**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.

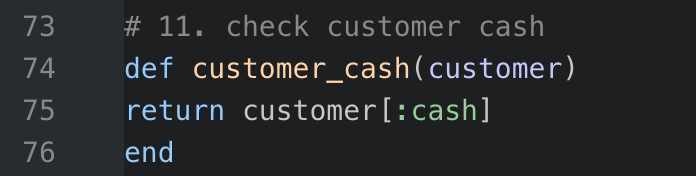
**Week 1**

| 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 | |

**Paste Screenshot here**

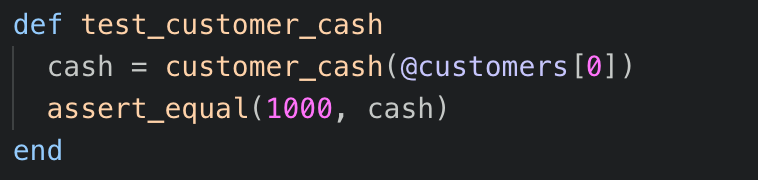
**Description here**

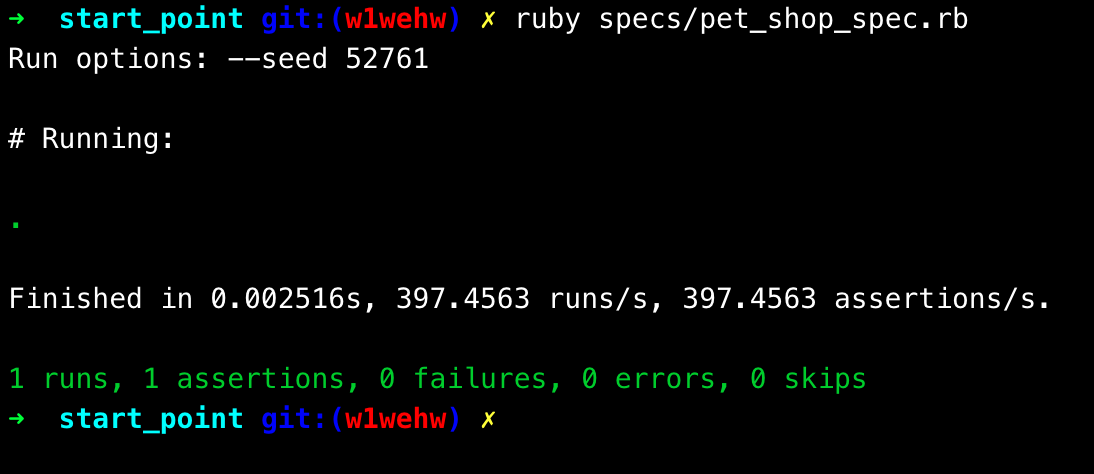
This screenshot shows the @customers hash being created in order to initialize the test spec.

**Paste Screenshot here**

**Description here**

The hash **@customer** being created in the previous screenshot is being passed into the function here with an index. This function is taking the value of **cash** for the customer specified in the index and returning it.

**Paste Screenshot here**

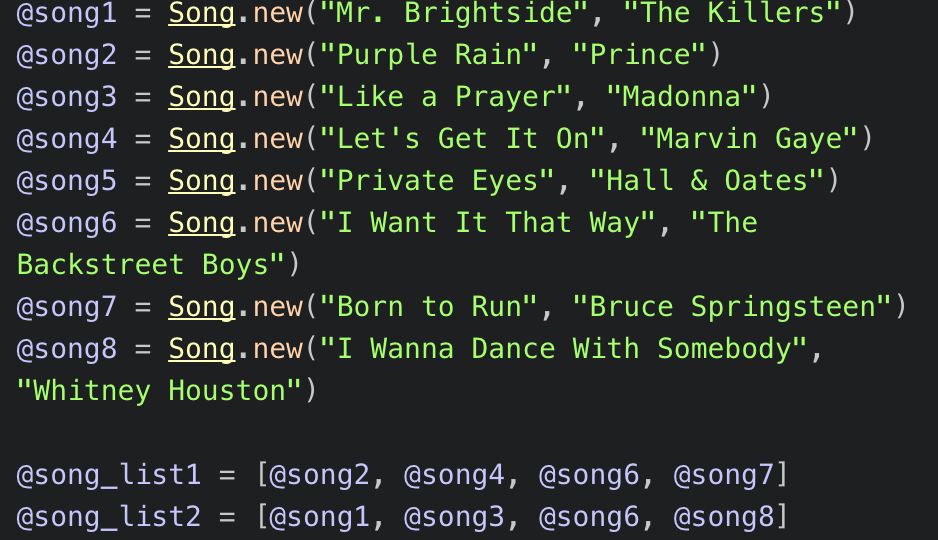
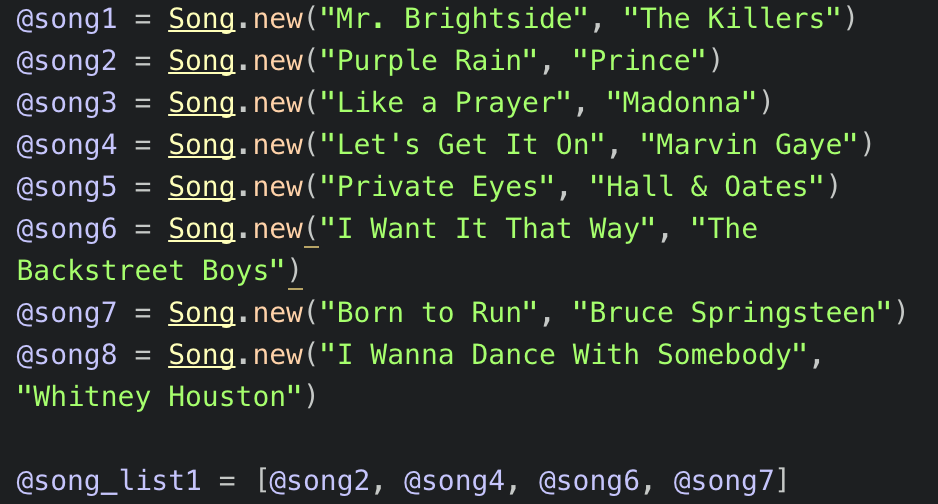
****

**Description here**

The above images show the function in which the hash is used, and the result when run. The function tests that the value of **cash** for the first customer (index 0) is correct (1000).

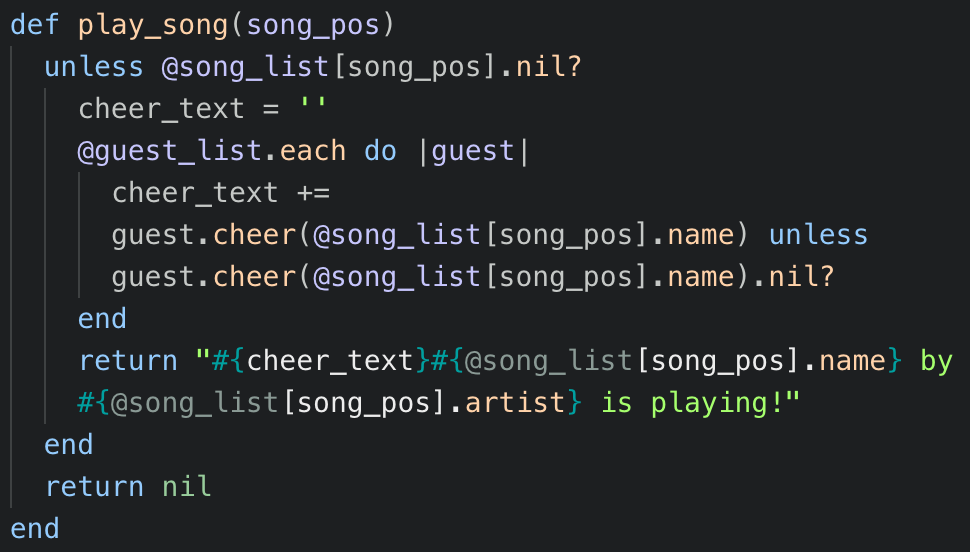
**Week 2**

| 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 | |

**Paste Screenshot here**

**Description here**

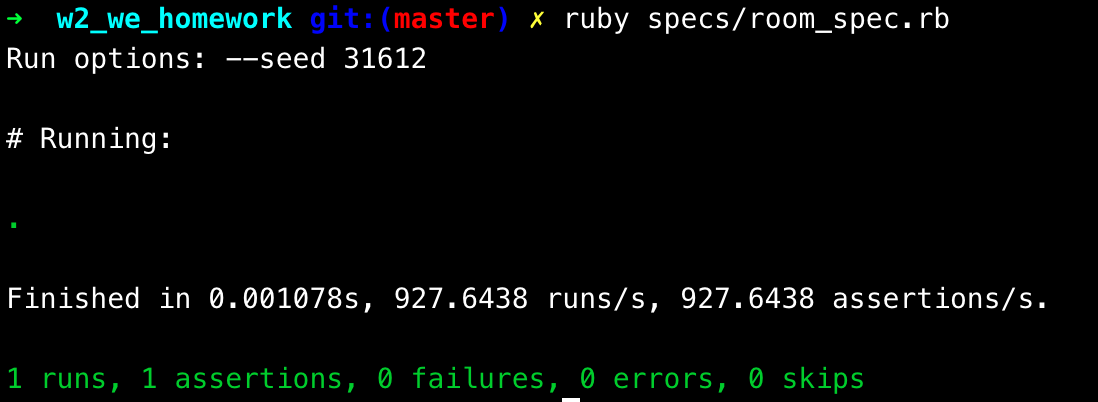
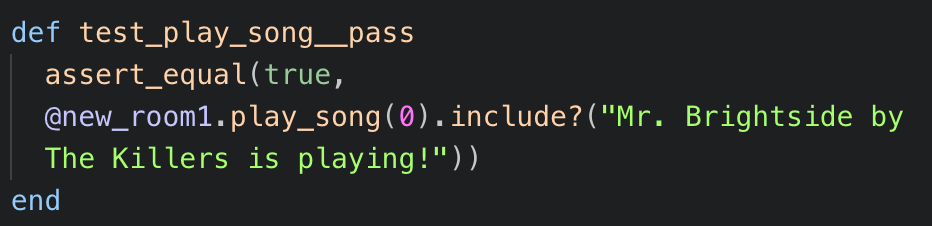
This screenshot shows 2 arrays being created out of a selection of 8 song objects created above. I will use **@song\_list2** in the example below.

**Paste Screenshot here**

**Description here**

This screenshot shows that the barrage @song\_list2 is pulled in as a property of @new\_room1. The function takes a song position as an integer and checks the array (@song\_list2) for the song at the position. It then displays the text, ***“<song name> by <song artist> is playing!”***. It then also loops through each guest in the room and compares the song’s title to the guest’s favourite song. If they match, an additional cheer\_text string (“Wooo!!”) Is added to the beginning of the return.

