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

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

```
1  const james = {
2    firstName: 'James',
3    surName: 'Berry',
4    age: 31,
5  };
6
7  function haveBithday(person){
8    person.age +=1;
9  }
10
11 console.log(`${james.firstName} ${james.surName} is ${james.age} years old`);
12 haveBithday(james);
13 console.log(`HAPPY BIRTHDAY! ${james.firstName} is now ${james.age}!`);
14
```

```
➜  examples git:(master) ✗ node family.js
James Berry is 31 years old
HAPPY BIRTHDAY! James is now 32!
➜  examples git:(master) ✗
```

'James' is an object with a first name, surname and age.
The function 'have birthday' has a person passed to it as an argument and increases the value of age by 1.

The function is invoked on line 12. The outputs on lines 11 and 13 show that the value of age has increased by 1.

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

```
1  const james = {
2    firstName: 'James',
3    surName: 'Berry',
4    age: 31,
5  };
6
7  const family = [];
8
9  function addToFamily(person){
10   family.push(person);
11 }
12
13 addToFamily(james);
14 addToFamily(jo = {firstName:'Jo',surName:'Berry'});
15
16 console.log('This family contains');
17 family.forEach((familyMember) => {
18   console.log(`${familyMember.firstName} ${familyMember.surName}`);
19 })
```

```
➜  examples git:(master) ✗ node family.js
This family contains
James Berry
Jo Berry
➜  examples git:(master) ✗
```

‘Family’ is an empty array. The function ‘add to family’ pushes a person onto the array. On lines 13 and 14 we invoke this function and push the objects ‘James’ and ‘Jo’ onto the array.

The outputs on lines 16-18 then iterate over the array and provide details of each object it contains.

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

```
class Car {
  constructor() {
    this.distance = 0;
    this.getDistance = function() {
      return this.distance;
    };
    this.setDistance = function(a) {
      this.distance += a;
    };
  }
}
```

```
const Car = require ('../encapsulation');

describe('a car', () => {
  let myCar;

  beforeEach(() => {
    myCar = new Car ();
  });

  test('getDistance() should start by returning 0', () => {
    expect(myCar.getDistance()).toBe(0);
  });

  test('distance should be a encapsulated', () => {
    expect(myCar.distance).toBe(undefined);
  });

  test('setDistance() increase the distance travelled', () => {
    myCar.setDistance(10)
    expect(myCar.getDistance()).toBe(10);
  });
})
```

```
FAIL specs/car.test.js
  a car
    ✓ getDistance() should start by returning 0 (2ms)
    ✕ distance should be a encapsulated (3ms)
    ✓ setDistance() increase the distance travelled

  ● a car › distance should be a encapsulated

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

    Expected: undefined
    Received: 0

      13 |
      14 |   test('distance should be a encapsulated', () => {
    > 15 |     expect(myCar.distance).toBe(undefined);
         |                            ^
      16 |   });
      17 |
      18 |   test('setDistance() increase the distance travelled', () => {

      at Object.toBe (specs/car.test.js:15:28)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 2 passed, 3 total
Snapshots:   0 total
Time:        0.9s, estimated 1s
Ran all test suites.
npm ERR! Test failed.  See above for more details.
```

The second test has failed, where the variable 'distance' should not be accessible. The class 'Car' was then changed and the tests were re-ran.

```
class Car {
  constructor() {
    let distance = 0;
    this.getDistance = function() {
      return distance;
    };
    this.setDistance = function(a) {
      distance += a;
    };
  }
}
```

```
PASS specs/car.test.js
  a car
    ✓ getDistance() should start by returning 0 (3ms)
    ✓ distance should be a encapsulated
    ✓ setDistance() increase the distance travelled (1ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.002s
Ran all test suites.
```

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

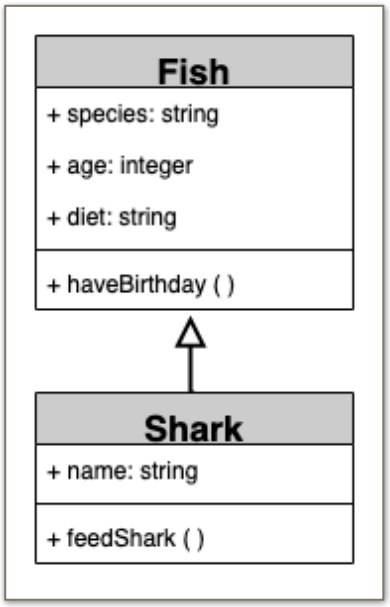
```
class Car {
  constructor() {
    let distance = 0;
    this.getDistance = function() {
      return distance;
    };
    this.setDistance = function(a) {
      if (a>0){
        distance += a;
      };
    };
  }
}

var myCar = new Car();
console.log(`I have driven ${myCar.getDistance()} miles`);
myCar.setDistance(20);
console.log(`I have now driven ${myCar.getDistance()} miles`);
console.log(`My car has travelled ${myCar.distance} miles`);
```

```
➜  examples git:(master) ✗ node encapsulation.js
I have driven 0 miles
I have now driven 20 miles
My car has travelled undefined miles
```

In this example the variable has been encapsulated and is only accessible and alterable via the getDistance() and setDistance() functions.

This could be important as it should be impossible to reduce the distance a car has travelled or re-set it to zero.

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



This inheritance diagram shows the class of Fish and Shark.

The Fish constructor has the variables; species, age and diet. It also has the method haveBirthday where the age of the fish would be incremented by 1.

The class Shark inherits species, age, diet and haveBirthday from Fish. In addition it has the variable name and an additional method of feedShark.

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 Fish{
  constructor(species, age, diet){
    this.haveBirthday= function() {
      age++
    }
  }
}

class Shark extends Fish{
  constructor(name, species, age, diet){
    super(species, age, diet)
    this.feedShark=function(){
      console.log(`${name} gets fed ${diet}`);
    }
  }
}

const molly = new Shark ('Molly', 'reef shark', 5, 'cephalopods')

molly.feedShark();
```

```
➜  examples git:(master) ✗ node inheritance.js
Molly gets fed cephalopods
```

The class ‘Fish’ has a constructor with ‘species’ ‘age' and ‘diet’ variables, the class ‘Shark’ extends that that and adds an additional variable ‘name’.

Shark also has a function which returns a string including the variable name from the ‘Shark’ class and the diet from the ‘Fish’ class.

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.

```
 9    dealCards(){
10      this.startingDeck.forEach((card,index)=>{
11        if (index%2===0){
12          this.players[0].deck.push(this.startingDeck[index]);
13        }else{
14          this.players[1].deck.push(this.startingDeck[index]);
15        };
16      });
17    };
```

The function ‘dealCards’ takes an array ‘startingDeck’ and deals the cards out between players[0] and players[1].

As “index%2” will be zero for every other element in the ‘startingDeck’ array, it will alternate between between the two players exactly as you would expect a human to deal out cards.

This function does not change the order or size of the ‘startingDeck’

```
62    chooseGameWinner(){
63      let winner = null
64      this.players.forEach((player)=>{
65        if (player.deck.length === this.startingDeck.length){
66          winner = player.name;
67          return winner;
68        };
69      });
70    return winner;
71    };
```

The function ‘chooseGameWinner’ checks whether one player holds all the cards. As the previous function ‘dealCards’ did not change the size of the ‘startingDeck’ we can check if that array length is the same as the player’s deck. If there are no winners then it returns null.

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

```
/* UPDATE a mark with id :id. */
router.put('/', function (req,res){
  SqlRunner.run('UPDATE marks SET score = $1 WHERE mark_id = $2',
  [req.body.score, req.body.mark_id])
  .then((result) => {
    res.status(200).json(result);
  });
});
```

mark_id	score	student_id	assessment_id
1	85	1	1
2	90	2	1
3	85	3	1

The above code is a PUT request for the server-side of an API request. It is updating data held in an SQL database. To do so it needs to search for the correct entry. In the above example it is passed ‘mark_id’ to search for the object in question.

‘SqlRunner.run’ can be found below: (this was pre-written and provided before I started my individual project)

```
class SqlRunner {
  static run(sqlQuery, values = []) {
    return pool.query(sqlQuery, values).then(results => {
      return results;
    });
  }
}
```

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

```
5   /* GET all students. */
6   router.get('/', function(req, res, next) {
7     SqlRunner.run("SELECT * FROM students ORDER BY student_id ASC")
8     .then((result)=>{
9       res.status(200).json(result.rows);
10    })
11  });
```

student_id	name
1	Arvin Kelly
2	Connar Mccallum
3	Calvin Prince

The above code is a get request for the server-side of an API request. It is getting all the students from an SQL database and ordering them by ‘student_id’ before sending them in JSON format.

It is also using the ‘SqlRunner.run’; the same as the previous example. (this was pre-written and provided before I started my individual project)

```
7   class SqlRunner {
8     static run(sqlQuery, values = []) {
9       return pool.query(sqlQuery, values).then(results => {
10        return results;
11      });
12    }
13  }
```

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

