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

An example of an object literal in a program, from a small project which provided a way for users to score or rate albums:

```
1   var album = {
2     title: "Disreali Gears",
3     artist: "Cream",
4     releaseDate: 1967,
5     ukChartHigh: 5,
6     usChartHigh: 4,
7     label: "Reaction",
8     criticRating: 96
9   }
```

A function using the above object. The ***changeRating*** function takes a rating (from 1 to 100) as an argument and uses it to amend the ***criticRating*** property. It also logs out a confirmation that the change has taken place. In this example the new rating (97%) has been hard-coded for demonstration purposes:

```
11  function changeRating(newRating) {
12    album.criticRating = newRating;
13    console.log("Rating changed to: ", newRating);
14  }
15
16  changeRating(97);
17
```

The result of the function above running:

```
➜ code_examples git:(master) ✗ node IT6_object.js
Rating changed to:  97
➜ code_examples 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

A simple example of an array in a program, from some homework early in the course:

```
1  var coffees = [
2    { type:"Latte", price:2.65 },
3    { type:"Cappuccino", price:2.75 },
4    { type:"Flat white", price:2.95 },
5    { type:"Mocha Deluxe", price:3.79 }
6  ];
```

A function using the above array. This iterates over the array, using a ***for / of*** loop, then formats and logs out information about each of the items in the array:

```
8  function priceLog() {
9    for (coffee of coffees) {
10     console.log(coffee.type + "s cost £" + coffee.price + " each");
11   }
12 }
13
14 priceLog()
15
```

The result of the function above running:

```
➜ code_examples git:(master) ✗ node IT5_array.js
Lattes cost £2.65 each
Cappuccinos cost £2.75 each
Flat whites cost £2.95 each
Mocha Deluxes cost £3.79 each
➜ code_examples git:(master) ✗
```

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

Some faulty test code. This uses the ***jest*** framework, and is meant to check that a small calculator app correctly adds two single-digit numbers (1+4 - the test description is on line 10):

```
2   var Calculator = require('../../public/js/calculator.js')
3   var assert = require('assert')
4
5   describe('calculator', function () {
6     beforeEach(function () {
7       calculator = new Calculator()
8     });
9
10    test('adds 1 plus 4', () => {
11      calculator.previousTotal = 1;
12      calculator.add(4);
13      expect(calculator.runningTotal).toBe(6);
14    });
```

This is the console output from running the above test. The log shows that it failed because the expected result (which had been incorrectly stated as 6) did not match the actual result (5):

```
> jest

 FAIL  tests/unit/calculator_spec.test.js
  calculator
    ✕ adds 1 plus 4 (6ms)

  ● calculator › adds 1 plus 4

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

    Expected: 6
    Received: 5

      11 |     calculator.previousTotal = 1;
      12 |     calculator.add(4);
    > 13 |     expect(calculator.runningTotal).toBe(6);
         |                                     ^
      14 |   });
      15 |
      16 |   // test('subtracts 4 from 7', () => {

      at Object.toBe (tests/unit/calculator_spec.test.js:13:37)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        1.042s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
➜  js_calculator_start_point
```

This version of the test code has been corrected, so that the expected result (in line 13) is now 5:

```
2   var Calculator = require('../../public/js/calculator.js')
3   var assert = require('assert')
4
5   describe('calculator', function () {
6     beforeEach(function () {
7       calculator = new Calculator()
8     });
9
10    test('adds 1 plus 4', () => {
11      calculator.previousTotal = 1;
12      calculator.add(4);
13      expect(calculator.runningTotal).toBe(5);
14    });
```

The console output from re-running the amended test now indicates that it has passed successfully:

```
> jest

 PASS  tests/unit/calculator_spec.test.js
  calculator
    ✓ adds 1 plus 4 (3ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.996s, estimated 1s
Ran all test suites.
➜  js_calculator_start_point
```

All the above examples show only a single test. This is for the sake of clarity, since testing would ordinarily involve running a whole suite of such tests, covering all functions of the software, and a comprehensive range of test cases for each.

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

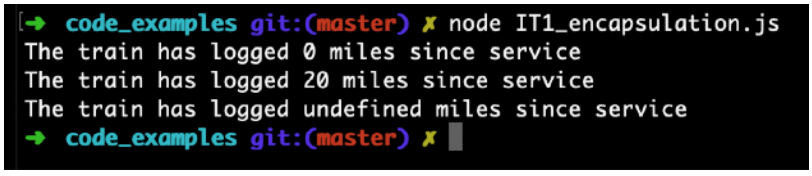
This screenshot shows the use of encapsulation within a program:

```
2   class Train {
3     constructor() {
4       var serviceMileage = 0;
5       this.getMileage = function() {
6         return serviceMileage;
7       };
8       this.setMileage = function(journeyLength) {
9         if (journeyLength > 0){
10          serviceMileage += journeyLength;
11        };
12      };
13    }
14  }
```

The next screenshot shows code to demonstrate the encapsulation shown above:

```
16  var currentTrain = new Train();
17
18  console.log(`The train has logged ${currentTrain.getMileage()} miles since service`); //returns 0
19  currentTrain.setMileage(20);
20  console.log(`The train has logged ${currentTrain.getMileage()} miles since service`); //returns 20
21  console.log(`The train has logged ${currentTrain.serviceMileage} miles since service`); // returns undefined
22
```

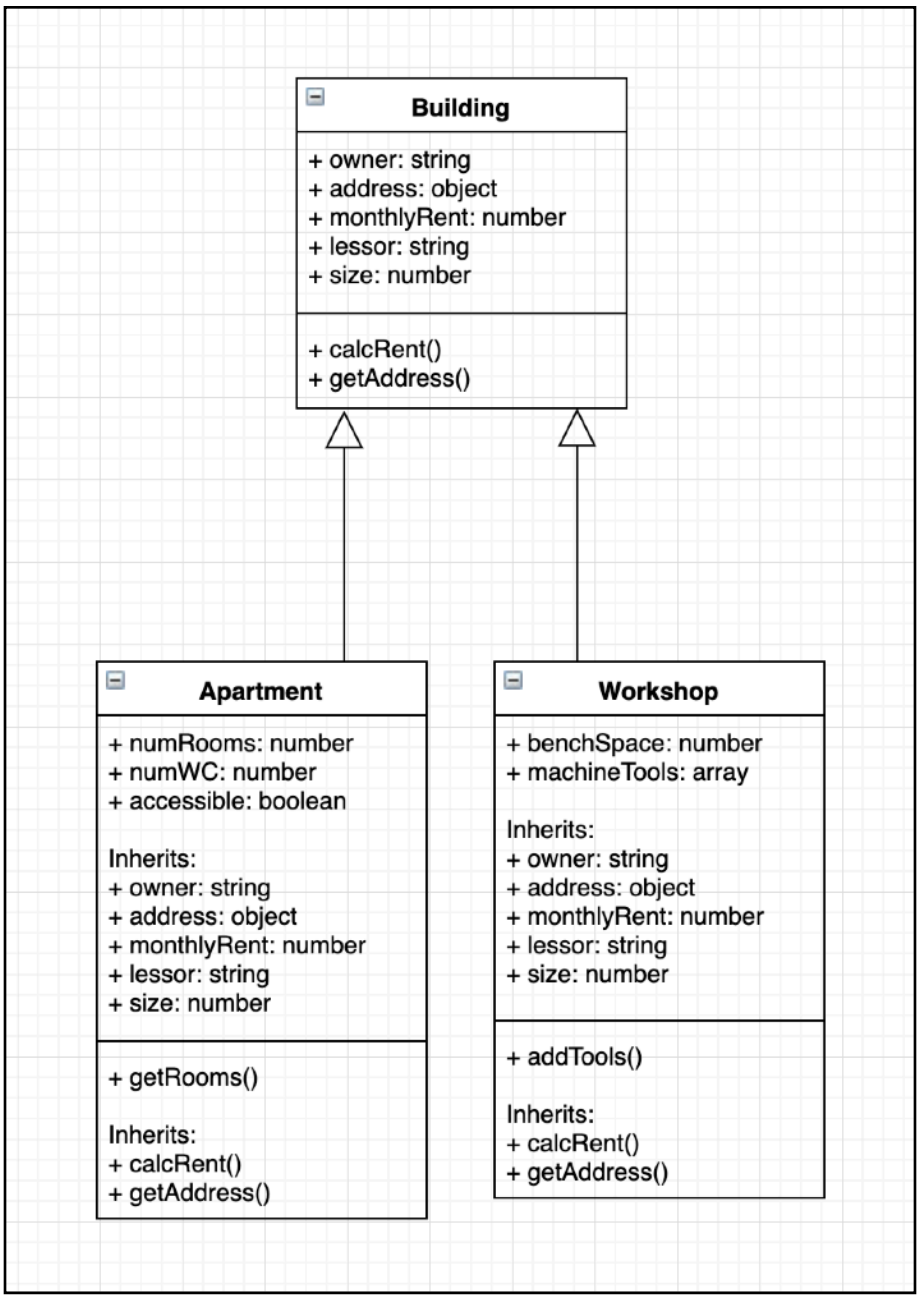
The final screenshot shows the result of running the above code:



```
➜  code_examples git:(master) ✗ node IT1_encapsulation.js
The train has logged 0 miles since service
The train has logged 20 miles since service
The train has logged undefined miles since service
➜  code_examples git:(master) ✗
```

Note that the ‘serviceMileage’ variable in the Train class is accessible only by using the getter and setter functions within the constructor, thus the first 2 console logs successfully retrieve the mileage, which is successfully amended in between the calls. However, the last console log shows ‘undefined’ as a result of attempting to read the variable directly.

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



In this inheritance diagram all buildings have a common set of properties and methods, which are inherited by each different type of building, such as the apartment or workshop shown. Each building type also has its own methods and properties in addition to those inherited from their parent class.

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.

The examples in this section refer to the inheritance diagram from the section immediately above. First, there is a screenshot showing the definition of the parent class ***Building***:

```
class Building {
  constructor(owner, address, monthlyRent, lessor, size) {
    this.building = {
      owner: owner,
      address: address,
      monthlyRent: monthlyRent,
      lessor: lessor,
      size: size
    }
  }
  calcRent(period) {
    console.log("Calculated rent for the period: ", period * this.building.monthlyRent);
  }
  getAddress() {
    console.log("Address: ", this.building.address);
  }
}

module.exports = Building;
```

Then, the next screen shows the definition of a child class, ***Apartment***. This is followed by code which instantiates a new Apartment object, after which two methods are called, one of which is inherited from the parent class:

```
const Building = require('./Building.js');

class Apartment extends Building {
  constructor(owner, address, monthlyRent, lessor, size, numRooms, numWC, accessible) {
    super(owner, address, monthlyRent, lessor, size)
    this.numRooms = numRooms,
    this.numWC = numWC,
    this.accessible = accessible
  }
  getRooms() {
    console.log(`The apartment has ${this.numRooms} rooms and ${this.numWC} bathrooms`);
  }
}

let apartment = new Apartment(
  "Propert Investors",
  "22 High Street",
  750,
  "John Smith",
  290,
  3,
  1,
  false
);

apartment.getRooms();
apartment.getAddress();
```

Finally, this output shows the logs resulting from running the code as described above:

```
➜  code_examples git:(master) ✗ node Apartment.js
The apartment has 3 rooms and 1 bathrooms
Address:  22 High Street
➜  code_examples git:(master) ✗
```

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.

Algorithm No. 1 - An algorithm for a game of 'scissors/paper/stone' - in particular, the code between lines 21 and 27, to decide on a winner.
There are 9 possible outcomes for a round of this game: player 'A' selects 1 of 3 options, and for any of those choices player 'B' also selects 1 of 3 options.

scissors/scissors	scissors/paper	scissors/stone
paper/scissors	paper/paper	paper/stone
stone/scissors	stone/paper	stone/stone

To avoid a complicated nested if, or 9-way switch statement, I used the index of each choice in the 'plays' array (line 4) to calculate a difference (x) from which a much simpler decision can be coded, as shown in the switch statement beginning on line 22.

