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.

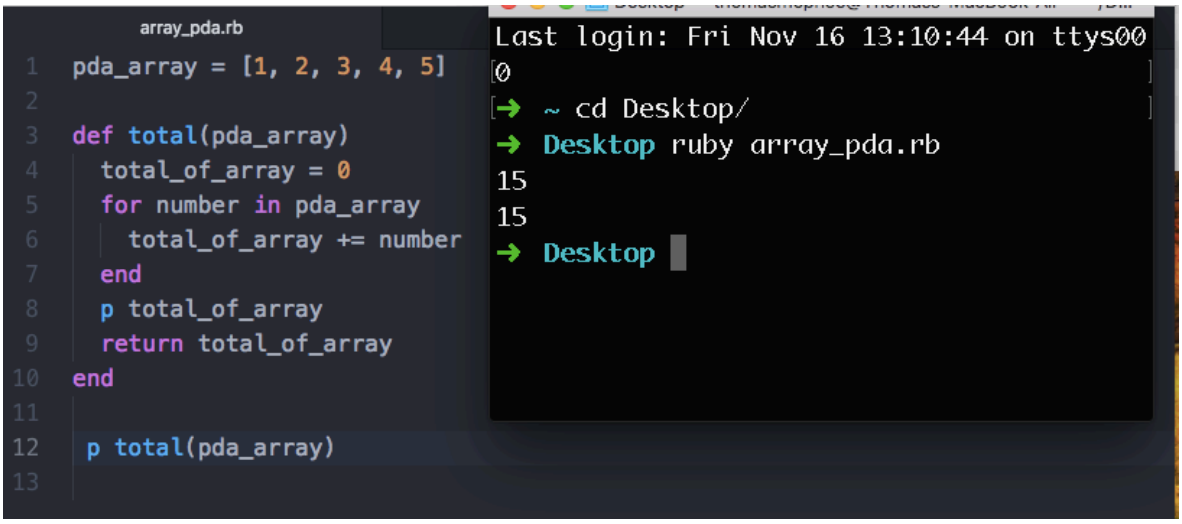
Please fill in each point with screenshot or diagram and description of what you are showing.

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:<br>*An array in a program<br>*A function that uses the array<br>*The result of the function running |
| | | **Description:** |

**Paste Screenshot here**

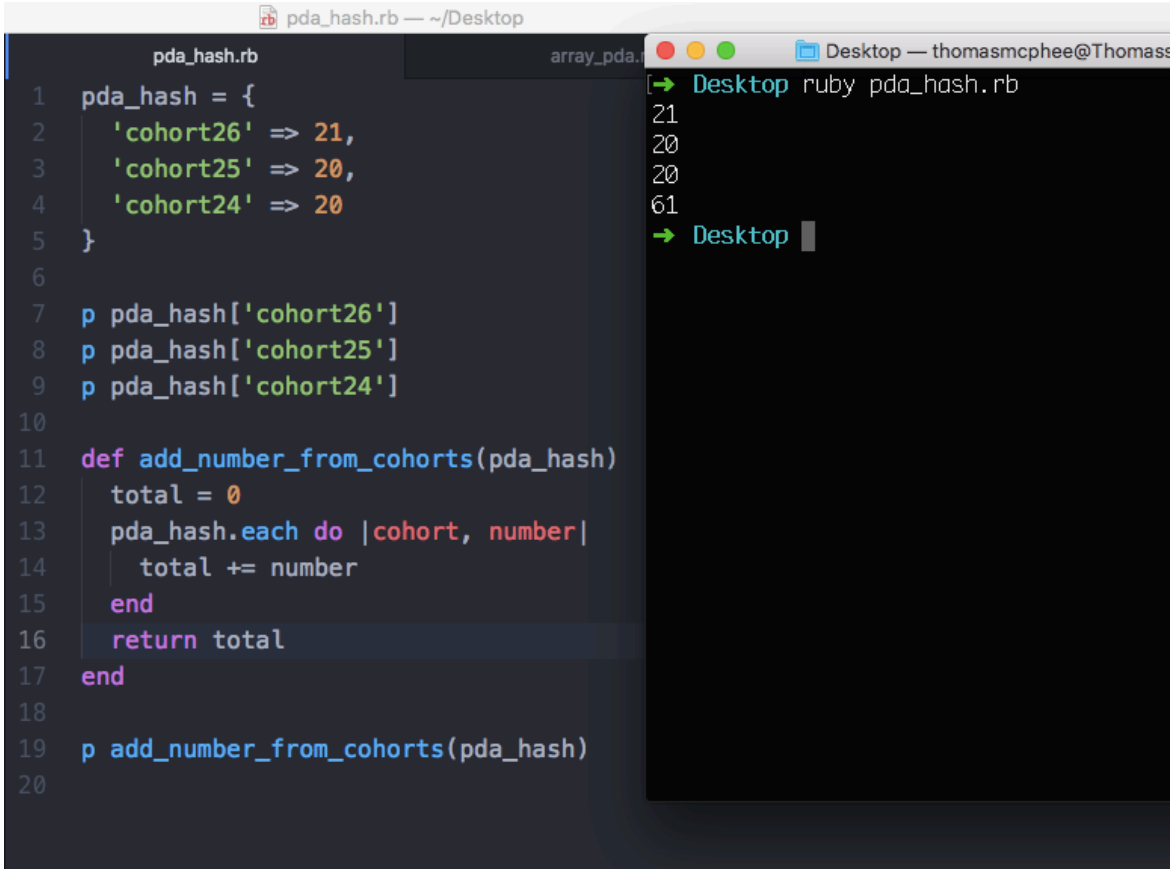



**Description here**

This is a use of an array I have created. I wanted to add all the values of items in the array by looping through each value and adding it to a variable. The result should be 15 and it is.

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

**Paste Screenshot here**




**Description here**

Above I have written a hash called pda_hash. I have first asked to print all values of each key. This is shown by '21', '20' and '20' appearing in the terminal. I have then written a function to find the total of each key value pair. I have used the .each method and it returns and prints out the total of 61.

**Week 3**

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

**Paste Screenshot here**

```ruby
def self.all()
  sql = "SELECT * FROM animals"
  values = []
  result = SqlRunner.run(sql, values)
  return result.map { |animal| Animal.new(animal)}
end
```

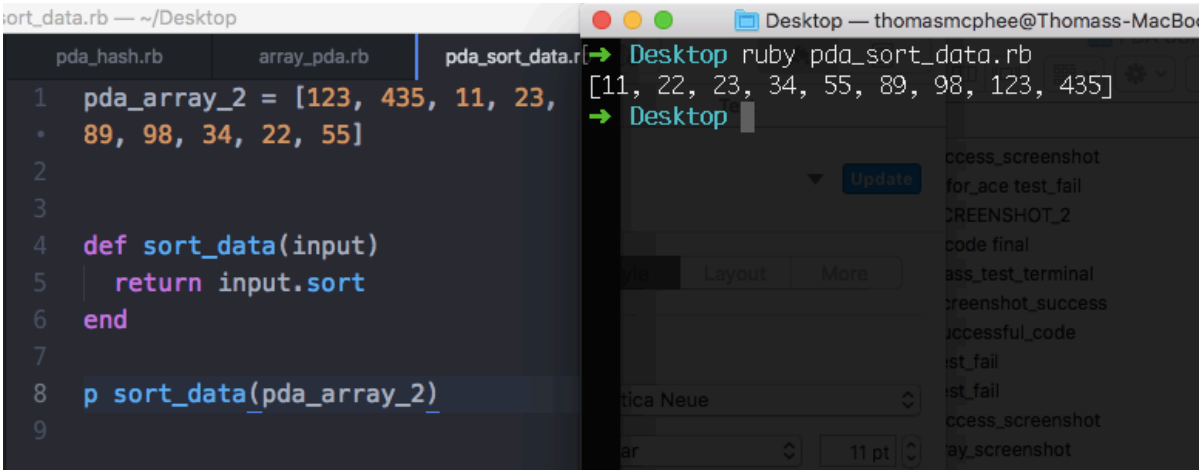
```
animal_shelter — ruby db/seeds.
[3] pry(main)> Animal.all
=> [#<Animal:0x007fa84b1fb4e0
  @adoption_status="Adoptable",
  @age="2",
  @breed="Husky",
  @day_admitted="2018-08-14",
  @id=41,
  @name="David Tennant",
  @owner_id="31",
  @type="Dog">,
 #<Animal:0x007fa84b1fb2b0
  @adoption_status="Adoptable",
  @age="10",
  @breed="Bald One",
  @day_admitted="2008-02-15",
  @id=42,
  @name="Christopher Eccelston",
  @owner_id="31",
  @type="Cat">,
 #<Animal:0x007fa84b1fae28
  @adoption_status="Not Adoptable",
  @age="14",
  @breed="Ginger One",
  @day_admitted="2004-02-15",
```

**Description here**

Above I have posted a class function called ‘self.all’ This injects a sql query from Ruby into an sql database. This query asks for all information in a table called animals and returns a view of all the information (via result.map). I viewed the result of this function using pry.

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

**Paste Screenshot her**



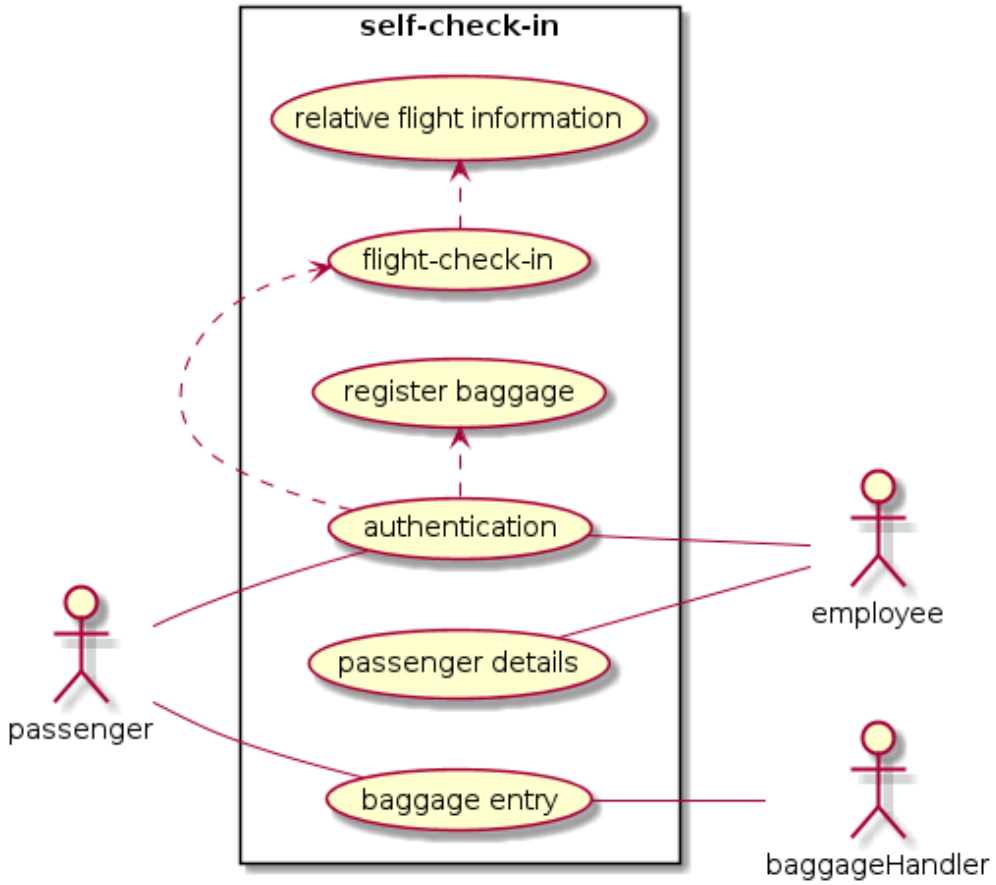

**Description here**

I have created an unsorted array called pda_array_2 and I have called a function called ‘sort_data’ which uses a ruby method .sort to return all values in ascending order.