```
componentDidMount(){
  fetch('https://restcountries.eu/rest/v2/all')
    .then(res => res.json())
    .then(data => {
      this.setState({countriesData: data})
    })
}
render(){
  return(
  <div>
    <Options
      countriesData = {this.state.countriesData}
      populateSelectedCountriesData = {this.populateSelectedCountriesData}
    />
    <CountriesContainer
      selectedCountriesData = {this.state.selectedCountriesData}
    />
  </div>
  )
}
```

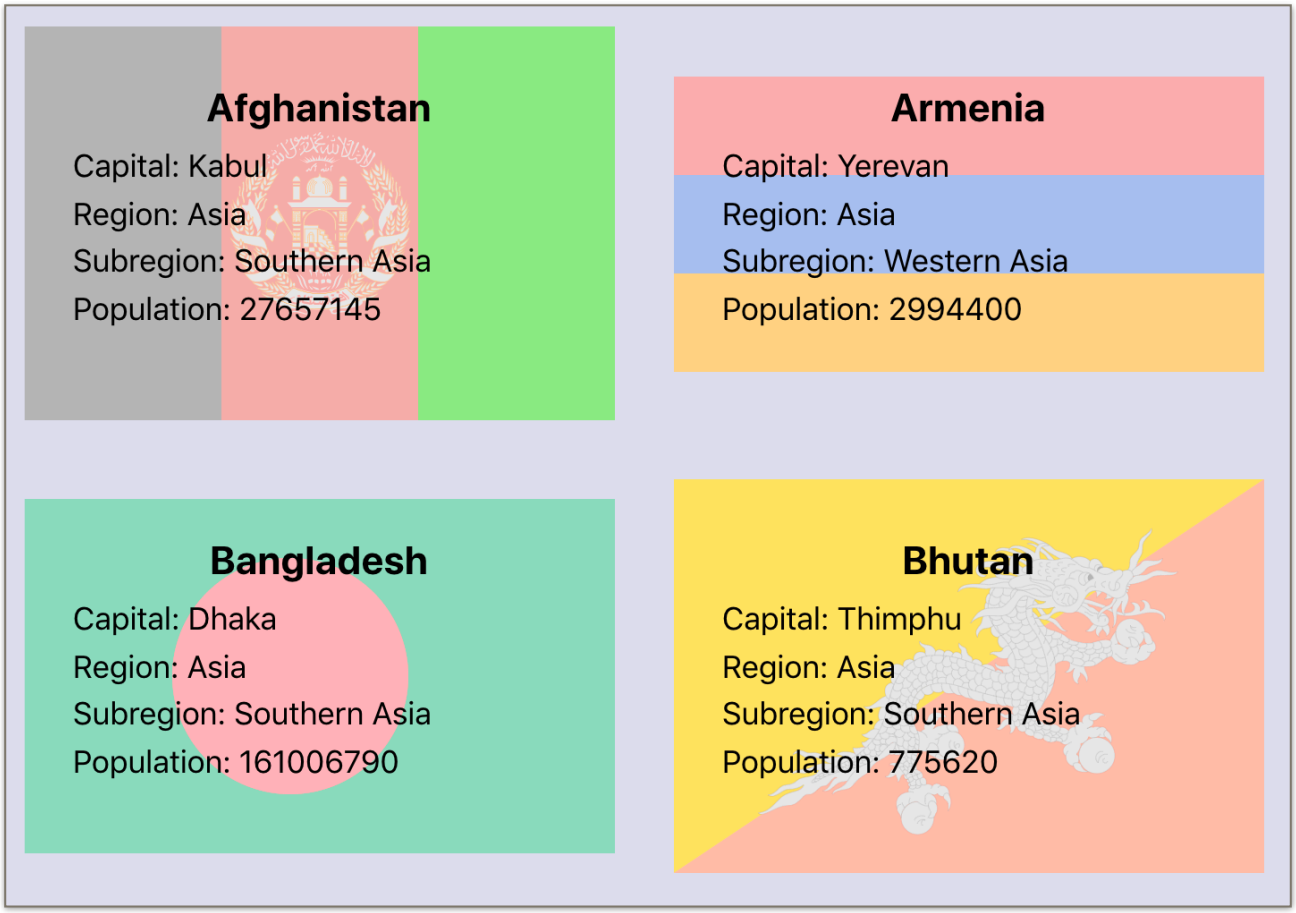
The above code is part of a react app. It calls the ‘Rest Countries API’ when the DOM element loads, it then stores this data in state and passes it to an ‘Options’ and ‘CountriesContainer’ DOM elements.

```
const CountriesContainer = (props) => {
  const countriesList = props.selectedCountriesData.map((country,index)=>{
    return(
      <CountriesTile key={index} country={country}/>
    )
  });
    return(
      <div className = 'CountriesContainer'>
        {countriesList}
      </div>
  )
}
```

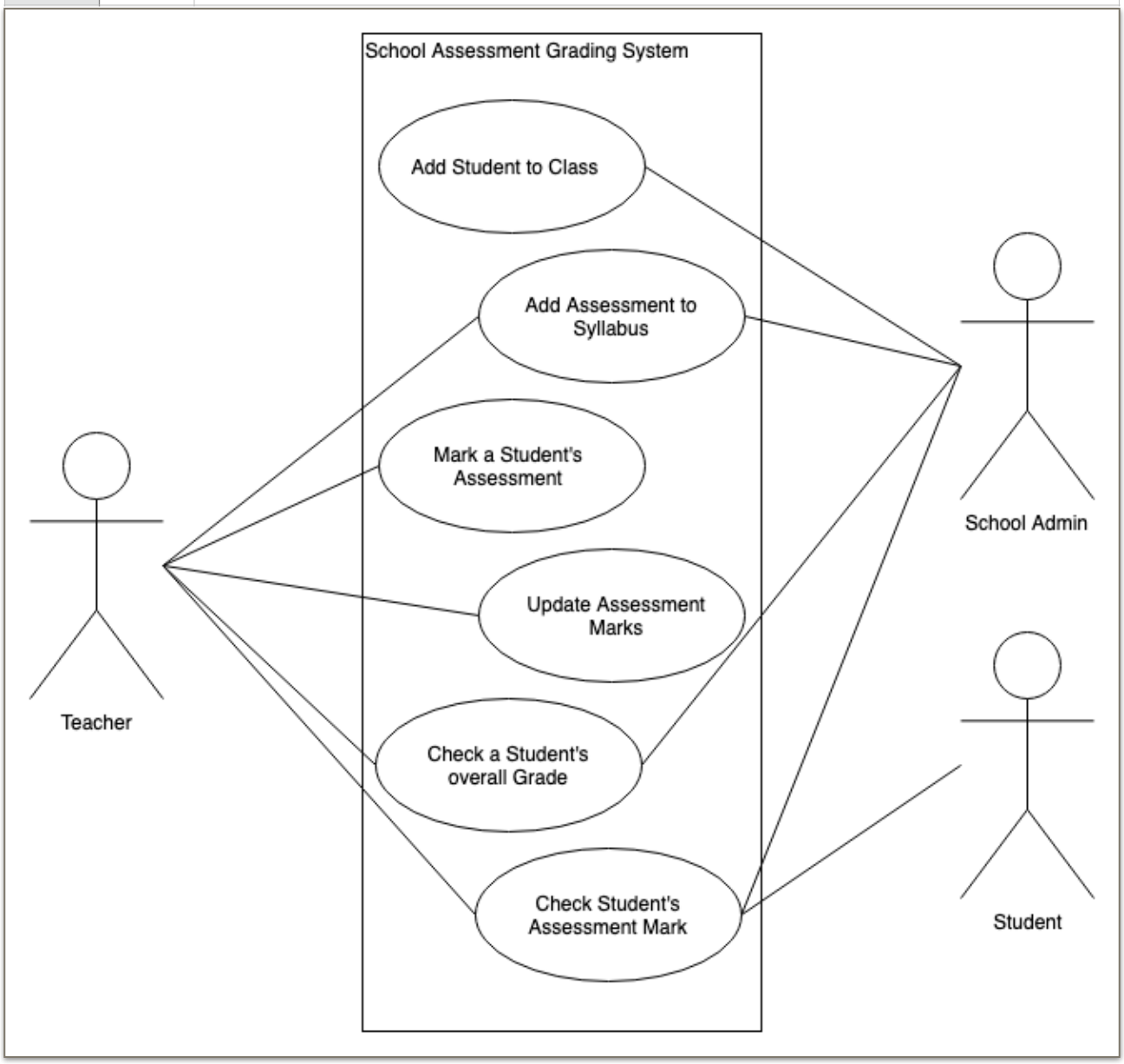
The ‘CountriesContainer’ maps over the data it has been provided and produces an array of ‘CountriesTile’ DOM elements; one for each element in the array.

```
    populateStat(country, stat){
      if (this.props.country[stat]!= "") {
        return(
          <p> {stat}: {this.props.country[stat]} </p>
        )
      }
    };

