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.

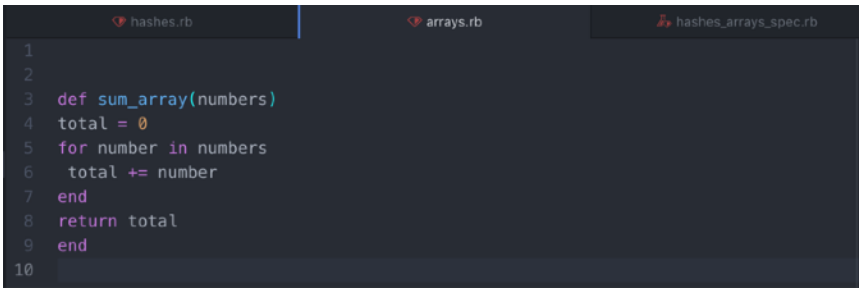
Each point that required details the Assessment Criteria (What you have to show) along with a brief description of the kind of things you should be showing.

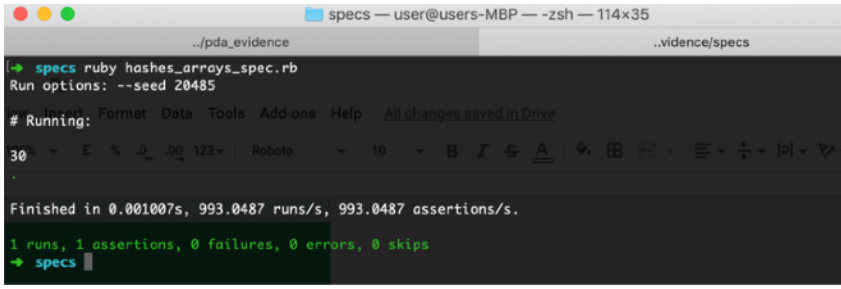
Please fill in each point with screenshot or diagram and description of what you are showing.

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:

Paste Screenshot here





Description here

This is a function that adds together all the integer values in a given array. The function first creates a variable called total and sets it to zero to begin at zero. It then loops over every item in the array and says for every number, increment the running total with this number. The loop then stops once it has run through the array once and return the total. The test also prints the result so you can see what "result" is when the function is called on this particular array.

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:

Paste Screenshot here

hashes.rb | hashes_spec.rb | arrays.rb

```ruby
def stock_count (books)
  return books [:books].count
end
```

```ruby
require( 'minitest/autorun' )
require( 'minitest/rg' )
require_relative( '../hashes' )

class TestBookShop < MiniTest::Test

def setup

  @book_shop = {

    books: [
      {
        name: 'Harry Potter and the Goblet of Fire',
        genre: 'Fantasy'
      },
      {
        name: 'GreenFeast',
        genre: 'Cookery'
      },
      {
        name: 'Carrie',
        genre: 'Horror'
      }
    ]
  }
end

  def test_stock_count
      count = stock_count(@book_shop)
      p count
      assert_equal(3, count)
    end
```

```
../pda_evidence  ..vidence/specs
➜ specs ruby hashes_spec.rb
Run options: --seed 54427

# Running:

3
.

Finished in 0.000921s, 1085.7763 runs/s, 1085.7763 assertions/s.

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

Description here

The inbuilt method .count is called the hash books, which contains nested hashes of individual books with their names and genres, and counts each individual nested hash before returning the number of books.

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:

Paste Screenshot here

data_search_spec.rb | data_search.rb

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

data_search_spec.rb | data_search.rb

```ruby
  @pet_shop = {
    pets: [

      {
        name: "Sir Lancelot",
        pet_type: :dog,
        breed: "Pomsky",
        price: 1000,
      },
      {
        name: "Arthur",
        pet_type: :dog,
        breed: "Husky",
        price: 900,
      },
      {
        name: "Merlin",
        pet_type: :cat,
        breed: "Egyptian Mau",
        price: 1500,
      }
    ],
    name: "Camelot of Pets"
  }
  end

def test_find_pet_by_name__returns_pet
  pet = find_pet_by_name(@pet_shop, "Arthur")
  p pet
  assert_equal("Arthur", pet[:name])
end

end
```

../pda_evidence | ..vidence/specs

```
➜ specs ruby data_search_spec.rb
Run options: --seed 9032

# Running:

{:name=>"Arthur", :pet_type=>:dog, :breed=>"Husky", :price=>900}
.

Finished in 0.001021s, 979.4319 runs/s, 979.4319 assertions/s.

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

Description here

count is called the hash books, which contains nested hashes of individual books with their names and genres, and counts each hash before returning the number of books.

Description here

This function searches data by looking through the hash to see if the name given as an argument in the function matches the name of a pet in the pets hash. If it does, the function returns the pet object and the test prints this out as a hash before comparing only the name part to check it is the right pet.

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:

Paste Screenshot here

```ruby
  @pet_shop = {
    pets: [
      {
        name: "Sir Percy",
        pet_type: :cat,
        breed: "British Shorthair",
        price: 500
      },
      {
        name: "King Bagdemagus",
        pet_type: :cat,
        breed: "British Shorthair",
        price: 500
      },
      {
        name: "Sir Lancelot",
        pet_type: :dog,
        breed: "Ponsky",
        price: 1000,
      },
      {
        name: "Tristan",
        pet_type: :dog,
        breed: "Basset Hound",
        price: 800,
      }
    ]
  }
end

def test_sort_all_pets_by_breed__found
    pets = sort_pets_by_breed(@pet_shop, "British Shorthair")
    p pets
    assert_equal(2, pets.count)
  end


end
```

```ruby
def sort_pets_by_breed(pet_shop, breed)
  same_breed_pets = []
  for pet in pet_shop[:pets]
    same_breed_pets << pet if(pet[:breed] == breed)
  end
  return same_breed_pets
end

