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.

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 1

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

```
  admin: {
    total_cash: 1000,
    pets_sold: 0,
  },
```

```
def total_cash(pet_shop)
    p pet_shop[:admin][:total_cash]
end
```

```
➜  specs git:(master) ✗ ruby pet_shop_spec.rb
Run options: --seed 6440

# Running:

1000
.

Finished in 0.001036s, 965.2510 runs/s, 965.2510 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Description here

Image 1 contains the admin hash with two keys and two values. The keys are total_cash and pets_sold. The values are 1000 and 0.

Image 2 contains a function that used a hash. The function accesses the pet shop instance variable (not shown) then accesses the admin hash via [:admin] and then accesses the total_cash key via [total_cash].

Image 3 contains the result of the function total_cash running.

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

```ruby
def initialize(room_type)

  @room_type = room_type
  @guests_in_room = []
  @song_in_room = []
```

```ruby
def add_song_to_room(song)
  p @song_in_room << song.song_name
end
```

```
➜  specs git:(master) ✗ ruby rooms_specs.rb
Run options: --seed 24491

# Running:

["Lose Yourself"]
.

Finished in 0.001038s, 963.3911 runs/s, 963.3911 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Description here

Image 1 contains an instance variable called @song_in_room that is an empty array.

Image 2 contains a function that is shovelling the argument of song into the empty array. The song argument is an object from another class.

Image 3 contains the result of the function add_song_to_room running. The song “Lose Yourself” has been added to the array.

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

```ruby
@pet_shop = {
  pets: [
    {
      name: "Sir Percy",
      pet_type: :cat,
      breed: "British Shorthair",
      price: 500
    },
```

```ruby
def find_pet_by_name(pet_shop, pet_name)
  for pet in pet_shop[:pets]
    if pet[:name] == pet_name
      return pet
    end
  end
  return nil
end
```

```ruby
def test_find_pet_by_name__returns_pet
  pet = find_pet_by_name(@pet_shop, "Sir Percy")
  assert_equal("Sir Percy", pet[:name])
end
```

```
➜  specs git:(master) ✗ ruby pet_shop_spec.rb
Run options: --seed 64302

# Running:

.

Finished in 0.001013s, 987.1668 runs/s, 987.1668 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Description here

Image 1 shows the @pet_shop object which contains an array called pets. This array contains a list of hashes. Shown is the first hash in that array.

Image 2 contains the function that will find a pet by its name. It loops through each hash in the array and if a hash's name key is the same as the argument passed to it then it will return that animals name.

Image 3 contains the test for the find function. It contains the two arguments that will be held as parameters in the find pet by name function.

Image 4 is the result of the function running.

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

```ruby
animals = ["tiger", "elephant", "wolf", "highland cow", "bat"]