    render(){
      return(
        <div className = "CountriesTile"
          style = {{backgroundImage: `url(${this.props.country.flag})`}}
        >
          <h4>{this.props.country.name}</h4>
          {this.populateStat(this.props.country,'capital')}
          {this.populateStat(this.props.country,'region')}
          {this.populateStat(this.props.country,'subregion')}
          {this.populateStat(this.props.country,'population')}
        </div>
      )
    }
  }
```

Each ‘CountriesTile’ displays that information as an individual element in the browser. Multiple elements can be seen below (with some CSS styling)



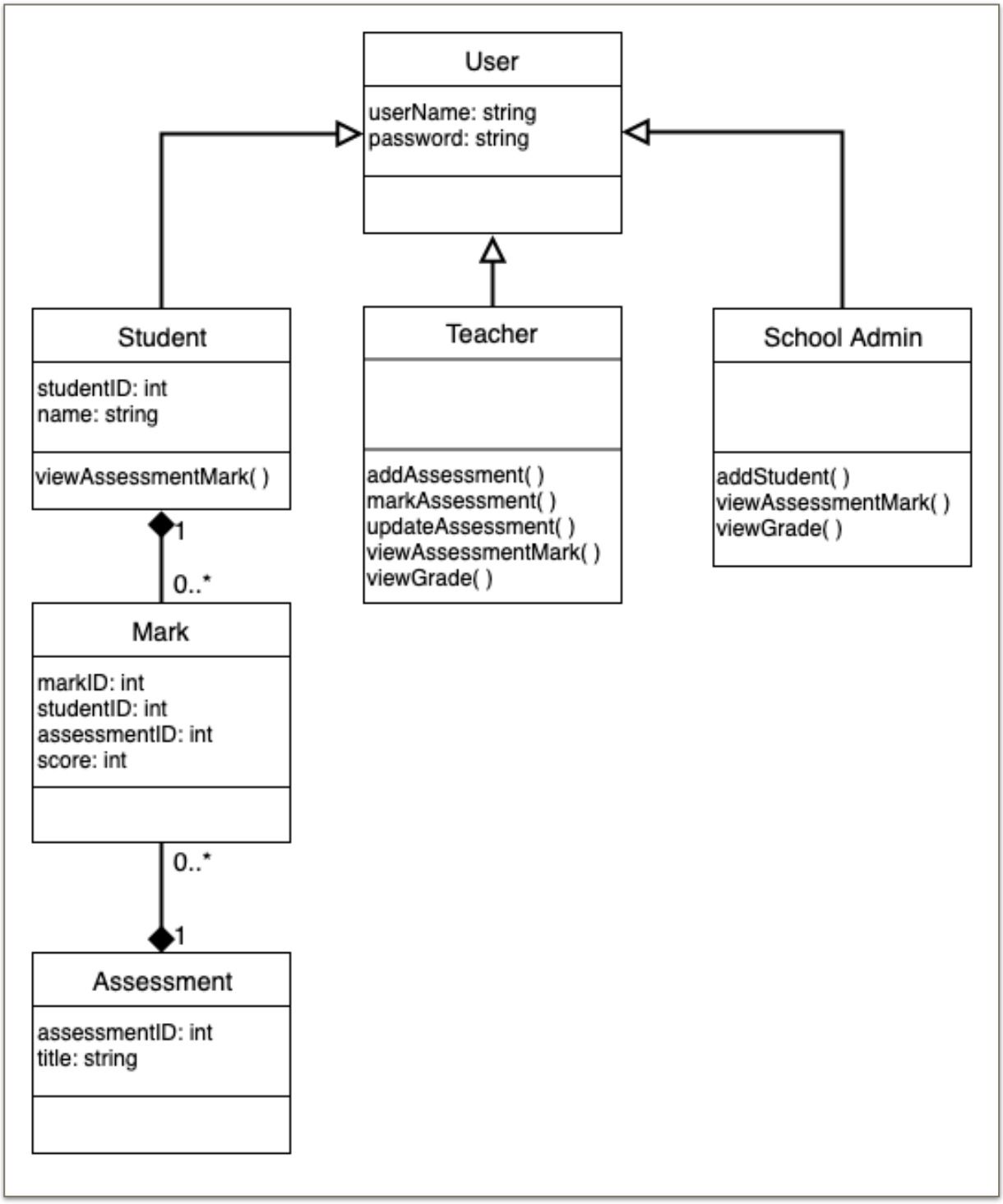
Unit	Ref	Evidence
A&D	A.D.1	Produce a Use Case Diagram



For my solo project I produced a full-stack system that allowed teachers to create a syllabus, mark and update assessments that students have sat. They were also able to view those marks and get a current overall-grade for their students.

A student would want to be able to view the marks of their individual assessments. School administrators would also want to be able to add new students to the class and monitor the marks and grades of the students.

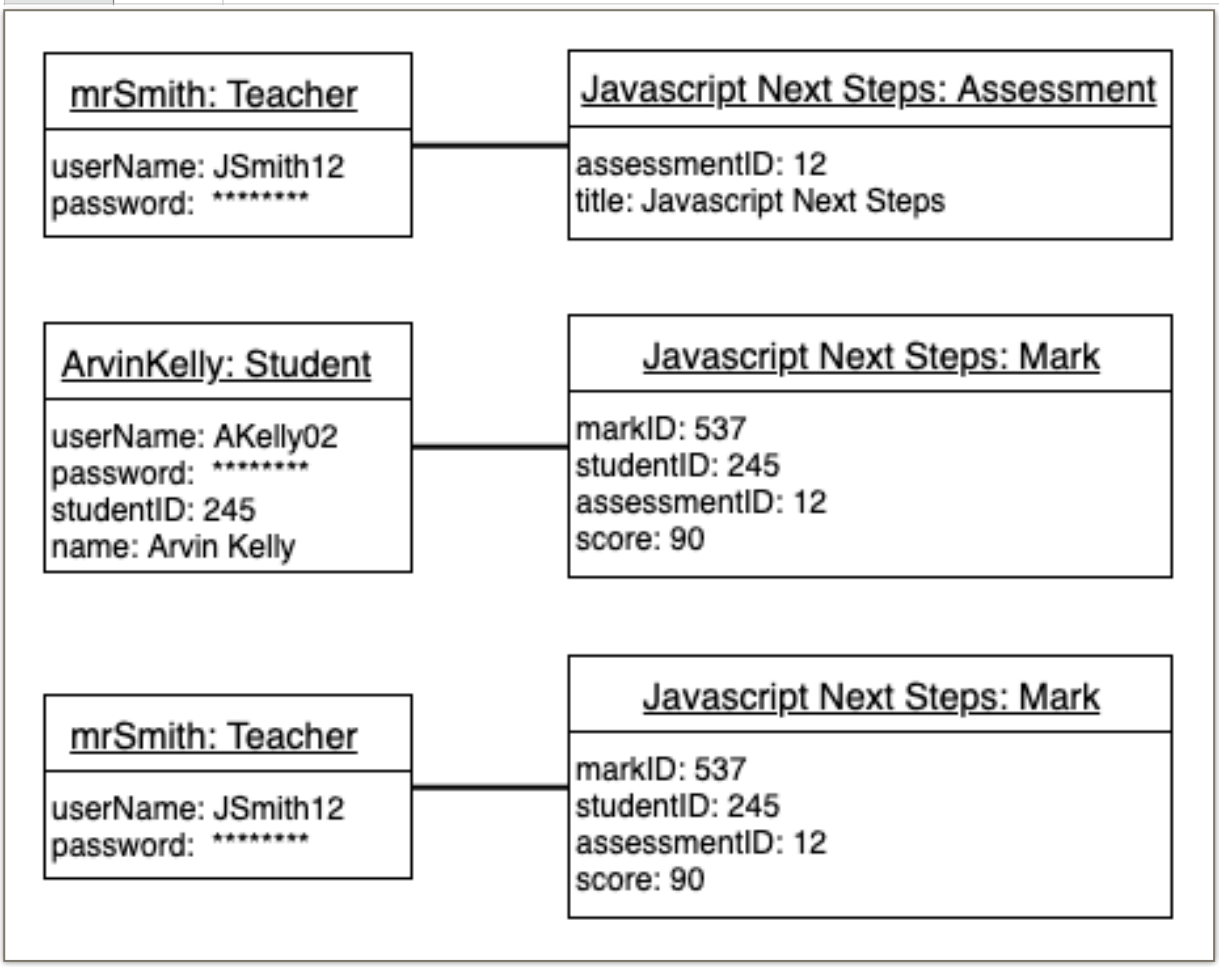
Unit	Ref	Evidence
A&D	A.D.2	Produce a Class Diagram



There are the three classes ‘Student’, ‘Teacher’, and ‘School Admin’ who all inherit from the ‘User’ class. Similarly to the *Use Case Diagram* a teacher will have the methods to add, mark and update assessments, as well as to view individual assent marks and overall grades.

Each student will receive many marks and each assessment will be sat by many students. Each time a student sits an assessment they will receive a mark.

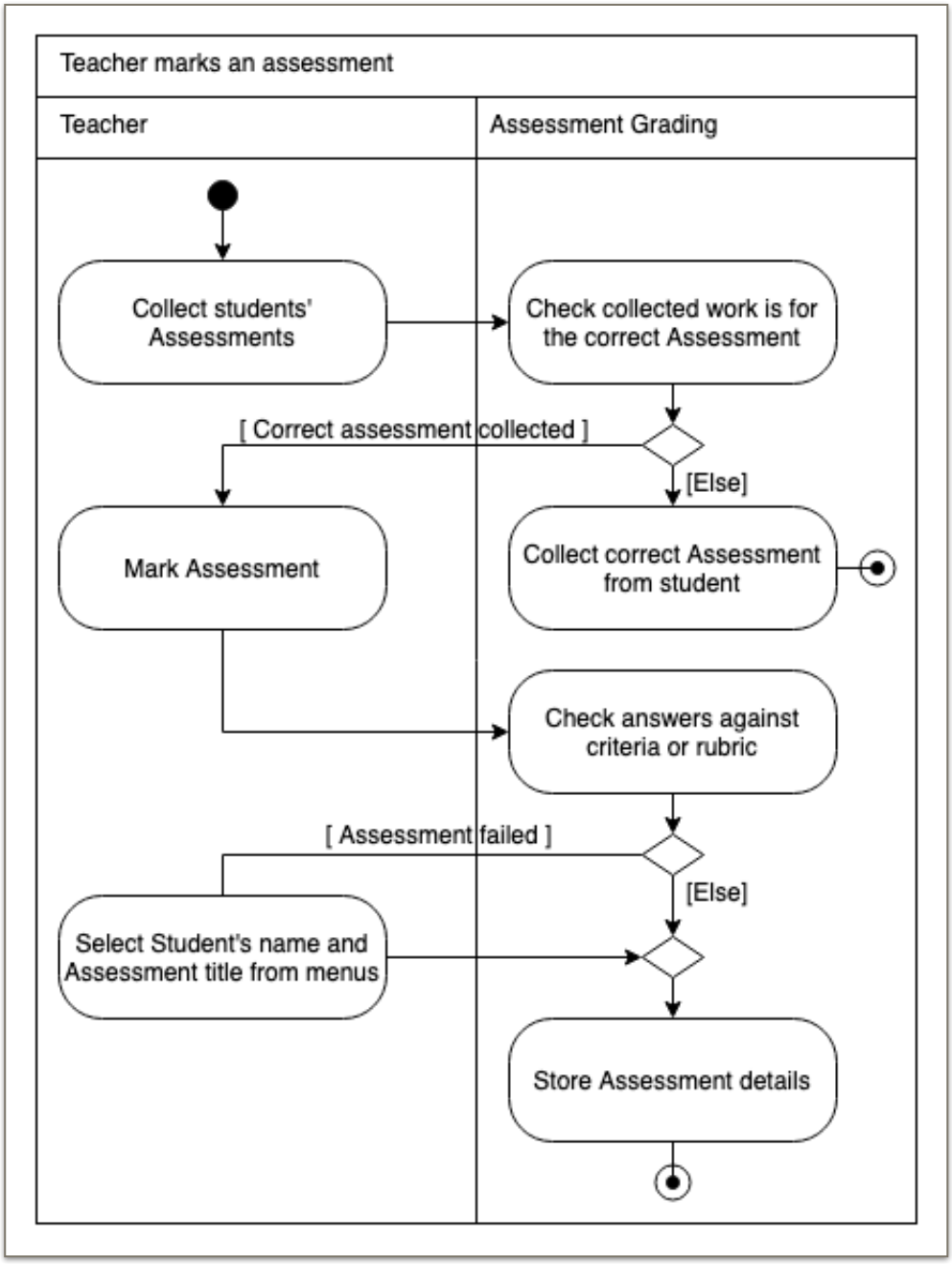
Unit	Ref	Evidence
A&D P	A.D.3 P.8	Produce three Object Diagrams



These three *Object Diagrams* are showing the relationship between:

- The ‘Teacher’ Mr Smith and their ability to add an assessment.
- The ‘Student’ Arvin Kelly and their ability to view their mark.
- The ‘Teacher’ Mr Smith and their ability to view or update the mark of a student.

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



This diagram shows the activity of a teacher logging in and recording the mark of an individual assessment. It starts with the login details being verified; if they are incorrect then there is the option to reset the Username or Password.

The teacher will need to select the relevant option from the menu and then complete the form with student details, the assessment title and the score. Once submitted the system then stores them in the database.

Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time

Constraint Category	Implementation Constraint	Solution
Hardware and Software Platforms	The system will need to be accessible in a computer browser. This will allow for a great majority of users to easily access the system.	It should be produced using Javascript, HTML and CSS. This will allow for access in a computer browser.