```

```
../pda_evidence                                  ..vidence/specs
➜ specs ruby data_sort_spec.rb
Run options: --seed 49184

# Running:

[{:name=>"Sir Percy", :pet_type=>:cat, :breed=>"British Shorthair", :price=>500}, {:name=>"King Bagdemagus", :pet_
type=>:cat, :breed=>"British Shorthair", :price=>500}]

Finished in 0.001087s, 919.9632 runs/s, 919.9632 assertions/s.

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

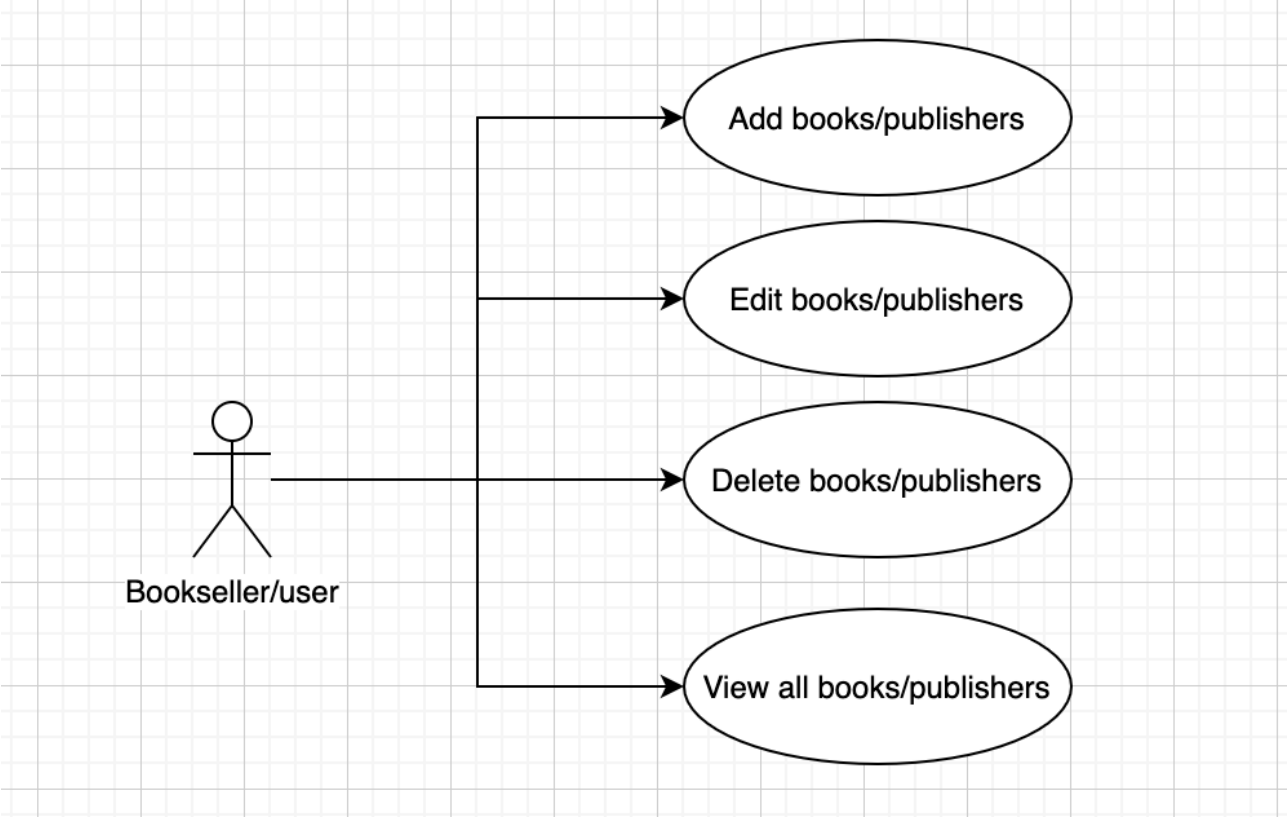
Description here

This function sorts the pets by breed. It first makes an empty array to put the results of running the function into called same_breed_pets. It then runs a loop over all the pets in the pet shop by checking if their breed matches the breed given as an argument in the function - if it does, the result is pushed into the array of same_breed_pets and returned as an array.

Week 5 and 6

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

Paste Screenshot here

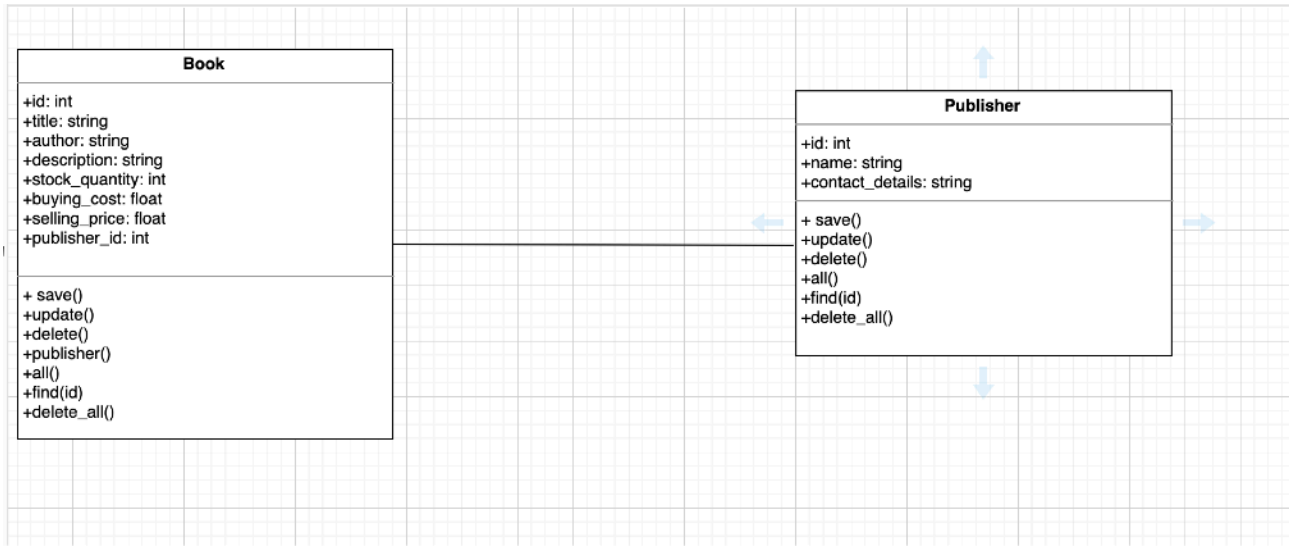


Description here

This use-case diagram gives an overview of the usage requirements for a system. Here, a bookseller must be able to add books and publishers separately, edit books and publishers, delete books and publishers, and view all books and publishers present in the system.

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

Paste Screenshot here

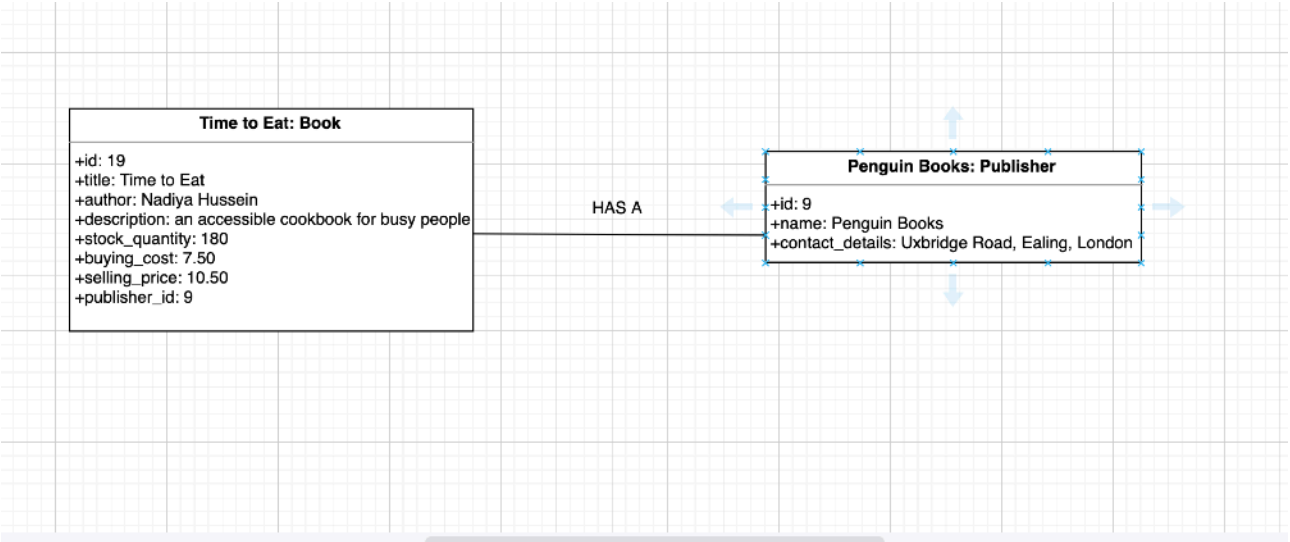


Description here

Book is a class that has the properties of id, title, author, description, stock_quantity, buying_cost, selling_price, and a publisher id. Its methods are save, update, delete, find publisher for book, show all, find by id, and delete all.
Publisher has the properties of an id, a name, and contact details, as well as the methods of save, update, delete, show all, find by id and delete all.

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

Paste Screenshot here

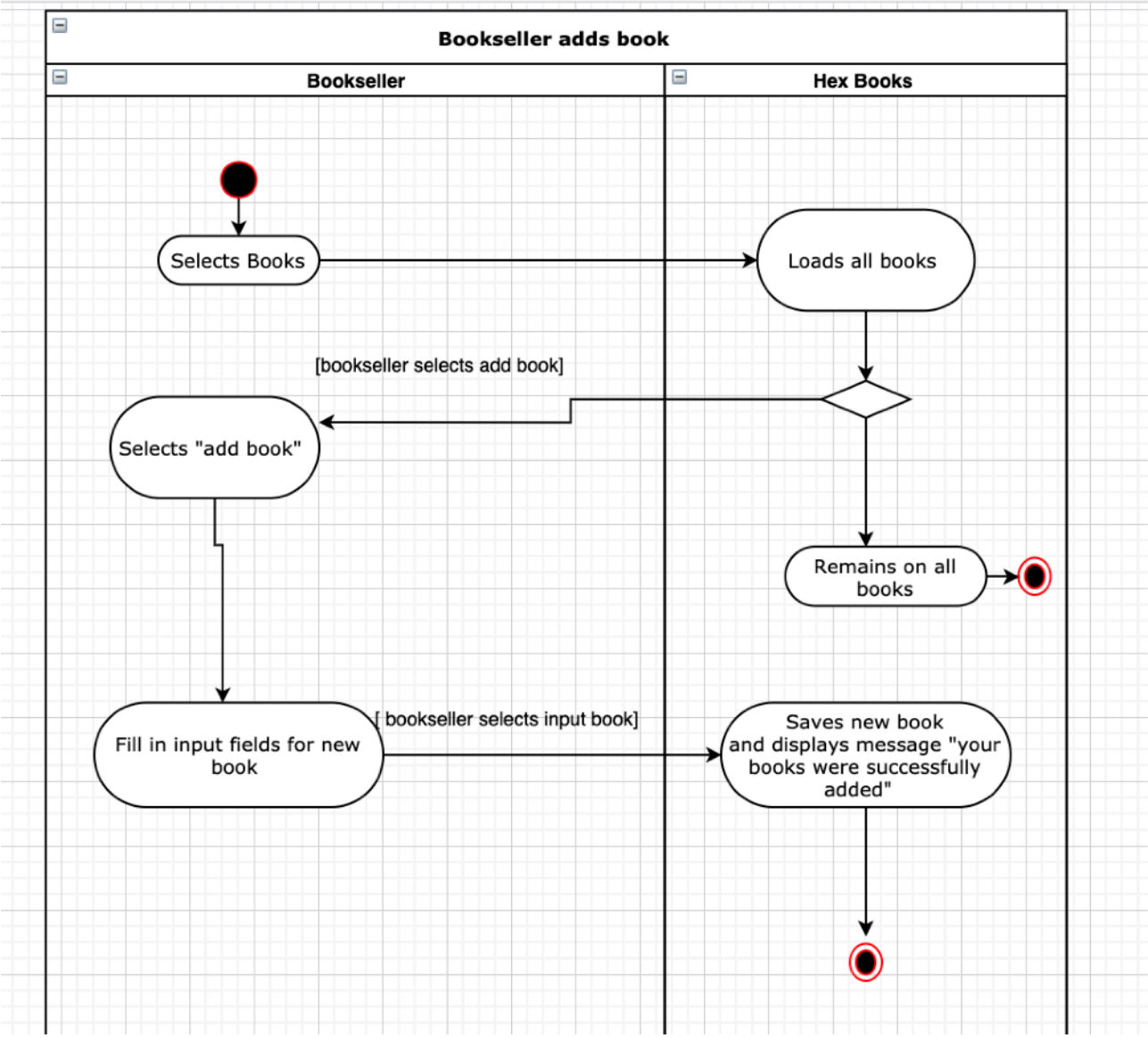


Description here

As an example of the class diagram above, this Book object “Time to Eat” has a Publisher called Penguin Books.

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

Paste Screenshot here



Description here

The bookseller user selects the “books” section of the app, which will then display all books currently stored. The user then either clicks the “add book” button which opens a form to input a new book’s details, or does nothing, which leaves the user on the all books page. If the user adds fills in the input fields for a new book, they then click the “add book method” at the bottom of the form which saves the book and displays the message “your books were successfully added”

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:

Paste Screenshot here

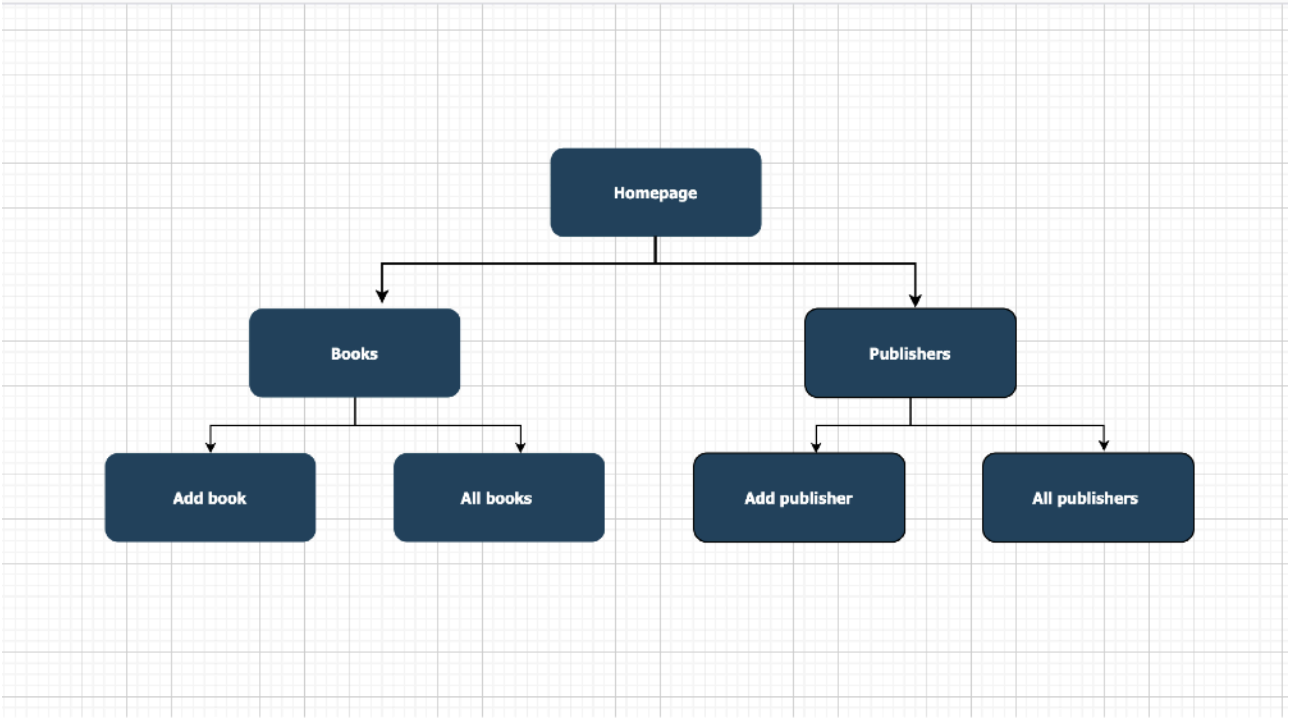
This covers things that might constrain the project and prevent it from reaching its full potential.

Constraint Category	Implementation Constraint	Solution
Hardware and Software platforms	The app is not currently mobile friendly, which may inconvenience managers who wish to check the status of books whilst they are on the shop floor.	The app should be made mobile friendly.
Performance requirements	If the app became popular, it would require larger servers and better memory as there are many bookshops who could benefit from its functionality. It is currently a very small app with basic functionality and thus would not be able to handle large volumes of information. It is also not accessible offline yet and this could affect its popularity as not all bookshops (and especially smaller bookshops) do not have wifi (or consistently good wifi)	The app should use a server with good memory and be accessible offline.
Persistent Storage and Transactions	The content is currently not securely stored, which could prove a problem if the app increases in popularity and or stores extra information (such as financial details for facilitating the buying and selling of books) that needs to be handled securely. The app also has no current ability to track previously deleted books or publishers which could prove a problem if a user wanted to check if a particular author or book had ever been stored - or was declared out of print, for example.	The app needs to implement secure storage of sensitive information, as well as functionality to show which books and publishers have been deleted from the system.

Usability	The app is straightforward to use due to its simple design, but is not visually engaging due to lack of time to spend on CSS. It also does not have any functionality other than CRUD. This means it currently doesn't have wide appeal as most booksellers and bookshop managers expect more functionality from their management software (such as a way to show book covers, the amount stored in different branch, ability to order in from a different branch)	More time would allow for an app with a better design and more functionality.
Budgets	The current front end design is poor and thus budgeting for a designer would be necessary. The lack of an appealing design seriously affects user experience and would necessitate a professional touch.	Hiring a front-end designer or further time for the current developers to improve their CSS skills would be necessary.
