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.

Evidence Key: A&D - Analysis and Design Unit
I&T - Implementation and Testing Unit
P - Project Unit

Week 1

Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of an object literal in a program. Take screenshots of: *An object literal in a program *A function that uses the object *The result of the function running

Here is an object called 'holiday' that includes three key value pairs. A function is used on this object called cutShort that updates the amount of 'days' by dividing it by three. This means the amount of days changes to 3 after the function is ran (as seen in the console log below).

```
const holiday = {
    location: 'Thailand',
    days: '21',
    airport: 'Edinburgh'
  };

  const cutShort = function(days){
    holiday.days = holiday.days / 7
  }
  cutShort();

  console.log(holiday.days);
```

```
Last login: Mon Apr 29 16:50:17 on ttys004
➜  day_03 git:(master) ✗ node Homework.js
3
➜  day_03 git:(master) ✗
```

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

```
var coffees = [
  { type:"Latte", price:2.65 },
  { type:"Cappuccino", price:2.75 },
  { type:"Flat white", price:2.95 },
  { type:"Super-dooper Mocha Deluxe", price:3.79 }
]
for (var coffee of coffees){
  console.log(coffee.type + " cost £" + coffee.price + " each");
}
```

```
Latte cost £2.65 each
Cappuccino cost £2.75 each
Flat white cost £2.95 each
Super-dooper Mocha Deluxe cost £3.79 each
➜  hw_js_fundamentals git:(master) ✗
```

This is an array (coffees) of objects, with various key value pairs. The loop function then goes through each item in the array and returns the type, price and text, which can be seen in the above screenshot.

Week 2

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('can get pets by breed', () => {
  expect(myFunctions.getPetsByBreed(petShop, "British Shothair")).toBe(2);
});
```

```
  ● pet shop › can get pets by breed

    expect(received).toBe(expected) // Object.is equality

    Expected: 2
    Received: 0

      109 |
      110 |   test('can get pets by breed', () => {
    > 111 |     expect(myFunctions.getPetsByBreed(petShop, "British Shothair")).
toBe(2);
          |
      112 |   });
      113 |
      114 |   test('returns 0 if not pet by breed found', () => {

      at Object.toBe (specs/pet_shop.test.js:111:69)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 22 passed, 23 total
Snapshots:   0 total
Time:        1.217s
Ran all test suites.
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! pet_shop@1.0.0 test: `jest`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the pet_shop@1.0.0 test script.
npm ERR! This is probably not a problem with npm. There is likely additional log
ging output above.

npm ERR! A complete log of this run can be found in:
npm ERR!     /Users/a9854261/.npm/_logs/2019-05-02T21_11_13_960Z-debug.log
➜  specs git:(master) ✗
```

Here is a test to see if pets can be obtained by breed, the code is in the first screenshot. The test code fails because of the misspelling of “British Shorthair”. After this is corrected the test passes.

```
test('can get pets by breed', () => {
  expect(myFunctions.getPetsByBreed(petShop, "British Shorthair")).toBe(2);
});
```

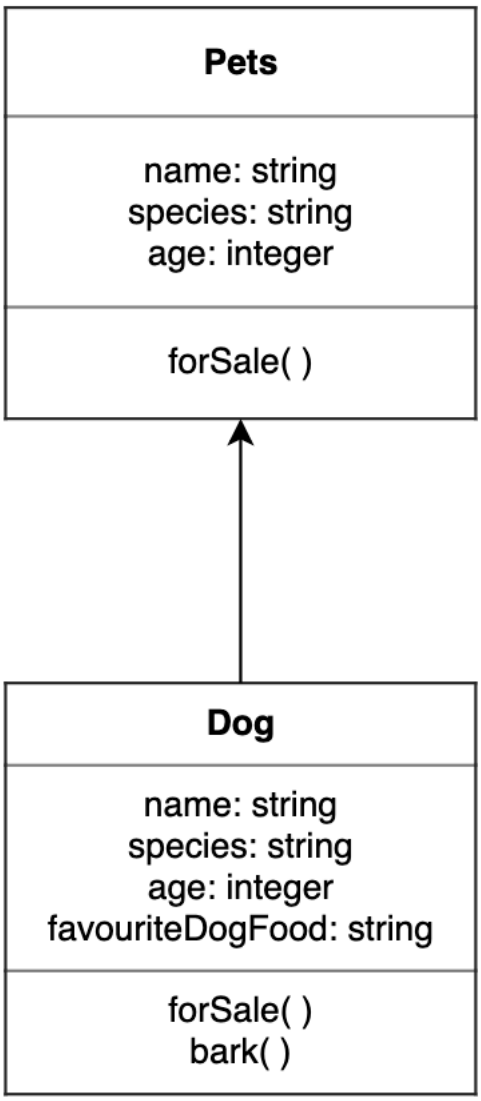
```
Test Suites: 1 passed, 1 total
Tests:       23 passed, 23 total
Snapshots:   0 total
Time:        1.109s
Ran all test suites.
➜  specs git:(master)
```

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

```
class Bag {
  constructor() {
    let weight = 5;
    this.getWeight = function() {
      return weight;
    };
    this.setWeight = function(itemWeight) {
      weight += itemWeight;
    };
  }
}
var myBag = new Bag();
console.log(myBag.getWeight());
myBag.setWeight(12);
console.log(myBag.getWeight());
console.log(myBag.weight);
```

Line 3 creates a private instance of the weight data, this private variable can then be accessed by the method setWeight on line 7. A new instance of the class is then created, which would have the weight of 5 on the first log. Because the setWeight() function is then ran the weight would be updated to 12, which is what the second log will give. This is because it can access the function getWeight. However the third log will return ‘undefined’ because it cannot access the property ‘weight’.

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



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.

```
class Pets {
  constructor(name, species, age) {
    this.name name;
    this.species = species;
    this.age = age;
  }

  forSale() {
    console.log(`Welcome to the shop - ${this.name} is a ${this.species} that is for sale!`);
  };

}
```

Here a class is created of Pets which has the method forSale() within it. Another class is then created called Dog which inherits (using super) the name, species and age of the pet. However, as favouriteDogFood is specific to the Dog class, this is added into the constructor. A new instance of this class is created which uses the method forSale that it inherited from the Pet class. It also has a bark method from its own class.

```
class Dog extends Pets {
  constructor(name, species, age, favouriteDogFood) {
    super(name, species, age);
    this.favouriteDogFood = favouriteDogFood;
  }
  bark() {
    console.log(`Woof!`);
  };
}

let rover = new Dog('Rover', 'Bulldog', 2, 'Chicken')


rover.forSale();
rover.bark();
```

Week 3

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.

```
highestCard(card1, card2){
  if(card1.value > card2.value){
    return card1.name
  }
  else{
    return card2.name
  }
},
```

Both of these functions were used in a card game app. The first calculates the total value of cards someone has, by looping through each card in their hand and adding the value onto the running total. It then returns this total in a string at the end. The second function determines which card is higher out of the two played by checking if the value of the first is higher than the second - if so it returns the name of the first, if not the second.

```
cardsTotal(cards){
  let total = 0;
  for(let card of cards){
    total += card.value;

  }
  return "You have a total of " + total;
}
```

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

id	title	main_star	year_released
2	Beetlejuice	Alec Baldwin	1988

```
//GET a particular film from ID
router.get('/:id', function(req, res) {
  SqlRunner.run("SELECT * FROM films WHERE id = $1", [req.params.id]).then(
    result => {
      res.status(200).json(result.rows);
    });
});
```

Here is the code from my backend (using SQL) that searches the data by an id. If I run this function with the id 2 I get the screenshot above.

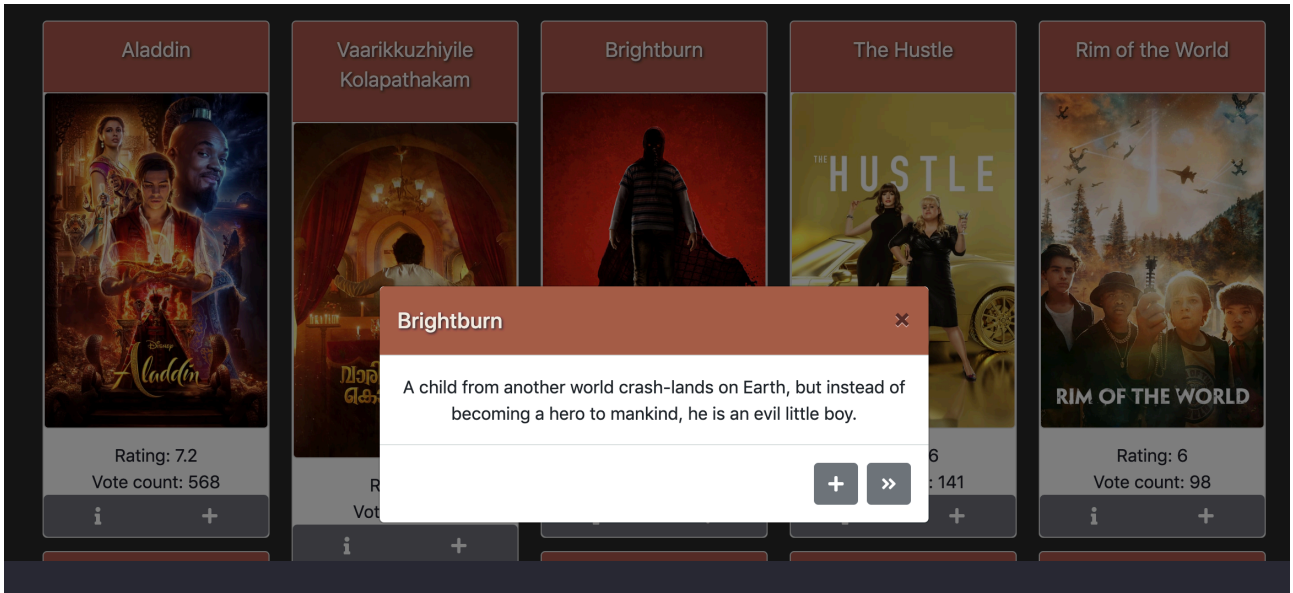
Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running

