CodeClan_
Bridging the digital skills gap.

Evidence Gathering Document for SQA Level 8 Professional Developer Award.

This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

Please fill in each point with screenshot or diagram and description of what you are showing.

Each point requires details that cover each element of the Assessment Criteria, along with a brief description of the kind of things you should be showing.

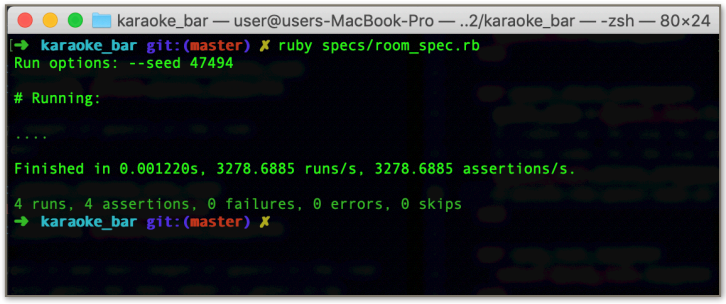
Week 2

Unit	Ref	Evidence
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running
		Description:

Paste Screenshot here

```
def setup
  @room1 = Room.new("1", 3, 15)

  @guest1 = Guest.new("Charlie", 50)
  @guest2 = Guest.new("Sam", 100)
  @guest3 = Guest.new("Alison", 150)
  @guest4 = Guest.new("Ross", 70)

  @guest_list = [@guest1, @guest2, @guest3, @guest4]
end

def test_room_has_a_name
  assert_equal("1", @room1.name)
end
# passed test

def test_check_for_guests_checked_into_room
  assert_equal(0, @room1.guests_number())
end
# checking length of guests in the room
# passed test

def test_guests_checked_into_room
  @room1.guest_check_in(@guest1)
  assert_equal(1, @room1.guests_number())
end
# check guest in and test length of guests in the room
# to make sure guest has been added to guest array
# passed test

def test_check_in_two_guests
  @room1.guest_check_in(@guest1)
  @room1.guest_check_in(@guest2)
  assert_equal(2, @room1.guests_number())
end
# check in two guests as above
# passed test

def test_guests_checked_out_of_room
  @room1.guest_check_in(@guest1)
  @room1.guest_check_in(@guest2)
  @room1.guest_check_out(@guest2)
  assert_equal(1, @room1.guests_number())
end
# check guest out and test length of guests in the room
```

```
def initialize(name, capacity, entry_fee)
  @name = name  #MVP
  @capacity = capacity ## ext 1
  @entry_fee = entry_fee##ext 1
  @guest_spend = guest_spend ##ext 2 (entry fee/dri
  @guest_list = []    #MVP
  @song_list = []     #MVP

end

def guests_number()
  return @guest_list.length()
end

def guest_check_in(guest)
  @guest_list.push(guest)
end

def guest_check_out(guest)
  guest_name = @guest_list.find_index(guest)
  @guest_list.delete_at(guest_name)
end
```



Description here

Screenshot 1: Array set-up in minitest to test several functions.
Screenshot 2: The array is used here in the **guest_check_in** and **guest_check_out** functions to add and remove guest elements from the array. **guests_number** is a function to check the length of the guest list.
Screenshot 3: All tests are passed.

Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running
		Description:

Paste Screenshot here

phone_numbers.rb

```ruby

def numbers_count(directory)
  return directory.count
end

def add_name(directory, name)
  directory.push(name)
end

```

phone_numbers_spec.rb

```ruby
require( 'minitest/autorun' )
require( 'minitest/rg')
require_relative( '../phone_numbers' )

class TestFriends < MiniTest::Test

  def setup
   @new_person = {
      name: "Shona", mobile_number: 7788888, home_number: 1215557654
    }

    @directory = {
      name: "Harry", mobile_number: 7725438, home_number: 1215674567
    },
    {
      name: "Gemma", mobile_number: 7712345, home_number: 01136673444
    },
    {
      name: "Ross", mobile_number: 9948374, home_number: 14547765566
    }
  end

  def test_numbers_count
      count = numbers_count(@directory)
      assert_equal(3, count)
    end

  def test_add_name_to_directory
      add_name(@directory, @new_person)
      result = numbers_count(@directory)
      assert_equal(4, result)
    end
end
```

```
phone_numbers — user@users-MacBook-Pro — ..phone_numbers — -zsh — 8...
➜ phone_numbers git:(master) ✗ ruby specs/phone_numbers_spec.rb
Run options: --seed 27727

# Running:

..

Finished in 0.000967s, 2068.2523 runs/s, 2068.2523 assertions/s.

2 runs, 2 assertions, 0 failures, 0 errors, 0 skips
➜ phone_numbers git:(master) ✗
```

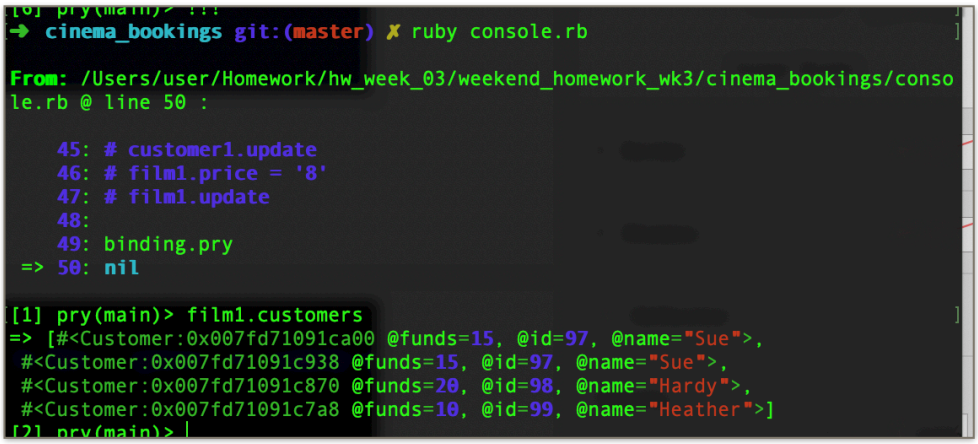
- **Screenshot 1:** hash of names and phone numbers - @directory
- **Screenshot 2:** result of @new_person being added to @directory hash

Week 3

Unit	Ref	Evidence
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running
		Description:

Paste Screenshot here

```ruby
def customers()
  sql = "SELECT customers.* FROM customers
  INNER JOIN tickets
  ON tickets.customer_id = customers.id
  WHERE tickets.film_id = $1"
  values = [@id]
  customers_hash = SqlRunner.run(sql, values)
  return customers_hash.map{ |customer| Customer.new(customer)}
end
```




```ruby
customer1 = Customer.new( {'name' => 'Sue', 'funds' => '15'})
customer1.save
customer2 = Customer.new( {'name' => 'Hardy', 'funds' => '20'})
customer2.save
customer3 = Customer.new( {'name' => 'Heather', 'funds' => '10'})
customer3.save

film1 = Film.new( {'title' => 'Apocalypse Now', 'price' => '5'})
film1.save
film2 = Film.new( {'title' => 'The Departed', 'price' => '5'})
film2.save
film3 = Film.new( {'title' => 'Some Like It Hot', 'price' => '3'})
film3.save

ticket1 = Ticket.new({'customer_id' => customer1.id, 'film_id' => film1.id})
ticket1.save
ticket2 = Ticket.new({'customer_id' => customer1.id, 'film_id' => film1.id})
ticket2.save
ticket3 = Ticket.new({'customer_id' => customer2.id, 'film_id' => film1.id})
ticket3.save
ticket4 = Ticket.new({'customer_id' => customer3.id, 'film_id' => film3.id})
ticket4.save
ticket5 = Ticket.new({'customer_id' => customer3.id, 'film_id' => film1.id})
ticket5.save
```

Description here

Screenshot 1: the function **customers( )** is used to search for customers booking to see a film.
Screenshot 2: The result is returned by running the **ruby console.rb** file and calling the function **film1.customers**. There are 4 results returned.
Screenshot 3: Test data.

Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running
		Description:

Paste Screenshot here

```ruby
def self.all_sorted_by_name()
  sql = "SELECT * FROM customers
  ORDER BY name"
  values = []
  customers = SqlRunner.run(sql, values)
  result = customers.map{ |customer| Customer.new(customer)}
  return result
end
```

```
From: /Users/user/Homework/hw_week_03/weekend_homework_wk3/cinema_bookings/console.rb @ line 54 :

    49: # customer1.update
    50: # film1.price = '8'
    51: # film1.update
    52:
    53: binding.pry
 => 54: nil

[1] pry(main)> Customer.all_sorted_by_name
=>[#<Customer:0x007ffdae480478 @funds=50, @id=32, @name="Amy">,
 #<Customer:0x007ffdae4802e8 @funds=20, @id=30, @name="Hardy">,
 #<Customer:0x007ffdae480180 @funds=10, @id=31, @name="Heather">,
 #<Customer:0x007ffdae480068 @funds=15, @id=29, @name="Sue">,
 #<Customer:0x007ffdae47bef0 @funds=25, @id=33, @name="Zed">]
[2] pry(main)>
```

```ruby
customer1 = Customer.new( {'name' => 'Sue', 'funds' => '15'})
customer1.save
customer2 = Customer.new( {'name' => 'Hardy', 'funds' => '20'})
customer2.save
customer3 = Customer.new( {'name' => 'Heather', 'funds' => '10'})
customer3.save
customer4 = Customer.new( {'name' => 'Amy', 'funds' => '50'})
customer4.save
customer5 = Customer.new( {'name' => 'Zed', 'funds' => '25'})
customer5.save
```

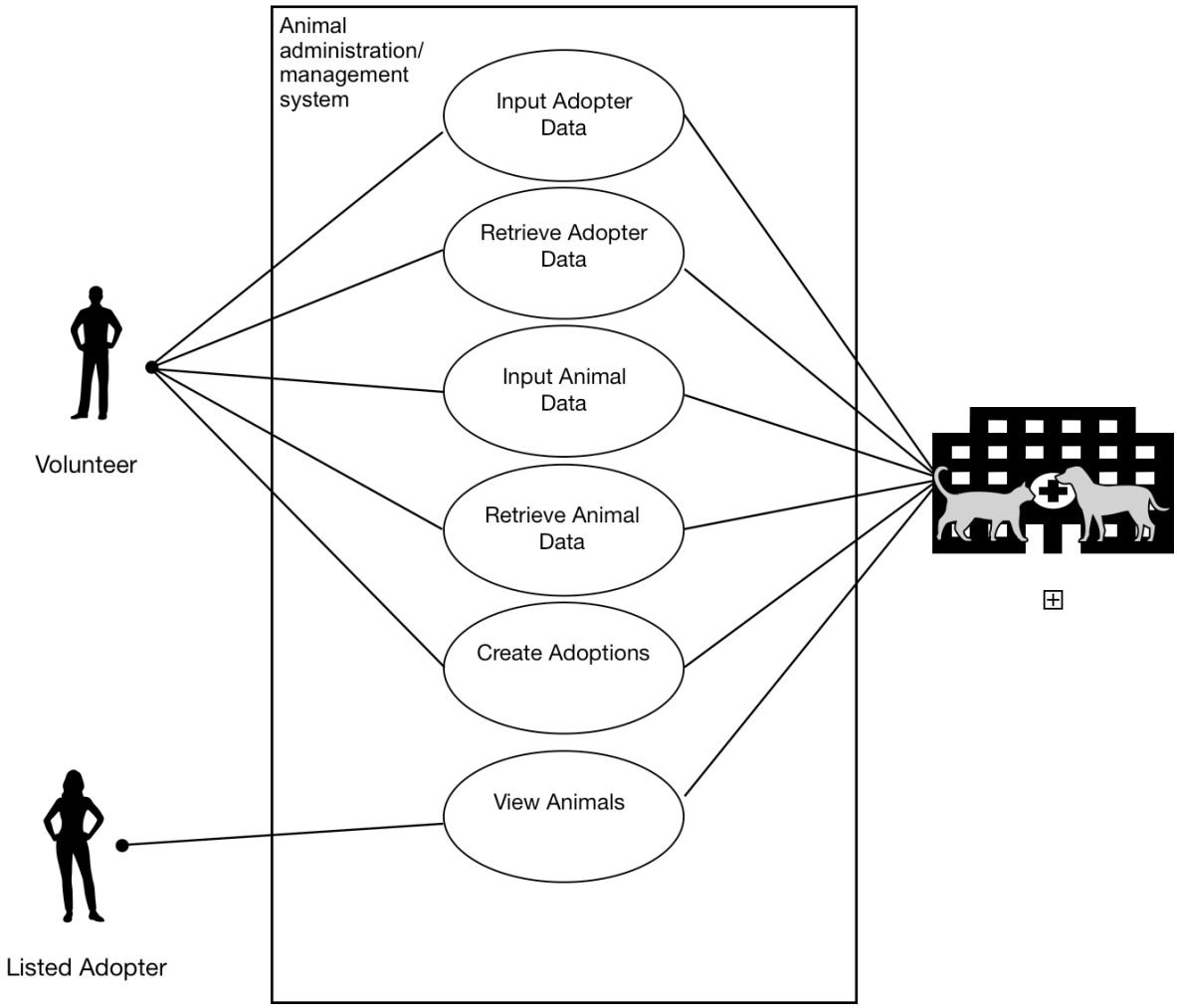
Description here

- **Screenshot 1:** the function **self.all_sorted_by_name()** is used to search for customers booking to see a film and returns the result ordered by name in alphabetical order.
- **Screenshot 2:** The result is returned by running the **ruby console.rb** file and calling the function **Customer.all_sorted_by_name**. There are 5 results returned.
- **Screenshot 3:** Additional customers added to test data.

