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:

```
def initialize(number, destination, passengers)
  @number = number
  @destination = destination
  @passengers = passenger
end

def drive()
  return "broom broom"
end

def passenger_count
  return @passengers.count
end

def pick_up(passenger)
  @passengers << passenger
end

def drop_off(passenger)
      @passengers.delete(passenger)
end

def empty_bus
  @passengers = []
end

end
```

This program uses an array for passengers on a bus - see the @passengers defined at **bottom** of screenshot and initialised at **top** of screenshot.

In addition, this screenshot also shows a function:
(def passenger_count) which uses the passenger array. This function starts 9 lines down from the top.

```
class BusTest < MiniTest::Test

 def test_bus_has_route_number
   bus1 = Bus.new(11, "", [])
   assert_equal(11, bus1.number)

 end

 def test_bus_has_destination
   bus1 = Bus.new(11, "Princes Street", [])
   assert_equal("Princes Street", bus1.destination)

 end

 def test_bus_can_drive
   bus1 = Bus.new(35, "Riccarton", [])
   noise = bus1.drive
   assert_equal(noise, "broom broom")

 end

 def test_bus_start_empty
   bus1 = Bus.new(22, "Silverknowles", [])
   assert_equal([], bus1.passengers)
 end

 def test_count_passengers
   bus1 = Bus.new(11, "Princes Street", ["Rick", "Shabs", "John"])
   assert_equal(bus1.passenger_count, 3)
 end
```

On the left is the test file (bus_spec.rb) which has a test for the passenger array at the end - see def test_count_passengers.

When this test file is run, all 8 tests pass - see screenshot below. Please note there are 3 additional tests on the bottom not included in this screenshot.

```
bus_lab — user@users-MBP — ..day_2/bus_lab — -zsh — 80×17

........

Finished in 0.001307s, 6120.8878 runs/s, 6120.8878 assertions/s.

8 runs, 8 assertions, 0 failures, 0 errors, 0 skips
➜  bus_lab git:(master) ruby specs/bus_spec.rb
Run options: --seed 27059

# Running:

........

Finished in 0.001204s, 6644.5191 runs/s, 6644.5191 assertions/s.

8 runs, 8 assertions, 0 failures, 0 errors, 0 skips
➜  bus_lab git:(master)
```

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:

friends_spec.rb

```
1  require( 'minitest/autorun' )
2  require( 'minitest/rg')
3  require_relative( '../friends' )
4
5  class TestFriends < MiniTest::Test
6
7    def setup
8
9      @person1 = {
10       name: "Shaggy",
11       age: 12,
12       monies: 1,
13       friends: ["Velma","Fred","Daphne", "Scooby"],
14       favourites: {
15         tv_show: "Friends",
16         snacks: ["charcuterie"]
17       }
18     }
```

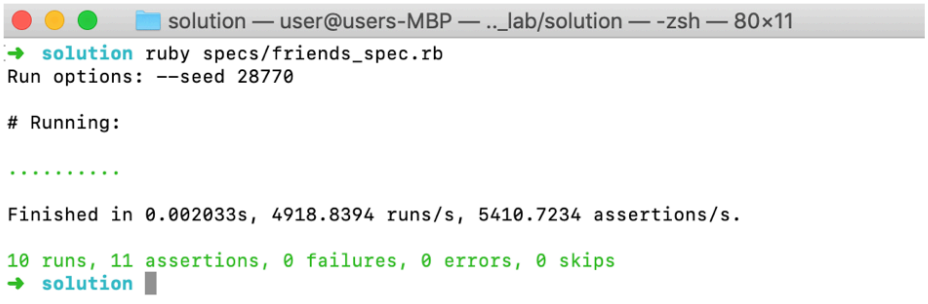
The friends_spec.rb program uses a Hash. See line 9 on the left for an example of a hash.
@person1 has **key:value** pair
name:Shaggy
age: 12
Monies: 1
Friends: Velma, Fred, Daphne, Scooby
Tv-show: Friends
Snack: charcuterie

friends_spec.rb

```
69
70   def test_getting_name
71     result = get_name(@person1)
72     assert_equal("Shaggy", result)
73   end
74
```

On the left is an example of using the Hash @person1.

Below-left is an example of the test passing (all the tests in freinds_spec.rb passed.



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: From week3 weekend homework. **Below is screenshot of selecting from customers**

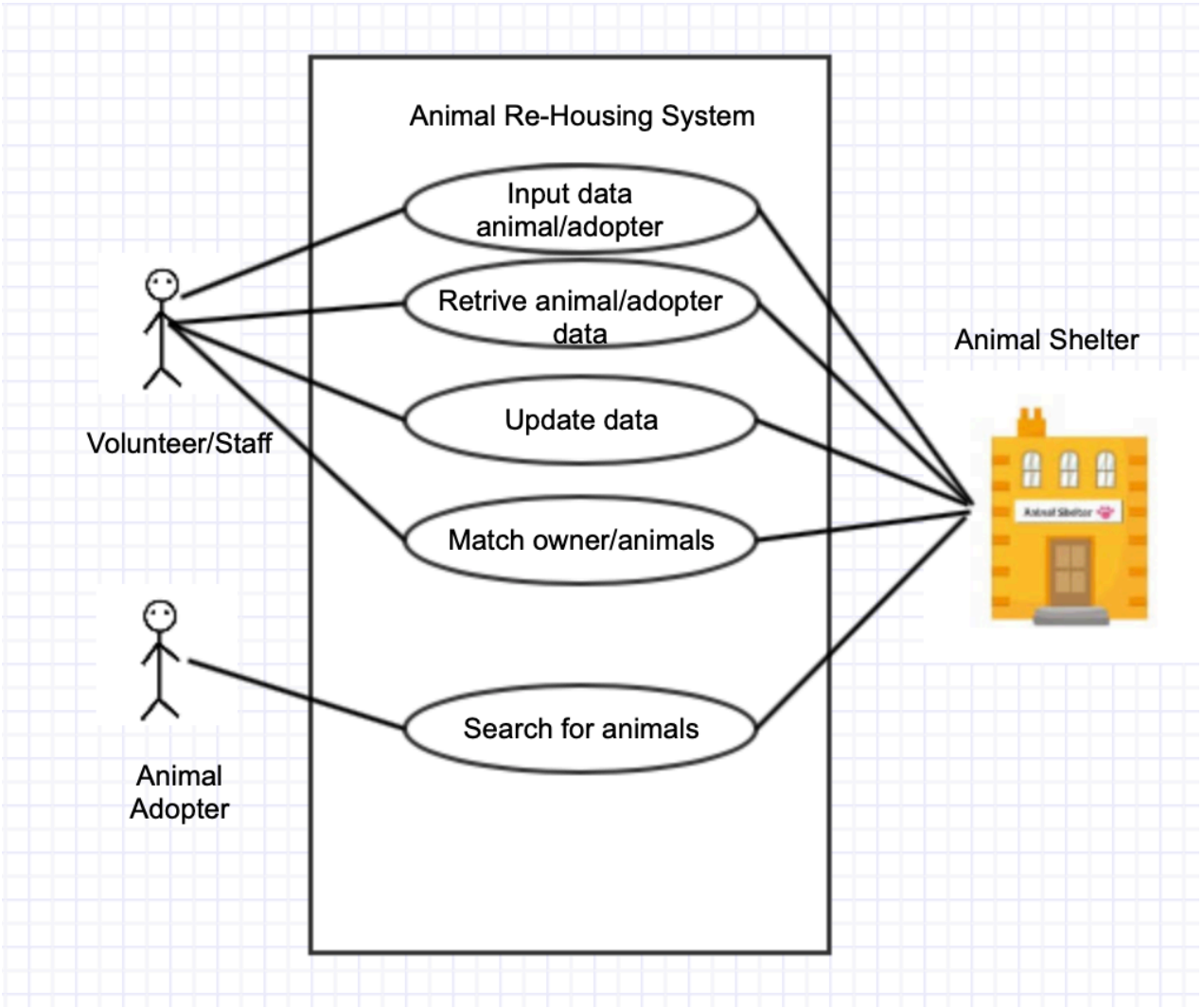
```
17
18    def customer()
19      sql = "SELECT * FROM customers WHERE id = $1"
20      values = [@customer_id]
21      result = SqlRunner.run(sql, values)
22      customer_hash = result[0]
23      return Customer.new( customer_hash)
24      # above linking film with customer.
25    end
26
```

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: From week 3, weekend homework. The function below uses the map function to sort out films.