```
//GET all the films
router.get('/', function(req, res) {
  SqlRunner.run("SELECT * FROM films ORDER BY title ASC").then(result => {
  res.status(200).json(result.rows);
  });
});
```

id	title	main_star	year_released
2	Beetlejuice	Alec Baldwin	1988
6	Corpse Bride	Johnny Depp	2005
1	Edward Scissorhands	Johnny Depp	1990
4	Nightmare Before Christmas	Danny Elfman	1993

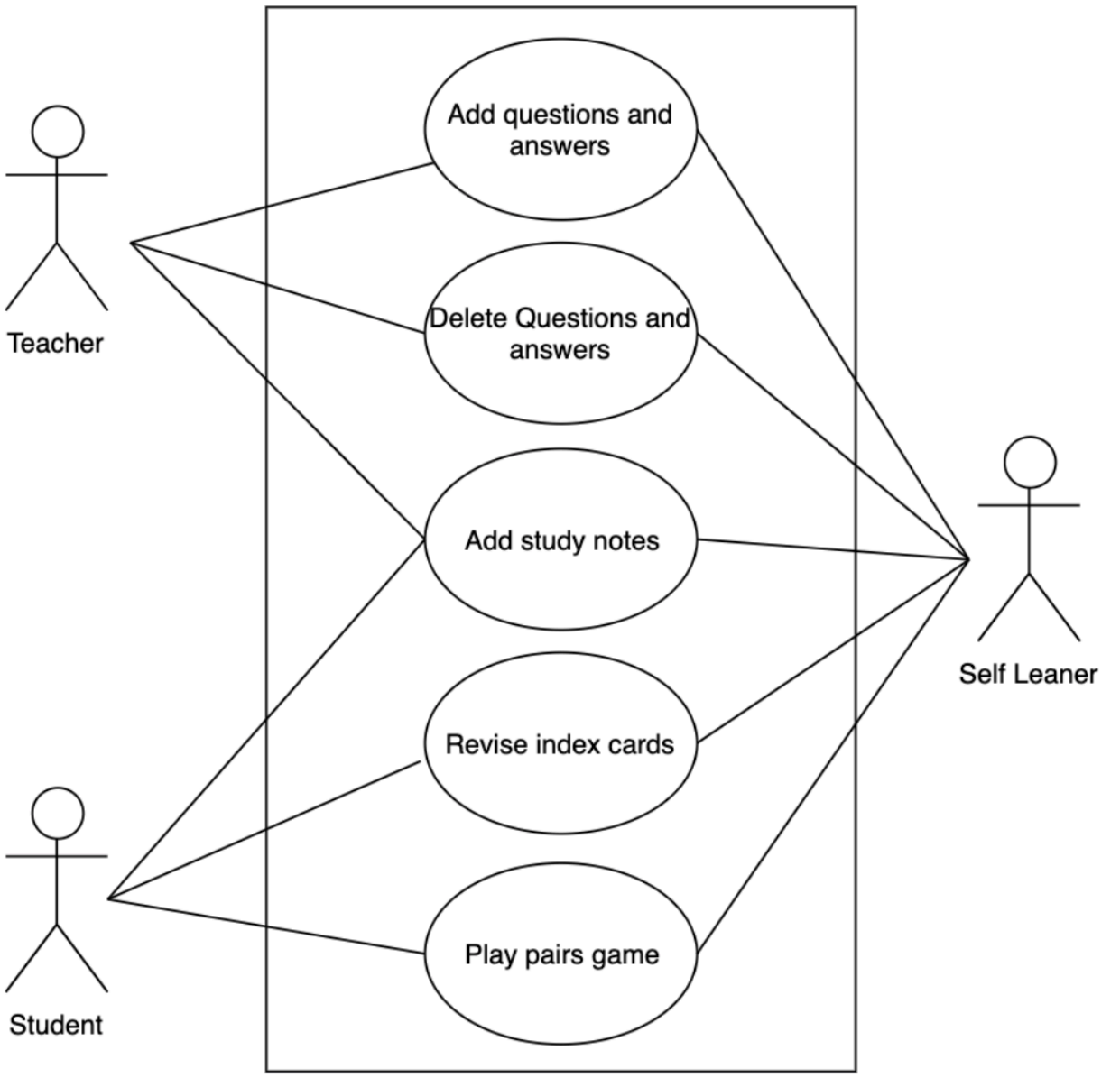
Here my backend is sorted in ascending order by title using SQL. The table to the left is the result.

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



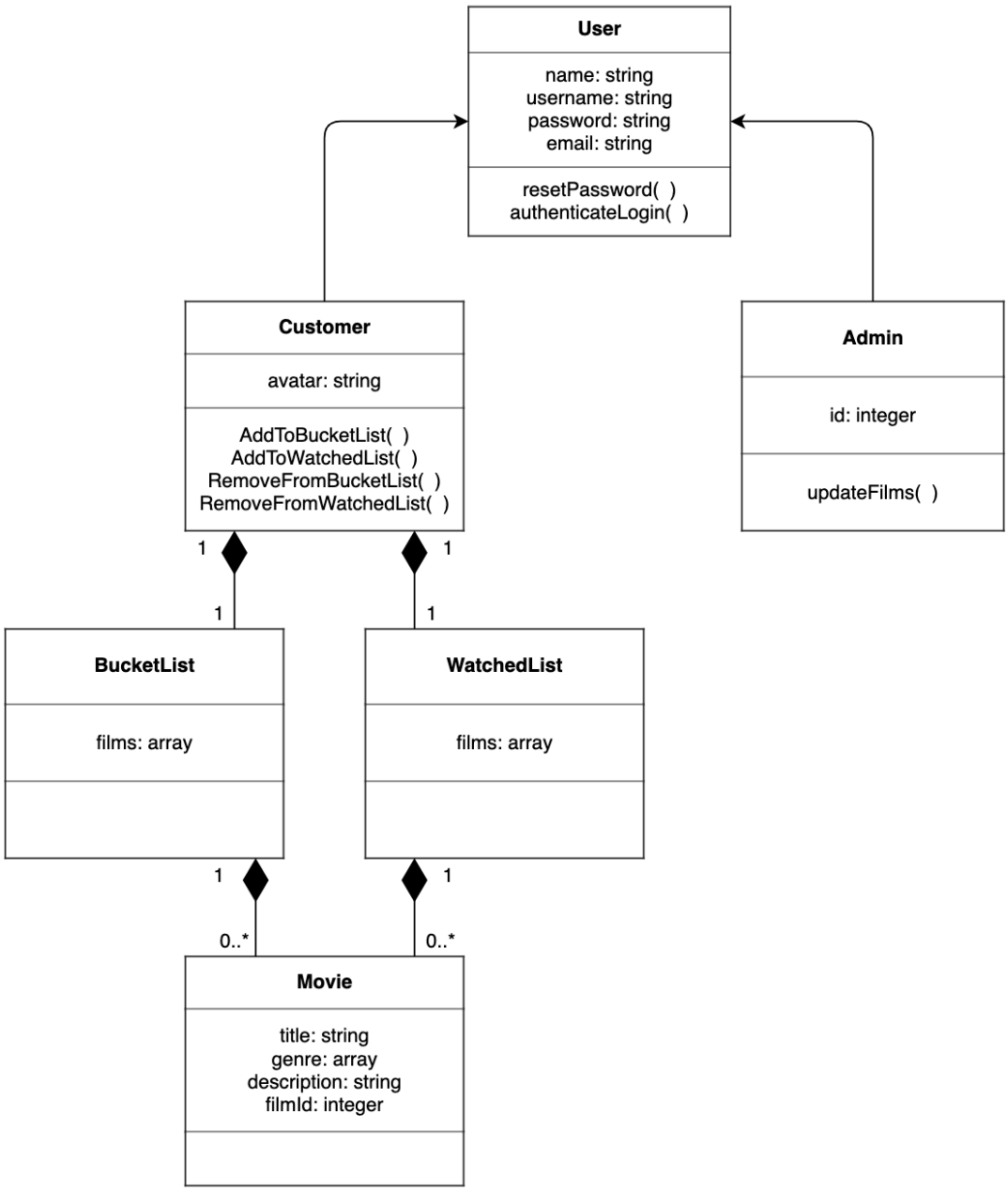
The getPopularData method uses an API to pull through the details of the top 20 films. This includes the poster of the film (for example) which is rendered on the page using the code above which you can see screenshots of above.

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



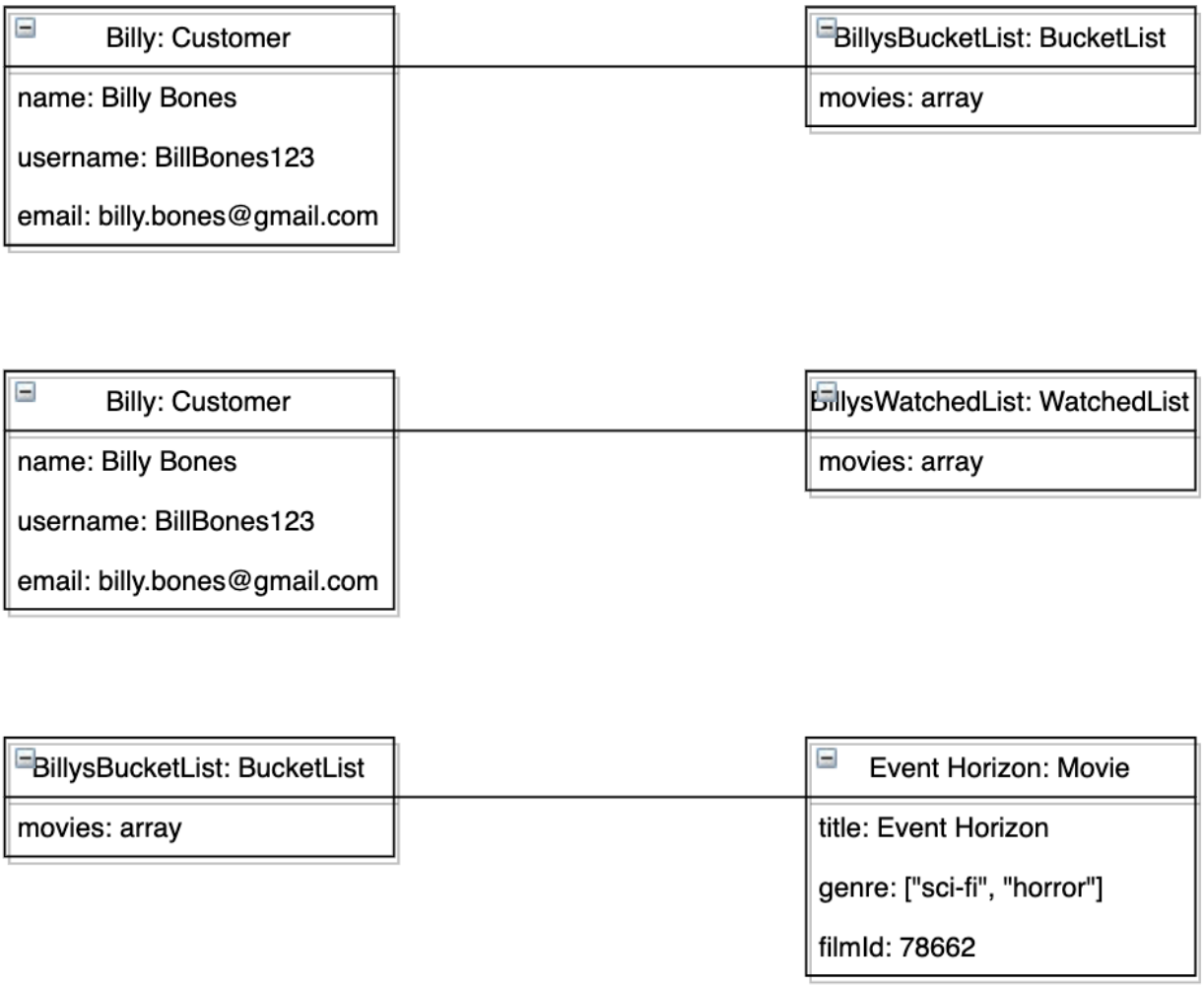
Here is the use case diagram for a revision app, which shows how different users have different needs from the application. For example a teacher may want to amend content i.e. adding/ deleting questions, answers and study notes. A student may also want to add their own study note but might also want to revise the index cards and play a revision game, which the teacher has no use for. Finally a self learner may want to use all aspects of the app, as they add and then revise their own content.

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



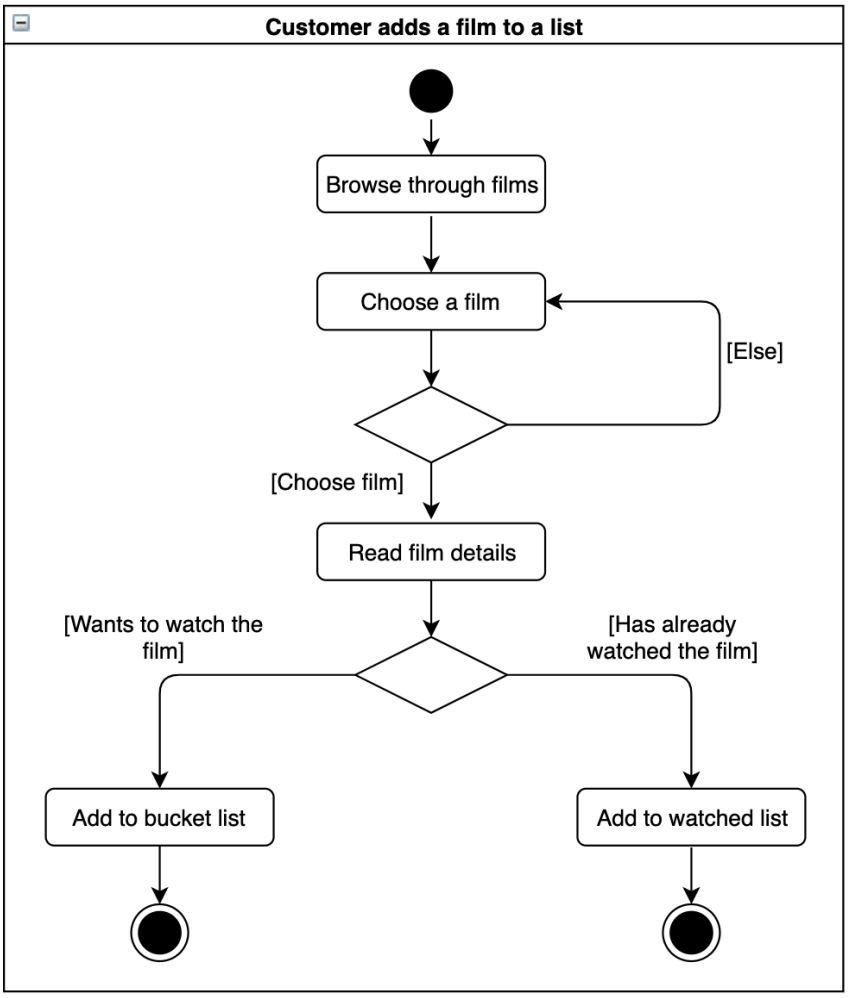
Here is a class diagram for an app where users can create bucket lists of films they'd like to watch, as well as keeping a record of ones they've already watched. This will mean there will be two types of users (as shown above) which will be admin and customers. Although admin may want to update the full list of films available on the app, they won't need their own lists. The customer however will have one bucket list and one list of films they've watched, both of which will contain many movies.

Unit	Ref	Evidence
A&D P	A.D.3 P.8	Produce three Object Diagrams



Above are three examples of objects from the same movie list app. The first is a customer (Billy) who has a bucket list. Billy is an instance of the class 'customer' and he has an instance of the class 'BucketList'. In the same way the second diagram shows how he has a watched list too. The final diagram shows BillysBucketList and one instance of a movie which is Event Horizon - its likely however that he would have many of these objects within his list.

Unit	Ref	Evidence
A&D	A.D.4	Produce an Activity Diagram