**Paste Screenshot here**

****

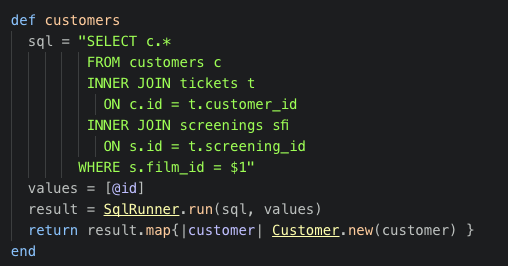
**Description here**

This screenshot shows the test comparing the output of the function to the expected string, and also shows the test passing.

**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 | |

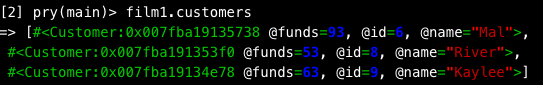
**Paste Screenshot here**

****

**Description here**

This is a method on the Film object that returns an array of Customer objects associated with the Film. It does this by querying the database and joining the Tickets and Screenings tables to the Customer table in order to bring back the relevant ID’s, and then maps these into new objects.

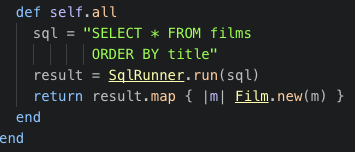
**Paste Screenshot here**

****

**Description here**

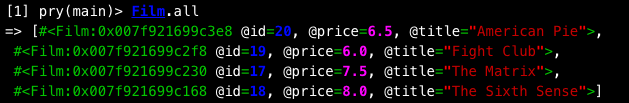
This shows the results of the method ‘customers’ above. It is an array of Customer objects created from the results of the SQL that ran against the database.

| 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 | |

**Paste Screenshot here**

**Description here**

This function brings back all the fields from the ‘films’ table in the database and sorts them by the ‘title’ field. It then maps the values into new ‘film’ objects.

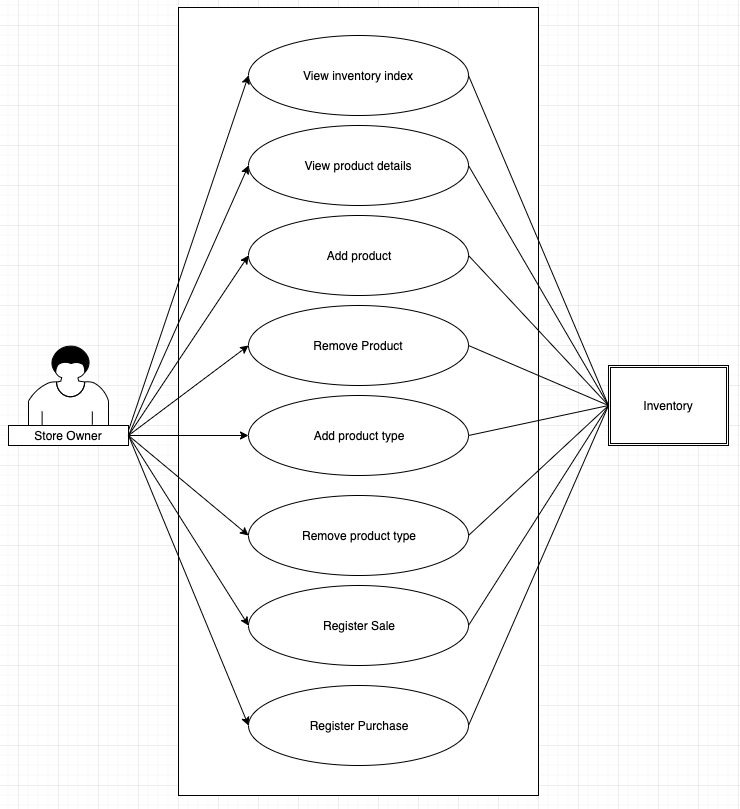
**Paste Screenshot here**

**Description here**

The results of the function shows that the objects have been returned into an array in alphabetical order by Title.

**Week 4**

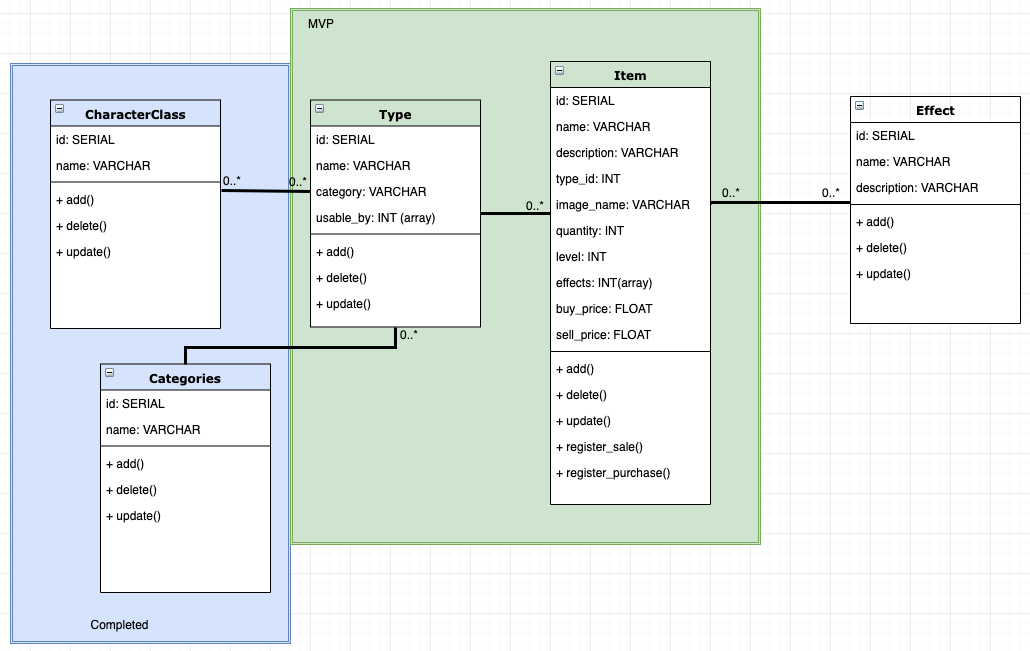
| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| A&D | A.D.1 | A Use Case Diagram | |

**Paste Screenshot here**

**Description here**

This shows the use case for a shop inventory system that I developed for my project. As a basic inventory management system, the only user would be the store owner, who would need to be able to view the items in stock, drill down to the product details, add and remove products, add and remove product types, and register sales and purchases.

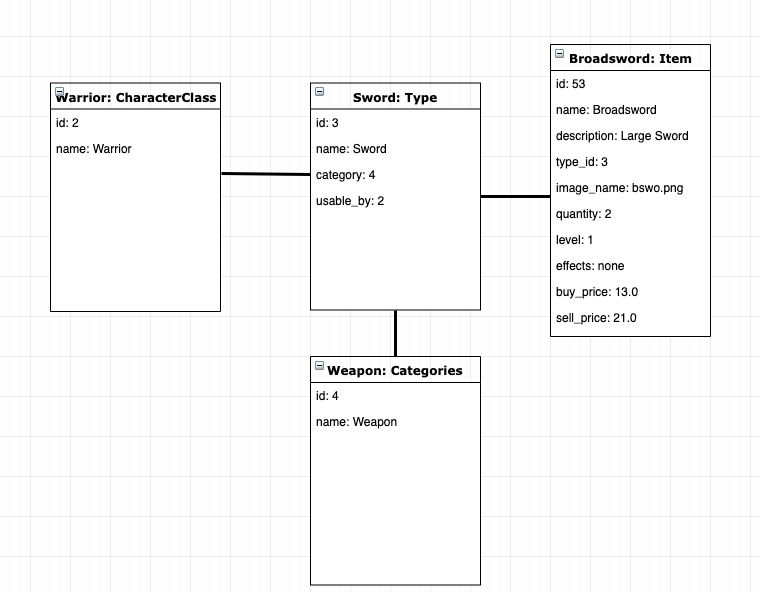
| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| A&D | A.D.2 | A Class Diagram | |

**Paste Screenshot here**

**Description here**

This is the class diagram for my project. The two key classes for the mvp were the Item and Type tables.As an extension, I was also able to add CharacterClass and Categories. As a further extension I had planned to add an Effect class, but did not have time in the end. CharacterClass is a many to many relation to Type, which also required the relationships to be built separately.

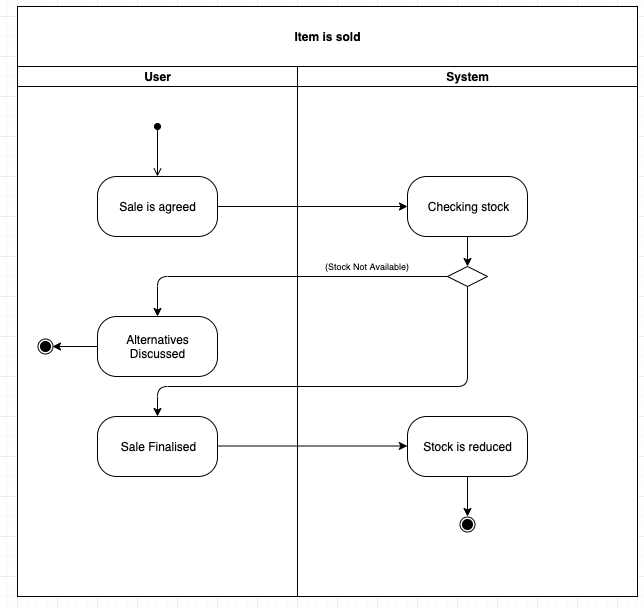
| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| A&D | A.D.3 | An Object Diagram | |

**Paste Screenshot here**

**Description here**

This object diagram shows on the right hand side a Broadsword item, which is linked to the Sword type. Sword types are assigned to the category of Weapon. Sword types are usable by the Warrior Character Class.

