11th May 2016

GroomK9.com

A Dog Grooming Management System

Business and Technical Report

National College of Ireland

Alan Rice, X11105038, Alan.Rice@students.ncirl.ie

Declaration Cover Sheet for Project Submission

SECTION 1 *Student to complete*

Name:	Alan Rice
Student ID:	X111105038
Supervisor:	Semester 1 - Mr. Mikhail Timofeev Semester 2 – Mr. Ralf Bierig

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.

Complete the sections above and attach it to the front of one of the copies of your assignment,

What constitutes plagiarism or cheating?

The following is extracted from the college's formal statement on plagiarism as quoted in the Student Handbooks. References to "assignments" should be taken to include any piece of work submitted for assessment.

Paraphrasing refers to taking the ideas, words or work of another, putting it into your own words and crediting the source. This is acceptable academic practice provided you ensure that credit is given to the author. Plagiarism refers to copying the ideas and work of another and misrepresenting it as your own. This is completely unacceptable and is prohibited in all academic institutions. It is a serious offence and may result in a fail grade and/or disciplinary action. All sources that you use in your writing must be acknowledged and included in the reference or bibliography section. If a particular piece of writing proves difficult to paraphrase, or you want to include it in its original form, it must be enclosed in quotation marks and credit given to the author.

When referring to the work of another author within the text of your project you must give the author's surname and the date the work was published. Full details for each source must then be given in the bibliography at the end of the project

Penalties for Plagiarism

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. Where the Disciplinary Committee makes a finding that there has been plagiarism, the Disciplinary Committee may recommend

- that a student's marks shall be reduced
- that the student be deemed not to have passed the assignment
- that other forms of assessment undertaken in that academic year by the same student be declared void
- that other examinations sat by the same student at the same sitting be declared void

Further penalties are also possible including

- suspending a student college for a specified time,
- expelling a student from college,
- prohibiting a student from sitting any examination or assessment.,
- the imposition of a fine and
- the requirement that a student to attend additional or other lectures or courses or undertake additional academic work.

1 Table of Contents

1. Executive Summary

According to Time magazine Americans are spending more money than ever on their pets. They forked out a record $55.7 billion in 2013 of which Dog Grooming accounted for $5 billion. It's a similar situation in Europe with the European Pet Food Industry Federation (FEDIAF) estimating the Pet Market to be €24 billion in 2010. Both regions have also witnessed a growth rate of 4.8% per annum in recent years. Despite this rapid growth many Dog Grooming businesses do not use modern technologies to manage their business.

GroomK9 is a software solution which aims to capitalize on this profitable and rapidly growing market by helping Dog Groomers efficiently manage the day to day running of their business including their growing demand for information management, appointment management, and social media marketing opportunities.

This new software is provided as a Software as a Service (SaaS) solution which allows to easy deployment and adoption by existing Groomers as they will not be burdened with any upfront costs for hardware or ongoing maintenance. The software was developed with flexibility in mind with the backend service offering a full RESTFul API. This flexibility will provide the Groomer with many options and reduce any barriers to using the software to help their business.

The initial client made available is a Web Application utilising AngularJS and the Google Material Design spec framework which will provide an emerging standard. These two technologies combined provide a graphical user interface which is responsive in terms of the devices and screen size but also in how quickly different sections will be displayed.

This solution was created after working closely with a very successful and growing Dog Grooming business in Co. Kildare, Ireland. Capitalising on first-hand experience to help improve the day to day management of their business.

2 Background

According to Time magazine Americans are spending more money than ever on their pets. They forked out a record $55.7 billion in 2013—or about $10 billion more than Germany's entire defence budget. This figure is based on American Pet Products Association (APPA) annual report. Dog Grooming makes up $5 billion of this market and is currently growing at 4.8% per annum. In Europe we can see similar sized markets with the European Pet Food Industry Federation (FEDIAF) estimating the Pet Market to be €24 billion in 2010.

(Brough, 2015)

Professional dog grooming has been around since the 1960s but often worked with very limited equipment with groomers being isolated from other groomers. In the last 50 years we have seen a number of key changes in this market. The business has become more professional with set training qualifications, groomer competitions and groomer seminars. There has also been a cultural change in how animals have become more important as pets with over 37% of Americans having a dog (67% having any pet), 36% dog ownership in Irish Homes, and 22% in Britain.

Dog Grooming is now seen more as a skilled trade which has to be learned and crafted over years. This explains why the value of this market is quiet large but at the same time tends to be made up of many small business owners. These groomers typically start out as sole traders who organically grow their business over time while they build a client base and learn their craft.

Despite this growth in the business and a key change in culture which is moving Groomers to use modern technology in terms of their technique and tools, the management of their business tends to still be antiquated. Many of which them tend to use plain old pen and paper ad-hoc management, while some might venture as far as an excel file on their computers. As with all young professions they will eventually become more professional, and that will require them to use system to manage their Customers, their staff and time, marketing (particularly social media elements) and accounts.

In terms of market competition there are companies currently providing software that cater to groomers. However, the selection is poor given the market size, many are overpriced, and most of them only offer basic functionality. The software services in place are predominantly desktop based although there is a handful of web software as a service products becoming available.

Given the tight margins of start-up groomers many of the products on offer now are overly expensive and a fixed price. In an industry where a groomer can command a higher wage over time the provision of software should be flexible as their business grow i.e. have a very low setup and running cost with the ability to grow as the business does.

This is why a SOA Cloud based approach is ideal for this industry. It will reduce barriers to entry for Groomers in terms of initial cost by leveraging the cloud. It will do this by allowing the groomer to use a thin client application which does not require any investment in expensive hardware (servers, powerful computers, or the cost to maintain them). It will provide a very fast setup and rather than the groomer having to outlay a large investment they can pay for it as a service on a low ongoing basis with flexibility to grow easily if needed. The improved management of their business will lead to increased efficiency by reducing time lost in booking, improve customer relations by having information on hand and social media improvements, provide key insights into which and what type of Pets are the most profitable. It will help them manage their staff and accounts.

This Grooming Management App will be able to provide superior functionality and features than existing marketing providers with a price structure which is more flexible than anyone else on the current market.

3 Aims

The objective of this project is to create a complete Dog Grooming Management System which will help business owners manage the day to day activities of their business these core objectives include:

3.1 Customer Management

- Maintain Contact and Address information
    - Provide inquiry options to show previous visits, billing, and feedback.
- Animal Management
    - Ability to record multiple dogs against their owners including all standard information e.g. name, breed, or any notes.
    - Inquiry Options
    - Record details preferred cuts, animal personality/behaviour, and potential trouble or sore areas on the dog.
- Appointment Management
    - Common calendar functionality
    - Allow owner view and manage appointments linking to particular jobs and dogs
- Billing
    - Ability to record all grooming activities and charges
    - Create PDF reports for customers
    - Record and maintain default services in the system for easy entry
    - Data Analysis / Reporting
- Provide data analysis and statistics
    - Provide export of information through reports
- Manage Photo uploads both for Clients and their Pets

3.2 Provision of Software Objectives

Provide a robust software system which can be incrementally updated and improved over time. Have a separation of concerns by using a SOA approach by dividing up the system into manageable components and to decouple front end interfaces from business logic. Facilitate this through the provision of JSON or XML web service API to allow for easier integration and support for multiple device and multiple platforms.

4 Technologies

4.1 Data Layer

The data layer utilizes a PostgreSQL database. PostgreSQL is an enterprise object relational database management system. Anecdotal evidence has shown Postgres to give better performance than MySQL the other main database consideration. It is open source and is more compliant with the SQL:2011 standard than MySQL. While MySQL is hugely popular and arguably has better tooling options (in particular their excellent workbench management system), Postgres performance, licencing, and tight integration with Heroku (the top proposed hosting options used with this project) made it the best choice for this project.

It also has a unique feature in terms of how it can manage Multi-Tennent setups that offers the best of both worlds and is discussed later in this document in the Design and Architecture: Database Tier section (6.3)

4.2 Business Layer

This project uses Ruby with the Rails framework as the primary language for the business logic layer.

Ruby on Rails (ROR) is responsible for creating the API and web services. The API is RESTful so that it can easily be consumed by different front end clients and integrate with other services.

Other languages were considered such as Java EE, .net, Node.js, and ColdFusion. ROR was chosen as due to its popularity has incredible documentation and tutorials (both written and video) and a thriving community which will help throughout this project. There are some real nice commercial infrastructures available such as Heroku which will enable quick prototyping with a minimum of setup time and cost. Due to the popularity of this option the tooling options can improve development quality and time. The Rails framework is also appealing due to its convention over configuration which avoided having large configuration files. The other languages discussed are also very capable and some share similar features, but as a complete package ROR seemed more appealing.

Due to the frameworks popularity many problems have already been overcome and there will be no shortage of experienced developers should this project become commercially successful and need to expand.

4.3 Presentation Layer

The first presentation layer is in the form of a web frontend which is viewable by any browser. As this project uses a RESTful API other native clients can be developed easily at a later date e.g. native iOS and Android.

To keep the web front end fast and modular this project uses Angular.JS which helps decouple the presentation (HTML) files from directly binding to JavaScript events. Using AngularJS also made testing easier due to its modular nature.

A responsive design was completed using Google’s Material Design specification and their own framework. This decision was made due to its fresh design choices and optimisation for mobile devices. It also offers a break and an opportunity to stand out from the bootstrap framework, which in its default form has become quite pervasive for websites and web applications.

5 Requirements Specification

To obtain a list of requirements I closely liaised with a professional dog groomer in Kildare spending two different days on their premises to observe the day to day running of the operations.

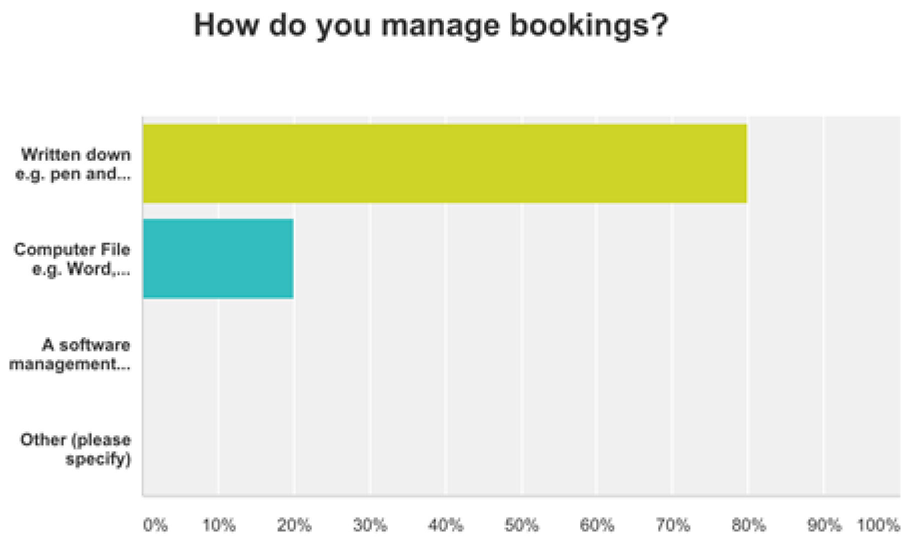
During these days we discussed how they manage their business and what they need to make their life easier. A lot of options came up but one of the most important was the requirement to have access on the go as clients can call outside normal working hours and it can be awkward to check for availability. Typically, this was done by checking one of two diaries. Following this days, a list of functional requirements was created below.

In addition to the groomer completed a survey and asked a number of their peers to do so as well. This provided a small sample of information but was enough to get a first draft of what was needed for this project.

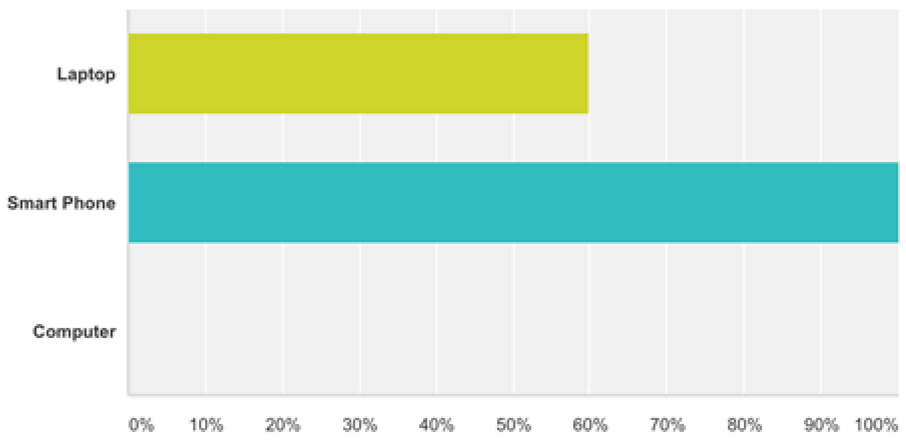
5.1 Survey Results

The survey revealed a number of key areas which included:

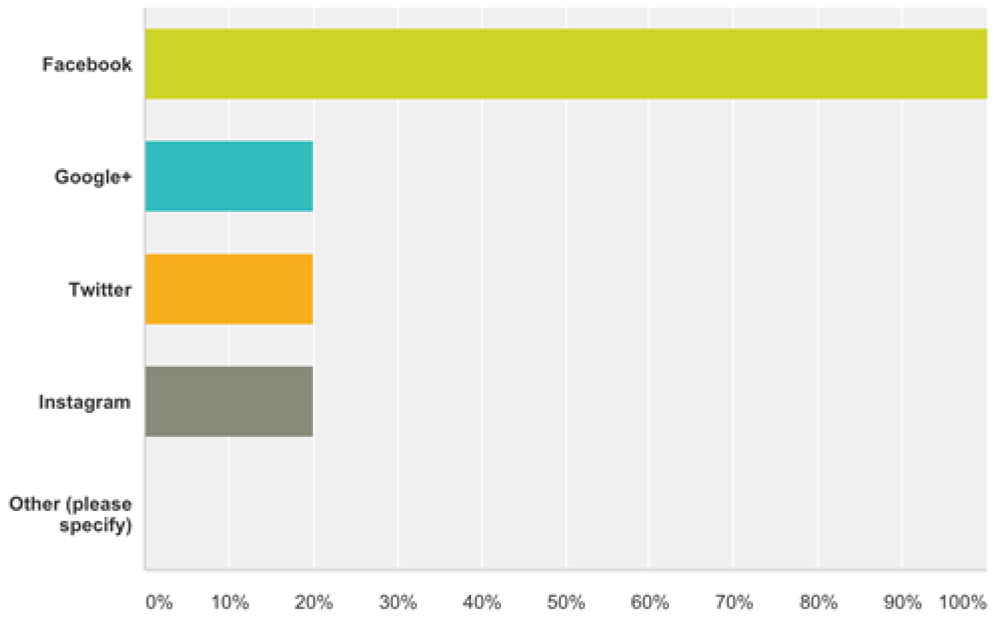
- 100% of the surveyed Groomers did not currently use a dedicated software system to manage their bookings.
- More than 90% were somewhat interested or very interested in using a software system to manage their business
- 100% of surveyed Groomers owned a smart phone and used Facebook on a daily basis



Which of these devices do you own (tick as many as appropriate)?



Which of the following social networking websites do you currently have an account with? (Check all that apply)



5.2 Functional Requirements

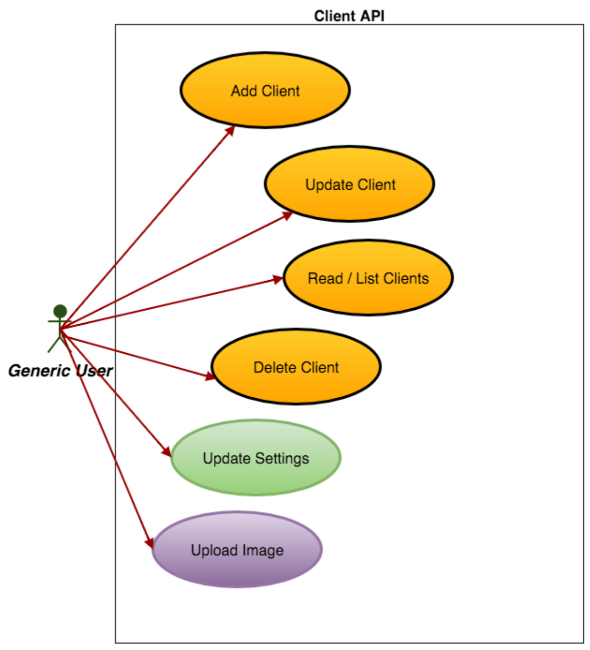
As the main front end will be provided by different GUIs to the user for the purpose of this section the ‘client’ shall refer to any application client which provides the GUI to the user. This will include the Web Application, Native mobile device of whatever the user must directly interact with to use the service.

Please note that the application contains a section for the user’s Clients. To avoid confusion in the following sections the User’s Clients and resources will be referred to as Customers to differentiate them from a GUI client.

The list of these requirements are listed directly below with finer details in the following pages:

- The system will allow for the creation of a user account
- The system will allow authentication using a unique identifier (email or username) and a password. In exchange it should provide a token for subsequent access.
- The system will provide authorisation to all restricted sections using the unique identifier and a provided token
- The system will allow authenticated users to perform add, update, create, and delete (CRUD) operations on the following resources:
    - Customers
    - Pets
    - Appointments
    - Preferences
- The system will allow authenticated users perform CRUD operations for Customer and Pet associations
- The system will allow authenticated users upload, retrieve, and delete images

5.3 Use Case Diagram



5.3.1 Use Case Index

Use Case ID	Use Case Name	Scope	Complexity	Priority
1	User Registration	In	Low	1
2	User Authentication	In	High	1
3	User Authorisation	In	Medium	1
4	Create Customers	In	Medium	1
5	Read Customers	In	Medium	1
6	Update Customers	In	Medium	1
7	Delete Customers	In	Medium	1
8	Create Pets	In	Low	1
9	Read Pets	In	Low	1
10	Update Pets	In	Low	1
11	Delete Pets	In	Low	1
12	Create Appointments	In	High	1
13	Read Appointments	In	High	1

14	Update Appointments	In	High	1
15	Delete Appointments	In	High	1
16	Upload and Manage Images	In	High	2

5.3.2 Requirement 1: User Registration

5.3.2.1 Description & Priority

This use case describes how the system will allow the user register with the system

5.3.2.2 Scope

The scope of this use case is to allow the user to register an account

5.3.2.3 Description

This use case describes how the user will register using a Customer

5.3.2.4 Flow Description

5.3.2.4.1 Precondition

The system is ready and the user is not signed in

5.3.2.4.2 Activation

The user fills out required fields which include a valid email address and password subject to minimum requirements

5.3.2.4.3 Main Flow

1. The client displays a registration form
2. The user enters the required details
3. The system checks that there is not already an account matching that email
4. The system validates the required fields
5. The system creates a new account and returns an authorisation token to the client

5.3.2.4.4 Alternate Flow

A1 Failure to provide required items: A valid email and valid password

1. The system sends a warning message to the client which is then displayed to the user informing them of missing or invalid details
2. The client prompts the user to enter the correct details and resumes from point 3 of the main flow

5.3.2.4.5 Termination

1. The system provides an authorisation token to the client
2. The client displays the home screen to the user

5.3.3 Requirement 2: User Authentication

5.3.3.1 Description & Priority

This use case describes the process by which the user authenticates themselves using an email address and password at the sign in resource

5.3.3.2 Scope

The scope of this use case is to allow a user authenticate themselves

5.3.3.3 Description

This use case describes the process of how a user authenticates themselves and receives an authorization token in place of their email and password

5.3.3.4 Flow Description

5.3.3.4.1 Precondition

The system is ready and the user does not have a valid authorization token

5.3.3.4.2 Activation

The user signs in using their email and password

5.3.3.4.3 Main Flow

1. The client sends a HTTP POST request to the source which includes the user's email and password
2. A HTTP response is returned by the server which contains the user's email and token

5.3.3.4.4 Alternate Flow

A1 Failure to a valid authorisation token

1. The authorisation token provided by the client is expired or invalid
2. The server will respond with a 401 Unauthorised message

A2 Provide an invalid end point

1. The client requests an invalid email and password combination
2. The server will respond with a 401 Unauthorized HTTP header response

5.3.3.4.5 Termination

1. The system returns an authorization token
2. The client requests the home page with the authorization token

5.3.4 Requirement 3: User Authorisation

5.3.4.1 Description & Priority

As an API provides no session every request must be sent with authentication as it is stateless.

5.3.4.2 Scope

The scope of this use case is to allow a user access restricted access points in the API

5.3.4.3 Description

This use case describes how the user will request access to different resources

5.3.4.4 Flow Description

5.3.4.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.4.4.2 Activation

The user requests access to resource via the client

5.3.4.4.3 Main Flow

3. The client sends a HTTP request to the source
4. A HTTP response is returned by the server. This will contain a JSON response for any GET requests but headers status messages are only required for POST, PUT, and DELETE operations

5.3.4.4.4 Alternate Flow

A1 Failure to a valid authorisation token

3. The authorisation token provided by the client is expired or invalid
4. The server will respond with a 401 Unauthorised message

A2 Provide an invalid end point

3. The client requests an invalid resource address
4. The server will respond with a 404 not found HTTP Header

5.3.4.4.5 Termination

3. The system fulfils the requested HTTP operation on the resource
4. The client displays the result in the GUI

5.3.5 Requirement 4: Customers Create

5.3.5.1 Description & Priority

This use case describes the process by which a user performs Create operations on the Customers resource

5.3.5.2 Scope

The scope of this use case is to allow a user to perform Create operation on the Customers resource

5.3.5.3 Description

This use case describes how the user will perform Create operation on the Customers resource

5.3.5.4 Flow Description

5.3.5.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.5.4.2 Activation

The user submits a form with the Customer's information

5.3.5.4.3 Main Flow

1. The application client sends a HTTP POST request to the server containing the Customer's information
5. A HTTP response is returned by the server. This will contain a JSON which will include an Identification key for the newly created client

5.3.5.4.4 Alternate Flow

A1 Failure to a valid authorisation token

1. The authorisation token provided by the client is expired or invalid
2. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Details

1. The user does not provide valid or required information
2. The server will respond with a not found HTTP JSON response alerting the user to the error

5.3.5.4.5 Termination

1. The system presents the new Customer on screen

5.3.6 Requirement 5: Customer Read

5.3.6.1 Description & Priority

This use case describes the process by which a user performs Read operations on the Customers resource

5.3.6.2 Scope

The scope of this use case is to allow a user to perform Read operation on the Customers resource

5.3.6.3 Description

This use case describes how the user will perform Read operation on the Customers resource

5.3.6.4 Flow Description

5.3.6.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.6.4.2 Activation

The user requests to view a Customer's record

5.3.6.4.3 Main Flow

1. The application client sends a HTTP GET request to the server containing Customers URI which contains the Customers Unique Identifier
2. A HTTP response is returned by the server. This will contain a JSON which will include the Customers information

5.3.6.4.4 Alternate Flow

A1 Failure to a valid authorisation token

1. The authorisation token provided by the client is expired or invalid
2. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Identifier Specified

1. The user does not provide a valid Customer identifier
2. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.6.4.5 Termination

The system presents the requested client on screen

5.3.7 Requirement 6: Customer Update

5.3.7.1 Description & Priority

This use case describes the process by which a user performs UPDATE operation on the Customer's resource

5.3.7.2 Scope

The scope of this use case is to allow a user to perform UPDATE operation on the Customer's resource

5.3.7.3 Description

This use case describes how the user will perform UPDATE operation on the Customer's resource

5.3.7.4 Flow Description

5.3.7.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.7.4.2 Activation
The user requests to update a Customer's record

5.3.7.4.3 Main Flow
1. The application client sends a HTTP PUT request to the server containing Customer's unique identifier and updated information
2. A HTTP response is returned by the server with the updated Customer information and a 200 HTTP header

5.3.7.4.4 Alternate Flow
A1 Failure to a valid authorisation token

1. The authorisation token provided by the client is expired or invalid
2. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Identifier Specified

1. The user does not provide a valid Customer identifier
2. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.7.4.5 Termination
The system presents the requested client on screen with an updated notification

5.3.8 Requirement 7: Customer Delete

5.3.8.1 Description & Priority
This use case describes the process by which a user performs a DELETE operation on the Customer's resource

5.3.8.2 Scope
The scope of this use case is to allow a user to perform a DELETE operation on the Customer's resource

5.3.8.3 Description
This use case describes how the user will perform DELETE operation on the Customer's resource

5.3.8.4 Flow Description
5.3.8.4.1 Precondition
The system is ready and the user has received an a valid authorisation token

5.3.8.4.2 Activation
The user requests to delete a Customer record

5.3.8.4.3 Main Flow

1. The application client sends a HTTP DELETE request to the server using the Customer's resource end point
2. A HTTP response is returned by the server with the updated Customer's information and a 204 No Content HTTP header

5.3.8.4.4 Alternate Flow

A1 Failure to a valid authorisation token

3. The authorisation token provided by the client is expired or invalid
4. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Identifier Specified

3. The user does not provide a valid client identifier as part of the resource URI
4. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.8.4.5 Termination

The system returns the user to the Customers index section

5.3.9 Requirement 8: Pet Create

5.3.9.1 Description & Priority

This use case describes the process by which a user performs Create operations on the Pets resource

5.3.9.2 Scope

The scope of this use case is to allow a user to perform Create operation on the Pets resource

5.3.9.3 Description

This use case describes how the user will perform Create operation on the Pets resource

5.3.9.4 Flow Description

5.3.9.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.9.4.2 Activation

The user submits a form with the Pet's information

5.3.9.4.3 Main Flow

2. The application client sends a HTTP POST request to the server containing the Pet's information
6. A HTTP response is returned by the server. This will contain a JSON which will include an Identification key for the newly created client

5.3.9.4.4 Alternate Flow

A1 Failure to a valid authorisation token

3. The authorisation token provided by the client is expired or invalid
4. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Client Details

3. The user does not provide valid or required information
4. The server will respond with a not found HTTP JSON response alerting the user to the error

5.3.9.4.5 Termination

2. The system presents the new Pet on screen

5.3.10 Requirement 9: Pet Read

5.3.10.1 Description & Priority

This use case describes the process by which a user performs Read operations on the Pets resource

5.3.10.2 Scope

The scope of this use case is to allow a user to perform Read operation on the Pets resource

5.3.10.3 Description

This use case describes how the user will perform Read operation on the Pets resource

5.3.10.4 Flow Description

5.3.10.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.10.4.2 Activation

The user requests to view a Pet's record

5.3.10.4.3 Main Flow

3. The application client sends a HTTP GET request to the server containing Pets URI which contains the Pets Unique Identifier
4. A HTTP response is returned by the server. This will contain a JSON which will include the Pets information

5.3.10.4.4 Alternate Flow

A1 Failure to a valid authorisation token

3. The authorisation token provided by the client is expired or invalid
4. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Identifier Specified

3. The user does not provide a valid Pet identifier
4. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.10.4.5 Termination

The system presents the requested Pet on screen

5.3.11 Requirement 10: Pet Update

5.3.11.1 Description & Priority

This use case describes the process by which a user performs UPDATE operation on the Pet's resource

5.3.11.2 Scope

The scope of this use case is to allow a user to perform UPDATE operation on the Pet's resource

5.3.11.3 Description

This use case describes how the user will perform UPDATE operation on the Pet's resource

5.3.11.4 Flow Description

5.3.11.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.11.4.2 Activation

The user requests to update a Pet's record

5.3.11.4.3 Main Flow

3. The application client sends a HTTP PUT request to the server containing Pet's unique identifier and updated information
4. A HTTP response is returned by the server with the updated Pet information and a 200 HTTP header