```
1   document.addEventListener('DOMContentLoaded', () => {
2
3     let pooterPick = null;
4     const plays = ['rock', 'paper', 'scissors'];
5     const playerOP = document.querySelector('#players-choice');
6     const pooterOP = document.querySelector('#computers-choice');
7     const resultOP = document.querySelector('#result');
8
9     function randomPlay(plays) {
10      return plays[Math.floor(Math.random() * Math.floor(plays.length))];
11    };
12
13    function resultOut(playerPick, pooterPick, winner) {
14      playerOP.textContent = `You played: ${playerPick}`
15      pooterOP.textContent = `Computer played: ${pooterPick}`
16      resultOP.textContent = `So the result is: ${winner}`
17    };
18
19    function playGame(playerPick) {
20      pooterPick = randomPlay(plays);
21      let x = plays.indexOf(playerPick) - plays.indexOf(pooterPick);
22      switch (x) {
23        case -2: case 1: resultOut(playerPick, pooterPick, 'player'); break;
24        case -1: case 2: resultOut(playerPick, pooterPick, 'computer'); break;
25        default: resultOut(playerPick, pooterPick, 'draw'); break;
26      }
27    };
28
29    const rock = document.querySelector('#rock');
30    rock.addEventListener('click', () => {
31      playGame(event.target.value);
32    });
33
34    const paper = document.querySelector('#paper');
35    paper.addEventListener('click', () => {
36      playGame(event.target.value);
37    });
38
39    const scissors = document.querySelector('#scissors');
40    scissors.addEventListener('click', () => {
41      playGame(event.target.value);
42    });
43
44  });
```

Algorithm No. 2 - An algorithm to populate a dropdown with items filtered according to their initial letter.

This application provided a user with the option of selecting countries of the world from a drop-down menu. The country data was obtained from an API which provided information on 250 different countries, principalities, dependencies, etc., making an uncomfortably large list from which to choose.

To minimise the size of the drop-down, I created a clickable alphabet which allowed a user to choose an initial letter. The code in the screen shot below shows the algorithm for populating the drop-down with only countries whose name begins with the chosen letter.

```
30  selectOptions() {
31    if (this.state.filterLetter == null) {
32      return this.props.countries.map((country, index) => {
33        return <option value={index} key={index}>{country.name}</option>
34      });
35    } else {
36      let filteredCountries = []
37      for (var i = 0; i < this.props.countries.length; i++) {
38        let firstLetter = this.props.countries[i].name.slice(0,1)
39        if (firstLetter === this.state.filterLetter) {
40          let country = {details: this.props.countries[i], indexNum: i}
41          filteredCountries.push(country)
42        };
43      };
44      return filteredCountries.map((country) => {
45        return <option value={country.indexNum} key={country.indexNum}>{country.details.name}</option>
46      });
47    };
48  };
```

Week 4

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

In the following example, the function ***findCar*** conducts a linear search of the array ***garage*** to find and return details of a particular ***car***, given a search argument of the manufacturer's name.

```
var garage = [
  { manufacturer: "Jaguar", model: "E-Type", year: 1961, speed: 150 },
  { manufacturer: "Maserati", model: "Bora", year: 1971, speed: 171 },
  { manufacturer: "Ferrari", model: "Daytona", year: 1968, speed: 174 },
  { manufacturer: "Aston Martin", model: "DB6", year: 1965, speed: 150 },
  { manufacturer: "Lamborghini", model: "Miura", year: 1966, speed: 180 },
  { manufacturer: "Porsche", model: "924", year: 1976, speed: 142 },
  { manufacturer: "Jensen", model: "Interceptor", year: 1966, speed: 135 },
  { manufacturer: "Lotus", model: "Esprit", year: 1976, speed: 138 },
  { manufacturer: "AC", model: "Cobra 427", year: 1965, speed: 164 }
]

function findCar(car) {
  for (var i = 0; i < garage.length; i++) {
    if (garage[i].manufacturer == car) {
      return garage[i];
    }
  }
}

console.log(findCar("Lotus"));
```

This screenshot shows the successful conclusion of a search for a ***Lotus***:

```
➜  code_examples git:(master) ✗ node IT3_search.js
{ manufacturer: 'Lotus', model: 'Esprit', year: 1976, speed: 138 }
➜  code_examples git:(master) ✗
```

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

The following code is a portion of the SQL from the back end server from my week 5 project. It sorts data being retrieved from a relational database in ascending order of organisation's names:

```
6   router.get('/', function(req, res) {
7     SqlRunner.run("SELECT
      businesses.id,category_id,organi...addressline1,addressline2,addressline3,phonenumber,
      url,category FROM busi...sses INNER JOIN business_c...egories ON businesses.category_id =
      business_categories. ... ORDER BY organization ASC")
8       .then(result => {
9           res.status(200).json(result...);
10    });
11  });
```

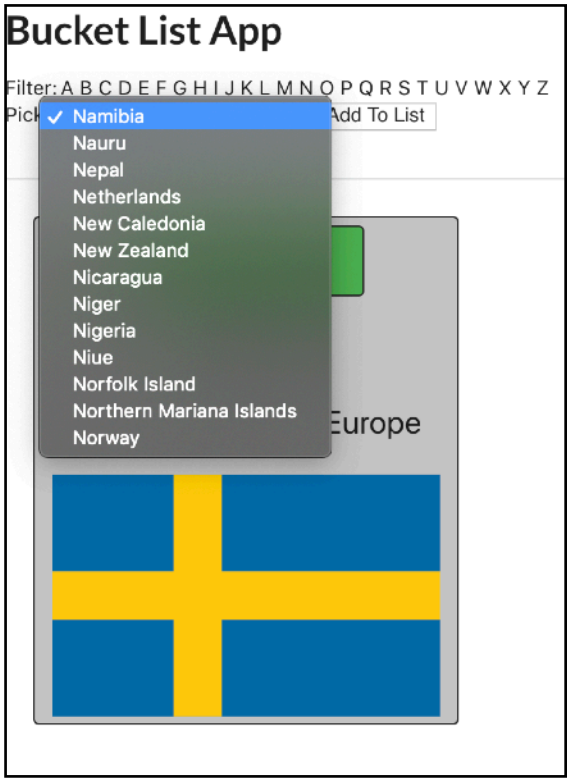
…and this screenshot shows the resulting data displayed by the application in ascending alphabetical order:

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

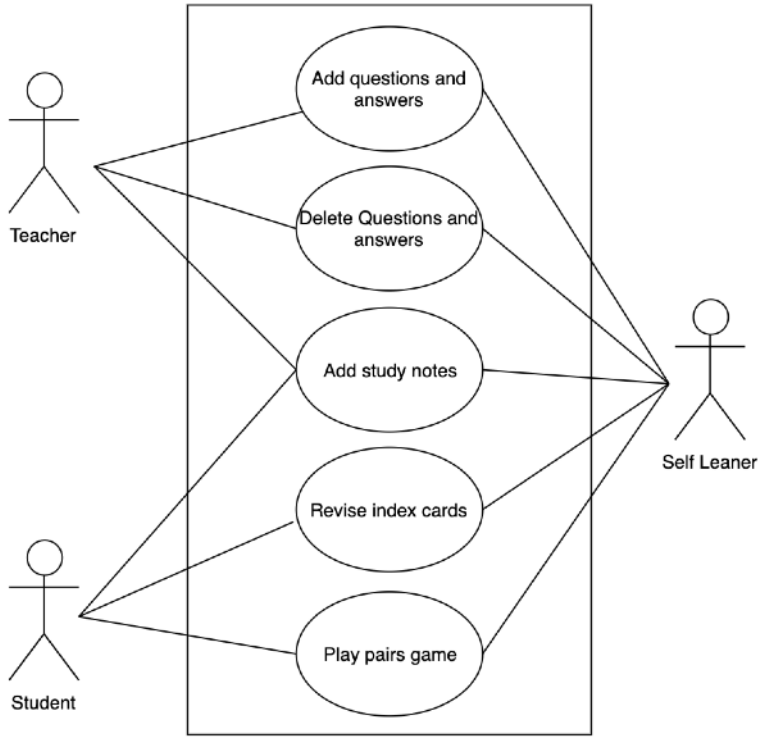
The following code calls the ***rest countries*** API, which provides statistical and geographic information about all the countries of the world:

