University of Glasgow | Department of Computing Science

Walk Around The World

Ryan Wells

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 20, 2014

Abstract

For many, cardiovascular exercise is an activity that we do not get enough of [1] . Studies have shown that physical exercise between 1 hour a day for children over five years of age and as little as 2.5 hours a week for adults has a huge positive impact on our physical and emotional wellbeing [2, 3] .

This project aims to encourage users to do oudoor cardiovascular exercise by positive encouragement through a Mobile Application. By tracking the distance the user travels in each exercise session through device GPS and translating this into well-known outdoor pursuits allows the user to quantify their exercise in terms of well known physical and cultural achievements. These achievements will take badge-like form and other metrics of success such as leaderboards will also exist.

Outdoor pursuits will contain: actual routes such as the West Highland Way and the climb to the top of Everest; popular culture references such as "Route 66" and the (approximate) distance that *Frodo* and *Sam* travelled in J.R.R.Tolkein's "The Lord of the Rings"; and Global distances such as the distance between capitol cities and simple metrics such as the first one hundred miles.

The user will exercise with a compatible device on their person. During exercise the device will track and log the distance that the user has travelled and add this to their accumulators. If this device is their smart phone then the Mobile Application will notify them immediately when they have completed an outdoor pursuit, otherwise they will be notified when data is collected from the dedicated hardware.

This application will encourage social exercise through outdoor pursuits that can only be completed through teamwork with other users that have met in real life. This intends to extend the influence of rewarding exercise by encouraging users to include their peer group in exercising with them and sharing their successes. For a user, this is incentivised by unlocking achievements only achievable through this social interaction.

An analysis of user trends such as: frequency and duration of exercise with regards to duration of application use; change in exercise duration when achievements are awarded during exercise; and quantity of social interaction with regards to application use life cycle will be used to indicate whether or not gamification techniques could have a valid application in cardiovascular exercise.

Contents

Chapter 1

Introduction

1.1 Motivation / Context

The number of smartphones being sold per year is increasing [4, 5] and nearly three-quarters of exercisers use technology to support their workouts in some way [6]. This project aims to utilize this mass market acceptance of technology in exercise and combine it with game like elements to begin an evaluation on the effects of games and exercise. Specifically, this project will look at outdoor cardiovascular exercise where the user is required to exercise with their phone with them.

The main problem I aim to address is the boredom that can arise through exercise or the reluctance to do so in the first place.

Consider a frequent exerciser who repeatedly travels the same route, or the same collection of routes when exercising. This individual may easily tire of the journey or bad weather may discourage the individual just enough that they decide to move their exercise to a controlled environment such as a gym where the benefits are not as profound.

Also consider a different individual who has a relatively short distance to travel and who doesn't exercise regularly. This individual could either travel this distance on their own accord or use a service such as a car or take public transport. With no outside encouragement, and low self control, the individual may be likely use the service based travel method and lose out on all benefits of exercise.

Both individuals above would benefit from positive encouragement for exercise. I intend to provide this positive encouragement through bringing game like elements to the exercise routine in such a way that it can cater for heavy and casual users alike.

The project will allow users to travel along a "virtual route" while they traverse the physical route they are exercising on and will gain achievements for completing these "virtual routes".

1.2 Research Questions

The application of game like elements to exercise is nothing new. As discussed in Section 2.2, other applications have taken different approaches to incentivising exercise. To the best of my knowledge no other product uses exactly the same game experience as the one I propose in this dissertation, so the main points for analysis are with regards to the distinctive features of this experience.

This dissertation aims to analyse the main two following points within the scope of the project:

1. How valid is the application of badge-based gamification to the context of outdoor exercise? I intend to gauge this through measuring the distance travelled and length of exercise duration of users and determine if a change occurs.
2. How valid is the platform for this kind of encouragement? With user feedback as a basis, I will propose whether or not mobile phones are a suitable platform based on preferences of users and ease of use.

1.3 Summary of the Contributions

This project required producing a mobile app to provide the gamification and a webserver to manage data provision and result gathering.

The mobile app takes the form of an Android App which is hosted on the Google Play Store [7]. It is recommended for convenience to download the app directly from here, however you can install from source or the generated apk if desired. The version that is hosted on the app store will not change between the time of submission and results being published.

The webserver was developed in a RESTful style using the django middleware framework [8] and tastypie, a django module for generating a REST style API [9]. Some business logic does not fit with the REST methodology, such as registration and exercise session management, so specific views have been implemented to cater for this. These are constructed using solely the django framework.

When in use, the mobile app requires an internet connection to obtain data from the webserver and to communicate location information to the webserver. Since this is an application intended for use when the user is not on a home network, care has been taken to minimise the network traffic required. Content is managed in the app by utilising explicit caching: this retains local copies of data objects that will not change during the runtime of the app.

Care was also taken when designing the interaction for managing an exercise session, with the overall goal of further reducing the network cost. Inspiration was taken from some of the REST principles, but this work flow does not adhere to all of these principles.

These are explained in full in Section **??**.

1.4 The Structure or Outline of the Report

We will start by examining the current state of the market by discussing other implementations of gamification in mobile apps. Through this examination, we will discover the benefits and pitfalls of each implementation and use this to form the basis of our implementation.

We will then define the overall goals of the project: how we will use gamification to deliver our implementation and what we hope to achieve in the scope of this project.

An analysis of our implementation will then be presented. This is based on user testing and evaluation of the overall suitability of the platform and experience.

Chapter 2

Related Work and Background

2.1 Gamification and technology in exercise

Life Fitness in 2012 released a survey concerning technology in the exercise environment [6] and it was found that over half of gym users used a smartphone or tablet when exercising. This study focused solely on gym users which is not our target market for this product (we are targeting users who exercise outdoors), however it is not unreasonable to assume similar statistics for our target demographic. We use this metric as a basis of platform validity: we know people use this platform in our targeted situation and so we are justified to capitalise on this.

2.1.1 Defining gamification in our context

Sridharan, Hrishikesh, & Raj [10]

This may also be relevant, need to get when in uni `http://dl.acm.org/citation.cfm?id=1125805`

2.1.2 Targeted users

The users we are targeting are those who do not exercise often and those who spend some time travelling on foot outdoors where their travel is not intentional exercise.

For users who do not exercise often, we aim to motivate these users to exercise more frequently by setting a clear reward structure that does not punish users for taking time between sessions. We also hope that this encourages these users to travel by foot on journeys that they otherwise may opt to take transport for. The short term rewards will initially help these users the most: since they do not exercise often they may need quicker encouragement to help push them to exercise more regularly.

With users that spend some time travelling on foot without it being intentional exercise we should endeavour to retain the thought that it is not explicit exercise. For these users, the time they spend travelling on foot may be their morning commute to work or walking children to school or nursery - time which is not set aside as exercise time but can still be enhanced by the benefits of exercise. An extension to their commute may be easily achieved, taking the "long route" for example, where both the long and short term rewards will encourage the user to spend more time on foot.

