Evidence Gathering Document for SQA Level 8 Professional Developer Award.

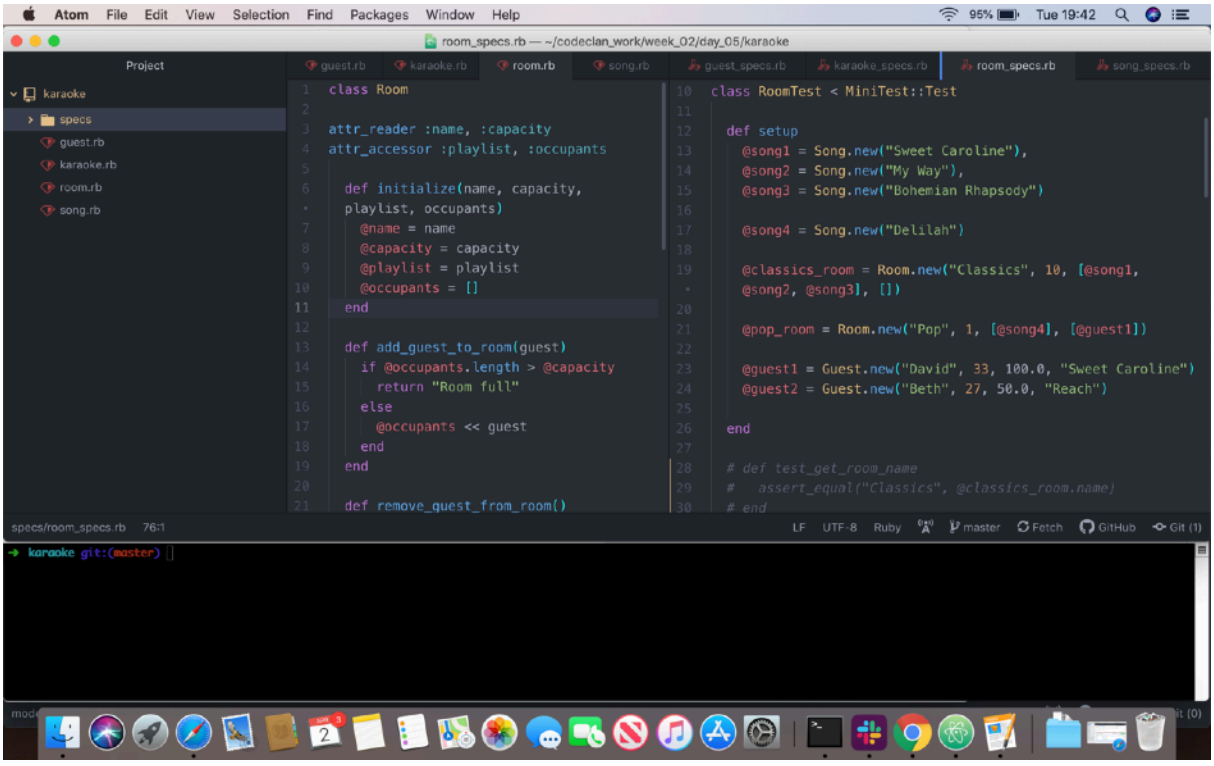
This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

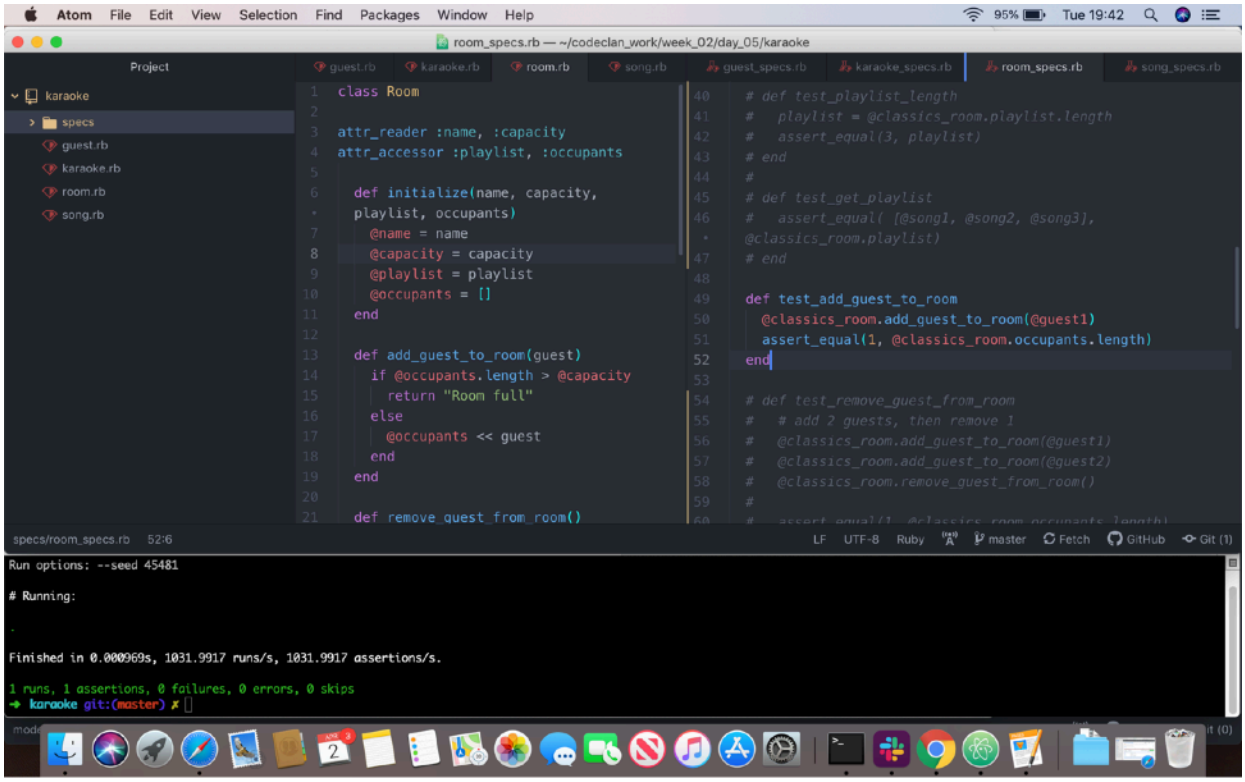
Each point that required details the Assessment Criteria (What you have to show) along with a brief description of the kind of things you should be showing.

