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

```
{
  name: "Sir Percy",
  pet_type: :cat,
  breed: "British Shorthair",
  price: 500
},
```

Above is an example of a hash. It has four key/value pairs for; a pet name, their type, breed and price.

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

Above is a function that searches for the pet by name. It searches through the :name key of all the pets looking for one that has a matching value with the pet name entered by the user.

```
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 7029

# Running:

.

Finished in 0.001467s, 681.6632 runs/s, 681.6632 assertions/s.

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

Above is a test for the find_pet_by_name function and the result of running the test, which successfully passes.

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

```
@song1 = Song.new("I Want to Break Free","Queen")
@song2 = Song.new("Sweet Child O' Mine", "Guns N'Roses")
@song3 = Song.new("Eye Of The Tiger", "Survivor")
@song4 = Song.new("Smells Like Teen Spirit", "Nirvana")
@song5 = Song.new("I'm Gonna Be 500 Miles", "The Proclaimers")

@songs = [@song1, @song2, @song3]
```

Above is an example of an array of songs created for a Karaoke Bar task. In this initial set up three songs are put into the array.

```
def add_song(song, room)
  return room.playlist.push(song)
end
```

Above is an example of a function to add a song to the array. It adds a song specified by the user to the end of the array using .push

```
def test_add_song_to_room
  @bar1.add_song(@song5, @room2)
  assert_equal(4, @room2.playlist.count)
end
```

```
→  specs git:(master) ✗ ruby karaoke_bar_spec.rb
Run options: --seed 59982

# Running:

.
```

Above is a test for the add_song function. It counts the number of songs in the array checking that there are now four songs 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
def details()
  sql = "SELECT * FROM tickets WHERE id =$1;"
  values = [@id]
  return SqlRunner.run(sql, values).map{|ticket_details| Ticket.new(ticket_details)}
end
```

Above is a function which searches through the tickets table in the sql database. It uses the unique ticket id to find the ticket information being searched for.

```
[2] pry(main)> ticket1.details
=> [#<Ticket:0x00007fd480252b58 @customer_id=21, @film_id=22, @id=33>]
```

Above shows the result of calling the ticket details function which returns all of the ticket information.

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
def pets_by_breed(shop, breed_type)
  result = []
  for pet in shop[:pets]
    result.push(pet) if(pet[:breed] == breed_type)
  end
  return result
end
```

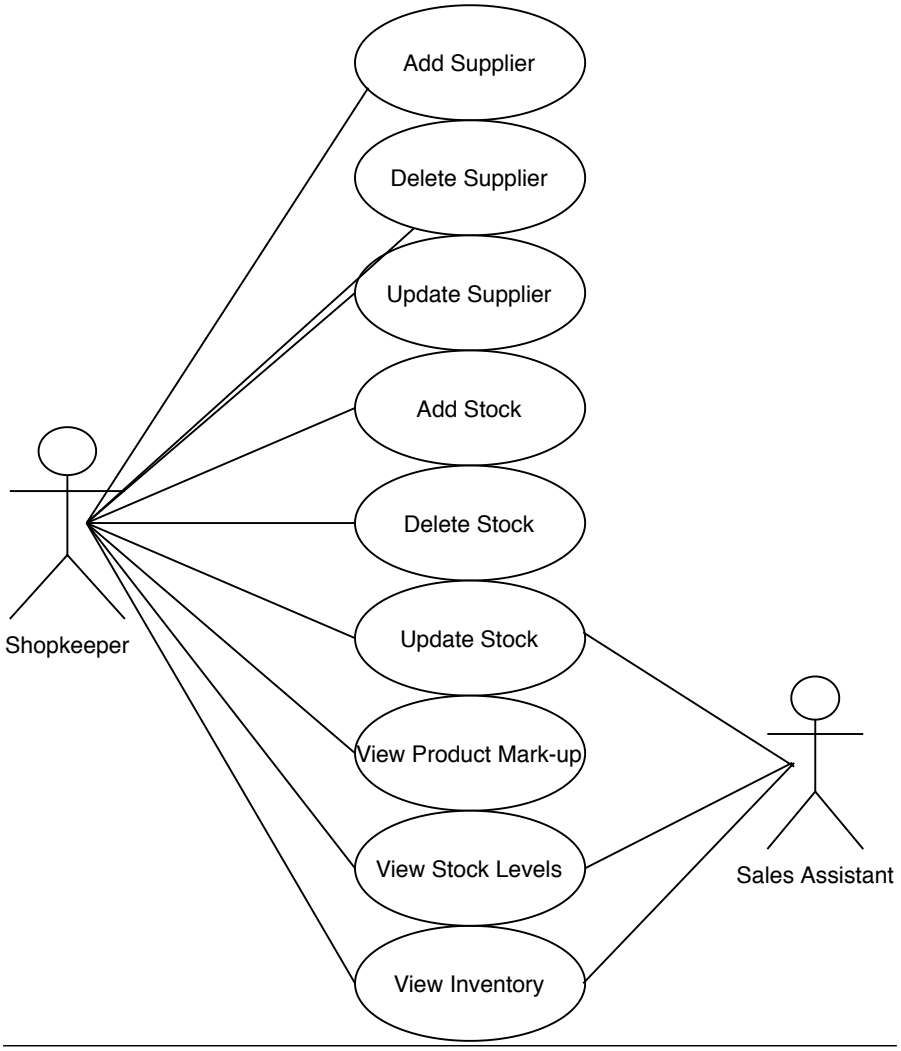
Above is a function that sorts through an array of pets and searches for one specified breed. It then returns a new array of all pets that match that breed.

```
From: /Users/mary/codeclan_work/week_01/weekend_homework/start_point/specs/pet_s
hop_spec.rb @ line 123 TestPetShop#test_all_pets_by_breed__found:

    120: def test_all_pets_by_breed__found
    121:   pets = pets_by_breed(@pet_shop, "British Shorthair")
    122:   binding.pry
 => 123:   assert_equal(2, pets.count)
    124: end