5.3.11.4.4 Alternate Flow

A1 Failure to a valid authorisation token

5. The authorisation token provided by the client is expired or invalid
6. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Pets Identifier Specified

3. The user does not provide a valid Pet identifier

4. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.11.4.5 Termination

The system presents the requested client on screen with an updated notification

5.3.12 Requirement 11: Pet Delete

5.3.12.1 Description & Priority

This use case describes the process by which a user performs a DELETE operation on the Pet's resource

5.3.12.2 Scope

The scope of this use case is to allow a user to perform a DELETE operation on the Pet's resource

5.3.12.3 Description

This use case describes how the user will perform DELETE operation on the Pet's resource

5.3.12.4 Flow Description

5.3.12.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.12.4.2 Activation

The user requests to delete a Pet record

5.3.12.4.3 Main Flow

5. The application client sends a HTTP DELETE request to the server using the Pet's resource end point
6. A HTTP response is returned by the server with the updated Pet's information and a 204 No Content HTTP header

5.3.12.4.4 Alternate Flow

A1 Failure to a valid authorisation token

7. The authorisation token provided by the client is expired or invalid
8. The server will respond with a 401 Unauthorised message

A2 Invalid Pet Identifier Specified

7. The user does not provide a valid client identifier as part of the resource URI
8. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.12.4.5 Termination

The system returns the user to the Pets index section

5.3.13 Requirement 12: Appointment Create

5.3.13.1 Description & Priority

This use case describes the process by which a user performs Create operations on the Appointments resource

5.3.13.2 Scope

The scope of this use case is to allow a user to perform Create operation on the Appointments resource

5.3.13.3 Description

This use case describes how the user will perform Create operation on the Appointments resource

5.3.13.4 Flow Description

5.3.13.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.13.4.2 Activation

The user submits a form with the Appointment's information

5.3.13.4.3 Main Flow

3. The application client sends a HTTP POST request to the server containing the Appointment's information
7. A HTTP response is returned by the server. This will contain a JSON which will include an Identification key for the newly created client

5.3.13.4.4 Alternate Flow

A1 Failure to a valid authorisation token

5. The authorisation token provided by the client is expired or invalid
6. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Client Details

5. The user does not provide valid or required information
6. The server will respond with a not found HTTP JSON response alerting the user to the error

5.3.13.4.5 Termination

3. The system presents the new Appointment on screen

5.3.14 Requirement 13: Appointment Read

5.3.14.1 Description & Priority

This use case describes the process by which a user performs Read operations on the Appointments resource

5.3.14.2 Scope

The scope of this use case is to allow a user to perform Read operation on the Appointments resource

5.3.14.3 Description

This use case describes how the user will perform Read operation on the Appointments resource

5.3.14.4 Flow Description

5.3.14.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.14.4.2 Activation

The user requests to view an Appointment's record

5.3.14.4.3 Main Flow

5. The application client sends a HTTP GET request to the server containing Appointments URI which contains the Appointments Unique Identifier
6. A HTTP response is returned by the server. This will contain a JSON which will include the Appointments information

5.3.14.4.4 Alternate Flow

A1 Failure to a valid authorisation token

5. The authorisation token provided by the client is expired or invalid
6. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Customer Identifier Specified

5. The user does not provide a valid Appointment identifier
6. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.14.4.5 Termination

The system presents the requested Appointment on screen

5.3.15 Requirement 14: Appointment Update

5.3.15.1 Description & Priority
This use case describes the process by which a user performs UPDATE operation on the Appointment's resource

5.3.15.2 Scope
The scope of this use case is to allow a user to perform UPDATE operation on the Appointment's resource

5.3.15.3 Description
This use case describes how the user will perform UPDATE operation on the Appointment's resource

5.3.15.4 Flow Description
5.3.15.4.1 Precondition
The system is ready and the user has received an a valid authorisation token

5.3.15.4.2 Activation
The user requests to update an Appointment's record

5.3.15.4.3 Main Flow
5. The application client sends a HTTP PUT request to the server containing Appointment's unique identifier and updated information
6. A HTTP response is returned by the server with the updated Appointment information and a 200 HTTP header

5.3.15.4.4 Alternate Flow
A1 Failure to a valid authorisation token

9. The authorisation token provided by the client is expired or invalid
10. The server will respond with a 401 Unauthorised message

A2 Invalid Groomer Appointments Identifier Specified

5. The user does not provide a valid Appointment identifier
6. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.15.4.5 Termination
The system presents the requested client on screen with an updated notification

5.3.16 Requirement 15: Appointment Delete

5.3.16.1 Description & Priority
This use case describes the process by which a user performs a DELETE operation on the Appointment's resource

5.3.16.2 Scope
The scope of this use case is to allow a user to perform a DELETE operation on the Appointment's resource

5.3.16.3 Description
This use case describes how the user will perform DELETE operation on the Appointment's resource

5.3.16.4 Flow Description
5.3.16.4.1 Precondition
The system is ready and the user has received an a valid authorisation token

5.3.16.4.2 Activation
The user requests to delete an Appointment record

5.3.16.4.3 Main Flow
9. The application client sends a HTTP DELETE request to the server using the Appointment's resource end point
10. A HTTP response is returned by the server with the updated Appointment's information and a 204 No Content HTTP header

5.3.16.4.4 Alternate Flow
A1 Failure to a valid authorisation token

11. The authorisation token provided by the client is expired or invalid
12. The server will respond with a 401 Unauthorised message

A2 Invalid Appointment Identifier Specified

11. The user does not provide a valid client identifier as part of the resource URI
12. The server will respond with a 404 not found HTTP JSON response alerting the user to the error

5.3.16.4.5 Termination
The system returns the user to the Appointments index section

5.3.17 Requirement 16: Media Upload

5.3.17.1 Description & Priority
This use case describes the process by which a user performs an Upload operation

5.3.17.2 Scope
The scope of this use case is to allow a user to perform an Upload operation

5.3.17.3 Description
This use case describes how the user will perform Upload operation

5.3.17.4 Flow Description

5.3.17.4.1 Precondition

The system is ready and the user has received an a valid authorisation token

5.3.17.4.2 Activation

The user requests to upload an image

5.3.17.4.3 Main Flow

13. The application client sends a HTTP POST request to the server using the Upload end point
14. The server relays the post request with credentials to image hosting API (Cloudinary or other)
15. A HTTP response is returned by the server with the updated Upload information and a 200 HTTP Header with option JSON response with upload details

5.3.17.4.4 Alternate Flow

A1 Failure to a valid authorisation token

13. The authorisation token provided by the client is expired or invalid
14. The server will respond with a 401 Unauthorised message

5.3.17.4.5 Termination

The system notifies the user of the image uploads success

5.4 Data Requirements

- User data must be isolated from each other without risk of leaking user data between different users.
- Data must be easily backed up without impacting other users.
- All inputted input data must be validated at the server and database level using appropriate data types for relevant fields e.g. longs for numbers, decimal for decimal, varchar for text, bit for Boolean values etc.
- Foreign Keys should be established to ensure the consistency of the data.

5.5 User Requirements

Users will require an internet enabled device is capable of loading a modern HTML5, JavaScript, and CSS3 enabled website.

5.6 Environmental Requirements

- System capable or running Ruby (Windows, Mac, Linux based PC/Laptop)
- Mac and Developer Licence if for deploying a Cordova app. Also required for a native application
- Linux hosting provider
  This may include Amazon AWS, PaaS such as Heroku, or a virtual environment using Docker hosted with web host provider such as Digital Ocean or Linode

5.7 Usability Requirements

A great deal of focus was placed on increasing the usability of this application as the user's aversion to newer technology is a barrier which had to be overcome.

The GUI was developed using strict guidelines from Googles Material design and the web client opted to use Googles Own Material Design web framework to help meet these requirements.

5.7.1 Understand-ability

The main interface has been kept simple and uncluttered with clearly defined menu items for all the basic actions. Labels are used where space allows and where there is limited space a simple icon is displayed instead with a tooltip for extra help.

5.7.2 Learnability

Online documentation will be provided for the user using a wiki system so that it easy to update which should reduce the time and hurdles. Online video tutorials on how to use the system will also provide an alternative approach for users who prefer to watch rather than read instructions. Finally training support will also be available in person for users who wish to avail of it.

5.7.3 Operability

The use of the framework and colour scheme provide a simple and consistent user experience between all sections in the app. All search pages have a common feel with controls such as buttons and inputs in the same position.

HTML5 tags have been used to help users enter valid data such as required fields. Date picker controls are also provided to help users enter valid data quickly without issues. These steps will help reduce errors while providing informative information to the user why something is not valid.

5.7.4 Attractiveness

The Google Material Design specification provides to a standard style guide which details what colours can be used and how controls are positioned and spaced out. This helps provide a pleasant user experience without being too flashy to busy.

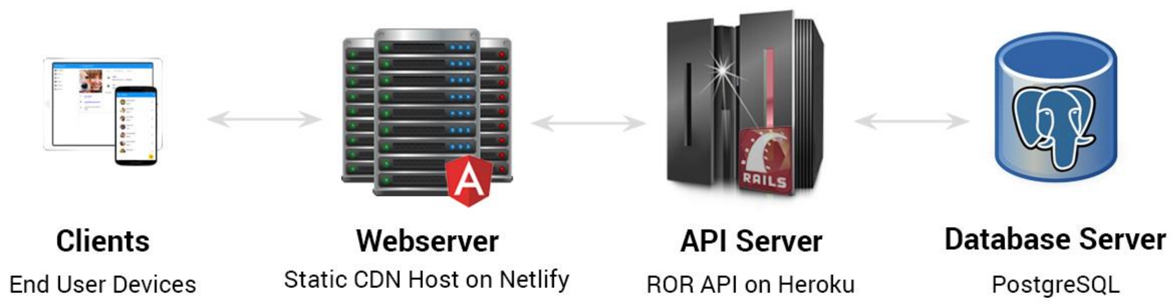
5.8 Research Requirements

Mainly for research and references:

- Book: Web Development with Rails 4 (Pragmatic Programmers)
- Code School – Rails, and Rails API Tutorials
- Plural Sight – Angular JS + Bootstrap Tutorials

6 Design and Architecture

The system architecture for this project utilizes an n-tier design pattern.



6.1 Presentation Tiers (Clients + Webserver)

The main presentation tier is an AngularJS HTML5 and JavaScript application. This tier can easily be deployed to a number of different clients including Desktop via a website and to mobile phones and tablets via either a hosted website of contained within a Phone Gap application.

6.2 Business Logic Tier (API Server)

The business logic layer is fulfilled by a Ruby powered JSON API, which uses the Rails framework. It is responsible for all business logic in creating, updating, reading, and deleting information in the application.

6.3 Database Tier (PostgreSQL)

All data is stored in the database server utilizing a PostgreSQL database. Multi-tenancy is provided by feature unique to PostgreSQL called Postgres Schemas. This is not to be confused with the generic database term schemas but is in this case is used to hold multiple copies of the same tables within the one database not dissimilar to name scoping in other languages. The main advantage to this is that all users' data is kept separate from each other while using the same names for database tables and fields. It provides many of the benefits of having separate databases per user without the additional overhead.

6.4 Hardware Architecture

From market research we expect that the demands on hardware architecture to be very low. The load on the server will be initially will be low and undemanding, typically only requiring simple input of data with light processing.

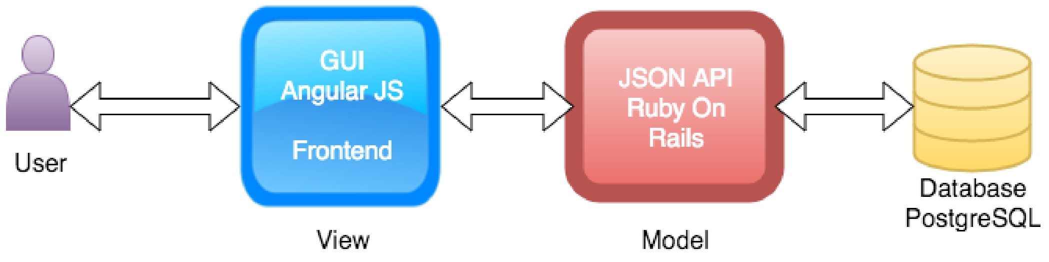
This project will be hosted on Heroku (Heroku, 2015) cloud hosting which provides a Platform as Service (PAAS) architecture. This was chosen as it outsources much of the administration and allows more attention to be given to the development of the project. As it is a virtualized service it will also allow the hardware to be scaled up easily as the project and hardware requirements grow. Other alternatives considered were Amazon Web Services (AWS) and Digital Ocean. Heroku was preferred due to their simplified deployment model and their simple pricing scheme.

6.5 Software Architecture

This project will use the Ruby on Rails framework, which includes Active Record. This is an ORM, which allows for concise code to be written to interact with the database and manage structured relational data. The framework also provides a powerful routing feature, which is very useful in creation, an API and allows for following of best practices.

The initial client will be an AngularJS application. This forms the main View in MVC of the application. Within this view AngularJS follows a MVVM approach, which allows for a structured layout of code. It will allow code to be modularized while handling dependencies as one of its key features. Two-way data binding is possible and it uses the model to determine what is shown in the Angular view.

The models used in the backend will be mapped directly to the database design, which is discussed later in this document but includes models for Appointments, Clients, Pets, and the relationships between each item.



6.6 Standards

6.6.1 RESTFUL JSON API

The main model of this application will be a JSON API in a restful fashion. This will be created using Ruby and the Rails framework.

In terms of standards for this for this JSON API this project followed the JSONAPI.org specifications. This is a developing standard created in 2013 (Katz, 04).

6.6.2 Graphical User Interface (GUI)

In order to create a modern and consistent interface this project will follow the Google Material Specification and Guidelines (Google, 2015).

Figure 1 Google Material Paper

6.6.3 AngularJS – Style Guide

AngularJS does not force the developer to structure their code in any particular way. This is very flexible but as a first time user to the framework you are constantly asking what's the best way to do this, or how and where should modules be placed. With many schools of thought this project used the 'John Papa' style guide (Anon., 2016) which is one of the most popular and widely used approaches. It has in depth documentation about how to structure AngularJS code to ensure easy maintenance and growth. Where possible this project followed these guidelines.

6.6.4 Security Architecture

Steps were taken to provide a secure application. In terms of the API a number of key areas were addressed to follow official Ruby Security (Anon., 2016) guidelines and OWASP's RESTFul security Cheat Sheet (Anon., 2016)

6.6.4.1 End to End Encryption

As this project is composed of two applications (API Backend, HTML5 + AngularJS frontend) encryption had to be setup for both. The SSL on the frontend uses TLS1.2 using Let's Encrypt as the signing authority while the backend API uses the SSL provided by Heroku's wild card domain. It is possible to setup SSL using a custom domain with Heroku and this was also completed but given the low budget for this project and the $40 per month charge from Heroku proved too expensive, especially while the wild card Heroku provided certs is more than adequate.

6.6.4.2 User Authentication and Authorisation

This application follows best standards and trades a user's email and password for a secure token. This token is only ever transferred over SSL and only ever in the header. Finally, this token is compared using Devise compare function to help resist brute force and time attack issues. Tokens can easily be cycled and reset.

All passwords are hashed using BCrypt and salted.

User's data is isolated using PSQL schemas where user's only have access to their own tables which scoped connections.

6.6.4.3 SQL Injection Prevention

Active record has been used for 99% of all queries to the database and in those cases where it was not possible to use active record all variables were sent to the database separate to the query using parameters.

6.6.4.4 Minimum Password Policy

There is a minimum password policy of 8 characters currently in place with further quality checks and optional two factor authentication as a future development.

6.6.4.5 Protected HTTP Methods

Routes are only provided where needed with many views only offering basic GET and read only operations

6.6.4.6 JSON Encoding

All JSON responses are created using standard approved serializers libraries

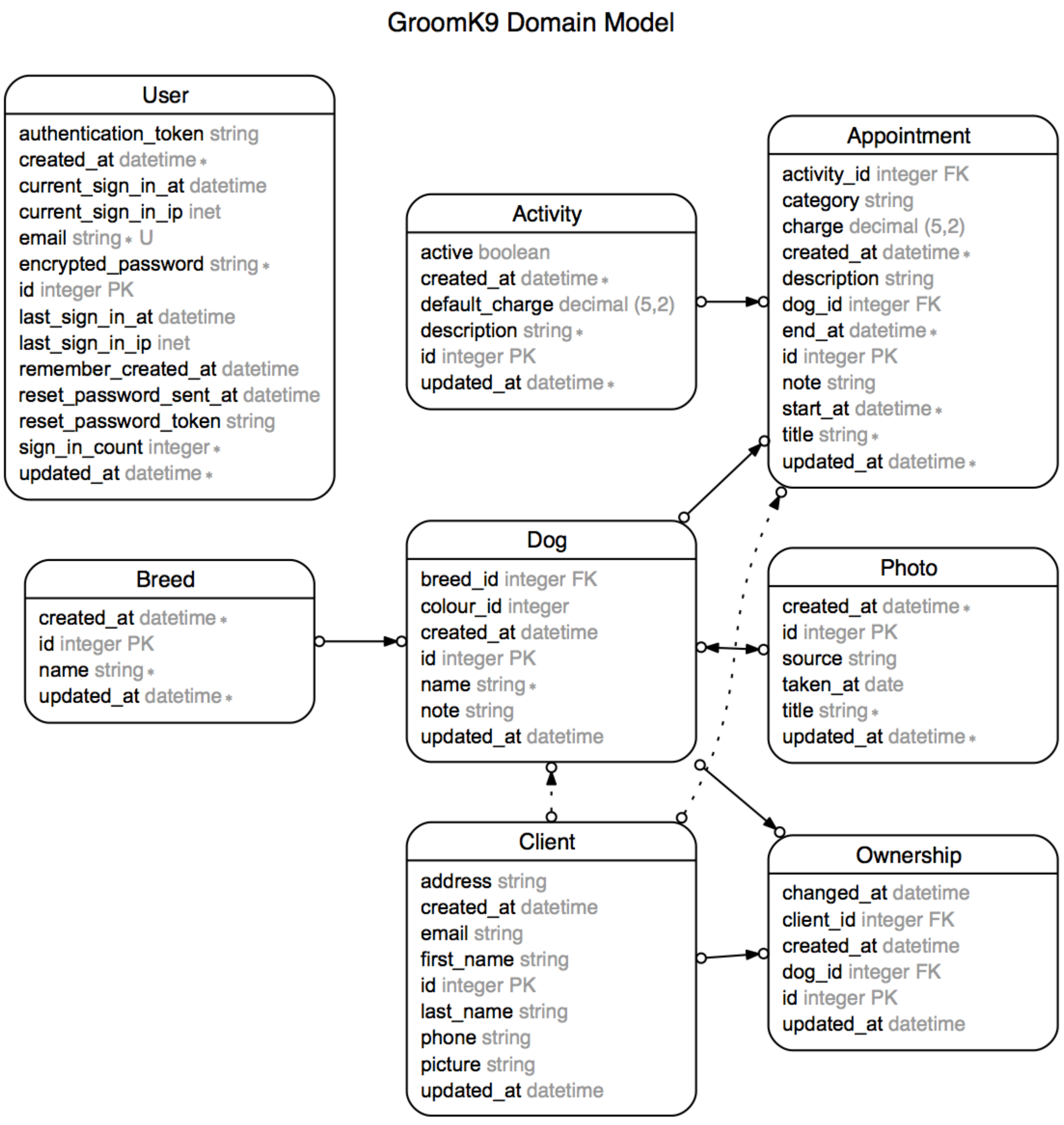
6.6.4.7 HTTP Return Codes

As per OWASP recommendations the RESTFul API uses more than just 200 for success and 404 for error return codes

7 System Design

7.1 Database Design

The primary data storage is for the day to day management of a Dog Grooming Parlor. The main entities are the Clients, their Pets, and their Appointments. The below represents the current entities and their relationships:



7.2 Application Program Interfaces (APIs)

As mentioned earlier in this document the entire backend of this application is an API. It has many end point but the primary ones will include CRUD operations following RESTFUL guidelines. This includes Create, Read, Update, and Delete using the HTTP verbs POST, GET, PUT, and DELETE for each of the following: Appointments, Client and Pets.

Third party API services have also been integrated to provide a mashup and extend the functionality of the application. These include MailGun and Cloudinary which are explained below.

7.2.1 Mail Gun – Transactional E-mail Service

Sending e-mails is a tricky business so I always find its best to outsource this to avoid headaches. MailGun is an established provider for this with an extensive and easy to use RESTFul api which seamlessly integrated with the ROR backend.

Site: https://www.mailgun.com/

7.2.2 Cloudinary – Image Storage Service

This service Insures that images uploaded are hosted on robust CDNs to improve the speed of loading for clients. It also comes with backup service as standard with the option with easy expansion. Another great feature of integrating this service is the extra functionality you get for free. For example, any photos uploaded of Clients in Groom K9 have facial recognition enabled which aids cropping the image at the optimum point.

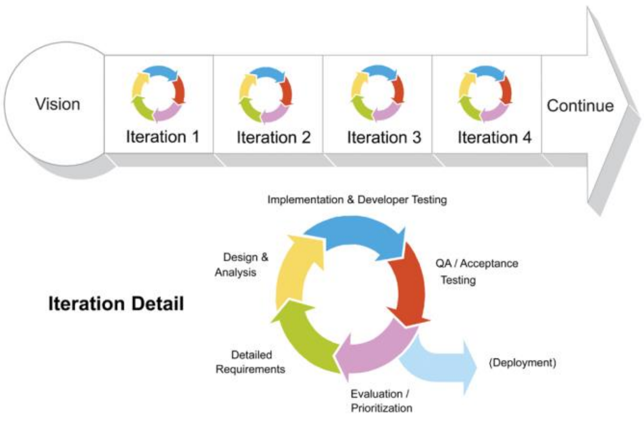
Site: http://cloudinary.com/

7.3 User Interface Design

This application will follow the Google Material Design Guidelines which can be found here: https://www.google.com/design/spec/material-design/introduction.html

8 Implementation

This project will be developed using agile methodologies in particular the use of the Scrum management framework. This will involve an iterative approach designing a prototype which will go through the phases of Evaluation, Requirements Gathering, Design and Analysis, and Implementation, and final user acceptance testing.



(Decoster, 2014)

Using this method will improve quality through constant testing (not just at the end); improve customer relations through iterative updates and allowing them to have early input. This will give direction to the project while preventing scope creep.

Gantt chart using Microsoft Project with details on implementation steps and timelines

This section of the document details how this project was implemented, display some code snippets and the design decisions behind them.

8.1 Project Plan

The original project plans are included in the appendix: (Project Plan and Microsoft Project)

8.2 Code Snippets & Breakdown

8.2.1 Backend

This software project capitalises on a number of the Ruby on Rails framework principles including MVC architecture, ORM, and Resource Routing.

8.2.1.1 Routing

A key area for a RESTFul API are its end points and routes as this is the primary interface any client application must interact with. A common convention is to use different HTTP Verbs (GET, POST, PUT, PATCH, DELETE) on the same end point to perform different actions. With GET used to retrieve information and never to update it, POST for new items and so on.

Rails convention driven framework mean that defining routes its very clean. Below is an example of routes for the backend in this application and we can see that it includes versioning, name spacing. These routes are all directed to different Controllers by naming convention. There is also a key word resources which when used in routes will generate routes for all the common CRUD commands (GET, PUT/PATCH, POST, DELETE).

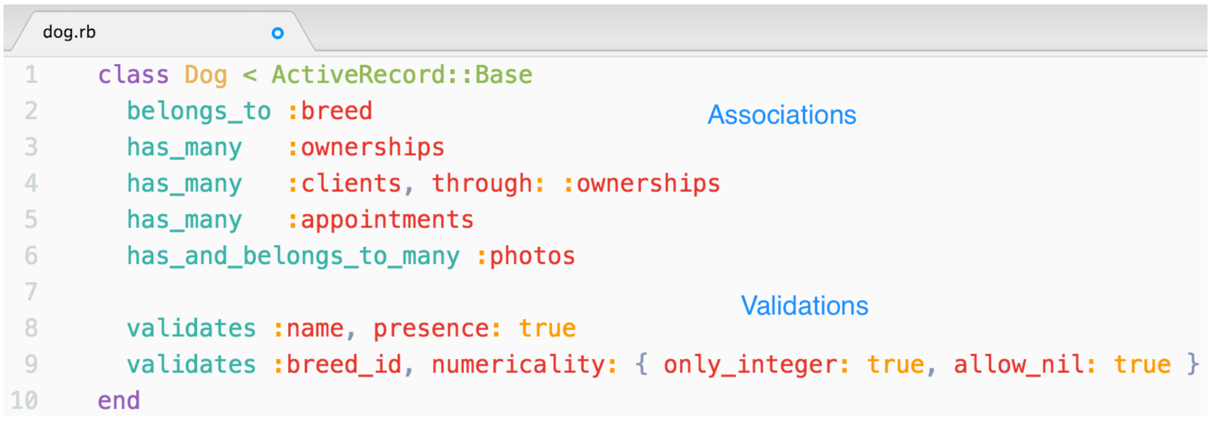
Extra comments have been added in the below example to highlight how the Resource route works by writing out the paths that will be generated.

```ruby
Rails.application.routes.draw do

  devise_for :users, skip: [ :sessions ]

  # API CORS
  resources :cors

  namespace :api, defaults: {format: 'json'} do

    namespace :v1 do

      # New Accounts + Sign Up
      resources :users, :only => [:create]
      post '/login' => 'sessions#create'

      # GET    /api/v1/photos       - returns all photos
      # GET    /api/v1/photos/:id   - Returns photo for with matching ID
      # POST   /api/v1/photos       - adds a new photo
      # PUT    /api/v1/photos/:id   - Updates a Photo with matching ID
      # DELETE /api/v1/photos/:id   - Deletes a Photo with matching ID
      resources :photos
```

8.2.1.2 Models

Models in the application are responsible for any business logic and any database interaction. The Rails framework provides a convention over configuration option so a little code can do a lot.