Time Limitations	Due to there being only a week to complete a full-stack app, the app does not have much functionality other than CRUD. The appearance is also significantly lacking in appeal. The basic functionality and design means the app currently lacks wide appeal for booksellers and bookshop managers who will have much higher standards for the software they use to organise their stock.	More time to complete the project would result in a better design, more functionality (e.g a calculate mark-up button for each book) and a better user experience.

Unit	Ref	Evidence
P	P.5	User Site Map
		Description:

Paste Screenshot here

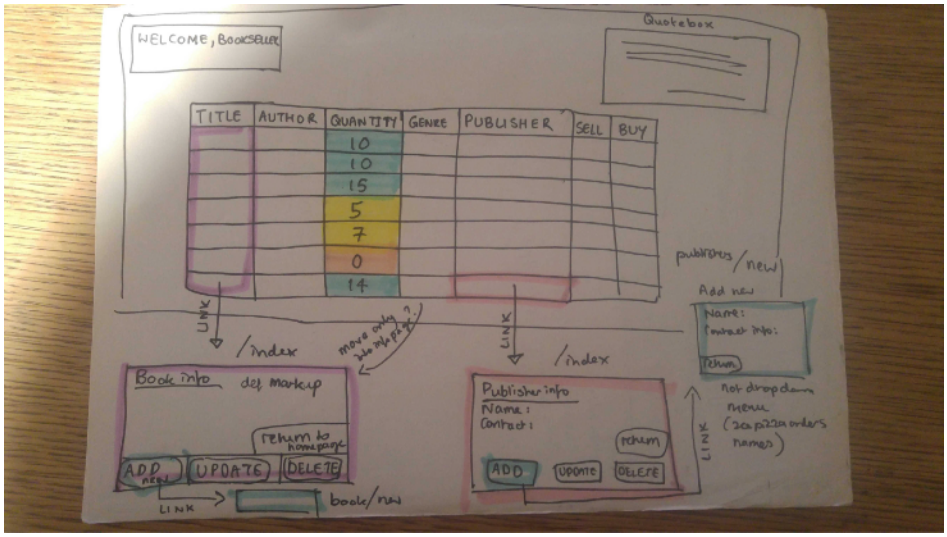


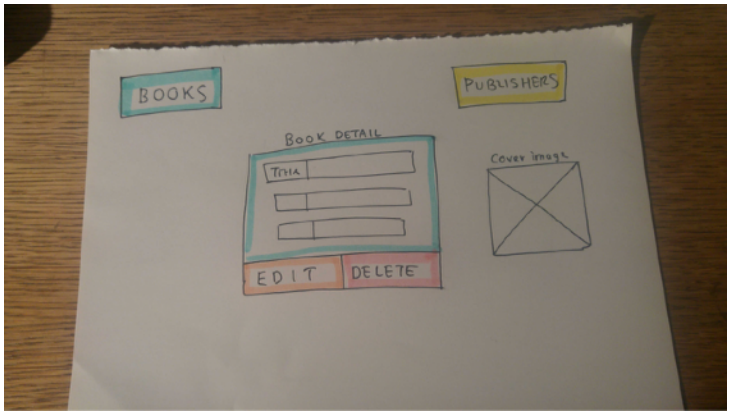
Description here

This is the user site map. It displays in a simple way the simple Ruby app for organising book stock. The current stock of books is stored under

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description:

Paste Screenshot here





Description here

These are two wireframe diagrams for my Ruby project, an app to manage a stock of books and the publishers the books were associated with. The first wireframe is of the homepage and what would be loaded if the cells in the table were clicked on; the second is the detail box of what would be displayed if a particular book was selected.

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

Paste Screenshot here

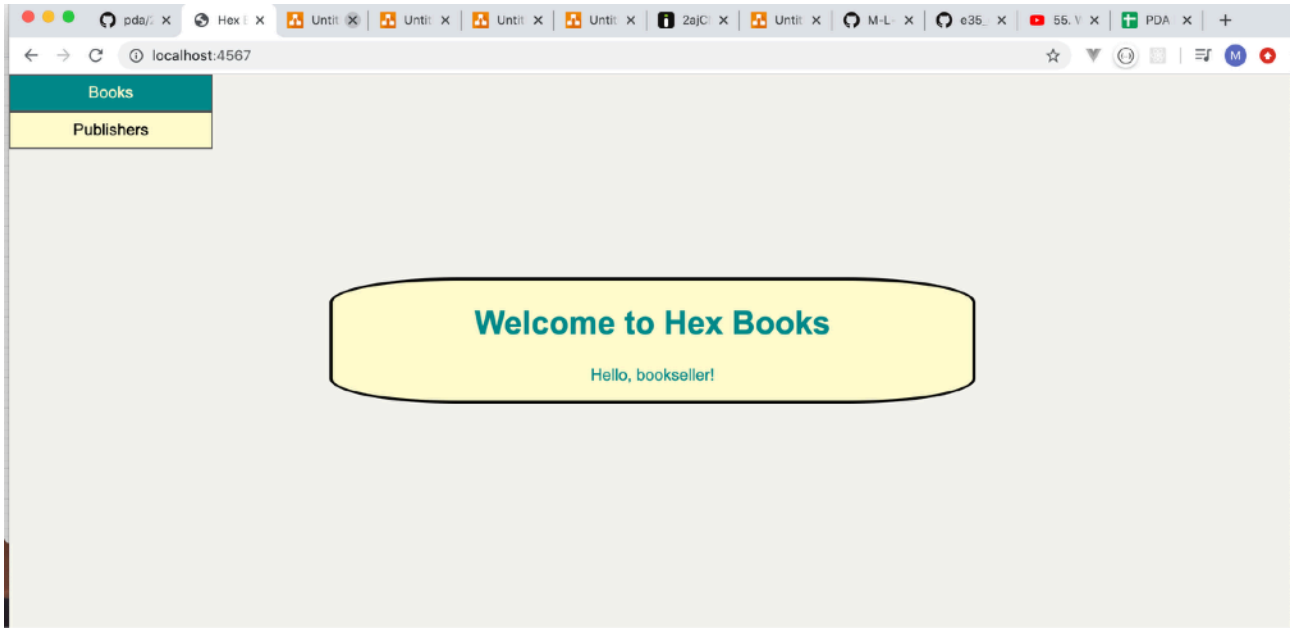
```
39
40      //bookPassenger must check check there are seats to book a passenger into and then push
41      //added passenger into array, it should not return anything
42      //1) Count available seats using availableSeats()
43      //2) if there are available seats, push whichever passenger has been passed as argument
44      //into the array of passengers and do not return anything
45
46      public void bookPassenger(Passenger passenger) {
47          if (availableSeats() > 0) {
48              passengers.add(passenger);
49          }
50      }
51
52      public int availableSeats() {
53          return this.plane.getSeats() - this.passengers.size();
54      }
55  }
56
```

Description here

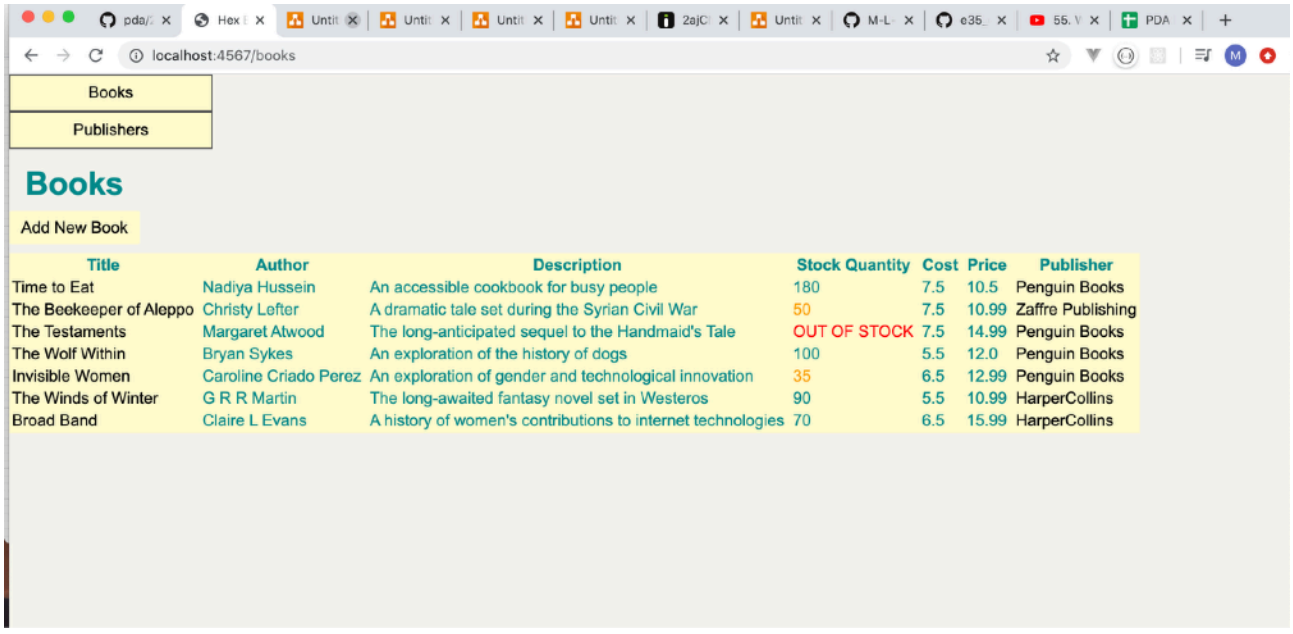
The method to book a passenger must only work if there are seats to put a passenger in, so the method first uses a separate method availableSeats( ) to check there is more than 0, and then pushes whichever passenger has been passed as an argument into the passengers array.

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:

Paste Screenshot here



User is at the homepage and inputs “books” (as seen by the CSS colour-change when the button for books is hovered over)

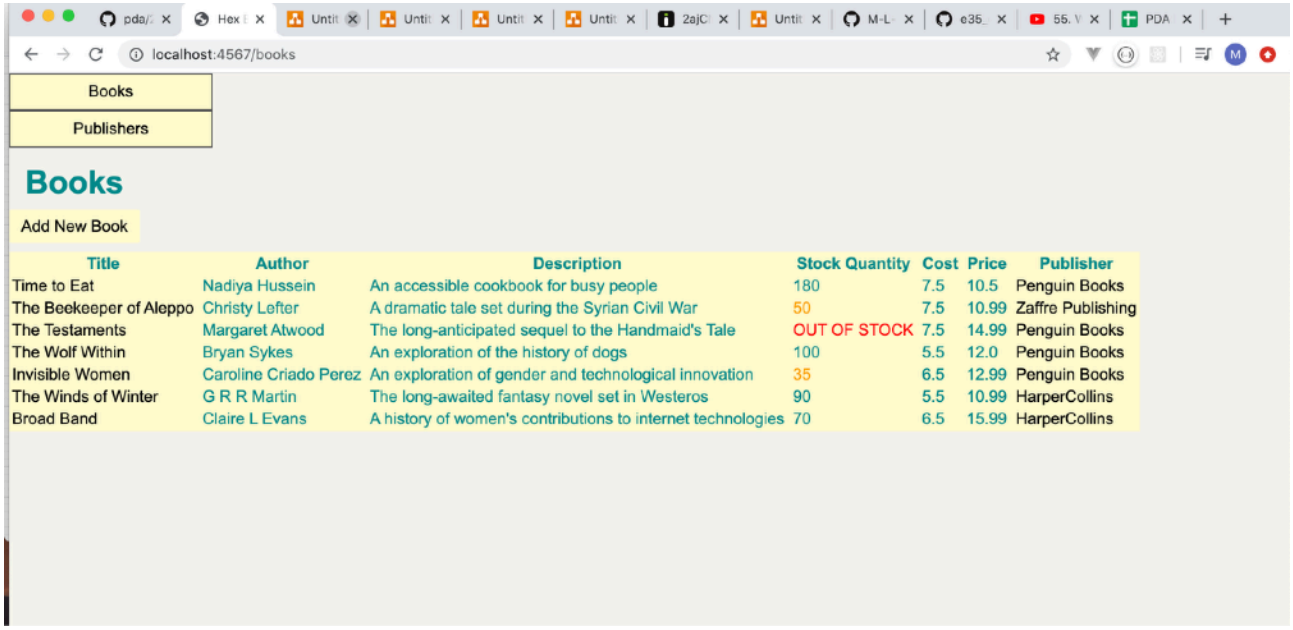


Title	Author	Description	Stock Quantity	Cost	Price	Publisher
Time to Eat	Nadiya Hussein	An accessible cookbook for busy people	180	7.5	10.5	Penguin Books