[1] pry(#<TestPetShop>)> pets.count
=> 2
[2] pry(#<TestPetShop>)> pets
=> [{:name=>"Sir Percy",
  :pet_type=>:cat,
  :breed=>"British Shorthair",
  :price=>500},
 {:name=>"King Bagdemagus",
  :pet_type=>:cat,
  :breed=>"British Shorthair",
  :price=>500}]
[3] pry(#<TestPetShop>)>
```

Using prybyebug to pause the test you can see above that the function has successfully returned an array with the two pets that have the breed “British Shorthair”.

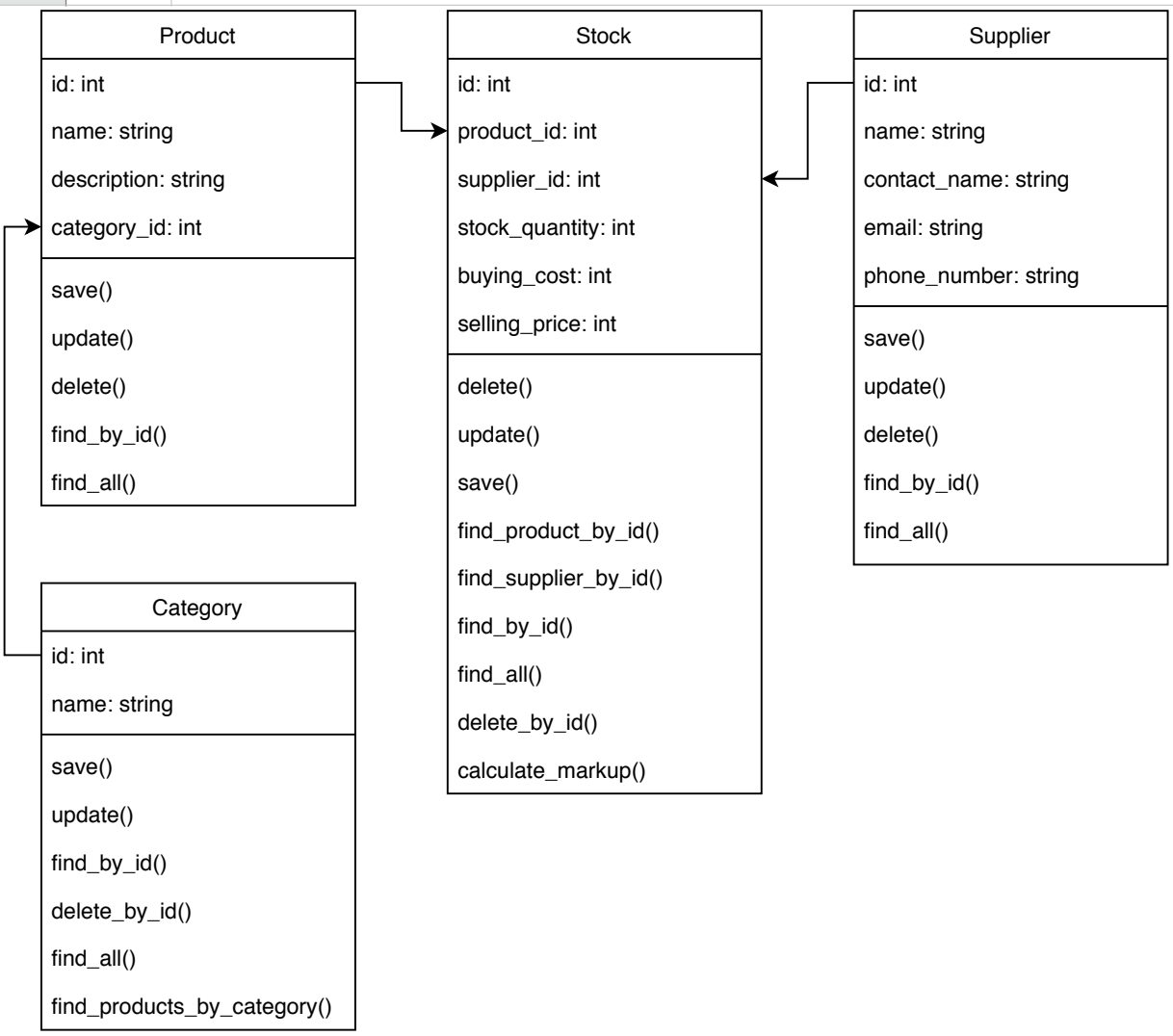
Week 4

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



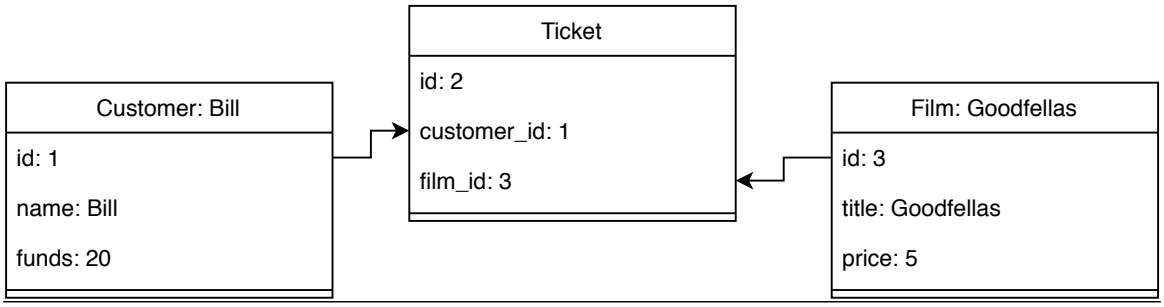
This shows a use case diagram for an application built to manage stock for a small shop. It shows all the functions the Shopkeeper would need to be able to access.
There would also need to be a lower level of access for a sales assistant who would just need to be able to view the inventory and stock levels to answer queries and to update stock when a sale was made.

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



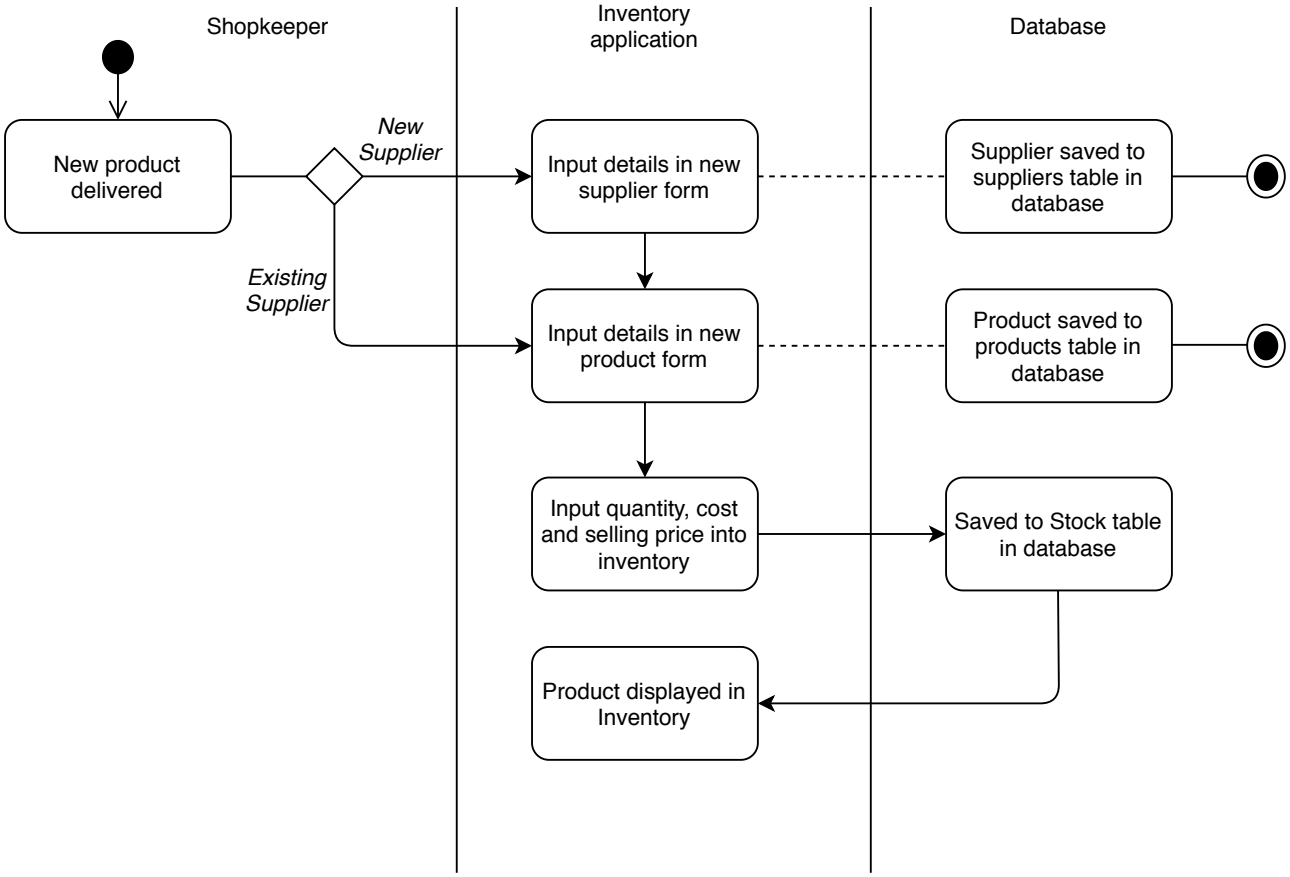
Above is an example of a class diagram. It shows the four classes used in my shop application. The diagram shows the parameters and methods each class will have. It also shows relationships between classes, for example the Product class links to the Category class by the category_id.

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



Above is an object diagram that shows a specific instance of each class and how they interact with each other.

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

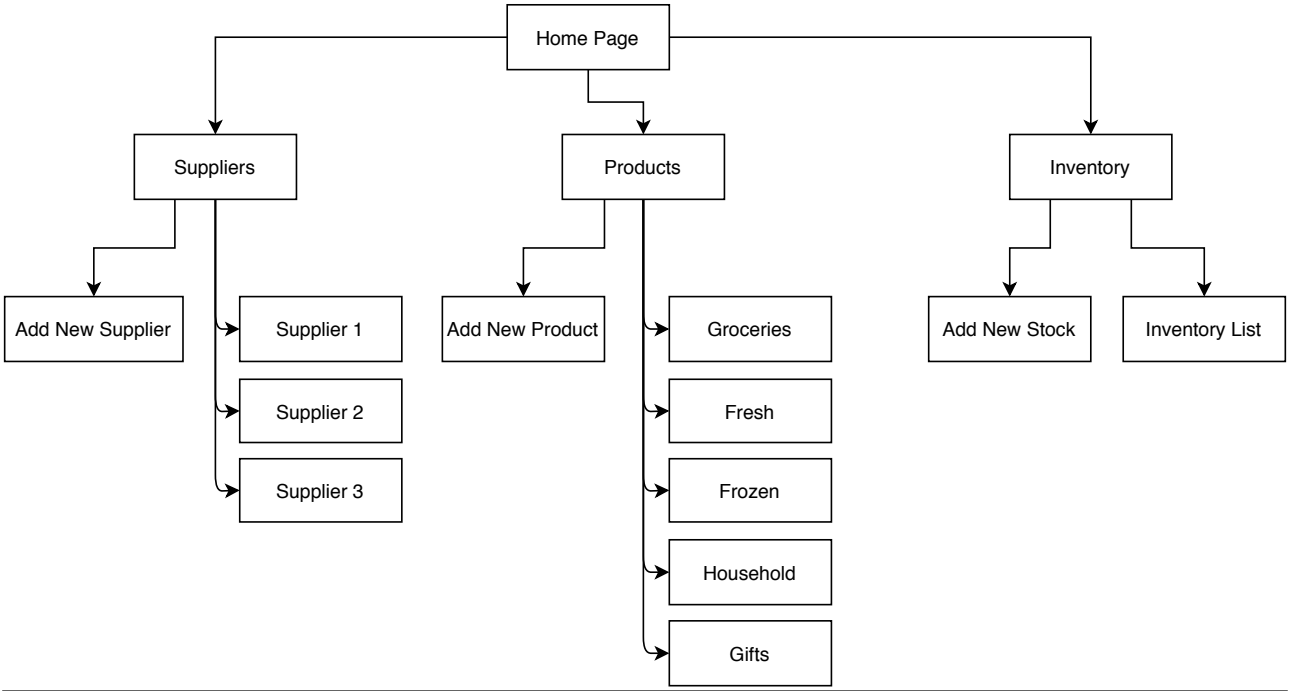


Above is an activity diagram showing the process of receiving a new item of stock, entering it into the system and then displaying it in the inventory.

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

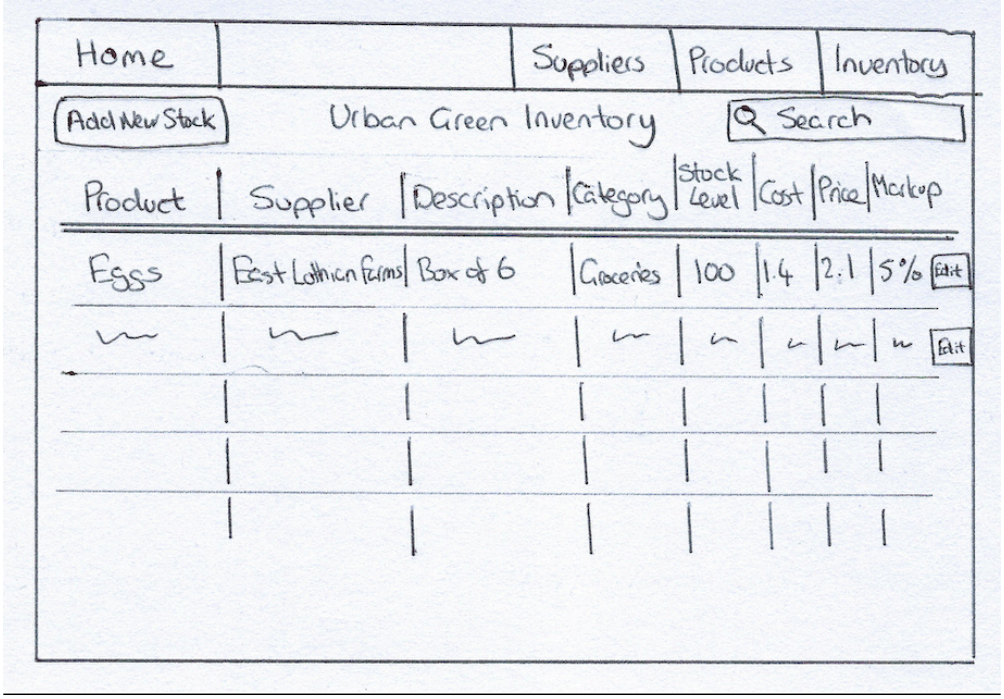
Constraint Category	Implementation Constraint	Solution
Hardware and software platforms	The initial planning was for a web based app however market research has shown a strong customer need for the app to be usable on mobile devices, particularly in smaller stores. The current proposal is for the app to be built with Ruby, if the app is to be fully responsive, especially on mobile devices, this may not be the best solution.	Delay the start of the build to allow a short period of additional planning and research into the best languages/interfaces to use if the future of the app is to be more mobile based than initially predicted. The long term cost saving will be in getting the right solution from the start.
Performance requirements	At the first iteration the product will require a good internet connection to work. This may be an issue for customers, especially smaller stores, where internet may be unreliable or not currently in place, incurring an extra cost to clients to use our system.	Explore implementing an offline option as early in the development as possible to allow access to customers without a regular internet connection.
Persistent storage and transactions	Secure data storage is essential for the success of the project and as the product scales the demands for this will increase quickly. Currently there is limited cloud storage expertise within the team.	For the first stages of the project data will be hosted by a 3rd party cloud storage/management company. This will be an extension of a current contract with a company we already work with and know to be reliable.