def sort(animals)
animals.sort_by {|species| species.length}
end

puts sort(animals)
```

```
[➜  ruby_pda_function ruby sort_function.rb
bat
wolf
tiger
elephant
highland cow
```

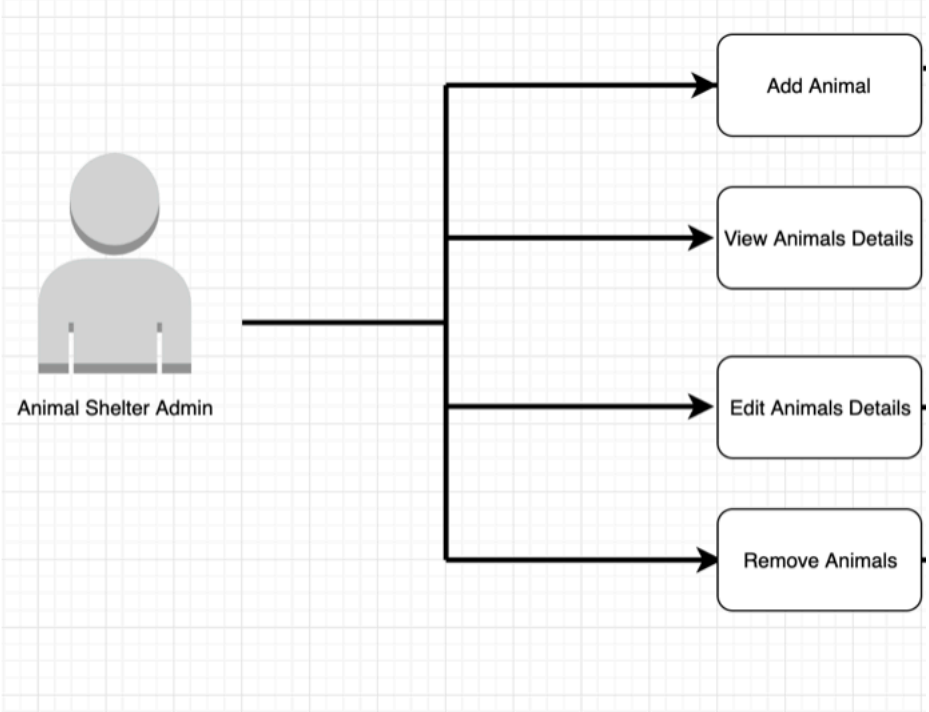
Description here

Image 1 contains an array called animals. This array contains a list of strings. It is then followed by a function which will loop through each element within the array and put them into order based on the number of characters each string contains.

Image 2 is the result of that function running.

Week 4

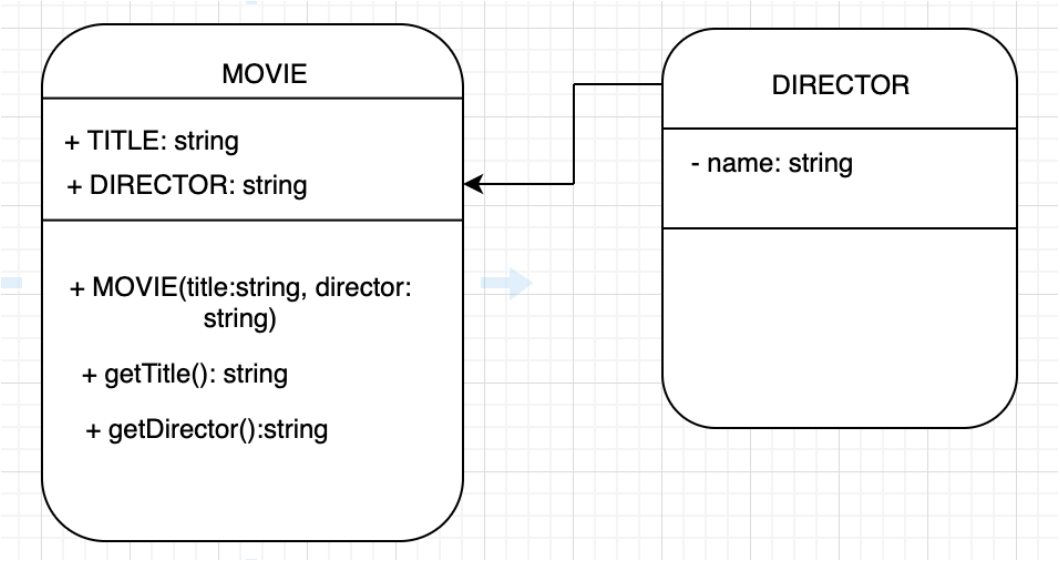
Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram



Description here

The above image is of an admin user who can perform various actions (functionality) with the animal shelters information and database.

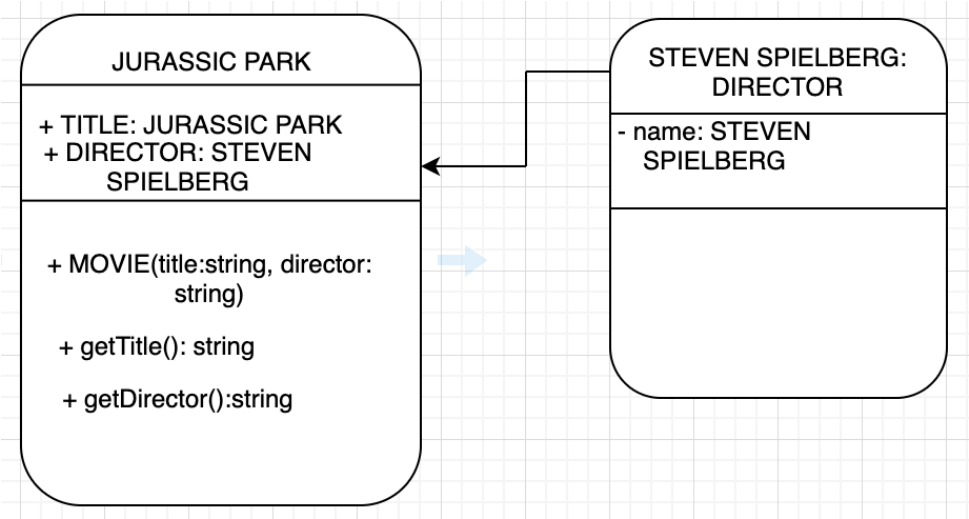
Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram



Description here

The above image shows a class diagram that contains both a movie and director class. The diagram shows the proposed attributes and methods of both classes. The director class has a name attribute that is a string. The movie class has the attribute of title but also director which it attains from the director class. The director class has a direct relationship with the movie class, as indicated by the arrow.

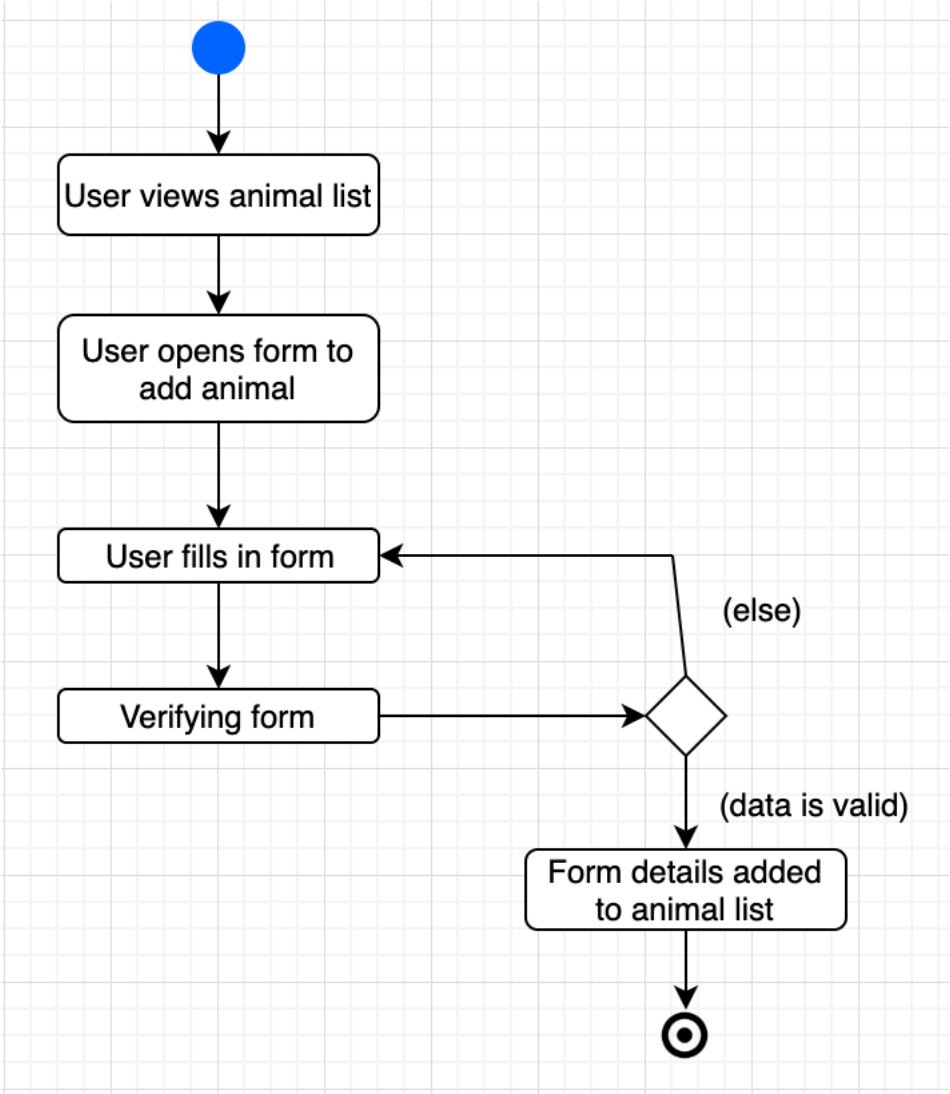
Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram



Description here

The above image is an object diagram of the above class diagram. Jurassic Park is an object of the movie class and Steven Spielberg is an object of the director class. The movie object attains its director attribute from the director class.

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



Description here

Above is an activity diagram showing the process of what a shelter employee is to do in order to update the database when a new animal arrives at the shelter. First the user selects the link to go to display the animal list. The user then opens a new animal form and enters the details of the animal and submits the information. The app then checks that the animal details are all valid (i.e. not a string in the date section). If the details are not valid the app keeps the user on the same page to try again; if not then the animal details are saved to the database.

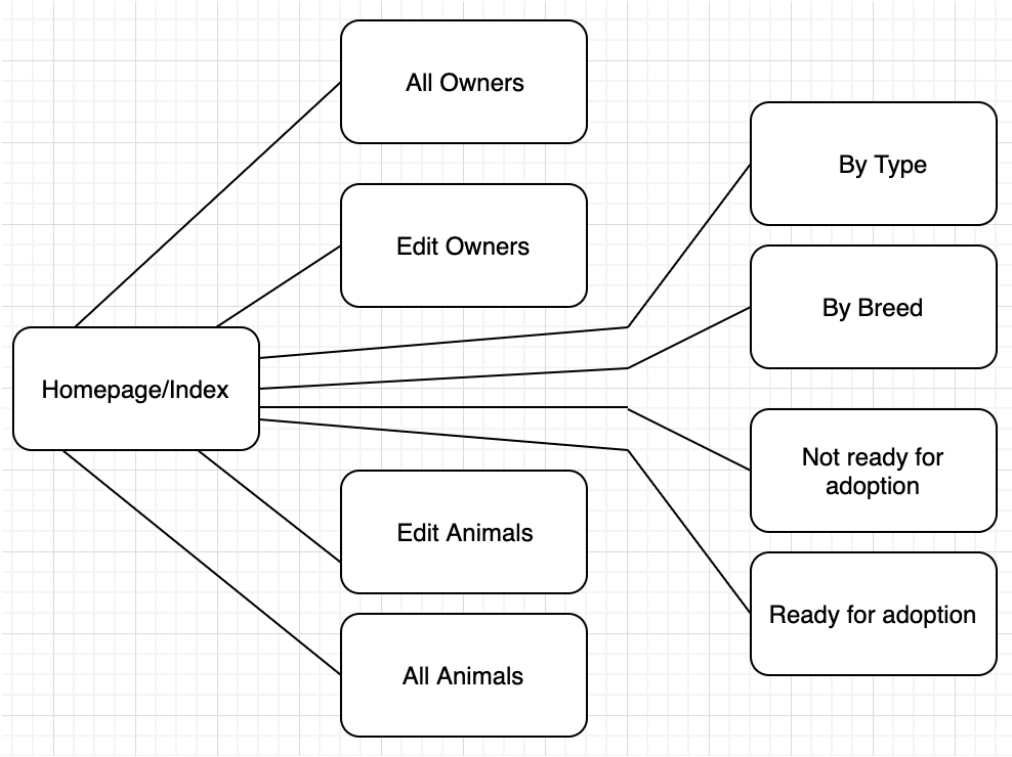
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time

Implementations Constraint Table - Animal Shelter		
Constraint Category	Implementation Constraint	Solution