**Week 5 and 6**

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

**Paste Screenshot here**

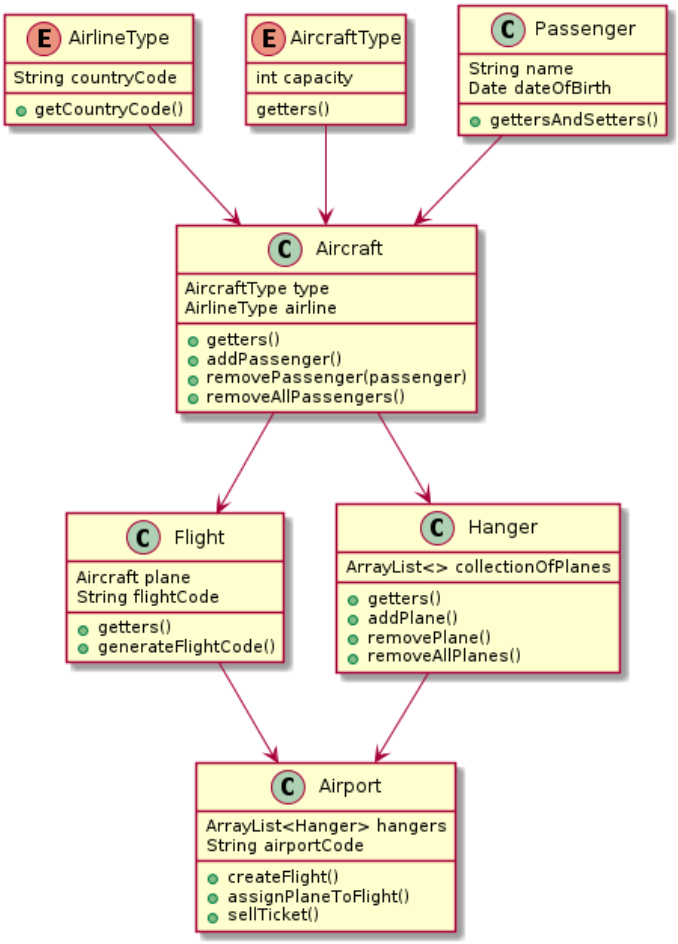



**Description here**

This is a simple case diagram for a. Passenger checking in for a flight.
The passenger has the ability to register baggage, check in to their flight and see flight information but ONLY through authentication. The employee should be able to access authentication incase passenger needs help. Employee should also be able to access current passenger info but a passenger should NOT be able to.

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

**Paste Screenshot here**

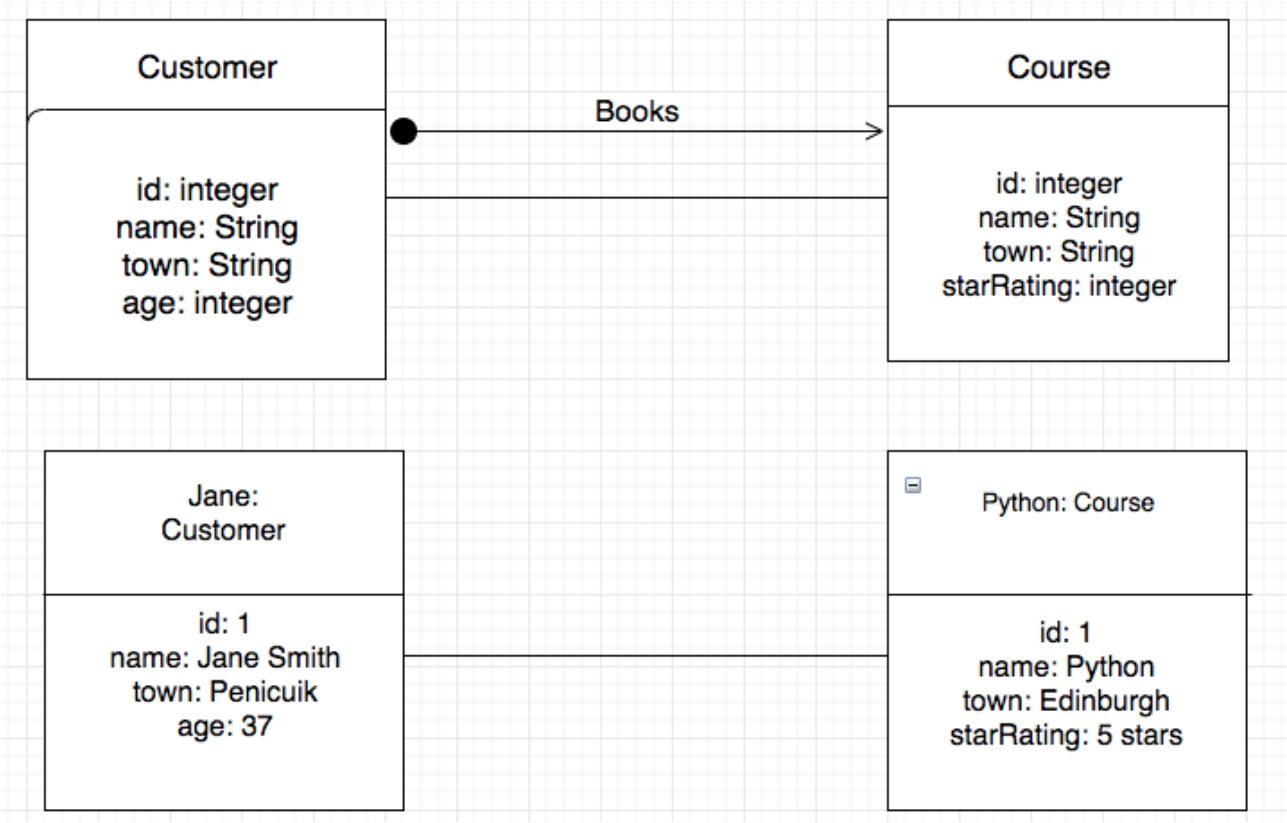



**Description here**

This is a class diagram of a program to model an airport.
The airport would need to be able to create Flights, assign and aircraft to flights, sell tickets and potentially others. This would require an aircraft, passenger, flight and hanger classes to be created. An aircraft should have a collection of passengers. A hanger should have a collection of aircraft, a flight should be assigned an aircraft. And these should interact with Airport.

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

**Paste Screenshot here**

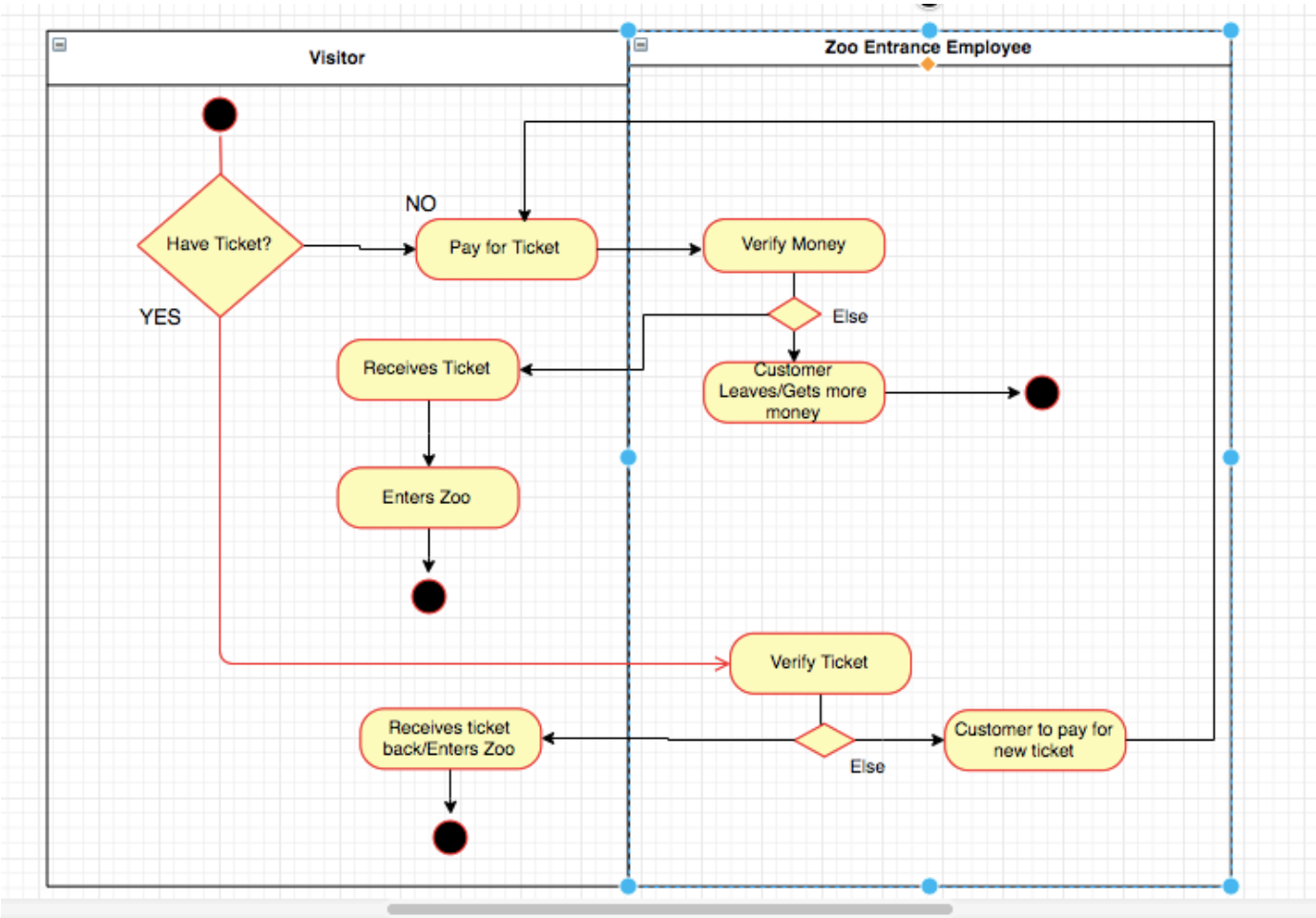



**Description here**

This is an object diagram showing the which shows the class of the object with its attributes. It also shows an instance of the class with precise attributes. The customer books the course.