Here is an example of an activity diagram where the activity is a customer adding a film to one of their lists. First the customer will log in to the app - of the details are authenticated then popular films will be displayed, if not they will return to the same log in page. Then the customer will make a decision on whether to choose a film, if they do they will click a film and this will display that particular films details. Another decision is then made about whether the customer chooses the ‘Add to bucket list’ or the ‘Add to watched list’ button which will add it to the relevant list.

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	If the application isn't available on mobile then it will not display correctly - this could deter users from using it on the go	Ensure the application is mobile optimised
Performance requirements	There is a limit to the amount of requests that can be made to a third party API, which could mean that the data doesn't pull though for the films so customers can't see the details/images.	The popular films is loaded from the API which is within the limit, but when a film is added to the watched or bucket list it is stored in a database so doesn't need to call the API again
Persistent storage and transactions	All of the information for the lists will need to be stored in a database for an indefinite amount of time, which means storage is required which could potentially get corrupted. This might deter customers from storing their films.	Data will be stored within the backend database which will be backed up.
Usability	Some of the buttons are icons so it may be difficult for users to be confident in what a button does, making it inaccessible for some. This might mean customers stop using the app if it's not clear how they should.	Included tooltips on each button to say what action it will perform
Budgets	A budget is set that the project will need to abide to, so extra cash wouldn't be available if needed for a new feature. This could mean the project isn't fully completed.	Ensure that resources and costs are planned carefully from the beginning, allowing an acceptance leeway if it exceeds these.
Time	A deadline is set that the project will need to be launched by, if it is not launched by the date the money will have been wasted on marketing.	Core features must be prioritised and resources planned wisely

Week 5

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

Here is an example of pseudocode that was written in order to explain what a function would do prior to it being written in code. The function will loop through an array of films and if the films has been ‘watched’ it will display that film a card.

