FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE
IN
Computer Science

PROJECT REPORT

Name: Jake Fallows

ID Number: K1514226

Project Title: Organisational and Monitoring System for Dog Breeders

Project Type: Build

Date: 24/04/2019

Supervisor: Beryl W. Jones

Kingston University London

Did you discuss and agree the viability of your project idea with your supervisor?	Yes
Did you submit a draft of your proposal to your supervisor?	Yes
Did you receive feedback from your supervisor on any submitted draft?	Yes

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date: 24/04/2019

Signature:

Contents

Declaration 1
Abstract 5
1 Introduction 6
1.1 Background 6
1.2 Problem 6
1.3 Proposed Solution 6
1.4 Aims and Objectives 6
1.5 Project Management 7
2 Literature Review 9
2.1 Introduction 9
2.2 Breeding Licences 9
2.2.1 Suitable Environment 9
2.2.2 Suitable Diet 10
2.2.3 Protection from Pain, Suffering, Injury and Disease 10
2.3 Small Businesses and Online Presence 11
2.3.1 Brand Building 11
2.3.2 Market Reach 12
2.4 Data Protection 12
2.4.1 Data Protection Principles 12
2.4.2 Data Protection Act 1998 13
2.4.3 General Data Protection Act 2018 13
2.5 Conclusions 14
3 Analysis 15
3.1 Introduction 15
3.2 Stakeholder Analysis 15
3.3 System Actors 17
3.4 User Stories 18
3.5 Functional and Non-Functional Requirements 19
3.5.1 Functional 19
3.5.2 Non-Functional 21
3.6 Use Case Diagram 22
3.7 Use Case Textual Descriptions 23

3.8 MoSCoW Analysis 24
3.9 SWOT Analysis 26
3.10 Risk Analysis 27
3.11 Summary 27
4 Design 28
4.1 Introduction 28
4.2 Interface Design 28
4.2.1 Page Identification 28
4.2.2 Site Flow 32
4.2.3 Wireframes 33
4.2.4 Prototyping 33
4.3 Database Design 34
4.3.1 Primary Entity Identification 34
4.3.2 Primary Attribute Identification 35
4.3.3 Class Diagram Development 37
4.3.4 Constraints and Assumptions 40
4.3.5 Data Dictionary 41
4.3.6 Discussion on Decisions 41
4.4 Summary 42
5 Implementation 43
5.1 Introduction 43
5.2 Production Pipeline 43
5.3 Technologies and Libraries 44
5.3.1 Interface 44
5.3.2 Database 45
5.3.3 Libraries Used 46
5.4 Database Implementation 47
5.4.1 Creation 47
5.4.2 Data Population 48
5.5 Server-side Application 49
5.5.1 Application Structure and Development 49
5.5.2 Security 52
5.5.3 Design Conflicts 54

5.5.4 Other Notable Functions 56
5.6 Summary 58
6 Validation 59
6.1 Usability Testing 59
6.2 Performance Testing 60
6.2.1 Database Validation 60
6.2.2 Security Validation 62
6.2.3 Non-Function Requirements Validation 63
6.3 Client Feedback 64
6.4 Summary 64
7 Critical Review 65
7.1 Requirements Review 65
7.2 Project Management Approach 65
7.3 Changes to the Implementation 66
7.4 Time Management Strategies 66
7.5 Conclusions 67
7.5.1 Review of Aims and Objectives 67
7.5.2 Strengths and Limitations 67
7.5.3 Future Work 67
7.5.4 Final Conclusions and Closing Summary 68
Bibliography 69
Appendices 71
Appendix A 71
Appendix B 88
Appendix C 108
Appendix D 114
Appendix E 119

Abstract

With the laws of dog breeding in the UK becoming stricter on breeders to help prevent puppy farms, it is more important than ever for breeders to keep proper track of their dogs and their litters.

Dog breeders often produce a large number of litters, sometimes at the same time, this can often lead to confusion as to which puppy has been reserved by which prospective owner, furthermore it can be even more confusing to keep track of each puppy's health accurately when the puppies have the same parents and can look near identical.

An application would be produced that can track each of the puppies in detail, alert users to discrepancies, link puppies with prospective owners and increase a user's ability to manage their business.

1 Introduction

1.1 Background

The number of breeds registered on the Kennel Club in 2017 was 243,290 (Breed registration statistics, 2018) new dogs registered every year. A large portion of these new registrations are from litters. As the Kennel Club chargers breeders to register their dogs many breeders may not have registered, the true number of litters should be considered too be much larger.

1.2 Problem

This figure can be used to show just how many puppies are bred each year and lends itself as a good baseline for why it can be difficult for breeders to track their litters effectively. When breeder has a new litter of puppies they need to ensure that they can identify each puppy, track health records and prospective owners to help prevent confusion. As the number of concurrent litters increases, the difficulty of these tasks also increases.

1.3 Proposed Solution

A database driven application would be developed to track important details about puppies, such as; weight, potential owners, current payment progress, parents and defining characteristics. This database would be displayed with a web-based application to show; puppy's weights over time to monitor health, puppy's owners to ensure each owner receives the correct puppy, parents to allow the breeder to easily keep track of produced litters and a visualisation of the family tree to help prevent accidental in-breeding.

1.4 Aims and Objectives

Primary Aim: To produce a platform for dog breeders to; improve their ability to ensure prospective owners and guardian homes are correctly matched with their selected puppies, track the family tree of their dogs and closely monitor the health of their puppies as they grow.

Objectives:

- To design and implement a relational database to hold information about potentially thousands of litters and an interface for the user to access the data to allow them to track their puppies effectively.
- To produce a system for breeders to be able to track the health of each puppy in a current litter and flag any puppies that are underweight by a user-specified margin.
- To produce a system for breeders to enable them to track dogs in "guardian homes" so that they can ensure that the terms of the contract for the dog are fulfilled.
- To produce a web application and equivalent mobile application, which can handle multiple breeders being logged in simultaneously, for breeders to input desired data regarding litters and display potential issues.
- To produce a front end "network" view of related dogs that can show up to 50 different images of dogs at one time without a drop in responsiveness.

Secondary Aim: To produce a platform for dog owners and prospective dog owners to allow them to; meet with other owners/breeders for dog socialisation and recreational activities, find local breeders in their area and the characteristics of their breeds and keep in touch with their breeder.

Objectives:

- To implement an event and notifications system, that should update in real time, so that stakeholders can plan walks and other socialising events with other owners.
- To implement a Google maps API for localisation of owners, breeders and prospective owners.
- To implement a "contact breeder" feature for prospective owners, so that breeders can be queried about their upcoming litters quickly and without relying on existing messaging platforms.
- To implement a separate login for breeders and dog owners so dog owners have different functionalities from that of the breeders.
- To implement a "feed" so that breeders can update about potential litters for prospective owners and so that owners can keep the breeder updated about the growth of their puppy, in real time.

1.5 Project Management

The main bulk of the project will be produced by using a sequential design process, Waterfall, to produce a minimum viable product. Once that is produced, a series of user stories will be drafted and presented to the client to be ranked by importance. Additional features will be added to the project using an iterative design process, DSDM, so that new features are considered as they are drafted.

The project is being completed in this fashion as it will be the most efficient way to produce a product that the client will be happy with, with the ability to extend the project in a way that doesn't allow focus on supplementary and unnecessary features until there are no more features on the backlog.

Once using the iterative process, the project will follow a plan of 3-week cycles to assess how the requested work is completed. This allows for a cost/benefit analysis for each new feature and the ability to ignore those that are not beneficial. These user stories and features will be tracked with the use of a Kanban Board technology, such as Trello. This will allow the project to be tracked in a format that encourages iterative development.

During the sequential phase four major milestones have been outlined;

- Complete Front-end
- Complete Database
- Complete Validation Testing
- Complete Project

There are also several intermediate milestones;

- Create a prototype of pages
- Complete database class diagram
- Elicit SQL statements / Link database to front-end

These are milestones that are notable phases of the project that are also defined as tasks, as they require work to be completed at their level to reach the milestone.

2 Literature Review

2.1 Introduction

This literature review will focus on discussion regarding three subject areas that relate heavily to the scope of this project, dog breeding regulations, the effect of an online presence on a small business and the principles and legality of data protection.

2.2 Breeding Licences

As the project relates to the management of a dog breeding business it is important to discuss the regulations that surround such a business. The most important legality related to the business is the need of a breeder to obtain a licence that will allow them to breed and sell dogs. The requirements that affect whether a breeder can hold a licence have changed as of October 2018, these requirements detail that should a breeder produce three or more litters of puppies in any 12-month period or breed dogs and advertising a business of selling dogs then they are required to have a breeders licence. (The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018, 2018).

The animal welfare regulations of 2018 dictate the specific conditions in which breeding dogs should be kept, detailing conditions such as the environments, diets, training and the health of the dogs in question. These areas will be explored in more detail in this literature review, specifically a comparison between the old regulations and the new current regulations.

2.2.1 Suitable Environment

One of the major new changes to the regulations surrounding the breeding of dogs is the environment in which the breeding bitch and the resulting litter is kept. As specified by (New Dog Breeding Regulation, 2019), the housing specified on model conditions have been in use for a number of years. This indicates that, before the introduction of the new regulations in 2018, the conditions for the looking after breeding animals was the responsibility of the local council that distributed the licence.

In direct comparison to the previous regulations, Breeding and Sale of Dogs Act of 1999 and the Breeding of Dogs Acts of 1973 and 1991, the new Animal Welfare (Licensing of Activities Involving Animals) Regulations 2018 indicates specific guidelines that have to be followed in order for a breeder to obtain a licence, these laws dictate that each dog must have access to; a sleeping area free from draughts and a separate exercise area with sufficient space to stand fully, lie down fully stretched out, wag its tail, walk, turn around without touching another dog or the walls of the sleeping area (The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018, 2018).

When comparing the old and new regulations it should be apparent that the Animal Welfare Act of 2018 aims to add significant detail to the rule surrounding the environment in which breeding animals are kept. This aims to ensure that all breeders are subject to the same, well defined, restrictions to give breeding dogs and respective litters adequate space to live in humane conditions, compared to the old regulations which were defined by the local council and therefore were subject to change dependant on the councils inclination and as a result were not consistent across all breeders.

2.2.2 Suitable Diet

Regarding the diets of breeding dogs and litters the (New Dog Breeding Regulation, 2019) also specifies that the feeding regulations, in the same vein as the housing, was dictated by the local council rather than the government.

In the new regulations it is specifically stated when puppy's should start being weaned off of the mothers food at the correct times, each breeding bitch and puppy is fed appropriately and that each puppy is monitored so that it ingests the correct share of the feed provided (The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018, 2018).

The regulations on the diets of the dogs in the care of the breeders is not as specific as those dictated for housing, relying on the breeder themselves to gauge when is a suitable time to wean the puppies and what is the appropriate feed and quantity for both the breeding bitch and the puppies. Because of this it is clear that these new regulations are unlikely to have as much of an impact as that of the environment changes.

2.2.3 Protection from Pain, Suffering, Injury and Disease

From an ethical standpoint the health and well-being of the dogs in the care of a breeder is the most important aim of the regulations surrounding dog breeders and animal welfare. The conditions and diets of dogs are two minor factors that contribute to the overall health of an animal. Whether an animal is actually healthy and in good standing could be considered to be most affected by the ability of the owner to protect the animal from injury and disease. The new regulations set out a large set of specific rules regarding this concept.

Previously the idea of protecting the animal in any one individuals care was set out in the Animal Welfare Act of 2006. This specified that should a person cause unnecessary suffering through "an act of his, or a failure of his to act." (Animal Welfare Act 2006, 2006). The (Breeding and Sale of Dogs (Welfare) Act 1999, 1999) extends this notion to not only include animals in a person's care but also to breeding bitches with regulations such as; bitches should not be mated under a year in age and should not produce more than a single litter each year with a total litter count of no more than six in their lifetime. These rules ensure that the breeding animals themselves are kept and treating in a fashion that ensures their well-being whilst they are in the breeders care.

The new regulations improve upon the older concepts in a handful of ways. Firstly, they extend the obligations of the breeder to the breeding bitch so that as well as recording the previous details they also have to ensure that the number of matings and the age at said matings are recorded. It goes further to state that all vet treatments are recorded and that the number of litters that a dog can produce is reduced should that dog have undergone C-section surgery prior. (The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018, 2018). These new regulations can only be viewed as an improvement on the previous, they do not remove nor redefine the existing rules but seek to extend upon them so that the edge cases, such as C-section dogs, are accounted for.

The new regulations also seek to ensure that puppies are also treated in the same manner as the breeding bitches. Previously, the legislation relied upon the general statements laid out in the Animal Welfare of

2006 so that owners ensured that the animals in their care did not encounter any unnecessary suffering. In the new regulations it is detailed more specifically as puppies have needs that are more involved than that of normal pets. Because of the fragility of a new-born animal "the health, safety and welfare of each dog must be checked at the start and end of every day and at least every four hours during the daytime."(The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018, 2018). As well as this, breeders are also required to record the puppies' date of birth, sex, weight and the number of puppies in the litter. Recording the weight of the puppies allows the breeder to better keep track of each puppies' health and give the information required to react accordingly, opposed to the old rules in 1999, this method ensures that instead of breeders estimating the health of their dogs, they are forced to record details that will show without any reason of a doubt, the health progression of their dogs.

2.3 Small Businesses and Online Presence

Small service businesses "ultimately all measure success by their ability to build offline relationships – and the web can play a huge part in that." (Shoosmith, 2012). Small businesses rely upon their customer base far more than that of medium and large businesses, this is due to small businesses often holding a local customer base where the larger sized businesses tend towards nationwide and global customer bases. Because of this it is hugely important for small businesses to not only bolster good relationships with their consumers but to also be easily accessible to potential new customers and to reach as many customers in their local area as possible to maximise the success of their business.

2.3.1 Brand Building

Before considering how an online presence effects a business's brand it is important to understand what a brand actually is. According to (Keller, 1993) a brand is a combination of names, logos, symbols, colours and characters. This is to say that a brand is a collection of graphics and concepts grouped in a fashion that is recognisable to consumers and acts as a front facing representation of the company that it is associated with. One of the important notes regarding brands is the idea that it holds brand equity, this is the value that is associated with the customer's perception of a brands name in place of the value of the product or service itself. Managing this perceived value is the reason that companies can sell products at prices that far exceed the cost of the components and man-hours it took to produce. "SMEs in most economies are now more than ever exposed to greater competition and therefore require the development and management of sustainable brands which are able compete both locally and internationally" (Odoom, 2016).

"Building customer-based brand equity requires the creation of a familiar brand that has favorable, strong, and unique brand associations." (Keller, 1993). Here Keller provides a link between brand equity and brand associations, where the association is not a push for the user to buy the product but instead provides the user with familiarity and differentiation from other products. (Chiang et al., 2008) states that the "most important key to an online business's success is to create 'brand value' for the customers." Brand association is one method of managing brand value by allowing the company to create a positive image of themselves in the minds of their consumers.

"As the number of online companies multiplies, only the organizations that have built online brands can enjoy stronger customer-company relationships"(Chiang et al., 2008). This, coupled with the ideas set out

previously, means that as more companies move online it becomes more important to establish an online presence for a company to foster a strong brand equity and that a strong brand equity is equally important for a company's success.

2.3.2 Market Reach

(What is market reach? definition and meaning, n.d.) describes market reach as the "estimated number of the potential customers it is possible to reach through an advertising medium or a promotional campaign." In the scope of this discussion it will be considered in the domain of advertising mediums rather than promotional campaigns, however it is worth noting that an online presence helps to increase a small businesses ability to run promotional campaigns in a similar fashion to the way that advertising capabilities are enhanced.

In the UK, as of 2018, there are 47.56 million internet users, ranging from 1.25m users in Northern Island to 6.67m users in the South East (Office for National Statistics (UK), 2018). From this it is clear to see that there are a significant number of internet users in the UK. With a population of 66.47 million (United Kingdom - Total population 2012 to 2022 | Statistic, 2016) this means that over 70% of people in the UK are internet users, without an online presence this is a large bracket of potential customers that are being overlooked. As small businesses thrive with larger customer bases, having the ability to reach 70% of people in the area is a big advantage for businesses.

2.4 Data Protection

Data protection presents itself as both a legality issue as well as a methodology for the protection of data from modification and loss. Before exploring data protection in this section it is important to clearly define data (What is data?, 2016) defines data as "information that has been translated into a form that more convenient to move or process." However, when discussing data in the context of data protection, data can be split into two subsections for definition; personal data and sensitive personal data. This split exists as such because of the way that data protection standards change when dealing with the two separate types of data. This concept will be explored more in the sections detailing the two legal acts that have been passed relating to data protection; Data Protection Act 1998 and General Data Protection Regulation 2018.

2.4.1 Data Protection Principles

There are eight main principles when regarding the protection of an individual's data, Richardson (2003) details these eight principles as such:

1. Fairly and lawfully processed
2. Processed for limited purposes
3. Adequate, relevant and not excessive
4. Accurate
5. Not kept for longer than necessary
6. Processed in accordance with the data subject's rights
7. Secure
8. Not transferred to countries without adequate protection

Whilst it these principles were set out in the 1998 Data Protection Act, Laybats and Davies (2018) outlines the fact the new regulations doesn't replace the data protection principles but rather enshrines them, making them as relevant today as they were in 1998. These principles have survived 20 years of scrutiny without being changed, this shows that they are both fully comprehensive and do not hold any superfluous detail.

2.4.2 Data Protection Act 1998

The collection of worker information was used heavily for management functionality so that the company could make informed decisions about their direction, however as technology has improved the ability to collect larger and larger quantities of data has grown (Ford, 1999). As time progressed this data collected branched into topics such as health, ethnicity and sexual orientation so that the companies could prove that they were adhering to different discrimination laws. "But the potential for abuse is clear. Information may be collected for purposes which are irrelevant to performance at work" (Ford, 1999). Thus, as a result of this potential, for abuse a new legislation and definition of what data is was needed to help protect individuals.

"The first transitional period under the Data Protection Act 1998 ended on the 23 October 2001" (Richardson, 2003). This new act brought the eight principles of data protection, outlined previously, as well as introducing much stronger rights for the individual, allowing access to the data that was being held on them and how that data was intended to be used (Ford, 1999). With the act's first principle, the fair and lawful processing of data, a new concept was introduced, the data controller and data processor. "The data controller determines the purposes for which and the means by which personal data is processed." Whereas, "the data processor processes personal data only on behalf of the controller" (What is a data controller or a data processor?, n.d.). The data controller is usually the party that is collecting the data, for example; an employer or a website. The data processor is typically a third party that is handling the data, these new definitions helped establish who is responsible for the correct and lawful handling of the data collected.

2.4.3 General Data Protection Act 2018

One of the major changes with the new regulations, introduced in October of 2018, is the idea that when an individual's data is being collected they must be given clear privacy notices to highlight that their data is being collected and that it will have an impact upon their privacy, so they can make informed decisions on whether they give consent (Laybats and Davies, 2018). This change is set to increase the clarity of data collection and force the requirement of an opt-in structure to data collection, previously individuals would have their data collected unless they specifically sought to be protected. This is an important improvement as it moves the control of data collection into the hands of the individual rather than the corporation.

This idea of giving the individual control is extend further with the idea that "this consent can be withdrawn at any time"(Laybats and Davies, 2018). The ability to withdraw the consent of data is important as it prevents corporations from continuing to collect data beyond the scope of the initial agreement should the individual not consent to the change. It acts as a preventative measure, ensuring that individuals cannot be shoehorned into agreeing to data collection that they have no way out of. This

change was not required with the opt-out structure that existed previously as the very nature of opting-out means that users can prevent their data being collected at any point.

Laybats and Davies (2018) form an interesting conclusion from these regulation changes. "In the past, individuals have not been so savvy and aware of their personal data use and misuse, organizations have (if they wished to) been able to monitor, store and analyse vast amounts of personal data unquestioned." The limitation of corporation's ability to scrape vast amounts of data is a massive leap in a positive direction regarding data protection. Article 8 of the Human Rights Convention illustrates that people have a right to hold a private life, data protection seeks to act as an extension of this right and is making large steps in the correct direction to do so.

2.5 Conclusions

During the course of this literature review three topic areas have been covered, this section seeks to summarise the content covered in each section and to provide an accurate conclusion for each topic.

The new breeding licence regulations introduced in 2018 reinforces the most common and long-adhered to regulations that were set up by the local council previously. They detail specifics regarding the animals environmental, dietary, health and well-being requirements and, in doing so, ensure that all local councils follow the same regulations when distributing licences. This is a positive evolution subjecting all animal owners and breeders to the same, well-defined, rules and it is more difficult the skirt the legislation by moving businesses to areas with less strictly defined conditions.

For small businesses one of the major key factors to their success is their ability to create and maintain a loyal customer base. The larger the customer base, the greater the success of the business. Having an online presence allows a business to increase their customer base in a couple of very effective ways, firstly it increases their ability to produce a brand that invokes a sense of positivity and loyalty in customers, secondly it allows them to expand their market reach so that they can increase their chances of creating new loyal customers and expanding their customer base. From these factors is it easy to see that small businesses have a lot to gain from having an online presence and that financial success in the modern age is heavily influenced by an online presence.

Finally, the new GDPR regulations introduced in October of 2018 enthrone the initial eight principles of data protection that were set out in 1998 but provide individuals with the power to control the use of their data. This has yielded multiple benefits, allowing users to become more aware of how and why their data is used and collected, giving power to individuals to protect data that is more sensitive to them and withdraw any consent given at their discretion. The changes are a giant step in the direction of allowing individuals the ability to hold a private life as described in Article 9 of the Human Rights Convention.

3 Analysis

3.1 Introduction

This chapter is focused on defining, eliciting and prioritising the requirements of the system, it will also look at the positives and negatives of the proposed system by way of a SWOT analysis.

Firstly the main stakeholders of the project will be identified and assessed in a power/interest matrix to help to define which stakeholders' opinions and needs are the most important in the scope of the project. With this knowledge in hand, user stories will then be crafted to help to rough out what exactly is expected from the project. These rough ideas will then be refined and separated into functional and non-functional requirements, complete with descriptions of each use case.

Once the requirements have been fully defined, they will inform a use case diagram to demonstrate the relationships between the use cases and the system actors. Some of the use cases are more in-depth than others, ones that require more explanation will then be elaborated upon with use case textual descriptions. This will include the requirements flow of actions and alternate flows. Finally the requirements will be prioritised and displayed in a MoSCoW format.

3.2 Stakeholder Analysis

There are six major stakeholders for the system, three of which will be the users of the system, Dog Breeders, Dog Owners and Prospective Dog Owners. Two of the interested parties are stakeholders that may have interest in the system for their own reasons, they are; Local Councils and The Kennel Club. Finally there is also the Project Owner. The base interests and relative interest and power rating of each of the stakeholders can been seen in **Appendix A (A.1, A.2).**

The project owner has a large level of interest in the system as they desire to complete the project more so than any other stakeholder. However as the project is being completed for a client, they do not have the most power in the system. The project owner still has a large influence and therefore should be kept happy to ensure the completion of the project.

The client for the project is a dog breeder, as a result of this they are the stakeholder that holds the most power, however their interest is lower than that of the project owner as they would benefit from the final product but are capable of existing without it.

Dog owners have the least power and interest as a stakeholder of the system, they have little influence over the system as they are not main target audience. Their interest is also less than the other main stakeholders as they already own a dog and therefore do not need the system unless they are looking to buy another dog, however this would make them a part of the Prospective Dog Owner's stakeholder group.

The interest of a Prospective Dog Owner is much higher than that of a Dog Owner, this is because they are looking to own a dog and the system has features that would work to facilitate this interest. Their influence is also higher as they would also potentially act as a user of the system, but they do not have the

influence that a Dog Breeder has as they are not the client and do not have as much of a vested interest in the final product.

Councils may have an interest in the system due to the recent changes in the legislation for dog breeding within the UK, discussed in depth in section 2.2 of this report, as a result the councils may wish to have access to the data that the system stores. This may be so that they are able to easily detect when dogs are not being looked after in the conditions outlined in the legislation or so that they can easily find breeders that are producing too many litters without the correct license. However the chances that this system will become large enough for local councils to have a use for this data is small, as a result the estimated interest for this stakeholder is relatively low. However if they were to gain interest, then they could end up having a large amount of power in the system.

The Kennel Club is the stakeholder that has the least interest and the least influence in the project. They have an interest in the system as it could be used to promote Kennel Club registration for both Breeder and Prospective Owners. They influence in the system is as low as they will not be a user of the system and therefore are not to be considered as greatly as Breeders and Prospective Owners.

In conclusion with consideration of the stakeholders and their respective powers and interests in the system, the project should work to manage the Breeders, Prospective Owners and the Project Owner closely as they hold the most power and interest in the system. It should also work to monitor and inform the Local Councils, The Kennel Club and the Dog Owners as their interest in the system is less to the point that just monitoring them should be sufficient in the scope of the project. Special care should be taken to ensure that the Local Councils are happy and satisfied with the project as they hold a larger amount of power but very little interest in the project.

3.3 System Actors

From the list of stakeholders, the users of the system have been isolated and detailed descriptions of them as system actors have been created. The users of the system have been described as the primary actors of the system whereas the services that aid the system, e.g. Databases and APIs, have been identified as secondary actors.

As a result of The Kennel Club, Local Councils and the Project Owner not being users of the system, they have not been identified as system actors.

Actor Name: Dog Breeder
Type: Primary
Role Description: It is the responsibility of the breeder to update the entries in the database that relate to the dogs that they own and breed, using the provided interface. They will be able to add new dogs, edit the details of both existing dogs and themselves, produce contracts for new puppies and be able to generate the family tree for any dog that belongs to them.
Actor Goals: • Track dogs health and prospective owners • Add new dogs to the system • Track guardian home dogs
Related Requirements: Register new account, Login to existing account, Update account name, Update account email, Update password, Update tagline, Update bio, Update photos, Update Twitter link, Add feed to main page, Add puppy health entry, View puppy health, Update acceptable deviance, Notify health outside deviance, View dog details, View dog's DOB, Update dog's DOB, View dog's gender, Update dog's gender, View dog's name, Update dog's name, View dog's description, Update dog's description, View dog's collar, Update dog's collar, View breed details, Update breed details, View dog family tree, Update family tree, View dog's owner, Update dog's owner, Add new dog / puppy, Remove dogs, Add puppy to litter, Remove puppy from litter, Add puppy visits to calendar, Generate contracts, Update contract template, Locate breeders, View update feed, Update interested feeds, Message users, View guardian homes, Notify guardian home seasons

*Table 3.3-1 – Dog breeder system actor (**Appendix A (A.3)** contains remaining system actors)*

3.4 User Stories

As...	~Condition	I want to ...	So that...
Dog Breeder		Add puppy health entry	I can ensure the continuing health of puppies in my care
Dog Breeder		Be notified if puppy health is outside of defined range	I can ensure the continuing health of puppies in my care
Dog Breeder		Track puppies owners	I can ensure puppies are collected by the correct owners
Dog Breeder		View family tree	I can see how closely a wide range of dogs are related
Dog Breeder		Add a new dog / puppy	I can add new puppies, sires and bitches to the system when they become part of my business
Dog Breeder	Who has concurrent litters	Group puppies into a litter	I can differentiate puppies from each litter
Dog Breeder	Who offers guardian home contracts	View guardian home dogs details	Keep up-to-date with when guardian dogs are in season and ready to breed
Dog Breeder	Who offers guardian home contracts	Be notified of when a guardian home dog is coming into season	Keep up-to-date with when guardian dogs are in season and ready to breed
Dog Breeder		Add puppy visits to my calendar	I can organise and monitor visits to ensure that I am available at required times
Dog Breeder		Generate contracts from a template	I can easily create and distribute professional documents with ease
Dog Breeder		Update feed of interested parties	I can advertise new litters through the application

*Table 3.4-1 – Dog breeder user stories (**Appendix A (A.4)** contains remaining user stories)*

3.5 Functional and Non-Functional Requirements

The requirements for this project has been separated into functional and non-functional requirements, non-functional requirements are constraints to ensure the quality of the system whereas the functional requirements are the expected features that the system should deliver. The following requirements have been elicited from the user stories that were crafted in the previous section.

The missing requirements can be seen in **Appendix A (A.5)**, they have not been included here as there is a long list of requirements and the removed requirements are expected to act as sub-sections to UC15 – View Dog Details, and requirements that are expected of the system, e.g. Registering accounts and account details.

3.5.1 Functional

ID	Use Case	Description
UC06	Update tagline	Breeders have the ability to change the tagline that is linked to their profile
UC07	Update bio	Breeders are able to edit the bio section of their account
UC08	Update photos	Users are able to add and remove photos associated with their account
UC09	Update Twitter link	Breeders are able to change which twitter feed is shown on their page, none is also an option
UC10	Add feed to main page	Users are able to add feeds that interest them to their personal splash pages
Uc11	Add puppy health entry	Breeders are able to add a date and a weight so that the overall health of a puppy can be monitored effectively
UC12	View puppy health	Breeders are able to see the data linked to a dogs health so that they are able to make their own judgements on the health
UC13	Update acceptable deviance	Breeders are able to update the range at which a puppy is considered to be dangerously underweight compared to its littermates
UC14	Notify health outside deviance	Breeders are alerted when the weight of a puppy is lower than has been set as the acceptable range
UC15	View dog details	Users are able to see details about dogs that they have permission to see
UC28	View dog family tree	Users have the ability to view the family tree of dogs that they have permission to view
UC29	Update family tree	Breeders have the ability to update the relationships between their dogs that allow the system to build the family tree view
UC32	Add new dog / puppy	Breeders have the ability to add new dogs to the system
UC33	Remove dogs	Breeders have the ability to remove dogs from the system in the case of an incorrect entry

UC34	Add puppy to litter	Breeders have the ability to add an existing dog to a grouping of dogs, represented as a litter
UC35	Remove puppy from litter	Breeders have the ability to remove dogs from a litter in case they are added incorrectly
UC36	Add puppy visits to calendar	Breeders are able to add events to a calendar to better organise their puppy related events
UC37	Generate contracts	Breeders have the ability to generate pdf contracts using a template and information about owners and puppies
UC38	Update contract template	Breeders have the ability to specify the own contract by use of a WYSIWYG editor
UC39	Locate breeders	Users have the ability to find breeders using a set of filters including but not limited to; distance, price and breed
UC40	View update feed	Users have the ability to view a "feed" of updates from breeders and other twitter feeds that are of interest to the user
UC41	Update interested feeds	Users have the ability to update the list of Twitter feeds that are of interest to the user and therefore appear on their main splash page
UC42	Message users	Users have the ability to send messages to other users of the system
UC43	View guardian homes	Breeders have the ability to view dogs that were sold with the guardian home contract
UC44	Notify guardian home seasons	The system should alert Breeders when guardian home dogs are coming into season so that the breeder can return on their investment

Table 3.5-1 – Function requirements

3.5.2 Non-Functional

1. The interface should be responsive with only a few seconds of delay for the user.
2. The system should have an uptime of 99% or greater to ensure an uninterrupted service.
3. The system should be scalable, able to handle large volumes of simultaneous usage – a minimum of 50 concurrent users on the un-upgraded base server.
4. Passwords and users' personal data should be encrypted, passwords should be hashed with a salt.
5. Authenticating into the system should be fast, taking a maximum of a minute to complete to ensure the user has a fluid experience.
6. The system should be backed-up every major change to the system so that lost data can easily be recovered.
7. The system should be maintainable, code needs to be readable and modular
8. Updates to the system should be released incrementally to ensure a fluid release without an interruption of service.
9. The interface should be intuitive so that users have an easy experience.
10. The server should be able to hold a reasonable of photos per user, should be high enough that a user never experiences a refusal to accept more photos.
11. The system should censor information that is not accessible to the current user, e.g. the exact location of a Breeder.