Hardware & Software	The app was designed to run on MacOS with chrome browser compatibility. It is possible the app may not run correctly on other hardware/software configurations. This matters as the volunteers will likely not have macbooks and thus will not be able to use the app.	Test the app on other hardware/software configurations and modify the product if required to ensure it works correctly regardless of the hardware/software config.
Performance Requirements	The app has low performance requirements but performance with larger databases is unknown which may cause an issue down the line when the db size increases. This matters as if the performance is too slow then the volunteers may be unable to use the app and affect the running of the shelter.	Test the product with a larger database to ensure adequate performance.
Persistent Storage & Transactions	The storage will be local to the machine. As such the app cannot be accessed by other devices. This matters as if the shelter runs out of local storage they will be unable to add to the database.	Remote hosting could be deployed so that the database can be accessed from multiple devices.
Usability	The user interface should be non-technical as users are unlikely to be technically savvy. This matters as if the app is too complex the volunteers may struggle to use it.	Carefully create the UI to be simple and intuitive to use.
Budgets	No budget is available and thus proprietary software cannot be used for this product. This matters as if proprietary software is used the shelter would most likely be unable to use it due to lack of funds.	Design and plan the product based on having a single developer and having no paid resources.
Time	A one week deadline limits the amount of functionality that can be included within the app. This matters as if too ambitious during the planning stage, the MVP may be incomplete at the end of the week.	Evaluate any additional functionality that goes beyond the scope of the MVP and prioritise accordingly. Create a schedule for all aspects of the planning and implementation phases to maximise productivity.

Description here

Above is an implementation and constraints plan for the animal shelter app. It was created with the presupposition that workers within the animal shelter would be volunteers and possibly older, meaning they would not have much in the way of computer experience and so creating a simple, easy to use app was of most importance.

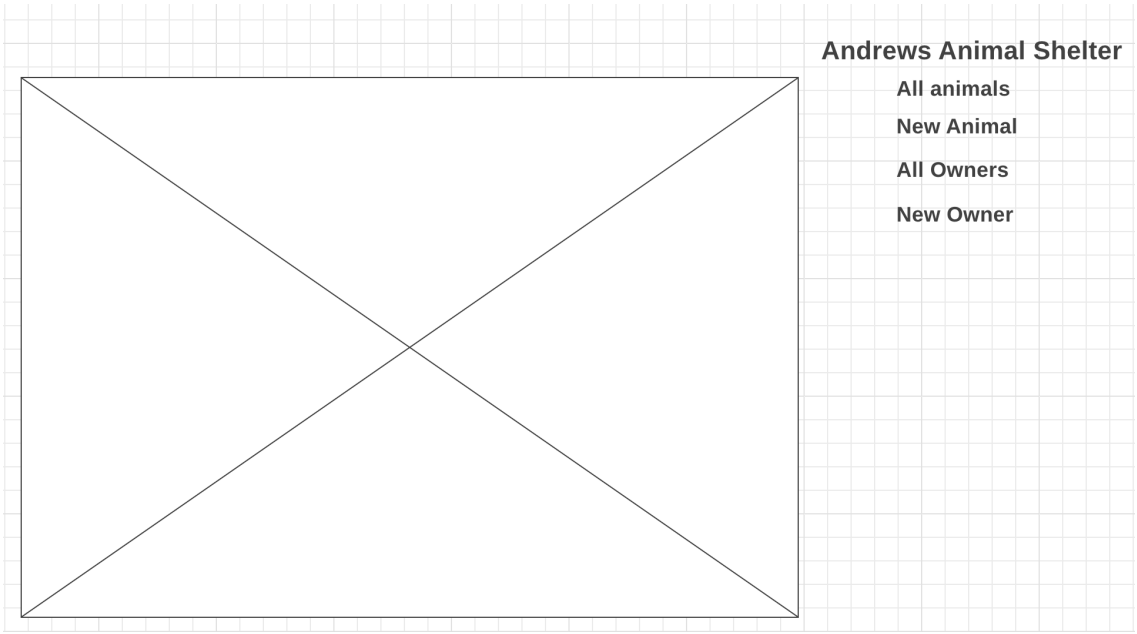
Unit	Ref	Evidence
P	P.5	User Site Map

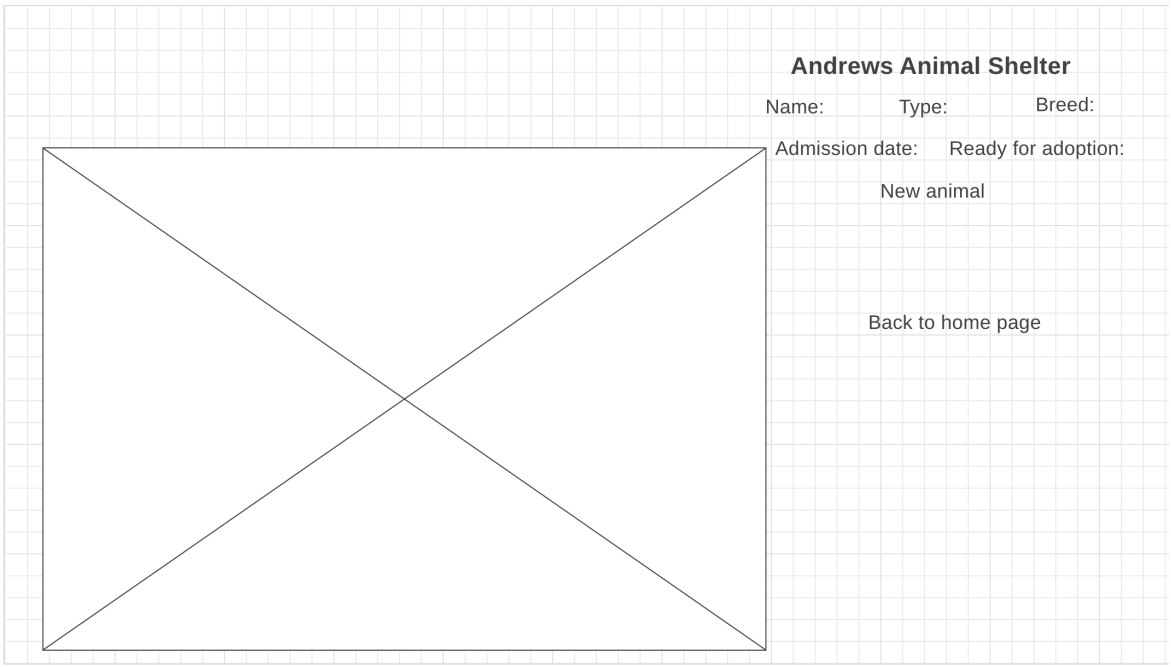


Description here

Above is a user site map that links all pages back to the homepage/index.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams





Description here

Image 1 shows the wireframe for the home page.

Image 2 shows the wireframe for the new animal page.

Week 5

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