The Beekeeper of Aleppo	Christy Lefter	A dramatic tale set during the Syrian Civil War	50	7.5	10.99	Zaffre Publishing
The Testaments	Margaret Atwood	The long-anticipated sequel to the Handmaid's Tale	OUT OF STOCK	7.5	14.99	Penguin Books
The Wolf Within	Bryan Sykes	An exploration of the history of dogs	100	5.5	12.0	Penguin Books
Invisible Women	Caroline Criado Perez	An exploration of gender and technological innovation	35	6.5	12.99	Penguin Books
The Winds of Winter	G R R Martin	The long-awaited fantasy novel set in Westeros	90	5.5	10.99	HarperCollins
Broad Band	Claire L Evans	A history of women's contributions to internet technologies	70	6.5	15.99	HarperCollins

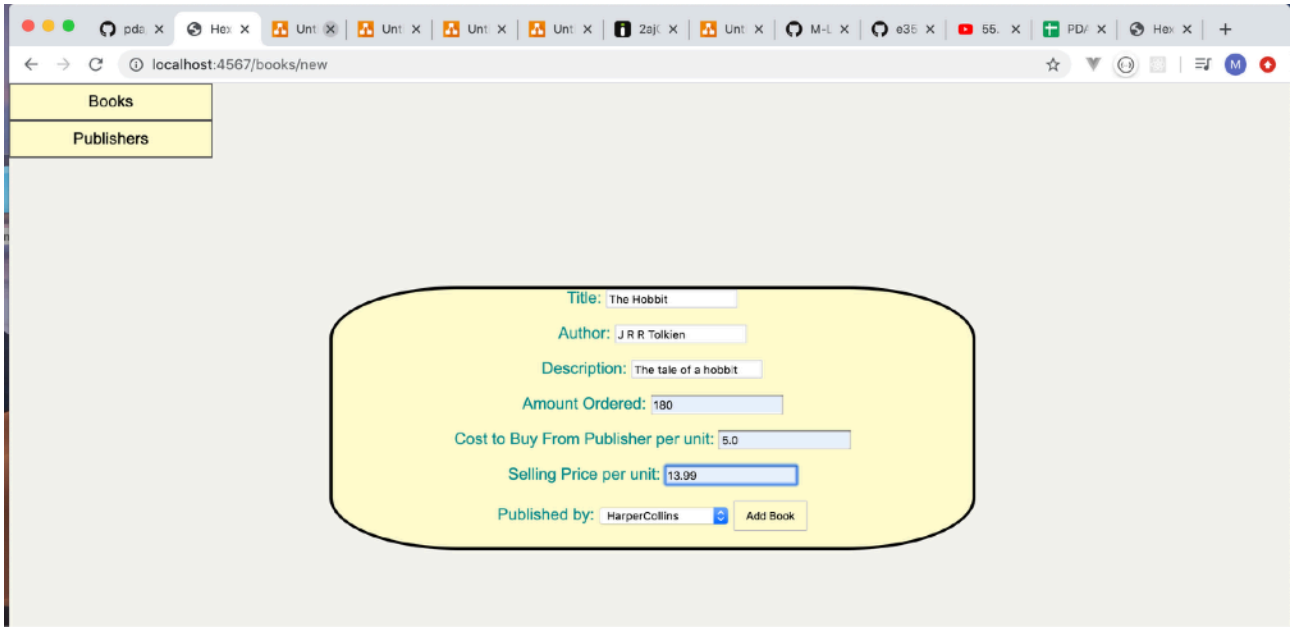
User’s input of clicking on books is processed and brings up the /books page which displays all books.

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:

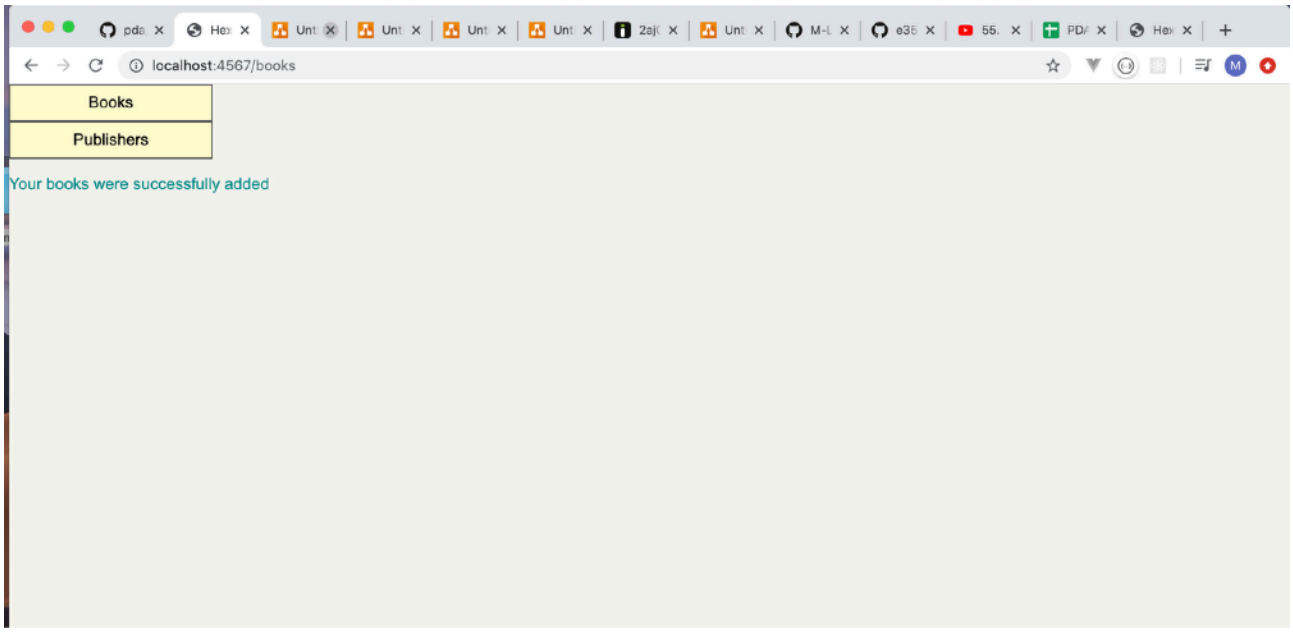
Paste Screenshot here



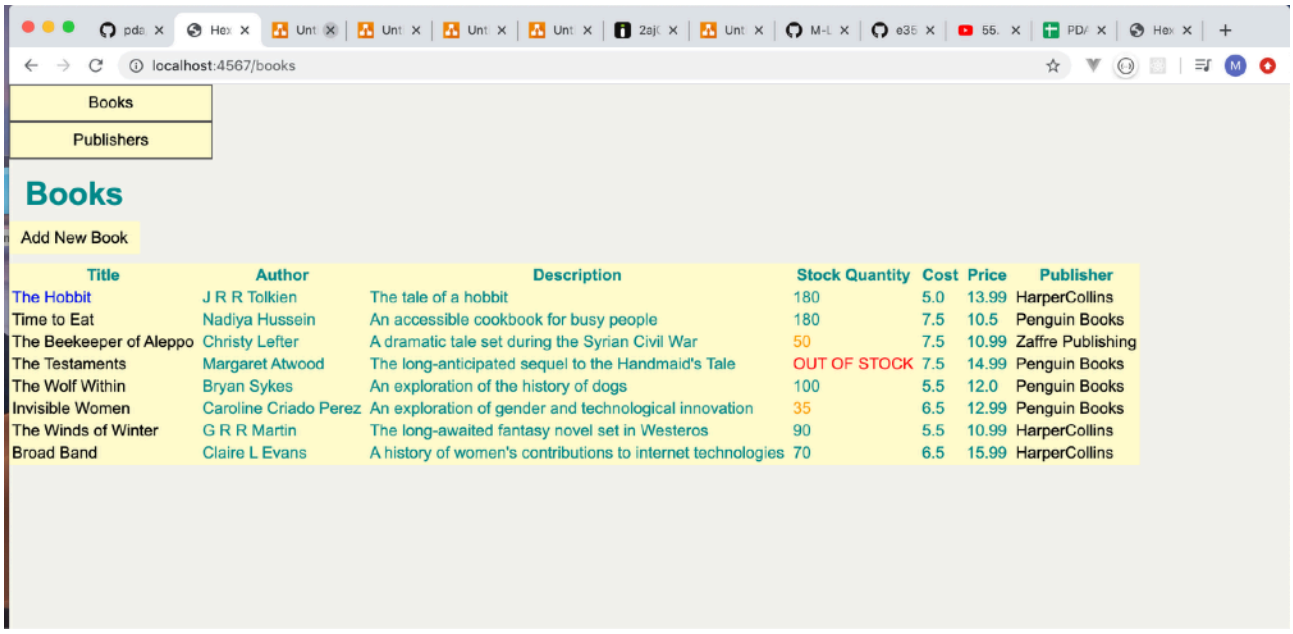
The user wishes to add a new book to the store.



The user fills in the fields required to add the new book and selects ‘add book.’



The user receives confirmation that the book was successfully added.



The new book - “The Hobbit” - is now in the /books page.

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:

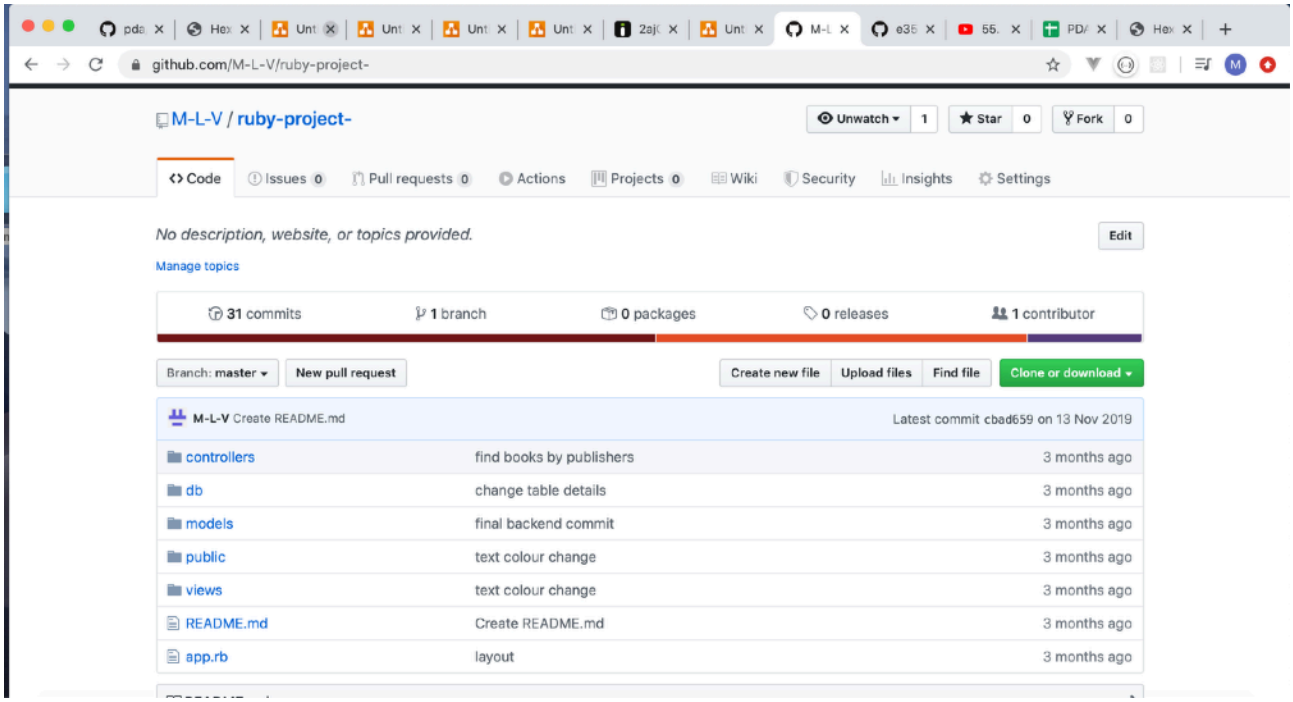
Paste Screenshot here

The user selects a book and selects “delete book” button.

User navigates back to /books - selected book “The Winds of Winter” is no longer there.

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

Paste Screenshot here

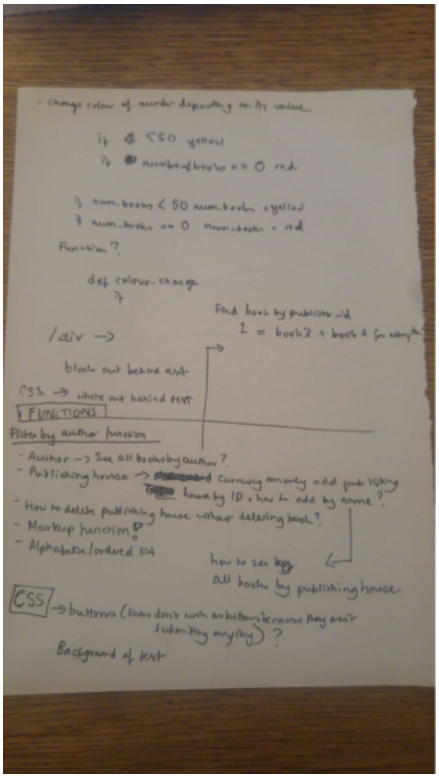


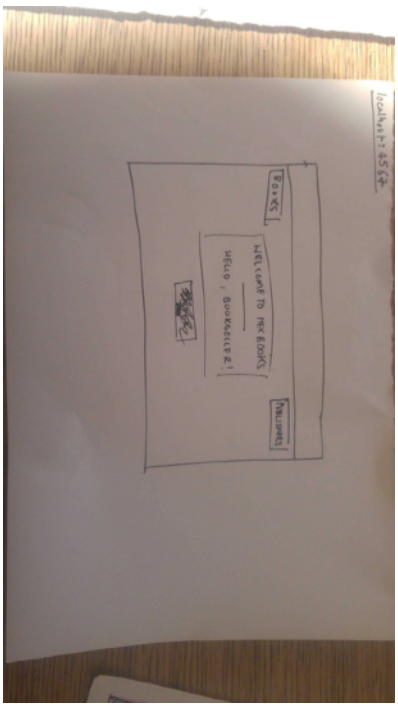
Description here

This is my Ruby project, which can be found at https://github.com/M-L-V/ruby-project-

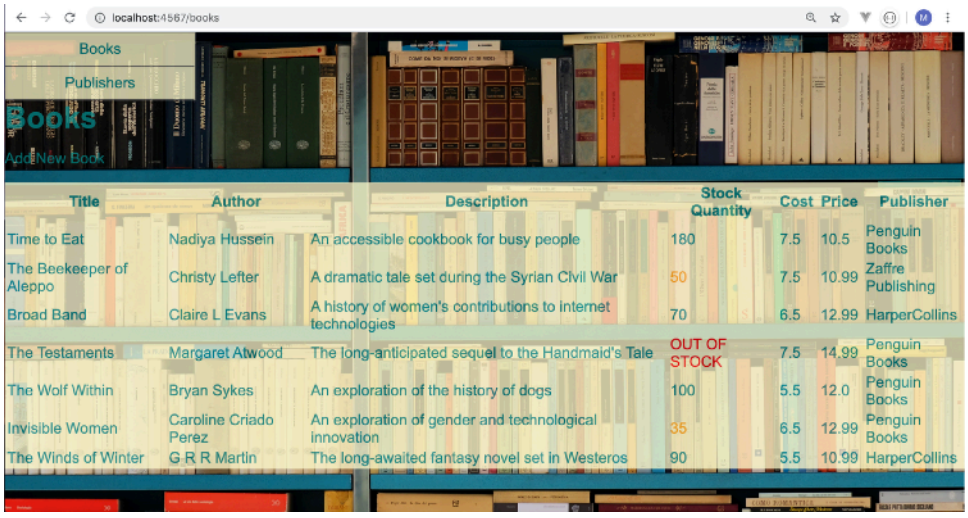
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description:

Paste Screenshot here





This shows some very early planning of functionality and CSS colour changes depending on value of the number of books in stock, as well as what the homepage would potentially look like.

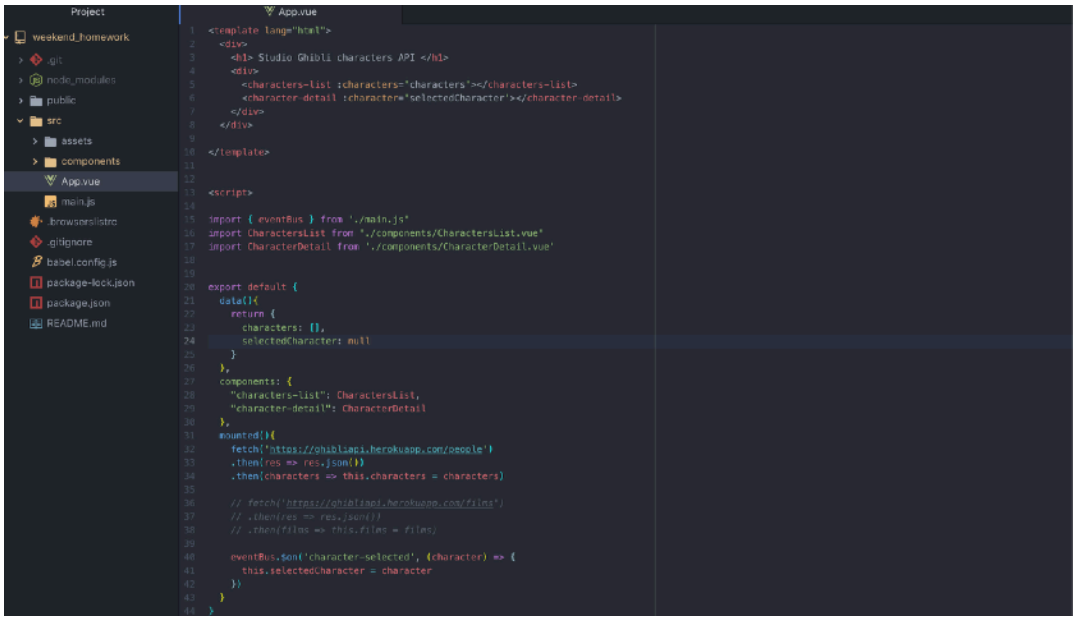


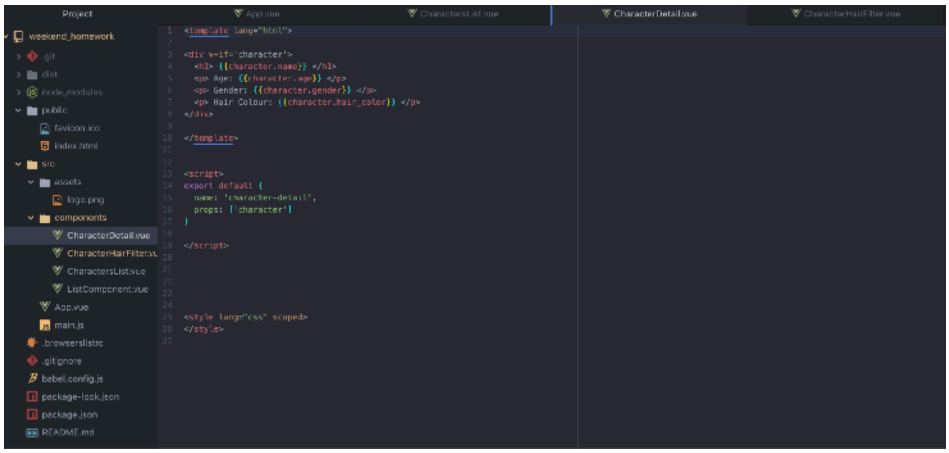
This screenshot shows some early CSS before it was changed due to being too garish and not giving a good user experience.

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:

Paste Screenshot here





Description here

This app utilises a Studio Ghibli character API to display information about characters. The code fetches the data from the API and the character detail component renders the name, age, gender, and hair colour in the browser.

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:

Paste Screenshot here