Usability	The app needs to be intuitive and easy for temporary shop staff to familiarise themselves with quickly to avoid delays to service.	Carry out user testing to ensure app is suitable for use.Offer short training sessions for shop staff when installing product.
Budgets	Current budget will support the app to be developed and deployed. There is not currently funding for the next stage of features development and for ongoing support and bug fixes.	Aim to get a functional prototype built early in the planning to allow the sales team to secure advance orders. The package sold should include a fee for ongoing support.
Time	Currently the team are committed to 20 working days for this project over the next month, with no additional time confirmed.	To ensure the project is completed successfully a review has been agreed at the halfway point (10 working days). If at this stage it is forecast we will need to increase the developers time on the project there is flexibility to extend by an additional 10 days.

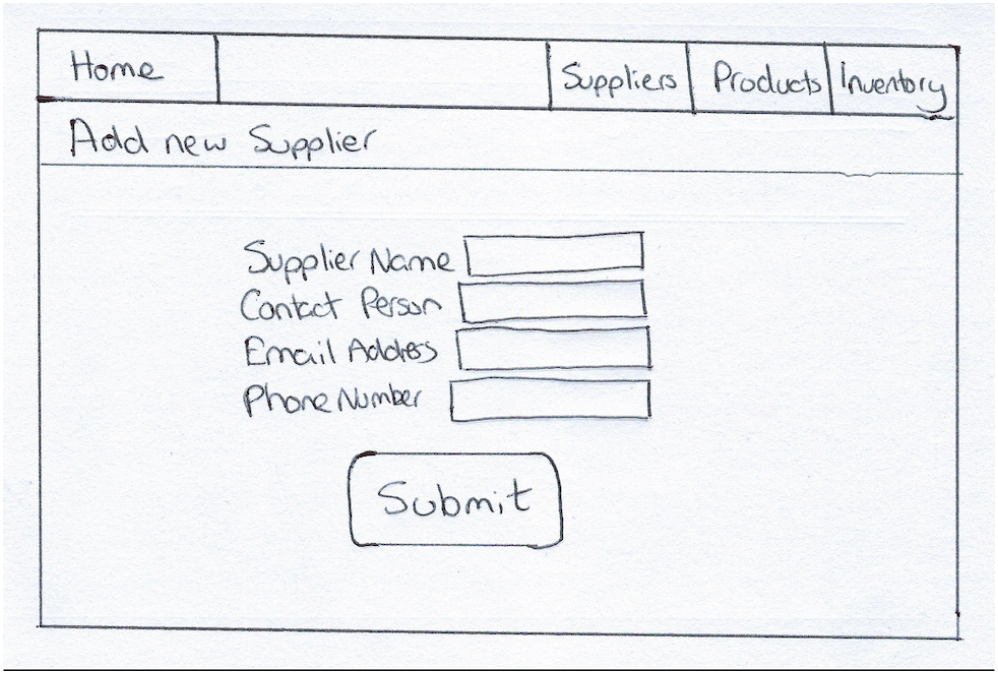
Unit	Ref	Evidence
P	P.5	User Site Map



Above is a user site map for a shop inventory application. It shows the layout of the pages within the site and how the user can navigate through the site. It is useful as starting point to building the app as it shows the overall structure of the site and how the separate pages relate to each other.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams





Above are two initial wireframe diagrams I drew for my individual Ruby shop application Project. They show the layout of the overall site with a top navigation bar. The first diagram is the Inventory page, I used it to work out how the page might look and how a large amount of information could be displayed neatly. The second diagram shows the page for adding a new supplier, the design was deliberately kept clear and clutter free to improve the user experience.

Week 5

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

Takes the big array of threats e.g. [poaching, climate change, poaching, building, deforestation]

Sorts thrugh the array comparing each pair of values by the number of times they occur (by using .length)

Takes the count of the number of second items away from count of first items

If it is a positive number then sort knows that item occurs more times and it is place at the end of the array

Sort iterates through the array doing this for every item comparing it with the next item

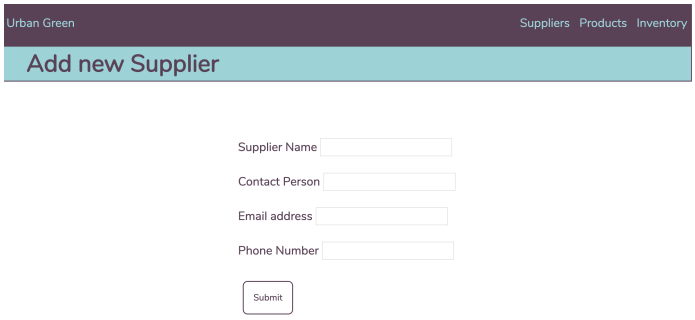
At the end the array is sorted with the items that occur least at the start and that occur most at the end

The last item of the array is then returned

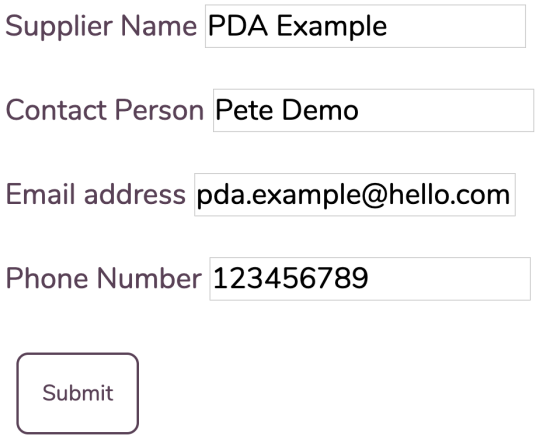
Above is an example of Pseudocode I wrote while trying to figure out a method for our JavaScript group project. The method needed to do several different things to achieve the final result and I was struggling to write it. By writing the steps out in Pseudocode it clarified for me what each part needed to do and made writing the final method much easier.
Below is the final method and how it fits within the larger method I wrote:

```
biggestThreats(){
  this.threats = []
  this.favouriteAnimals.forEach(animal => this.threats.push(animal.threats))
  const newArray = [].concat.apply([], this.threats)
  console.log(newArray);
  this.biggestThreat = this.findThreat(newArray)
  this.findThreatObject(this.biggestThreat);
},

findThreat(array){
  return array.sort((a,b) => {
  array.filter(v => v===a).length
  - array.filter(v => v===b).length
}).pop();
},
findThreatObject(threat) {
  this.biggestThreatObject = this.threatObjects.find(threatObject => threatObject.name === threat)
}
```

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

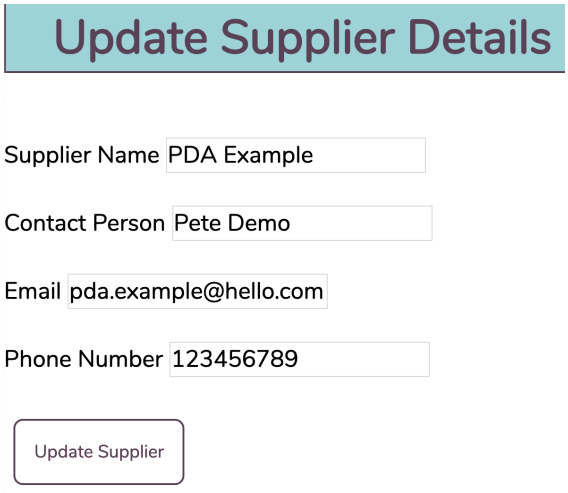


This shows the “Add New Supplier” page from my shop inventory project.



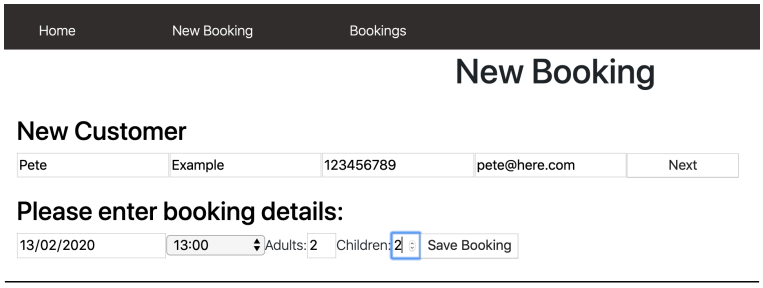
This shows some example data being entered by the user. They then click submit to save the information.

The supplier is now displayed in the suppliers list. The user can click on the supplier name to view their details.



From the view page the user can then delete the supplier or edit their details. Editing brings up a form pre-populated with the saved information which the user can then amend and then click the ‘Update Supplier’ button to save the new information.

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



Above is an example of a booking being entered into a Restaurant Booking System built as my final group project. The user has a choice of using an existing customer or entering a new customer, in this case a new customer is being created. The customer and booking details are then saved in the database. The images below show that the information is now successfully saved. (The new customer and new booking are the bottom line in each table).

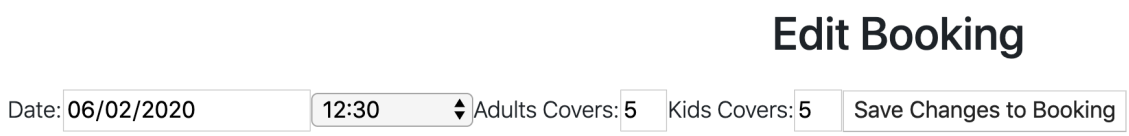
```
restaurantbookingservice=# select * from customers;
 id |        email         | first_name | last_name  | phone_number
----+----------------------+------------+------------+--------------
  1 | joseph@gmail.com     | Joseph     | Adams      | 07111111111
  2 | nelson@gmail.com     | Nelson     | Mandela    | 07222222222
  3 | fitzer@gmail.com     | Fitzer     | Konig      | 07333333333
  4 | eugene@gmail.com     | Eugene     | Hualala    | 07444444444
  5 | paul@gmail.com       | Paul       | McLaren    | 07384638463
  6 | harrold@gmail.com    | Harrold    | Williamson | 07283649876
  7 | jojo@gmail.com       | Jojo       | Russotto   | 0738452987
  8 | luise@gmail.com      | Luise      | Camaro     | 07193845208
  9 | dominika@gmail.com   | Dominika   | Panas      | 0728364091
 10 | john@gmail.com       | John       | Stevenson  | 0718293674
 11 | mussolini@gmail.com  | Paolo      | Mussolini  | 07999999999
 12 | terminator@gmail.com | Marcin     | Terminator | 07293710983
 13 | salmonPink@gmail.com | Naz        | SalmonPink | 0719298370497
 14 | stevenson@gmail.com  | Steven     | Stevenson  | 0729384762
 15 | neverAngry@gmail.com | Michael    | NeverAngry | 07873298374
 16 | pete@here.com        | Pete       | Example    | 123456789
(16 rows)
```

