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

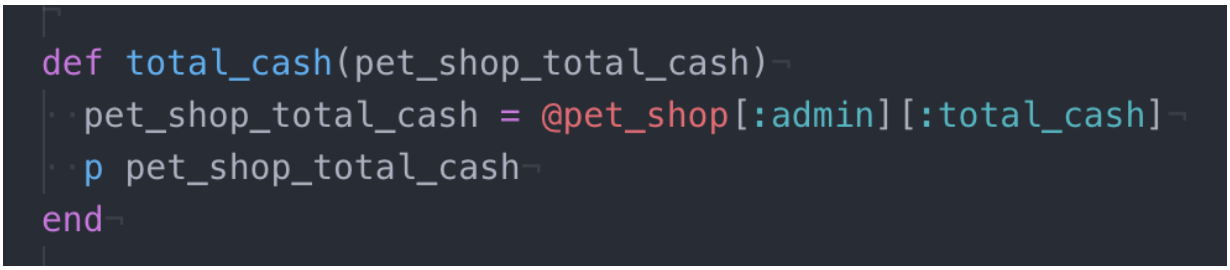


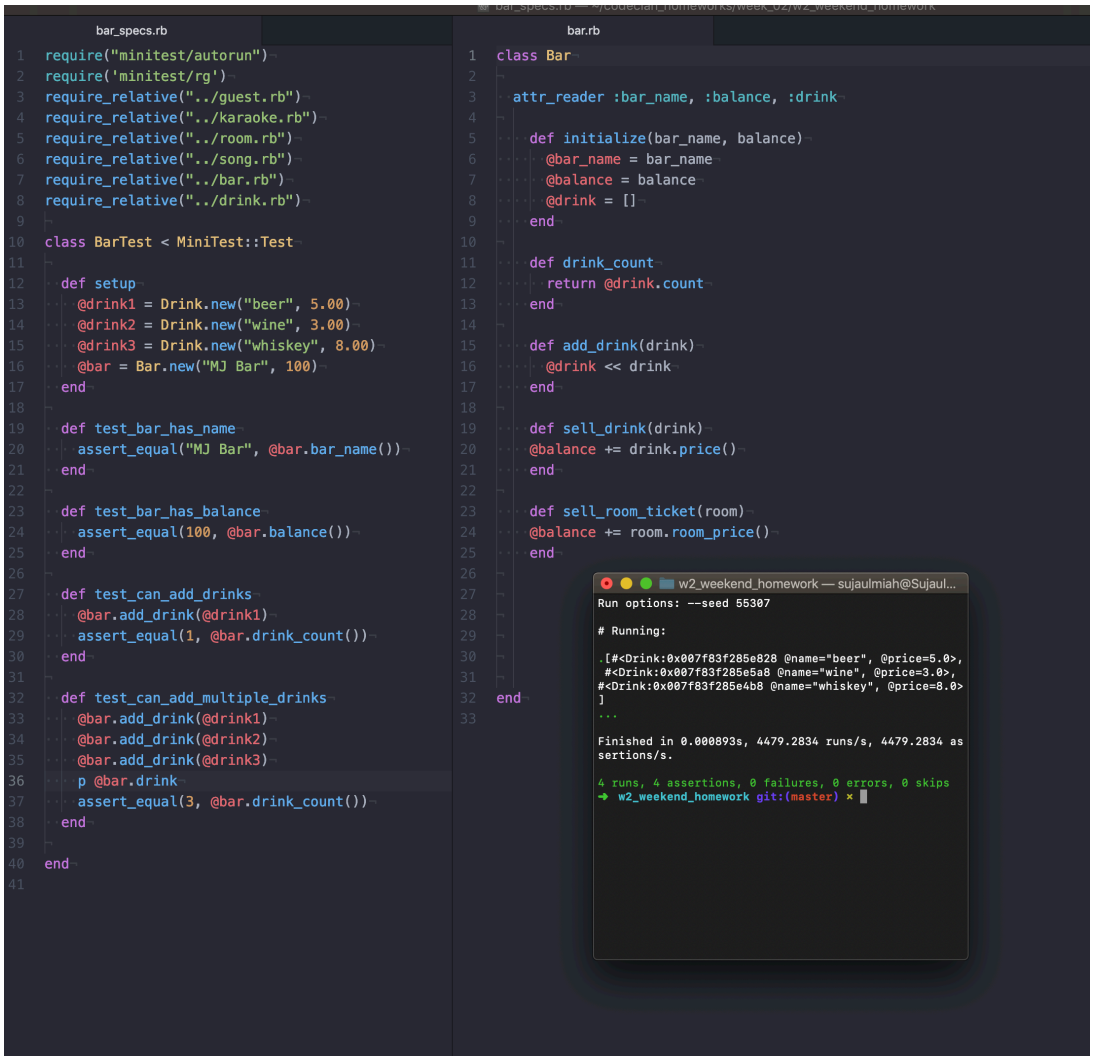
Image 1 has the has for admin which has two keys and two values -> total_cash with the value 1000 and pets_sold with the value 0.

Image 2 contains the function that uses the hash. The function will access the pet shop instance, then access the admin hash with [:admin] then access the value with [:total_cash].

Image 3 is the result showing 1000 for the total cash the pet shop has

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



bar_specs.rb

```
require("minitest/autorun")
require('minitest/rg')
require_relative("../guest.rb")
require_relative("../karaoke.rb")
require_relative("../room.rb")
require_relative("../song.rb")
require_relative("../bar.rb")
require_relative("../drink.rb")

class BarTest < MiniTest::Test

  def setup
    @drink1 = Drink.new("beer", 5.00)
    @drink2 = Drink.new("wine", 3.00)
    @drink3 = Drink.new("whiskey", 8.00)
    @bar = Bar.new("MJ Bar", 100)
  end

  def test_bar_has_name
    assert_equal("MJ Bar", @bar.bar_name())
  end

  def test_bar_has_balance
    assert_equal(100, @bar.balance())
  end

  def test_can_add_drinks
    @bar.add_drink(@drink1)
    assert_equal(1, @bar.drink_count())
  end

  def test_can_add_multiple_drinks
    @bar.add_drink(@drink1)
    @bar.add_drink(@drink2)
    @bar.add_drink(@drink3)
    p @bar.drink
    assert_equal(3, @bar.drink_count())
  end

end
```

bar.rb

```
class Bar

  attr_reader :bar_name, :balance, :drink

    def initialize(bar_name, balance)
      @bar_name = bar_name
      @balance = balance
      @drink = []
    end

    def drink_count
      return @drink.count
    end

    def add_drink(drink)
      @drink << drink
    end

    def sell_drink(drink)
    @balance += drink.price()
    end

    def sell_room_ticket(room)
    @balance += room.room_price()
    end


end
```

```
w2_weekend_homework — sujaulmiah@Sujaul...
Run options: --seed 55307

# Running:

.[#<Drink:0x007f83f285e828 @name="beer", @price=5.0>,
 #<Drink:0x007f83f285e5a8 @name="wine", @price=3.0>,
#<Drink:0x007f83f285e4b8 @name="whiskey", @price=8.0>
]
...

Finished in 0.000893s, 4479.2834 runs/s, 4479.2834 as
sertions/s.

4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
➜  w2_weekend_homework git:(master) ✗
```

The file bar.rb has an array for the drinks the bar has to offer which is on line 8 “@drink”. The function add_drink line 15 is called when a drink is to be added to the array “@drink”. In the bar_specs file we have three drinks created “@drink1”, “@drink2”, “@drink3”.

In the test “test_can_add_multiple_drinks” we call the function “add_drink” to add three drinks and when this test is ran in the terminal with the help of “p @bar.drink” we print out the drinks now in the array.

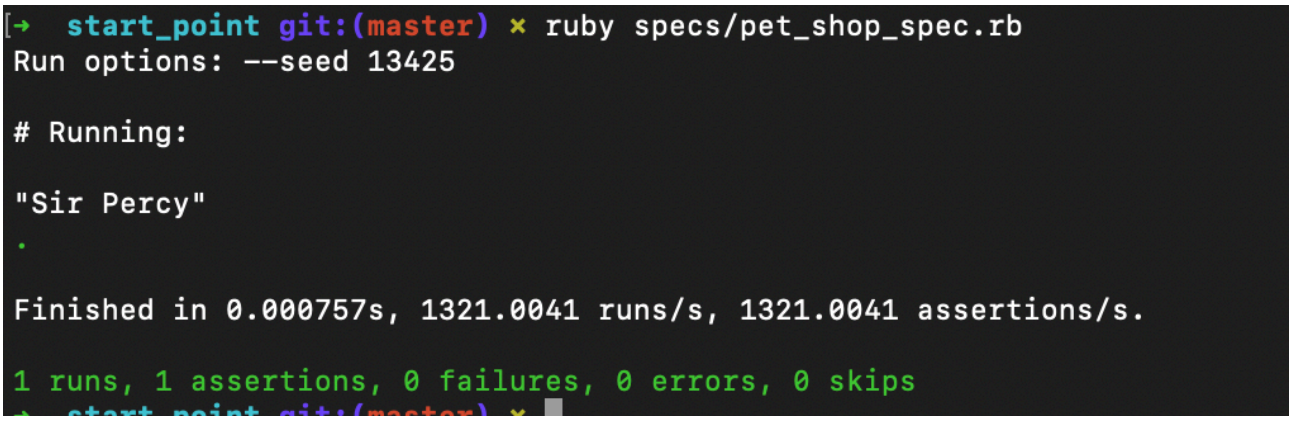
Week 3

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, name)
  for pet in pet_shop[:pets]
    if pet[:name] == name
      p 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
```



```
[→  start_point git:(master) × ruby specs/pet_shop_spec.rb
Run options: --seed 13425

# Running:

"Sir Percy"
.

Finished in 0.000757s, 1321.0041 runs/s, 1321.0041 assertions/s.

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

Image 1 shows the @pet_shop object which contains an array called pets, this array contains a list of hashes the first hash being Sir Percy.