```
const mapDispatchToProps = (dispatch) => ({
  getCountriesData() {
    dispatch(() => {
      fetch('https://restcountries.eu/rest/v2/all')
      .then(res => res.json())
      .then(countriesData => {
        console.log('in app, fetched countriesData: ', countriesData);
        dispatch({
          type: 'ADD_COUNTRIES',
          newCountries: countriesData
        })
      })
    })
  }
})
```

…and this screenshot shows a list of countries derived from the information returned by the API, filtered to include only those whose name begins with the letter N. (Reference is made to the creation of the filtered list in the section about algorithms above.)



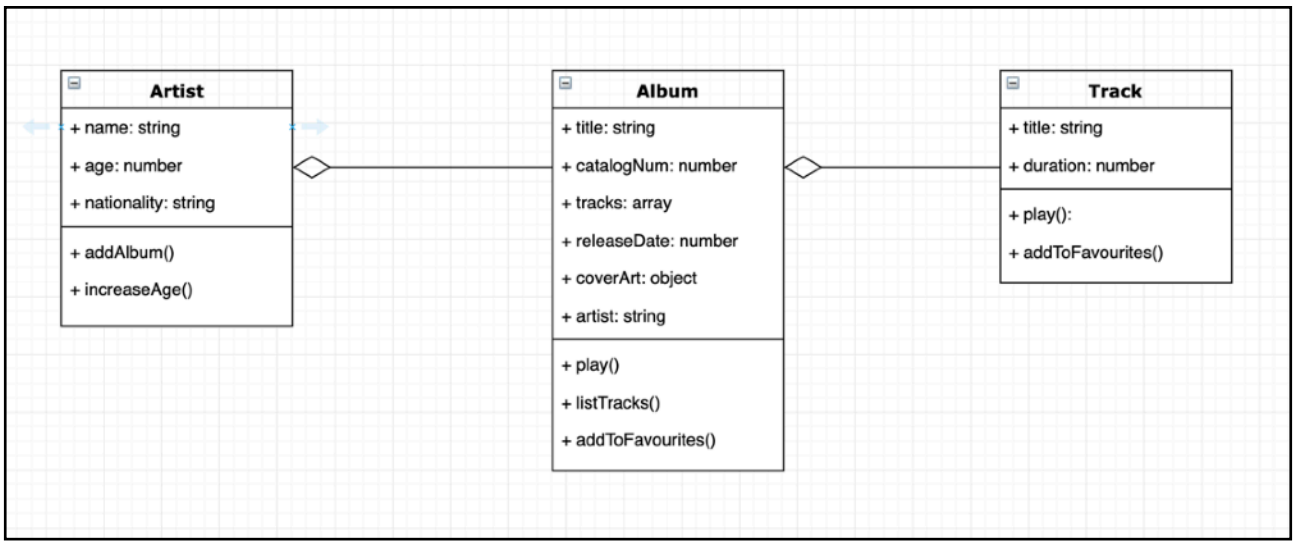
Unit	Ref	Evidence
A&D	A.D.1	Produce a Use Case Diagram



Shown above is a use case diagram for my group project from weeks 9 & 10. The diagram refers to the functionality of the application at the conclusion of the project. This is beyond the MVP, but short of the many possible extensions.

The actors all have different requirements of the system: an autodidact could potentially make use of the full range of functionality, whereas teachers / lecturers / tutors may make use of only a subset of the functionality to prepare revision questions for their pupils / students, who would use a different subset of the functions in order to access those questions.

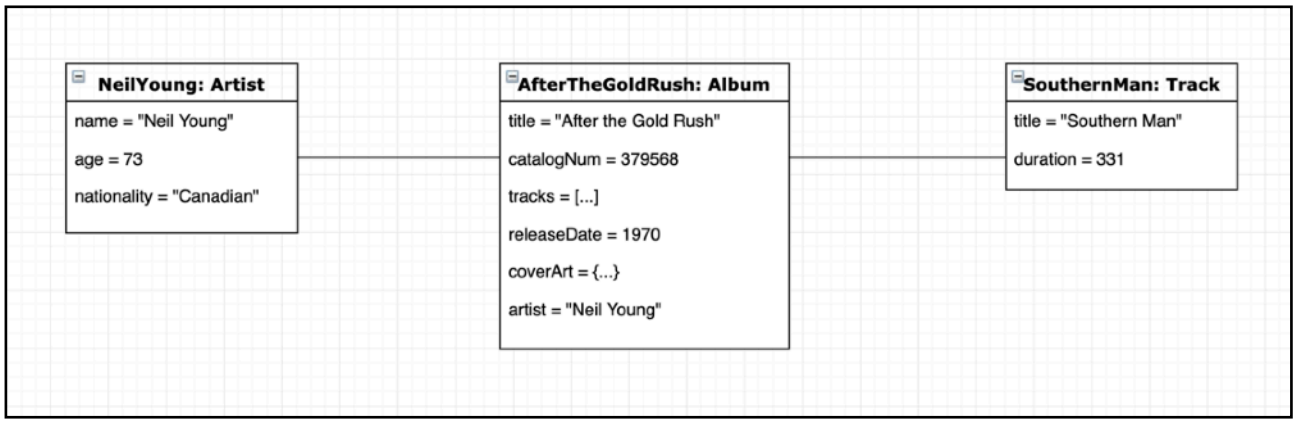
Unit	Ref	Evidence
A&D	A.D.2	Produce a Class Diagram

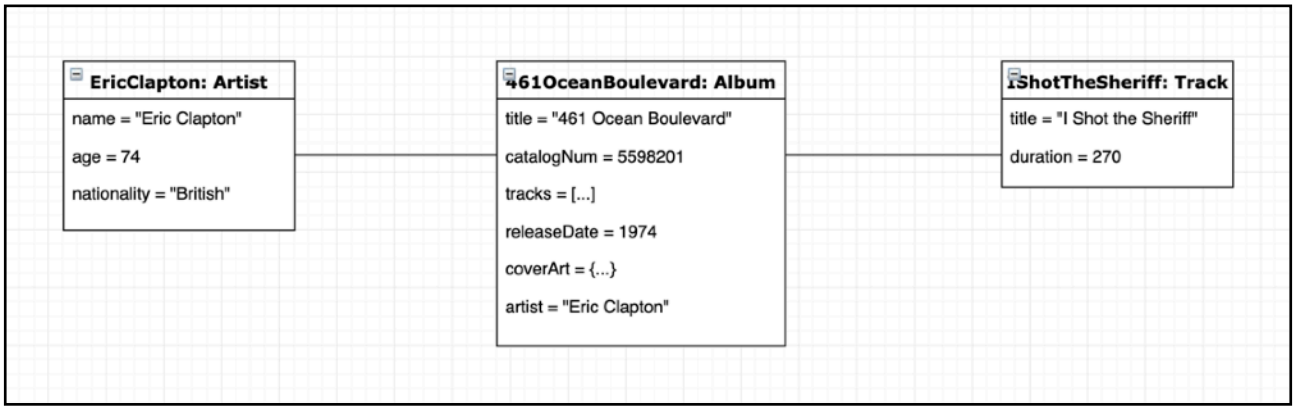


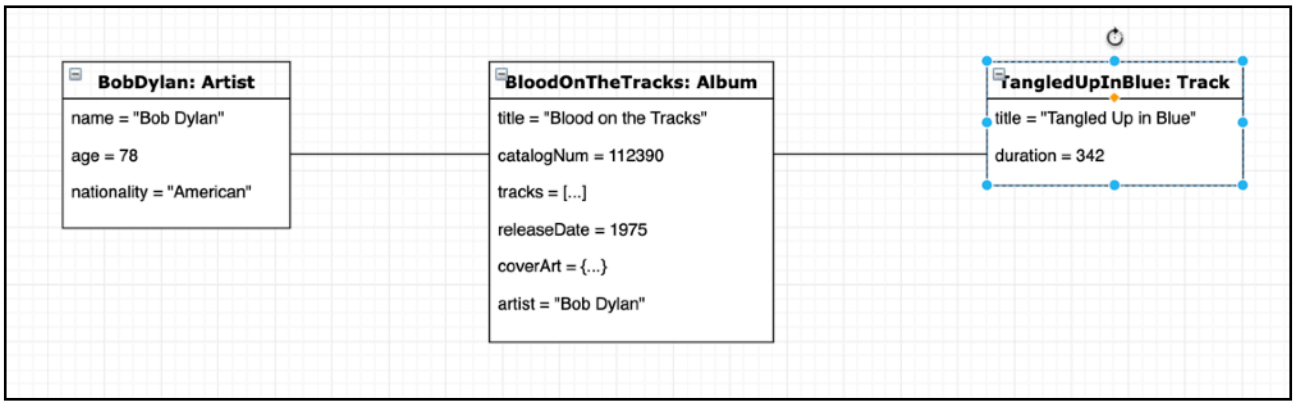
The class diagram above shows a (partial) representation of a system, designed and built during an early group project, which allowed listeners to score or rate music. It describes 3 classes of object, their attributes and methods, and the relationships between them.

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

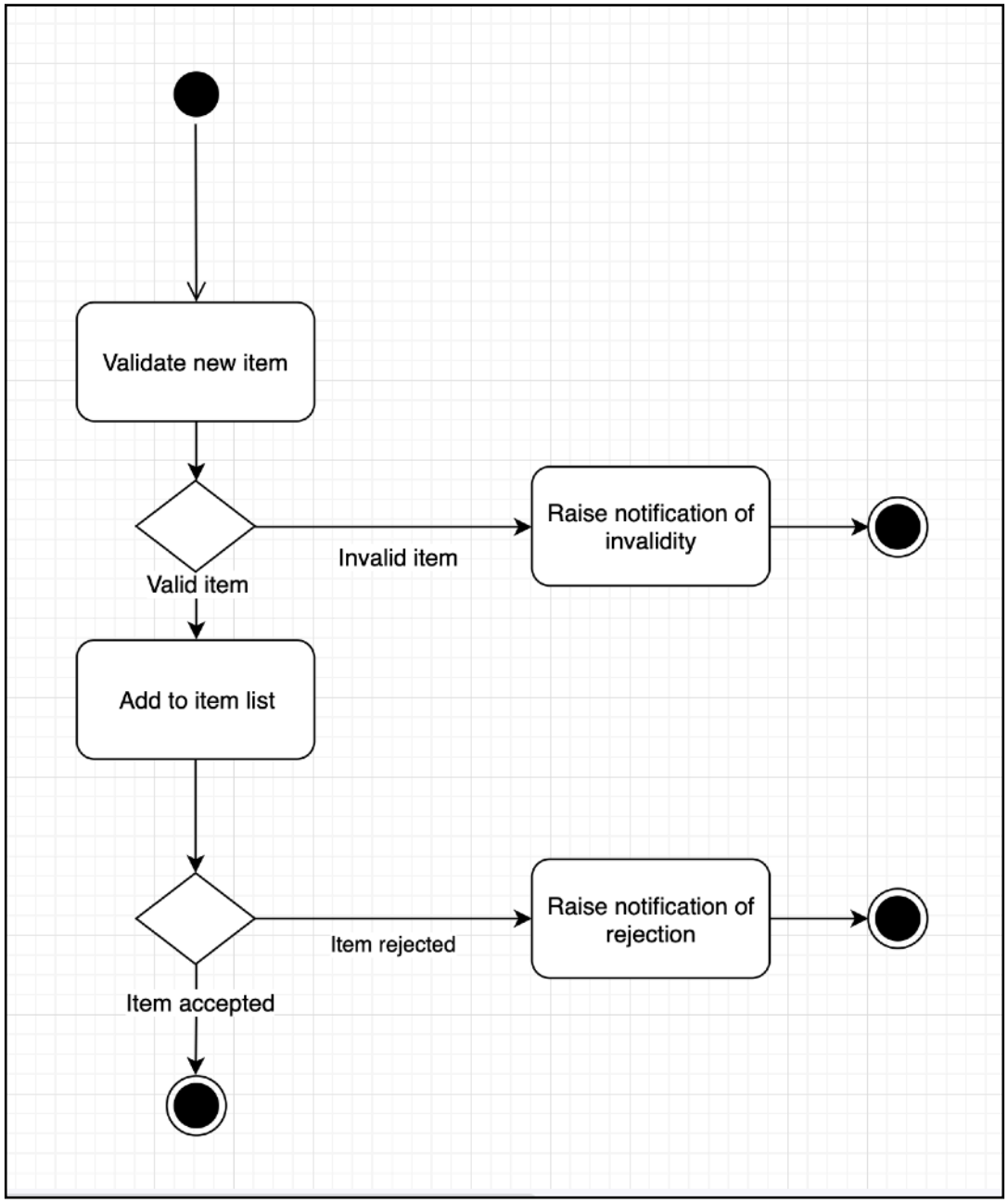
The figures below show 3 object diagrams, all of which are based on the class diagram from the preceding section.







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



Shown above is an activity diagram describing the primary flow of an operation to add a new item to a ‘To Do’ list for a personal organiser.

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	Constraint	Solution
Hardware & Software Platforms	• The application must be fully functional on all common hardware and operating system platforms. • Reduced portability will restrict the target audience.	• Build as a web application for maximum portability. • Test on a representative set of platforms.