```
def check_out_guests(guest)
  @guests_in_room.delete(guest)
end
```

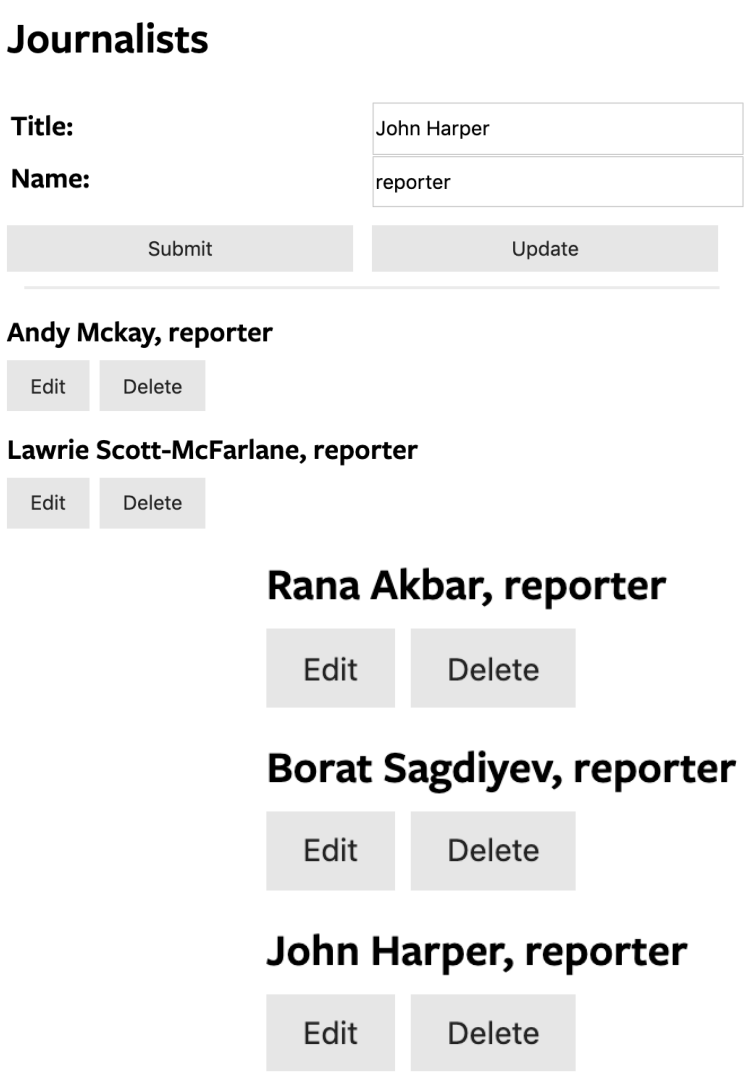
```
# get guest1 and use them as a parameter for the check_out_guests method
# use the count_guests_in_room method on the room1 object
# once this happens it will show how many people are left in that room
def test_check_out_guests
  @room1.check_out_guests(@guest1)
  assert_equal(0, @room1.count_guests_in_room)
end
```

Description here

Image 1 shows the check_out_guests method that will hold our guest parameter in image 2.

Image 2 shows the test method along with pseudocode above. This is an example of pseudocode for a function to remove a guest from a room. Once the guest is removed the count_guest_in_room function checks how many people are left in the room; with the answer being 0.

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

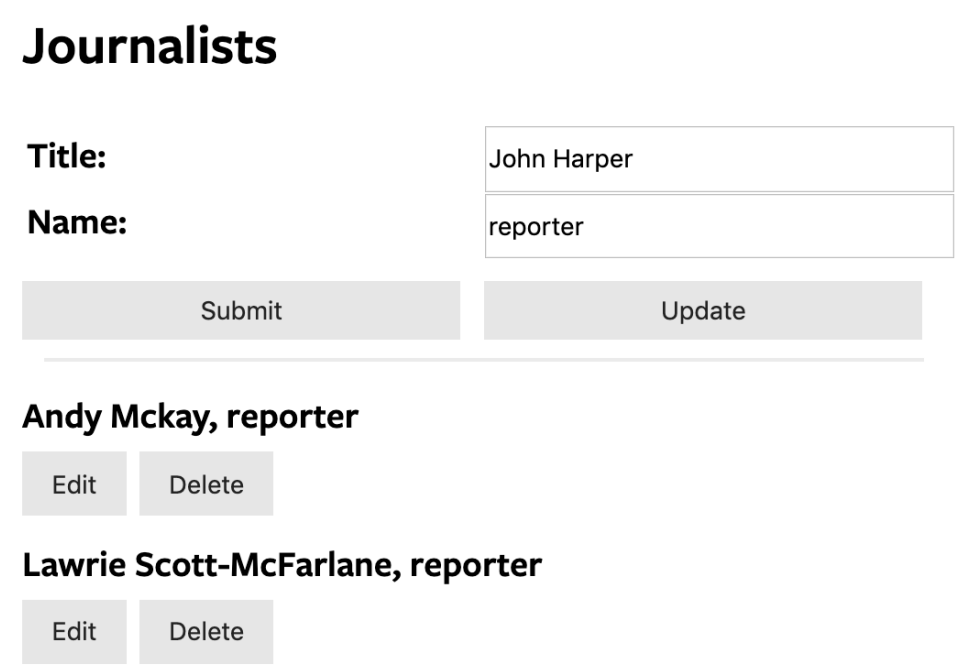


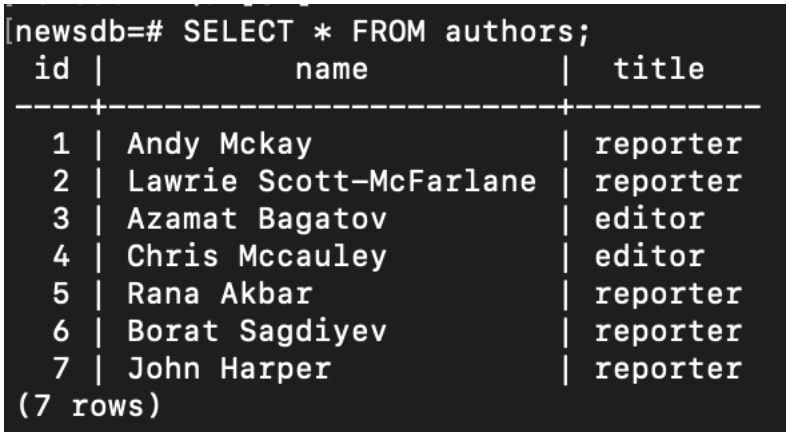
Description here

Image 1 shows an admin user filling out a form to add a new journalist to a database.

Image 2 shows the result of that new journalist being added to the database and displayed in the browser.

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





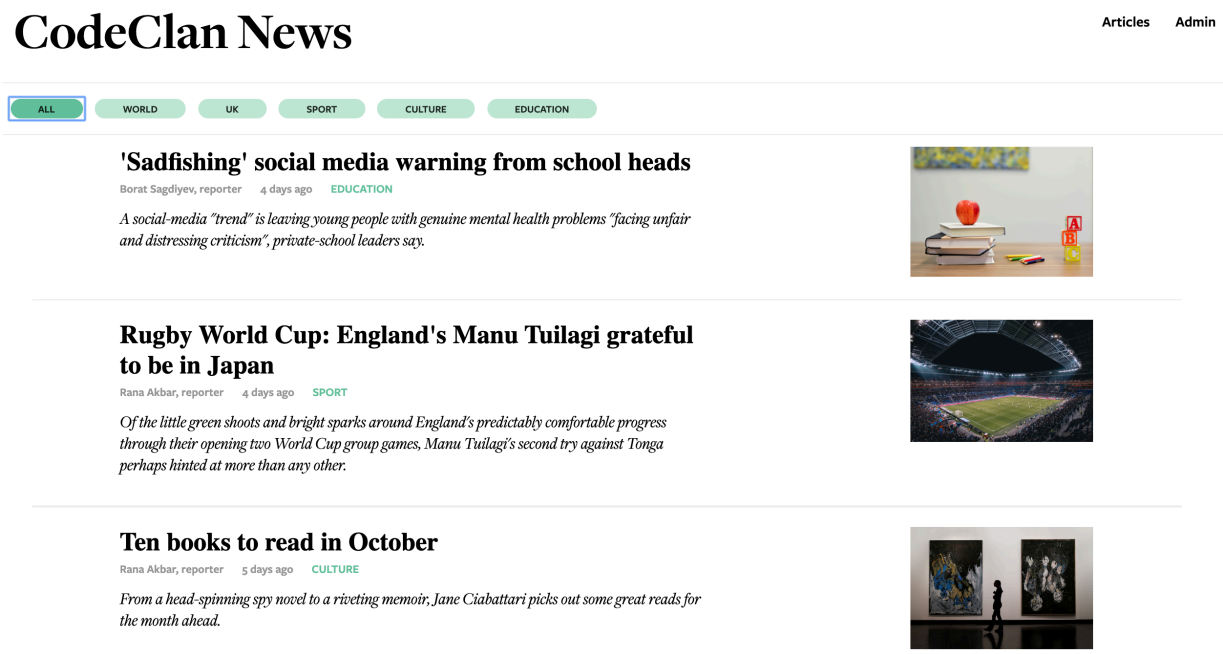
```
newsdb=# SELECT * FROM authors;
 id |          name          |  title
----+------------------------+----------
  1 | Andy Mckay             | reporter
  2 | Lawrie Scott-McFarlane | reporter
  3 | Azamat Bagatov         | editor
  4 | Chris Mccauley         | editor
  5 | Rana Akbar             | reporter