3.6 Use Case Diagram

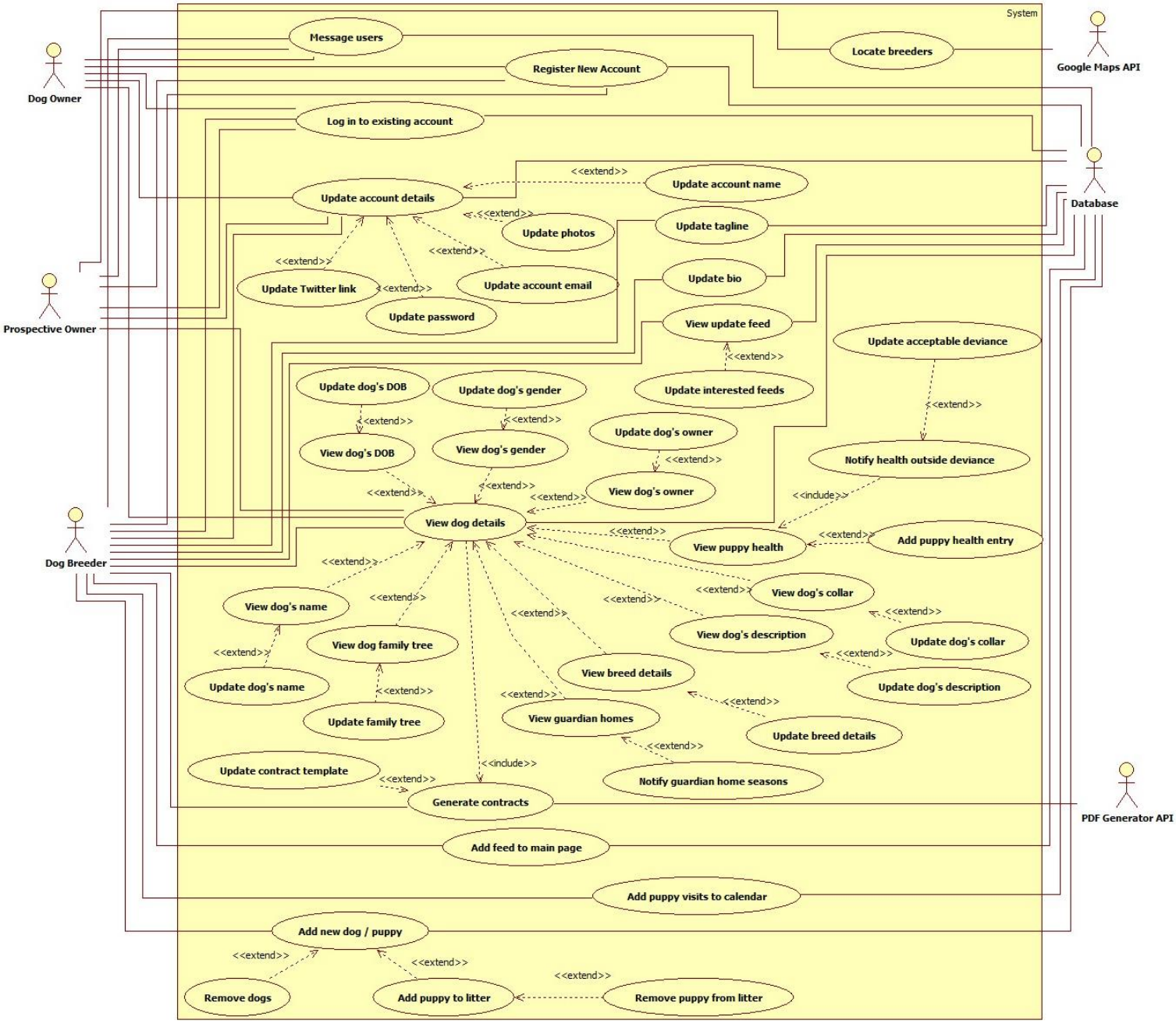


Figure 3.6-1 – Use case diagram

3.7 Use Case Textual Descriptions

Some of the identified use cases require extra detail and have potential alternative flows from the expected result. For these use cases, textual descriptions have been created to demonstrate the conditions required for the use case, the end conditions, which actor is doing the use case and the flow of events to complete the use case. As there are over forty different use cases for this project, not every single use case has been converted into a textual description. Some of the earlier use cases have been as examples and use cases that have more complex conditions and flows have also been converted.

UC02	**Register new account**	
Actor(s)	Dog Breeders, Dog Owners, Prospective Dog Owners	
Description	The user is able to use their email and password to gain access to the system and subsequent features and functionalities	
Pre-conditions	The actor has an account on the system	
Success End Condition	The actor has authenticated and logged onto the system	
Failed End Condition	The user fails to authenticate and isn't logged onto the system	
Main Flow	Step	Action
	1	User accesses webpage
	2	User clicks the log in button
	3	User enters their details
	4	User logs onto the system
Alternative Flows	Step	Action
	2a	User has forgotten their log in details
	2b	User clicks the forgotten details button
	2c	User enters their email
	2d	System generates a link for the user to reset their password and sends it to their email
	3a	The user enters the wrong details
	3b	The system redirects the user to enter their details again

*Table 3.7-1 – UC02 textual description (**Appendix A (A.6)** contains the remaining textual descriptions)*

3.8 MoSCoW Analysis

The requirements have been prioritised according to their importance for the system to function and for the client's satisfaction.

Must		Should	
UC01	Register new account	UC13	Update acceptable deviance
UC02	Login to existing account	UC14	Notify health outside deviance
UC03	Update account name	UC28	View dog family tree
UC04	Update account email	UC29	Update family tree
UC05	Update password	UC34	Add puppy to litter
UC06	Update tagline	UC35	Remove puppy from litter
UC07	Update bio	UC39	Locate breeders
UC08	Update photos	UC43	View guardian homes
Uc11	Add puppy health entry	UC44	Notify guardian home seasons
UC12	View puppy health		
UC15	View dog details		
UC16	View dog's DOB		
UC17	Update dog's DOB		
UC18	View dog's gender		
UC19	Update dog's gender		
UC20	View dog's name		
UC21	Update dog's name		
UC22	View dog's description		
UC23	Update dog's description		
UC24	View dog's collar		
UC25	Update dog's collar		
UC30	View dog's owner		
UC31	Update dog's owner		
UC32	Add new dog / puppy		
UC33	Remove dogs		

Could		Won't Have	
UC09	Update Twitter link	UC36	Add puppy visits to calendar
UC10	Add feed to main page	UC42	Message users
UC26	View breed details		
UC27	Update breed details		
UC37	Generate contracts		
UC38	Update contract template		
UC40	View update feed		
UC41	Update interested feeds		

*Table 3.8-1 – MoSCoW analysis of functional requirements (Justifications in **Appendix A (A.7)**)*

3.9 SWOT Analysis

This SWOT analysis aims to identify the Strengths, Weaknesses, Opportunities and Threats that exist in the scope of this project.

Strengths and Weaknesses are factors that are internal to the project, aiming at ideas that come from the team working on the project and the resources at the projects disposal. Strengths are beneficial internal factors whereas Weaknesses are detrimental internal factors.

Opportunities and Threats are external factors, these could be other projects, resources that are out of or inside the projects reach or a lack of familiarity with technologies that are required for the project. Opportunities are beneficial external factors whereas Threats are detrimental external factors.

Strengths	Weaknesses
• Familiarity and experience with web design in PHP • Familiarity and experience with Database Management with MySQL • Familiarity and experience with HTML and CSS • External server at disposal with good Linux knowledge to optimise usage • External server allows for server-side processes rather than entirely client based • Source control with private repository	• Lack of experience in project and time management • Large scale project with a very small team to produce • External server requires more initial setup than provided hosting • External server requires more maintenance than provided hosting • Accessing users location requires SSL connection, unfamiliar with processes involved
Opportunities	**Threats**
• Large quantity of available resources to pool, lecturers and online resources • Support from supervisor • Existing APIs to produce pdf files from HTML documents allows easy fulfilment of generating contracts use case • Constant contact with the projects client allows for consistent feedback and requirement elicitation, also allows for user testing that is guaranteed to fit the clients vision	• Time constraints could result in the project not reaching completion • Reliance on external APIs could leave system vulnerable to deprecation as the APIs update their processes • Resources on external server may not be enough to cope with the many processes that could be running at one time • Constant contact with client could result in feature creep as they request more features that there is time to properly implement

Table 3.9-1 – SWOT analysis of project

3.10 Risk Analysis

The risks for the proposed system have been given probability, impact and priority values, these values have been used to create a risk table, which can be seen in **Appendix A (A.8)**. For each risk a plan to mitigate the risk and a contingency has been drafted, this should help to prevent the project from running into these risks and to help minimise the damage if they are encountered.

3.11 Summary

This section of the report serves to show how the project's requirements were elicited and prioritised by considering the users and stakeholders of the system and their different wants and needs. By determining the power and interest of all the stakeholders, prioritising the requirements was simplified so that the stakeholders with the most interest and power were considered above those with a waning interest.

This section also shows an analysis of the project itself, determining the advantages and disadvantages that are applicable. A risk analysis has been produced to help to mitigate the more impactful risks and provide contingencies for those are more difficult to prevent.

Applying the results from this section will help to ensure that the project is delivered in a state that is acceptable as the minimum viable product to the client (Dog Breeders) and that as little time as possible is wasted on the correction of mistakes and the redesign of the system to accommodate for emerging requirements.

4 Design

4.1 Introduction

This section of the report will be separated into two main sections, focusing on the main aspects of the design. It is worth noting that due to the time difference between the design of the interface and the database, the messaging requirement and the calendar requirement were dropped due to their complexity to complete. Because of this, these functions are present in the interface design but not in the database design.

The interface design section will focus on the front end of the project, the web interface. It will tackle the production of the web pages and the early prototyping of the interface, as well as the structure of the pages themselves and the separation of the different users.

The second half will look at the design of the database that will serve to hold all the data that will ensure that the application can serve dynamic content and store information relevant to the application. It will look an identifying the main elements of the project and will then expand upon this to produce a class diagram that will show the relationships in the data. This section will also cover the decisions taken when producing the main design and why certain directions were followed over some suitable alternatives.

The section will complete with a summary of the choices made and will act to give an overall idea of where the design of the project is headed and what should be expected from the final product.

4.2 Interface Design

For the interface design, the requirements identified in the analysis will be grouped together based on their ability to be completed with the use of similar functions. This grouping will also serve to provide definition for different users view and which views will be shared among all users. This will help to identify what pages the application will require to serve the identified functions.

Once the pages have been identified, they will then be drafted into a site map that will show the links between pages and how a user would navigate the application. This will allow for the implementation of the pages to process smoothly and without reiteration.

Finally, wireframes will be produced for the pages, only low-fidelity wireframes will be produced for the application, this is because the product is more focused on providing functionality and choices such as typefaces are less important for the system and can be focused on after the completion of the MVP.

Once the wireframes have been completed, a low function prototype will be produced in the place of the high-fidelity wireframes.

4.2.1 Page Identification

The user stories show that they are three different users for the system; Breeders, Owners and Prospective Owners. When considering these three entities in the system it is important to make the distinction that an Owner could also act as a Prospective Owner should they both own their own dog and be looking to get another. For this reasoning it is clear that the Owner and Prospective Owner will

require the same view when using the application. As a result the two views for the system will be a Breeder view and an Owner view, there will also be some pages that both views will have access to, and these will be identified first.

4.2.1.1 Shared Views

There are several requirements identified that would be needed for both users, such as; logging into the system, functionalities surrounding profiles and account information, receiving notifications and viewing the family tree of their dogs.

Other pages, such as breeder's profile are viewable by both users, but under different circumstances, as a Breeder the page is viewed to update their details but as an Owner the page is viewed to see the Breeder

Table 4.2-1 shows all the pages identified to be shared across all users, it displays which use cases a facilitated by which page and justifications for each decision.

Pages	Use Cases Fulfilled	Justifications
Login	UC02	A log in page allows the user to log into the system which is the basis for UC02.
Notifications	UC14, UC44	All notification-based use cases will be handled by the notifications page.
Messages	UC42	Sending users messages is able to be handled via a messages page
Family Tree	UC28, UC29	This will show the relationships in a graphical format to fulfil the use cases.
My Dog	UC15 – UC31	Viewing the dog's details will be facilitated with the implementation of the dog's detail page. The family tree in this case will be shown as pictures of directly related family and not the full tree.
Breeder Profile	UC10	Adding a feed to the main page is a functionality that a user will be able to perform when they are viewing a breeder's profile page.
Register	UC01, UC03, UC04, UC05	Registering a new account requires the user to create details to allow for logging into the system, UC01, 03, 04 and 05 all facilitate this.
Lost Details	UC03, UC05	Updating of usernames and passwords are done of the lost details page.
Index	UC40, UC41	Viewing a feed and updating what is shown on that feed will be done on the homepage, this is because these are updates that should be of interest to the user and so should be the first thing that they see when they log in.
Account	UC03, UC04, Uc05	From a user's account they are able to update their account details.

Table 4.2-1 – Shared pages

4.2.1.2 Breeder View

The Breeder view requires some functionalities that do not apply to the Owner, the Owner will only own a few dogs and will not need to have a page dedicated to listing out all of their dogs. The Breeder, however, could have many, many puppies in the system and will need to be able to list them out and to be able to filter what the list is showing. Hence Breeders have the "Dog List" page whereas the Owners do not.

The ability to see the details about a puppy is a feature that is required for the system, but the Owner does not need to see these details. So this is another page that is specifically designed to serve the Breeders requirements. This is also the case for the "Health Graph" page.

The other three pages for the Breeder work in the same fashion, acting to serve use cases that only apply to the Breeder and not to the Owner. Contracts are something that only the Breeder needs to generate as they are the users that are selling opposed to the Owners. Calendars and Editing the Breeders profile are also features that are only required by the Breeder and not by the Owner.

Table 4.2-2 shows the use cases fulfilled by the pages crafted and the justifications for each of those pages.

Pages	Use Cases Fulfilled	Justifications
Dog List	UC32 - Uc35, UC43	From this page the breeder will be able to add a new dog / litter and edit existing litters. This is a list page that can be specified to show certain types of dogs, such as guardian homes.
Puppy Details	UC11, UC12, UC15 – UC31, UC43	Viewing the details of puppies will be done here, this is different to the dog page as it also shows the health of a dog and whether that dog is a part of a guardian home.
Health Graph	UC11 UC12, UC13	The health view page will work the same as the details page but is separate so that these details can be viewed alone. This is the only place to update the acceptable deviance.
Contracts	UC37, UC38	A page for contracts will be able to allow the user to set the template for their contracts and allow them to produce contracts from this template.
Calendar	UC36	The calendar page will allow for visits to be added to a calendar
Edit Profile	UC06, UC07, UC08, UC09, UC10	A breeder's profile page contains extra details that an owner's page does not, thusly it requires the ability to update those details.

Table 4.2-2 – Breeder only pages

4.2.1.3 Owner View

The Owner only has two views that the Breeder does not and this is the ability to search and locate Breeders. This is an Owner only view because the Breeder does not need to be able to find other Breeders, Table 4.2-3**Error! Reference source not found.** shows these pages.

Pages	Use Cases Fulfilled	Justifications
Map	UC39	The map page will show the results of trying to locate breeders and as combined with the "Search View" page will facilitate UC39.
Search View	UC39	The ability to search means that users are able to locate breeders with the help of the "Map" page.

Table 4.2-3 – Owner only pages

4.2.2 Site Flow

To be able to accurately create the pages identified in the previous section they will be modelled into an activity diagram acting as a model for the expected flow of the site. This model will attempt to show the relationships between pages and how a user would be expected to travel between them, it will also touch on pages that will only be access under certain circumstances.

Figure 4-1 shows the completed diagram, it demonstrates how the Breeder and Owner views are disconnected and unable to view each other's pages. There are several pages that both can access, but only from one direction, e.g. the Notifications page can be accessed from both indices but the indices cannot be accessed from the Notifications page.

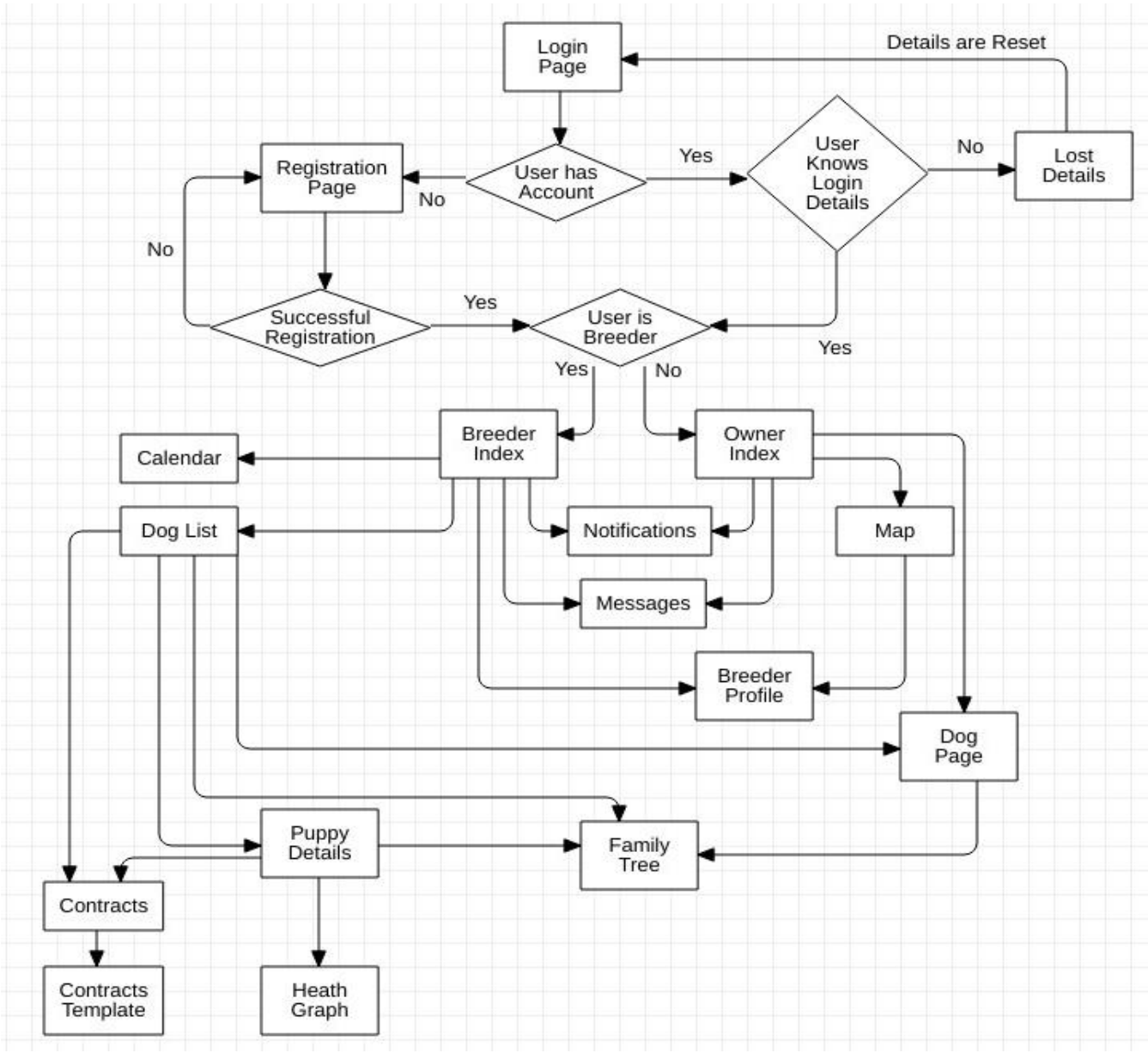


Figure 4.2-1 – Site map

Two pages identified in the previous section are notably absent from the diagram, these are the "Edit Profile" and the "Search View" pages. These are missing from the diagram because both are considered to be subsections of already existing pages, "Search View" accessible on the "Owner Index" and "Edit Profile" accessible from the "Breeder Profile" page. It is worth noting that only the Breeder will have access to "Edit Profile" despite its parent being accessible to both users.

4.2.3 Wireframes

With the relationships between pages completed, wireframes were produced and can be seen in **Appendix B (B.1)**. All pages that shared relations with the index pages were added to the navigation bar so that they could be accessed from any point on the site without first directing to the index page.

For this project only low fidelity wireframes were produced, this allowed the project to progress far faster than otherwise would've been the case. Additionally, features such as logos, typefaces and colour schemes that would be chosen with the use of high fidelity wireframes are features that can be implemented in the prototype in ways that allow for a quick swap to test out large varieties of settings without producing time-costly wireframes.

To compensate for the lack of high fidelity wireframes, the low fidelity wireframes have been produced with a higher level of detail than would otherwise be expected. They include details of icon usage and button placements.

4.2.4 Prototyping

The final step of the interface design was to produce a prototype to show roughly how the interface should be expected to look. The implementation of this prototyp will be discussed in more detail in the implentation section of this report, however it is mentioned here as was the finalisation of most of the details that were drafted in the wireframes.

The prototype acts to show the interface without any major style choices applied and acts as an interactive high fidelity wireframe, with the links between pages active and the content only being generated from hard-coded values and not drawn from any database. Figures 4-2 and 4-3 show parts of this prototype. The rest can be found in **Appendix B (B.2).**

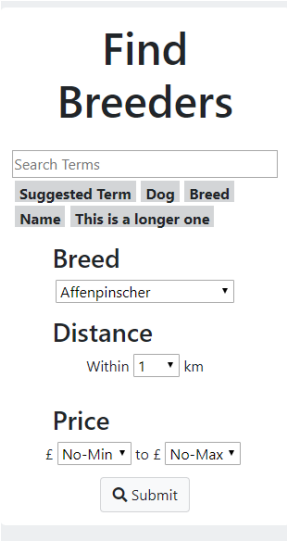


Figure 4.2-2 – Search view

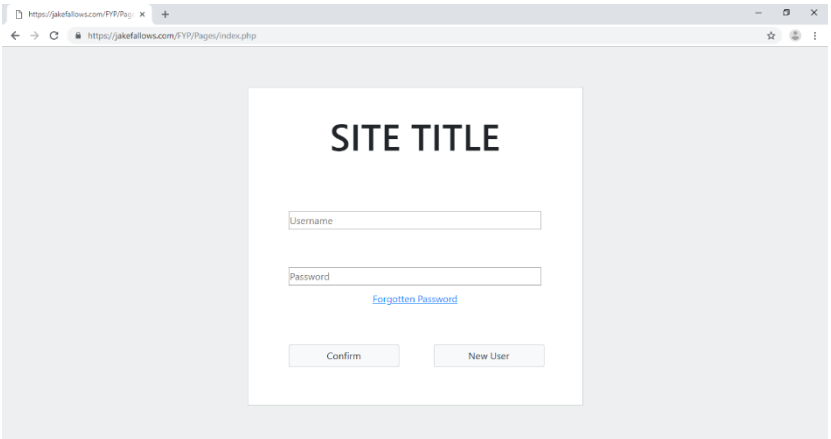


Figure 4.2-3 – Prototype login page

4.3 Database Design

The database design will start by identifying the primary components of the system. This will be done by analysing the system and user stories. The primary identification of the systems required entities and attributes will show the first steps of the design process. The entities and attributes will be drafted into a class diagram to model the relationships.

This model will then be iterated to improve the design by extracting lists into their own tables and resolving the many-to-many relationships, whilst this will increase the complexity of the diagram and resulting database it will reduce the time to complete queries and the efficiency of the database.

4.3.1 Primary Entity Identification

The main entity of the system is the ***Breeder***, this is because they are the client that has a need for the system. However, since the ***Breeder*** entity is a user of the system, it is reasonable to assume that there are likely to be other users that will require similar attributes to that of the ***Breeder***, as a result it should be identified that another entity of the system is the ***Account***, this is the class that will hold all the important information for a user to be able to log into the system. The ***Breeder*** will then be able to act as an extension of the ***Account***, storing details that are only required for the ***Breeder*** entity and not any other user of the system.

The main use of the system is to track data surrounding the ***Dogs*** that a ***Breeder*** owns, it makes sense that another required class would be the ***Dog*** class, so that the data in question can be stored for the ***Breeder***.

One of the user stories for this project is giving the ***Breeder*** the ability to group different ***Dogs*** together into a ***Litter***. For this reason, a ***Litter*** is another identified entity.

Accounts require the ability to store and display ***Photos***, it would make sense to store that data in its own separate table and so ***Photo*** is another primary entity of the database.

Finally, the system needs to have the ability to send users alerts, this functionality should be facilitated with the addition of a **Notification** table, which will store which user has received the ***Notification*** and what the content of the ***Notification*** is. The reason that the ***Notification*** is a separate entity and not an attribute of the ***Account*** entity is due to the ability of a user to receive many notifications having this start as a separate entity removes one step of normalisation when making attributes atomic.

4.3.2 Primary Attribute Identification

The ***Account*** entity acts as the main body of the system, allowing users to log in and access the application. It requires attributes to identify each users log in credentials, the users ***Email*** and ***Password***. To ensure that the system is secure the passwords will be salted before they are encrypted, to facilitate this a ***Salt*** will also be stored in the ***Account*** entity. The table will need to be uniquely identifiable, so an ***Account ID*** is required. Finally the user has the ability to view different ***Twitter*** streams on their homepage, this will also be stored here.

For the ***Breeder***, their location will need to be stored so that other users are able to fine ***Breeders*** local to them. Their profile will display with their own ***Biography***, ***Tagline***, ***Breed*** and ***Twitter*** and as a result they will need to be stored in the table. As the ***Breeder*** entity is an extension of the ***Account*** entity it will be have ***Account ID*** attribute to link the two together.

The ***Photo*** entity requires only two attributes, the location of the file stored as its ***URL*** and the ***Account ID*** of the account that uploaded the photo.

The ***Dog*** entity requires the most attributes, as storing dog data is the main focus of the system. The ***Dog ID*** will be used to uniquely identify the data and the dog will be linked to the ***Account*** entity with the ***Owner*** attribute. The details of the dog are required to be stored, this includes it's; ***Name***, ***Gender***, ***Guardian Home***, ***Collar colour***, ***Description***, the ***Breeders Notes*** and finally a record of its ***Health***.

The ***Litter*** entity needs to know the ***DOB*** of the dogs related to it and the parents of the ***Litter*** itself, these are the main attributes of the ***Litter*** entity. Finally the ***Litter*** has a ***Litter ID*** to act as the unique identifier of the ***Litter*** class.

Finally the ***Notification*** entity, as this is a simple form of sending users ***Messages***, it will need to be able to store the ***Message*** itself, the ***Account*** that it is bound for, the ***Date*** of the message and a ***Link*** to the cause of the notification if it is required.

Table 4.3-1 shows the full list of entities and attributes identified for the start of the database design.

Entity	Attribute
Account	• Account ID • Name • Email • Password • Followed Twitters
Breeder	• Location • Twitter Handle • Bio • Tagline • Breed • Account ID
Photo	• URL • Account ID
Dog	• Dog ID • Owner (Account ID) • Name • Gender • Guardian Home • Collar • Description • Notes • Health Record
Litter	• Litter ID • DOB • Sire (Dog ID) • Bitch (Dog ID)
Notification	• Notification ID • Account ID • Date • Message • Link • Type

Table 4.3-1 – Primary entities and attributes

4.3.3 Class Diagram Development

All figures in bold can be found in **Appendix B (B.2).**

Figure B.3-1 - Version 1 of Class Diagram

shows the first iteration of the class diagram, it shows the six identified entities and the addition of the *health record* entity. The addition of this entity acts to resolve the many entries into the health record attributes that would have to exist to facilitate the needs of the *dog* entity.

Figure B.3-2 - Version 2 of Class Diagram shows the second stage of the class diagram, there are many changes between the first and the second versions of the diagram. The first of the major changes is the resolution of the many-to-many relationships in the first model, the relation between Account and Notification, Figure 4.3-1, becoming *Account Notification* and holding the *date, time* and *sender* attributes as these are attributes that are only relevant to the single occurrence of an accounts notification. It is worth noting that because multiple user could receive the same notification, the *message* attribute of the notification has remained in the *Notification* entity.

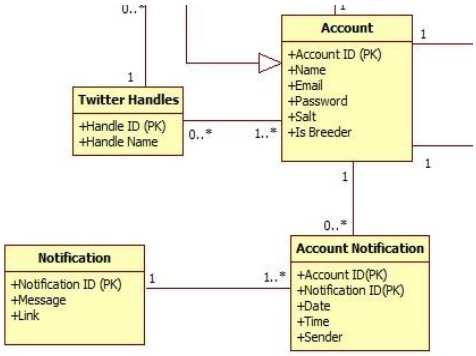


Figure 4.3-1

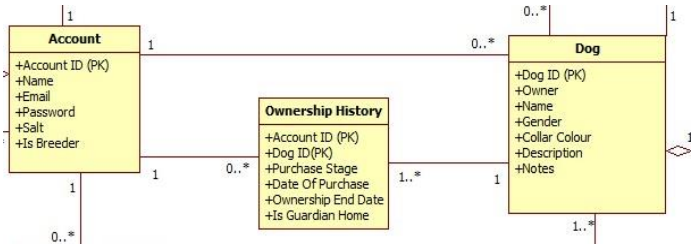


Figure 4.3-2

Another resolved many-to-many relationship that was resolved was the relationship between *Dog* and *Account* that represented the ownership of a dog. This was changed to include the *Ownership History*, Figure 4.3-2, entity and acts as a record of the ownerships of a dog, with the entry with a null *end date* acting as the current owner.

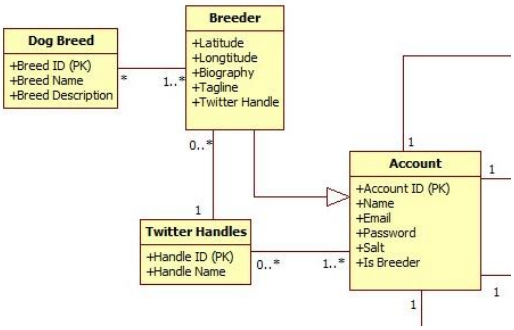


Figure 4.3-3

In addition to the resolved relationships, several new entities were added. The *Dog Breed* and *Twitter Handles*, Figure 4.3-3, were added as they are attributes that could be considered to have many recurring entries of the same data, these new entities mean that the data entered in relation to these qualities is limited and far less likely to be duplicated. With the *Twitter Handles* entity, it has a relationship with both *Account* and *Breeder* as a breeder has their own handle and an account can follow many different handles.

Finally, the last change in the second version is the addition of the relationship between *Dog* and *Photo*. This relationship acts

to serve as an easy link for the system to query for photos of dogs in the system, this is a many-to-many link as there could be many dogs in one photo but also many photos of the same dog.

Figure B.3-3 - Version 3 of Class Diagram shows the third version of the class diagram, the main changes in this diagram is the resolution of the newly created many-to-many relationships. There are three main places in which new entities have been added to resolve these connections, *Followed Handles, Breeder Dog Breeds* and *Dog Photos*, Figure 4.3-4. Another noteworthy change in this version of the diagram is the addition of the *FK* label to denote the foreign keys.

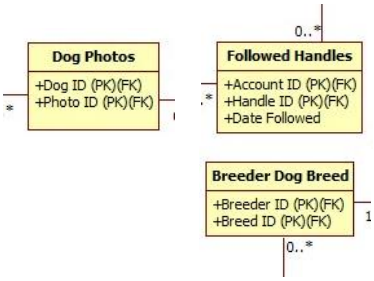


Figure 4.3-4

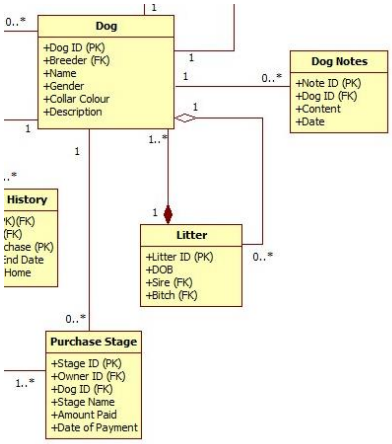


Figure B.3-4 - Version 4 of Class Diagram shows the penultimate class diagram, the main changes in this version of the diagram revolve around the *Dog* entity. The *Notes* attribute has been extracted into its own class. This is due to the ability of the *Breeder* to add and remove notes about a dog and having this data stored in a separate table instead of a block of text in the dog table ensures this functionality with ease.

Another addition is the entity of *Purchase Stage*, this is an entity designed to allow the system to show the stage of purchase for a dog. This is currently not a major requirement of the system and this entity has been included to make implementation of this feature easier in the future, Figure 4.3-5.

Figure 4.3-5

The final change in the fourth version is the addition of the *Owner Details* entity, this was required because, up until this point, for a dog to have an owner, that owner would have to have an account on the system. This table enables the breeder to enter details about an owner but not force their clients to create an account on the system. It also serves as a way to store the client's information.

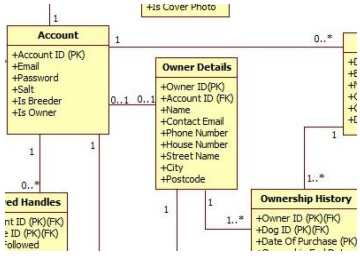


Figure 4.3-6

Finally, Figure 4.3-7 shows the final version of the class diagram, the main changes in this diagram is the *Purchase Stage* enitity merging with the *Ownership History* table, Figure 4.3-6, this is due to the two enities sharing primary keys and the relationships with the nearby tables.

The other main change is the change in the relationship between *Dog* and *Account*, this is the relationship that represented the dog and its breeder. However, this didn't make sense as there is also a *Breeder* entity, the change swaps that relationship to be between the *Dog* and the *Breeder.* As a result of this, the diagram had to be rearranged so that there were not overlapping lines, this makes the diagram look vastly different when in truth these were the only two changes since the last version of the diagram.