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

**Paste Screenshot here**

**Description here**

The activity diagram above shows the process for selling at item using the stock inventory system. If a sale is agreed, stock must be checked using the app. If stock is available, the sale can be finalised, and the stock level reduced in the app.

| 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 | |

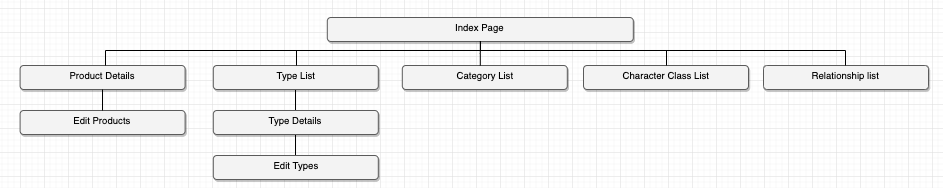
**Paste Screenshot here**

|  |  |  |
| --- | --- | --- |
| **Constraint Category** | **Implementation Constraint** | **Solution** |
| Time Limitations | There may not be enough time to complete all the features. This could mean there is no working application by the deadline. | Restrict the features to be built before the deadline to the minimum viable product. The remaining features can then be built as extensions to ensure there is always a workable product. |
| Budget | There is no budget for Art assets. This is an issue because the application is very abstract, and images would help to sell the concept. | Use creative commons licensed assets. This will allow images to be used for free as long as they are credited, and if budget becomes available down the line they can be swapped out in a patch. |
| Hardware / Software Platform | The user must be connected to the internet to interface with the system. This is an issue in areas of low network reliability, and some consumers may want to use the product offline. | The user must accept the constraint in the short term, however an extension could be planned to move the data to local storage if there is demand. |
| Persistent Storage & transactions | The inventory system will not be connected to the user’s financial ledger or customer details. This means that there will be manual work for the user to make sure that and additional bookkeeping is aligned. | The user must keep any additional records separately. However an extension could be planned to add bookkeeping functionality to the system if there is demand. |
| Usability | The user may not be familiar with this type of system. This could lead to errors in data entry which would reduce the functional benefit of the app. | User training must take place to reduce the likelihood of user error. |
| Performance Requirements | Large numbers of records may be required to be kept by the application. This could cause slowdowns and reduce the app’s functionality. | The back end should be designed to be as extensible as possible to ensure large volumes are handled efficiently. |

**Description here**

The implementation constraints plan developed for this project is shown above. The biggest concern here is the time constraint.

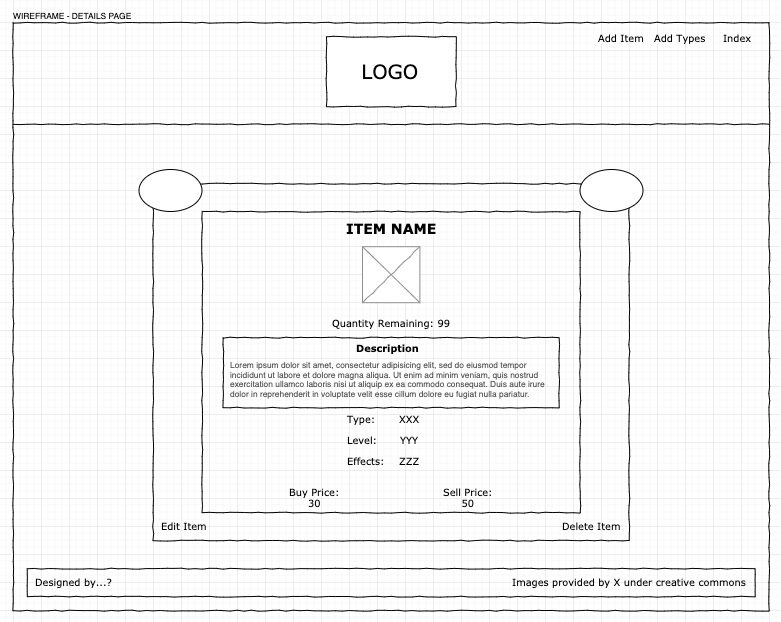
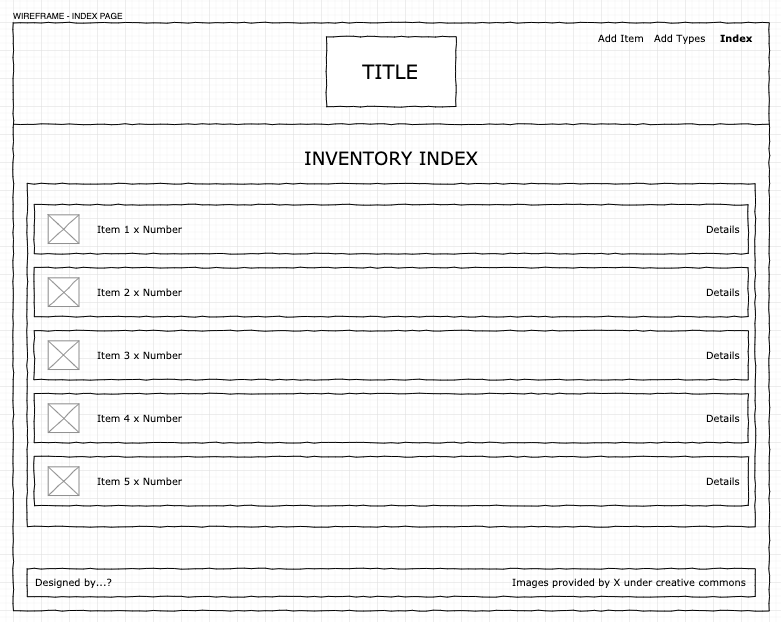
| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| P | P.5 | User Site Map | |

**Paste Screenshot here**

**Description here**

Above is the site map for the project.

| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| P | P.6 | 2 Wireframe Diagrams | |

**Paste Screenshot here**

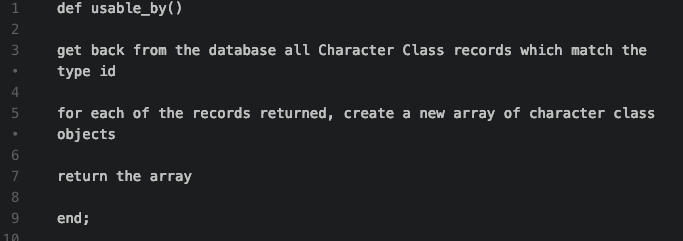
**Description here**

These are the wireframes for the inventory index page and for the item details page. The final result was very close.

**Week 5**

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

**Paste Screenshot here**

****

**Description here**

This pseudocode describes what the method **usable\_by** should be doing on the types object.

Using pseudocode was helpful in planning what steps the method needs to go through to accomplish the required result.

| 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 | |

**Paste Screenshot here**

**Description here**

The above screenshots show the application screen that allows the user to create a new item. The first screenshot is the data entry form, and the second screenshot shows the resulting page which retrieves the stored data entered above.

| 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 | |

**Paste Screenshot here **

****

**Description here**

In the above screenshots, you can see the data being input to create a new item type. In the second screenshot, you can see that the entered item type has become persistent, and can be used to create new items.

| 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 | |

**Paste Screenshot here**

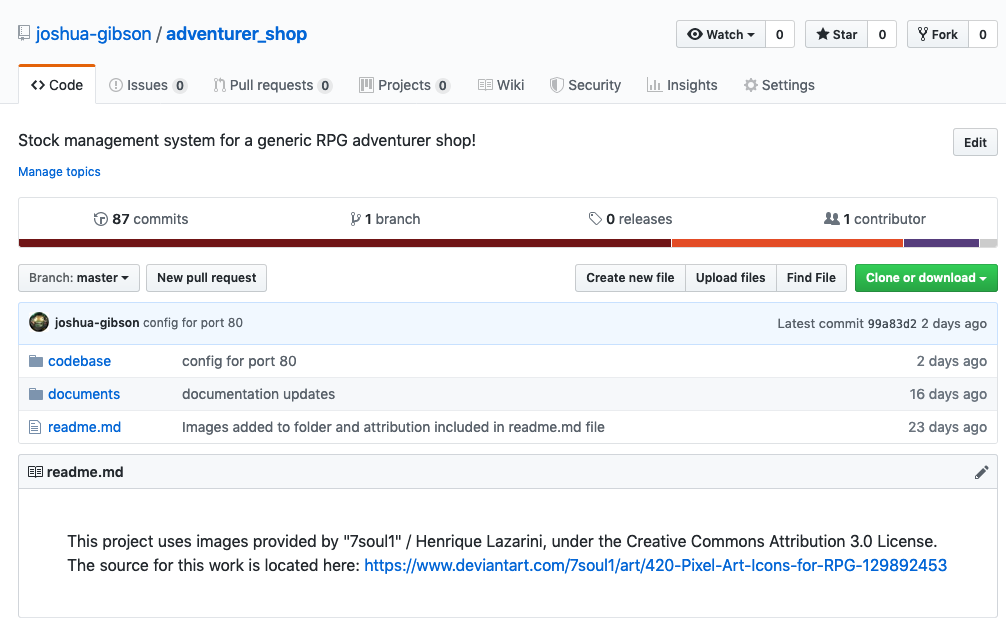
****

**Description here**

It the first screenshot, you can see in the ‘sell item’ square, the user has entered input that they wish to sell 2 of the selected items.

In the second screenshot, after the user has pressed the sell button, you can see that the quantity has reduced by 2.