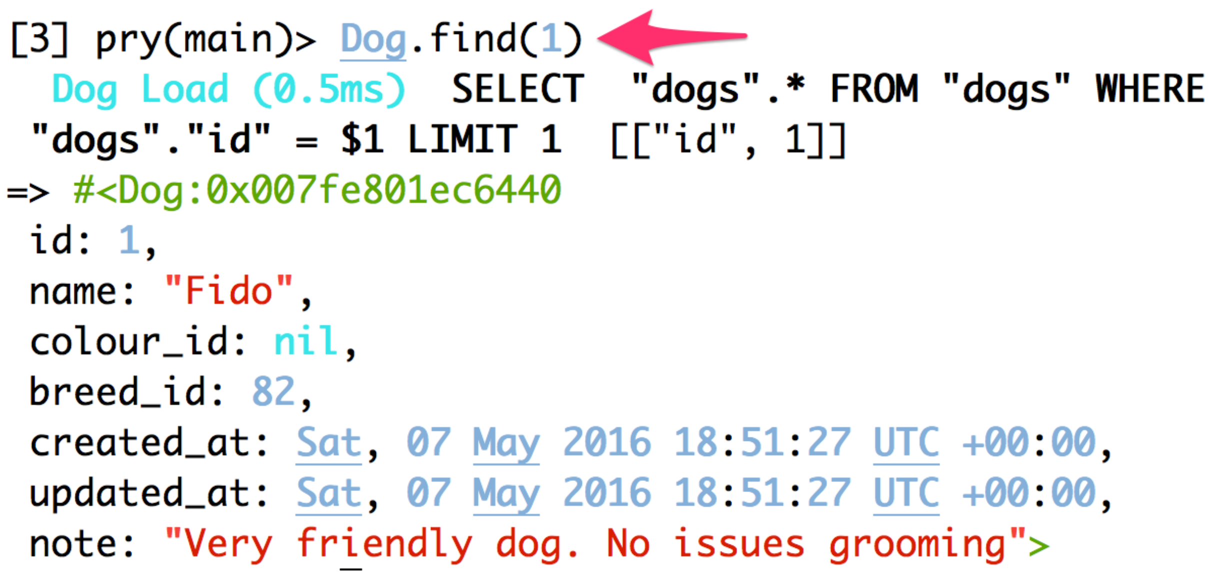
As an example the below file is the Model definition for the Dog in the system. The below 10 lines have created an Active record class, defined all its association/relationships to other classes in the system and even some validations before attempting to save a record to the database.



```ruby
class Dog < ActiveRecord::Base
  belongs_to :breed
  has_many   :ownerships
  has_many   :clients, through: :ownerships
  has_many   :appointments
  has_and_belongs_to_many :photos

  validates :name, presence: true
  validates :breed_id, numericality: { only_integer: true, allow_nil: true }
end
```

Retrieving information from the database can be as concise as follows:

DOG = DOG.FIND(DOGID) is all that's needed using Active record to retrieve a Dog for a given ID



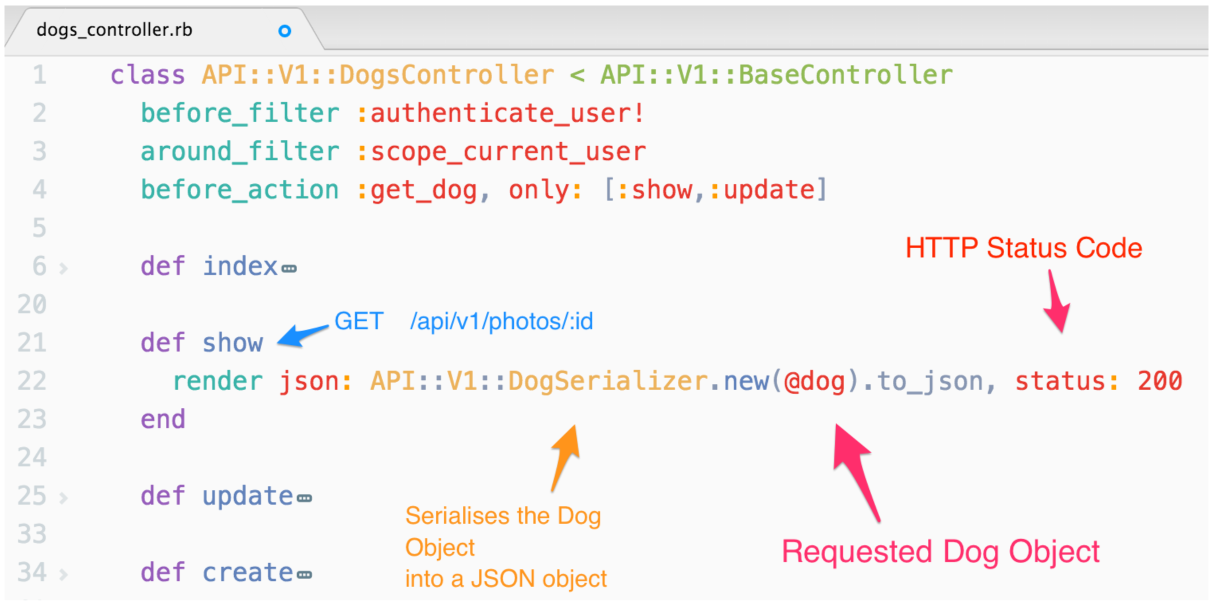
```
[3] pry(main)> Dog.find(1)
  Dog Load (0.5ms)  SELECT  "dogs".* FROM "dogs" WHERE
 "dogs"."id" = $1 LIMIT 1  [["id", 1]]
=> #<Dog:0x007fe801ec6440
 id: 1,
 name: "Fido",
 colour_id: nil,
 breed_id: 82,
 created_at: Sat, 07 May 2016 18:51:27 UTC +00:00,
 updated_at: Sat, 07 May 2016 18:51:27 UTC +00:00,
 note: "Very friendly dog. No issues grooming">
```

The above represents a simple class, yet very rich with functionality. For a more complex example please view the User class in the project code (not included here for brevity's sake).

8.2.1.3 Controllers

As in any MVC architecture the Controller is the glue which joins the Business/Model with the View. In the case of an RESTFul API this is the very same except the View is not hosted on the same machine and is fed via HTTP and JSON data.

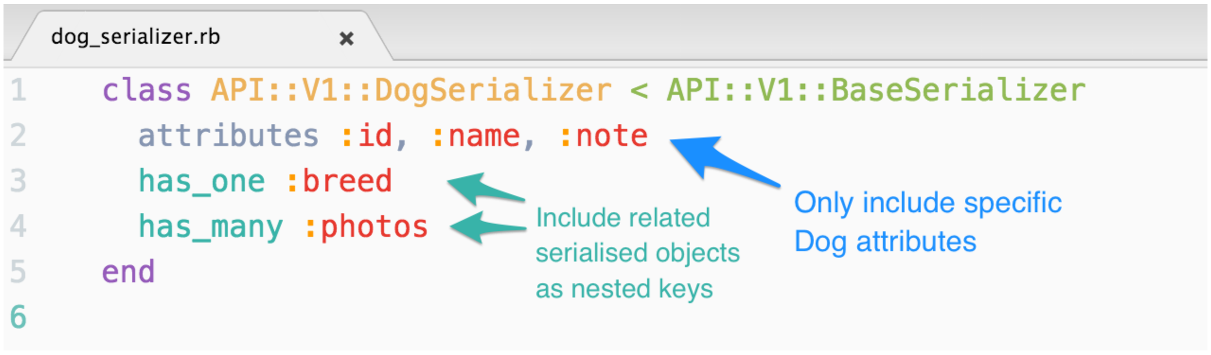
All the views in relation to the API are in their own folder /controllers/api/v1/*.rb



```
class API::V1::DogsController < API::V1::BaseController
  before_filter :authenticate_user!
  around_filter :scope_current_user
  before_action :get_dog, only: [:show,:update]

  def index

  def show
    render json: API::V1::DogSerializer.new(@dog).to_json, status: 200
  end

  def update

  def create
```

8.2.1.4 Serializers

To help follow a DRY (Do Not Repeat Yourself) approach as per Ruby guidelines a Serializer is used to serialize the Dog object into a JSON object. The benefits of this is that it can be reused anywhere a Dog needs to be returned. This serializer can also be used within another serializer.

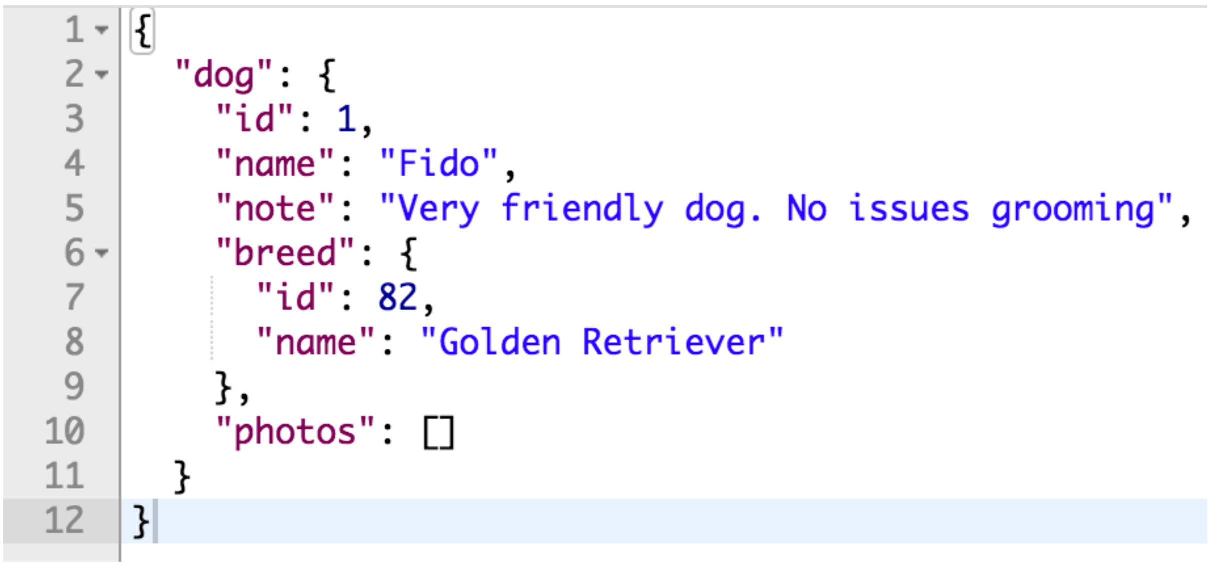


dog_serializer.rb

```ruby
class API::V1::DogSerializer < API::V1::BaseSerializer
  attributes :id, :name, :note
  has_one :breed
  has_many :photos
end
```

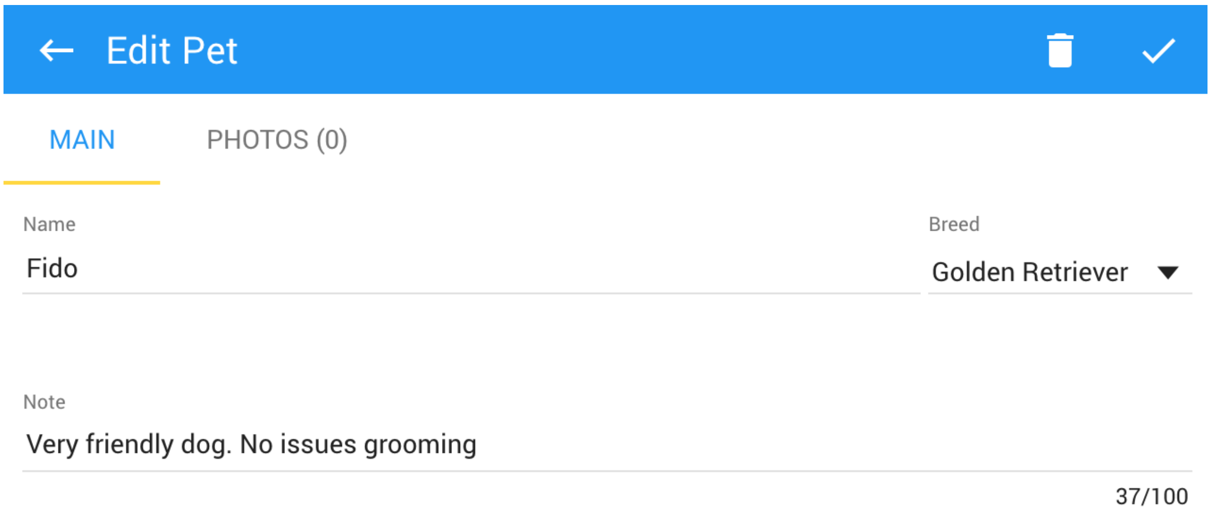
Only include specific Dog attributes
Include related serialised objects as nested keys

The combination of the above Controller (returning data from Model) and then serializing it with the Dog Serializer results in the following JSON structure



```json
{
  "dog": {
    "id": 1,
    "name": "Fido",
    "note": "Very friendly dog. No issues grooming",
    "breed": {
      "id": 82,
      "name": "Golden Retriever"
    },
    "photos": []
  }
}
```

Finally, this JSON is used by the Client. In this project it is used in the AngularJS web app on the Edit Dog section:



Edit Pet

MAIN   PHOTOS (0)

Name: Fido

Breed: Golden Retriever

Note: Very friendly dog. No issues grooming

37/100

8.2.1.5 PDF Reports with Prawn

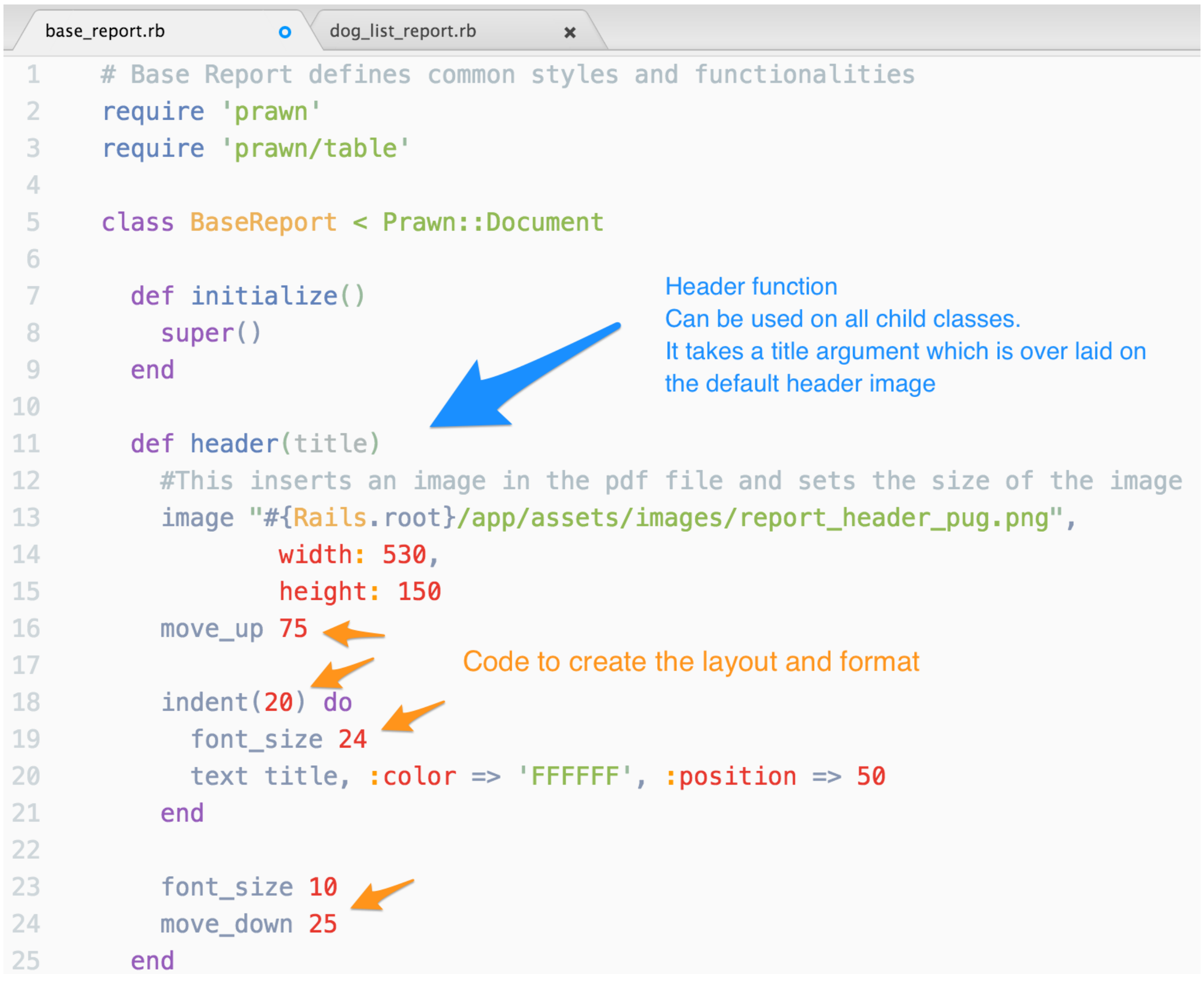
Prawn is a pure Ruby PDF generation library that provides a lot of great functionality while trying to remain simple and reasonably performant (Anon., 2016)

Prawn has two main methods of generating PDFs. It can take a HTML file and create a PDF based on it or using its own DSL[1]. While the HTML method is easy to use the DSL method offers greater power and flexibility at the cost of having to learn new syntax. For the purposes of this project the DSL version was chosen as this extra flexibility will be more important as future reports grow in complexity.

How does it work?

Reports are defined in Ruby class files so the power of Ruby is available to generate the reports. This includes Inheritance, Mix-ins, helper function, and so on. All reports in this project inherit from a base class which defines common methods for items such as headers and footers.

Sample Base report from which all other reports inherit from:

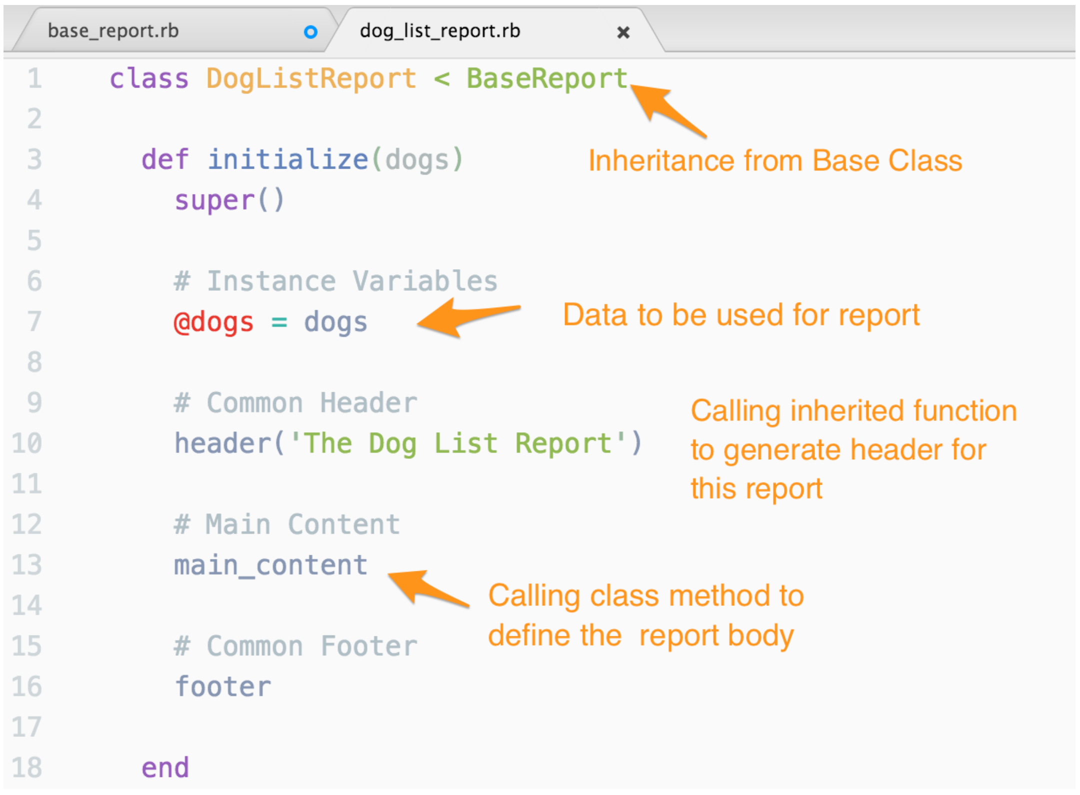

base_report.rb
dog_list_report.rb
Header function
Can be used on all child classes.
It takes a title argument which is over laid on the default header image
Code to create the layout and format

```ruby
# Base Report defines common styles and functionalities
require 'prawn'
require 'prawn/table'

class BaseReport < Prawn::Document

  def initialize()
    super()
  end

  def header(title)
    #This inserts an image in the pdf file and sets the size of the image
    image "#{Rails.root}/app/assets/images/report_header_pug.png",
            width: 530,
            height: 150
    move_up 75

    indent(20) do
      font_size 24
      text title, :color => 'FFFFFF', :position => 50
    end

    font_size 10
    move_down 25
  end
```

[1] DSL: Domain-specific language (DSL) is a computer language specialized to a particular application domain

This example shows a report class inheriting from Prawn::Document. There is header definition which creates a report heading which includes an image overlaid with a custom title. There is also a number of different formatting and adjusting where the class moves the current print position.

Sample Inherited Report using the parent's methods:


base_report.rb
dog_list_report.rb
Inheritance from Base Class
Data to be used for report
Calling inherited function to generate header for this report
Calling class method to define the report body

```ruby
class DogListReport < BaseReport

  def initialize(dogs)
    super()

    # Instance Variables
    @dogs = dogs

    # Common Header
    header('The Dog List Report')

    # Main Content
    main_content

    # Common Footer
    footer

  end
```

The above image displays how clean and straight forward is to create a list report with clearly defined methods for each task.

The @dogs instance variable is passed from the calling source and is used in the 'main_content' function to print a table on the report. A screenshot of this is included in the appendix (Prawn Table Creation Method).

Once the above report definitions are created a new report can be created by instantiating a new report object and telling it to create a new PDF.

```ruby
@dogs = Dog.all
pdf = DogListReport.new(@dogs)
send_data pdf.render, filename: 'report.pdf', type: 'application/pdf'
```

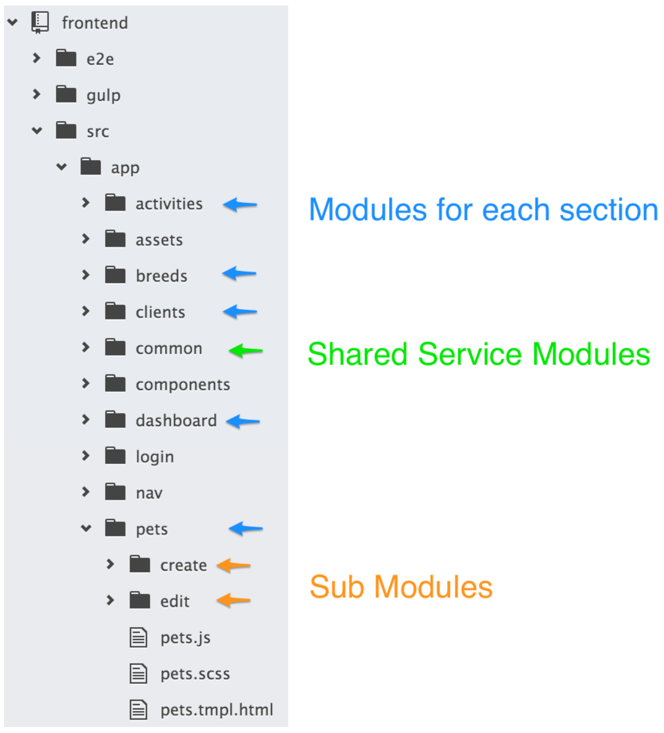
8.2.2 Frontend

There is a lot of design patterns and conventions used in the Frontend probably even more than in the backend. This section will describe some of these key areas. While AngularJS represents the V in MVC from an entire project point of view, with in that view is an additional MVVM approach which in turn also has different layers for Model, Controllers and Views.

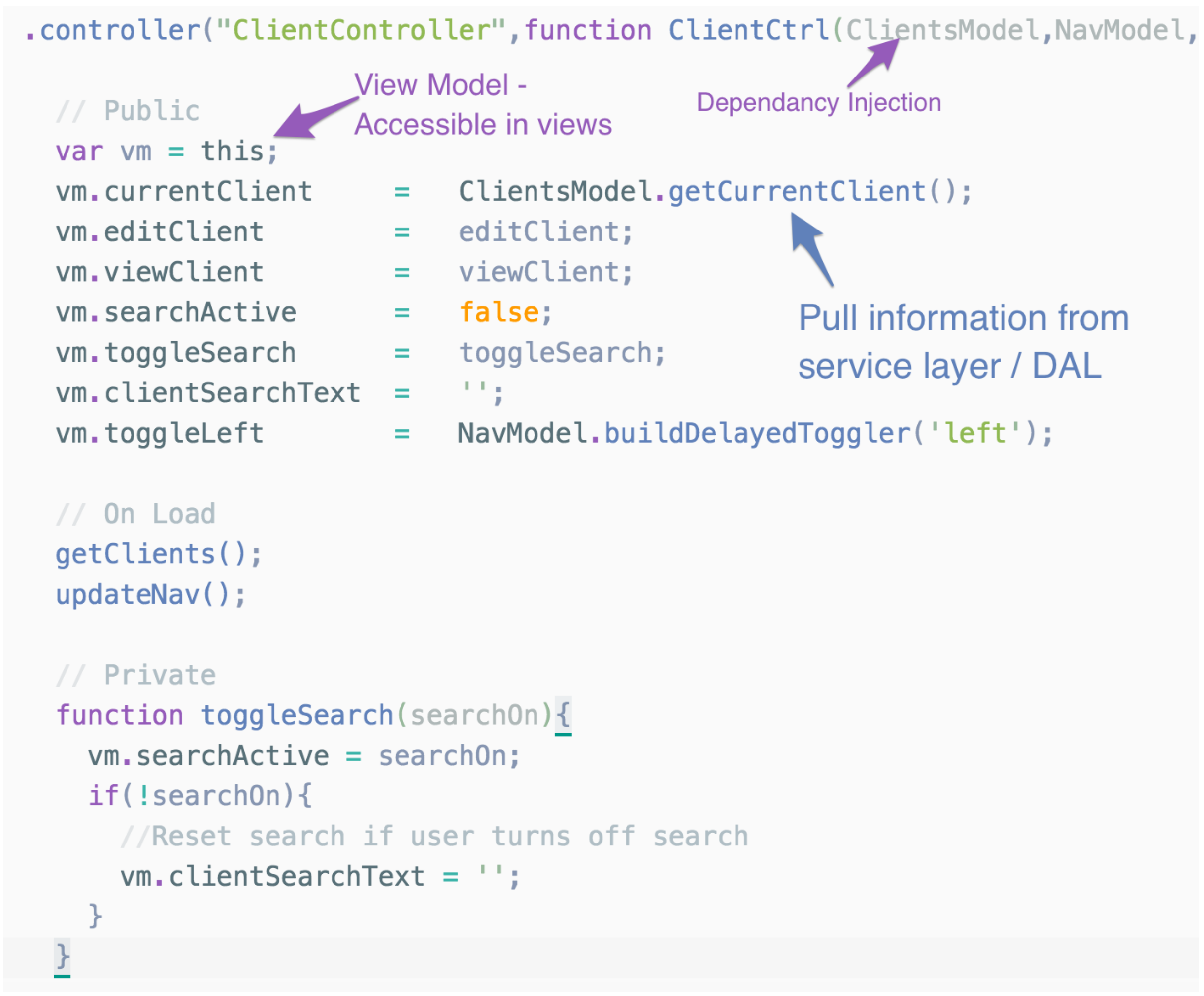
8.2.2.1 Code Organization

There are many conventions on how to organise code in AngularJS but the approach that was used in this project was to keep all related items in modules which are organised by folder i.e. View Template files, Controller Files, Directives are all grouped by area rather than by type; e.g. The Dog module will contain all the required templates and controllers. Some other angular conventions group files by type, i.e. all controllers together. I believe the former approach helps keep code more modular and easier to organise. As described in the earlier Standards section this organisation was greatly influence by the Papa John Approach (Anon., 2016) and a course on EggHead.io entitled 'AngularJS Architecture' (Ruebbelke, 2016). Any other resources needed by the module that are not common are also included within the folder e.g. SCSS files which includes styles just for that module.

The exception to the above structure was to create a common folder. This folder contains code which acts as a Service or Data access which are used to access information from the API but also share that information once it's been downloaded across many areas. E.g. any gateway code to access data that might be used through the system e.g. Dog information as its used on the Appointments Screen, Dog Screen, and Clients Screen. This layer also provides caching to reduce calls to the server.