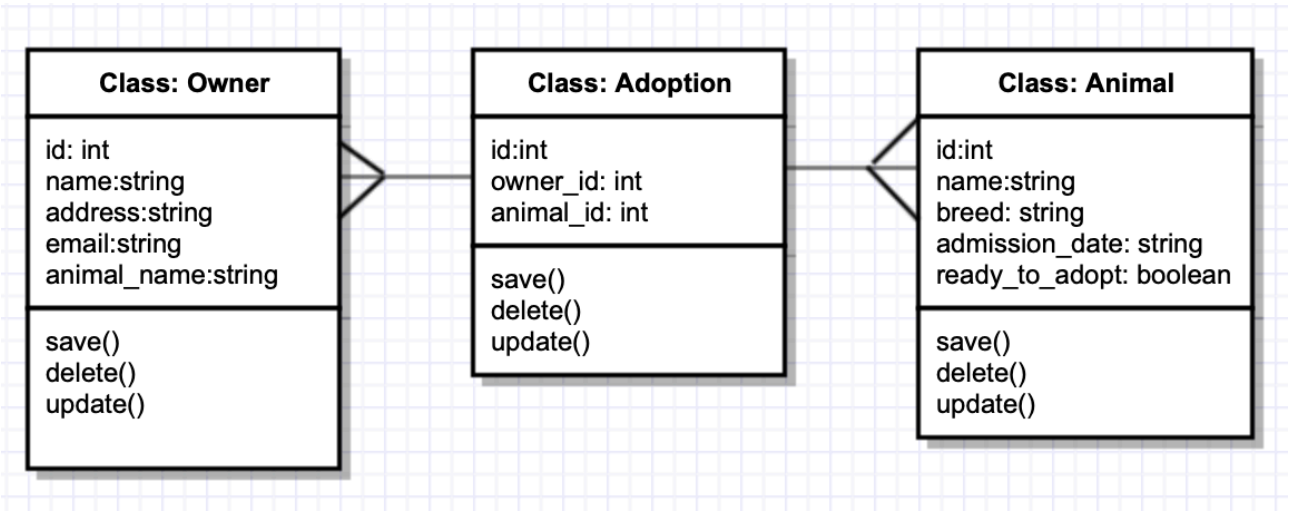
```
70
71    def self.all()
72      sql = "SELECT * FROM film"
73      customers = SqlRunner.run(sql)
74      return film.map { |film|
75        Film.new(film) }
76    end
```

Week 5 and 6

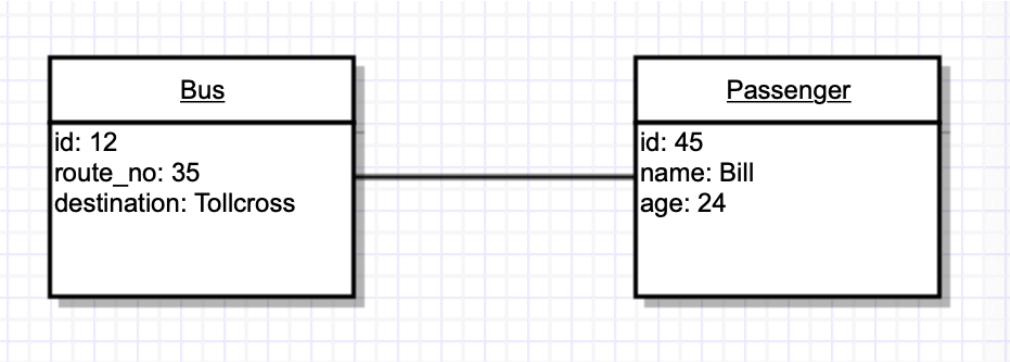
Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram
		Description: The Animal Re-Housing System used by the animal shelter has a database of animals and potential owners. Animals/owners are matched up by staff. Animal adopter are able to view animals available.



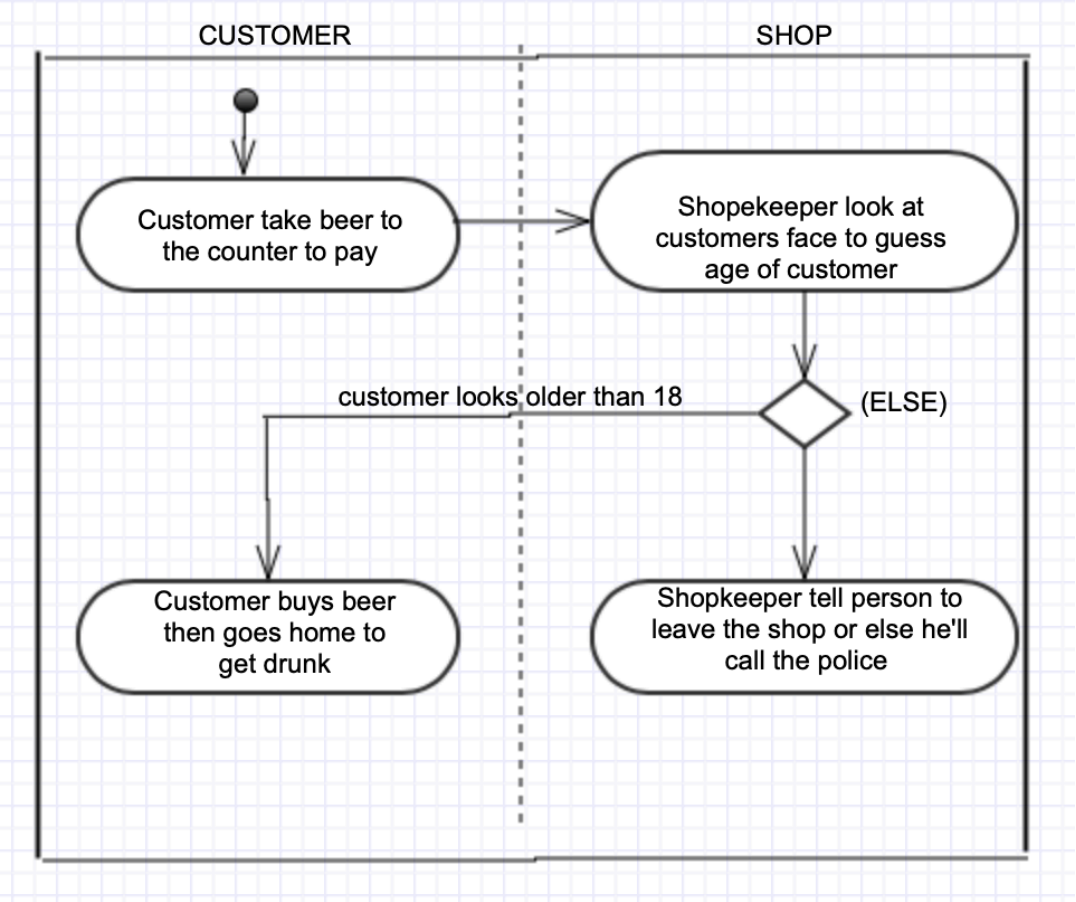
Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram
		Description: Below is the class diagram for animal shelter. Its a many-to-one and one-to-many releationship.



Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram
		Description: Object diagram of Bus and Passenger objects



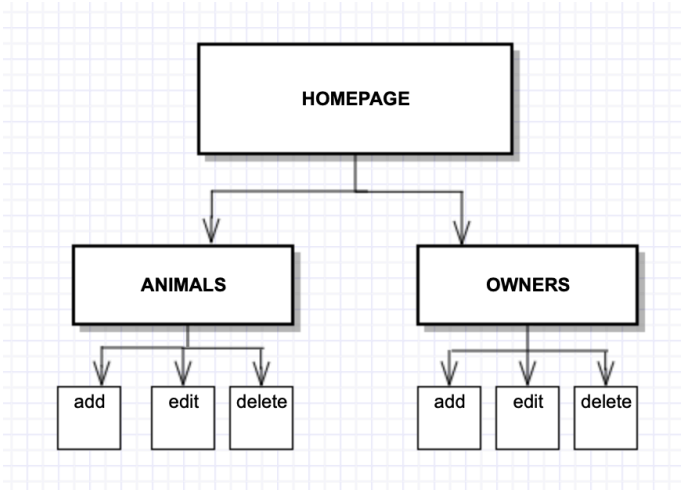
Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram
		Description: Activity of customer and shopkeeper, decision based on age of customer.



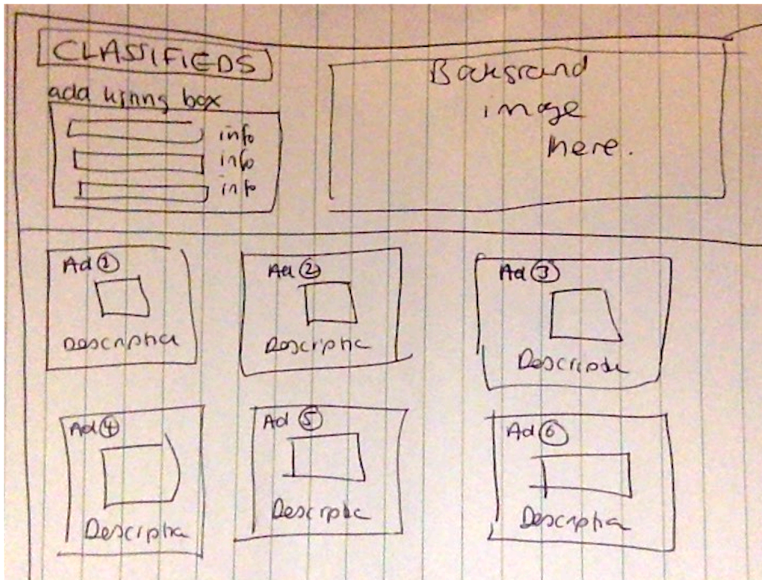
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time
		Description: See table below

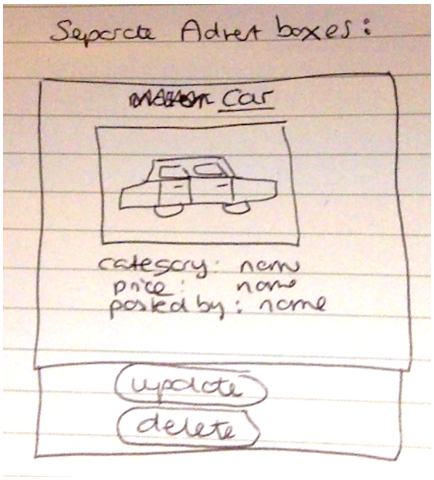
Constraint Category	Implementation Constraint	Solution
Hardware and Software Platforms	App does not run on device	Ensure correct platform is available to run the app on this (and other) devices
Performance Requirements	The app runs slowly and with has error messages	Make sure program has been tested properly and errors resolved before release of App
Persistent Storage and Transactions	Space for storing data is limited or running out	Ensure enough storage space is available to run App
Usability	User interface is complicated to operate	Use UX design to improve UX features.
Budgets	Cost of using device is expensive due to high energy usage/large storage space	Re-analyse energy usage/storage space and improve design for those constraints.
Time Limitations	More time needed to produce good App	Use Agile system, short sprints and monitor progress on a more regular basis.

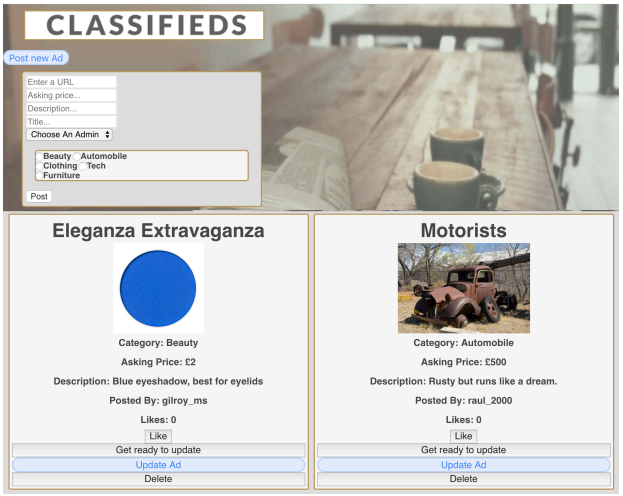
Unit	Ref	Evidence
P	P.5	User Site Map
		Description: The user site map below shows that the user can move to different pages. From the different pages (animals and owners) they have access to add, edit and delete functions.



Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description: Wireframe of homepage, wireframe of individual advert (car). For comparison, an actual screenshot of the homepage is also included at end.