Image 2 contains the function which will find the pet by its name, the function loops through each hash until the pet[:name] matches name passed in the argument. Once this is met the function will print the name with p pets[:name].

Image 3 shows the test calling the function and the argument being passed to the function -> “Sir Percy”

Image 4 shows 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
cars = ["Bmw", "Audi", "Mercedes", "Bentley", "Ferrari"]

def sort(cars)
  cars.sort { |a, b| a <=> b }
end

puts cars
puts ""
puts sort(cars)
```

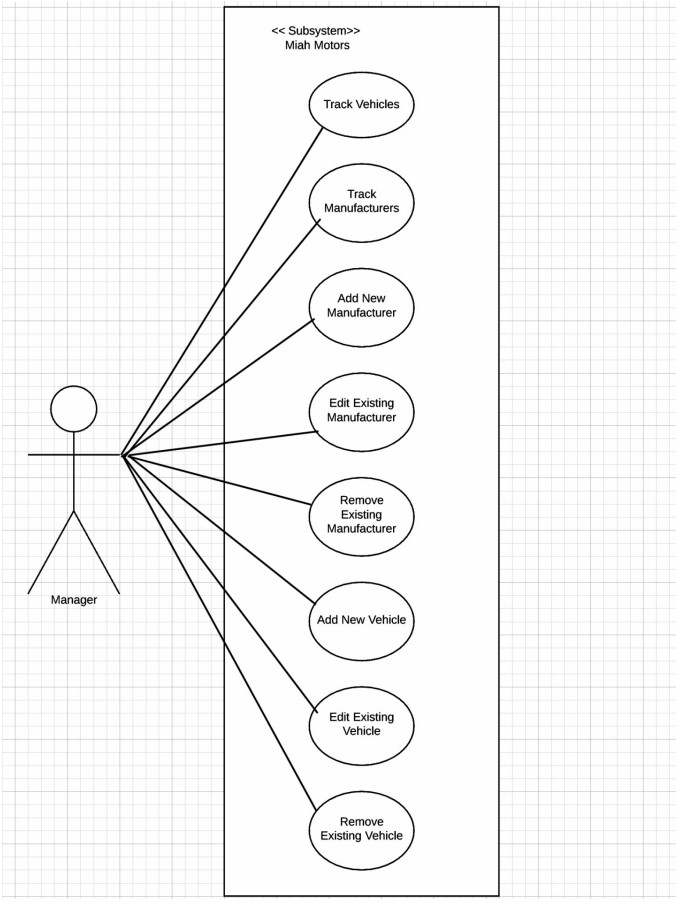
```
→  week_3 git:(master) ✗ ruby sort.rb
Bmw
Audi
Mercedes
Bentley
Ferrari