Sample Controller in frontend (not dissimilar to the ones in Ruby On Rails):



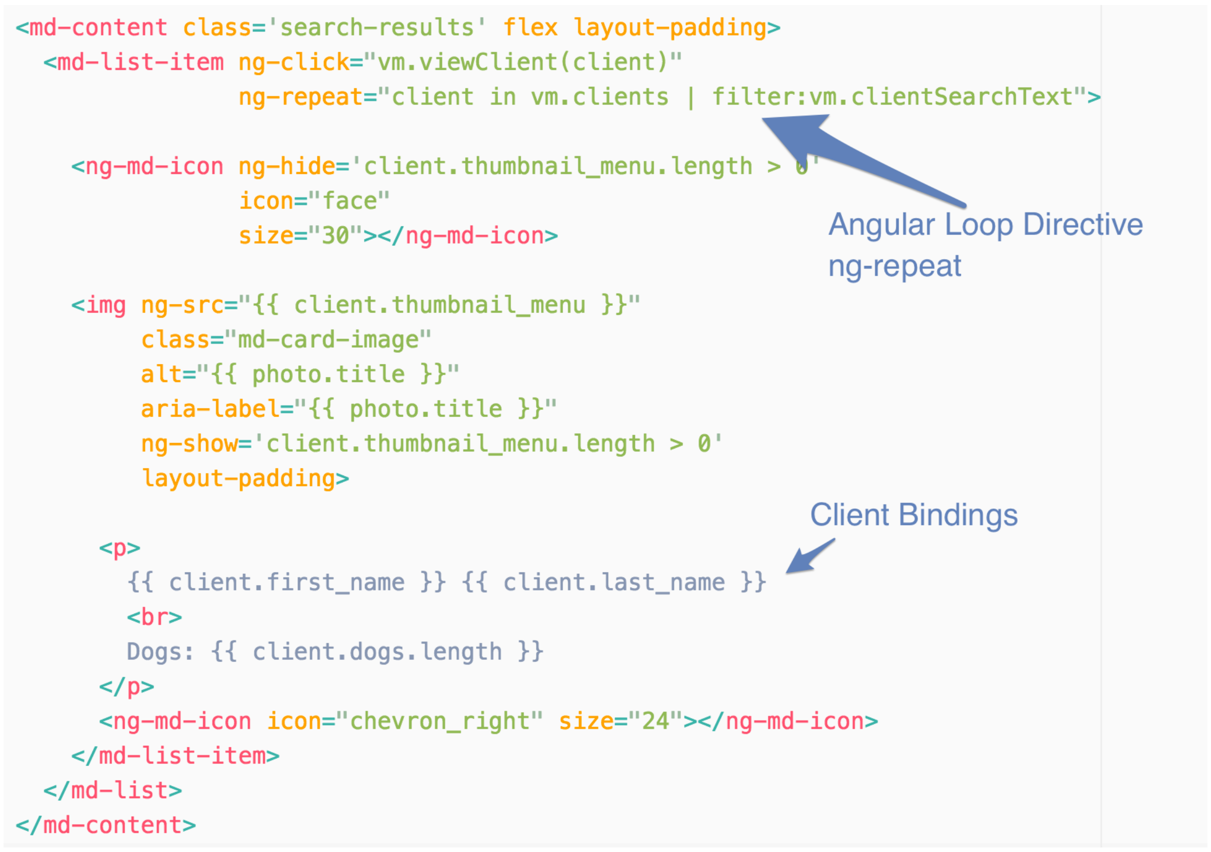
```
.controller("ClientController",function ClientCtrl(ClientsModel,NavModel,

  // Public
  var vm = this;
  vm.currentClient     =   ClientsModel.getCurrentClient();
  vm.editClient        =   editClient;
  vm.viewClient        =   viewClient;
  vm.searchActive      =   false;
  vm.toggleSearch      =   toggleSearch;
  vm.clientSearchText  =   '';
  vm.toggleLeft        =   NavModel.buildDelayedToggler('left');

  // On Load
  getClients();
  updateNav();

  // Private
  function toggleSearch(searchOn){
    vm.searchActive = searchOn;
    if(!searchOn){
      //Reset search if user turns off search
      vm.clientSearchText = '';
    }
  }
```

Key conventions in this were to use the newer Angular ‘controllerAsSyntax’ so that all calls are made available through the view model variable called VM in this example. All public methods are attached to this and always at the top of the file. On load and private functions follow, finally with routing information (not displayed in screenshot) at the bottom

Controllers can also manage state, in this case ‘searchActive’ but other controllers can have ‘isEditing’ or ‘isSaving’. These are used by views to determine what to display.

This controller can then be referenced in a route (similar to ROR already discussed) and linked to a view. AngularJS uses two way binding so any values in the View Model are reflected in the view and vice versa.

Example View:

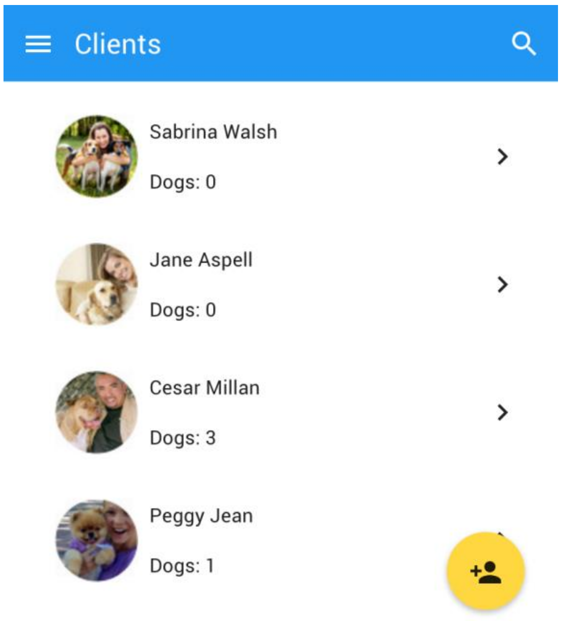


```
<md-content class='search-results' flex layout-padding>
  <md-list-item ng-click="vm.viewClient(client)"
                ng-repeat="client in vm.clients | filter:vm.clientSearchText">

    <ng-md-icon ng-hide='client.thumbnail_menu.length > 0'
                icon="face"
                size="30"></ng-md-icon>

    <img ng-src="{{ client.thumbnail_menu }}"
         class="md-card-image"
         alt="{{ photo.title }}"
         aria-label="{{ photo.title }}"
         ng-show='client.thumbnail_menu.length > 0'
         layout-padding>

      <p>
        {{ client.first_name }} {{ client.last_name }}
        <br>
        Dogs: {{ client.dogs.length }}
      </p>
      <ng-md-icon icon="chevron_right" size="24"></ng-md-icon>
    </md-list-item>
  </md-list>
</md-content>
```

This is snippet of code highlights how two-way binding is used within this HTML file. To insert a binding a value can be placed inside 4 curly brackets:
e.g. {{ vm.someVariable }}

The md-* tags are part of Angular Material Design directives and offer different layouts and controls to the page. Aria attributes are used for accessibility and screen readers.

The above Angular code is responsible for the rendered view to the right.



Clients

Sabrina Walsh
Dogs: 0

Jane Aspell
Dogs: 0

Cesar Millan
Dogs: 3

Peggy Jean
Dogs: 1

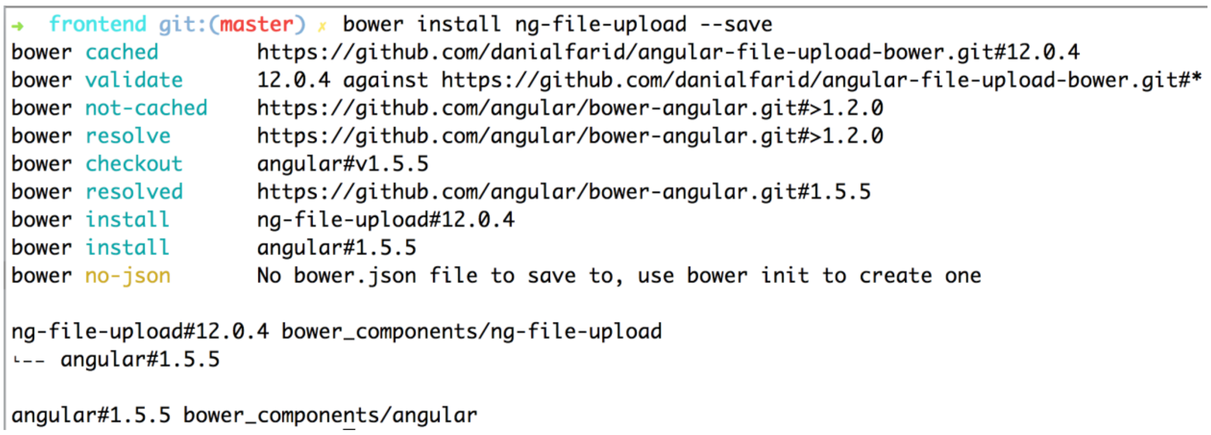
8.2.2.2 *NPM & Bower for Dependency Management*

As the frontend of this application is a single page app utilising AngularJS there are a lot of third party libraries and resources which needed to be managed.

NPM & Bower simplify the downloading of these libraries and any dependencies that they require. The main advantage of this is that these files do not have to be downloaded by hand and also that they can be easily left out of source control which is a good thing as it reduces the repositories size and frivolous commits to update third party libraries.

Downloading and Installing an AngularJS Library before Angular is installed using Bower:

Bower is able to determine that the ‘ng-file-upload’ library has a dependency on AngularJS and so that is also downloaded and copied to the folder.



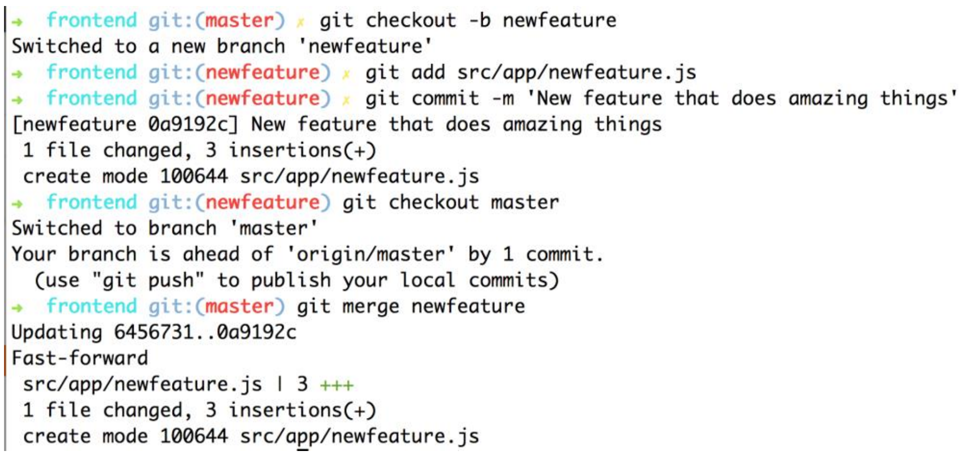
```
➜  frontend git:(master) ✗ bower install ng-file-upload --save
bower cached        https://github.com/danialfarid/angular-file-upload-bower.git#12.0.4
bower validate      12.0.4 against https://github.com/danialfarid/angular-file-upload-bower.git#*
bower not-cached    https://github.com/angular/bower-angular.git#>1.2.0
bower resolve       https://github.com/angular/bower-angular.git#>1.2.0
bower checkout      angular#v1.5.5
bower resolved      https://github.com/angular/bower-angular.git#1.5.5
bower install       ng-file-upload#12.0.4
bower install       angular#1.5.5
bower no-json       No bower.json file to save to, use bower init to create one

ng-file-upload#12.0.4 bower_components/ng-file-upload
└── angular#1.5.5

angular#1.5.5 bower_components/angular
```

8.2.2.3 *GIT Version Control*

GIT is a distributed version control system and has been used since the very start of this project. There were two GIT projects setup, one for the frontend and one for the backend.

As part of the development iterative approach, new branches were created for any new functionality and only merged back into the master branch once they had been reviewed, tested, and approved.

Example:



```
➜  frontend git:(master) ✗ git checkout -b newfeature
Switched to a new branch 'newfeature'
➜  frontend git:(newfeature) ✗ git add src/app/newfeature.js
➜  frontend git:(newfeature) ✗ git commit -m 'New feature that does amazing things'
[newfeature 0a9192c] New feature that does amazing things
 1 file changed, 3 insertions(+)
 create mode 100644 src/app/newfeature.js
➜  frontend git:(newfeature) git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
➜  frontend git:(master) git merge newfeature
Updating 6456731..0a9192c
Fast-forward
 src/app/newfeature.js | 3 +++
 1 file changed, 3 insertions(+)
 create mode 100644 src/app/newfeature.js
```

8.2.2.4 GULPJS

GULPJS describes itself as a task manager. For this project is has been used to perform a number of actions for the front end. The initial GULPJS routines were generated using Yeoman.

8.2.2.5 Sass (Syntactically Awesome Style Sheets)

There are multiple SASS located throughout the frontend project. They are global styles as well as SASS files for individual modules and features. GULPJS runs a job which watches these files and if a change is made will convert the changes to CSS, combine them into one file, and compress them on the fly.

8.2.2.6 JavaScript

JavaScript is the primary language used in the frontend site and as such there are many individual files. There is a GULPJS task which monitors changes to these files, and again combines them into one and compresses. Any syntax errors are highlighted at build time.

8.2.2.7 Webserver + BrowserSync

GULPJS serves the frontend webpage using the BrowserSync. This is a tool which automatically monitors changes in files and will reload the webpage in the browser once these changes occur. This means that as a developer you no longer have to change code and then refresh to see the changes, it does it automatically. Another nice addition in using this tool for this project is that you have many different devices looking at the webpage e.g. laptop, tablet, phone and as soon as code is changed all three devices refresh. This proved very helpful with developing the responsive design layout.

8.2.2.8 Building Front Distributions

Once the frontend code is ready for a release there is a GULPJS task called 'build' which creates a deployment. Jobs included in this include importing third party libraries, compressing JS and SASS, injecting new links into the main HTML file and exporting to a build folder. This folder can then easily be deployed to the frontend webserver (https://www.netlify.com/ for this project)

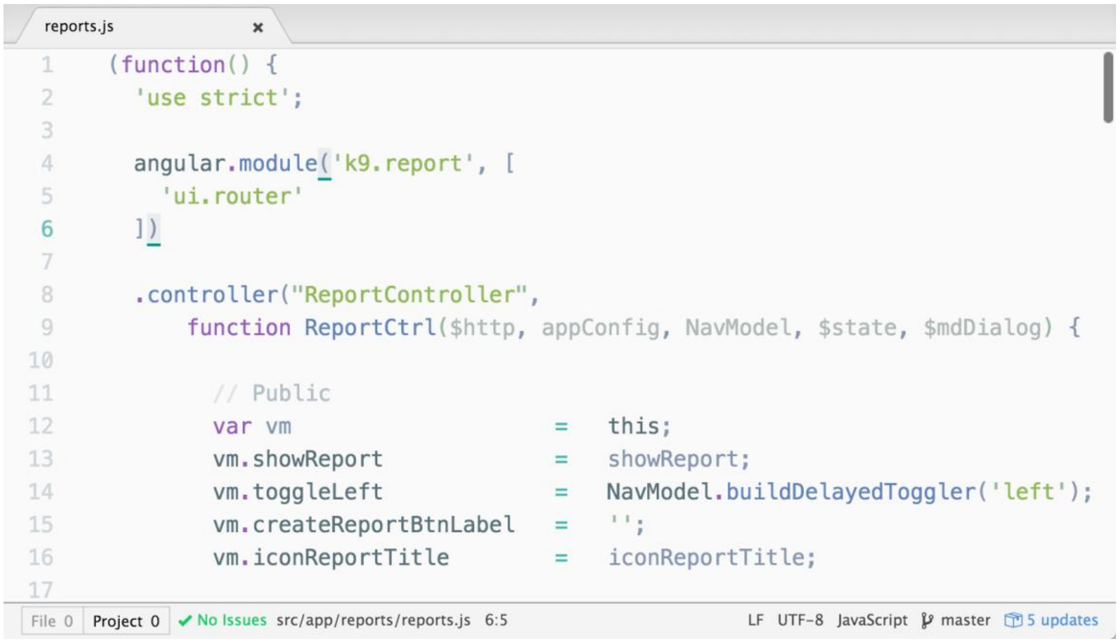
8.2.2.9 Linters

Linters were used during the development of this project which greatly impacted in a positive way on the implementation process. Linters are programs which help verify code quality by highlighting coding syntax errors but also bad practice.

For this project linters were used with the Atom editor (although they are available for lots of editors and languages). This tool highlights code errors as you type are were very useful to highlight problems as they were written. This is particularly useful in non-compiled languages such as JavaScript where you may otherwise not notice the error until run time.

Sample Report File in Project with No Errors

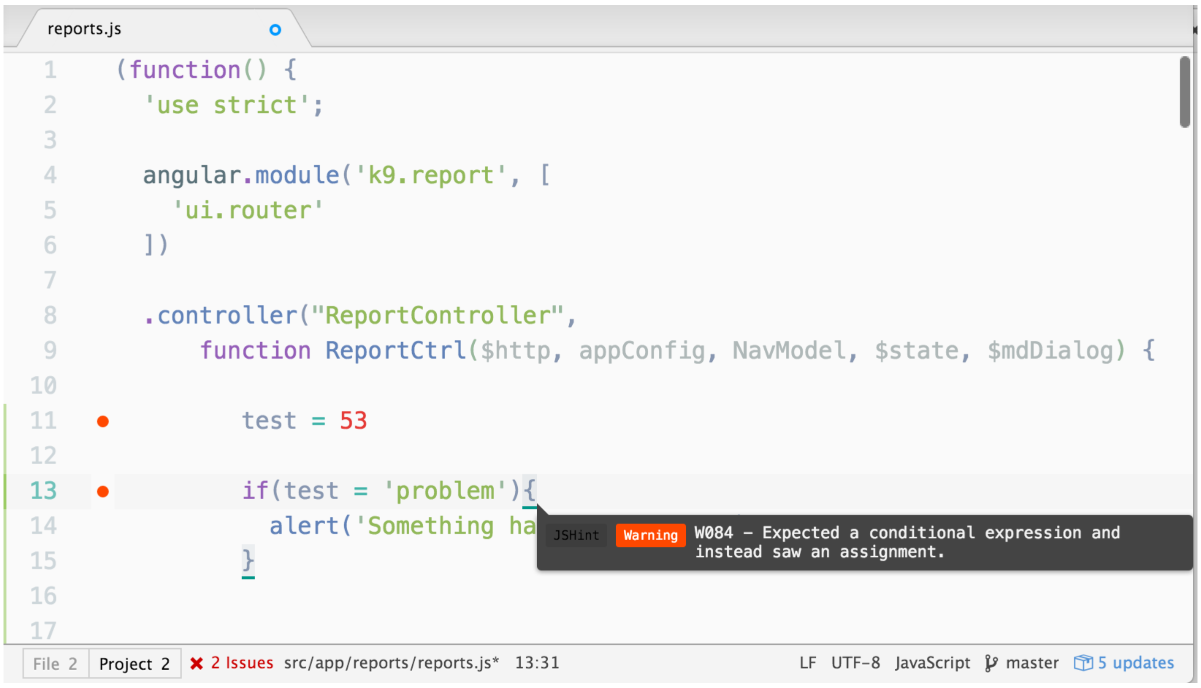
Below is a screenshot of a JavaScript file in the project with no errors. The bottom bar informs us that there are 'No Issues'



```
reports.js
(function() {
  'use strict';

  angular.module('k9.report', [
    'ui.router'
  ])

  .controller("ReportController",
      function ReportCtrl($http, appConfig, NavModel, $state, $mdDialog) {

        // Public
        var vm                        =   this;
        vm.showReport                 =   showReport;
        vm.toggleLeft                 =   NavModel.buildDelayedToggler('left');
        vm.createReportBtnLabel       =   '';
        vm.iconReportTitle            =   iconReportTitle;

File 0  Project 0  ✔ No Issues  src/app/reports/reports.js  6:5     LF  UTF-8  JavaScript  master  5 updates
```

Sample Report File in Project with 2 Errors

Each line that has an error there is a red dot placed next to it and it is also noted in the bottom toolbar. Clicking near these lines gives a full explanation. In this example the first line is missing a semi-colon and the second line is using an assignment operator rather than a comparison operator.



```
reports.js
(function() {
  'use strict';

  angular.module('k9.report', [
    'ui.router'
  ])

  .controller("ReportController",
      function ReportCtrl($http, appConfig, NavModel, $state, $mdDialog) {

●         test = 53

●         if(test = 'problem'){
            alert('Something ha
          }

File 2  Project 2  ✖ 2 Issues  src/app/reports/reports.js*  13:31     LF  UTF-8  JavaScript  master  5 updates
```

JSHint Warning W084 - Expected a conditional expression and instead saw an assignment.

8.2.3 Deployment

The below sections cover how the application is deployed for the backend and the frontend.

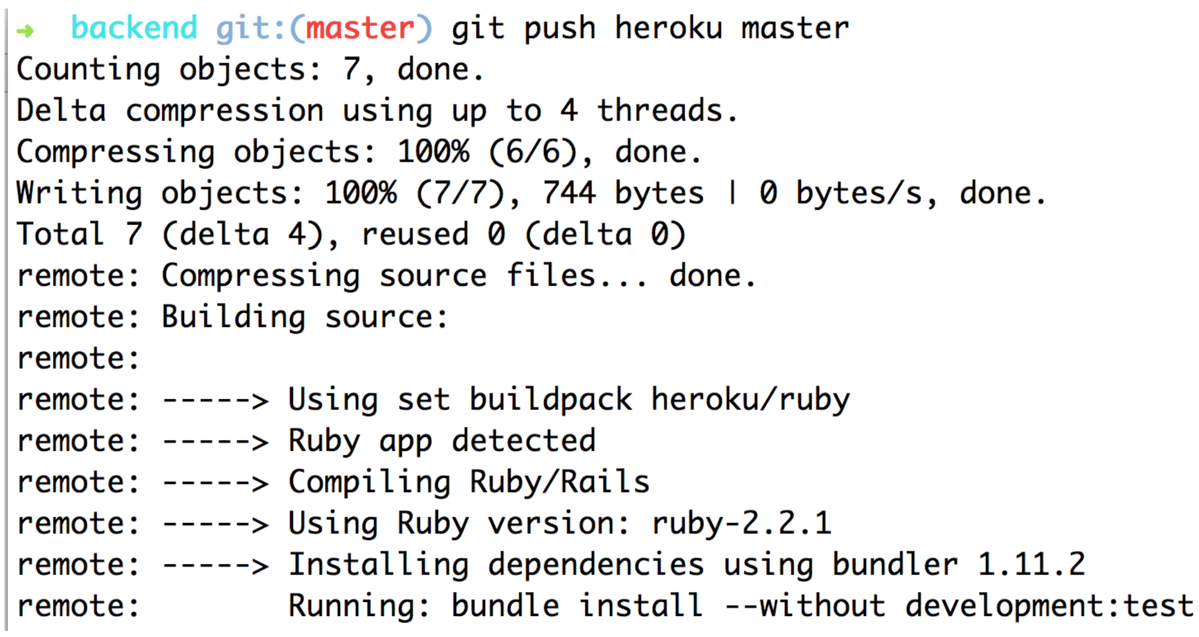
8.2.3.1 Backend ROR API

The backend Ruby On Rails API application is deployed to Heroku server. The primary means to this is via GIT which was discussed earlier.

To update the live server, the following command simply has to be run:

"git push Heroku master"

Once pushed to the server Heroku will start a build process. Once this is complete database migrations can be called by running *"Heroku run rake db:migrate".* As this project makes use of multiple PSQL schemas the above migrate only makes changes to the default database. To perform migrations on all the other schemas a custom task has been setup which can be called by running: *"rake users:db:migrate"*



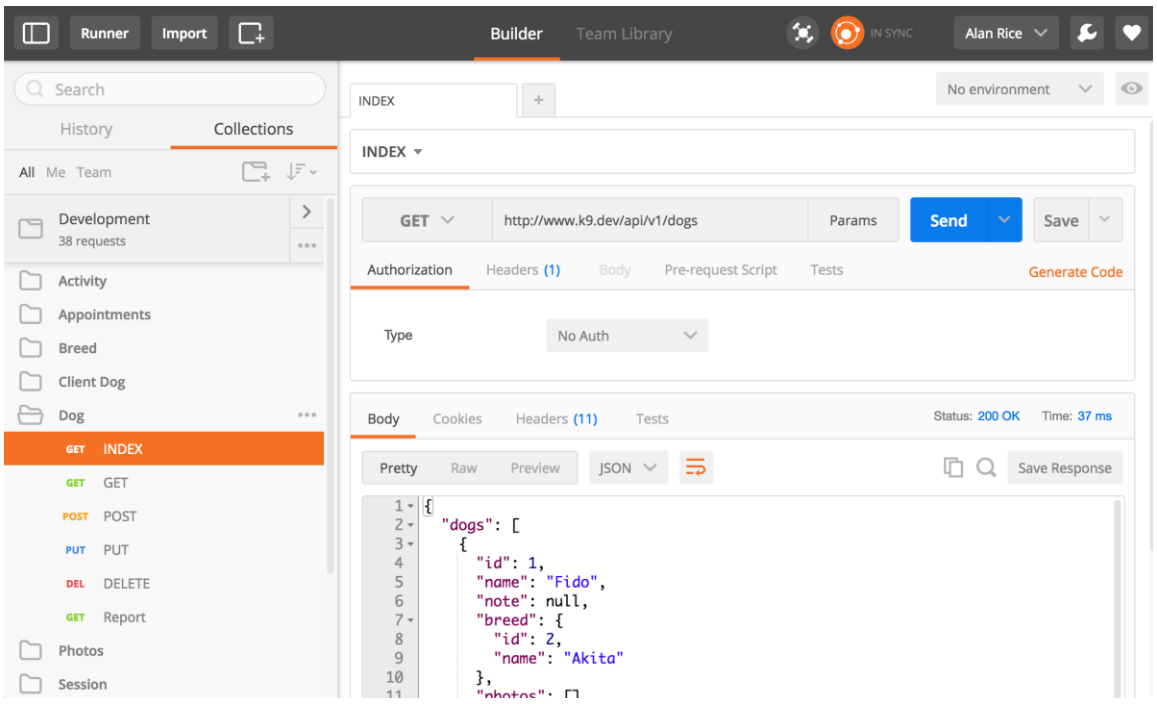
```
➜  backend git:(master) git push heroku master
Counting objects: 7, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 744 bytes | 0 bytes/s, done.
Total 7 (delta 4), reused 0 (delta 0)
remote: Compressing source files... done.
remote: Building source:
remote:
remote: -----> Using set buildpack heroku/ruby
remote: -----> Ruby app detected
remote: -----> Compiling Ruby/Rails
remote: -----> Using Ruby version: ruby-2.2.1
remote: -----> Installing dependencies using bundler 1.11.2
remote:        Running: bundle install --without development:test
```

8.2.3.2 Frontend AngularJS

As discussed earlier a build can be created for the Angular frontend by using the GULP job. Once build this folder can simply be uploaded to Netlify.com where it will be deployed on their server. There are additional options for builds and auto deployments using git but given the easy of the GULP job I did not find a need for this.