2.2 Comparison of related applications

Incentivising exercise is not a new concept in the mobile market. From simple applications that mimic a pedometer to immersive alternate realities, there has been a varied approach to encouraging exercise. These approaches can be shown through the following applications.

Table 2.1 gives a brief overview of requrements and user statistics for the compared products gained from the Google Play Store.

Name	Interface	Requirements	User base
Charity Miles	Mobile App	App and GPS	10K - 50K
Zombies, Run!	Mobile App	App and GPS, headphones	1M - 5M
WalkJogRun	Mobile App	App and GPS	unknown
Fitbit	Dedicated hardware and Mobile App	Dedicated Hardware	1M - 5M

Table 2.1: Comparison of exercise gamification implementations from the Google Play store

2.2.1 Charity Miles

Charity Miles encourages you to exercise by facilitating you to raise awareness of charities and help others as you exercise. For every mile that you run or walk $0.25 is donated to a charity of your choice, and for every mile you cycle $0.10 is donated. The money currently comes from a pool of $1,000,000 gathered by the parent company - so you as an end user don't have to pay anything.

The money you accrue during an exersize session is called your "sponsorship" and is only passed onto your charity when you share your success to a social network (facebook or twitter) [11]. Charity Miles say this is to increase awareness of the charity, but it is also a good advertising tool for them.

The incentive here is clear - run to donate. In essence, you are both being paid to exercise and exercising to help out a charity. The monetary backing allows you to quantify your exercise in terms you are very familiar with whilst generating a feeling of satisfaction by helping a charity.

The main drawback here is that there is no defined behaviour for what happens when the $1,000,000 pool runs out. From a business perspective, the app could easily be re-purposed for non-centralised pool funding where the end user could find local sponsorship. However this entirely changes the user experience and explicitly requires effort on the users side to gather this sponsorship.

2.2.2 Zombies, Run!

Zombies, Run! is a companion app that encourages you to exercise through audio cues, rich story and a base-building game. As you run you are being "chased" by a hoard of zombies that get closer and further away from you as you exercise - the closer they are to catching you the louder they become.

In the app you have a base from where you defend yourself from the zombie hoard and are required to leave the base to gather supplies. The story narritive tells you what you need to gather and you find these supplies based on the distance you travel and the route you take. Your success is measured by the fortifications of your base, which you can improve by running faster/further than before.

You are required to wear headphones and carry your mobile phone while exercising. The headphones give you audio queues to immerse you in the story, but it is a requirement of the game experience that you listen to the soundscape provided - you cannot listen to your own music. As endearing as groaning zombies may be, it is understandable that users may tire of hearing it. Also, a user may be used to listening to their own music and use that to zone out of the exercise so some users might be reluctant to adopt this.

2.2.3 WalkJogRun

WalkJogRun helps you discover new routes that are geographically close to where you are and record your favourite routes to share with your friends. From your current location it finds and proposes new exercise routes given a certain route length. It can provide turn-by-turn direction help when you are exercising to ensure you do not get lost and will also record an exact route as you run so you can create a new route with the minimum of effort.

Your workout is saved and can be later analysed so you can accurately monitor your progress.

Since the routes are user provided, there is no guarantee that the routes are safely traversable or applicable to all types of exercise - a route that can be walked may not be applicable for a runner due to terrain type, for example.

There is no clear gamified experience in this application, so the incentive is self-provided. The user can only use this to improve their exercise experience as there is not much in the way of encouragement.

2.2.4 Fitbit

I have included a comparison with Fitbit products to display other mediums in which this application can be used. Although it is not mainly based around a mobile phone use it is an interesting comparison.

Fitbit provide a collection of dedicated hardware devices capable of accurately measuring your distance travelled (and in some models the height you have ascended in terms of “flights of stairs”) as a means of tracking your activity. At the core, it is a glorified pedometer with web and phone apps to accompany it and display progress. The devices and apps also make a prediction about the calories you burn each day based upon the distance you travel when wearing the device, your weight and stride length.

Each device can show you the progress toward your daily goal (either steps taken or predicted calorie burn) with various visualisation techniques - the simplest of which is a row of five LED lights that fill up as you near your goal. Synchronization with compatible phones, or through a computer if your device is not compatible, allows a more in depth visualization of your progress.

The gamification aspect is inherent in the design of these products - you set your own goal and you can easily view your progress at a glance. Upon completion of your goal the device vibrates and lights up to notify you. You are also awarded badges for daily completion such as walking 10,000 steps a day and lifetime completion goals such as travelling 100 miles in total.

There is a clear downside to this - you are required to purchase a bespoke device to fully utilize the experience provided by Fitbit. When compared to an app that may be \$1.99 or thereabouts it is a much larger startup cost than the previous examples. However this larger startup cost may be an encouragement unto itself: the user may feel obligated to commit to using the device since they are more financially invested, which may help with the onboarding required to fully benefit from these systems.

Most of these devices are reasonably discreet, the more advanced the devices get the larger they become. Since these devices must be worn mainly on the wrist there are social implications involved. Does the user want everyone to know they are using a fitness tracker? The other applications are more discreet by design, since they are a mobile app they can be hidden easier.

2.3 Consideration of competitors concepts in this project implementation

Each of the above products utilise gamification in an effective way and each implementation differs in how they utilise game elements. This project builds upon some of the design decisions these competitors have used, aiming to remove some of issues that make the competitors products either unsustainable or flawed.

First and foremost, this project should not require any form of additional hardware for the end user or monetary investment. A key demographic that we are targeting are those users who do not exercise outside often, so we do not want to create a barrier between those users and them achieving their exercise goal. This is in reference to the Fitbit product discussed in Section 2.2.4 which requires a dedicated hardware wristband, and the "Zombies, Run!" product (Section 2.2.2) which requires the user to purchase the mobile app. Both of these could be reasons for an under encouraged user to not take the first step to starting the exercise process.

This project should also not require a social aspect to achieve the core goals for this application, unlike WalkJogRun (Section 2.2.3) which has a heavy reliance on social contributions from users in your area. Although WalkJogRun does not provide a gamified experience, it does enhance your exercise experience by showing you new running routes. These routes are user provided and reliance on crowdsourcing for content, gamified or otherwise, is too risky for the short period of this project.

Chapter 3

Design and Specification

3.1 Aim/vision

The idea of the game is to travel around the world without physically having to be there, inviting the user to realise the scale of the world whilst gaining the positive benefits of outdoor exercise. A user may decide to run around areas in Scotland (the Scotland Mission) and this is achieved through routes such as Glasgow to Edinburgh. Each route is further broken up into manageable stages of a few kilometres and users are awarded badges based on stages/routes completed and overall distance and time. Therefore the user can gain short-term success while working towards the satisfaction of larger goals.

This approach is to examine whether or not this type of encouragement could potentially work with outdoor exercise. Metrics will be tracked for individual users to monitor their exercise duration and frequency and will be used to encourage that user to exercise. These metics will then be used to form the basis of our conclusion.