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

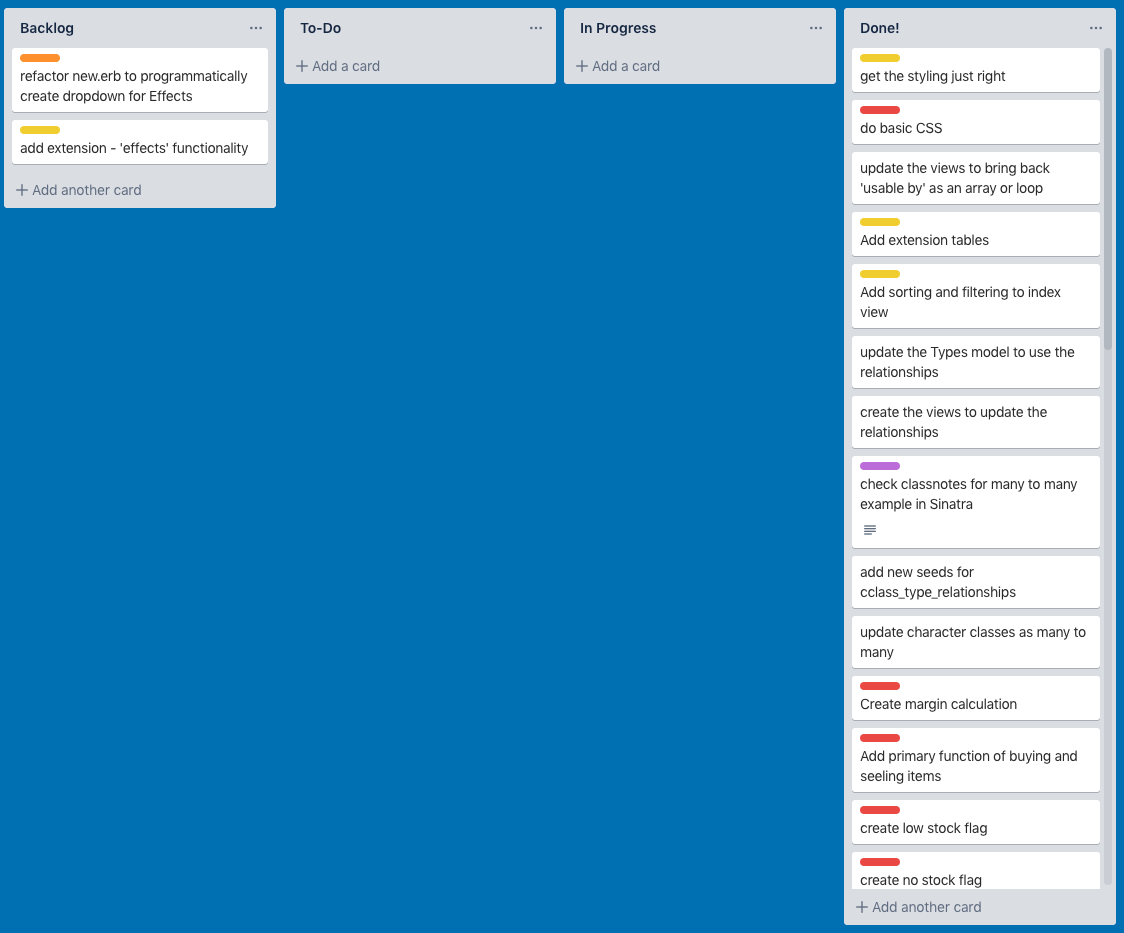
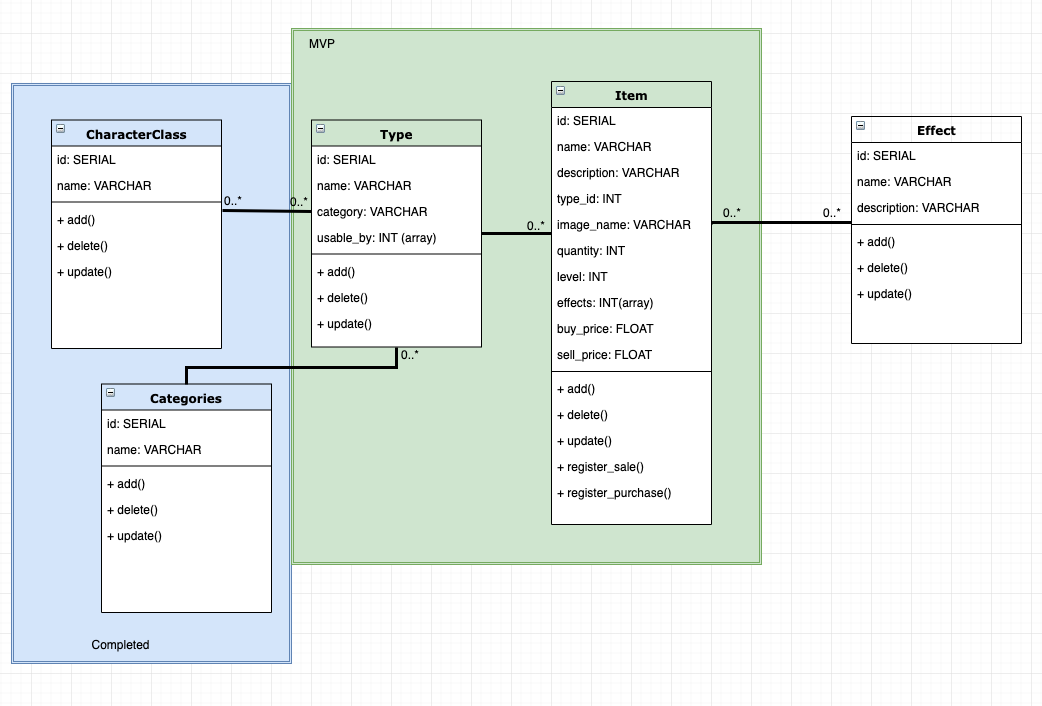
**Paste Screenshot here**

**Description here**

The above screenshot shows the Github repo for my solo project. This can be viewed in the browser via the following URL:

<https://github.com/joshua-gibson/adventurer_shop>

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

**Paste Screenshot here**

**Description here**

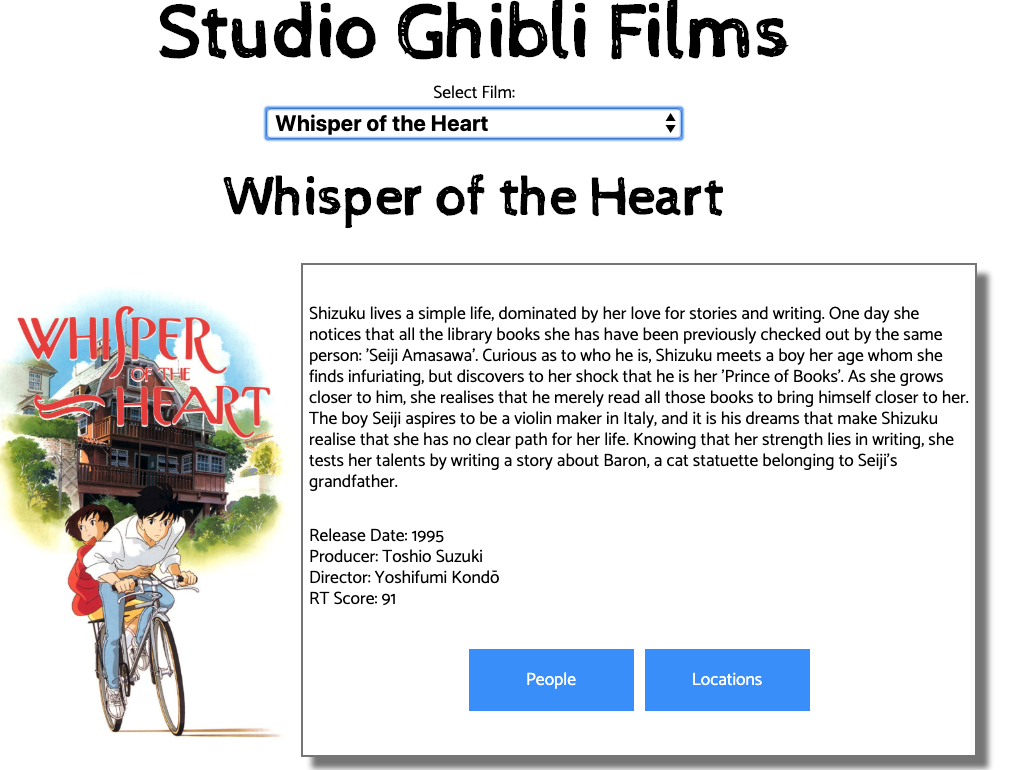
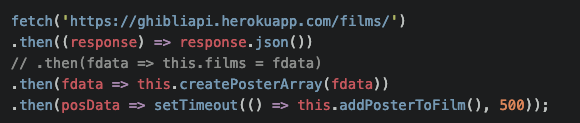
The screenshots above shows how the project was planned and how that changed over time. In the first screenshot, you can see in the green border the ‘MVP’, or minimum viable product. This Wass completed about half way through the project. At that stage the classes in the blue border were added as an extension.

The second screenshot shows the Trello board which was used in the planning process. As tasks were completed, these moved from the Backlog column through to the Done column.

**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 | |

**Paste Screenshot here**

****

**Description here**

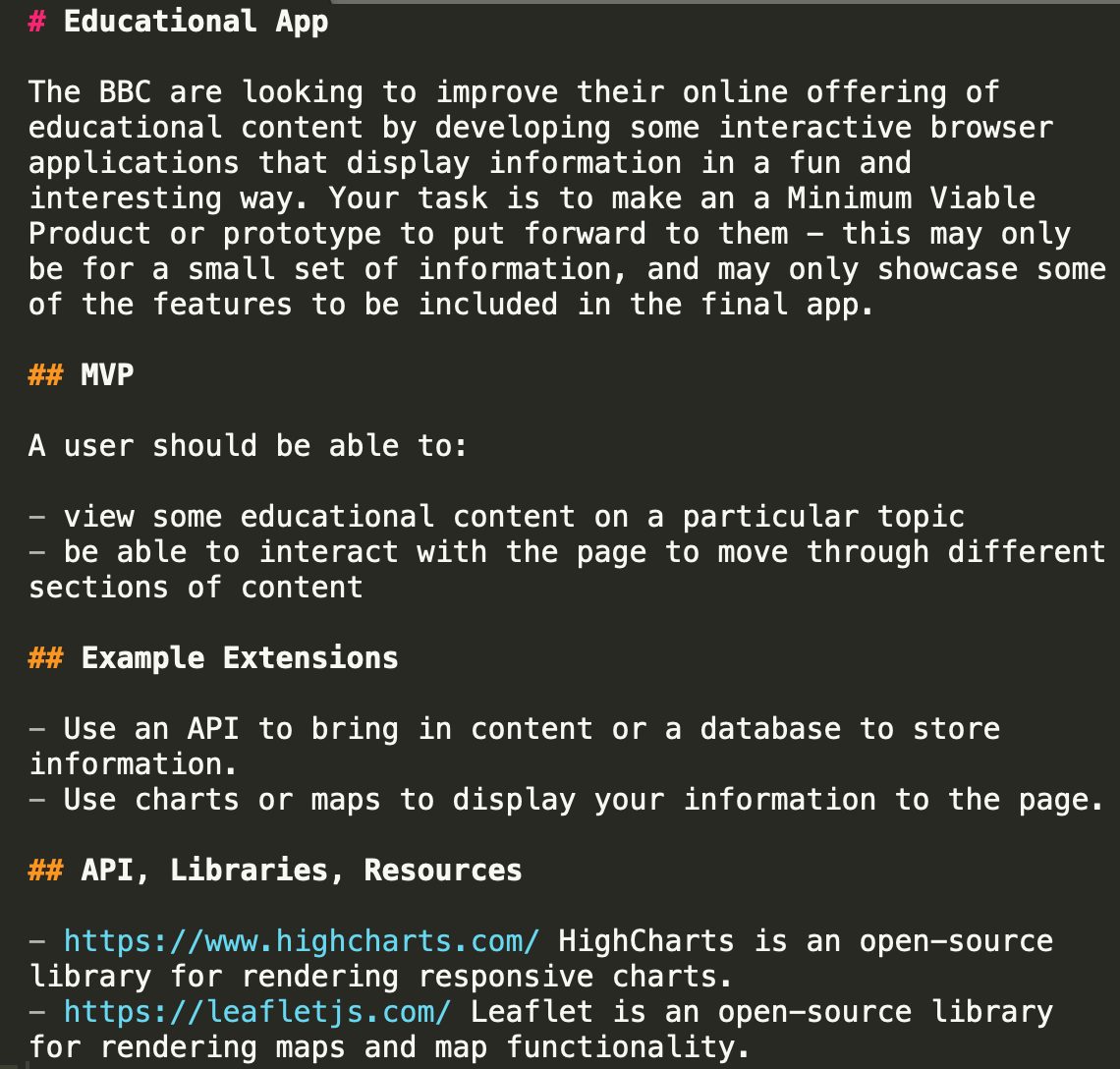
The first screenshot shows the API call to the Studio Ghibli API (<https://ghibliapi.herokuapp.com/>).

This API allows access to information about Studio Ghibli films.