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

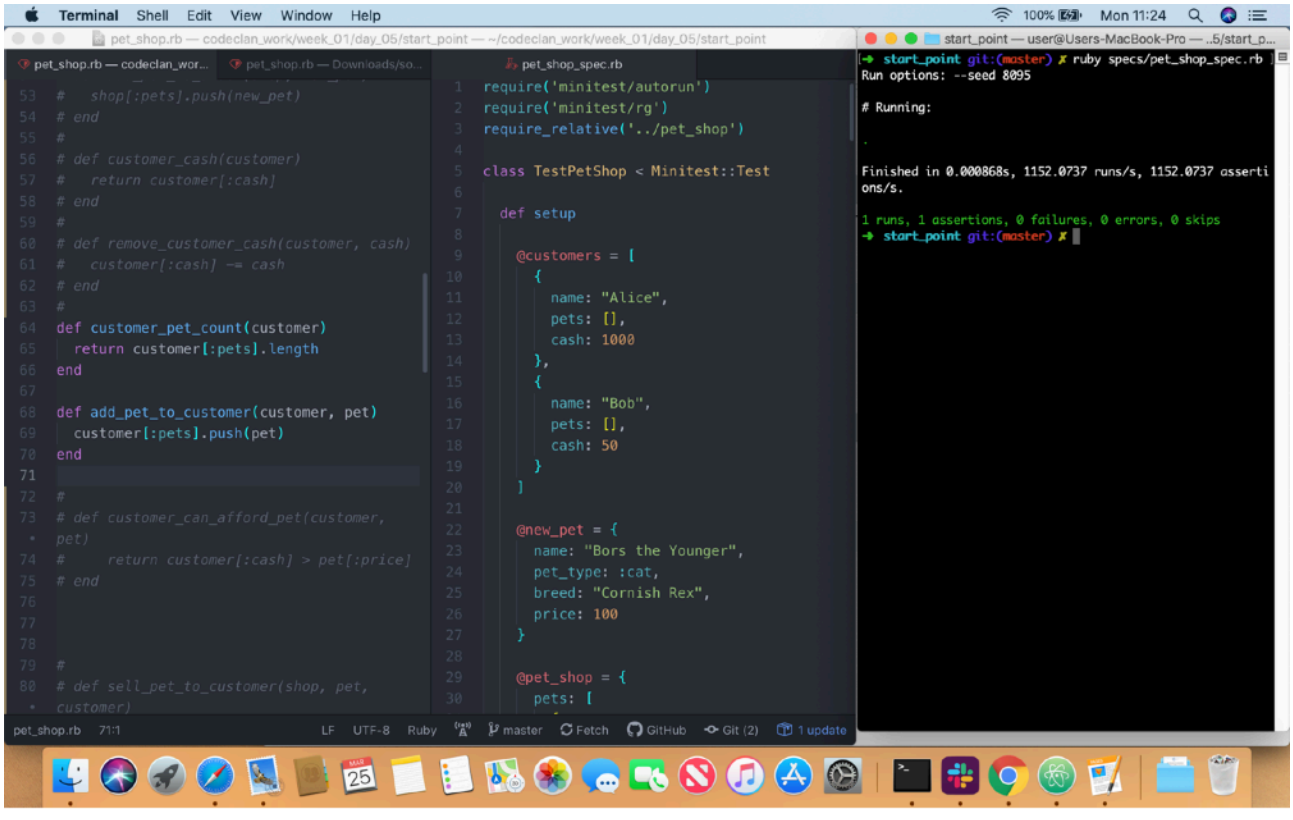
Week 2

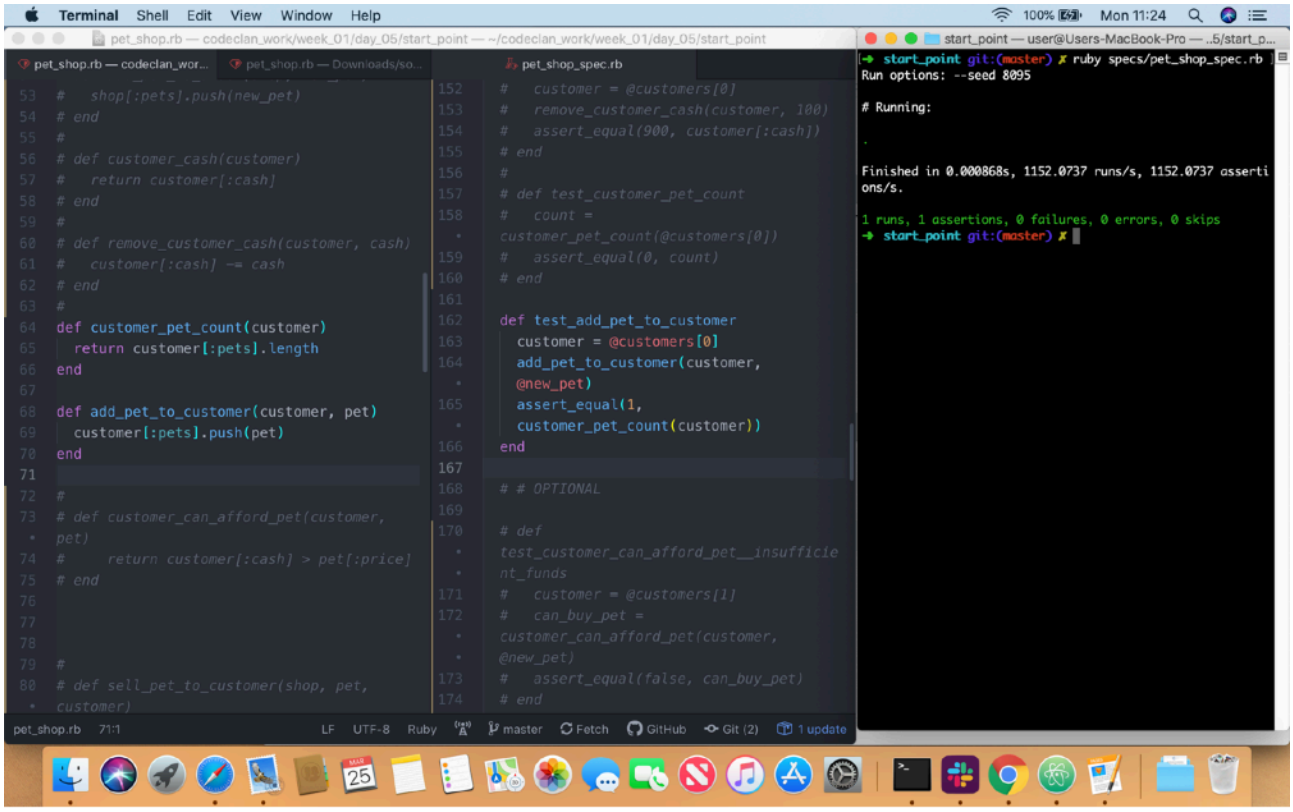
Unit	Ref	Evidence
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running
		Description: This was a program which contained a few arrays. The photos below show the test result of a function which was created to add a guest to a specific room. A guest is added to the empty 'occupants' array which is then counted to show that in now contains 1 item (or guest).





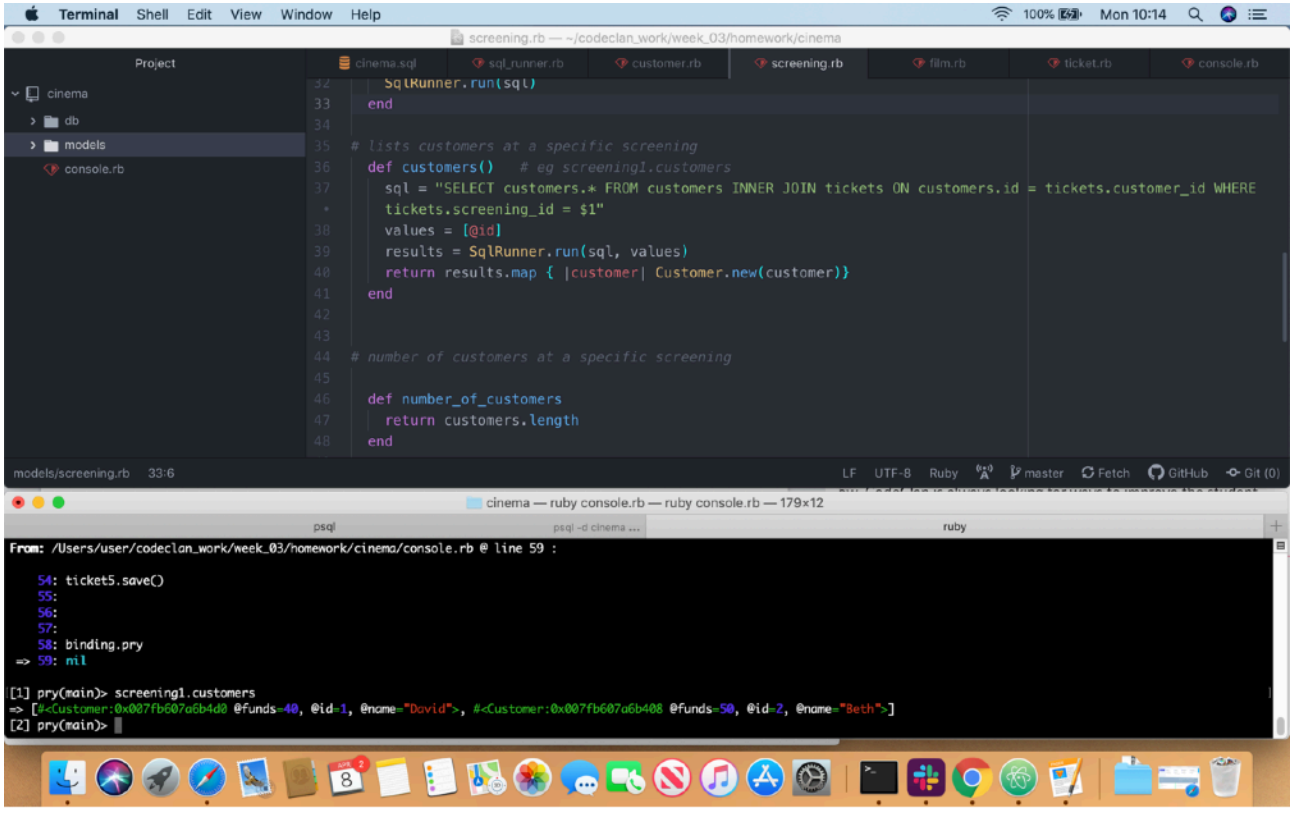
Unit	Ref	Evidence
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running
		Description: This program shows an array of hashes. The test shows the result of adding a pet to the customer pets array contained in a hash.



