PetPal – Reuniting Pets with their Owners

Brian Murphy, x12533897

BSc in Networking & Mobile Application Development

Technical Report

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name: Brian Murphy
Student ID: x12533897
Supervisor: Sam Cogan

SECTION 2 Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read the College statement on plagiarism (summarised overleaf and printed in full in the Student Handbook) and that the work I have submitted for assessment is entirely my own work.

Signature: ____ Date: ____________

NB. If it is suspected that your assignment contains the work of others falsely represented as your own, it will be referred to the College's Disciplinary Committee. Should the Committee be satisfied that plagiarism has occurred this is likely to lead to your failing the module and possibly to your being suspended or expelled from college.

Table of Contents

1.1 Executive Summary

The objective of this project is to build a Hybrid application that is ported to Android with the option to port with many other platforms such as iOS, Windows and other. Due to research it was most feasible to build the application with real time capabilities to allow for Pet Owners and people that find their pet to communicate instantaneously through a simplified reactive user interface. The application provides Pet Owners with the ability to create a profile based off their pet's details. This Meteor.js application gives Pet Owners and Founders of pets the ability to communicate through a real time messaging service on any smartphone, desktop or tablet. The project model is built to allow Pet Owners manage their profiles through the Android build, whilst the Web Application allowing the Founder of a pet to view the public profiles and begin interacting with the system to start the process of re-uniting the pet with the owner.

2.1 Introduction

The background to this idea is that as a pet owner, I became aware of the risk of no additional security measures being taken when owning a dog. If a dog was to go missing the only possibility of the dog being returned is for the person who found the dog to bring it to a vet and get the microchip scanned. The founder of a dog may not have a vet nearby, furthermore the vet could be closed and the founder may not have a mode of transport.

It is a worrying time when the situation arises when you become aware your pet has gone missing. The only security measure taking by law now is to have your dog microchipped which holds the owners contact details. Sometimes people forget to update their details such as phone number, address which leads to the dog been sent to the DSPCA dog shelter.

50% of dogs go missing a year which leads to the reasoning for this new additional measure to be introduced alongside microchipping. This additional measure can be potentially quicker and more efficient than microchipping as it does not require a vet to scan a chip or update contact details.

(ISPCA, 2016)

The staggering statistic above demonstrates that some people still do not microchip their dogs still even with the new law introduced this year, with an application built of inspiration from microchipping, this can allow for a fun, trendy and modern way to manage dogs details through a public profile accessible by people that may find a missing dog.

The reasons above discussion great inspiration to build an application that is built on top of the idea of microchipping in a more efficient and future proof way. Giving the user the option to choose their type of communication to the owner of a pet was huge factor for the reasoning of creating this application. By communicating with professional dog trainers, their feedback was that this idea could potentially be a success with owners of a variety of different animals with the goal of re-uniting pets with their owners.

2.2 Aims

The main aims for this project was to create a real time Hybrid application that can be ported to a Native build on any popular mobile operating system. The Web Application version of the project must be optimised for all devices, with a user friendly GUI throughout.

The application is to be consistent across all platforms regarding the GUI, events and usability. The application is to allow Pet Owners to build profiles for their pets, perform well without any constraints on device hardware and provide an intuitive way to connect the Founders of pets with their owners within seconds. This application should also create awareness to Pet Owner regarding the importance of providing multiple ways of been contacted if your pet was to go missing. The application should be include industry standard commercialisation whilst also allowing for future improvements outside the time frame of the project.

3.1 Technologies

3.1.1 Hybrid Mobile Application

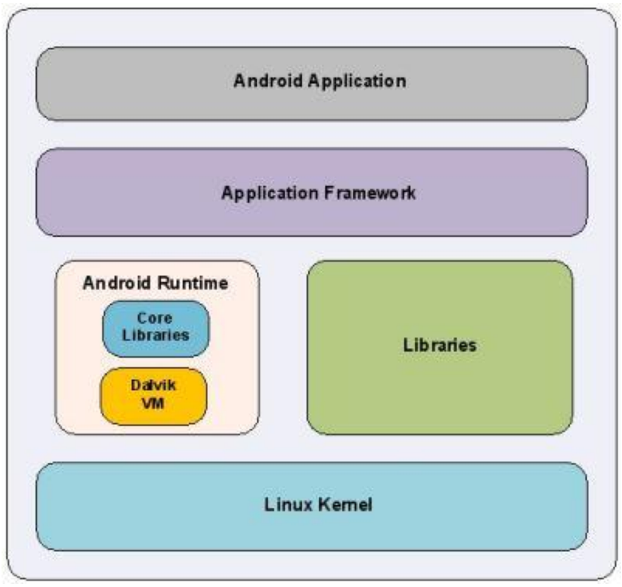
Hybrid mobile applications are typically defined as a web app that run in a browser, built using HTML5, CSS and JavaScript, this can be wrapped in a native build such as iOS or Android. The process involved to complete this is by either using PhoneGap or a framework that supports the Apache Cordova library. PhoneGap is actually a distribution of Cordova although they achieve different things. Hybrid mobile applications are becoming increasingly popular due there only been a need to build one version of an application that can be distributed across different devices instantaneously. Any changes to an application build take affect across all of the devices in one instance. Hybrid mobile applications deliver the same native like User Interface compared to a Native application of both iOS and Android. There are many frameworks that can be used to build Hybrid mobile applications such as the following JavaScript frameworks - Backbone.js, JQuery Mobile, AngularJS, and Meteor.js. These frameworks all allow a developer to build Hybrid mobile applications only the development process is different and also the features provided by each differ.

(Prime Module, 2016)

3.1.2 Native Applications

Native applications are installed on a device and are accessed through using an icon on the actual device. The application itself is installed through an application store such as Apple or Google Play store. Native applications are developed with the focus on one platform. With a Native build you can take full advantage of device hardware such as camera, GPS, mic etc. Increasingly popular nowadays with Native builds you can allow users to interact with the system by using gestures, these can be used offline along with most features of a Native application. Choosing Native development on a project means a company would have to make duplicate versions of the same application providing the exact matching functionality.

3.1- Android Architecture



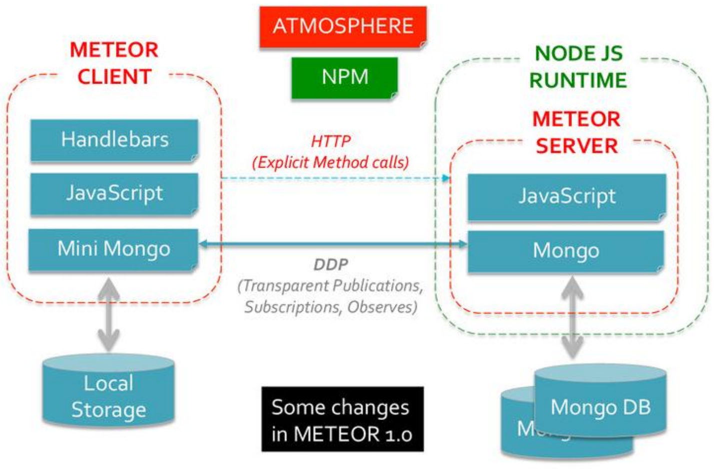
Source: EE Times

3.2.3 Meteor.js

Meteor.js is an open source Full Stack JavaScript framework for building web, mobile and desktop applications. Full Stack Meteor.js allows for development of applications to be real time by default. This framework handles HTML templates using Handlebars.js, this allows for an applications client side to be updated live across multiple clients. MongoDB is integrated in the framework with live database connectors that use a Meteor.js package named Blaze. Blaze is a reactive rendering library that runs in the background on request. Blaze allows for HTML tags to have the ability to be changed in real time. The entire framework is built on top of the cross platform runtime environment Node.js to create a server environment for the applications, it also allows for Meteor.js to be integrated with the npm package manager which allows for an application developed in Meteor.js to be combined with a multitude of different packages from client to server packages by using simple commands to add them to a project (meteor add package name). By using Meteor.js as a development tool it allows for Hybrid applications to be built at a face pace using one programming language for the front and back end (Client, Server). It is integrated with Apache Cordova so it gives a developer the option to port a Hybrid application to a Native mobile build whether it be iOS or Android. Meteor.js gives development an always open option to reuse the one application built across different platforms. For this project Meteor.js will be used to create the application structure, build real time functionalities, and access MongoDB for storage and also to deploy the application to a server using Node.js

(Meteor.js, 2016)

3.2- Meteor.js Appication Structure



Source: MongoDB

3.2.4 Ionic

Ionic is a mobile first framework that allows for quick front end prototyping. It offers front end designs similar of a Native look like iOS or android. It uses HTML, CSS and JavaScript components to create interactive web applications. Along with the framework comes Ionic Icons which allow a developer to use icons you would be familiar with in a native application.

(Ionic, 2016)

3.2.5 Apache Cordova

Apache Cordova is a development framework which allows you to build applications using one codebase that can be ported across multiple platforms as a Native build. It is free and open source. All of the rendering in a Cordova ported application is done via web views. With native applications you have full access to the device hardware, Cordova has built in functions that allow Hybrid applications to acquire these same features.

(Apache Cordova, 2016)

The decision to choose building a Hybrid application rather than developing natively was due to the benefits of developing in this architecture, also the model of this project primarily suited more to Hybrid over Native. The model of this project needs to have the following: web application accessed via a desktop browser, mobile browser and also a native built for the owners of pets to manage their accounts. Building a Hybrid application allowed for this, using Meteor.js allowed for a full stack development style using a built in sub divided version of MongoDB, can easily multiple import packages cutting the development time and cost and also allowing to build one application using one codebase and been available across almost any of the latest smartphone device through whether it be iOS, Android, Windows or other. An major factor influencing the decision of building Hybrid especially for this project model that has been emphasised above; is that 98% of the potential market can be accessed through the application using one build across Android, iOS, Windows etc. Google Maps API inclusion is vital in this project to gather the geo coordinates of where pets have been found and present them to the owner.

4.1 System

4.1.1 Purpose

The purpose of this document is to set out the requirements for the development of the PetPal application. The system is a hybrid mobile application that can be ported to an apk (Android). Building hybrid applications has become increasingly popular, particularly with software developers and companies such as Twitter and Amazon. By building hybrid applications you can cut cost, time and be able to scale to multiple platforms.

(Venturpact, 2016)

This project will consist of a web application as well as an Android ask. The reasoning been for testing. The apk will be specifically for the pet owners to create and update their profiles- both owner and pet details may be stored. If a person finds a lost pet and wishes to use the app. they will not have to download the app to visit the profile. They will simply be able to view a profile through their browser.

The intended users are pet owners who are interested in acquiring an additional layer of security for their pet, alongside microchipping. Microchipping is useful if a person who finds a pet has a mode of transport to bring this pet to a vet. However, if this person does not have access to transport it can be difficult for them to get a pet to a vet to be scanned. Also microchipping is obsolete after working hours at veterinary clinics and local authorities. The application is intended to achieve the same process as microchipping, only accessed at any time through the application.

Only 15% of dogs are returned safely per year. These pets are not always microchipped and some have no collar attached. 15% of pets that are microchipped are found. To counteract this problem, the application will provide information of which pet stores supply engraved URL collars to get setup on the application and also provide are more efficient way of contacting a missing pet owner. With future improvements of the application there would be a system to allow pet owners to design their own 3D printed collar with their pet profile details. This would be possible by integrating with 3D printing hubs across the country.

(Vet Street, 2014)

This project will fulfil the following:

- Personal profile for a pet and the owner
- Additional measure of security for pet owners to compliment microchipping, allowing the founder of a pet to communicate instantly at anytime
- Tracking feature to determine the pet's location when found

4.1.2 Project Scope

The scope of the project is to develop a web application that is also ported to android. The application will be optimized for all devices. The application will use a real time framework and Google Maps API to track the pet. An online storage tool will be used to store uploaded images e.g. Amazon Web Services S3 bucket.

This system should allow for users to register an account and set up their own contact details, as well as their pet's details, a brief description and a profile image. Many pet owners agree that microchipping should not be the only method to track your pet if they were to go missing. The application will allow the user to include details such as the pet's name, age. A real time chat messenger will be included to allow any person who finds a lost pet, to directly contact the owner.

4.1.3 Definitions, Acronyms, and Abbreviations

Geo-location: identifying a person by the geographical location of a person through the Internet.

Meteor.js: full stack JavaScript framework for developing real time hybrid applications.

MongoDB: open source, NoSQL document orientated database (JSON documents).

Iron Router: a client and server package built for Meteor.js to define routes for an application.

Ionic: mobile first front end framework for building interactive User Interfaces.

Google Maps API: a mapping service provided by Google.

Node.js: JavaScript cross-platform environment for running server side web applications

4.1.4 User Requirements Definition

An Android device is a device that runs on the Android operating system. The operating system is purposely intended for use on mobile devices that feature an operating system, applications and middleware. To build an application with technologies such as HTML, JavaScript and CSS, Cordova will be required to port the web based application to android.

The application should be fully based on real time e.g. updating profile information, real time map tracking as well as real time messaging. The messaging should be secure as well as reliable. Real time tracking on Google maps should only be viewed between the owner and person who finds a lost pet.