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

**Paste Screenshot here**




**Description here**

The above diagram shows the flow when a visitor wishes to enter a Zoo. The visitor decides if they have a ticket or not and then follows a series of actions based on that. If the visitor does not have a ticket, the visitor will purchase one from the employee. If it is the correct amount the visitor will receive a ticket and enter the park. If they do have a ticket already, the employee will verify the ticket and, if acceptable, the visitor will enter the park.

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

**Paste Screenshot here**

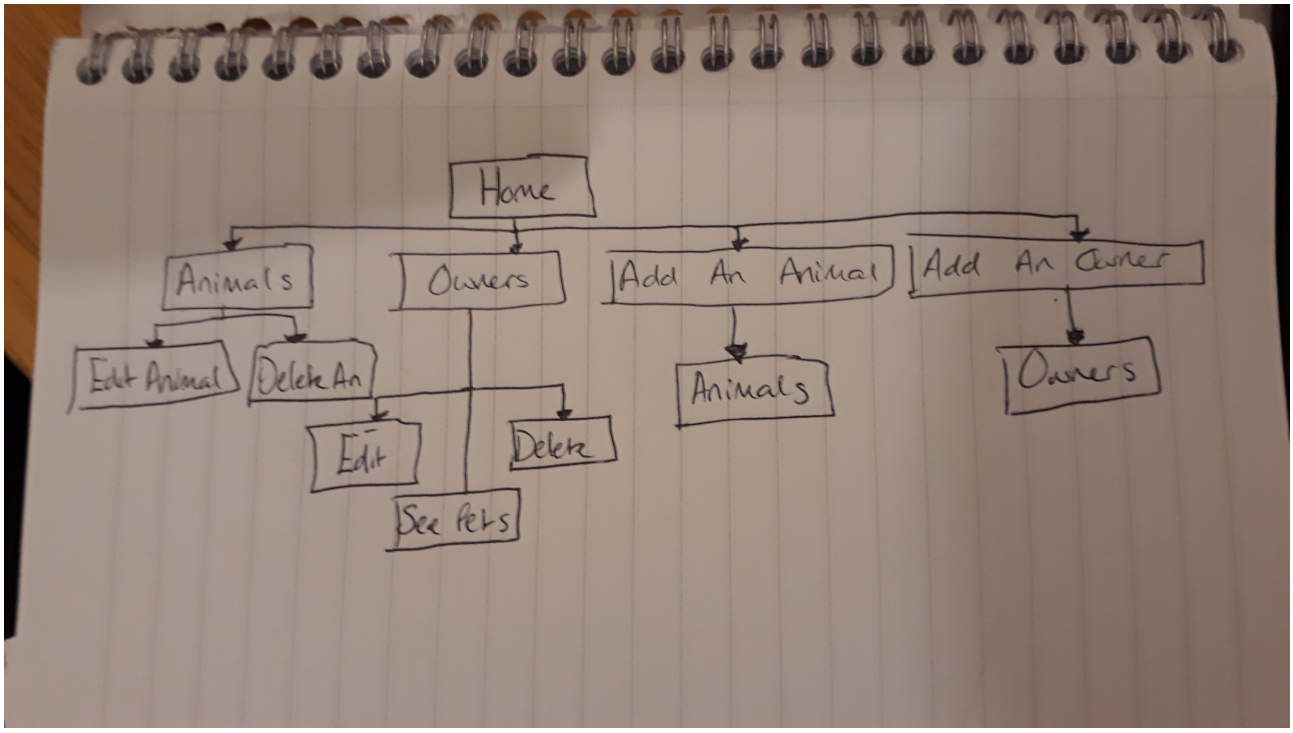
| ***Constraint Category*** | ***Implementation Constraint*** | ***Solution*** |
|---|---|---|
| **Hardware and software platforms** | The application will be built on Mac OS. This could cause issues for users on different systems. | Host application on an external server (a service such as Heroku) or use a containerised deployment service such as AWS or Docker |
| **Performance requirements** | Makes regular requests to a server. If server goes down then users will not be provided with any news data for application. | A local script that is called if API fails at any point. This would provide user with the last loaded data from API until server issues are resolved. |
| **Persistant storage and transactions** | Data for news articles will be stored in a local database. Similar to the problem above, a user would be impacted if data is lost or corrupted. | Have back-up systems in place to hold data. These can be used in the event of database breakdown. |
| **Usability** | The application is for a user to edit and place articles written by journalists. The app needs to be clear and responsive for a fast paced environment. | Make interactive elements bright (bold primary colours). Use Material UI which imports elements already written by Google employees. These are responsive elements which are already used throughout various websites. |
| **Budget** | Low budget. No room for reactive editing and reworking. | Extensive UX research. A clear **User Persona** which shows very clearly who we are building for.<br>**user stories.** Define precisely what a user wants to happen when using the app. Define their expectations clearly |

**Description here**

The above shows my constraints plan for my final project. The project was an application for controlling news articles.

| Unit | Ref | Evidence |
|---|---|---|
| **P** | P.5 | User Site Map |
| | | **Description:** |

**Paste Screenshot here**

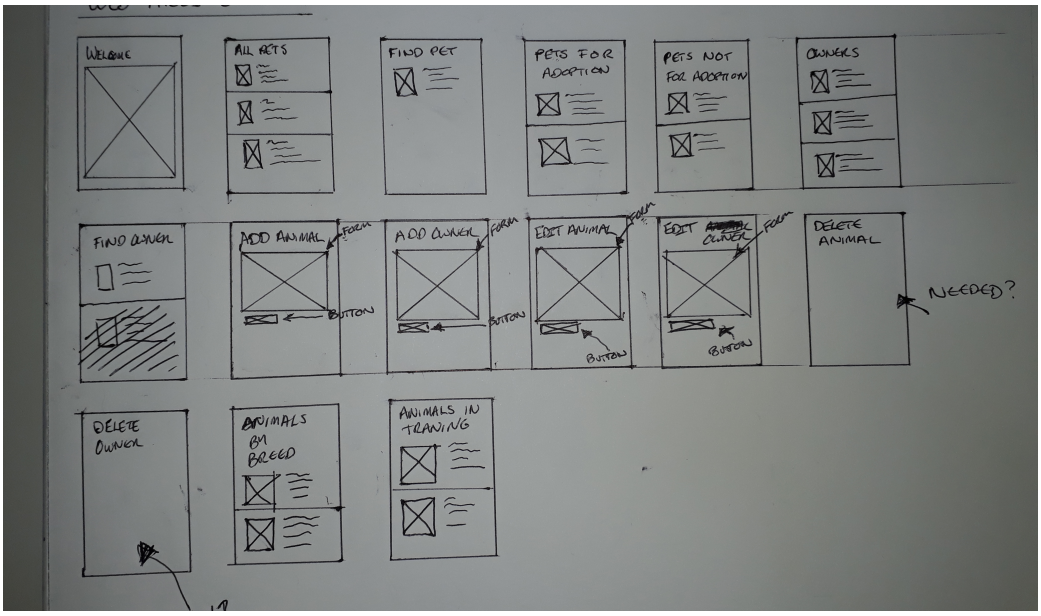



**Description here**

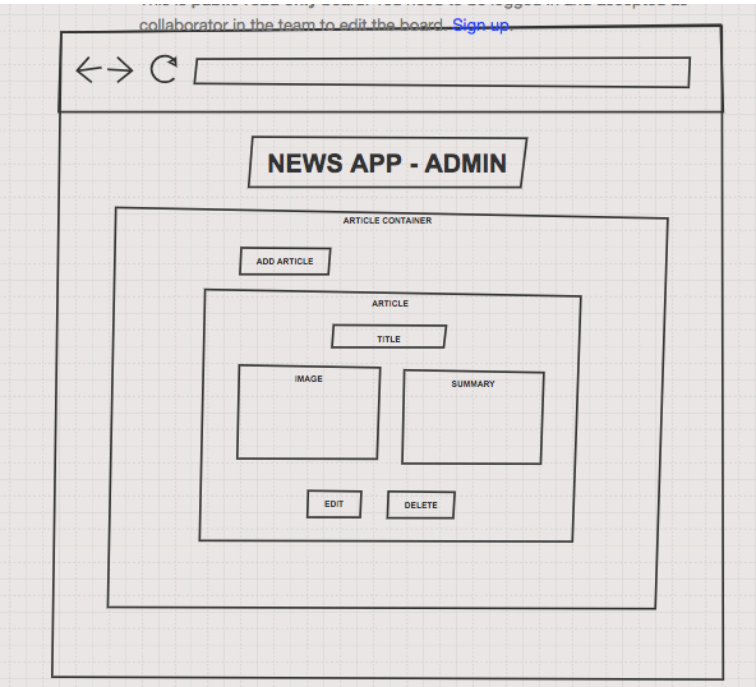
This is a site map of my solo Ruby project for a Scottish Animal Shelter. The user begins at the homepage and can navigate to see animals, owners and access forms to add animals and owners to the database. The user has the option to edit and delete owners on their respective pages.

| Unit | Ref | Evidence |
|---|---|---|
| **P** | P.6 | 2 Wireframe Diagrams |
| | | **Description:** |

**Paste Screenshot here**




**ANIMAL SHELTER WIREFRAME**




**Description here**

The top diagram is wireframes drawn up for my Ruby project. These are a series of pages and their layout which I wanted to implement.
The second image is that of my final project and a general view of how we wanted our application to look.