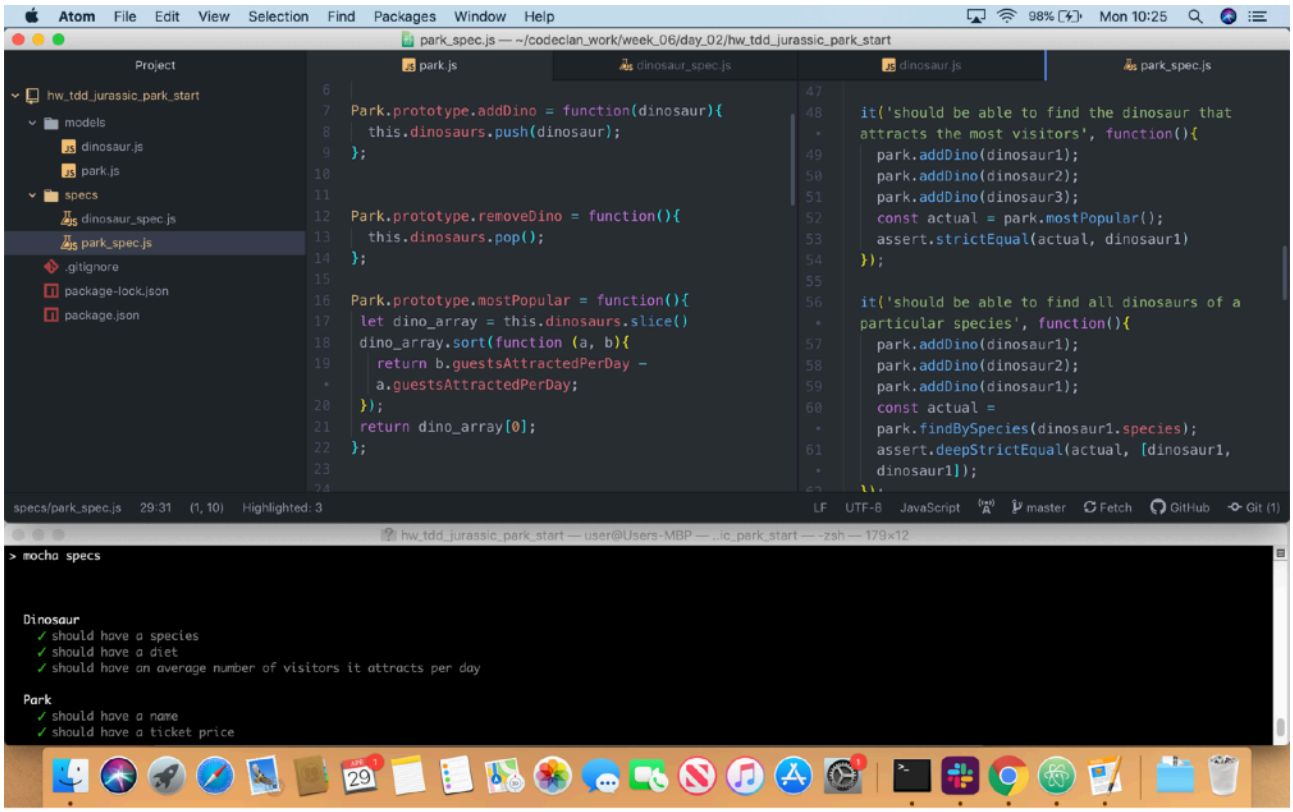


Week 3

Unit	Ref	Evidence
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running
		Description: Below we have created a function that searches for all the customers at a specific screening. The terminal output at the bottom of the screenshot shows the two customers who were at screening 1.