Audi
Bentley
Bmw
Ferrari
Mercedes
→  week_3 git:(master) ✗
```

Image 1 shows the sort function take an argument (an array of car manufacturers) and sort the argument into alphabetical order

Image 2 shows the result of the function - it first puts the array out inits original state then once again in alphabetical order.

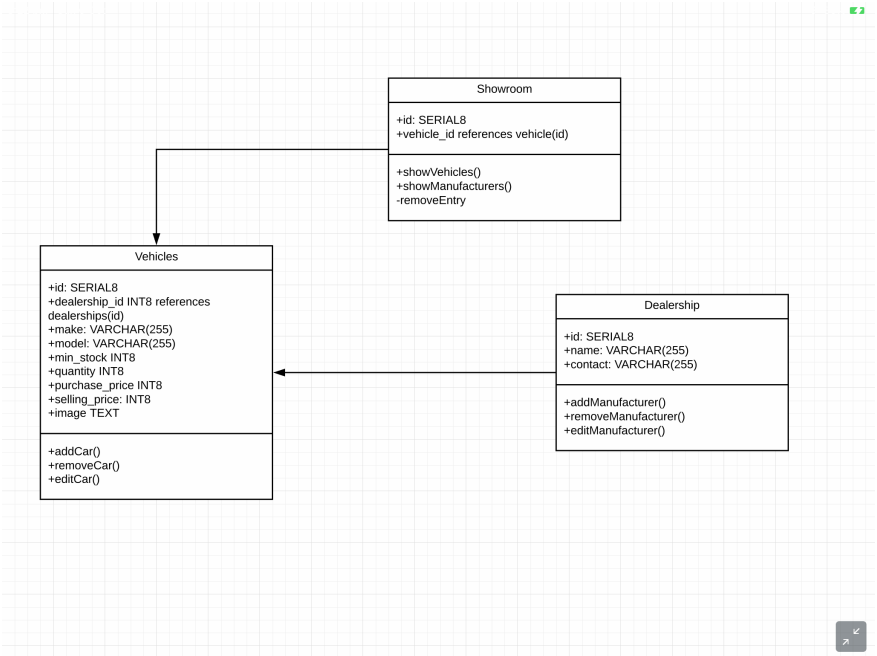
Week 4

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



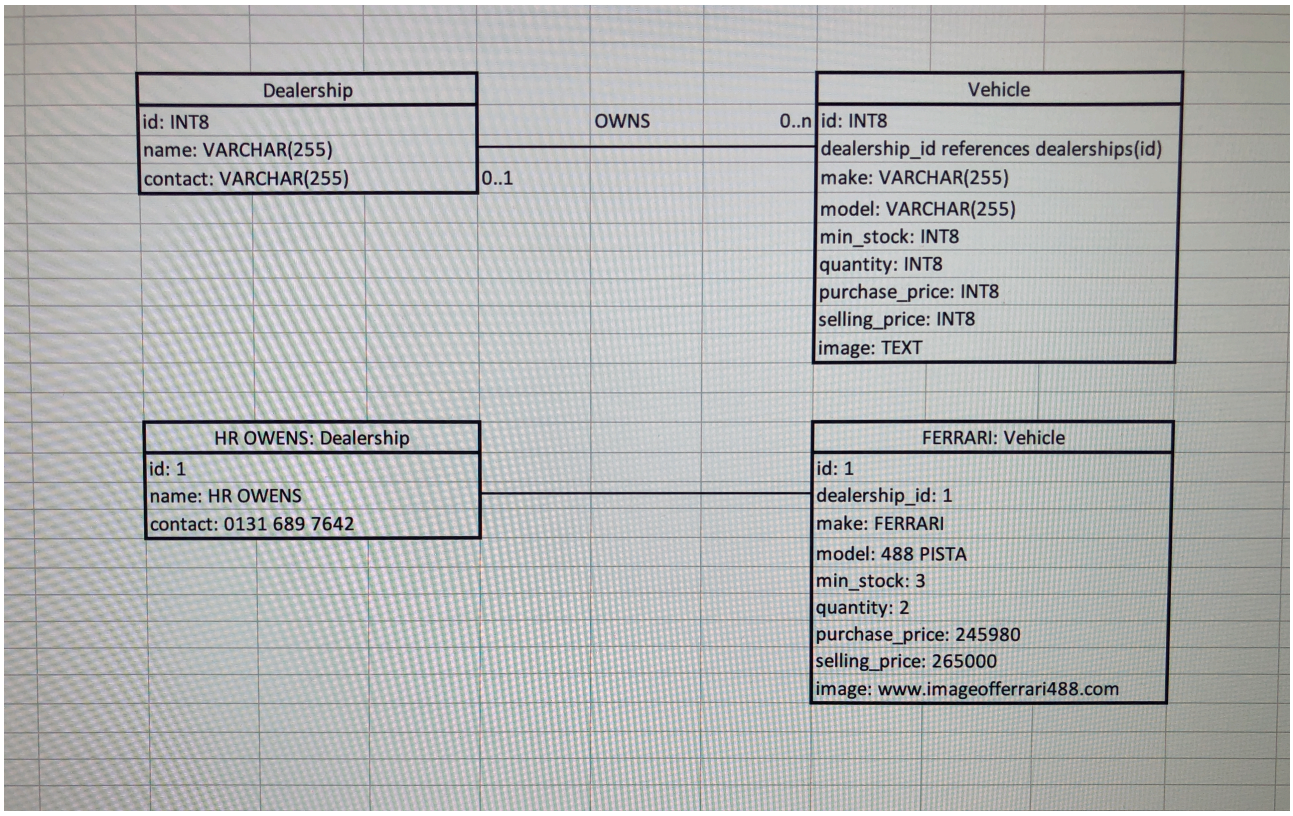
The image to the left shows the manager who is the user of the Miah Motors system who can perform various actions/functions such as “Edit Existing Manufacturer” and “Add New Vehicle”.

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



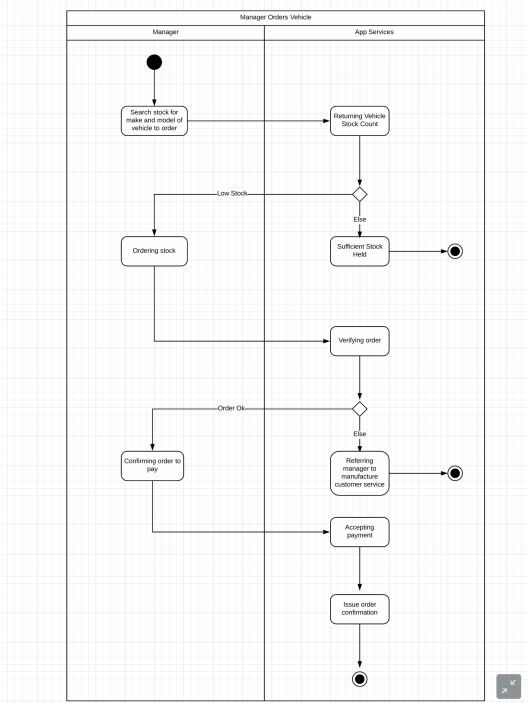
The image to the left shows a class diagram that contains a showroom, vehicle and dealership class. The diagram shows the attributes and methods for all classes. The dealership class has a name and contact attribute which are set as string. The vehicles class has an attribute of dealership_id which it gets from the dealership class. The dealership class has a direct relationship with the vehicles class.

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



The above image is an object diagram of the above class diagram. Ferrari is an object of the vehicle class and HR Owens of the dealership class. The vehicles object attains the dealership_id attribute from the dealership class.

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



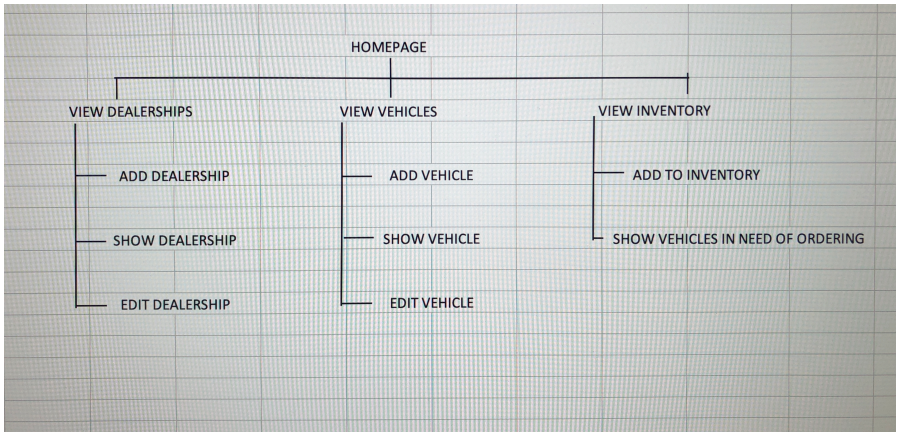
The activity diagram on the left shows the process of ordering new stock. At the start of the journey the manager will search for the make and model of a vehicle to order this will return the current stock count. If there app shows sufficient stock the manager can exit the process however if there is low stock the manager can order stock, this will lead to the app verifying the order. If the order cannot be verified the manager will be referred to the customer service of the manufacturer and will exit the process at that point however once the order is verified then the manager will confirm order to pay, this will lead the app to accept the payment followed by issuing a confirmation and exit the process.

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

Implementation Constraints Table - Miah Motors		
Constraint Category	Implementation Constraint	Solution
Hardware & Software	The app was designed to work on a Mac OSX operating system with Chrome as the default browser, however it is possible that this app may not work as expected when used with Firefox or any other browser or other operating systems. This is important as all users of the app must have a system running Mac OSX and have Chrome as a Browser Choice.	Test the app on other hardware/software configurations and configure the app in order to ensure it works correctly and as expected regardless of what the hardware/software configuration asked for.
Performance Requirements	The app currently has low performance requirements but as the tables continue to grow with more vehicles and dealership information the database site will also continue to grow. This is important because if the databases gets big enough to slow the app significantly down then the manager might not be able to manage Miah Motors adequately.	A solution to this problem would be to test the product with larger database to see how much It can slow the app down and configure it to prevent it.
Persistent Storage & Transactions	The storage related to the app will be local to the machine it is running on, this is a problem because multiple devices can't run the app with the same data unless its individually added to each and every device. Another issue is if the local storage becomes full then new vehicles cannot be added.	A possible solution for this would be to store the database remotely in the cloud so that is available to multiple devices and much storage space can be bought/added.
Usability	The user interface has been developed in a way that users who aren't tech savvy can still use the app. It was important to develop it this way as if it was too complex the manager and future users may struggle to use the app correctly.	Create a simple and intuitive UI
Budgets	No budget has been made available and as a result we cannot implement remote databases in the cloud which is a problem for future scalability	Design the application with the intention of it having one user at a time and no paid resources.
Time	The app came with a one week deadline which limits how much in terms of functionality can be added to the product while also maintaining a simple to use interface. This can be a problem as lack of proper planning can lead to the MVP not being fulfilled	A solution to this would be to work out the MVP and complete that within the timeframe and plan for any additional features after the fact.

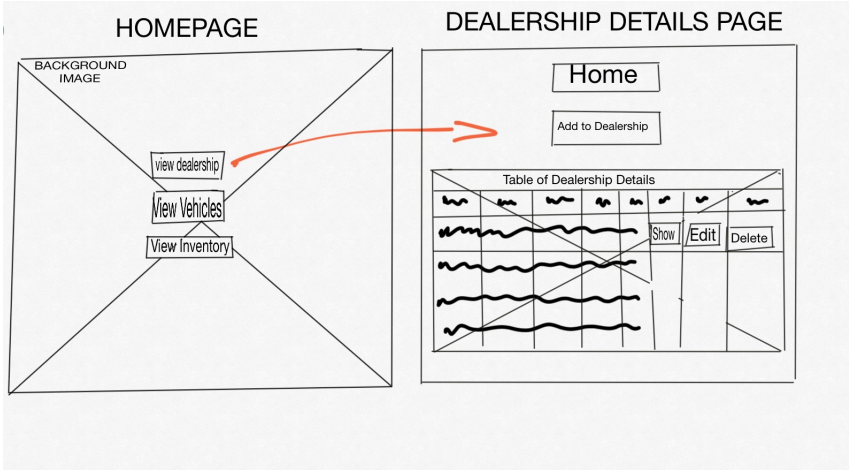
Above is an implementation and constraints plan for the Shop inventory app. It was created with the understanding that workers within the Shop would be using this app to update inventory levels so it had to be simple and an easy to use application.

Unit	Ref	Evidence
P	P.5	User Site Map



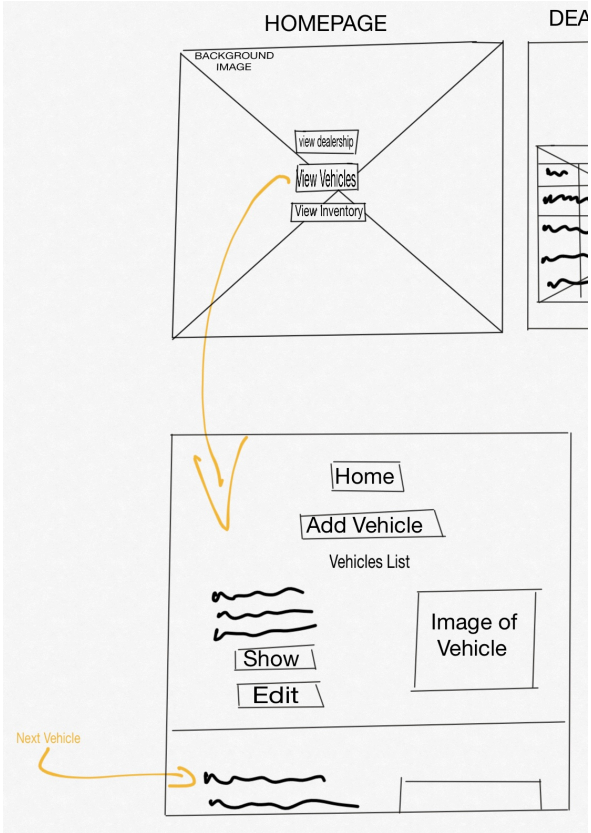
To the left is a user site map. From the homepage you can choose to access either the dealership, vehicles or inventory page. From the dealership page you can add show and edit a dealership.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams



The image to the left shows the homepage wireframe - Background image with three buttons in the centre to take you to the three pages of the app

When view dealership is selected it takes you to the Dealership details page shown as wireframe. Which displays the details in the form of a table, allowing the user to show, edit and delete dealerships from the table.



The image to the left shows the homepage wireframe - Background image with three buttons in the centre to take you to the three pages of the app

When view vehicles is selected it takes you to the vehicles details page shown as wireframe. Which displays the details in the form of a list, allowing the user to show, edit and add vehicles in the list

Week 5

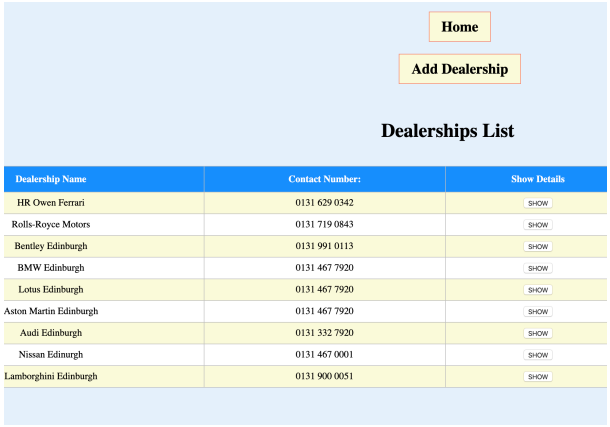
Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method

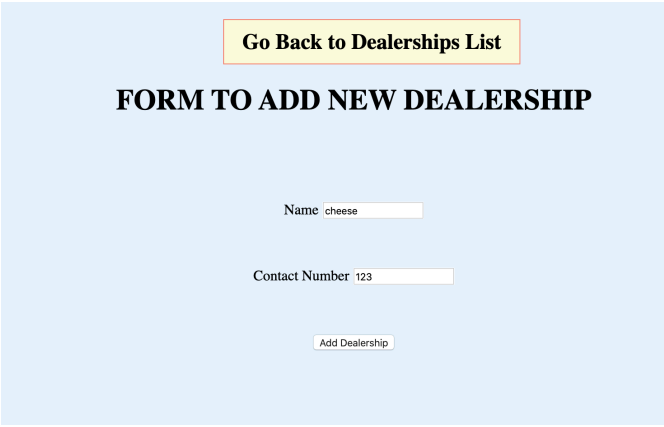
```
# get cars and use as a parameter for the sort method.
# use sort method to run through cars array before
# sorting into alphabetical order
# display list in alphabetical order

cars = ["Bmw", "Audi", "Mercedes", "Bentley", "Ferrari"]