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

**Paste Screenshot here**

pseudocode.js

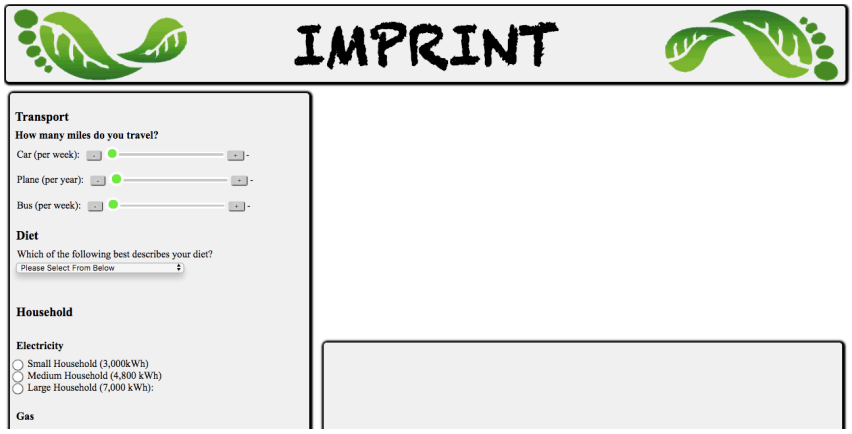
```
pseudocodeIncrementer(){
  // arrayfrom 1 - 10
  // starting count of 0

  // start for loop over array
  // add each array value to count
  // finish loop

  // return count
}
```

**Description here**

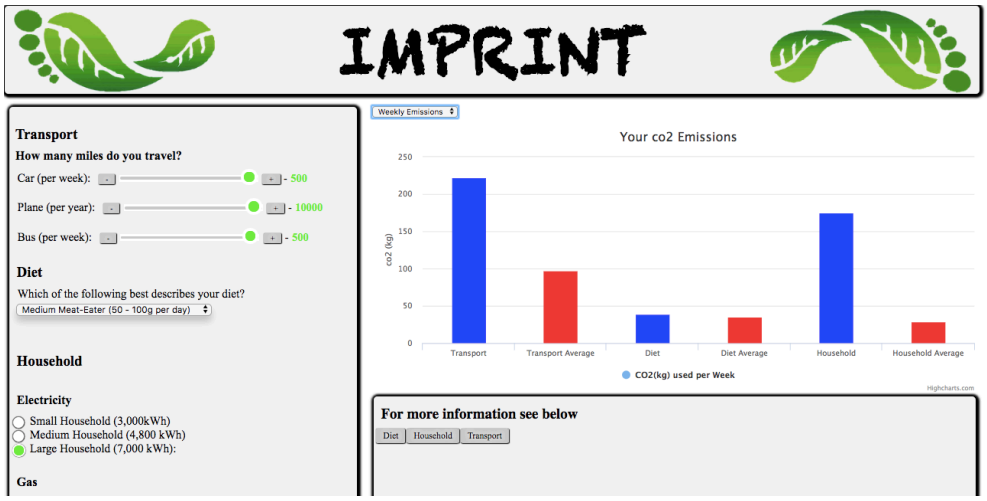
This example is pseudocode for looping through an array and returning a value.
An array from 1-1- would be created, looped over and then returning a sum of value.

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

**Paste Screenshot here**




**Before user input**



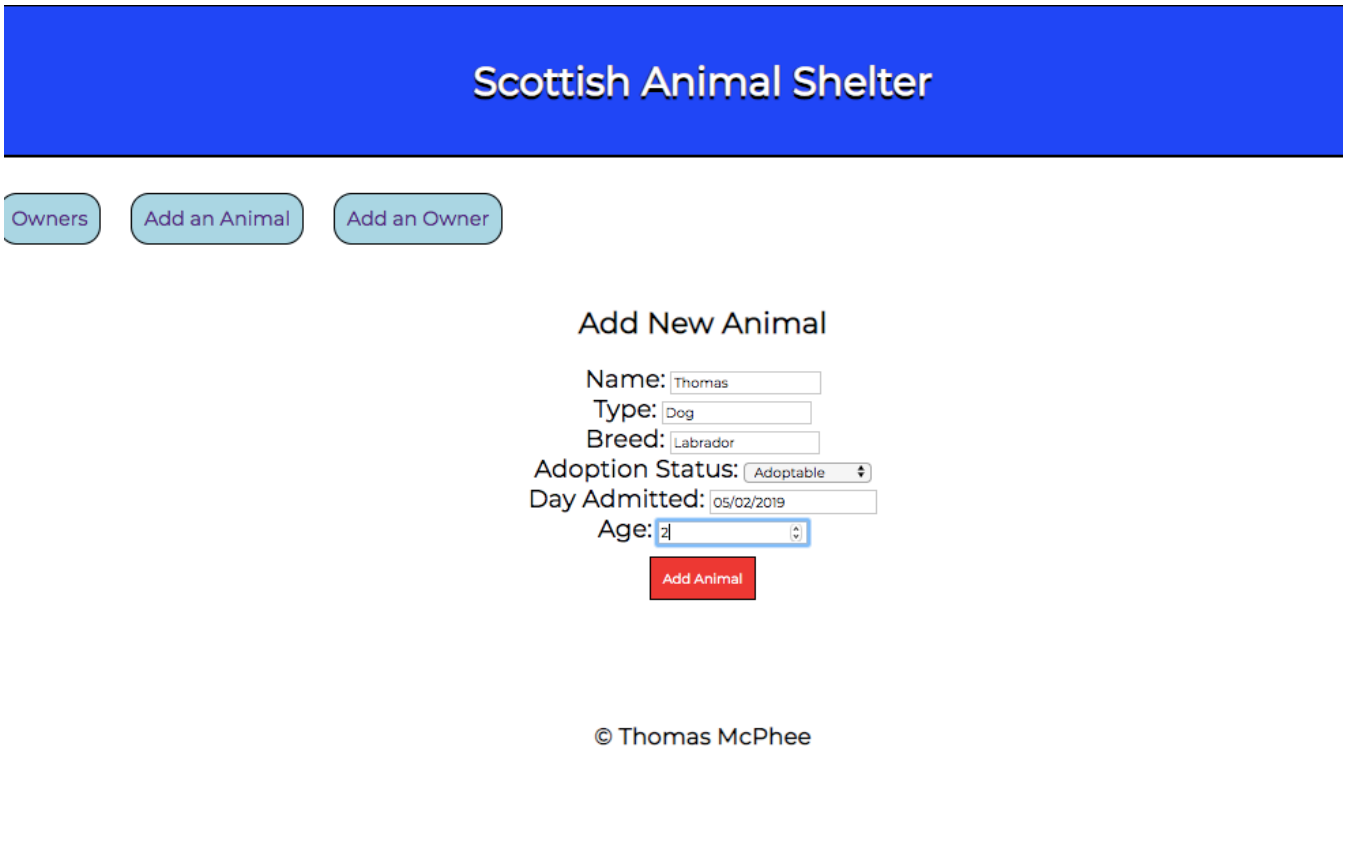

**After user input**

**Description here**

The above shows an application that allows a user to input data into a form to see their CO2 Emissions. The first picture shows the application before the user has inputted any data. The second shows how the application uses that data for a visual response.

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

**Paste Screenshot here**




Name: Thomas

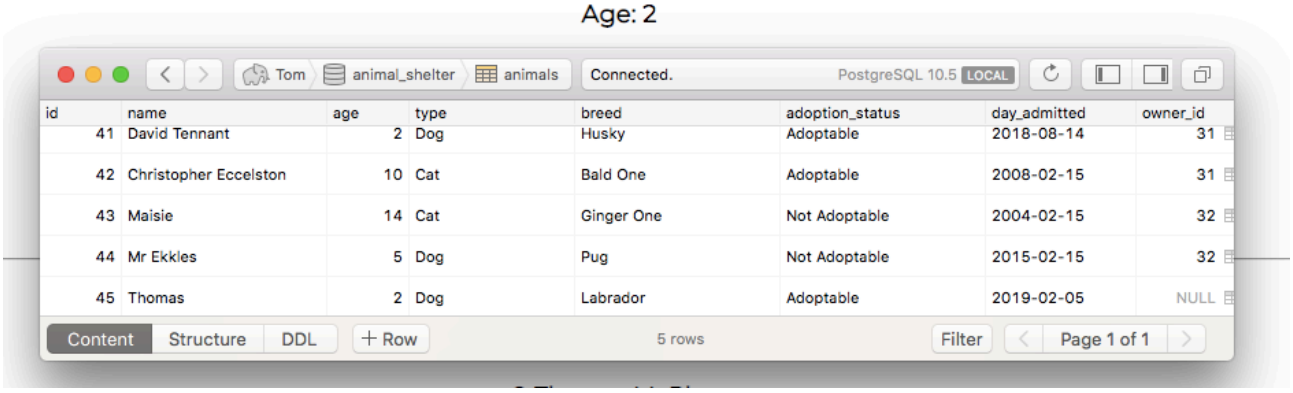
Type: Dog

Breed: Labrador

Adoption Status: Adoptable

Day Admitted: 2019-02-05

Age: 2



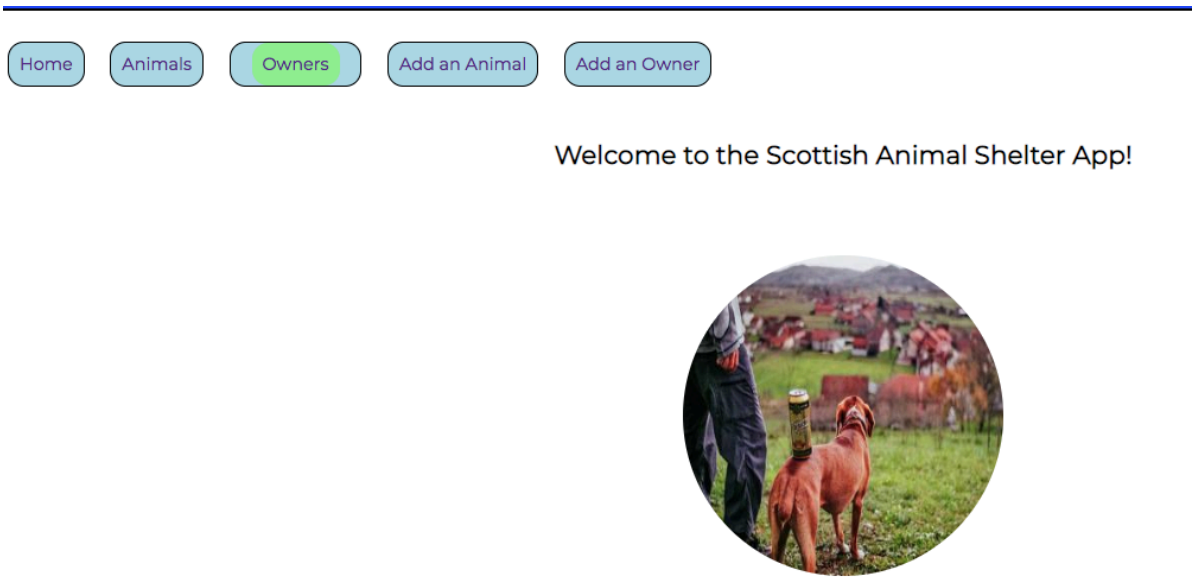

| id | name | age | type | breed | adoption_status | day_admitted | owner_id |
|---|---|---|---|---|---|---|---|
| 41 | David Tennant | 2 | Dog | Husky | Adoptable | 2018-08-14 | 31 |
| 42 | Christopher Eccelston | 10 | Cat | Bald One | Adoptable | 2008-02-15 | 31 |
| 43 | Maisie | 14 | Cat | Ginger One | Not Adoptable | 2004-02-15 | 32 |
| 44 | Mr Ekkles | 5 | Dog | Pug | Not Adoptable | 2015-02-15 | 32 |
| 45 | Thomas | 2 | Dog | Labrador | Adoptable | 2019-02-05 | NULL |