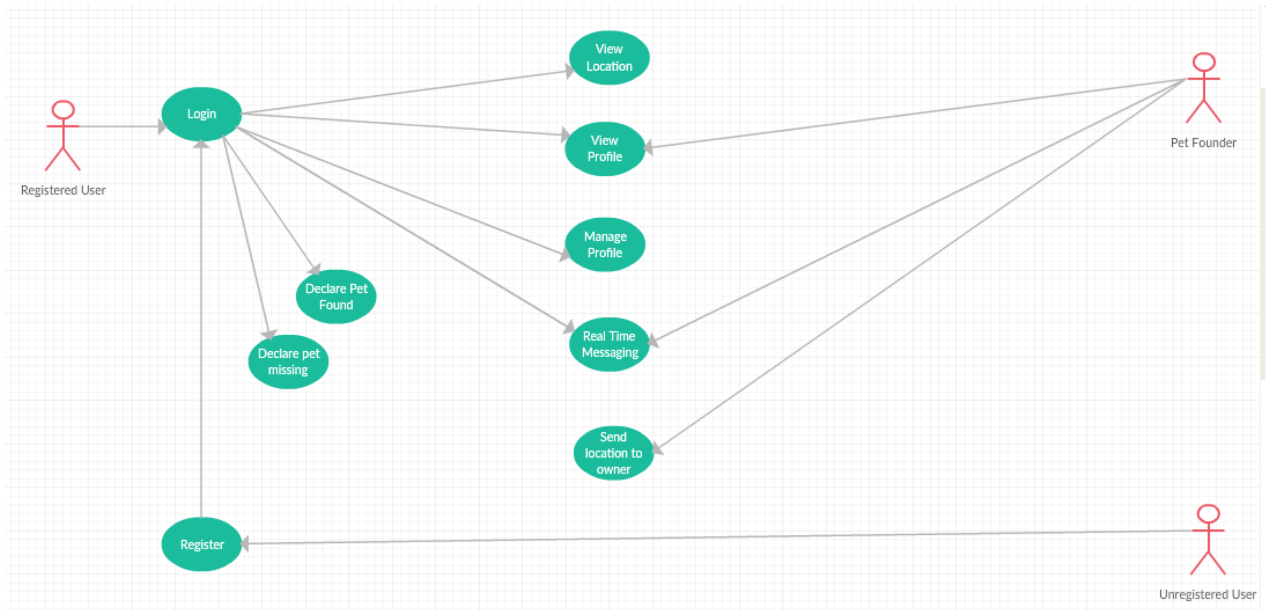
4.1.5 Requirements Specification- Functional Requirements

1. *Register/Login* - The user will be able to register an account by entering their personal details such as an email address, a username and a password to associate with the account. After personal details are entered, the next step will prompt users to add their pets details, including an image of the pet (this is optional but user will be advised to upload an image). When the user's details and pet's details are filled in, registration is complete. The user can then view and update their profile.

2. *Real Time Messaging* - The pet founder and owner will be able to communicate directly by using a real time chat feature within the application. This can be used with map tracking also.

3. *Real Time Map Tracking* - The pet founder will be able to visit the pet profile through a URL entered in the browser. They will then be able to send their location to the pet owner (where the pet has been found). The location will update in real time which can be seen by the pet founder and pet owner.

4.1- Use Case Diagram



Source: Self

4.1.5.1 Requirement 1 <User Registration>

Description & Priority

This requirement is the first step a user must take within the application. The user will either already have an account or will have to register. Registration must be complete before a user can login and create a profile, or have any sort of usage across the application. However a pet founder can access a profile without registering, if they enter a specific URL from a registered pet collar.

Use Case

User Registration & Login

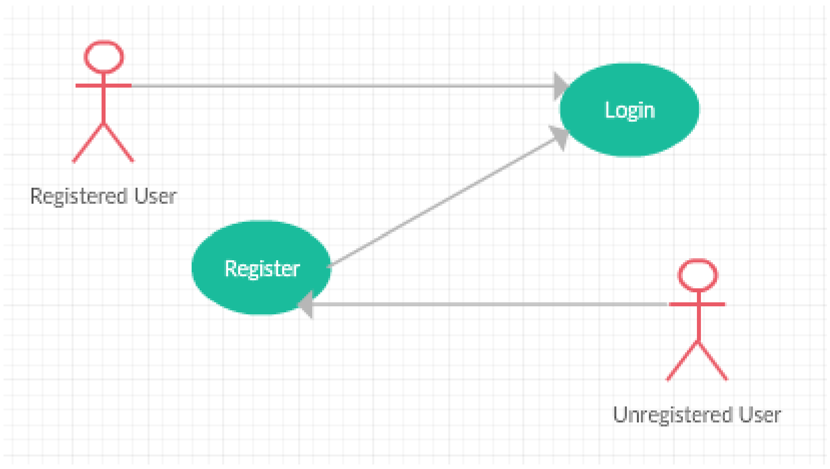
Scope

The scope of this use case is to register a user and login that user with their account.

Description

This use case describes the process of users opening the application on their android device and registering an account to log in.

4.2- Use Case Diagram



Source: Self

4.1.5.2 Flow Description

Precondition

A user has an android device with the application installed, or a web browser that visited the URL.

Activation

This use case starts when an <Unregistered User> clicks on the register account button.

Main Flow

1 User launches Register/Login screen.
2 The <Unregistered User> navigates to register and enters the required information to register an account. (See A1)
3 The system stores this information in the database if the combination of passwords is correct as well as username not already taken and email not already in use.(See E1)
4 The <Registered User>can now login using the details they used to register.

Alternate Flow

5 System validation of the username/password combination fails due to incorrect entries by the user.
6 The system requests the <Unregistered User> to re-enter the username/password combination as it was incorrect. System displays an error message.
7 Goes back to main flow 2.

Exceptional Flow

8 System validation does not find the user in the database.
9 System displays a message alerting the user that the record does not exist.
10 The system terminates.

Termination

The system terminates if the record does not exist in the database.

Post condition

The system is connected and performing tasks as required by the user.

4.1.6 Requirement 2 <Real Time Messaging>

Description & Priority

This requirement hugely important to allow for <Pet Founder> to communicate with <Registered User> directly in real time.

Use Case

Real Time Messaging

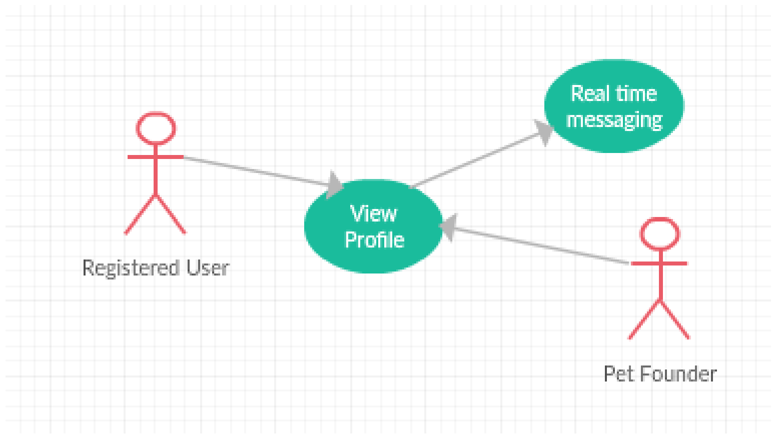
Scope

The scope of this use case is to allow the founder of a pet to communicate with the owner in real time to ensure the pet is returned.

Description

This use case describes how a pet founder can contact the owner of a pet through the pet's profile.

4.3- Use Case Diagram



Source: Self

4.1.6.2 Flow Description

Precondition

The application is installed on the registered user's android device, the pet founder has typed the URL of the profile in their browser.

Activation

This use case starts when the <Pet Founder> views to profile of the pet and navigates to messaging.

Main flow

5 The system identifies that the pet is missing.
6 The <Pet Founder> visits the profile of the pet and navigates to real time messaging(See A1)
7 The system allows for the <Pet Founder> to access the real time chat by signing in as a guest or founder (See E1)
8 The <Pet Founder> sends a message to the <Registered User>
9 The <Registered User> receives a notification from the application and then navigates to the message received in messaging section of the profile.

Alternate flow

A1 : <Reset>

11 The <Pet Founder> receives no response from the owner
12 The <Registered User> forgets credentials and gets locked out of account which leads to credentials been reset.
13 Registered user logs in with new credentials.
14 Goes to main flow 9.

Exceptional flow

E1 : <Communication>

15 There is no network connection while sending the message.
16 The <Unregistered User> aborts the use case.
17 The owner and pet founder speak different languages.

Termination

The system presents the other requirements throughout the application through a navigation bar.

Post condition

The system sends and receives real time messages.

4.1.7 Requirement 3 <Real Time Map Tracking>

Description & Priority

This requirement allows for the founder of the pet to send the location of where it was found to the owner. The owner can then view their pet's current location in real time.

Use Case

Real Time Map Tracking

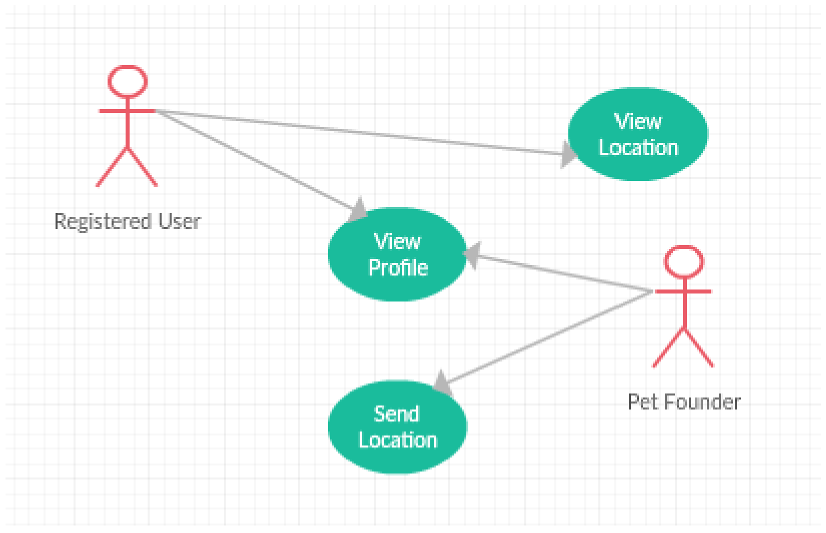
Scope

The scope of this use case is to allow the pet founder to send the coordinates of their location to the pet's owner. The location will be viewed in real time by the owner.

Description

This use case describes how a pet founder can contact the owner of a pet through the pet's profile.

4.4- Use Case Diagram



Source: Self

4.1.7.2 Flow Description

Precondition

The application is installed on the registered user's android device, the pet founder has typed the URL of the profile in their browser.

Activation

This use case starts when the <Pet Founder> views to profile of the pet and navigates to location.

Main Flow

1 The system identifies that the pet is missing.
2 The <Pet Founder> visits the profile of the pet and navigates to location(See A1)
3 The <Pet Founder> sends the location of where the pet was found to the <Registered User>.
4 The <Registered User> receives a notification from the application and can view the location of the pet in real time.

Alternate flow

A1 : <title of A1>
5 The <Registered User> has notifications turned off on their android device.
6 When the <Registered User> signs in the location will be changed to the location that the pet was found.

Exceptional flow

E1 : <Location>

7 The GPS on the <Pet Founder> android device is off.
8 No cellular coverage in the area.
9 The location is not accurate.

Termination

The system presents other requirements of the application through a navigation bar.

Post condition

The system sends the location to the owner via notification.

4.2. Requirement Specification- Non-Functional Requirements

4.2.1 Performance/Response Time Requirement

The response time of the application will be carefully considered during the application development. The registration of users will be smooth throughout as it will be based off a real time framework. The uploading of images as well as updating user's accounts will all be committed in real time. As the entirety of the application will be in real time views, the maps and real time messaging implementation will work well with these technologies.

4.2.2 Recover Requirement

If a user gets locked out of their account by means of forgetting their password or username etc. A recovery option will be available to recover the account. This option can be achieved by selecting recover my account from the settings menu. A security question and email address must be answered before the recovery link can be sent.

4.2.3 Security Requirement

The main security for this application is to keep all details that a user submits to the database secure. Information, such as an email address and password will be stored by the input of the user. Passwords will be hashed with a cryptographic hashing/salt algorithm Bcypt. The user accounts should be immune to any form of hacking.

4.2.4 Reliability Requirement

The application will have to be reliable to achieve the project goals. If certain functionalities within the application do not react as they should, there is clearly a problem within the application. In this case, testing will be carried out to ensure the application is reliable. Functionality, Usability, Automation, Interface, Compatibility, Performance and Security testing will be carried out through the cycle of the application building and completion.

4.2.5 Maintainability Requirement

The application can be easily upgraded to the latest standards and frameworks. As the application will be hybrid, changes can be easily made to codebase which will work across all devices. Any enhancements to frameworks can be implemented by making slight changes to existing code. Using source control will document any changes made to code, allowing back tracking of conflicts with future implementations to be examined through previous versions of the project. GitHub is the tool used in this project.