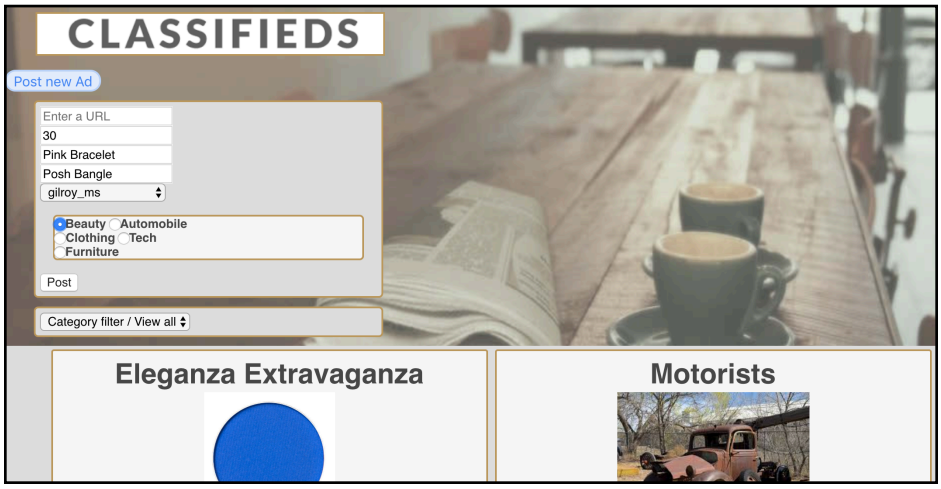


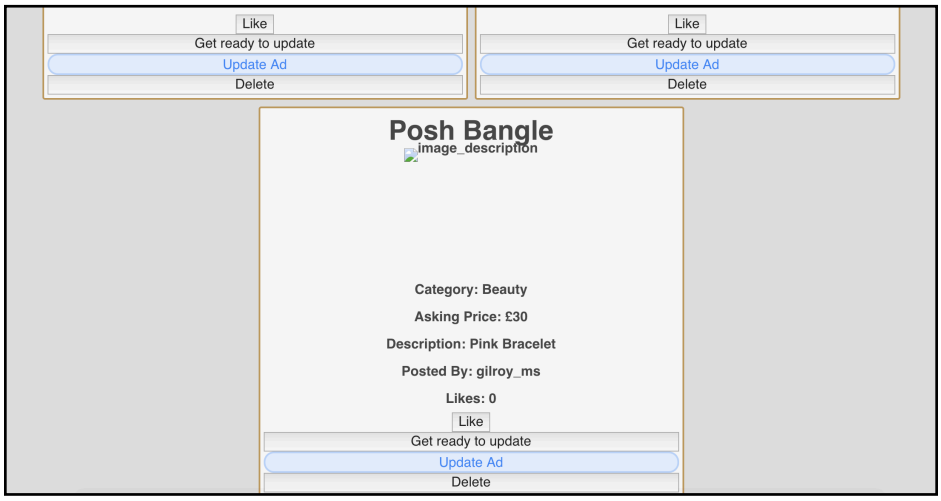


Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		Description: See the example below from cinema homework week3

```
43
44  def films()
45    sql = "SELECT films.*
46    FROM films
47    INNER JOIN screenings
48    ON films.id = screenings.film_id
49    INNER JOIN tickets
50    ON tickets.screening_id = screenings.id
51    WHERE tickets.customer_id = $1"
52    values = [@id]
53    film_data = SqlRunner.run(sql, values)
54    return Film.map_items(film_data)
55  end
```

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: First screenshot - data being inputted. **Second screenshot - data (advert) now displayed**

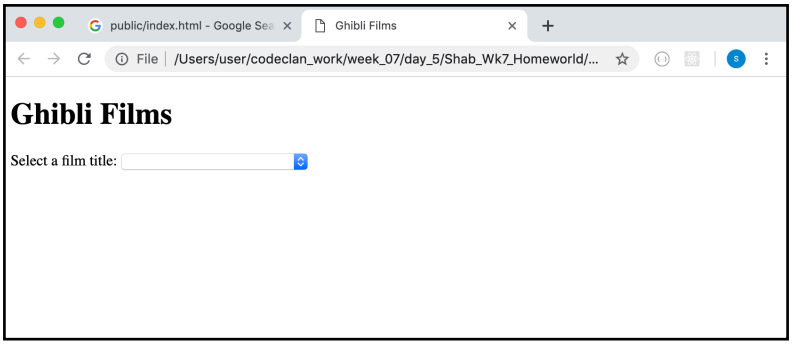


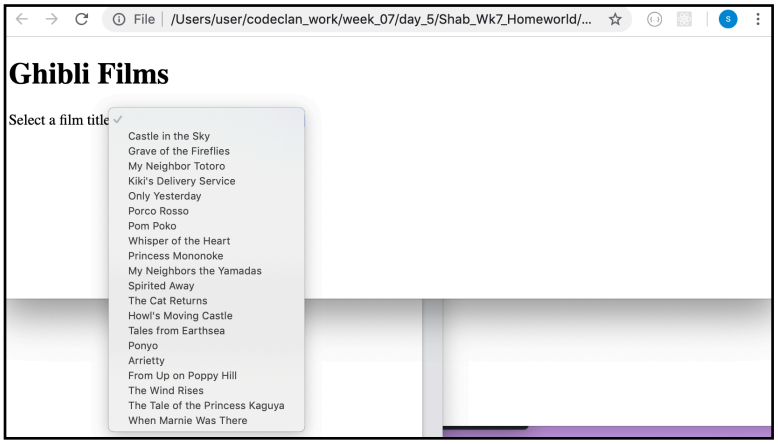


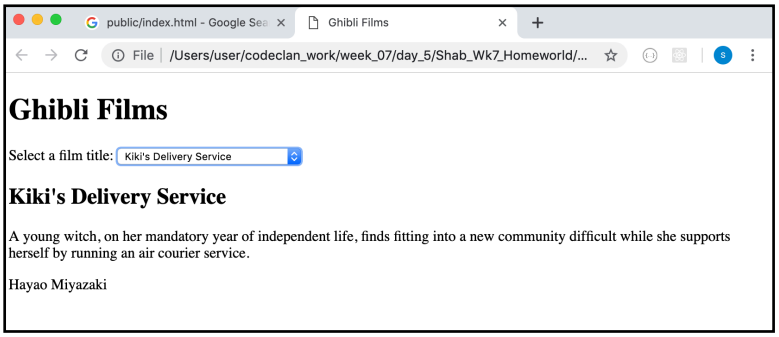
Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved
		Description:

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:

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: Week7 weekend homework - I worked alone on this. **Github link: https://github.com/GitHubShabs/Wk7WkendHomeworkGhibli**



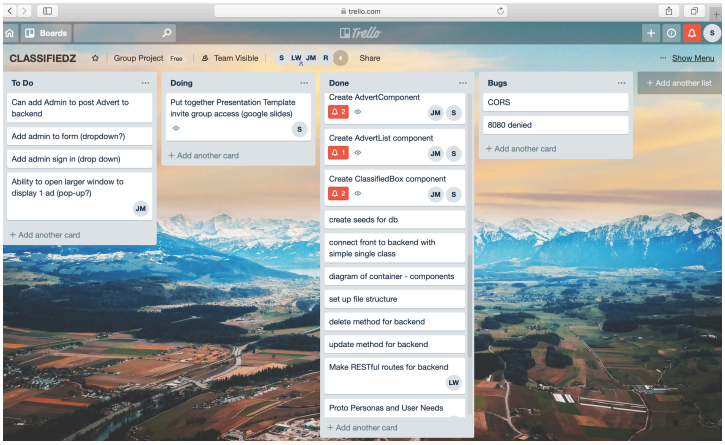


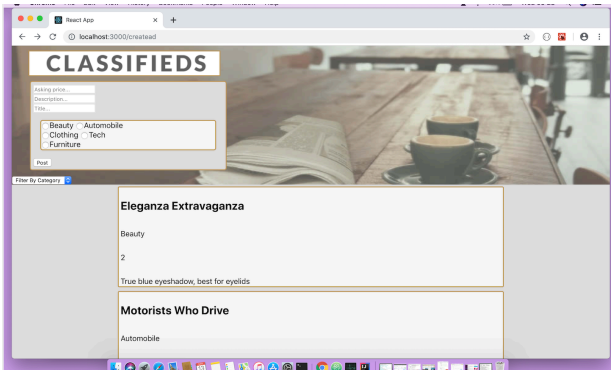


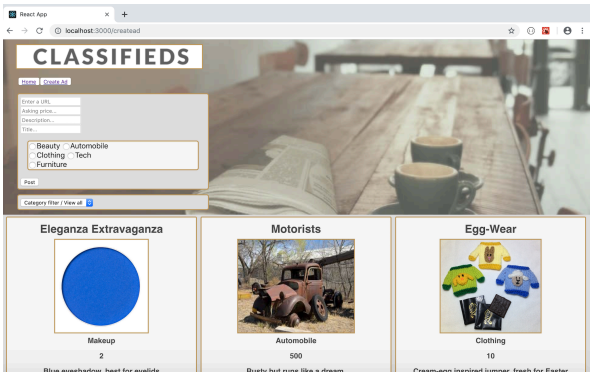
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description: Initial User Persona. Trello board, Evolution of Homepage (bottom two screenshots).