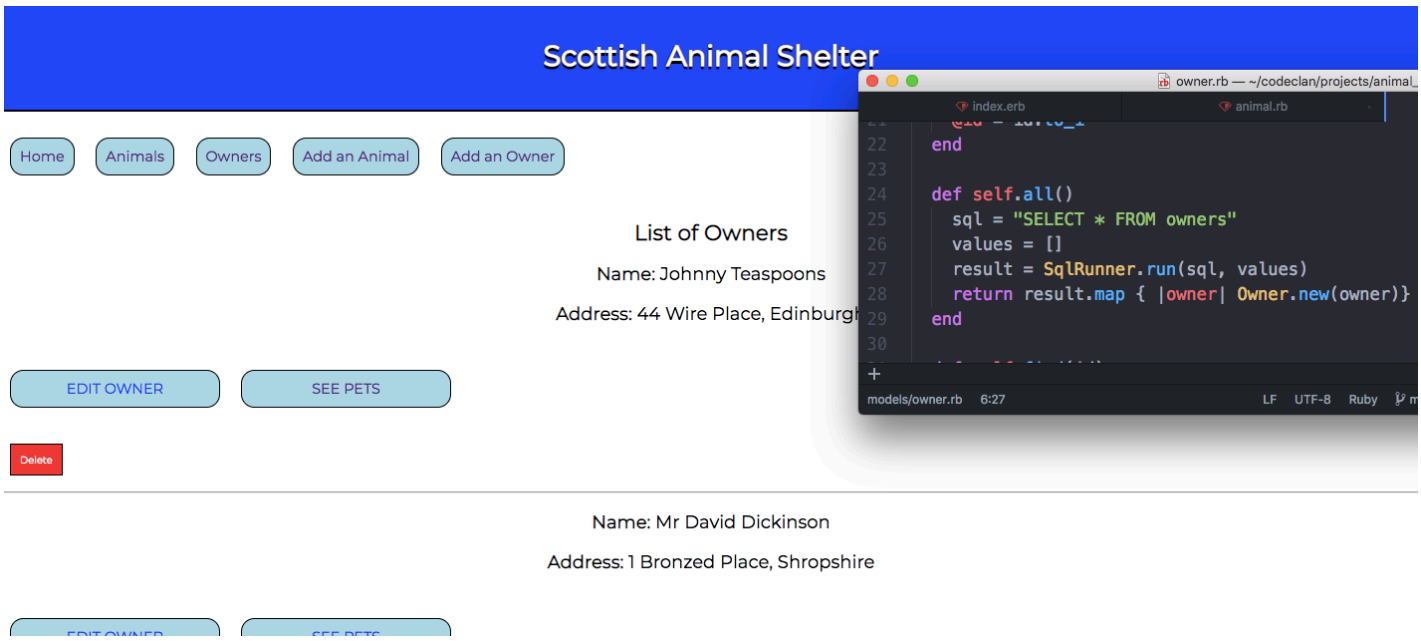
**Description here**

The two screenshots above show an animal being added using a website form. The second shows that data being confirmed within the website but also the data being added to the database in Postico.

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

**Paste Screenshot here**



**Description here**

The top screenshot shows the user highlighting the ‘owners’ button.
The second shot shows the request being processed with the code request on the left and the results show on the page.

| 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**

**RUBY SOLO PROJECT**

**Description here**

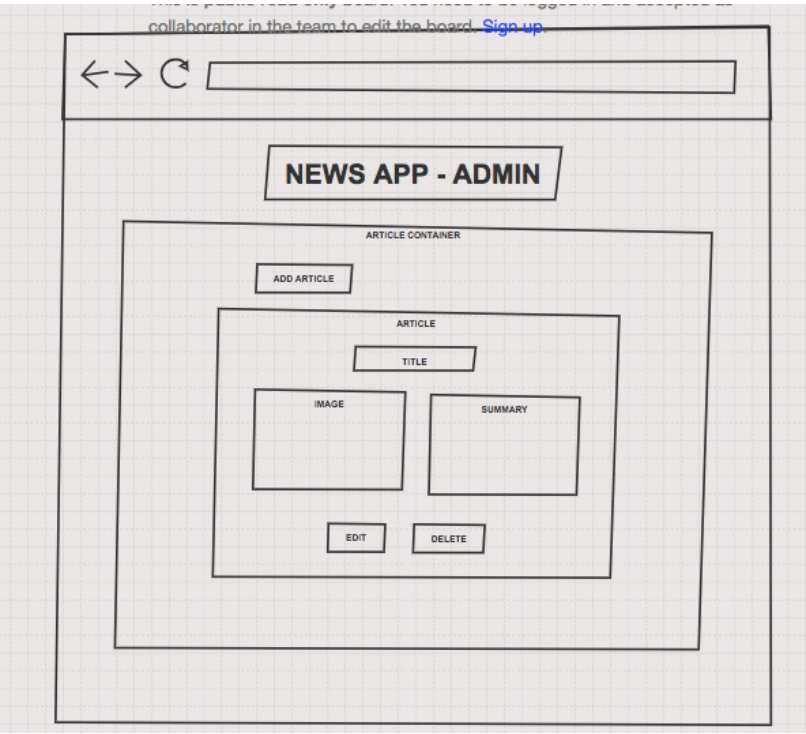
This was the week 5 project using Ruby, SQL, HTML and CSS. My brief described an Animal Shelter that had he ability to add/remove/update animals into a database using RESTFUL Routes. And the same for Animal Owners. The app had the ability to show any owners and their adopted animals though SQL.

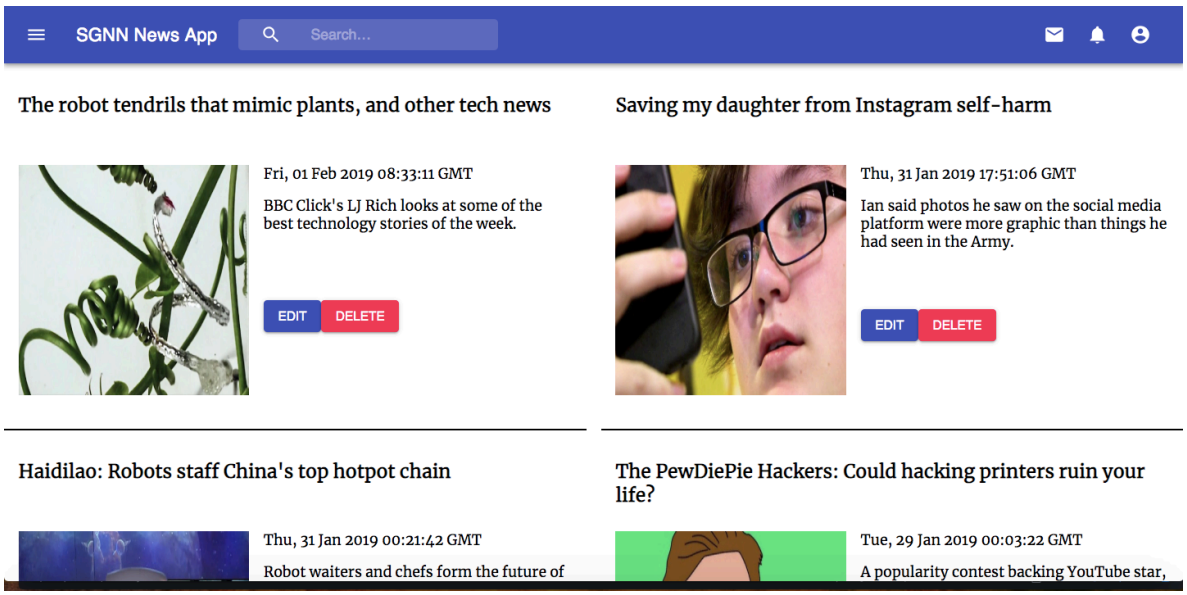
https://github.com/ThomasMcP/Animal-Shelter-Application

| 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**

| As a | i want to | so that |
|---|---|---|
| News consumer | be able to see a list of articles | i can catch up on the news |
| News consumer | Be able to search for keywords | I can find specific stories |
| News consumer | Be able to see related articles (those with same keyword) | So I can find related stories |
| News outlet | Be able to add articles | I can update the news database |
| News outlet | Be able to add journalists | I can track who has written each article |
| News outlet | Be able to edit articles | I can fix typos |
| News outlet | Be able to delete articles | We can keep our lawyers happy |
| News outlet | Be able to search for keywords | I can find specific stories |
| News outlet | Be able to order articles | I can display them in a specific order |






**Description here**

These pictures show some of the progress that happened during my Java/Spring and React project.

