**C868 – Software Capstone Project Summary**

**Task 2 – Section C**

|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | http://www.idevnews.com/views/images/uploads/general/wgu_logo.png  Christie’s Companions |
| **Student Name:** | Michael C Irick |

Contents

[Contents 2](#_Toc27506767)

[Task 2 Part C – C868 Software Development Capstone 4](#_Toc27506768)

[Application Design and Testing 4](#_Toc27506769)

[Design Document 4](#_Toc27506770)

[Class Design 4](#_Toc27506771)

[UI Design 5](#_Toc27506772)

[Unit Test Plan 8](#_Toc27506773)

[Introduction 8](#_Toc27506774)

[Purpose 8](#_Toc27506775)

[Overview 8](#_Toc27506776)

[Test Plan 8](#_Toc27506777)

[Items 8](#_Toc27506778)

[Features 9](#_Toc27506779)

[Deliverables 9](#_Toc27506780)

[Tasks 9](#_Toc27506781)

[Needs 10](#_Toc27506782)

[Pass/Fail Criteria 10](#_Toc27506783)

[Specifications 10](#_Toc27506784)

[Procedures 11](#_Toc27506785)

[Results 12](#_Toc27506786)

[C4. Source Code 12](#_Toc27506787)

[C5. Link to Live Version 12](#_Toc27506788)

[User Guide 12](#_Toc27506789)

Task 2 Part C – C868 Software Development Capstone

# Application Design and Testing

# Design Document

## Class Design

Below is the Entity-Relation Diagram that reflects the class design for the Christie’s Companions application. Due to the fairly standard implementation of MVC (Model/View/Controller) by the Ruby-on-Rails framework used in the project, only the classes involved in the core function of the application as a database will be shown and discussed.

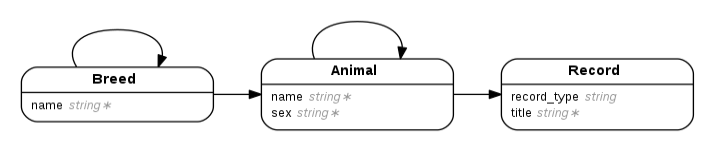
The primary classes are Animal, Breed, and Record, while DashboardMessage and User are utilized in a secondary manner for presenting customized dashboard messages and restricting access to the administrator panel respectively. Each Breed has a one to many relationship with Animal representing all the animals belonging to that Breed, as well as a one-to-many relationship between Breed and itself representing an optional hierarchy of Breeds, i.e. Dog is the parent of Chihuahua. Animal has the inverse many-to-one relationship with Breed, two optional many-to-one relationships with Animal representing each Animal’s paternal and maternal relationships, and inverse one-to-many relationships with Animal for the Animal’s progeny, though only one of which will contain any relations. Record has a many-to-one relationship with Animal representing the various types of records, e.g. pedigree, vaccinations, that an Animal may have.

Figure . ERD

## UI Design

Below are the low-fidelity versions of the user interface for the catalogue; high-fidelity user interface is out of scope for this project. Provided are the Catalogue and the Profile.

Figure 2 shows the main catalogue page. It shows each animal along with a profile picture, its sex, breed, sire, and dam, as well as how many images and how many records are available. By clicking the name of each animal, the view changes to the profile for that animal, see figure 3. At the top right hand side is a link to the admin panel, where new animals may be added.

Figure 3 shows for the selected animal the profile picture enlarged, sex, and breed. In addition to these basic attributes, there are 3 collapsible panels each for images, progeny, and records. These panels only appear if they contain data. The progeny panel uses the same components as the catalogue. The images panel shows larger versions of all attached images for the animal. Similarly, the records panel shows all the associated records with the animal.