Performance Requirements	• System responses must be quick enough to meet business requirements. • Degraded response times create poor UX. • Presence on comparison sites is a key business requirement, and such sites will not tolerate high latency.	• Consider response time at the design stage and plan accordingly. • Conduct performance testing.
Persistent Storage & Transactions	• Sufficient capacity must be available for persistent storage of all transactions. • Insufficient storage may degrade service. • Adequate data retention must be provisioned to meet legal and regulatory requirements.	• Forecast transaction volumes. • Specify sufficient storage to cope with predicted usage. • Conduct load/volume testing.
Usability	• Users must be familiar & comfortable with all interfaces and controls. • Poor UX limits usage. • International accessibility standards must be maintained.	• Deploy standardised interfaces and controls wherever possible. • Use standard UX design methods. • Engage users in design process. • Provide help files and user documentation.
Budgets	• Project must be completed within the allotted budget. • Senior management sign-off will be required for any over-run.	• Use best practise budget control techniques. • Monitor & report on progress.
Time Limitations	• Project must be built, tested and ready to go live by the implementation date. • Late delivery will impact adversely on business plans.	• Use best practise project planning techniques. • Monitor & report on progress.

Week 5

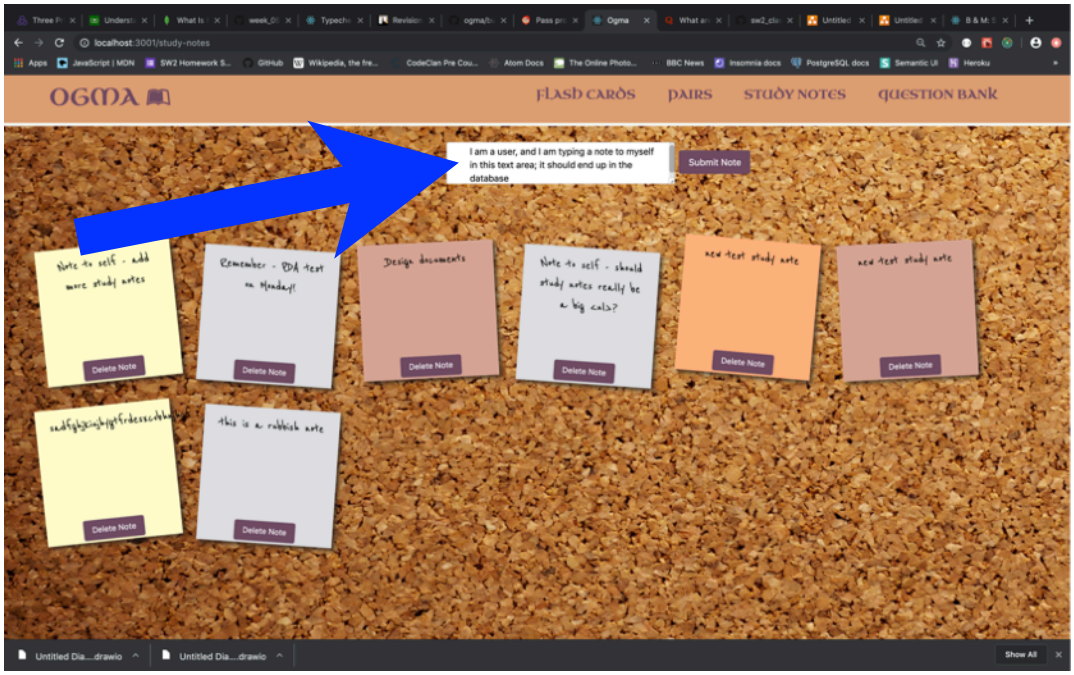
Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method

The pseudocode below describes a method used in my week 5 project to create a digital business card for display within the business directory application. The project was created in vanilla JavaScript, rather than React, which involved a few lines of code to realise each step.

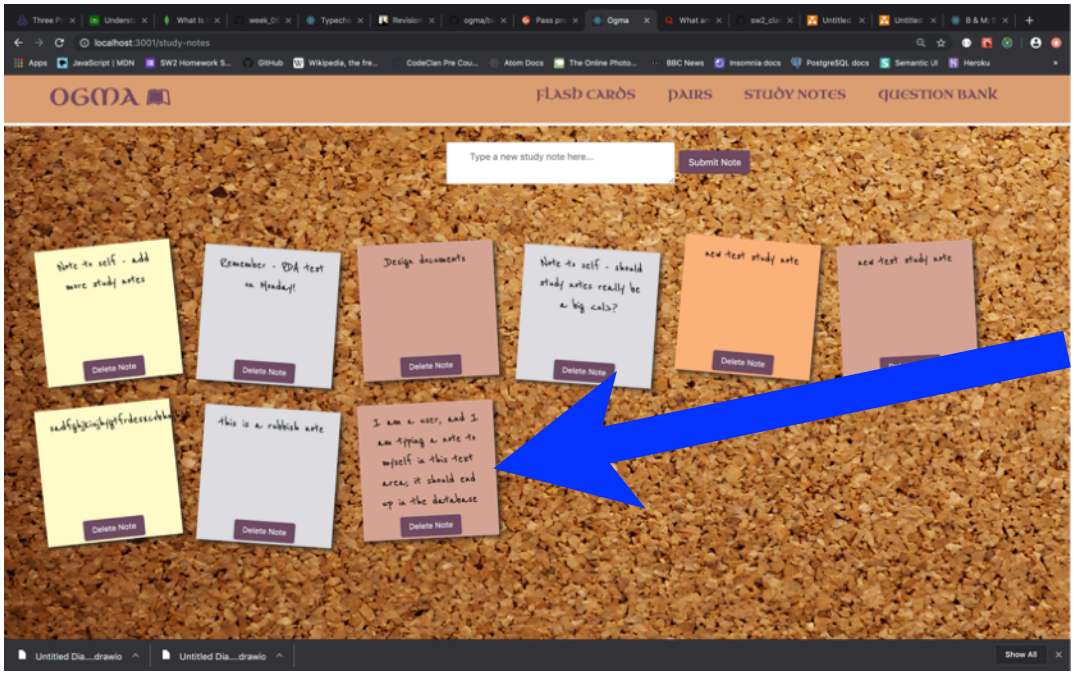
```
method createCard (takes a business object as an argument)
  create a set of HTML divs to contain each part of the business card:
    category, address, contact details, business name
  create an overarching div to contain the elements of the content
  append each of the sub-divs to the main div in turn
  return the card
end method
```

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

User input (adding a ‘study note’ to my group project learning/revision application):



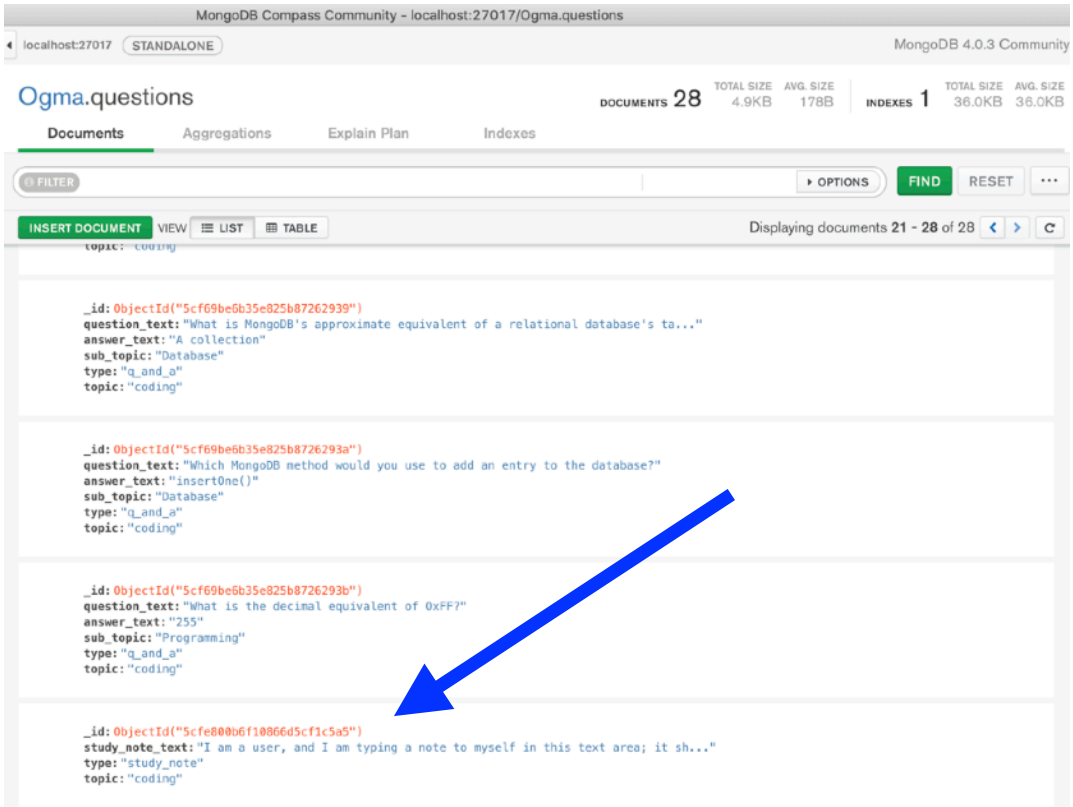
The input being used (to display the inputted note on a ‘noticeboard’.



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

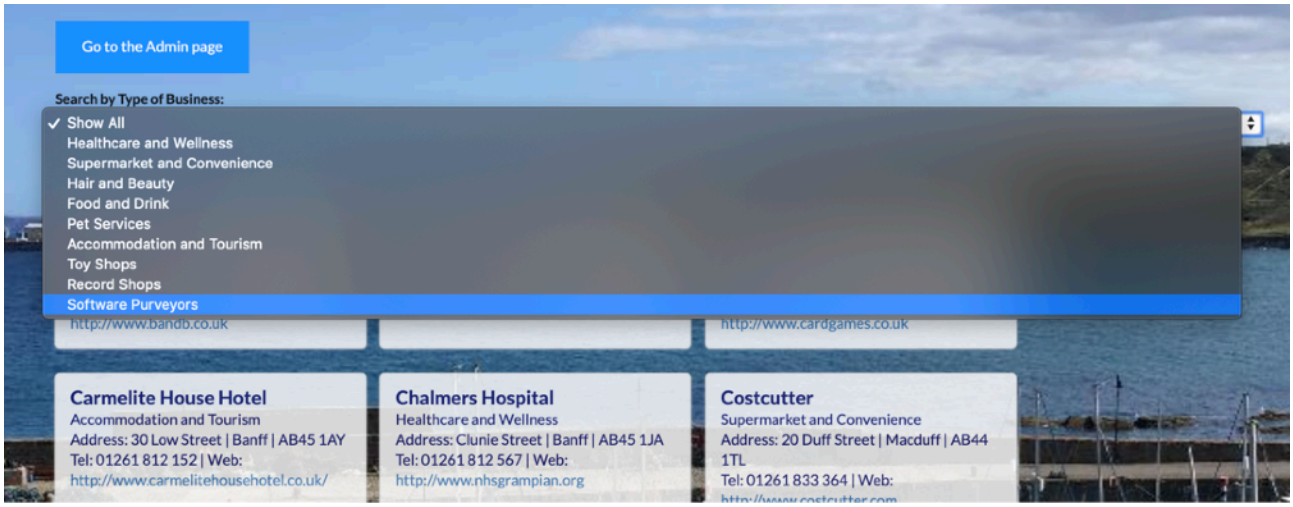
The same user input as in P.13 above:

Confirmation of the data being saved (as a document record in the back-end Mongo database):

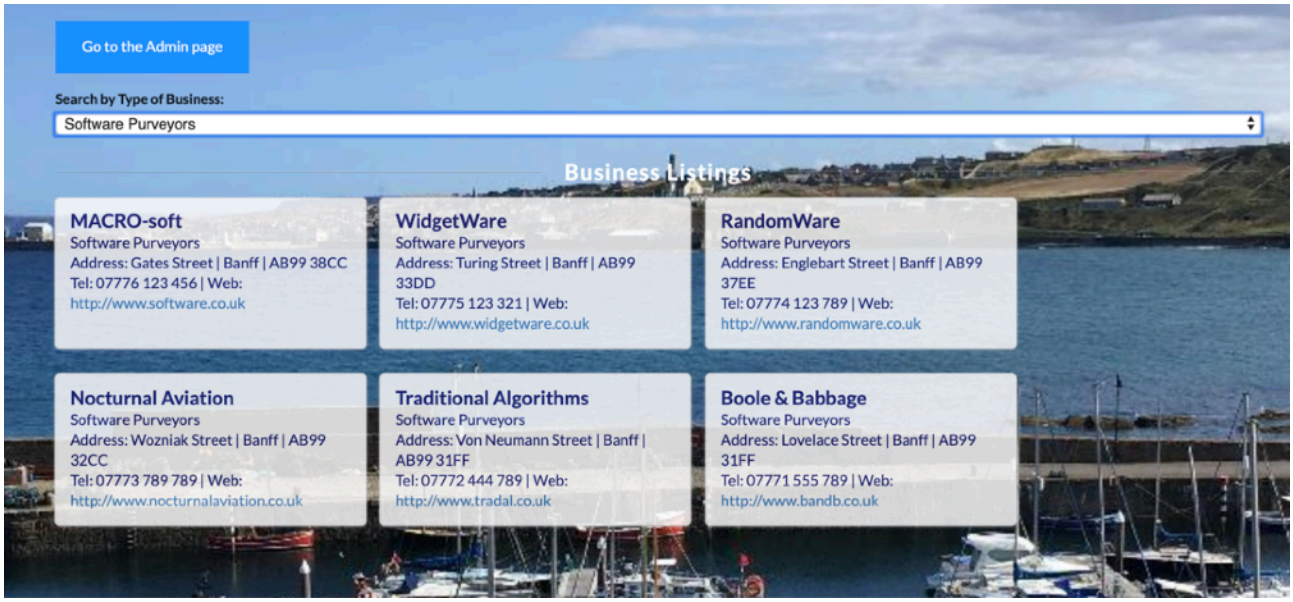


Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program

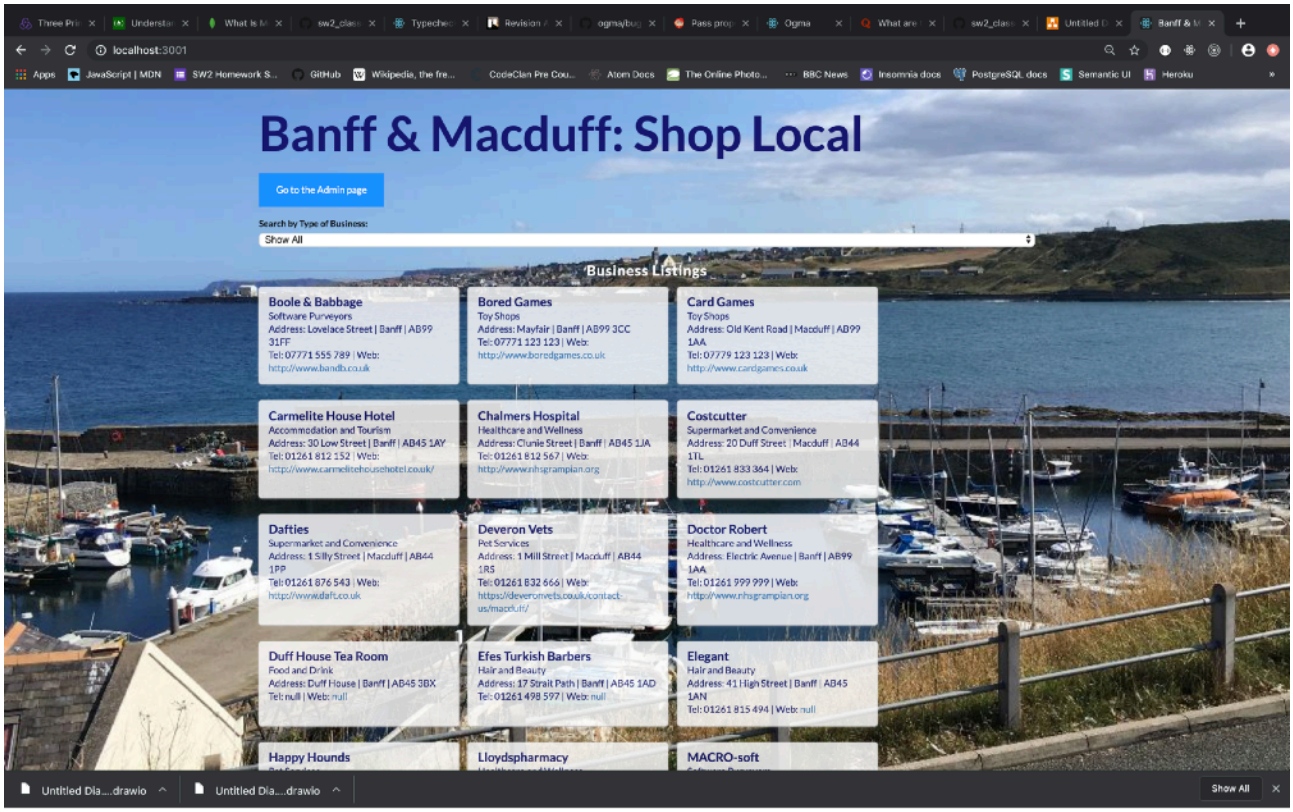
Screenshot of a user requesting an action to be performed (specifically, instructing the app to filter its display to a specific business type by selecting from a drop-down):



Screenshot of the result of the request being processed correctly and demonstrated in the program, which now shows only the chosen type of business (software developers):



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



The screenshot above is from my week 5 project which implemented a local business directory, and was completed alone.

The project's brief came from a real-world situation in which The Banff & Macduff Heritage Trail sought to create a website containing, among other things; practical information, historical features, heritage stories, and a "what's on" section, including a "shop local" feature. The "shop local" feature formed the basis of this project, and comprised a searchable display-list of digital business cards from local businesses.

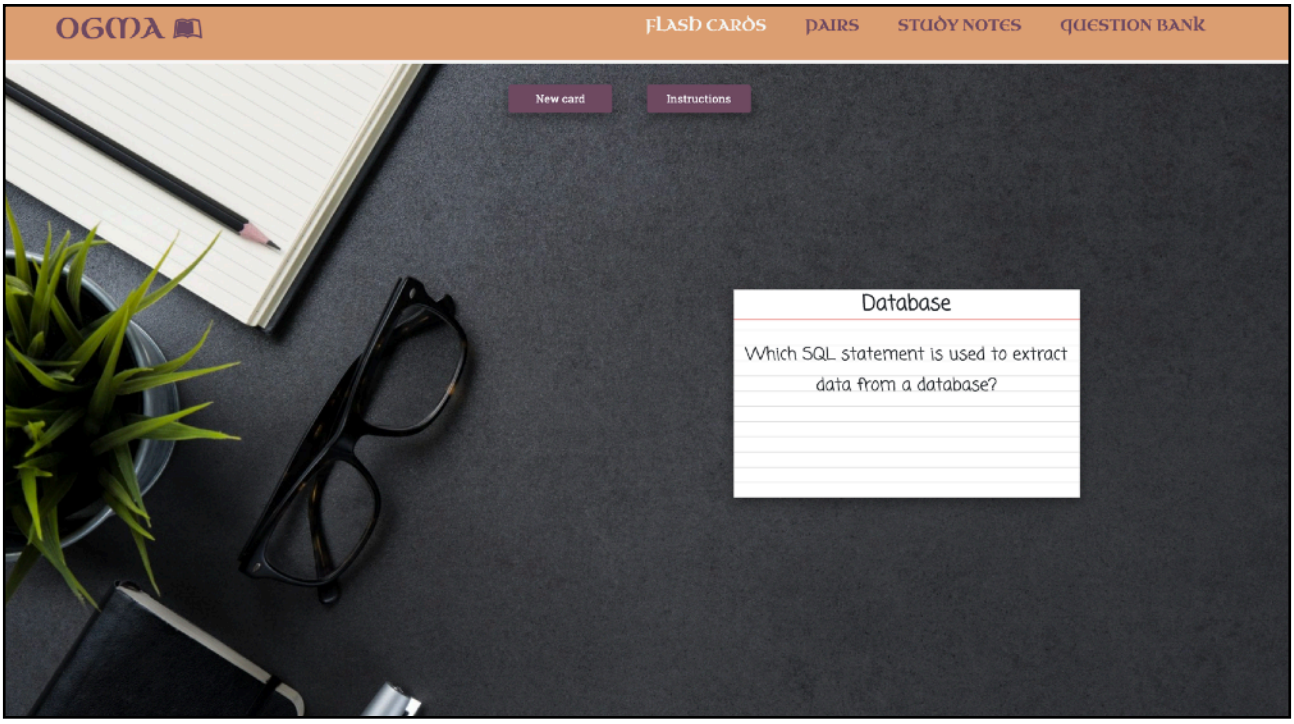
The project repository can be found on GitHub at the link below:

https://github.com/JimFullerton/week_05_project

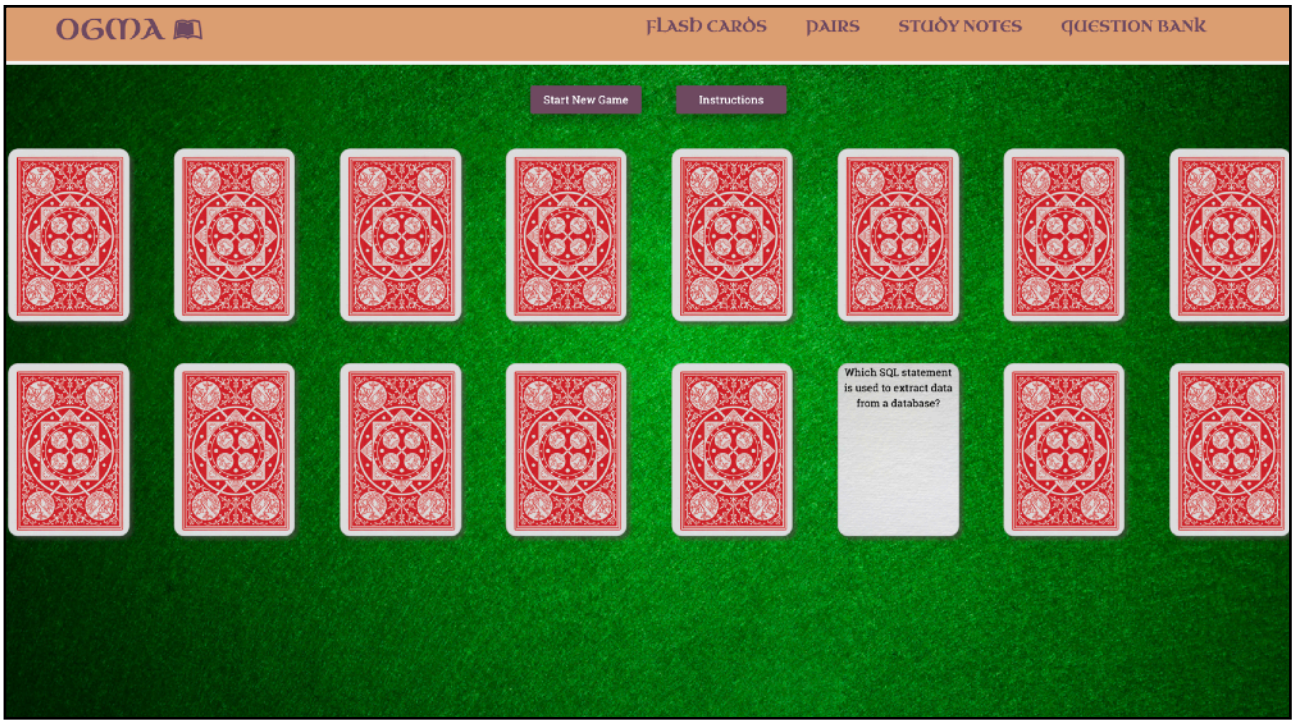
Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.

The following screenshots clearly show different stages of development in my group project from weeks 9 and 10:

MVP:

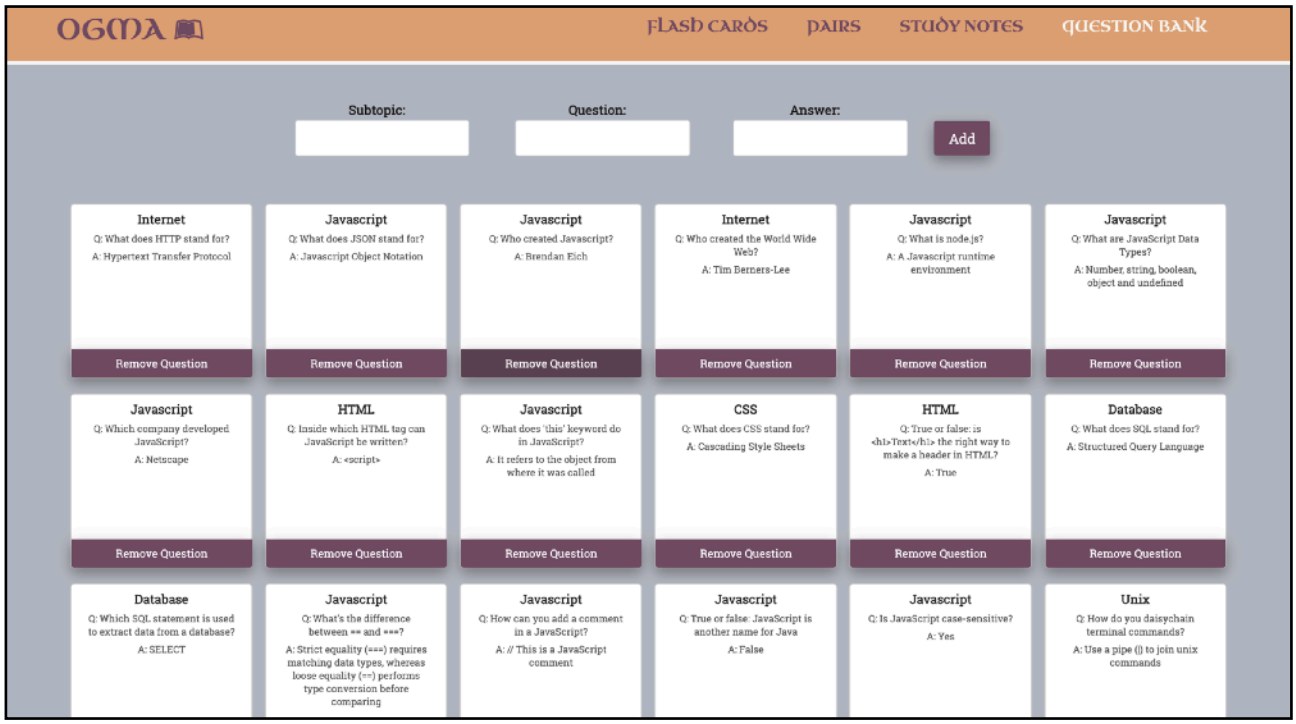


1st Extension:



2nd Extension:

3rd Extension:



Week 7

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

Revision App Bug Tracker

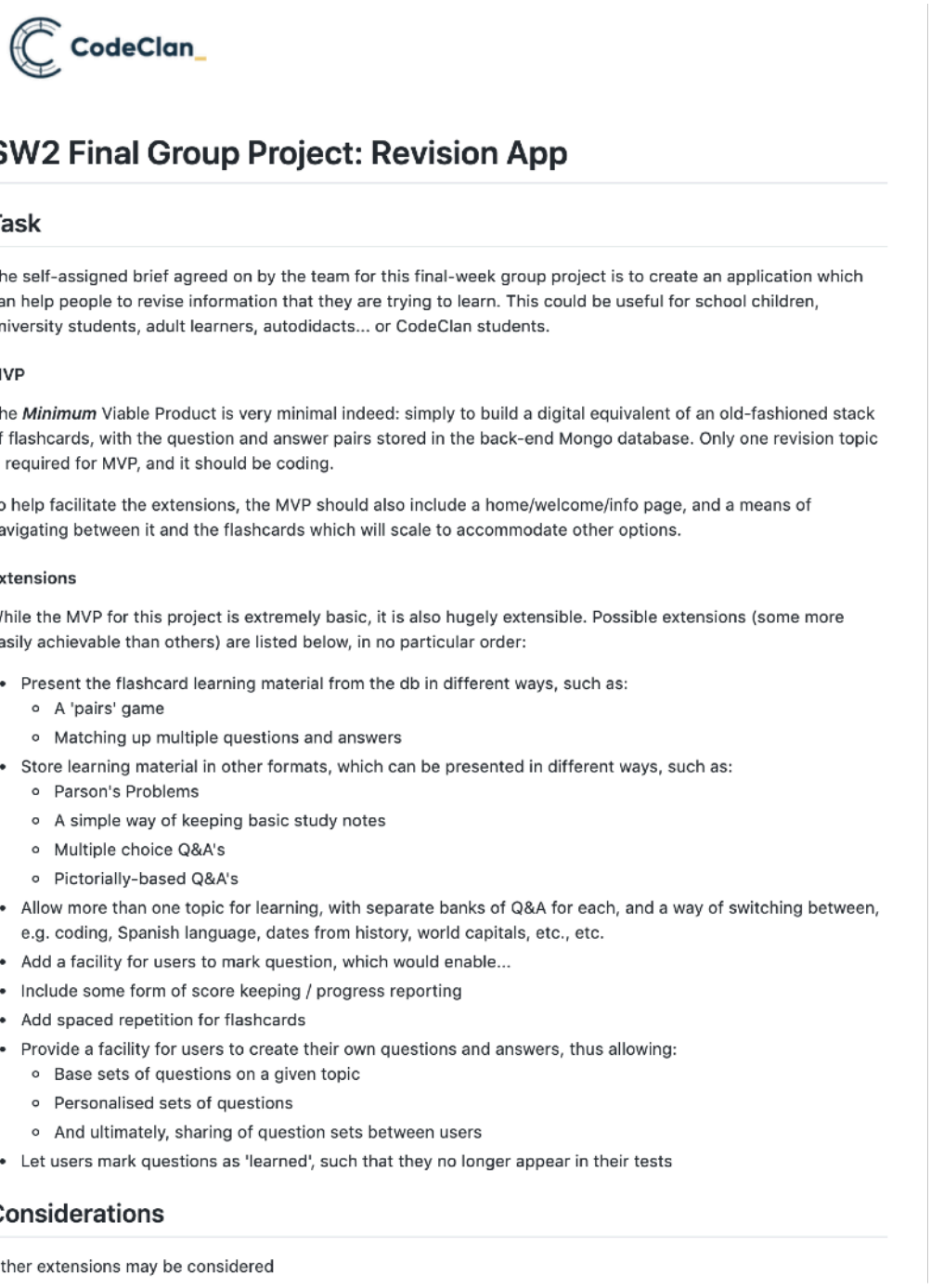
Ref. No.	Bug Description	Resolution	Date Reported	Closed
1	Type error, in reducer	Needed to initialise store as []	30/05/2019	Y
2	Type error, in FlashCard component	Added conditional not to try to render empty array	30/05/2019	Y
3	Flipping flashcard causes random re-selection of Q/A	Added conditional 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

Bug Tracker: The screenshot above is of the bug tracking report which we used on our group project. This proved useful in prioritising which (if any) bugs should be worked on at any given time.

The tracker was created and maintained by myself on behalf of the project team.

Week 9

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



CodeClan_

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

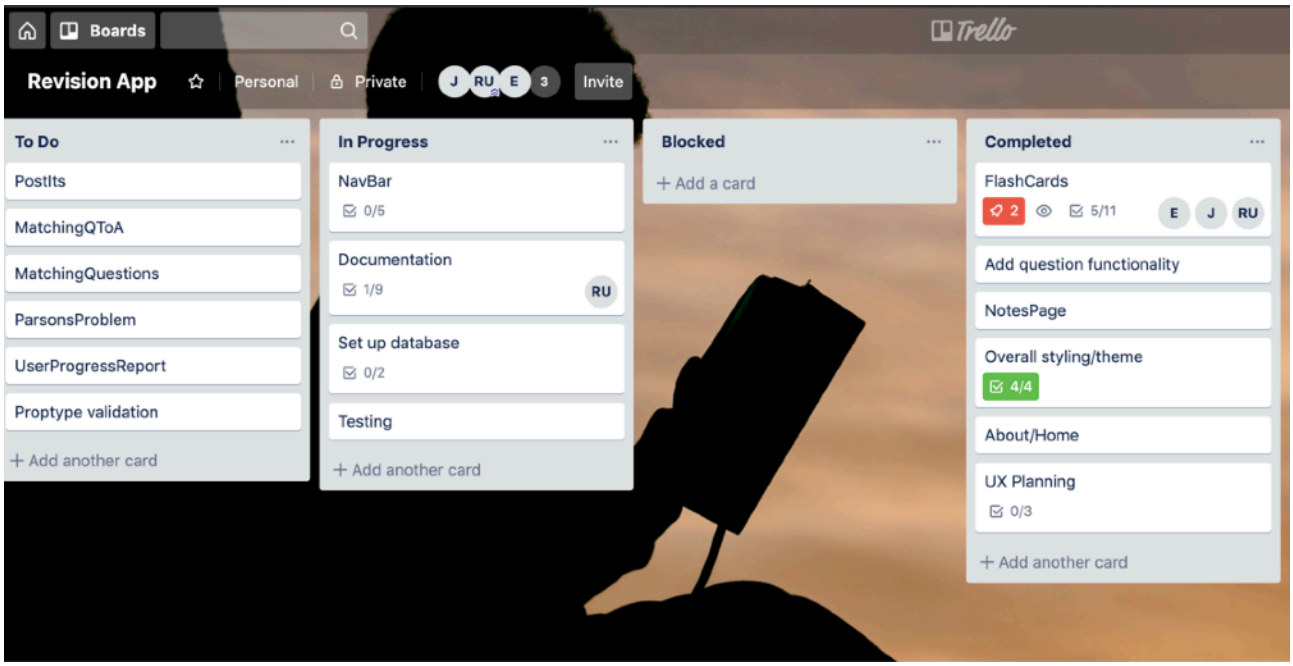
Considerations

Other extensions may be considered

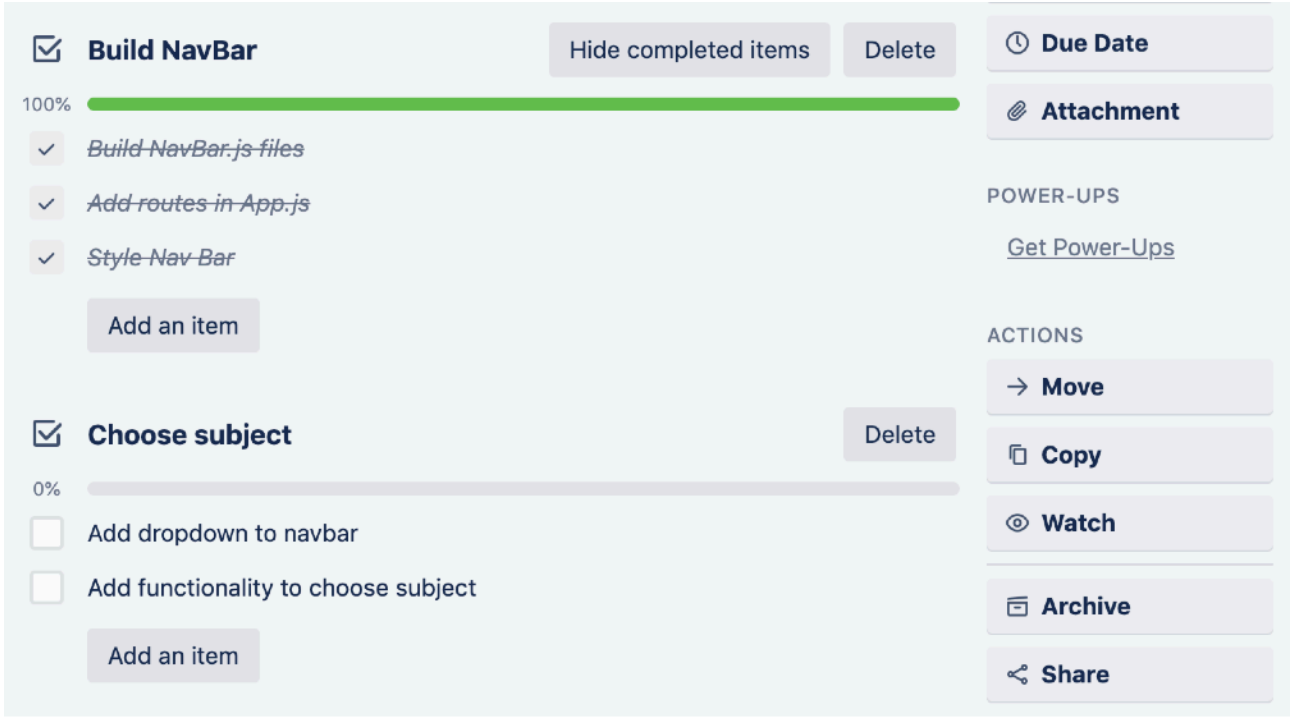
Project Brief: The screenshot above was taken from the GitHub repository for my group project from weeks 9 & 10. It shows the brief for the project, including a general outline of the proposed application, a description of the MVP, and a list of potential extensions. The list of extensions is not exhaustive, and owing to time constraints only 3 of them were completed.