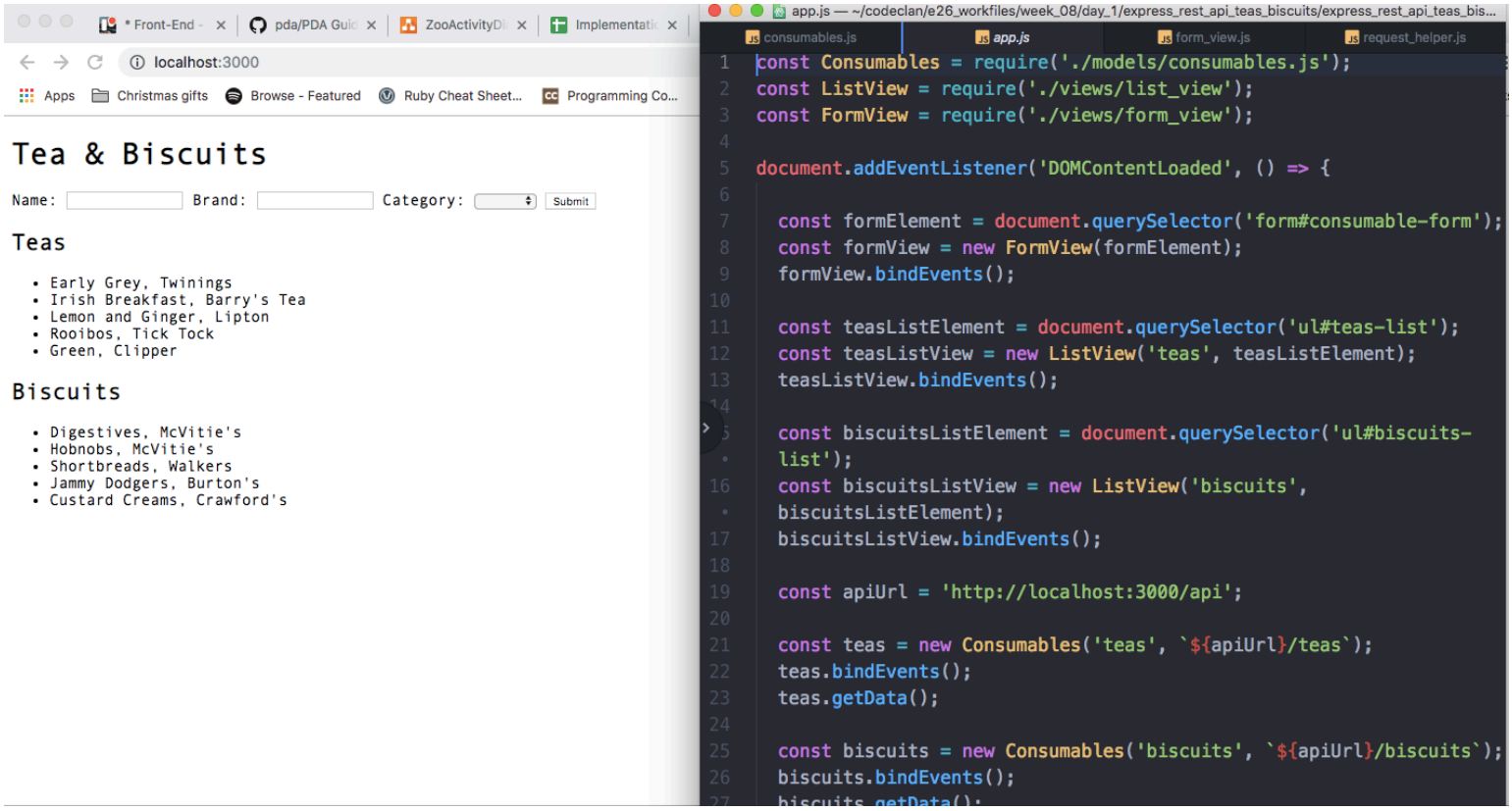
The first is our group using user journeys to discover and figure out precisely what functionality a user would require from our application. We then visualised some of these by using sketchboard.io. The second picture shows our designs and our layout for our application and what we hoped to build.

The third picture shows the application is process of being written. While it is slightly different than our wireframe, these were design decisions taken during writing as it seemed more efficient.

# Week 7

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

## Paste Screenshot here




## Description here

The API is being used to retrieve information about different brands of teas and biscuits. The program then uses this information to display the different brands to the user. The user has the ability to add different brands. This will add them to the database and the API will then retrieve them for the user to see.

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

**Paste Screenshot here**

```ruby
def checkforAce(card)
  if card.value = 1
    return true
  else
    return false
  end
end
```

```
➜ specs git:(master) ✗ ruby testing_task_2_spec.rb
Run options: --seed 13406

# Running:

.....E

Finished in 0.001663s, 3607.9374 runs/s, 3006.6145 assertions/s.

  1) Error:
TestCardGame#test_cards_total:
NoMethodError: undefined method `cards' for #<CardGame:0x007f902812a790>
    testing_task_2_spec.rb:37:in `test_cards_total'

6 runs, 5 assertions, 0 failures, 1 errors, 0 skips
➜ specs git:(master) ✗
```

```ruby
def check_for_ace(card)
  if card.value == 1
    return true
  else
    return false
  end
end
```

```
➜ PDA_Static_and_Dynamic_Task_A git:(master) ✗ ruby specs/testing_task_2_spec.r
b
Run options: --seed 36811

# Running:

......

Finished in 0.001429s, 4198.7404 runs/s, 4198.7404 assertions/s.

6 runs, 6 assertions, 0 failures, 0 errors, 0 skips
➜ PDA_Static_and_Dynamic_Task_A git:(master) ✗
```

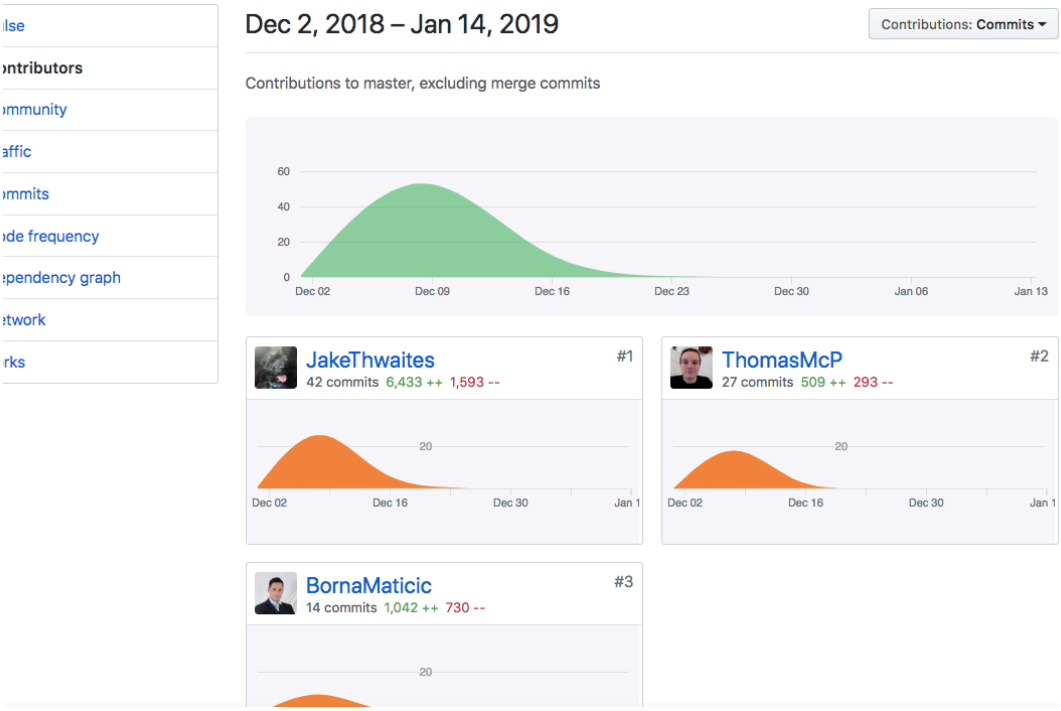
**Description here**

I have inserted screenshots of my code above. I had written a simple test to test the function above. The first screenshots are of it failing. And the second two are of my changes to the function and then it running successfully.

# 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




**THE THREE COLLABORATORS FOR THE JAVASCRIPT PROJECT**

## Description here

This is a screenshot of the contributors page on the group project GitHub repository.
In the group there were three members as shown above.

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

**Paste Screenshot here**

# Personal CO2 Footprint Checker

You have been approached by a lifestyle consultancy company dealing with environmental sustainability. Your task is to build a personal CO2 footprint checker app that calculates a user's CO2 footprint based on their lifestyle.

## MVP

A user should be able to:

- to submit values for various aspects of their lifestyle (e.g. diet, commute, recycling and heating routine, holiday habits, etc) and view their CO2 footprint. You'll need to create your own (simple) tested model to calculate this.
- to update the values to see the effect on their CO2 footprint.
- view the CO2 footprint result in a visually interesting ways.

## Example Extensions

- Calculate and visualise projections of CO2 savings based on a user's input.
- Show the CO2 footprint result before and after the user has updated the values.

## API, Libraries, Resources

- https://www.highcharts.com/ HighCharts is an open-source library for rendering responsive charts with good documentation.

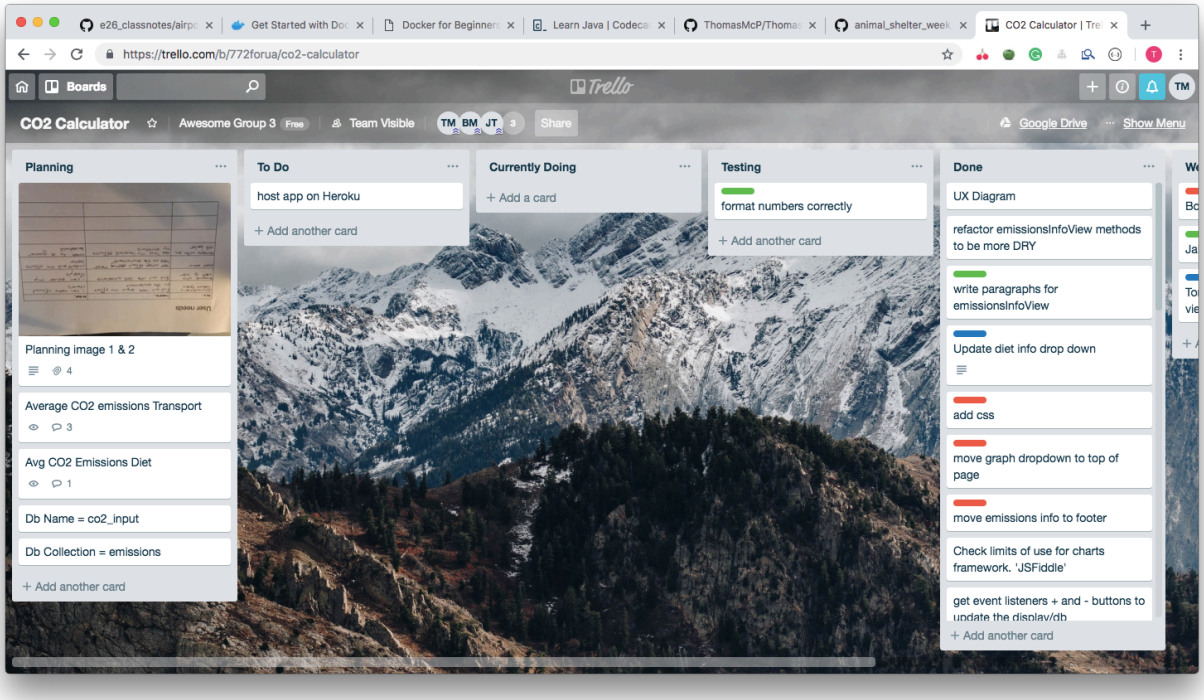
**BRIEF FROM JAVASCRIPT CO2 CALCULATOR GROUP PROJECT**

**Description here**

The above screenshot shows the brief for our selected group project.

| 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**

This Trello board above shows the different planning stages and dividing work between the three contributors. We each had a different colour associations we would know what each of us had to complete each day.