```
restaurantbookingservice=# select * from bookings;
 id | adults_covers |    date    | kids_covers | time  | customer_id
----+---------------+------------+-------------+-------+-------------
  1 |             6 | 2020-02-05 |           9 | 12:00 |           1
  2 |             5 | 2020-02-05 |           0 | 14:00 |           2
  3 |             3 | 2020-02-05 |          12 | 17:00 |           3
  4 |             3 | 2020-02-05 |           6 | 17:30 |           4
  5 |             3 | 2020-02-06 |           1 | 12:30 |           5
  6 |             3 | 2020-02-06 |           2 | 13:30 |           6
  7 |             3 | 2020-02-06 |           0 | 16:15 |           7
  8 |             5 | 2020-02-06 |           0 | 19:00 |           8
  9 |             7 | 2020-02-06 |           1 | 19:30 |           9
 10 |            11 | 2020-02-07 |           4 | 12:15 |          10
 11 |             2 | 2020-02-07 |           0 | 12:30 |          11
 12 |             8 | 2020-02-07 |           8 | 16:00 |          12
 13 |             8 | 2020-02-07 |           8 | 16:15 |          14
 14 |             8 | 2020-02-07 |           8 | 16:30 |           6
 15 |             7 | 2020-02-07 |           0 | 17:30 |          13
 16 |             5 | 2020-02-07 |           1 | 19:15 |          14
 17 |             2 | 2020-02-07 |           0 | 20:15 |          15
 18 |             2 | 2020-02-13 |           2 | 13:00 |          16
(18 rows)
```

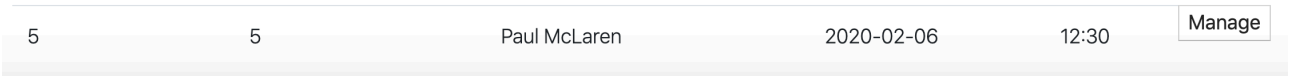
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program

Kids Covers	Adults Covers	Customer Name	Date	Time	
1	3	Paul McLaren	2020-02-06	12:30	Manage
2	3	Harrold Williamson	2020-02-06	13:30	Manage
0	3	Jojo Russotto	2020-02-06	16:15	Manage
0	5	Luise Camaro	2020-02-06	19:00	Manage
1	7	Dominika Panas	2020-02-06	19:30	Manage

The restaurant booking system project allows staff to manage their bookings. They can display all the bookings for any date. If a customer calls up and wants to make a change then they can click ‘Manage’ to view and edit the booking. In this example ‘Paul McLaren wants to change his booking from 3 adults and 1 child to 5 adults and 5 children.



In the edit booking screen the user updates the number of covers with the new information and clicks ‘Save Changes to Booking’



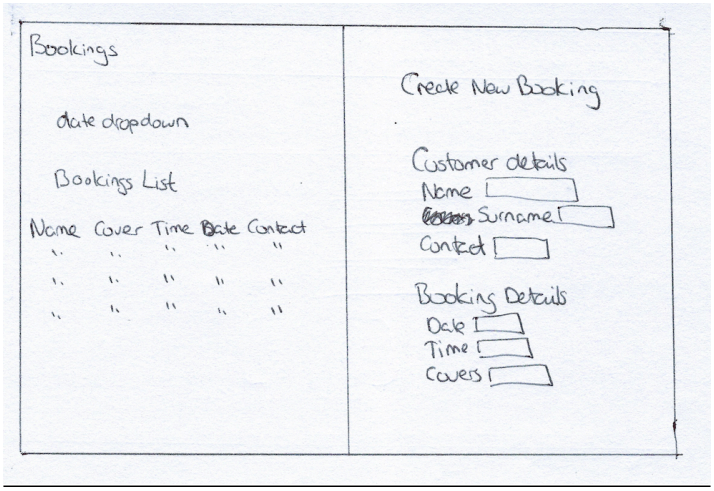
5	5	Paul McLaren	2020-02-06	12:30	Manage

The new information is then displayed to them on the bookings screen, confirming the change has been made successfully.

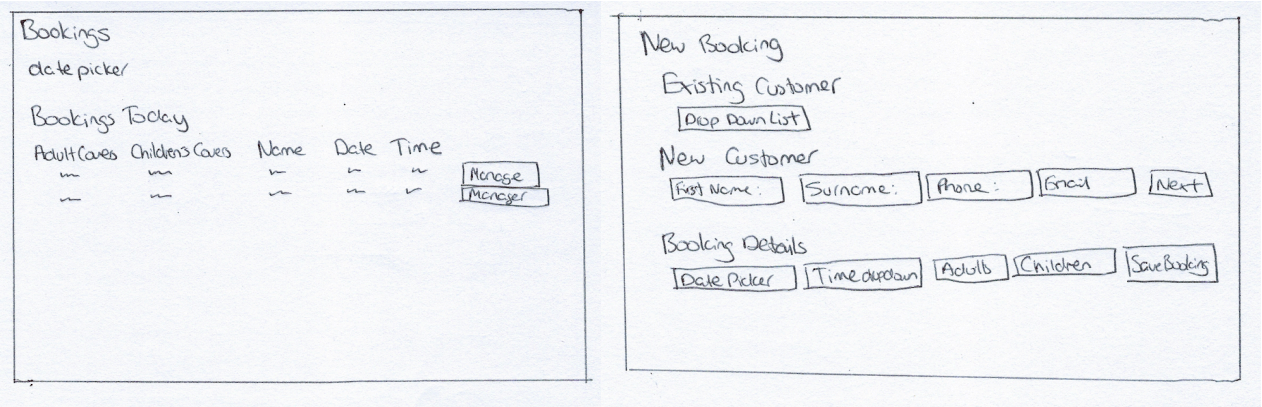
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.

This is the home page of my solo Ruby shop inventory project. The Github link is: https://github.com/MaryBobs/ruby_project_urban_green

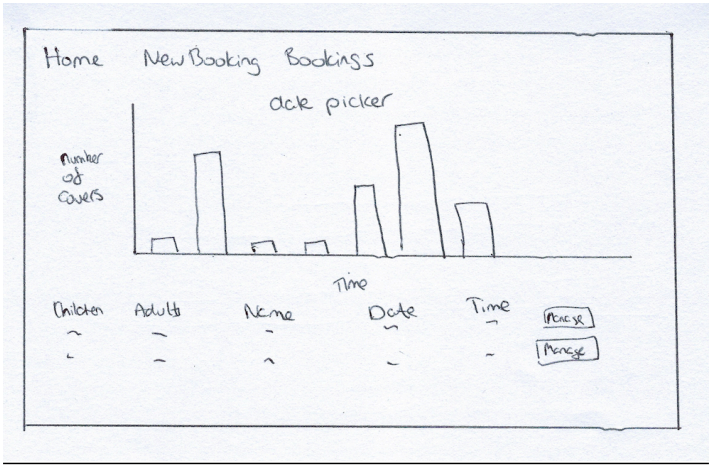
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.



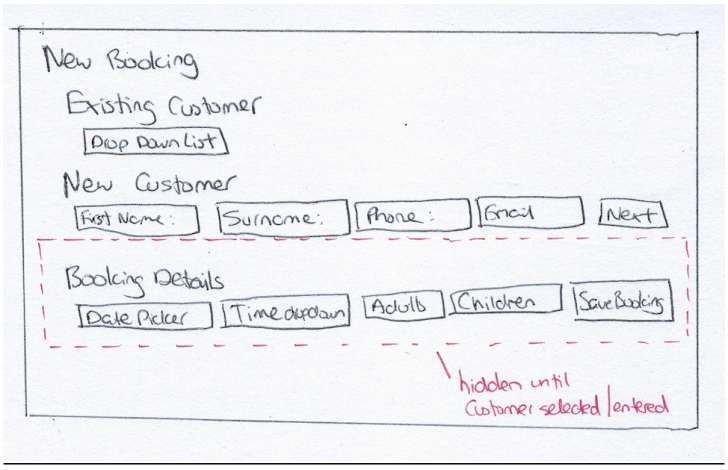
The initial draft sketches for our group project restaurant booking app had one main page where the user managed everything. They could see a list of bookings for a day and create new bookings.



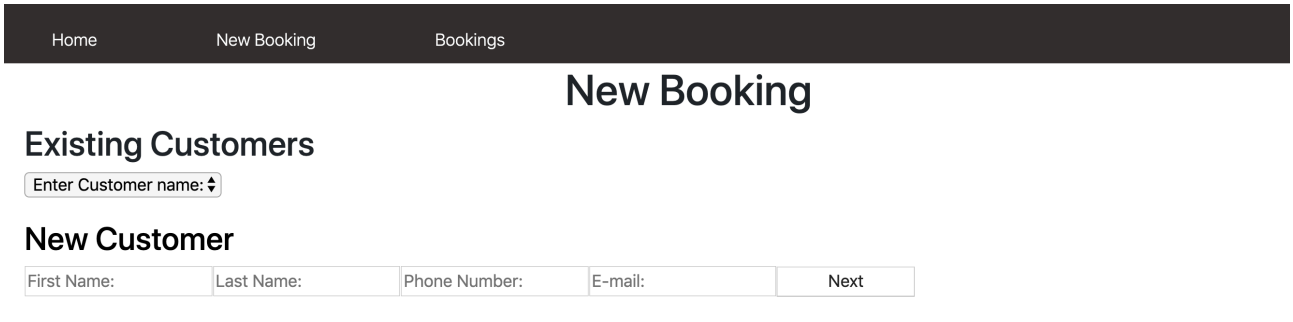
By the end of the planning stage we had re-thought the design to make it less cluttered and split the app into two sections, separating displaying bookings from creating new bookings.
East access to each separate page would still be maintained through a navigation bar at the top of the screen. We also added the ability to manage bookings from the Bookings List page by adding manage buttons and we decided to use a date picker rather than a drop down menu for choosing the date as it felt more user friendly. On the New Booking side we added the ability to add an existing customer or create a new customer when making a new booking. This would allow us to keep data about customers visits and we aimed to implement offers for regular customers as an extension.



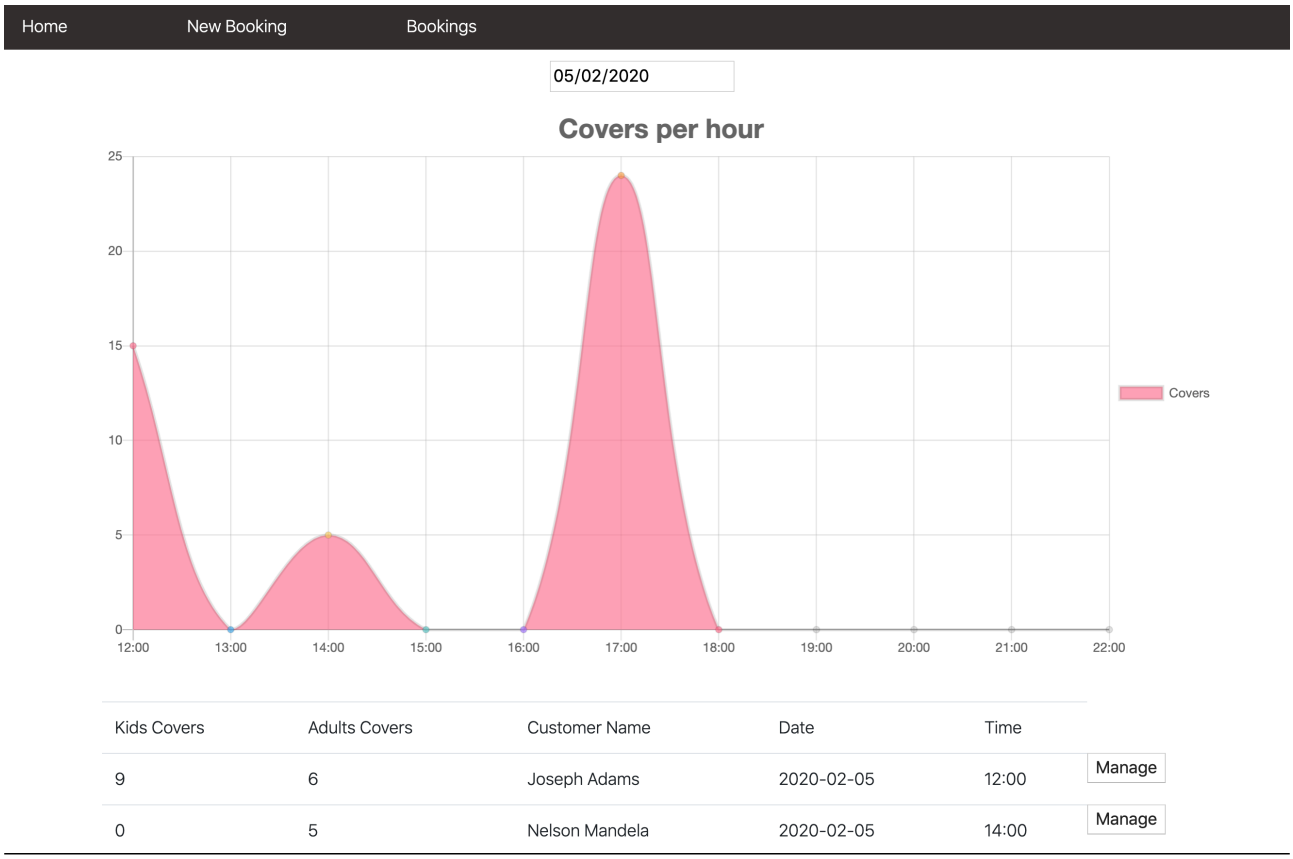
As the project developed we decided a visual representation of the number of bookings expected across a day would be more useful to busy restaurant staff. So we altered the design of the bookings page again to include a bar chart.



We also realised there was a potential issue with the design of the create booking page as a user could attempt to add both an existing and new customer to a booking. To fix this we decided to hide the booking details section until either an existing customer was selected or a new customer's details added. The booking details form then appears and the customer option not being used is hidden.



The final new booking page design did not change much more and looks like this, with the new booking form appearing once a customer choice is made.



There was one further change to the bookings page, we decided to use a different style of chart to better show the flow of bookings across the day, rather than the more rigid bar chart we had first tried.

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

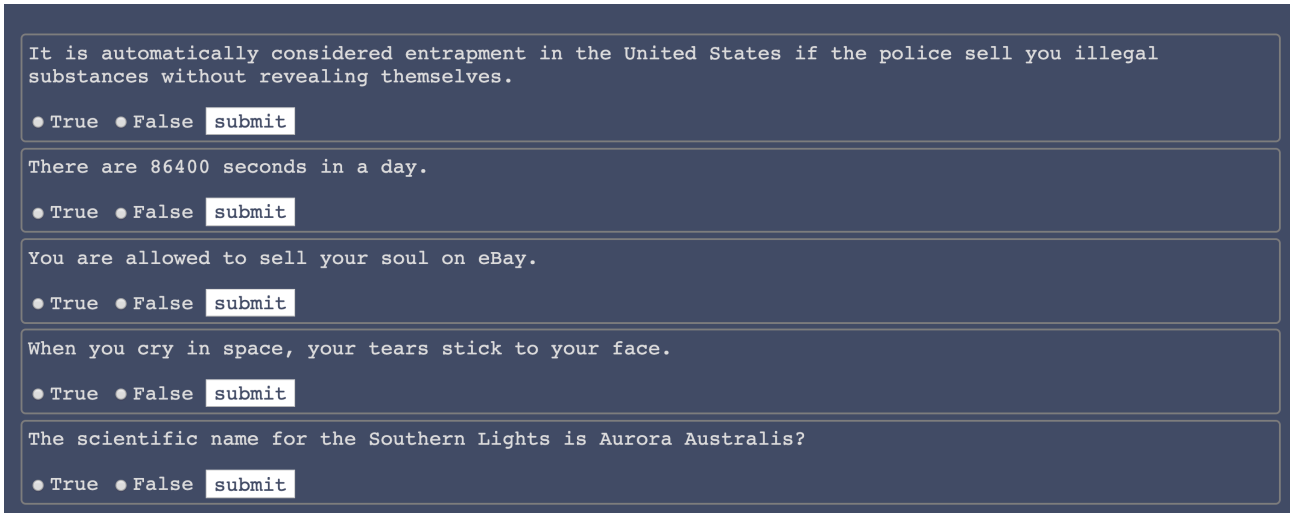
```
mounted(){
  fetch("https://opentdb.com/api_category.php")
  .then(res => res.json())
  .then(categories => this.categories = categories.trivia_categories)
```

The above shows my program using a trivia API. The initial API call populates a list of categories available.

The user is then asked to input a number of questions(up to ten) and choose a category.