User Persona - Classified Ads App

	 Lulu Zippy (32) Software Engineer	 Sebastian Ford (29) Fashion Designer	 Albert Okeke (69) Retired Car Mechanic
Behaviours	• Lives alone in large house • Inherited house from rich parents • Loves drinking and fine dining • Spends most evenings out	• Shares flat with 3 flatmates • Likes to keep up to date with latest fashion trends • Spends a lot of time online looking at designer clothes etc	• Lives with wife and 3 cats • Hobby is repairing/selling old cars and motorbikes • Introduced to internet last year and spends a lot of time on Ebay
Demographic Information	• 32 year old Software Engineer • Lives in central Glasgow • Earns £40k/year	• 29 year old fashion designer • Owns posh boutique in Mayfair London. • Earns £32k/year	• 69 year old retired mechanic • Runs a small local magazine called "*Cars and Stuff*" • Lives in Hull
Needs and Goals	• Needs a simple to use app for selling family antique furniture • Needs a record of everything sold for future reference • Want to arrange items into simple categories for easy sale	• Wants an app with ability to display bright colourful photos • Layout must be modern to attract a younger clientele • Categories should included makeup and clothes	• Wants an app to sell his finished cars, motorbikes and parts • Likes simplicity and hates small print • Must have clear instructions and easy to use

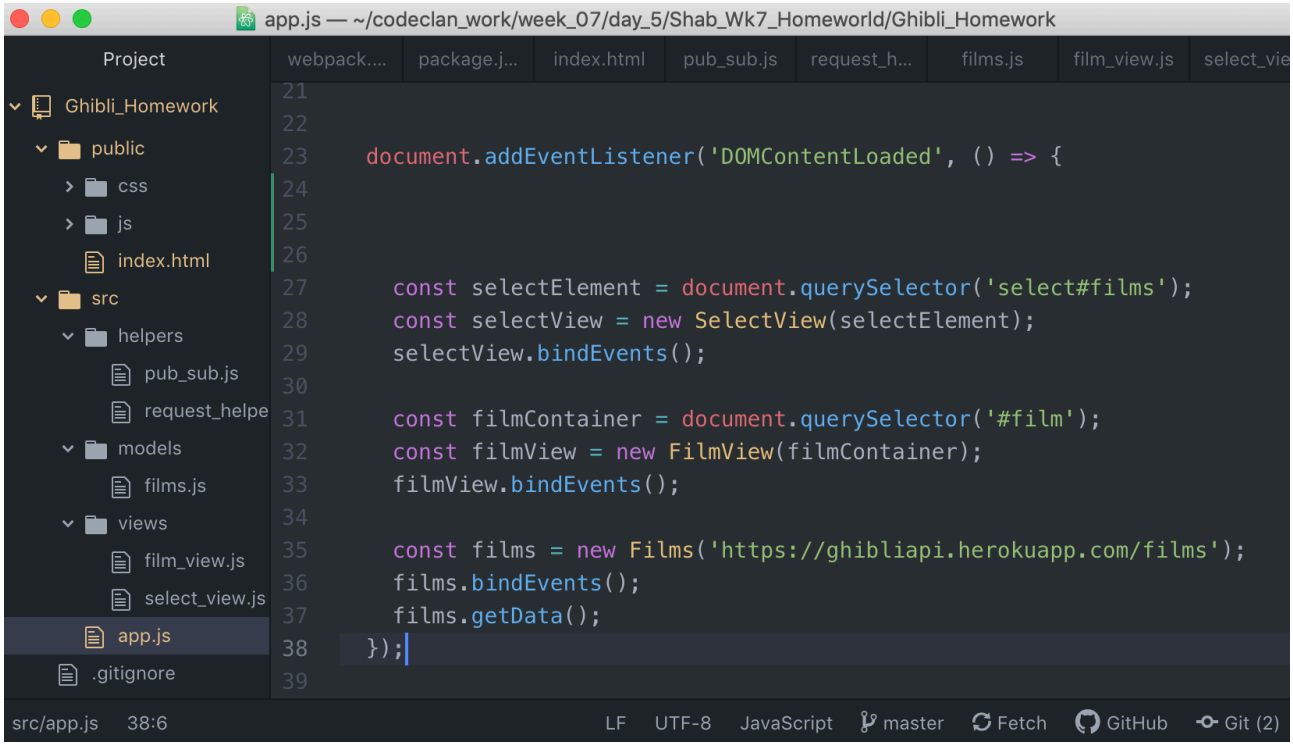


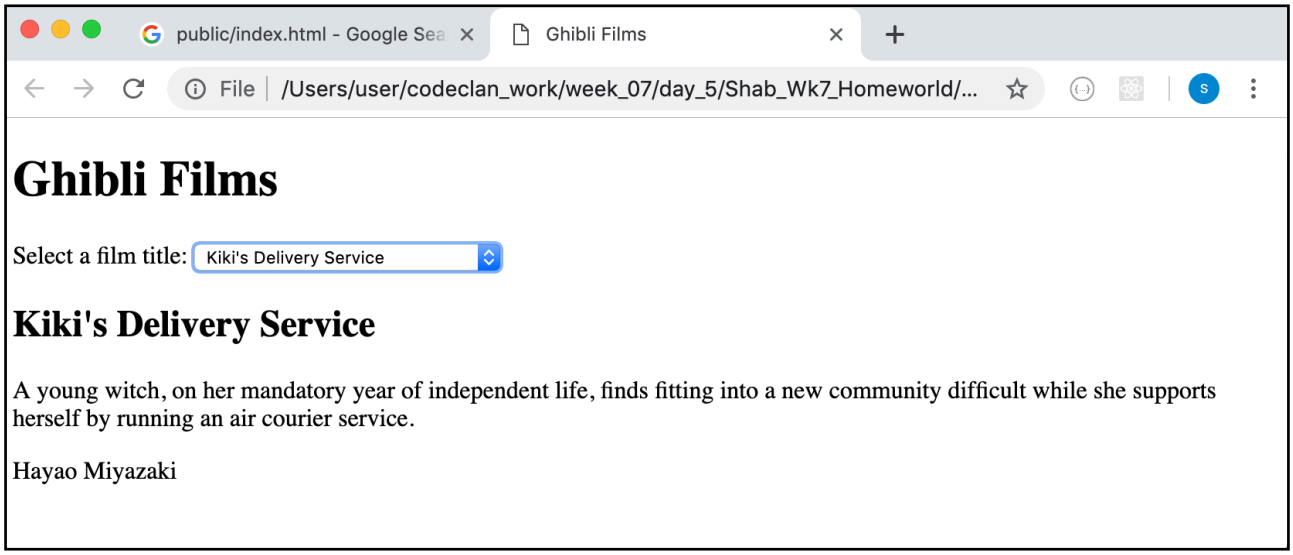




Week 7

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: Week7 weekend homework uses API - Ghibli Homework **Code that implements the API is on first screenshot below.**





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: See code below: food_spec.rb

food_specs.rb

```ruby
require("minitest/autorun")
require("minitest/rg")

require_relative("../food")

class FoodTest < MiniTest::Test

  def setup

    @food1 = Food.new("Kebab", 10.00, 2)

  end

def test_food_has_name
  assert_equal("Burger", @food1.food_name)
end

def test_food_has_price
  assert_equal(10.00, @food1.food_price)
end

def test_food_has_rejuvenation_level
  assert_equal(2, @food1.rejuvenation_level)
end
```

This is an example of test code. Its from Pub Lab (week 2/day3). It has 3 separate tests:
1. Test food has name
2. Test food has price
3. Test food has rejuvenation level.