  6 | Borat Sagdiyev         | reporter
  7 | John Harper            | reporter
(7 rows)
```

Description here

The first image shows an admin user inputting both a form to add a new journalist.

The second image is the database table where all journalists are stored. The result of the form input “John Harper” has been saved to this table.

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



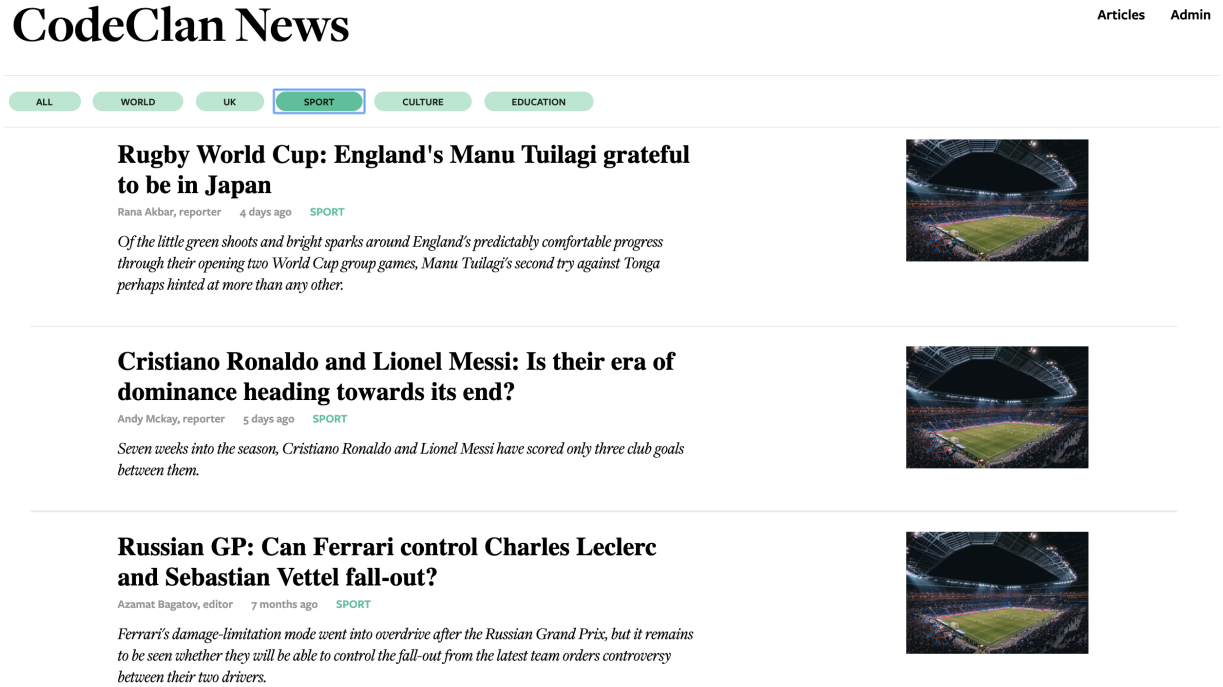
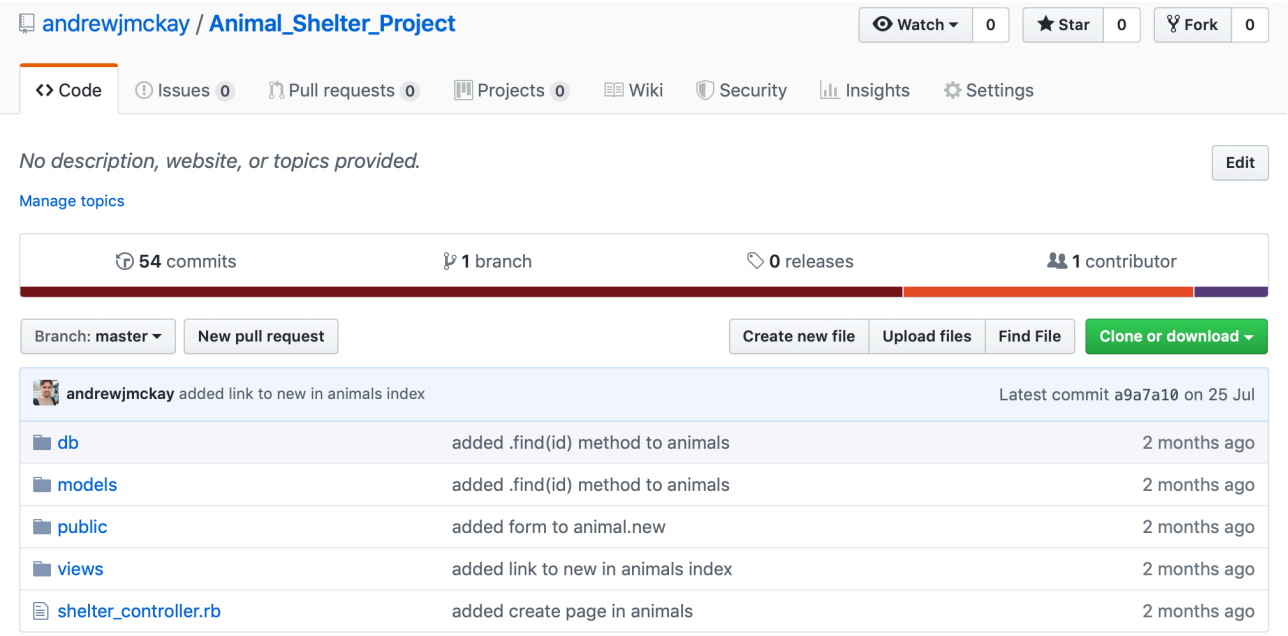


Image 1 displays the main page of a news website. By default all articles are displayed. The user can filter by type of news from the buttons at the top of the articles.
Image 2 shows the result of the user selecting sport as the news topic. Only sports related articles will be returned - as seen by the stadium picture attached to the sport genre.

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

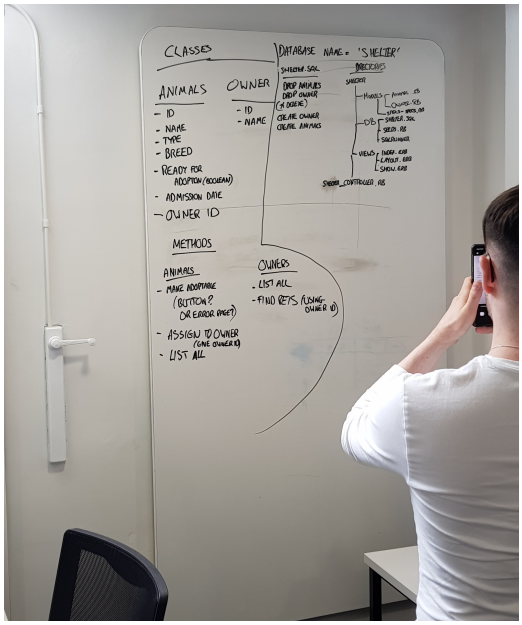


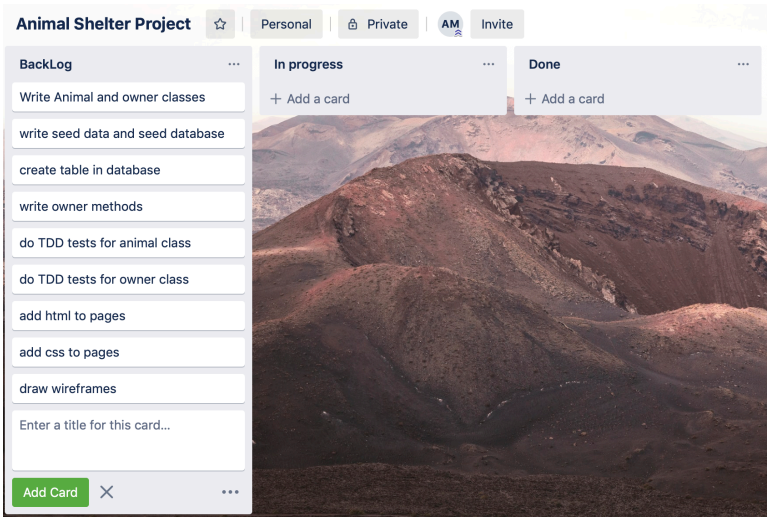
Description here

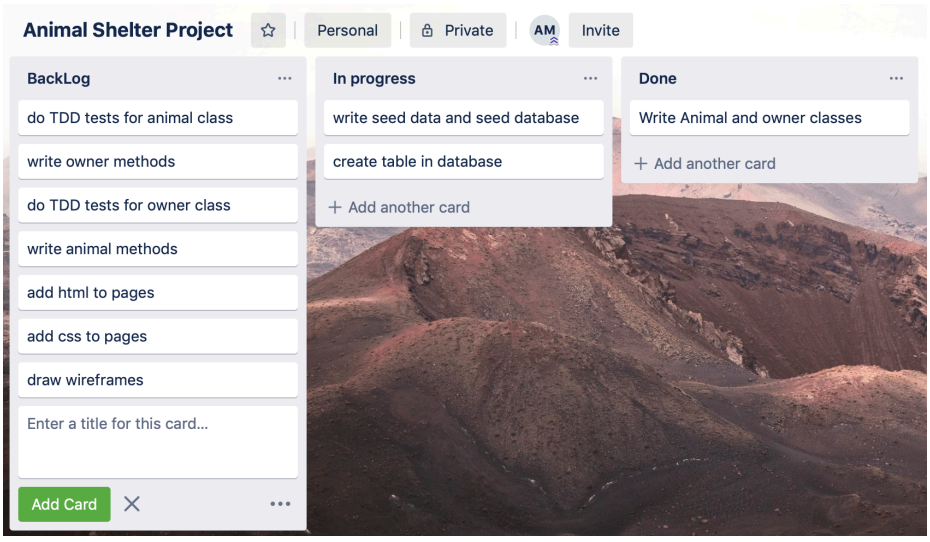
The Github link to this project is;

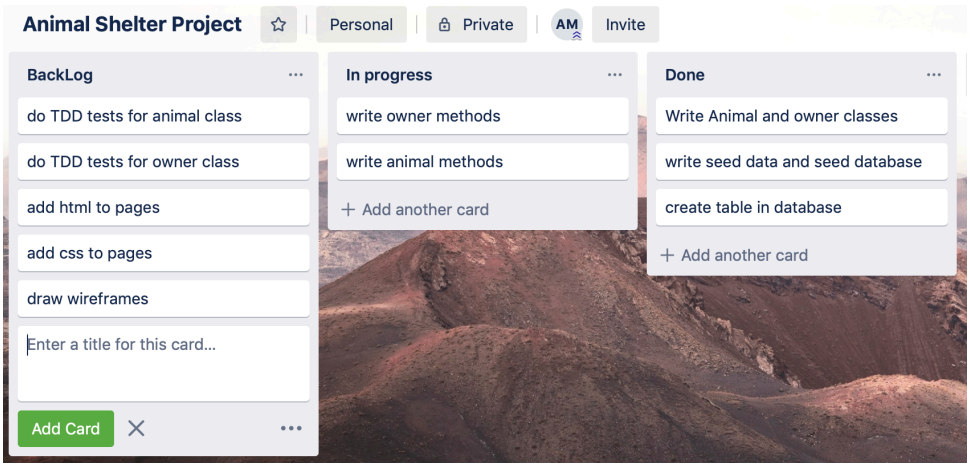
https://github.com/andrewjmckay/Animal_Shelter_Project

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









Description here

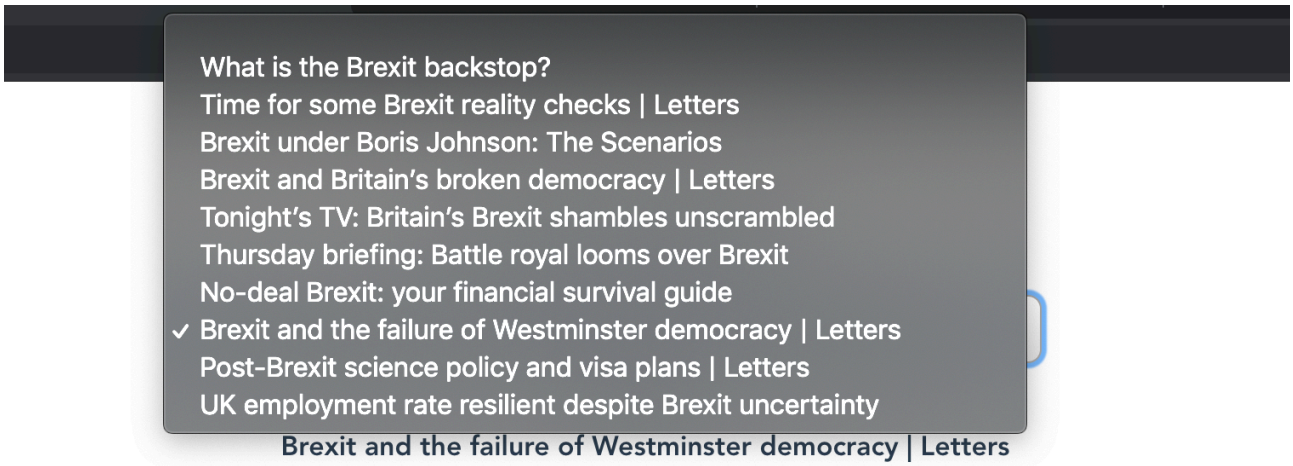
The photo on the top left was our groups initial thought process on what we should include and we used the whiteboard to lay out these ideas.

The other photos from top to bottom shows the Trello board used for planning and progress as tasks were completed.

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