```
fetchNewQuiz: function() {
  const url = `https://opentdb.com/api.php?amount=${this.quizNumberOfQuestions}&category=${this.quizCategory}&type=boolean&encode=url3986`;
  fetch(url)
  .then(res => res.json())
  .then(data => this.questions = data.results)
},
```

These choices are then included in the next API call which returns the correct number of questions for the chosen category.



Week 8

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

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:

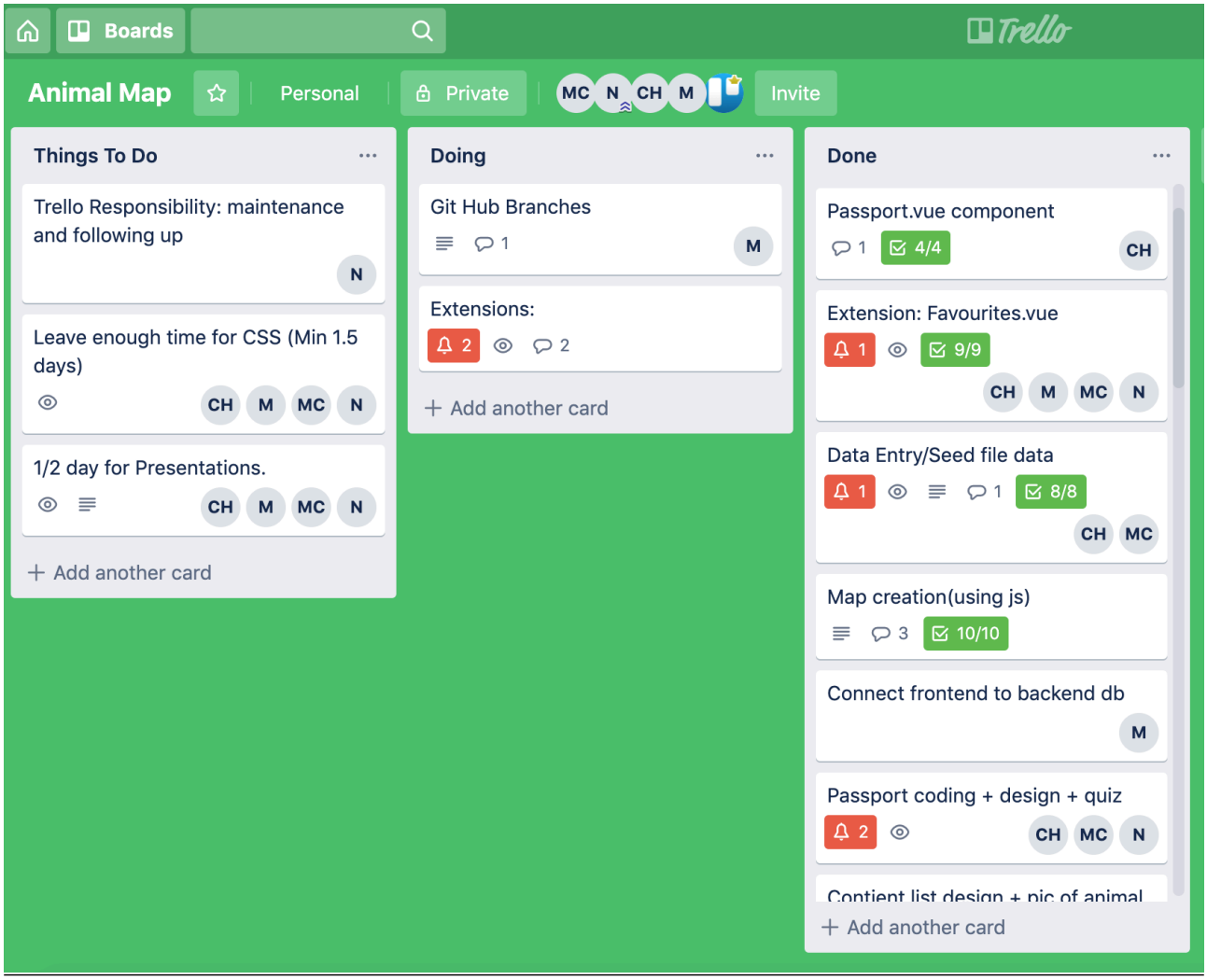
- view some educational content on a particular topic
- be able to interact with the page to move through different sections of content

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.

This is the brief from my first group project. We built an Educational App aimed at teaching primary school age children about endangered species.

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



This is an example of our Trello board from the Educational App group project. We used Trello to keep track of tasks, assign tasks to individuals and to make sure everyone knew what was being worked on at any time.

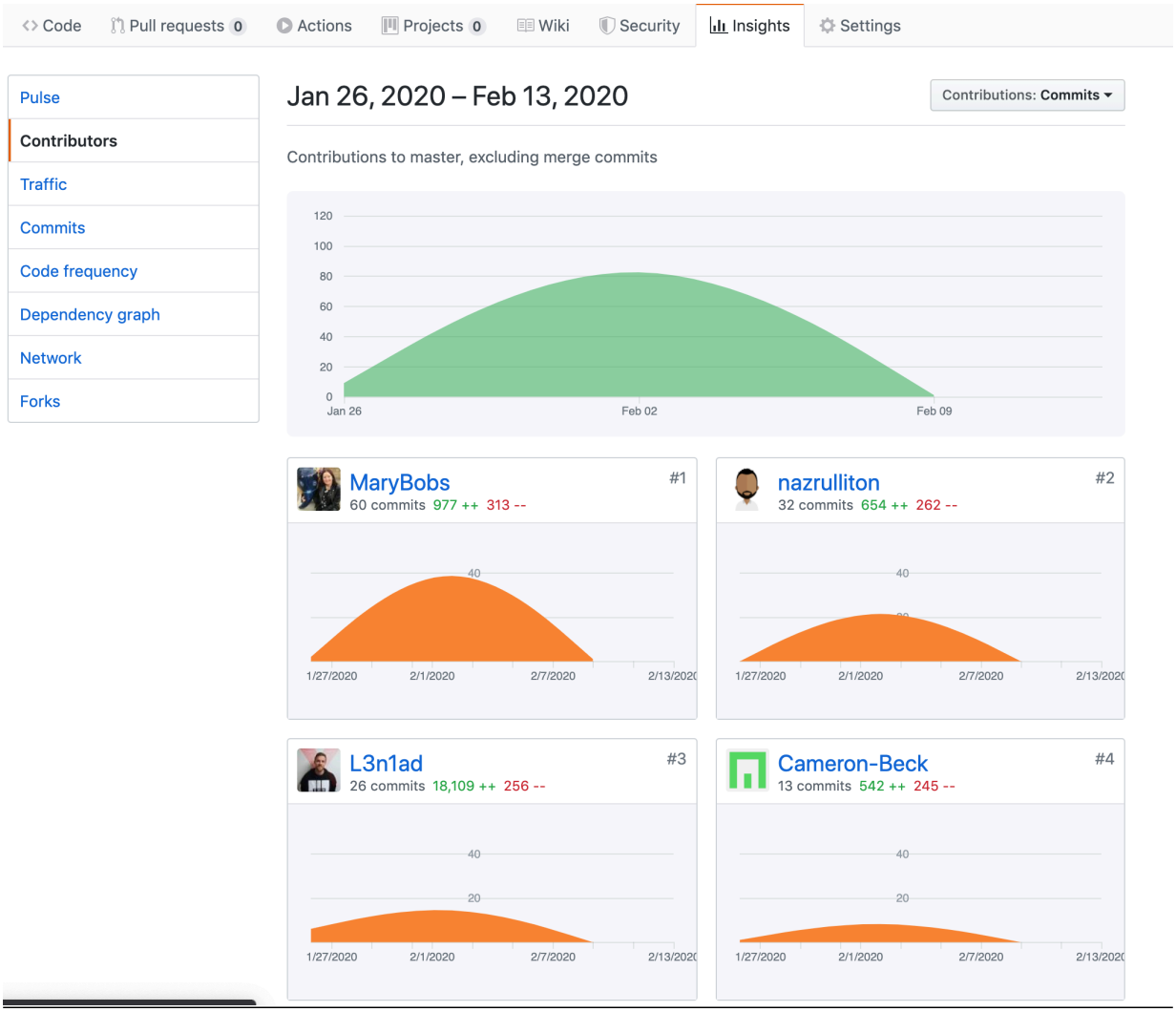
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

Acceptance Criteria	Expected Result	Pass/Fail
The user is able to select a continent.	The user clicks on the paw print above a continent and a pop up opens displaying the continent name and five of its endangered animals.	Pass
The user can see an individual animals ‘passport’.	The user clicks on an animal picture and a pop up opens displaying a ‘passport’ page with more details about the animal.	Pass
The user can add an animal to their favourites.	The user clicks on the heart symbol next to the animal’s passport picture. The heart changes colour to pink and the animals picture appears in the favourites bar.	Pass
The user is told when they answer a quiz question incorrectly.	The user selects an incorrect answer and a large red cross appears on the passport.	Pass
A science fact is displayed when a user answers a quiz question correctly.	The user selects the correct answer and a science fact appears on the passport along with a large green tick.	Pass

This is part of an acceptance criteria and test plan for our Endangered group project. It focuses on the ‘passport’ element of the application and how the user interacts with it.

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.

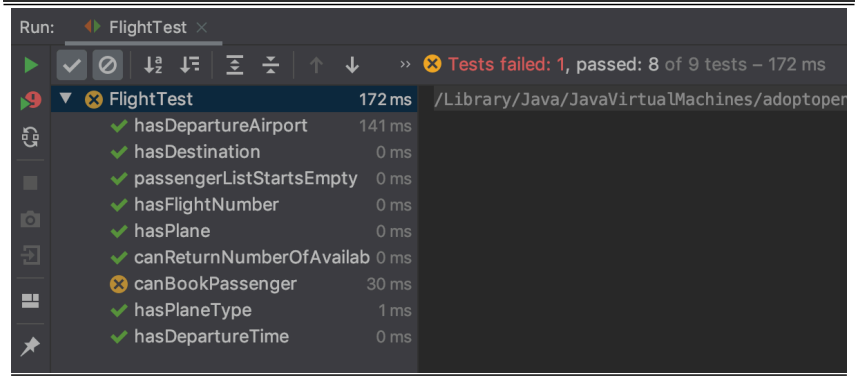


This shows the team I worked with for the final group project, a restaurant bookings application.

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