```
//render films
//loop through each film in the film array
//for each film confirm whether it's status is set to watched
//if it is watched ...
//... then create a card for this film
```

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

Below you can see the user has clicked the ‘tick’ button which records the film as watched. The screenshot underneath shows film then appearing in the ‘watched’ tab.

POKÉMON Detective Pikachu

Rating: 5.00
Vote count: 6

Cold Pursuit

Rating: 5.30
Vote count: 361

Kamen Rider Heisei Generations FOREVER

Rating: 4.90
Vote count: 11

Captain Marvel

Guardians of the

POKÉMON Detective Pikachu

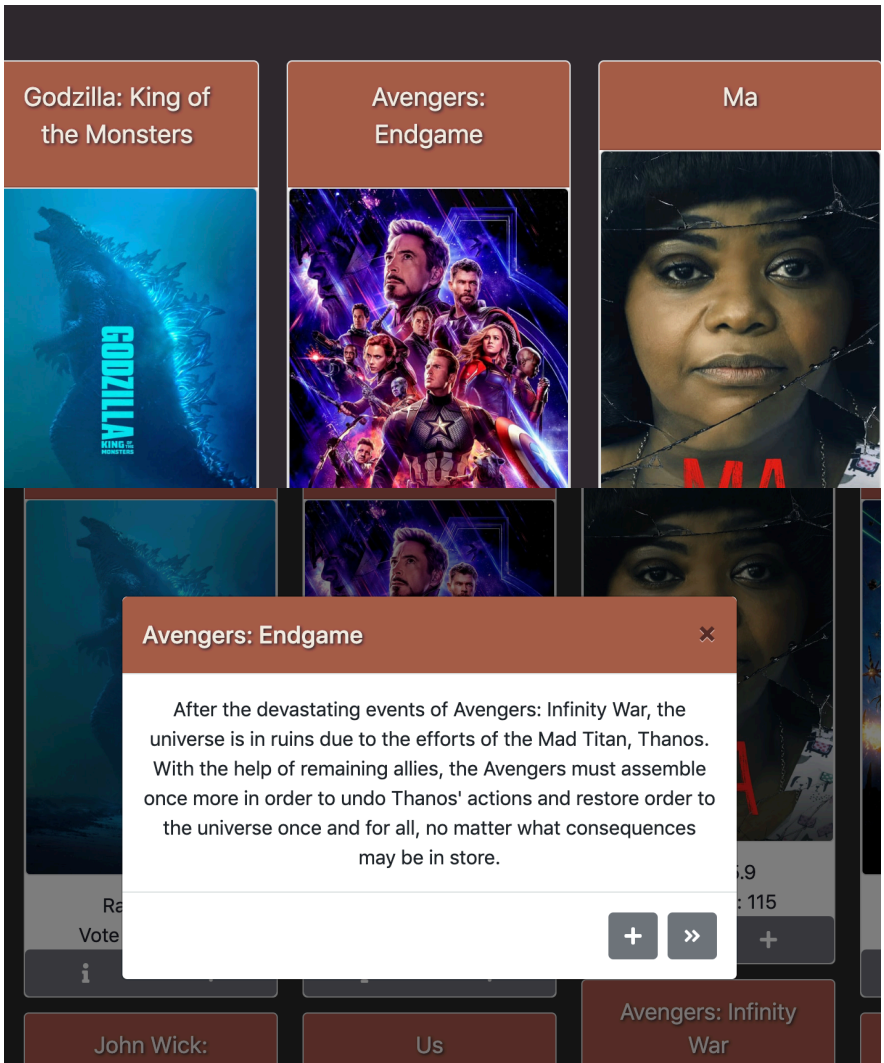
Rating: 5.00
Vote count: 6

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

id	film_id	title	vote_average	vote_count	poster_path	watched
7	447404	POKÉMON Detective Pikachu	5.00	6	/wgQ7APnFpf1TuviKHXeEe3KnsTV.jpg	TRUE
8	283995	Guardians of the Galaxy Vol. 2	7.70	12120	/y4MBh0EjBlMuOzv9axM4qJlmhzz.jpg	FALSE
11	438650	Cold Pursuit	5.30	361	/otK0H9H1w3JVGJjad5Kzx3Z9kt2.jpg	FALSE
12	299537	Captain Marvel	7.10	4510	/AtsgWhDnHTq68L0lLsUrCnM7TjG.jpg	FALSE

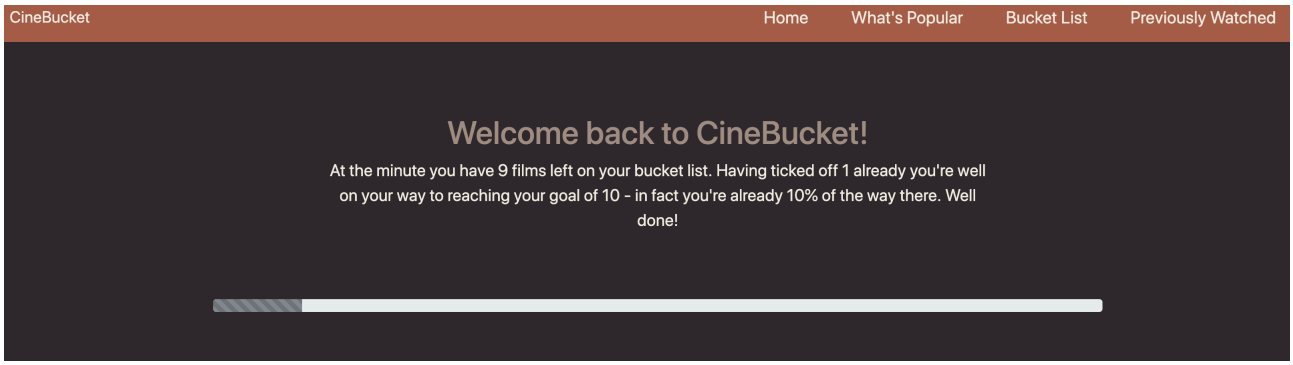
Using the first screenshot of the last question you can see the user inputting that they have watched a film (Detective Pikachu) which then updated the watched field in the database from false to true.

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



Here the user has clicked the ‘i’ or information button on the film ‘Avengers: Endgame’ which has triggered a modal to appear which gives the film description as seen below.

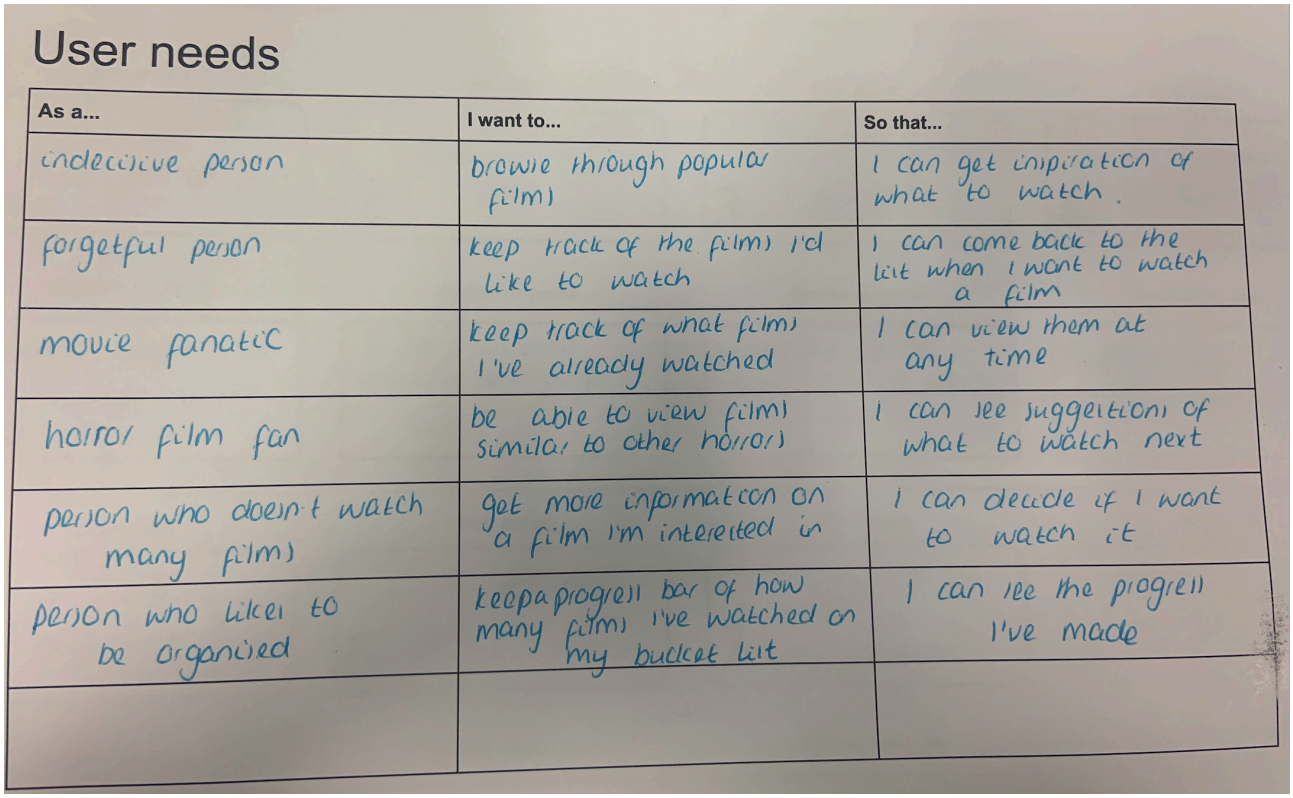
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.



Here is a screenshot of my film list app that I made alone with the link below. This is a private repository as it includes an API key.

https://github.com/roseulldemolins/cinebucket

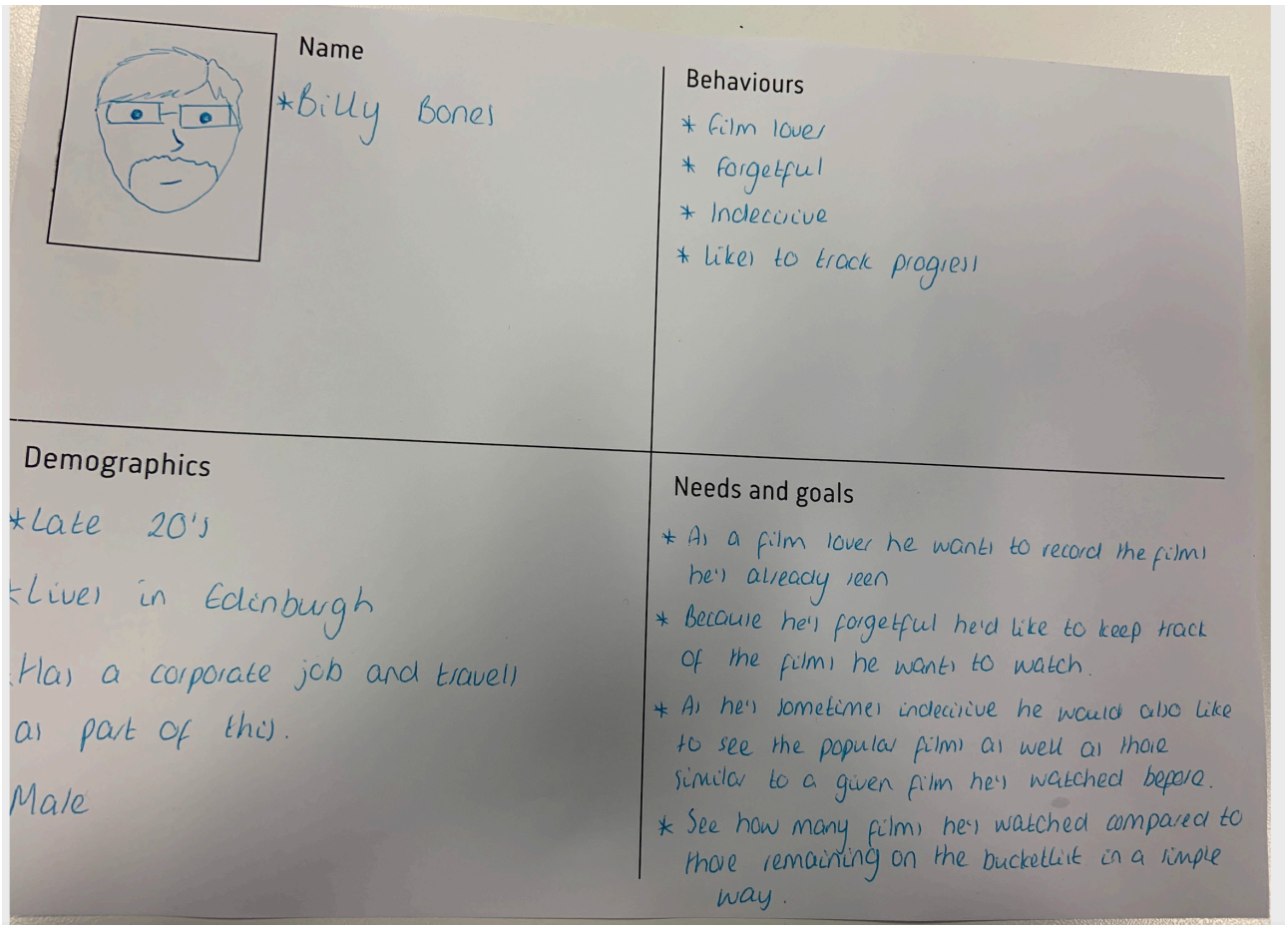
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.



User needs

As a...	I want to...	So that...
indecisive person	browse through popular films	I can get inspiration of what to watch.
forgetful person	keep track of the films I'd like to watch	I can come back to the list when I want to watch a film
movie fanatic	keep track of what films I've already watched	I can view them at any time
horror film fan	be able to view films similar to other horrors	I can see suggestions of what to watch next
person who doesn't watch many films	get more information on a film I'm interested in	I can decide if I want to watch it
person who likes to be organised	keep a progress bar of how many films I've watched on my bucket list	I can see the progress I've made

Here is my UX planning which allowed me to look at different user needs and plan how I would build my app in order to fulfil these.



Name

*Billy Bones

Behaviours

* Film lover
* Forgetful
* Indecisive