Performance Requirements	It is important to ensure that the system is usable on a broad range of computer specs. Back-end needs to be able to process high amounts of data, often in short amounts of time. E.G: Start of year with new students being added to classes, or end of day where multiple teachers will be logging assessment scores for their classes.	Testing for the front-end should be completed on a variety of computers with various specs and display sizes to ensure that the web-page displays as intended. Need to ensure that the hardware and connection for the back-end can handle the anticipated data-load.
Persistent Storage and Transactions	Back-end needs to be able to store details for the time that it is relevant.	Will need to ensure that regular backups are made in the event of hardware failure.
Usability	The user journey will need to be as intuitive as possible for all users.	Sufficient UX mapping will need to be completed. Additional feedback may need to be sought during implementation to ensure that the user-journey is as efficient as possible.
Budgets	The project requires sufficient budget in order to produce a viable product. The first sprint is anticipated to take 1 week and should produce an MVP.	Sufficient planning needs to be done to ensure that the required development costs are within the budget and that there is margin in the case of unforeseen circumstances
Time Limitations	The first sprint is anticipated to take 1 week and should produce an MVP. Following feedback it may be required to re-assess the total project timescale to produce a complete product for customer use.	Planning needs to be completed with benchmarks and points within the project where certain tasks or components need to be completed by.

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

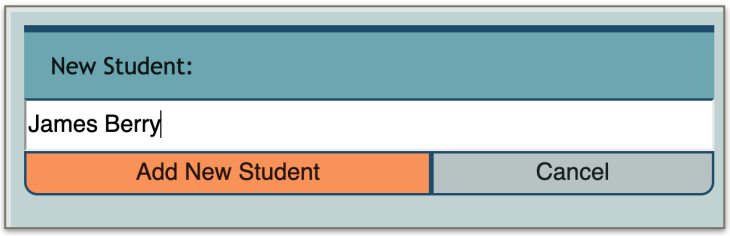
```
renderStudentData(){
  for each student in the class ...
    create a tile containing ...
      their name
      average mark %
      current grade (A-F)
    and attach it to the list DOM element
  then attach the list element to the DOM
}
```

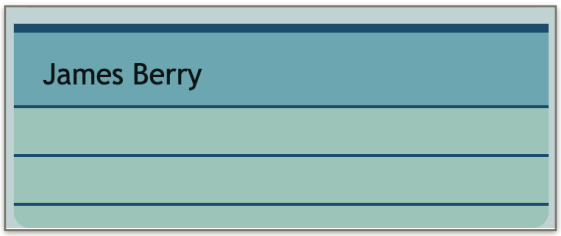
This piece of pseudocode was originally produced to take in an array of students from a class, iterate over that list and then produce a ‘preview tile’ for each student.

These tiles were then appended together into a larger DOM element and then that larger element was then appended to the DOM so they would render in the browser.

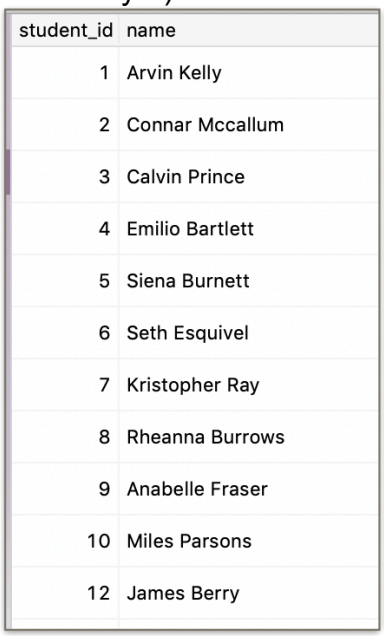
This final code was written in pure javascript so each step was at least 3 lines of code where the DOM element was created, it was given its text content and then appended to the DOM element above it in the tree.

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





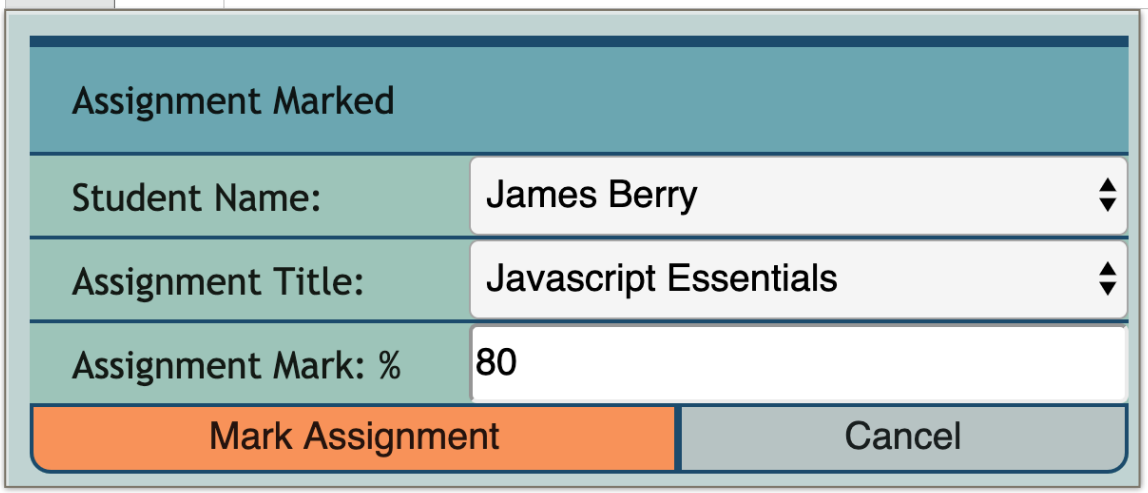
The above screenshots show part of my final app. The first image shows a user form where a new student can be added to the class. Once they have been added it creates a new preview tile for them (with no other content as they have not had any assessments marked yet)



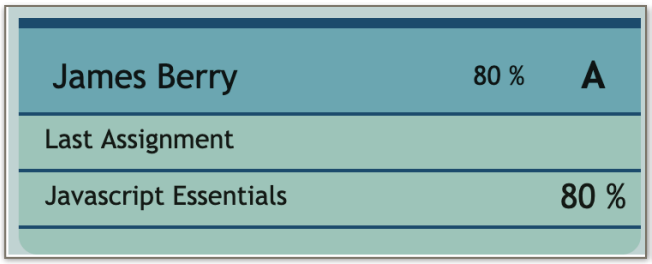
student_id	name
1	Arvin Kelly
2	Connar Mccallum
3	Calvin Prince
4	Emilio Bartlett
5	Siena Burnett
6	Seth Esquivel
7	Kristopher Ray
8	Rheanna Burrows
9	Anabelle Fraser
10	Miles Parsons
12	James Berry

When the SQL database is also inspected we can see that the new student has been added. When the assignment marking form is then opened the new student is also added to the drop-down list.

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