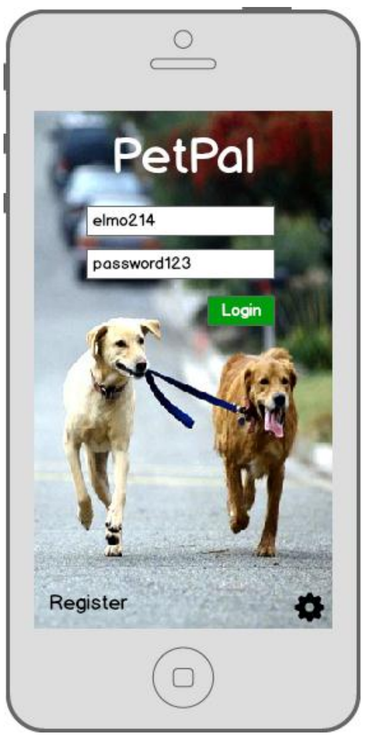
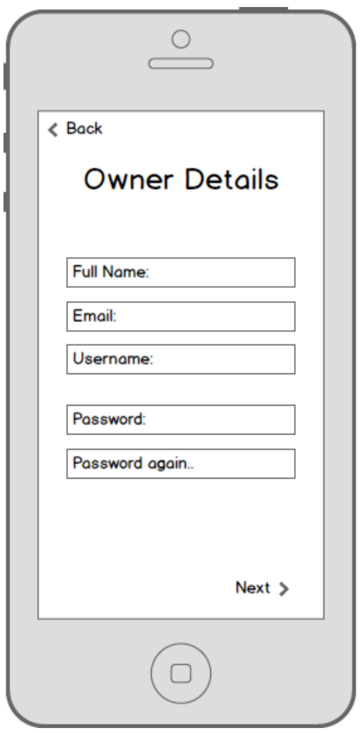
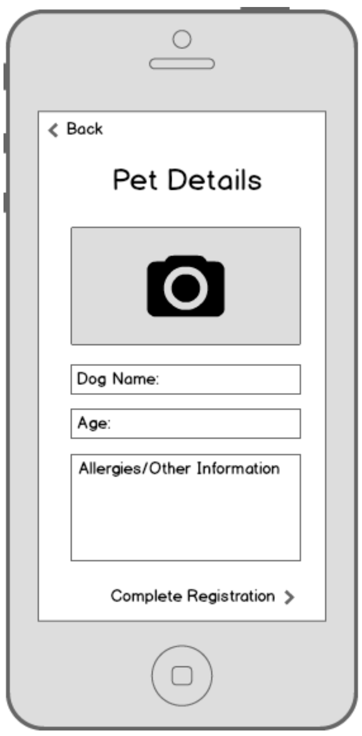
4.2.6 Portability Requirement

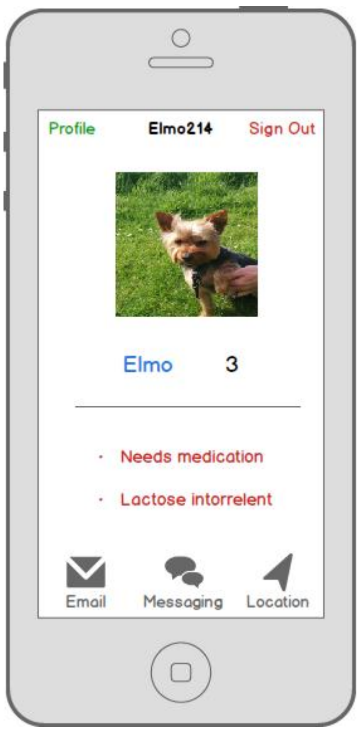
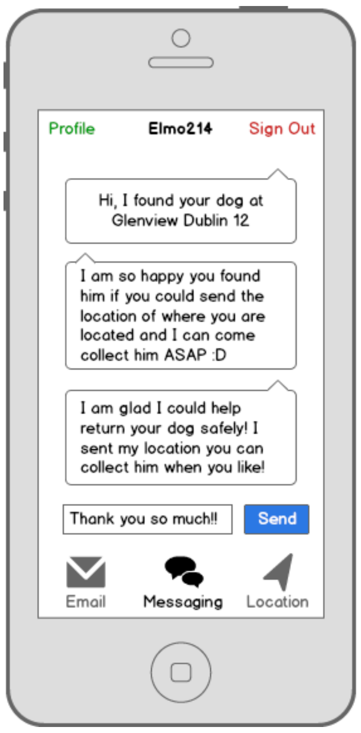
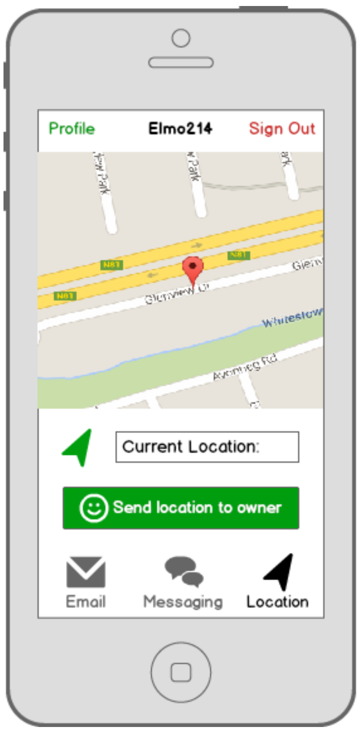
The application will be a Hybrid Application ported to Android. The Android application can be used to allow users to manage their profile. The web application can achieve the same as the android version, but will be more focused on pet founders. The hybrid application can be easily ported to a mobile application across all devices using Cordova, as well as being viewed in any web browser.

4.2.7 Reusability Requirement

The application will be fully reusable in future projects at it will be using web based languages such as HTML, JavaScript and CSS. So the project can be integrated with future implementations at ease especially with the increase in Hybrid application development, Using Cordova these languages can achieve a ported Android build.

4.2.8 Interface Requirements

1	2	3
	 Back Owner Details Full Name: Email: Username: Password: Password again.. Next	 Back Pet Details Dog Name: Age: Allergies/Other Information Complete Registration
This is the first screen the users will see when the application is opened. On this screen the users can login with an existing account or register a new one to and set up their profile.	If a user chooses to register an account they will need to fill in information which can be seen above. The information entered must not contain e.g. an email or username that already exists. Also the password	This is the next step to complete the registration of an account. A profile image can be uploaded for the pet as well as other relevant information being entered to display on the profile.

	combination must match.	
4	5	6
		
After registration is complete the main view of the profile will be displayed. From here users can navigate through the application.	The founder of the pet can contact the owner using real time messaging through the application.	The founder can also navigate to the location which will update the location on google maps of where the dog was found.

Profile Elmo214 Sign Out

Elmo 3

- Needs medication
- Lactose intorrelent

Email Messaging Location

Profile Elmo214 Sign Out

Hi, I found your dog at Glenview Dublin 12

I am so happy you found him if you could send the location of where you are located and I can come collect him ASAP :D

I am glad I could help return your dog safely! I sent my location you can collect him when you like!

Thank you so much!! Send

Email Messaging Location

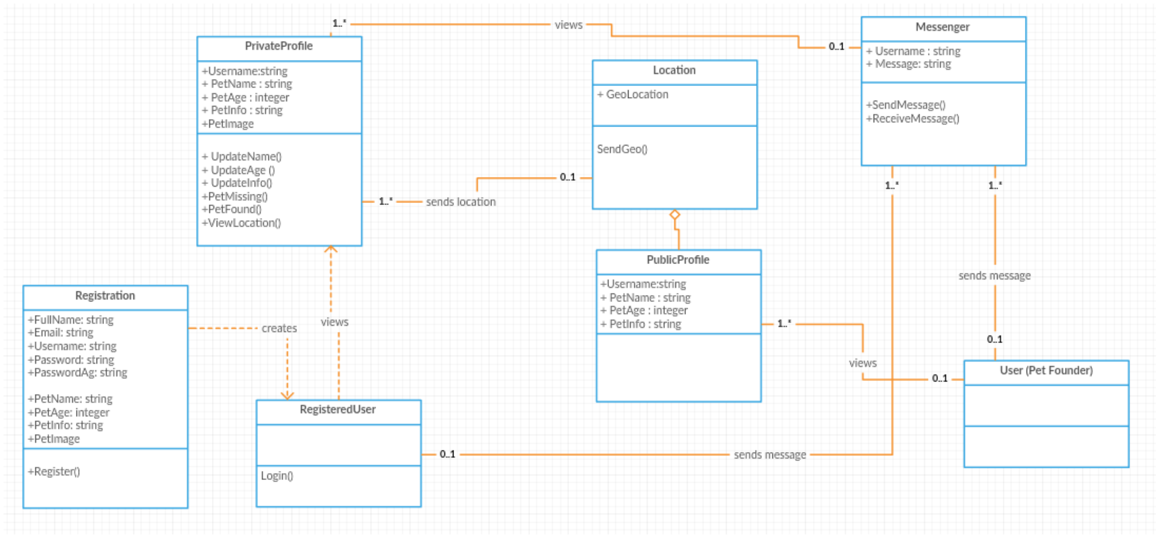
Profile Elmo214 Sign Out

Current Location:

Send location to owner

Email Messaging Location

5.1 System Architecture



6.1 System Evolution

With future implementations, to improve this application it would be functional to add multiple OAuth logins to build a profile. Also allowing users to upload images as they go through the registration process. Pushing the entire application to a secure server on Amazon Web Services would be a future proof implementation for storage, image manipulation through manually installed software such as GraphicsMagick, it would also allow for the application to be easily scaled with an increase in the amount of registered users.

Including a service within the application to create a 3D printed collar that could be personally designed by a pet owner and sent to a local 3D printing hub. Porting the application to iOS would be have to be completed through Cordova after a development licence is obtained. That means the application would be able to run on the two most popular mobile operating system iOS and Android.

Including a forum for pet owners to establish connections with other users, creating posts on training, eating habits and general pet related queries.

7.1 Analysis & Design

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Brian Murphy	02/10/16	Brian Murphy	02/10/16	Design Definition draft
1.1	Brian Murphy	07/05/16	Brian Murphy	07/05/16	Amendments

Introduction

7.1.1 Assumptions / Constraints / Standards

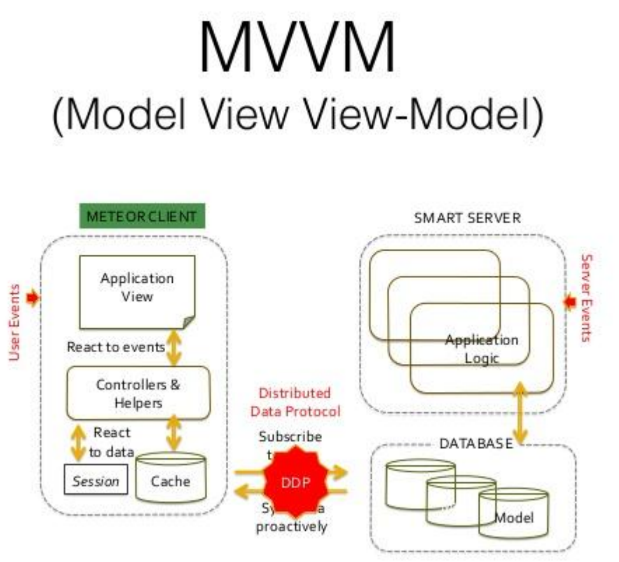
The application design of this project should be implemented in such a way that there is little downtime between the content that is pushed to the client from the server. The most challenging constraint would be the time between data been stored in the database and the client reacting in real time. As Meteor.js ships with MongoDB integrated, these factors must be heavily tested to obtain a smooth application throughout. Google Maps API plays an important role within the application. It is important to prevent constraints on GPS coordinates. GPS coordinates are typically considered sensitive information especially when been push around an application. They must transmit and store in secure manner to and from the database. Instant messaging between users can also have constraints such as private information been included in a chatroom. The messages must be sent in a way that only validated registered users can view and delete messages. A lot of functionalities in this application require usage of outside the framework API's. Including oAuth with Facebook, Google Maps and Ionic. Testing must be carried out to ensure the application can withstand the use of such API's according to web standards.

7.1.2 Architecture Design

7.1.2.1 Logical View

With a Meteor.js application it is broken into 3 important segments. These been the Client, Smart Server and Database. The Client is where all the front-end and backend communications are executed to the server and database. The Client is built around a reactive UI to react to changes made on the application. The Smart Server is the backbone of the architecture of the framework which allows for deployment over a Node.js server environment. Although the database is broken into a third segment, it is lying on the server integrated into the application. Meteor.js applications use DDP as its primary data transfer protocol to communicate with the database with live returning queries. A developer can prevent specific elements from been accessed by using the provide Meteor.js server and client methods. If Meteor (isClient) or if Meteor (isServer).

7.1- Meteor.js Model



Source: Slide share

7.1.2.2 Hardware Architecture

There is no detailed hardware architecture needs. The Hybrid Mobile application build allows for this project to function across any device. With the option to export as Native build, this allows to dedicate the application to specific platforms using the one design. The application is dependent on user's interaction with the system. The users input will navigate the application by touch, content appears and is displayed on the screen.

7.1.2.3 Software Architecture

The pet owner's user end will consist of an application which is Hybrid and ported to Android. This allows this user to manage a public profile, send messages and view locations of the locations their pet may have been found. The person who finds a pet may access the application through ideally a web browser or alternatively download a Native build on their mobile device and search a user. Public profiles are handled by the Meteor.js functions and reactive UI. The messaging system is also build upon Meteor.js methods. Whereas the location functionality is dependent on the Google Maps API. Using this API allows for the user to track their geo-location and send it to the owner of the pet found. The above is all handled on the client side but interacts with the server database. MongoDB is located on the server which is responsible for storing geo-coordinates, user account details and messages. The database is document based, meaning it presents data input from the user to the database in JSON form. An important feature of MongoDB is the storing of unstructured data which plays an important role when storing messages and geo-coordinates from multiple users.

7.1.2.4 Security Architecture

The main architecture of security for this application is the sending of data between two user's applications and server. When a user sends the location of where a pet has been found or sends a message, the application must ensure the sensitive data such as passwords, geo-coordinates, phone numbers, addresses etc. are not compromised. Passwords are secured in a Meteor.js application using the Bcrypt hashing/salting algorithm. To prevent any of this data from been accessed and leaked a Meteor.js deny rule can be added to a publish method within the application. A deny rule is basically a variable that can be declared to prevent a user from seen or accessing specific information. Meteor.js use DDP (Distributed Data Protocol) as its primary protocol to communicate between client and server. DDP is a protocol based on JSON that uses web sockets as its message transportation.

(Meteorhacks, 2016)

7.1.2.5 Performance

There is no performance issue with the sending of data in this application due to the architecture been real time. There is no major requirements on a devices hardware due to no access to it needed other than GPS been enabled only to gather geo-coordinates from a device. GPS is not needed after the location is found, this prevents heavy usage of a devices hardware been needed.