```
fetch('https://content.guardianapis.com/search?q=brexit&format=json&api-key=test')

.then(res => res.json())

.then(resjson => this.articles = resjson.response.results);
```



https://www.theguardian.com/politics/2019/aug/05/brexit-and-the-failure-of-westminster-democracy

Description here

Image 1 is the main App view fetching the API under the mounted() function.

1. Fetch pulls the information from the API.

2. ".then (res => res.json())" is turning the result of the API into json format.

3. ".then(resjson => this.articles = resjson.response.results); takes the result of calling the json function on the result of the API and saves it in a variable named resjson. "resjson.response.results" is accessing specific objects within the API and storing it into an empty array called "articles" which is found under return of data in export default.

Image 2 is the result of the API running in the browser. A dropdown allows the user to specify what article to read and the result of that action returns some information about that article, including a link to the article itself.

Week 8

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

Description here

Educational App

The BBC are looking to improve their online offering of educational content by developing some interactive browser applications that display information in a fun and interesting way. Your task is to make an a Minimum Viable Product or prototype to put forward to them - this may only be for a small set of information, and may only showcase some of the features to be included in the final app.

MVP

A user should be able to:

- view some educational content on a particular topic
- be able to interact with the page to move through different sections of content

Example Extensions

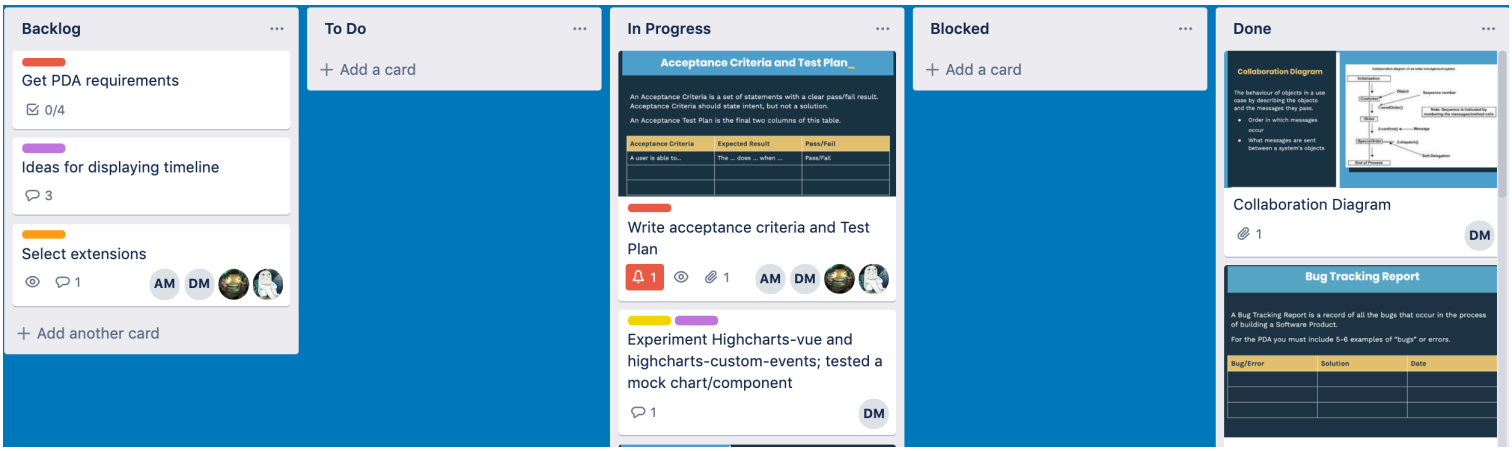
- Use an API to bring in content or a database to store information.
- Use charts or maps to display your information to the page.

API, Libraries, Resources

- https://www.highcharts.com/ HighCharts is an open-source library for rendering responsive charts.
- https://leafletjs.com/ Leaflet is an open-source library for rendering maps and map functionality.

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

Description here



Description here

Above is the Trello board for our JavaScript project. Backlog contained all tasks we wanted to accomplish. To Do contained the most important tasks from backlog that we wanted to focus on next. In Progress contained the tasks currently being done. Blocked was for if we were struggling with a task and wanted to leave it there to come back to it at another time. Done was for finished tasks.

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

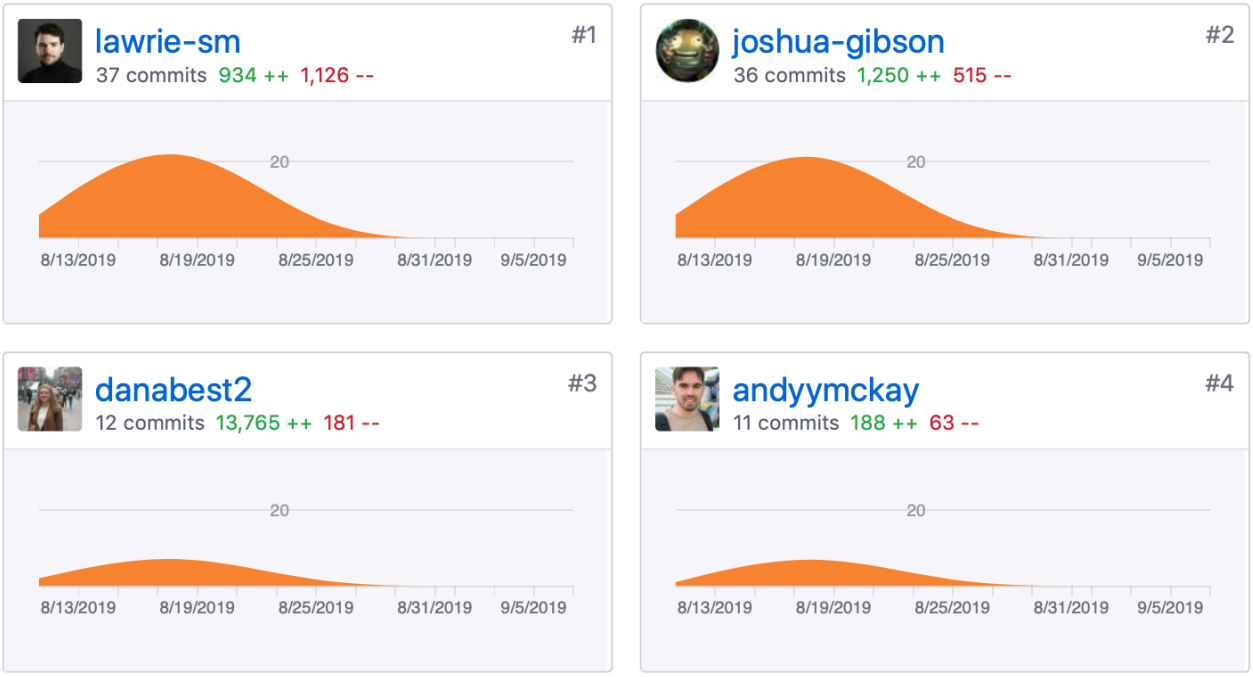
Acceptance Criteria	Test Plan	Pass / Fail
The user is able to navigate through all future space launches by scrolling down.	The page automatically refreshes when the user reaches the bottom of the page and loads up the next space launches from the API.	Pass
The user is able to search for specific space launches via a date search bar.	The app returns only the launches from the API that fall between the two dates provided by the user.	Pass
The user is able to click on a card component which expands each individual space launch and show more information about each launch.	Once clicked the card component will expand; displaying more information such as country, flag, short info about the launch and google map component from where the launch took place.	Pass
The user is able to use google maps to see each space flight's launch location	When the card component is clicked the interactive map is displayed and uses coordinates from the API to get the correct location	Pass
The user is able to observe a countdown till the next space launch	When the splash page loads, the countdown will display under the app title and uses the API to pull in the time till the next space flight	Pass

Description here

Above is an acceptance and criteria plan from our javascript project that was to develop an interactive educational app.

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.



Week 11

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

```ruby
def test_remove_customer_cash
  customer = @customers[0]
  remove_customer_cash(customer, 100)
  assert_equal(899, customer[:cash])
end
```

```
[➜  specs git:(master) ✗ ruby pet_shop_spec.rb
Run options: --seed 26451

# Running:

F

Finished in 0.001165s, 858.3693 runs/s, 858.3693 assertions/s.

  1) Failure:
TestPetShop#test_remove_customer_cash [pet_shop_spec.rb:160]:
Expected: 899
  Actual: 900

1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
```

```ruby
def test_remove_customer_cash
  customer = @customers[0]
  remove_customer_cash(customer, 100)
  assert_equal(900, customer[:cash])
end
```

```
Run options: --seed 18459

# Running:

.

Finished in 0.000949s, 1053.7408 runs/s, 1053.7408 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Description here

The first image shows the test for the method remove_customer_cash. The customer variable is equals to an array which contains a list of hashes. The first customers cash key has a value of 1000. We then subtract 100 from this value using the remove_customer_cash function.

The second image shows this test failing as the assert equals is set to 899, which is incorrect.

The third image shows the assert equals being set to 900.

The final image shows this test passing as 1000 - 100 = 900.

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

```
public class Car {

    private CarType type;
    private CarModel model;
    private ArrayList<Passenger> Passengers;
```

```
public CarModel getCarModel() {return model;}

public void setCarModel(CarModel model) {this.model = model;}
```

Description here

This shows encapsulation within the class Car. The variable model is private. This means it is only accessible within the class Car and in order to access or change it, the getCarModel or setCarModel methods must be used.