Week 5 and 6

Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram
		Description:

Paste Screenshot here

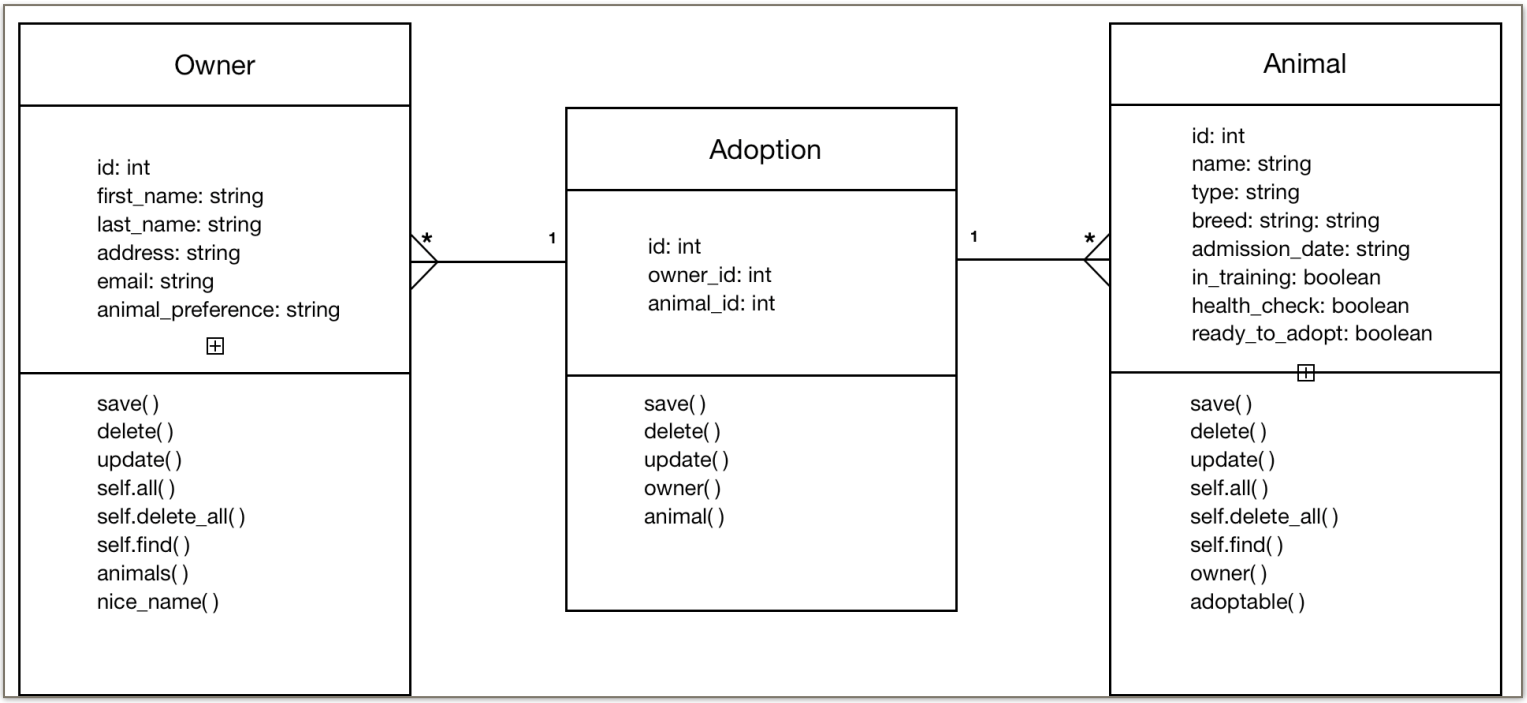


Description here

This system would be used by a volunteer admin user to input and update data about animals and potential owners/adopters for storage on a database. They would also have the functionality of being able to then mark an animal as adopted. The user would be presented with forms with editing functions to do this.
The listed adopter would be able to view animals ready to adopt.

Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram
		Description:

Paste Screenshot here

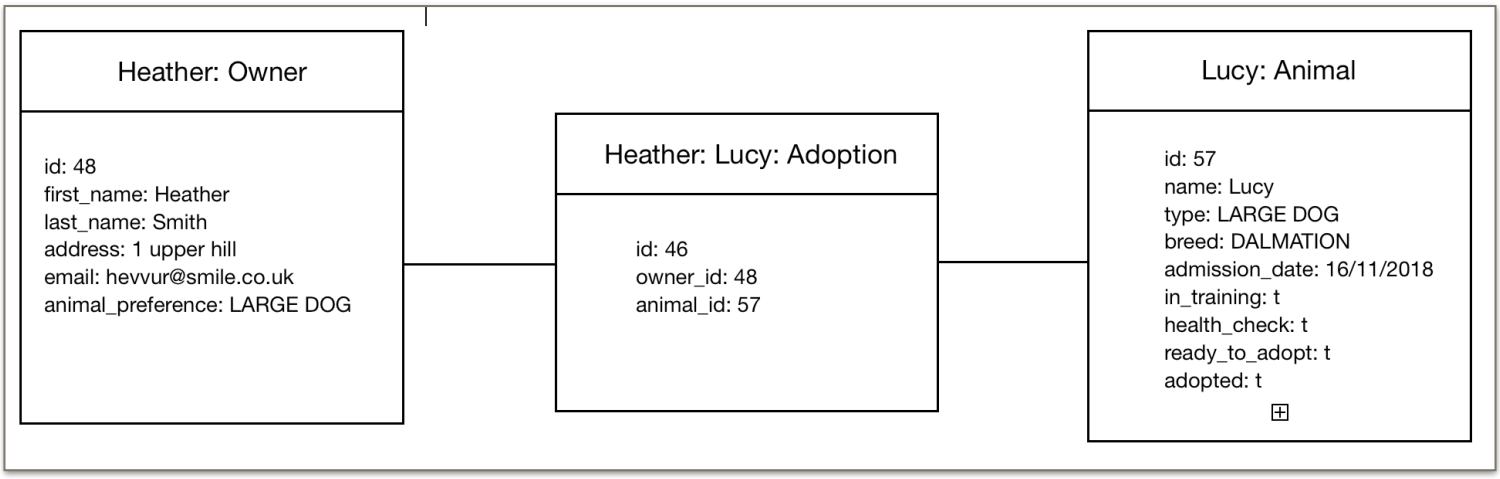


Description here

A class diagram of three classes in a many to many relationship. Each class is part of a process to adopt an animal.

Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram
		Description:

Paste Screenshot here

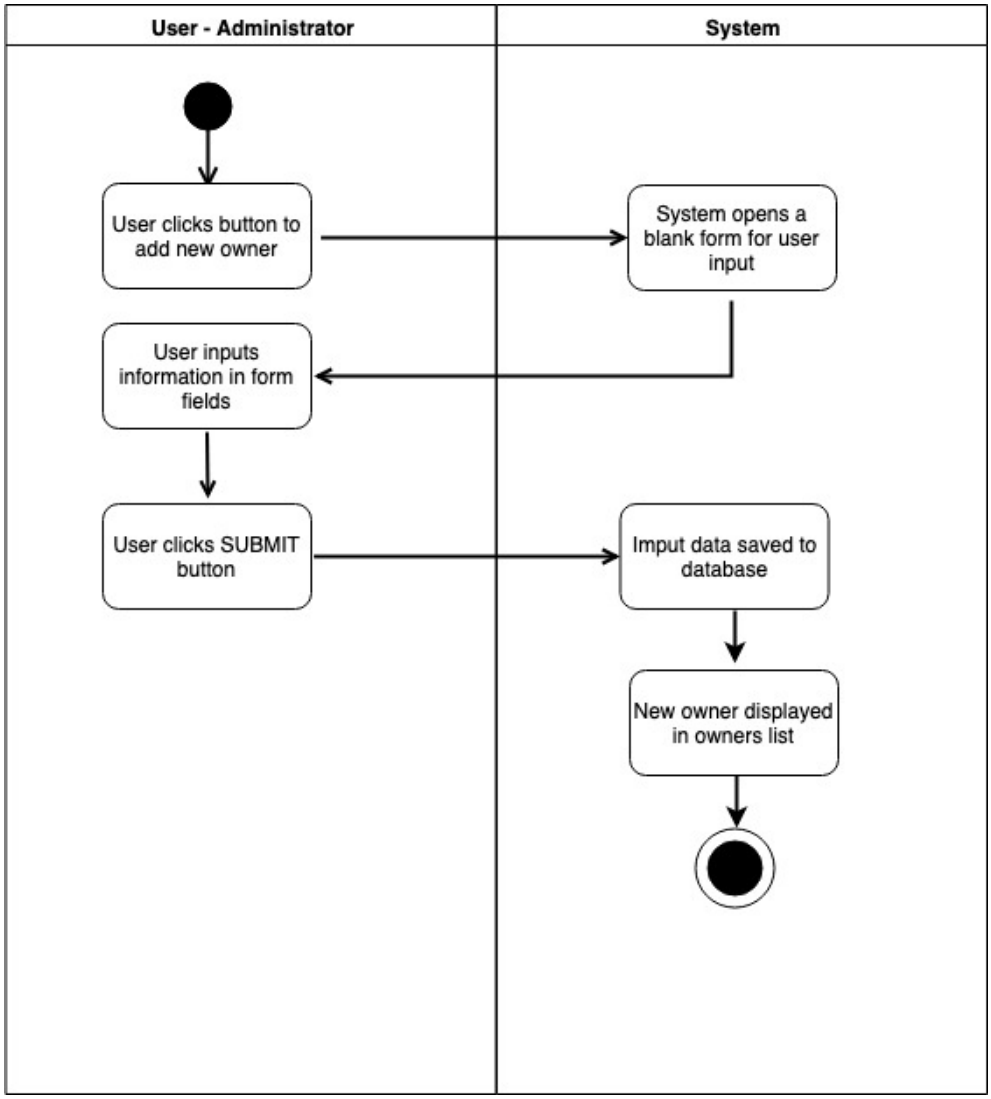


Description here

Object diagram of an owner object having adopted an animal object.

Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram
		Description:

Paste Screenshot here



Description here

Adding a new owner to a list of owners database for display on a full ldist of owners

Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time
		Description:

Paste Screenshot here

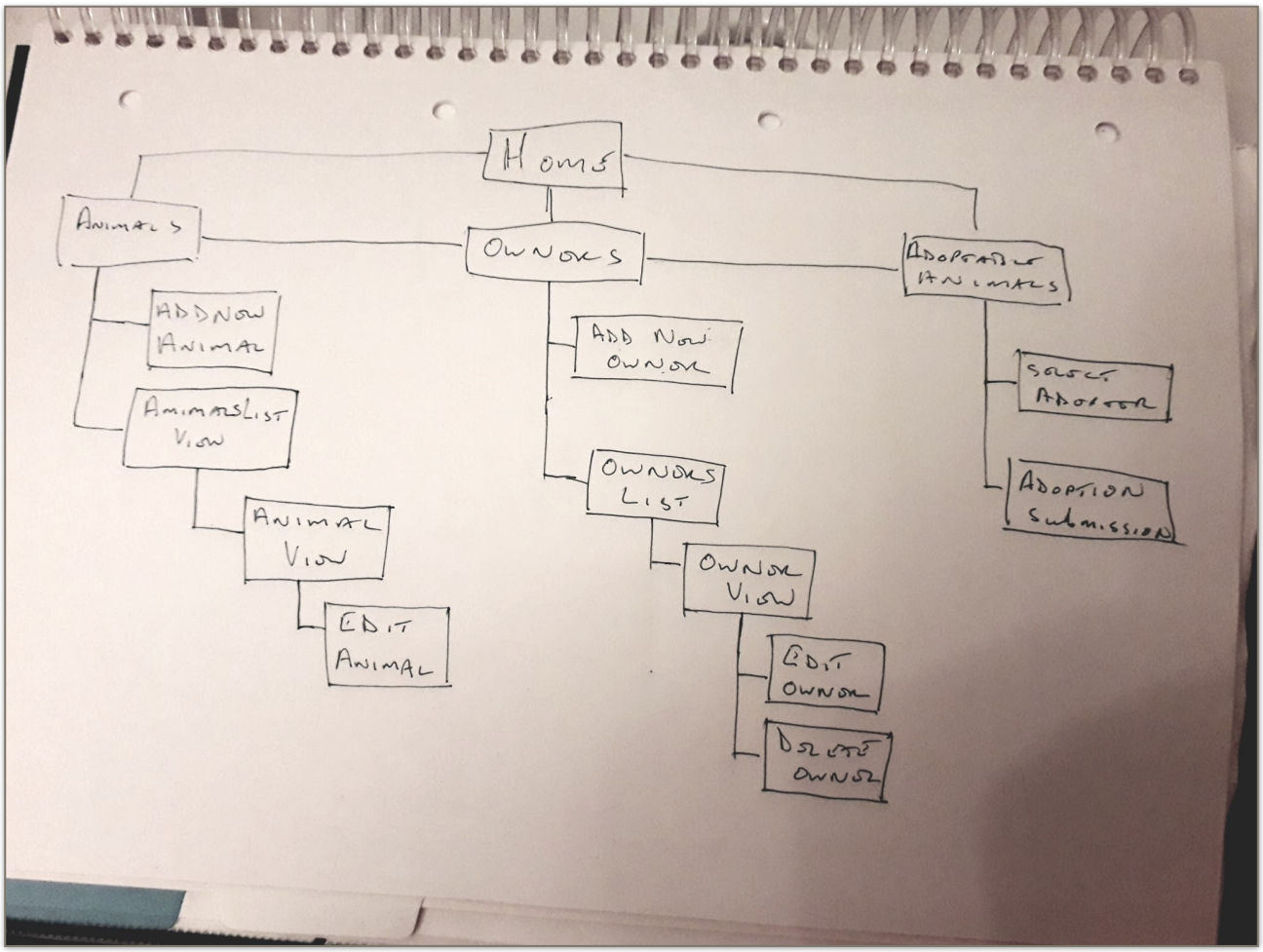
Constraint Category	Implementation Constraint	Solution
Hardware and software platforms	Apps may not run on all devices.	Choose platforms most appropriate to devices desired to run the app.
Performance requirements	Depending on data being stored, the app may run slowly.	Make sure the database can fulfil performance expectations.
Persistent storage and transactions	Local storage of data may become a problem if limits are low.	Implement immediately, or make plans, for remote storage of data.
Usability	Difficulty navigating the app using complicated UX design.	Spend time in planning and considering the user side.
Budgets	Long term data storage may require a larger budget.	Ensure only necessary data is stored and requirements reviewed regularly.
Time	6 days duration.	Awareness of the MVP functionality requirements. Set small manageable goals. Review regularly. Extension can be added if time allows.

Description here

- **Screenshot:** considerations for constraints on production of an app and solutions to address these constraints.

Unit	Ref	Evidence
P	P.5	User Site Map
		Description:

Paste Screenshot here

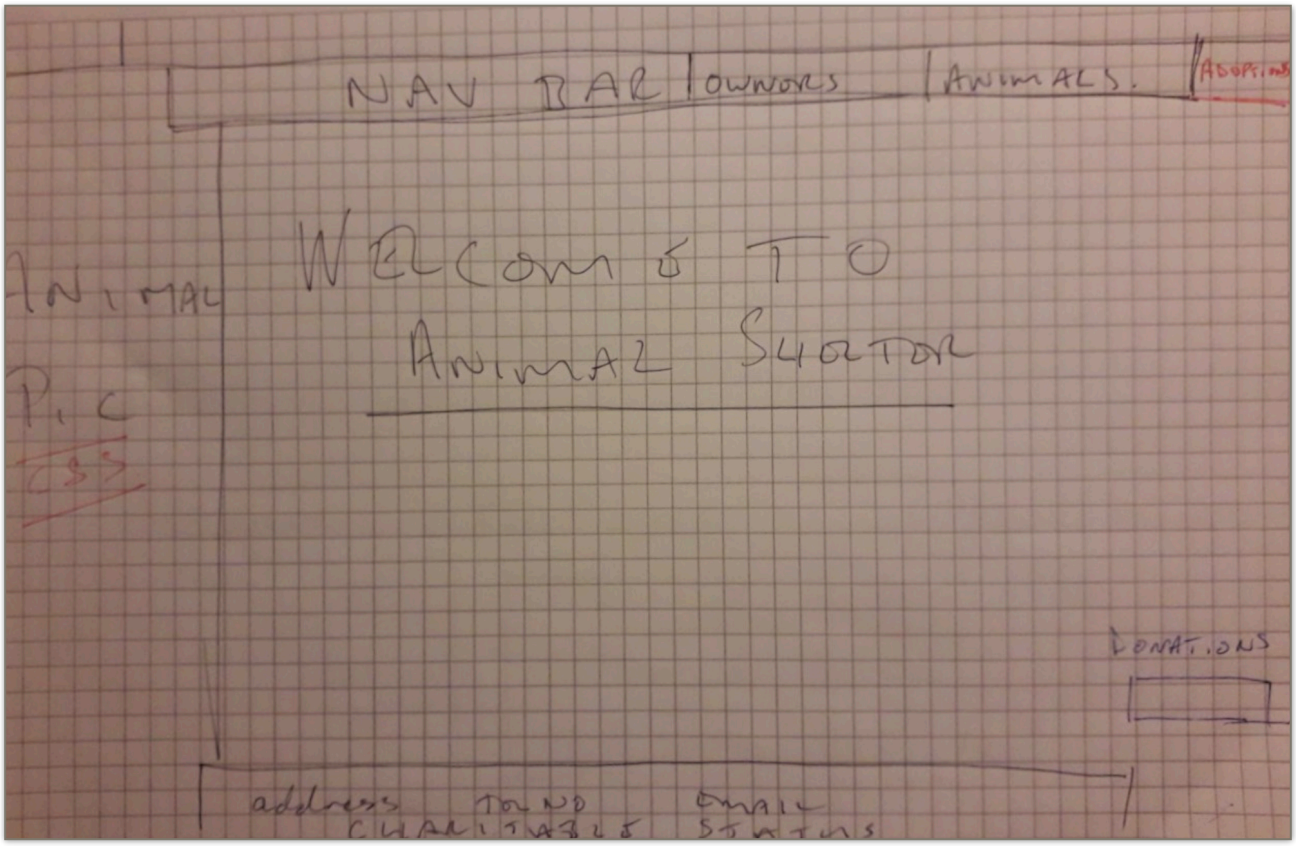


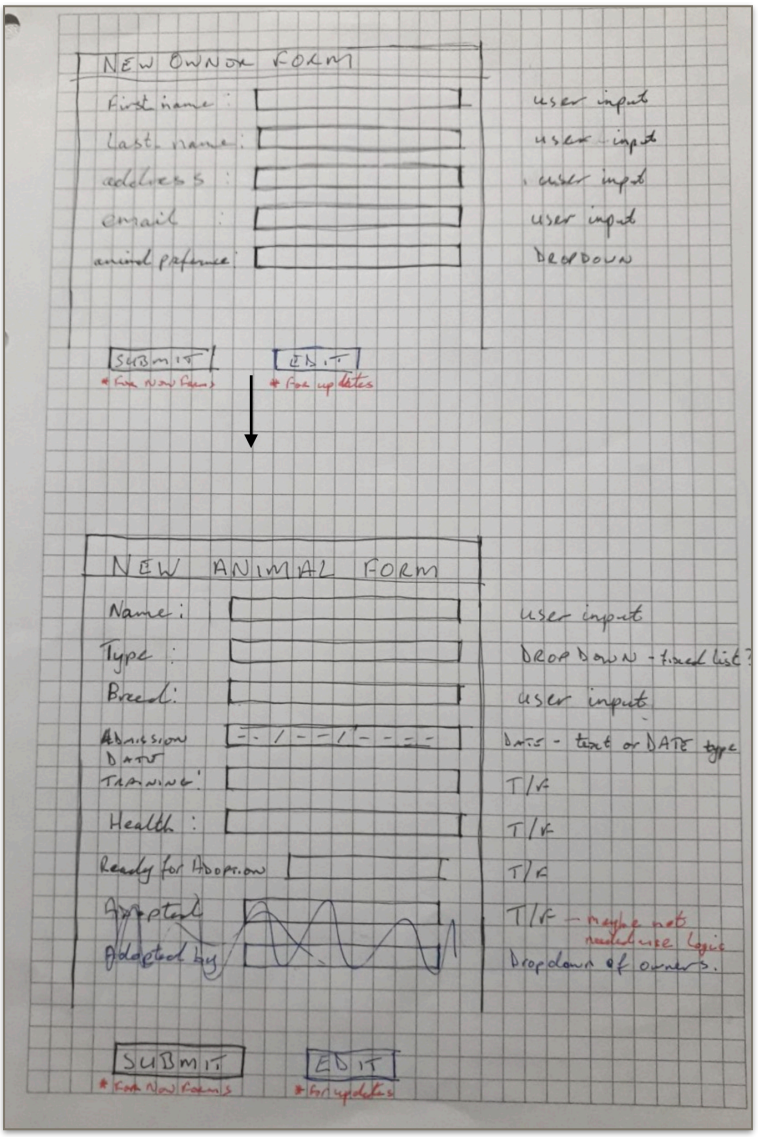
Description here

- **Screenshot:** animal shelter project user site map of website content

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description:

Paste Screenshot here



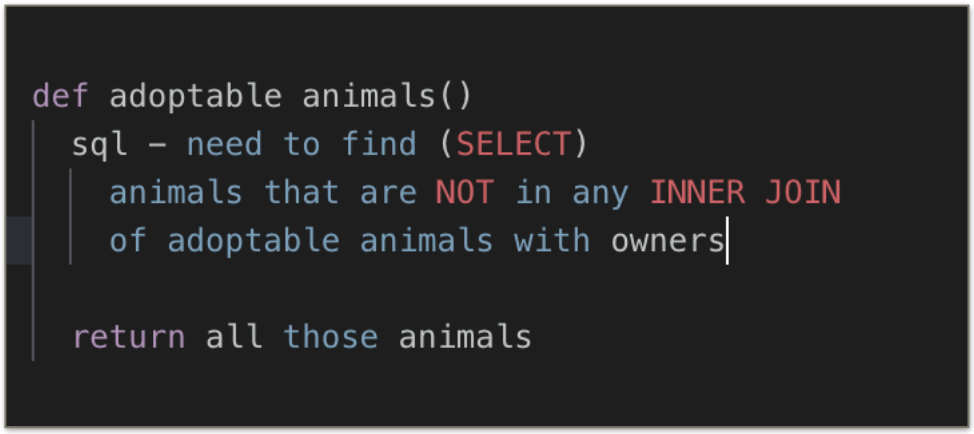


Description here

- **Screenshot 1:** the wireframe for home / index page for Animal Shelter project
- **Screenshot 2:** the wireframe for forms for new owner and new animal details to be added to the shelter database.
- The wireframes are structured to emulate a paper based form. Designing this way helped constructing the forms different input types.

Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		Description:

Paste Screenshot here



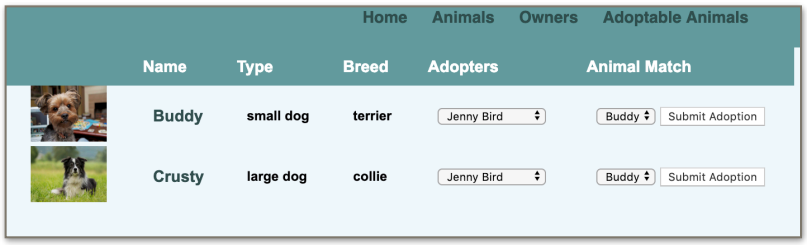

```
def self.adoptable()
  sql ="SELECT * FROM animals
    WHERE animals.id NOT IN
    (SELECT animals.id FROM animals
    INNER JOIN adoptions
    ON animals.id = adoptions.animal_id)
    AND ready_to_adopt = TRUE"
  values = []
  animals = SqlRunner.run(sql, values)
  return animals.map{ |animal| Animal.new(animal)}
end
```

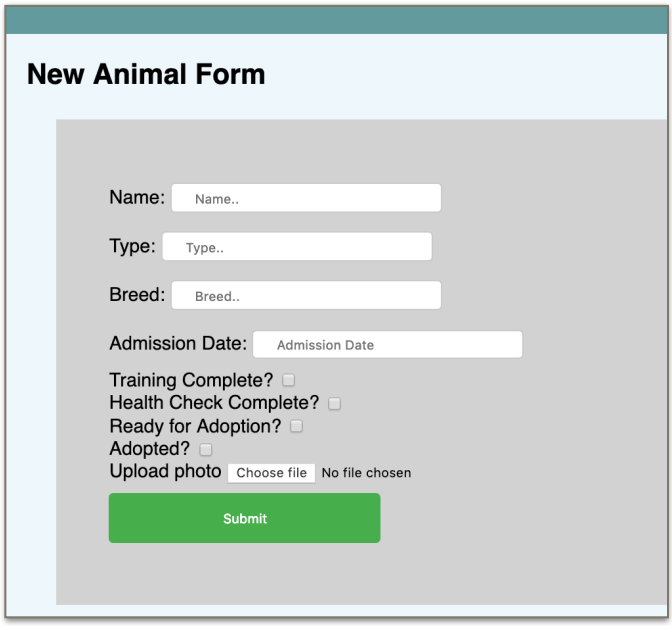
Description here

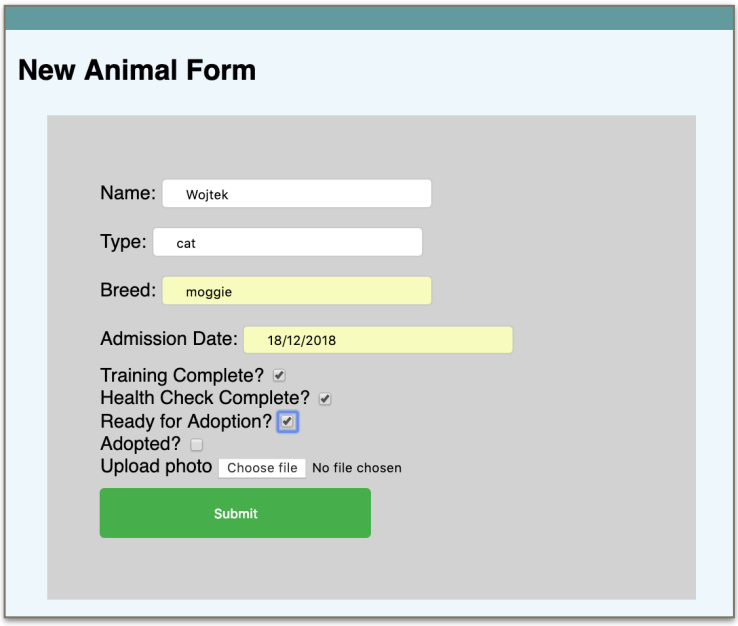
- **Screenshot 1:** the pseudocode written to figure out the requirement to extract data from a many to many schema where the data was outside the Inner Join.
- **Screenshot 2:** the final working method

Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way
		Description:

Paste Screenshot here





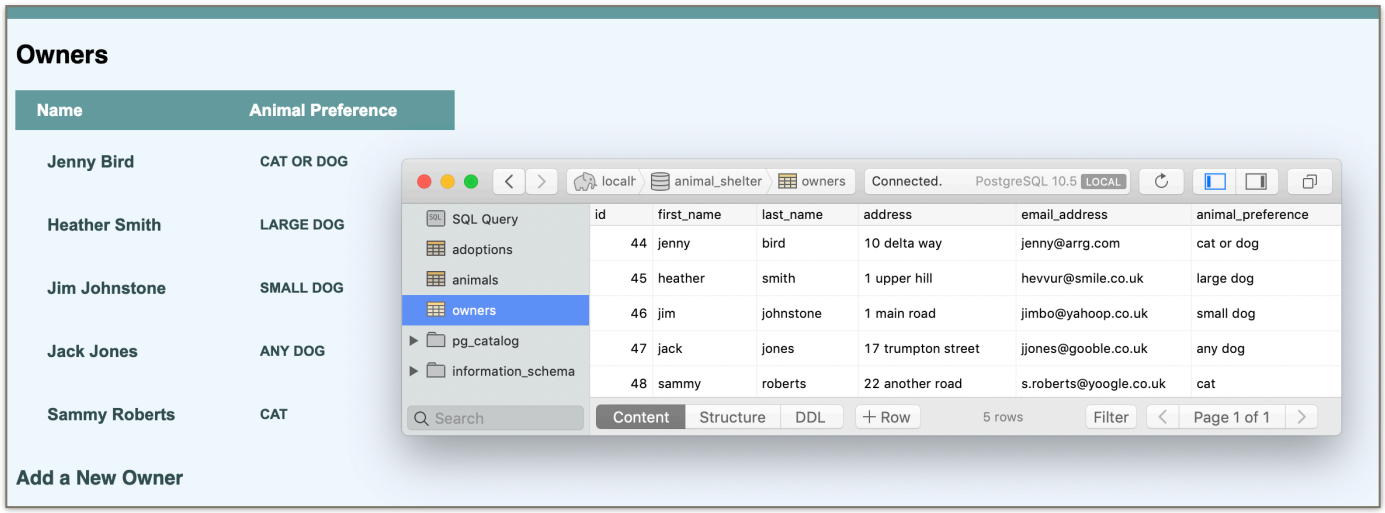


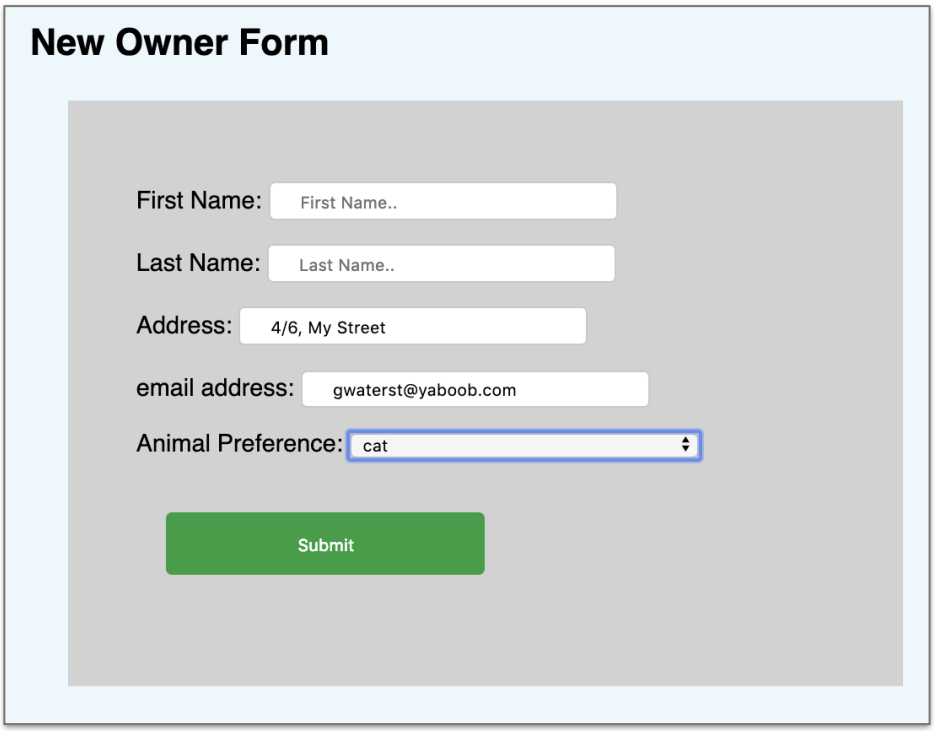
Description here

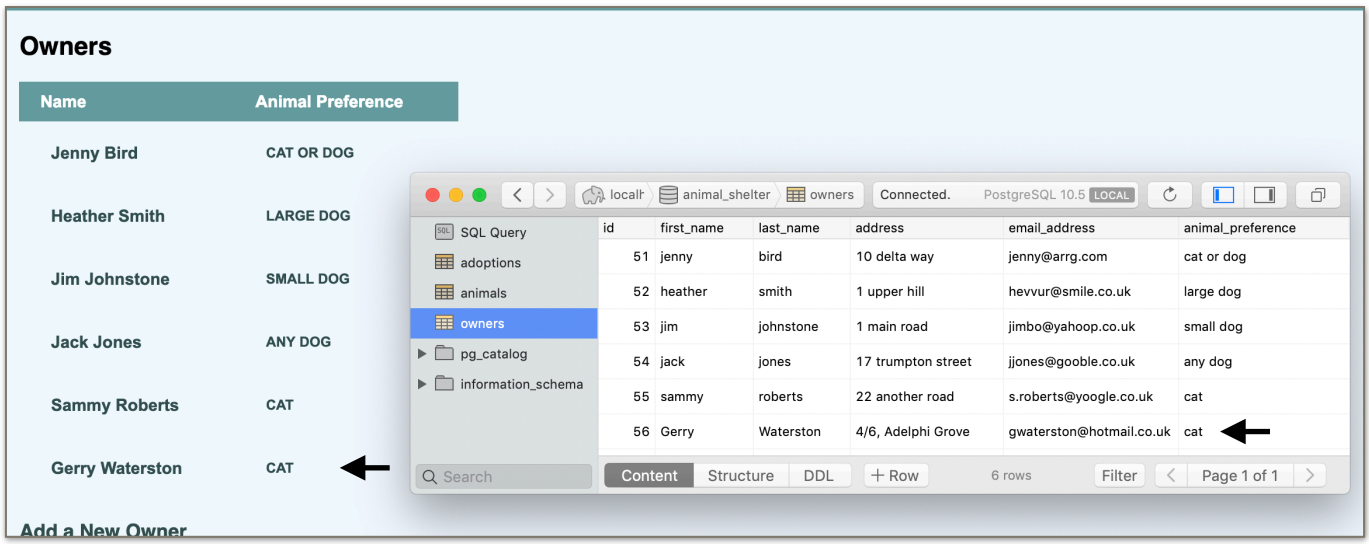
- **Screenshot 1:** list of current adoptable animals
- **Screenshot 2:** blank form to input new animal details and mark ‘ready to adopt’
- **Screenshot 3:** completed form saved on **Submit.**
- **Screenshot 4:** new animal added to list on adoptable animals page.

Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved
		Description:

Paste Screenshot here





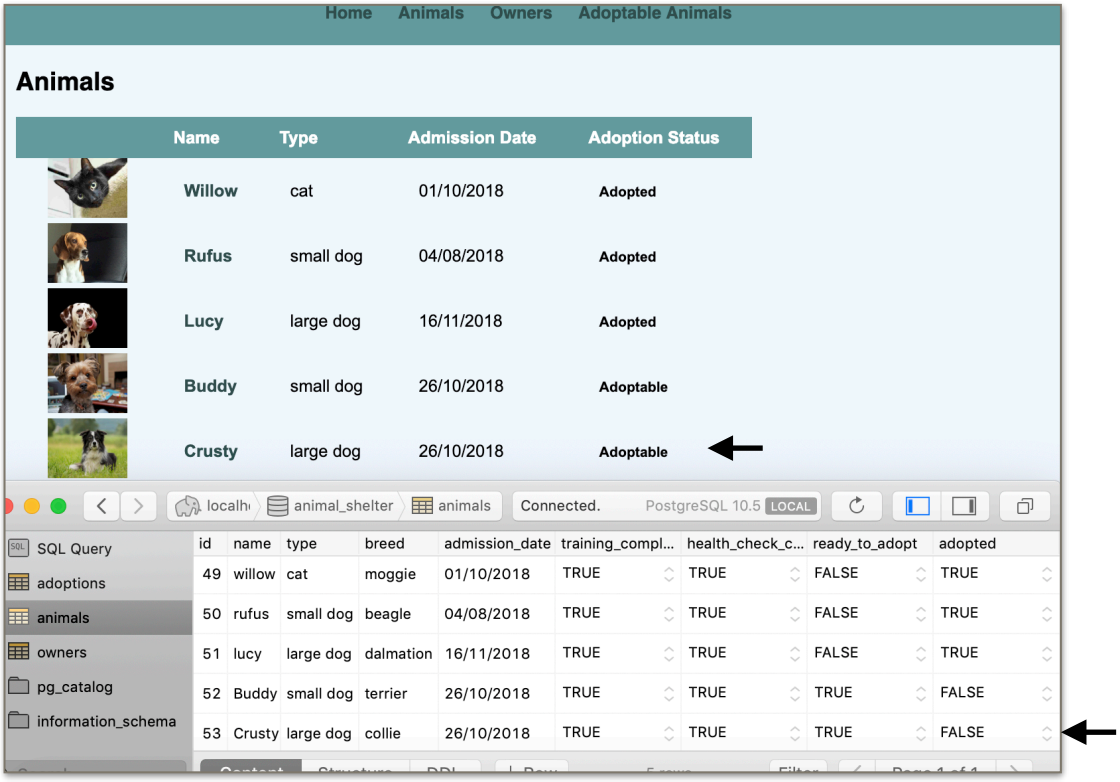


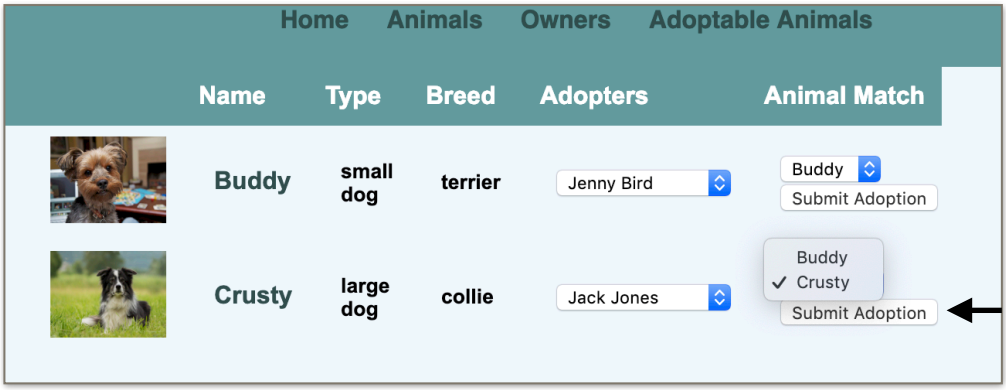
Description here

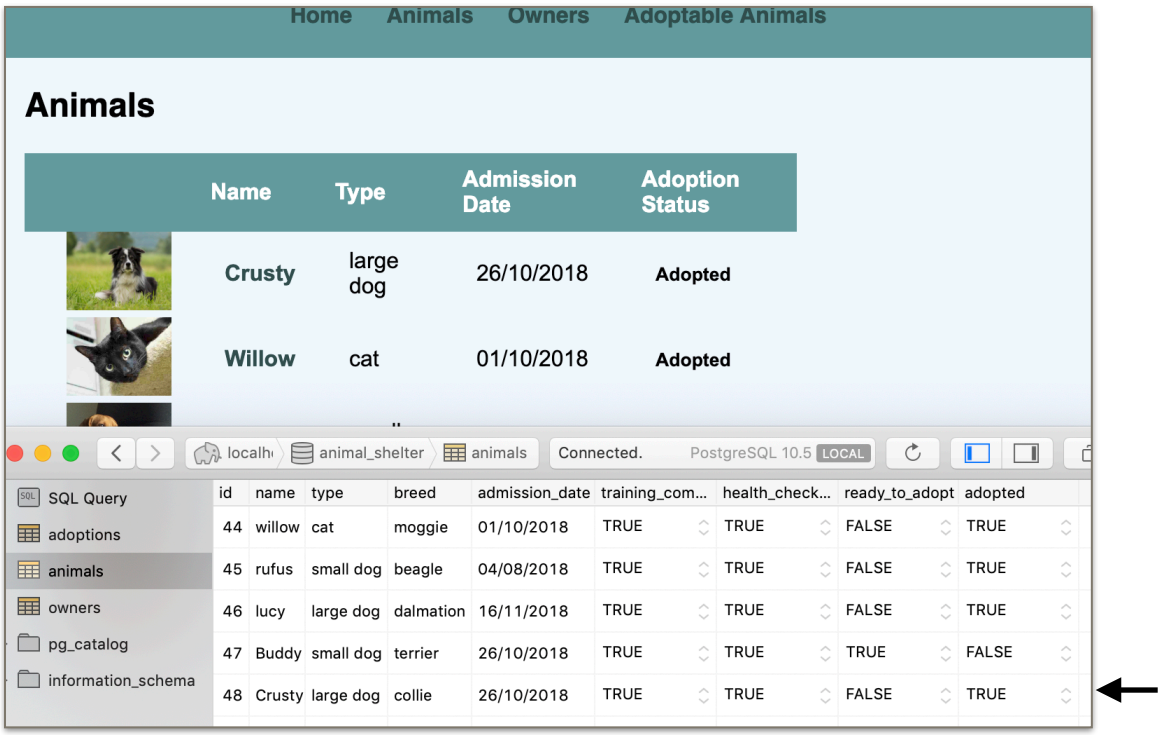
- Screenshot 1: list of current owners.
- Screenshot 2: blank form to input new owner details.
- Screenshot 3: completed form saved on **Submit** and updated on database.

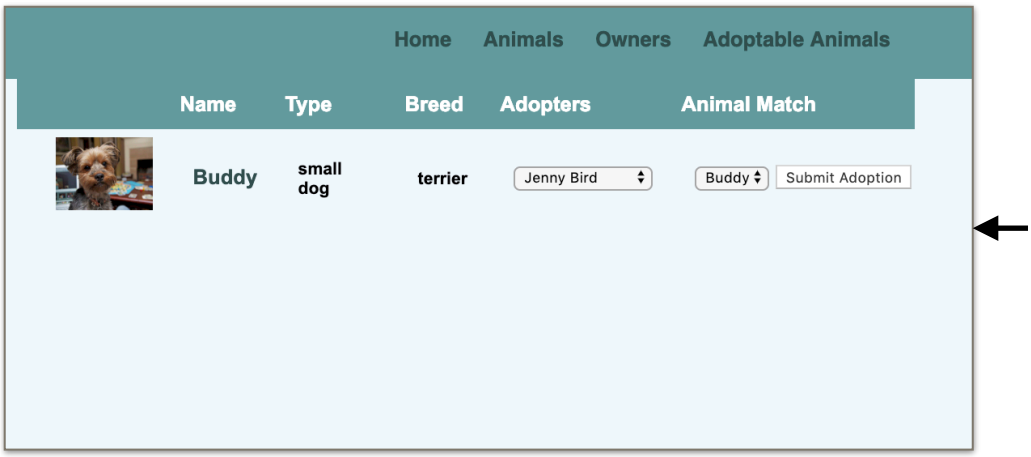
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program
		Description:

Paste Screenshot here







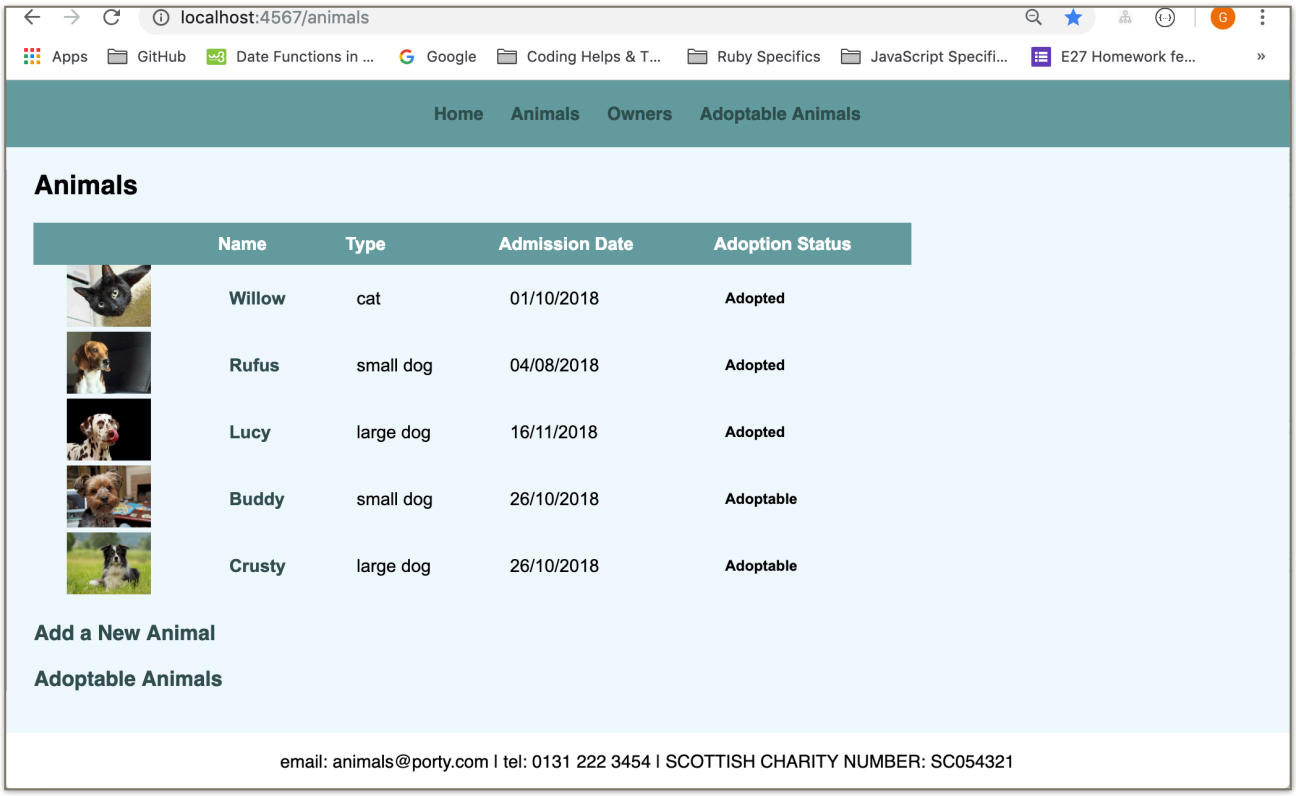


Description here

- **Screenshot 1:** a list of animals with adoption status and database showing true / false.
- **Screenshot 2:** user action of animal ‘Crusty’ being adopted.
- **Screenshot 3: ‘Crusty’ status now adopted and showing TRUE on database.**
- **Screenshot 4:** ‘Crusty’ removed from Adopted Animals view showing the user that the request has been processed.

Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.
		Description:

Paste Screenshot here



Description here

https://github.com/Ger-onimo/animal_shelter_project1

Project Brief:
Animal Shelter
The Scottish Animal Shelter accepts orphaned or stray animals and takes care of them until they can be adopted by a new owner. The shelter has a list of potential new owners for the animals. Animals may take a while to be trained up and made healthy before being available for adoption. They are looking for a management system to keep track of their animals and owners.

