|  |
| --- |
| BISCHE – 4th Year |
| Groom K9 |
| Product Design Specification |
|  |
| **Alan Rice – X11105038** |
| **04 December 2015** |

|  |
| --- |
| High level Analysis & Design Specification for the Groom K9 Application. RESTFul JSON API application with a rich Angular JS and Material front end. Application is to help manage the day to day running of a Dog Grooming business. |

Contents

[2 Version History 2](#_Toc437035413)

[3 Introduction 3](#_Toc437035414)

[3.1 Purpose of the Product Design Specification Document 3](#_Toc437035415)

[3.2 General Overview and Design Guidelines/Approach 3](#_Toc437035416)

[3.3 Constraints 3](#_Toc437035417)

[3.4 Standards 3](#_Toc437035418)

[3.4.1 RESTFUL JSON API 3](#_Toc437035419)

[3.4.2 Graphical User Interface (GUI) 3](#_Toc437035420)

[4 Architecture Design 4](#_Toc437035421)

[4.1 Presentation Tier 4](#_Toc437035422)

[4.2 Business Logic Tier (Web Server) 4](#_Toc437035423)

[4.3 Database Tier 4](#_Toc437035424)

[4.4 Hardware Architecture 5](#_Toc437035425)

[4.5 Software Architecture 5](#_Toc437035426)

[4.6 Security Architecture 5](#_Toc437035427)

[4.6.1 Authentication 6](#_Toc437035428)

[4.6.2 Authorization 6](#_Toc437035429)

[5 System Design 6](#_Toc437035430)

[5.1 Use-Cases 6](#_Toc437035431)

[5.1.1 Scope 6](#_Toc437035432)

[5.1.2 Description 7](#_Toc437035433)

[5.1.3 Flow Description 7](#_Toc437035434)

[5.1.4 Alternate flow 7](#_Toc437035435)

[5.1.5 Termination 7](#_Toc437035436)

[5.1.6 Post condition 7](#_Toc437035437)

[5.1.7 Database Design 7](#_Toc437035438)

[5.2 Application Program Interfaces 8](#_Toc437035439)

[5.3 User Interface Design 8](#_Toc437035440)

[7 Bibliography 9](#_Toc437035441)

[8 Appendix A: Key Terms 10](#_Toc437035442)

# Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
| 1.0 | Alan Rice | 04 Dec 2015 | Alan Rice | 04 Dec 2015 | Initial Design Definition draft |

# Introduction

## Purpose of the Document

This document defines and records the architecture and system design decisions in order to provide guidance during the development cycle.

This document is created during the Planning Phase and is aimed at the developers of the Groom K9 system.

Some parts of the user interface (UI) may on occasion be shared with the clients, and other stakeholder whose input/approval into the UI is needed.

## General Overview and Design Guidelines/Approach

This section describes the principles and strategies to be used as guidelines when designing and implementing the system.

## Constraints

The Groom K9 application is web based and will require an active Internet connection to operate correctly. There is no planned off functionality for the scope of this document or the initial development cycle.

This project uses a Model View Controller design pattern. One of the goals of the project is that it will provide a platform independent solution with the ability to easily add native front-end clients for both known and yet unknown platforms. The initial view component will in the form of an AngularJS (Google AngularJS, 1) frontend application and use the Google Angular Material design framework for GUI. This setup provides a lot of power and flexibility and can be used on any modern browser. For this reason a constraint for the GUI is that a modern browser with support for CSS3 and JavaScript is require and it cannot support older browsers such as Internet Explorer less than version 10.

## Standards

### 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 will follow the JSONAPI.org specifications. This is a developing standard created in 2013 (Katz, 04).

### http://static.dezeen.com/uploads/2014/07/Google-Material-Design_dezeen_sq.jpgGraphical 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

# Architecture Design

The system architecture for this project will utilize a 3-tier design pattern.

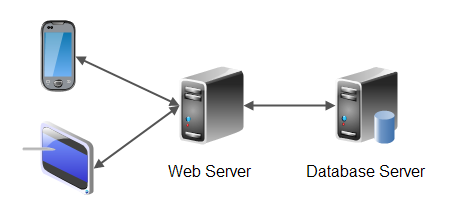


Figure 2: http://tutorials.jenkov.com/

## Presentation Tier

Initially the Presentation tier will be an AngularJS HTML 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.

## Business Logic Tier (Web Server)

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

## Database Tier

All data will be stored in the database server utilizing a PostgreSQL database. Multi-tenancy will be provided by feature unique to PostgreSQL called Postgre 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.

## 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.

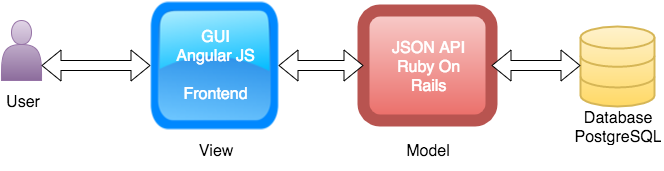
This project will be hosted on Heroku (Heroku, 2015) cloud hosting which provides a Platform as Service (PAAS) architecture. This was chosen as it outsources much of the administration and allows more attention to be given to the development of the project. As it is a virtualized service it will also 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.