def sort(cars)
  cars.sort { |a, b| a <=> b }
end

puts cars
puts ""
puts sort(cars)
```

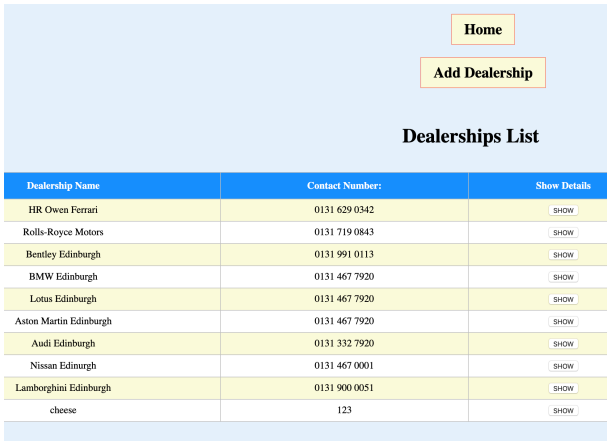
Description Here

The above shows the sort method taking the argument cars which is an array of makes before sorting it into alphabetical order.

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



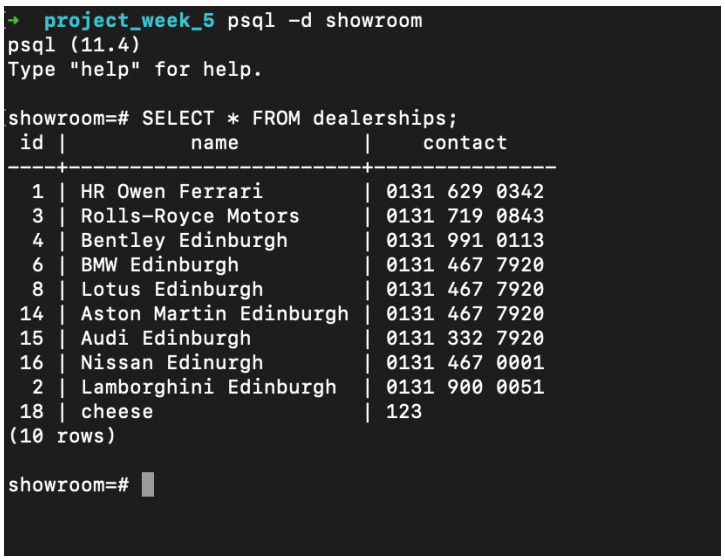




Description Here

The top left image shows the dealership list before the user inputs in new data. Top left shows the information the user is wanting to add to the dealership list. The bottom left shows the table once the information has been processed and added

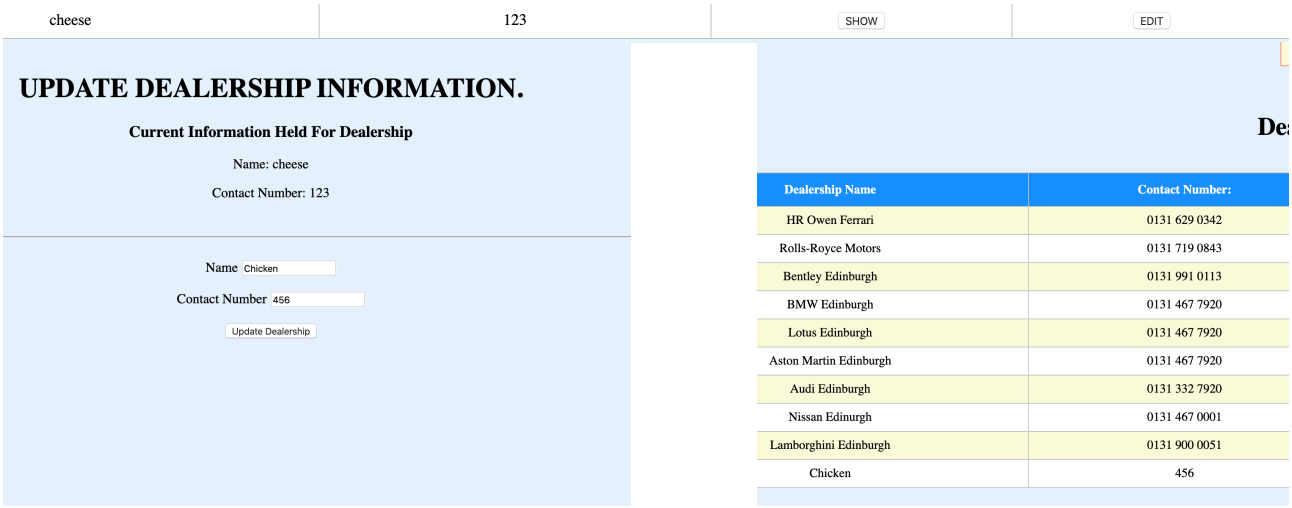
Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved



Description Here

The image on the left is conformation that the user input in the above images has been saved to the database.

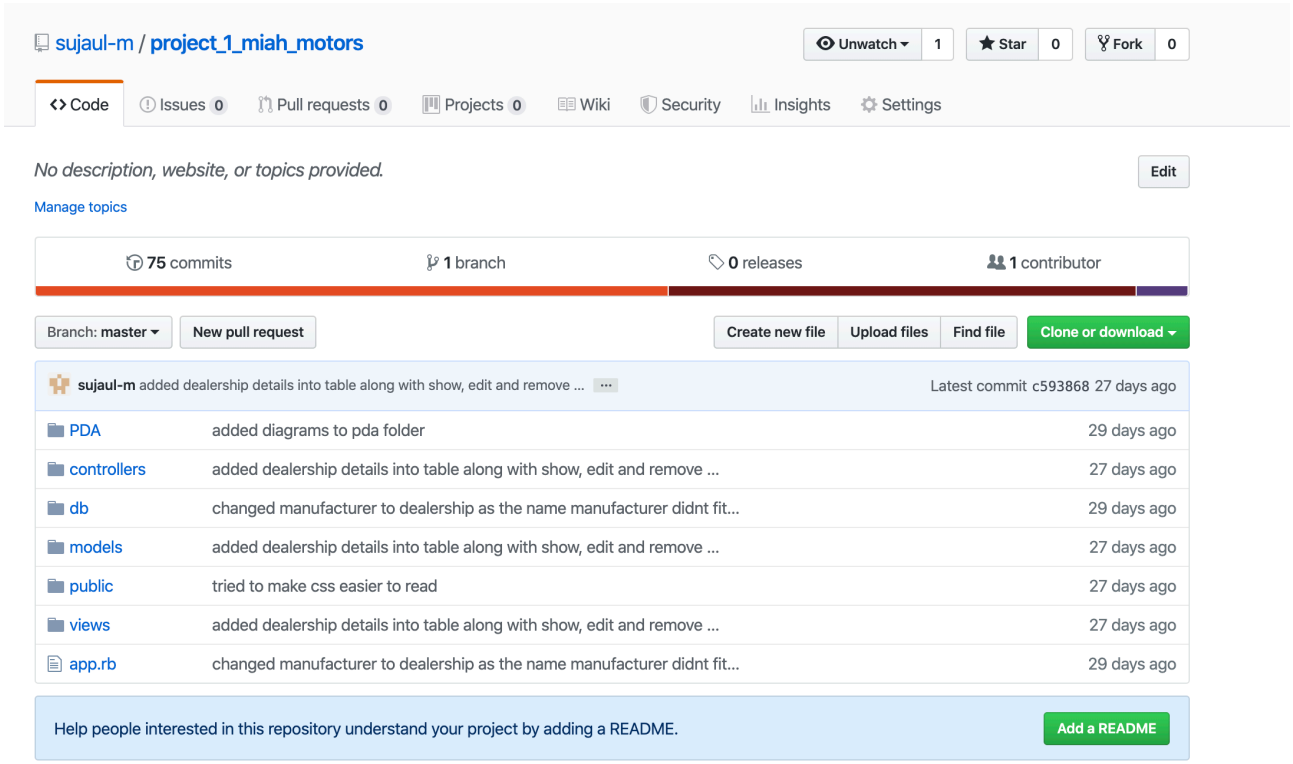
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program




```
showroom=# SELECT * FROM dealerships;
 id |          name           |    contact
----+-------------------------+---------------
  1 | HR Owen Ferrari         | 0131 629 0342
  3 | Rolls-Royce Motors      | 0131 719 0843
  4 | Bentley Edinburgh       | 0131 991 0113
  6 | BMW Edinburgh           | 0131 467 7920
  8 | Lotus Edinburgh         | 0131 467 7920
 14 | Aston Martin Edinburgh  | 0131 467 7920
 15 | Audi Edinburgh          | 0131 332 7920
 16 | Nissan Edinurgh         | 0131 467 0001
  2 | Lamborghini Edinburgh   | 0131 900 0051
 18 | cheese                  | 123
(10 rows)