The aims of the project are then as follows:

1. To provide incentives that become intrinsically rewarding. Rewards are primarily linked to stage and route completion and these accumulate to the completion of the entire mission - every achievement brings the user closer to a larger success.
2. To provide long and short term goals to fulfil the sense of achievement. The long term goals are Route completion and these are fulfilled by the easily achievable short term goals of Stages - for example the Central Park Route contains four stages, one for each edge of the park and these stages are easily achievable in an exercise session.
   (a) Stage completion - unlock a photograph of the stage or of a key landmark near the stage. This is achieved by completing a specific stage.
   (b) Route Completion - a gold, silver or bronze medal is awarded for completing a route in a given time period. A route is completed when all the stages in a route are completed.

3.2 Specification

The implementation specifics of this project are defined herein with the overall development of the project shown in the following sections.

An overarching goal for this implementation is to minimise the end cost for the user. This cost takes two forms: response time for data loading and network costs incurred through data loading.

Initial response time for loading data cannot be avoided without prepackaging content inside the app. We do not want the user to be forced to update the app every time we want to add another route to run down so prepackaging will not work. We can however cache the data we know does not change and reference that instead - which is much faster than a network response time.

Caching also reduces the overall network cost: by caching the data received we remove the need to make the request again. This inherently reduces the network cost by reducing the number of network requests.

These are discussed in more detail in Section 3.3 and will be referenced in the following subsections.

3.2.1 Platform and Frameworks

The mobile application is developed using PhoneGap [12], a tool which allows you to develop a mobile app as if it were a web app, and then package it as a mobile app. The decision to use PhoneGap instead of building a native app is primarily because I come from a web development background but also so we can deploy to multiple platforms easily without needing to change the code base.

Because of this, our application is written using HTML, JavaScript, and CSS3. The twitter Bootstrap CSS framework [13] was used because of the excellent support for small screen devices and layout management.

Titanium, developed by appcelerator [14], is an alternative platform to PhoneGap that was also considered. This required a greater knowledge of android than PhoneGap does but the biggest drawback was the state of development when the project started. After discussion with members of the startup community who were using Titanium, it was apparent that Titanium was still a work in progress and probably a risk to go with. The platform is now flourishing and would probably have been the better choice to go with in retrospect, but at the begining of the project the safer choice was to develop for PhoneGap.

AngularJS [15] is the framework used for the application. This framework has a major benefit of explicit data linking between the DOM(Document-Object Model) and the javascript controlling that section of the DOM. In AngularJS, a controller is defined to manage the behaviour of a specific section of a page. By defining these controllers, we can utilise this explicit data linking to manage how information is displayed in the DOM - if a variable changes in a controller then it also updates in the DOM. This linking is bi-directional, so the reverse is also true.

The AngularJS framework also facilitates many other control mechanisms: DOM tree manipulations, CSS class manipulations, frontend routing (a process where a section of the DOM tree and an associated JavaScript controller are swapped out based on the current URL) and wrappers around the browsers native AJAX implementation. All of this functionality make AngularJS the ideal framework for the project.

The web application is written in Python and uses Django [8] and Django-Tastypie [9]. Django is used as a middleware platform for interfacing with the database and creation of specific workflow interactions; while Django-Tastypie is used for managing and building a REST style API for well-defined database object access.

Django exposes an Object-Relation Mapping (ORM) inside the framework for manipulating and creating database objects. This is utilised by Django-Tastpie for creation of the REST API. We use a REST style API so that the mobile app is able to uniquely identify an object in the database in a consistent and reliable manner. This is used in session management in Section 3.3.3.

Moreover, there is a large number of database objects that we require knowledge about in the app. A tool which creates a well defined way of accessing these objects and managing their access permissions is invaluable for consistency and reliability.

No session-like state is required in this application as the database object relations are defined in such a way that they hold all information required.

3.2.2 N-Tier Diagram

The high level components of this system are reasonably simple (figure 3.1). The user requires a mobile phone with GPS enabled, an active internet connection and the mobile application installed. This communicates with a Django Webserver with a REST-style API (with a small number of bespoke views since some behaviour does not fit well with the REST specification) which in turn uses Django's Object Relation Model (ORM) to persist these to a database.

The mobile application communicates with the devices GPS location management system to obtain location information for the user. We request that the users devices provides us with the most accurate GPS location it can provide, so it will try satellite positioning first and then try network provided information.

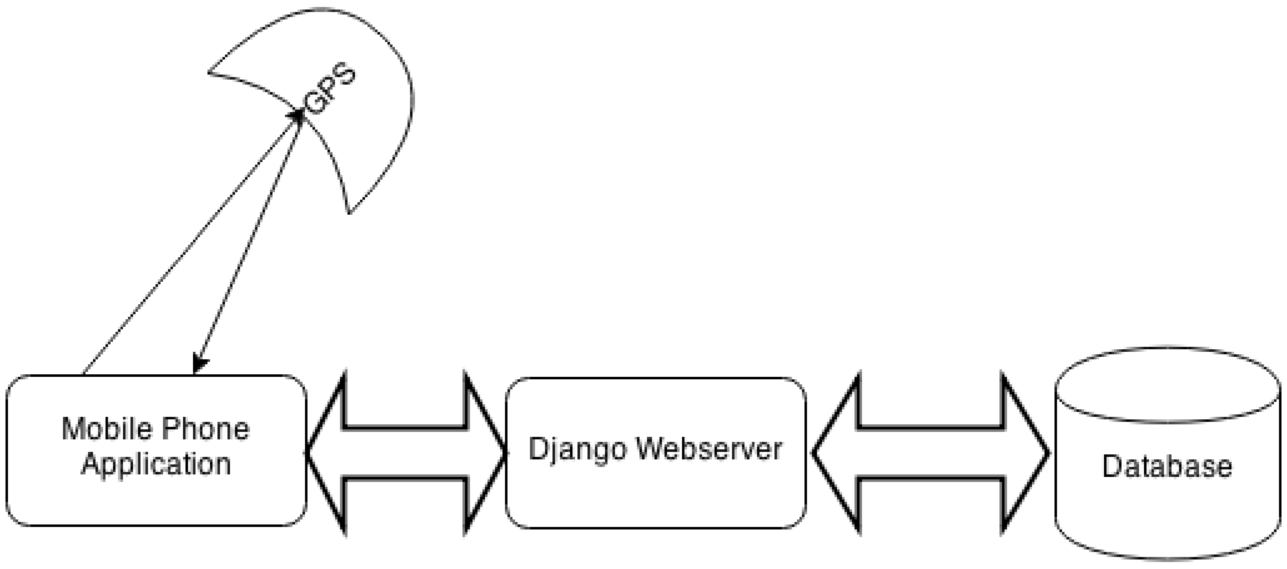

Figure 3.1: N-Tier diagram

3.2.3 Entity Relation Diagram

There have been two significant iterations of the Entity Relation (ER) diagram in this project - the first (figure 3.2) is the final theoretical ER model and the second is the real world implementation with modifications and extra linking for data transfer optimisation and changing requirements.

The main reason for the implementation modifications is to keep data transfer low between the mobile device and the API. Since the mobile device will be using non-wifi based communication methods the number of bytes transferred will be logged by their network provider and will cost the end user. Since we care for the monetary cost that using the app may incur to the end user (Section 2.3), we can sacrifice having a non-redundant database in order to optimise the data transfer for the end user.