The brief was created by myself on behalf of the project team.

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



Planning: The screenshot above shows the Trello board which my team created for our group projects. It may seem to lack detail, but some of the items contained sub-tasks with further information, as shown in the example below, where some sub-tasks are completed and some still outstanding:



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

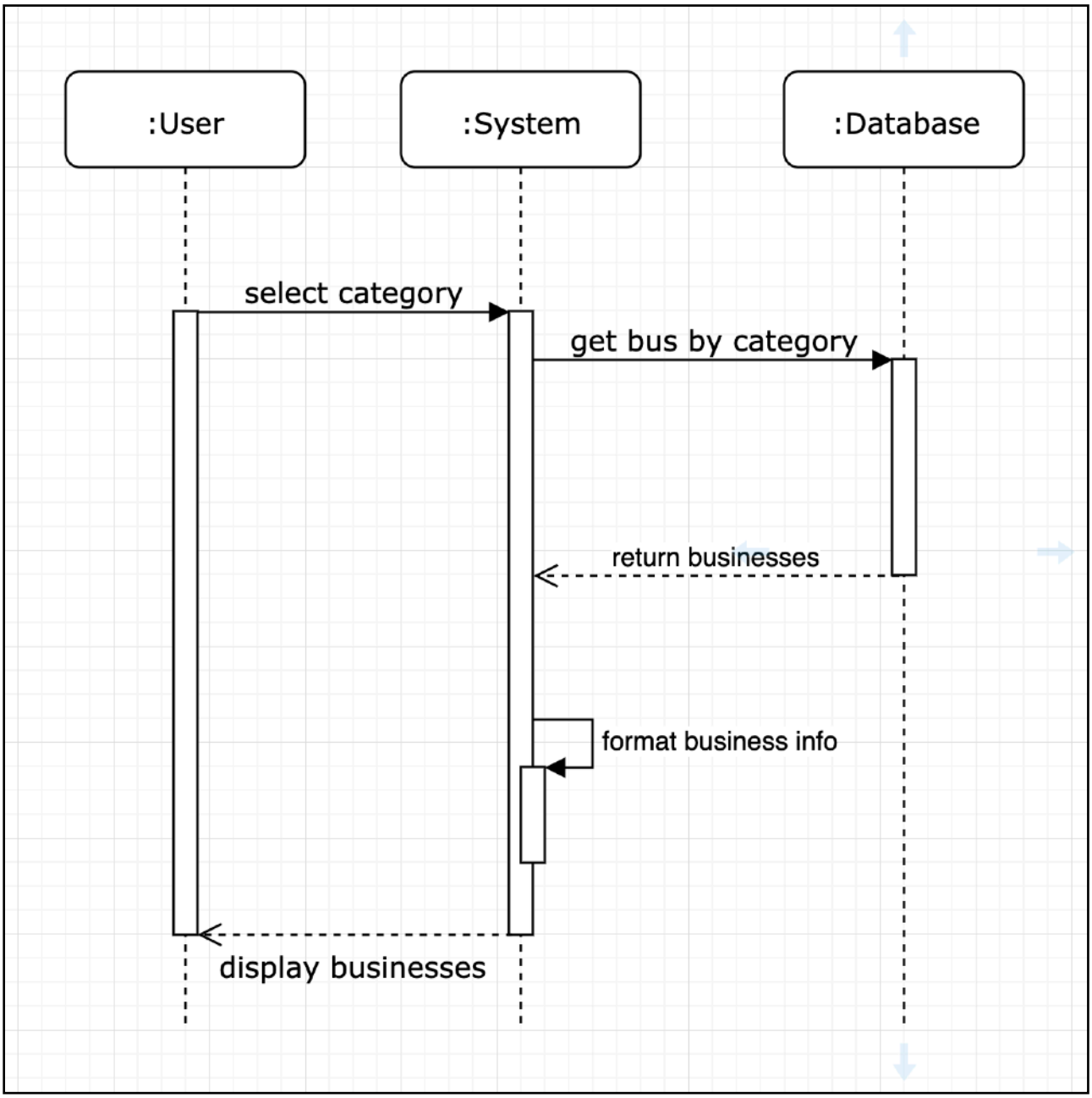
Acceptance Criteria & Test Plan

Ref	Acceptance Criterion	Expected Result	Pass / Fail
1	A user can see a list of businesses	The site shows a list of all businesses on loading	Pass
2	A user can filter businesses by category	A drop-down shows the available categories for businesses	Pass
3	A user can filter businesses by category	The display filters appropriately on selection from the drop-down	Pass
4	A user can undo the filter to return to a full listing	The drop-down has an option to select all	Pass
5	A user can undo the filter to return to a full listing	The display shows all businesses on selection of 'show all'	Pass
6	An administrator can log on to the admin page	The site re-directs to admin on clicking the 'admin' button	Pass
7	An administrator can see a list of businesses on the admin page	The admin page shows a list of all businesses on loading	Pass
8	An administrator can filter businesses by category on the admin page	A drop-down shows the available categories for businesses	Pass
9	An administrator can filter businesses by category on the admin page	The display filters appropriately on selection from the drop-down	Pass
10	An administrator can undo the filter to return to a full listing on the admin page	The drop-down has an option to select all	Pass
11	An administrator can undo the filter to return to a full listing on the admin page	The display shows all businesses on selection of 'show all'	Pass
12	An administrator can add a new category of business	Filling in & submitting the 'new category' form adds a business category to the database	Fail: not yet implemented
13	An administrator can add a new business	Filling in & submitting the 'new business' form adds a business to the database	Pass
14	An administrator can amend details for a business	Clicking 'update' takes the administrator to an update page	Fail: not yet implemented
15	An administrator can delete a business	Clicking 'delete' removes the business from the database	Pass