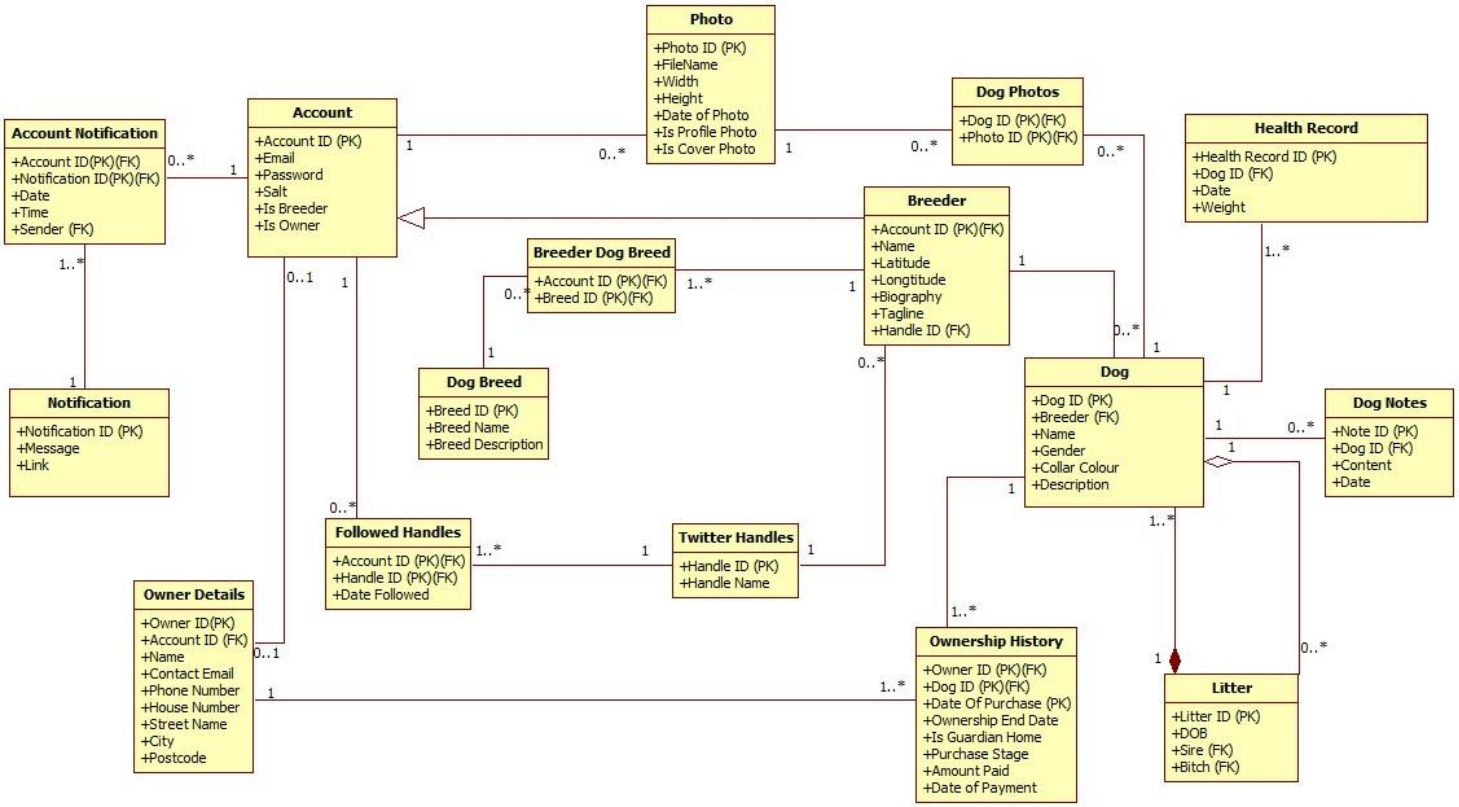


Figure 4.3-7 - Final Version of Class Diagram

4.3.4 Constraints and Assumptions

4.3.4.1 Dogs and Litters

- An entry for both parents will exist in the *Dog* table before a *Litter* is entered into the system.
- A breeder will create an entry for a Sire dog if they do not personally own the dog.
- A litter is only born on one day, occurrences over midnight will be recorded as the later date.

4.3.4.2 Health Records

- Weight is the only measure of a healthy puppy.
- A user doesn't need to add notes to the health records.
- A breeder takes regular measurements of a puppy's weight.

4.3.4.3 Photos

- Photos that are not of dogs will either be cover photos or profile photos.
- Photos will be stored in a location derived from a combination of user/dog details and so only the filename is required for the photo.
- A user will want to connect their photos to the dogs that they are taking pictures of.
- A user will only be uploading photos of their dogs.

4.3.4.4 Twitter

- Breeders have a Twitter account.

4.3.4.5 Breed

- Breeders enter accurate data about dog breeds and do not sugar-coat their descriptions.

4.3.4.6 Account

- Users do not forget the email used in the system.
- Users do not forget the password used in the system.
- Users have access the email used in the system.
- User passwords have to pass a strength test before being stored in the database.

4.3.4.7 Owner Details

- Details stored in the database are done with the owner's consent.
- Details stored in the database are encrypted with a secret key that can only be accessed by the server.

4.3.5 Data Dictionary

The data dictionary shown in this section serves to show the attributes of each of the entities and their data types, lengths and constraints. This table will be used to generate the create table statements that will be used to initialise the database.

The data dictionary can be seen in **Appendix B (B.4).**

4.3.6 Discussion on Decisions

4.3.6.1 Litters and Puppies Relationship

One of the major design decisions for the database was how to model the relationship between *Dog* and the *Litter*. The *Dog* entity includes both the parents of the litter and the puppies of the litter. There is an argument to be made for the removal of the *Litter* entity and the implementation of a self-referencing relationship for the *Dog* entity to model this.

The reason that this was not chosen as the implementation for the relationship is due to the frequency of different queries. The self-referencing implementation would benefit a query in the nature of finding all dogs, regardless of their litter and would limit the ability to find all dogs of a specified litter without a long complex query.

The current implementation benefits the opposing idea, making queries for litters much simpler but complicating the ability to find the litters parents. Because of the nature of the application is likely to be querying for litters groupings more frequently than single queries for parents, this is the reason for using the implementation proposed in the class diagram.

4.3.6.2 Dog Photos

The relationship between *Dog* and *Photo* was a design decision to allow for the ability to easily query for photos that contain certain dogs. As a design decision this acts purely for the ease of queries for dog photos, the implementation of this functionality relies on the user actually inputting data on is contained within the photos that they upload.

4.3.6.3 Purchase Stage and Ownership History

The choice to combine *Purchase Stage* and *Ownership History* does come with some drawbacks, with the two acting as separated entities then it is possible to keep a historical record of transactions. This ability is lost when the two tables are combined as the production of this record would flood the *Ownership History* with are large amount of duplicated data.

The choice to not record the historical transaction data is because the inclusion of transaction data was requested for the purpose of viewing the current status of ongoing transactions and not as a means to record over a period of time.

As the ability to store historical data was not required in the scope of the project, the decision to merge the tables yields more benefits than having them separated. With the tables merged, it will be far easier to query the current purchase stage of ownerships as the table holds both set of data.

4.4 Summary

In this section, the projects interface and supporting database were designed. The interface was created by identifying the different pages that would be required for the application and then mapping those pages out into an easily traversable layout for the site flow. The wireframes of those pages were produced to outline a rough idea of what would be expected for each of the pages. Those wireframes were them used to help model the prototype of the interface, showing an interactive side to the design and allowing the design to flourish in a fashion that meant that the style could be applied later after the implementation stage was completed.

The database design was then completed, eliciting of the main elements of the system and extending that identification to the attributes of each of the elements in turn. This data was then modelled using a class diagram, to show both the entities and their attributes and the relationships between the different entities themselves. The constraints and assumptions undertaken in the production of the diagram were then discussed and evaluated. Finally the data was entered into a data dictionary so that the create statements for the database could be easily constructed.

The results of this section will heavily impact the implementation of the project and it is highly important that this section is precise and accurate in its conclusions as mistakes in this section will have a compounded cost to correct in the implementation stage of the project.

5 Implementation

5.1 Introduction

The implementation stage of the project will focus on taking the ideas crafted in the design and constructing a working product from them. The process will be separated into three major sections; the interface, the database and the business logic of the application.

During the design of the interface, a prototype was produced. For the implementation of the interface, this prototype will be taken and updated so that it is responsive to device width and appropriately styled with logos and typeface choice.

The database implementation will involve the conversion of the data dictionary into a series of table creation queries and discussions regarding the complications faced during this process. It will also include the generation and insertion of test data to ensure that the database is able to hold the data in the expected format.

Finally the last section will tackle the business logic of the application, the technologies and the methodology used will be discussed and illustrated.

5.2 Production Pipeline

The code for this project was maintained and managed using version control software named git. This software was used to enable the ability to preserve versions of the working code and separate the development of new features from the releases.

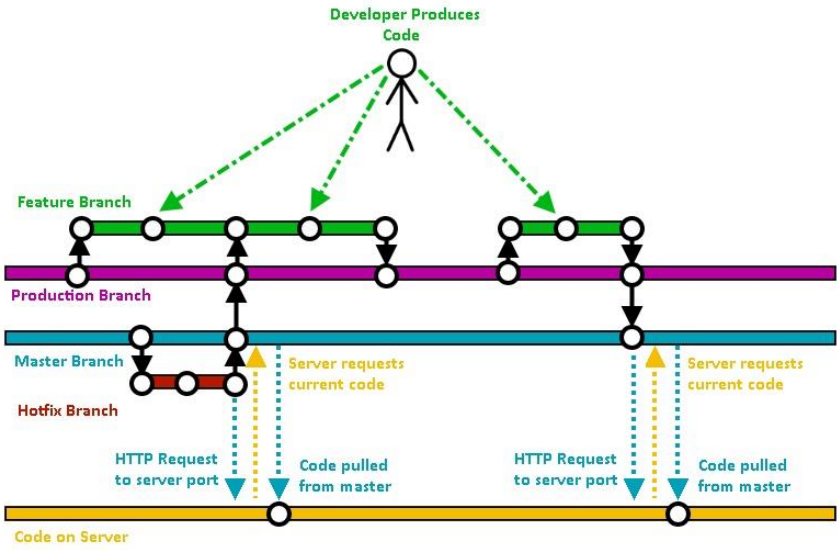


Figure 5.2-1 – Diagram of code flow in production pipeline

Figure 5.2-1 shows the particular pipeline that was used in the implementation of the interface and business logic of the application. As can be seen in the diagram, the developer is only able to work on the code using the feature branches, which in turn only push their code into the production branch once the feature is complete. The master branch is used to represent the state of code on the public server. This setup allows for the application to develop new features without affecting the existing product.

This pipeline is fulfilled on the git hosting platform of GitLab, this is a service that holds remote code repositories, allowing the developer to work on the code from any machine. GitLab further supports the production pipeline by providing the ability to send http requests upon certain events. For this pipeline

GitLab was set up so that every time a commit was merged into the master branch it would send a request to the public facing server on a defined port.

Meanwhile the server is running a program called "git-auto-pull" (Belghiti, 2017), that listens for requests on a defined port. When the request is received from GitLab, the program prompts the repository on the main server to pull the code from the master branch, updating the public facing product to the latest code, allowing instant and automatic deployment of new code to the platform anytime that production code is merged into the master.

5.3 Technologies and Libraries

This section will address the technologies that were used in the implementation of the interface and the database and will justify the reasons for those choices.

5.3.1 Interface

5.3.1.1 PHP

PHP: Hypertext Preprocessor (PHP) is an open source scripting language that is heavily associated with web environments. It has the ability to be embedded directly into HTML and acts to transform plain and static HTML pages into dynamic content, tailored to each user.

PHP has a range of features that makes it suitable to this project, PHP Data Objects (PDO) allow for a consistent, lightweight and adaptable method for database access. PDO acts as an abstracted data access method that will act in the same manner no matter which database it is connected to. Furthermore, PHP introduced the ability to create objects and follow the Object Orientated Programming (OOP) paradigm.

Finally, because PHP runs on the server and not on the client's browser it isn't dependant on the client having the correct version of the language or using their processing power, it is protected from being malicious users scanning the code for vulnerabilities and often has lower loading times than alternatives.

5.3.1.2 JQuery

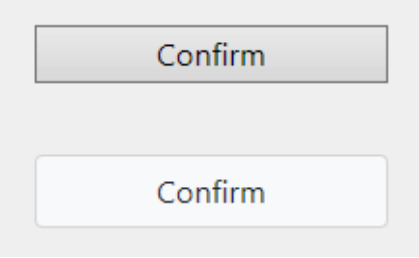
JQuery is a library for JavaScript that aims simply and enhance the usual functionalities that JavaScript provides. It removes a large amount of "Boilerplate Code", this is code that is often repeated without little change. Figure 5.3-1 shows one way in which JQuery does this, the first line is traditional JavaScript and the second line is JQuery. Both lines act the same, but it is clear that the JQuery method is much shorter.

```
document.getElementById("example-id");
$('#example-id');
```

Figure 5.3-1 – How JQuery reduces boilerplate

5.3.1.3 Bootstrap

Bootstrap is a framework for CSS and includes a library of JavaScript, this framework works to make creating web development easier and faster. There is a range of styles that are used in Bootstrap that look good enough to act as the final version of the product. Figure 5.3-2 shows a Bootstrap button compared with a normal HTML button.



In addition to including a range of styles, Bootstrap also acts to allow fast development of responsive web applications. By measuring the screen ratio of the viewing device, Bootstrap is then able to set the sizes of different classes to values that make sense for each device's screen ratio.

Figure 5.3-2 – Normal button (Top), Bootstrap button (Bottom)

Finally Bootstrap includes some JavaScript that facilitates the need for a more complex responsive components, such as navigation bars that may require the ability to shift their menu options to dropdown menus on smaller devices.

5.3.1.4 Google Maps

Google Maps has an API that allows key holders to access functions such as converting postcodes to latitudes and longitudes, using those coordinates to place markers on an interactive map and embedding maps into webpages. This API will be used for the project as it is the only API that could be found that would provide the same level of functionality.

5.3.1.5 Twitter API

One of the requirements of the project is to include the functionality for users to create posts that will appear on other users feeds. This concept comes with a large amount of overhead for such a simple function. For this reason, the Twitter API will be used, this will allow users to post content and follow people of interest without having to implement all the functionality that would usually come with an updates system.

Twitter's API allows for the embedding of a stream of tweets, by limiting this stream to a length of 1, we can use the API to only collect the latest tweet from a user.

5.3.2 Database

5.3.2.1 MySQL

MySQL is a Relational Database Management System (RDBMS) that uses the Structured Query Language (SQL). In the analysis section of this report, it was concluded that the database structure would be a relational model because of the ability to link data together through their relationships. Because the data model is a relational model it makes sense to use a RDBMS for the database technology.

MySQL was chosen over alternatives such as PostgreSQL and Oracle due to the simplicity of the setup and accessibility of the software. Due the lack of overly complex queries, MySQL is a common and well supported choice. It could be argued that PostgreSQL is potentially a stronger contender for the RDBMS technology to be used in this project. However, should the need to exchange the database arise, both MySQL and PostgreSQL use the same syntax for their SQL commands and the swap should be a simple exercise

5.3.2.2 phpMyAdmin

phpMyAdmin is a web GUI that facilitates the management of a MySQL. It is a free and open source PHP application that enables the user to easily be able to create and manage: user permissions, databases, tables, and data. It also allows for the exporting and importing of entire databases, tables and individual rows.

It is used in this project to trivialise the setup of new users and permissions in the database, as well as increasing the ease of executing raw SQL commands on the server, such as the table creation scripts or data population.

5.3.3 Libraries Used

5.3.3.1 Kingraph

A JavaScript application that executes server-side, this library is the basis for the "family tree" functionality across the application. It takes the contents of a YAML configuration file and produces an SVG diagram of the family described in the file. For this project the major hurdle in using this library was the conversion of the relationships stored in the database into a YAML file that could be used by the application.

Figure 5.3-3 shows the final result of the program when rendered on the webpage. Kingraph was produced by (Cruz, 2017).

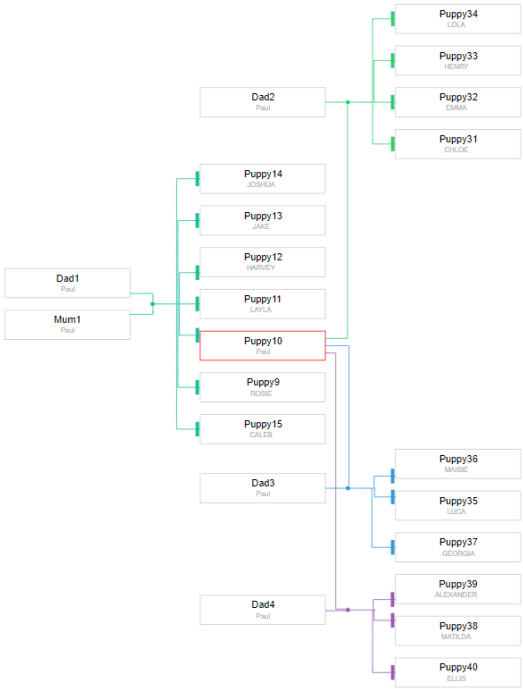


Figure 5.3-3 – Resulting SVG from kingraph

5.3.3.2 Jquery-Svg-Pan-Zoom

This library is a plugin for JQuery that enables panning and zooming on SVG images. (Bernardes, 2015). Because the output of kingraph is an SVG image, this library was used to enhance the functionality of the image so that the larger models are able to be viewed by zooming in.

5.3.3.3 Chart.js

Chart.js is a framework that utilises JavaScript and the HTML5 canvas element to make producing charts quick and simple. This library was used to help speed up production as well as allowing for a range of customisation for aesthetic results. Chart.js is a production of (chartjs, 2019).

5.3.3.4 NameThatColor

A desired functionality of some of the charts that were used throughout the application is that the line for a puppy reflects the colour of its collar. As the values for these collars is stored in the database as only a string, the library by (Zec, 2014) was adapted so that providing a colour name would result in an RGB value for that colour.

5.3.3.5 Git-auto-pull

As mentioned in section 5.2, this is a library that facilitates the deployment of the application to the hosting server. Produced by (Belghiti, 2017) this application, is written in JavaScript and utilises asynchronous functions to update git repositories when it receives a request on the specified port.

5.4 Database Implementation

5.4.1 Creation

Using the data dictionary that was constructed in section 4.3.5 create table statements were generated. To ensure that the foreign key constraints didn't cause the table creation to fail, the columns were ordered so that tables without foreign keys were created first and the ones with the most relationships were created last. **Appendix C (C.1)** shows all the statements that were used to create the tables, in the order that they were created. Figure 5.4-1 shows the tables that were created as a result of the creation statements.

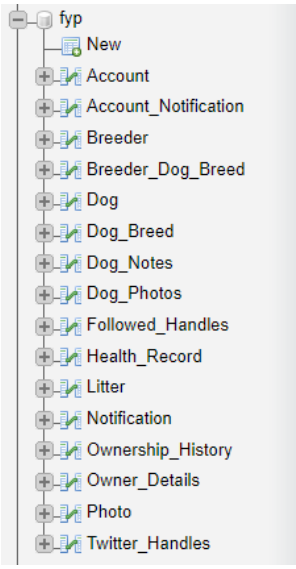


Figure 5.4-1 – Tables created in the database

Due to several issues that were encountered during the creation process, a series of drop commands were constructed so that the entire database could be reset quickly, these commands can be seen in **Appendix C (C.2).**

Some of the issues encountered during the creation process included:

- Date Types not needing a length to be specified
- Check constraints not existing in MySQL
- Cyclical Foreign Keys of the litter and dog tables

These issues were dealt with by removing the specified length from date, removing the check constraints and ensuring that the validation of these constraints occurred on the server-side application.

The cyclical dependency was tackled by removing the Foreign Key constraint from the Dog table and the

```
ALTER TABLE Dog ADD CONSTRAINT litter_id FOREIGN KEY (litter_id) REFERENCES Litter(litter_id);
```

Figure 5.4-2 – Alter statement used to add the Foreign Key to the Dog table

alter statement in Figure 5.4-2 was used to add the key after the creation process was completed.

After the creation of the database, the phpMyAdmin visualisation tool was used to generate the class diagram that is shown in Figure 5.4-3. Comparing this diagram to the one produced in section 4.3.3, it can be seen that both diagrams are very similar, showing us that our design translated into an actual database correctly.

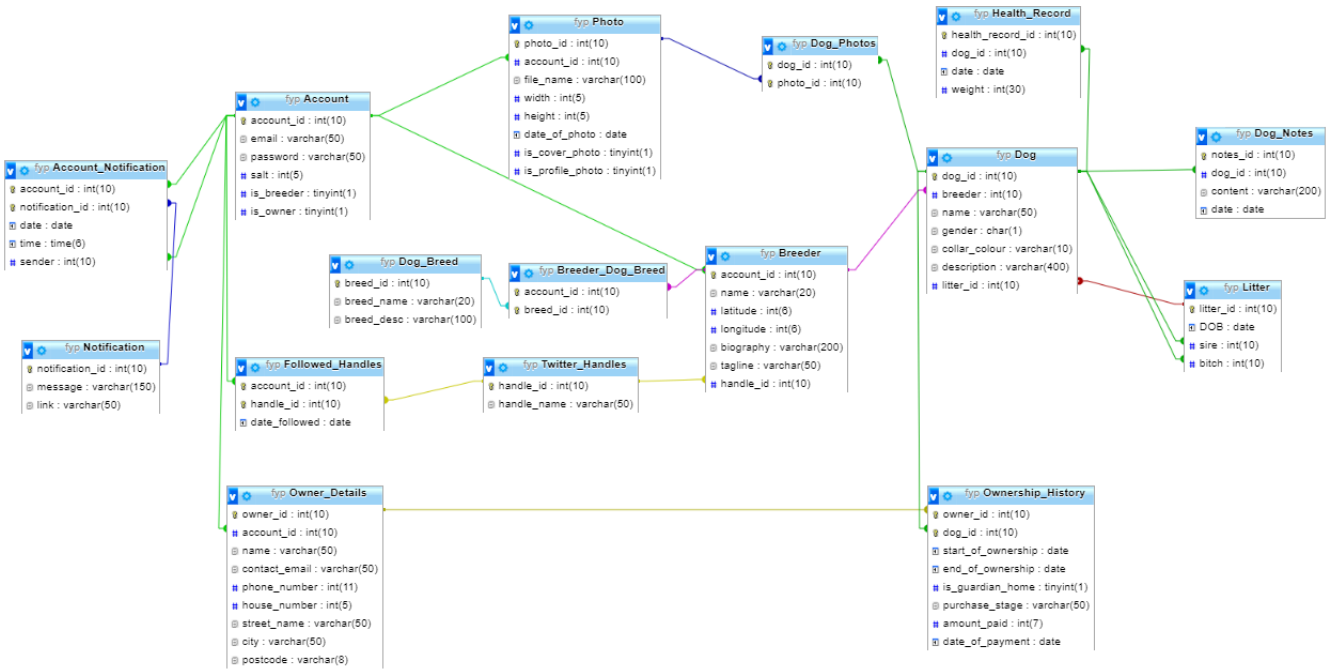


Figure 5.4-3 – Class Diagram produced from the visualisation tool in phpMyAdmin

Twitter Handles	
handle_id	handle_name
1	katyperry
2	justinbieber
3	BarackObama
4	rihanna
5	taylorswift13
6	ladygaga

Figure 5.4-4 – Example raw data sample

5.4.2 Data Population

To ensure that the business logic of the application is working as expected and that the interface is able to show the models that result from that business logic, there needed to be a series of data in the tables. This data is also required in the system to be able to provide an accurate demo of the application.

The process to create this testing / demonstration data was to create tables in Excel that contained the raw data for the database. Figure 5.4-4 is an example of the raw data that was used, however it should be noted that this table actually contained 100. The full raw data will not be included in this report as there are over 1000 records in some tables. Instead a truncated version of this data can be seen in **Appendix C (C.3).**

This raw data was then processed in a similar manner to the data dictionary to concatenate statements to mass insert the raw data. The full commands will not be included in the report, **Appendix C (C.4)** shows a truncated version of these statements.

Some of the data could not be randomised strings or passages borrowed from Wikipedia, examples of these rows are twitters handle names and longitudes and latitudes. The twitter handles were pulled from a list of the Top 100 followed handles on twitter, whereas the coordinates were generated from a list of random postcodes, using the Google Maps API.

Figure 5.4-5 shows the code that was used to perform this conversion. The results are displayed in a table so that they would paste correctly into excel with minimal alterations.

```
echo "<table>";
foreach($postcodes as $postcode){
    echo "<tr>";
    echo "<td>" . $postcode ."</td>";
    $latLong = getLatLong($postcode);
    echo "<td>" . $latLong['long'] ."</td>";
    echo "<td>" . $latLong['lat'] ."</td>";
    echo "</tr>";
}
echo "</table>";

public function getLatLong($code){
    $mapsApiKey =                                                    ;
    $query = "https://maps.googleapis.com/maps/api/geocode/json?address=".urlencode($code)."&key=".$mapsApiKey;
    $data = file_get_contents($query);
    // if data returned
    if($data){
        // convert into readable format
        $data = json_decode($data, true);

        $long = round($data["results"][0]["geometry"]['location']['lng'], 4);
        $lat = round($data["results"][0]["geometry"]['location']['lat'],4);
        return array('lat'=>$lat,'long'=>$long);
    }else{
        return false;
    }
}
```

Figure 5.4-5 – Code used to convert list of postcodes into coordinates

5.5 Server-side Application

The function of the server-side application is to model the data into a format that can be used by the interface, update existing data and insert inputs into the database.

5.5.1 Application Structure and Development

Separation of Concerns was the main principle adhered to during the production of the server-side application. The segmentation of the application was focused around the different tables of the database and their interactions with the interface. This lead to a pattern of one class per table that would access, process and pass on the data from the database.

As the application grew and the concerns became more clearly defined, the segmentation of the application became more refined and the need for greater segmentation emerged. At this point of production that the application began to mirror the Table Data Gateway design pattern, henceforth referred to as gateway. This lead to a refactoring of the application to better implement the pattern and separate business logic out of the database access classes.

The final structure of the server-side application concluded with a series of database access classes, named gateways, which inherited from one major gateway class. For each table in the database, one gateway was constructed to communicate with the related table. To process the data from the gateway classes, a series of manager classes were constructed. The viewable pages of the application access the manager to read data from the database whereas the utility pages would access the managers for the functionality to create, update or delete data.

5.5.1.1 Separation of Concerns

Separation of concerns is a design principle of software development to allow for the segmentation of an application into different sections of functionality. The benefit that this principle brings is a higher level of freedom and manageability to the code. In addition to this it also enables the application to replace segments of functionality if the requirements change. An example of this would be a change to the database technology, without a separation of concerns this change would require changes throughout the applications business logic and potentially the interface itself. When the separation of concerns is adhered to, a change in the database would only require a change in the segment that accesses the database.

5.5.1.2 Table Data Gateway

(Fowler, 2003) states that the Table Data Gateway is "An object that acts as a Gateway to a database table. One instance handles all the rows in the table." He explains that the gateway often has a simple interface and will hold all of the SQL statements for an application so that they do not interfere with the business logic. The Gateway is further defined to be "an object that encapsulates access to an external system or resource" (Fowler, 2003). This design pattern allows the developer to segment their database interaction from their logic whilst providing a layer of encapsulation.

Figure 5.5-1 shows an example of a gateway used in the final project and shows how the Table Data Gateway pattern has been applied throughout the application.

```php
class AccountGateway extends TableGateway
{
    public function getAccountDetailsWithoutPassword($fromWhichColumn, $identifier){
        return $this->selectEntry("Account", "account_id, username, email, is_breeder, is_owner",  $
    }

    public function getAllAccountDetails($fromWhichColumn, $identifier){
        return $this->selectEntry("Account", "*", $fromWhichColumn, $identifier,false);
    }

    public function createAccount($username, $email, $password, $breeder){
        //$sql = "INSERT INTO AccountGateway (account_id, username, email, password, is_breeder, is_
        return $this->insertEntry("Account", array('username' => $username, 'email' => $email, 'pass
    }

    public function updateAccount($account_id, $columns_values){
        return $this->updateEntry("Account", $columns_values, array("account_id" => array('identifie
    }

}
```

Figure 5.5-1 – Account Gateway code performing SQL statements

5.5.1.3 Managers

The Manager classes are a series of classes that were produced to better facilitate the need for segmentation within the project. Their primary function is to access the gateway classes to interact with the database. However, the also act as a method of encapsulation for the gateway classes, limiting the application to only be able to access the database through the managers.

```php
public function getMostPopularHandles($account_id, $number_to_return){
    // this function aims to deliver the top 10 handles using the count of followed handles.
    $handle_names= array();
    $current_handles = $this->getAllFollowedHandlesFromID($account_id);
    // Get the handles of the most popular handles
    $mostPop = $this->followedHandleGateway->getMostPopularHandleIDs($number_to_return);
    foreach($mostPop as $handle_id){
        $add_to_top = true;
        //if the user is already following the handle then dont recommend it
        foreach ($current_handles as $current){
            if($current['handle_id'] == $handle_id['handle_id']){
                $add_to_top = false;
                break 1;
            }
        }
        if($add_to_top){
            $handle_names[$handle_id['handle_id']] = $this->getTwitterHandle($handle_id['handle_id'])['handl
        }
    }
    return $handle_names;
}
```

Figure 5.5-2 – Example manager function performing business logic on gateway data

The managers are the main driving force behind the business logic for the application, accessing and writing data after performing their own validation and functionalities. Unlike the gateways, however, there is not a one-to-one pairing of managers to gateways, a

manager can access many different gateways and this is often the case. The six main areas of the project were used for the manager classes, with access to join tables often occurring across multiple managers. Figure 5.5-2 shows an example function performed by the **TwitterManager** class to filter out handles that are already followed by the current user.

5.5.1.4 Utility Pages

Throughout the interface of the application, there are many opportunities for users to interact with the system and to insert their own data. Whilst it is the main function of the managers to interact with this data, it is the concern of the utility pages to access this data from the POST and GET variables and to pass it to the managers for processing. The utility pages are also responsible for redirecting users after this interaction is complete by processing the results returned from the manager classes.

```php
<?php
    session_start();
    include "../../Gateways/TableGateway.php";
    include "../../Gateways/AccountGateway.php";
    include "../../Managers/AccountManager.php";
    $register = new AccountManager();
    if($_POST['password'] != $_POST['password_verify']){
        $_SESSION["login_message"] = "Passwords do not match";
        header( string: "Location: /FYP/Pages/index.php");
    }
    $result = $register->registerAccount($_POST['username'], $_POST['email'], $_POST['password'], $_POST['breeder']);
    print_r($result);
    if($result["status"]){
        // AccountGateway was created successfully, we should redirect them to be logged in
        header( string: "Location: /FYP/Pages/index.php");
    }
    else {
        $_SESSION["login_message"] = $result["msg"];
        header( string: "Location: /FYP/Pages/index.php");
    }
```

Figure 5.5-3 – Example utility page that handles users input when registering

Figure 5.5-3 shows the register utility page. As can be seen, this page instantiates an instance of the **AccountManager** and proceeds to pass it the data it has received from the post request. It also shows how the result retrieved from the manager is used to set a session variable on a failed registration and how the file deals with redirecting the user.

5.5.1.5 Utility Statements

The Create, Read, Update and Delete (CRUD) functions are four main interactions with a database. For this project, to assist with the execution of these functions and to improve the applications portability, several utility statements were constructed. These statements use the raw data that is required and constructs the SQL statement that is then executed by an SQL executor function. Figures 5.5-4 through 5.5-7 show these statements.

```php
protected function selectEntry($table, $selectColumns, $whereColumn, $identifier, $fetchAll = true){
    $sql = "SELECT $selectColumns FROM $table WHERE $whereColumn = :identifier";
    if($fetchAll){
        return $this->execPrepState($sql, array(':identifier' => $identifier))->fetchAll();
    }
    else {
        return $this->execPrepState($sql, array(':identifier' => $identifier))->fetch();
    }
}
```

Figure 5.5-4 – Utility function for reading data

Using a table name and an array of columns and values to enact the function, they convert these raw values into prepared statements, discussed in section 5.5.2.4. The placeholder values for the prepared statements were

```php
protected function insertEntry($table, $columns_values){
    $array = array();
    $sql = "INSERT INTO $table (";
    foreach($columns_values as $column => $value){
        $sql .= "$column, ";
        $array[':'.$column] = $value;
        $columns_values[$column] = ':'.$column;
    }
    $sql = substr($sql,0,-2);
    $sql .= ") VALUES (";
    foreach($columns_values as $value){
        $sql .= "$value, ";
    }
    $sql = substr($sql,0,-2);
    $sql .= ")";
    return $this->execPrepState($sql, $array);
}
```

Figure 5.5-5 – Utility function for inserting data

```php
protected function deleteEntry($table, $identifiers){
    $array = array();
    $sql = "DELETE FROM $table WHERE ";
    foreach($identifiers as $key => $value){
        $sql .= "$key = :$key AND ";
        $array[':'.$key] = $value;
    }
    $sql = substr($sql, 0, -4);
    return $this->execPrepState($sql, $array);
}
```

created by adding a ":" to the front of the column name that was to be affected by the statement.

Figure 5.5-6 – Utility function for deleting data

```php
protected function updateEntry($table, $columns_values, $where_conditions){
      array('where_column' => array('identifier'=> $idendifer, 'join' => null/AND/
    if(count($columns_values) == 0){
        return false;
    }
    // Example UPDATE statement = UPDATE $table SET $column = $value WHERE $whereC
    $array = array();
    $sql = "UPDATE $table SET ";
    foreach($columns_values as $column => $value){
        $sql .= "$column = :$column, ";
        $array[':'.$column] = $value;
    }
    $sql = substr($sql,0,-2);
    $sql .= " WHERE ";// $where_column = :identifier";
    foreach($where_conditions as $where_column => $identifier_join){
        $sql .= $where_column ." = :where_" . $where_column;
        if(isset($identifier_join['join'])){
            $sql .= " " . $identifier_join['join'] . " ";
        }
        $array[':where_'.$where_column] = $identifier_join['identifier'];
    }
    return $this->execPrepState($sql, $array);

}
```

Figure 5.5-7 – Utility function for updating data

5.5.2 Security

5.5.2.1 Passwords

Passwords are something that are required for a vast majority of applications on the internet and are potentially the most sensitive data that a user can input through a web form. Therefore, it is imperative that this information is protected.

When a user first registers on the application they provide their password in the masked input box, this data is then sent over POST and handled by the utility page showed in Figure 5.5.1.4-1 which in turn passes the information the **AccountManager** and its **registerAccount** function, to perform a salt and hash on the password before storing it in the database.

A hash is a mathematical function is irreversible and the same input will always produce the same result. As it is not reversible, if the database is breached, the password cannot be recovered from the hash.