Week 12

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

```java
import java.util.ArrayList;

public class Game {

    private ArrayList<Monster> monsters;
    private ArrayList<IObject> objects;
    private Room currentRoom;

    public Game(ArrayList<Monster> monsters,ArrayList<IObject> objects){
        this.monsters = monsters;
        this.objects = objects;
        this.currentRoom = null;
    }

    public void addObject(IObject object){
        objects.add(object);
    }
```

```java
package Potions;

import Interfaces.IObject;

public class Potion implements IObject {

    private String type;
    private int effect;
    private int doses;

    public Potion(String type, int effect){
        this.type = type;
        this.effect = effect;
        this.doses = 4;
    }

    public int getDoses() { return doses; }

    public void setDoses(int doses) { this.doses = doses; }

    public String getType() { return type; }

    public int getEffect() { return effect; }
}
```

```java
package Weapons;

import Interfaces.IObject;

public class Weapon implements IObject {

    private String type;
    private int damage;

    public Weapon(String type, int attack){
        this.type = type;
        this.damage = attack;
    }

    public String getType() { return type; }

    public int getAttack() { return damage; }
}
```

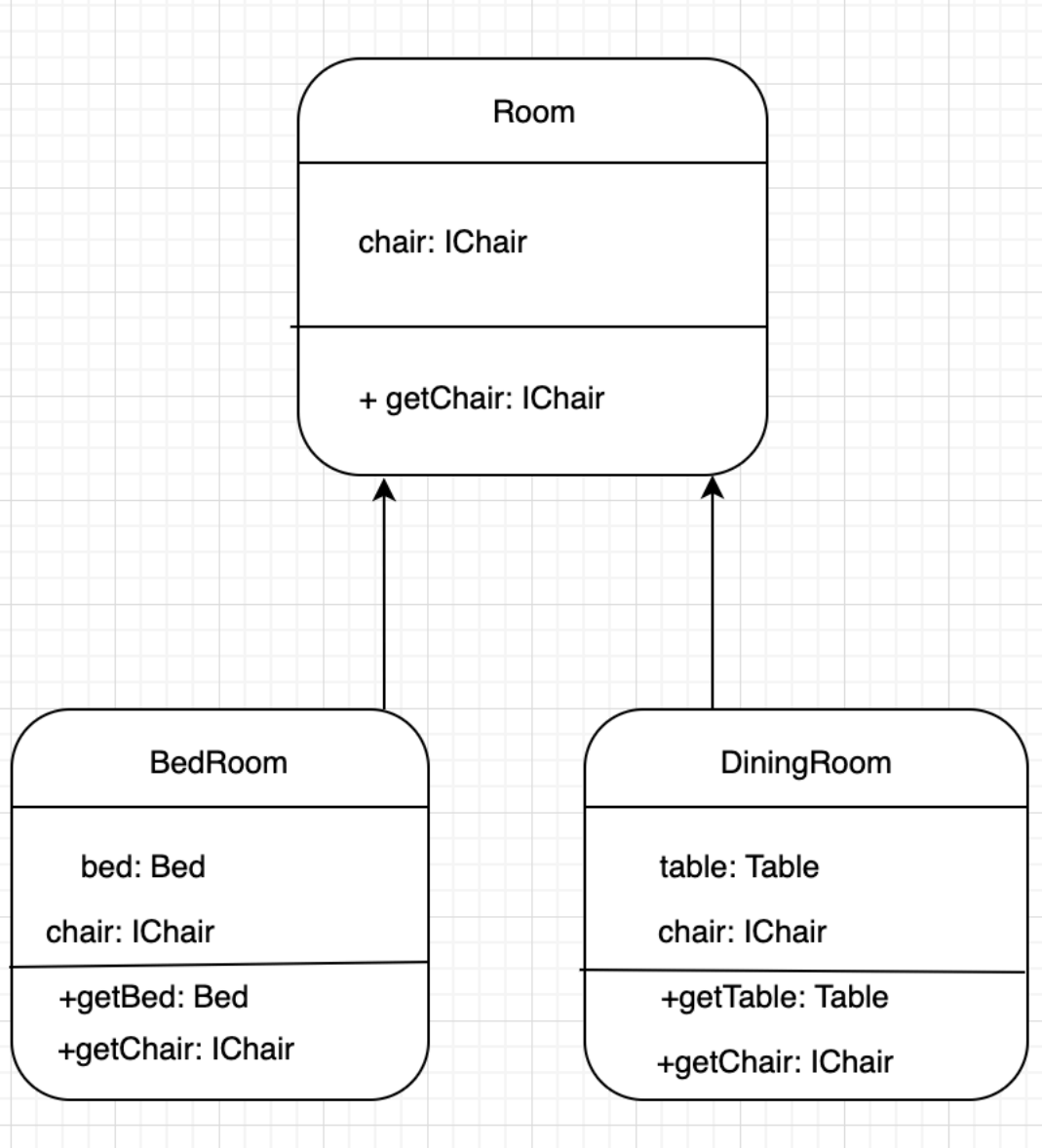
```java
package Interfaces;

public interface IObject {
    String getType();
}
```

Image 1 shows the Game class which contains an ArrayList of objects which holds items that implement the IObject interface. It also shows an addObject method which is used to add items which which implement the Object interface into the object ArrayList.

Image 2 and image 3 contain two classes which implement IObject; the Potion class and Weapon class. These are two different types of object, but they both implement the IObject interface, meaning that they can both be added to the same ArrayList.

Image 4 shows the IObject interface which contains the method getType() which will return a String.

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



Description here

The diagram above shows the inheritance of properties and methods from Room into the classes of BedRoom and DiningRoom. The BedRoom and DiningRoom will inherit the IChair object from the Room as well as the method of getChair. The classes BedRoom and DiningRoom will have their own characteristics and methods. For example BedRoom has a property of bed and has a method of getBed. DiningRoom has a property of table as well as the method of getTable.

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.

```java
package Rooms;

import Game.Game;
import Interfaces.IObject;

public abstract class Room {

    protected IObject object;

    public Room(IObject object) { this.object = object; }

    public IObject getObject() {
        return object;
    }

}
```

```java
package Rooms;

import ...

public class MonsterRoom extends Room {

    private Monster monster;

    public MonsterRoom(IObject object,Monster monster){
        super(object);
        this.monster = monster;
    }

    public Monster getMonster() { return monster; }

    public boolean isMonsterDefeated(){
        if (this.monster.getHp() > 0){
            return false;
        } else {
            return true;
        }
    }
}
```

```java
import Armour.Armour;
import Monsters.Monster;
import Rooms.MonsterRoom;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class MonsterRoomTest {

    MonsterRoom monsterRoom;
    Monster monster1;
    Armour armour;


    @Before
    public void before() {

        monster1 = new Monster( hp: 50,  type: "Orc",  attack: 4,  defence: 7);

        armour = new Armour( type: "Iron", defence: 6);

        monsterRoom = new MonsterRoom(armour,monster1);

    }

    @Test
    public void canGetObject(){
        assertEquals(armour, monsterRoom.getObject());
    }

    @Test
    public void canGetMonster() { assertEquals(monster1,this.monsterRoom.getMonster()); }

}
```

Description here

The first screenshot (top left) shows the parent class and the properties all rooms should have. Each room should contain an object of IObject.

The second screenshot (top right) is the MonsterRoom class which inherits from Room, but also has its own property of Monster. It also inherits the property of object using the 'super' keyword in the constructor.

The third screenshot show an Object of the MonsterRoom class being created (MonsterRoom monsterRoom = new MonsterRoom(armour, monster1). The object monsterRoom then uses the inherited method of getObject() from Room to find the object that is in that room.

Week 14

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.

```
def serve_customer_check(customer, drink)
  if !check_age_before_serving(customer) || customer.drunkenness >= 5
    return "We can't serve you"
  else
    sell_drink_to_customer(customer, drink)
  end
end
```

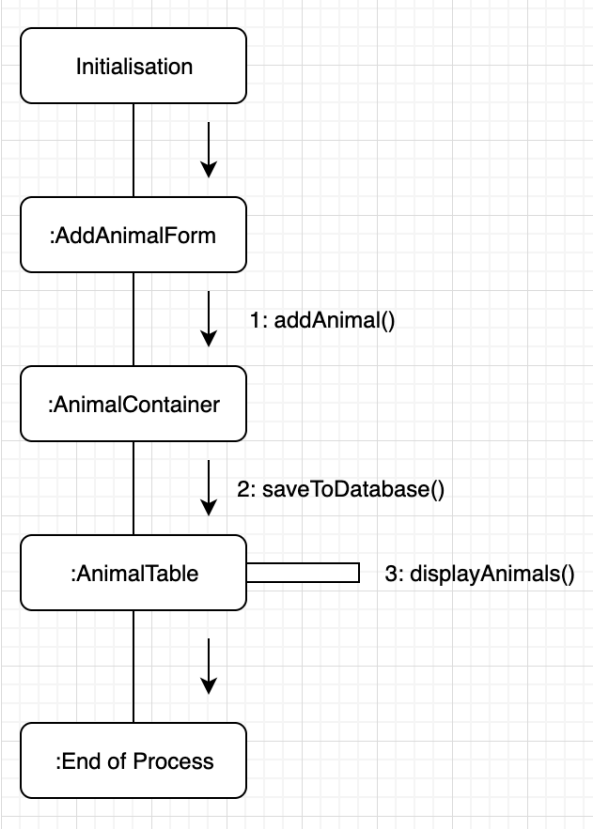
```
fetch('https://content.guardianapis.com/search?q=brexit&format=json&api-key=test')
.then(res => res.json())
.then(resjson => this.articles = resjson.response.results);
```

Description here

Image 1 is an example of the use of a logical OR operator. I used it in one of the daily homework. The algorithm checks the customers age and drunkenness level and if either of them fail to meet set criteria then a string of "we can't serve you" is returned to the customer. Otherwise it will sell a drink to a customer.