MVP:

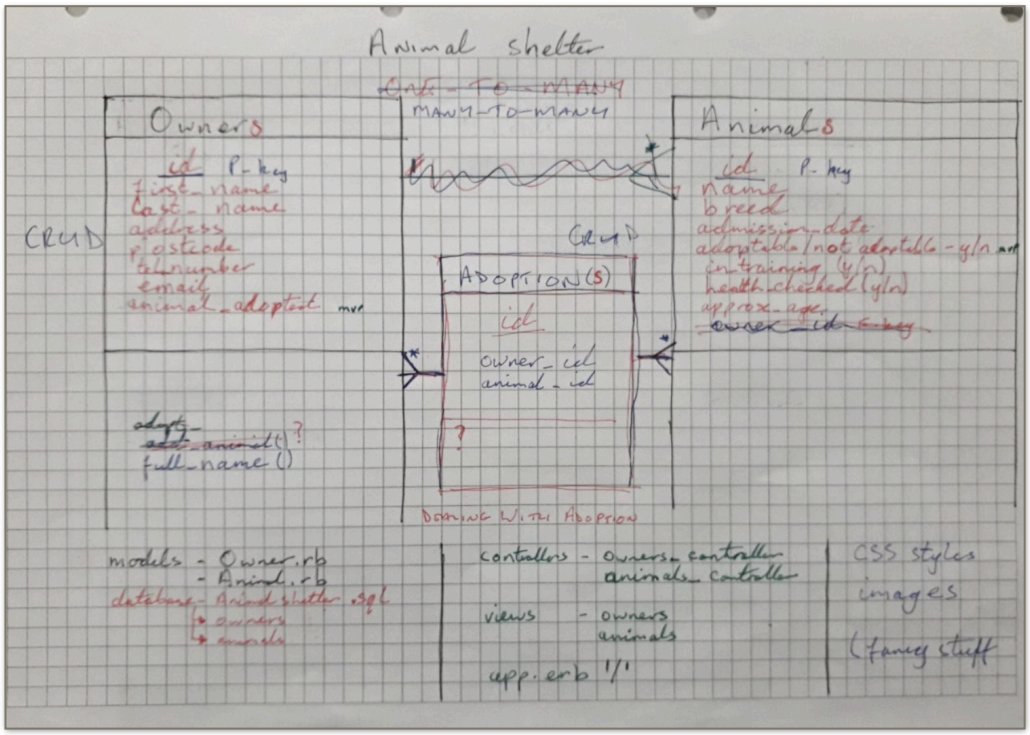
A list of all their animals and their admission date - **completed**
Mark an animal as being adoptable/not adoptable - **completed**
Assign an animal to a new owner - **completed**
List all the owners and their adopted animals - **completed**

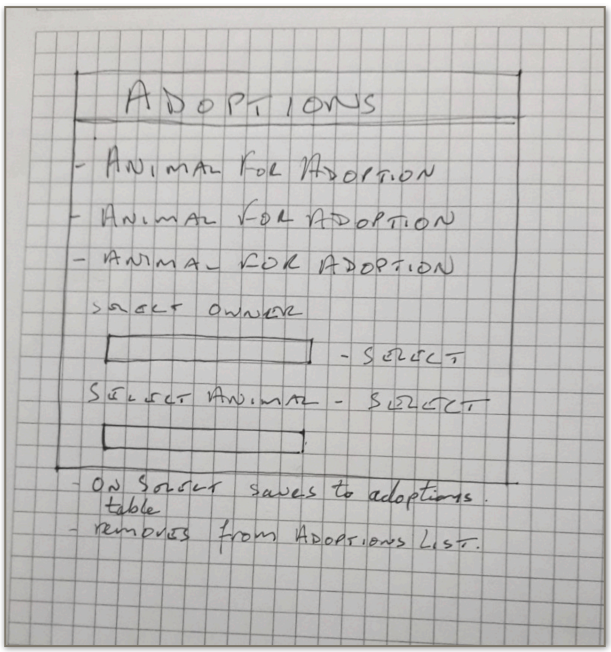
Possible Extensions:

CRUD actions for animals/owners - **completed**
Have separate pages for animals ready for adoption and ones still in training/vet care - **part done**
Search for animals by breed/type
Any other ideas you might come up with

Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description:

Paste Screenshot here





Description here

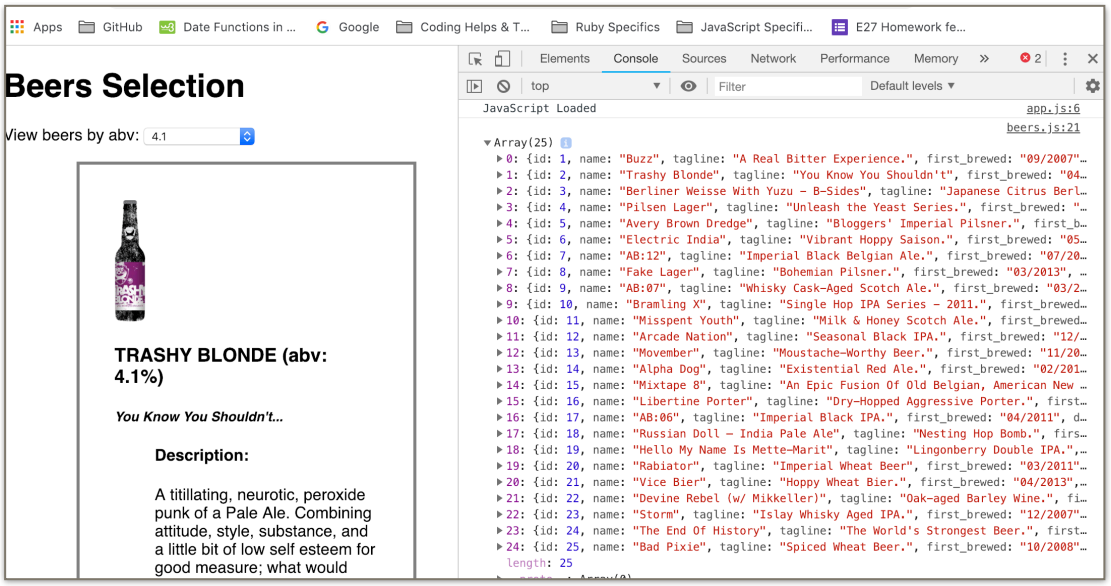
- **Screenshot 1:** database schemas and class diagram - initially set-up as one to many, but then changed to many to many to allow for expansion of functionality.
- **Screenshot 2:** wireframe of an additional form added after the relationship change.

Week 7

Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running
		Description:

Paste Screenshot here

```
15  };
16
17  Beers.prototype.getData = function () {
18    const request = new RequestHelper('https://api.punkapi.com/v2/beers');
19    request.get().then((data) => {
20      this.beersData = data;
21      console.log(data);
22      PubSub.publish('Beers:beers-data-ready', this.beersData);
23      this.publishAbvs(data);
```



Description here

- **Screenshot 1:** the code that requests the api to get all the data.
- **Screenshot 2:** Brewdog beers API being used in app. Beers being selected by abv.

Unit	Ref	Evidence
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing
		Description:

Paste Screenshot here

```ruby
class CardGame

  def initialize(suit, value)
    @suit = suit
    @value = value
  end

```

```
  1) Error:
CardGameTest#test_CardGame_has_suit:
NoMethodError: undefined method `suit' for #<CardGame:0x007fd01105f738 @suit="clubs",
alue=2>
    specs/task_b_spec.rb:12:in `test_CardGame_has_suit'

1 runs, 0 assertions, 0 failures, 1 errors, 0 skips
➜  Static_and_Dynamic_Task_A git:(master) ✗
```

```ruby
class CardGame

  attr_reader :suit, :value

  def initialize(suit, value)
    @suit = suit
    @value = value
  end
```

```ruby
class CardGameTest < MiniTest::Test

  def setup
    @card = Card.new("clubs", 9)
  end

  def test_CardGame_has_suit
    assert_equal("clubs", @card.suit())
  end

  def test_CardGame_has_value
    assert_equal(9, @card.value())
  end

end
```

```
Run options: --seed 64121

# Running:

.

Finished in 0.000880s, 1136.3636 runs/s, 1136.3636 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
➜  Static_and_Dynamic_Task_A git:(master) ✗
```

Description here

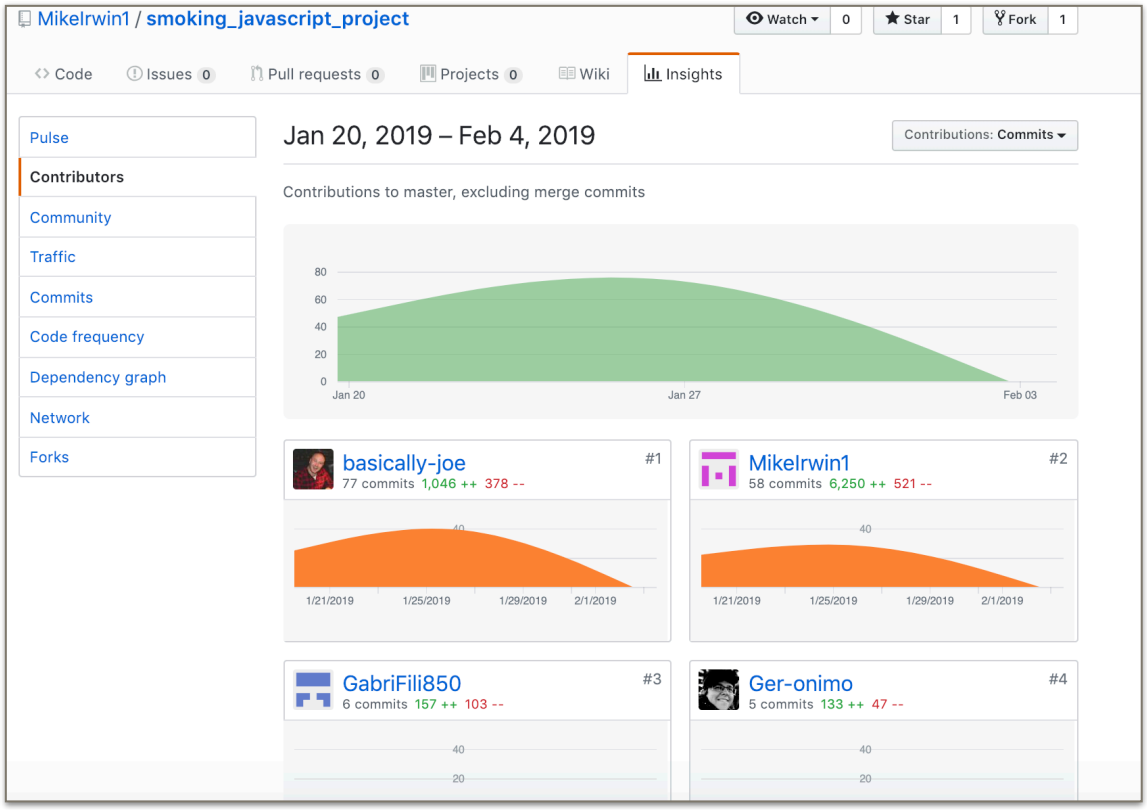
- **Screenshot 1:** Missing attr_reader accessors for **suit** and value
- **Screenshot 2:** Test failing - undefined method **suit**
- **Screenshot 3:** attr_reader added for **suit** and value
- **Screenshot 4:** Test code
- **Screenshot 5:** Test passing for **suit**

  *same process followed to test **value**

Week 9

Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.
		Description:

Paste Screenshot here

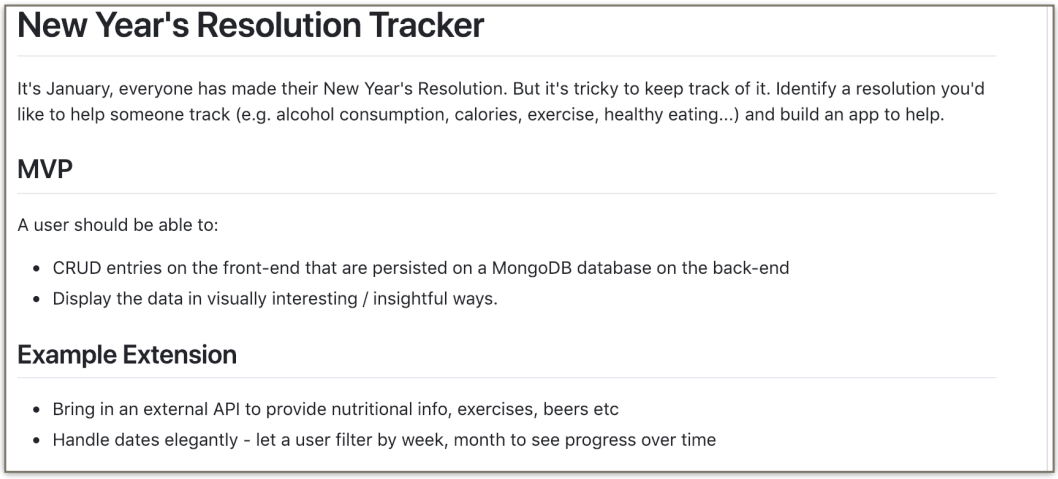


Description here

- **Screenshot:** 4 contributors

Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.
		Description:

Paste Screenshot here



New Year's Resolution Tracker

It's January, everyone has made their New Year's Resolution. But it's tricky to keep track of it. Identify a resolution you'd like to help someone track (e.g. alcohol consumption, calories, exercise, healthy eating...) and build an app to help.