The second screenshot shows my app which makes use of this information to display it to the user based on their inputs.

**Week 8**

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

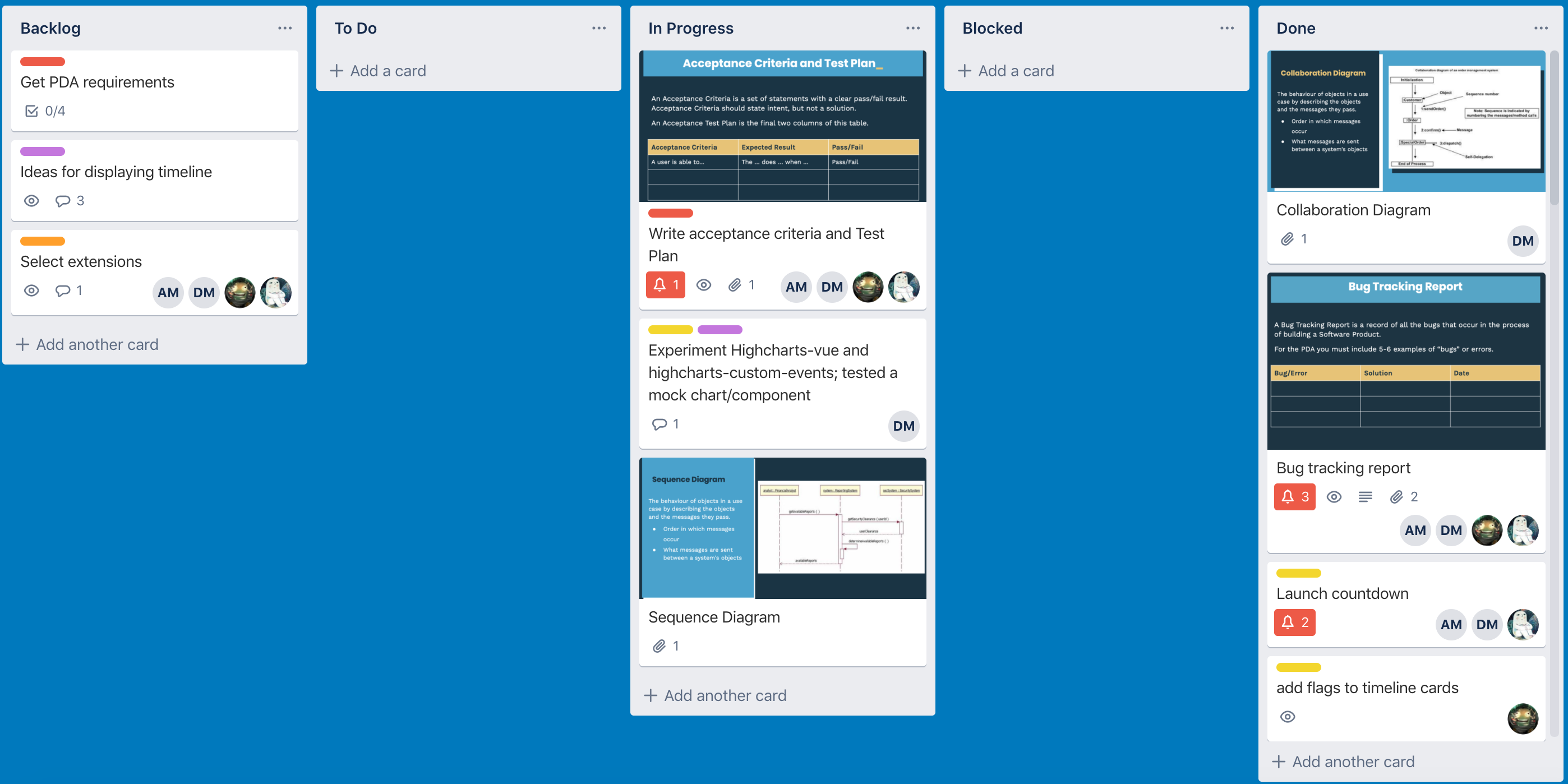
**Paste Screenshot here**

**Description here**

This is the project brief. Based on the remit, we decided to build an app that shows space launches, both historically and future launches via an existing API.

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

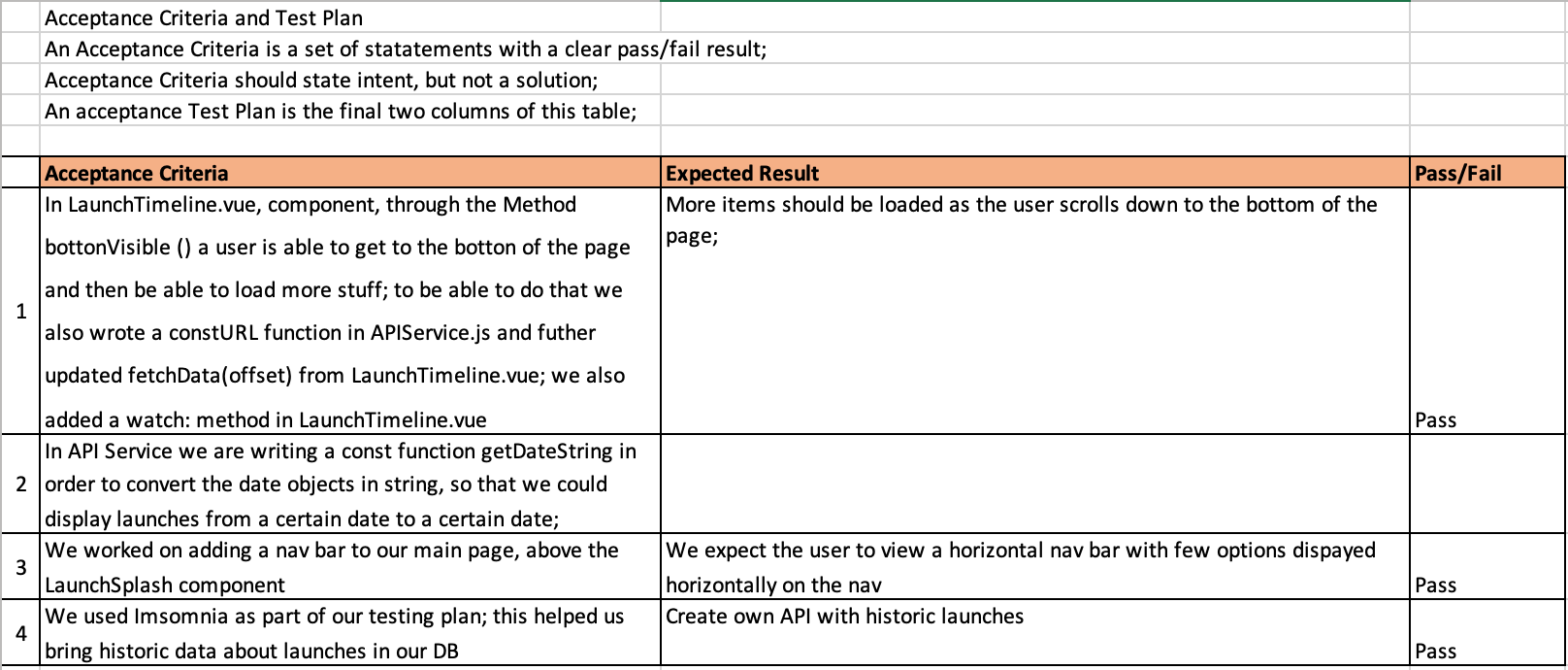
**Paste Screenshot here**

****

**Description here**

Above is the Trello board we used to manage our group project.

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

**Paste Screenshot here**

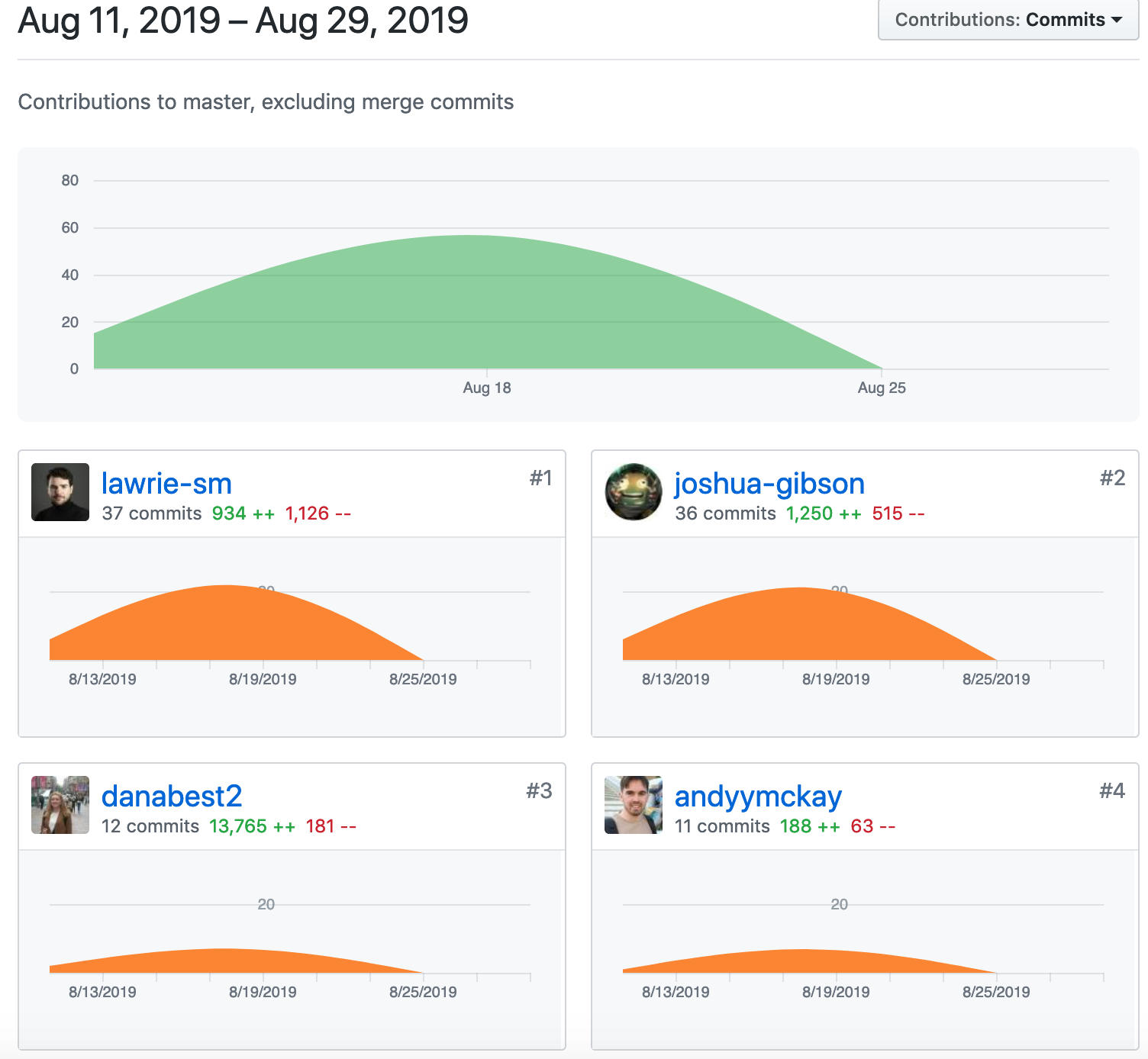
**Description here**

Above is the acceptance criteria and test plan built by the group as part of the project.

