during the timeframe involved.
- GitHub will be used to provide versioning and code repository.

6 Evaluation

The unit tests will be done using the JUnit testing library. Test Driven Development will be utilized during development of Tripency. This involves first writing a test for a requirement, and then developing the functionality to make this test pass.

Integration testing will be performed on the hosting company when the hosting company has been decided upon. Early in the development cycle, a narrow yet cross functional smoke test of all technologies (angularjs, Spring, Hibernate, MySQL) will be performed on the host to ensure there are no fundamental issues with these technologies on the host. An integration test completion report will be written and an update will be provided in the project journal.

Finally, the developer has contacts in the travel industry which will be leveraged for end user testing and feedback.

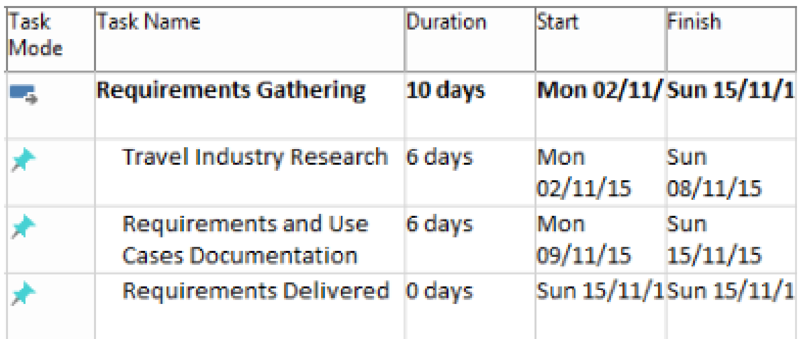
7 Consultation with Academic Staff

Eugene McLaughlin was consulted about this project, specifically about the validity of reusing a project idea from a previous project. Eugene's feedback was that this approach is permissible provided work will not be submitted for grading twice. Given this Tripency will be a complete redevelopment using newer technologies with considerably more functionality, the developer is confident this is the case.

8 Proposed Supervisor

Eugene McLaughlin

3.2 Project Plan



Task Mode	Task Name	Duration	Start	Finish
▬	**Requirements Gathering**	**10 days**	**Mon 02/11/**	**Sun 15/11/1**
📌	Travel Industry Research	6 days	Mon 02/11/15	Sun 08/11/15
📌	Requirements and Use Cases Documentation	6 days	Mon 09/11/15	Sun 15/11/15
📌	Requirements Delivered	0 days	Sun 15/11/1	Sun 15/11/1

3.3 Monthly Journals

October

What I have done:

The core focus of this early stage is to complete the initial documents, specifically the project proposal and the project specification. I have a number of ideas for my project but the idea I am most keen about is a travel web application that automatically generates a tourist's itinerary based on the interests that person has. That was an idea that I used in a previous year's project although I was never fully satisfied with its implementation and I felt there was scope for further development. I discussed the validity of using an idea from a previous project for this final year project. He said it was valid as long as I'm not reusing material that was already submitted for grading. The previous project was built using simple Java servlets and JSPs. My new final year project will be a complete rewrite using Spring and angularjs, so that was deemed valid.

Any problems/risks:

During the summer months, I learned the Spring framework together with the Hibernate ORM library. There is a significant amount of material and complexity in such a framework, and although I have a good working knowledge of it, this project will be the first time I'll have built a full scale application with it. At this early stage, it is almost impossible to predict where the gaps in my knowledge lie, and therefore I plan to tackle each issue as it arises.

Next Steps:

The plan for next month is to write the requirements. I have already spoken to a number of people about the Tripency concept (including a view in the travel industry) so I have a good indication of the requirements needed to capture their expectations of such an application.

November

What I have done:

I have completed the requirements document which allows for the main Tripency web application and a small supporting android app that allows users to view their itineraries offline. I have also spent a lot of time putting together a skeleton design of the application from both a theoretical and practical viewpoint. I have already drafted a brief overview of the main classes in the project, how they will interact and their relationship to the database tables. From the practical viewpoint, I have created a simple CRUD application (create, retrieve, update, delete) using Spring and Hibernate. This is my main “smoke test”, i.e. ensuring I can use my chosen technologies to connect to a database and display that information to the user via a web page. The web page, for the moment, is JSP because I have not yet spent much time learning angularjs. I suspect the demo that will be performed during the mid-point presentation will be using JSPs as the view technology, and at some point soon thereafter, I will migrate to angularjs. I need to learn it first though!

Any problems/risks:

I suppose my desire to use angularjs in the project despite knowing very little about it is a risk at the moment. I deliberately learned Spring during the summer months because I knew it would take a long time to get proficient with it. Learning angularjs while in college with submissions for both this and other modules is an extra challenge. However, I am not taking the easy route by using JSPs. Even though I am very familiar with that technology and perfectly functional, it is an old technology. I want to challenge myself to learn a newer and more modern view technology and I am sticking with my desire to learn angularjs for this project.

Next Steps:

Learn angularjs!!...and prepare the design document.