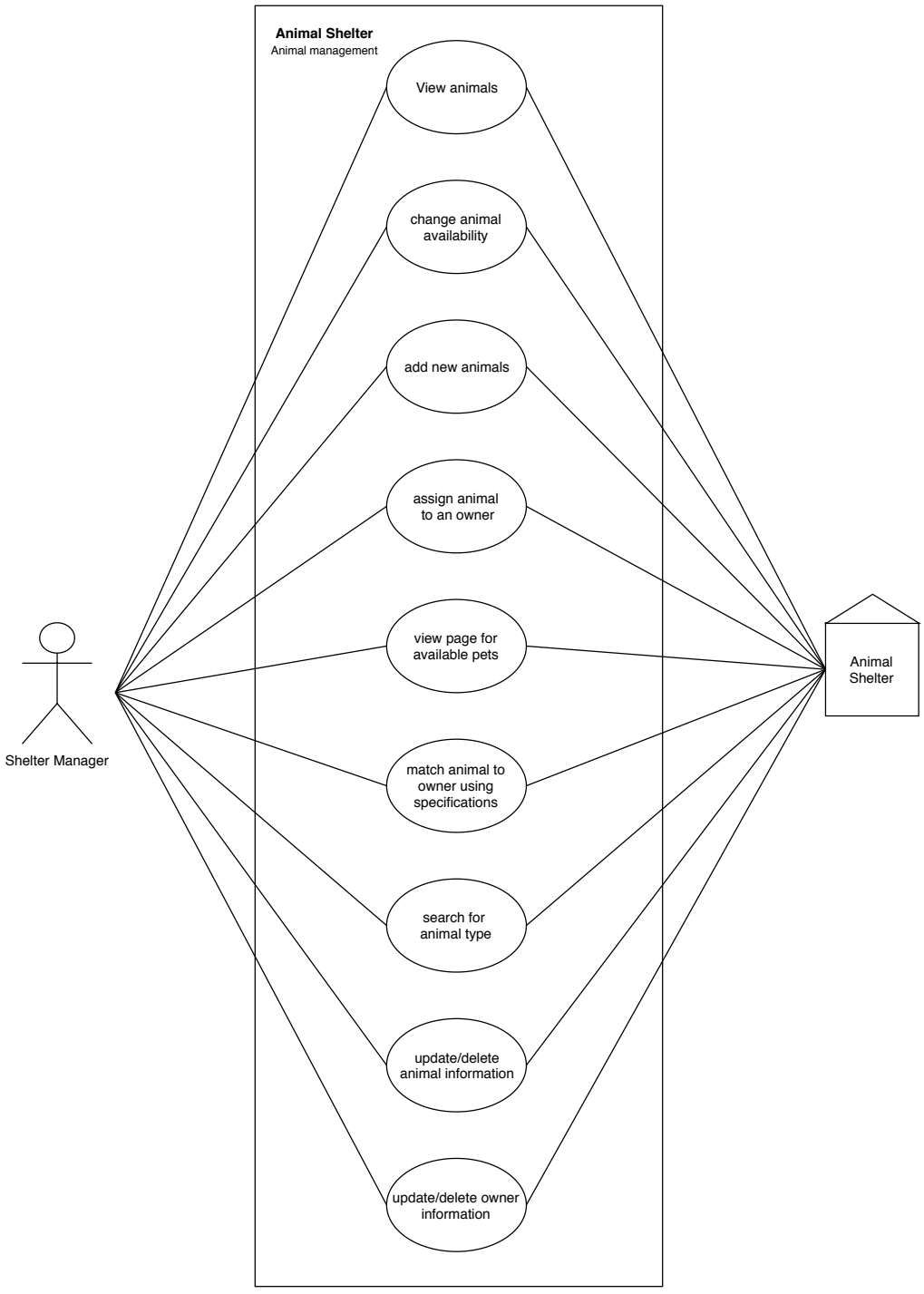


Unit	Ref	Evidence
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running
		Description: The program below has a function to find the dinosaur that attracts the most visitors. To do this I sorted the dinosaurs in the array by the value of guestsAttractedPerDay and then returned the first result. The result in the test spec shows that dinosaur1 was the expected outcome as he had the highest value for guestsAttractedPerDay.

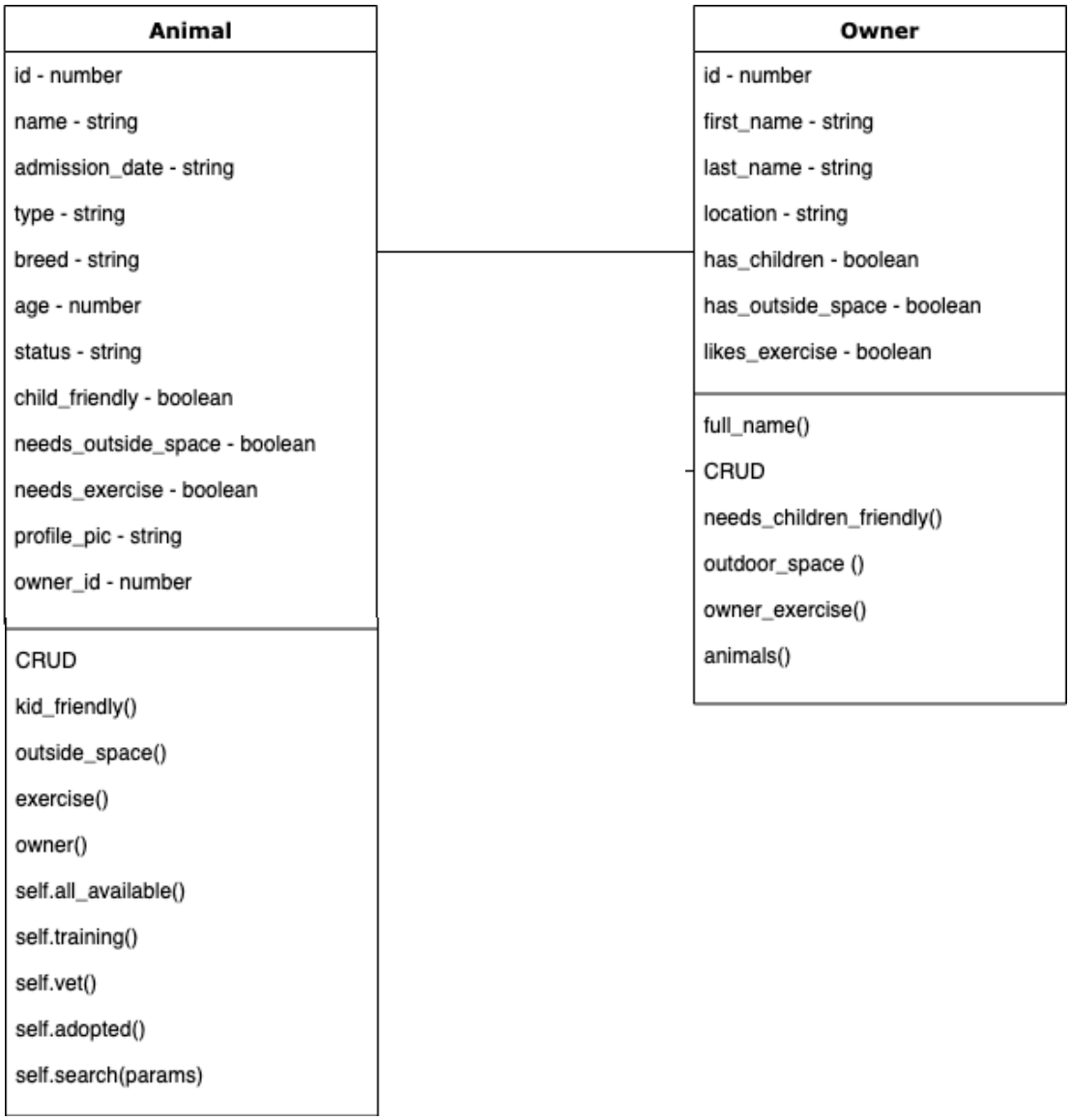


Week 5 and 6

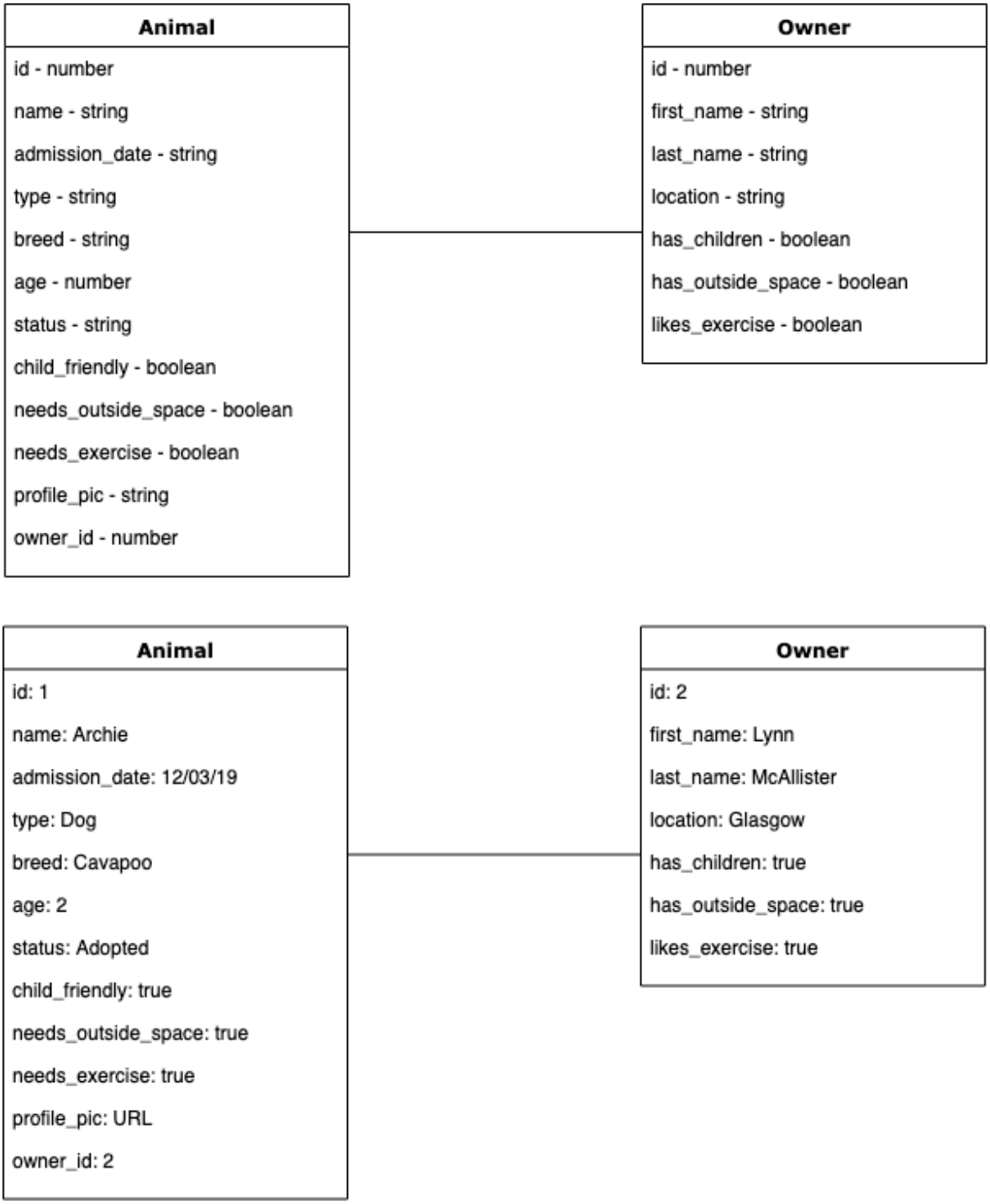
Unit	Ref	Evidence
A&D	A.D.1	A Use Case Diagram
		Description: the use case diagram below shows us all the tasks the shelter manager would need to do on any given day. As the program is not intended to be used by the general public, the shelter manager will be the only user.