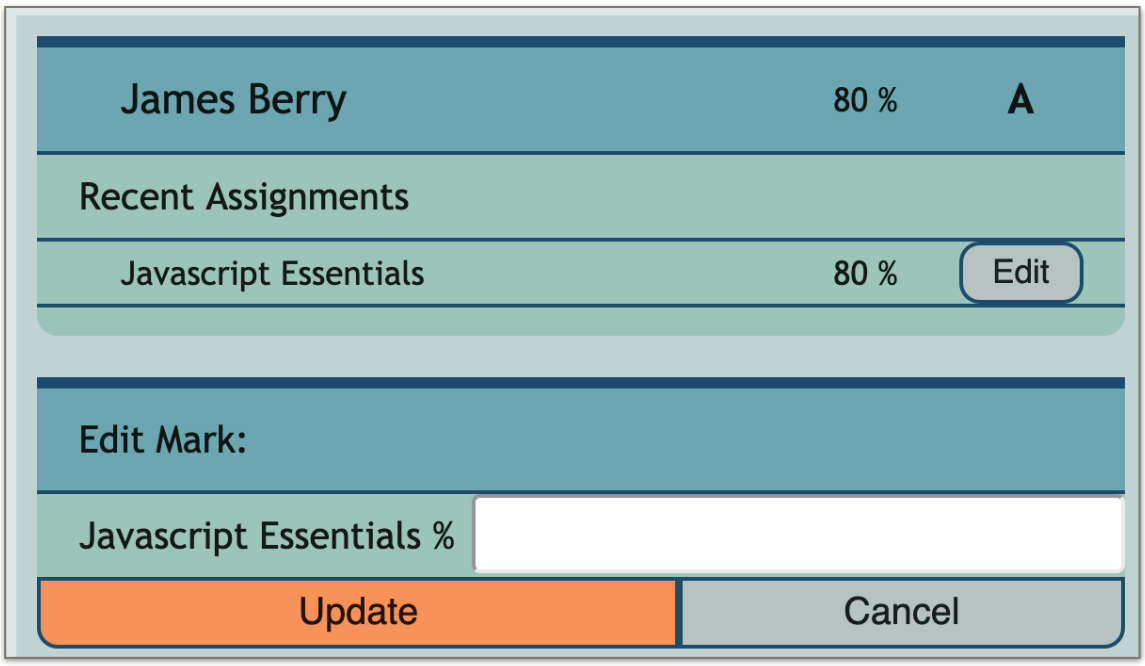


If we continue to complete the assignment marking form and submit it then the preview tile that we have just produced will then be populated. When we inspect the SQL database we can also see that a new mark has been added. From the previous piece of evidence (P.13) we can see that the new student_id is ’12’ and so the bottom entry of the below table is the newly entered data.

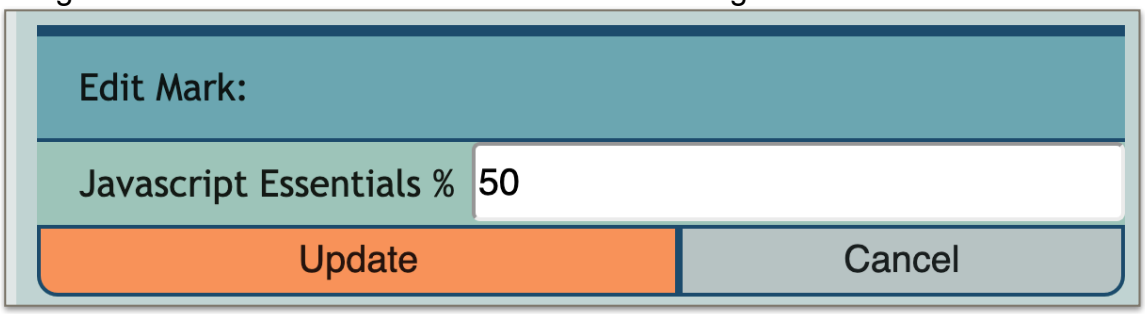


mark_id	score	student_id	assessment_id
30	100	1	7
31	0	2	4
32	80	12	1

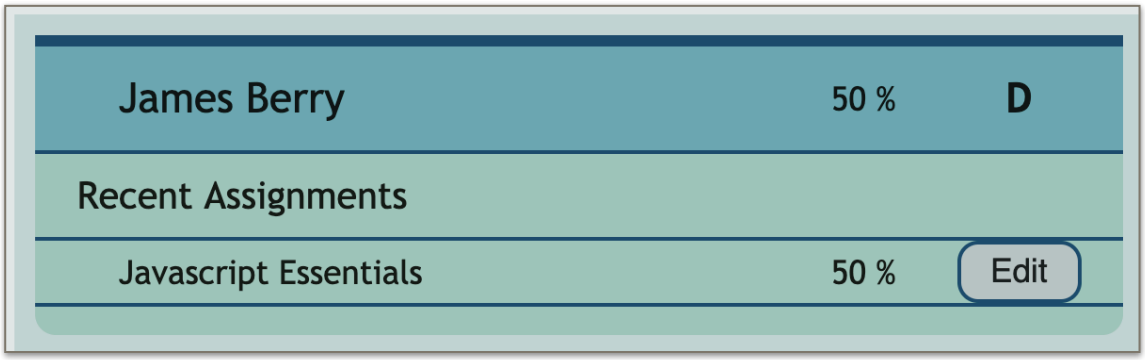
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program



Here we want to edit the mark that we have just entered. When we click the 'Edit' button we get a new menu render which allows us to change the mark.

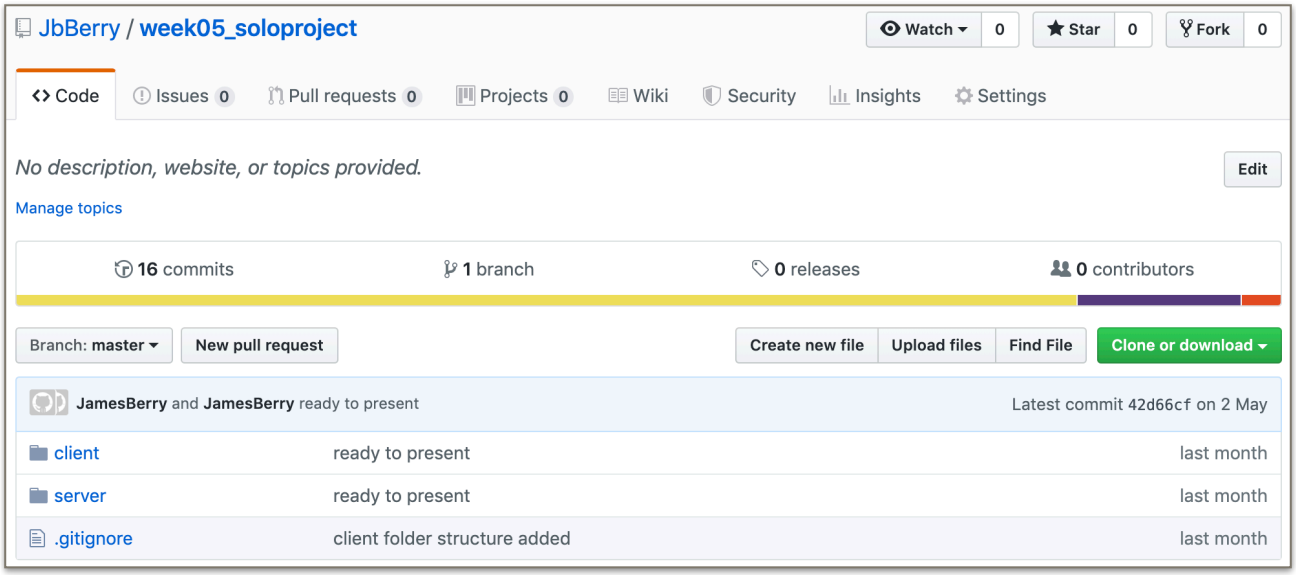


Once this has been completed and we click 'Update'



We can see that the score for that assessment has been updated, as has the average and the overall grade.

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



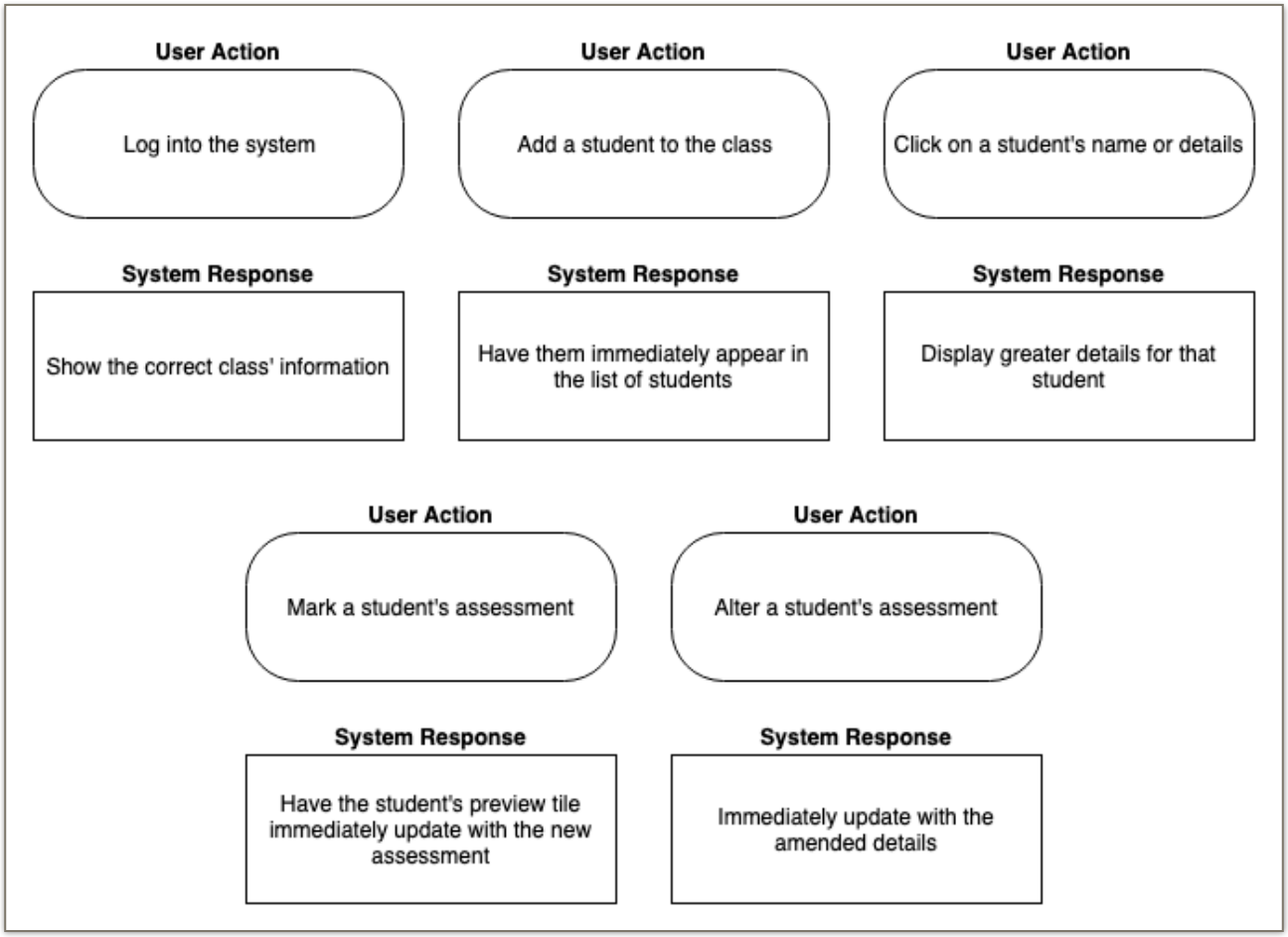
https://github.com/JbBerry/week05_soloproject

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

Whilst planning my individual project I started with a basic user needs diagram, as the teacher would be the main user I first of all concentrated on their needs.

As A…	I Want To…	So That…
Teacher	Be able to record my students marks	All the information is available in one place
Teacher	Keep track of my student's performance	I can help them where they may be struggling
Teacher	Update grades when required	I can keep my records up-to-date

I also produced a User Summary including the basic actions that a user would take and how the system should respond.



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

For the group project, we kept a bug tracking report to help track workload during multiple short sprints.

Bug/Error	Solution	Date
Following initial setup. Does not render/compile	Syntax error in Blackjack.js (line 10).	29 May 2019
Hand value logic does not calculate correctly when hand contains multiple aces.	Changed order of logic so values over 21 are removed after being added instead of not adding them in the first instance. (allows Ace, Ace to equal 2 or 12 as expected)	30 May 2019
When Blackjack starts dealer sometimes does not draw a hand, player can still act as expected.	Unknown why error occurs. Have added link on page to ‘re-draw’ hand in the event of the error occurring.	30 May 2019
	Error occurs if ‘Start Game’ button is clicked before deck has been loaded into the redux store following the API call. ‘Start Game’ button should only render once the deck is available. No time to apply fix before project deadline.	05 June 2019
MongoDB does not launch following the feature branch being merged to develop	Database files and front-end files have been saved together in the same folder. Node modules have been mixed into a single large folder. Time spent to move the front/back-end files into separate folders, major changes to the file structure on newest git-hub commit. Each member of the team needs to take care on next branch merges to or from develop.	31 May 2019