When I ran this test, it had one fail (see below). It failed because @food1 was named as Kebab but was tested as Burger in the first test. Hence there was a mis-match i.e. actual was different from expected result for @food1.

```
week2_day3_pub_lab — user@users-MBP — .._day3_pub_lab — -zsh — 80×13
# Running:

.F.

Finished in 0.001256s, 2388.5354 runs/s, 2388.5354 assertions/s.

  1) Failure:
FoodTest#test_food_has_name [specs/food_specs.rb:15]:
Expected: "Burger"
  Actual: "Kebab"

3 runs, 3 assertions, 1 failures, 0 errors, 0 skips
➜  week2_day3_pub_lab git:(master) ✗
```

~/codeclan_work/week_02/day_3/pub_lab/week2_day3_pub_lab

food_specs.rb

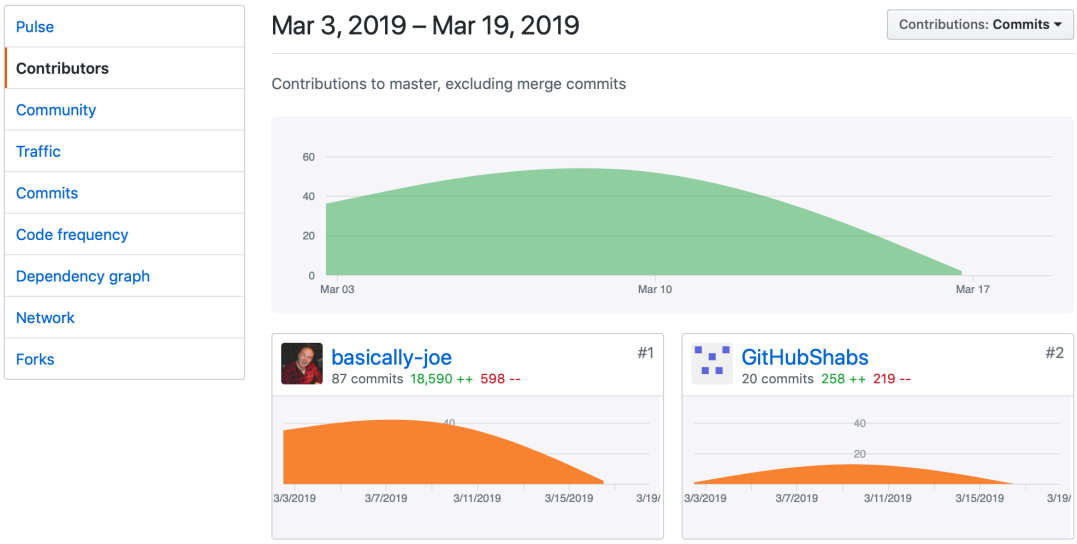
```ruby
require("minitest/autorun")
require("minitest/rg")

require_relative("../food")

class FoodTest < MiniTest::Test

  def setup

    @food1 = Food.new("Kebab", 10.00, 2)

  end

def test_food_has_name
  assert_equal("Kebab", @food1.food_name)
end
```

When this error was corrected i.e. Burger changed to Kebab on the first test (see screenshot on the left), the test actually passed - see the screenshot below, all 3 runs passed.

```
week2_day3_pub_lab — user@users-MBP — .._day3_pub_lab — -zsh — 80×13
3 runs, 3 assertions, 1 failures, 0 errors, 0 skips
➜  week2_day3_pub_lab git:(master) ✗ ruby specs/food_specs.rb
Run options: --seed 12425

# Running:

...

Finished in 0.000934s, 3211.9910 runs/s, 3211.9910 assertions/s.

3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
➜  week2_day3_pub_lab git:(master)
```

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:



Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.
		Description: See brief below which includes the MVP and Extensions (all completed).

Group Project Brief : Classified Ads

You are required to write a web application for a company listing classified ads.
The company would like to be able to create and post adverts to their listings.

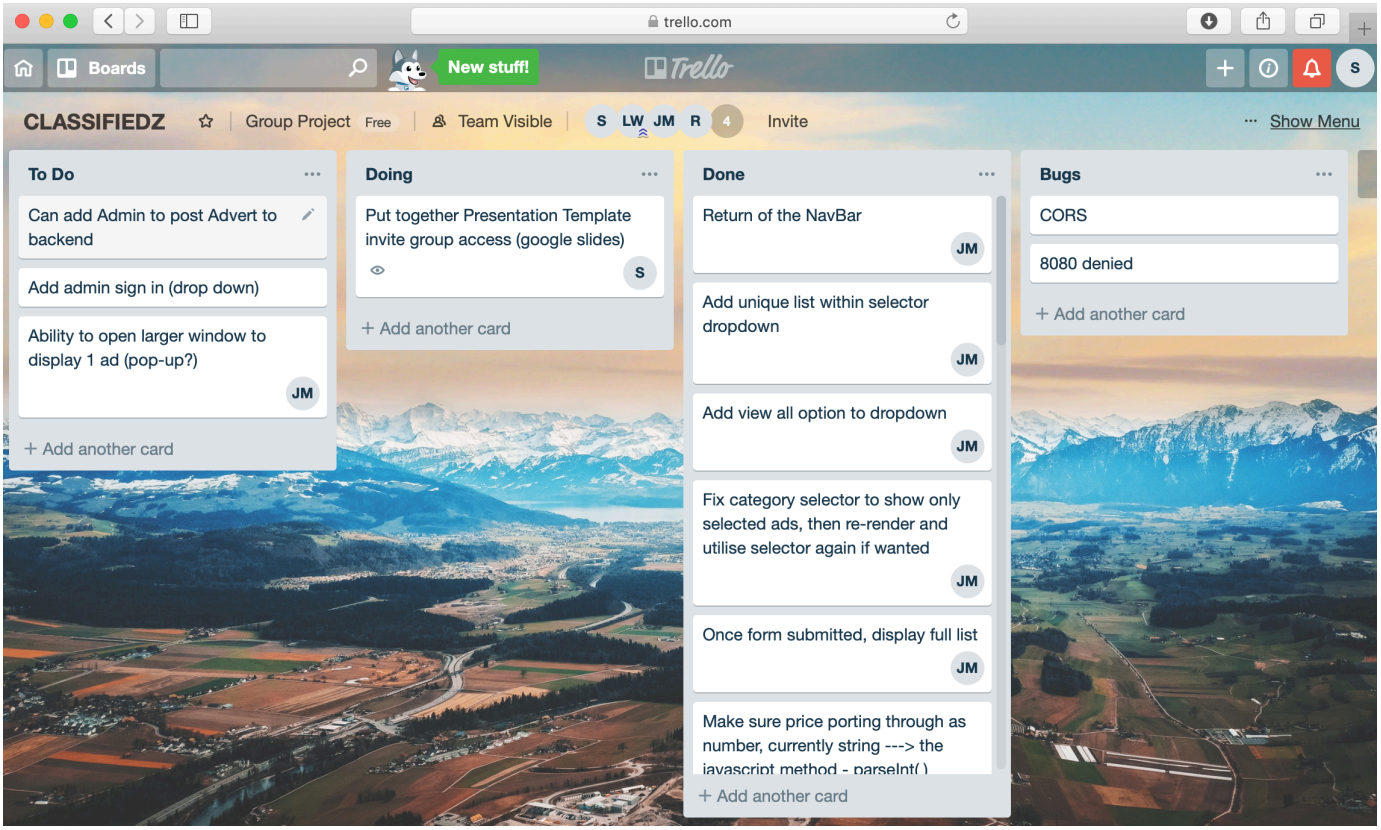
MVP:

- View all adverts posted ✅
- View all adverts in a category ✅
- Create and post adverts to specific categories ✅
- Delete adverts ✅

Project Extensions:

- Each advert contains a ‘likes’ button - displays total likes for particular item ✅

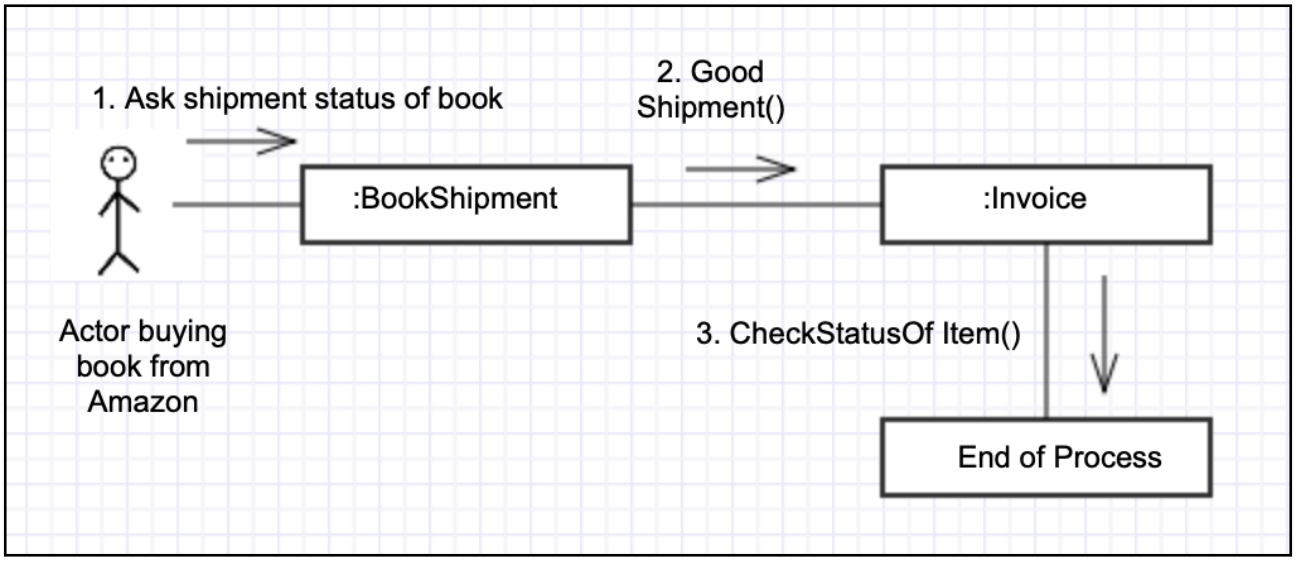
Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.
		Description: We used a troll board for the group project - see below:

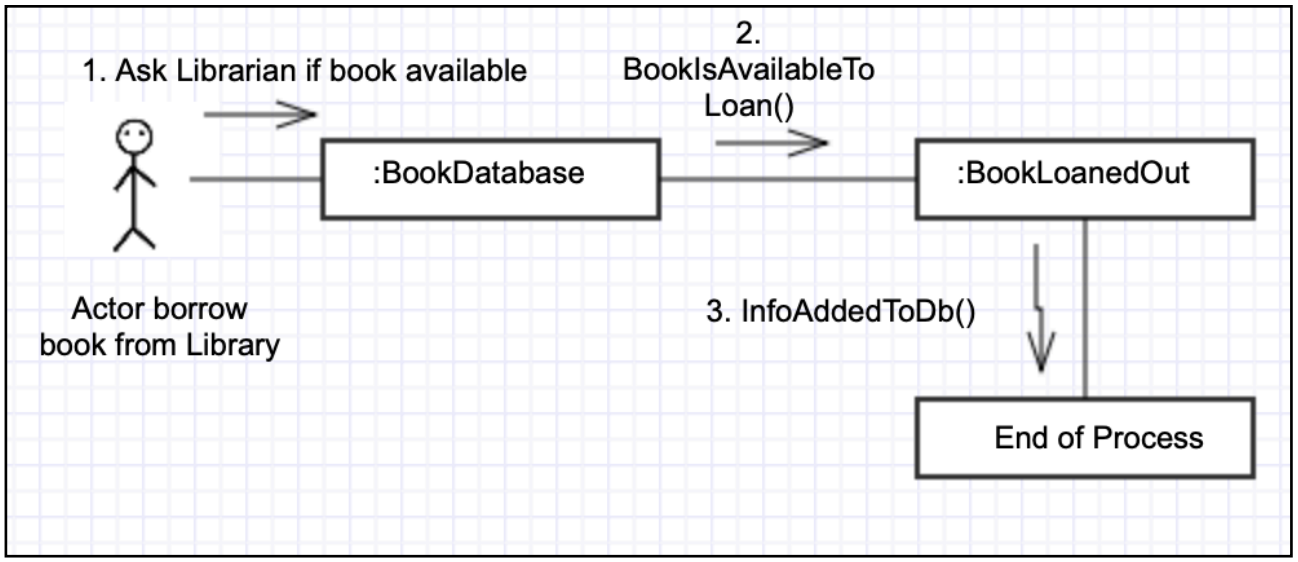


Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.
		See Below:

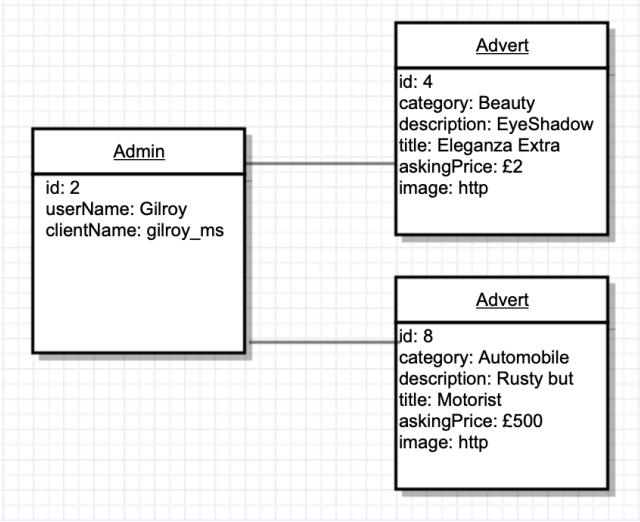
Acceptance Criteria	Expected Result/Outcome	Pass or Fail
A user is able to view all the adverts in the list	Each advert is visible on the main homepage. This means the database has been seeded with advert data.	Pass
The user add new adverts to the list	The user should be able to add title, category, asking price and description to each advert.	Pass
The user can delete adverts	The user should be able to delete an advert from the homepage.	Pass
The user can update adverts	The user should be able to update/modify one or many of the features of the advert e.g. title, category, description, price.	Pass

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





Unit	Ref	Evidence
P	P.8	Produce two object diagrams.
		Description: See object diagram used for group project - Classified Ads



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

Error (Bug details)	Solutions/Correction	Date of Error
Failed to compile	Forgot to add export default Navbar on bottom of NavBar.js file. Export added to file and now app works	13 March 2019
Failed to compile	Typo in the background image (index.css). The spelling was changed from inthenews.jpeg to inthenews.jpg. Image was restored and error message disappeared.	14 March 2019
Description in adverts squashed & overlapped and hard to read.	Adjusted the max-width of individual-ad-box-column size from 350px to 500px. Now the box size is bigger and you can read the text properly.	14 March 2019

Week 12

Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description:Wk12D4 lab has examples of polymorphism. The Cleric inherits IHeal, IHeal can run the function canHeal(). Hence the Cleric take on the character (function) of healing.