8.2.4 Postman

This was a key piece of software for development and testing. It's a frontend tool which lets you save a collection of REST API calls place them in categories. It proved to be very useful in testing the API backend without having to rely on AngularJS or another intermediary client.



```
{
  "dogs": [
    {
      "id": 1,
      "name": "Fido",
      "note": null,
      "breed": {
        "id": 2,
        "name": "Akita"
      },
      "photos": [
```

9 Evaluation & Testing

Automated testing should ideally be done before any code is written in a Test Driven Development or Behaviour Driven Development process.

9.1 API Backend Testing

Testing took a Bottom Up Testing approach and leveraged many of the Ruby on Rails (ROR) features to perform automated tests.

The separation of presentation and business logic into Models, Views, Controllers (MVC) design pattern in ROR allowed the development to follow a Test-driven Development (TDD) process. This resulted in key components can being unit tested more easily with less complex setups.

9.1.1 Integration Testing with RSpec

RSpec is a behaviour-driven development (BDD) framework available for the Ruby programming language.

RSpec

As the main server application code is provided as an API rather than a normal Rails web application the main type of test that we need to perform is an integration test.

The following code is a sample of how an RSpec test is written in Ruby. Test information is entered directly into the database using FactoryGirl (discussed in next section).

The test then requests the data from the server using the API end point (GET /dogs/dogid). Once the server replies the RSpec test checks two things:

1. That the HTTP header code is 200 i.e. the request was successful
2. Checks that there was a JSON response, that it contains the correct keys (the dog's name), and finally that the dog's name matches up with the dog that was created at the start of the test.

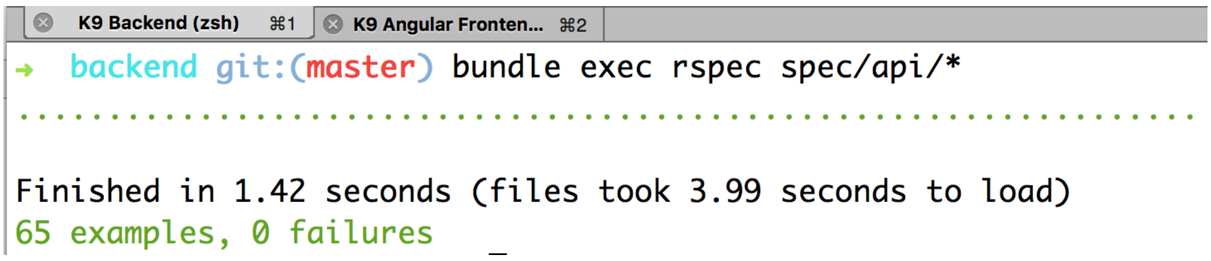
```ruby
describe "GET #show" do

  # Create record for Dog, then call get dog endpoint
  before(:each) do
    @dog = FactoryGirl.create :dog
    get :show, id: @dog.id, format: :json
  end

  # Returns Correct Header Code
  it { should respond_with 200 }

  # Correct body information in JSON format
  it "returns the information about a dog on a hash" do
    dog_response = JSON.parse(response.body, symbolize_names: true)
    expect(dog_response[:dog][:name]).to eql @dog.name
  end

end
```

RSpec tests have been written to cover all the main API end points and include checks for Creating, Updating, Reading, and Deleting each of the resources. In addition to these checks there are also negative checks i.e. checks where we provide bad data to the API endpoint and check that an error code is returned.

This type of testing has greatly improved the quality of the code in this project and has helped prevent regression bugs in part due to how easy it is to run the tests.

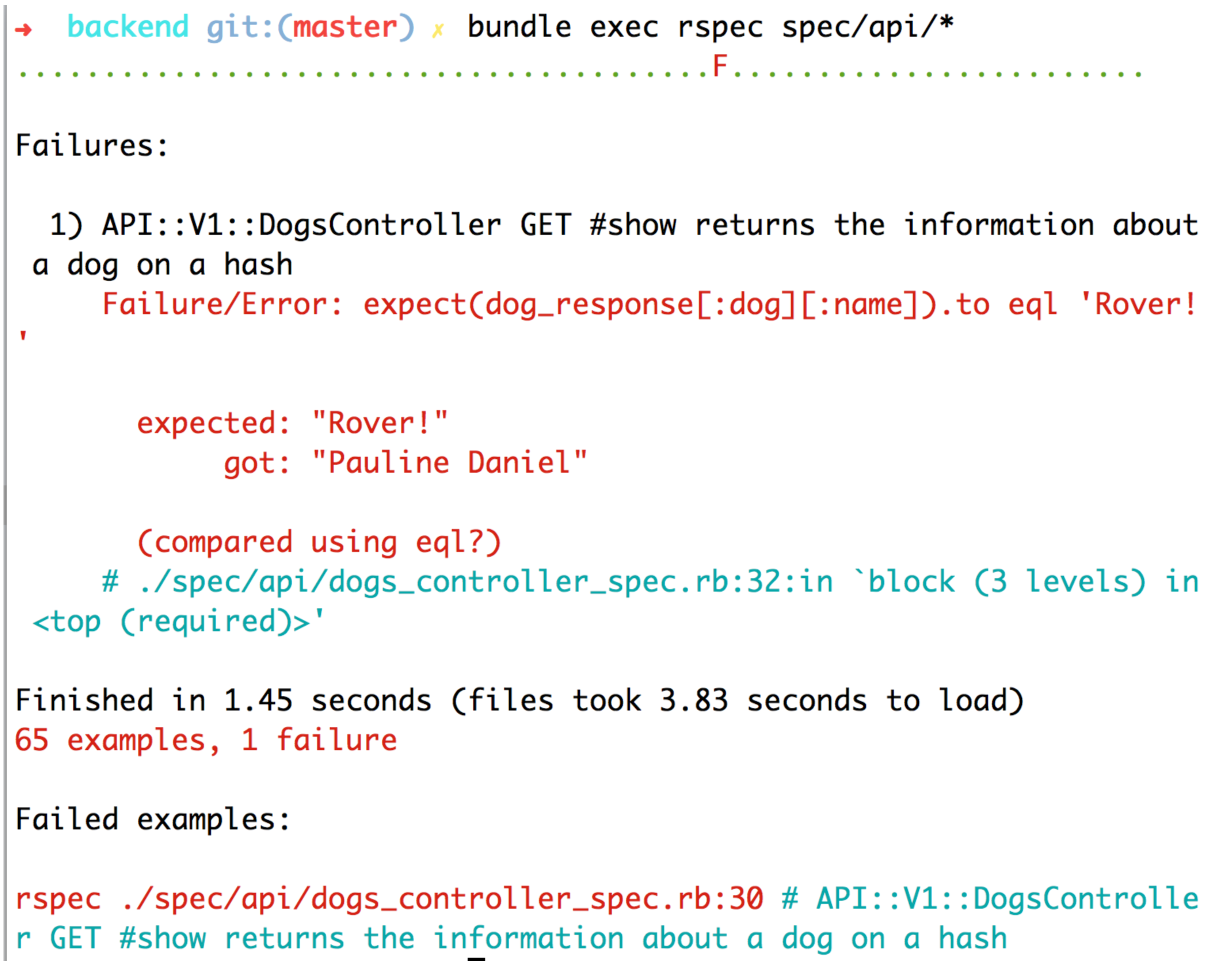
e.g. to run all tests in the console the following command can be run in the terminal by:
*bundle exec rspec spec/api/**

RSpec with all API Tests Passing Example



```
K9 Backend (zsh)  ⌘1    K9 Angular Fronten...  ⌘2
➜  backend git:(master) bundle exec rspec spec/api/*
.................................................................

Finished in 1.42 seconds (files took 3.99 seconds to load)
65 examples, 0 failures
```

RSpec with one test failing



```
➜  backend git:(master) ✗ bundle exec rspec spec/api/*
......................................F........................

Failures:

  1) API::V1::DogsController GET #show returns the information about
 a dog on a hash
     Failure/Error: expect(dog_response[:dog][:name]).to eql 'Rover!
'

       expected: "Rover!"
            got: "Pauline Daniel"

       (compared using eql?)
     # ./spec/api/dogs_controller_spec.rb:32:in `block (3 levels) in
 <top (required)>'

Finished in 1.45 seconds (files took 3.83 seconds to load)
65 examples, 1 failure

Failed examples:

rspec ./spec/api/dogs_controller_spec.rb:30 # API::V1::DogsControlle
r GET #show returns the information about a dog on a hash
```

9.1.2 FactoryGirl and FFaker

As outlined above an important part of testing the API is the ability to reset the database and then insert the records needed to test the functionality each time the test is run.

FactoryGirl is a tool which allows the easy creation of this test data by 'stubbing' out different models or structures. FFaker is a Ruby gem which works with FactoryGirl and helps create randomised inputs into the model/structure, having generators for creating random names, emails addresses etc. Testing with random data makes the tests more robust with better coverage than using static values.

Sample Factory Girl Factory used in this application for stubbing out a basic Appointment:

```
FactoryGirl.define do
  factory :appointment do
    title { FFaker::Name.name }
    start_at { rand(10.days).seconds.ago.in_time_zone.iso8601 }
    end_at { rand(1.day).seconds.from_now.in_time_zone.iso8601  }
  end
end
```

9.1.3 RSpec Unit Testing

In addition to the API tests outlined above RSpec has also been used to automate testing of other components in the system such as Rails Models and Controllers. The tests syntax is the very same but is much easier to write and execute as the Ruby objects can be queried directly as opposed to having to call the RESTFul API via HTTP

e.g. RSpec for Rails Dog Model:

```
require 'spec_helper'

describe Dog do
  # Create dog stub
  before { @dog = FactoryGirl.build(:dog) }

  # Specifiy test subject
  subject { @dog }

  # Perform Tests
  it { should respond_to(:name) }
  it { should be_valid }
end
```

9.2 Frontend Testing

In terms of front end presentation code, the decision to use Angular.js was in part due to its modularity. It follows a method close to Model, View, View Model (MVVM). Similar to the ROR setup this allows the front end to use different modules and models which again can be individually unit tested.

The evaluation of these tests resulted in either a pass or a fail.

9.3 User Acceptance Testing

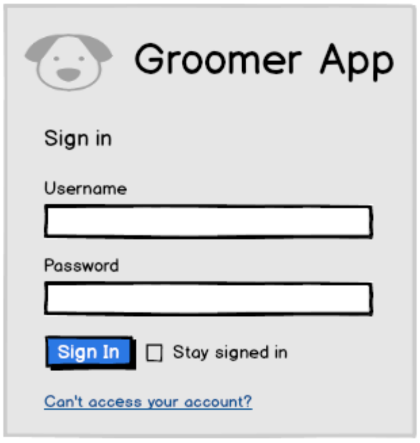
User stories were constructed with sample end users and agreed upon. A story was played out during each sprint and marked successful or not. This was done on an ongoing basis with consultations with the end user. This process reassured the user that progress is being made and in the right direction while giving focus to the development of the project.

Evaluation was conducted on each of these tests with the end user to determine if a process had the expected result.

10 Graphical User Interface (GUI)

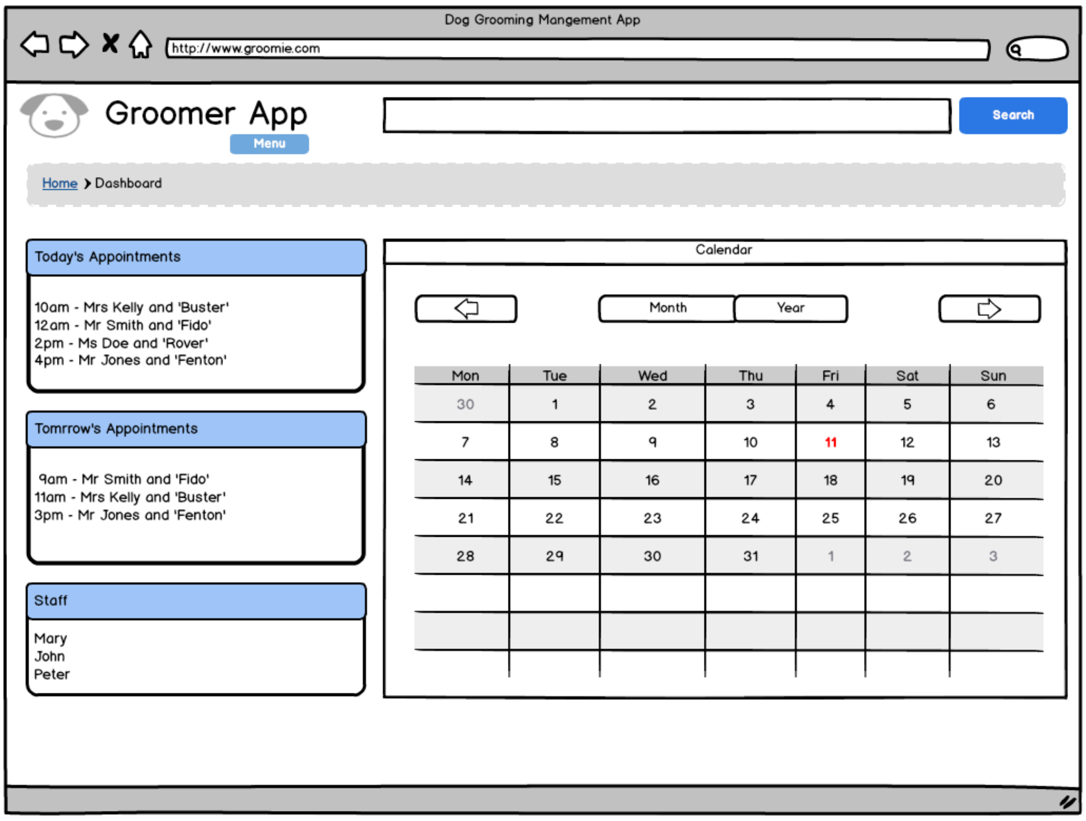
10.1 Early Mock Ups

10.1.1 Customer Sign in Screen



10.1.2 Dashboard

The idea behind the dashboard was to provide the user with a quick overview of the 2 days' appointments and a calendar to quickly add new items. The top of the screen provides a search bar which can search all Clients and Dogs and any other key areas system wide.



Today's Appointments

10am - Mrs Kelly and 'Buster'
12am - Mr Smith and 'Fido'
2pm - Ms Doe and 'Rover'
4pm - Mr Jones and 'Fenton'

Tomrrow's Appointments

9am - Mr Smith and 'Fido'
11am - Mrs Kelly and 'Buster'
3pm - Mr Jones and 'Fenton'

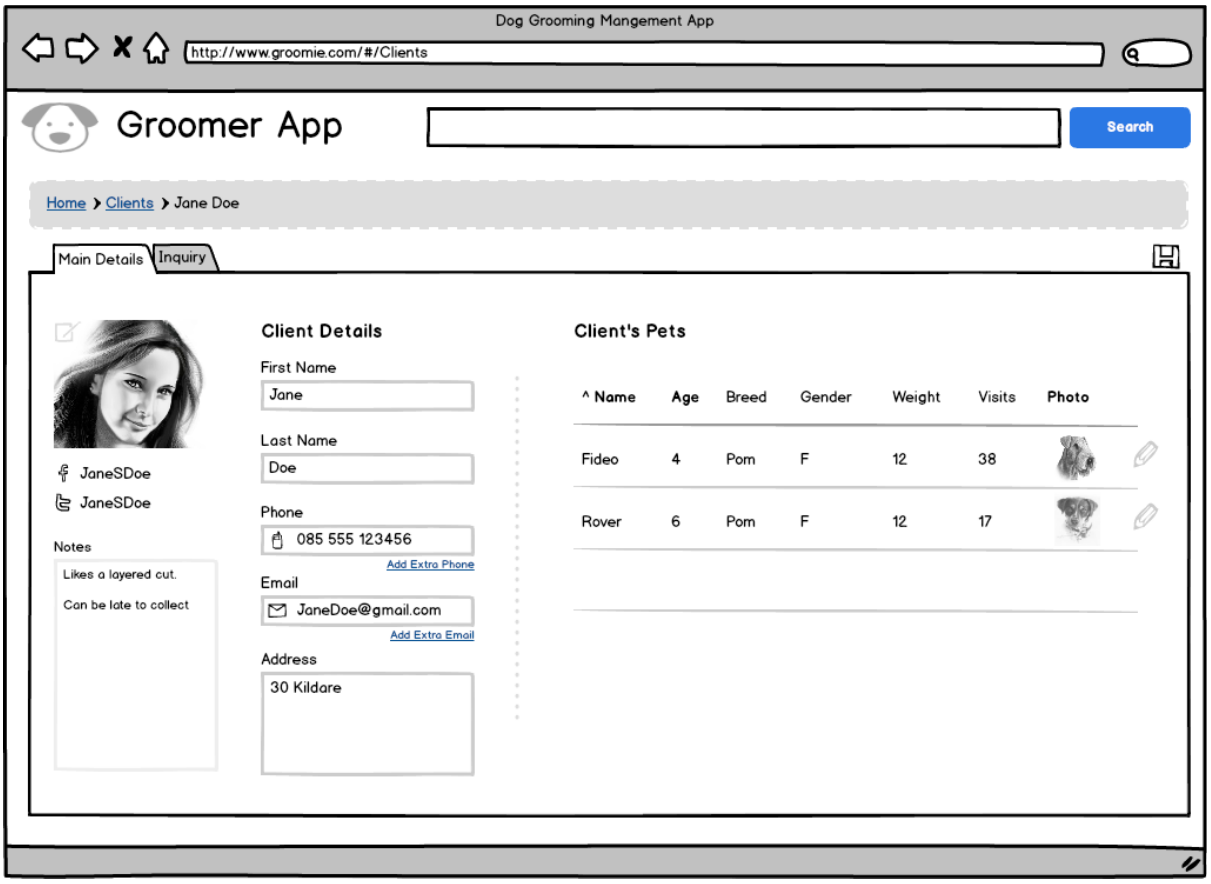
Staff

Mary
John
Peter

Mon	Tue	Wed	Thu	Fri	Sat	Sun
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3

10.1.3 Client Section

The Client section is a very important section to the app, it should display key contact information quickly and friendly as the user must be able to navigate quickly into this section and find the information they need. It should also display a list of any pets owned by the owner to help improve on the spot inquiries and phone calls.


Dog Grooming Mangement App
http://www.groomie.com/#/Clients
Groomer App
Search
Home > Clients > Jane Doe
Main Details
Inquiry

JaneSDoe
JaneSDoe

Notes
Likes a layered cut.
Can be late to collect

Client Details

First Name	Jane
Last Name	Doe
Phone	085 555 123456
Email	JaneDoe@gmail.com
Address	30 Kildare

Add Extra Phone
Add Extra Email

Client's Pets

^ Name	Age	Breed	Gender	Weight	Visits	Photo
Fideo	4	Pom	F	12	38	
Rover	6	Pom	F	12	17	

10.2 Final Version

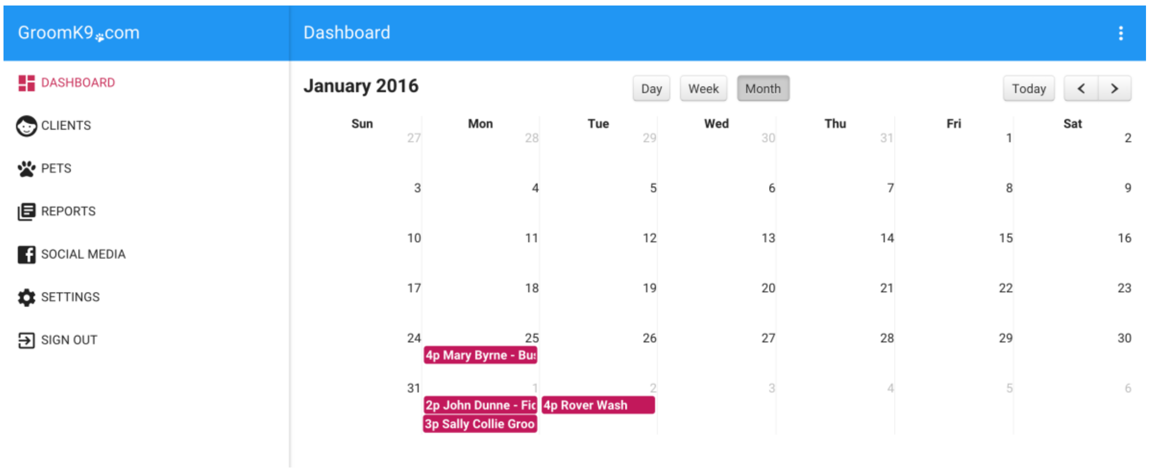
The final version used Google Material Design pattern and utilised the Google Material Design framework. When the mock-ups were created it was proposed to use Bootstrap framework.

The project was moved to Material Design as it had positive feedback from the Client and provided something that was different than the normal bootstrap website. In terms of technology this framework provided components which were more responsive and better fit with AngularJS development methodologies.

On the downside I found this framework to be less mature than bootstrap with early versions failing to even have a working Date picker widget and sparse level of documentation compared to Bootstraps detailed examples.

10.2.1 Dashboard

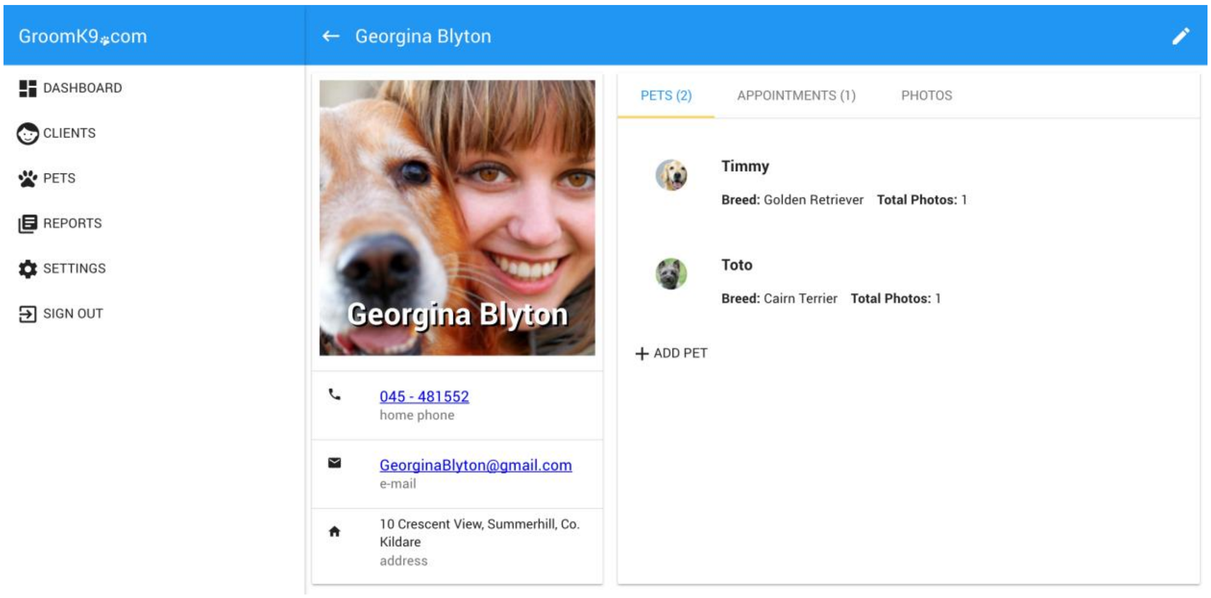
Listing Appointments in pink on the calendar



Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25 4p Mary Byrne - Bu	26	27	28	29	30
31	1 2p John Dunne - Fic 3p Sally Collie Groo	2 4p Rover Wash	3	4	5	6

10.2.2 Client Detailed View

Listing main client details including tabs for additional information including Pets, Appointments, and Photos



045 - 481552
home phone

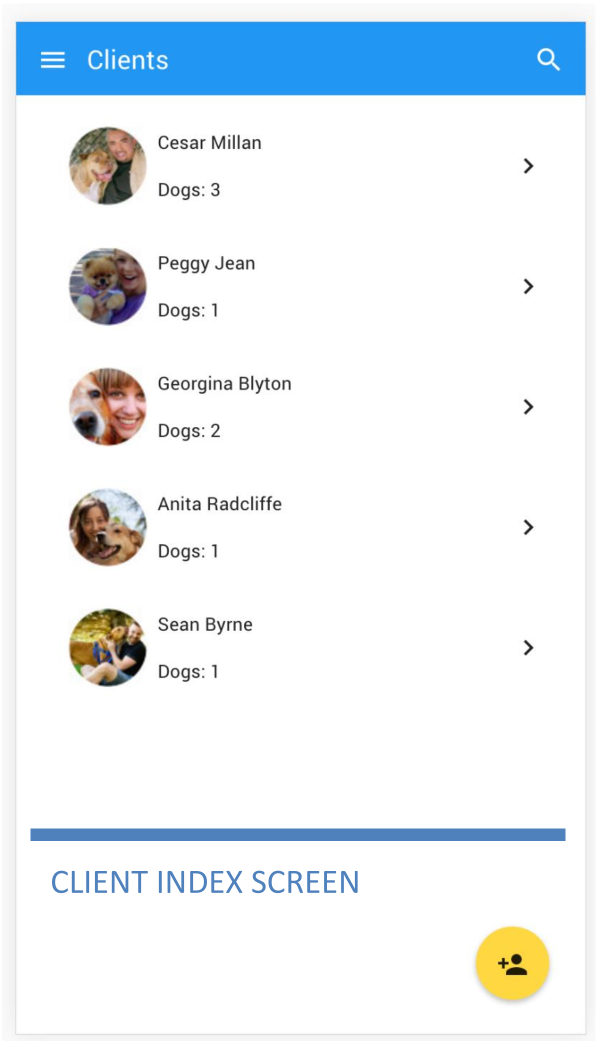
GeorginaBlyton@gmail.com
e-mail

10 Crescent View, Summerhill, Co. Kildare
address

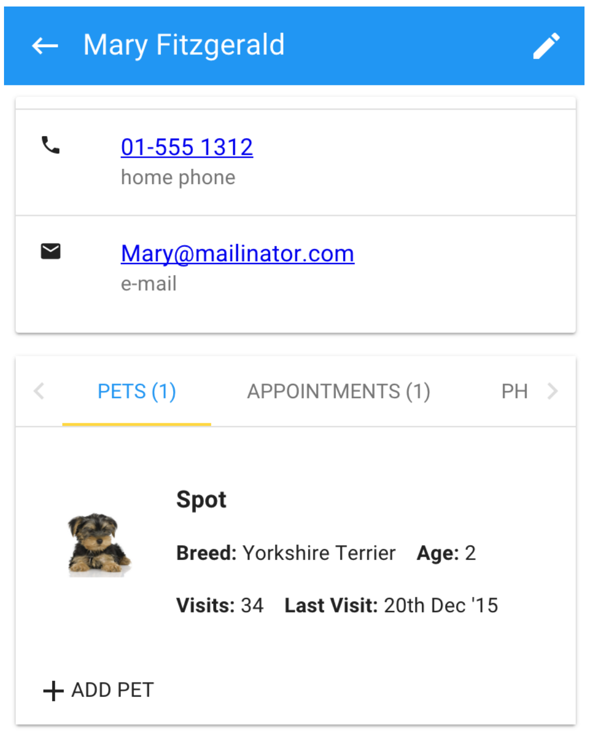
Timmy
Breed: Golden Retriever **Total Photos:** 1