* Likes to track progress

Demographics

*Late 20's

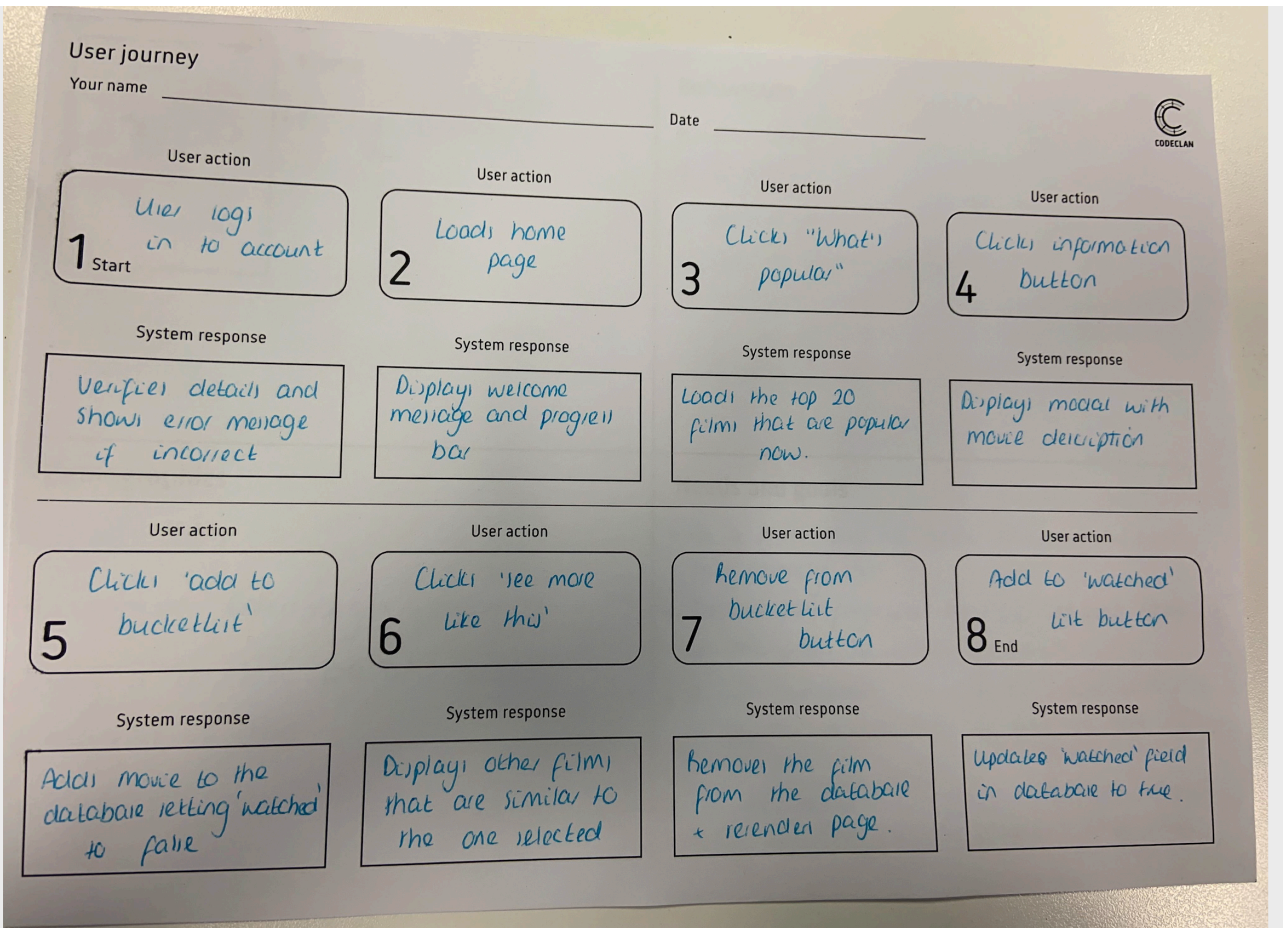
*Lives in Edinburgh

*Has a corporate job and travels as part of this.

Male

Needs and goals

* As a film lover he wants to record the films he's already seen
* Because he's forgetful he'd like to keep track of the films he wants to watch.
* As he's sometimes indecisive he would also like to see the popular films as well as those similar to a given film he's watched before.
* See how many films he's watched compared to those remaining on the bucketlist in a simple way.



User journey

Your name _____ Date _____

CODECLAN

User action: 1 Start — User logs in to account	User action: 2 — Loads home page	User action: 3 — Clicks "What's popular"	User action: 4 — Clicks information button
System response: Verifies details and shows error message if incorrect	System response: Displays welcome message and progress bar	System response: Loads the top 20 films that are popular now.	System response: Displays modal with movie description
User action: 5 — Clicks 'add to bucketlist'	User action: 6 — Clicks 'see more like this'	User action: 7 — Remove from bucketlist button	User action: 8 End — Add to 'watched' list button
System response: Adds movie to the database setting 'watched' to false	System response: Displays other films that are similar to the one selected	System response: Removes the film from the database & refreshes page.	System response: Updates 'watched' field in database to true.

Week 7

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

Revision App Bug Tracker

Ref. No.	Bug Description	Resolution	Date Reported	Closed
1	Type error, in reducer	Needed to initialise store as []	30/05/2019	Y
2	Type error, in FlashCard component	Added conditional not to try to render empty array	30/05/2019	Y
3	Flipping flashcard causes random re-selection of Q/A	Added conditional not to pick new Q/A on re-render	30/05/2019	Y
4	Flipping flashcard on Answer side reveals answer on next card	Reset "isFlipped" for new card	30/05/2019	Y
5	No content showing below NavBar	Rearranged router	31/05/2019	Y
6	New items not appearing in DB	Update DB *as well as* state/store	01/06/2019	Y
7	Blank study notes can be added	Low priority	02/06/2019	N
8	Flashcards sometimes show blank	Filtered out non Q&A cards	03/06/2019	Y
9	Pairs game uses duplicate Q's	Make unique Q array	03/06/2019	Y
10	Restart pairs not retrieving new cards	unresolved	04/06/2019	N
11	Study note delete crashes app	Amended reducer & dispatch	04/06/2019	Y
12	Pairs game throws error on refresh	Added conditional to render "blank" game	04/06/2019	Y
13	Extra button appears on completion of pairs game	unresolved	05/06/2019	N
14	New study note text area remains populated after subbing	unresolved	05/06/2019	N
15	Very long words overspill study note "post-its"	low priority	05/06/2019	N
16	Scaling: pairs cards don't all fit on screen at >= 100%	unresolved	05/06/2019	N