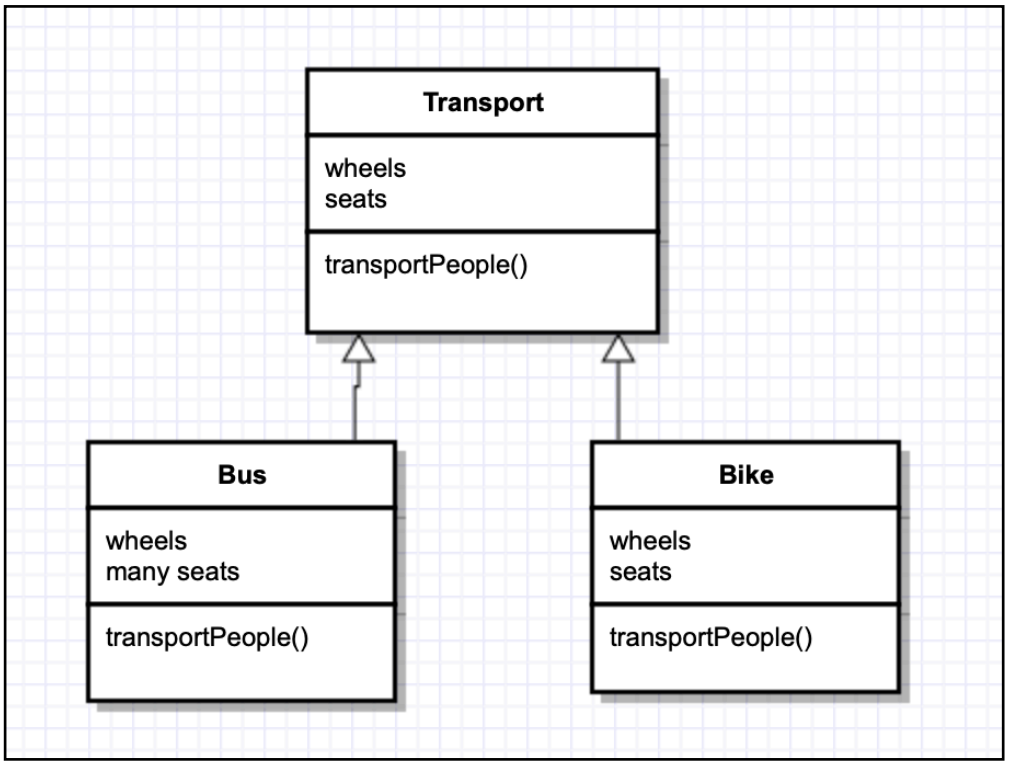
Cleric.java

```java
package Heal;

public class Cleric implements IHeal {
    @Override
    public String canHeal() {
        return null;
    }
}

```

IHeal.java

```java
public interface IHeal {

    public String canHeal();
}

```

Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram
		Description: Using inheritance the child classes (Bus & Bike) take on the parents function of transportPeople() and has wheels and seats (from Transport - parent).



Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description: Is used to protect data (access) and only allow access using getters and setter. Example from wk12d2 theme park homework. As name is private String, it can only be accessed with public String getName() i.e. a getter.

```
public abstract class Stall implements ITicketed, IReviewed, ISecurity {
    private String name;
    private String ownerName;
    private int parkingSpot;

    private int rating;

    public Stall(String name, String ownerName, int parkingSpot, int rating)
        this.name = name;
        this.ownerName = ownerName;
        this.parkingSpot = parkingSpot;
        this.rating = rating;
    }


    public String getName() {
        return name;
    }

    public String getOwnerName() {
        return ownerName;
    }

    public int getParkingSpot() {
        return parkingSpot;
    }

    public int getRating(){
        return this.rating;
```

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: A class: Weapon - screenshot 1. **Sword inherits from Weapon - screenshot 2.** **Sword (inherited class) can “stab” - screenshot 3.** **The Weapon class can do damage (int) and this is method is passed to Sword - last screenshot.**

```java
package Attack;

public abstract class Weapon {

    int damage;

    public Weapon(int damage) {
        this.damage = damage;
    }

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

```java
package Attack;

public class Sword extends Weapon {

    String action;

    public Sword(int damage, String action)
        super(damage);
        this.action = "stab";
    }

    public String getSwordAction() {
        return action;
    }
```

```java
package Attack;

public class Sword extends Weapon {

    String action;

    public Sword(int damage, String action)
        super(damage);
        this.action = "stab";
    }

    public String getSwordAction() {
        return action;
    }
```

```java
package Attack;

public abstract class Weapon {

    int damage;

    public Weapon(int damage) {
        this.damage = damage;
    }

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

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: First one from wk11d3 lab, the last function makes sure each room does NOT have more than the allocated room capacity, rooms have different number of beds (single, double, etc). 2nd: wk11d4 lab, I really like the last algorithm because is MAKES a whole deck of cards using ENUMS Rank type and SuitType in the perfect combination to make a full deck of cards (small two screenshots, bottom right of page).

ConferenceRoom.java

```java
import java.util.ArrayList;

public class ConferenceRoom {

    private String roomName;
    private int roomCapacity;
    private ArrayList<Guest> guests;

    public ConferenceRoom(String roomName, int roomCapacity){
        this.roomName = roomName;
        this.roomCapacity = roomCapacity;
        this.guests = new ArrayList<>();
    }

    public void addGuest(Guest guest){
        this.guests.add(guest);
    }
    public void removeGuest(Guest guest){
        this.guests.remove(guest);
    }
    public boolean hasSpace(){
        return this.guests.size() < this.roomCapacity;
    }
```

```java
import java.util.ArrayList;
import java.util.Collections;

public class Deck {

    private ArrayList<Card> cards;

    public Deck() {
        this.cards = new ArrayList<>();
    }

    public void addCardToDeck(Card card) {
        this.cards.add(card);
    }

    public int cardCount() {
        return this.cards.size();
    }


    public void createDeck(){
        for (SuitType suit : SuitType.values()){
            for(RankType rank : RankType.values()){
                Card card = new Card(suit, rank);
                this.cards.add(card);
```

```java
public enum SuitType {
    HEARTS,
    DIAMONDS,
    SPADES,
    CLUBS,
}
```

```java
public enum RankType {
    ACE(1),
    TWO(2),
    THREE(3),
    FOUR(4),
    FIVE(5),
    SIX(6),
    SEVEN(7),
    EIGHT(8),
    NINE(9),
    TEN(10),
    JACK(11),
    QUEEN(12),
    KING(13);
```