Toto
Breed: Cairn Terrier **Total Photos:** 1

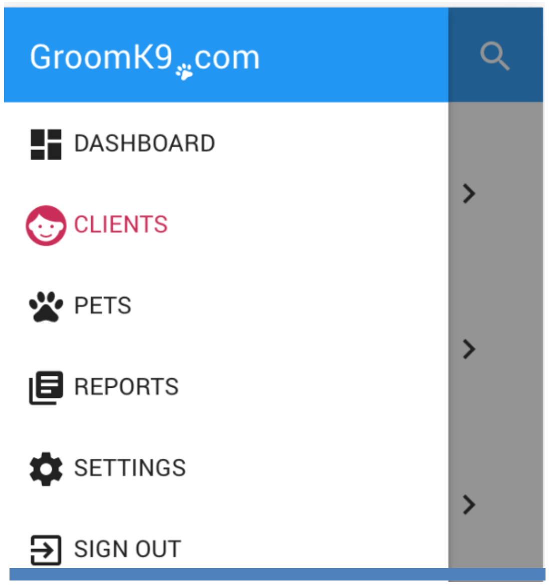
10.2.3 Responsive Samples for Smaller Screens



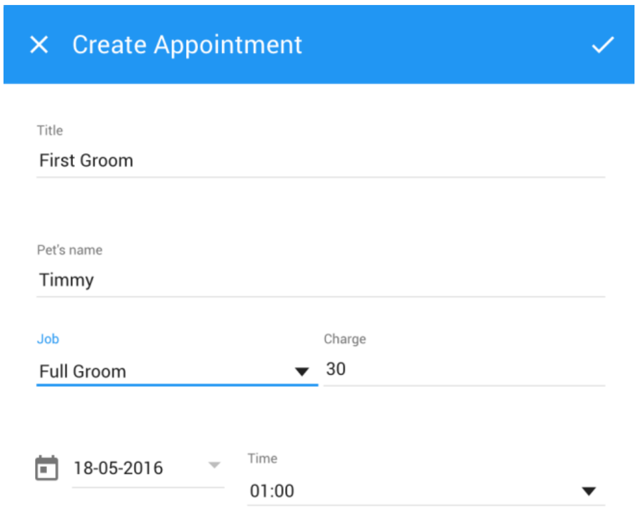
CLIENT INDEX SCREEN



CLIENT DETAIL SCREEN



SIDE BAR MENU

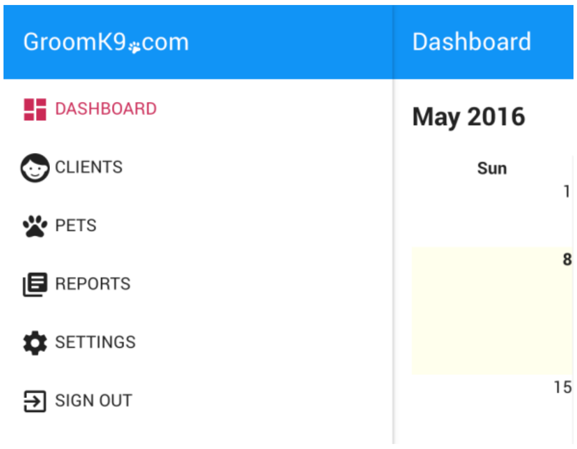


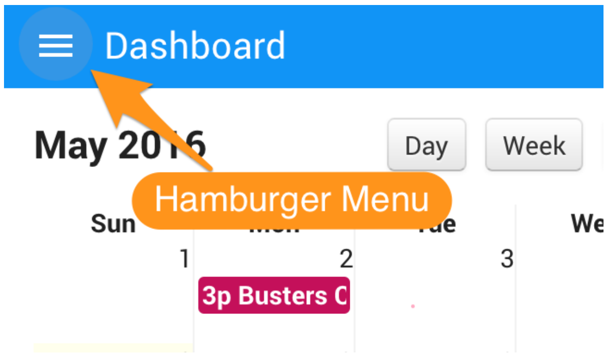
CREATE APPOINTMENT

11 User Manual

11.1.1 Navigation

The primary navigation is located on the left hand side of the screen. It lists the main navigation links. These can be clicked to view each respective sections. On larger screens this menu is pinned open but on smaller screens such as phones the menu is hidden until the 'hamburger' icon is clicked



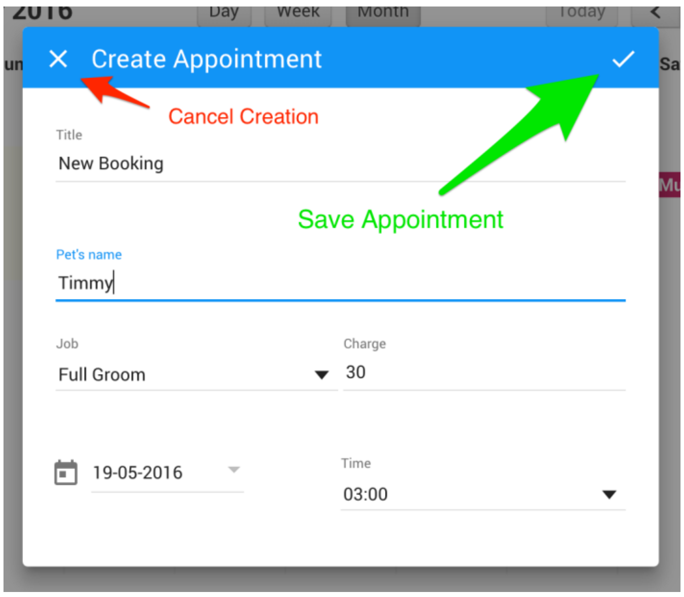


11.1.2 Dashboard

The dashboard section shows any existing appointments on Calendar.

11.1.2.1 Add New Appointment

To add a new appointment, click on any date block on the calendar. A new popup will appear with the date that matched the date clicked. Once the popup appears all sections must be filled in.



Title: New Booking
Pet's name: Timmy
Job: Full Groom
Charge: 30
19-05-2016
Time: 03:00

Notes:

- The pet's name uses an autocomplete option so new options will be suggested with each letter entered.
- Jobs can be setup via the activities settings window

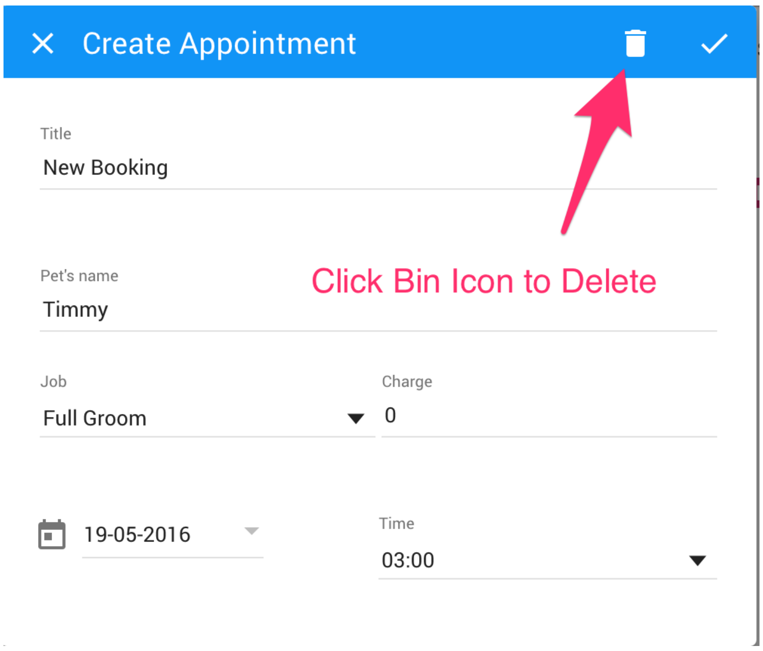
Once the correct details are entered click the Tick icon on the top right to Save and return to the calendar

11.1.2.2 Edit Existing Appointment

Click on the pink appointment listing on the calendar that needs to be edited. A popup will appear. Make any amendments and click the tick save icon

11.1.2.3 Delete Appointment

To delete an appointment, open the appointment as described in the Edit Existing Appointment section. Once open clicking on the Bin Icon will delete the appointment and



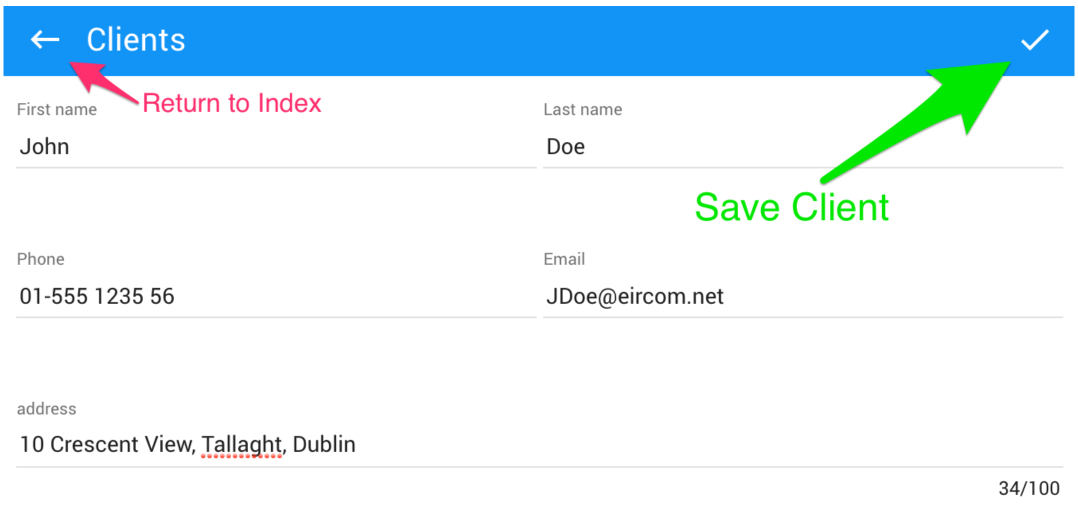
Title	New Booking
Pet's name	Timmy
Job	Full Groom
Charge	0
	19-05-2016
Time	03:00

11.1.3 Clients

11.1.3.1 Add New Client

To add a new client, click the orange add icon and the Client form will appear.

Fill in the form and click the Tick to save. Optionally clicking the Back Arrow will return to the index screen.



First name	John
Last name	Doe
Phone	01-555 1235 56
Email	JDoe@eircom.net
address	10 Crescent View, Tallaght, Dublin
	34/100

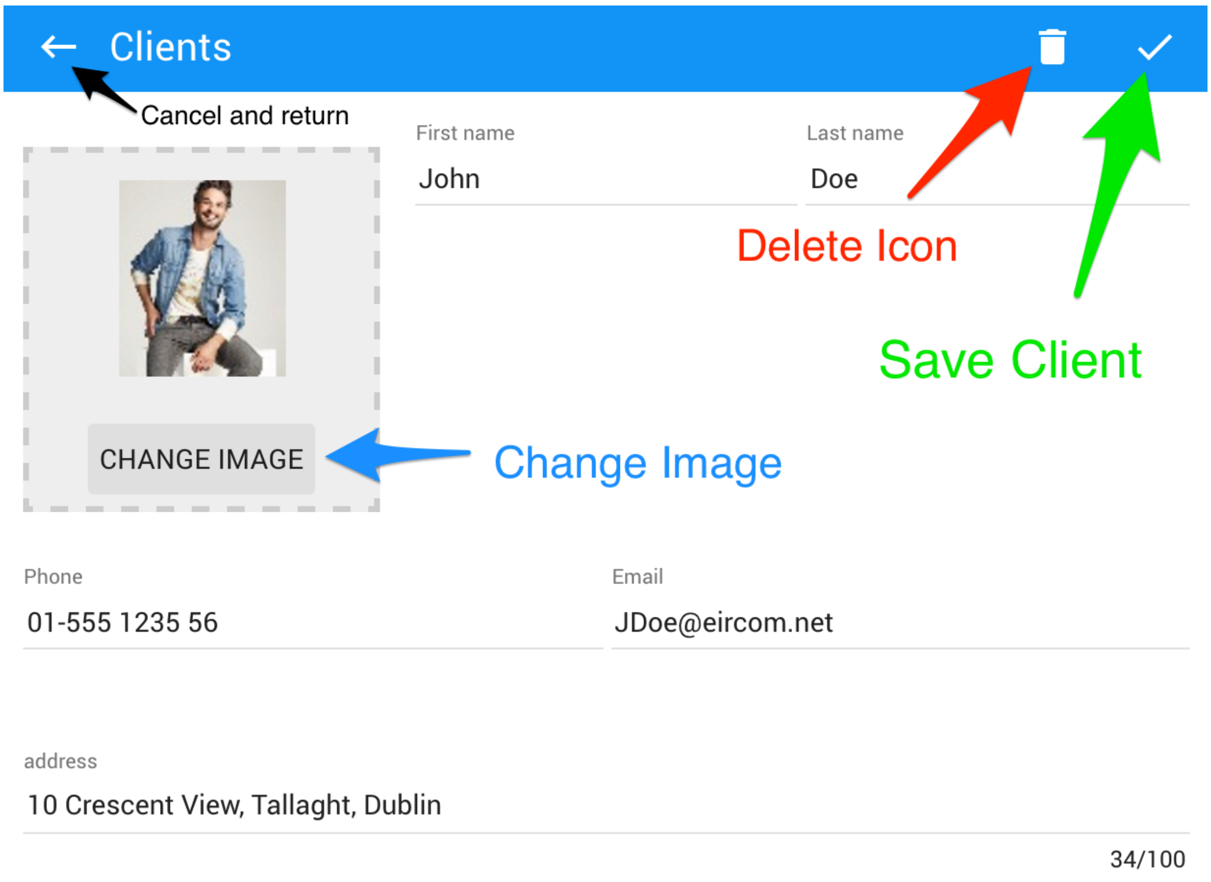
11.1.3.2 View Existing Client

Click on the Client row on the index screen to view their main details

11.1.3.3 Edit Existing Client

View a client then click on the edit pencil icon on the top right to display the edit form. After editing the form click the Tick Save icon to save the details and return to the client view screen.

Note: Images can only be uploaded to existing clients



First name: John

Last name: Doe

Phone: 01-555 1235 56

Email: JDoe@eircom.net

address: 10 Crescent View, Tallaght, Dublin

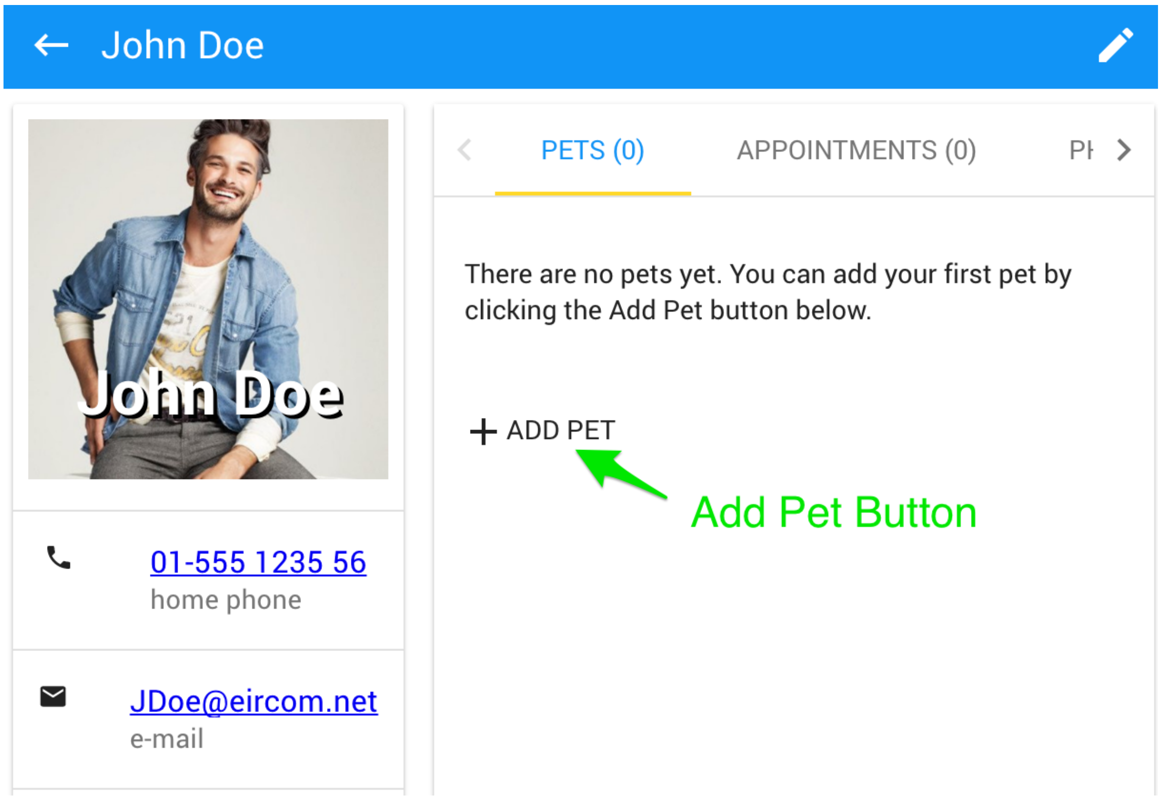
34/100

11.1.3.4 Delete Client

Edit the client as described above then click the Bin icon to delete

11.1.3.5 Add Pet to Client

On the Client view screen click the Add Pet button and fill in the form. Click the Tick Save icon to complete.



There are no pets yet. You can add your first pet by clicking the Add Pet button below.

11.1.3.6 Secondary Navigation Notes

When viewing a Client there are a number of different options. There are primary contact links listed under the Client's photo. These can be clicked to activate the appropriate app.

There is also a series of tabs which can be viewed by clicking on their title. The number in the title indicates the total number of items within.

- Pets – This tab shows a list of pets assigned to the Client
- Appointments – Lists appointments for any of the Clients dogs and is sorted by Dog name
- Photos – Lists any photos that have been saved against the Client's pet records

11.1.4 Pets

11.1.4.1 Add New Pet

As described in the Client's section a Pet can be added by clicking the Add Pet button. Pets can also be added independently of Clients by using the Pets section on the navigation.

To add a Pet, click the add icon and fill in the form. Once complete click the Tick Save icon.

11.1.4.2 Edit an Existing Pet

To edit and existing Pet click on the Pets name in the main Pet index. After editing the form click the Tick Save Icon

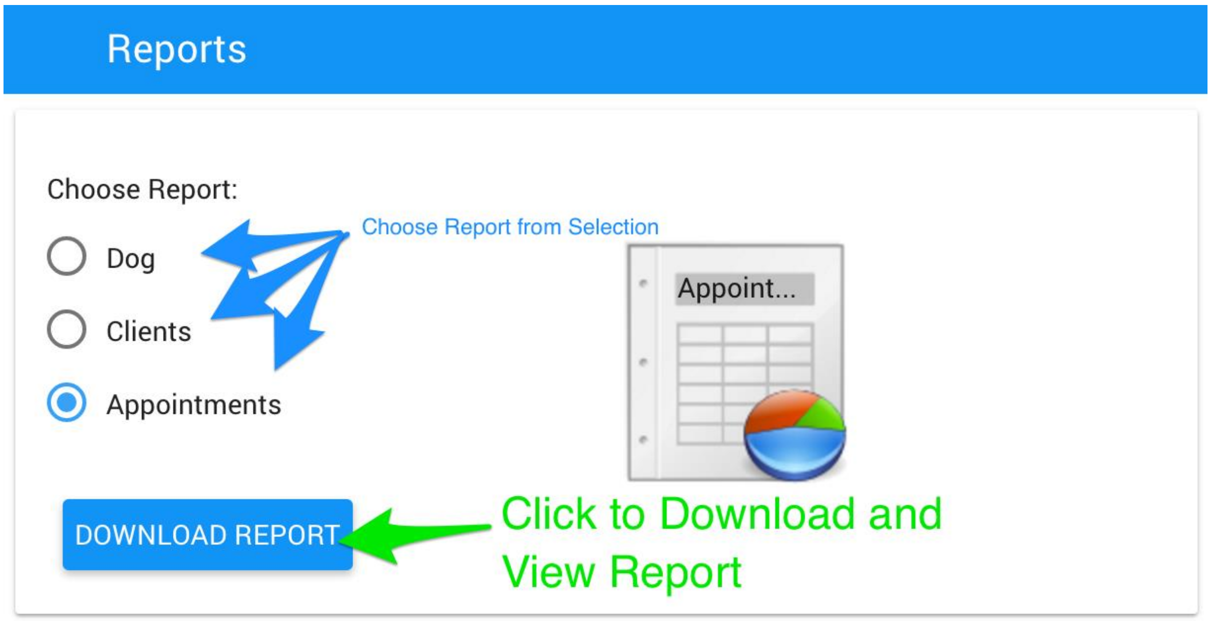
11.1.4.3 Delete Pet

To delete a Pet, enter the edit screen as described above and then click the Bin Icon to delete and return to the Index.

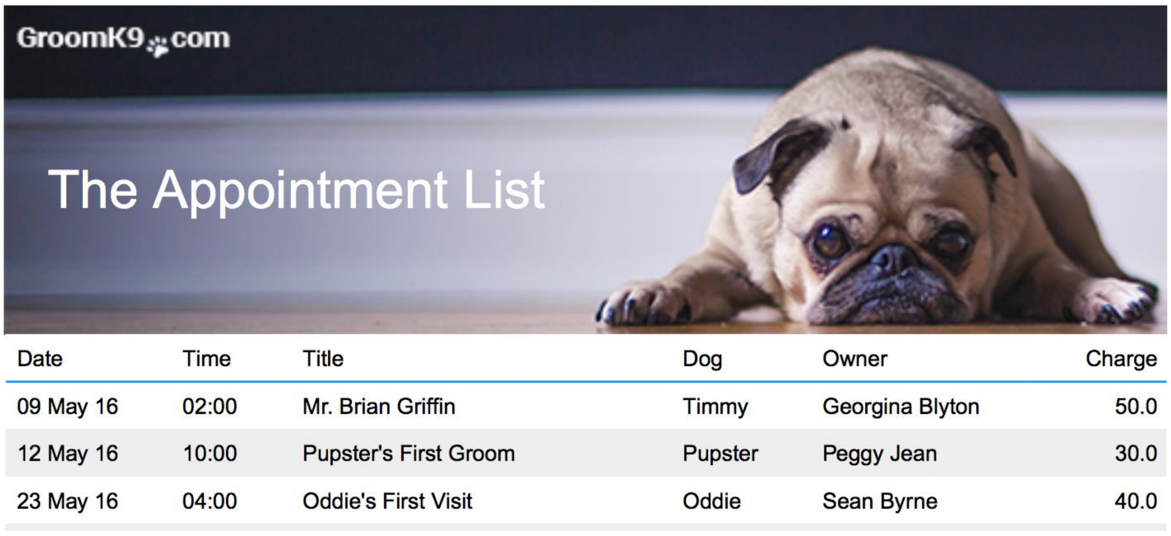
11.1.5 Reports

To create a report simply click the report type that is required. Once selected click the Download Report option to download the report to your devise.

Note: Reports will open in a new window/tab so permissions may need to be set to open these.



Report Sample:



Date	Time	Title	Dog	Owner	Charge
09 May 16	02:00	Mr. Brian Griffin	Timmy	Georgina Blyton	50.0
12 May 16	10:00	Pupster's First Groom	Pupster	Peggy Jean	30.0
23 May 16	04:00	Oddie's First Visit	Oddie	Sean Byrne	40.0

11.1.6 Settings

All settings can be maintained in the same way below. This applies to Activity and Breed settings.

11.1.6.1 Add Activity/Breed

To add a new item simply click the yellow Add button at the bottom of the Index screen. Fill in the form and click the Tick Save icon to Save

11.1.6.2 Edit Activity/Breed

Click on the item on the Index screen to view the Edit Form. Update the form and click Save to update the setting

11.1.6.3 Delete Activity/Breed

To delete an item, view the Edit screen for the item (described above). Once in the Edit view click the Bin icon to delete the item.

12 Conclusions

Following the surveys, it became very evident that cost will play an important role with the provision of this service. While the Dog Groomers were keen on making their lives easier it must be done at a price. A solution to overcome this barrier is to offer a nearly free service with basic options and let the user pay more for special features. This will increase the up take but also allow the service to grow as the Groomers become to relay on it more and see its real benefits.

Overall I am happy with my progress with this project. Choosing so many new technologies for your final year project is a bit of a double edged sword. On one hand the extra pressure to complete a project focuses your attention and gives you motivation to quickly learn new tools, but on the other hand time is very limited in the final year and the weight of trying to complete something while learning how it works can be difficult.

I believe that developing an application as a RESTFul API and incorporating some SOA and Cloud features really does add benefit to the final project. Using an API provides an elegant way to decouple frontend clients and backend servers in a way that a standard all in one MVC be it standard Rails or other can compete with.

AngularJS may be a vast framework which takes a lot of time to get the most out of but once things start lining up you can really see the benefits both in terms of modularised code with dependency management to the product itself feeling fast and responsive. Managing state and caching data also too begin be a better fit without having to force these concepts into a different framework.

Google Material Design I believe is still in early stages and though not yet as mature as something like Bootstrap I feel it definitely has the better look and feel. It also nicely compliments AngularJS in how its components are constructed.

I believe this project has all the core elements needed to manage a Dog Groomer business and with a very strong foundation it will have ample opportunity to develop new integrations and features maintaining very high level of quality and standards.

12.1 Further Research and Development

In terms of future development, I would like this project to grow and integrate with more third party services in particular:

Facebook integration for easy Sign in and Image sharing to help Dog Groomer’s easily promote their business.

Trello Integrations to easily send SMS reminders to Clients about bookings.

Mail Chimp's integrations to organise e-mail campaigns, again for Groomers to help grow their business.

Beyond these integrations I believe this project has a solid foundation and other features can grow organically at the demand of existing users who will continue to suggest new reports and new processes to improve their experience of the application.

It is also worth noting that at the time of writing this document that Ruby On Rails 5 in Release Candidate stages, as is Angular 2. Both of these updates would be interesting to bring into this project.