Dealers cards are visible before @Keyframes animation once delay has been implemented. Does not ‘flip’ as intended.	CSS changed so cards have opacity:0 before animation and immediately become opaque on animation start.	31 May 2019
Snap Game does not pause as intended once ‘snap’ is called	Removed ‘this.game’ variable from the constructor. setInterval() and clearInterval() now act as intended.	03 June 2019
Snap Game does not detect a match when it should.	State was not correctly destructured in the reducer. ‘…’ added to snap_reducer.js (line 10).	04 June 2019

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

Project Brief

Create an online card game application that will allow a user to play one or more card games against the computer.

MVP:

There should be at least one fully functional game and the option for a user to register their high score.

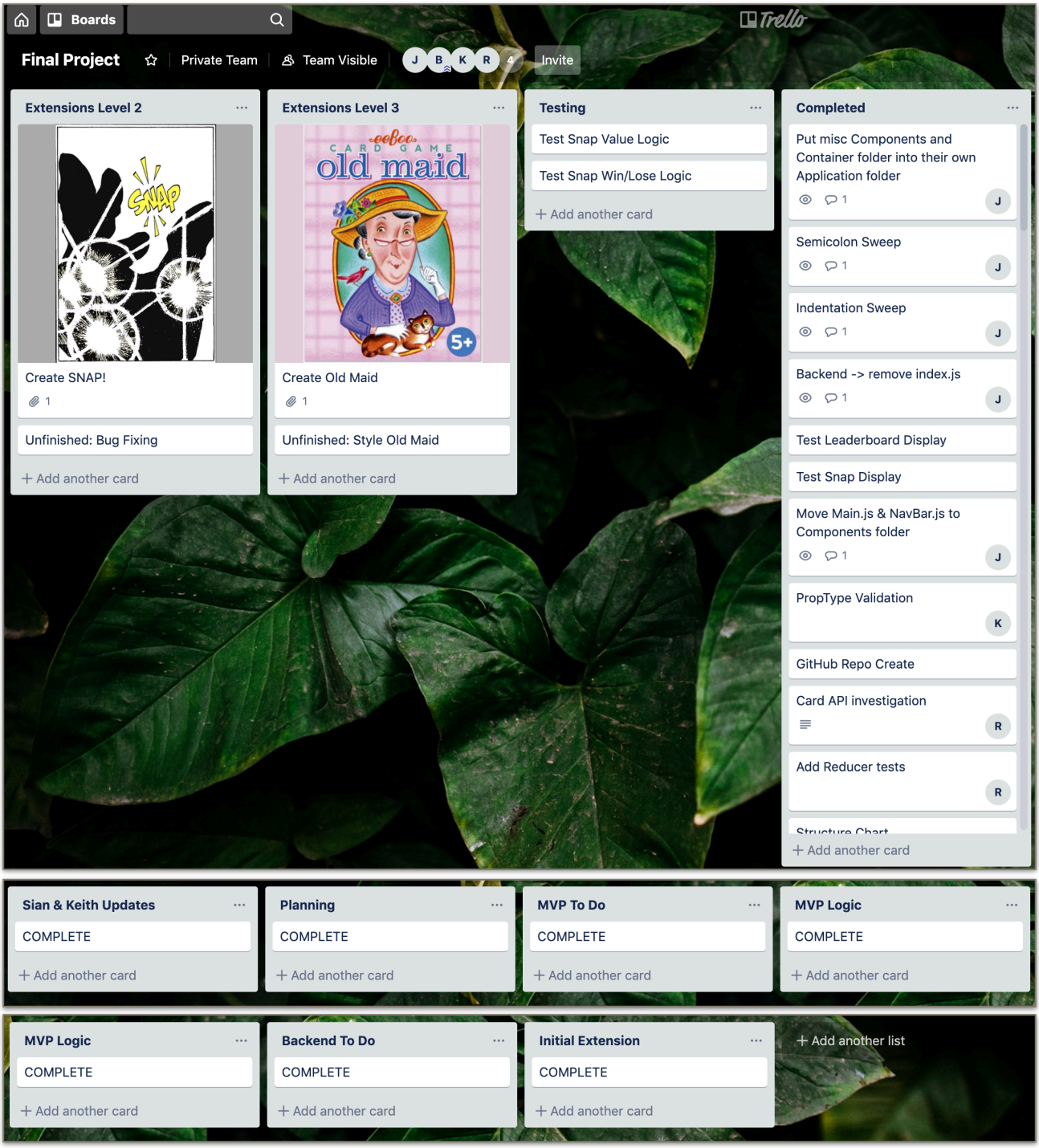
Expectations:

- All members of the group contributing to the planning, development and presentation of the project
- Members supporting each other to make sure everyone can get the most they can from the week
- Some application of Agile concepts e.g. a morning standup, sprints, a kanban board (Trello)

Technical Requirements:

- TDD - unit testing of models and Redux reducers where appropriate
- Frontend testing using react-testing-library
- API calls
- React, Redux and MongoDB
- Regular Git commits and use of branches and PRs. We are looking to see at least 100 commits.
- Validating props with prop-types

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



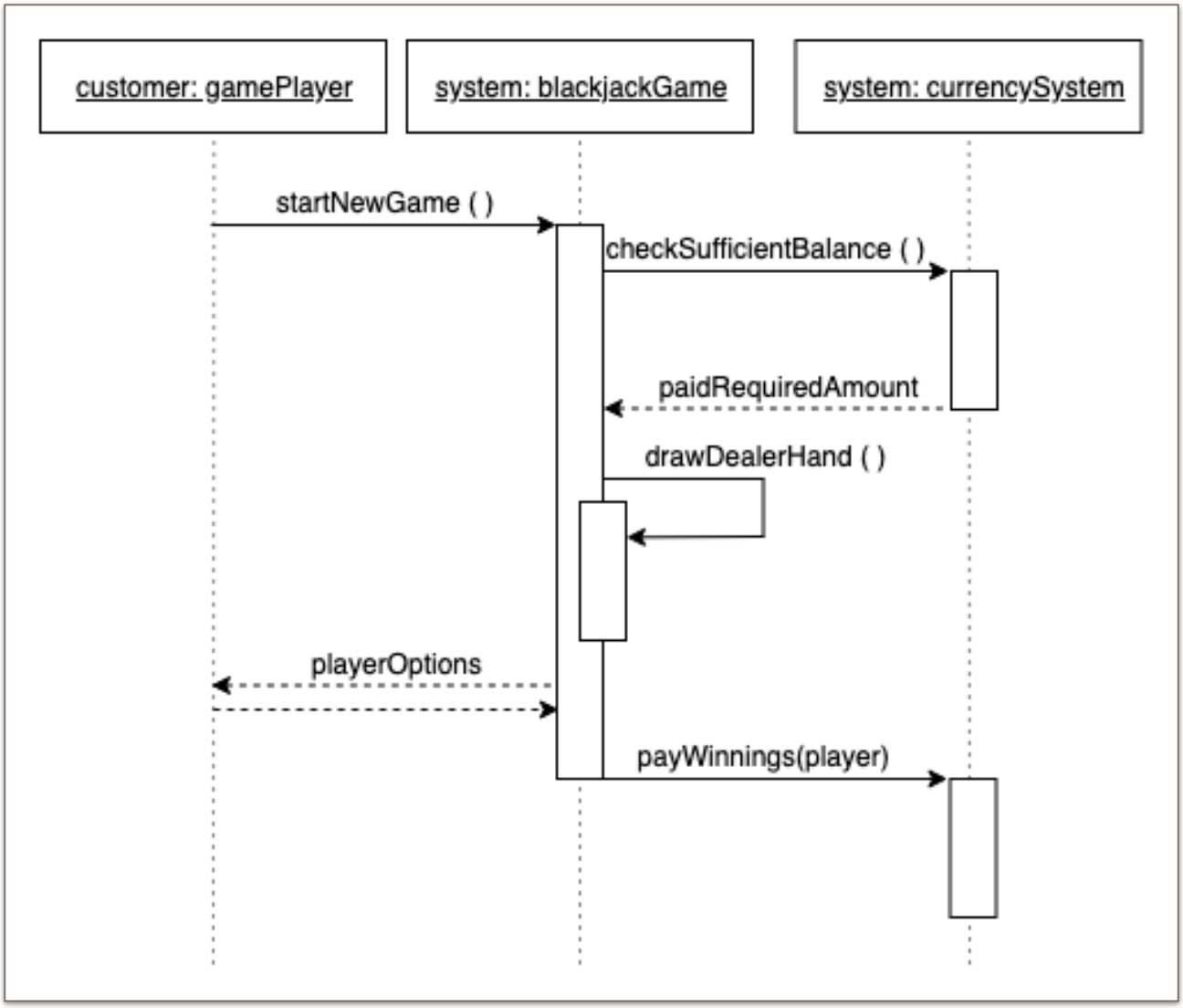
As we completed various tasks we marked each list as complete and any tasks within that list were moved to the ‘Completed’ list. As we had a strict time limitation some of the extension tasks were not completed.

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

As we completed the minimum product and moved into our group project extensions we added to the acceptance criteria. In each game we broke down each logical step into a testable series of results.

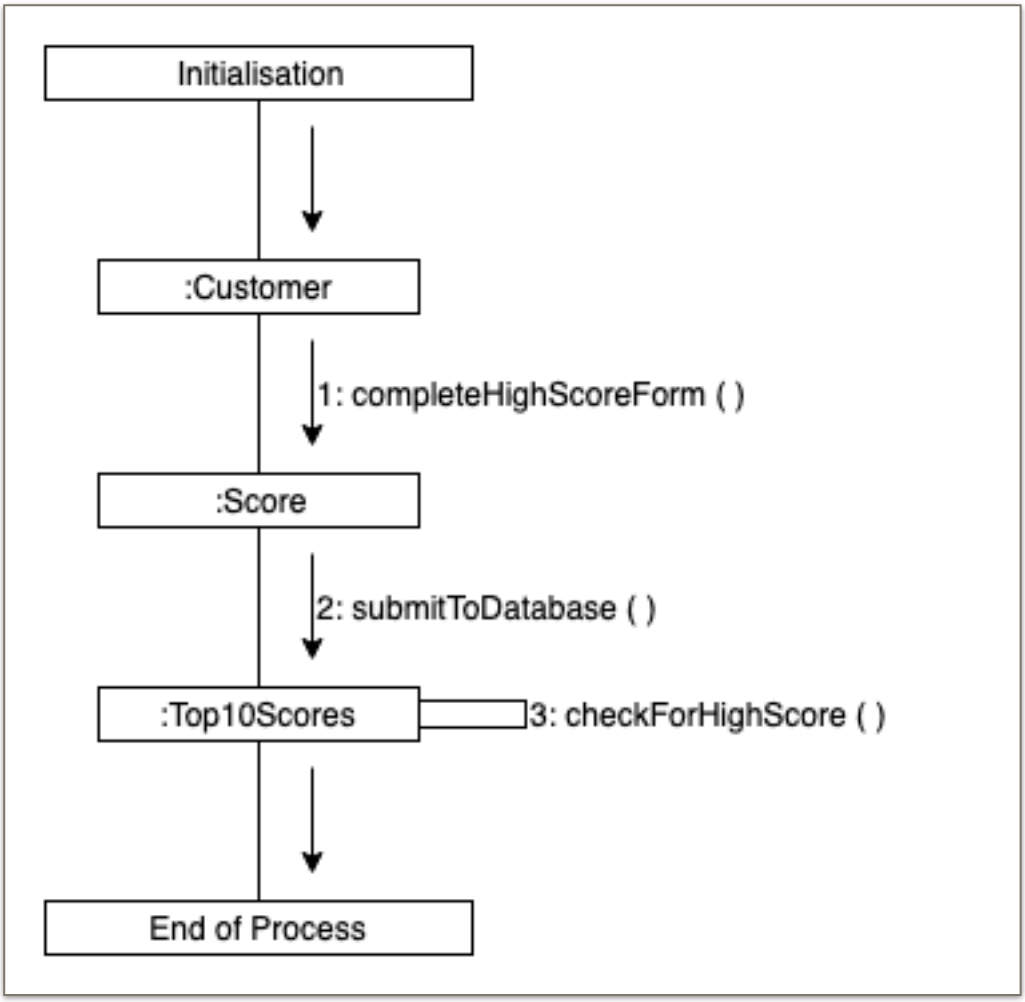
Acceptance Criteria	Expected Result	Pass/Fail
The user is able to start a game a blackjack	The game starts when the user clicks ‘Start Game’: • A hand is generated for the dealer with their hand value. • A button is presented for the user to deal their opening hand.	PASS
The user is able to twist in blackjack	When the user clicks ‘Twist’: • The game draws the user an additional card which is presented in-game: • The user’s hand value is updated	PASS
The user is able to stick in blackjack	When the user clicks ‘Stick’: • The game evaluates the two hands and correctly declares a winner. • If the user has won then their coin total is increased by 20.	PASS
The user is able to start a new game once the previous game of blackjack has finished	The user is presented with a ‘Start Game’ button once the previous game has been won/lost. When the user clicks ‘Start Game’ it starts a new game of blackjack	PASS
The user is able to start a game of snap	The game starts when the user clicks ‘Start Game’: • Cards are dealt one at a time in regular intervals onto a single ‘pile’ in game.	PASS
The user is able to call snap when the cards match	When the user clicks ‘Snap’: • The game stops dealing cards. • The game is able to determine the top two cards match value. • The user’s coin total is increased by 10.	PASS