7.1.3 System Design

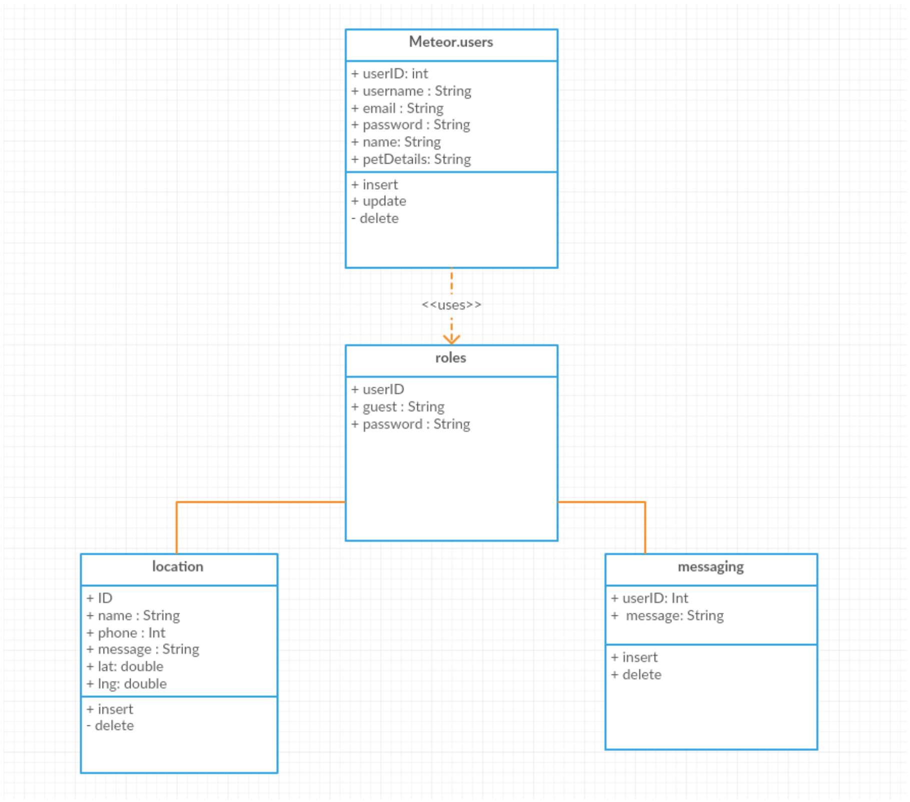
7.1.3.1 Use-Cases

The use cases of this project can be found under the requirements specification heading.

7.1.3.2 Database Design

The database consists of 4 collections including, users, roles, locations and messages. Each field in the database will handle storage of user input. Each collection has a unique ID which helps when returning the data relative to the users. User accounts include signup information such as name, email and username along with pet profile information such as name, age etc. These are known as fields in the database collection. Roles collection is built off the user accounts only thee users that are added to the role are regarded as guest users giving them limited access to certain features such as editing pet owner accounts. The location collection basically stores the geo-location coordinates of the current location of the pet founder along with their details. The messages collection essentially stores each message that is transferred through the one-to-one service.

7.2- Database Design



Source: Self

7.1.3.3 Data Conversions

Due to time constraints and suitability to this project. The integrated MongoDB database was the best choice to use for storing any data to be converted into a form presentable to the user. With MongoDB you can query the JSON stored in a collection by using the find () method or using the fetch () query to pull out specific fields. The JSON format is converted when passed to the client using mini mongo database connecters. This format is presented to the user in a presentable form using a reactive UI Meteor.js provides called blaze.

7.1.3.4 Application Program Interfaces

The system will interact with two API's, those been Google Maps API and Facebook API. Each playing a pivotal role in the system. Google Maps API is responsible for getting the users geo-location and given the user the ability to send these coordinates to the database. Once stored a marker can be placed on places pets have been found. Facebook API is used to login a user to the system and search a pet owner's user profile.

7.1.3.5 User Interface Design

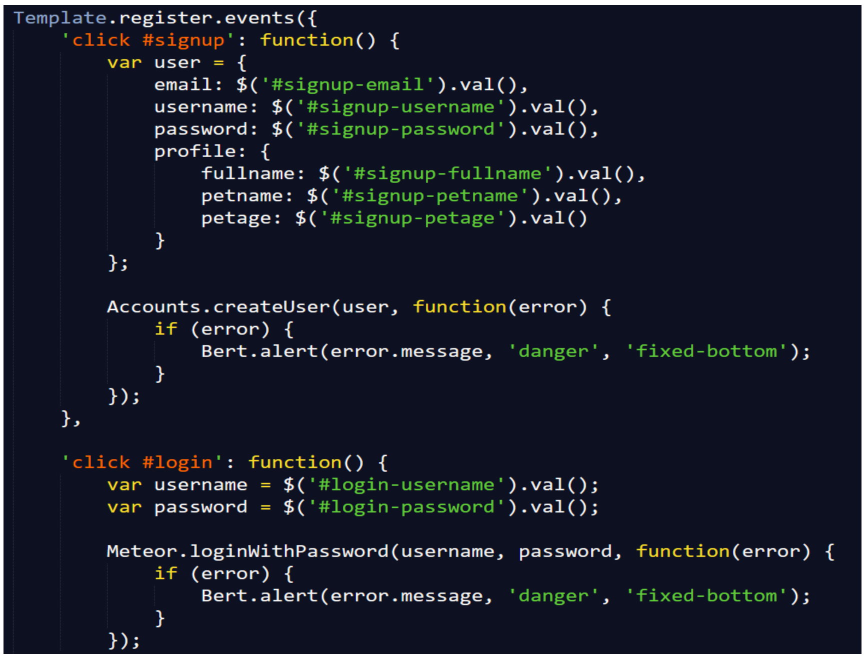
There will be a need for multiple templates for the client UI. These been the login screen the user first sees when the application has loaded, register section, not found error page, user account dashboard, profiles, geo-location template, messaging template and lastly the location found template. All these User Interfaces can be combines easily as the Handlebar.js templating engine allows you to easily call in any templates to any page by using {{> templatename}}. This basically uses sections of a HTML page in another HTML page, combining and re-using existing code.

8.1 Implementation

To develop a meteor application, first you must download the windows or mac installer via meteor.com. Once installed a developer can create a project through the command line using meteor create app name. This builds the essential meteor backend files in the projects file structure which includes MongoDB as well as server bundles using Node.js. It also allows to test the application via a localhost or easily deploy to a web hosting service, all possible through the Meteor.js shell. The 3 main functional requirements implementations is discussed below with code snippets showcasing how it works.

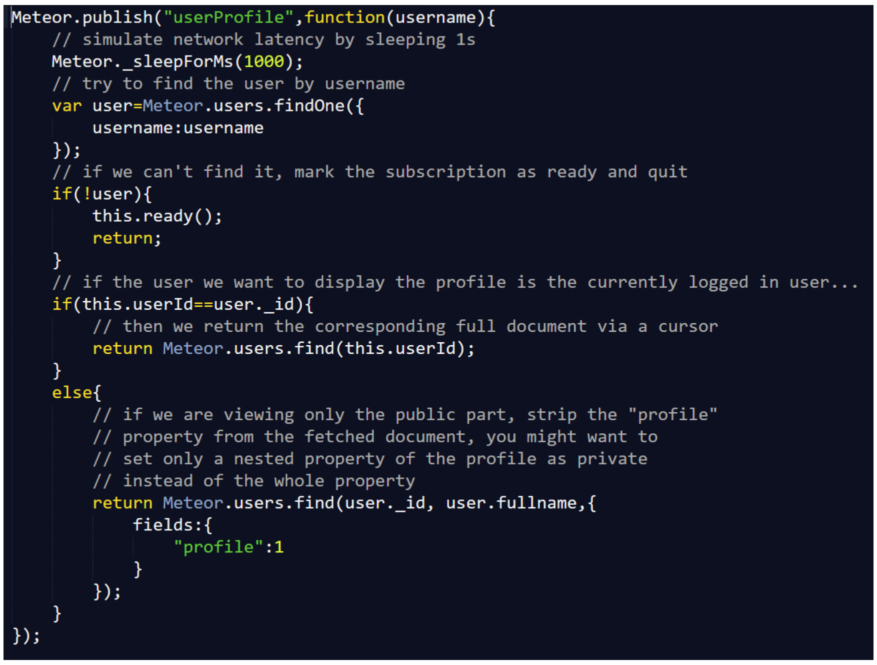
8.1.1 User Accounts

To build a user account system in a Meteor.js application there are a few packages a developer must add to the project based off their requirements, accounts-password is a package that is needed to securely store user passwords by hashing and salting the passwords using Bcrypt. Bcrpyt is a password hashing function used widely in Linux based systems to prevent attacks on applications. Another package to be added in the case of this project is oAuth, facebook-accounts. By using the Meteor.js package manager it adds all the required files to the project, except the developer must add their secret key and access id to a settings.json file. The settings.json file is used in a Meteor.js application to establish connections to API's outside the Meteor.js framework. To create connections to these package to create user accounts, there is client side templating events that are required to send data the end user inputs to the MongoDB on the server side. Below you can find a screenshot of an example of how the userAccounts template event gets the input from and sends this input to the database to register a user or login an existing user.



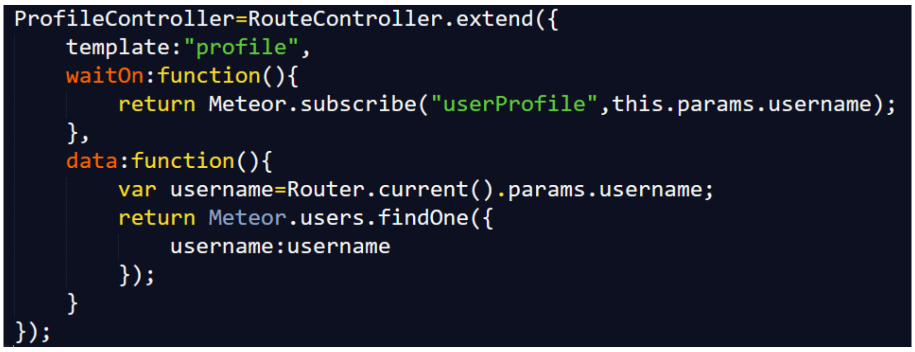
```
Template.register.events({
    'click #signup': function() {
        var user = {
            email: $('#signup-email').val(),
            username: $('#signup-username').val(),
            password: $('#signup-password').val(),
            profile: {
                fullname: $('#signup-fullname').val(),
                petname: $('#signup-petname').val(),
                petage: $('#signup-petage').val()
            }
        };

        Accounts.createUser(user, function(error) {
            if (error) {
                Bert.alert(error.message, 'danger', 'fixed-bottom');
            }
        });
    },

    'click #login': function() {
        var username = $('#login-username').val();
        var password = $('#login-password').val();

        Meteor.loginWithPassword(username, password, function(error) {
            if (error) {
                Bert.alert(error.message, 'danger', 'fixed-bottom');
            }
        });
```

For the above events to execute, subscribe must be invoked to Meteor.publish method. A publish method is used to allow specific information to be seen on the client side. In this project user details must be presented on a profile so a publish method is required. This is to allow the database to be searched by the client so that when it logs a user in, the correct details are presented to them. Below is a screenshot of a Meteor.js publish method sending a query to the database to present searchable information to client side, in this case to search by username and to register and login.



```
Meteor.publish("userProfile",function(username){
    // simulate network latency by sleeping 1s
    Meteor._sleepForMs(1000);
    // try to find the user by username
    var user=Meteor.users.findOne({
        username:username
    });
    // if we can't find it, mark the subscription as ready and quit
    if(!user){
        this.ready();
        return;
    }
    // if the user we want to display the profile is the currently logged in user...
    if(this.userId==user._id){
        // then we return the corresponding full document via a cursor
        return Meteor.users.find(this.userId);
    }
    else{
        // if we are viewing only the public part, strip the "profile"
        // property from the fetched document, you might want to
        // set only a nested property of the profile as private
        // instead of the whole property
        return Meteor.users.find(user._id, user.fullname,{
            fields:{
                "profile":1
            }
        });
    }
});
```

To build the public profiles to work based off the user accounts in a Meteor.js application, using iron-router allows for the username of registered user to be found by search. Iron-router is a package that can be added to a Meteor.js application to send the users to a page based on their interaction with the system. Example: user enters petpalapp.com/register, this sends the user to a template that is rendered based on the contents of the register template. The username is accessible as a publish method was implemented to send it to the client side. The router controller method query's the database to find the username field and allow it to be publically viewed if the user enters the username as a route such as petpalapp.com/username. Below you can view the code which invokes the publish method to create a public profile to be searched by username.



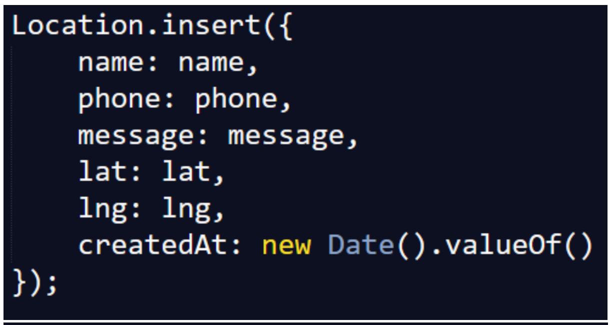
```
ProfileController=RouteController.extend({
    template:"profile",
    waitOn:function(){
        return Meteor.subscribe("userProfile",this.params.username);
    },
    data:function(){
        var username=Router.current().params.username;
        return Meteor.users.findOne({
            username:username
        });
    }
});
```

8.1.2 Location Services

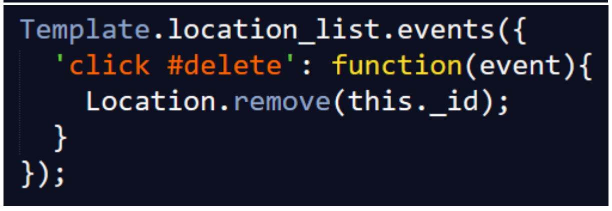
An important functional requirement in this application is using the Google Maps API to get the geographical coordinates of the person who finds the pet to send the location to the owner. Once the application receives the coordinates of where the person is, the map is then loaded with a marker placed on the geo-location of that person. These coordinates can then be sent to the owner of the pet along with a name, phone number and message. When the location is sent along with the other details, this data is placed on a map only visible to owner of the pet. latLng.lat and latLng.lng are two variables used to get the latitude and longitude of this map. These coordinates are store in the database with the user details with a timestamp. A MongoDB insert is performed in JavaScript to store this data and reflect these changes within the application in real time using the live queries database connectors incorporated in the Meteor.js framework.

Insertions and Deletions from the database

The below code uses a template event to collect the input from the use, stores this information with a timestamp. A MongoDB insert is performed when the click event is triggered. Below you can also find a template event which removes the data stored above based on the ID that was given when first sent to the database.