A salt is used to add uniqueness to the password, whilst a hash function is impossible to reverse, it is possible to catalogue the hash results of common passwords. A quick search can often reveal a common password. The salt is a random string that is both added to the password before the hash, and appended to the hash itself so that the same salt can be used when comparing if a supplied password is correct when the user attempts to log in.

By comparing the value of hashed passwords to see if a supplied password is correct, the user never has to have their password exposed as plaintext to any part of the system outside of the initial input.

5.5.2.2 Location Data

The application uses a range of location data that users could consider to be personal and private. For owner location data that is input by the breeder, it is assumed that the breeder has the permission of the owner to use their location data, as they will be the only users that will be able to see it. But for breeders it is important to protect their exact locations as they can be accessed by any user. To ensure this data is protected, only the breeder's latitude and longitude is stored in the database, but is limited to two decimal places. Reducing the accuracy of the information to pointing up to 1km away from the actual location of the breeder, increases the security of the application by reducing its accuracy.

5.5.2.3 URL Variables

Throughout the application there are references to other pages that use variables in the URL to pass data from one page to another. This represents a security risk for obvious reasons, as a user could potential "URL hack" to gain access to pages that they wouldn't otherwise be able to see. To properly handle this potential vulnerability any URL based variables, before they are accessed, are passed through the function shown in Figure 5.5-8. This function takes both the URL variable and the current users account id and checks that the user has the permission to view the resource that they are requesting. The function provides a clean exit to prohibited data by finding the closest match to the requested data and returning that instead, minimising the amount of errors screens that a user might see when tampering with the URLs.

```php
public function handleDogIDFromGet($dog_id, $account_id){
    // This functon should take in the requested dog id and check that the user requesting access to
    $dogsBreeder = $this->dogGateway->getDogsBreeder($dog_id);
    if($dogsBreeder !== $account_id){
        // redirect the user to the dog id that the own that is the closest to the requested value
        $dog_ids = array();
        $ownedDogs = $this->dogGateway->getAllDogsByBreeder($account_id);
        foreach($ownedDogs as $dog){
            array_push( &array: $dog_ids, $dog['dog_id']);
        }
        $new_id = $this->dogGateway->getClosest($dog_id, $dog_ids);
        if(isset($new_id)){
            return $new_id;
        }
        // user has no data relocate them!
        header( string: "Location: ../../Pages/index.php");
    }
    else {
        return $dog_id;
    }
}
```

Figure 5.5-8 – Handle GET variable function

5.5.2.4 Database Protection

When providing input interfaces for users that interact with a database, there is the chance that some malicious users may attempt to perform SQL injection attacks. These are attacks that attempt to halt the original intended SQL statement in order to execute their own statements. These could be statements that attempt to read sensitive data or could be destructive commands that attempt to drop database tables and break the application.

To protect against these attacks, two major mitigation strategies have been implemented. Firstly is the introduction of the **execPreState** function that is declared in the **TableGateway** class. This is the main function in which all SQL commands are funnelled to perform their interaction with the database. This function, shown in Figure 5.5-9, executes the SQL command in what is known as a prepared statement. A prepared statement protects against SQL injection attacks as the query is formed before the parameters are passed to it, this means that it is impossible for an attack to prematurely end a statement to execute their own, as by the time their input is read, the statement is already formed.

```php
protected function execPrepState($sql, $array){
    try {
        $pdo = $this->connect();
        $stmt = $pdo->prepare($sql);
        $stmt->execute($array);
        return $stmt;
    } catch (PDOException $e) {
        echo "Connection Failed: " . $e->getMessage();
        return false;
    }
}
```

Figure 5.5-9 – execPrepState function that is used for all database communication

A second layer of protection is formed from using a newly defined database user to perform all interactions with the database, rather than using the admin user. This new user has limited permissions that allow them to interact with the data in the database, but prevents them from altering database structure and other powerful functions.

5.5.2.5 Redirection

One of the key security concerns of this project is the existence of two distinct users that have separate views and access to the application. It is important for the applications security that both of these users are only able to view data and functionalities that their accounts allow.

```php
$currentURL = $_SERVER['REQUEST_URI'];
if(strpos($currentURL, needle: "/Breeders/") !== false){
    // Then we are at a breeder section
    $inBreeder = true;
}
else if(strpos($currentURL, needle: "/Owners/") !== false){
    // Then we are at an owner section
    $inOwner = true;
}
```

Figure 5.5-10 – URL check for redirector

To help ensure that this is the case and that users are only able to view their own content, a utility page was produced and included on every viewable page of the application. The function of this utility page is to check the URL that the user is currently attempting to visit and determine whether it is a page that they have access to, Figure 5.5-10. In the case that the user is on a page that they should not be viewing, then the page redirects them to the index page of their viewable content, Figure 5.5-11. It achieves this functionality by checking session variables and abusing the directory structure of the application to see which section is currently being viewed.

```php
} else if($_SESSION["usertype"] == 0) {
    // This is a non-breeder user
    if($inBreeder){
        header( string: "Location: /FYP/Pages/Owners/index.php");
    }
}
```

Figure 5.5-11 – Example of redirection for an Owner attempting to access Breeder content

5.5.3 Design Conflicts

Throughout the implementation of the server-side application there were sections of the design that had to be scrapped or altered. This section addresses these conflicts and the compromises made.

5.5.3.1 Interface

The initial design of the interface proposed several pages that had to be scrapped due to time constraints, complexity of implementation and having a low MoSCoW priority. The main functions that were scrapped included:

- A events planning system
- A calendar that displayed the planning system
- Contract generation and related template pages

While contract generation was of a higher priority, it was scrapped as a concept due to the technical complexity of implementing the functionality compounded with a limit of the APIs that facilitate the requirement.

Contrastingly to the removed functionalities, there were a range of unplanned pages that were included in the interface of the application. The vast majority of these pages were input forms that were required to facilitate the ability of the system to deal with user interaction. The newly included forms included:

- Adding new parents
- Adding new litters
- Editing/removing existing parents

- Editing/removing existing litters
- Adding health entries
- Editing/removing health entries
- Adding dog notes
- Editing/removing dog notes
- Editing account details

Another page that was added to the interface as a result of the server-side application's implementation is the loading page for the family tree. This page was included due to the time taken for the converting of data, writing and reading from files and the kingraph process. Without the loading page, the user would experience a severe delay which emulated the experience of a server hang, the loading page alerts the user that there is a process that is happening and that it might take some time if the request is larger.

The family tree page is another page that experienced changes from the original design. Originally the design showed the tree orientated vertically, with the parent at the top of the tree. However, this was not the case in the final implementation, this change resulted in the library that was used to generate the tree, kingraph, this program does not allow for the desired orientation. A compromise was made by including the ability to pan and zoom on the resulting image, a functionality that was valued higher than the orientation.

5.5.3.2 Database

In addition to the changes enacted on the interface, there were a series of design choices that had to be revised for the database. These changes and their reasoning are:

- **Account** "password" length was increased to 255 characters as the initial 50 wasn't long enough to cater for the hashed values.
- **Breeder's** "twitter_handle" is nullable as a user is not required to have a twitter account to use the application.
- **Breed's** "description" is null by default and not changed anywhere in the application, this functionality was never implemented.
- **Dog's** "collar_colour" is nullable, this is because parent dogs do not have a collar colour as it is a method of identification.
- **Photo's** "width" and "height" are nullable and set to null by default, the ability to use these values to format images was instead hard-coded into the application where photos appeared.
- **Breeder's** "latitude" and "longitude" values were changed from integers to decimals as the integer type does not allow for decimal points.
- **Breeder's** "biography" was increased to hold 1500 characters as the initially proposed length was not long enough to fit the style of the page.

As well as fields being altered for the database, there were a series of fields that were either added, deleted or moved between tables. The fields affected by these changes and their reasoning are:

- Removing the "salt" from the **Account** because the PHP hashing function appends the salt to the password removing the need to store the salt separately.

- **Account** had the field of "username" added so that each user has a personal and unique value that they can use to identify themselves on login.
- **Account_Notification** had the field of "unread" added so that the application can correctly display the number of new notifications for a user.
- The "link" field was moved from **Notification** to **Account_Notification** to allow for different links for different users. The initial implementation meant that it was not possible to customise the links outside of directing users to a static page.
- The **Owner_Details** table had the "latitude" and "longitude" fields added, this is to reduce the number of API calls that were being made to fetch these values when displaying parent statistics. These calls were increasing the loading time of the page as abusing the resource every time the page was loaded.

One of the more notable changes to the database structure was the change to **Account_Notification's** primary key, initially this key was a composite primary key of both "account_id" and "notification_id". However, this is a very limiting key as it meant that each user could only receive each type of notification once. To combat this limitation, the primary key was extended to include the "date" and "time" values as a 4 field primary key. Meaning that users can receive many notifications rather than just one. This change was made rather than creating a new field as a primary key as it allowed for the preservation of existing data, it is noted that a single primary key would be easier and computationally cheaper to query.

Finally, although no changes were made due to this, it is worth noting that the **Dog_Photo** table never saw any implementation and holds no data. The project time-scale didn't allow for this functionality to be implemented, and as a result the table is not used throughout the application.

5.5.4 Other Notable Functions

During the implementation there were several functions that were produced that should be noted, but aren't fitting with any of the previous headings, those functions will be explained here.

5.5.4.1 Photo Uploads

While some of the photo functionality was never implemented, there was also some that was successfully implemented. The implemented photo uploads centred around the photos that can be seen on a

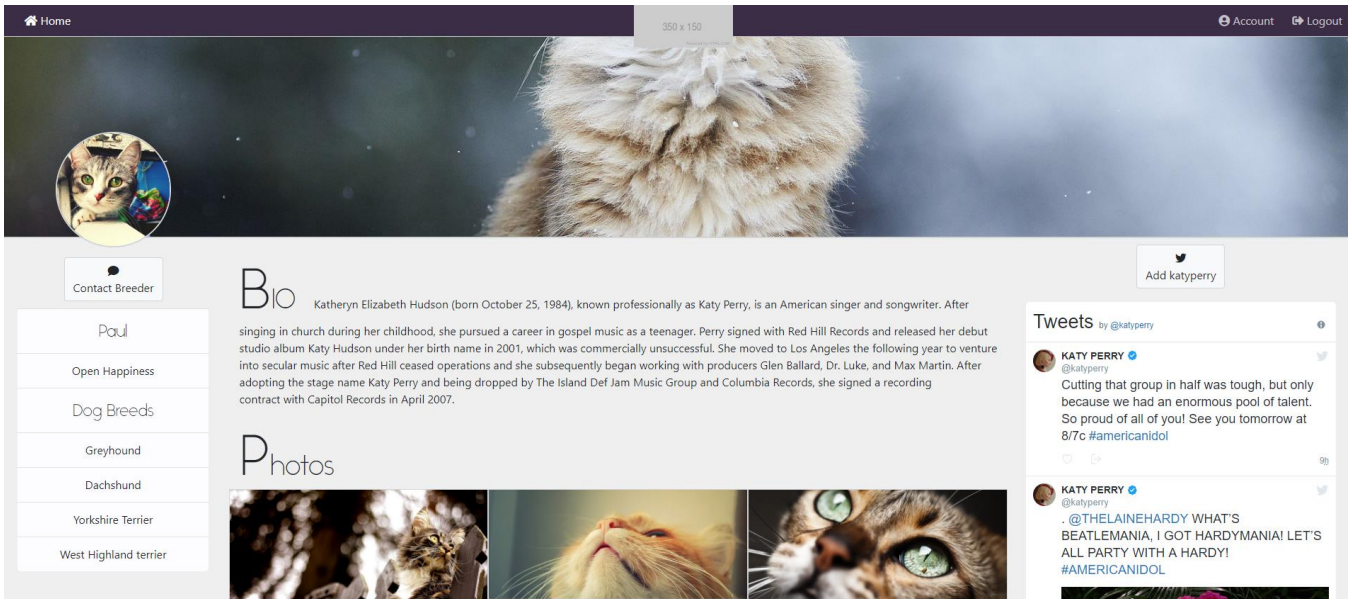


Figure 5.5-12 – Ironic dog breeder's profile page filled with cat photos

breeder's public profile, Figure 5.5-12 shows an example of a profile page.

To preserve the privacy of the user, these photos are not saved in the wwwhome, and are therefore inaccessible without being logged into the application. Initially this function was limited for the use of the dog photos but is also used here.

Because the photos are not stored on the web home directory, showing them on the page isn't as simple as providing a link to the file, instead a utility page had to be constructed. This page worked in conjunction with the **PhotoManager** to return a header with an image content-type and the contents of the requested image. This essentially converted the utility page into the requested image and could be

```
<div class="puppy-profile">
    <img id="profile-image" src="../Utils/imageFetch.php?profile=1&account_id=<?=$_GET['breeder_id']?>" />
</div>
```

Figure 5.5-13 – Example of utility function being used as an image reference

used to reference the requested images, see Figure 5.5-13.

The utility page also, with the help of the manager, ensures that the user has the permission to view the requested file. However, as this functionality was only ever implementated of the public facing profile of the user, the security features of the page go unutilised.

5.5.4.2 AutoLoader

Because PHP doesn't natively support packages, classes cannot interact with one another without using a reference to the class file. When a file uses a larger number of classes, it can become arduous to ensure that it has all of its dependencies. The autoload utility aims to alleviate this inconvenience.

Figure 5.5-14 shows this autoload utility, running the function everytime a class is loaded. Using a callback function, the utility checks if the classname contains "Gateway" and depending on the result either attempts to include the class from the Gateaways or the Managers directory. By including this file in all classes, and utility pages ensuring all dependancies are loaded becomes one less thing to worry about.

```
spl_autoload_register(function ($class_name) {
if (strpos($class_name, needle: 'Gateway') !== false) {
    include_once $_SERVER['DOCUMENT_ROOT'] . "/FYP/Gateways/" . $class_name . '.php';
}
else {
    include_once $_SERVER['DOCUMENT_ROOT'] . "/FYP/Managers/" . $class_name . '.php';
}
});
```

Figure 5.5-14 – Autoload utility

5.5.4.3 Twitter Account Verification

When a breeder adds their Twitter handle to their account, or when a new handle is followed, it is important to ensure that the handle in question is a real Twitter handle so that the implementation of showing these handles doesn't break.

To facilitate this requirement a function was created in the **TwitterManager** that sends a HTTP request to "https://twitter.com/users/username_av

```
$query = "https://twitter.com/users/username_available?username=".$handle;
$data = file_get_contents($query);
if($data) {
    // convert into readable format
    $data = json_decode($data, assoc: true);
```

Figure 5.5-15 – Checking accounts existence

ailable?username=<handle>" URL where <handle> is the inputted handle name. By checking the returned JSON from this request it could be determined whether a handle was "taken" or not. If the handle was taken then it was confirmed to be an existing account, see Figure 5.5-15.

However, using this URL sometimes gave unusual results, for example this method would return that the handle "BarackObama" is not a taken account. This is not the case. This happens because checking certain handles would return the status of "is_banned_word" even though an account could exist. Figure 5.5-16 shows the code that was used to confirm this edge case, by accessing the URL for a handles feed page and checking the returned headers to see if the account exists. This is an expensive process as the returned value from the query is the entire page, for this reason this check only occurs after the "is_banned_word" status is returned.

```
else if($data['reason'] == "is_banned_word"){
    // Some accounts have banned words in them and require a tad m
    // eg. BarackObama is banned but a legit handle
    $headers = get_headers( url: "https://twitter.com/".$handle);
    if(strpos($headers[0],  needle: '404') !== false ) {
        return false;
    } else {
        return true;
    }
```

Figure 5.5-16 – Code that checks the edge case

5.6 Summary

During the implementation process a deployment pipeline was created and used to deploy changes to the live public application running on the host server. These changes included the development of the interface, using HTML, CSS, PHP and JQuery and the implementation of the data using MySQL and PHPMyAdmin.

The database was populated with a series of test data to ease the production of the interface and to properly demonstrate the applications capabilities. The creation and population of the database was extracted into a series of repeatable SQL commands for a deployable process, changes to the testing database do not propagate to the hosting server using the pipeline.

Once the interface and the database were implemented the server-side application, that serves the data to the front-end from the database, was created. This PHP application was built to handle the requirements of the system and heavily followed the Table Data Gateway design pattern to manipulate, verify and authenticate functions. The bulk of the project's implementation was focused on the creation of this middleware.

Finally the design conflicts that arose from during the implementation were addressed and resolved. In the next section of the report, the resulting implementation of the project will be validated. This validation will determine the usability of the system and will detail which requirements have and have not been fulfilled by the final product.

6 Validation

This chapter attempts to test the implementation from the previous chapter. It does this by testing how well the final product fulfils the initial requirements and analysing feedback provided from the client.

6.1 Usability Testing

Black box testing is when the testing agent has no knowledge of the technical workings of the application that they are testing. Using this technique allows for a removal of bias from the testing process. Table 6.1-1 shows each of the functional requirements that were identified in the analysis chapter and how an uninformed user interacts with the system when asked to attempt to complete the requirements using the application. Table 6.1-1 also details what is expected of the system and what the actual results of the testing showed.

ID	Requirement	Expected Outcome	Actual Outcome
UC01	Register new account	User is able to register a new account quickly, without issue and without assistance	As expected.
UC02	Login to existing account	With an existing account, as user is able log into the system without issue	As expected.
UC03	Update account name	With a logged in account, a user is able to find the option to update their account name	User took longer than expected to find the options to change their personal details, but was able to make the changes once found.
UC04	Update account email	With a logged in account, a user is able to find the option to update their account email	After UC03 was completed, user had no issues.
UC05	Update password	With a logged in account, a user is able to find the option to update their account password	After UC03 was completed, user had no issues.
UC06	Update tagline	With a logged in account, a user is able to find the option to update their account tagline	After UC03 was completed, user had no issues.
UC07	Update bio	With a logged in account, a user is able to find the option to update their account biography	After UC03 was completed, user had no issues.
UC08	Update photos	With a logged in account, a user is able to find the option to update their account photos, with the ability to upload new phots and remove old ones	User expect to find this option in the same place as the other account options. Much longer to find this option and user struggled with the interface more than expected.
UC09	Update Twitter link	With a logged in account, a user is able to find the option to update	After UC03 was completed, user had no issues.

		their account twitter handle	
UC10	Add feed to main page	With a known Twitter handle a user is able to add that feed to their main page	As expected

*Table 6.1-1 - Table containing the results of the black box testing, see **Appendix D (D.1)** for the remaining requirements*

6.2 Performance Testing

Performance testing of the application is a process defined in this project as testing the core functionalities of the application, e.g. whether the database is capable of performing basic and complex queries, whether the security is up-to-scratch and how the application works under pressure.

This testing is conducted to ensure that the application is deployment ready and capable of scaling functionality as new requirements appear.

6.2.1 Database Validation

The database validation will be performed using the PHPMyAdmin utility to allow us to execute SQL commands with ease, validation will conducted by taking three common requirements of the application and testing their functionality and that their constraints act as expected.

Some of the typical requirements of the system include, adding a new dog entry, creating a litter of dogs and viewing the owner details of a litter.

When inserting data into the Dog table, there are two foreign key constraints that must be adhered to. Figure 6.2-1 shows the result of a successful query, showing that it is possible to fulfil the requirement of inserting new Dog entries.

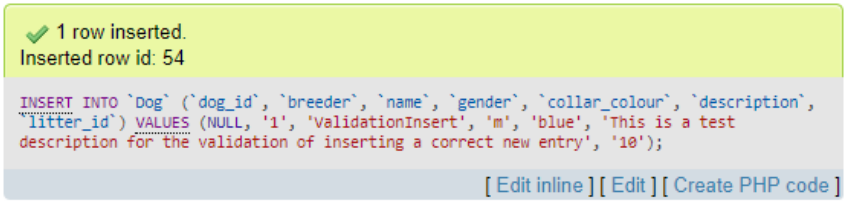


Figure 6.2-1 – Successfully inserting data into the Dog table

After confirming that it is possible to insert data into the Dog table, it is also important to ensure that the constraints work and prevent data from being entered that break them. Figures 6.2-2 and 6.2-3 show that these constraints work and violations prevent data from being entered successfully.

Error

SQL query:

```
INSERT INTO `Dog` (`dog_id`, `breeder`, `name`, `gender`, `collar_colour`, `description`, `litter_id`) VALUES (NULL, '1', 'IncorrectLitterID', 'f', 'Blue', 'This should fail to insert entry as the litter id doesn\'t match an exiting litter', '11')
```

MySQL said:

```
#1452 - Cannot add or update a child row: a foreign key constraint fails (`fyp`.`Dog`, CONSTRAINT `litter_id` FOREIGN KEY (`litter_id`) REFERENCES `Litter` (`litter_id`))
```

Figure 6.2-2 – Foreign key of litter has to be a real litter ID, violations prevent entry and give error

Error

SQL query:

```
INSERT INTO `Dog` (`dog_id`, `breeder`, `name`, `gender`, `collar_colour`, `description`, `litter_id`) VALUES (NULL, '1', 'ThisIsANameThatIsTooLongToInsertAsItIsGreaterThan50CharactersThisSohouldThrowAnError', 'f', 'Blue', 'This should fail to insert entry as the litter id does
```

MySQL said:

```
#1406 - Data too long for column 'name' at row 1
```

Figure 6.2-3 – Length of name must be under 50 characters of constraint is violated and error is thrown

1 row inserted.
Inserted row id: 11

```
INSERT INTO `Litter` (`litter_id`, `DOB`, `sire`, `bitch`) VALUES (NULL, '2019-04-02', '1', '2');
```

[Edit inline] [Edit] [Create PHP code]

Figure 6.2-4 – Data is successfully inserted into the litter table

Creating a litter of dogs is another common function that the application will perform. Thus, it is important to ensure that it is possible to insert data when all the constraints are adhered to. Figure 6.2-4 shows the successful insertion of data into the litter.

The main constraints for the litter table is the foreign keys of sire and bitch that must match existing dog ids. Figure 6.2-5 shows the results of this constraint being violated.

The last typical requirement of the system that will be validated is the ability to fetch the owner data for all the dogs in a litter. This requirement acts to ensure that the relationships in the database are conducive for more common complex queries and that they complete within a reasonable time frame. Figure 6.2-6 show the results of the query, proving that it is possible to join the tables correctly. It also shows that the query only took 0.0016 seconds, which is fast enough to be

Error

SQL query:

```
INSERT INTO `Litter` (`litter_id`, `DOB`, `sire`, `bitch`) VALUES (NULL, '2019-04-02', 100, 100)
```

MySQL said:

```
#1452 - Cannot add or update a child row: a foreign key constraint fails (`fyp`.`Litter`, CONSTRAINT `Litter_ibfk_1` FOREIGN KEY (`sire`) REFERENCES `Dog` (`dog_id`))
```

Figure 6.2-5 – Violations to the sire and bitch fields cause a an error to be thrown and data is not inserted

Showing rows 0 - 5 (6 total, Query took 0.0016 seconds.)

```
SELECT Owner_Details.* FROM Dog JOIN Ownership_History ON Dog.dog_id <=> Ownership_History.dog_id JOIN Owner_Details ON Ownership_History.owner_id = Owner_Details.owner_id WHERE Dog.litter_id = 2
```

Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

owner_id	account_id	name	contact_email	phone_number	house_number	street_name	city	postcode	longitude	latitude
44	NULL	ROSIE	rosie@mail.com	5490955816	55	Cedar Street	Sheffield	PO12 1FT	-1.1338	50.7906
47	NULL	LAYLA	layla@mail.com	1154546028	49	Bridge Street	Leicester	GU34 1LT	-0.9851	51.1444
84	NULL	HARVEY	harvey@mail.com	9300140803	67	10th Street	Bristol	TR13 0GA	-5.2569	50.1114
74	NULL	JAKE	jake@mail.com	9027849653	17	12th Street	Archway	E16 3NZ	0.0273	51.5121

Figure 6.2-6 – Results of a more complex data query with two joins

negligible in the system.

6.2.2 Security Validation

To validate the security of the system and its access to the database, an SQL injection attack will be simulated. For the demonstration a destructive SQL command will not be used, as damaging the database is not intended. Instead an insert command will be used to determine whether the attacker has the ability to execute SQL commands from the forms provided in the interface. Figure 6.2-7 shows the form and injection attempt.

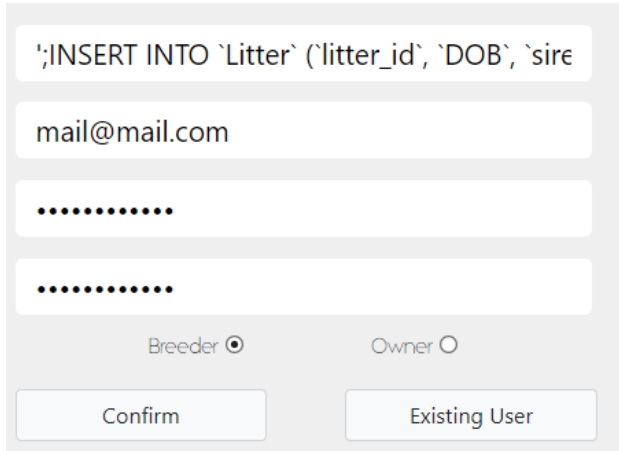


Figure 6.2-7 – Account registration form with a nasty SQL injection attack.

The attack is formed so that it ends the quotation of the original SQL statement and attempts to execute the attackers command as well, in a real attack this statement would likely be in the form of a table drop.

Figure 6.2-8 shows the Account table after the submission of the form, it reveals that rather than ending the original statement, the attack is now an entry in the Account table. This is the behaviour that proves that the prepared statements, mentioned in the previous chapter, work correctly. To futher

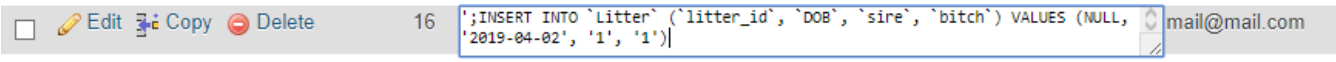


Figure 6.2-8 – Resulting entry in the Account table after the simulated attack

ensure that the command has not been executed, Figure 6.2-9 shows the full Litter table. This Figure confirms that the test attack has not managed to enter its data into the table, as there is no entry that has the DOB set as '2019-04-02' which is the DOB of the attack insert.

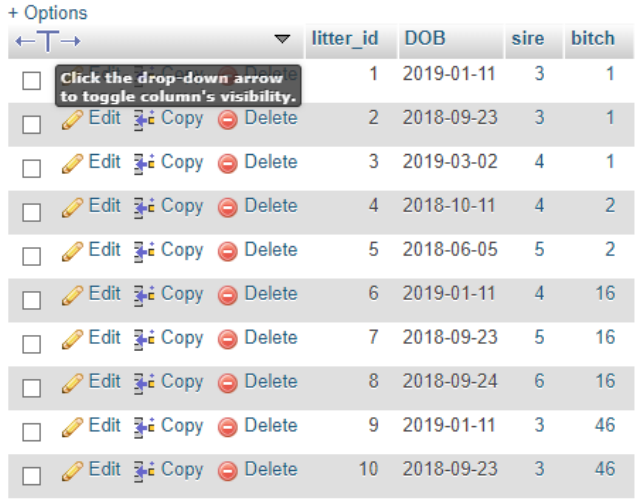


Figure 6.2-9 – Litter table after the simulated attack.

6.2.3 Non-Function Requirements Validation

This section of the validation will focus on the non-functional requirements identified in section 3.5.2. Each of these requirements will be tested to see whether the implementation of the project has successfully adhered to the requirements. Table 6.2-1 holds the tests performed for each requirement and whether that requirement was satisfied. All figures used as evidence for the non-functional requirements can be found in **Appendix D (D.2).**

No.	Requirement Summary	Tests	Outcome	Evidence Figures
1	Responsive interface	Measure loading times for pages on the interface.	The average loading time of pages is satisfactory. The longest page has unacceptable loading times, the impact is mitigated with the loading page.	D.2-1, D.2-2
2	Uninterrupted service	**Untestable** – Currently there are no tools available to test the uptime of the hosting server, from general usage the uptime appears to be consistent.		
3	Serve 50 minimum concurrent users	Use the apache benchmarking service to test the server ability to handle concurrent users.	For a bulk of 5000 requests, 50 concurrent requests were completed with an average of 1 millisecond per request. The system handled 100 concurrent request with a reduction to 0.6 milliseconds per request. 200 concurrent was the limitation of the current server.	D.2-3
4	Passwords hashed	Create password via the interface and check the resulting entry in the database.	Passwords created using the interface are inserted into the database as hashed values.	D.2-4
5	Fast authentication	Log into the application a series of 10 times and average the time taken.	Average load time was recorded to be 1.6 seconds, which is well within the acceptable range specified in the initial requirement.	D.2-5, D.2-6
6	System backups	**Not Implemented**		
7	Modular and maintainable code	**Untestable** – The structure of the code implies that the code should be modular. The code should be maintainable due to the production pipeline and the source control that it uses.		
8	Continuous delivery	**Untestable** – There is the option to implement continuous delivery with the applications production pipeline that allows for automatic update to make to the application.		
9	Intuitive interface	**Previously Tested** – Interface testing was conducted in section 6.1 with user testing.		

10	User storage capacity	**Not Implemented**		
11	Censoring of location data	Create location data via the interface and check that the resulting entry is limited to two decimal places	All location data that was enter into the application was rounded to two decimal places, obscuring the data as expected.	D.2-7

Table 6.2-1 – Table containing non-functional requirements testing and their results

6.3 Client Feedback

From the client's feedback, **see Appendix E (E.2),** there were two major functions that were identified to be improved in the next iteration of the product. These areas were, the ability to view guardian homes and owner details from the main menus and the error handling of the system.

Figure E.2-1 shows the clients ratings for the main areas of the project, from this it can be concluded that that the client is satisfied with the main areas identified in the projects objectives and that the client is satisfied with the overall result of the project.

6.4 Summary

The usability testing section of the validation showed that the vast majority of the requirements were fulfilled to a standard that allowed for users to be able to complete the expected functionality with ease. Furthermore, the usability testing showed the requirements that need improvement on the next iteration of the project.

The performance testing showed that the produced system is more robust than was specified as a non-functional requirement. The system can handle a large volume of concurrent users and the security features of the application act as they should.

Finally, the feedback received from the client indicates that the project has been completed to a satisfactory standard.

7 Critical Review

This chapter will review the project by looking at the requirements identified in the analysis and their completion. The project management and implementation will be critiqued and alternatives will be discussed.

7.1 Requirements Review

Out of the initial requirements identified in the analysis there were eight requirements that were not completed and five that were partially completed. Out of the requirements that were not completed, two had a MoSCoW priority of S, whereas the other six were C and W. The two requirements with the higher priority were UC13 (Update Acceptable Deviance) and UC43 (View Guardian Homes). These were not implemented into the final product as their initially identified priority was reclassified later on in the project as less important than other S requirements.

The partially completed requirements are classed as such due to the functionality existing in the system but the usability testing revealed that they were not as obvious and intuitive as they should be.

The full results of the requirements review is available in **Appendix E (E.1).**

7.2 Project Management Approach

The project management strategy that was planned for the creation of this project was a sequential design process, where a waterfall strategy was to be used to produce a minimum viable product (MVP) that would then have new features identified with user stories implemented using an iterative design process, such as DSDM.

This process was thought to act as the most efficient path to creating a product that the client would be most satisfied with. Where the most important features were implemented first and shifting focus away from unnecessary and supplementary features.

It reality this project management style quickly gave way to large periods of disjointed development, where large chunks of the application were developed at once and separate from end-to-end considerations. The entire interface of the application was developed first, followed by the entire database and finishing with the business logic of the application. This meant that there were many changes that had to be made to the interface and database when the business logic was applied. This created a large time sink where initial designs and implementations had to be reconsidered after their final development.

If this project were to be reconsidered, it would've been better to adopt the iterative development style from the start. This is because the user stories were actually crafted before development, rather than at the MVP, allowing for the prioritisation to be followed in a manner that allows the interface and database design could be segmented around features, rather than taking the whole project into consideration. This method would also account for the shortcomings in the requirements analysis as S level requirements should have been implemented before the C requirements that reached completion.

7.3 Changes to the Implementation

The design pattern used for the implementation was Table Data Gateway pattern, however whilst this pattern allowed for the separation of concerns, it added a large abstraction layer between the business logic and the database. Changes to the business logic would often lead to jumping around multiple gateways for a slight change to the end result. This was due to how the gateways would pass data, often just a specific SQL queries, and due to the inherent one-to-many relationship between managers and gateways. To better improve this structure, the implementation could have applied a one-to-one relationship with the managers, with one manager for every table in the database. Whilst this option would alleviate the aforementioned concerns, it would increase the complexity of the application structure.

An alternative pattern that could have been used, that still maintained the separation of concerns principle, would be the Model View Controller (MVC) pattern. This pattern aims to separate data, front end and business logic components in a similar fashion to the gateway pattern, but it does so with the use of objects instead of callable functions. This method could also have been applied to the project to reduce the complexity of the final code.

Other technical alterations that could have been made to the project include:

- Moving the large amount of inline JavaScript in their own separate files, this would reduce the amount of repeated code and reduce the clutter of the HTML pages.
- Creating templates of common and shared page elements, whilst this is done with the NavBar, there are several html snippets that could be refactored to reduce the code footprint.
- Implementation of AJAX functions, particularly for the form outputs. This would allow for a more seamless interaction and increased security as the utility pages could be made only accessible via AJAX requests.
- With a hosted server comes the ability to send emails to users, this could be implemented for functionalities such as, password recovery, account validation, contacting other users.