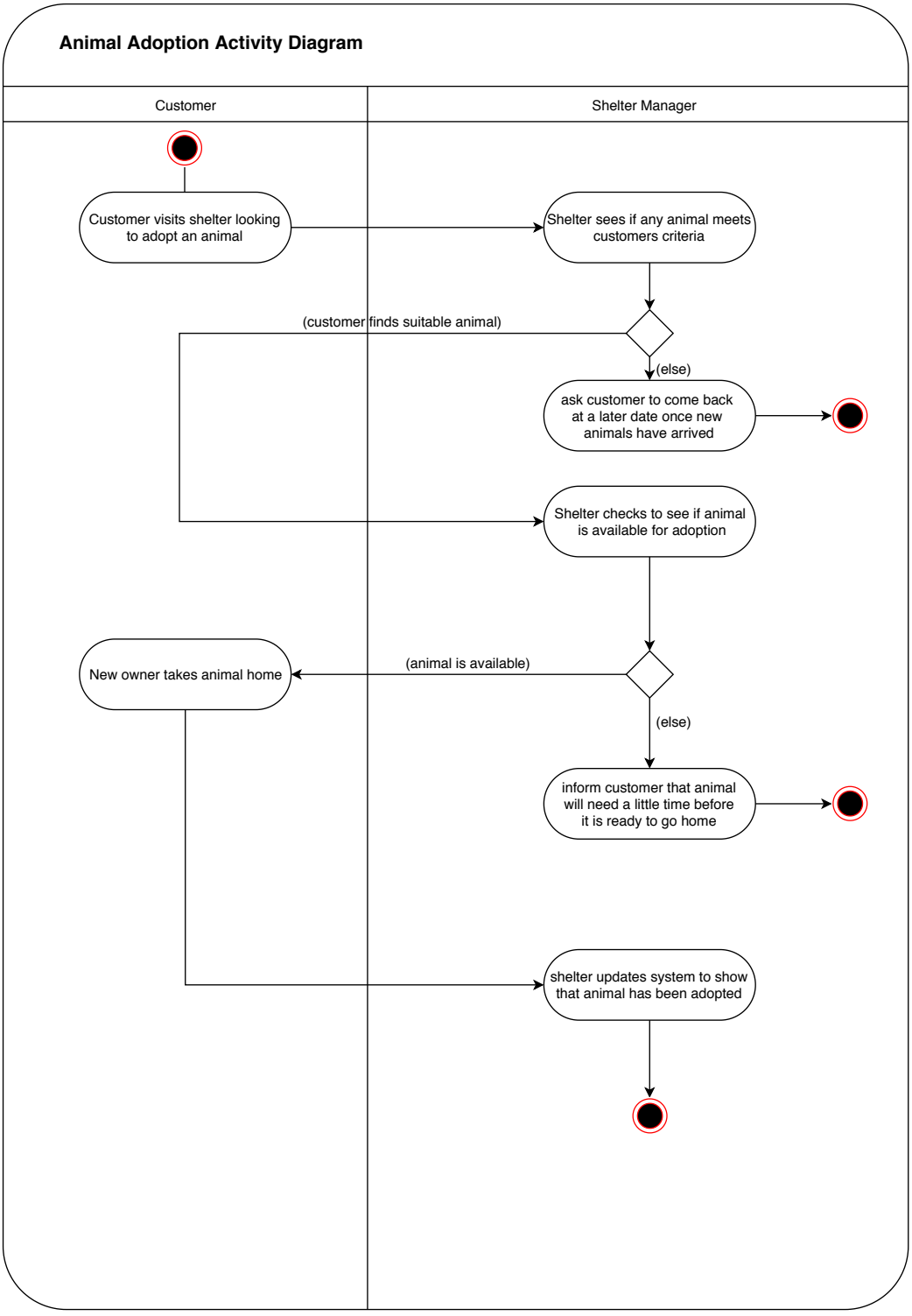
Unit	Ref	Evidence
A&D	A.D.2	A Class Diagram
		Description: The diagram below shows the attributes for each class as well as the methods that will be required. The type of each attribute is also shown.



Unit	Ref	Evidence
A&D	A.D.3	An Object Diagram
		Description: The object diagram shows an example of the data structure for each class.