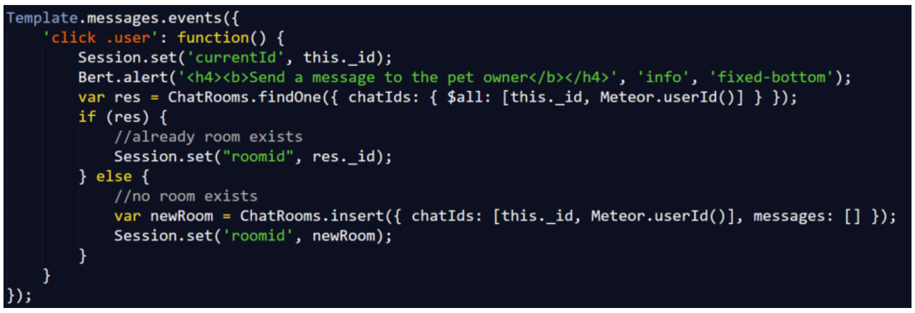
```
Location.insert({
    name: name,
    phone: phone,
    message: message,
    lat: lat,
    lng: lng,
    createdAt: new Date().valueOf()
});
```



```
Template.location_list.events({
  'click #delete': function(event){
    Location.remove(this._id);
  }
});
```

8.1.3 Messaging

The messaging service within this application must provide a pet owner and founder the ability to communicate in real time through one-to-one messages. To do this in Meteor.js you create a session between the two users. First a session is started with the current logged in user, then if this logged in user attempts to send a message to another user, a session is created between them both. The session involves two user ID’s of the users, this creates a chat room between only those two users. When a message is sent they are sorted by the time and date they were entered by using a timestamp. Again a MongoDB find () query is used to return the messages by user ID in this case. If a room between these users was not already set, an insert is performed to create a session between them, if a session was already created the messages are displayed instantly according to the user ID’s in that session room ID. Below is a snipped of code demonstration how the chat room is created on a one-to-one basis.


```
Template.messages.events({
    'click .user': function() {
        Session.set('currentId', this._id);
        Bert.alert('<h4><b>Send a message to the pet owner</b></h4>', 'info', 'fixed-bottom');
        var res = ChatRooms.findOne({ chatIds: { $all: [this._id, Meteor.userId()] } });
        if (res) {
            //already room exists
            Session.set("roomid", res._id);
        } else {
            //no room exists
            var newRoom = ChatRooms.insert({ chatIds: [this._id, Meteor.userId()], messages: [] });
            Session.set('roomid', newRoom);
        }
    }
});
```

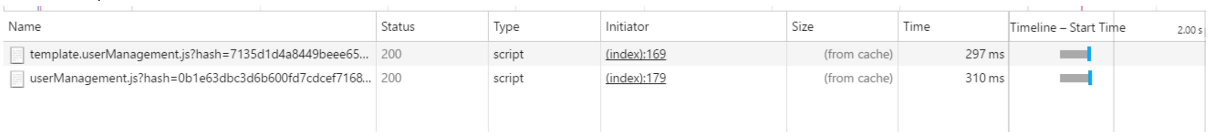
9.1 Testing

9.1.1 Performance Tests

9.1.1.1 Web Browser Console

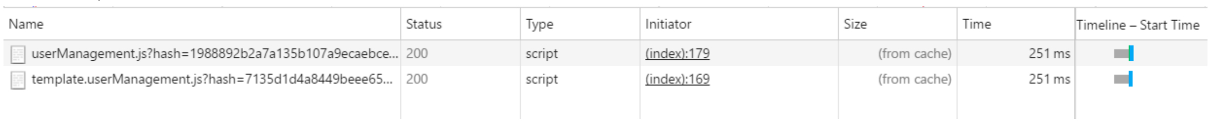
Several network tests were ran using a web browser console. These tests were used to identify loading times of each file and overall loading time of the application. Below you can find screenshots of a test that was carried out on the userManagement template and JavaScript file.

JavaScript code before minimized:



Name	Status	Type	Initiator	Size	Time	Timeline – Start Time 2.00 s
template.userManagement.js?hash=7135d1d4a8449beee65...	200	script	(index):169	(from cache)	297 ms	
userManagement.js?hash=0b1e63dbc3d6b600fd7cdcef7168...	200	script	(index):179	(from cache)	310 ms	

JavaScript code after minimization:



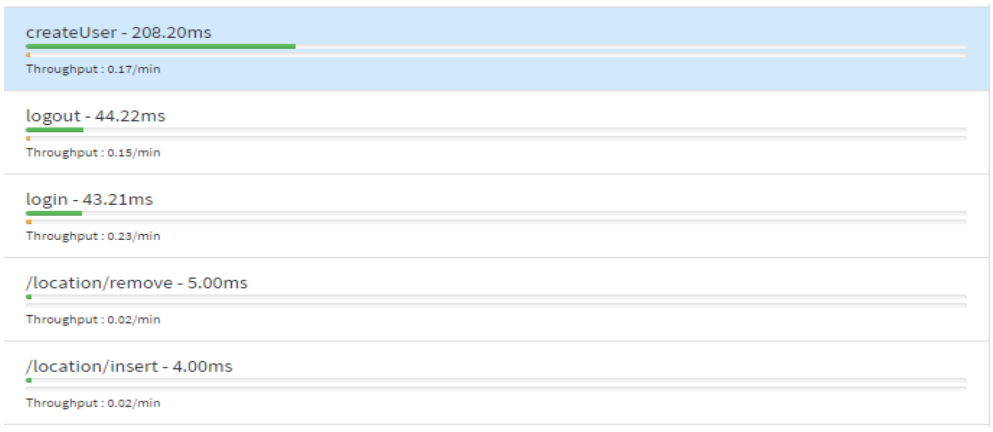
Name	Status	Type	Initiator	Size	Time	Timeline – Start Time
userManagement.js?hash=1988892b2a7a135b107a9ecaebce...	200	script	(index):179	(from cache)	251 ms	
template.userManagement.js?hash=7135d1d4a8449beee65...	200	script	(index):169	(from cache)	251 ms	

As you can tell from the above screenshots minimizing your code can have a major influence on the performance of in this project a hybrid web application. The user management files are extremely important within the application as these files allow the users to create and manage their accounts, these files need to be loaded particularly quick in order to process a login and registration form to the end user.

9.1.1.2 Kadira Performance Monitoring Tool for Meteor.js Applications

Kadira is a performance monitoring tool built specifically for Meteor.js Applications. This is usually one of the first packages that is added as a Meteor.js developer. It is an important package that helps a developer or large teams to track errors, loading issues. An important feature of this tool is that it allows a developer see the errors before the end user does, this is conducted on both client and server side in real time. Tests on Meteor.js methods load time are completed with huge focus on improving the load time. Below you can find the results of various Meteor.js methods running in the application.

(Kadira Performance Monitoring, 2016)


createUser - 208.20ms
Throughput : 0.17/min
logout - 44.22ms
Throughput : 0.15/min
login - 43.21ms
Throughput : 0.23/min
/location/remove - 5.00ms
Throughput : 0.02/min
/location/insert - 4.00ms
Throughput : 0.02/min

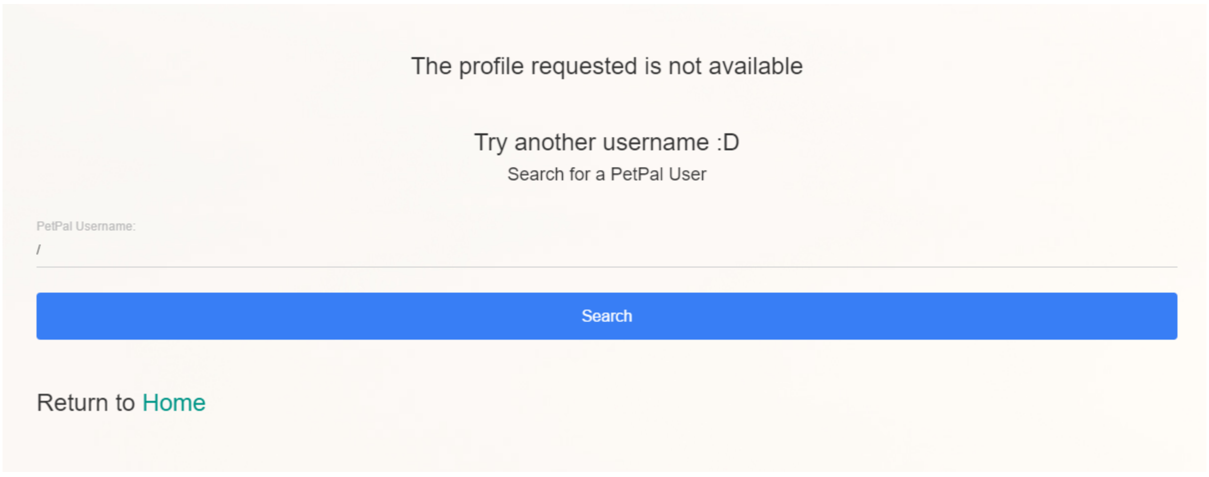
.

The results show the application requires additional loading time creating users and allowing those users to login. This is due to the amount of information and methods it has to process before completion.

9.1.2 Functionality Testing

Invalid Redirects/Broken Links/Validation

Testing was carried out on the entire application to ensure the end user is sent or redirected to the correct link. This application uses Iron Router which is a routing package to send the user on paths according to their interaction with the system. As there was no routes set for when the end user does not find the request page or profile, this means a page would be displayed issuing the developer to set routes for the page requested. To resolve this issue the following was needed: adding error handling to user searches, invalid logins, and pages that are not found, a template to be displayed when a page cannot be found and providing alternative routes. Below you can find screenshots of error messages that are returned to inform the user of what has occurred and also the alternative route when a page is not found.


The profile requested is not available

Try another username :D

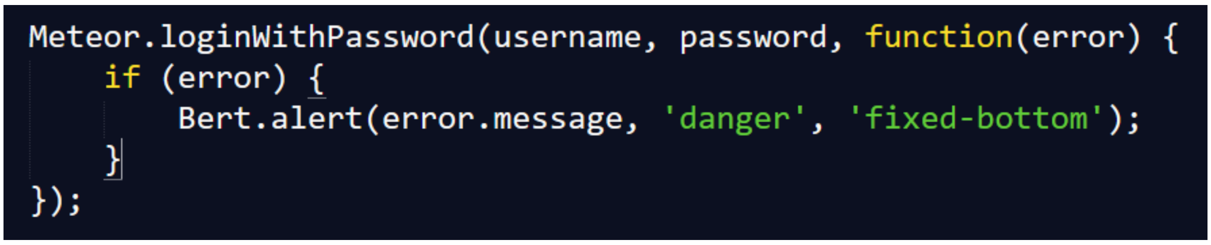
Search for a PetPal User

PetPal Username:

/

Search

Return to Home



```
Meteor.loginWithPassword(username, password, function(error) {
    if (error) {
        Bert.alert(error.message, 'danger', 'fixed-bottom');
    }
});
```

The above code provides validation to the user through meteor-accounts package. The errors returned are based off interaction with a user trying to login with their credentials. Based off correct or incorrect credentials success or error message popups are displayed to the user.

9.1.3 Compatibility Testing

Compatibility testing is a non-functional testing technique typically used to test software across different environments. As the application is built using a hybrid architecture it is important to ensure the application has the capability of working across different web browsers, whilst achieving the same functionality within the android build. Compatibility tests needed to be carried out to examine the differences of the application working across these different platforms and browsers. These differences could be display issues, performance, database issue etc.

By conducting Compatibly Tests, it was clear that Google Maps does not load on Mozilla Firefox due to the application not been deployed on a secure protocol such as HTTPS. Although it may not load on HTTPS, it works across all browsers on Local Host. Due to using a HTTPS server been such a costly process, it was best to run the application on Local, while still deploying the application to the instance on EC2, Amazon Web Services. HTTPS would be a future solution to this problem and would acquire great benefits such as encrypting all communication across the application.

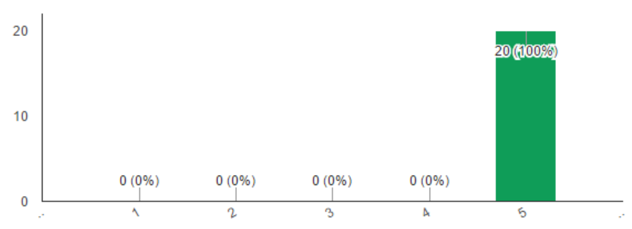
9.1.4 User Interface Testing

This testing technique is used to gather defects of an application in relation to the Graphical User Interface (GUI). This test is conducted by running the application across a broad range of devices such as mobile, desktop, tablet and also devices running different operation systems. This test is very important as the GUI must be consistent across all devices and operating systems, the end user must receive the same user experience no matter what device or operating system in use. With User Interface Testing it is essential to test the navigation, usability and screen validations of application. It is important to take not of events when navigating an application while testing the GUI, as events could vary on different web browsers. The outcome of the test is that the GUI displayed consistently across all browsers due the front-end frameworks used in this project been purposely built to provide applications with a consistent front-end. These applications been

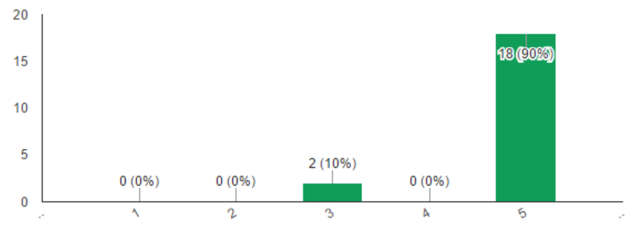
9.1.5 Usability Testing

Usability testing is a black box technique used to test how comfortable the users feel with the system. This is usually based off the flow, content, user interface, speed and user friendliness. Regarding this application Usability tests are used to determine if the user feels the system is easy to use. To run these tests potential users registered an account, login and view their profile. These users were gathered from the same users which completed the survey when the requirements. Below you can find screenshots of the results from the survey.