7.4 Time Management Strategies

The initial time management strategy that was planned for the project was the use of milestones in conjunction with the waterfall project management strategy. As previously discussed, the waterfall strategy was abandoned, this also lead to the disposal of the milestone strategy. Thus the time management strategy that was actually used for the project was a self-management strategy that was dictated by project deadlines rather than milestones.

This strategy was effective for the project, however this lead to an unstructured method of production with large time periods between deadlines. To improve upon this strategy a series of backlog epics would be constructed around the iterative development cycle of three weeks. This would allow for a dynamically populating list of epics with a shorter turn around period. This would work more effectively than the initial strategy as it is linked to each of the development cycles rather than the waterfall segments.

7.5 Conclusions

7.5.1 Review of Aims and Objectives

The primary aim of this project was to improve a breeder's ability to track and manage the details of their produced litters. In review of this aim and the implementation of the project, it can be concluded that the project has successfully completed this main aim. The primary objective that relates to the ability to view and monitor guardian homes is the only primary objective that was not completed. Whilst this objective itself was not completed as it is not possible to view guardian homes only, it is possible to see whether a dog is a part of a guardian home.

In consideration of the secondary aim, only half the identified aim was completed. The social platform aspects, contacting other users and event organisation, were not implemented but the searching functionality and the updates feeds were implemented with the use of the Google Maps API and the Twitter API respectively.

Overall, it can be concluded that the aims and objectives for this project have been completed to a high level of quality, despite the incomplete objectives, as the requirements related to the completed objectives were the requirements with the overall highest priority.

7.5.2 Strengths and Limitations

Strengths	Limitations
Modular and deployable code	Time management
Security of technologies used	Project management
Production pipeline and source control	One of the main requirements was not fulfilled
Strong application infrastructure	Small team
External hosting with near permanent uptime	

Table 7.5-1 – Table of projects strengths and limitations

7.5.3 Future Work

There are many directions that this project could explore, with the range of requirements that were not fulfilled. These features could be implemented to improve the functionality of the application for the Breeders and the Owners. Expanding the application so that it can handle viewing of guardian homes and other potential features such as waiting lists for puppies.

By following the goal of the secondary aim, the application could grow to serve as a social media, with event organisation, messaging and connecting with other users. The reliance on external APIs, such as Twitter, could be reduced and a more integrated alternative could be crafted.

The lessons that have been learnt from this project will be applied to all future endeavours, professional and personal alike. An enhanced understanding of project and time management will be beneficial for future projects and will be applied as such.

7.5.4 Final Conclusions and Closing Summary

This project focused on the production of an application that works to serve a breeder's need to manage and track the litters that they produce. During the course of the project, requirements were elicited and used to produce prototypes and designs, which in turn were used to implement a working product. The product was tested and the aims were reviewed to determine the validity of the project.

Overall, the project has been successful, meeting the main requirements of the system with an intuitive and user friendly application, modular and deployable enough to allow for a continuous delivery of features as the application expands in future work.

Bibliography

Animal Welfare Act 2006 (2006). Available at: https://www.legislation.gov.uk/ukpga/2006/45 (Accessed: 4 November 2018).

Belghiti, J. (2017) *git-auto-pull*. Keenpoint. Available at: https://github.com/Keenpoint/git-auto-pull (Downloaded: 30 March 2019).

Bernardes, D.H. (2015) *jquery-svg-pan-zoom*. Available at: https://github.com/DanielHoffmann/jquery-svg-pan-zoom (Downloaded: 30 March 2019).

Breed registration statistics (2018). Available at: https://www.thekennelclub.org.uk/registration/breed-registration-statistics/ (Accessed: 15 October 2018).

Breeding and Sale of Dogs (Welfare) Act 1999 (1999). Available at: https://www.legislation.gov.uk/ukpga/1999/11 (Accessed: 4 November 2018).

chartjs (2019) *Chart.js*. Chart.js. Available at: https://github.com/chartjs/Chart.js (Downloaded: 30 March 2019).

Chiang, I.-P., Lin, C.-Y. and Wang, K.M. (2008) Building Online Brand Perceptual Map. *CyberPsychology & Behavior*, 11 (5): 607–610. doi:10.1089/cpb.2007.0182.

Cruz, R.S. (2017) *kingraph*. Available at: https://github.com/rstacruz/kingraph (Downloaded: 30 March 2019).

Ford, M. (1999) Recent legislation. The Data Protection Act 1998. *Industrial Law Journal*, 28 (1): 57–60. doi:10.1093/ilj/28.1.57.

Fowler, M. (2003) *Patterns of Enterprise Application Architecture*. The Addison-Wesley Signature Series. 4th ed. Boston, Mass: Addison-Wesley.

Keller, K.L. (1993) Conceptualizing, Measuring, Managing Customer-Based Brand Equity. *Journal of Marketing*, 57 (1): 1–22.

Laybats, C. and Davies, J. (2018) GDPR: Implementing the regulations. *Business Information Review*, 35 (2): 81–83. doi:10.1177/0266382118777808.

New Dog Breeding Regulation (2019). Available at: https://www.thekennelclub.org.uk/breeding/dog-breeding-regulation/ (Accessed: 3 November 2018).

Odoom, R. (2016) Brand-building efforts in high and low performing small and medium-sized enterprises (SMEs). *Journal of Small Business and Enterprise Development*, 23 (4): 1229–1246. doi:10.1108/JSBED-05-2016-0067.

Office for National Statistics (UK) (2018). Available at: https://www.statista.com/statistics/556744/number-of-internet-users-in-the-uk-by-region/ (Accessed: 8 November 2018).

Richardson, B. (2003) Principles of the Data Protection Act: an evaluation. *Nursing and Residential Care*, 5 (3): 138–141. doi:10.12968/nrec.2003.5.3.11117.

Shoosmith, G. (2012) Why your business needs a website. *The Guardian*, 19 October. Available at: https://www.theguardian.com/small-business-network/2012/oct/19/businesses-need-websites (Accessed: 4 November 2018).

The Animal Welfare (Licensing of Activities Involving Animals) (England) Regulations 2018 (2018). Available at: http://www.legislation.gov.uk/uksi/2018/486/made (Accessed: 1 November 2018).

United Kingdom - Total population 2012 to 2022 | Statistic (2016). Available at: https://www.statista.com/statistics/263754/total-population-of-the-united-kingdom/ (Accessed: 8 November 2018).

What is a data controller or a data processor? (n.d.). Available at: https://ec.europa.eu/info/law/law-topic/data-protection/reform/rules-business-and-organisations/obligations/controller-processor/what-data-controller-or-data-processor_en (Accessed: 7 November 2018).

What is data? (2016). *Progressive Digital Media Technology News; London*, 24 June. Available at: http://search.proquest.com/docview/1799566542/citation/A7EDF619D5CB484EPQ/1 (Accessed: 5 November 2018).

What is market reach? definition and meaning (n.d.). Available at: http://www.businessdictionary.com/definition/market-reach.html (Accessed: 8 November 2018).

Zec, R. (2014) *name-that-color*. Available at: https://github.com/ryanzec/name-that-color (Downloaded: 30 March 2019).

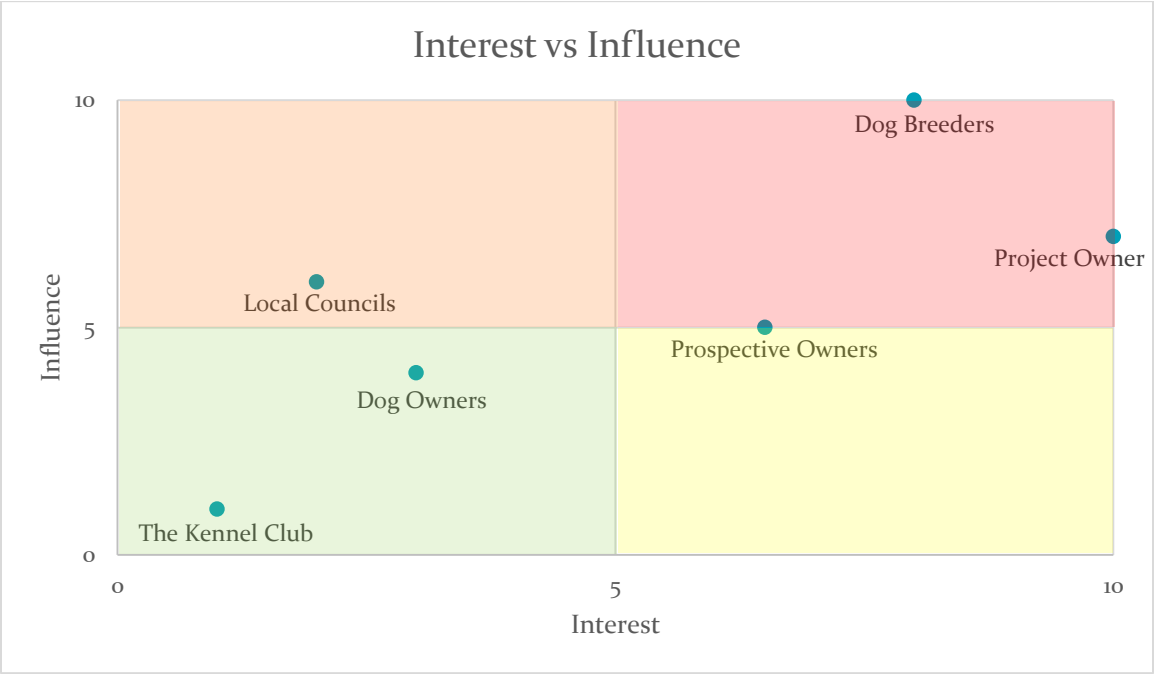
Appendices

Appendix A

A.1 Stakeholder Identification

Stakeholder	Interests	Estimated Interest (1-10)	Estimated Influence (1-10)
Project Owner	Project Completion Avoiding Liability Maximisation of Profits	10	7
Dog Breeders	Monitoring of Dog health Tracking dogs and owner relations Tracking of Guardian homes Protection of data Expanding business Storing of contacts and documents	8	10
Dog Owners	Protection of data Locating other owners in their area Staying "updated" with breeders and owners they "follow"	3	4
Prospective Owners	Finding breeders in their area Researching possible options (Breeders, Breeds) Contacting breeders Arranging viewings	6.5	5
Local Councils	Ensuring that breeders have the correct license for breeding Conditions in which dogs are kept	2	6
The Kennel Club	Registering of new dogs Genetic tracking application in close competition	1	1

A.2 Influence / Interest Matrix



A.3 System Actors

Actor Name: Prospective Dog Owner
Type: Primary
Role Description: The prospective owner must sign up to the system so that they have the ability to search their local area to find breeders. The prospective owner will use the system purely to contact breeders using the system to enquire about visiting, upcoming litters and breed details.
Actor Goals: • To find a breeders that fit to their own specification of distance, price and breed • To contact breeders to organise visits and purchasing routes • To purchase a puppy from the breeder
Related Requirements: Register new account, Login to existing account, Update account name, Update account email, Update password, Update photos, Add feed to main page, View dog details, View dog's DOB, View dog's gender, View dog's name, View dog's description, View breed details, View dog family tree, Locate breeders, View update feed, Update interested feeds, Message users

Actor Name: Dog Owner
Type: Primary
Role Description: The dog owner is a user that already owns a dog, if they are looking to purchase another dog then they fall into the prospective owner actors. Owners that are not looking for a new dog are users that would use the system to find other owners to organise social events and stay updated with "feeds" that interest them.
Actor Goals: • Find other owners • Message other owners • Follow the feeds that are interesting to the individual • Keep the details on their dog up-to-date
Related Requirements: Register new account, Login to existing account, Update account name, Update account email, Update password, Update photos, Add feed to main page, View dog details, View dog's DOB, View dog's gender, View dog's name, View dog's description, View breed details, View dog family tree, Locate breeders, View update feed, Update interested feeds, Message users

Actor Name: Database
Type: Secondary
Role Description: The role of the database is to store the details relating to primary actors of the system. It should also

hold the data for the dogs that are related to each of the actors and those relationships themselves. It must also allow for querying to return the needed data for each actor.

Actor Goals:

- Store information for primary actors
- Store dogs related to actors and the relationships
- Serve the data when requested

Related Requirements:

Register new account, Login to existing account, Update account name, Update account email, Update password, Update tagline, Update bio, Update photos, Update Twitter link, Add feed to main page, Add puppy health entry, View puppy health, Update acceptable deviance, Notify health outside deviance, View dog details, View dog's DOB, Update dog's DOB, View dog's gender, Update dog's gender, View dog's name, Update dog's name, View dog's description, Update dog's description, View dog's collar, Update dog's collar, View breed details, Update breed details, View dog family tree, Update family tree, View dog's owner, Update dog's owner, Add new dog / puppy, Remove dogs, Add puppy to litter, Remove puppy from litter, Add puppy visits to calendar, Generate contracts, Update contract template, Locate breeders, View update feed, Update interested feeds, Message users, View guardian homes, Notify guardian home seasons

Actor Name: Google Maps API

Type: Secondary

Role Description:

The role of the Google Maps API is to take addresses and covert them to longitude and latitudes. It should also use the users' location to find other locations from the database that are within a specified range of the current location.

Actor Goals:

- Convert addresses into lats and longs
- Display a map for the user
- Find the users current location
- Find addresses within a specified range of the current location

Related Requirements:

Locate breeders

Actor Name: PDF Generation API

Type: Secondary

Role Description:

The PDF generator's role is to take a populated HTML template and to produce a viewable and printable pdf document for the breeder to print and use with prospective owners.

Actor Goals:

- Convert HTML page into a pdf document

Related Requirements:

View dog details, View dog's DOB, View dog's gender, View dog's name, View dog's owner, Generate contracts

A.4 User Stories

As...	~Condition	I want to...	So that...
Dog Owners		Find other dog owners near me	I can organise walks and socialising events
Dog Owners	Who is looking to get another dog	Find breeders near me	I can organise visits to see dogs near me
Dog Owners		View a "feed" of updates from followed owners and breeders	I can keep up-to-date on news relating to my dog and other dogs in the local area

As...	~Condition	I want to...	So that...
Prospective Dog Owners		Find breeders in my local area	I can organise visits to see dogs near me
Prospective Dog Owners		See details about breeds	I can make an informed decision on the dog breed I am looking to buy
Prospective Dog Owners		View a "feed" of litter updates for litters that I am interested in	I can keep as up-to-date as possible on litters that I may want to buy a puppy from
Prospective Dog Owners	Who have found a puppy that they are interested in	Message breeders about their litters through the platform	I can organise a visit to the puppies in question and organise purchase of puppies without giving out my personal information

A.5 Functional Requirements

ID	Use Case	Description
UC01	Register new account	The user can use their email to create an account as a Breeder, Dog Owner or a Prospective Owner.
UC02	Login to existing account	The user is able to use their email and password to gain access to the system and subsequent features and functionalities
UC03	Update account name	The user has the ability to update how they are referred to by other users of the system
UC04	Update account email	The user has the ability to change the email that is associated with their account
UC05	Update password	The user is able to change the password that is used to access the system
UC06	Update tagline	Breeders have the ability to change the tagline that is linked to their profile
UC07	Update bio	Breeders are able to edit the bio section of their account
UC08	Update photos	Users are able to add and remove photos associated with their account
UC09	Update Twitter link	Breeders are able to change which twitter feed is shown on their page, none is also an option
UC10	Add feed to main page	Users are able to add feeds that interest them to their personal splash pages
Uc11	Add puppy health entry	Breeders are able to add a date and a weight so that the overall health of a puppy can be monitored effectively
UC12	View puppy health	Breeders are able to see the data linked to a dogs health so that they are able to make their own judgements on the health
UC13	Update acceptable deviance	Breeders are able to update the range at which a puppy is considered to be dangerously underweight compared to its littermates
UC14	Notify health outside deviance	Breeders are alerted when the weight of a puppy is lower than has been set as the acceptable range
UC15	View dog details	Users are able to see details about dogs that they have permission to see
UC16	View dog's DOB	Users are able to view the DOB of dogs that they have permission to view
UC17	Update dog's DOB	Breeders have the ability to change a dog's DOB in the case that the data was entered incorrectly
UC18	View dog's gender	Users are able to view the gender of dogs that they have permission to view
UC19	Update dog's gender	Breeders have the ability to change a dog's gender in the case that the data was entered incorrectly

UC20	View dog's name	Users are able to view the names of dogs that they have permission to view
UC21	Update dog's name	Breeders and Owners have the ability to change a dog's name as they change ownership
UC22	View dog's description	Users have the ability to see the description of dogs that they have permission to see
UC23	Update dog's description	Breeders have the ability to update the description of a dog as it matures and it's features and personality changes
UC24	View dog's collar	Breeders have the ability to view the collar colour of dogs so that they have a way to reference information on the system to the dogs that they own
UC25	Update dog's collar	Breeders have the ability to update a dog's collar colour in case they outgrow the collar or it is lost
UC26	View breed details	Users have the ability to view descriptions of different dog breeds
UC27	Update breed details	Breeders have the ability to update descriptions about the breed of the dogs that they own
UC28	View dog family tree	Users have the ability to view the family tree of dogs that they have permission to view
UC29	Update family tree	Breeders have the ability to update the relationships between their dogs that allow the system to build the family tree view
UC30	View dog's owner	Breeders have the ability to see which owners have reserved which of their puppies
UC31	Update dog's owner	Breeders have the ability to update which user owns which dog
UC32	Add new dog / puppy	Breeders have the ability to add new dogs to the system
UC33	Remove dogs	Breeders have the ability to remove dogs from the system in the case of an incorrect entry
UC34	Add puppy to litter	Breeders have the ability to add an existing dog to a grouping of dogs, represented as a litter
UC35	Remove puppy from litter	Breeders have the ability to remove dogs from a litter in case they are added incorrectly
UC36	Add puppy visits to calendar	Breeders are able to add events to a calendar to better organise their puppy related events
UC37	Generate contracts	Breeders have the ability to generate pdf contracts using a template and information about owners and puppies
UC38	Update contract template	Breeders have the ability to specify the own contract by use of a WYSIWYG editor
UC39	Locate breeders	Users have the ability to find breeders using a set of filters including but not limited to; distance, price and breed
UC40	View update feed	Users have the ability to view a "feed" of updates from

		breeders and other twitter feeds that are of interest to the user
UC41	Update interested feeds	Users have the ability to update the list of Twitter feeds that are of interest to the user and therefore appear on their main splash page
UC42	Message users	Users have the ability to send messages to other users of the system
UC43	View guardian homes	Breeders have the ability to view dogs that were sold with the guardian home contract
UC44	Notify guardian home seasons	The system should alert Breeders when guardian home dogs are coming into season so that the breeder can return on their investment

A.6 Use Case Textual Descriptions

UC01	Register new account	
Actor(s)	Dog Breeders, Dog Owners, Prospective Dog Owners	
Description	The user can use their email to create an account	
Pre-conditions	The actor has an email address	
Success End Condition	The user has an account on the system	
Failed End Condition	The user does not have an account on the system	
Main Flow	Step	Action
	1	User accesses webpage
	2	User clicks the register button
	3	User enters their details
	4	User logs onto the system
Alternative Flows	Step	Action
	N/A	N/A

UC03	Update account name	
Actor(s)	Dog Breeders, Dog Owners, Prospective Dog Owners	
Description	The actor can change the display name of their account	
Pre-conditions	The actor has an account and is logged into the system	
Success End Condition	The user's display name has been changed	
Failed End Condition	The user's display name is unchanged	
Main Flow	Step	Action
	1	User clicks "My Account"
	2	User clicks "Edit"
	3	User completes the required changes
	4	User clicks "Save"
	5	The system updates the database to reflect the change
Alternative Flows	Step	Action
	N/A	N/A

UC04	Update account email	
Actor(s)	Dog Breeders, Dog Owners, Prospective Dog Owners	
Description	The actor can change the email of their account	
Pre-conditions	The actor has an account and is logged into the system	
Success End Condition	The user's email has been changed	
Failed End Condition	The user's email is unchanged	
Main Flow	Step	Action
	1	User clicks "My Account"
	2	User clicks "Edit"

	3	User completes the required changes
	4	User clicks “Save”
	5	The system updates the database to reflect the change
Alternative Flows	Step	Action
	N/A	N/A

UC05	**Update password**	
Actor(s)	Dog Breeders, Dog Owners, Prospective Dog Owners	
Description	The actor can change the password of their account	
Pre-conditions	The actor has an account and is logged into the system	
Success End Condition	The user’s password has been changed	
Failed End Condition	The user’s password is unchanged	
Main Flow	Step	Action
	1	User clicks “My Account”
	2	User clicks “Edit”
	3	User completes the required changes
	4	User clicks “Save”
	5	The system updates the database to reflect the change
Alternative Flows	Step	Action
	3a	User enters a invalid password
	3a1	User enters a password that isn’t strong enough
	3a2	User is given a warning and description of what is considered a strong password
	3a3	System does not redirect the user
	3b1	Password doesn’t match the “verify password” field
	3b2	User is alerted that the passwords do not match
	3b3	System does not redirect the user

A.7 MoSCoW Analysis Justifications

ID	Requirement	Justification	M/S/C/W
UC01	Register new account	For the user to be able to use the system they have to be able to log onto the system in the first place. Without an account with related name, email and passwords they would be unable to log on and therefore would be unable to use the system at all, for this reason, these requirements are a must.	M
UC02	Login to existing account		M
UC03	Update account name		M
UC04	Update account email		M
UC05	Update password		M
UC06	Update tagline	The breeder has a profile that other users are able to see, they have to be able to update the details that are shown on this page otherwise all profiles would be the same. The ability to do this is a must.	M
UC07	Update bio		M
UC08	Update photos		M
UC09	Update Twitter link	The ability for users to have a Twitter feed related to their account fulfils the user story of having an update feed without a large time investment to implement a feed system. This requirement is only a "should" as the system would function without an update feed system in place.	C
UC10	Add feed to main page		C
Uc11	Add puppy health entry	Being able to monitor the health of a puppy is one of the main requirements of the system, having infrastructure in place to support this is necessary, being able to add entries and view the health facilitates this need.	M
UC12	View puppy health		M
UC13	Update acceptable deviance	Using the health data to alert a breeder to an unacceptable deviance of weight in a puppy is a very important requirement. However, this is only a "should" requirement as it isn't a feature that has been specifically requested from the client and it is possible to detect without an automated system in place.	S
UC14	Notify health outside deviance		S
UC15	View dog details	Being able to view the details of the dogs in the system is a "must" requirement as the system will not be able to function without the users being able to see details about the dogs. In addition to this, it isn't enough to be able to view the dog's details, the Breeder should also be able to update the details about the dogs so that they are as up-to-date as possible. UC16 – UC25 and UC30 and UC31 are	M
UC16	View dog's DOB		M
UC17	Update dog's DOB		M
UC18	View dog's gender		M
UC19	Update dog's gender		M
UC20	View dog's name		M

UC21	Update dog's name	extensions of UC15, because this requirement is a "must", by extension the sub-section requirements are also "must"s.	M
UC22	View dog's description		M
UC23	Update dog's description		M
UC24	View dog's collar		M
UC25	Update dog's collar		M
UC26	View breed details	Despite these requirements acting as extensions of the dog's details, they are only a "could" level requirement as they are not as important as being able to see a dog's collar colour as this is a detail that identifies them, details about breeds are heavily documented online already and providing an explanation here wouldn't be able to compete with the level of detail on the internet.	C
UC27	Update breed details		C
UC28	View dog family tree	Being able to view the family tree of a dog is one of the original requirements of the system as specified by the client. However, this have been prioritised as a "should" because the time element of implementing this feature may not be worth the resulting outcome.	S
UC29	Update family tree		S
UC30	View dog's owner	See the justification for UC15, view dog details.	M
UC31	Update dog's owner		M
UC32	Add new dog / puppy	The system has to be able to accept new entries and have the ability to remove entries, without these requirements the system would be static and wouldn't be able to remain relevant.	M
UC33	Remove dogs		M
UC34	Add puppy to litter	Being able to group dogs together into a litter is not a requirement of the client, however for the system to be able to function efficiently it would be beneficial to group new dogs together so that they can easily be compared and the family tree is easier to distinguish.	S
UC35	Remove puppy from litter		S
UC36	Add puppy visits to calendar	Being able to add calendar entries for puppy visits is a function that would allow Breeders to easily organise their business, however there are many applications that exist that allow users to manage their calendar entries. For this system to do this effectively it would have to integrate entries from other calendars, other	W

		calendars already do this and making the system do this is outside the scope of this project due to time constraints.	
UC37	Generate contracts	The client's user stories include the ability for the system to generate contracts from a template using the information in the system. This is a very time-consuming process and the time investment versus the benefits that the feature provides. Should the project have time then this requirement will be implemented, for this reason, this is a "could" requirement.	C
UC38	Update contract template		C
UC39	Locate breeders	For the system to satisfy the requirements for the Prospective Dog Owners, the system needs to be able show them Breeders. The reason that this requirement is only a "should" is because the Prospective Dog Owners are not the main stakeholders of the project and are less likely to use the system than Dog Breeders.	S
UC40	View update feed	See the justification for UC09 and UC10 as that justification applies to these requirements	C
UC41	Update interested feeds		C
UC42	Message users	The system's ability for users to message one another would allow Breeders and other users to communicate effectively and within the bounds of the system. However, the time constraints of the project would not allow for the implementation of a messaging system.	W
UC43	View guardian homes	Tracking the guardian homes in the system allows the Breeder to capitalise on their investments more effectively. This requirement is only a "should" because it is not as important as allowing the Breeder track the puppies that they currently have. It is still important for the system, this is why this requirement is not a "won't" or a "could" requirement.	S
UC44	Notify guardian home seasons		S

A.8 Risk Table

ID	Description	Probability	Impact	Priority	Mitigation	Contingency
R1	Project isn't completed within timeframe	M	H	H	Work on completing the MVP and the report before considering delivering supplementary features	Complete the report with priority over the project as this is the main deliverable. If applicable, apply for mitigating circumstances.
R2	Implementation overruns the allocated time	L	H	H	Ensure that milestones are reached on time or early by working diligently and prioritising work according to MVP	Focus on delivering the "Must" level requirements, work extra if required
R3	Client isn't kept up-to-date with the relevant information	M	L	L	Keep the client updated with anything that relates to completion of their requirements.	Prepare reports to catch up the client should they wish to know more than they have heard.
R4	Team is unable to work due to personal issues	L	H	L	Keep in good health and ensure that physical and mental health come before the completion of the project	Apply for mitigating circumstances
R5	Scope creep	M	M	M	Stick to the MoSCoW analysis for implementing features. Only once all the existing requirements are completed are new requirements allowed to be considered	When implementing features, refer to the MoScoW analysis to see if the feature is truly required.
R6	Database is compromised and personal data is leaked	L	H	H	Ensure that all personal data, ie. Location, name, contact details, is encrypted. All passwords should be salted before encrypted to ensure that they aren't susceptible to cryptanalysis	Have systems in place to detect when the database has been compromised and ensure that all users are advised to change their passwords, just in case, although information gained should be useless.
R7	PDF generation API runs out of allowed	L	M	M	Store generated PDFs so that they are only generated when a new	Upgrade to a paid version of the API should the system become too popular to sustain

	generations (free version allows 1000 generations per month)				version is needed, work to minimise the number of times that the API is called	with the free version of the API
R8	Server cannot handle the number of requests for the service	L	H	H	Work to reduce load on the server, caches and cookies could be used to make the client-side work harder to ease work on the server	Upgrade to a bulkier server should the popularity of the system outweigh the capability of the server
R9	Local Councils may want access to the data collected by the system	L	H	H	Draft a legal agreement for the use of the service that covers the weight of the service to distribute information to local councils if the users consent	Ensure that users of the system have the ability to "opt in" to sharing their information with local councils and that the use of their data is clearly explained to the user so that the system complies with GDPR
R10	The Kennel Club may want access to the data in the system	L	M	L	Implement a feature that allows the system to anonymise the data in the system so that only genetic data is available	Only share data with The Kennel Club if that data cannot be linked to any users of the system
R11	Lack of experience with technologies increases the time taken to implement the project	L	M	M	Plan out what is required for the implementation stage and employ self-learning on those technologies before attempted to deploy them	Employ self-learning by attempting to implement the inexperienced technologies, likely to lead to messier implementations
R12	Project doesn't adhere to the Gantt chart and its milestones	H	L	L	Ensure that the time required to complete each section is given in full	Complete the sections with the highest priority and then work down the priorities
R13	Users upload too many photos and take all the server space	H	H	H	The server has limited capacity, limited the amount of photos that users can upload to the server and work to compress the photos to maximise the quantity of uploads	Upgrade the server so that it can handle much more space being used
R14	Users forgets	H	L	L	Minimise the amount of	Implement a forgotten

	their log in credentials				times that a user has to authenticate into the system. Use cookies to hold log in sessions where possible	password system so that users can recover their lost credentials