13 Bibliography

Anon., 20116. *Strategies for Rails Logging and Error Handling - Rails on Maui.* [Online]
Available at: http://www.railsonmaui.com/blog/2013/05/08/strategies-for-rails-logging-and-error-handling/

Anon., 2016. *Angular Material Icons.* [Online]
Available at: https://klarsys.github.io/angular-material-icons/

Anon., 2016. *Angular Stlye Guide.* [Online]
Available at: https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md

Anon., 2016. *AngularJS User Registration and Login Example & Tutorial.* [Online]
Available at: http://jasonwatmore.com/post/2015/03/10/AngularJS-User-Registration-and-Login-Example.aspx

Anon., 2016. *Building an API with Rails: Parsing Data.* [Online]
Available at: http://www.yonbergman.com/2014/08/20/building_api_with_rails_parsing_data/

Anon., 2016. *CSS Guidelines (2.2.4).* [Online]
Available at: http://cssguidelin.es/

Anon., 2016. *Date Picker - Material-UI.* [Online]
Available at: http://www.material-ui.com/#/components/date-picker

Anon., 2016. *Flat Icon.* [Online]
Available at: http://www.flaticon.com/

Anon., 2016. *Google Announces Material Design.* [Online]
Available at: http://www.aumcore.com/blog/2016/04/18/google-announces-material-design-heres-how-it-will-affect-your-website-design/

Anon., 2016. *LumX.* [Online]
Available at: http://ui.lumapps.com/

Anon., 2016. *MailCatcher.* [Online]
Available at: https://mailcatcher.me/

Anon., 2016. *Material Design Lite.* [Online]
Available at: https://getmdl.io/

Anon., 2016. *Material-Design-For-Full-Calendar.* [Online]
Available at: https://github.com/jackyliang/Material-Design-For-Full-Calendar

Anon., 2016. *Netlify.com - Static Web Hosting.* [Online]
Available at: https://www.netlify.com/

Anon., 2016. *Part1: Get started Angularjs,Simple login, best practices, angularjs - YouTube.* [Online]
Available at: https://www.youtube.com/watch?v=4Y6LM4Y_K6c

Anon., 2016. *Prawn: Fast, Nimble PDF Generation For Ruby.* [Online]
Available at: http://prawnpdf.org/api-docs/2.0/

Anon., 2016. *Protocol Workflow.* [Online]
Available at: https://hueniverse.com/oauth/guide/workflow/

Anon., 2016. *REST Security Cheat Sheet.* [Online]
Available at: https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

Anon., 2016. *Ruby on Rails Security Guide.* [Online]
Available at: http://guides.rubyonrails.org/security.html

Anon., 2016. *SVG Morpheus - Morph SVG icons.* [Online]
Available at: http://alexk111.github.io/SVG-Morpheus/

Brough, C., 2015. *Pet businesses will prosper: Industry trends for 2014 and beyond.* [Online]
Available at: http://www.multibriefs.com/briefs/exclusive/pet_businesses_will_prosper.html

Decoster, J., 2014. *WHY AGILE PROJECTS WORK.* [Online]
Available at: http://johandecoster.wordpress.com/2013/01/14/why-agile-projects-work/

Google, 2015. *Google Material Design Specification.* [Online]
Available at: https://www.google.com/design/spec/material-design/introduction.html
[Accessed 09 09 2015].

Heroku, 2015. *Heroku.* [Online]
Available at: https://www.heroku.com
[Accessed 01 01 2015].

Katz, Y., 04. *jsonapi.org.* [Online]
Available at: http://jsonapi.org/
[Accessed 2015 12 04].

Ruebbelke, L., 2016. *AngularJS Architecture: File Structure.* [Online]
Available at: https://egghead.io/lessons/angularjs-angularjs-architecture-file-structure

14 Appendix

14.1 README.md

I use GIT for every project I complete. I find it to be a life saver and to give me piece of mind. In addition to this I also upload my code to an online repository usually GitHub or Bit Bucket.

Out of habit I also create a README.md file which is a markdown file (a simple but quick way to format a document). This has become habit as the README file is always displayed on these online repos as the home page. I find it very useful to fill in key information about the project here as should anyone be looking at the repository it gives them helpful information. Sometimes that anyone is myself a few months or years later and I always give out when I haven't left a document explaining how to get the code up and running.

This is a long introduction to say, I have created two README.md files one for each section of my project and will paste them below but they will also be available in the source, and possibly more up to date (but maybe people reading this document might not have that).

14.1.1 Backend API README.md

K9 - Dog Grooming Management App

JSON RESTful API App

- Ruby Version: 2.2.1

Some Features + Technologies Used

- devise - https://github.com/plataformatec/devise
- action model serializers - https://github.com/rails-api/active_model_serializers
- unicorn - http://unicorn.bogomips.org/
- httparty - https://github.com/jnunemaker/httparty

Running Tests with RSpec

Run all specs

```
bundle exec rspec
```

Run only model specs

```
bundle exec rspec spec/models
```

Run only specs for Dogs Controller

```
bundle exec rspec spec/api/breeds_controller_spec.rb
```

DB Migrations

As this application uses PSQL Schemas migrations must be performed on each user schema. The following task completes this task: rake users:db:migrate

Guides & Docs Used in creation of this app:

- Code School - http://www.codeschool.io
  - Token Based Authentication in Rails: http://blog.codeschool.io/2014/02/03/token-based-authentication-rails/
- Rails Casts - http://www.railscasts.com
  - Securing an API: http://railscasts.com/episodes/352-securing-an-api?view=asciicast
  - plus many more - entire site is a goldmine even if hasn't been updated in a while
- TutsPlus - Hands-On Angular
  - Course covers creation of Rails API and Angular Frontend : https://code.tutsplus.com/courses/hands-on-angular
- EggHead - http://egghead.io/
  - Site is focused on JS and Angular but did have some backend Ruby examples
- Building Rails APIs - some nice information on resouce links, subdomains etc' *Collectiveidea.com : http://collectiveidea.com/blog/archives/2013/06/13/building-awesome-rails-apis-part-1/
- Rails API Mini Guide (general good guide, snippet on caching)
  - http://www.yoniweisbrod.com/rails-api-mini-guide/
- Parsing Data + Active Model Serializers
  - Yonbergman.com: http://www.yonbergman.com/2014/08/20/building_api_with_rails_parsing_data/
- JSONAPI : A specification for building APIs in JSON.
  - Conventions for creating JSON resources: http://jsonapi.org/

14.1.2 Frontend README.md

Dog Grooming Management App Front end using AngularJS and Bootstrap for K9 Restful

Setup + Running

After cloning:

- bower install - will download dependancies
- npm Install
- gulp - process scss and js
- gulp serve - will start webserver

Deploy Guide + Notes

- Run Gulp Build - this creates a generated version in the dist folder.
- Note there is an existing bug in that angular-layout css file is not included in the extract so index.html needs to be updated with a new ref to this e.g. styles/angular-material.layouts.min.css.
- Once the dist folder is complete simply drag and drop into Netlify or any other static hosting service.

Some Features + Technologies Used

- AngularJS v1 (JS Framework MVVM) - https://angularjs.org/
- Angular UI Router (provides stateful routing) - https://github.com/angular-ui/ui-router
- Netlify.com used for front end hosting
- NG File Upload - angular directive to help upload files to brower https://github.com/danialfarid/ng-file-upload

Build Tools + Frameworks

- Angular Material - https://material.angularjs.org
- GulpJS (Build System/Task Runner) - http://gulpjs.com/
- Bower (Package Manager) - http://bower.io/
- Yeoman (Scaffolding tool) - http://yeoman.io/

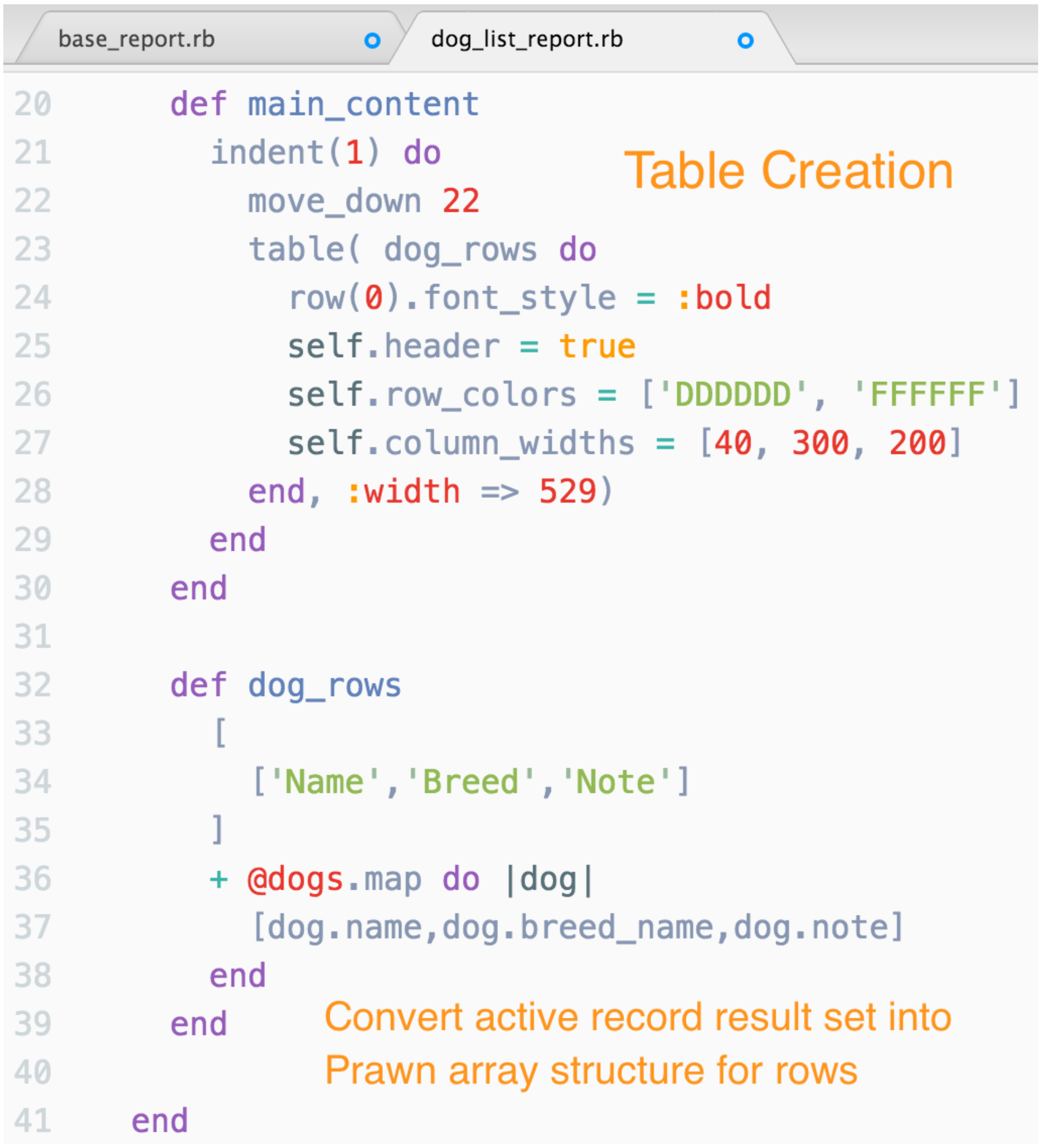
Guides & Docs Used in creation of this app:

- PluralSight - AngularJS: Get Started - by Scott Allen
  - Found this to be the easiest to follow when first learning
  - http://www.pluralsight.com/courses/angularjs-get-started#angularjs-get-started-m1-introduction
- EggHead - http://egghead.io/
  - Site is focused on JS and Angular but did have some backend Ruby examples
- Code School - http://www.codeschool.io
  - Shaping Up With Angular Beginner : https://www.codeschool.com/courses/shaping-up-with-angular-js
  - Staying Sharp With Angular : https://www.codeschool.com/courses/staying-sharp-with-angular-js
  - Soup To Bits - (live coding walkthrough pt1) : https://www.codeschool.com/screencasts/soup-to-bits-shaping-up-with-angular-js
  - Soup To Bits (live coding walkthrough pt2) : https://www.codeschool.com/screencasts/soup-to-bits-shaping-up-with-angular-js
  - Token Based Authentication in Rails: http://blog.codeschool.io/2014/02/03/token-based-authentication-rails/
- Rails Casts - http://www.railscasts.com
  - Securing an API: http://railscasts.com/episodes/352-securing-an-api?view=asciicast
  - plus many more - entire site is a goldmine even if hasn't been updated in a while
- TutsPlus - Hands-On Angular
  - Course covers creation of Rails API and Angular Frontend : https://code.tutsplus.com/courses/hands-on-angular

- scotch.io - Angular and Routing Tutorial - Basic file structure
    - https://scotch.io/tutorials/single-page-apps-with-angularjs-routing-and-templating
    - Site also has other nice tutorials such as animation but did not yet delve into these
- jsonapi : A specification for building APIs in JSON.
    - Conventions for creating JSON resources: http://jsonapi.org/
- Front end
    - Material Design Lite (Light Material Design Framework): http://www.getmdl.io/
    - Lumx (Material Design Framework): http://ui.lumapps.com/
    - MaterialUp (Sample designs) - http://www.materialup.com
    - Ionic (Front End Hyprid apps)- http://ionicframework.com/
    - Bootstrap for Angular - https://angular-ui.github.io/bootstrap/
    - Material Design Guidelines - http://www.google.com/design/spec/material-design/introduction.html
    - Materialize CSS Framework - http://materializecss.com/

14.2 Additional Screenshots

14.2.1 Prawn Table Creation Method


base_report.rb
dog_list_report.rb
Table Creation
Convert active record result set into Prawn array structure for rows

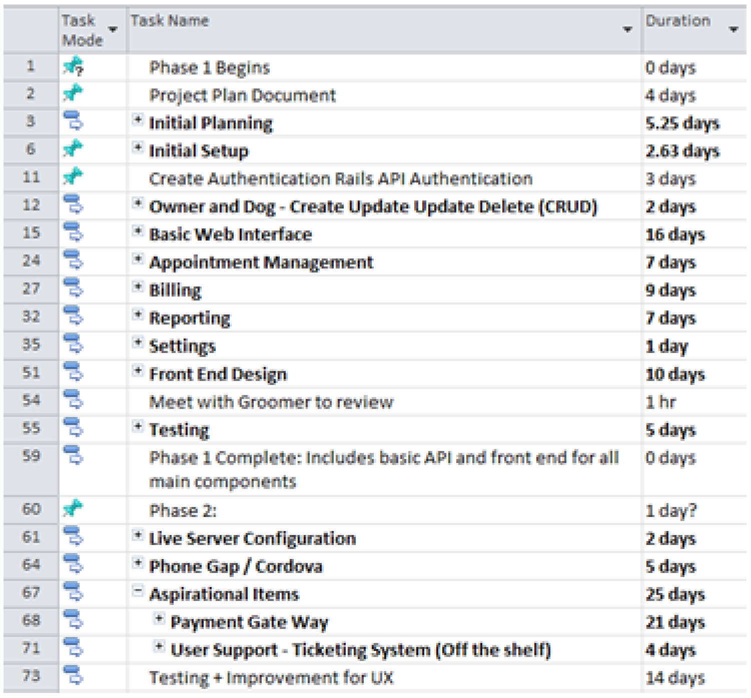
```ruby
    def main_content
      indent(1) do
        move_down 22
        table( dog_rows do
          row(0).font_style = :bold
          self.header = true
          self.row_colors = ['DDDDDD', 'FFFFFF']
          self.column_widths = [40, 300, 200]
        end, :width => 529)
      end
    end

    def dog_rows
      [
        ['Name','Breed','Note']
      ]
      + @dogs.map do |dog|
        [dog.name,dog.breed_name,dog.note]
      end
    end

  end
```

14.3 Project Plan and Microsoft Project

Part of the Implementation section but included at end of document due to large images.

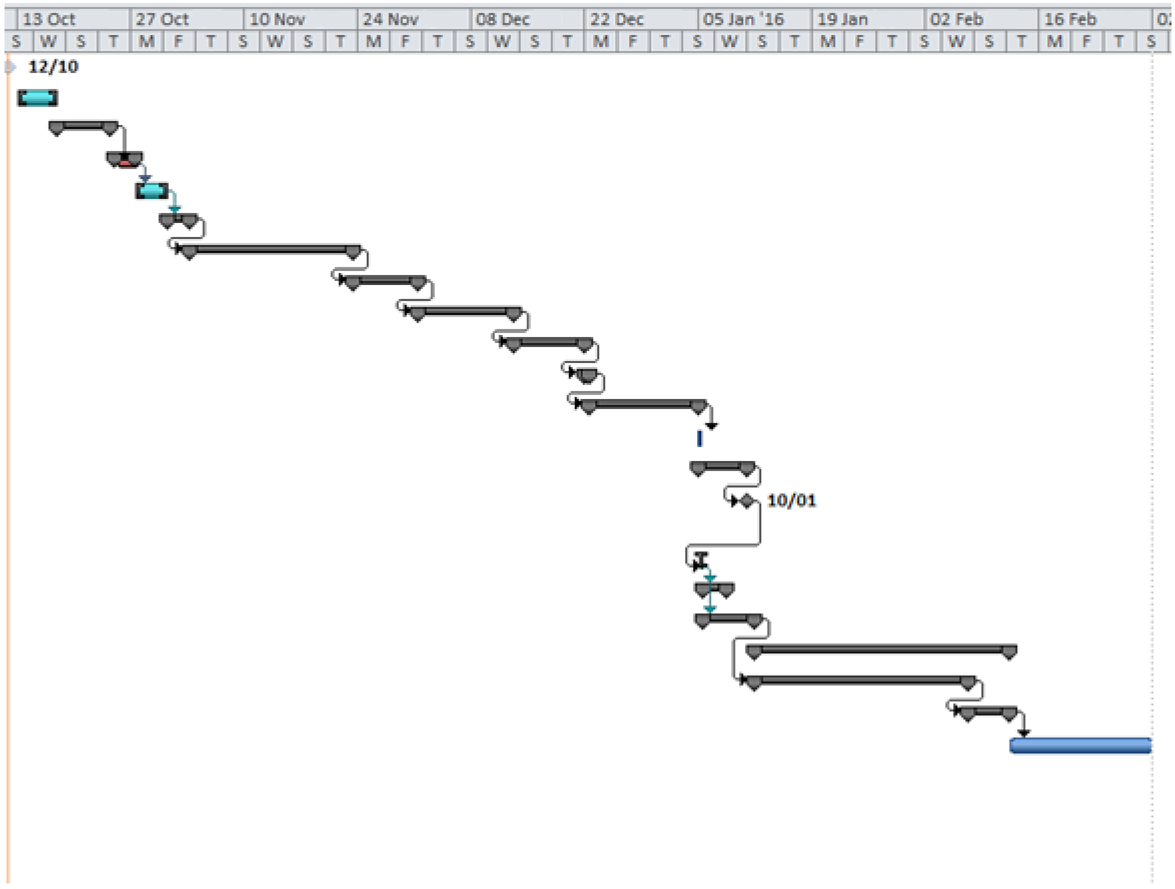
14.3.1 Project Plan Overview

Below shows the MS Project Overview

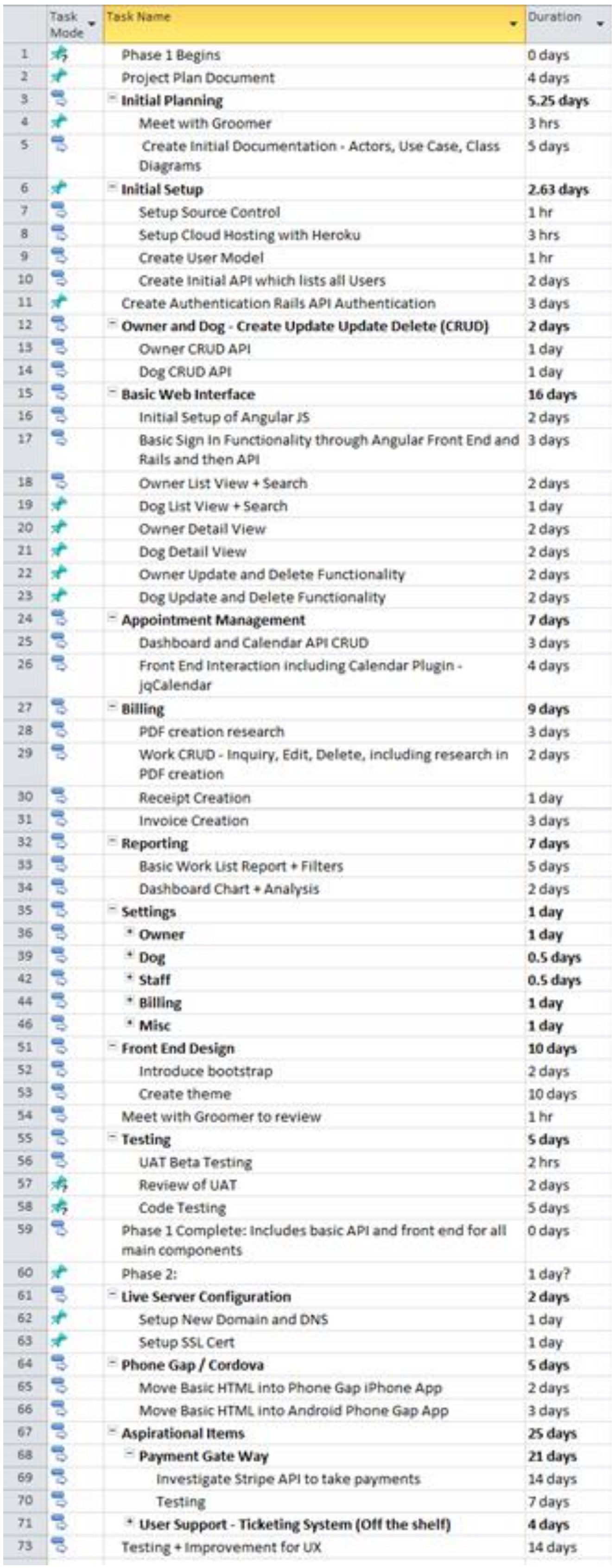


	Task Mode	Task Name	Duration
1		Phase 1 Begins	0 days
2		Project Plan Document	4 days
3		**Initial Planning**	**5.25 days**
6		**Initial Setup**	**2.63 days**
11		Create Authentication Rails API Authentication	3 days
12		**Owner and Dog - Create Update Update Delete (CRUD)**	**2 days**
15		**Basic Web Interface**	**16 days**
24		**Appointment Management**	**7 days**
27		**Billing**	**9 days**
32		**Reporting**	**7 days**
35		**Settings**	**1 day**
51		**Front End Design**	**10 days**
54		Meet with Groomer to review	1 hr
55		**Testing**	**5 days**
59		Phase 1 Complete: Includes basic API and front end for all main components	0 days
60		Phase 2:	1 day?
61		**Live Server Configuration**	**2 days**
64		**Phone Gap / Cordova**	**5 days**
67		**Aspirational Items**	**25 days**
68		**Payment Gate Way**	**21 days**
71		**User Support - Ticketing System (Off the shelf)**	**4 days**
73		Testing + Improvement for UX	14 days

Gant Chart:


13 Oct
27 Oct
10 Nov
24 Nov
08 Dec
22 Dec
05 Jan '16
19 Jan
02 Feb
16 Feb
12/10
10/01

14.3.2 Project Plan Detailed



	Task Mode	Task Name	Duration
1	📌	Phase 1 Begins	0 days
2	📌	Project Plan Document	4 days
3	➔	− **Initial Planning**	**5.25 days**
4	📌	Meet with Groomer	3 hrs
5	➔	Create Initial Documentation - Actors, Use Case, Class Diagrams	5 days
6	📌	− **Initial Setup**	**2.63 days**
7	➔	Setup Source Control	1 hr
8	➔	Setup Cloud Hosting with Heroku	3 hrs
9	➔	Create User Model	1 hr
10	➔	Create Initial API which lists all Users	2 days
11	📌	Create Authentication Rails API Authentication	3 days
12	➔	− **Owner and Dog - Create Update Update Delete (CRUD)**	**2 days**
13	➔	Owner CRUD API	1 day
14	➔	Dog CRUD API	1 day
15	➔	− **Basic Web Interface**	**16 days**
16	➔	Initial Setup of Angular JS	2 days
17	➔	Basic Sign In Functionality through Angular Front End and Rails and then API	3 days
18	➔	Owner List View + Search	2 days
19	📌	Dog List View + Search	1 day
20	📌	Owner Detail View	2 days
21	📌	Dog Detail View	2 days
22	📌	Owner Update and Delete Functionality	2 days
23	📌	Dog Update and Delete Functionality	2 days
24	➔	− **Appointment Management**	**7 days**
25	➔	Dashboard and Calendar API CRUD	3 days
26	➔	Front End Interaction including Calendar Plugin - jqCalendar	4 days
27	➔	− **Billing**	**9 days**
28	➔	PDF creation research	3 days
29	➔	Work CRUD - Inquiry, Edit, Delete, including research in PDF creation	2 days
30	➔	Receipt Creation	1 day
31	➔	Invoice Creation	3 days
32	➔	− **Reporting**	**7 days**
33	➔	Basic Work List Report + Filters	5 days
34	➔	Dashboard Chart + Analysis	2 days
35	➔	− **Settings**	**1 day**
36	➔	+ **Owner**	**1 day**
39	➔	+ **Dog**	**0.5 days**
42	➔	+ **Staff**	**0.5 days**
44	➔	+ **Billing**	**1 day**
46	➔	+ **Misc**	**1 day**
51	➔	− **Front End Design**	**10 days**
52	➔	Introduce bootstrap	2 days
53	➔	Create theme	10 days
54	➔	Meet with Groomer to review	1 hr
55	➔	− **Testing**	**5 days**
56	➔	UAT Beta Testing	2 hrs
57	📌	Review of UAT	2 days
58	📌	Code Testing	5 days
59	➔	Phase 1 Complete: Includes basic API and front end for all main components	0 days
60	📌	Phase 2:	1 day?
61	➔	− **Live Server Configuration**	**2 days**
62	📌	Setup New Domain and DNS	1 day
63	📌	Setup SSL Cert	1 day
64	➔	− **Phone Gap / Cordova**	**5 days**
65	➔	Move Basic HTML into Phone Gap iPhone App	2 days
66	➔	Move Basic HTML into Android Phone Gap App	3 days
67	➔	− **Aspirational Items**	**25 days**
68	➔	− **Payment Gate Way**	**21 days**
69	➔	Investigate Stripe API to take payments	14 days
70	➔	Testing	7 days
71	➔	+ **User Support - Ticketing System (Off the shelf)**	**4 days**
73	➔	Testing + Improvement for UX	14 days

14.4 Journals

14.4.1 October Journal

Summer:

- Completed http://tryruby.org intro tutorial to Ruby.
  Feels very different and confusing especially the lack of parenthesis but has some nice features
- Completed Rails for Zombies and Rails for Zombies 2 Courses on Code School (http://railsforzombies.org/ , http://railsforzombies.org/). Excellent course to do. Very well setup with video, interactive tests, and pdf documentation.
- Setup local testing environment
  Works well once setup.
- Setup and launch website using Heroku
  Lots of little teething problems but works well. Really need to rely on log files for when things go wrong

Week 1

Begin reading Agile Web Development with Rails 4 (www.amazon.co.uk/Agile-Development-Rails-Pragmatic-Programmers). Having taken off the last month of the summer from learning Ruby due to holidays and work I feel a very rusty and takes the week how to remember to do basic things and get back into it.

Week 2

Introduction was given about course and outline of future classes. Good to get started. It was recommended to keep a simple journal to note progress. Next week can discuss project ideas. Continue to code in Ruby using the rails framework. Create basic website with login functionality with a number of different models and views.

Week 3

Lecturers did not have enough time to discuss project ideas with cloud students, will be discussed next week instead. Continue with book and test project. Anxious to discuss project and get it signed off on it. I also have a query over whether to use a language I know well (ColdFusion) or go with Ruby which I am a beginner at but seems to have lots of modern approaches to development and better support for creating an API SOA service which I think would make my project technically much better.

Week 4

Very anxious to discuss project idea and to get it signed off, due to large proposal of work and new technologies. Bombshell dropped that there was a strong focus on the final year specialisation and that proposed projects were unsuited which was in stark contrast to third year answers, which said the final project could be completely unrelated to the specialisation. All project work on hold this week while the confusion was clarified and began looking at alternative project ideas.

This issue was clarified by the end of the week and original project idea was accepted.

Week 5

This week I created project proposal and detailed MS project plan including estimates.
There is a realisation that there are an awful lot of unknown aspects including unknown technologies. I have discussed these in the next section.

Week 6

This week I have spent most of my time focused on other college work. In particular, the Cloud project. For this project I've chosen to use Ruby on Rails and it has given me a good opportunity to complete a smaller project and gain experience using Ruby on Rails as well as consuming an API, using Bootstrap, and setting up hosting which should also help development with my main project.