showroom=# SELECT * FROM dealerships;
 id |          name           |    contact
----+-------------------------+---------------
  1 | HR Owen Ferrari         | 0131 629 0342
  3 | Rolls-Royce Motors      | 0131 719 0843
  4 | Bentley Edinburgh       | 0131 991 0113
  6 | BMW Edinburgh           | 0131 467 7920
  8 | Lotus Edinburgh         | 0131 467 7920
 14 | Aston Martin Edinburgh  | 0131 467 7920
 15 | Audi Edinburgh          | 0131 332 7920
 16 | Nissan Edinurgh         | 0131 467 0001
  2 | Lamborghini Edinburgh   | 0131 900 0051
 18 | Chicken                 | 456
(10 rows)
```

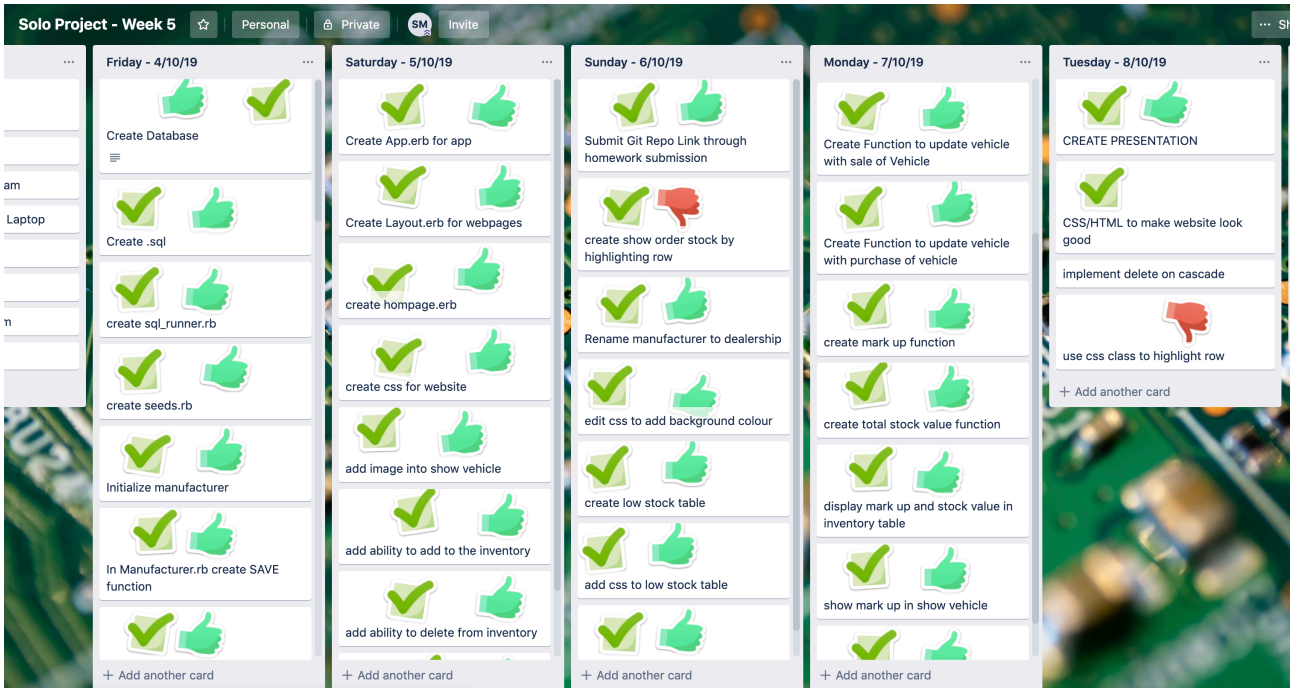
The very top image shows the entry for cheese, when the user clicks the edit button they are taken to the update dealership information page, here the name of the dealership is updated to Chicken and contact number to 456. When the user clicks update dealership the result is shown in the table in the picture on the right. If you look at the terminal picture you can see that the dealership with the id 18 has been updated from cheese to chicken.

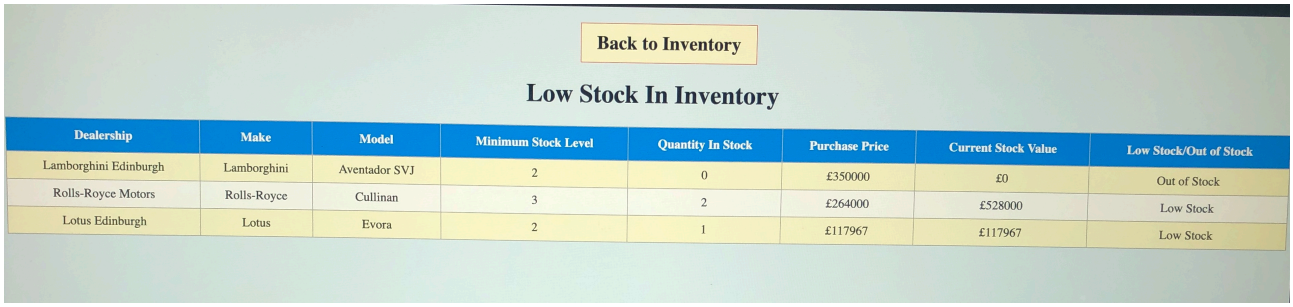
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.



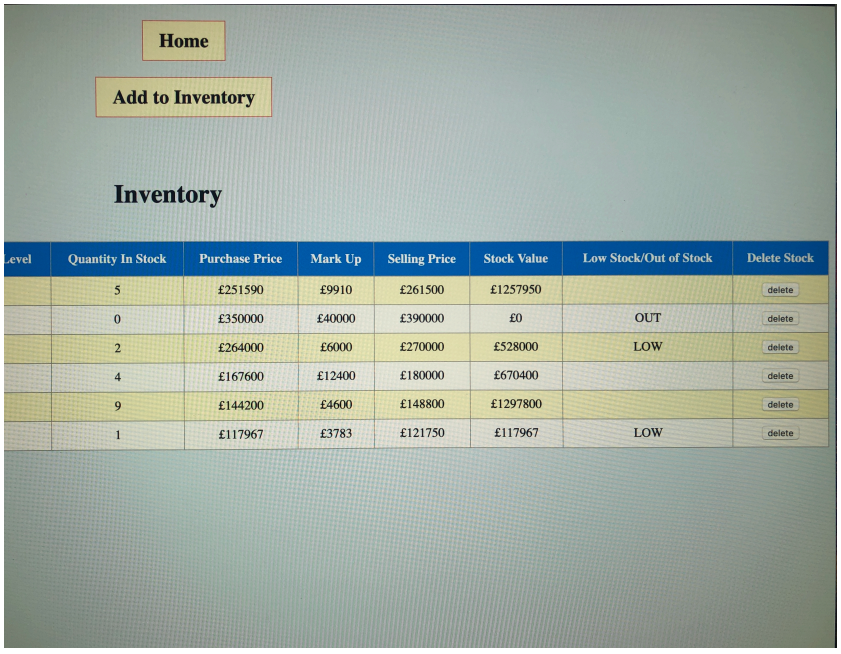
https://github.com/sujaul-m/project_1_miah_motors

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





Dealership	Make	Model	Minimum Stock Level	Quantity In Stock	Purchase Price	Current Stock Value	Low Stock/Out of Stock
Lamborghini Edinburgh	Lamborghini	Aventador SVJ	2	0	£350000	£0	Out of Stock
Rolls-Royce Motors	Rolls-Royce	Cullinan	3	2	£264000	£528000	Low Stock
Lotus Edinburgh	Lotus	Evora	2	1	£117967	£117967	Low Stock



Level	Quantity In Stock	Purchase Price	Mark Up	Selling Price	Stock Value	Low Stock/Out of Stock	Delete Stock
	5	£251590	£9910	£261500	£1257950		delete
	0	£350000	£40000	£390000	£0	OUT	delete
	2	£264000	£6000	£270000	£528000	LOW	delete
	4	£167600	£12400	£180000	£670400		delete
	9	£144200	£4600	£148800	£1297800		delete
	1	£117967	£3783	£121750	£117967	LOW	delete

Description Here

At the very top is the troll board used during planning and working on the project. The middle and bottom picture show change as originally the low/out of stock was shown as a separate table but it was later changed to show in the inventory table saving the user from clicking a link to a second table.

Week 7

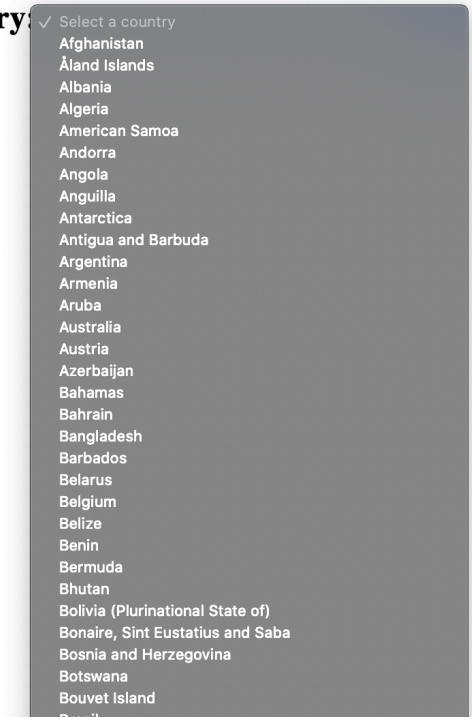
Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running