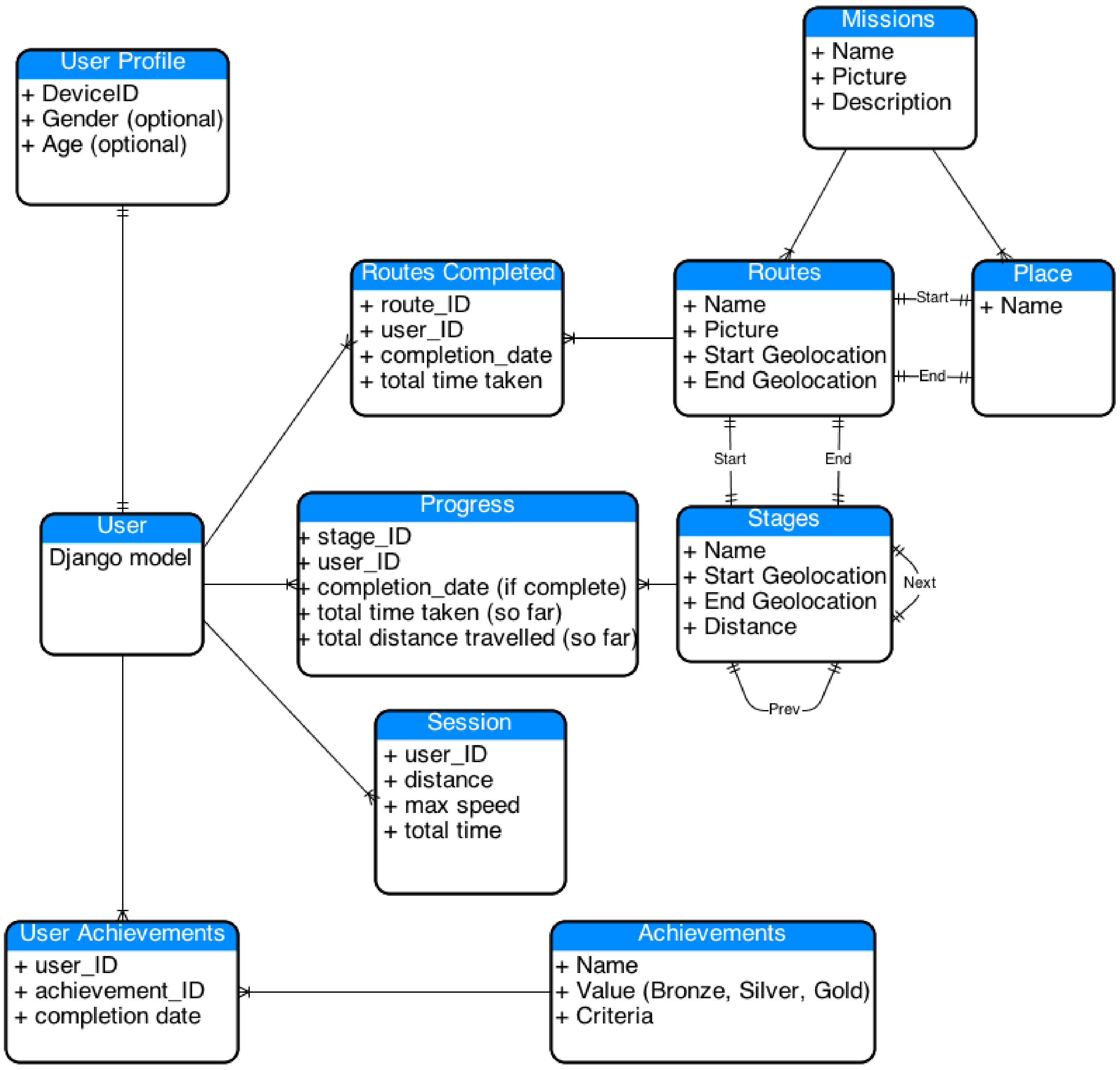

Figure 3.2: ER diagram, final theoretical model

Do implementation ER diagram

3.2.4 Walkthrough of Wireframes

THIS NEEDS REDONE WITH NEW WIREFRAMES

When the app is launched it silently registers with the server allowing the user to use the app immediately. The user is then shown the middle-top screen.

1. From the middle top screen, the user can follow arrow 1 by clicking on the middle button “Pick Mission” to pick a Mission (Run around Arran or Egg for example) and then pick a start and end location. After confirming these choices, the user is taken back to the middle top screen, or at any time can click the “Home” button to return.

2. The user can also view their current acheivements by clicking the “Achievements” button on the middle top screen, following arrow 2. These achievements will be grouped by tabs by category - Distance, Time, Stage and Mission based achievements.

3. The user can follow arrow 3 from the middle top screen to notify the app that they are starting an exercise period, telling the app to track their distance. If a Mission and start and end location are not picked (as in point 1) then they will instead be redirected to this screen and are unable to start exercising until this choice has been made. Once they have successfully advanced to this screen, it will display their current progress as they move showing the user how close to completion of their current stage and overall route they are.

4. When the user has finished exercising, they will click the “End Session” button and be taken to the first summary screen - following arrow 4. Here statistics from their exercise will be shown and the option to share this on several social media outlets.

5. The user can then move to the second and final summary screen, following arrow 5, where they will be shown any achievements they were awarded during that session. The user will also have the option to share these on social media outlets. From here, the user can click the “Home” button and be taken back to the middle top screen.