```
@Test
public void canBookPassenger() {
    flight.bookPassenger(passenger1);
    assertEquals( expected: 1, flight.countPassengers());
}
```

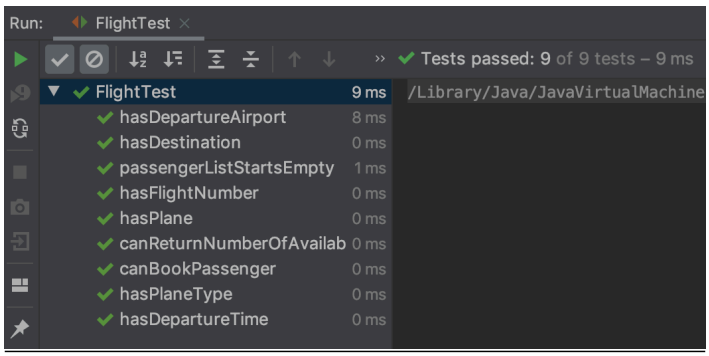


Above is a test to check if the bookPassenger method correctly adds a passenger to the plane passenger array. As the screenshot shows the test is currently failing.

```
public void bookPassenger(Passenger passenger) {
    if (this.plane.getCapacity() == this.countPassengers()) {
    this.passengers.add(passenger); }
}
```

Inspecting the function (above) shows that there is an error in it's construction, it is checking to see if the plane capacity is equal to the passenger count, when it should be checking if the plane capacity is greater than the passenger count and therefore another passenger can be booked on. Once corrected (below) the test now passes.

```
public void bookPassenger(Passenger passenger) {
    if (this.plane.getCapacity() > this.countPassengers()) {
    this.passengers.add(passenger); }
}
```



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

```java
public class Flight {

    private ArrayList<Passenger> passengers;
    private Plane plane;
    private String flightNumber;
    private AirportCode destination;
    private AirportCode departureAirport;
    private String departureTime;

    public Flight(Plane plane, String flightNumber, AirportCode destination, AirportCode departureAirport, String departureTime) {
        this.passengers = new ArrayList<Passenger>();
        this.plane = plane;
        this.flightNumber = flightNumber;
        this.destination = destination;
        this.departureAirport = departureAirport;
        this.departureTime = departureTime;
    }

    public int countPassengers() {
        return this.passengers.size();
    }

    public Plane getPlane() {
        return this.plane;
    }

    public PlaneType getPlaneType() {
        return this.plane.getType();
    }

    public String getFlightNumber() {
        return this.flightNumber;
    }

    public AirportCode getDestination() { return this.destination; }
```

The above demonstrates the use of Encapsulation in a programme. Also known as data hiding, encapsulation wraps data variables together with the methods that act on the data. The variables are set as private so will be hidden from other classes, they can only be accessed through the public methods provided in the class.
Encapsulation gives more control to a class over how it stores data without external users of the class changing anything.

Week 12

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

```
public class Wallet {
    private String name;
    private ArrayList<IScan> items;

    public Wallet(String name) {
        this.name = name;
        this.items = new ArrayList<IScan>();
    }

    public void addItem(IScan item) {
        this.items.add(item);
    }
```

This programme has a Wallet class that contains an array list of different items. All items in the wallet implement the IScan interface which means any of them can be added to the items array list using the addItem method which requires an item of type IScan. It does not matter if the IScan item is a ticket, debit card, loyalty card etc, as long as it is on type IScan it can be added.

```
public class LoyaltyCard implements IScan {
    private String barCode;
    private String vendor;

    public LoyaltyCard(String barCode, String vendor) {
        this.barCode = barCode;
        this.vendor = vendor;
    }
```

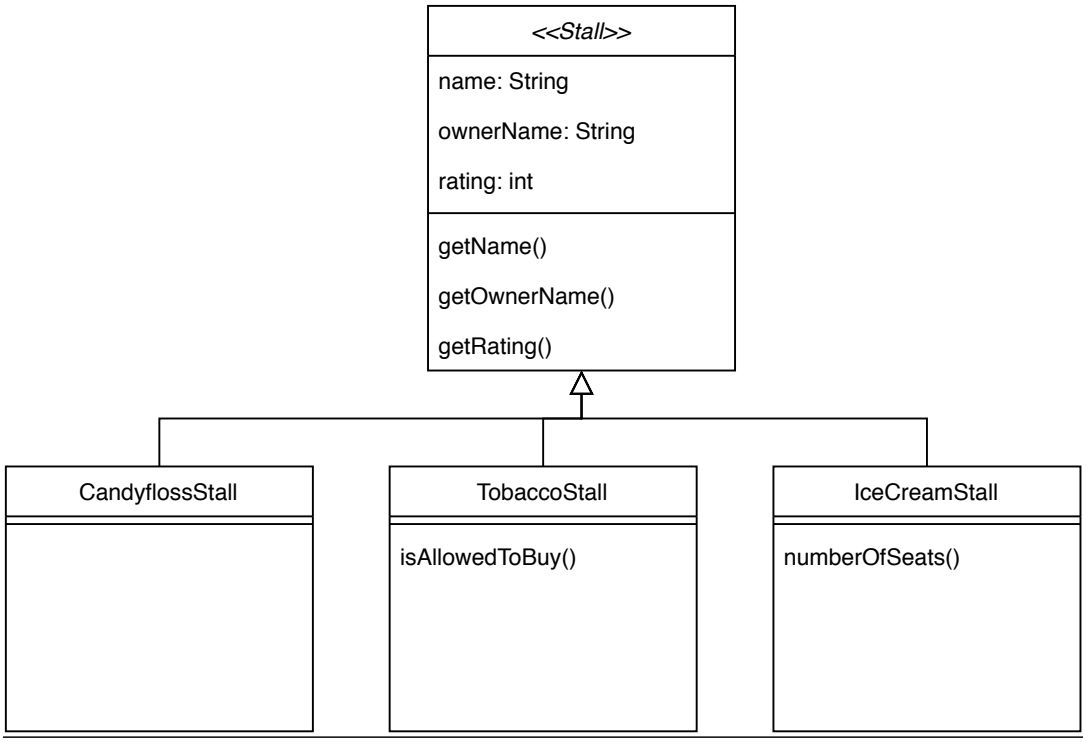
```
public class Ticket implements IScan {
    private String bookingRef;
    private String qrCode;
    private String date;

    public Ticket(String bookingRef, String qrCode, String date) {
        this.bookingRef = bookingRef;
        this.qrCode = qrCode;
        this.date = date;
    }
```

Above are examples of two classes that implement the IScan interface.

```
public interface IScan {
    public String scan();
}
```

The IScan interface which returns a String.

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



Above is an example of an inheritance diagram. The diagram shows the parent class Stall and three child classes CandyflossStall, TobaccoStall and IceCream stall. The three children inherit all the variables and methods in the parent class. Two of the children also have addition methods specific to them which are not inherited.

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 abstract class Attraction implements IReviewed {
    private String name;
    private int rating;
    private int visitCount;

    public Attraction(String name, int rating) {
        this.name = name;
        this.rating = rating;
        this.visitCount = 0;
    }

    public String getName() {
        return name;
    }

    public int getRating() {
        return rating;
    }
```

```
public class Dodgems extends Attraction implements ITicketed {

    public Dodgems(String name, int rating) {
        super(name, rating);
    }

    public double defaultPrice() {
        return 4.50;
    }

    public double priceForVisitor(Visitor visitor) {
        if (visitor.getAge() < 12 ) {
            return this.defaultPrice() / 2; }
        else {
            return this.defaultPrice();
        }
    }
}
```

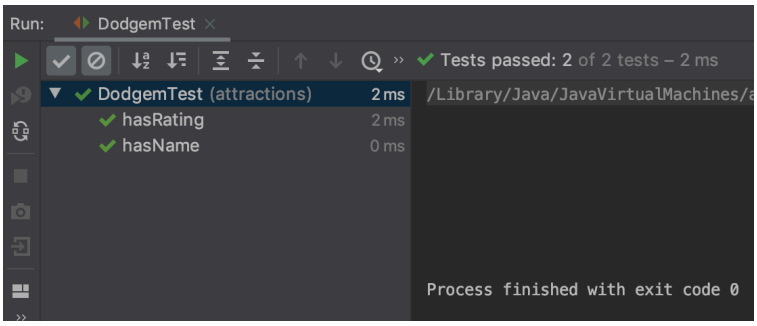
Above (left) shows the class Attraction, which has name and rating parameters and methods to getName and getRating.