```
55      @Test
56      public void canSellProduct(){
57          musicShop.addToStock(musicBook);
58          musicShop.sellProduct(musicBook);
59          assertEquals( expected: 1010.00, musicShop.getTill(),  delta: 0.01);
60          assertEquals( expected: 0, musicShop.getStock().size());
61      }
62
63  }
```

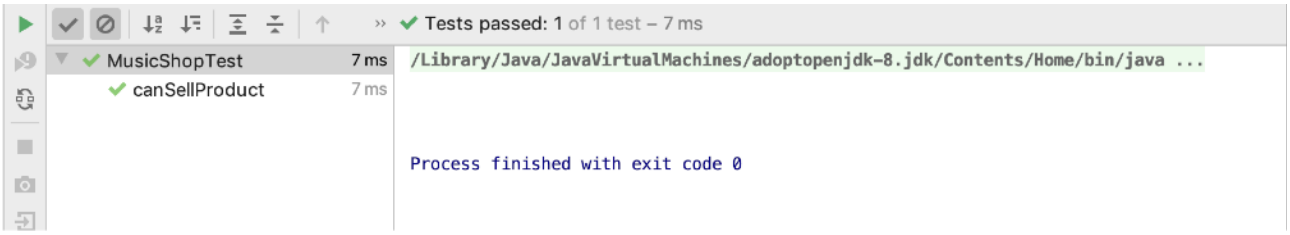
This test code is to check that a music shop's sellProduct( ) method works. It should sell a product and put the profit in the till.

The test has failed, saying there is 1 where 0 was expected for the length of the stock array in the music shop.