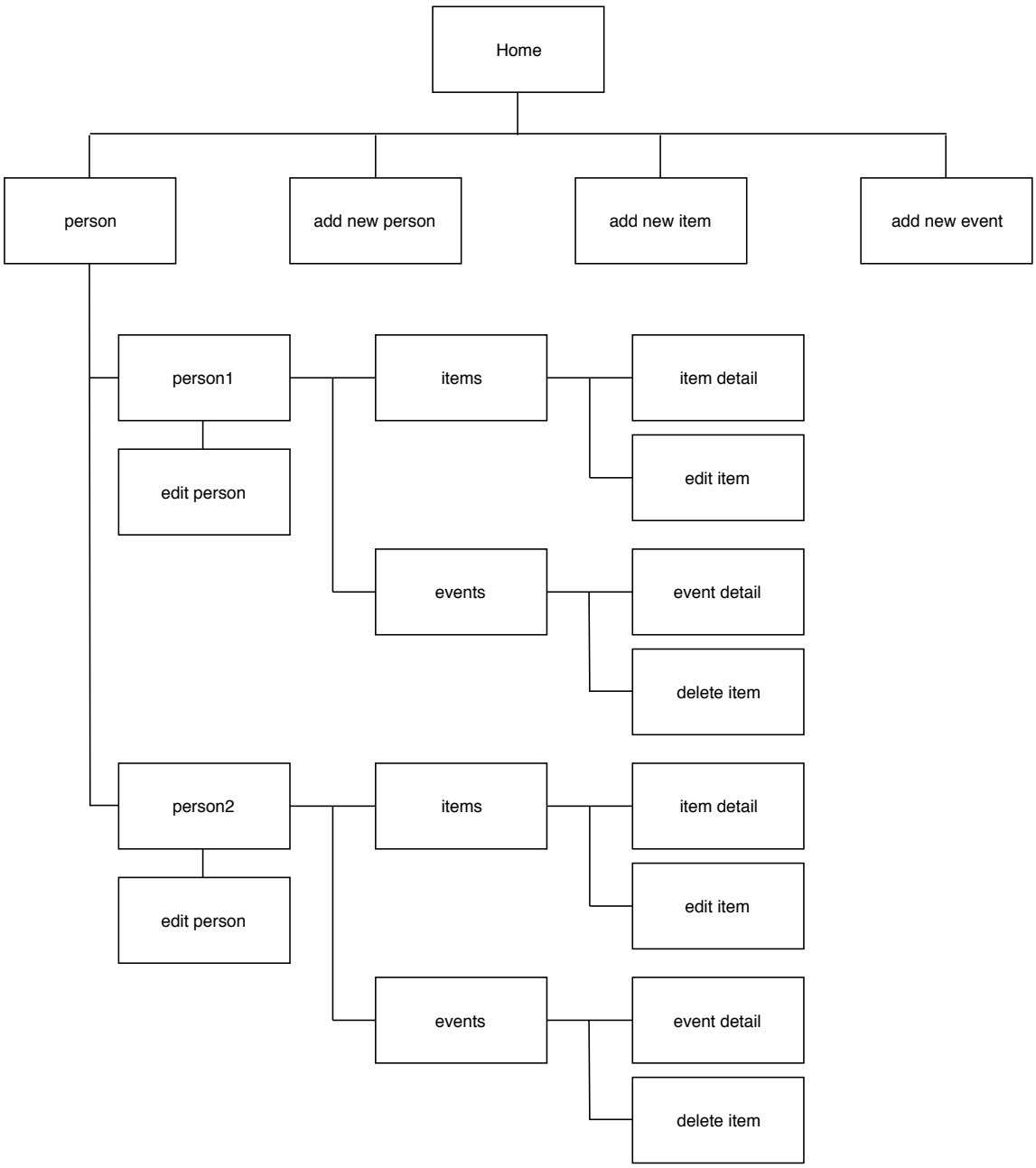
Unit	Ref	Evidence
A&D	A.D.4	An Activity Diagram
		Description: The activity diagram below shows the main journey, from start to finish, for an adoption at the shelter. The diagram shows the outcome from each decision/choice and how that will affect the final outcome.



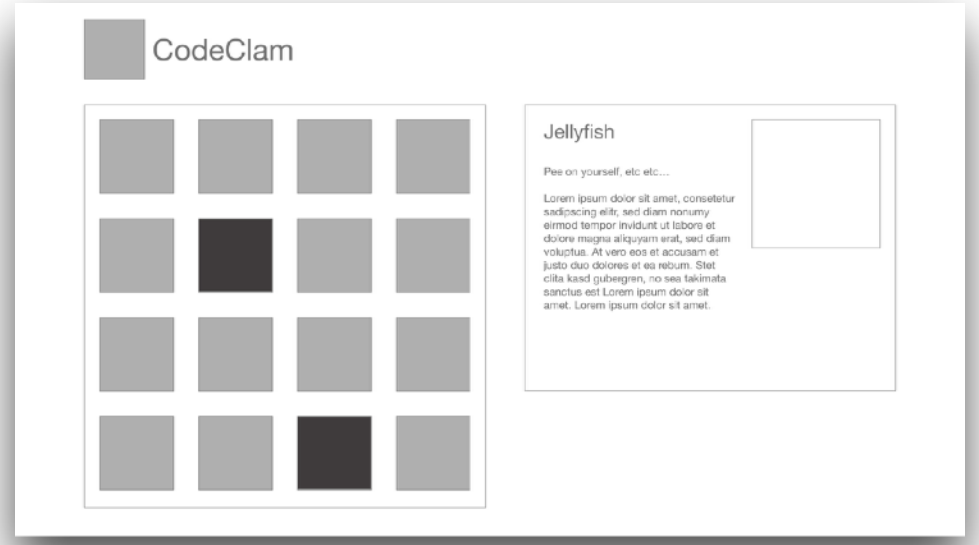
Unit	Ref	Evidence
A&D	A.D.6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time
		Description: Below is an example of an implementations constraints plan. As the website was designed for an imaginary Animal Shelter with no plans to upscale it, many of these potential issues were void.

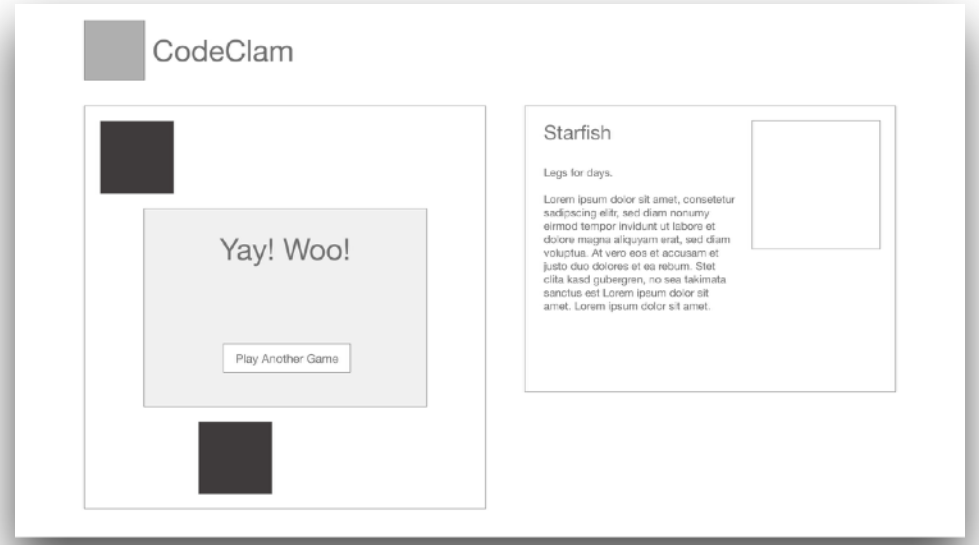
Constraint Strategy	Implementation constraints	Solution
Hardware and software platforms	No issue with hardware, laptop will be sufficient to build app. By using software we know how to use, will ensure there should be minimal disruption in the design process.	Used Ruby, Sinatra and PostgreSQL which will be easy to set up and maintain.
Performance requirements	App must update instantly to keep records up to date.	Ensured all routes and CRUD methods are working to ensure there will be no issues once the app is launched.
Persistent storage and transactions	Small scale app with no plans to upscale so a small server is more than sufficient.	Storage on Heroku will be sufficient for the purposes of this app.
Usability	App must be simple and clear how to use. It is being asked for to simplify and manage the pets and owners for the animal shelter.	Clear distinction between sections. All routes are clearly labelled.
Budgets	No budget. Need to work on project without any hired support.	Use free storage facilities.
Time	Only had a week to plan and build the application.	Keep the design simple. Done is better than perfect.

Unit	Ref	Evidence
P	P.5	User Site Map
		Description: This site map is for the present planner wish list website we created in the last project. It shows the different pages a user could navigate to when using the website.



Unit	Ref	Evidence
P	P.6	2 Wireframe Diagrams
		Description:The diagrams below show examples of wireframe diagrams for the Codeclam matching pairs game project. The diagram show an early prototype for what the main page was going to be like as well as the final page on game completion. As the project grew the design did change but the use of the wireframes early on, certainly had a major impact on how the final page looked.