Image 2 is an example of the fetching of API data into Vue.js. I used it in a weekend homework. The algorithm starts with fetch which is a promise to request the api when the application launches. The second line takes the result of the objects within the api (res) and uses the .json method that exists within the api and returns objects that can be used by javascript. The third line takes the result of the .json method being used (resjson) and adds it to the articles array which exists within the data() return{ section of the component.

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



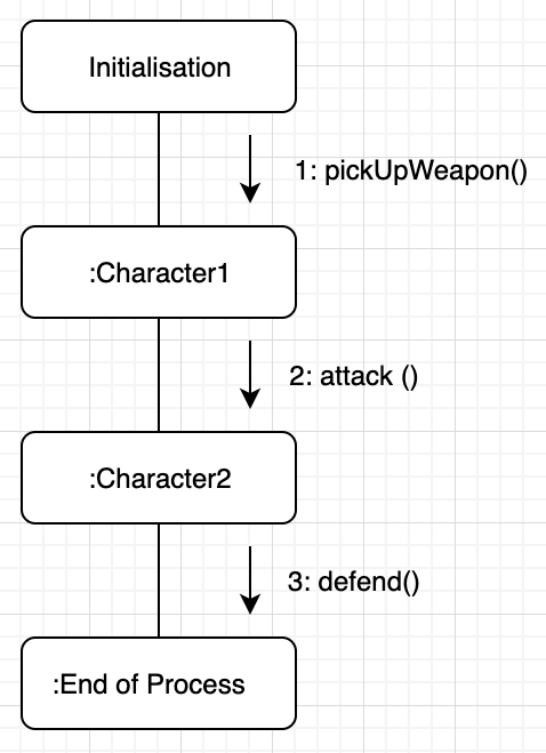
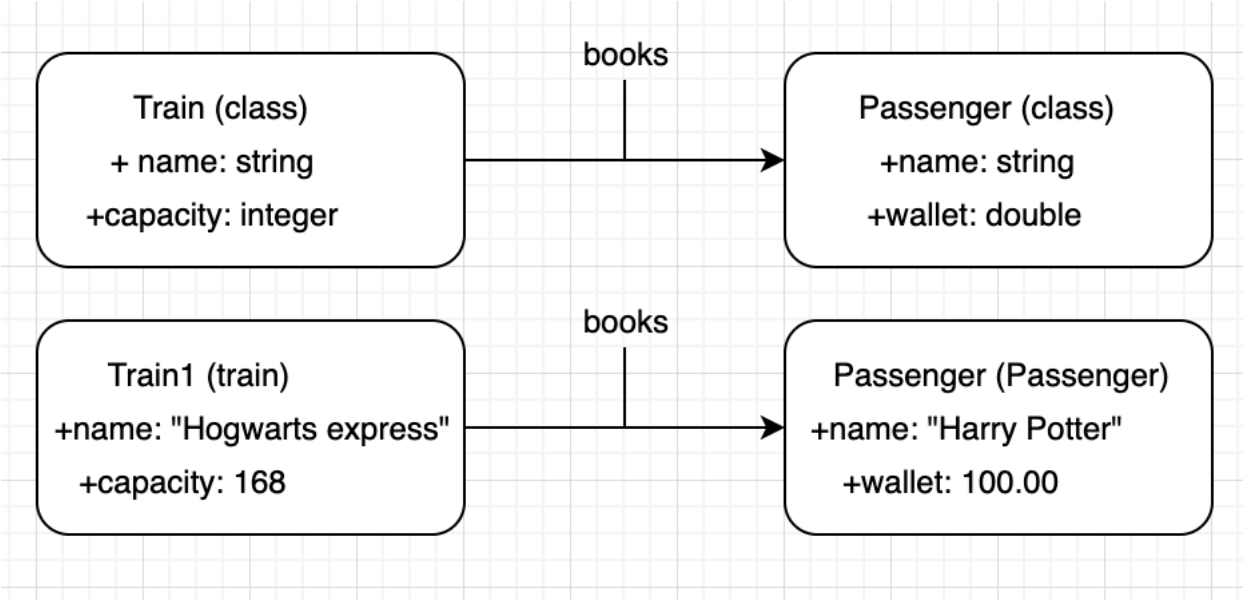
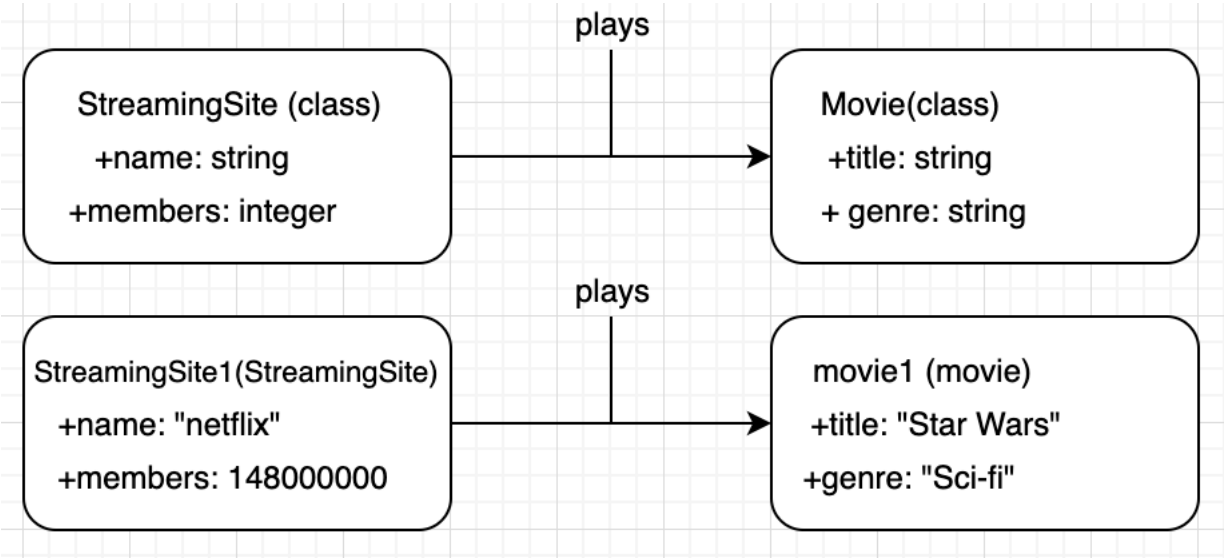


Image 1 (top left) is a collaboration diagram outlining the messages passed between objects and the order in which they occur for an admin user to add an animal to an animal shelter app.

Image 2 (top right) is a collaboration diagram from an interaction in a fantasy game. Character1 picks up a weapon and then attacks character2, who then defends from that initial attack.

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





Description here

Image 1 shows two classes; Train and Passenger. The passenger can be booked on the train. The train has a string for “name” and integer for its “capacity”. The passenger’s “name” is a string and their “wallet” is a double.

An instance of the Train class has the name “Hogwarts Express” and its capacity is 168. An instance of the passenger class has the name of “Harry Potter” and has 100.00 in his wallet.

The second image has two classes; StreamingSite and Movie. The movie can be played by the streaming service. The streaming site has a string for “name” and integer for “members”. The movie’s “title” is a string and its “genre” is a string.

An instance of the StreamingSite class has a name of “netflix” and has 148000000 members. An instance of the movie class has the title of “Star Wars” and is the genre of “Sci-fi”.

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

Bug Tracking Report

	Bug/Error	Solution	Date
1	In the LaunchSplash component, we had an issue where the timeline was off centered and was displaying in the splash page above.	In CSS within the timeline container class I added “justify-content: center” & “padding-top: 50px”, which centered the timeline and moved it down out of the splash page	19.08.19
2	In the server/DB we were getting “launch promise objects” instead of data.	In DB build_submit.js our group got the promise chain working properly: we re-ordered our promise .then statements so that they correctly reference the data portion of the object	20.08.19
3	In the server our group needed to find a range of dates; this was not working	We realised that the API was returning string and not mongod date objects. Fixing the type resolved the error.	20.08.19
4	The nav bar from the NavBar.vue was not properly implemented because we were passing the dates directly from the URL and the DB was not updating the view correctly	We needed to create another function that would return another set of dates.	20.08.19
5	The countdown that sat above our splash page was displaying NaN instead of a time remaining	We discovered that we were utilising a ‘coerce’ function that was removed from the current VUE package; we deleted that and moved the filter function from App.vue to our LaunchCountdown component. We then had to delete our computed methods and add a new mounted window.setInterval() method to fix the countdown.	21.08.19

Description here

Above is a bug tracking report for our javascript project.