MVP

A user should be able to:

- CRUD entries on the front-end that are persisted on a MongoDB database on the back-end
- Display the data in visually interesting / insightful ways.

Example Extension

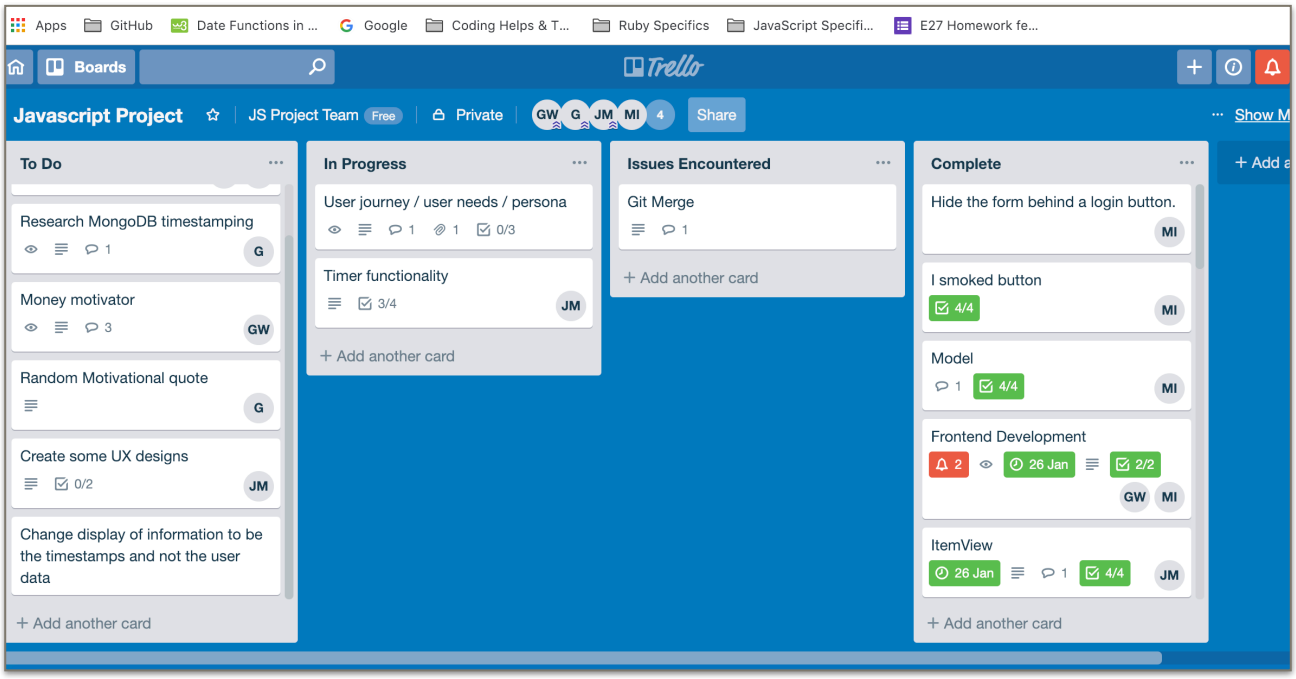
- Bring in an external API to provide nutritional info, exercises, beers etc
- Handle dates elegantly - let a user filter by week, month to see progress over time

Description here

- **Screenshot:** the group chose a New Years resolution app which would help people to stop smoking

Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.
		Description:

Paste Screenshot here



Description here

- **Screenshot:** Trello board shared with 3 others

Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

Paste Screenshot here

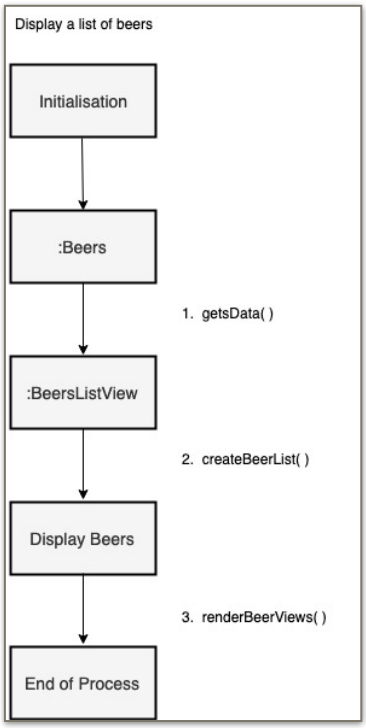
Acceptance Criteria	Expected Result/Output	Pass/Fail
A user is able to…		
Click a button and see the amount of money saved by not smoking	The accumulated saving does display when the savings button is clicked	Pass
Click a button and see the history by date of all individual cigarettes smoked since stopping smoking	The history of individual cigarettes smoked does display by date when the button is clicked	Pass
See the amount of money saved as a real time accumulation	The full running total of savings is displayed on the page when the user logs in	Fail
See the amount of time passed between cigarettes smoked	The time passed is displayed as a running total when the user logs in	Pass
Click a button when they smoke to reset the running timer to zero	The full running total will reset to zero	Pass

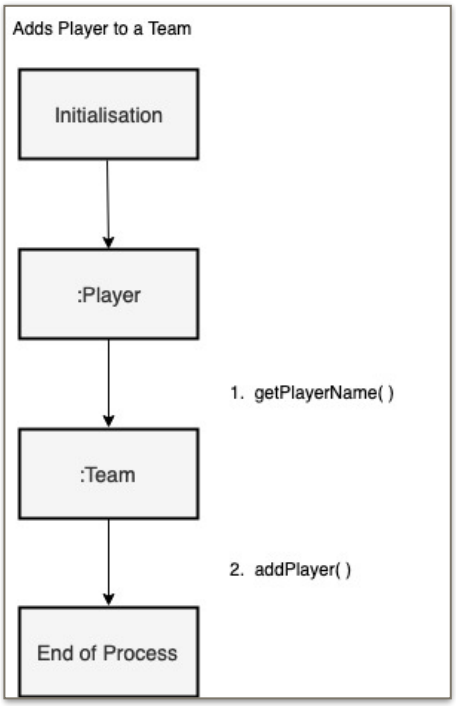
Description here

- **Screenshot:** Criteria for features of an app

Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).
		Description:

Paste Screenshot here



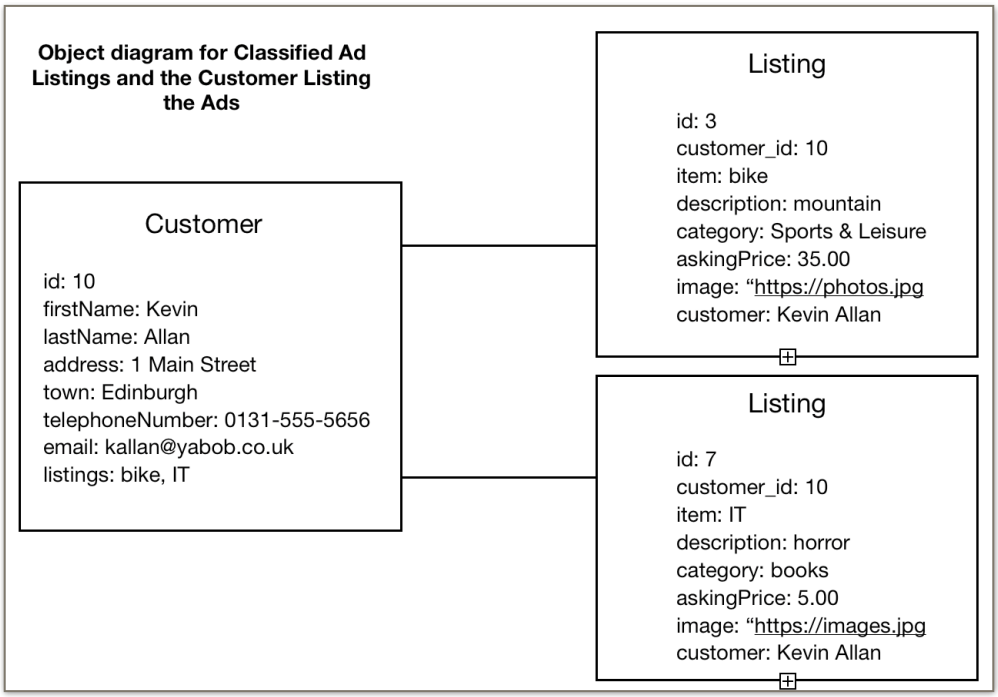


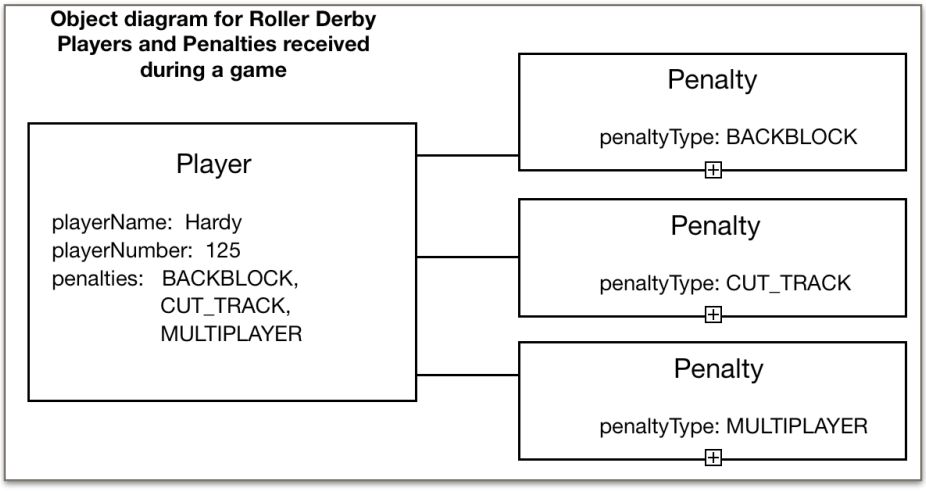
Description here

Screenshot 1: shows sequence for the rendering of a list of beers
Screenshot 2: gets a player object and adds to a list of players in a team

Unit	Ref	Evidence
P	P.8	Produce two object diagrams.
		Description:

Paste Screenshot here





Description here

- **Screenshot 1:** Object diagrams for classified add listings
  - Customer details - primary key
  - Listing details - primary key and foreign key

- **Screenshot 2:** Object diagrams for players and penalties

Unit	Ref	Evidence
P	P.17	Produce a bug tracking report
		Description:

Paste Screenshot here

Bug/Error	Solution	Date
Method undefined on dailySavingsCalculator	Corrected method naming error (spelling mistake) from where it was being called from	26/01/2019
Data value not being returned	Data not being called on variable in the constructor, outside the scope of the code block. Create new local variable inside the method.	28/01/2019
Timer counter displaying twice	Timer button - timer container refactor to check if the element container exists first if it doesn't exist create the timer.	29/01/2019
Multiple clicks registered to hide or show an element.	Call the hide function higher in the bind events	29/01/2019
I smoked button	Timer container refactored with an if statement	30/01/2019

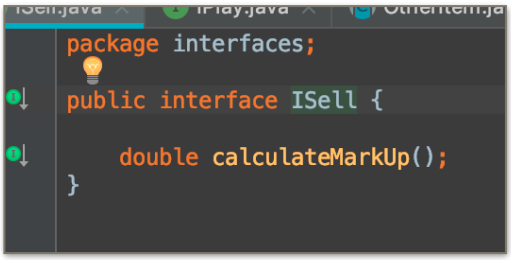
Description here

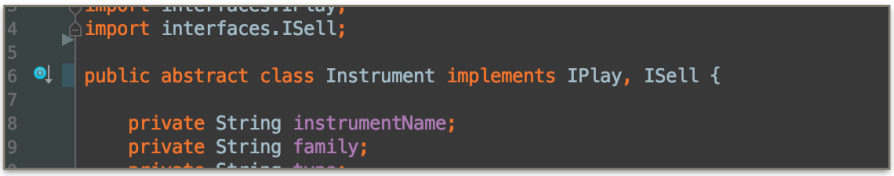
- **Screenshot:** bugs and errors resolved using debugger and console log