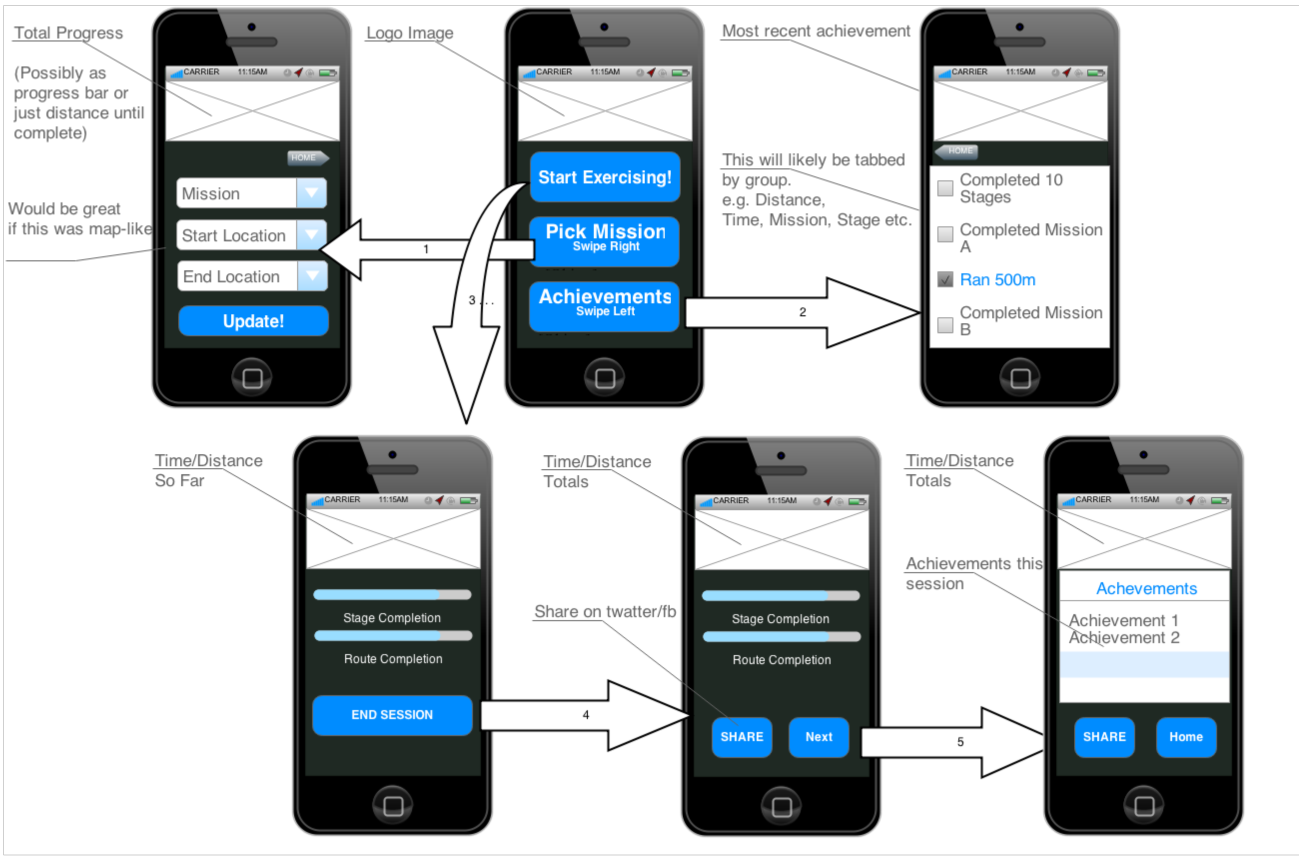

Figure 3.3: Wireframes, initial design

3.3 Notable design decisions

3.3.1 Explicit caching

As was mentioned in Section 3.2, the minimisation of network traffic is of great consideration when dealing with mobile devices. We did not want the mobile device to make repeated API calls for resources that will not change in the short term - resources such as what missions are available, what routes belong to what missions and what stages belong to each route.

The main case for this is as follows. A user may open the app for the sole purpose of browsing all missions and their routes, and any progress they have on these routes before they start using the app for exercise. In an unmanaged situation the app could call the API each time it lands on a menu to pick a mission or the routes for a mission - if the user is browsing then they may see the same information provided several times.

If we remove data transfer from the equation all together and focus solely on the user experience then caching will give a better user experience in terms of loading speed. Since caching means we do not have to hit the API each time we want to see something we have seen before, we can load selection menus faster.

Note that only data endpoints that are unlikely to change in the short term or those that cannot feasibly change under certain circumstances are cached. The "progress" endpoint which provides the data about a users progress along given stages will not change if the user does not initiate an exercise session - the user has not made any more progress so it could not have changed.

These caching services are used as singleton-like facilities in the application to ensure consistent data throughout the application, an update to this data will be reflected throughout the entire application.

3.3.2 Promise Based Module Interfaces

Each module that has been created for managing either a specific set of objects(routes, missions or the user object) has been defined with a common interface. This common interface brings a regular and well defined way of interacting with these modules, especially if they utilise explicit caching (section 3.3.1) or execute asynchronous methods.

There are two ways that this consistent interface could be implemented: a function callback explicitly provided as an argument to the module function call; or by using promises.

Function callbacks passed as parameters are a common JavaScript paradigm. They are used in the PhoneGap framework to interface with the device [16]. A frequent problem with function callbacks is what is colloquially known as "callback hell" [17] where nested anonymous functions are used to bring an order of execution to asynchronous tasks. This can become unmanageable and hard to maintain in larger projects but can be overcome with good software development procedures. An example of how the callback approach works is show in Figure 3.4.

The biggest problem for us is the framework we are using - AngularJS only knows about events that happen in its own scope. This means that when we invoke the callback function with the data from a module, the data will be updated in the controller but AngularJS does not know to update any reflections of this data in the DOM or elsewhere. There are ways to notify the framework that it should run an update (known as a digest cycle) [18] but if we invoke this when a digest cycle is currently in progress then the framework throws an error and drops the request.

```
function async(callback){
  // Do something with the args asynchronously.
  // Then explicitly call the callback function.
  callback(data);
}

function callback(data){
  // do something with the data returned from the asynchronous task.
  console.log(data);
}

// Invoke the asynchronous function and process the returned data.
async(callback);
```

Figure 3.4: Callback approach to asynchronicity

Another option is to use the AngularJS wrapping for the standard "setTimeout" function with a delay of zero. The "setTimeout" function executes a callback function after a certain time period [19]. While this may seem frivolous, a subtlety of this wrapping avoids the digest cycle clash of the above problem. The AngularJS wrapper "$timeout" will invoke the function passed to it after the delay expires, but importantly will cause a natural digest cycle to occur. So if this is called using a delay of zero then we make AngularJS update without risk of the framework erroring. This is the method that we use inside the "core" set of modules that wrap the PhoneGap device API methods [20] as they use callbacks as their own notification method. Wrapping using "$timeout" can be shown in figure 3.5 (note this is an example that is stripped of the necessary declarations to work in the framework, it is intended as an example only).

```
\\ PhoneGap function to get the users location,
\\ success and failure are two callback functions.
\\ Note the similarity to the previous example.
geolocation.getCurrentPosition(callback)

\\ This is defined in our controller or other module.
function success(data){
  \\ Call the $timeout function with delay of zero to safely invoke
  \\ on next digest.
  $timeout(function(){
    \\ do something with the location data
    \\ AngularJS knows about this update.
  }, 0);
}

\\ Invoke request for location information.
geolocation.getCurrentPosition(success);
```

Figure 3.5: Using "$timeout" to wrap PhoneGap API

Internally within modules the "$timeout" method is used for notifying the framework of necessary change as PhoneGap uses callbacks as its own notification method. Callbacks could also be used for inter-module communication but would require each callback function to use the "$timeout" method if any asynchronicity occurs. Since we want to develop a consistent interface between all modules this is not appropriate: we need to know when we should and should not require a call to "$timeout" when we write the callback function, and this can be different between modules.

We can remove all these complexities by using promises. A promise allows us to manage deferred (asynchronous) callbacks and synchronous callbacks in the same way and the AngularJS framework has promises integrated in with the digest cycle management.