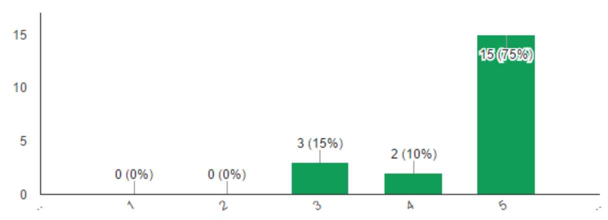
Creating an account was simple? (20 responses)



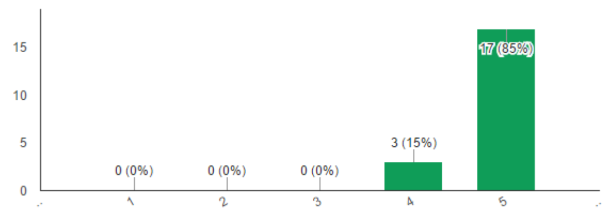
The application was quick and easy to navigate? (20 responses)



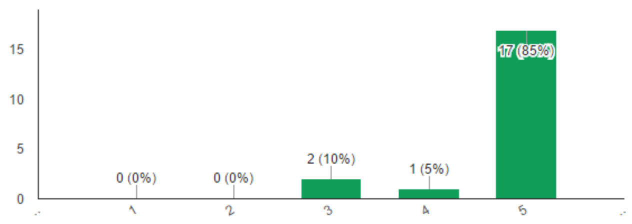
A Pet Profile included the correct fields? (20 responses)



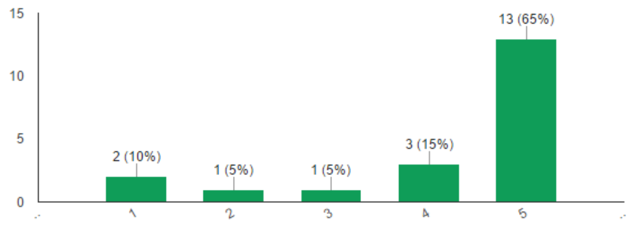
If you found a dog, the correct functionality is there to communicate with the owner?
(20 responses)



The overall user experience of the application was good? (20 responses)



Would you use this application again? (20 responses)



9.1.6 Automation Tests

9.1.6.1 Protractor for End-to-End Testing

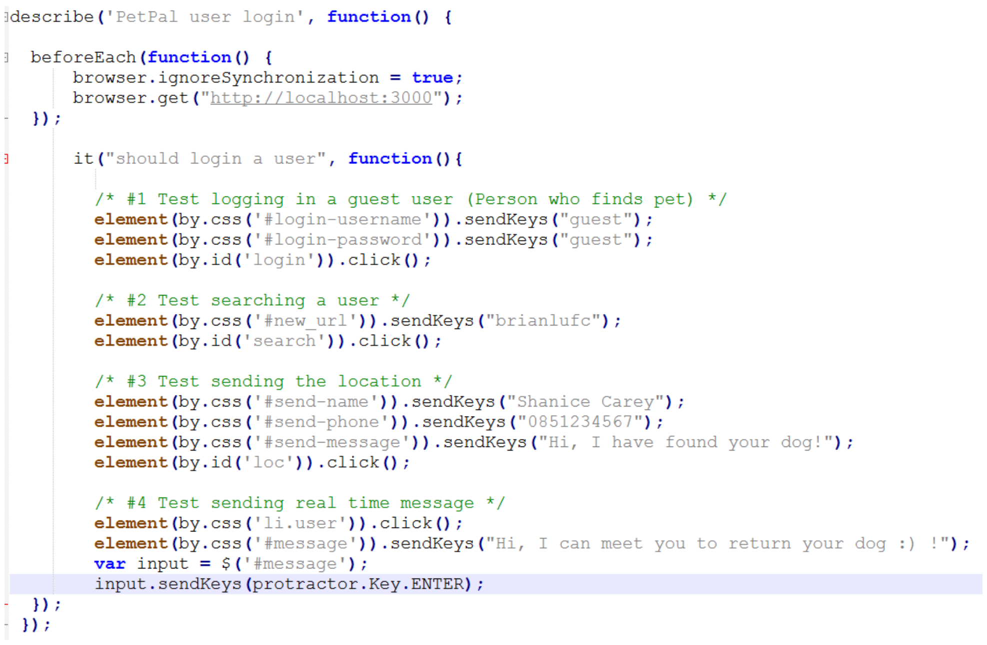
Flow Control

The Protractor automation testing framework is used to examine the flow of an application from start to finish. In case of the this project model where the person who finds a pet needs to visit the profile of a particular pet owner, the flow of this model needed to be tested to ensure the flow is correct and also to ensure each required functionality of the person who finds a pet executes as requested. The test was carried out the following on the below functions.

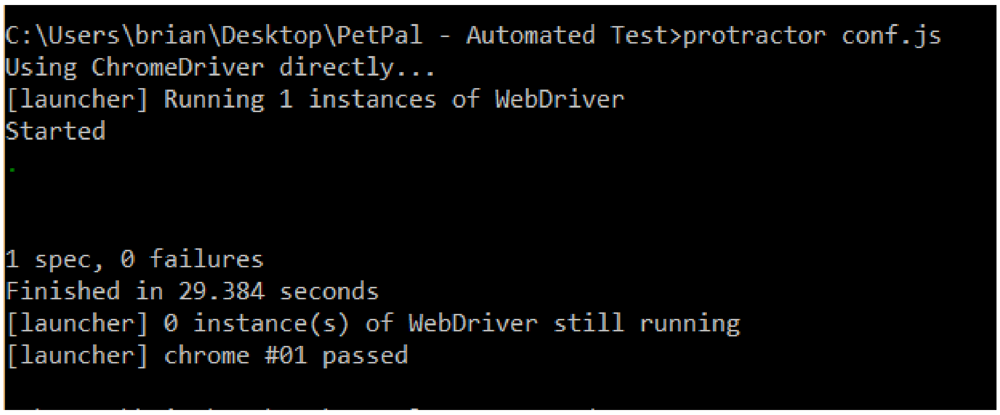
(Protractor Automation Testing, 2016)

- The person who finds a pet logs in as a guest user
- Searches the username of the pet owner
- Sends the location of where the pet was located
- Sends a real time message

(Test Script)



```
describe('PetPal user login', function() {

  beforeEach(function() {
      browser.ignoreSynchronization = true;
      browser.get("http://localhost:3000");
  });

      it("should login a user", function(){

        /* #1 Test logging in a guest user (Person who finds pet) */
        element(by.css('#login-username')).sendKeys("guest");
        element(by.css('#login-password')).sendKeys("guest");
        element(by.id('login')).click();

        /* #2 Test searching a user */
        element(by.css('#new_url')).sendKeys("brianlufc");
        element(by.id('search')).click();

        /* #3 Test sending the location */
        element(by.css('#send-name')).sendKeys("Shanice Carey");
        element(by.css('#send-phone')).sendKeys("0851234567");
        element(by.css('#send-message')).sendKeys("Hi, I have found your dog!");
        element(by.id('loc')).click();

        /* #4 Test sending real time message */
        element(by.css('li.user')).click();
        element(by.css('#message')).sendKeys("Hi, I can meet you to return your dog :) !");
        var input = $('#message');
        input.sendKeys(protractor.Key.ENTER);
  });
 });
```



```
C:\Users\brian\Desktop\PetPal - Automated Test>protractor conf.js
Using ChromeDriver directly...
[launcher] Running 1 instances of WebDriver
Started
.


1 spec, 0 failures
Finished in 29.384 seconds
[launcher] 0 instance(s) of WebDriver still running
[launcher] chrome #01 passed
```

Running the above automation tests demonstrated the correct flow of the application and ensured that the end user can complete the individual processes such as login, search user etc.

9.1.7 Security Testing

In this Meteor.js application a security test is needed to examine how data is stored in the built in Mongo Database. The data includes user accounts, locations and messages. By default when a Meteor.js application is created on the command line using meteor create app_name, the database is fully accessible from the client side, meaning a user can access the database and perform insertions, deletions and updates through a web browser console. This is set by default purely for prototyping reasons. But if a developer forgot or wasn't aware of this it leaves the database open to potential hackers. To prevent the database been accessed from the client side, two packages must be removed from the application. These are insecure and autopublish. The insecure package basically stops the application from data been sent to the database without necessary allow and deny rules. The autopublish package essentially stops the database been published to the client side and server side restricting the user to seen only data they have permission to see.

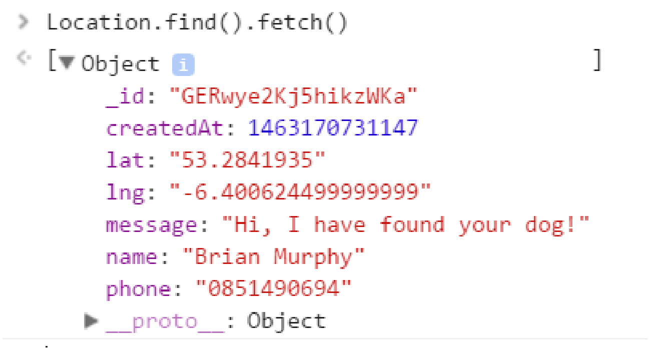
- meteor remove autopublish
- meteor remove insecure

(Atmosphere Autopublish, 2016)

(Atmosphere Insecure, 2016)

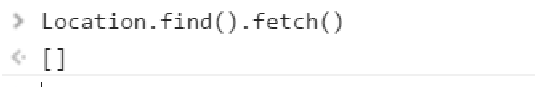
The above commands remove the packages from the application, this should only be done when prototyping is complete as including them allows for quick and easy testing of the data been stored.

If these packages are not removed on release the below screenshot is an example of a query that can be ran through a browser console accessing the data stored.



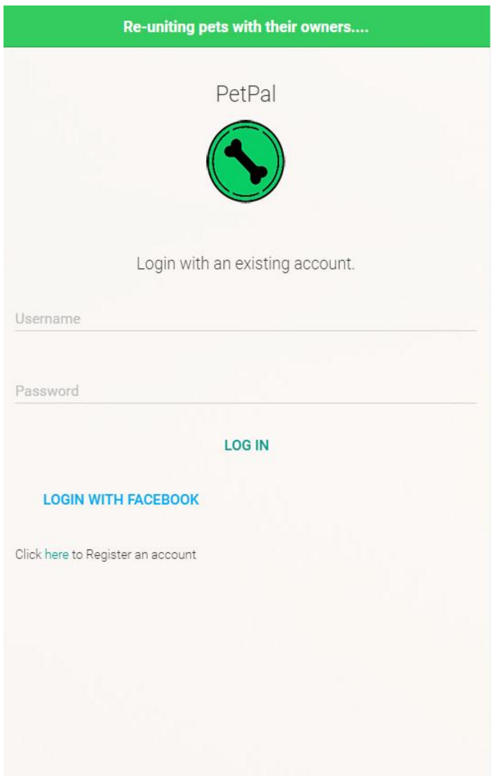
```
> Location.find().fetch()
<· [▼Object                                    ]
      _id: "GERwye2Kj5hikzWKa"
      createdAt: 1463170731147
      lat: "53.2841935"
      lng: "-6.400624499999999"
      message: "Hi, I have found your dog!"
      name: "Brian Murphy"
      phone: "0851490694"
    ▶ __proto__: Object
```

As you can see above all the details that was sent to the database is fully accessible. Removing these two packages result in an empty result from the database which is shown below:



```
> Location.find().fetch()
<· []
```

10.1 Final Graphical User Interface



Re-uniting pets with their owners....

PetPal

Login with an existing account.

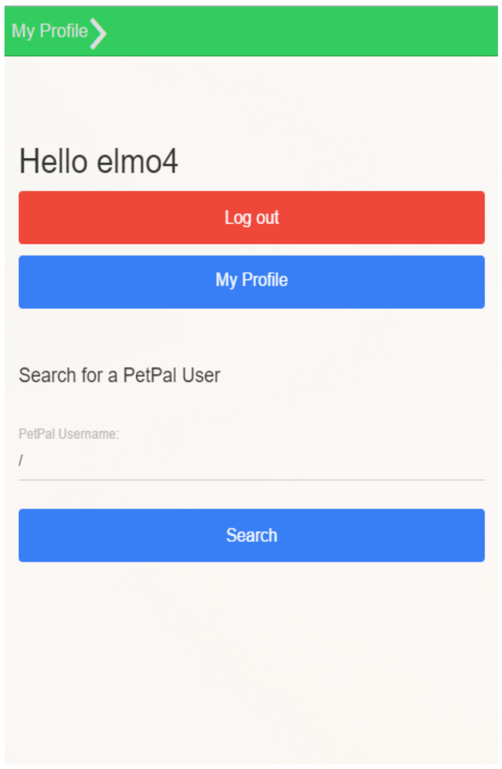
Username

Password

LOG IN

LOGIN WITH FACEBOOK

Click here to Register an account



My Profile

Hello elmo4

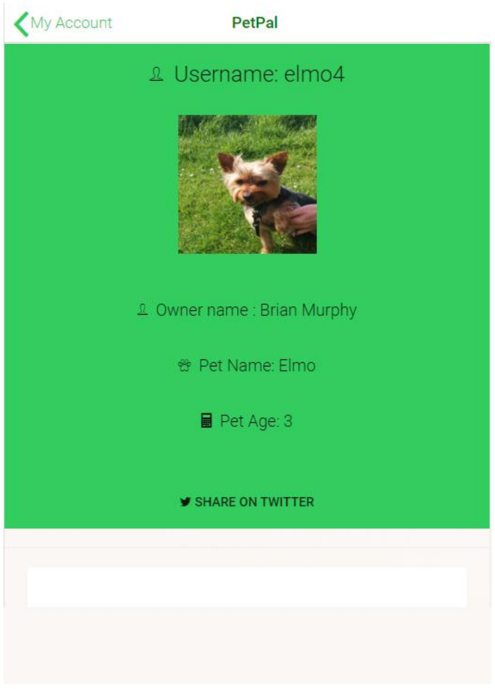
Log out

My Profile

Search for a PetPal User

PetPal Username:

Search



My Account PetPal

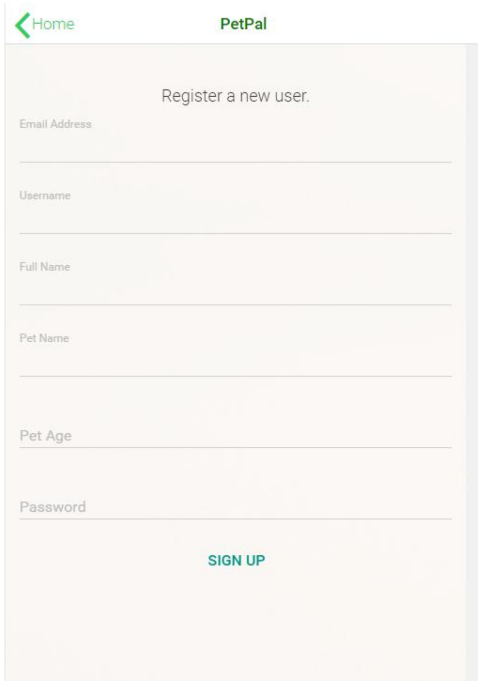
Username: elmo4

Owner name : Brian Murphy

Pet Name: Elmo

Pet Age: 3

SHARE ON TWITTER



Home PetPal

Register a new user.