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

**Paste Screenshot here**

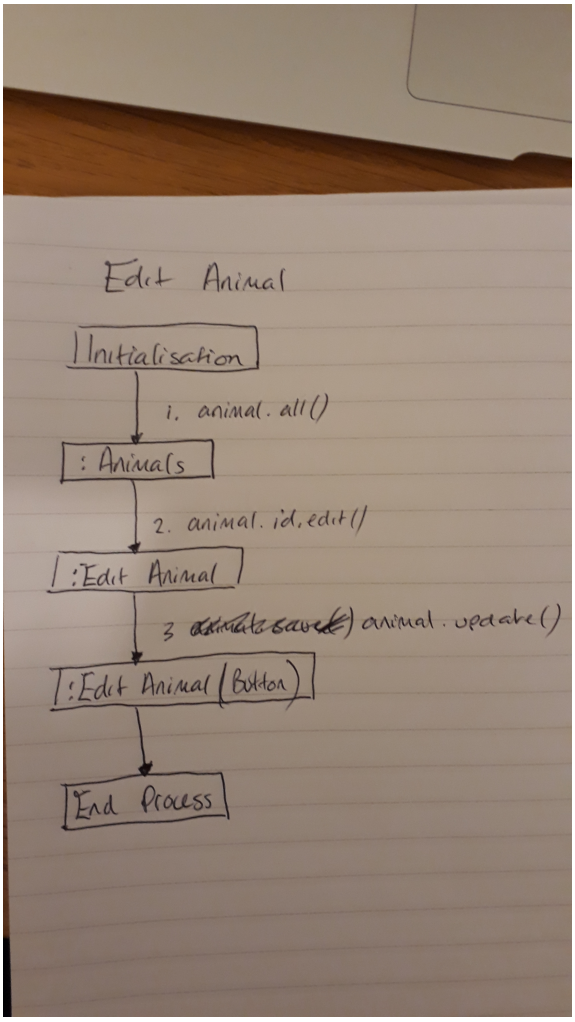
| Acceptance Criteria | Expected Result | Pass/Fail |
|---|---|---|
| A user is able to see a list of all animals within the shelter | The website displays a list of all animals within the database | PASS |
| A user should be able to add an animal to the shelter | The website will add an animal to the list of animals and add to the database once the user inputs the data. | PASS |
| A user should be able to delete an animal from the shelter | Website removes animal from the list and removes from the database. | PASS |
| A user should be able to see a list of animals owned by an owner. | Website shows a list of animals shown as owned by the owner. | FAIL |

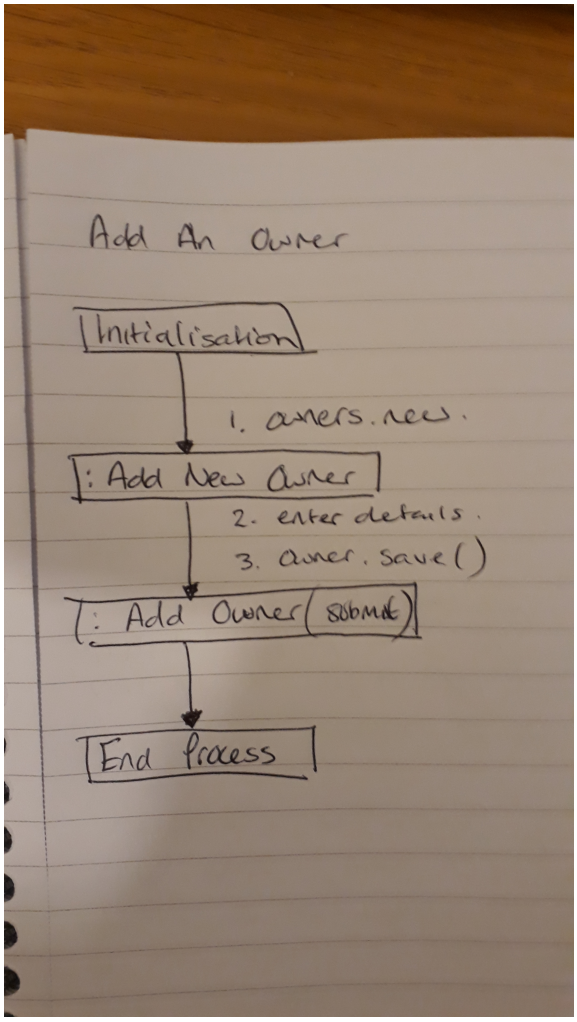
**Description here**

The above is the criteria and test plan for my solo Ruby project.

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

**Paste Screenshot here**




**Description here**

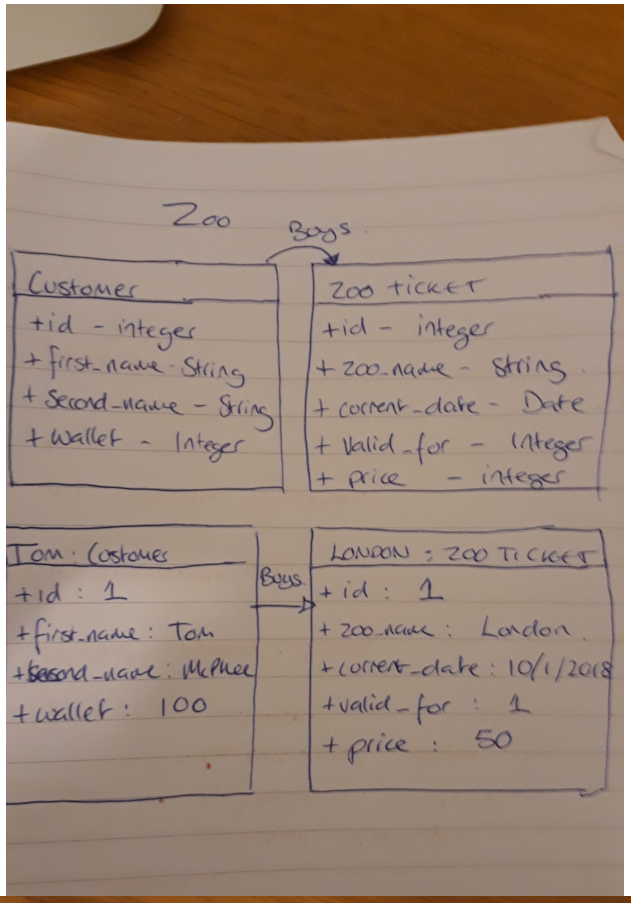
The collaboration diagrams above show the processes needed to edit an animal and add an owner with my Scottish Animal Shelter App.

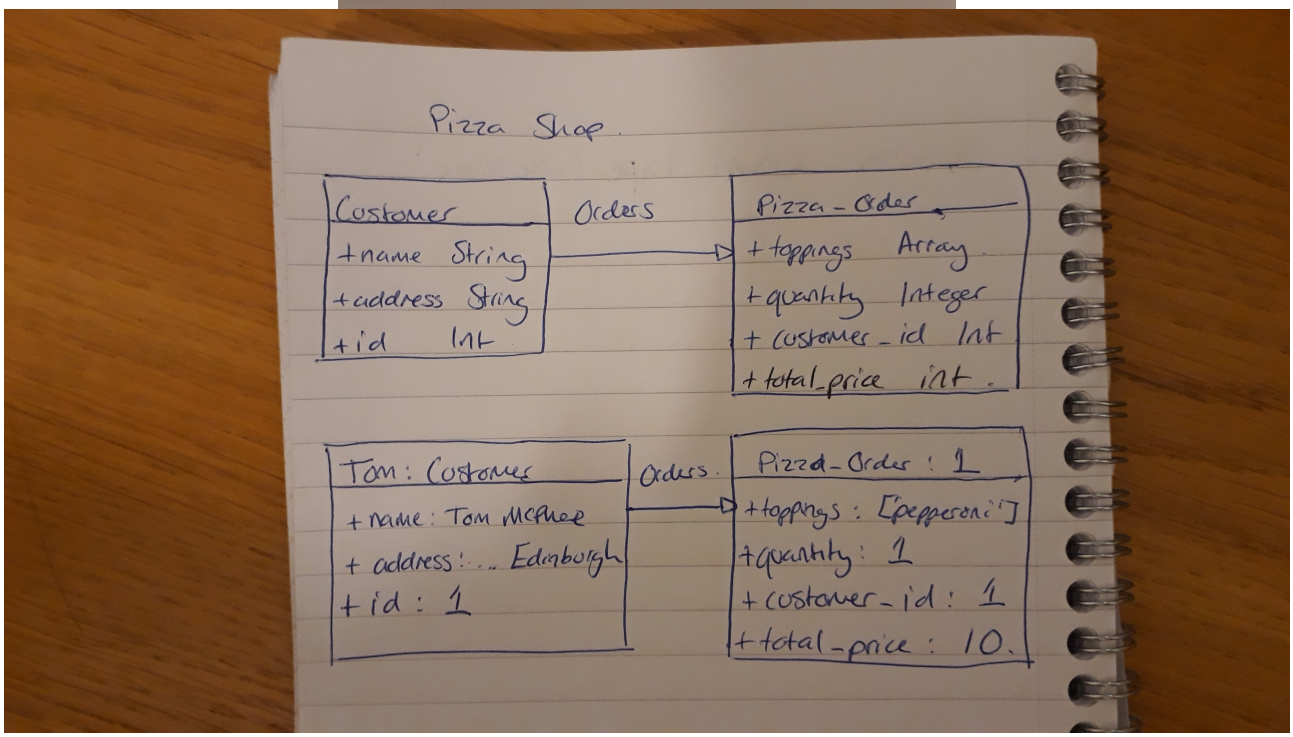
Edit animal process requires the system to load all animals using animal.all() method which takes the user to the list of animals. The user can then select the ‘edit’ button which calls the animal.id.edit method. The user then inputs updated info to the animal form and then presses the edit button. This calls the animal.update() method.

The second image is the process of adding an owner. The user goes to the ‘add new owner’ page and this calls owner.new(). The user enters an owners details and then saves them by pressing the button. This calls the owner.save() method.

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

**Paste Screenshot here**



**Description here**

The first diagram shows a customer and a zoo ticket. The customer can buy a ticket for the zoo if the amount in their wallet is higher than or equal to the price of the zoo ticket.

The second diagram shows a Customer and a Pizza Order. The customer can order a pizza and select what they want and how many.