The screenshot above is an excerpt from an acceptance criteria & test plan document relating to my week 5 solo project, referred to in several sections above. The application's user interface was relatively simple, but still generated an abundance of acceptance criteria and their associated tests. Failed tests relate to project extensions which were not implemented owing to time constraints.

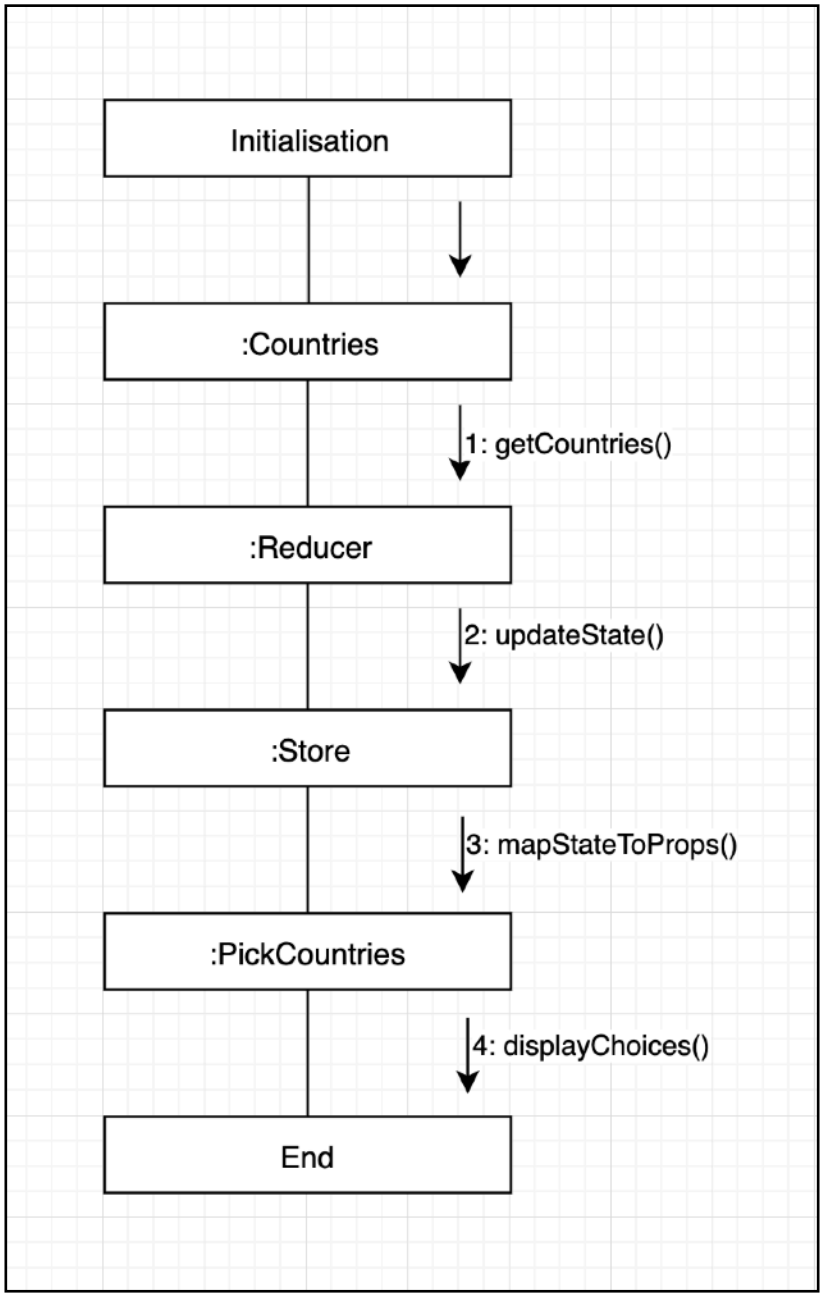
Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).

The sequence diagram below relates to my week 5 project, and describes the behaviour of different parts of the system, and the messages passing between them. Order is implicit in the flow from top to bottom of the page.



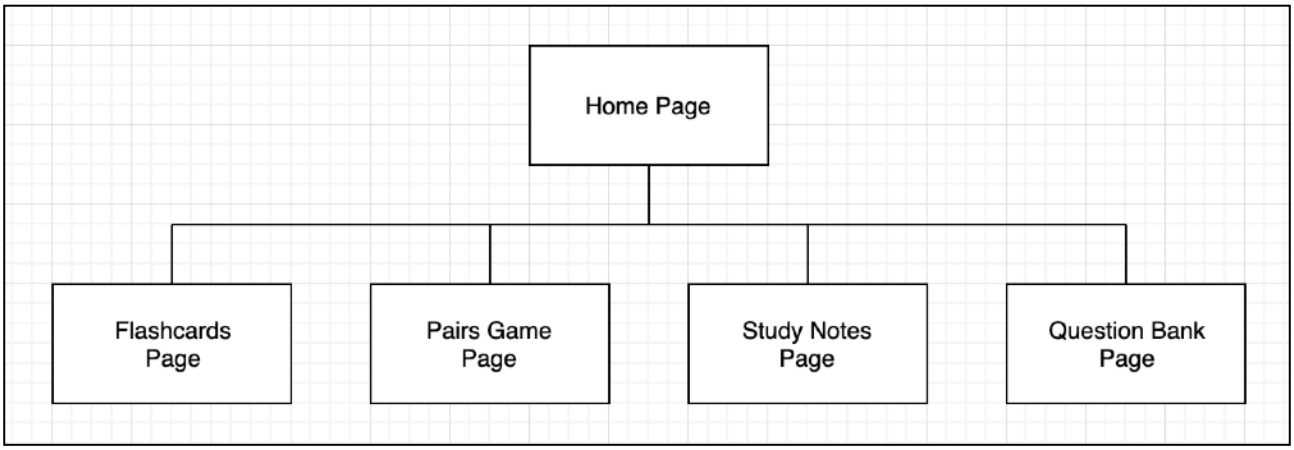
The second diagram in this section is a collaboration diagram, based on a homework assignment which built an application on the rest countries API (referred to elsewhere in this PDA document).

This diagram concerns the process of loading the application and preparing it for its first use.



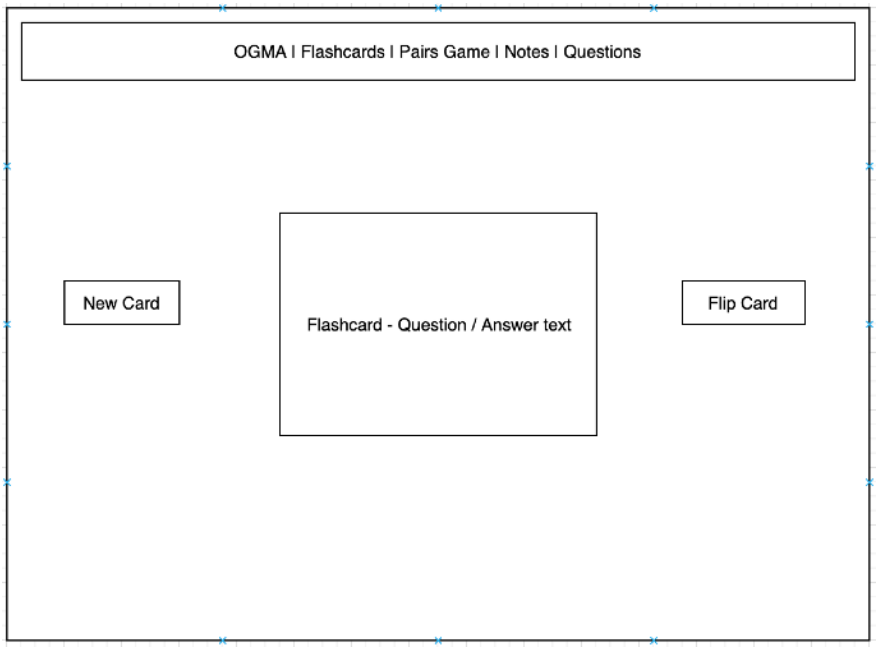
Week 10

Unit	Ref	Evidence
P	P.5	User Site Map

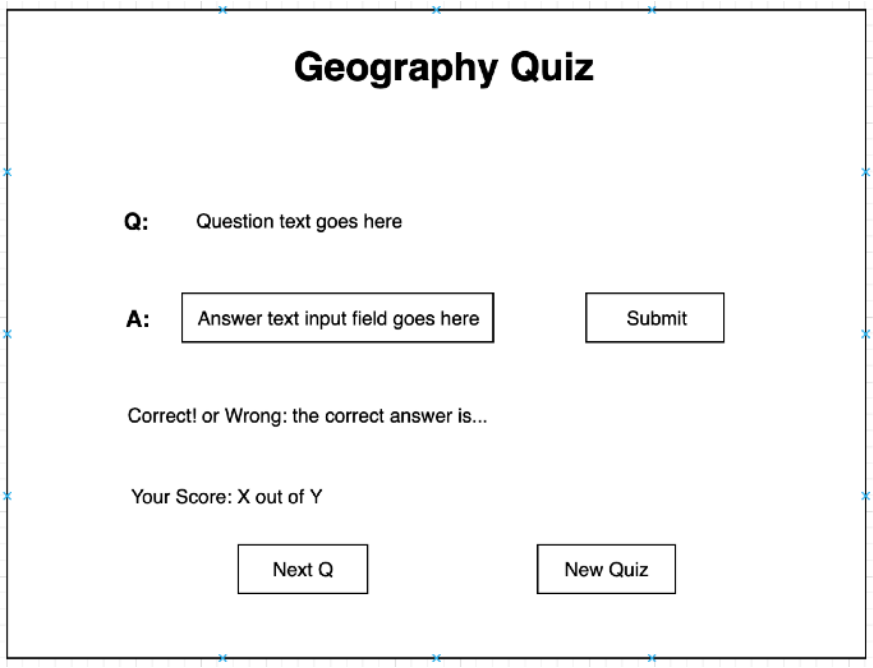


The screenshot above is a site map for my group project. The diagram shows the site at the point we reached by the end of the allotted time; it has a very flat structure, and would grow mainly in a horizontal way as extensions were added. The pages shown in the diagram are all reached by a nav bar running across the top of the screen which is common to all the pages.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

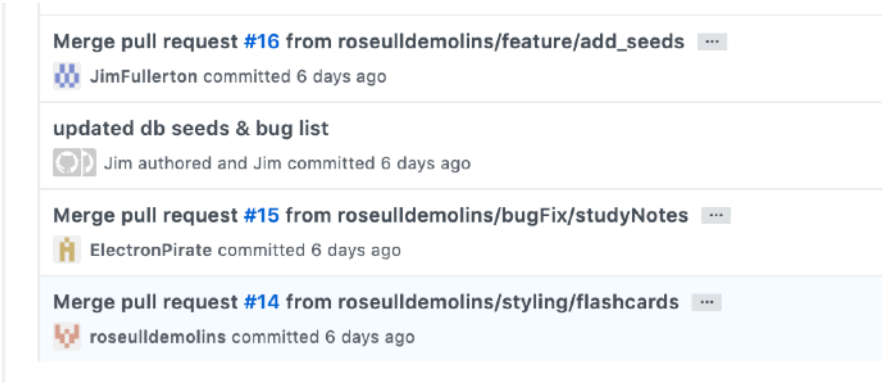


The diagram above shows a wireframe for the MVP for my group project (a single page and a nav bar, even though, initially, other tabs on the nav bar would lead only to holding pages). The finished project retained the essential character of the early wireframe, although it was heavily styled with CSS.



The second wireframe relates to an earlier project in which the brief was simply to make use of a publicly available API. This shows the basic design for a geography quiz driven by data from the rest countries API (https://restcountries.eu/).

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.



Group working: The GitHub account for my group project was set up incorrectly, and shows just myself as the one and only contributor. However, the screenshot above is an extract from the repository's change history, and clearly shows pull requests being made by myself (JimFullerton) and my 2 collaborators, Pawel (ElectronPirate) and Rose (roseulldemolins).

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

1. Definition of the class 'Guitar' (other similar definitions exist for different instrument classes):