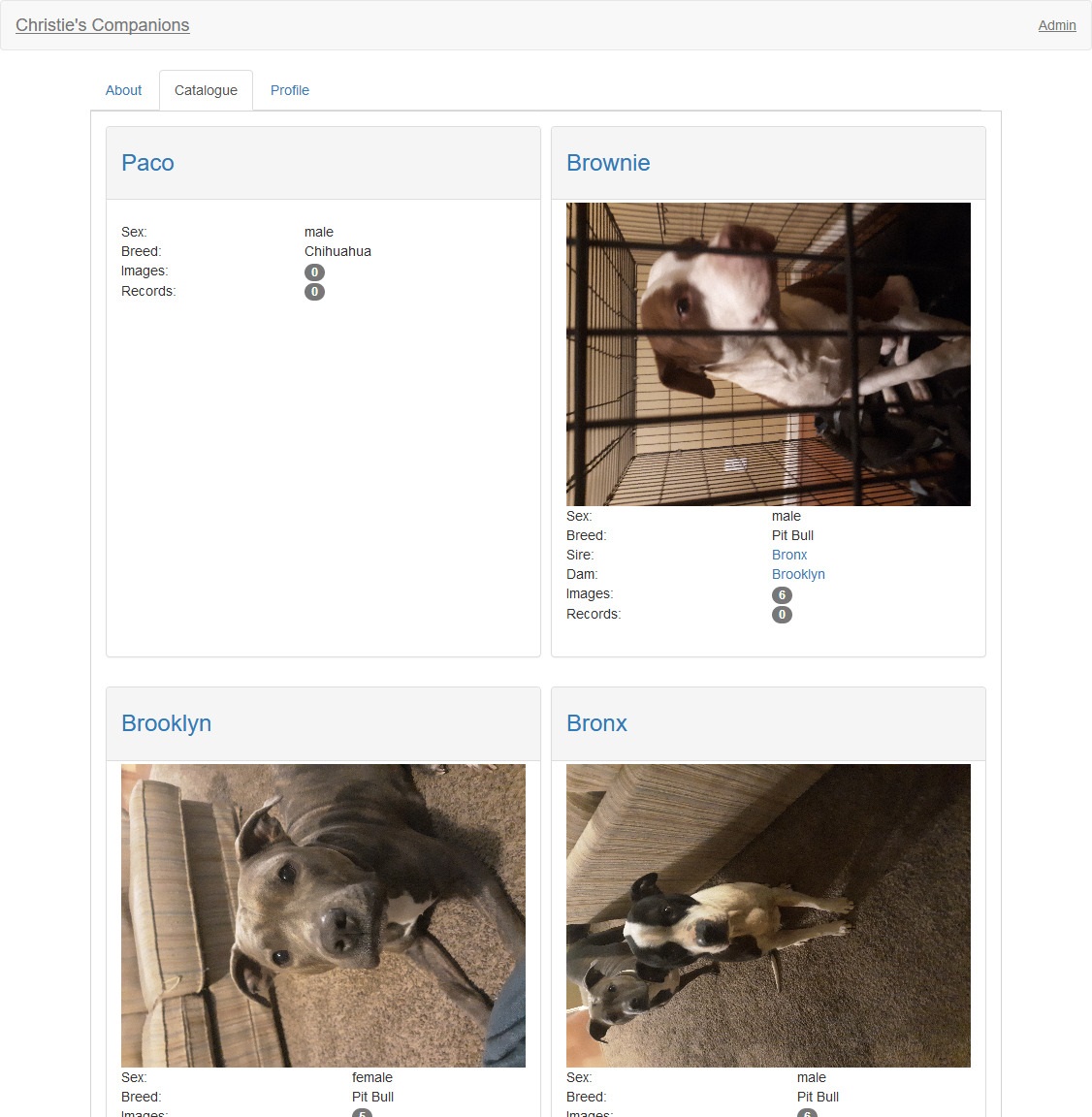