Work from the last five weeks

Multitenancy Database Storage

Research on Multitenancy Data – a key requirement for my web application

There are 3 main methods to approach this:

Separate Databases OR Shared Database – Separate Schemas

I have used both these methods before (mostly the separate schemas approach) and they work well. Guarantees a complete isolation of each user's data and makes backups and restoration much easier. Can be a little more work for schema updates and is more resource intense. The biggest problem here is that it goes against the Rails paradigm and it generally not recommended to be done with this framework. There are ways to do it but it can cause issues.
Having multiple databases on Heroku can also dramatically increase costs. It is important for this project to keep costs low.

Shared Database – Shared Schema

Involves using the same database and the same set of tables to host multiple tenants' data. A given table can include records from multiple tenants stored in any order; a Tenant ID column associates every record with the appropriate tenant.

The rails framework provides some nice helper functions to manage this such as applying a tenant filter to every query. This may be easier to implement but this approach can be harder to backup an individual tenant's data and restore it. There is also a greater risk of tenant data getting shared with other tenants, which is a big concern.

Unique Postgres SQL Solution - Schemas

Postgres has a unique feature which it unfortunately called 'schemas' which are not schemas in the traditional sense. In Postgres these are more like name spacing so that while you still only have one database you can scope your queries per client. This has the advantage of easily separating your queries while having the reduced overhead of only having to manage one database. I have decided to research and try to follow this method as it seems like an eloquent solution and PostgreSQL has a very good reputation which active development. I have read many recommendations about it over using mySQL or MSSQL. I have decided against using a non-relational database model as I do not think it would benefit my application and there is already almost too much new technology (for me) proposed for this project.

Research on REST APIs using Rails

This has been done using web resources in particular Code School and Rails Casts (excellent video resources). The basics seem straightforward and I have completed a number of mini projects. While I find the basics easy to complete adding in different authentication methods is proving more difficult particularly with the multitenancy. Before I progress with this I feel I have to get the proposed multitenancy solution completed using a standard rails application and sign in process.

Angular JS

Angular is a frontend JavaScript MVC like framework created by google. It allows for two-way data binding between a JavaScript object and a HTML view. I have completed a course on Plural site by Scott Allen (http://www.pluralsight.com/courses/angularjs-get-started) and it seems excellent. It seems like a great fit if developing a REST API with rails as it can lead to very clean code and does a great job at consuming JSON resources.

On the 27th Oct Angular 2.0 was announced. This is long after I had completed the tutorial and was happy in my decision. Google have announced a major change to how Angular 2.0 will work leaving many developers annoyed. It is almost a new language and will be completely different to Angular 1. I have not made any decisions about this, but there are alternative solutions such as a combination of backboneJS and KnockOutJS. The new version of Angular 2.0 will not be released until the end of 2015 and Angular 1.0 still fits my needs well so I may still stick with it. The use of Angular in my project if implemented will not be until other major work is complete so I have time to find alternatives and think about whether I should use it or not.

14.4.2 November - December Journal

Meeting with project sponsor (Nov 12th)

This month I met with my project sponsor Mikhail Timofeev and discussed my project proposal. We have decided that I should look into utilising more existing services rather than re-create them from scratch. I agreed to investigate the possibility of using Instagram for image hosting and Google for calendar functionality.

We also came to the conclusion that I should if possible get in touch with multiple Dog Groomers to interview to get primary feedback.

Finally, it was suggested that I should open a Twitter account with the purpose of creating a history of research with Dog and Grooming related articles about the dog grooming business. I never got a chance to do this due to issues discussed below.

API Research (Month Long)

I continued my research through Nov’ learning how to write an API.

During this time, I got completely bogged down with what I can only assume is Analysis Paralysis. I had hoped to get the basics up and running within a week but could not get my head around how I was going to handle the API authentication and was finding it hard to get something working. Ultimately something that I had hoped would take less than a week took a month to get right. My real issue with this was that I knew what the agreed best way to do API Authentication was but as I had never done it before I had jumped in at too complex a level. What worked in the end was getting a very basic API setup which used arguable the most basic method 'Basic Auth' and once I got this working I was then very slowly able to improve it to use a token based authentication method. At the start I was immediately looking at setting up an OAuth system as it was recommended but ultimately not needed and just lead me down the wrong path.

That's the bad news covered. The good news is that once I did get the Authentication working I was able to jump ahead much quicker in other tasks, which included:

- A public facing website which allows the user to register an account
- On registration it completes another big hurdle I had and that is that it will now create a new PostgreSQL Schema for their details. This is how I will be handling multi-tenancy so I was very happy to get this working; it means that all user data is isolated to their own schema. Getting this right means that I do not need to go the route of having to put a User ID on every record, which means less complex relationships between data. Fantastic!
- On registration it also creates a default API token that can then be used with a client to authenticate with the API.
- Creation of an API service, which lets users Create, Update, Edit, and Delete Client records.
- Creation of an API end point, which lets users Create, Update, Edit, and Delete dogs related to clients. This is quiet nice as the end point to these functions is below the clients end point which means when creating a dog for example we do not need to choose an owner as due to the path we already know it's going to be assigned to a particular client
- The API is also versioned correctly as is best practice so a new version can be easily added at later date if needed without breaking V1 APIs functionality.
- I have been using Google Chromes add on 'Postman' to test the existing API and it works great.
- I have made a very basic client website which calls the API. A challenge here was to overcome the Cross Site Scripting restriction as the API resides on a subdomain e.g. http://api.k9.dev and the main site is http://www.k9.dev. Browsers will not allow API calls to be done between domains like this. I investigated JSONP as an option but it is not suitable as it cannot do post requests. In the end I was able to setup the CORS option and that is working well.
- I have investigated using Instagram's API for storing images in my project but it is not feasible as while the Instagram's API allows for reading and setting data it does not let 3rd parties upload photos, only their official app can do that. I have found a possible alternative in Flickr, which does allow image uploads via their API.
- I have also learned a lot about consuming other API's using the HTTParty library in Ruby and I have learned a lot about handlebars.js and mustache.js for displaying JSON data (typical response from APIs). I learned many of these in creating a project for my cloud computing class and setup the site here: http://www.togatoga.me
- I have not had time to do much more AngularJS development but I hope to do that next month now that the basic API is getting shape. This is the next big concern for me as it can have a steep learning curve.
- Setup http://pow.cx/ for local development.
- Finally, I have completed my midpoint presentation and have a basic working demo.

RESOURCES

During this month especially with the issues I had with learning the API have leaned heavily on the Code School API course and a number of courses on Rails Casts. I had to go through some of these especially the Rails Casts tutorials a number of times.

- #353 OAuth with Doorkeeper - http://railscasts.com/episodes/353-oauth-with-doorkeeper
- #352 Securing an API – http://railscasts.com/episodes/352-securing-an-api
- #350 REST API Versioning - http://railscasts.com/episodes/350-rest-api-versioning
- Surviving APIs with Rails - https://www.codeschool.com/courses/surviving-apis-with-rails
- Soup to Bits: Surviving APIs with Rails - https://www.codeschool.com/screencasts/soup-to-bits-surviving-apis-with-rails
- APIs on Rails
  http://apionrails.icalialabs.com/book/chapter_one
  This is a great resource but it is also one of the big things that lead me astray, in terms of developing an API it starts off very easy but then half way through takes a massive increase in complexity without much notice.

In addition to the above I have read a number of SOA books, while it is all still very new and I'm not there yet I am getting a better idea about how it should work and how my applications can be split up.

One of the biggest realisations, which didn't sit well with me, is that to do SOA you will be breaking foreign key constraints, which has to happen, as they should be independent services. It makes sense when you think about it, but having used foreign key constrains to ensure data integrity for so long it just felt odd giving up that extra safety net.

The other thing that's bugging me is the whole authentication token. In a lot of system this token may never expire. I understand that it can be reaaaaaaaaallly hard to guess but possible especially as more keys become available. I don't understand what's to stop someone accidently using it even if chances of doing so are astronomically huge. It just feels less likely that someone would accidently guess two pieces of information like a username and password. I'm sure the odds must be the same but still feels odd. Some systems obviously do expire tokens and may use Oath as a way for a client to request a new token etc. but a lot of articles don't mention this and seem fine with just letting users creating a unique token that doesn't expire. It's something to find out.

Overview

My biggest concern at the minute is with the scope of the project. For this project to work I need the 3 highlighted things above (REST API using Rails, Multitenant database using Postgres SQL, and a new front end framework AngularJS). Each of these items is very new to me and my concern is that I might get bogged down researching and learning them without progressing the project. It is hard to break the project up into small parts really need to come together to work well. I find it hard to focus on one of these things at a time as they will be so integrated. I have decided to focus a site working with multitenancy logging in first and then complete a basic API.

So far I have used the 'Devise' gem to handle authentication for my websites using ruby. It works really well and handles a lot of security concerns. The problem that I have discovered using this gem is that it doesn't work very well multitenancy. For this reason, I may have to create my own login system with the use of Warden which is what Devise uses internally anyway.
I have allowed until the end of December to learn these items as well as get rough prototypes working. If at this stage, I feel I am too short on time my contingency will be to drop the REST API and Angular aspects and focus on creating a Rails only web application with multitenancy support and focus on creating a rich application here with more standard features. This is something I will need to discuss with my sponsor who has been contacted but we have not met yet.

14.4.3 January Journal

Since the last journal entry authorisation has been completed. It took a bit of back and forth of removing devise but eventually it was possible to add it back in and working with multi-tenant databases and connections.

As this was one of the big hurdles and now working any new time has been spent improving the API and adding basic functionality to the AP.

A nice addition added to the APIs is the use of Active Model serializers. These provide a way to describe how a model should be serialized. Attributes can be selected and associations added. An issue in implementing this was getting around having a circular dependency e.g. defining a serializer for the Dog model where it has many Clients, and then creating a Client serializers which has many Dogs. This can create an endless loop. For the time being Dogs have been included on Client but not vice versa as this is not yet needed and the Client information can easily be found by a second API call. This issue is scheduled for fixing in the new version of Active model serializers.

The create appointment screen has also been improved to allow Users record associated information and includes an autocomplete for selecting an existing dog in the system and dropdowns for activities.

In terms of style I've experimented with a number of different CSS frameworks including Bootstrap, MDL, and Google Material. MDL seems to have better documentation but I ultimately went back to Google Material Design spec framework as I found it to be more responsive and slightly more polished. I steered away from Bootstrap as I've used it a lot before and wanted something new and

14.4.4 February Journal

A large part of February was spent preparing the midpoint presentation and starting point for this document you are currently reading.

Presentation ready to go

New client builds were prepared for the presentation so that the application could be demoed as native apps on an Apple iPad and Android Nexus.

The frontend site has been setup on Netfli.com and a number of deployment bugs corrected. DNS records were updated for the custom domain GroomK9.com

As everything was geared towards this presentation the bulk of the work was spent on the frontend and AngularJS. In truth AngularJS seems to take up all my time when working on this project. It is incredibly broad and has a steep learning curve once over the basics. Thought difficult in places I continue to see the benefits and writing code in terms of state is beginning to feel more normal and using the view model to perform actions as well. This is a big change from previous projects and using jQuery and direct DOM manipulation.

14.4.5 March Journal

March has been dominated by other subjects on the course namely big projects for Distributed Systems and Cloud. Very little work has been completed on the main project this month. I had intended to use reading week to do a bit more but given the deadlines of other CAs this had to take a back seat.

14.4.6 April Journal

One week was used to prepare for Distributed Computing end of year exam.

Angular frontend was updated to include new Breed and Activity Settings screens with further improvements to appointments.

PDF reporting was added to the site. This was mostly a backend job using the Prawn gem to create PDFs. Prawn seems to be a very popular gem for this type of job. However, the recommended method of using it is to use its own Domain Specific Language (DSL). This works quiet well but again it's another thing to learn for the first time in this project where time is a factor but the results are good. I've been able to include custom headers including images and text. One issue I had with this DSL was to add tables as the table code was factored out into a separate ruby gem called prawn-tables. Adding this in involved reading yet more documentation.

I deliberately choose Rails, AngularJS and an API for this project as I knew the pressure would force me to learn them…which it has. However, it's a double edged sword as now that I am getting towards the end of it I am constantly aware that in terms of frontend functionality I could have achieve the same thing with languages I already know in a fraction of the time and that can be frustrating. I still maintain that an API provides so many benefits such as modularity and loosely coupled code and admire this approach. Overall I'm glad I choose to learn these technologies and to do it for the final project, I just don't think I could recommend that anyone else force this much new technologies for their final project as it does add a lot of pressure.

14.4.7 May Journal

This journal includes a bit of end of April which was not written at the time of the earlier diary. These last few weeks have been hugely productive. I think this is in part that all the foundation stuff is done for API and AngularJS. There is a sense that I now know enough to be dangerous and know how to complete new functionality without too much research or trial and error.

The main functionality added this month has been integration with Cloudinary. This was relatively quick and was completed within a couple of hours. The AngularJS side as usual took much longer and took a couple more evenings to get right as it required using a new plugin to handle the image upload from the front end and new file posts to the API. However once these were in place I was able to quickly add image uploads to both pets and clients. There has already been a nice benefit for this in that Cloudinary API is very easy to use and one feature added was the ability to use facial recognition on uploaded images to crop them appropriately. Cloudinary also has some nice transforming options to add a bit of style to my lists.

A new registration screen has been added AngularJS side to allow people self-register for the site. Mailgun Integration has also been added and can now send notification e-mails when someone signs up.

Lastly this month RSpec tests have been added for all API calls and some Models calls. I had looked at this earlier on but this time I sent a good couple of days to get it right and add tests for all functionality. These tests also highlighted some areas which were improved in terms of returning better errors when invalid calls are made. I know tests should be done before any functionality but as I was learning both Rails + API and AngularJS it felt like it was one too many things to worry about. I think however for future projects that I will include these from start, now that I know the ins and outs of the basic functionality and its easy once you know.

As this is the last journal entry my next focus will be getting this document printed as well as the poster for the end of year showcase.

14.5 Poster

GroomK9.com

Dog Groomer Management App

The Market

Pet products and services are a multibillion euro market and it is anticipated to grow by 4.4% in 2016 both in America and Europe.

The Problem

Dog Groomers need to keep and organise detailed records of their Clients and Pets. Cost is a factor as it's a very competitive business with low barriers for entry

The Solution

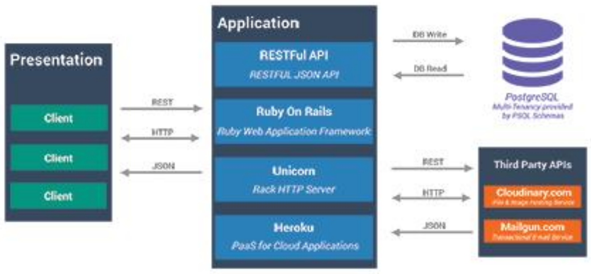
GroomK9.com provides Groomers with easy to use software on the web to manage their business:

- Appointment Scheduling
- Client and Dog information management
- Photo Cataloguing
- Custom PDF Reports

The Technology

All features of the software are accessible via a RESTFul API service which was built with Ruby On Rails.

The initial software client uses AngularJS (v1.5) combined with Angular Google Material Design framework to create a fast SPA which is deployed as a web app and also available as an iOS/Android app.



National College of Ireland

Alan Rice
BSc (Hons) Computing
Cloud Specialisation

14.6 Original Proposal Document

Follows on next page

PROJECT PROPOSAL

GroomK9 com

A Dog Grooming Management System

Alan Rice, X11105038, Alan.Rice@students.ncirl.ie

BSc (Hons) in Computing – Evening

Abstract: Software Project Proposal: Cloud based SOA Dog Grooming Management App

14.6.1 Objectives

The objective of this project is to create a complete Dog Grooming Management System which will help business owners manage the day to day activities of their business these core objectives include:

14.6.2 Customer Management

- Maintain Contact and Address information
  - Provide inquiry options to show previous visits, billing information, and feedback.
- Animal Management
  - Ability to record multiple dogs against their owners including all standard information e.g. breed, sex, date of birth, and chip ID.
  - Inquiry Options
  - Record details preferred cuts, animal personality/behaviour, and potential trouble or sore areas on the dog.
- Staff Management
  - Ability to maintain Departments and Job Positions
  - Record staff primary details including contact information and access rights throughout the system
- Appointment Management
  - Common calendar functionality
  - Allow owner view and manage appointments and assign staff
- Billing
  - Ability to record all grooming activities and charges
  - Create receipts / invoices for customers
  - Options to automatically charge and calculate tax
  - Record and maintain default services in the system for easy entry
  - Common exports to accounting packages such as Sage or QuickBooks
  - Regional and currency preferences
  - Data Analysis / Reporting
- Provide data analysis and statistics
  - Provide export of information through reports
  - Provide report filtering and sorting options

14.6.3 Provision of Software Objectives

Provide a robust software system which can be incrementally updated and improved over time. Have a separation of concerns by using a SOA approach by dividing up the system into manageable components and to decouple front end interfaces from business logic. Facilitate this through the provision of JSON or XML web service API to allow for easier integration and support for multiple device and multiple platforms.

Aspirational or future objectives will include integration with a payment gateway such as Stripe, PayPal, or Realex Payments. Software user help documentation and support features such as

Ticketed Help Desk Support. Functionality to allow the user to create a public facing website maintained via the system.

14.6.4 Background

According to Time magazine Americans are spending more money than ever on their pets. They forked out a record $55.7 billion in 2013—or about $10 billion more than Germany's entire defence budget. This figure is based on American Pet Products Association (APPA) annual report. Dog Grooming makes up $5 billion of this market and is currently growing at 4.8% per annum. In Europe we can see similar sized markets with the European Pet Food Industry Federation (FEDIAF) estimating the Pet Market to be €24 billion in 2010.

(Brough, 2015)

Professional dog grooming has been around since the 1960s but often worked with very limited equipment with groomers being isolated from other groomers. In the last 50 years we have seen a number of key changes in this market. The business has become more professional with set training qualifications, groomer competitions and groomer seminars. There has also been a cultural change in how animals have become more important as pets with over 37% of Americans having a dog (67% having any pet), 36% dog ownership in Irish Homes, and 22% in Britain.

Dog Grooming is now seen more as a skilled trade which has to be learned and crafted over years. This explains why the value of this market is quiet large but at the same time tends to be made up of many small business owners. These groomers typically start out as sole traders who organically grow their business over time while they build a client base and learn their craft.

Despite this growth in the business and a key change in culture which is moving Groomers to use modern technology in terms of their technique and tools, the management of their business tends to still be antiquated. Many of which them tend to use plain old pen and paper ad-hoc management, while some might venture as far as an excel file on their computers. As with all young professions they will eventually become more professional, and that will require them to use system to manage their clients, their staff and time, marketing (particularly social media elements) and accounts.

In terms of market competition there are companies currently providing software that cater to groomers. However, the selection is poor given the market size, many are overpriced, and most of them only offer basic functionality. The software services in place are predominantly desktop based although there is a handful of web software as a service products becoming available.

Given the tight margins of start-up groomers many of the products on offer now are overly expensive and a fixed price. In an industry where a groomer can command a higher wage over time the provision of software should be flexible as their business grow i.e. have a very low setup and running cost with the ability to grow as the business does.

This is why a SOA Cloud based approach is ideal for this industry. It will reduce barriers to entry for Groomers in terms of initial cost by leveraging the cloud. It will do this by allowing the groomer to use a thin client application which does not require any investment in expensive hardware (servers, powerful computers, or the cost to maintain them). It will provide a very fast setup and rather than the groomer having to outlay a large investment they can pay for it as a service on a low ongoing basis with flexibility to grow easily if needed. The improved management of their business will lead to increased efficiency by reducing time lost in booking, improve customer relations by having information on hand and social media improvements, provide key insights into which and what type of customers are the most profitable. It will help them manage their staff and accounts.

This Grooming Management App will be able to provide superior functionality and features than existing marketing providers with a price structure which is more flexible than anyone else on the current market.

14.6.5 Technical Approach

14.6.6 Requirements

14.6.7 Business Requirements

To better understand the user's requirements for this management system information primary research will be conducted with selected business owners as they use prototypes and discuss how they fit into their day to day business. They will be observed using these prototypes and interviewed to establish a plan which will allow them to easily complete their goal.

Based on this primary research information will be gathered and analysed to identify a list of key features and functionality that the service will create. The user's interaction with these products will be mapped using Use Case Diagrams to help establish the scope of the project and their interaction with the system. This will provide a clear direction to what the user will typically have to input (their level of work) and the result that will be returned to them.

14.6.8 Technical Requirements

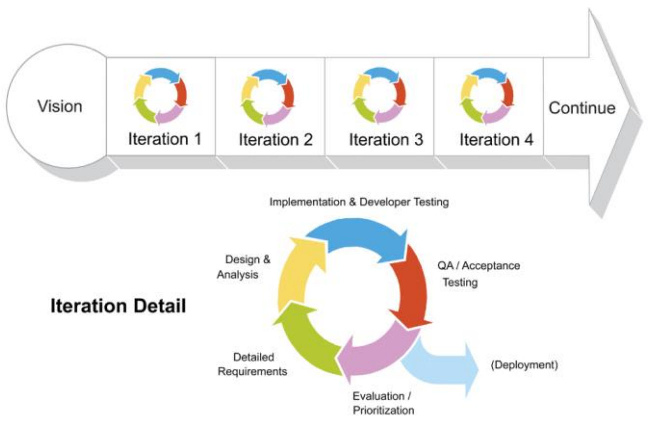
Ongoing research will need to be conducted and reviewed as the project progresses. Given the agile approach methodologies the development should be flexible enough to change focus to using different technologies in terms of hardware infrastructure or software should research prove a positive gain can be achieved.

14.6.9 Regulatory Requirements

These requirements should remain mostly static but online compliance of storage and security of user's private data will need to be in place.

14.6.10 Implementation

This project will be developed using agile methodologies in particular the use of the Scrum management framework. This will involve an iterative approach designing a prototype which will go through the phases of Evaluation, Requirements Gathering, Design and Analysis, and Implementation, and final user acceptance testing.



(Decoster, 2014)

Using this method will improve quality through constant testing (not just at the end); improve customer relations through iterative updates and allowing them to have early input. This will give direction to the project while preventing scope creep.

14.6.11 Special resources required

14.6.12 User Requirements

- A thin client (initially a browser) with internet connectivity.

14.6.13 Environment Requirements

- System capable or running Ruby (Windows, Mac, Linux based PC/Laptop)
- Mac and Developer Licence if developing a native client (aspirational goal)
- Linux hosting provider
  This may include Amazon AWS, PaaS such as Heroku, or a virtual environment using Docker hosted with web host provider such as Digital Ocean or Linode

14.6.14 Research Requirements

Mainly for research and references:

- Book: Web Development with Rails 4 (Pragmatic Programmers)
- Code School – Rails, and Rails API Tutorials
- Plural Sight – Angular JS + Bootstrap Tutorials

14.6.15 Technical Details

14.6.16 Data Layer

The planned data layer will utilize a PostgreSQL database. PostgreSQL is an enterprise object relational database management system. Anecdotal evidence has shown Postgres to give better performance than mySQL the other main database consideration. It is open source and is more compliant with the SQL:2011 standard than MySQL. While mySQL is hugely popular and arguably has better tooling options (in particular their excellent workbench management system), Postgres performance, licencing, and tight integration with Heroku (one of the top proposed hosting options to be used with this project) make it the best choice for this project.

14.6.17 Business Layer

This project will use Ruby with the Rails framework as the primary language which will be used for the business logic layer. Ruby on Rails (ROR) will be responsible for creating the API and web services. The API will be RESTful so that it can easily be consumed by different front end clients and integrate with other services.

Other languages were considered such as Java EE, .net, Node.js, and ColdFusion. ROR was chosen as due to its popularity has incredible documentation and tutorials (both written and video) and a thriving community which will help throughout this project. There are some real nice commercial infrastructures available such as Heroku which will enable quick prototyping with a minimum of setup time. Again due to the popularity of this option the tooling options can improve development quality and time. The Rails framework is also appealing due to its convention over configuration which avoided having large configuration files. The other languages discussed are also very capable and some share similar features, but as a complete package ROR seemed more appealing. Finally, once more stemming from its popularity many problems have already been overcome and there will be no shortage of experienced developers should this project become commercially successful and need to expand.

Other libraries will be evaluated as needed but initially will include Devise (Authentication solution).

14.6.18 Presentation Layer

Initially the presentation layer will be in the form of a web front end which will be viewable by any browser. Given the planned use of a RESTful API the aspirational goal will be to provide different client front ends which may encompass native iOS or Android programs, or use a web wrapper such as PhoneGap/Cordova.

To keep the web front end fast and modular this project will use Angular.JS which will help decouple our presentation (HTML) files from directly binding to JavaScript events. Using Angular.js will also make testing easier due to its modular nature.

A responsive design will be created and may use a CSS framework such as Bootstrap or Prototype.

14.6.19 Middleware

Different middleware will be investigated software solutions will be investigated dependant on which cloud infrastructure approach is applied.

14.6.20 Evaluation

14.6.21 Black-box Testing

Testing will take a Bottom Up Testing approach and leverage many of the Ruby on Rails (ROR) features to perform these tests.

A key reason for opting to use ROR is that it has a lot of testing options baked in. Many of the scaffolding functions that help create structure for your site code also create matching templates which can include test code.

Given the separation of presentation and business logic into Models, Views, Controllers (MVC) design pattern in ROR; it allows developers to follow a Test-driven Development (TDD) process. This means that key components can be unit tested more easily with less complex setups.

In terms of data the ROR framework again has another trick up its sleeve in that it can automatically create and reset its test databases for testing. For a developer this means that they do not have to reconfigure their environment to test a new patch or update.

Finally, ROR can run benchmarking tests which will profile the performance of the system.

In terms of front end presentation code, the decision to use Angular.js was in part due to its modularity. It follows a method close to Model, View, View Model (MVVM). Similar to the ROR setup this allows the front end to use different modules and models which again can be individually unit tested.

The evaluation of these tests will result in either a pass or a fail.

14.6.22 Acceptance Testing

User stories will be constructed with sample end users and agreed upon. A story will be played out during each sprint and marked successful or not. This will be done on an ongoing basis with many consultations with the end user sample. This process will reassure the user that progress is being made and in the right direction while giving focus to the development of the project.

Persona's will be created for each user of the project (Manager Vs Employee) and used to create simple story boards using wireframe models before prototyping begins.

Evaluation will be conducted on each of these tests with the end user to determine if a process had the expected result.

14.6.23 Consultation with Project Specialisation Coordinator

Eamon Nolan:
Great idea for a project, if in the business specialisation would have no problems.
Identify key problem areas early on and don't put off to the end.

Frank Slyne:
Conversation focused on the use of cloud components relation to the specialisation which caused surprise and concern as before the year began the year was informed that the software project was did not have to be related to our chosen specialisation. This was clarified as an error and that "projects will be judged on their own merits and functionality with respect to the marking rubric on Moodle. Students are free to choose a project on almost any topic assuming it is sufficiently complex for a final year project."

Having said that this proposed project is mostly a web application and many of the suggestions made by Frank were valid and will be taken on board where possible, these included the discussion of mashups and load balancers.

14.6.24 Consultation with Academic Staff

The project Specialisation Coordinator will point you to a suitable staff member for consultation. Please include the name of the second academic staff member consulted and a summary of their feedback.

Mikael Timofeev

14.6.25 Proposed Supervisor

Names of academic staff member that has agreed to act as a supervisor for this project.

Mikael Timofeev – Changed midterm to Ralf Bierig

____

Signature of student and date