Email Address

Username

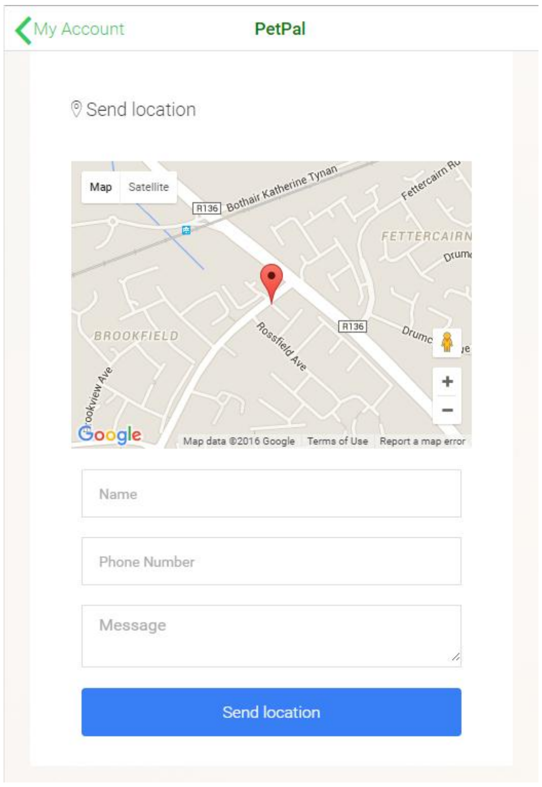
Full Name

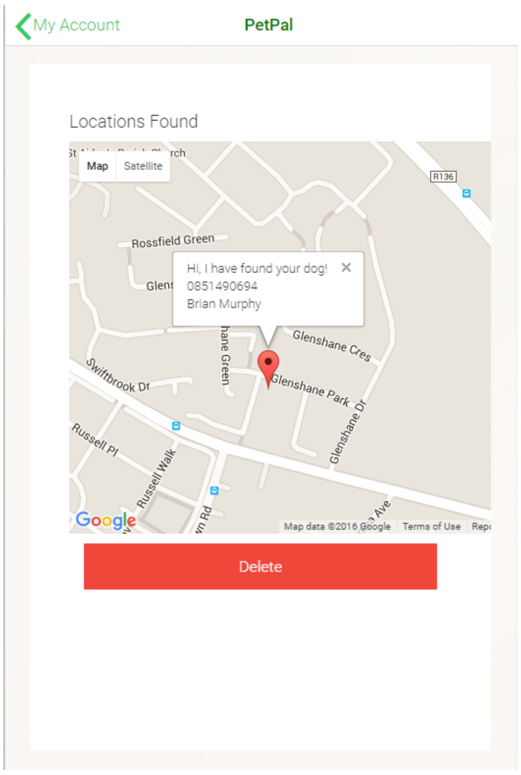
Pet Name

Pet Age

Password

SIGN UP





Hi, I have found your dog!
0851490694
Brian Murphy

11.1 Conclusion

This reports sets out the reasoning as to why I developed a Hybrid Application using Meteor.js. It demonstrated my passion, motivation and unconditional care for dogs and pets in general. It describes exactly what technologies were required and used and the reasoning behind choosing such technologies. The system requirements are gathered through customer involvement exercises using online surveys and one-to-one communication with Pet Owners. The requirements are discussed in detail including the three main functional requirements along with many needed non-functional requirements. Testing was carried out in depth with multiple different frameworks and testing tools that are widely used in industry. Using Facebook API and Google Maps API allowed for integration with Meteor.js real time backbone, which is not as straight forward with any other framework.

Working in depth with a full stack framework for the first time was a very interesting experience. It allowed me as a developer to gain knowledge of new upcoming real time technology's that is improving by month due to its large investment, I also gathered experience with MongoDB, Node.js and Ionic using this framework. Collaborating with multiple frameworks using the one stack was daunting at the beginning but once you carefully read documentation and run through example applications and tutorials, the development process begins to become more familiar and grounded. The project idea become really strong through the development lifecycle due to feedback from potential customers which gave me great inspiration to build the application to the best of my abilities. Creating interactive modernised User Interfaces with Ionic was a pleasant experience as I had never built a Native like GUI with a Hybrid Application of this stature. Overall full stack development in my opinion is going to become only more popular in coming years and I can foresee Meteor.js been used by a lot of high end IT companies in coming months especially to build enterprise applications in half the time it would take with a divided single stack of technologies.

Appendix

Project Proposal

Objective

The primary objective of this project is to create a web based application to allow pet owners to register their information of their pet to be stored securely. Users should be able to register using a form to input the information directly through the application or register and login using their Facebook account. Using Facebook to register should automatically collect the users details such as name, phone number, email address etc. whereas all information used within the app must not be viewable with other users or non-users. To do restrictions should be set on users that are not logged in as a pet owner.

The second objective is to generate a profile URL routes code based on the URL of the pet profile which can be printed and placed on the pet's collar. The URL will act as a direct link to the pet profile if the situation ever arose which the pet became missing. The founder of the pet should be able to visit the URL which brings you to the profile

The third objective is to use Google Maps API to allow for the user to set the location of where the pet was found. This location will then be sent to pet owner.

The fourth objective is to build a simple UI that reacts quickly with regards to signing up, logging in and viewing pet profiles. The design will also be responsive to all devices.

- User sets up the pet profile
- Prints URL on collar of pet.
- If the pet is missing the founder uses the code on the pet's collar to view the profile.
- The founder then can access the owner's details and also set the location of where the pet has been found.

Technical Approach

As this app will be used solely for finding pets, my main research will be carried out at local dog parks and pet stores. Sufficient surveying with 100 pet owners' cat and dogs specifically. Gather information based on whether pet owners would be attracted to using an additional measure for their pet.

A database populated based on user registration, image upload paths, locations and messages. These are broken into individual collections in the database.

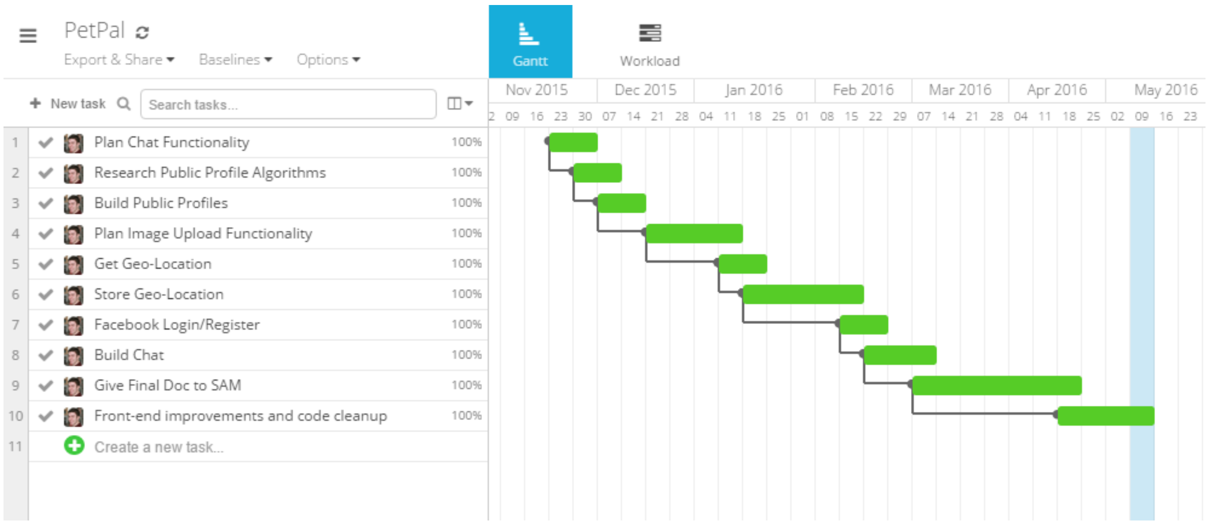
My web application will require the following:

- A network connection e.g. Wi-Fi, 3g, 4g, edge.
- Smartphone; android or iOS.
- Info such as name, phone number, pet name, pet allergies etc.
- The user will be required to upload an image of their pet.
- Optionally a Facebook account can be used to gather the owners name and phone number.

Special Resources Required

Android, Desktop, Tablet.

Project Plan



1	Plan Chat Functionality	100%
2	Research Public Profile Algorithms	100%
3	Build Public Profiles	100%
4	Plan Image Upload Functionality	100%
5	Get Geo-Location	100%
6	Store Geo-Location	100%
7	Facebook Login/Register	100%
8	Build Chat	100%
9	Give Final Doc to SAM	100%
10	Front-end improvements and code cleanup	100%

A more detailed overview of each task can be found in the implentation section of this report.

Technical Details

As it is such an early stage of my project cycle it is not feasible to decide what languages to definitely use but it is clear that using Node.js, Cordova and Google maps API will achieve the project model. (September – October 2015)

Evaluation

The data that is sent to the database on registration will be tested thoroughly. Also tests will be executed when sending and getting data for each pet profile that is created. Searches for profiles must obtain the correct results meaning return the user profile of what was input. The Google Maps location will be tested to ensure it matches the location of the user's geo-location. The entire site will be tested on a range of devices to ensure it is responding to each device individually.

Alternatively some outside users will test the application.

Reflective Journals

Reflective Journal – September

This month I put a lot of thought into my project idea. I had up to 5 completely different project ideas I could foresee. After careful thinking and a lot of conversation with friends, family and my project supervisor I decided I wanted to do a pet application. Sam Cogan my project supervisor played a huge part regarding my decision of my project idea as he gave me great advice and also some guidance with what programming languages and tools I could use, additionally some functionality I could add to the application.

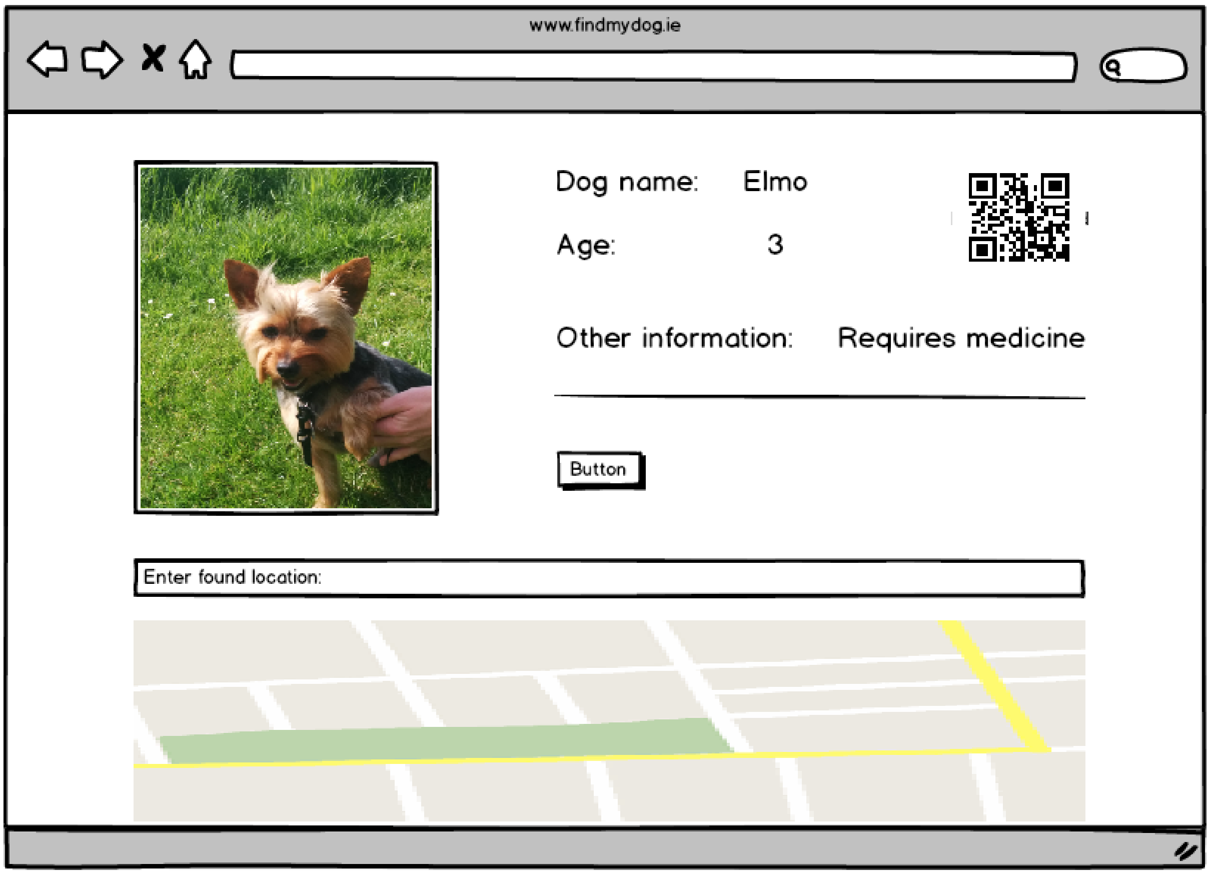
The purpose of such an application was to help pet owners by adding an additional security measure to their pet's collar by means of a URL to the profile. The application would act as a profile for the pet with relevant information such as pet name, any allergies and most importantly contact details via Facebook or direct to the pet owner. Additionally if the pet was missing and found by a person, this person could go to the pet profile and set the location of where the pet was found. When the pet is found a notification will be sent to the owner to confirm the pet has been found.