Promises and callbacks are theoretically similar as both provide a function to be called once something has occurred however promises do not suffer from the “callback hell” anti-pattern that callbacks alone do due to a flow control change. Kris Kowal, the creater of the promise library “Q” that AngularJS bases its promise library “$q” on, defines the flow control change as follows:

> The callback approach is called an “inversion of control”. A function that accepts a callback instead of a return value is saying, “Dont call me, Ill call you.”. Promises un-invert the inversion, cleanly separating the input arguments from control flow arguments. [21]

A function that uses promises to notify success or failure immediately returns a promise object. This is different to the callback approach as the program is allowed to continue to execute as the function immediately returns, unlike the callback approach which blocks for the response, and the function to handle the response is invoked when a response is ready. These response handlers are attached to the promise object by calling the “then” method of the promise, and these are called when the promise resolves. This can be shown in Figure 3.6.

```
// This could be in another module, but doesn't need to be.
function promiseFn(){
  \\ Create a new promise
  var deferred = $q.defer();
  var data = {}; \\ Data from (a)synchronous task stored in here.

  \\ Something asynchronous or sychonous here. Once it's completed...
  \\ Resolve the promise
  deferred.resolve(data);

  \\ return the promise
  return deferred.promise;
}

function processData(data){ ... //Do something with data }

// Call function and process the data it returns.
promiseFn()
  .then(processData);
```

Figure 3.6: Using Promises in AngularJS

Although promises are intended for use when there is asynchronous behaviour, they can also be used in synchronous situations. The exact user-perceived behaviour would also occur if the asynchronous task was made synchronous in figure 3.6. The functions passed to the “then” method of a promise will be called when the promise is in a state of being resolved, so they can be added before or after the promise is resolved and they will be called in both cases.

This then provides us a method that allows a consistent interface that integrates well with AngularJS. It is worth noting that the “$timeout” method is used with modules that interface with any of the PhoneGap device

API methods that are asynchronous. As is discussed previously, this is to notify AngularJS that a promise has changed state in a situation that it does not know about. So although promises solve most of our problems, they do not completely solve all of them.

3.3.3 Exercise Session Management

Due to the environmental implications brought on by GPS, we may not be able to identify their location. The user should not be penalised for this as it is outwith their control and is not a direct error on their part. Something as simple as running through a tunnel could initiate the loss of signal.

To counter this, the exercise session is designed in such a way as to not penalise or disadvantage the user if they should not be able to obtain location information.

When a user decides to exercise, a new session is created through the API. The users device is then given a unique ID for that session that only they have a knowledge of (in the first iteration of this implementation this ID is a direct mapping to the ID of the object in the database, but in a future version this could be something more obscure). Only this user can update this session and it can only be updated if you know the unique ID for that session.

This ID is saved on the users device for the duration that the user wants the session to be active and is deleted when the user either closes the app or explicitly ends the session. We can then update the session sucessfully when we have a GPS location and are ready to update, while indirectly “ending” the session when we delete the unique ID.

If the user cannot obtain location information then the device will retain the unique ID for the session and will update as soon as it can obtain location information. This is explored further in section 3.3.4.

This does guarantee that the user will gain all distance they should be awared. If the user starts an exercise session but cannot then obtain location information and either exits the app or ends the exercise session then the user will not be credited for the distance they travel. This is a situation we cannot control and so we can accept that this is a limitation of the implementation.

When a session is created we need to tell the server what route we intend to travel down and where we currently are (in terms of our longitude and latitude). To update a session, we only need to provide a timestamp, longitude and latitude positioning and the session ID. On receipt of a successful update the server will update what stage a user is on, automatically moving the user onto the next stage on a route when they complete their current stage.

Only the last known location (set of longitude and latitude coordinates) and the time that the user was at those coordinates are kept in the server. We only need the last location to work out distance a user has travelled since the last update.

We have two options when a user completes a stage: we can either accumulate the excess distance they travel and allow the user to “spend” this distance on whatever route they like when they finish; or automatically advance the user onto the next sensible route.

The latter option was chosen for this implementation as it is the most immersive option of the two. From a game perspective, the first option does not allow the user to fully engage in the game at the time they are playing as no further rewards are accumulated when distance is not being incremented while the second allows the user to continue progressing seamlessly.

3.3.4 Location Management

Phonegap exposes two main ways to obtain location information, either by requesting where the device is at this exact moment in time [16] or by requesting to be updated at a discreet time interval when the device position updates [22].

The original intention was to use the first method to manually control the time between requesting new location information so that if a request fails we can immediately send a request again. This method was dropped after closer inspection of the plugin that provides these services and research into how the Android Location Manager handles location data.

The plugin that provides the location data links directly to the Android Location Manager as expected. This location manager handles all requests for location data from all applications running on the device. The second method of obtaining location information registers a watcher with this Location Manager and allows the Location Manager to handle the polling for coordinates. If location information cannot be obtained then we rely on the Location Manager to deal with retrying.

As much as we may wish to immediately retry searching for coordinates with the intention of keeping accurate distance information for a user, this may have negative repercussions. Obtaining accurate location information is a costly operation requiring 50mA for each set of coordinates, more when there is no recent location information [23]. It is the fourth most power intensive operation an app developer can invoke, as shown in figure 3.7 [24].

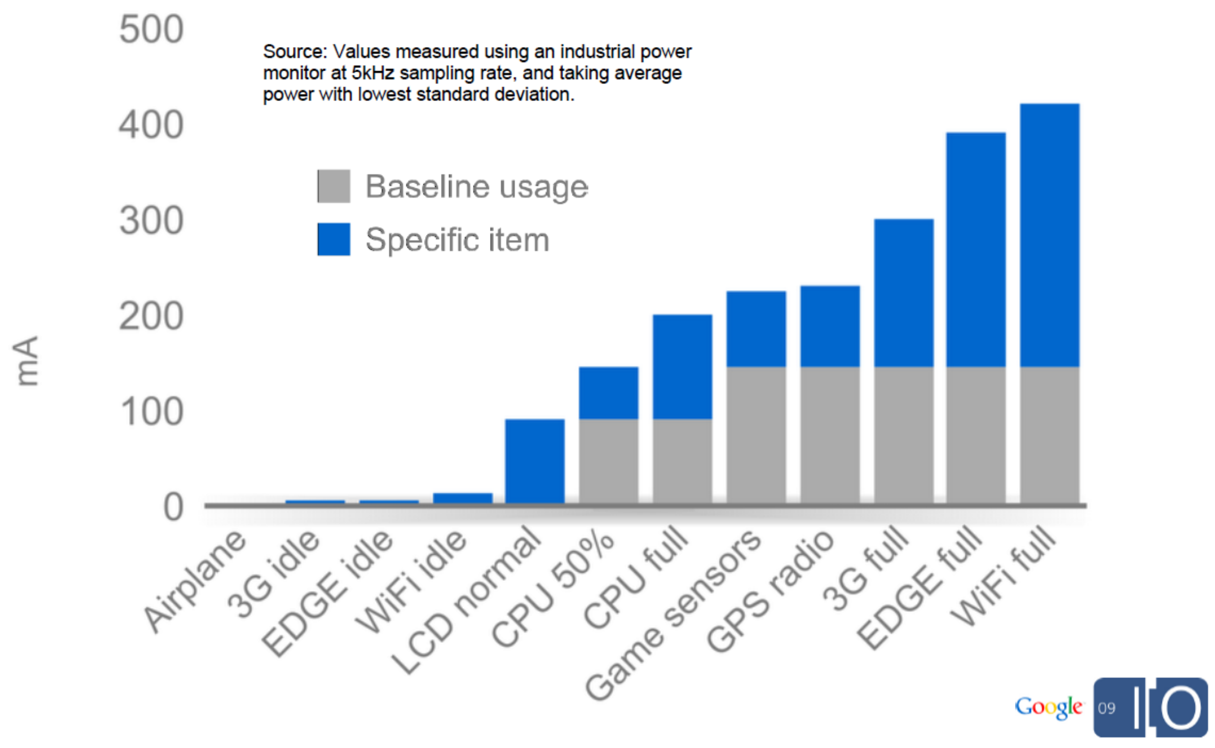

Figure 3.7: Power Consumption of sensors on Android Devices [24]