## 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 relationships between each item.



## Security Architecture

As the main model of this application will be a Ruby on Rails powered API this project must follow a number of security guidelines including:

* **Ruby on Rails Security Guidelines**  
  <http://guides.rubyonrails.org/security.html>
* **Owasp Ruby On Rails Cheat Sheet** – Complements ROR Security Guidelines  
  <https://www.owasp.org/index.php/Ruby_on_Rails_Cheatsheet>

This application will use proven methods and libraries rather than trying to create our own security libraries.

### Authentication

For Authentication the application will use the Devise gem, which is one of the most popular gems for Authentication. Internally amongst other features it uses BCrypt, which is the recommended hashing algorithm to slow, cracking attempts should a database be compromised. It also features a number of other key security features including hashing and salting, secure token generation, auto lockout out features, and password reset options.

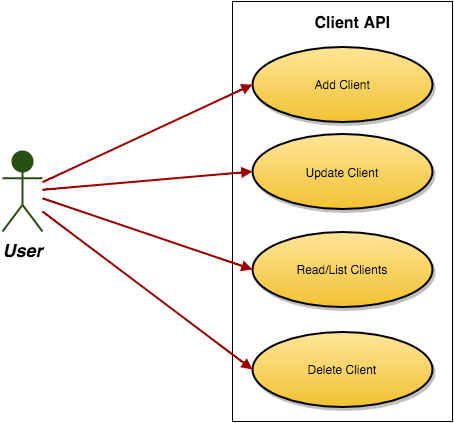
### Authorization

For authorization the Pundit gem has been chosen as it is well establish and allows the creation for an object oriented design approach which is robust and scaleable.

# System Design

## Use-Cases

In terms of use cases this application will provide Create, Read, Update, and Delete methods for Clients, Appointments, and Pets. For the purpose of brevity this document will just display the Client use case but the Appointment and Pet use-case follow the same pattern.



### Scope

The scope of this use case is to explain the process of adding a client.

### Description

This use case describes the input by the user of details relating to the client, which must be entered.

### Flow Description

#### Precondition

The user has selected the "Add Client" option from the main navigation.

#### Activation

This use case starts when a user is presented with the option of the "Add Client" page.

#### Main flow

1. The system presents the user with the option to select a "First Name" and "Last Name" for their client.
2. The system progresses to the "Destination" selection.
3. The user reviews data entered.
4. The user selects the Save button to add the client to the current client list.

### Alternate flow

#### A1: Change of mind

1. The user determines that they no longer want add a client.
2. The user selects the "Back Arrow" button.
3. The system is returned to the client Index page.

### Termination

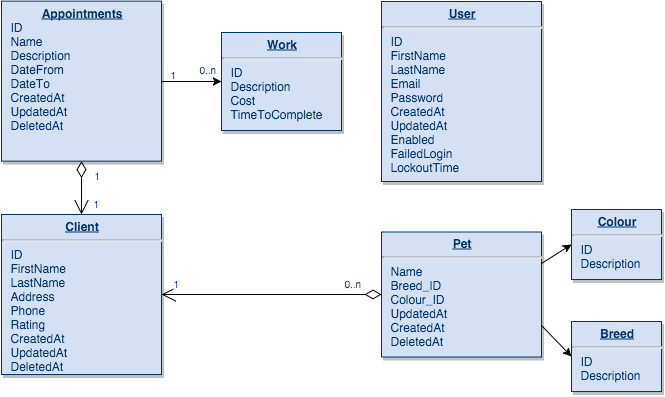
The user has selected to "Save" the client details entered.

### Post condition

The system waits returns to the client index and highlights the newly added client.

### Database Design

The focus of the database centers on the management for a Dog Grooming Parlor. The main entities in this will be the Clients, their Pets, and their Appointments. Below is a part of a database diagram, which includes the main tables and relationships between them. There will be more tables to hold additional information but this represents the core database design.



## Application Program Interfaces

As mentioned earlier in this document the entire backend of this application is an API. It will have 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.  
  
This application will also make use of third party APIs including

* MailGun (<https://www.mailgun.com/>) for sending email
* Cloudinary (<http://cloudinary.com/> ) for Image Hosting
* Facebook for social media needs

## 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>

# Bibliography

Google. (2015, 09 16). *Google Material Design Specification*. Retrieved 09 09, 2015, from Google Material Design Specification: https://www.google.com/design/spec/material-design/introduction.html

Google AngularJS. (1, 1 2015). *AngularJS*. Retrieved 1 2015, 1, from AngularJS: https://angularjs.org/

Heroku. (2015, 01 01). *Heroku*. Retrieved 01 01, 2015, from Heroku: https://www.heroku.com

Katz, Y. (04, 12 2015). *jsonapi.org*. Retrieved 12 2015, 04, from jsonapi.org: http://jsonapi.org/

# Appendix A: Key Terms

The following table provides definitions for terms relevant to this document.

|  |  |
| --- | --- |
| Term | Definition |
| GUI | Graphical User Interface |
| REST | Representational State Transfer  REST is the software architectural style of the World Wide Web |
| RESTFUL | To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) which web browsers use to retrieve web pages and to send data to remote servers.[4] REST interfaces with external systems using resources identified by Uniform Resource Identifier (URI), for example /people/tom, which can be operated upon using standard verbs, such as DELETE /people/tom. source: Wikipedia.com |