I have found it difficult to think of a name for the application to stand out and tie in with existing trends. I have started to ask friends, family, work colleagues for suggestions of a cool but relevant application name.

I have set up phone gap and node.js on my laptop to begin with coding the project. Node I believe is the best way to develop my application as there is so many packages available to add additional functionality to an application with ease.

My Achievements

This month, I was able to map out exactly what I wanted my project to do. This included that I would require a database to store all relevant information for the pet and their owner. My contributions to the project included researching various technologies I could use for the project as well as searching for applications that may be similar to the one I intend on developing. As I intended on developing a web application and my stream is partly focused on Mobile; I would like to use phone gap to port my web application to android specific to tick off both boxes for networking and mobile. I also mocked up a wireframe for the profile.



My Reflection

I felt, it worked well to map out exactly what functionality that will be included in the project. I have created a few wireframes for how I would like the application to appear across different devices. However, I was not successful in deciding on an application name.

Intended Changes

Next month, I will try to decide on the application name, start building the registration system and connecting to a database. Along with designing how the pet collar will appear.

I realised that I need to research alternatives to QR codes as 1 in 5 people use them which limits the amount of users that would use the application.

Reflective Journal - October

My Achievements

This month I focused on the research of various frameworks that I could use to begin my project. As my application will use real time map tracking and real time messaging, it makes sense to collaborate with one of the many real time frameworks available. I have tested and built sample applications using frameworks such as Meteor.js and Firebase. I have also become very familiar with Ionic. Ionic is a hybrid mobile application framework which allows for building applications, it has a similar interface to native android.

I have brainstormed ideas for the name of the application, and have decided to name it PetPal. I have conducted a small bit of market research by mentioning my product idea to pet owners including colleagues, friends and relatives. The feedback so far has been great, and many people seem interested in having an application that provides additional measures towards finding missing pets. I have also created a market research survey using survey monkey to ensure my project idea is actually feasible with pet owners. I will attempt to survey 50 pet owners in the coming weeks.

A couple of hours after my first meeting with Sam, I realised that I don't have to build just for android by porting using Cordova. I decided to focus the application on the user's too e.g. pet owners and the web application version for the people who find the pet and go to the profile. Overall, I have built an improved knowledge of real time application development with the use of frameworks.

My Reflection

This has been a productive month in relation to researching which frameworks I will use for my application. I have yet to select a framework due to such a wide selection of frameworks being available. I have tested some frameworks that I did not come across at the beginning of my research, such as PubNub which my supervisor Sam had mentioned, and also Firebase. My intentions are to compare each of the frameworks and work with the framework that best suits my project with regards to functionality, storage and response time.

Intended Changes

- Porting web application to android application, as well as using the web application version for pet founders that visit profiles.
- The name of the application will be PetPal.
- Using a simple URL to direct users to the profile of the pet, by an engraftment on the collar.
- Possibility of using real time map tracking, although static geo location is still the first goal.

Items discussed:

Real Time Frameworks, Technical Document, Comparing Frameworks, Online Storage, Amazon AWS.

I met with Sam Cogan to discuss my project and to ask him to be my project supervisor. I felt Sam would be a good support for me, as his background is very similar to certain technologies I intend on using throughout the project.

Later this month, I received an email to confirm that Sam Cogan was my project supervisor. I was recommended to meet with Sam regularly, in order to progress with my project lifecycle. I sent an email to Sam immediately after the confirmation to arrange our first meeting.

My first official meeting with Sam as my supervisor went very well. We thoroughly discussed my project as well as frameworks and document submissions. We discussed frameworks that would be suitable to meet the needs of the building of my project. As the pet application is partially in real time, it had become clear to use one of the many popular real time frameworks for building mobile applications, such as Meteor.js, PubNub, Sails and Firebase. Sam put a huge emphasis on documenting everything I test, research or even an idea that comes to my mind. He mentioned that it's very important, especially for the technical report as well as the overall outcome of the application. Sam said to take note of everything I do on a notepad, as well as saving links I have visited relating to frameworks I could use within the application.

We also discussed main features that would exist within the application, such as using the Google Maps API to allow for real time tracking as well as sending direct or real time messages.

Action Items:

- Test example apps of Real Time Frameworks
- Read documentation of frameworks
- Research requirements for application features
- Research storing methods of uploading images
- Document any testing, research etc. - It will help with writing the report

Reflective Journal - November

My Achievements

This month after deep research into frameworks for building real time applications, I made a decision to build my application with Meteor.js. Meteor.js as a framework has been to most impressive framework to build with through all of the testing I have done and I also feel it offers the most to achieve the functionality I plan on including within the application.

So I began to develop my application with Meteor.js. I setup a basic application using commands provided by Meteor. This was done in the command line by typing meteor create PetPal. Meteor sets up the basic file structure to begin coding your application. This includes in mini mongo version of MongoDB. The first thing I wanted to include in the application was the basics. This was to create a user management system. Meteor also makes this relatively easy as it provides a package with user accounts setup and ready to go with a collection stored in the database. To do this in the command line I simply typed meteor create accounts accounts-password. Although this done some work for me, I still had to ensure the data that the user input on the client was being stored so I created a userManagement JavaScript file which sends the data to the users collection.

As the android apk version of the project will be used specifically for the registered user it was important for me to set out reasons why I have chosen to do this. I have based the project model off a similar one to AirBnb where the hosts would use the app to communicate with their customers whereas the customers would use all platforms.

My Reflection

This month I feel I have nailed down what technologies I am going to use to build the project and I also think that it is very achievable with the technologies I have chosen. Although the framework is particularly new, my past project work will guide me through the frameworks unfamiliar functions.

Intended Changes

- Using Meteor.js to build application
- Using Cordova to port to android within Meteors package.
- Project model changed

Items discussed:

Meteor.js, User Accounts.

In this month's meetings with Sam we discussed in detail my plan with regards to using the Meteor framework to build my project. We also discussed briefly the application model which I would set out for potential customers. Sam mentioned that I should have a look at Asana.

Action Items:

- Sign up for Asana

Reflective Journal - December

My Achievements

This month has been very difficult to focus any time on the project due to other modules projects being due along with study for upcoming January exams. Although time has been limited this month I signed up to Asana to manage my time with my project more effectively. Within a week I found it to be very helpful to focus specific days on my project and leave other days open for other work for CA's and study etc.

My Reflection

I have not completed the work I set out for this month as it was very busy with other modules. I have used Asana to be more productive with project time after the January exams.

Intended Changes

No changes for this month due to the busy period.

Items discussed:

Asana.

Sam guided me how to effectively manage my project time by using Asana. A project management tool which notifies you via email when tasks are due. This really helped me set out a time scale to complete tasks.

Reflective Journal – January

My Achievements

After the exams I was able to focus more time on the project as we had a couple of weeks before we started second semester. I continued where I left off with building the user accounts system. I created a UI for login and registration using the Ionic framework. This allowed the users to register an account and login with a simple form with an attractive native like UI

After I had the basic user system created my next task which was noted on Asana was to create public profiles. I did this by using methods which are built into Meteor. I created a publication.js file which is used to publish user data to client. This allowed for public profiles to be accessed by a person that finds a pet through a URL found on the pet's collar. Data such as the pet name, owner name and age of the pet will be visible on the public profile. The next step was to add more fields to the user's collection in the database. With meteor accounts some fields are already setup in the collection such as email, username and password. I needed to create a sign up function which would add more fields to this collection. I did this by adding more fields to the profile { } inside the signup function. Once the fields were added and a user registered I needed to create a route which would send the user to their profile once they are logged in. I did this by using Iron Router which allowed me to check if the user was logged in to redirect to the user profile. To access the profiles publically and as a logged in user I wrote a function which finds the user by id or username.

This function would execute when a public profile is typed as a URL in the browser or when a pet owner is logged into their account and wants to navigate to their profile via a button click. Next up was to start preparing for the presentation. This is very difficult as there is a lot going on in other modules. But I still have a few weeks to prepare so it should be OK!

My Reflection

This month has been challenging as I started to write meteor functions within my application. The public profiles was a time consuming task as it required me to look in depth into the meteor documentation as well as online examples. Using Asana to manage my project time has allowed me to progress well with tasks. I would complete one task before beginning the next one and also document the steps in which I took to complete a task.

Intended Changes

Build simple user interfaces for each section of the application.

Supervisor Meeting

Items discussed: User Accounts, Asana, and Mid-point presentation.

I discussed with Sam my idea for the user accounts and public profiles. Sam helped me set up future tasks for the project so I could try keep on top of each individual task within a certain time. We also discussed what I should include in the presentation regarding content.

Reflective Journal – February

My Achievements

This month I began to work on a messaging system for the application, this is an integral part of the application due to it been one of the functional requirements. The logic behind the sending of messages between the founder of a pet and the owner should be only one-to-one, meaning no other user can send messages unless logged in and viewing that profile. To build this system I needed to re-use some knowledge I have gathered whist developing this Meteor application over the past few months. This knowledge included storing data in the database and taking advantage of the live querying database connectors. Using Asana has helped me a lot especially for managing time with each individual task. Outside building the messaging system, I looked over using oAuth implementation in Meteor applications. It seemed a relatively easy process, so the next day I created a Facebook login in a test version of the application then later integrated it with the current project. It required adding the facebook-accounts package and a small snippet of code to create the user session when a user clicks login.

My Reflection

Asana has begun to play an important role regarding managing my time effectively with the tasks that are yet to complete. Implementing oAuth was actually relatively straight forward in a Meteor application.

Intended Changes

Continue to work on messaging system, improve the UI by adding Ionic components.

Supervisor Meeting

Items discussed:

Next tasks, messaging logic.

I and Sam discussed the logic that I planned on including within the messaging system. We also discussed my next steps after the messaging system is complete.

Reflective Journal – March - May

My Achievements

This month I very busy with implementing the final functionalities of the application. This included finishing off the messaging system. As the messaging system was now complete I could then begin to work on the location requirement. This was to get the current location of the user which I already discussed, but then to store this location when sent along with a name, phone number and optional message. These details then are plotted on a map only visible to Pet Owners for security reasons. After I completed these functionalities the three main requirements were implemented into the project. I could now begin to work on the front-end of the application, making the GUI look modern and professional.

I also continued to work on the technical report, created test scripts to validate requirements and ensure the quality of the application was up to standard. I met with Sam more than once a week to keep him updated on my progress and to also talk through the upcoming final presentation which is coming in fast! This is the last time I will document this project, final year of my degree is now almost complete. It went by so fast!

I would personally like to thank my project supervisor Sam Cogan for been so supportive throughout my final year, his advice was so helpful. I would also like to thank my girlfriend Shanice Carey for been there for me when I needed someone to keep me focused on my goals with this project.

It's been a rollercoaster of a year but it's a good feeling to have it all completed now and in the coming months begin a career in Software Testing. Thanks for the memories NCI!

My Reflection

It's been a tough month but getting through the documentation and finalising the code was a good experience personally.

Intended Changes

None

Supervisor Meeting

We discussed certain parts of the upcoming presentation and how I plan on laying out my showcase as well as discussing tests I carried out on the project which I documented in the technical report..

Items discussed: Presentation, Documentation, and Showcase

References

Apache Cordova. (2016). *Cordova.* Available: https://cordova.apache.org/. Last accessed May 4th 2016.

Arden Moore. (2014). *what are the Chances of Finding My Lost Pet.* Available: http://www.vetstreet.com/our-pet-experts/what-are-the-chances-of-finding-my-lost-pet-aspca-survey-has-the-answers. Last accessed September 15th 2015.

Atmosphere. (2016). *Autopublish.* Available: https://atmospherejs.com/meteor/autopublish. Last accessed May 2nd 2016.

EE Times. (2012). *Android Architecture.* Available: http://www.eetimes.com/document.asp?doc_id=1279698. Last accessed April 21st 2016.

Francesco Corazza. (2014). *Meteor.js Architecture.* Available: http://www.slideshare.net/francescocorazza/meteor-js-35071602. Last accessed January 31st 2016.

Insecure. (2016). *Insecure.* Available: https://atmospherejs.com/meteor/insecure. Last accessed May 2nd 2016

Ionic. (2016). *Interactive User Interfaces.* Available: http://ionicframework.com/. Last accessed May 1st 2016.

ISPCA. (2016). *Why Micro-chipping your dog is a no-brainer.* Available: http://www.ispca.ie/blog/detail/why_micro_chipping_your_dog_is_a_no_brainer. Last accessed October 5th 2016.

Kadira. (2016). *Performance Monitoring.* Available: https://kadira.io/. Last accessed May 6th 2016.

Krzysztof Wyrykowski. (2016). *Hybrid mobile applications.* Available: http://blog.primemodule.com/hybrid-mobile-applications-mvp/. Last accessed May 1st 2016.

Meteor. (2016). *Real time applications.* Available: https://www.meteor.com/. Last accessed May 6th 2016.

MongoDB. (2015). *Meteor: Build iOS and Android Apps that are a Delight to Use.* Available: https://www.mongodb.com/blog/post/meteor-build-ios-and-android-apps-are-delight-use. Last accessed April 31st 2016.

Protractor. (2016). *End-to-end Testing.* Available: http://www.protractortest.org/#/. Last accessed May 6th 2016.

Venture Pact. (2016). *8 High Performance Apps that are Hybrid.* Available: http://blog.venturepact.com/8-high-performance-apps-you-never-knew-were-hybrid/. Last accessed 20th April 2016.