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

**Paste Screenshot here**

| Bug/Error | Solution | Date |
|---|---|---|
| getEmptyRooms method only returns 1 room when the expected result is 4 | Put the return statement after the loop has finished. | 9th January 2018 |
| getEmptyRooms method returns an array of occupied rooms | Add an '!' to if condition so it returns non occupied rooms. | 9th January 2018 |
| getNightlyIncome method shows an error which states that method should return a double not an integer | Change sum variable to a double to match that of the expected return type. | 9th January 2018 |
| Error shown when testing to book a room for a guest within the hotel. | Instantiate a room within the hotel before trying to add a guest. | 9th January 2018 |
| totalCapacity method returning 0 when expecting 1. | Corrected typo within the return statement | 9th January 2018 |

**Description here**

# 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**

```
public class Guitar extends Instrument implements IPlay, ISell {

    private GuitarType type;
    private int numberOfStrings;
```

```
public class Shop {

    private ArrayList<ISell> stock;
    private String shopName;

    public Shop(String shopName) {
        this.stock = new ArrayList<>();
        this.shopName = shopName;

    }

public void addToStock(ISell item) { stock.add(item); }
```

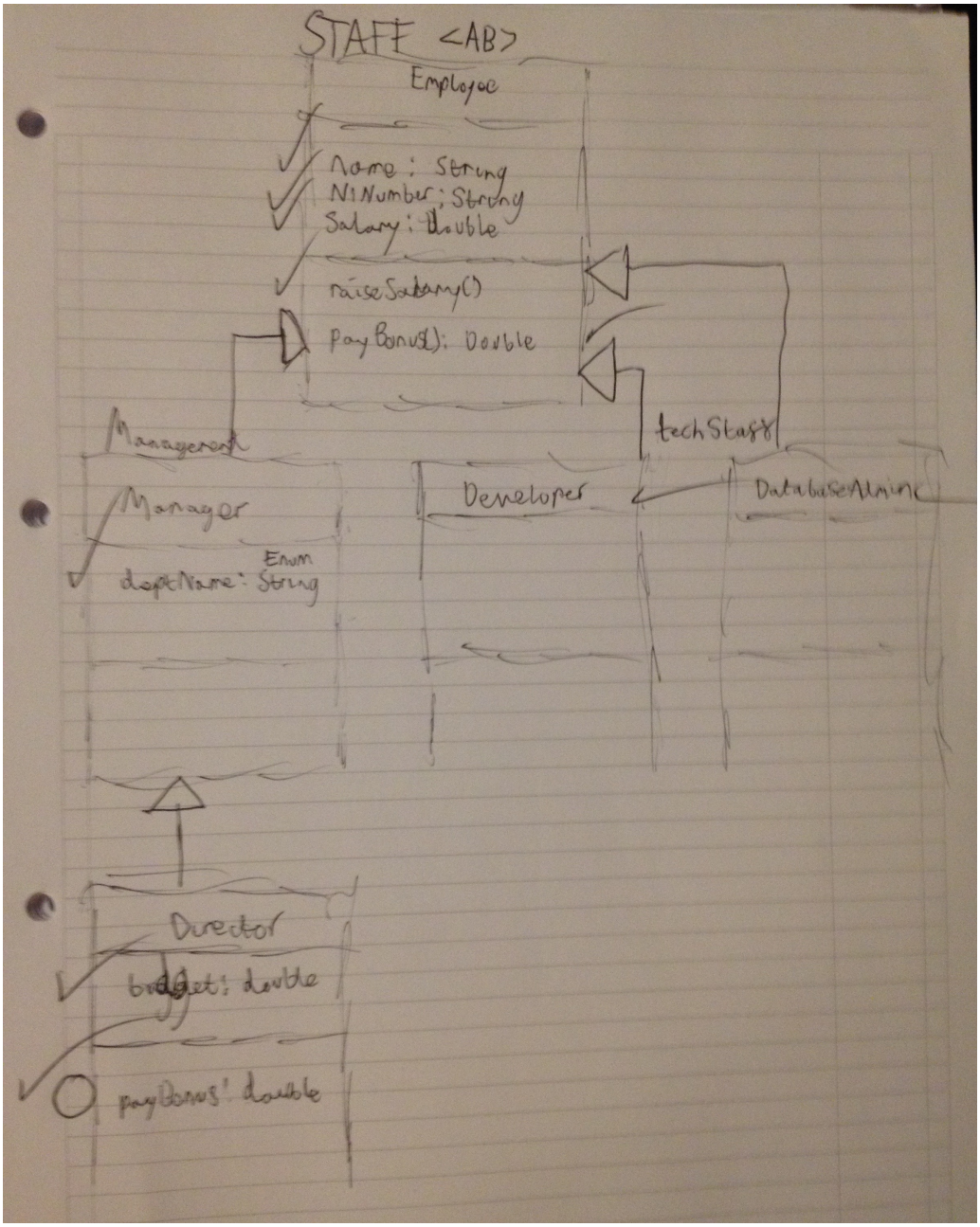
```
@Before
public void before(){
    shop = new Shop( shopName: "Record shop.Shop");
    guitar = new Guitar(InstrumentFamilyType.STRINGS, In
    keyboard = new Keyboard(InstrumentFamilyType.KEYBOAR
}

@Test
public void hasName() { assertEquals( expected: "Record s

@Test
public void stockIsZero() { assertEquals( expected: 0, sh

@Test
public void canAddItemToStock(){
    shop.addToStock(guitar);
    shop.addToStock(keyboard);
    assertEquals( expected: 2, shop.getStockAmount());
}
```

**Description here**

Above is a class type Guitar which implements an interface called ISell.
The Shop class has stock which is an ArrayList of Sell Items and it has a method addToStock which takes in an item of ISell type.
In the last picture you can see that the addToStock method has been called and is passed a 'guitar' which is of Guitar class.
This is successful only because the Guitar class implements the ISell interface and so can be treated as an ISell type item.

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

**Paste Screenshot here**




**Description here**

This is a screenshot of an inheritance diagram modelling software for a staff database. A Employee class has three attributes and two methods. Three more classes have been created, all of which inherit the same properties and methods with the exception of the 'Manager' class which has an extra property.

There is an extra inheritance. That of a ‘Director’ which  inherits directly from ‘Manager’ and not ‘Employee’ This has created clear parent and child inheritance flow.

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

**Paste Screenshot here**

```java
public class Hotel {
    private ArrayList<Bedroom> bedrooms;
    private String name;
    private int numberOfFloors;

    public Hotel(String name, int numberOfFloors){
        this.name = name;
        this.numberOfFloors = numberOfFloors;
        this.bedrooms = new ArrayList<>();
    }
```

**Description here**

The instance variables of hotel are all set to ‘private’. This means they cannot be accessed outside the Hotel class. It can only be accessed through a getter method. This is encapsulation and this stops any unauthorised access of the information only allowing access through the correct way (the getter method).

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

**Paste Screenshot here**

```java
public abstract class Instrument {
    private InstrumentFamilyType family;
    private InstrumentColourType colour;
    private InstrumentMakeType make;
    private InstrumentModelType model;
    private int price;

    public Instrument(InstrumentFamilyType family, InstrumentColourType colour, InstrumentMakeType make, InstrumentModelType model, int price) {
        this.family = family;
        this.colour = colour;
        this.make = make;
        this.model = model;
        this.price = price;
    }

    public InstrumentFamilyType getFamily() { return family; }

    public InstrumentColourType getColour() { return colour; }

    public InstrumentMakeType getMake() { return make; }

    public InstrumentModelType getModel() { return model; }

    public int getPrice() { return price; }
```

```java
public class Guitar extends Instrument implements IPlay, ISell {

    private GuitarType type;
    private int numberOfStrings;


    public Guitar(InstrumentFamilyType family, InstrumentColourType colour, InstrumentMakeType make, InstrumentMo
        super(family, colour, make, model, price);
        this.numberOfStrings = numberOfStrings;
        this.type = type;
    }

    public GuitarType getType() { return type; }

    public int getNumberOfStrings() { return numberOfStrings; }

    public String play(Instrument instrument) { return "Instruments.Guitar playing"; }

}
```

```java
    Guitar guitar;

    @Before
    public void before(){
        guitar = new Guitar(InstrumentFamilyType.STRINGS, InstrumentColourType.BLUE, InstrumentMakeType.FENDER, Ins
    }

    @Test
    public void canGetFamily() { assertEquals( expected: "Strings", guitar.getFamily().getValue()); }

    @Test
    public void canGetColourType() { assertEquals( expected: "#4156f4", guitar.getColour().getColourCode()); }

    @Test
    public void canGetModelType() { assertEquals( expected: "HA3455T", guitar.getModel().getModelNumber() ); }

    @Test
    public void getMake() { assertEquals(InstrumentMakeType.FENDER, guitar.getMake()); }

    @Test
    public void canGEtNumberOfStrings() { assertEquals( expected: 4, guitar.getNumberOfStrings()); }

    @Test
    public void canGetGuitarType() { assertEquals(GuitarType.ELECTRIC, guitar.getType()); }

    @Test
    public void canGetPrice() { assertEquals( expected: 500, guitar.getPrice()); }

    @Test
    public void canCalculateMarkUp() { assertEquals( expected: 1000, guitar.calculateMarkup(),  delta: 0.001); }
}
```

**Description here**

The top picture is an abstract class called Instrument which has a number of methods (getFamily(), getColour()) etc.
The next picture is that of a Guitar class which inherits from the Instrument class (extends Instrument) and has use of all of its methods.
The third picture shows this happening. I have created an object of Guitar class which runs tests to show that a Guitar type object can call Instrument type methods.

| 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**

```
public ArrayList<Bedroom> getEmptyRooms() {
    ArrayList<Bedroom> foundRooms = new ArrayList<>();
    for (Bedroom bedroom: this.bedrooms) {
        if (!bedroom.isOccupied()){
            foundRooms.add(bedroom);
        }
    }
    return foundRooms;
}
```

```
public int getNumberOfGuests() {
    int totalGuests = 0;
    for (Bedroom bedroom: this.bedrooms) {
        totalGuests += bedroom.numberOfGuests();
    }
    return totalGuests;
}
```

**Description here**

The first algorithm is from a hotel management application.
It finds empty rooms within the hotel.
It does this by creating a new and empty array list called 'foundRooms'. It then loops through an array list of all rooms (this.bedrooms) and checks to see if each room is NOT occupied(by using the isOccupied method). If that condition is true then it adds the bedroom to the foundRooms arraylist. I chose this algorithm because I started to truly appreciate Java during this project.

The second is an algorithm I have used countless times in my short career so far, which is exactly why I chose it. The ability to do swift calculations is something I have truly begun to appreciate. The method counts the number of guests within the hotel at any moment. The method has a variable called totalGuests which begins at 0. There is then a loop, which goes through each bedroom from the array list of rooms and adds the number of guests from each room to the totalGuests variable. It is then returned. This returns the user the total guests within the hotel.