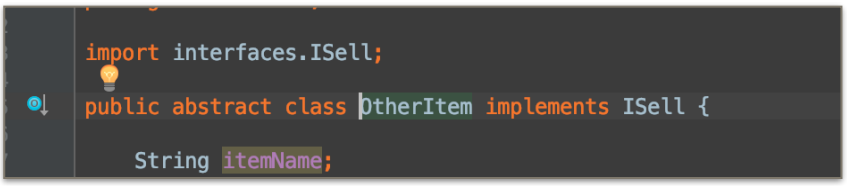
Week 12

Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description:

Paste Screenshot here





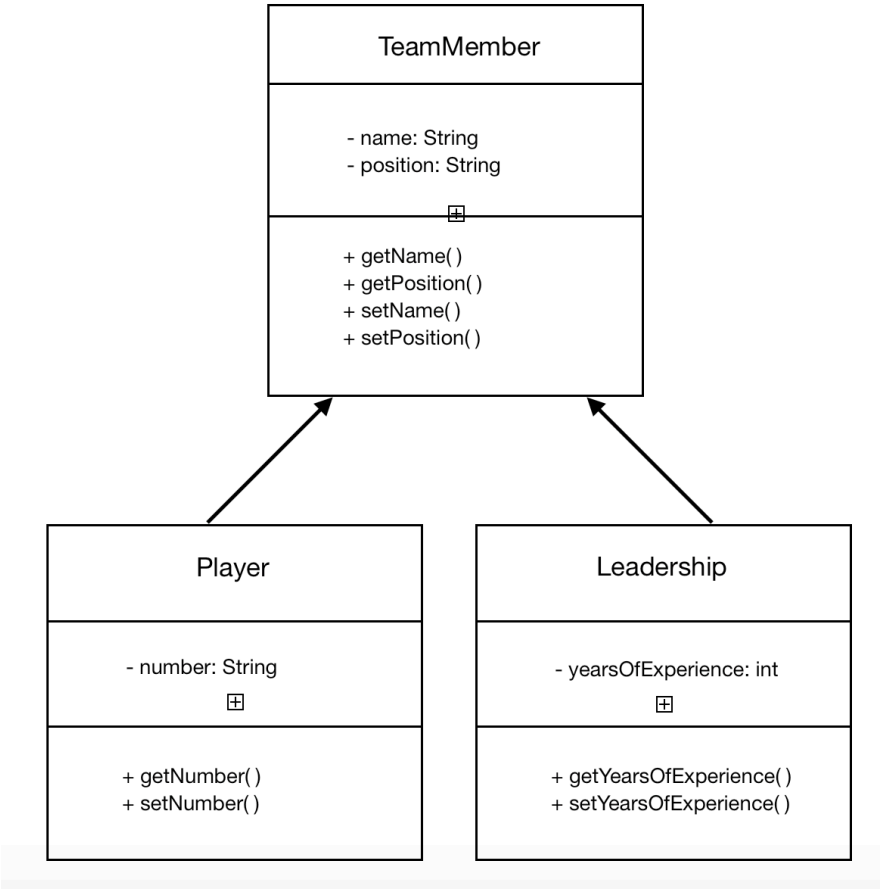


Description here

- **Screenshot 1:** iSell interface has method calculateMarkUp( )
- **Screenshot 2:** iSell implemented in Instrument class
- **Screenshot 3:** iSell implemented in OtherItem class
- **Screenshot 4:** Imported into Shop class and used in a list of instruments and other items that are sellable

Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram
		Description:

Paste Screenshot here

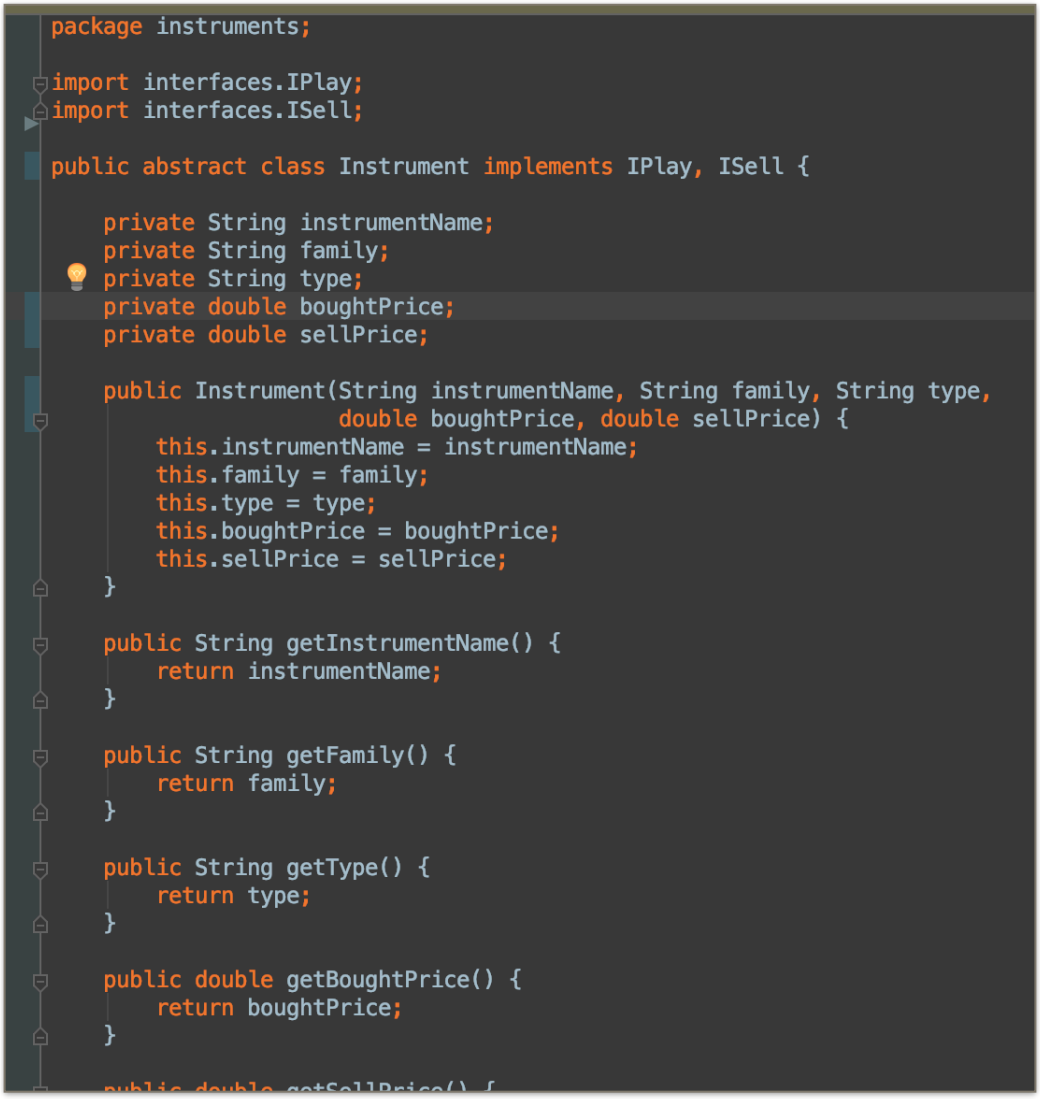


Description here

- **Screenshot:** an inheritance diagram showing members of a ‘team’. TeamMember class has the inherited attributes and behaviours.

Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description:

Paste Screenshot here



```
package instruments;

import interfaces.IPlay;
import interfaces.ISell;

public abstract class Instrument implements IPlay, ISell {

    private String instrumentName;
    private String family;
    private String type;
    private double boughtPrice;
    private double sellPrice;

    public Instrument(String instrumentName, String family, String type,
                      double boughtPrice, double sellPrice) {
        this.instrumentName = instrumentName;
        this.family = family;
        this.type = type;
        this.boughtPrice = boughtPrice;
        this.sellPrice = sellPrice;
    }

    public String getInstrumentName() {
        return instrumentName;
    }

    public String getFamily() {
        return family;
    }

    public String getType() {
        return type;
    }

    public double getBoughtPrice() {
        return boughtPrice;
    }
```

Description here

- **Screenshot:** encapsulation demonstrating property access modifiers being declared as private so they can only be accessed from within the class by the methods. Method access modifiers are declared here as public, so they can be accessed from outside the class by related classes.
- e.g. **private String instrumentName** can be accessed within **Instrument** class by the method **public String getInstrumentName( ) ,** this can be accessed by a subclass.

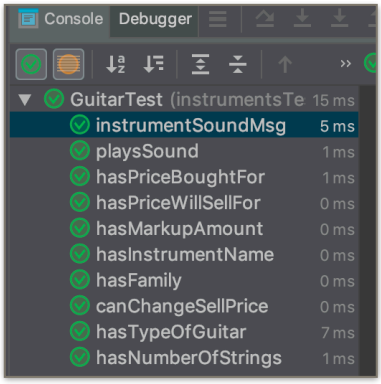
Unit	Ref	Evidence
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.
		Description:

Paste Screenshot here

```java
public abstract class Instrument implements IPlay, ISell {

    private String instrumentName;
    private String family;
    private String type;
    private double boughtPrice;
    private double sellPrice;

    public Instrument(String instrumentName, String family, String type,
                      double boughtPrice, double sellPrice) {
        this.instrumentName = instrumentName;
        this.family = family;
        this.type = type;
        this.boughtPrice = boughtPrice;
        this.sellPrice = sellPrice;
    }

    public String getInstrumentName() {
        return instrumentName;
    }

    public String instrumentSoundMsg(){
        return "The sound this " + this.instrumentName + " makes is ";
    }
```

```java
public class Guitar extends Instrument{

    private int numberOfStrings;

    public Guitar(String instrumentName, String family, String type, double b
        super(instrumentName, family, type, boughtPrice, sellPrice);
        this.numberOfStrings = numberOfStrings;

    }

    @Override
    public String instrumentSoundMsg(){
        return super.instrumentSoundMsg() + soundPlayed();
    }
// directly inherits from Instrument class

    public int getNumberOfStrings() {
        return numberOfStrings;
    }
```

```java
public class GuitarTest {

    Guitar guitar;

    @Before
    public void before(){
        guitar = new Guitar( instrumentName: "Guitar", family: "String",
                type: "Electric", boughtPrice: 50, sellPrice: 100, numberOfStrings: 6);
    }
```



Description here

- **Screenshot 1:** Instrument class
- **Screenshot 2:**
  - Guitar class inherits from Instrument class
  - Method instrumentSoundMsg( ) Overrides same method from Instrument class and uses the information
- **Screenshot 3:** Guitar guitar object created in test
- **Screenshot 4:** All tests passed on inherited information in Guitar object

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.
		Description:

Paste Screenshot here

```
<% @animals.each do |animal| %>
  <tr>
    <td><img src="/images/adoption_animals/<%= animal.name %>.jpg"
             width="75" height="56" alt="">  </td>
    <td><h4><a href="/animals/<%= animal.id %>">
             <%= animal.name.capitalize %></a></h4></td>
    <td><%= animal.type %></td>
    <td><%= animal.admission_date %></td>
    <td><% if animal.ready_to_adopt == 't'%>
          <h5>Adoptable</h5>
        <% elsif animal.ready_to_adopt == 'f' && animal.adopted == 't' %>
          <h5>Adopted</h5>
          <%else%>
          <h5>Not ready for adoption</h5>
        <% end %></td>
  </tr>
    <% end %>
 </table>
/div>
```

```
    public void addBook(Book book) {
        if (getBookCount() < this.capacity) {
            this.books.add(book);
        }
    }
```

Description here

- **Screenshot 1:** The logic in this algorithm works out the adoption status of an animal and outputs to the page. The challenge was getting the criteria in the correct order. The result of the algorithm saves to the database and then determines which animals are displayed on a separate *Adoptable Animals* page.
- **Screenshot 2:** adds a book object to a library array of books if the book count is less than the capacity of books that can be held.