The user is able to call snap when the cards do not match	When the user clicks ‘Snap’: • The game stops dealing cards. • The game is able to determine the top two cards’ values do not match • The user’s coin total is decreased by 10.	PASS
The user is able to continue playing after they they have called snap.	After the user has called ‘snap’ they are presented with a ‘continue’ button. When this button is clicked the game continues dealing cards one at a time in regular intervals onto a single ‘pile’ in game.	PASS
The user is able to start a new game once the previous game of snap has finished	The user is presented with a ‘New Game’ button once the game has dealt the whole deck into the single ‘pile’. When this button is clicked it starts a new game of snap	PASS
The user is able to ‘cash out’ and submit their score	The user is presented with a ‘Cash Out’ button and a field where they can enter their name. When a name is entered and ‘Cash Out’ is clicked: • Their current score is submitted to the database. • If the score is high enough it will be added to the top10 list in-game. • The users score is reset.	PASS

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



Above is a sequence diagram for the steps of playing blackjack.

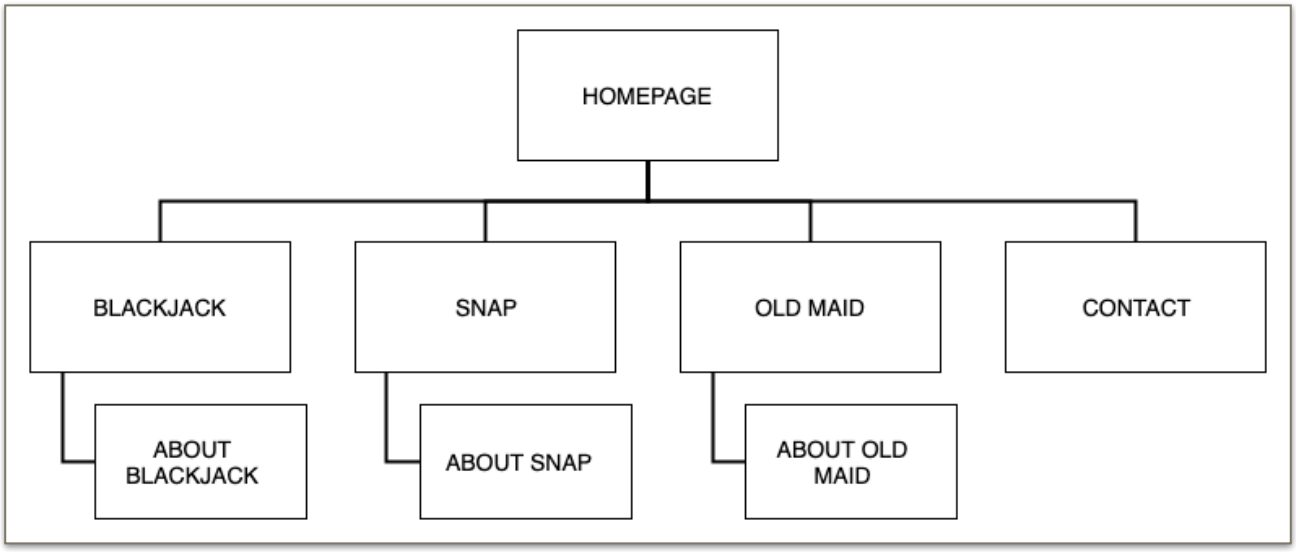
- The Player starts a new game. Before any actions are taken it will check that they have enough balance to pay and then pay to play.
- The game then automatically draws the dealer's hand and will give the player options to develop their hand.
- Once the player has finished with their options and has opted to 'stick' then the game will determine the winner and pay winnings to that person.



Above is a Collaboration diagram for the steps of a player submitting their score.

- The Customer will complete a high score form.
- That score will then be submitted and saved to the database.
- It will be then checked against the current high scores and if high enough will be added to the top 10 scores.

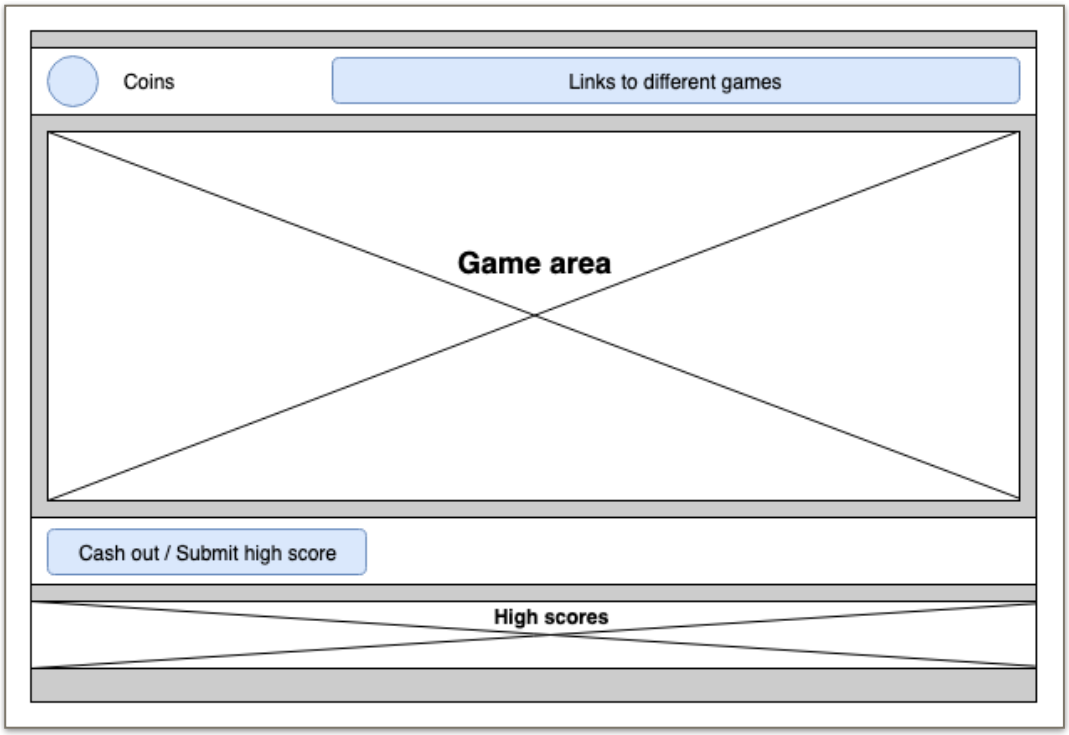
Unit	Ref	Evidence
P	P.5	User Site Map



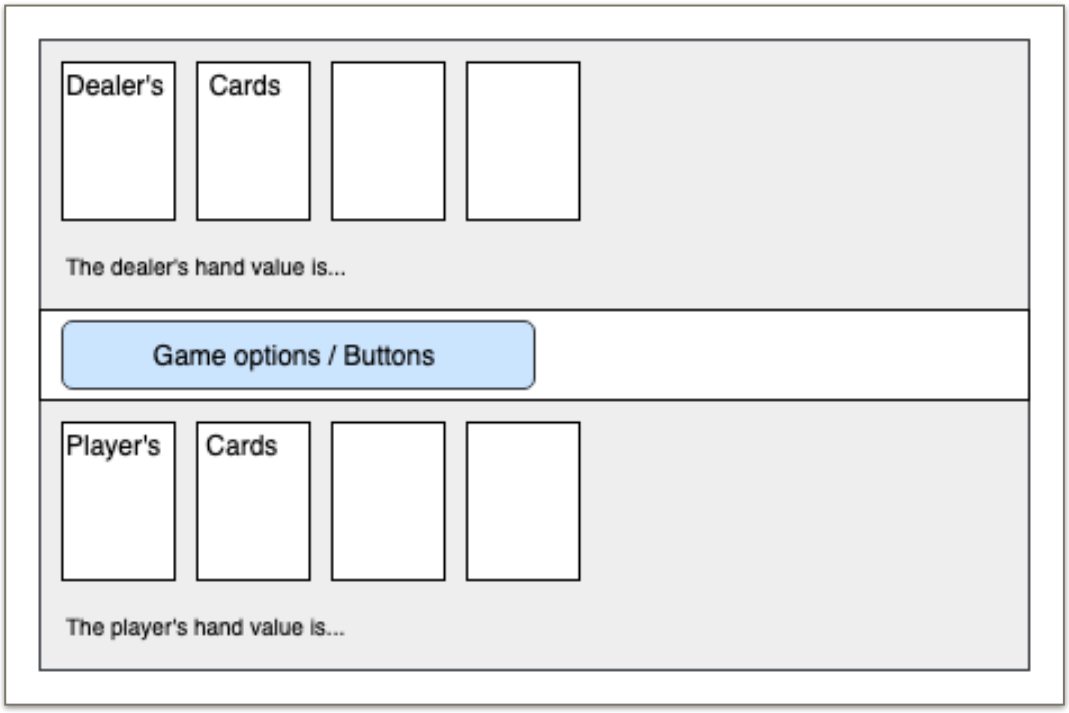
From the homepage there should be links to each game and a link to a contact page.

Within each game there should be a further link for some further information, help and advice of how to play.

Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams

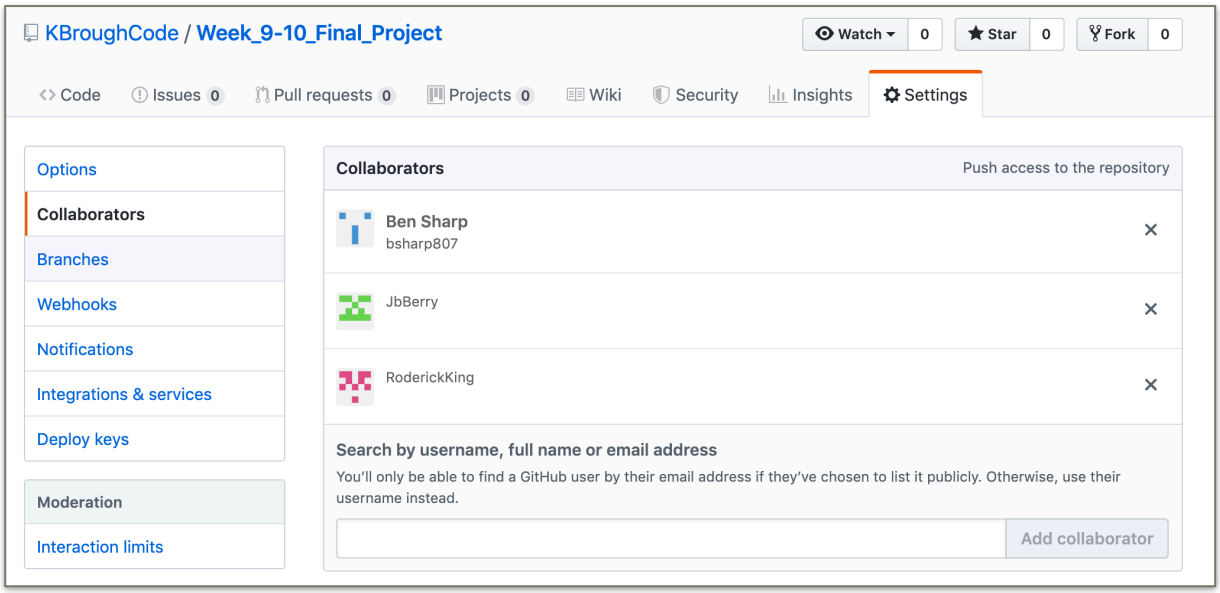


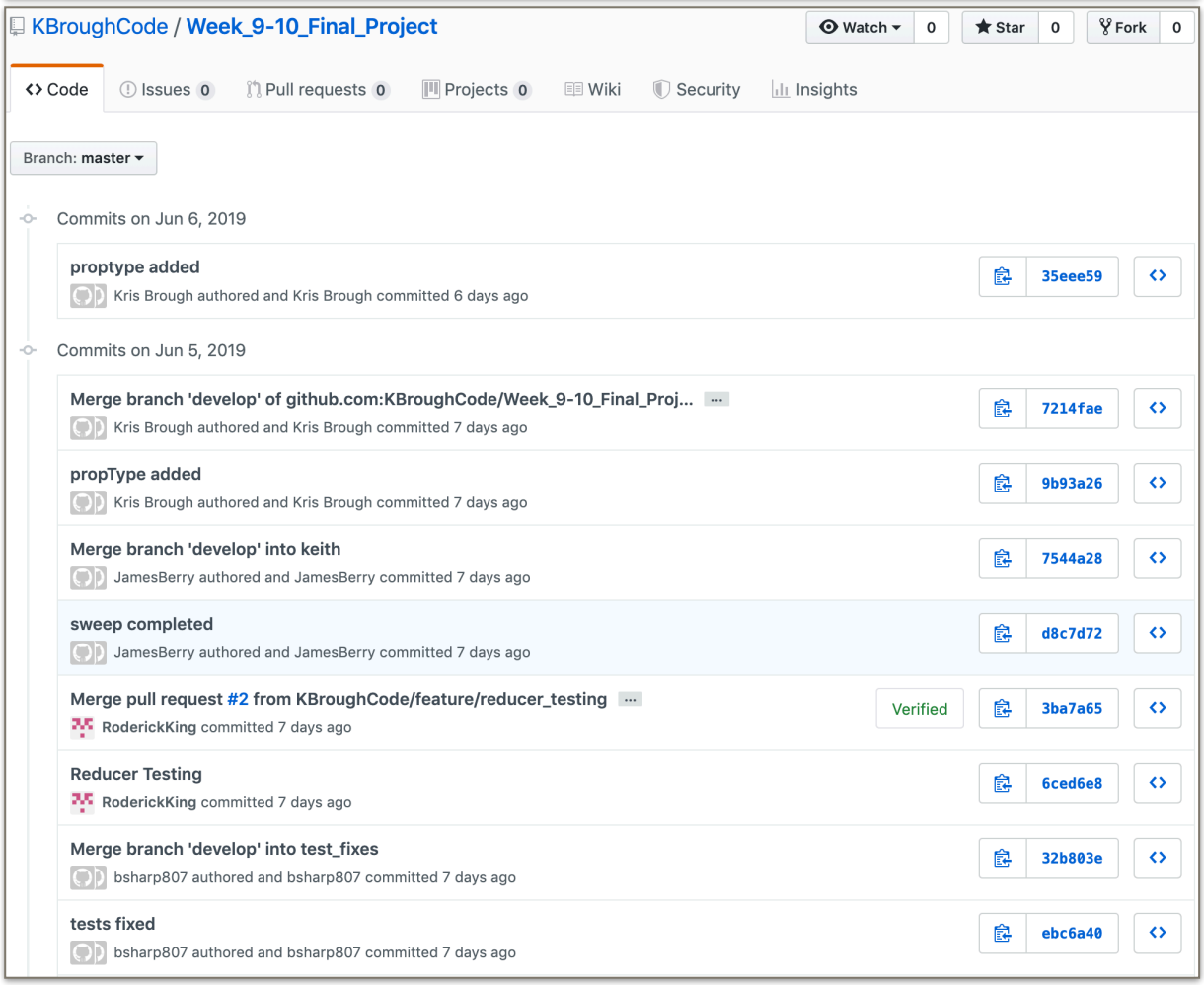
The above wireframe is the main screen for the group project. We felt it was important to make it feel app-like; so we ensured that there was a common header & footer. This way the user is able to easily move between games, is able to view their current score and easily ‘cash out’. The large space in the middle is for each game.



This second wire-frame is for the blackjack game. With the dealer’s hand and details at the top of the page and the player’s at the bottom. The central banner is for the players ‘start game’, ‘draw hand’, ‘stick’, ‘twist’ etc. buttons.

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.





The above screenshots show the collaborators and the final commits on our group project.

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