```
31
32
33
34 @    public void sellProduct(ISell product){
35          double profit = product.markUp();
36          till += profit;
37      }
38  }
39
40
```

```
31
32
33
34 @    public void sellProduct(ISell product){
35          double profit = product.markUp();
36          till += profit;
37          removeFromStock(product);
38      }
39  }
40
```

Method is corrected from not including the product being removed from stock to including the function removeFromStock( ) that takes the sold product out of the array of stock items.

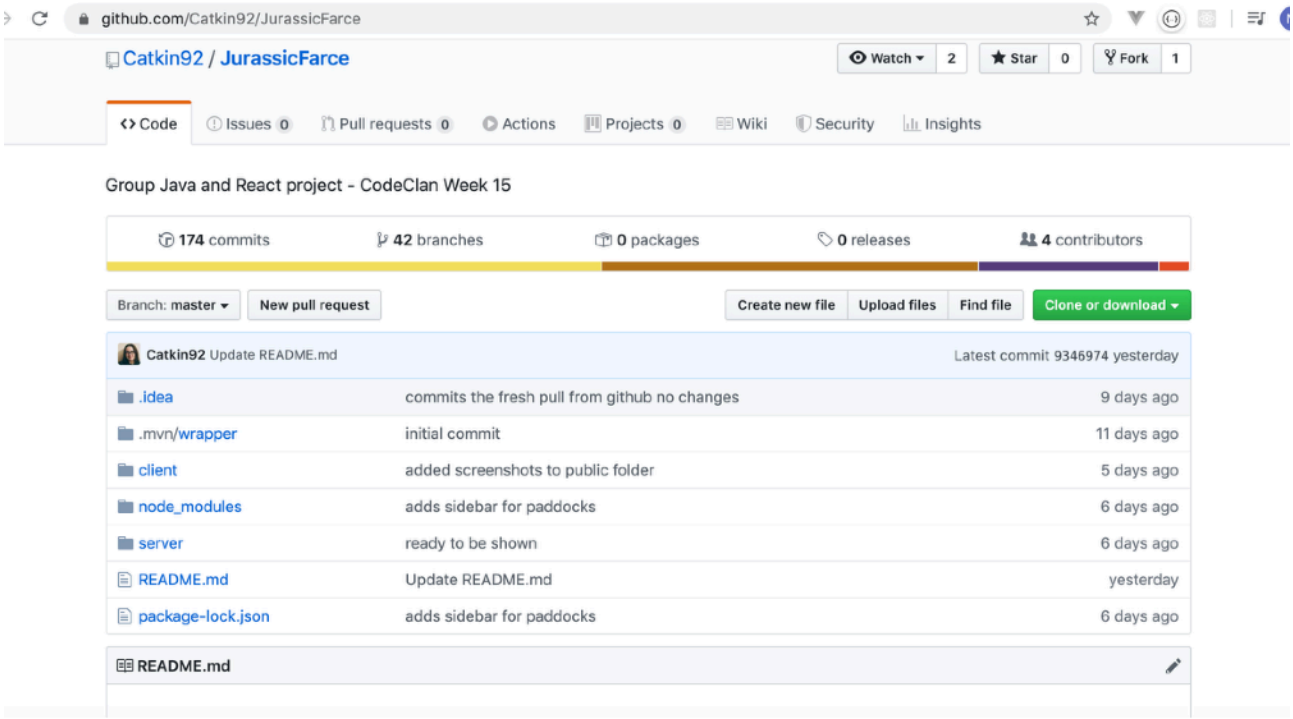


Test now passes.

Week 9

Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.
		Description:

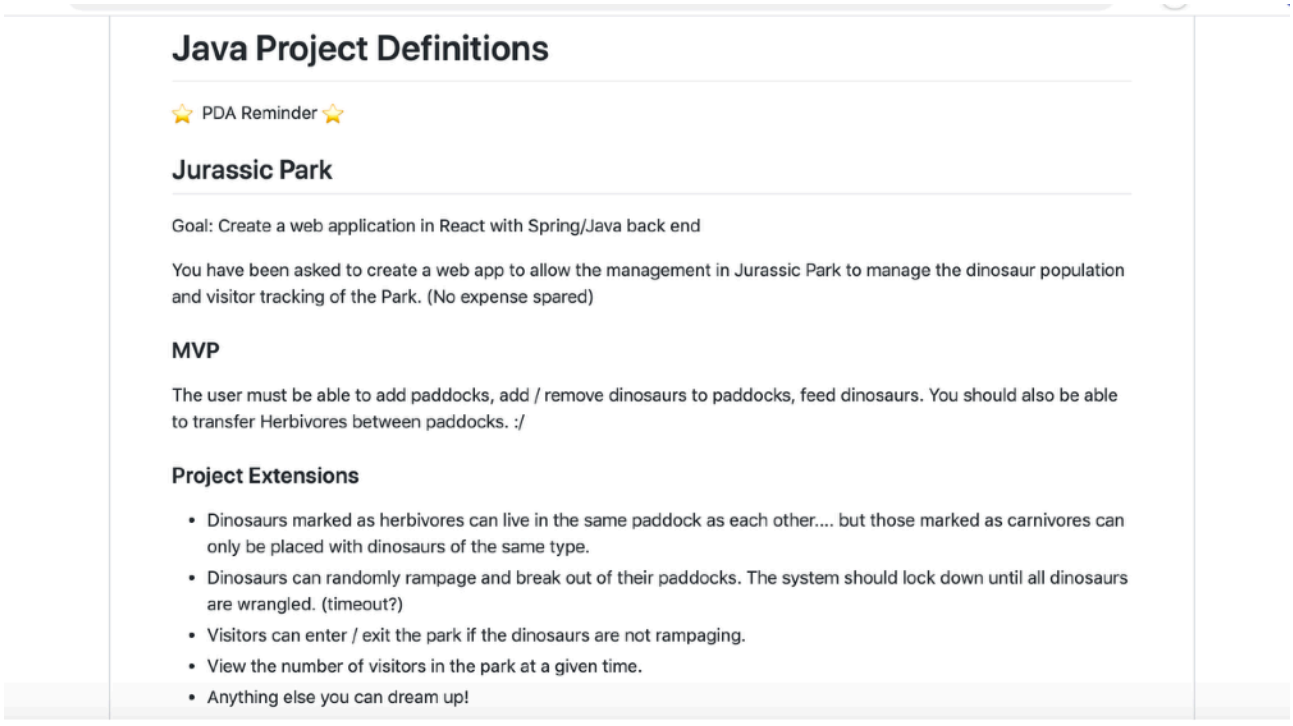
Paste Screenshot here



This is the contributors' page on GitHub where my team worked for the Java and React final project with 4 contributors, 42 branches, and 174 commits.

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

Paste Screenshot here



Java Project Definitions

⭐ PDA Reminder ⭐

Jurassic Park

Goal: Create a web application in React with Spring/Java back end

You have been asked to create a web app to allow the management in Jurassic Park to manage the dinosaur population and visitor tracking of the Park. (No expense spared)

MVP

The user must be able to add paddocks, add / remove dinosaurs to paddocks, feed dinosaurs. You should also be able to transfer Herbivores between paddocks. :/

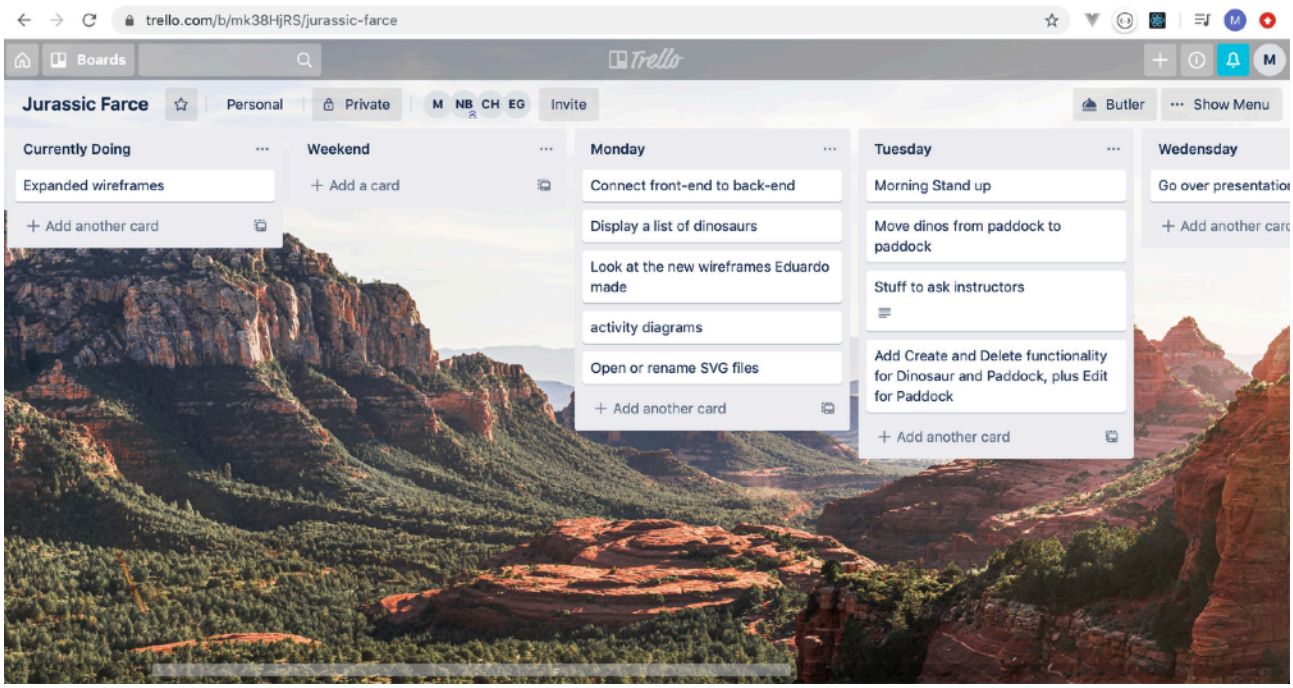
Project Extensions

- Dinosaurs marked as herbivores can live in the same paddock as each other.... but those marked as carnivores can only be placed with dinosaurs of the same type.
- Dinosaurs can randomly rampage and break out of their paddocks. The system should lock down until all dinosaurs are wrangled. (timeout?)
- Visitors can enter / exit the park if the dinosaurs are not rampaging.
- View the number of visitors in the park at a given time.
- Anything else you can dream up!

This is the project brief my team and I chose for the Java/React group project.

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

Paste Screenshot here



Description here

This is the Trello board used for the Java/React group project.

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

Paste Screenshot here

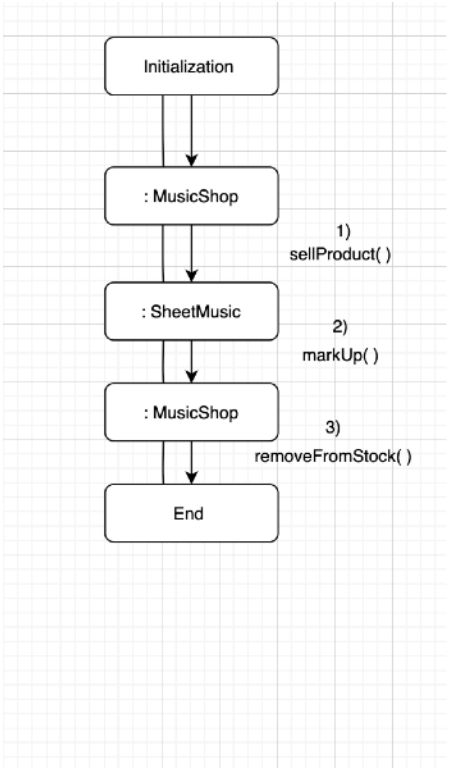
Acceptance Criteria	Expected Result	Pass/Fail
A user is able to view all dinosaurs	The page should load all dinosaurs currently saved if user selects “dinosaurs” from the top of the page	Pass
A user is able to add a dinosaur and specify its gender, name, and species	The page should give the option to select a species, and gender, and allow the user to fill in the name	Pass
A user is able to feed a dinosaur and see hunger decrease in a visual way	The feed button should reduce the hunger level and colour of bar should change from red (high hunger) to yellow(medium) to green (not hungry)	Pass
A user is able to check dinosaurs present in a paddock by selecting the paddock.	A paddock should display dinosaurs when selected by id.	Fail - the paddock does not display the dinosaurs present in it.

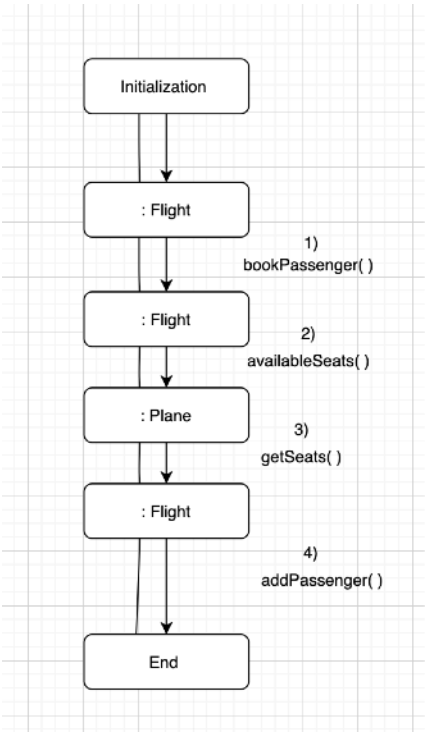
Description here

This is an acceptance criteria and test plan for the Java/React group project.

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

Paste Screenshot here





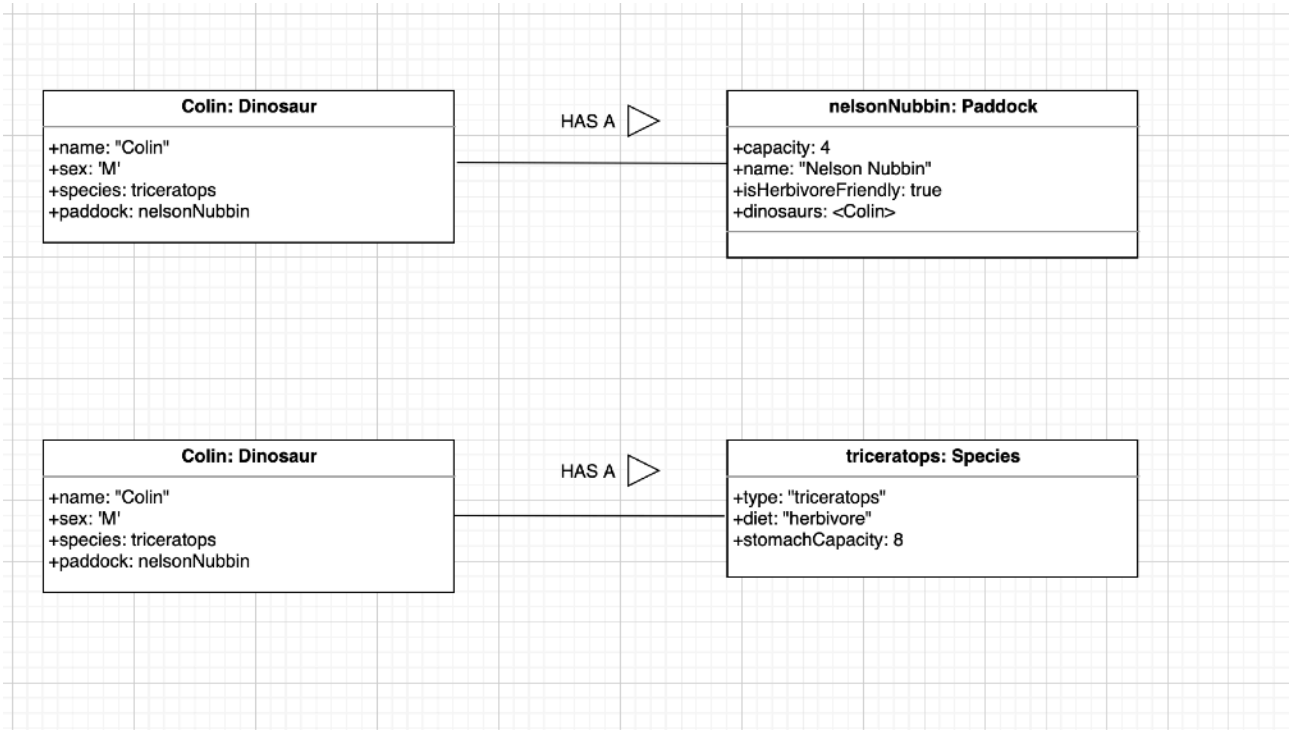
Description here

The first collaboration diagram demonstrates the selling of a product in a music shop. To sell a product, the music shop needs the mark-up on the product (here a sheet music object) - the product object uses its markUp method (implemented via an interface) to calculate how much profit it is giving, and then passes this back to the musicShop, which then removes the product from its stock and the sequence ends.

The second collaboration diagram is the process to book a passenger on a flight. The flight requires the available seats on it, so it passes this request to the Plane class, which counts the seats it has and passes this back so the Flight can subtract the number of current passengers from the number of seats it now knows it has from the Plane's information. If there are seats left, the Flight class books a Passenger.

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

Paste Screenshot here



Description here

Colin is a dinosaur object of the Dinosaur class and this object has a paddock of the Paddock class called Nelson Nubbin. Nelson Nubbin has an arrayList of dinosaur of the Dinosaur class of which Colin is a member.

Colin also has a Species property triceratops which is of the species class. Triceratops as an object of the species class details the the type, diet and stomach capacity of any dinosaur object with this particular species as a property.

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

Paste Screenshot here

Bug/Error	Solution	Date
Database was not loading at all and giving Spring error message "Failed to configure a data source"	"ArrayList" was changed to "List"	2/02/2020
Test for getSex() was failing Hard-coding in the sex as F in the constructor was causing a failing test when getSex() was called on a dinosaur with M	The sex property was changed from F to being defined when instantiated in the backend	3/02/2020
Dinosaur was initially declared as an abstract class and would not be instantiated	Dinosaur was changed to a non-abstract class	3/02/2020
Paddock could not be passed to a dinosaur in the database	Paddock was being passed as a whole object when it should have been passed as a localhost:8080 href link	4/02/2020
Data could not set state with 2 examples of componentdidmount in the same file	The two fetch requests were saved under variable names fetchPaddock and fetchSpecies and placed in a single componentdidmount	4/02/2020

Description here

This is the bug tracking report detailing some bugs found in the Java-React group project.

Week 12

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

Paste Screenshot here

MusicBookTest.java | Guitar.java | Piano.java | IPlay.java | SheetMusic.java | MusicShopTest.java