December

What I have done:

I have completed the design document as it currently stands. It will be sufficient for the mid-point report although I suspect it will be updated significantly by the time of final submission. Also, I worked through a course about angularjs on the online IT training website PluralSight. I have come to grips with the core concepts of angularjs and created as small test application to experiment with it (separate from Tripency). The framework is actually much more extensive than I had initially thought. Luckily, only a small handful of the angularjs components are needed to get a site up and running (specially the routeProvider and controller) and the rest are “optional extras”. We’ll see as the project progresses what additional modules I will need.

Any problems/risks:

This month has the Christmas/New Year season and next month have the end of semester exams, so I probably will be spending less time than usual on this project.

Next Steps:

Once my exams in January are out of the way, the next step is to get a working prototype ready for the midpoint presentation, preferably with the itinerary generation working to some extent.

January

What I have done:

I have completed a working prototype that I can use during the midpoint demo. As I suspected, the prototype’s view technology is JSP because I am not completely confident I can migrate to angularjs and get it working in time for the presentation. For the moment, JSPs are safe and that is my main concern for the presentation – the last thing I want is for an unknown error to occur in the middle of the demo.

Any problems/risks:

The main risk at the moment actually comes from another module I am doing this semester called Distributed Systems. That module involves a rather heavy CA workload including three separate project (two of which actually overlap). This schedule together with the fact that the module uses a fairly difficult (and let's be honest, outdated) technology called CORBA only adds to the problem. I suspect a lot of time will be spent on that module over the coming weeks which will naturally reduce the time available for this project.

Next Steps:

The midpoint presentation is next week, so that is obviously next on my to-do list. After that, I really need to focus on converting the view technology from JSP to angularjs because I can't have JSPs bleeding into the final few months of the project.

February

What I have done:

The midpoint presentation went well. I was the first to present on that morning so there was a little confusion about double-booked rooms, but aside from that, I feel everything went smoothly. The application ran successfully and generated an itinerary based on user input. The questions and feedback provided by the faculty didn't present any significant worries either. Actually, the only real annoyance was that my laptop (which isn't my primary computer and is fairly low spec) started running an autovirus scan just as I started the demo of the application. As the server and database are currently hosted locally on that machines, the response time of the application was much slower than I would have liked. Thankfully, it didn't adversely affect the demo and I am aware that it is a system resource issue with that laptop rather than a performance issue with the application itself.

After the midpoint, I finally found time to get rid of the JSPs and use angularjs. There were a view initial problems that necessitated examining the document and browsing stackoverflow for similar issues, but I finally got it working. It is good to see Tripency now behaving as a single page application. RIP JSPs!

Any problems/risks:

As I suspected, the CAs for the Distributed Systems module are very time consuming. I predict a significant drop in productivity during next month for this project because I'll need to focus on the deliverables for Distributed Systems.

Next Steps:

Simply, to work through the requirements and develop as many as possible while attempting to work around the Distributed Systems CAs.

March

What I have done:

I have started using a UI library called Travelo for Tripency. Travelo is a mostly CSS library built on top of Bootstrap that is developed specifically for web application within the travel industry. As with any new library, there was an initial learning curve but Travelo has decent documentation so it wasn't too much hassle. Of all the aspects of developing a web application, front end design is probably my weakest skill. I have fairly good working knowledge of CSS but it is the non-technical visual design that I can struggle with. I want the site to good sleak, modern and element, and my CSS skills are such that I can render any design. The problem is actually creating such a design. Thankfully, the Travelo library has the exact "look and feel" I required. I double checked with my supervisor if I could use such a CSS library, and he said there was no problem as long as I credited it. Although, while redesigning the UI, I decided it was a better user experience to select their preferred and disliked categories on the same page, as opposed to two separate pages.

Any problems/risks:

I want to start doing some development of that app. Although it is a small part of the project, it is still a requirement. However, one of my modules this semester is actually Android development, so I have been waiting until my project for that module is complete before starting the app for Tripency. My logic is that I can then use the skills learned from that project for the app. Also, exams are next month which will also take some time away from the Tripency project.

Next Steps:

As before, work through the requirements and develop as many as possible while attempting to work around the Distributed Systems CAs.

April

What I have done:

Once the exams and the Distributed Systems were out of the way, I went full steam ahead with Tripenecy. Now, at the end of April, I have about 90% of the coding complete. The Android development skills learned in my Android module came in very handy when developing the Tripency add. Consuming a rest API from Android was also surprisingly easy. Also, based on the first round of user testing, I have added a new piece of functionality that consumes a Google Maps directions API so that the distances and directions between each attraction is displayed to the user. This was not a requirment in the original spec but it proved popular with the test users as a desired new features.

Any problems/risks:

My chosen hosting company, Heroku, natively supports the *PostgreSQL* database. Tripency currently uses mySql, and although mySql can be used with Heroku using an add-on, it is strongly recommended by Heroku to use *PostgreSQL.* Thankfully, I am not using any specific mySql functionality and switching database types should be fairly easy based on my research into Hibernate. Therefore to avoid any hosting issues, I feel it is prudent to go with Heroku’s strong recommendation and migrate from mySql to Heroku.

Next Steps:

Perform the *PostgreSQL* migration, upload to Heroku and complete the remaining 10% development.