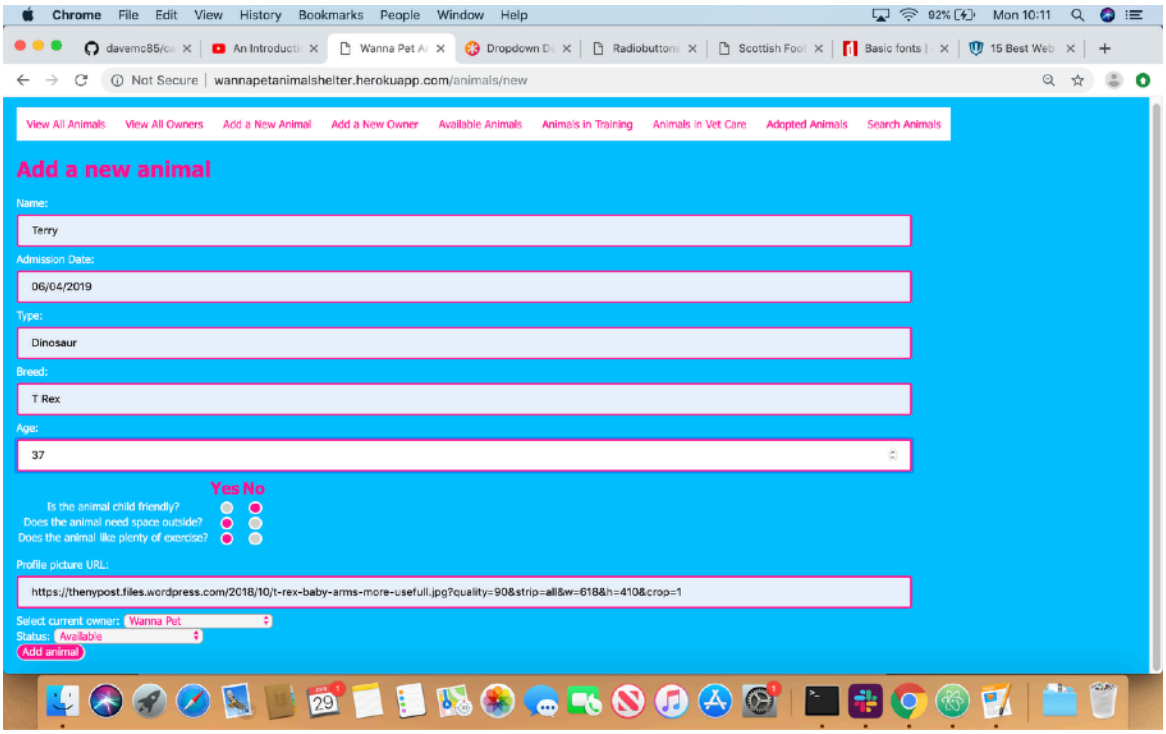


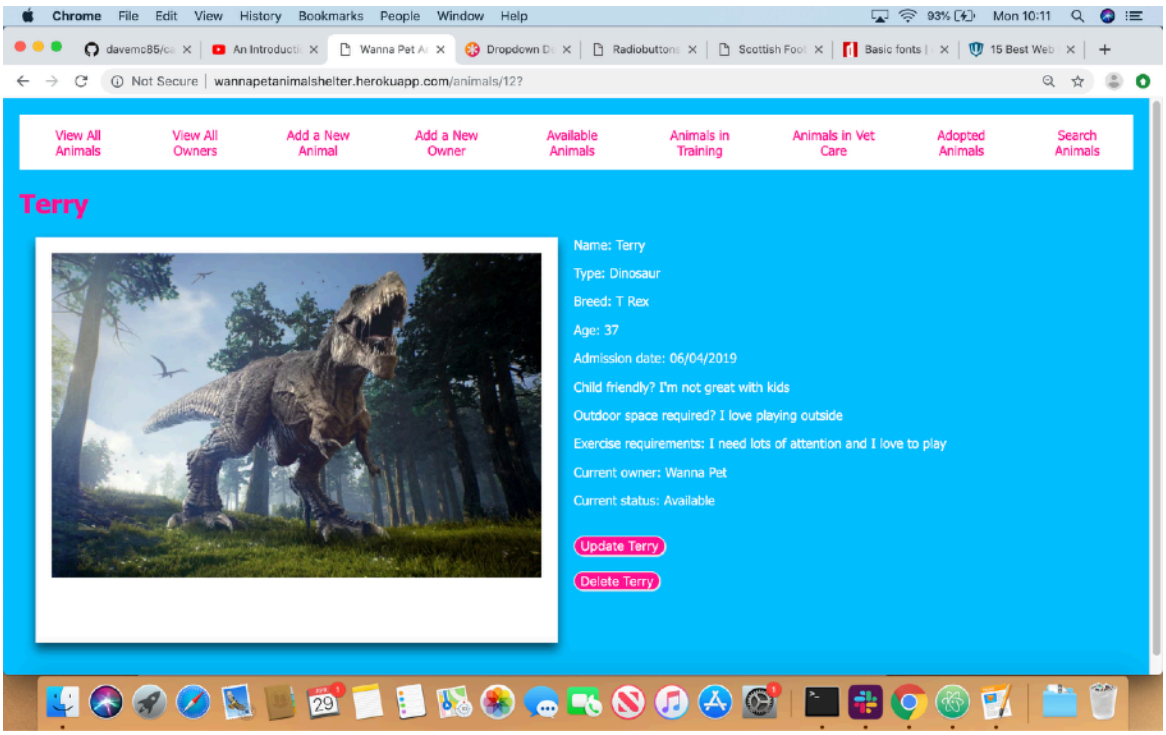


Unit	Ref	Evidence
P	P.10	Example of Pseudocode used for a method
		Description: The pseudocode here just describes exactly what the method is going to do. It will work through each if statement and if the change is greater it will add a coin to the returned coin array and move on to the next coin. It loops round the 20p coin twice as the change could be 40 as it reaches this point and as a result two 20p coins will be required.

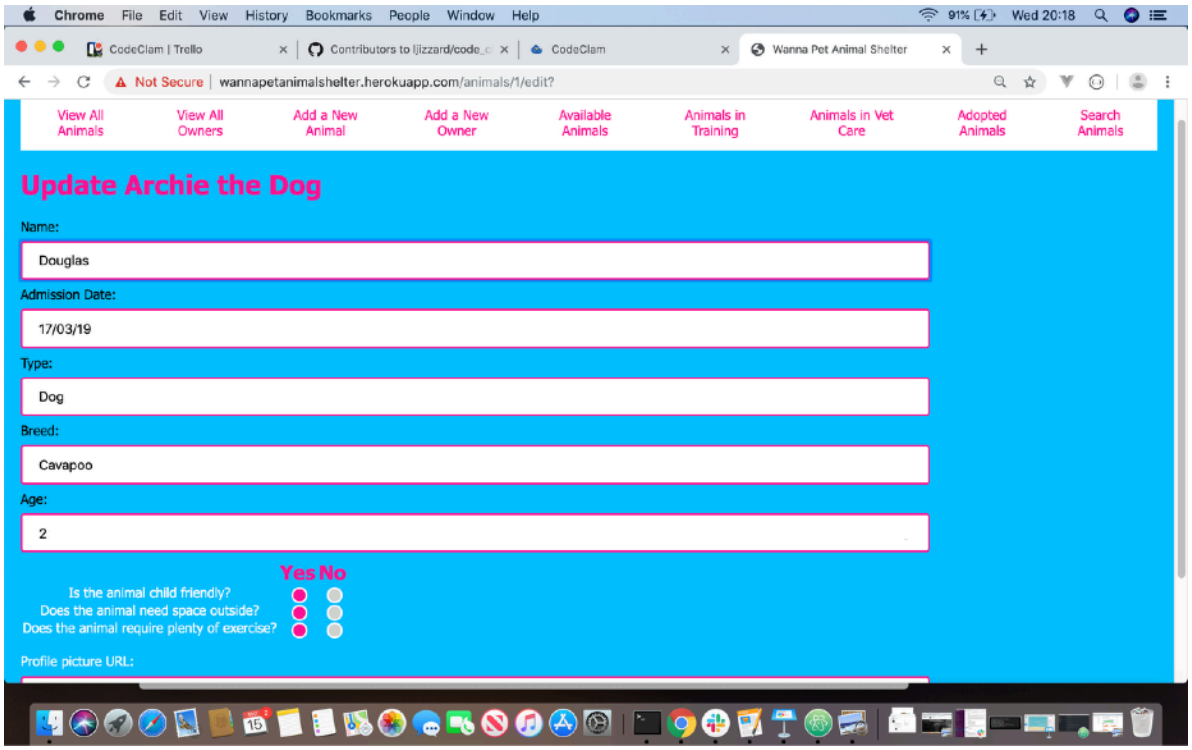
```
// this method is used to calculate what coins would be returned
// for a given amount of change

public void workOutChange(int change){
     ArrayList<Coin> returnedCoins = new ArrayList<Coin>();
    if(change >= 200){ // if change is more than 200 a two pound coin can be issued
        this.returnedCoins.add(twoPound);
        change -= 200;
    } if(change >= 100){ // if change is more than 100 a pound coin can be issued
        this.returnedCoins.add(pound);
        change -= 100;
    } if (change >= 50){ // if change is more than 50 a 50p coin can be issued
        this.returnedCoins.add(fifty);
        change -= 50;
    } if (change >= 20){// if change is more than 20 a 20p coin can be issued
        this.returnedCoins.add(twenty);
        change -= 20;
        if (change >= 20){ // if change is still more than 20 a 20p coin can be issued ie 40p change
            this.returnedCoins.add(twenty);
            change -= 20;
        }
    } if (change >= 10){ // if change is more than 10 a 10p coin can be issued
        this.returnedCoins.add(ten);
        change -= 10;
    } if (change >= 5){ //// if change is more than 5 a 5p coin can be issued
        this.returnedCoins.add(five);
        change -= 5;
    }
}
```