**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. | |

**Paste Screenshot here**

****

**Description here**

This is a screenshot of the contributors page for our project.

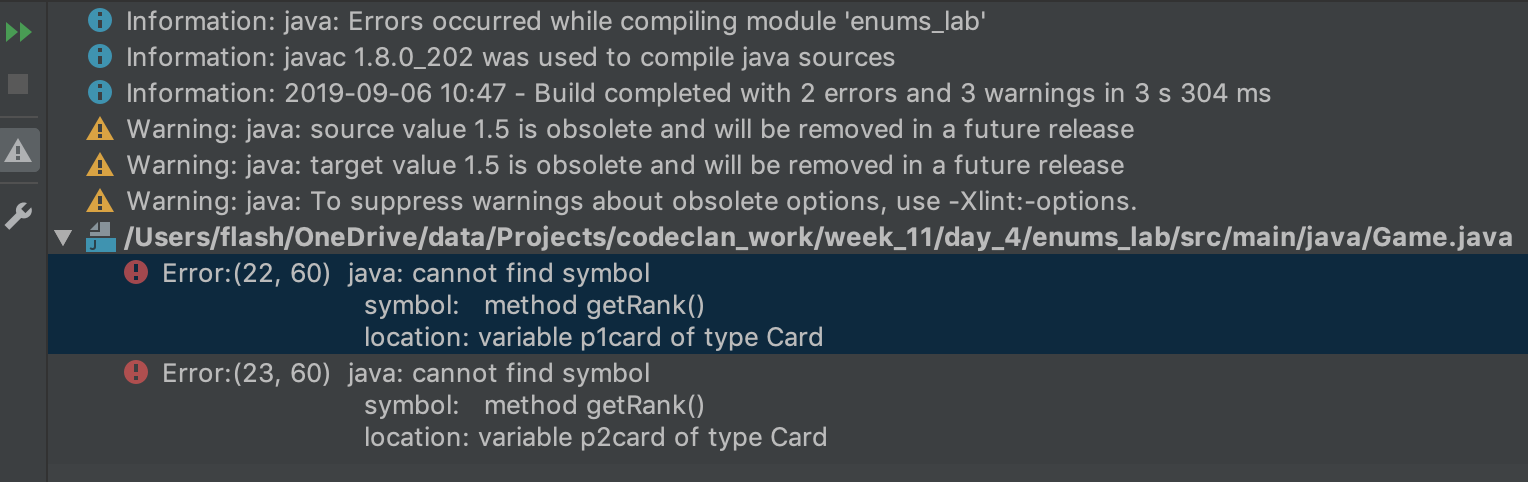
**Week 11**

| 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 | |

**Paste Screenshot here**

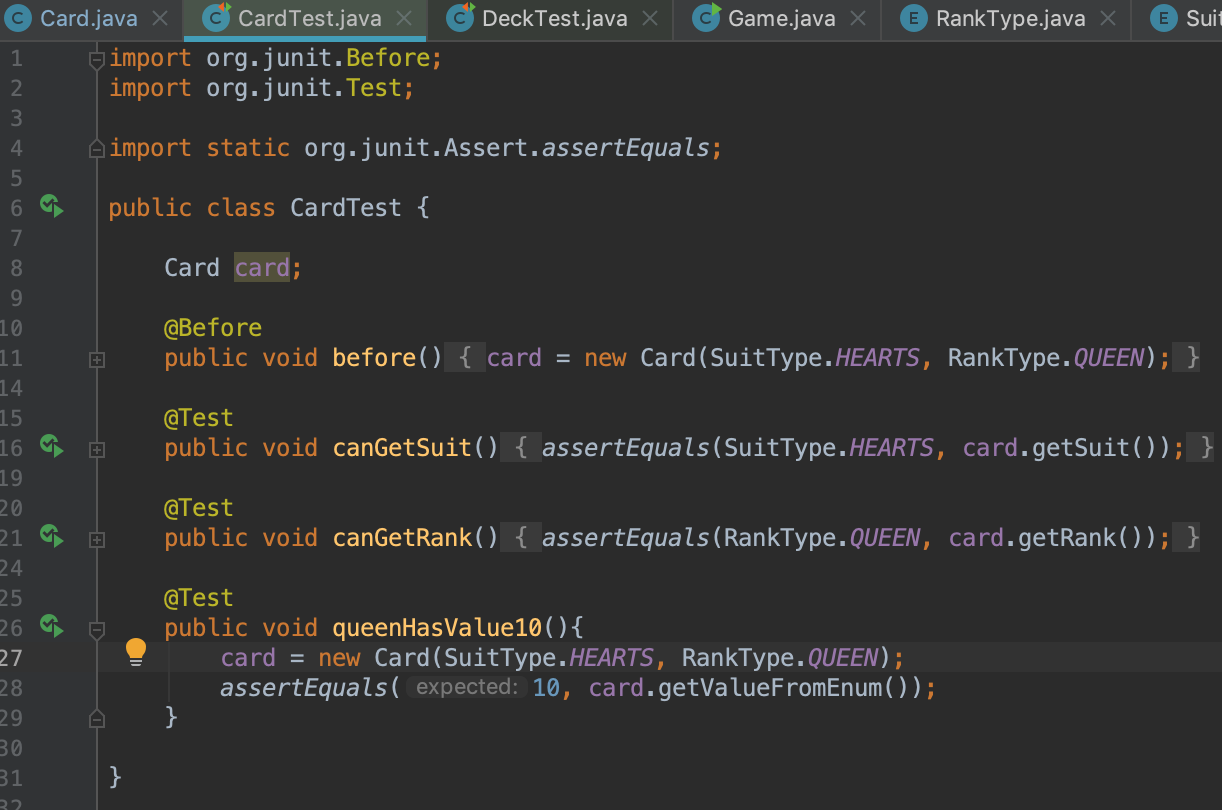
**Description here**

This shows the test code for the Card class. This will fail when run because the getRank() method hasn’t yet been built.

**Paste Screenshot here**

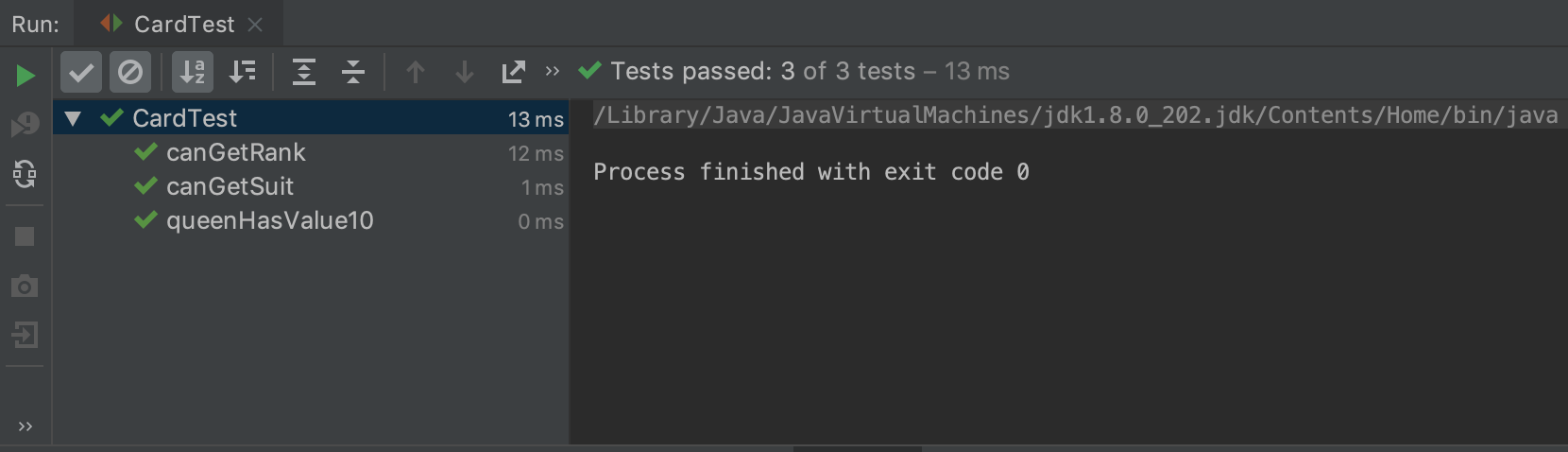
**Description here**

This shows the failure of the above tests, because of the missing method getRank().

**Paste Screenshot here**

**Description here**

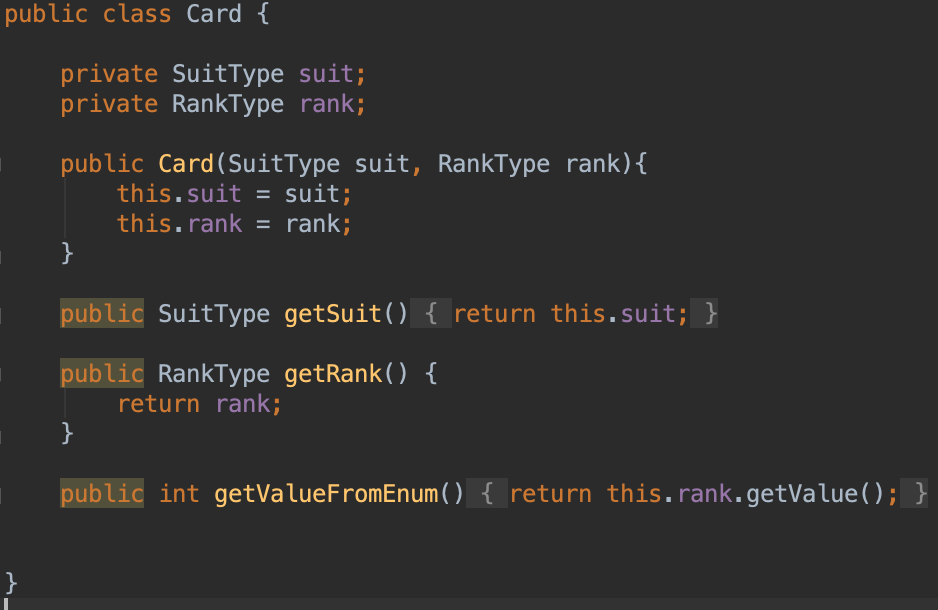
This shows the test code with the errors corrected, the getRank() method has now been created.