```
getCountries: function(){
    fetch("https://restcountries.eu/rest/v2/all")
    .then(res => res.json())
    .then(countries => this.countries = countries)
}
```

Countries App

Global population: 7349137231

Select a Country:



Description Here

The above image shows the rest countries api being accessed and data being fetched from it to populate the drop down selection for select a country. Fetch pulls the information from the api and .then (res => res.json) converts the result into json format.

Week 8

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

Animal Info App Aim: To have an animal information tool that can be shared with members of the public to help increase responsible pet ownership and avoid unintentional cruelty towards animals. This information tool should be accessed via the web but also via an app. It will be advertised through general public engagement activities but also through the Scottish SPCAs enforcement activities when it comes to working with adults to improve the welfare of the animals they are responsible for and ultimately keep that person together with their animal and preserve that human-animal bond. Some adults may be illiterate so use of imaging/icons would be a necessity. This app should focus on common pet animals initially (dog, cat, rabbit, hamster) but long term it should have the ability to evolve to include more species including wildlife. Ideally there should be the ability for the user to create their own animal profile relating to their pet (keeping in mind they may have more than one pet) using pre-set templates (where you can add age, sex, weight and upload a picture of your pet). There should also be a calendar feature attached to the animal profiles so the user can be reminded of when to feed their animal, routine vet checks, exercise, cleaning etc. Long term a child version could be created where the functionality would be the same but the user interface (icons, language etc) would be more child friendly.

Platform: Web and app (android, IOS)

User interface: easy to use, uses menus and icons to access information. Graphical interface initially but long term option for voice control interface?

MVP:

- Landing page: Option to choose from 3 different animal types (Domestic, Wildlife, Farm), initially focusing on domestic animals. Once an animal type is selected, it would refine further into new groups
    - 4-5 animal species with high-def images or pictograms (depending on client's needs), most likely dogs, cats, hamsters, rabbits - if time permits, add details of all animals on the back end.
    - Each animal type would lead to their own page in the app, that includes further details about how to care for them - pictures included if needed.
    - Each section for their care should have its details pulled from an object structured for the different needs of the animal - including text, a list of links to images, and anything else the client might provide - all supported by the back-end
- Add a contact page, where user can see/contact Scottish SPCA directly if needed.

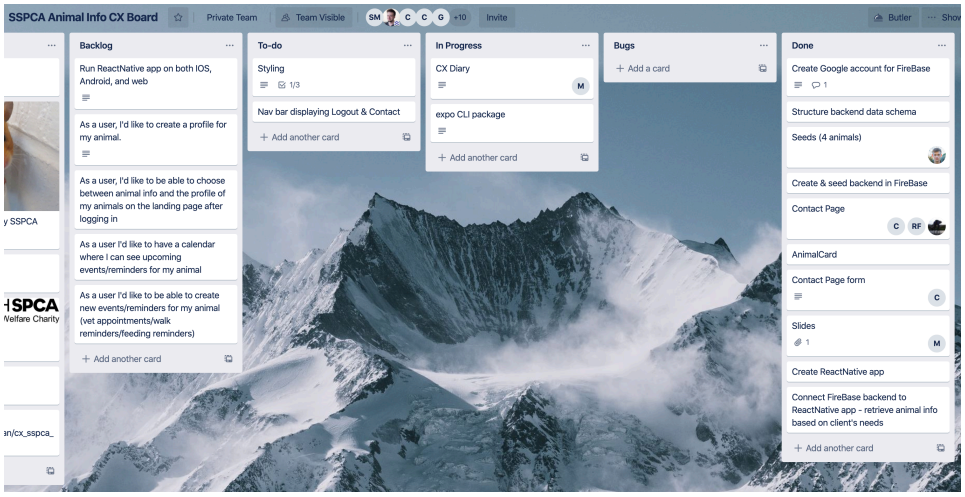
Extensions:

- Add authentication to the app, so users can have preferences set up and saved
- Add the option for the user to create a profile for their animals, including name, age, sex, and the option to take a picture and store it
- Create a calendar option with notifications to remind owner when to feed/health check/take animal to vet/exercise.

Description

This is the project brief for my 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 here

This is the planning for the group project set out on a TRELLO board

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

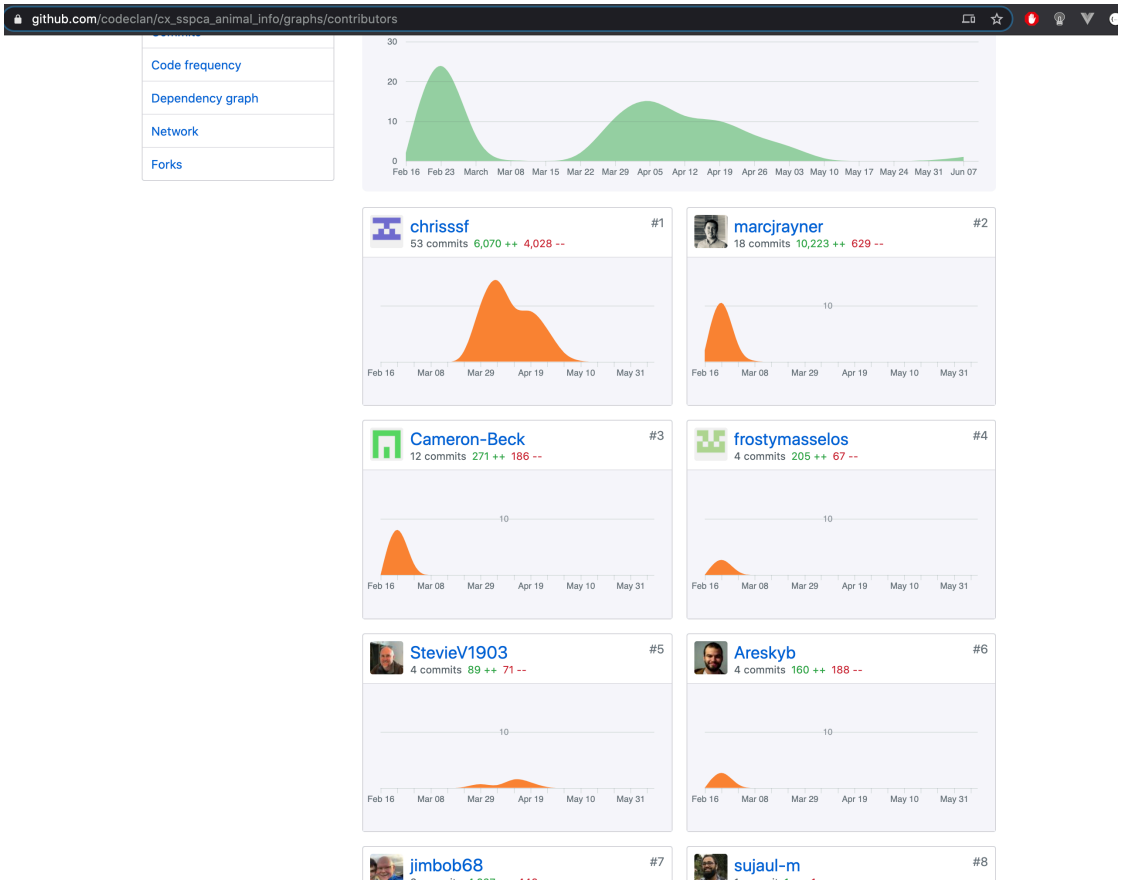
Acceptance Criteria	Test Plan	Pass / Fail
The user is able to favourite a recipe from the selection provided	Upon clicking the heart to favourite a recipe the action should be carried out and the recipe should be added to favourites without the current page requiring to be refreshed.	Pass
The user is able to set a recipe from the selection provided as currently in progress by clicking on the cooking pot on the recipe card	Upon clicking the cooking pot to set the recipe as in progress the action should be carried out and the recipe should be added to in progress without the current page requiring to be refreshed.	Pass
The user is able to click on a recipe card component which will expand into a step by step guide on how to make the baked good.	Upon clicking the card component the page should refresh into the step by step guide of the card clicked for example if the card for Mint Brownies is clicked it should refresh the page to show a step by step guide on how to make mint brownies.	Pass
The user is able to go forward and backwards when viewing the recipe guide	Once viewing the recipe guide the user should be able to go back and forward in the recipe by clicking on the buttons provided.	Pass
The user is able to view all recipes set as favourites at once.	Upon clicking the favourites button the user will be displayed all recipes set as favourites.	Pass

Description here

The above is an acceptance and criteria plan from the javascript educational application project which I developed

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.



Above is a screenshot of the contributor page of the SSPCA Animal Info Group Project. The Group Project was continued on from the last Cohorts work and was carried out during Covid-19 lockdown and as a result we had to make use of Zoom Video Call and Visual Studio Code's Live Share Collaboration feature in order to work on the project as a group while socially distancing. During the project the team collaborated together as a whole as we all came onto a zoom call and with the use of Live share we all took part in coding by communicating different ideas and having a go at trying snippets of codes to see how they affected the end product. By programming as a group we were able to work faster and effectively due to being able to plan, code, test, reflect and agree at every step as a team, thus we managed to accomplish a great deal during our time working on the project. I collaborated with my team to fix authentication issues we faced when trying to log into the app, adding new features such as a calendar for users to add tasks to, we managed to get images to load from the database and I also collaborated on improving the styling and general look of the app across all platforms. As a team we struggled with getting the drop down information for each pet to display fully across all platforms (web, iOS and Android) but through research and testing we managed to overcome this obstacle. Due to using VSCode's Live share feature we essentially did mob programming as everyday we used the same file from the same individual and as a result all commits were pushed by that one individual. Hence why viewing the above screenshot shows one individual with a lot more commits than anyone else in the group during the month of March, so at the end of the project as per the marking guidance I had to update the README file with my name in order to get myself on the contributor page to provide evidence for this section.

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_add_or_remove_cash__add
  add_or_remove_cash(@pet_shop,10)
  cash = total_cash(@pet_shop)
  assert_equal(1000, cash)
end
```

```
→  start_point git:(master) × ruby specs/pet_shop_spec.rb
Run options: --seed 4265

# Running:

F

Finished in 0.000848s, 1179.2453 runs/s, 1179.2453 assertions/s.

  1) Failure:
TestPetShop#test_add_or_remove_cash__add [specs/pet_shop_spec.rb:94]:
Expected: 1000
  Actual: 1010

1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
→  start_point git:(master) ×
```

```ruby
def test_add_or_remove_cash__add
  add_or_remove_cash(@pet_shop,10)
  cash = total_cash(@pet_shop)
  assert_equal(1010, cash)
end
```

```
→  start_point git:(master) × ruby specs/pet_shop_spec.rb
Run options: --seed 13687

# Running:

1010
.

Finished in 0.000867s, 1153.4025 runs/s, 1153.4025 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
→  start_point git:(master) ×
```

Description here

The first image show the test for the method add_or_remove_cash which adds or removes cash from the pet shop. We use the method to add 10 to the total cash held by the pet shop, which was previously 1000. The second image shows the test failing as the amount held by the pet shop is 1010 after adding 10 but the assert equal was set to 1000 which is incorrect. The third image shows the assert equal with the correct amount of 1010 and the fourth image shows the test now passing.