```
1  package behaviour;
2
3  public interface ISell {
4
5      double markUp();
6  }
7
```

MusicBookTest.java | Guitar.java | Piano.java | IPlay.java | SheetMusic.java | MusicShopTest.java

```
3  import behaviour.ISell;
4
5  public class MusicBook implements ISell {
6
7      private String author;
8      private String title;
9      private double buyingPrice;
10     private double sellingPrice;
11
12     public MusicBook(String author, String title, double buyingPrice, double sellingPrice) {
13         this.author = author;
14         this.title = title;
15         this.buyingPrice = buyingPrice;
16         this.sellingPrice = sellingPrice;
17     }
18
19     public String getAuthor() {
20         return author;
21     }
22
23     public String getTitle() {
24         return title;
25     }
26
27     public double getBuyingPrice() {
28         return buyingPrice;
29     }
30
31     public double getSellingPrice() {
32         return sellingPrice;
33     }
34
35     public double markUp(){
36         return sellingPrice -= buyingPrice;
37     }
```

ar.java | Piano.java | IPlay.java | MusicShop.java | SheetMusic.java | MusicShopTest.java | MusicBook

```
1  import behaviour.ISell;
2
3  import java.util.ArrayList;
4
5  public class MusicShop {
6
7      private ArrayList<ISell> stock;
8      private double till;
9
10
11     public MusicShop(double till) {
12         this.stock = new ArrayList<ISell>();
13         this.till = till;
14     }
15
16     public ArrayList<ISell> getStock() { return stock; }
19
20     public double getTill() { return till; }
23
24     public void addToStock(ISell product) { stock.add(product); }
27
28     public void removeFromStock(ISell product) { stock.remove(product); }
31
32
33
34     public void sellProduct(ISell product){
35         double profit = product.markUp();
36         till += profit;
37         removeFromStock(product);
38     }
39 }
40
41
42
43
```

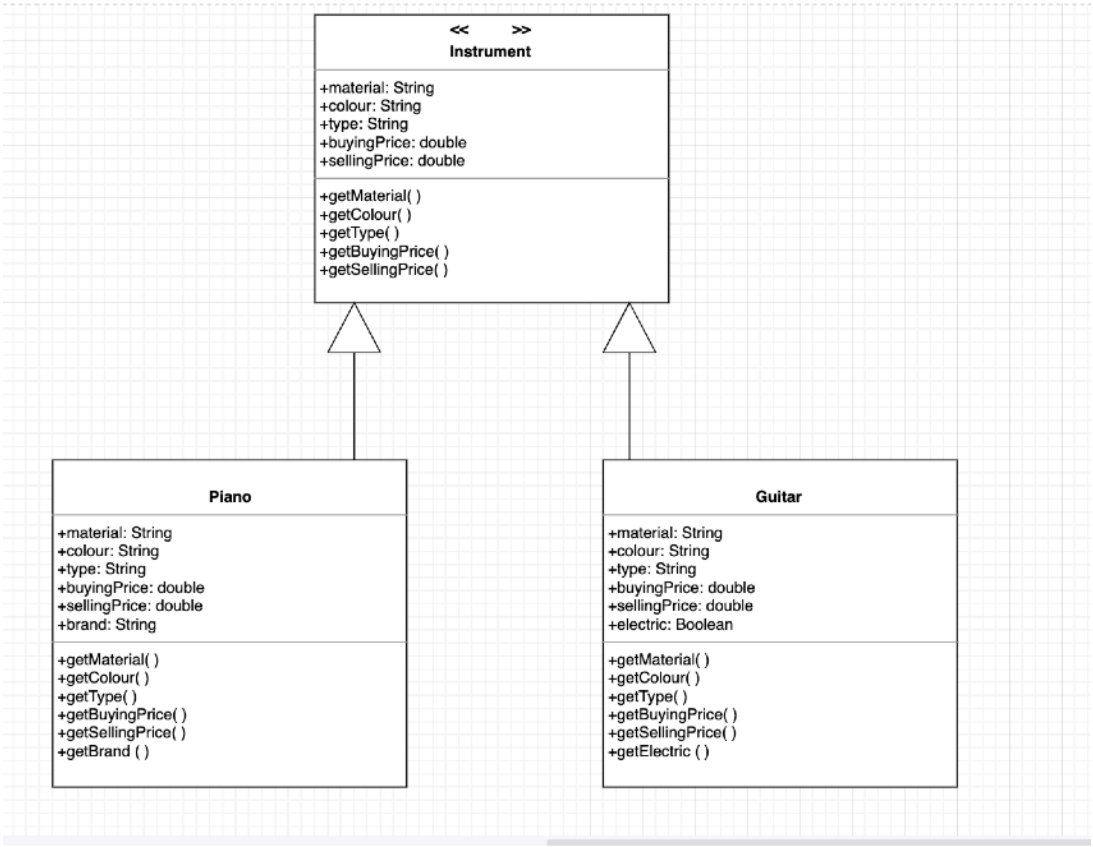
Description here

This is an interface called ISell that promises a markUp method. MusicBook is a class that implements this interface with a markup method that returns its sellingPrice int minus its buyingPrice to show the markup on individual music books when sold.

MusicShop is where the items are sold - it has an array list of all items that implement ISell and thus have a markup method, but can be treated as both themselves AND an ISell object. Thus the addtoStock and sell methods that belong to the musicShop class can be implemented on any class of item that uses iSell, such as musicBook.

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

Paste Screenshot here



Description here

Instrument is an abstract class that has properties of material, colour, type, buyingPrice, and sellingPrice, as well as getter methods for all of these properties. Piano and Guitar inherit all these properties and getter methods as they extend the abstract Instrument class, but also have one unique quality each - electric for Guitar and brand for Piano.

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

Paste Screenshot here

```
public class Passenger {

    private String name;
    private int numOfBags;

    public Passenger(String name, int numOfBags){
        this.name = name;
        this.numOfBags = numOfBags;
    }

    public String getName() {
        return name;
    }

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

    public int getNumOfBags() {
        return numOfBags;
    }

    public void setNumOfBags(int numOfBags) {
        this.numOfBags = numOfBags;
    }
}
```

Description here

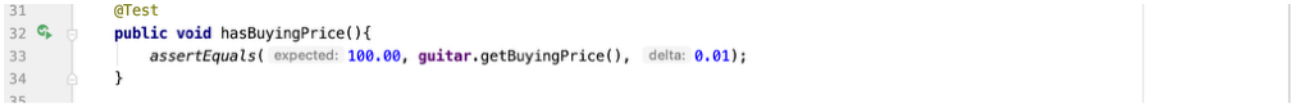
The variables of name and numOfBags of the Passenger class are declared as private initially but public getter and setter methods are provided to view and modify the variables. The variables of name and numofBags can therefore only be accessed through the methods of the Passenger class and are thus hidden from other classes.

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:

Paste Screenshot here

Piano.java | IPlay.java | MusicShop.java | SheetMusic.java | MusicShopTest.java | MusicBook.java

```java
package instruments;


public abstract class Instrument {

    private String material;
    private String colour;
    private String type;
    protected double buyingPrice;
    protected double sellingPrice;

    public Instrument(String material, String colour, String type, double buyingPrice, double selling
        this.material = material;
        this.colour = colour;
        this.type = type;
        this.buyingPrice = buyingPrice;
        this.sellingPrice = sellingPrice;
    }

    public String getMaterial() { return material; }

    public String getColour() { return colour; }

    public String getType() { return type; }

    public double getBuyingPrice() {
        return buyingPrice;
    }

    public double getSellingPrice() {
        return sellingPrice;
    }
```



Description here

The class guitar inherits from the class Instrument, but also has a Boolean of if it is electric or not when it is instatiated as a guitar object. Guitar uses the method “hasBuyingPrice” inherited from the Instrument class to show its buying price.

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.
		Description:

Paste Screenshot here

```
<td></td>
<td><%= book.description %></td>
<td></td>

<% if book.stock_quantity > 50 %>
<td><%= book.stock_quantity %></td>
<% elsif book.stock_quantity == 0 %>
<td><font color="#ff0000">OUT OF STOCK</font></td>
<% else %>
<td><font color="#ffa500"><%= book.stock_quantity %></font></td>
<% end %>

<td></td>
```

```
06
07    def self.markup()
08      profit = @selling_price.to_f - @buying_cost.to_f
09      markup_amount = (profit/@buying_cost.to_f) * 100
10      return markup_amount.round(2)
11    end
12
13
14
15
16
17
18
```

Description here

The first algorithm is one written for the Ruby bookseller's app - it is an if statement that changes the colour of the stock quantity or message displayed in the table of books depending on how many books there are. If there are above 50, the colour of the number remains blue, if there are none, a message of "OUT OF STOCK" is displayed in red, and if there are less than 50, the stock quantity is displayed in yellow to warn of low stock. This colour changing algorithm is necessary for a user to see at a glance if they are running out of an item.

The second algorithm is a simple mark-up function that calculates the mark-up on an individual book. This would be a necessary bit of functionality for an app that a bookseller would use to keep track of stock and the money they were making.