**Paste Screenshot here**

**Description here**

This screenshot shows the tests running successfully after having been corrected.

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

**Paste Screenshot here**

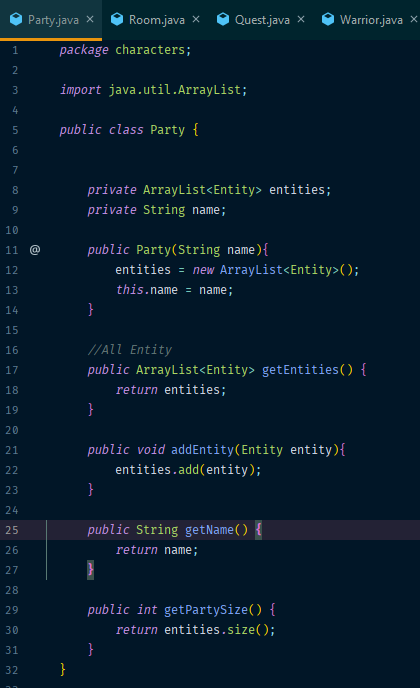
**Description here**

This screenshot is an example of encapsulation in the Card class. It has two properties, SuitType and RankType. These properties are set to ‘private’ at the top of the class so they cannot be accessed directly. The first two methods in the class are ‘getters’, their purpose is to allow the properties to be accessed within the principles on encapsulation. They allow the properties to be read, but only as the programmer intended, and the values cannot be set directly.

**Week 12**

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

**Paste Screenshot here**

****

**Description here**

The screenshot shows an object called Party, which includes an array list of Entities, which can be of different types (warrior, cleric, wizard). You can see also different methods which allow for the population of the array list, and retrieving information about it.

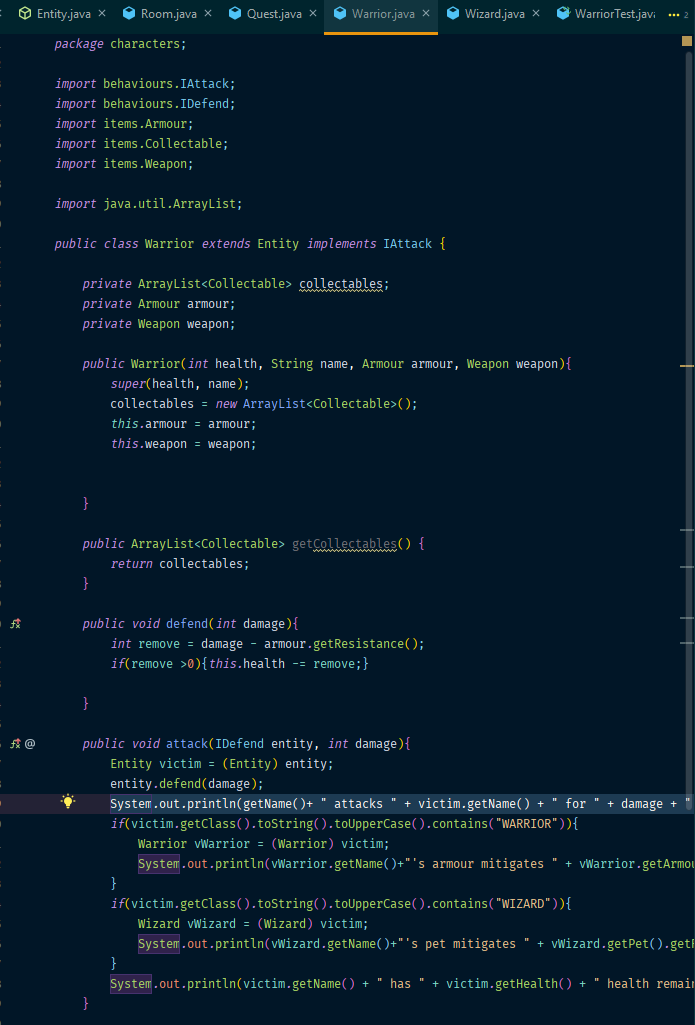
**Paste Screenshot here**



**Description here**

This shows the abstract class Entity, which is used in the arraylist on Party above. It implements the IDefend and IRecover interfaces, which allow the entity to defend from attach and recover health.

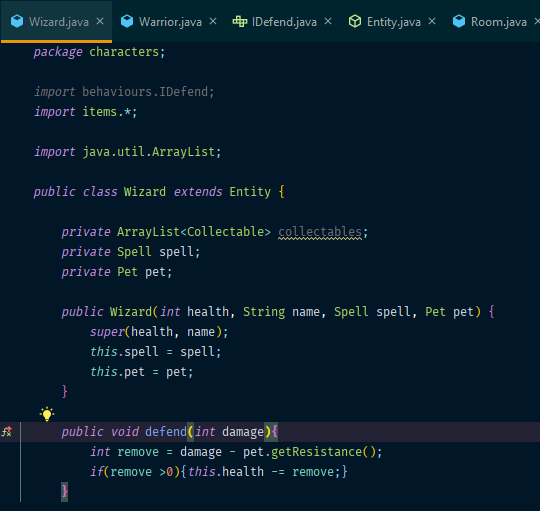
**Paste Screenshot here**



**Description here**

This is the Warrior class, which extends Entity. Here you can see that it implements the IDefend interface specified in the abstract class.

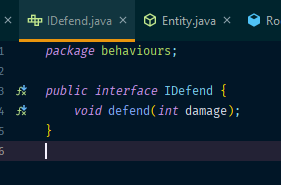
**Paste Screenshot here**



**Description here**

This is the Wizard class, similar to the Warrior class above, which is also extended from Entity and implements IDefend.

**Paste Screenshot here**



**Description here**

This is the IDefend interface code referenced above.

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

**Paste Screenshot here**

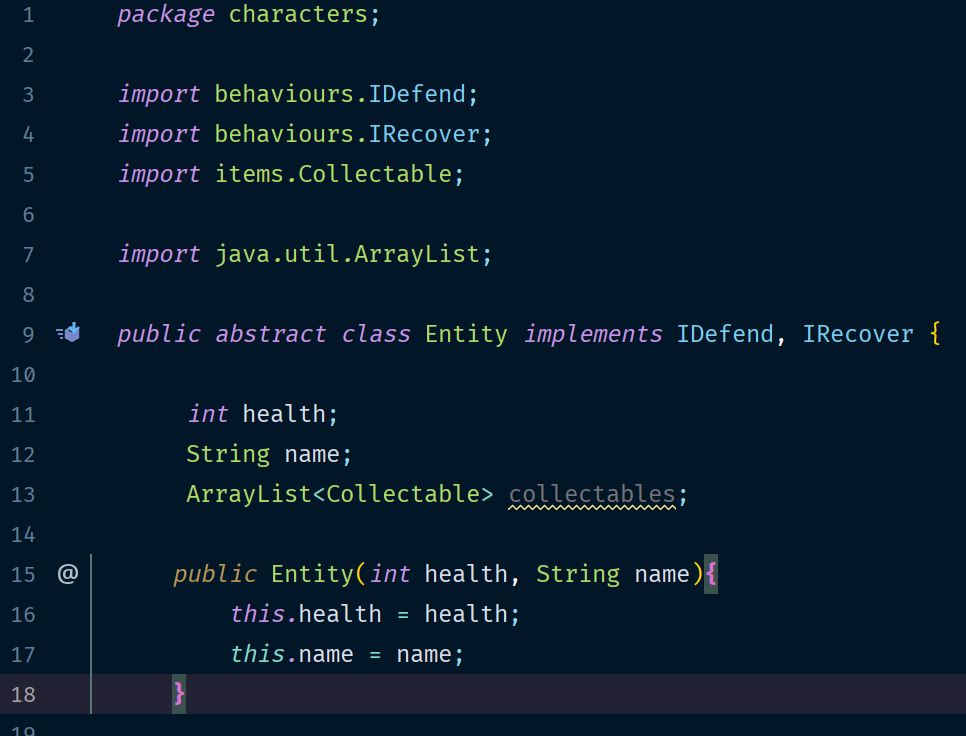


**Description here**

Above is the inheritance diagram for the Fantasy Game lab, in which an entity was created as an abstract class and the Warrior, Wizard, and Cleric classes were extended from it.

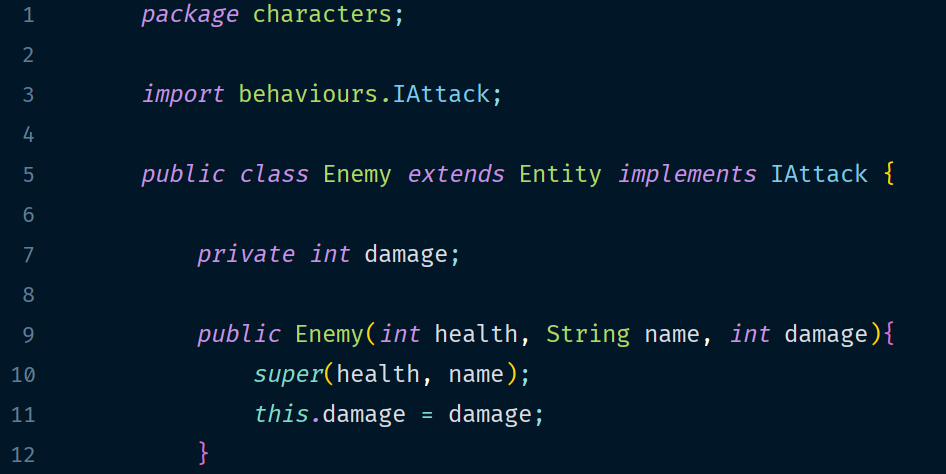
| Unit | Ref | Evidence |  |
| --- | --- | --- | --- |
| I&T | I.T.2 | Take a screenshot of the use of Inheritance in a program. Take screenshots of:  \*A Class  \*A Class that inherits from the previous class  \*An Object in the inherited class  \*A Method that uses the information inherited from another class. | |

**Paste Screenshot here**

****

**Description here**

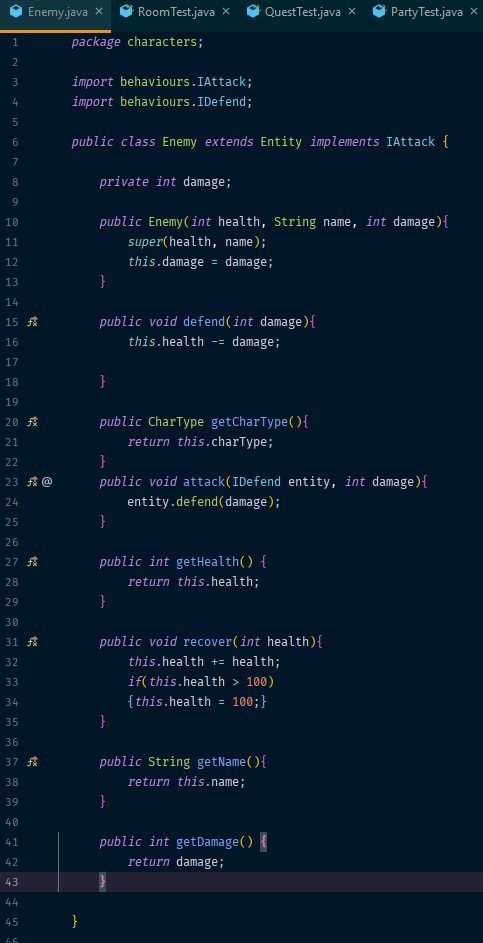
This screenshot shows the abstract class Entity.