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

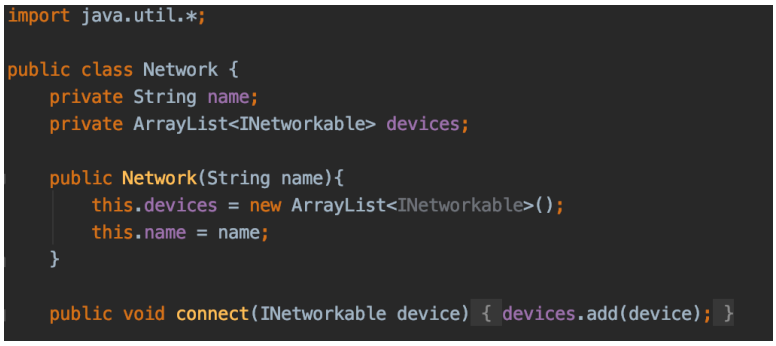
```java
public class Bear {

    private String name;
    private int age;

    public Bear(String name, int age){
        this.name = name;
        this.age = age;
    }

    public String getName() { return this.name; }

    public void setName(String newName) { this.name = newName; }

    public int getAge(){ return age; }

    public void setAge(int newAge) { this.age = newAge; }
}
```

Description here

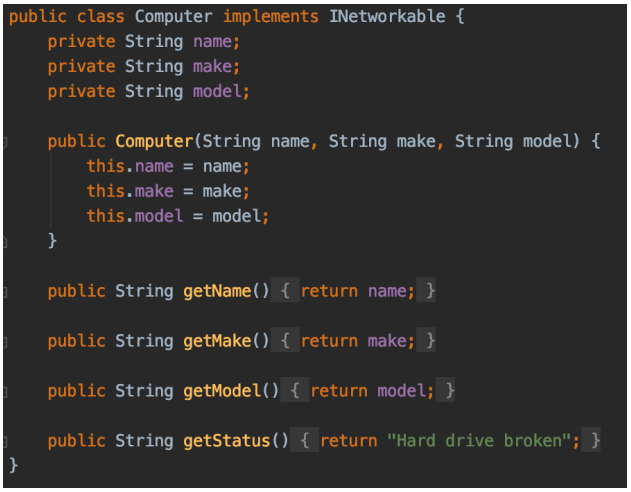
This shows the use of encapsulation within the class Bear. The variable for name and age is private. This means it is only accessible within the class Bear and to be able to access or change it, the getName/getAge or setName/setAge must be used. The getName returns the current string held for name whereas setName replaces the current string with the new string.

Week 12

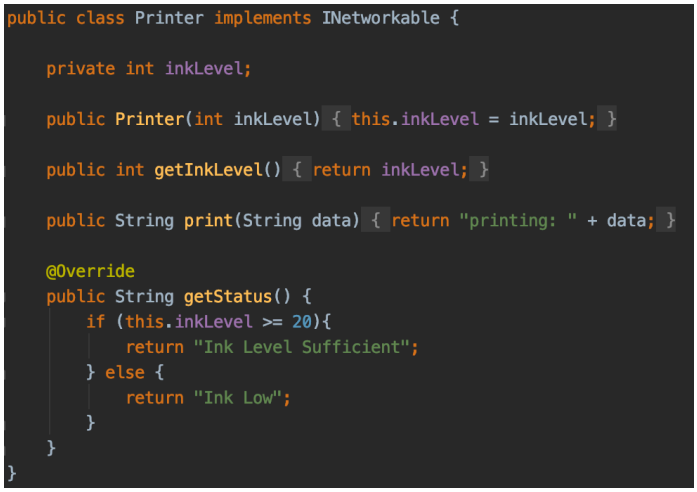
Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.



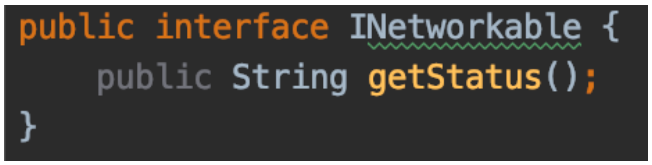
```java
import java.util.*;

public class Network {
    private String name;
    private ArrayList<INetworkable> devices;

    public Network(String name){
        this.devices = new ArrayList<INetworkable>();
        this.name = name;
    }

    public void connect(INetworkable device) { devices.add(device); }
```



```java
public class Computer implements INetworkable {
    private String name;
    private String make;
    private String model;

    public Computer(String name, String make, String model) {
        this.name = name;
        this.make = make;
        this.model = model;
    }

    public String getName() { return name; }

    public String getMake() { return make; }

    public String getModel() { return model; }

    public String getStatus() { return "Hard drive broken"; }
}
```



```java
public class Printer implements INetworkable {

    private int inkLevel;

    public Printer(int inkLevel) { this.inkLevel = inkLevel; }

    public int getInkLevel() { return inkLevel; }

    public String print(String data) { return "printing: " + data; }

    @Override
    public String getStatus() {
        if (this.inkLevel >= 20){
            return "Ink Level Sufficient";
        } else {
            return "Ink Low";
        }
    }
}
```



```java
public interface INetworkable {
    public String getStatus();
}
```

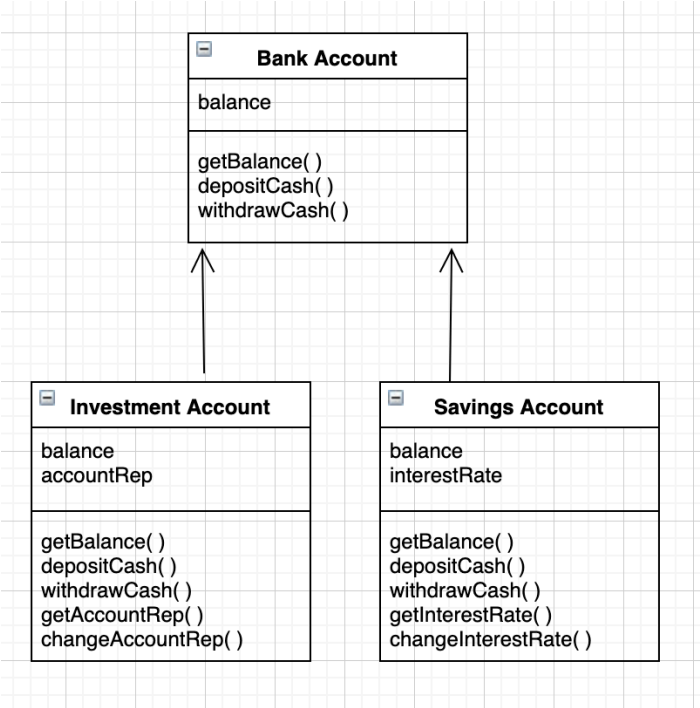
Description here

Image 1 at the top shows the Network class which contains an ArrayList of devices which holds items that implement the INetworkable interface. It also shows a “connect” method which is used to add items which implement the INetworkable interface into the device ArrayList.

Image 2 and image 3 contain two classes which implement INetworkable, the Computer class and the Printer class. These are two different types of objects but since they both implement the INetworkable interface this means they can both be added to the same ArrayList.

Image 4 shows the INetworkable interface which contains the method getStatus() which will return a string.

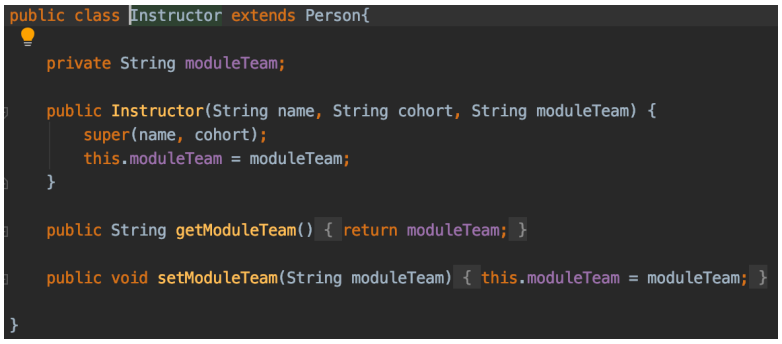
Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram



The diagram to the left shows the inheritance of properties and methods from the Bank Account into the classes of InvestmentAccount and SavingsAccount. The InvestmentAccount and SavingsAccount will inherit the balance object swell as the method getBalance,despositCash and withdrawCash. The classes InvestmentAccount and SavingsAccount will have there own methods and characteristics . For example savingsAccount has a property of interestRate and has additional methods of getInterestRate and changeInterestRate.

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.

```
public class Person {

    private String name;
    private String cohort;

    public Person(String name, String cohort) {
        this.name = name;
        this.cohort = cohort;
    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getCohort() { return cohort; }

    public void setCohort(String cohort) { this.cohort = cohort; }

    public String talk(String favLanguage) { return String.format("I love %s!", favLanguage); }
}
```



```
public class Instructor extends Person{

    private String moduleTeam;

    public Instructor(String name, String cohort, String moduleTeam) {
        super(name, cohort);
        this.moduleTeam = moduleTeam;
    }

    public String getModuleTeam() { return moduleTeam; }

    public void setModuleTeam(String moduleTeam) { this.moduleTeam = moduleTeam; }

}
```

```
public class InstructorTest {

    Instructor instructor;

    @Before
    public void before() { instructor = new Instructor( name: "Colin",  cohort: "G6",  moduleTeam: "Java"); }


    @Test
    public void hasName() { assertEquals( expected: "Colin", instructor.getName()); }

    @Test
    public void hasCohort() { assertEquals( expected: "G6", instructor.getCohort()); }
```

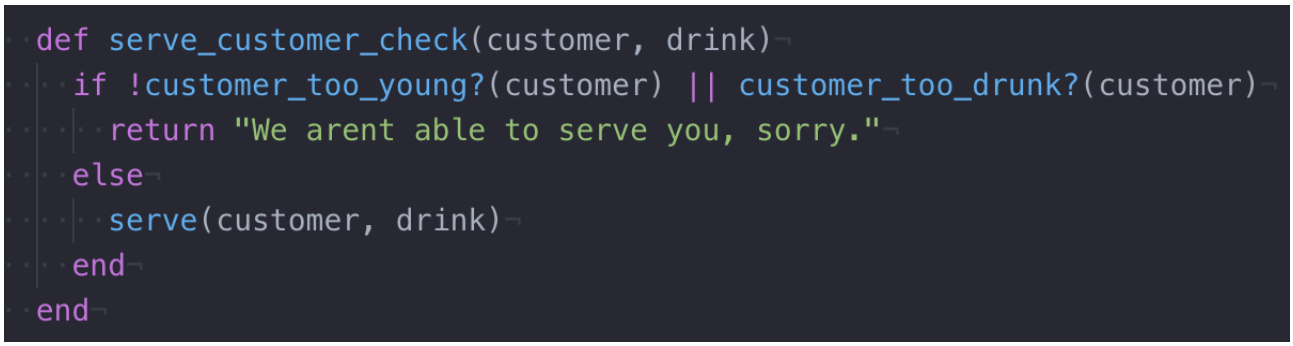
The first screenshot shows the parent class which is Person and the properties a person should have. Each person should have a string of name and cohort.