Appendix B

B.1 Low Fidelity Wireframes

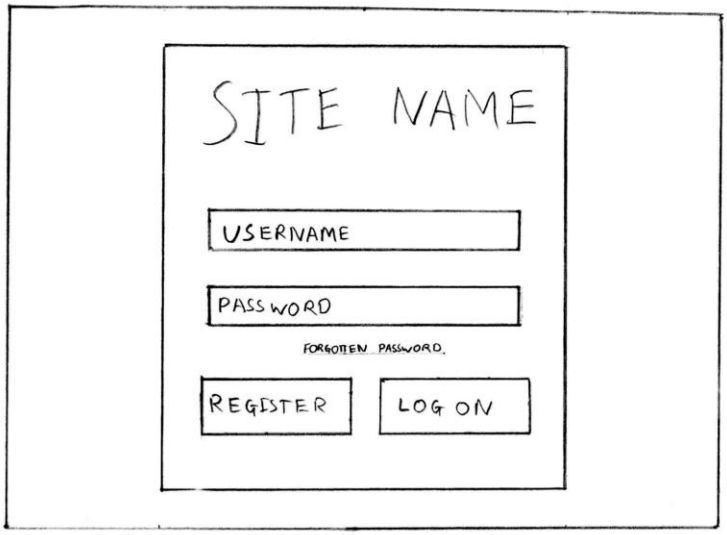


Figure B.1-1 - Login Page

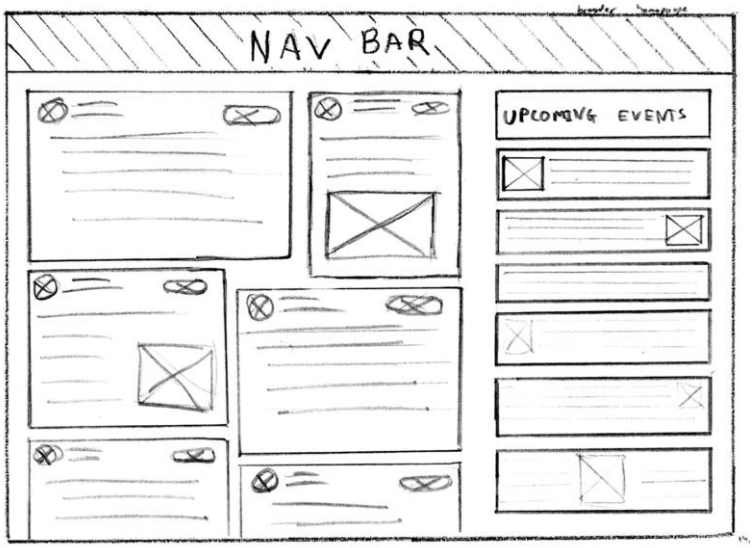


Figure B.1-2 - Breeder Homepage

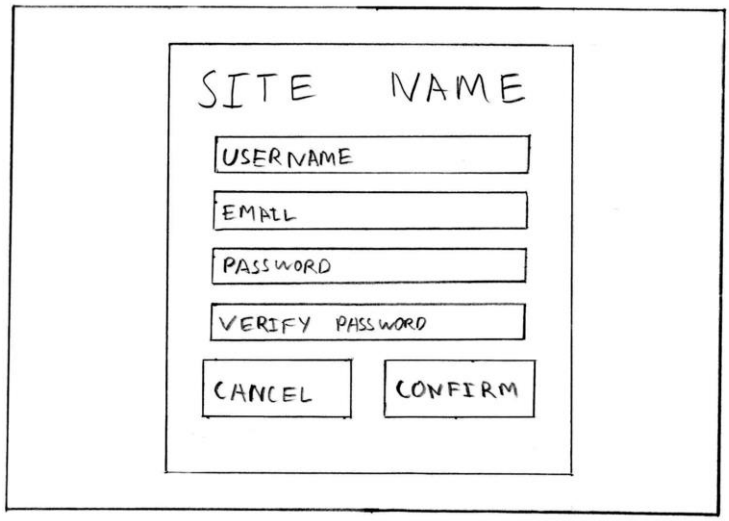


Figure B.1-3 - Register Account Page

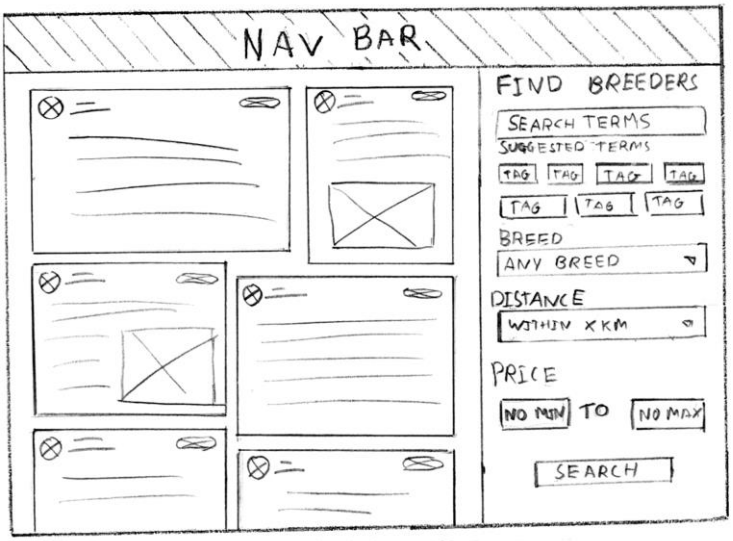


Figure B.1-4 - Prospective Owner Homepage

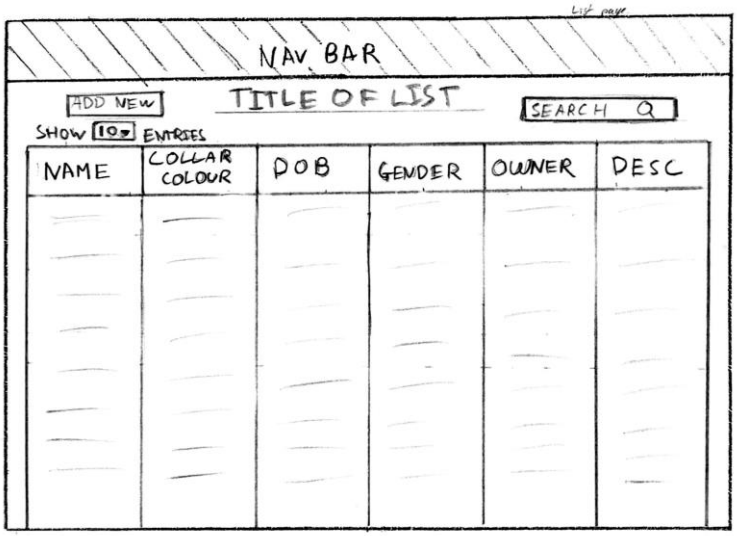


Figure B.1-5 - Dog List

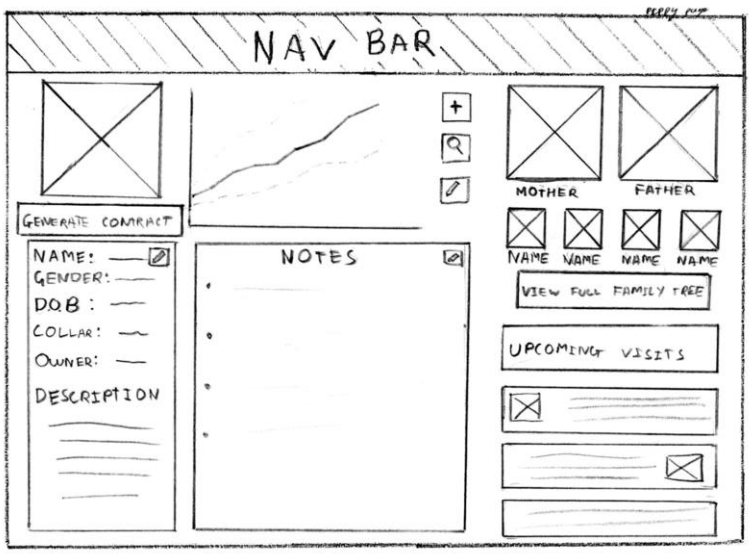


Figure B.1-6 – Puppy Details Page

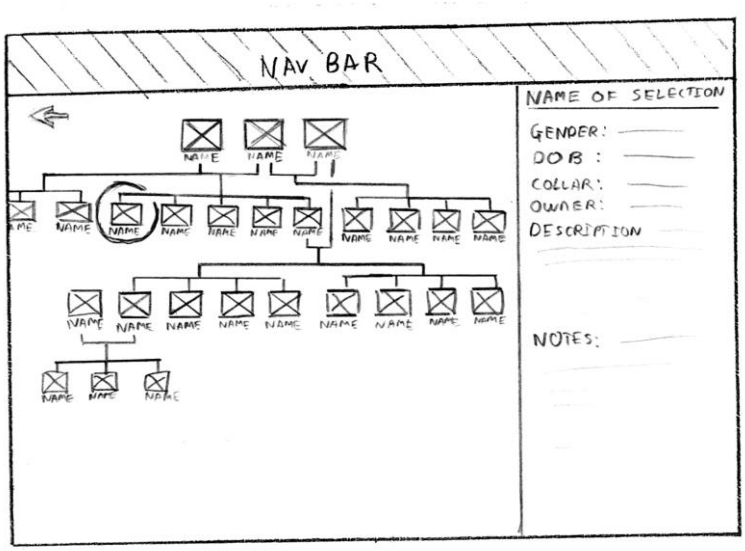


Figure B.1-7 - Family Tree Page

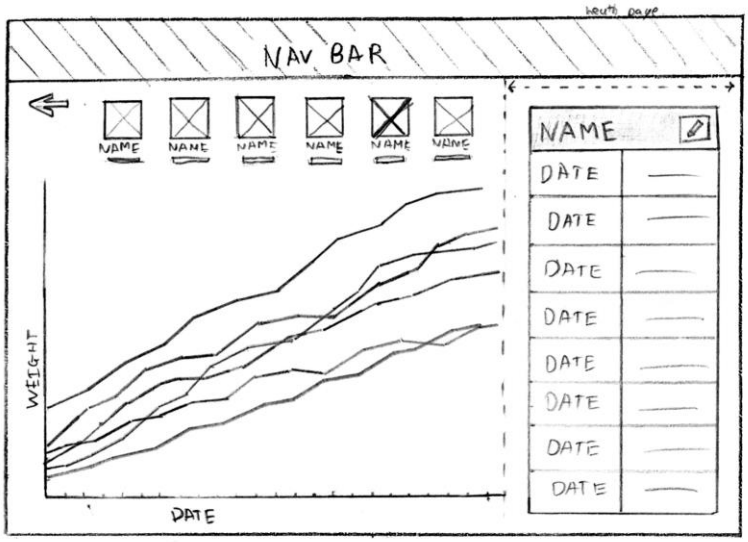


Figure B.1-8 - Dog Health Graph Page

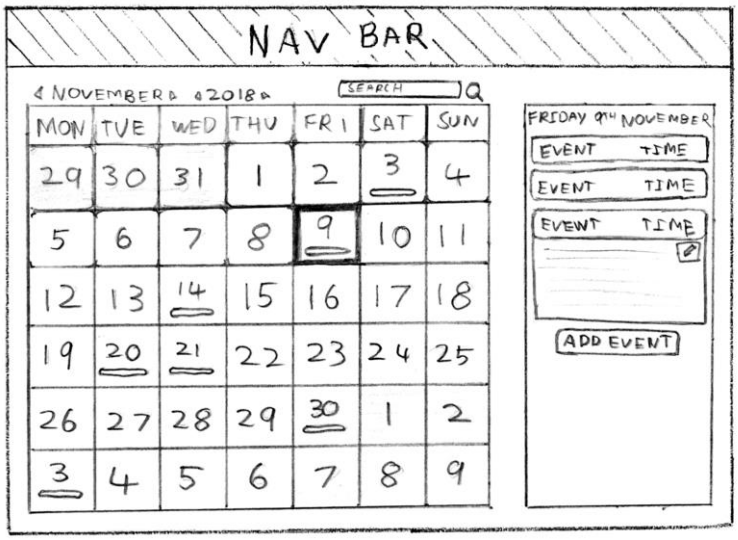


Figure B.1-9 - Calendar Page

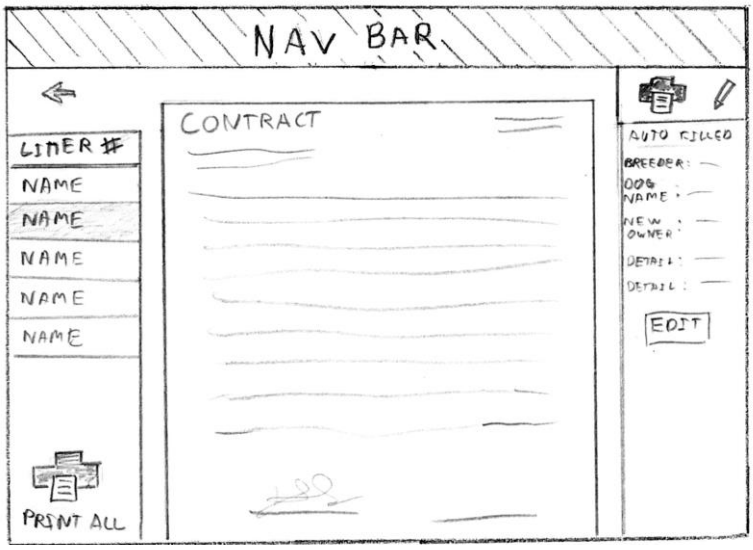
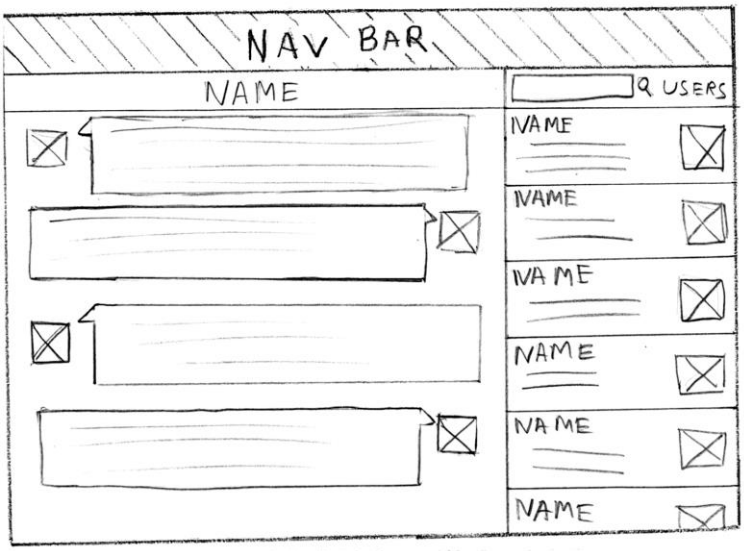


Figure B.1-10 - Contracts Page



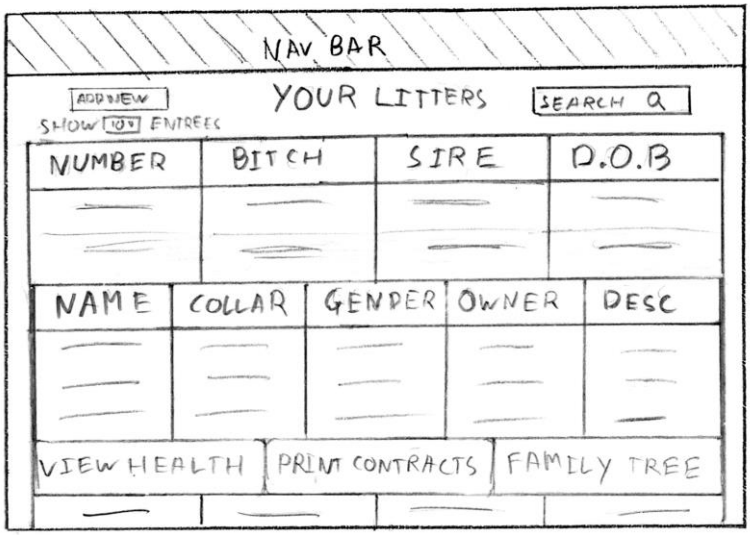


Figure B.1-11 - Messages Page

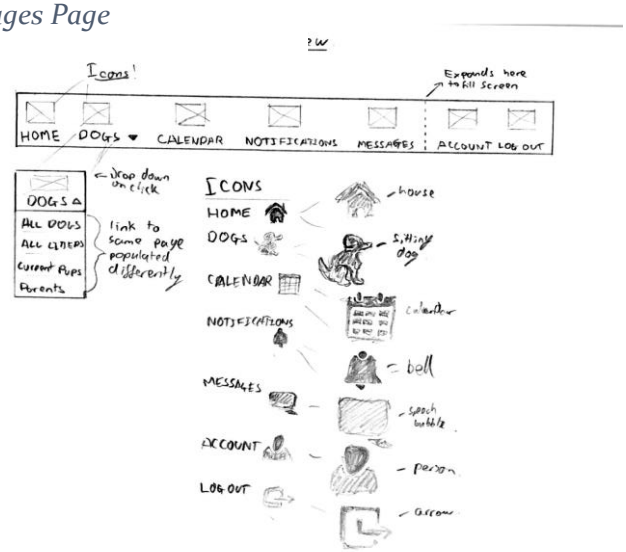


Figure B.1-12 - Litters List Page

Figure B.1-13 - Breeder NavBar

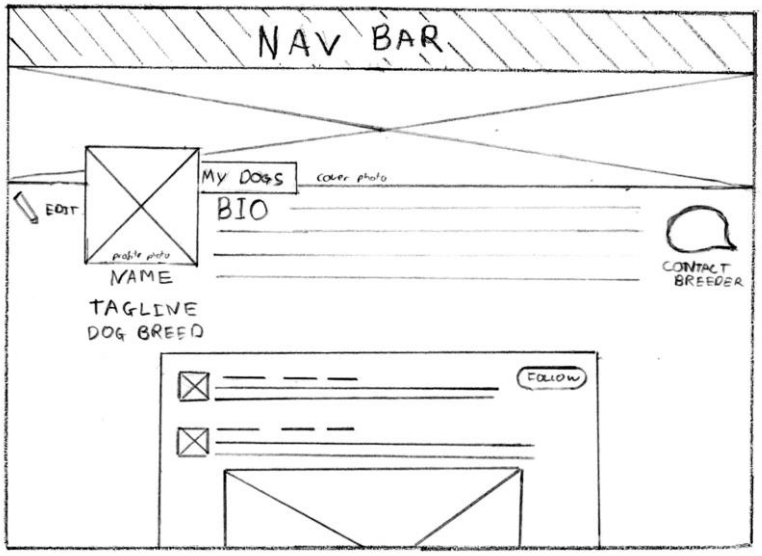


Figure B.1-14 - Breeders Account Page

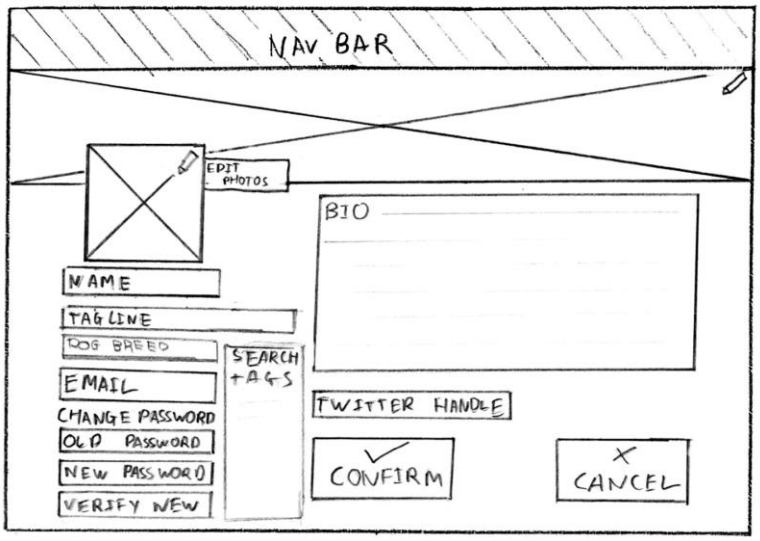


Figure B.1-15 - Editing Breeders Account Page

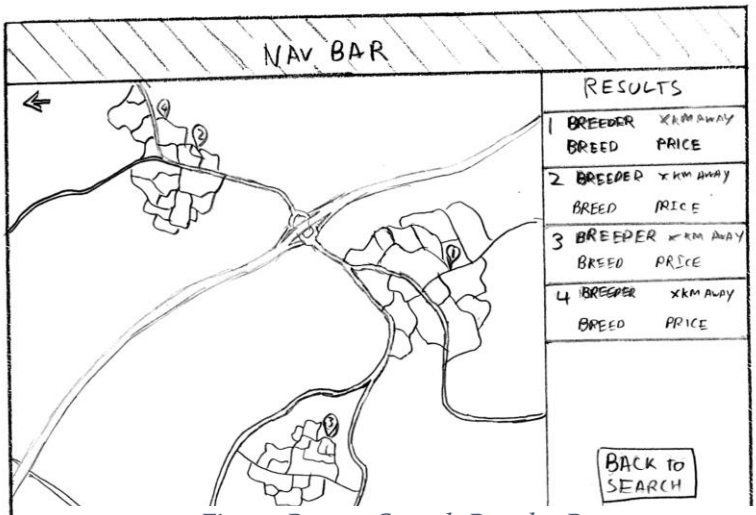


Figure B.1-17 - Search Results Page

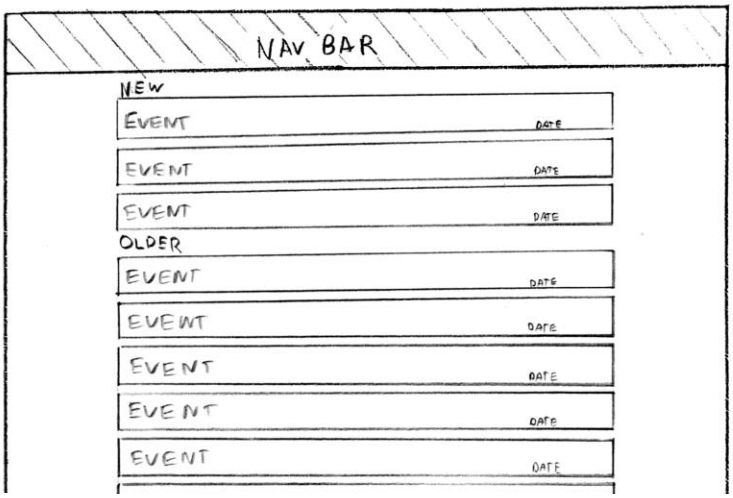


Figure B.1-16 - Notifications Page

B.2 Prototype

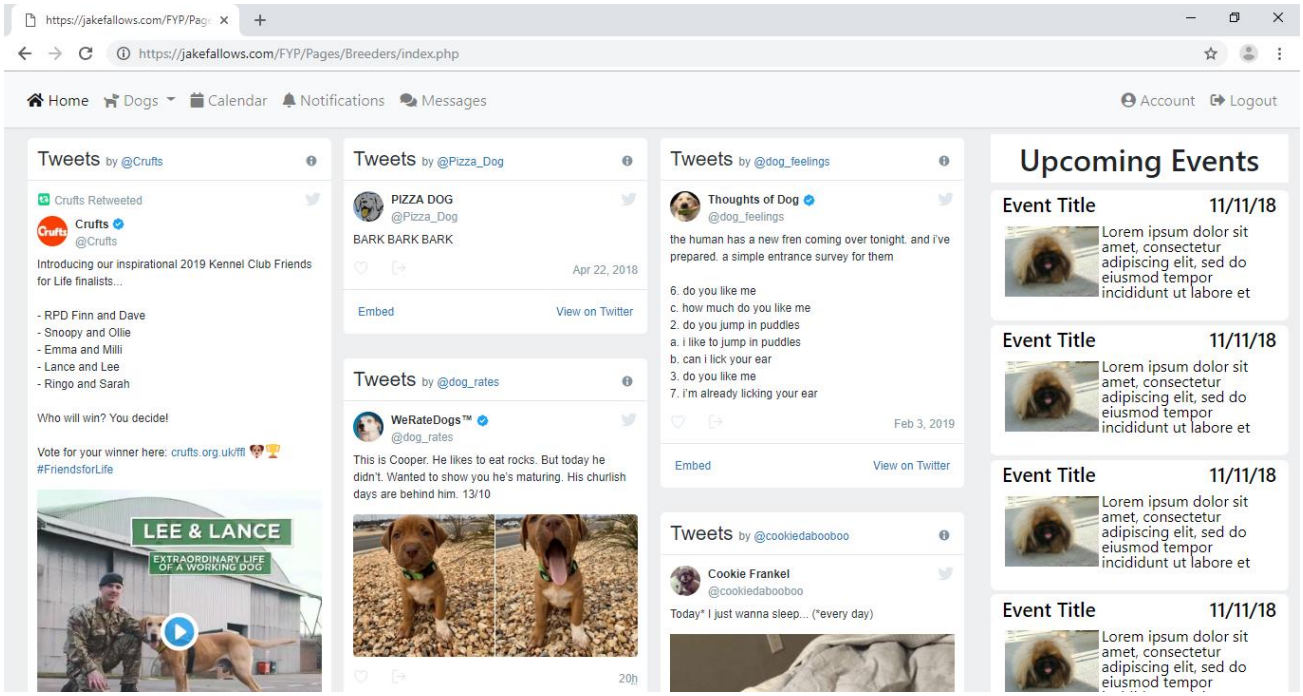
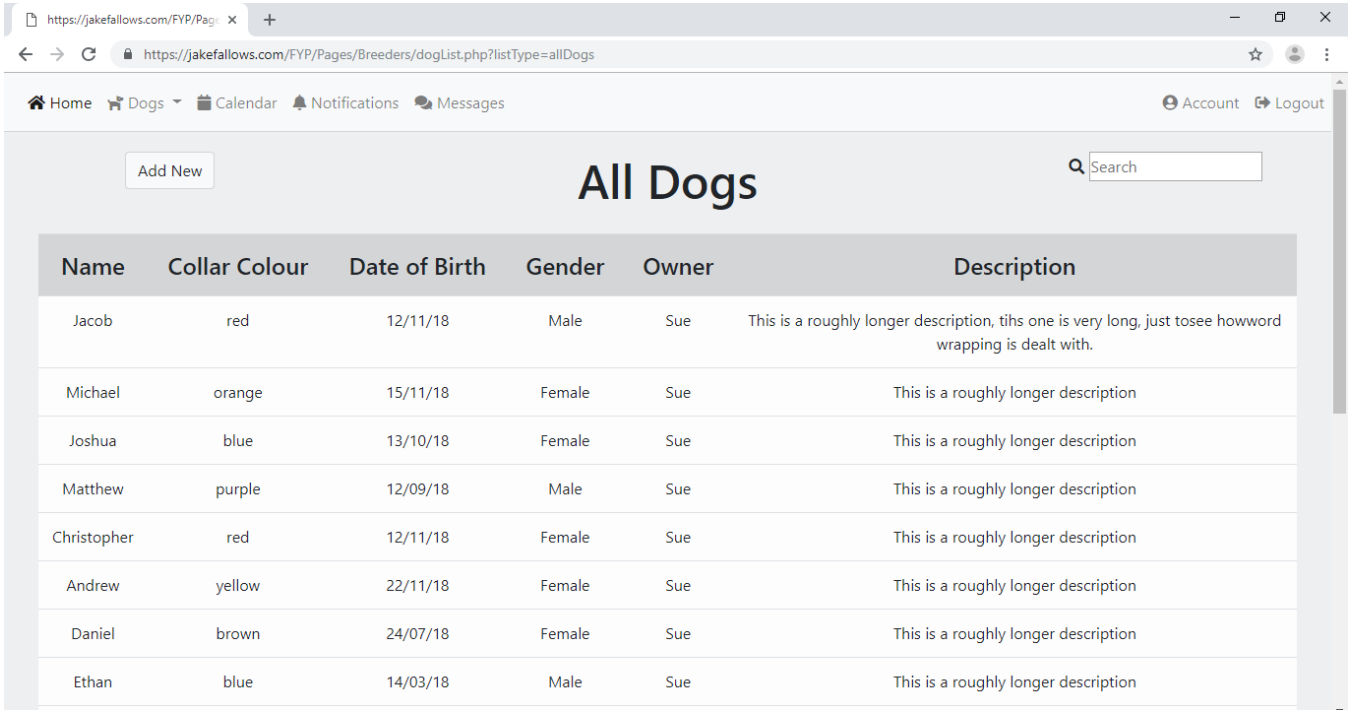


Figure B.2-1 - Breeder's Index Page



Name	Collar Colour	Date of Birth	Gender	Owner	Description
Jacob	red	12/11/18	Male	Sue	This is a roughly longer description, tihs one is very long, just tosee howword wrapping is dealt with.
Michael	orange	15/11/18	Female	Sue	This is a roughly longer description
Joshua	blue	13/10/18	Female	Sue	This is a roughly longer description
Matthew	purple	12/09/18	Male	Sue	This is a roughly longer description
Christopher	red	12/11/18	Female	Sue	This is a roughly longer description
Andrew	yellow	22/11/18	Female	Sue	This is a roughly longer description
Daniel	brown	24/07/18	Female	Sue	This is a roughly longer description
Ethan	blue	14/03/18	Male	Sue	This is a roughly longer description