```
public interface IInstrument{
  public String makeNoise();
  public String getBrand();
  public String getModel();
  public int getPrice();
}
```

```
public class Guitar implements IInstrument{

  private String brand;
  private String model;
  private int price;

  // constructor
  public Guitar(String brand, String model, int price){
    this.brand=brand;
    this.model=model;
    this.price=price;
  }

  public String makeNoise(){
    return "twang";
  }

  public String getBrand(){
    return this.brand;
  }

  public String getModel(){
    return this.model;
  }

  public int getPrice(){
    return this.price;
  }
}
```

Above we have the classes ‘Guitar’, ‘Trumpet’ and ‘Piano’. They all implement the interface called ‘IInstrument’.

This allows us to add objects of these three different classes together into a single array.

```
public class Trumpet implements IInstrument{

  private String brand;
  private String model;
  private int price;

  // constructor
  public Trumpet(String brand, String model, int price){
    this.brand=brand;
    this.model=model;
    this.price=price;
  }

  public String makeNoise(){
    return "toot";
  }

  public String getBrand(){
    return this.brand;
  }

  public String getModel(){
    return this.model;
  }

  public int getPrice(){
    return this.price;
  }
}
```

```
public class Piano implements IInstrument{

  private String brand;
  private String model;
  private int price;

  // constructor
  public Piano(String brand, String model, int price){
    this.brand=brand;
    this.model=model;
    this.price=price;
  }

  public String makeNoise(){
    return "plink plonk";
  }

  public String getBrand(){
    return this.brand;
  }

  public String getModel(){
    return this.model;
  }

  public int getPrice(){
    return this.price;
  }
}
```

```java
import java.util.*;

public class Shop{

  private ArrayList<IInstrument> instrumentStock;
  private String shopName;
  private int tillBalance;

  // constructor
  public Shop(String shopName){
    this.shopName=shopName;
    this.instrumentStock = new ArrayList<IInstrument>();
  }

  public void addInstrumentStock(IInstrument instrument){
    this.instrumentStock.add(instrument);
  }

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

  public ArrayList<IInstrument> getInstrumentStock(){
    return this.instrumentStock;
  }

  public static void main(String[] args) {
    Shop tootYourHorn = new Shop("Toot Your Horn");

    Guitar g = new Guitar("YAMAHA", "F310", 115);
    Piano p = new Piano("Hape", "E0320", 100);
    Trumpet t = new Trumpet("YAMAHA", "YTR-2330", 379);

    System.out.println("Welcome to " + tootYourHorn.getshopName());
    tootYourHorn.addInstrumentStock(g);
    tootYourHorn.addInstrumentStock(p);
    tootYourHorn.addInstrumentStock(t);
    for (int i=0;i<tootYourHorn.getInstrumentStock().size();i++) {
      System.out.println(
      tootYourHorn.getInstrumentStock().get(i).getClass().getSimpleName()+ " " +
      tootYourHorn.getInstrumentStock().get(i).getBrand() + " " +
      tootYourHorn.getInstrumentStock().get(i).getModel() + " costs £" +
      tootYourHorn.getInstrumentStock().get(i).getPrice());
    }
    System.out.println("Our shop goes ");
    for (int i=0;i<tootYourHorn.getInstrumentStock().size();i++) {
      System.out.println(tootYourHorn.getInstrumentStock().get(i).makeNoise());
    }
  }
}
```

```
➜ Music_Shop java Shop
Welcome to Toot Your Horn
Guitar YAMAHA F310 costs £115
Piano Hape E0320 costs £100
Trumpet YAMAHA YTR-2330 costs £379
Our shop goes
twang
plink plonk
toot
```

Here we have the class ‘Shop’.

‘tootYourHorn’ is an instance of the class Shop. It has an array ‘instrumentStock’ which contains the objects ‘g’, ‘p’, and ’t’ which are instances of the classes Guitar, Piano, and Trumpet respectively.

The system.out iterates over the array ‘instrumentStock’ and uses the getter methods from those Classes (and the interface).