In the situation where we will not be able to obtain location information, when a user enters a tunnel for example, it would be unnecessary to repeatedly ask for location information. The only outcome in this scenario is a faster drain on the users battery which could lead users to have a negative experience with the app. So even though we are leading with the best of intentions it is not the right decision to make. Due to the round trip time required to notify the server of new location information and the relatively short distances users can travel, it was decided that the provided interval of one update a minute was sufficient to ensure a balance between power and accuracy.

Polling for location information once a minute does suffer from one drawback we will define as the "Cul-de-sac Effect". The Cul-de-sac Effect arises between two location updates where the user travels a significant distance between these two updates but these location updates occur close to one another, resulting in a much smaller perceived distance. This effect arises when we reduce update frequency to increase power conservation and can be shown in figure 3.8 where we assume all location updates (noted as "GPS Location") occur consistently at a discreet time interval.

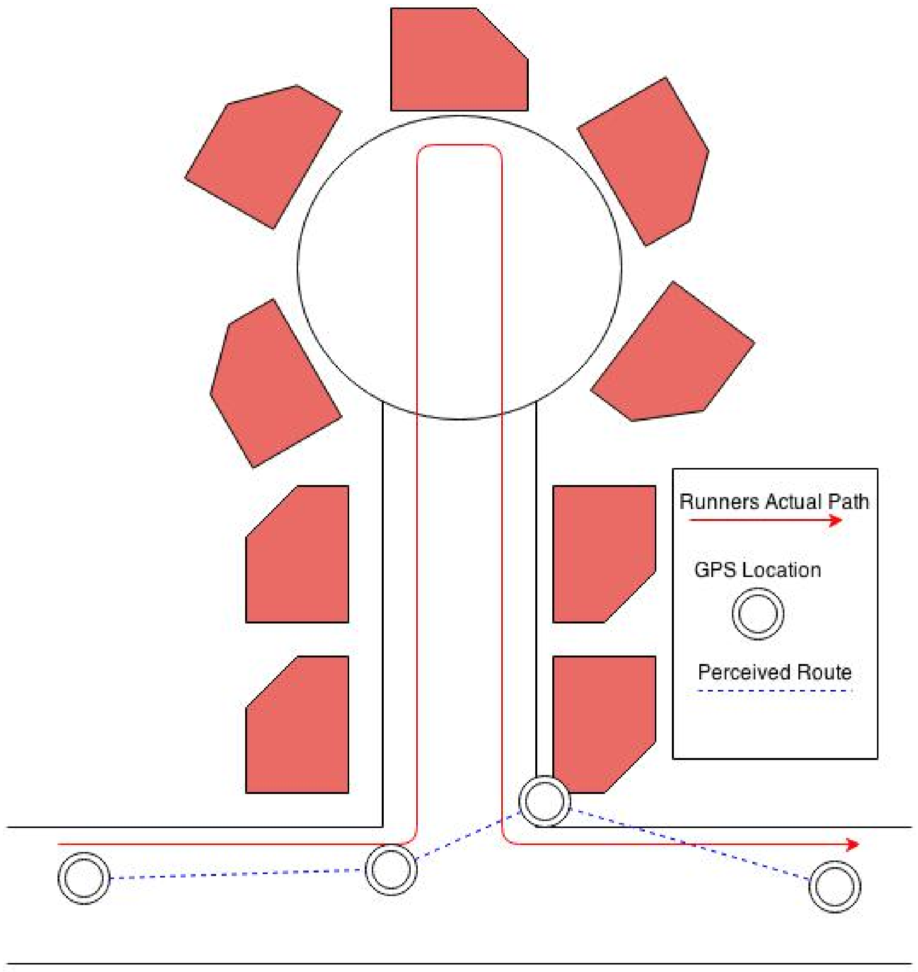

Figure 3.8: Example of the "Cul-de-sac Effect"

In the contrived example shown in figure 3.8 we have no evidence that the user has travelled into the cul-de-sac, as we base our distance calculations solely from the location information, and so we cannot award them the distance they have actually travelled. Therefore the user is unnecessarily penalised through the unfortunate circumstances of timing and their route choice.

As is noted in Section 3.3.5, the maximum speed we allow users to travel at is $6m/s$. In the worst case of this effect the users device will return location information that informs us the user has moved no distance (the coordinates are exactly the same). If we assume that we receive the location updates exactly once a minute then the worst case distance that the user could lose due to this effect is $360m$. This is a considerable distance and the unacknowledged effort could have a large negative effect on the user. Futher work is required to analyse this effect to find the best update frequency interval with respect to this effect and power consumption.

3.3.5 Distance Verification

To create an exercise session a set of longitude and latitude coordinates are required. By enforcing this requirement we are always able to know where a user was last and then work out the distance they have travelled when new coordinates are provided.

Converting two pairs of longitude and latitude coordinates to a distance in kilometers is done through the Haversine Function [25] with a python module of the same name [26].

We send the coordinates to the server and compute the distance there instead of on the mobile device in the interests of verifying that the distance is correct. If a user only sends a distance and not a set of coordinates then we have no mechanism of determining if that distance is valid or not, whereas if we are given coordinates then we have a reference point to compare to (since we retain the last known coordinates). It can be argued that even this is not enough to fully guarantee that the location information provided is honest but there is little we can do to ensure honesty in this case.

We can however limit the distance we update by based upon whether or not that distance is physically possible. A timestamp is required when updating with a set of coordinates and so we can compute the speed the user was travelling at over the time period since their last update. If this speed is greater than a fixed maximum travel speed then action is taken to reduce the distance to the maximum distance the user could have travelled in this interval. This process is shown in Figure 3.9.

$$distance = haversine(coordinates_{old}, coordinates_{new}) \tag{3.1}$$

$$time = timestamp_{new} - timestamp_{old} \tag{3.2}$$

$$speed_{actual} = \frac{distance}{time} \tag{3.3}$$

$$distance = \left\{ \begin{array}{ll} distance & \text{if } speed_{actual} \leq speed_{max} \\ speed_{max} \times time & \text{if } speed_{actual} > speed_{max} \end{array} \right\} \tag{3.4}$$

Figure 3.9: Limiting the distance a user can travel based on a maximum speed.

The maximum speed we allow users to travel at is $6m/s$ as this accommodates for sprinting and will facilitate cyclists, should they wish to use this app.