17	Scaling: lines on flash cards don't all show at some scales	intermittent	05/06/2019	N

Here is the bug tracking report which gives details of bugs that we faced during our group project to design a revision app. It gives a description, whether it was resolved (and if so how), the date the big was first reported and whether it is ongoing.

Week 9

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

SW2 Final Group Project: Revision App

Task

The self-assigned brief agreed on by the team for this final-week group project is to create an application which can help people to revise information that they are trying to learn. This could be useful for school children, university students, adult learners, autodidacts... or CodeClan students.

MVP

The ***Minimum*** Viable Product is very minimal indeed: simply to build a digital equivalent of an old-fashioned stack of flashcards, with the question and answer pairs stored in the back-end Mongo database. Only one revision topic is required for MVP, and it should be coding.

To help facilitate the extensions, the MVP should also include a home/welcome/info page, and a means of navigating between it and the flashcards which will scale to accommodate other options.

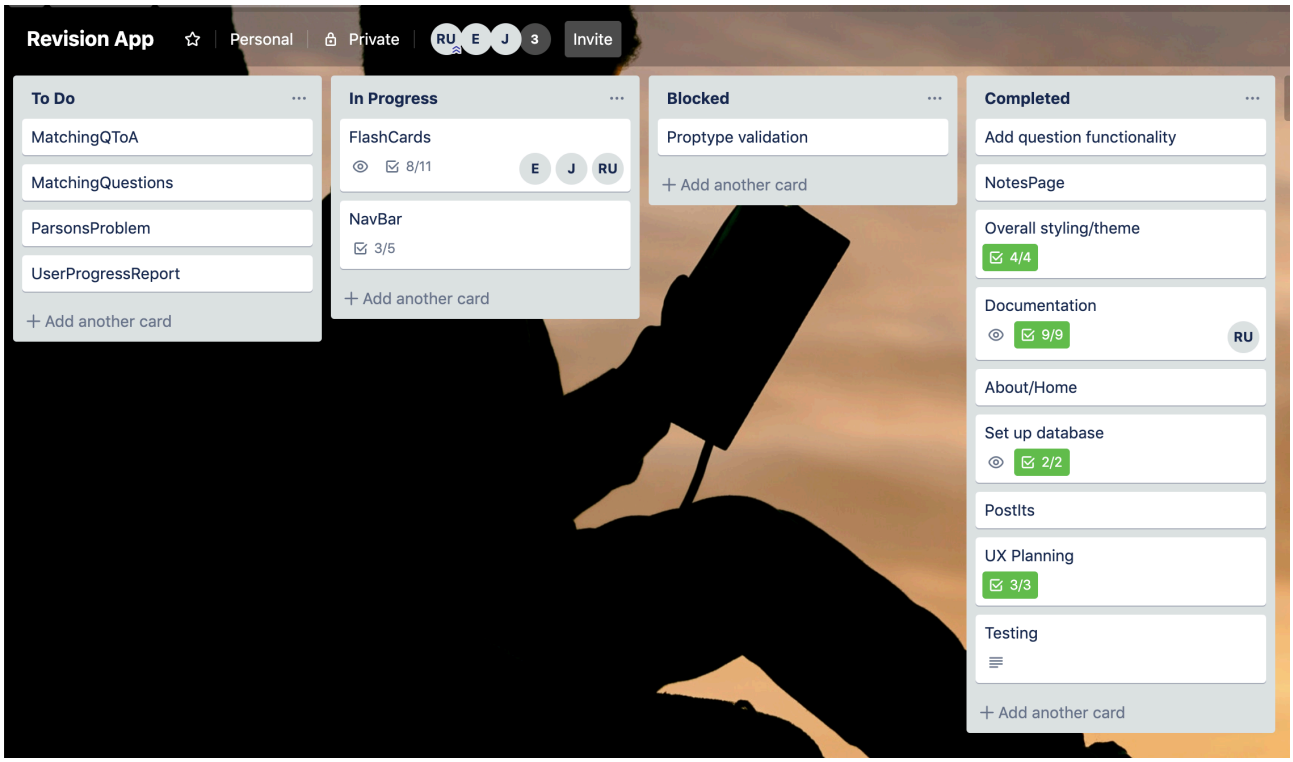
Extensions

While the MVP for this project is extremely basic, it is also hugely extensible. Possible extensions (some more easily achievable than others) are listed below, in no particular order:

- Present the flashcard learning material from the db in different ways, such as:
  - A 'pairs' game
  - Matching up multiple questions and answers
- Store learning material in other formats, which can be presented in different ways, such as:
  - Parson's Problems
  - A simple way of keeping basic study notes
  - Multiple choice Q&A's
  - Pictorially-based Q&A's
- Allow more than one topic for learning, with separate banks of Q&A for each, and a way of switching between, e.g. coding, Spanish language, dates from history, world capitals, etc., etc.
- Add a facility for users to mark question, which would enable...
- Include some form of score keeping / progress reporting
- Add spaced repetition for flashcards
- Provide a facility for users to create their own questions and answers, thus allowing:
  - Base sets of questions on a given topic
  - Personalised sets of questions
  - And ultimately, sharing of question sets between users
- Let users mark questions as 'learned', such that they no longer appear in their tests

Here is the brief which included a relatively achievable MVP whilst allowing for many different extensions. After completing the MVP early on this allowed us to determine which extensions we would like to prioritise and invest time and resource into next.

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