```
1   public class Guitar implements IPlay{
2
3     private int wholesaleCost;
4     private int retailPrice;
5     private int numberOfStrings;
6
7     // constructor
8     public Guitar(int cost, int price, int strings) {
9       this.wholesaleCost = cost;
10      this.retailPrice = price;
11      this.numberOfStrings = strings;
12    }
13
14    public String makeSound(){
15      return "Twang";
16    }
17
18  }
19
```

2. Definition of the interface ‘IPlay’:

```
public interface IPlay{
  public String makeSound();
}

```

3. Creation of an array of different instrument objects using the ‘IPlay’ interface:

```
public class Shop {

  private String shopName;
  private ArrayList<IPlay> shopStock;

  // constructor
  public Shop(String name) {
    this.shopName = name;
    this.shopStock = new ArrayList<IPlay>();
  }

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

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

  public ArrayList<IPlay> getSounds(){
    return this.shopStock;
  }

  public static void main(String[] args) {

    Shop scayles = new Shop("Scayles");

    Guitar g = new Guitar(300, 400, 6);
    Mandolin m = new Mandolin(200, 300, "Gibson");
    Trumpet t = new Trumpet(400, 500, "Green");
    Cymbal c = new Cymbal(100, 150, 14);

    scayles.addStock(g);
    scayles.addStock(m);
    scayles.addStock(t);
    scayles.addStock(c);

    for (int i=0; i<scayles.getSounds().size(); i++) {
      System.out.println(scayles.getSounds().get(i).makeSound());
    }

  }
}
```

In the examples above, the ‘Shop’ is a public class, and therefore available throughout the program. The shop has a list of its stock, in the form of an array of instruments. Each instrument is defined as a different class, and to allow them all to be stored as stock, they must adhere to the conditions of the interface ‘IPlay’, i.e. they must have a ‘makeSound’ method which returns a string. Thus, the various instrument objects are polymorphic (e.g. simultaneously a ‘Guitar’ and an ‘IPlay’).