This action to limit the distance travelled based on the time period will help to defend against cheating but will not negatively impact users who cannot obtain location information for an arbitrary time period. A user could still upload bogus coordinates to the server but this limiting will help to reduce the effectiveness of their cheating.

Chapter 4

Results and Analysis

4.1 Evaluation of Interface and Workflow

4.1.1 Initial Design

4.1.2 Final Design

4.2 Evaluation of Gamification

4.3 Evaluation of Platform

Phonegap has its merits as a mobile platform: you can achieve rapid development cycles by taking advantage of strong JavaScript and CSS frameworks, a "write once, deploy anywhere" strategy for multiple platform development, and good integration with the phones native functionality. However it is the details of the integration with the native functionality that can leave wanting, especially for geolocation information.

As is discussed in Section 3.3.4, we use the PhoneGap provided location watch service to receive location information. It was discovered during development that if the PhoneGap provided location watch service could not obtain location information for any reason then the request would fail silently. Geolocation information cannot always be obtained due to factors outwith our control and so failing silently does not allow the developer to enforce any form of persistence when searching for location information. In the case that we could not obtain location information it may be desirable to immediately restart searching for location information to provide the user with better quality of service, however other factors such as power consumption then come in to effect which may be more detrimental than a few missed waypoints. This would need to be further analysed in future work to properly justify what the correct course of behaviour is in this situation.

The biggest challenge with the PhoneGap platform was the inconsistency within the documentation of the platform and also the instability of the upgrade process.

The PhoneGap platform is built of two main modules: the PhoneGap module which is concerned with the developers interface to the phone, and the Cordova module which is concerned with the device management. There is disparity between platform installation documentation and the plugin documentation (for the geolocation module etc) [27–29]. This inconsistency does not fill the developer with confidence when the official recommendations cannot agree on correct procedures.

Considerable time was also lost during the first round of user evaluation due to an undocumented step required during platform updating. When an Android project is created a JAR is installed that is responsible for interfacing with the device and updating the PhoneGap project did not update this JAR. Since the JAR was not updated it did not have new functionality that the project was attempting to access and so was crashing the app. A solution was eventually discovered on an unrelated thread [30] and user evaluation was able to continue. Time was unnecessarily lost over bad documentation and wrong assumptions by the platform developers.

Other small issues like a provided build script to build a "release" version of the app for the Android Play Store builds a "debug" version instead, each time you build the app the version number and release number are reset to their default values, an XML configuration file for the app is ignored when searching for the main "index.html" file (root of the application), amongst others.

It is for these reasons that I could not recommend PhoneGap as a platform that is viable for a production quality app. Until the documentation and platform become more reliable and consistent the benefits of cross platform development do not outweight the loss of fidelity incurred by non native implementation.

Chapter 5

Discussion and Conclusion

5.1 Acknowledgements

I would like to thank ...

Bibliography

[1] National Health Service. Health survey for england. https://catalogue.ic.nhs.uk/publications/public-health/surveys/heal-surv-cvd-risk-obes-ad-ch-eng-2006/heal-surv-cvd-risk-obes-ad-ch-eng-2006-re pdf, 2006.

[2] UK Department of Health. Start active, stay active. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/216370/dh_128210.pdf, 2011.

[3] US Department of Health. Webstore for gps exercise devices. http://www.health.gov/paguidelines/midcourse/pag-mid-course-report-final.pdf, 2012.

[4] Gartner. Gartner says smartphone sales grew 46.5 percent in second quarter of 2013 and exceeded feature phone sales for first time. http://www.gartner.com/newsroom/id/2573415.

[5] The Guardian. Smartphone sales to hit 1bn a year for first time in 2013. http://www.theguardian.com/business/2013/jan/06/smartphone-sales-1bn-2013.

[6] Life Fitness. 2012 fitness and technology survey. http://www.lifefitness.co.uk/Survey.

[7] Ryan Wells. Google play store entry for ”urban explorer”, the app created for this project. https://play.google.com/store/apps/details?id=com.phonegap.urbanexplorer.

[8] Django Software Foundation. django web framework. https://www.djangoproject.com/.

[9] Daniel Lindsley. django-tastypie library. https://github.com/toastdriven/django-tastypie.

[10] Lena Sunali Raj Mithun Sridharan, Aditya Hrishikesh. An academic analysis of gamification. *UX magazine*, 2008.

[11] Charity Miles. Charity miles terms and conditions. http://www.charitymiles.org/terms.html. Section 1 ”How it works”.

[12] Adobe Systems Inc. Phonegap. http://phonegap.com.

[13] Twitter. Twitter bootstrap css3 framework. http://getbootstrap.com.

[14] appcelerator. apcelerator homepage for titanium alloy. http://www.appcelerator.com.

[15] Google. Angularjs. http://angularjs.com.

[16] Adobe Systems Inc. Phonegap “getcurrentposition” definition. http://docs.phonegap.com/en/3.0.0/cordova_geolocation_geolocation.md.html#geolocation.getCurrentPosition.

[17] Max Ogden. Callback hell. http://callbackhell.com/.

[18] Google. Angularjs $scope.$apply function definition. `http://docs.angularjs.org/api/ng/type/$rootScope.Scope#$apply`.

[19] Mozilla Foundation. Documentation for window.settimeout javascript function. `https://developer.mozilla.org/en/docs/Web/API/window.setTimeout`.

[20] Ryan Wells. Wrapping phonegap functions for the angularjs framework. `https://github.com/ryaanwells/urbanexplorer/blob/master/urbanexplorer_app/www/js/core/location.js`.

[21] Kris Kowal. Javascript promise library: q. `https://github.com/kriskowal/q`.

[22] Adobe Systems Inc. Phonegap “watchposition” definition. `http://docs.phonegap.com/en/3.0.0/cordova_geolocation_geolocation.md.html#geolocation.watchPosition`.

[23] Android Developers Documentation. Power profiles for android. `http://source.android.com/devices/tech/power.html`.

[24] Jeff Sharkey. Coding for life–battery life, that is. `http://dl.google.com/io/2009/pres/W_0300_CodingforLife-BatteryLifeThatIs.pdf`.

[25] Wikipedia. Haversine function. `http://en.wikipedia.org/wiki/Haversine_formula`.

[26] Balthazar Rouberol. Pypi entry for the haversine module. `https://pypi.python.org/pypi/haversine`.

[27] Adobe Systems Inc. Phonegap installation instructions. `http://phonegap.com/install/`.

[28] Adobe Systems Inc. Phonegap documentation for using the command line interface. `http://docs.phonegap.com/en/3.0.0/guide_cli_index.md.html#The%20Command-line%20Interface_add_features`.

[29] Adobe Systems Inc. Phonegap documentation for installing geolocation plugin. `http://docs.phonegap.com/en/3.0.0/cordova_geolocation_geolocation.md.html#Geolocation_accessing_the_feature`.

[30] David Rabinowitz. Plugin fix for upgrade between phonegap versions. `https://github.com/phonegap-build/PushPlugin/issues/92`.