Here is our Trello board which we decided to group by progress. We then split each section up into smaller tasks, which were broken down further into different checklists.

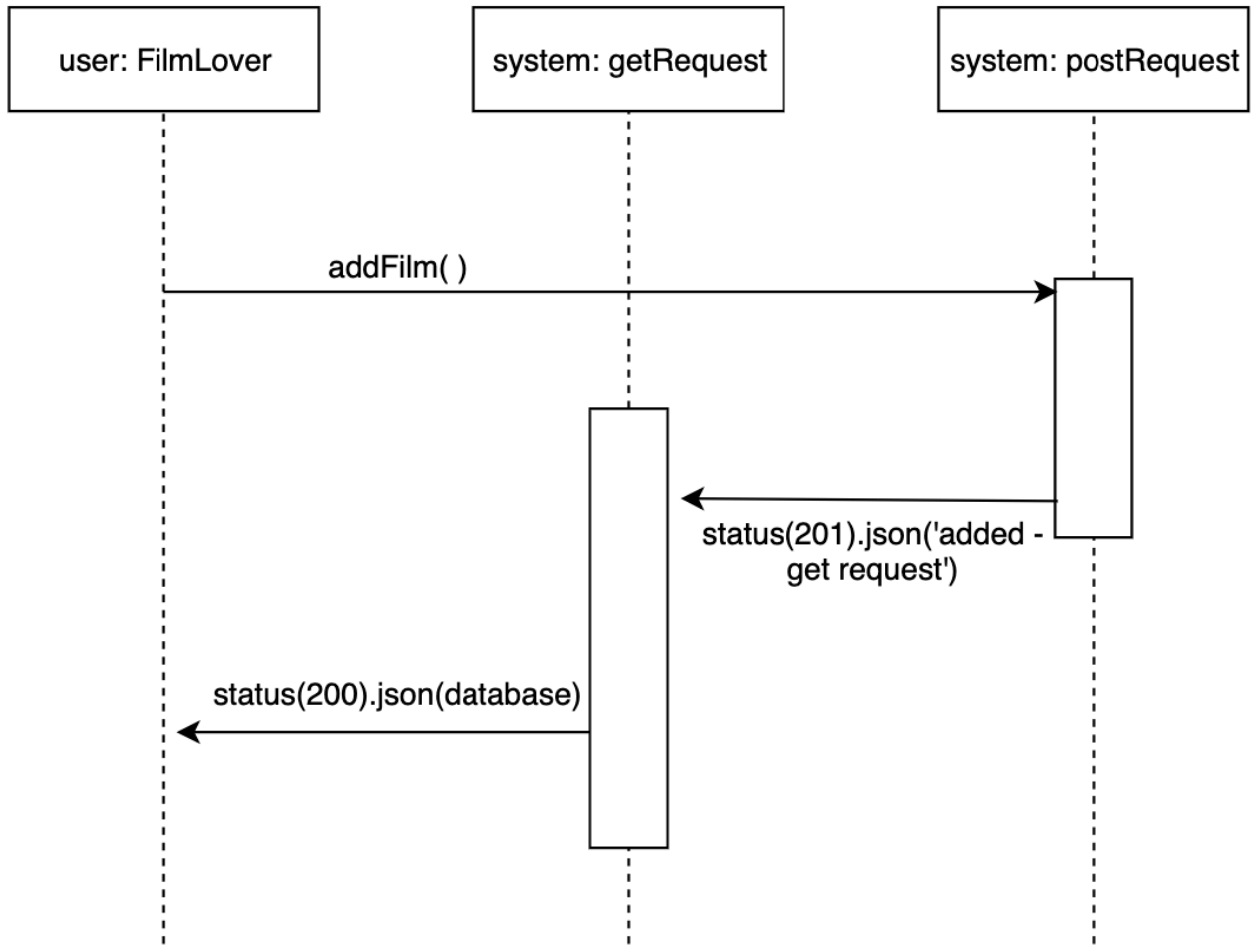
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.

Acceptance Criteria	Expected Result	Pass/Fail
User can add a film to the bucket list	When the user clicks the add button the film the database is updated to include the film, with the ‘watched’ field set to false.	Pass
User can add a film to the watched list	When the user clicks the tick button the ‘watched’ field of that film is updated to true in the database.	Pass
User can view popular films	When the user clicks the ‘Popular Films’ button on the nav bar, the top 20 films are displayed on the page	Pass

Acceptance Criteria	Expected Result	Pass/Fail
User can get more information on a film	When the user clicks the information button a modal appears that gives the films description	Pass
User can view similar films to the one they've selected	When the user clicks the more button on a film the page is rendered to display films that are similar	Pass
User is able to view their progress	When the user clicks the home button on the navbar they can see a progress bar and the number if films in each list, which is updated in the background every time they add a film.	Pass

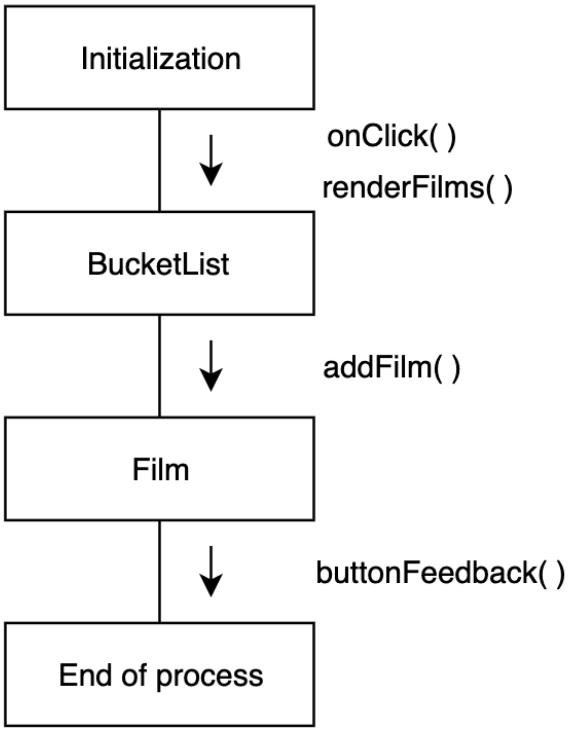
Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).

Sequence diagram

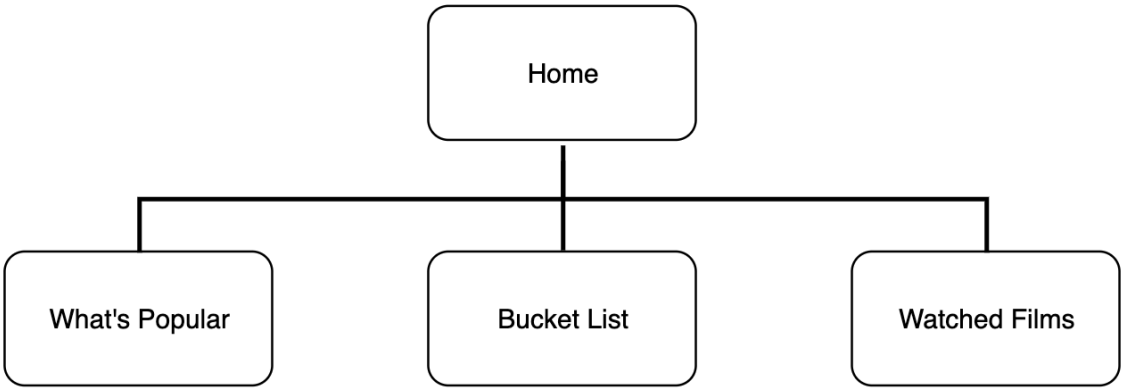


Week 10

Above is the sequence diagram for a user adding a film to their bucket list. This passes the addFilm method from the user to the system which triggers a postRequest to add the film to the database. This gives the message that it has been added and then runs getRequest to pull the data from the database, now it has been updated. The collaboration diagram to the right shows the actions of someone adding a film. The first step is an onClick where the user selects bucket list on the navigation bar, which renders the card for each film. This loads the bucket list, where a user can select one film add click the 'add' button to add the film to their watched list. This causes the button to turn green, so that the user knows it has been added successfully.



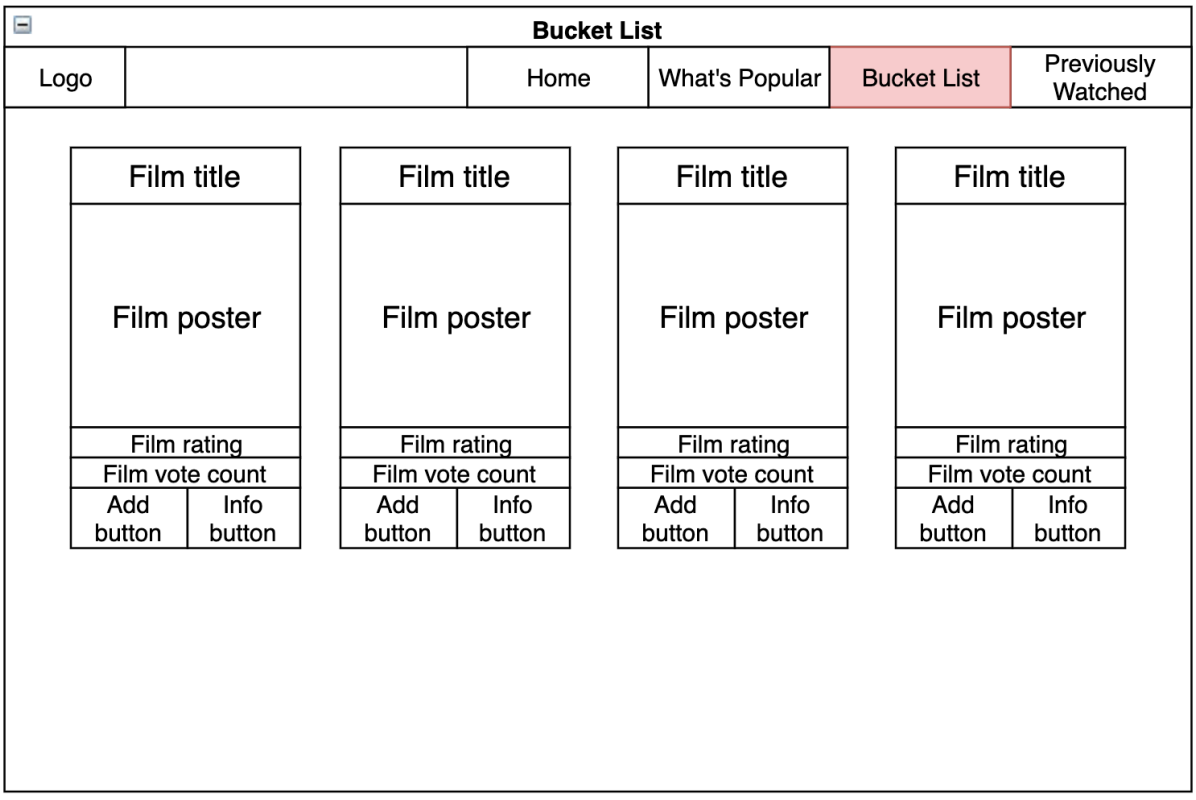
Unit	Ref	Evidence
P	P.5	User Site Map



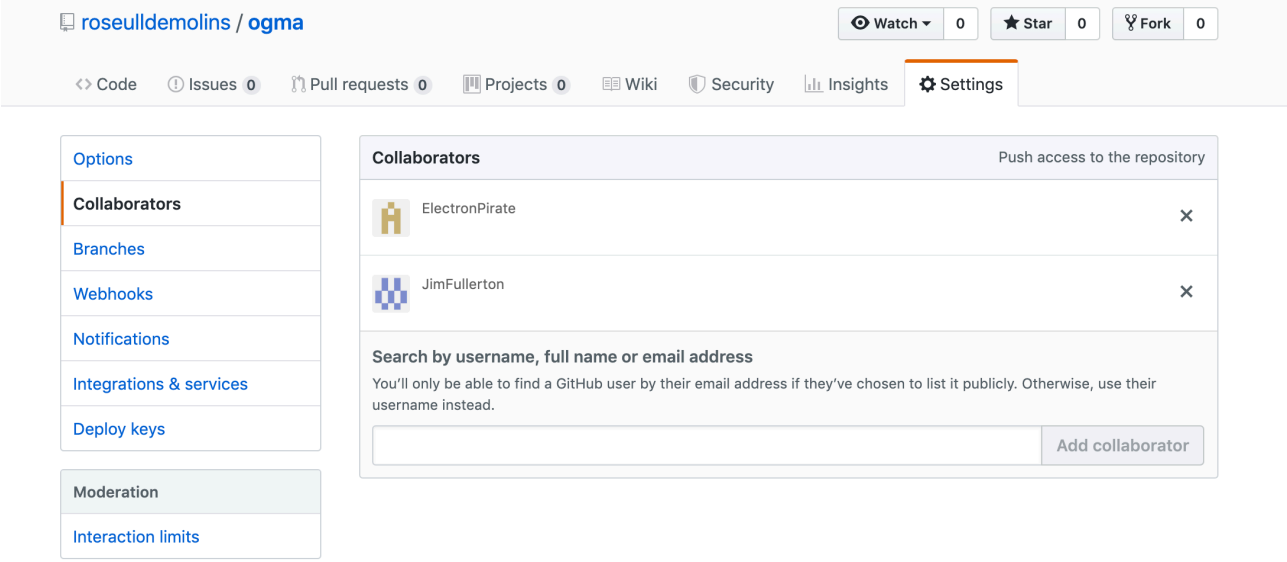
Here is my sitemap for my film list project. First it directs you to the homepage, but from here you can use the navigation bar to direct you to either the 'What's popular' page, the 'Bucket List' page or finally the 'Watched Films' page.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

Above is the wireframe of my home page for my film list application and below is the wireframe for the bucket list page.



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.



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

Drums.java | Guitar.java | IPlay.java