The second screenshot is the Instructor class which inherits from Person, but also has its own property of moduleTeam. It also inherits the property of name and cohort using the “Super” keyword in the constructor.

The third screenshot shows an object of instructor being created - instructor = new Instructor(“Colin”, “G6”, ”Java”). The object instructor then uses the inherited method of getName() to find the name of the instructor.

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.



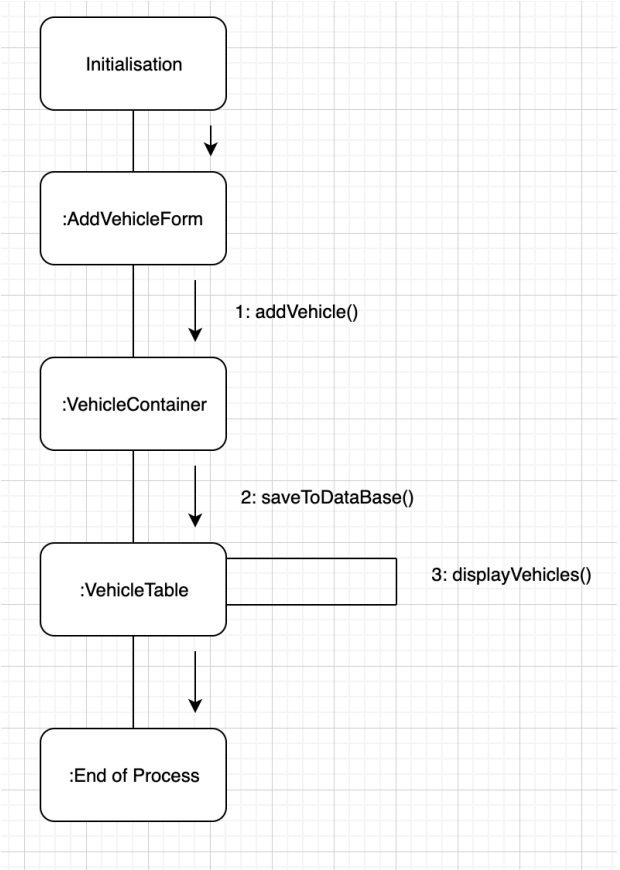
```ruby
def serve_customer_check(customer, drink)
  if !customer_too_young?(customer) || customer_too_drunk?(customer)
    return "We arent able to serve you, sorry."
  else
    serve(customer, drink)
  end
end
```

```javascript
fetch("https://restcountries.eu/rest/v2/all")
.then(res => res.json())
.then(countries => this.countries = countries)
```

Image 1 is an example of when I used a logical OR operator within an application. The algorithm checks the customers age to see if they are too young to drink and also checks to see if they are too drunk to be served and in the event either of them fails then the customer is not served another drink and the string “We aren’t able to serve you, sorry” is returned. Otherwise it will serve the customer a drink.

Image 2 is an example of fetching date from an API into Vue.js. This was used in the countries lab. The algorithm starts with fetching data. The second line then takes the result of the objects in the api known as “res” and then uses the .json method which exists within the api and returns the objects in a form that javascript understands. The third line then takes the result of the .json method and extracts the countries information from the API before populating the countries array with this.countires, this is then accessible by vue.js with countries.

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



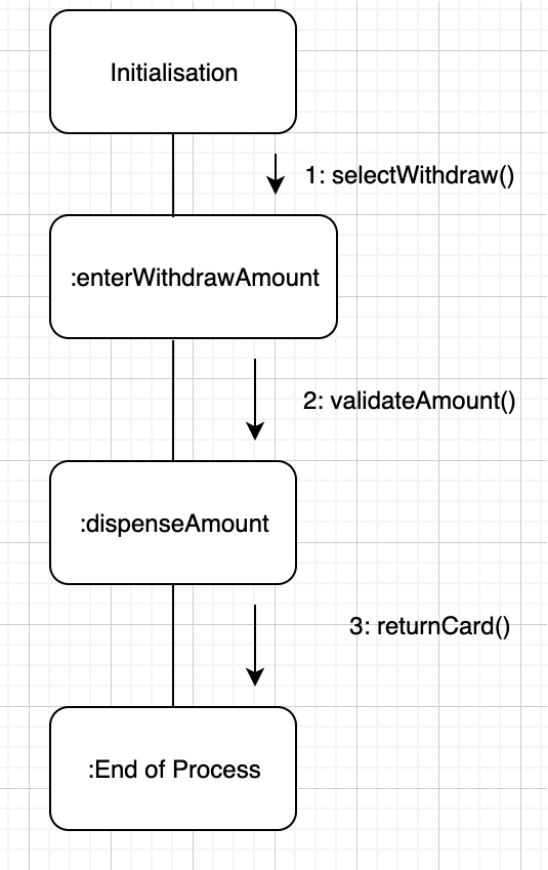


Image 1 on the left is a collaboration diagram outlining the messages passed between objects and the order in which they occur for the employee when adding vehicles to the Vehicle inventory app.

Image 2 on the right is a collaboration diagram to show the interaction in an ATM machine. The user selects to withdraw money. After the user enters an amount to be withdrawn the amount is validated against the money in the account and if its okay the amount is dispensed and the card is returned to the user ending the process.

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

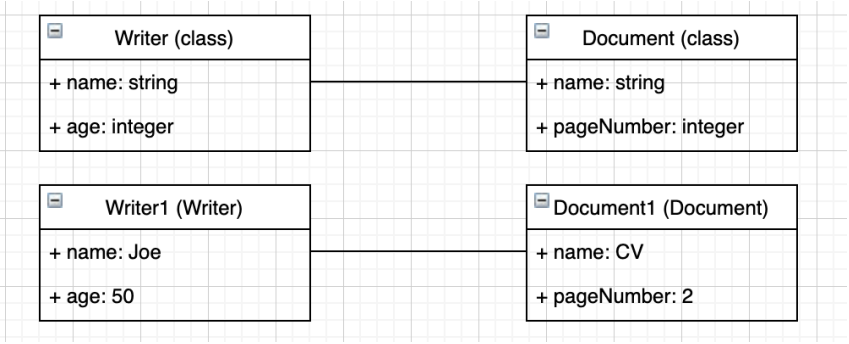


Image 1 on the left shows two classes Writer and Document. The Writer can create a Document. The writer has a string for “name” and an integer for “age” whereas the document has a string for “name” and an integer for “pageNumber”. An instance of the Writer class has the name of “Joe” and an age of “50” whereas the instance of the Document class has the name of “CV” and a pageNumber of “2”

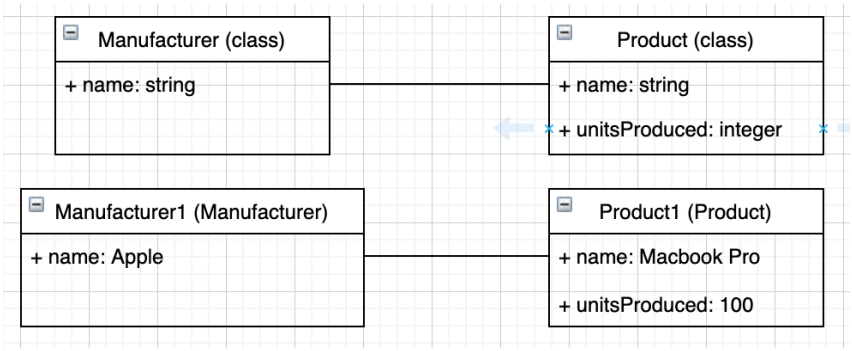


Image 2 on the left shows two classes Manufacturer and Product. The Manufacturer can create a Product. The Manufacturer has a string for “name” whereas the Product has a string for “name” and an integer for “unitsProduced”. An instance of the Manufacturer class has the name of “Apple” whereas the instance of the Product class has the name of “Macbook Pro” and a unitsProduced of “100”

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

Bug Tracking Report

	Bug/Error	Solution	Date
1	While running the RestaurantBookingSystemApplication we had an issue where the terminal was returning the error FATAL: database "restaurantbookinservice" does not exist.	Realised the database had a typo and should have been “restaurantbookingservice” missing the g at the end of booking. This resolved the issue	6/3/20
2	In the browser we got the error “Failed to Compile” Module not found: Can't resolve './component/ BookingSystemComponents/EditBookingForm'	This was resolved by correcting the path set out in BookingSystemBox for EditBookingForm	6/3/20
3	While running the RestaurantBookingSystem Application the database would not drop before running which meant the same instances of the data loader were being related multiple times	This issue was resolved by updating the properties file to spring.jpa.hibernate.ddl-auto = create-drop Instead of spring.jpa.hibernate.ddl-auto = update	6/3/20
4	During the creation of a new booking a user would be able to book with 0 adults and 0 kids present which should not be allowed as a minimum of 1 adult is required	This issue was resolved by adding the (min=“1”) to the input box for adult	7/3/20
5	The content in the browser was partially hidden behind the vertical nav bar when it was changed from a top bar to a vertical bar	This issue was resolved by inserting margin-left: 140px; in the css for any item partially covered by the nav bar. This ensured that the content was moved to the left by the width of the sidebar plus a little extra.	8/3/20

Above is a Bug Tracking report