Above (right) shows the Dodgems class which inherits from Attraction.

Below (left) shows the Dodgem test file, an instance of the Dodgems class is created and then two tests run to check if we can successfully use the .getName() and .get Rating() which exist in the parent Attraction class but not in the Dodgems class.

Below (right) shows the tests passing proving that we have successfully inherited the methods from the parent class.

```
public class DodgemTest {

    Dodgems dodgems;

    @Before
    public void setUp() throws Exception {
        dodgems = new Dodgems( name: "Bumper Cars", rating: 5);
    }


    @Test
    public void hasName() {
        assertEquals( expected: "Bumper Cars", dodgems.getName());
    }

    @Test
    public void hasRating() {
        assertEquals( expected: 5, dodgems.getRating());
    }
```



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.

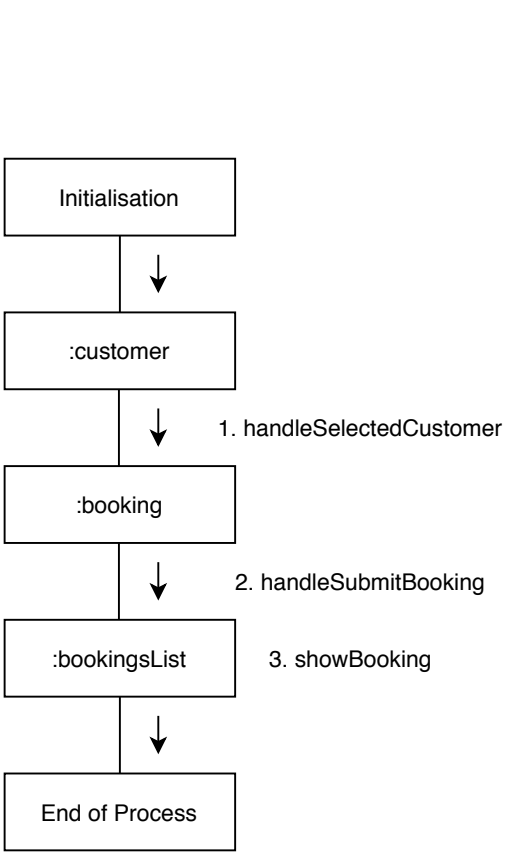
```ruby
def markup()
  profit = (@selling_price.to_f - @buying_cost.to_f)
  result = (profit / @buying_cost.to_f) * 100
  return result.round(2)
end
```

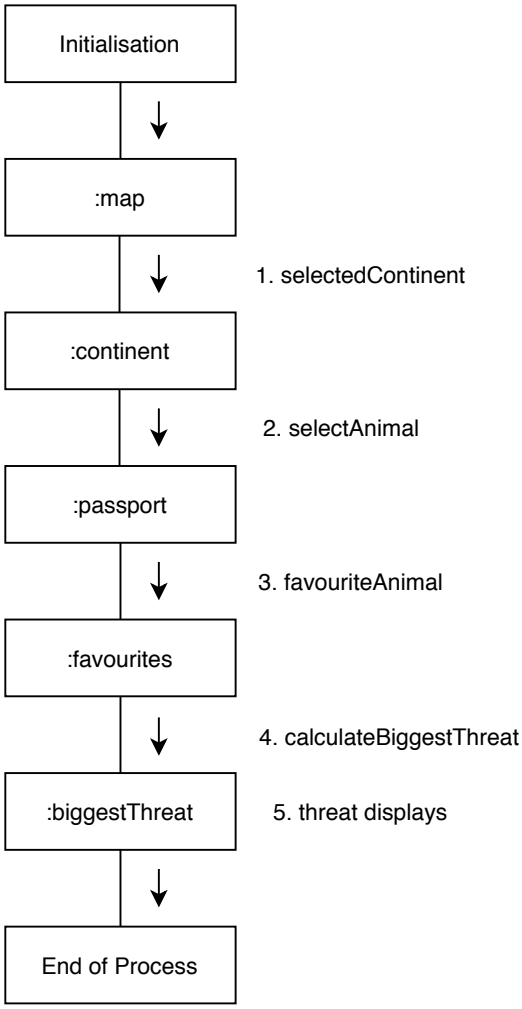
Above is small function from early in the course. It calculates the markup on an item in a stock inventory by taking the cost from the buying price and multiplying by 100. I've chosen it as although it looks like a simple function to me know it was one of the first functions I remember working out by myself when coding was still very new. It is also a small but essential part of the inventory application.

```java
public void battleEnemy(Room room) {
    while (room.getTotalPlayerHealth() > 0 && room.getEnemy() != null) {
        for (Player player : room.getPlayers()) {
            player.activateSkill(room.getEnemy());
            if (room.getEnemy().getHealth() < 0) {
                room.killEnemy();
                break;
            }
            room.getEnemy().attack(player);
        }
    }
    if (room.getTotalPlayerHealth() > 0) {
        for (Valuable treasure : room.getTreasures()) {
            room.getPlayers().get(0).collectTreasure(treasure);
        }
    }
    else {
        room.removeAllPlayers();
    }
}
```

Above is a function written towards the end of the course as part of a fantasy adventure homework. I chose it as I couldn't have imagined being able to write a longer more complex function like this when I wrote the first example at the start of the course. The battleEnemy function carries out a battle between and player and an enemy until one of them wins. It works by looping round as long as there is a player with health left in the room and the enemy is still in the room. In each loop each player in the room attacks (using activateSkill) the enemy. If after that the enemy health is reduced to less than 0 the enemy is killed and removed from the room by the killEnemy function and the players win. If the enemy is still alive then it attacks a a player. If the total player health reduces below 0 all players are removed from the room players array and the enemy wins. If the players win then they collect the treasure in the room.

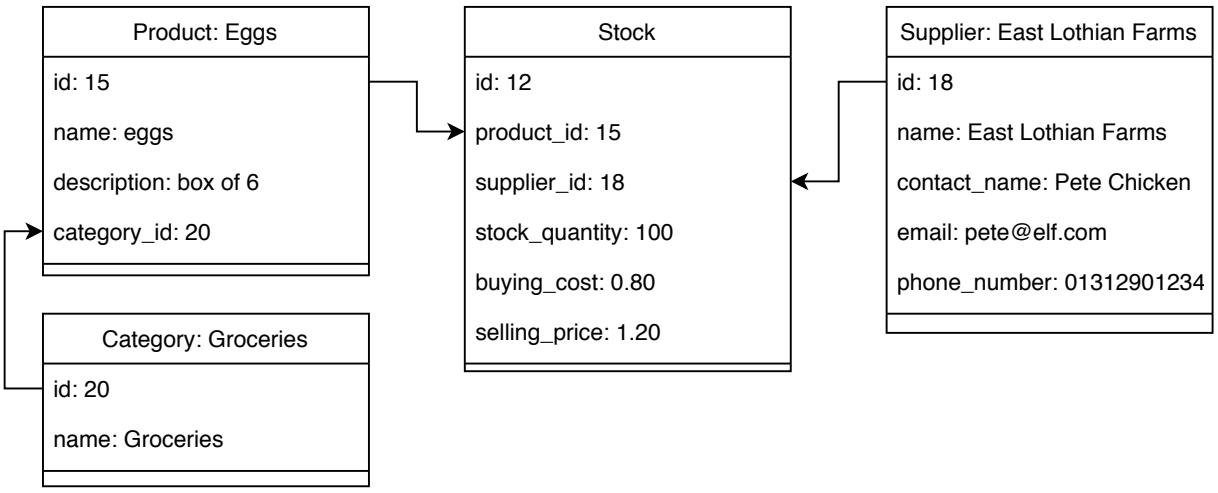
Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).



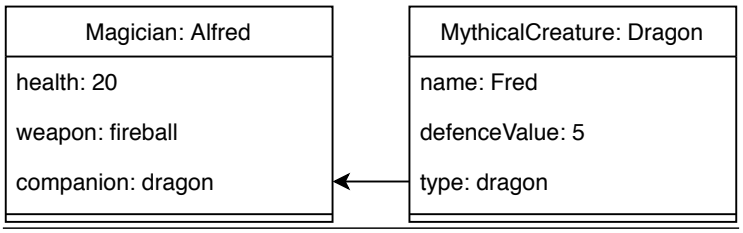


Above are to system interaction component diagrams. The first shows the interaction between components in a restaurant booking system application. The second shows the interactions needed to find a biggest threat in an education application.

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



This object diagram shows instances of each class in an inventory application and how they link together.



This object diagram shows instances of two classes from a fantasy adventure and the link between them.

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

Bug/Error	Solution	Date
“Cannot read property ‘map’ of undefined.”	There was a typo in the prop name ‘customers’ being passed down, this meant the map function couldn’t work as it did not know what it was supposed to be mapping through.	3/2/20
Warning pop up: “! Please fill in this field” meaning cannot create a booking with adding at least one child.	Field had been set as required, removed so bookings could be created without children. All other fields remain as required.	3/2/20
Possible for users to input any time into a booking, no restriction to when restaurant is open.	Refactor to remove user input of time and change to a drop down selector populated by an array of accepted times	4/2/20
Bookings chart title states “Covers per day” but displays covers per hour.	Title of chart corrected to “Covers per hour”	5/2/20
User can select an existing customer but then also enter new customer details.	Redesign of new booking page so the user chooses existing or new customer before new booking form is displayed.	5/2/20

The above is a sample of a bug tracking report from the final Restaurant booking system project.