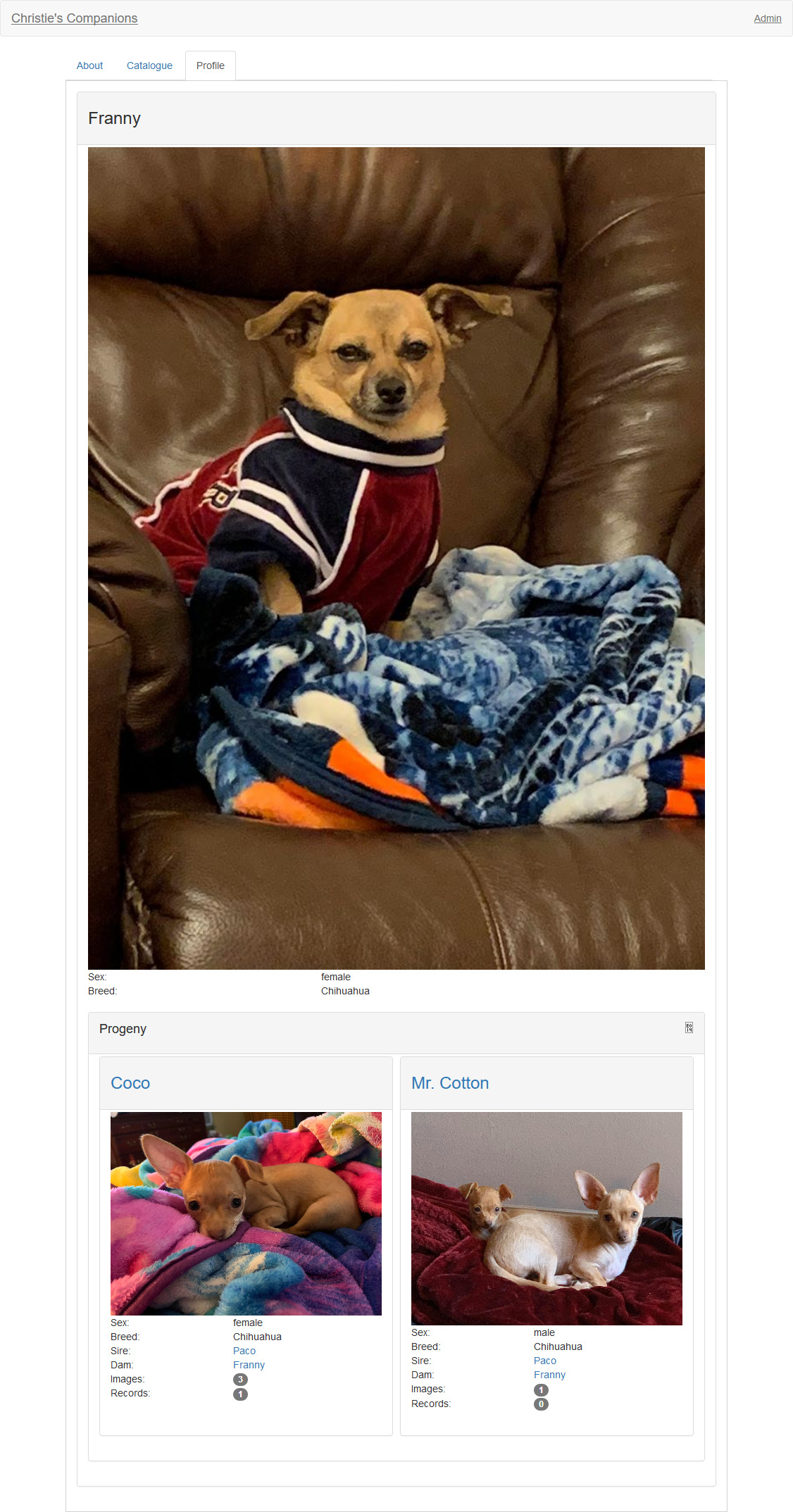
*Figure 2. Catalogue*