Figure B.2-2 - Dog List Page

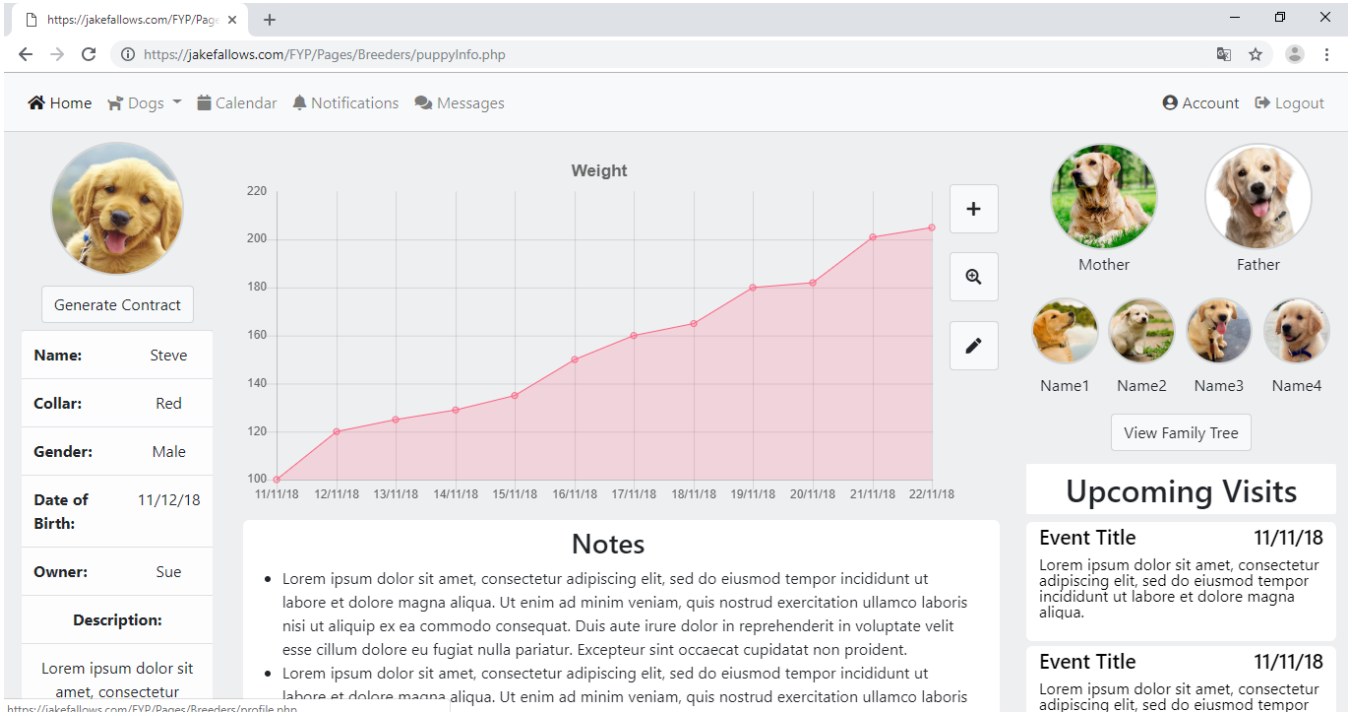


Figure B.2-3 – Puppy Details Page

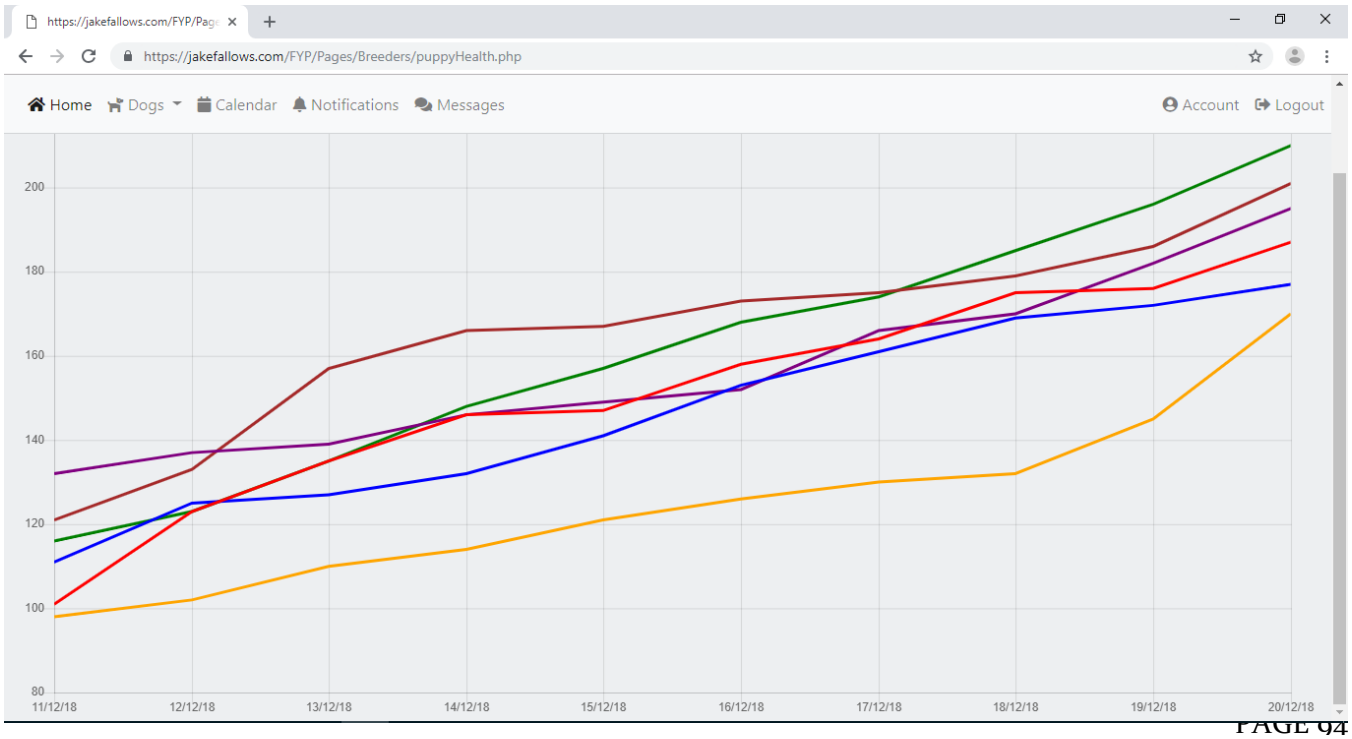


Figure B.2-4 - Puppy Health Page

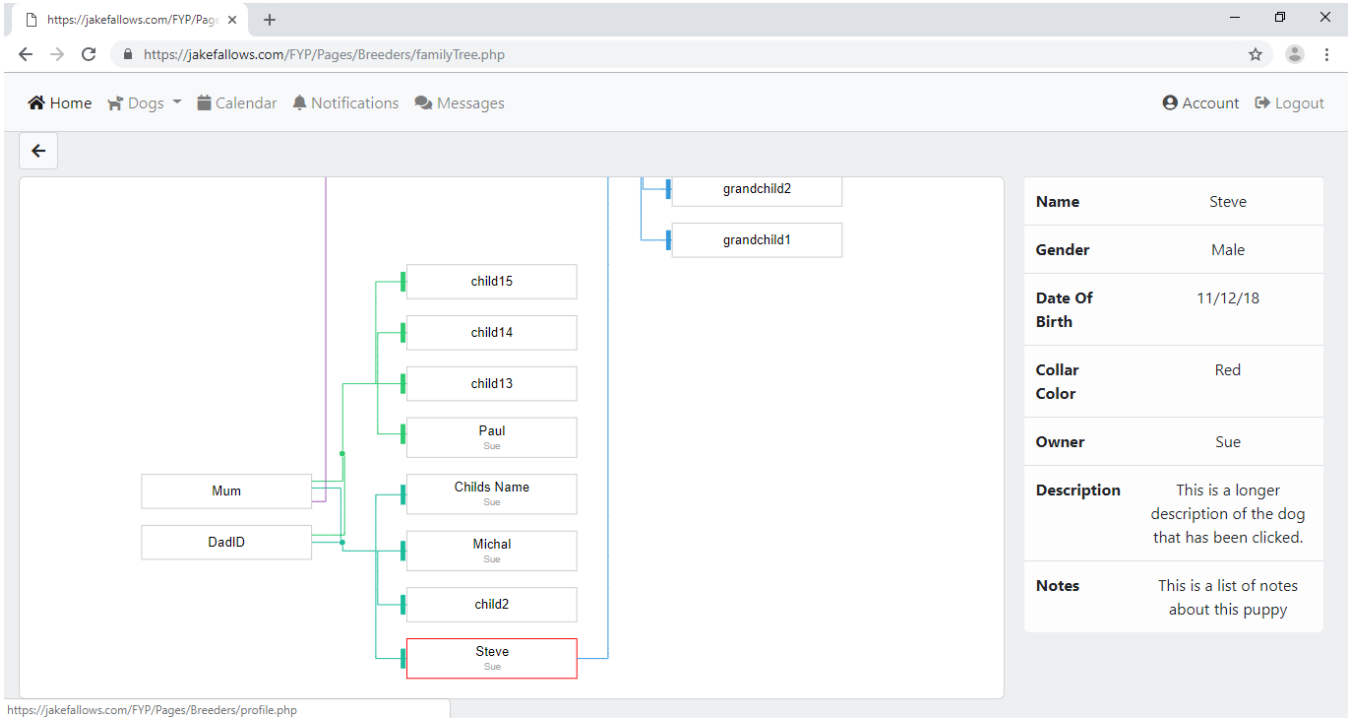


Figure B.2-5 - Family Tree Page

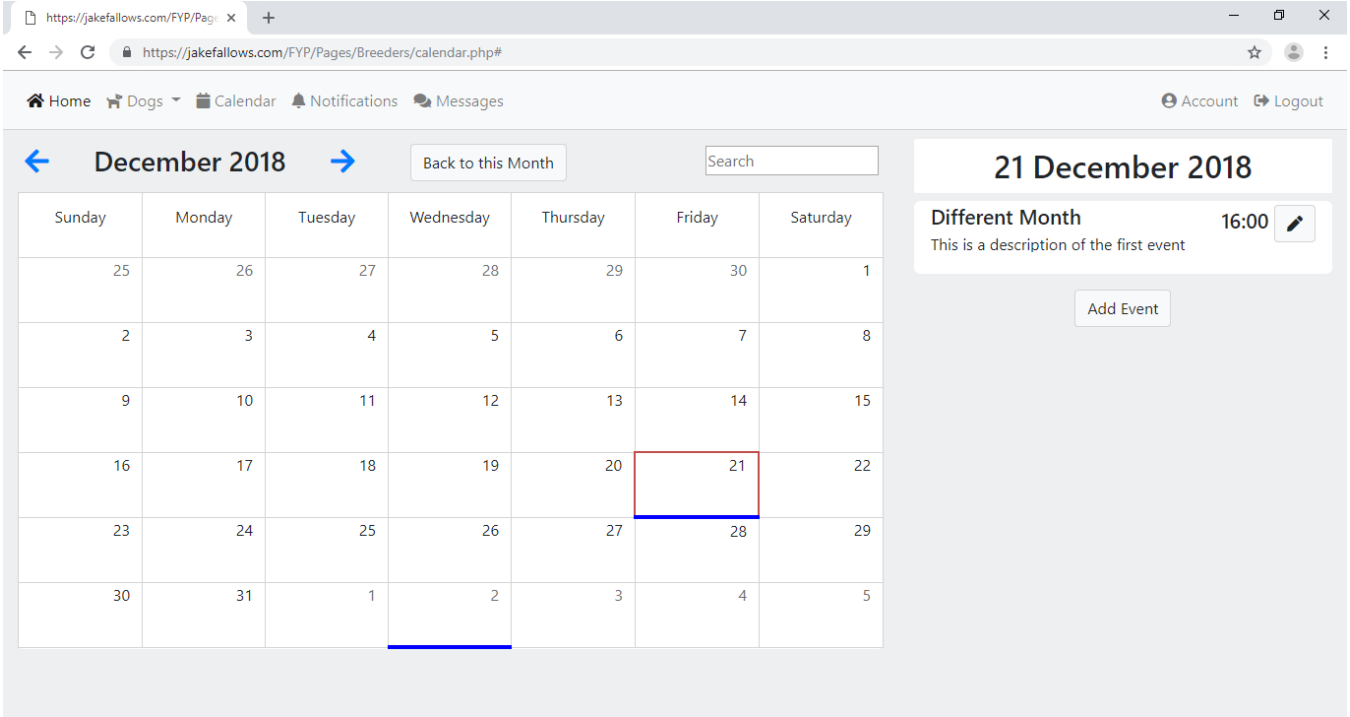


Figure B.2-6 - Calendar Page

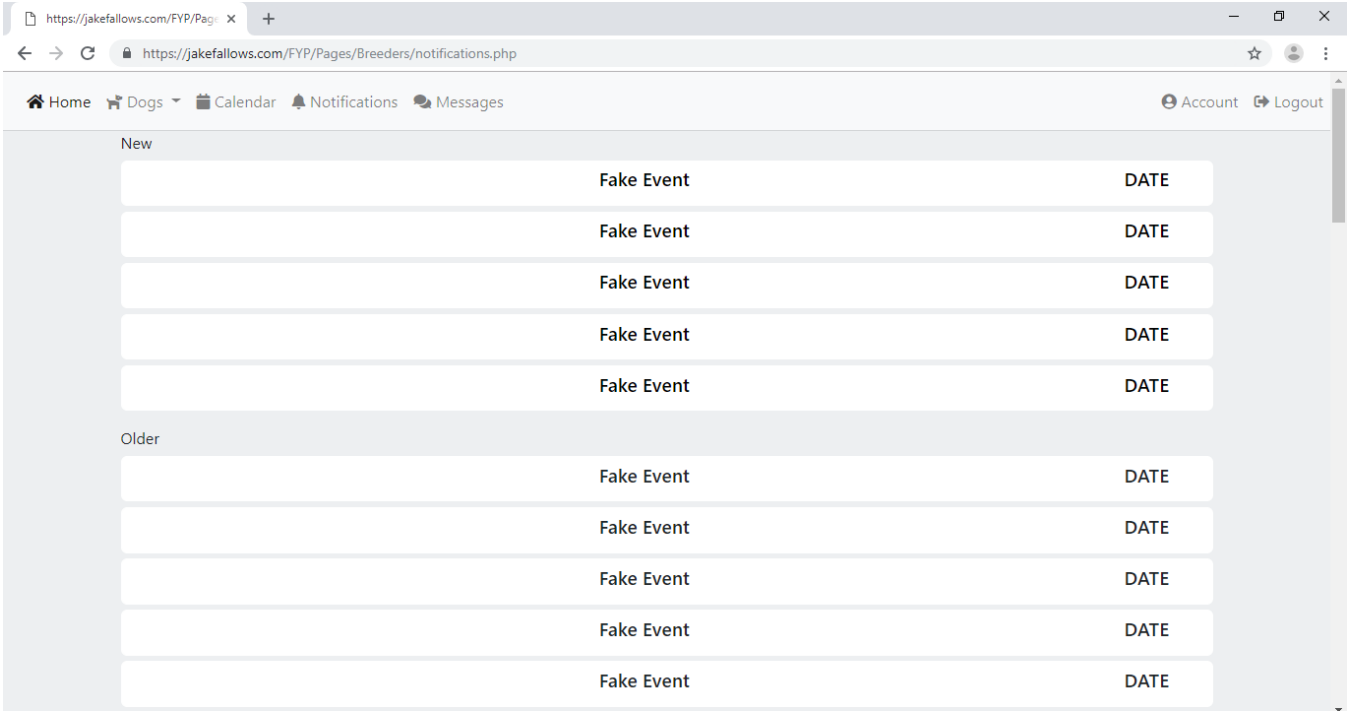


Figure B.2-7 - Notifications Page

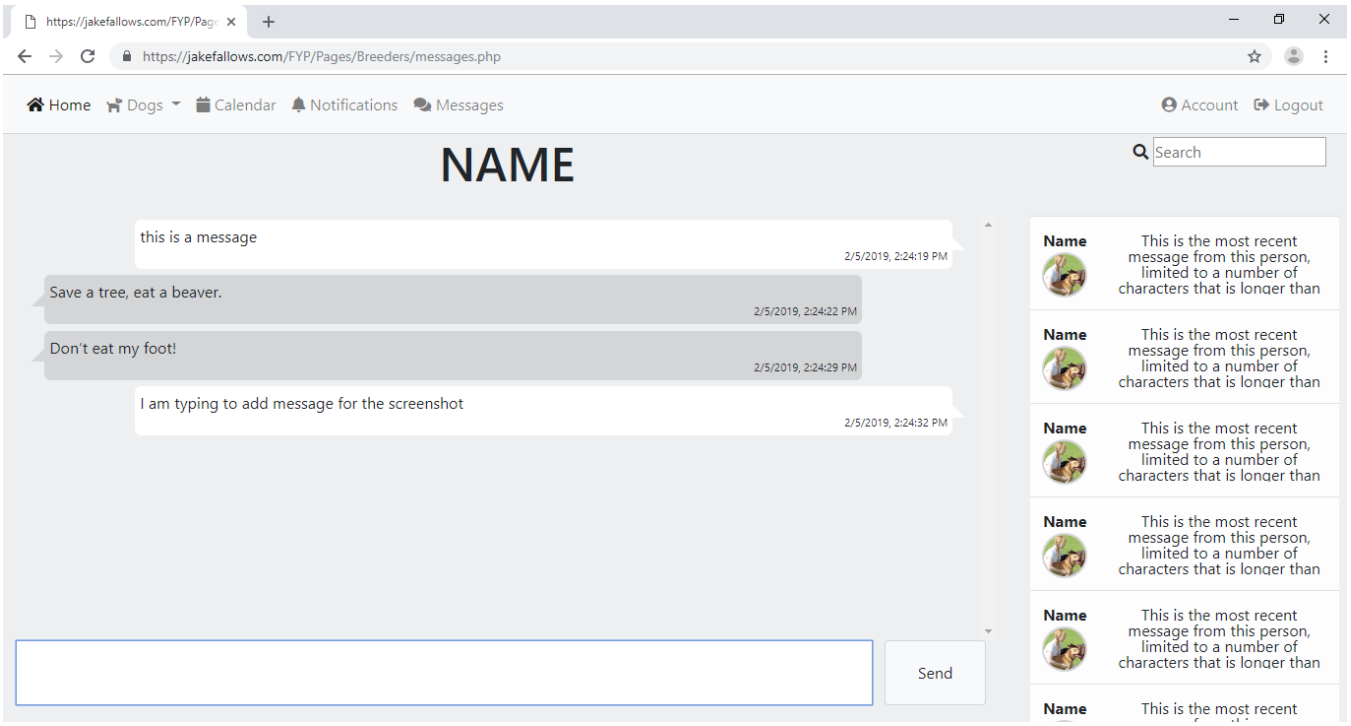


Figure B.2-8 - Messages Page

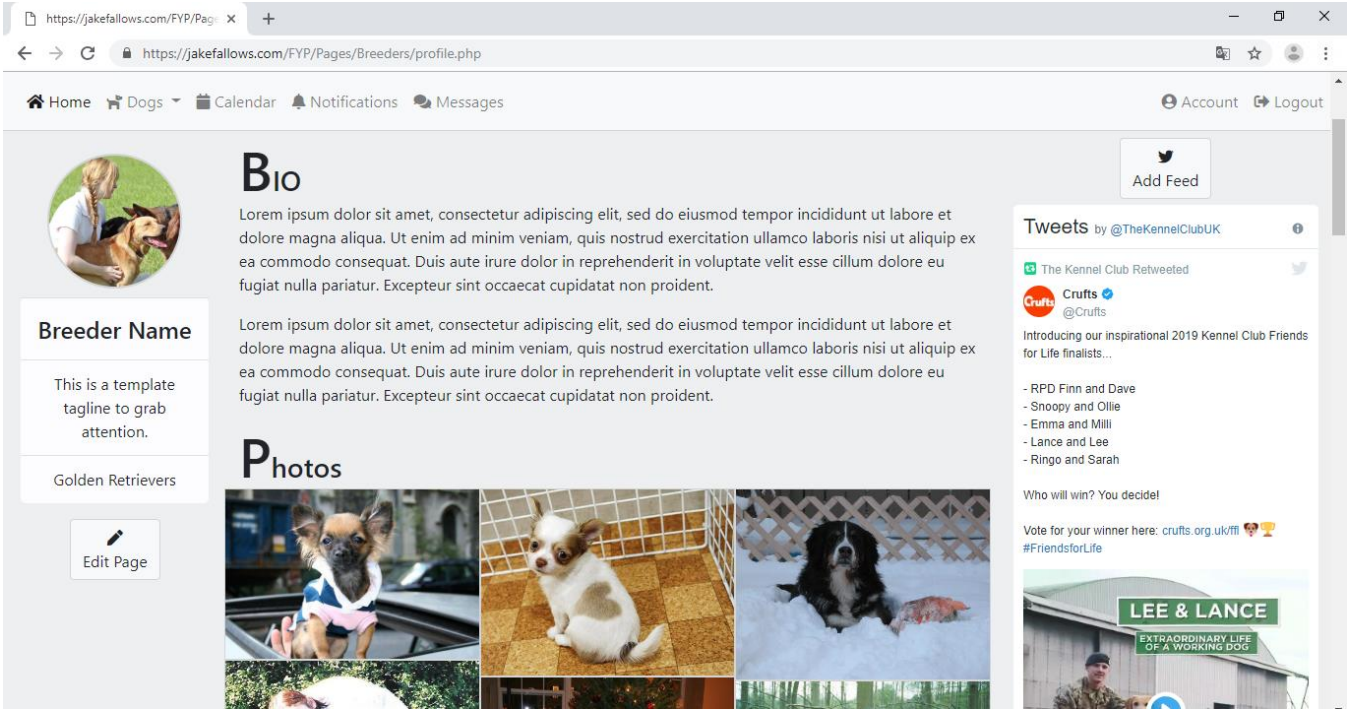


Figure B.2-10 - Breeder's Profile Page

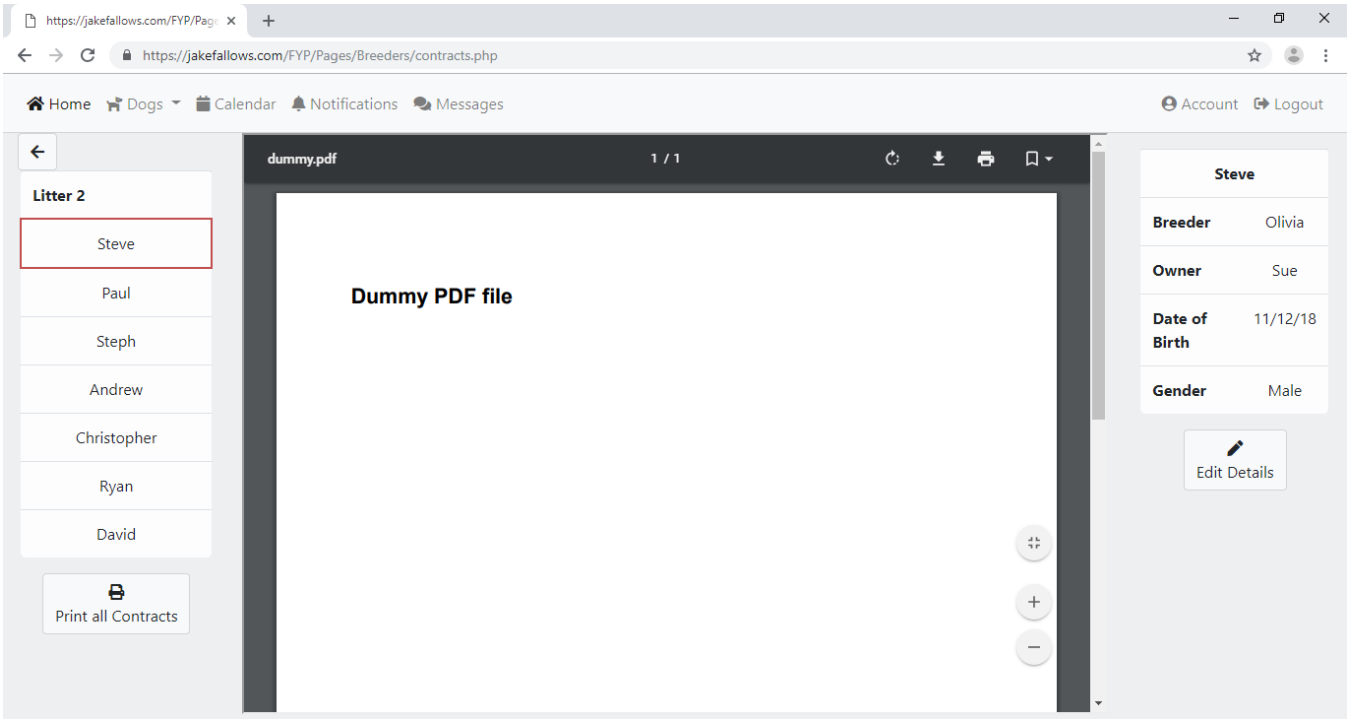


Figure B.2-9 - Contracts Page

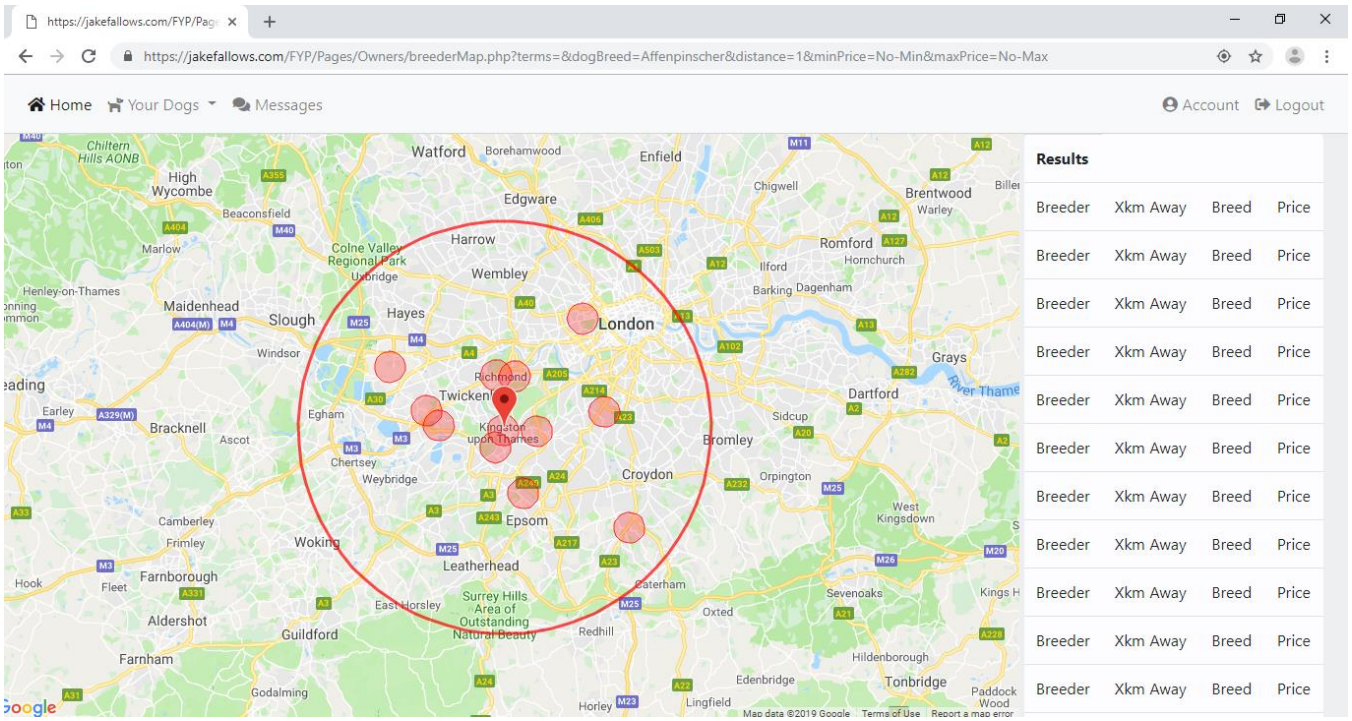


Figure B.2-12 -Search Results Page

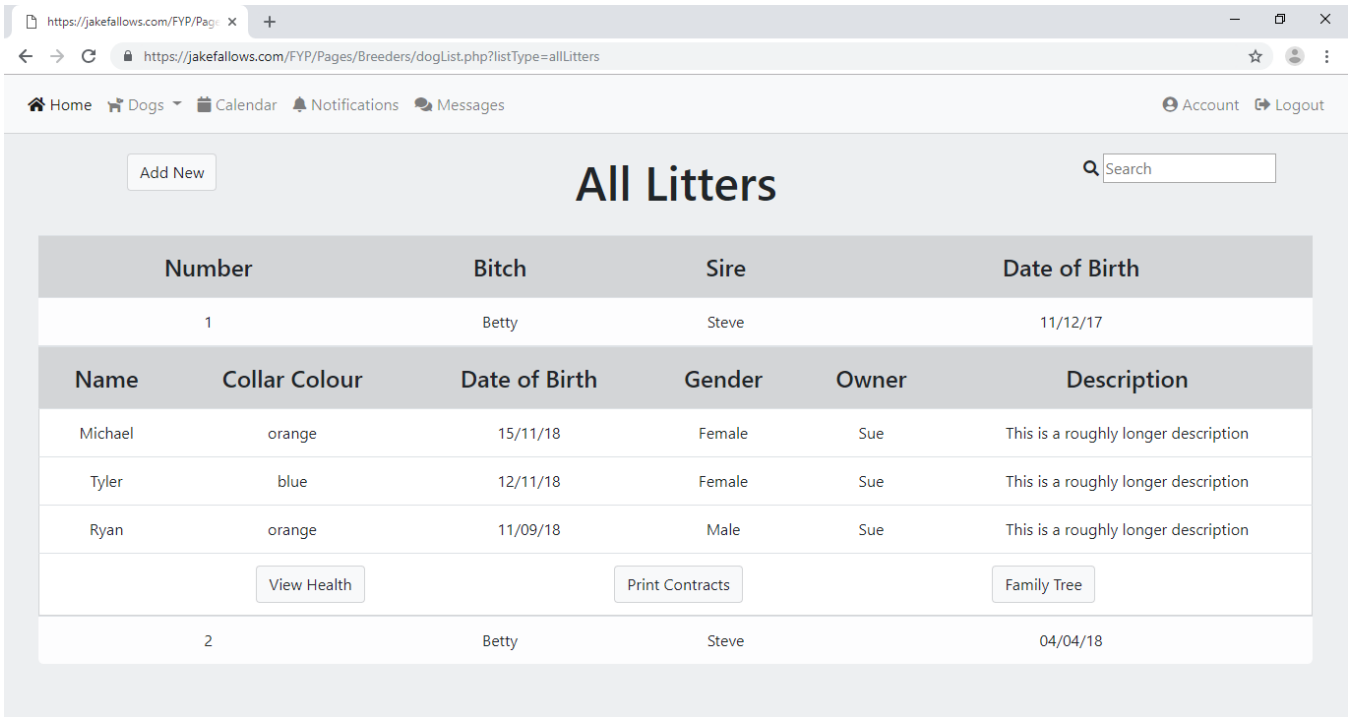


Figure B.2-11 - Dog Litters View

B.3 Class Diagrams

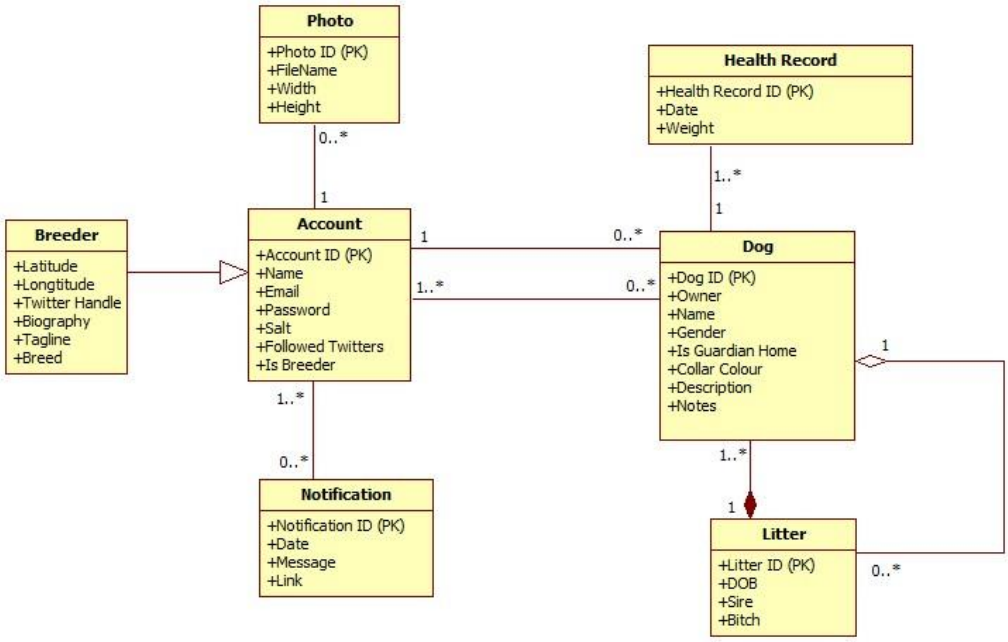


Figure B.3-1 - Version 1 of Class Diagram

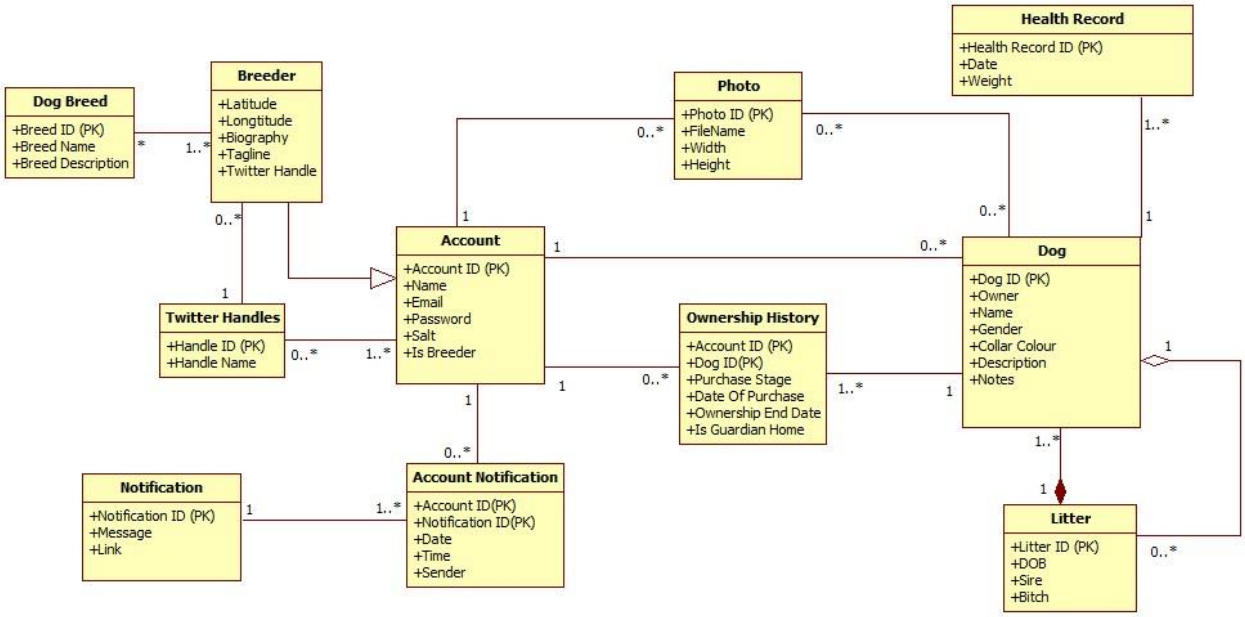


Figure B.3-2 - Version 2 of Class Diagram

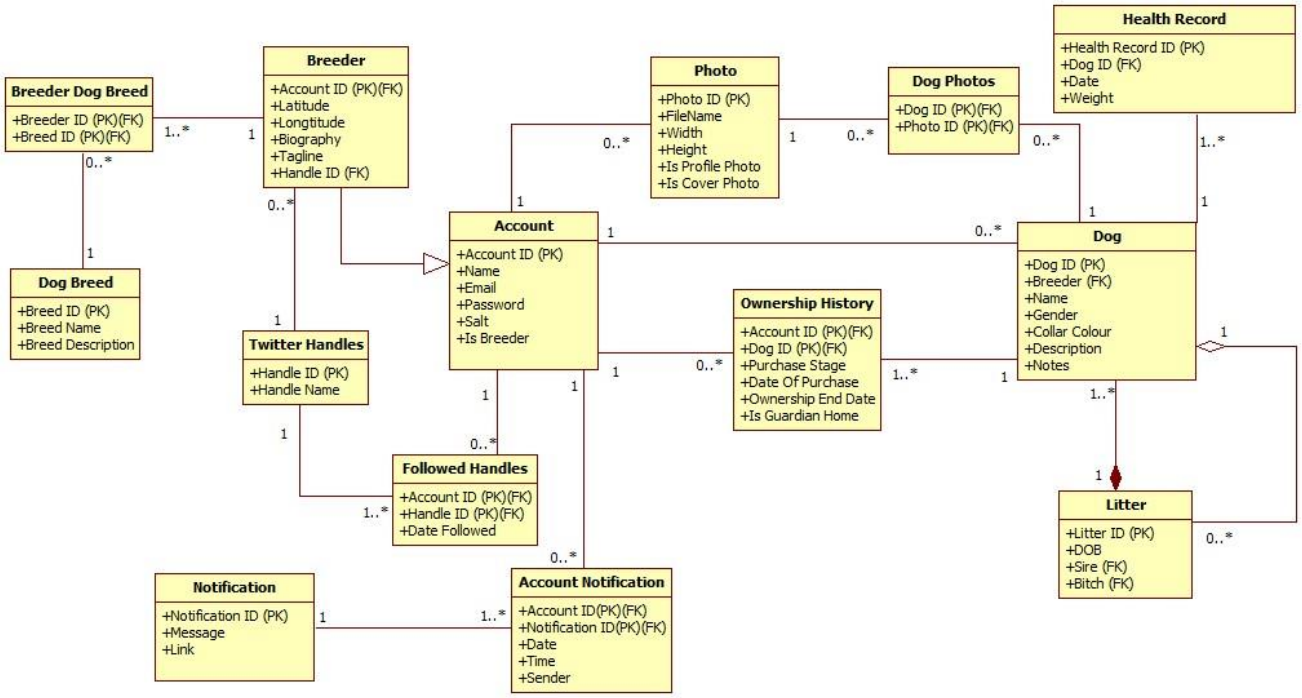


Figure B.3-3 - Version 3 of Class Diagram

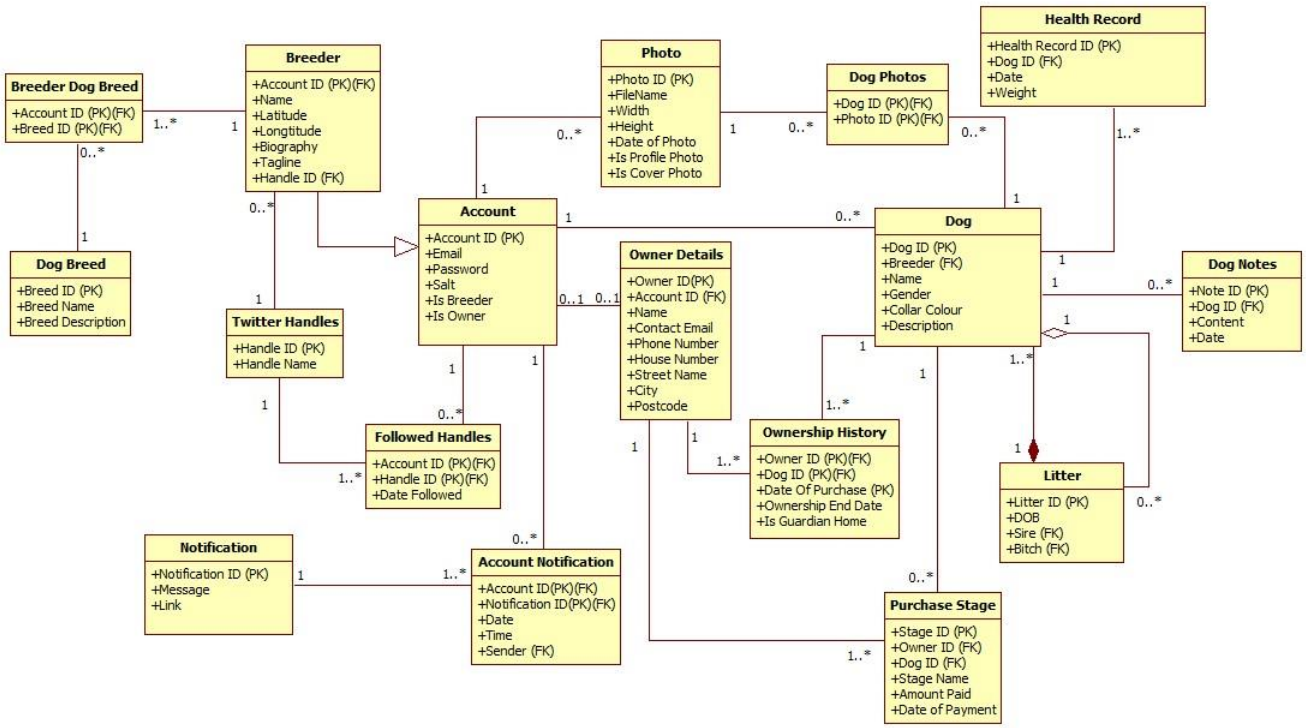


Figure B.3-4 - Version 4 of Class Diagram

B.4 Data Dictionary

Relation Name	Attribute Name	Data Type	Length	PK	FK	NOT NULL	Other Constraints
Dog_Breed	breed_id	INT	10	PK		NOT NULL	AUTO_INCREMENT UNIQUE
	breed_name	VARCHAR	20			NOT NULL	
	breed_desc	VARCHAR	100			NOT NULL	
Twitter_Handles	handle_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	handle_name	VARCHAR	50			NOT NULL	
Account	account_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	email	VARCHAR	50			NOT NULL	
	password	VARCHAR	50			NOT NULL	
	salt	INT	5			NOT NULL	
	is_breeder	TINYINT	1			NOT NULL	
	is_owner	TINYINT	1			NOT NULL	
Notification	notification_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	message	VARCHAR	150			NOT NULL	
	link	VARCHAR	50			NOT NULL	
Photo	photo_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	file_name	VARCHAR	100			NOT NULL	
	width	INT	5				

	height	INT	5				
	date_of_photo	DATE				NOT NULL	
	is_cover_photo	TINYINT	1			NOT NULL	
	is_profile_photo	TINYINT	1			NOT NULL	
Breeder	account_id	INT	10	PK	FK	NOT NULL	
	name	VARCHAR	20			NOT NULL	
	latitude	INT	6			NOT NULL	
	longitude	INT	6			NOT NULL	
	biography	VARCHAR	200				
	tagline	VARCHAR	50				
	handle_id	INT	10		FK	NOT NULL	
Breeder_Dog_Breed	account_id	INT	10	PK	FK	NOT NULL	
	breed_id	INT	10	PK	FK	NOT NULL	
Followed_Handles	account_id	INT	10	PK	FK	NOT NULL	
	handle_id	INT	10	PK	FK	NOT NULL	
	date_followed	DATE				NOT NULL	
Account_Notification	account_id	INT	10	PK	FK	NOT NULL	
	notification_id	INT	10	Pk	FK	NOT NULL	
	date	DATE				NOT NULL	

	time	TIME	6			NOT NULL	
	sender	INT	10		FK		
Owner_Details	owner_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	account_id	INT	10		FK	NOT NULL	
	name	VARCHAR	50			NOT NULL	
	contact_email	VARCHAR	50			NOT NULL	
	phone_number	INT	11			NOT NULL	
	house_number	INT	5			NOT NULL	
	street_name	VARCHAR	50			NOT NULL	
	city	VARCHAR	50			NOT NULL	
	postcode	VARCHAR	8			NOT NULL	
Dog	dog_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	breeder	INT	10		FK	NOT NULL	
	name	VARCHAR	50			NOT NULL	
	gender	CHAR	1			NOT NULL	
	collar_colour	VARCHAR	10			NOT NULL	
	description	VARCHAR	400			NOT NULL	
	litter_id	INT	10		FK		
Litter	litter_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT

	DOB	DATE				NOT NULL	
	sire	INT	10		FK	NOT NULL	
	bitch	INT	10		FK	NOT NULL	
Ownership_History	owner_id	INT	10	PK	FK	NOT NULL	
	dog_id	INT	10	PK	FK	NOT NULL	
	start_of_ownership	DATE					
	end_of_ownership	DATE					
	is_guardian_home	TINYINT	1			NOT NULL	
	purchase_stage	VARCHAR	50			NOT NULL	
	amount_paid	INT	7			NOT NULL	
	date_of_payment	DATE	10				
Dog_Notes	notes_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	dog_id	INT	10		FK	NOT NULL	
	content	VARCHAR	200			NOT NULL	
	date	DATE				NOT NULL	
Health_Record	health_record_id	INT	10	PK		NOT NULL	UNIQUE AUTO_INCREMENT
	dog_id	INT	10		FK	NOT NULL	
	date	DATE				NOT NULL	
	weight	INT	30			NOT NULL	

Dog_Photos	dog_id	INT	10	PK	FK	NOT NULL	
	photo_id	INT	10	PK	FK	NOT NULL	

Appendix C

C.1 Table Creation SQL Statements

```
CREATE TABLE IF NOT EXISTS Dog_Breed(
	breed_id INT(10) NOT NULL AUTO_INCREMENT UNIQUE,
	breed_name VARCHAR(20) NOT NULL,
	breed_desc VARCHAR(100) NOT NULL,
	PRIMARY KEY (breed_id)
);

CREATE TABLE IF NOT EXISTS Twitter_Handles(
	handle_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	handle_name VARCHAR(50) NOT NULL,
	PRIMARY KEY (handle_id)
);

CREATE TABLE IF NOT EXISTS Account(
	account_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	email VARCHAR(50) NOT NULL,
	password VARCHAR(50) NOT NULL,
	salt INT(5) NOT NULL,
	is_breeder TINYINT(1) NOT NULL,
	is_owner TINYINT(1) NOT NULL,
	PRIMARY KEY (account_id)
);

CREATE TABLE IF NOT EXISTS Notification(
	notification_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	message VARCHAR(150) NOT NULL,
	link VARCHAR(50) NOT NULL,
	PRIMARY KEY (notification_id)
);

CREATE TABLE IF NOT EXISTS Photo(
	photo_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	account_id INT(10) NOT NULL,
	file_name VARCHAR(100) NOT NULL,
	width INT(5) ,
	height INT(5) ,
	date_of_photo DATE NOT NULL,
	is_cover_photo TINYINT(1) NOT NULL,
	is_profile_photo TINYINT(1) NOT NULL,
	PRIMARY KEY (photo_id),
	FOREIGN KEY (account_id) REFERENCES Account(account_id)
);
```

```
CREATE TABLE IF NOT EXISTS Breeder(
	account_id INT(10) NOT NULL,
	name VARCHAR(20) NOT NULL,
	latitude INT(6) NOT NULL,
	longitude INT(6) NOT NULL,
	biography VARCHAR(200) ,
	tagline VARCHAR(50) ,
	handle_id INT(10) NOT NULL,
	PRIMARY KEY (account_id),
	FOREIGN KEY (account_id) REFERENCES Account(account_id),
	FOREIGN KEY (handle_id) REFERENCES Twitter_Handles(handle_id)
);

CREATE TABLE IF NOT EXISTS Breeder_Dog_Breed(
	account_id INT(10) NOT NULL,
	breed_id INT(10) NOT NULL,
	PRIMARY KEY (account_id, breed_id),
	FOREIGN KEY (account_id) REFERENCES Breeder(account_id),
	FOREIGN KEY (breed_id) REFERENCES Dog_Breed(breed_id)
);

CREATE TABLE IF NOT EXISTS Followed_Handles(
	account_id INT(10) NOT NULL,
	handle_id INT(10) NOT NULL,
	date_followed DATE NOT NULL,
	PRIMARY KEY (account_id, handle_id),
	FOREIGN KEY (handle_id) REFERENCES Twitter_Handles(handle_id),
	FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

CREATE TABLE IF NOT EXISTS Account_Notification(
	account_id INT(10) NOT NULL,
	notification_id INT(10) NOT NULL,
	date DATE NOT NULL,
	time TIME(6) NOT NULL,
	sender INT(10) ,
	PRIMARY KEY (account_id, notification_id),
	FOREIGN KEY (account_id) REFERENCES Account(account_id),
	FOREIGN KEY (sender) REFERENCES Account(account_id),
	FOREIGN KEY (notification_id) REFERENCES Notification(notification_id)
);

CREATE TABLE IF NOT EXISTS Owner_Details(
	owner_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	account_id INT(10) NOT NULL,
	name VARCHAR(50) NOT NULL,
	contact_email VARCHAR(50) NOT NULL,
```

```
	phone_number INT(11) NOT NULL,
	house_number INT(5) NOT NULL,
	street_name VARCHAR(50) NOT NULL,
	city VARCHAR(50) NOT NULL,
	postcode VARCHAR(8) NOT NULL,
	PRIMARY KEY (owner_id),
	FOREIGN KEY (account_id) REFERENCES Account(account_id)
);

CREATE TABLE IF NOT EXISTS Dog(
	dog_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	breeder INT(10) NOT NULL,
	name VARCHAR(50) NOT NULL,
	gender CHAR(1) NOT NULL,
	collar_colour VARCHAR(10) NOT NULL,
	description VARCHAR(400) NOT NULL,
	litter_id INT(10) ,
	PRIMARY KEY (dog_id),
	FOREIGN KEY (breeder) REFERENCES Breeder(account_id)
	#FOREIGN KEY (litter_id) REFERENCES Litter(litter_id)
);

CREATE TABLE IF NOT EXISTS Litter(
	litter_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	DOB DATE NOT NULL,
	sire INT(10) NOT NULL,
	bitch INT(10) NOT NULL,
	PRIMARY KEY (litter_id),
	FOREIGN KEY (sire) REFERENCES Dog(dog_id),
	FOREIGN KEY (bitch) REFERENCES Dog(dog_id)
);

CREATE TABLE IF NOT EXISTS Ownership_History(
	owner_id INT(10) NOT NULL,
	dog_id INT(10) NOT NULL,
	start_of_ownership DATE ,
	end_of_ownership DATE ,
	is_guardian_home TINYINT(1) NOT NULL,
	purchase_stage VARCHAR(50) NOT NULL,
	amount_paid INT(7) NOT NULL,
	date_of_payment DATE ,
	PRIMARY KEY (owner_id, dog_id),
	FOREIGN KEY (owner_id) REFERENCES Owner_Details(owner_id),
	FOREIGN KEY (dog_id) REFERENCES Dog(dog_id)
);

CREATE TABLE IF NOT EXISTS Dog_Notes(
```

```
	notes_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	dog_id INT(10) NOT NULL,
	content VARCHAR(200) NOT NULL,
	date DATE NOT NULL,
	PRIMARY KEY (notes_id),
	FOREIGN KEY (dog_id) REFERENCES Dog(dog_id)
);

CREATE TABLE IF NOT EXISTS Health_Record(
	health_record_id INT(10) NOT NULL UNIQUE AUTO_INCREMENT,
	dog_id INT(10) NOT NULL,
	date DATE NOT NULL,
	weight INT(30) NOT NULL,
	PRIMARY KEY (health_record_id),
	FOREIGN KEY (dog_id) REFERENCES Dog(dog_id)
);

CREATE TABLE IF NOT EXISTS Dog_Photos(
	dog_id INT(10) NOT NULL,
	photo_id INT(10) NOT NULL,
	PRIMARY KEY (dog_id, photo_id),
	FOREIGN KEY (dog_id) REFERENCES Dog(dog_id),
	FOREIGN KEY (photo_id) REFERENCES Photo(photo_id)
);
```

C.2 Drop Tables Statements

```
ALTER TABLE Dog DROP FOREIGN KEY litter_id;
DROP TABLE Dog_Photos;
DROP TABLE Health_Record;
DROP TABLE Dog_Notes;
DROP TABLE Ownership_History;
DROP TABLE Litter;
DROP TABLE Dog;
DROP TABLE Owner_Details;
DROP TABLE Account_Notification;
DROP TABLE Followed_Handles;
DROP TABLE Breeder_Dog_Breed;
DROP TABLE Breeder;
DROP TABLE Photo;
DROP TABLE Notification;
DROP TABLE Account;
DROP TABLE Twitter_Handles;
DROP TABLE Dog_Breed;
```

C.3 Truncated Raw Example Data

Account					
account_id	usernam	email	password	is_breede	is_owner

	e			r	
1	Breeder	breeder@mail.com	$2y$10$..g3Qysbru/	1	0
2	Breeder2	breeder2@mail.com	$2y$10$..g3Qysbru/	1	0
3	Breeder3	breeder3@mail.com	$2y$10$Vt1lk6n6QlS	1	0

Breeder

account_id	name	latitude	longitude	postcode	biography	tagline	handle_id
1	Paul	51.93	-0.59	Rf344kn	Katheryn Elizabeth ...	Open Happiness	1
2	Mary	53.34	-2.17	hf3230n	Justin Drew Bieber ...	The happiest place on earth.	2
3	Simon	50.82	-3.28	fd223fs	Born in Aberdeen, ...	Avis – We try harder.	3

Owner_Details

owner_id	account_id	name	contact_email	phone_number	house_number	street_name	city	postcode	longitude	latitude
1	4	OLIVER	oliver@...	3375676614	87	Main Street	London.	BT66 6JD	-6.3369	54.4681
2	5	HARRY	harry@...	3178981112	96	Church Street	Birmingham	HD8 8JH	-1.7105	53.5967
3	6	GEORGE	george@...	1493946747	42	Main Street North	Liverpool	S70 9SB	-1.4829	53.5542

Followed_Handles

account_id	handle_id	date_followed
2	99	2018-10-21
5	3	2017-08-30
3	13	2018-05-19

Dog_Breed

breed_id	breed_name	breed_desc
1	Greyhound	null
2	Chihuahua	null
3	Boxer	null

Breeder_Dog_Breed

account_id	breed_id
1	1
1	4

C.4 Truncated Insert Statements

INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('1','katyperry');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('2','justinbieber');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('3','BarackObama');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('4','rihanna');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('5','taylorswift13');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('6','ladygaga');
INSERT INTO Twitter_Handles (handle_id, handle_name) VALUES ('7','TheEllenShow');

INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('1','Greyhound','null');
INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('2','Chihuahua','null');
INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('3','Boxer','null');
INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('4','Dachshund','null');
INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('5','Golden Retriever','null');
INSERT INTO Dog_Breed (breed_id,breed_name,breed_desc) VALUES ('6','Yorkshire Terrier','null');

INSERT INTO Account (account_id,username,email,password,is_breeder,is_owner) VALUES
('1','Breeder','breeder@mail.com','$2y$10$B0B2ewY2diSU89RAFaqNNe5iYzmeZeYcsN0Uj/gau/ZEq//MZtIGa','1','0');
INSERT INTO Account (account_id,username,email,password,is_breeder,is_owner) VALUES
('2','Breeder2','breeder2@mail.com','$2y$10$..g3Qysbru/bKbbb2K02f.Oap8EfQ1VO5gohq5UqFdhZvViVNX/T6','1','0');

```
INSERT INTO Account (account_id,username,email,password,is_breeder,is_owner) VALUES
('3','Breeder3','breeder3@mail.com','$2y$10$Vt1lk6n6QlSr7zI3DRSwieebBfsICsf/jw2.7padH6cSyyiOxOVAO','1','0');
INSERT INTO Account (account_id,username,email,password,is_breeder,is_owner) VALUES
('4','Owner','owner@mail.com','$2y$10$..g3Qysbru/bKbbb2K02f.Oap8EfQ1VO5g0hq5UqFdhZvViVNX/T6','0','0');
INSERT INTO Account (account_id,username,email,password,is_breeder,is_owner) VALUES
('5','Owner2','owner2@mail.com','$2y$10$Vt1lk6n6QlSr7zI3DRSwieebBfsICsf/jw2.7padH6cSyyiOxOVAO','0','1');

INSERT INTO Breeder (account_id,name,latitude,longitude,postcode,biography,tagline,handle_id) VALUES
('1','Paul','51.93','-0.59','Rf344kn','Katheryn Elizabeth','Open Happiness','1');
INSERT INTO Breeder (account_id,name,latitude,longitude,postcode,biography,tagline,handle_id) VALUES
('2','Mary','53.34','-2.17','hf3230n','Justin Drew Bieber ...','The happiest place on earth.','2');
INSERT INTO Breeder (account_id,name,latitude,longitude,postcode,biography,tagline,handle_id) VALUES
('3','Simon','50.82','-3.28','fd223fs','Born in Aberdeen, ... ','Avis – We try harder.','3');
INSERT INTO Breeder (account_id,name,latitude,longitude,postcode,biography,tagline,handle_id) VALUES
('7','Logan','53.43','-2.98','L20 2HW','Katheryn Elizabeth ...','Open Happiness','24');

INSERT INTO Breeder_Dog_Breed (account_id,breed_id) VALUES ('1','1');
INSERT INTO Breeder_Dog_Breed (account_id,breed_id) VALUES ('1','4');
INSERT INTO Breeder_Dog_Breed (account_id,breed_id) VALUES ('1','6');
INSERT INTO Breeder_Dog_Breed (account_id,breed_id) VALUES ('1','8');

INSERT INTO Followed_Handles (account_id,handle_id,date_followed) VALUES ('2','99','2018-10-21');
INSERT INTO Followed_Handles (account_id,handle_id,date_followed) VALUES ('5','3','2017-8-30');
INSERT INTO Followed_Handles (account_id,handle_id,date_followed) VALUES ('3','13','2018-5-19');
INSERT INTO Followed_Handles (account_id,handle_id,date_followed) VALUES ('3','64','2018-4-5');
```

Appendix D

D.1 Usability Testing Table of Results

ID	Requirement	Expected Outcome	Actual Outcome
Uc11	Add puppy health entry	With an existing puppy, a user is able to add a new health record for that puppy	As expected.
UC12	View puppy health	With an existing puppy, a user is able to view the health records for a puppy	As expected.
UC13	Update acceptable deviance	**Not Implemented**	
UC14	Notify health outside deviance	When a puppy's health range is unacceptably low, the user should receive a notification alerting them to this fact	When presented with an alert, the user was able to quickly find which puppy the alert related to.
UC15	View dog details	With an existing dog, a user should be able to find the page that shows the details for that dog	User was slower than expected to find the dropdown menu containing the dog that they were looking for. As expected once dog was found.
UC16	View dog's DOB	With an existing puppy, a user should be able to find the date of birth for that dog	As expected.
UC17	Update dog's DOB	With an existing puppy, a user should be able to alter the dogs date of birth	As expected.
UC18	View dog's gender	With an existing dog, a user should be able to find the dog's gender	As expected.
UC19	Update dog's gender	With an existing dog, a user should be able to alter the dog's gender	As expected.
UC20	View dog's name	With an existing dog, a user should be able to find the dog's name	As expected.
UC21	Update dog's name	With an existing dog, a user should be able to alter the dog's name	As expected.
UC22	View dog's description	With an existing dog, a user should be able to find the dog's description	As expected.
UC23	Update dog's description	With an existing dog, a user should be able to alter the dog's description	As expected.

UC24	View dog's collar	With an existing dog, a user should be able to find the dog's collar colour	As expected.
UC25	Update dog's collar	With an existing dog, a user should be able to alter the dog's collar colour	As expected.
UC26	View breed details	**Not Implemented**	
UC27	Update breed details		
UC28	View dog family tree	With at least one litter present in the system, a user should be able to view the family tree of a selected dog and the family tree of all dogs in the system	As expected.
UC29	Update family tree	With at least one litter present in the system, a user should be able to update the relationships in the tree	User struggled to update the family tree, was able to perform the expected function after slight instruction to change the litter contents rather than the family tree.
UC30	View dog's owner	With an existing dog, a user should be able to view the dog's current owner and details about the person and payment process	Information could be more obvious to the user as identification of the information was not instant.
UC31	Update dog's owner	With an existing dog, a user should be able to alter the dog's current owner, or alter the state of the payment process	As expected.
UC32	Add new dog / puppy	With a logged in account, a user should be able to insert a new parent or puppy into the system	As expected.
UC33	Remove dogs	With a logged in account, a user should be able to remove a dog or puppy from the system	User identified that there is no way to remove parents and puppies could only be removed using the edit litter interface.
UC34	Add puppy to litter	With an existing litter, a user should be able to add a new puppy to the existing litter	User took more time than expected to find the edit litter interface.
UC35	Remove puppy from litter	With an existing litter, a user should be able to remove an existing puppy from the litter	See remove dogs.

UC36	Add puppy visits to calendar	**Not Implemented**	
UC37	Generate contracts		
UC38	Update contract template		
UC39	Locate breeders	With a logged in owner account, a user should be able to perform a search to find breeders that fit their specifications	As expected.
UC40	View update feed	With a logged in account, a user should be able to see a series of updates from users that they have followed in the system	As expected.
UC41	Update interested feeds	With a logged in account, a user should be able to remove updates that they follow and add new ones	As expected.
UC42	Message users	**Not Implemented**	
UC43	View guardian homes		
UC44	Notify guardian home seasons	When a guardian home enters a season period, a user should receive a notification from the system alerting them to the seasonal dog	When presented with a notification a user was able to identify the dog that was in season.

D.2 Non-Functional Requirement Validation

Name	Status	Type	Initiator	Size	Time	Waterfall
familyTree.php	200	docu...	generating...	9.6 ...	10.23 s	

Figure D.1-1 – Load time of the longest page

dogList.php?listType=al...	200	docu...	Other	2.8 ...	98 ms
bootstrap.css	200	styles...	dogList.ph...	(fro...	15 ms
jquery-ui.min.css	200	styles...	dogList.ph...	(fro...	17 ms
jquery-3.3.1.min.js	200	script	dogList.ph...	(fro...	0 ms
style.css	200	styles...	dogList.ph...	(fro...	18 ms
jquery-ui.min.js	200	script	dogList.ph...	(fro...	0 ms
all.css	200	styles...	dogList.ph...	(fro...	20 ms
bootstrap.min.js	200	script	dogList.ph...	(fro...	0 ms
chart.bundle.min.js	200	script	dogList.ph...	(fro...	0 ms
jquery-svg-pan-zoom.js	200	script	dogList.ph...	(fro...	0 ms
v4-shims.css	200	styles...	dogList.ph...	(fro...	20 ms
name-that-color.js	200	script	dogList.ph...	(fro...	0 ms
logo.png	200	png	dogList.ph...	(fro...	0 ms
Alcubierre.woff	200	font	dogList.ph...	(fro...	0 ms
fa-solid-900.woff2	200	font	dogList.ph...	(fro...	0 ms

Figure D.2-2 – Load time of an average page

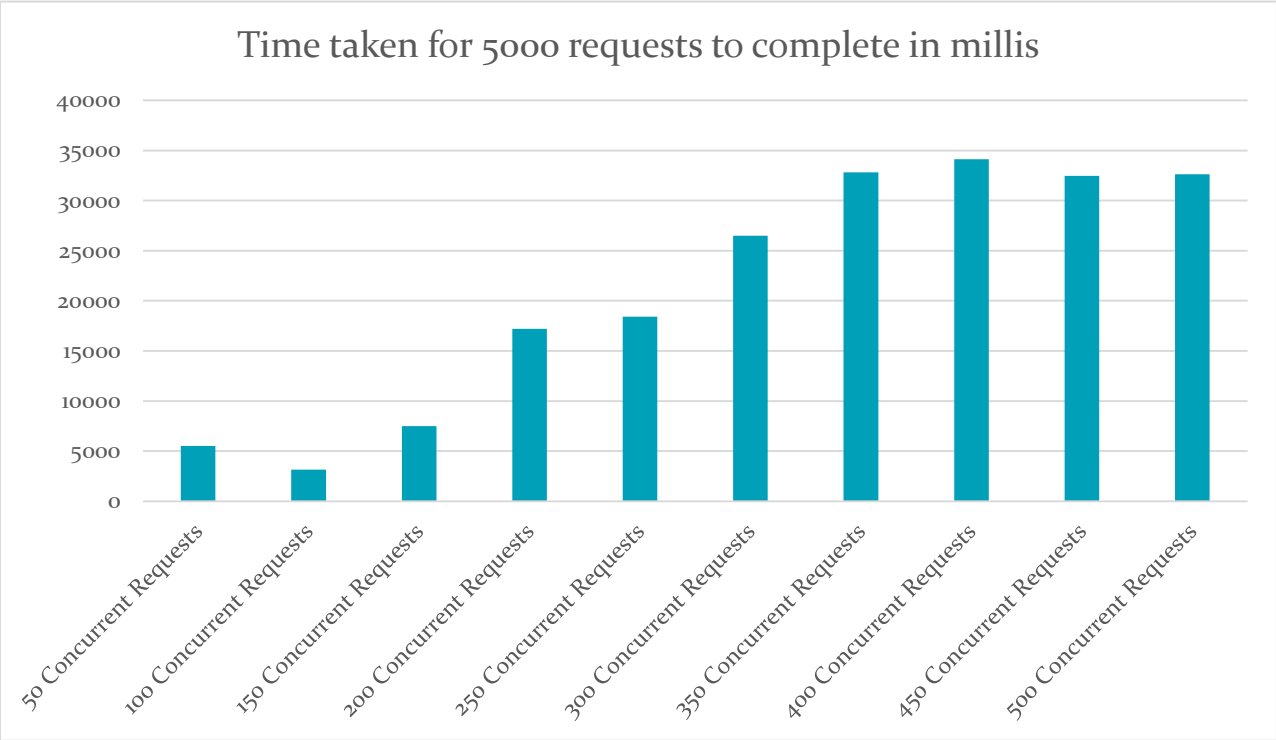
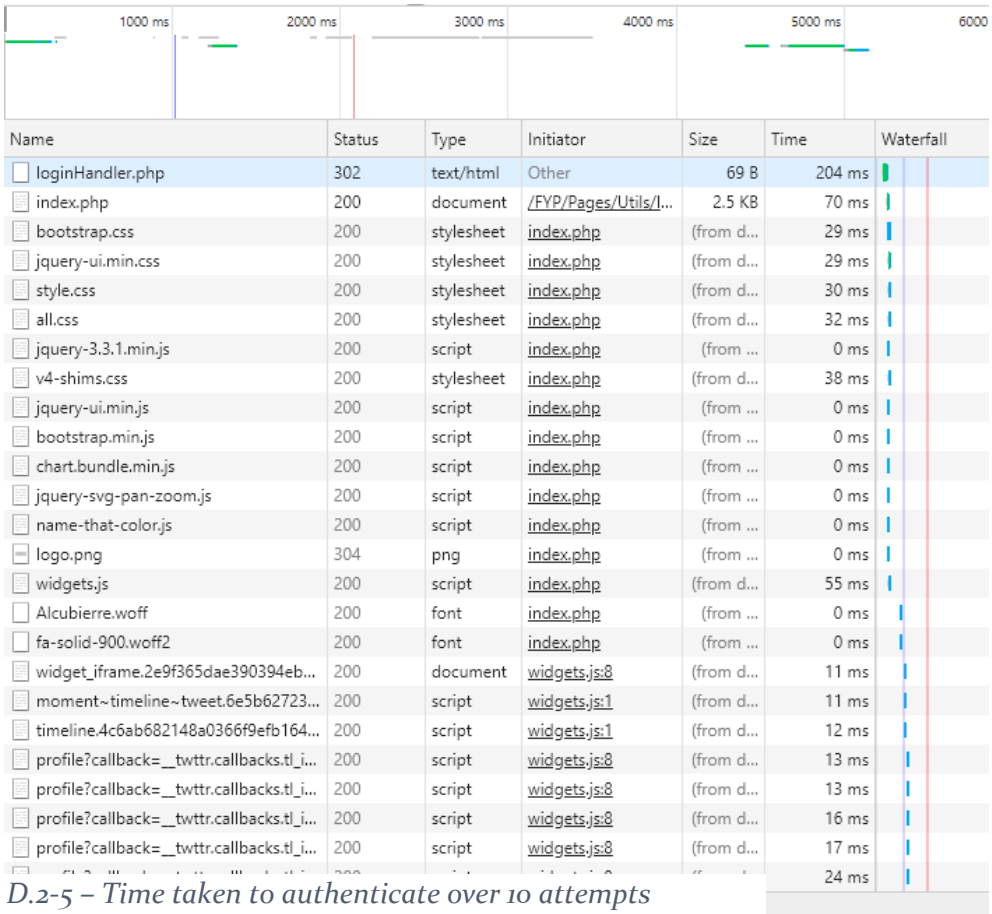


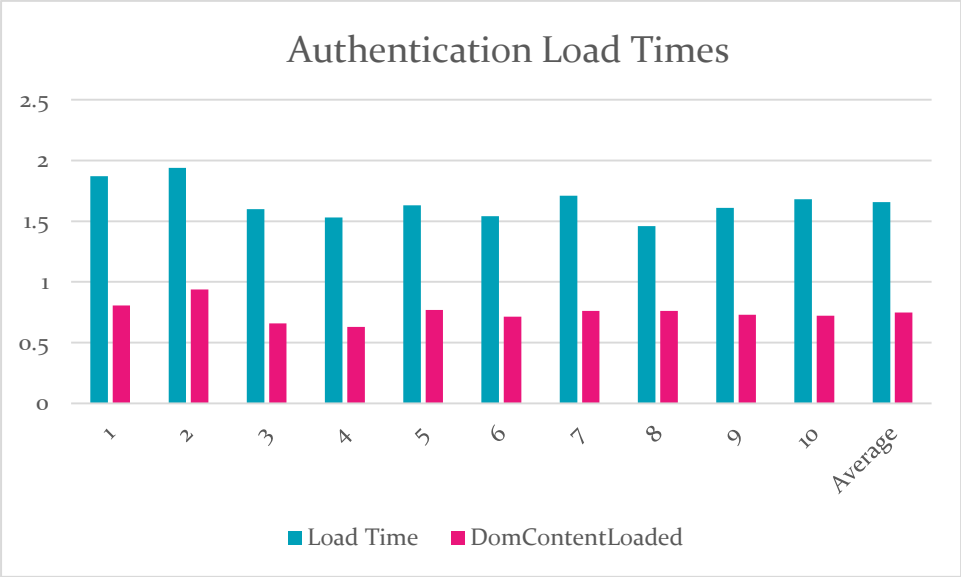
Figure D.2-3 – Apache benchmarking concurrency results



Name	Status	Type	Initiator	Size	Time	Waterfall
loginHandler.php	302	text/html	Other	69 B	204 ms	
index.php	200	document	/FYP/Pages/Utils/I...	2.5 KB	70 ms	
bootstrap.css	200	stylesheet	index.php	(from d...	29 ms	
jquery-ui.min.css	200	stylesheet	index.php	(from d...	29 ms	
style.css	200	stylesheet	index.php	(from d...	30 ms	
all.css	200	stylesheet	index.php	(from d...	32 ms	
jquery-3.3.1.min.js	200	script	index.php	(from ...	0 ms	
v4-shims.css	200	stylesheet	index.php	(from d...	38 ms	
jquery-ui.min.js	200	script	index.php	(from ...	0 ms	
bootstrap.min.js	200	script	index.php	(from ...	0 ms	
chart.bundle.min.js	200	script	index.php	(from ...	0 ms	
jquery-svg-pan-zoom.js	200	script	index.php	(from ...	0 ms	
name-that-color.js	200	script	index.php	(from ...	0 ms	
logo.png	304	png	index.php	(from ...	0 ms	
widgets.js	200	script	index.php	(from d...	55 ms	
Alcubierre.woff	200	font	index.php	(from ...	0 ms	
fa-solid-900.woff2	200	font	index.php	(from ...	0 ms	
widget_iframe.2e9f365dae390394eb...	200	document	widgets.js:8	(from d...	11 ms	
moment~timeline~tweet.6e5b62723...	200	script	widgets.js:1	(from d...	11 ms	
timeline.4c6ab682148a0366f9efb164...	200	script	widgets.js:1	(from d...	12 ms	
profile?callback=__twttr.callbacks.tl_i...	200	script	widgets.js:8	(from d...	13 ms	
profile?callback=__twttr.callbacks.tl_i...	200	script	widgets.js:8	(from d...	13 ms	
profile?callback=__twttr.callbacks.tl_i...	200	script	widgets.js:8	(from d...	16 ms	
profile?callback=__twttr.callbacks.tl_i...	200	script	widgets.js:8	(from d...	17 ms	
					24 ms	

Figure D.2-5 – Time taken to authenticate over 10 attempts

Figure D.2-6 – Measuring the load time of the index page



latitude	longitude
51.9300	-0.5900
53.3400	-2.1700
50.8200	-3.2800
53.4300	-2.9800
51.9100	-0.2000
53.0100	-1.1900
51.5900	-1.1200
56.0500	-3.4400
54.9000	-1.3600
54.6800	-1.2400

Figure D.2-7 – Location entries in the database

Appendix E

E.1 Function Requirements Completion

ID	Requirement	Status
UC01	Register new account	Completed
UC02	Login to existing account	Completed
UC03	Update account name	Completed
UC04	Update account email	Completed
UC05	Update password	Completed
UC06	Update tagline	Completed
UC07	Update bio	Completed
UC08	Update photos	Completed
UC09	Update Twitter link	Completed
UC10	Add feed to main page	Completed
Uc11	Add puppy health entry	Completed
UC12	View puppy health	Completed
UC13	Update acceptable deviance	Not Completed
UC14	Notify health outside deviance	Completed
UC15	View dog details	Completed
UC16	View dog's DOB	Completed
UC17	Update dog's DOB	Completed
UC18	View dog's gender	Completed
UC19	Update dog's gender	Completed
UC20	View dog's name	Completed
UC21	Update dog's name	Completed
UC22	View dog's description	Completed
UC23	Update dog's description	Completed
UC24	View dog's collar	Completed
UC25	Update dog's collar	Completed
UC26	View breed details	Not Completed
UC27	Update breed details	Not Completed
UC28	View dog family tree	Completed
UC29	Update family tree	Not Completed
UC30	View dog's owner	Partially Completed
UC31	Update dog's owner	Partially Completed
UC32	Add new dog / puppy	Completed
UC33	Remove dogs	Completed
UC34	Add puppy to litter	Partially Completed
UC35	Remove puppy from litter	Partially Completed
UC36	Add puppy visits to calendar	Not Completed
UC37	Generate contracts	Not Completed
UC38	Update contract template	Not Completed

UC39	Locate breeders	Completed
UC40	View update feed	Completed
UC41	Update interested feeds	Completed
UC42	Message users	Not Completed
UC43	View guardian homes	Not Completed
UC44	Notify guardian home seasons	Partially Completed

E.2 Client Feedback Form

FYP Client Feedback Form

What Improvements can be made?

I would like to see all of the guardian dogs in the main menu screen. It is difficult, at the moment, to see if a dog is a guardian or not.

I would like to be able to see a list of all the owners who have paid deposits or full amounts. I would like to create a waiting list for the owners who want a puppy but haven't got one yet.

The style could be a little cleaner and more feedback could be given on failed forms.

Figure E.2-2 – Client Feedback Form 2 (Client Suggestions)