```
public class Drums implements IPlay{

  private String name;
  private int pricebought;
  private int pricesold;
  private int pedals;

  public Drums (String name, int pricebought, int pricesold, int pedals){
    this.name = name;
    this.pricebought = pricebought;
    this.pricesold = pricesold;
    this.pedals = pedals;
  }

  public String makeNoise(){
    return "Dum";
  }
}
```

Drums.java | Guitar.java | IPlay.java

```java
public interface IPlay {
  public String makeNoise();
}
```

Drums.java | Guitar.java | IPlay.java

```java
public class Guitar implements IPlay{

  private String name;
  private int pricebought;
  private int pricesold;
  private int strings;

  public Guitar (String name, int pricebought, int pricesold, int strings){
    this.name = name;
    this.pricebought = pricebought;
    this.pricesold = pricesold;
    this.strings = strings;
  }

  public String makeNoise(){
    return "Ding";
  }
}
```

Here are the screenshots from each of the classes in Java files - I have created a class of ‘Guitar’ and ‘Drums’. They both belong to the IPlay interface but they have different underlying data. For example the Guitar class has a number of strings, which is irrelevant for the Drums class. Similarly Drums have a humber of symbols which isn’t needed in the Guitar class. They do however both need to inherit the makeNoise function from the Play interface which is achieved using polymorphism.

Drums.java | Guitar.java | IPlay.java | Shop.java

```java
import java.util.*;

public class Shop {

  private ArrayList<IPlay> myStock;
  private String myName;

  public Shop (String name){
    this.myName = name;
    this.myStock = new ArrayList<IPlay>();
  }

  public void addStock(IPlay stock) {
    this.myStock.add(stock);
  }

  public ArrayList<IPlay> getStock(){
    return this.myStock;
  }

  public static void main(String[] args){
    Shop richersounds = new Shop("Richer Sounds");

    Guitar g = new Guitar("Guitar", 300, 600, 6);
    Drums d = new Drums("Drums", 700, 1000, 2);

    richersounds.addStock(g);
    richersounds.addStock(d);

    for (int i=0;i<richersounds.getStock().size();i++) {
      System.out.println(richersounds.getStock().get(i).makeNoise());
    }
  }
}
```