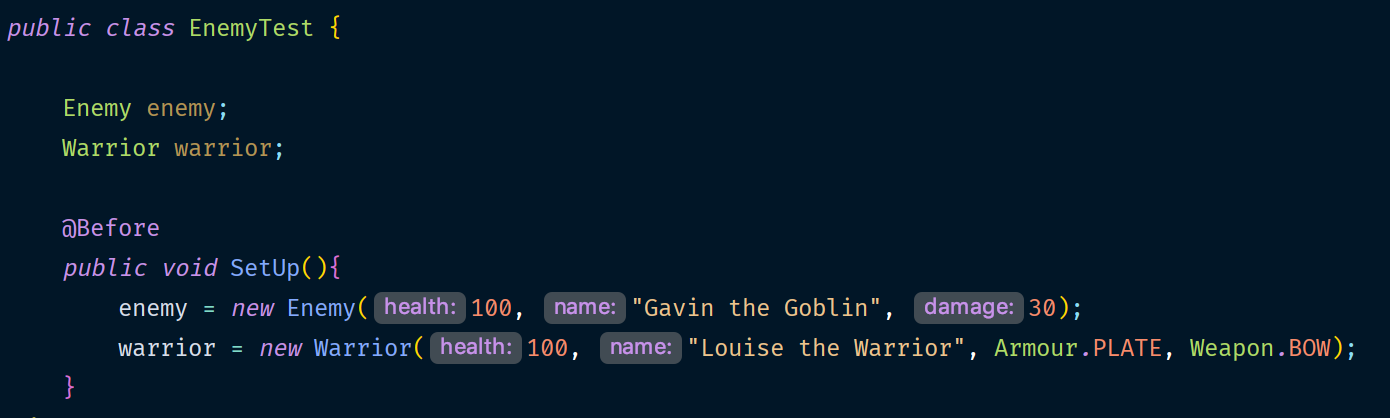
**Paste Screenshot here**

**Description here**

This screenshot shows the class Enemy, which extends the abstract class Entity.

**Paste Screenshot here**

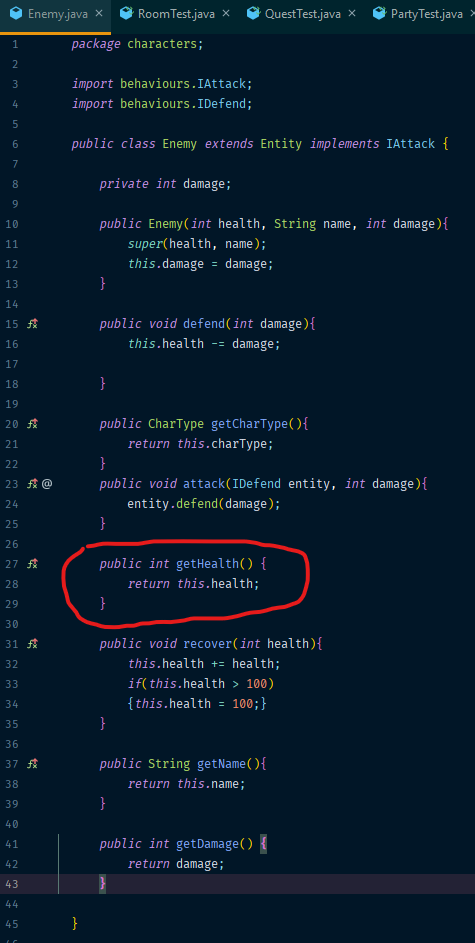
****

****

**Description here**

These screenshots shows an object ‘enemy’ (below), created from the inherited class ‘Enemy’ (above).

**Paste Screenshot here**

****

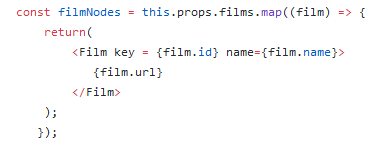
**Description here**

This method on the Enemy class returns ‘health’, which is inherited from the Entity class.

**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. | |

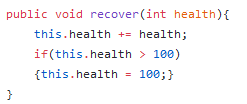
**Paste Screenshot here**

****

**Description here**

The algorithm above takes an ArrayList of films and for each creates a new container from the film’s id, name, and URL. I’ve chosen this because I quite like the efficiency of the ‘map’ method.

**Paste Screenshot here**

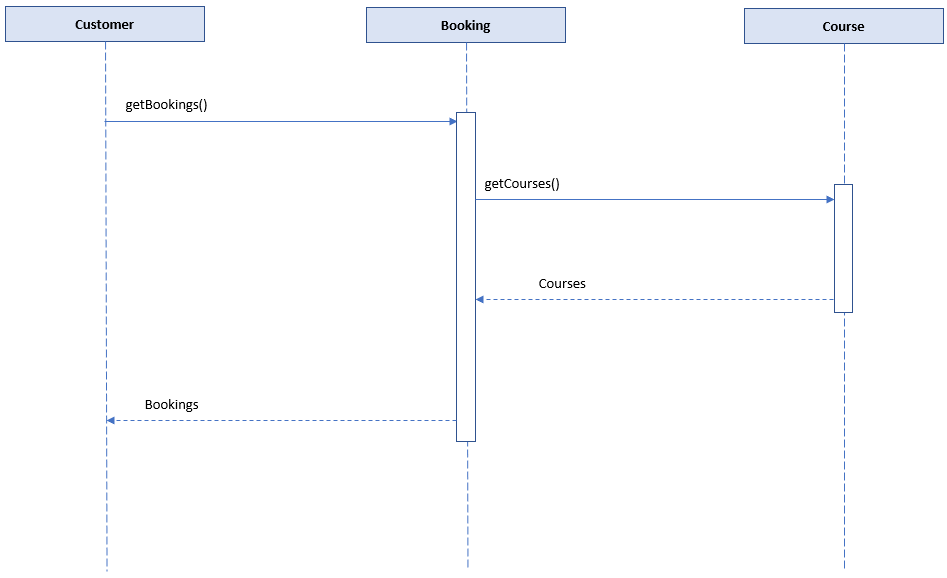
****

**Description here**

The above algorithm is part of the adventure project, and it is triggered whenever a character recovers health. It takes in the amount of health to be recovered, and limits any recovery to cap the characters health at 100 points. I chose this because it is a good example of code that is very simple and reusable.

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

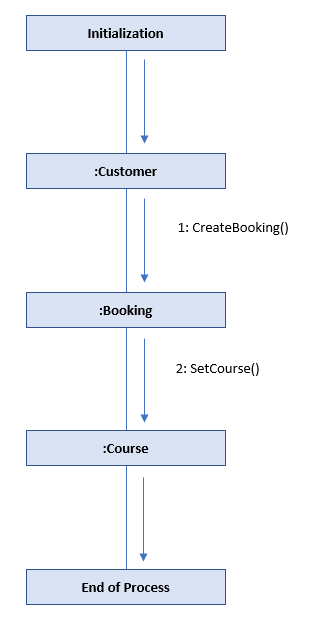
**Paste Screenshot here**

****

**Description here**

This screenshot shows a Sequence Diagram. You can see that when a Customer object invokes the getBookings method, the Booking object needs to retrieve the Course details from the Course object and return this information together to the Customer object.

**Paste Screenshot here**

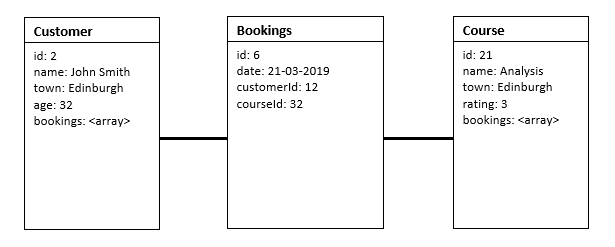
****

**Description here**

This shows a Collaboration diagram, in which a customer can create a booking. The Creation request sends a message to the Booking object, which in turn needs to assign a course which is being booked, so sends a message to the Course object.

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

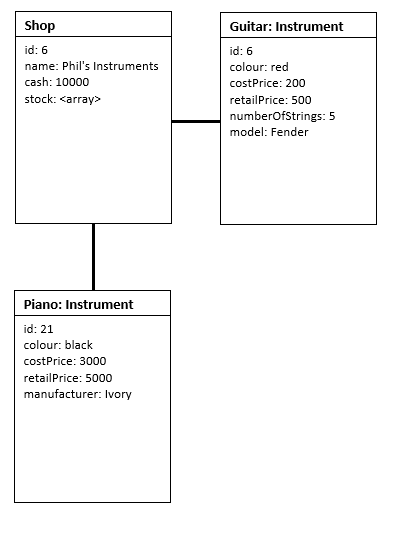
**Paste Screenshot here**

****

**Description here**

The above shows an object diagram of the course booking app. It shows that the Bookings table holds the relationship between the Customers and the Courses. Here we can see that the customer John Smith has booking ID 6, which means he is booked on the Analysis course on the 21st of March.

**Paste Screenshot here**

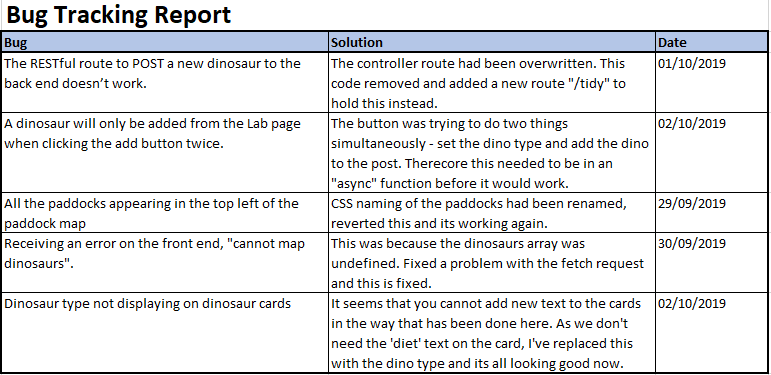
****

**Description here**

This shows an object diagram of an Instrument Shop app. This indicates that the Shop object holds an array of Stock, in which a Guitar and Piano are held, which are both Instruments.

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

**Paste Screenshot here**

****

**Description here**

This is the bug tracking report for the Jurassic Park lab. It shows some of the issues encountered and how they were resolved.