Figure 3. Profile



# Unit Test Plan

## Introduction

### Purpose

To ensure that the validation rules for the classes for Animal, Breed, and Record are enforced correctly, a series of unit tests were developed using the MiniTest library for Ruby. Each unit test will create an invalid object of each class and then check for a specific field that is required. The test will pass only if the specific field is set.

Provide a brief description of the testing method(s) that you used and what the results it yielded. Also, what remediation was required if necessary and how it would be performed.

### Overview

Since the application is designed to replace a paper-based record keeping system, it is vital to ensure that the required data is entered into the database. The validation rules get applied at the moment when a new record or changes to an existing record attempt to be persisted to the database which then sets an errors attribute on the record with messages of the failed validations; thus, by creating a new record missing the correct attribute and checking the error messages for the field to be tested, we can ensure that the database will only have valid data.

Being written in MiniTest, the test suite can be run from with the rails command which outputs the results of each test as well as a summary of all tests to the console.

## Test Plan

### Items

For each major class (Breed, Animal, and Record), there is a set of attributes that are required, and for each of which, there will be a test. The tests are as follows:

1. Breed
   1. Name
2. Animal
   1. Name
   2. Sex
   3. Breed
3. Record
   1. Animal
   2. Title

This gives a total of six tests.

### Features

To test each attribute, a new record object is created by calling the class method “new” and saving it to a variable. Then, the “save” method is called on the record object. Then the asset method is called which will cause the test to pass if the parameter evaluates to a “truthy” value. The parameter passed to asset is the negation of calling the “empty?” method on the records error messages mapped to the attribute tested.

### Deliverables

On a successful test, a green dot (“.”) will print to the console; on a failure, a detailed error message and stack trace is shown. Then, after all tests have been run, a summary of successes to failures will be displayed.

### Tasks

1. Setup the tests in the “test/models” directory according to MiniTest specification.
2. Create the code to be tested.
3. Run the test suite with the command “bundle exec rails test”.
4. Evaluate the result and fix any failed tests.

### Needs

The testing library used was MiniTest version 5.13.0, though any version greater than or equal to 5.1 but less than 6.0 will suffice.

### Pass/Fail Criteria

As described above, by checking whether the list of error messages for the tested attribute is empty, we can determine whether the record’s attribute is valid. A failing test would feature an error message for the tested attribute.

## Specifications

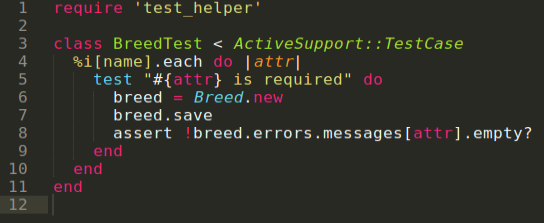
Figure 4: screenshot of the Breed name test from file “test/models/breed\_test.rb”: 

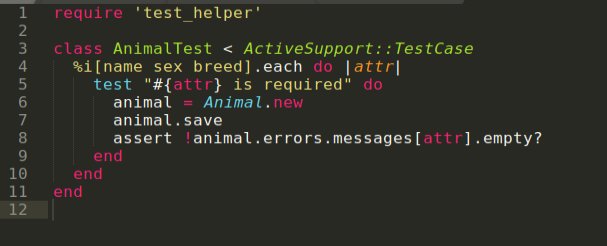
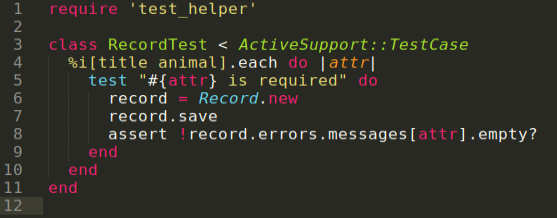
Figure 5: screenshot of the Animal name, breed, sex tests from file “test/models/animal\_test.rb”: 

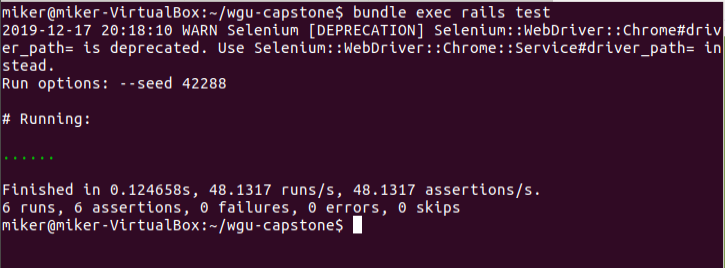
Figure 6: screenshot of the Record title, animal tests from file “test/models/record\_test.rb”: 

## Procedures

As shown above, each test is created dynamically by iterating over a list of attributes, then setting up the test using MiniTest specifications. This approach allows each test to be consistent with the others for the same class while also reducing the amount of code that needs to be written and maintained.

## Results

The following image shows the result of running the test suite on the command line. There are six green dots to indicate that all tests have passed as well as a summary that shows the percentage of passing tests as 100%.



# Source Code

The source code can be found in the archive file wgu-capstone-master.zip.

# Link to Live Version

The live version can be found at http://christiescompanions.hatchie.studio

# User Guide

The user guide can be found in the file README.md in the source code. The sky will include instructions on how to install the application for development as well as login to manage the data of the application.