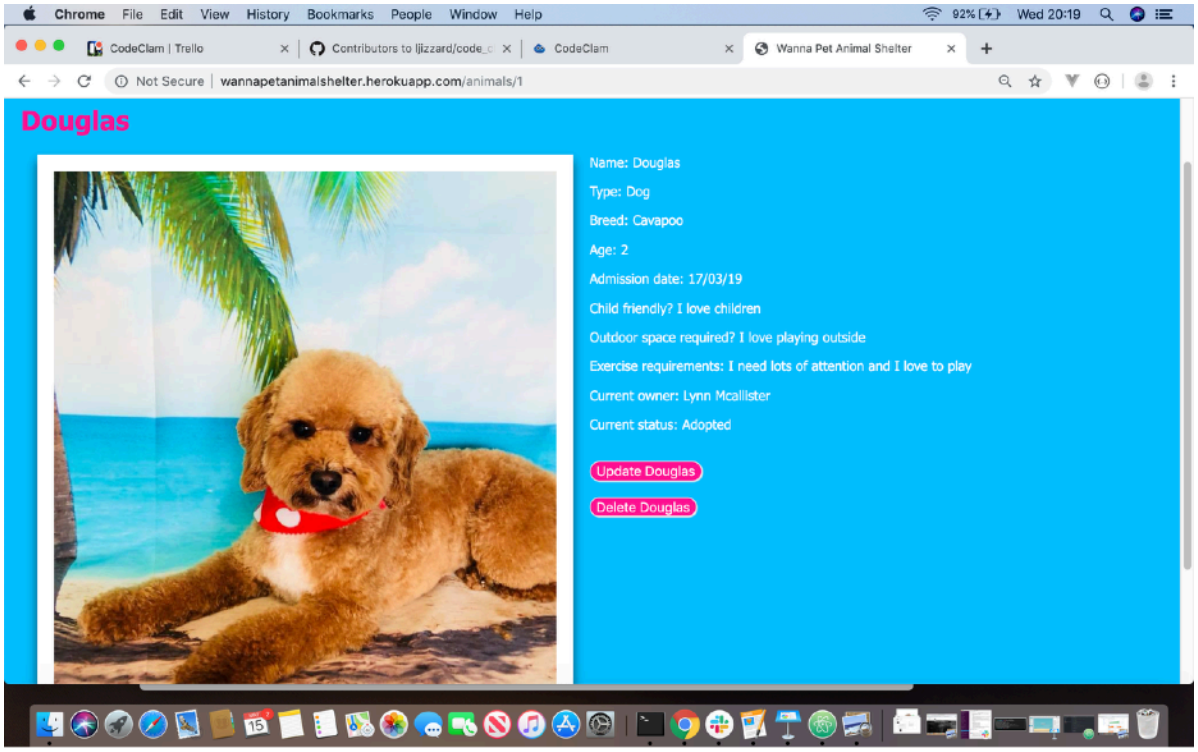
Unit	Ref	Evidence
P	P.13	Show user input being processed according to design requirements. Take a screenshot of: * The user inputting something into your program * The user input being saved or used in some way
		Description: The brief asked us to be able to add new animals to the shelter, Below shows the user form inputting an entry for a new dinosaur to the shelter. The bottom image shows the profile of Terry the T Rex after it has been saved to the database.



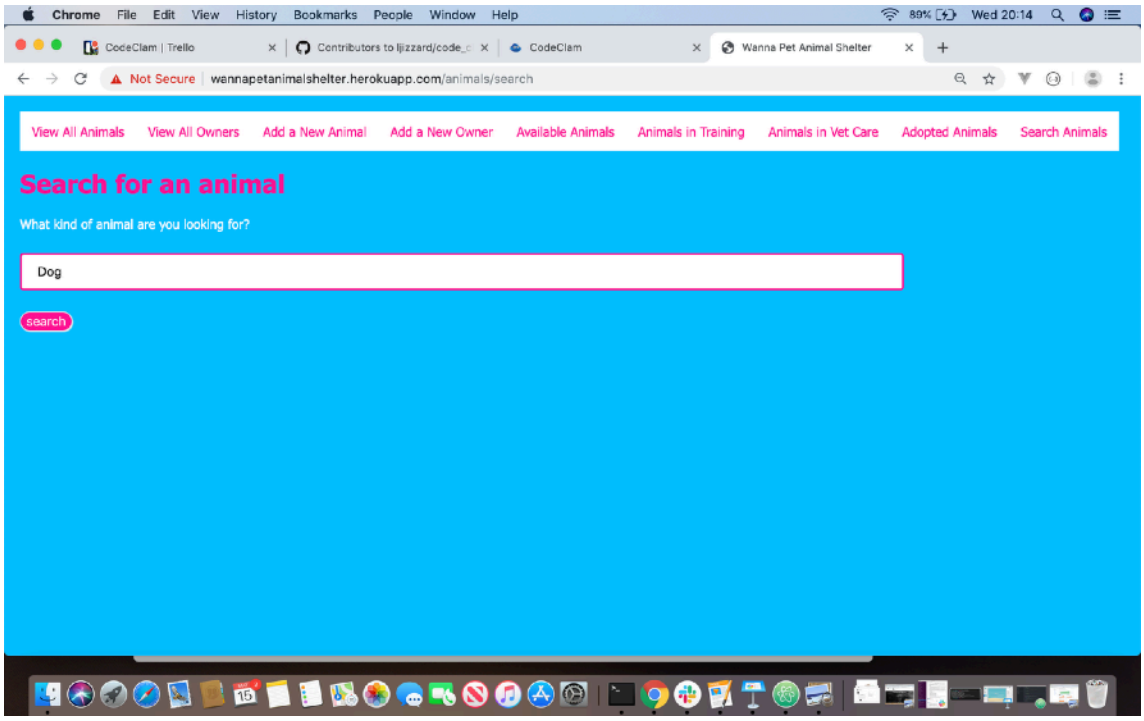


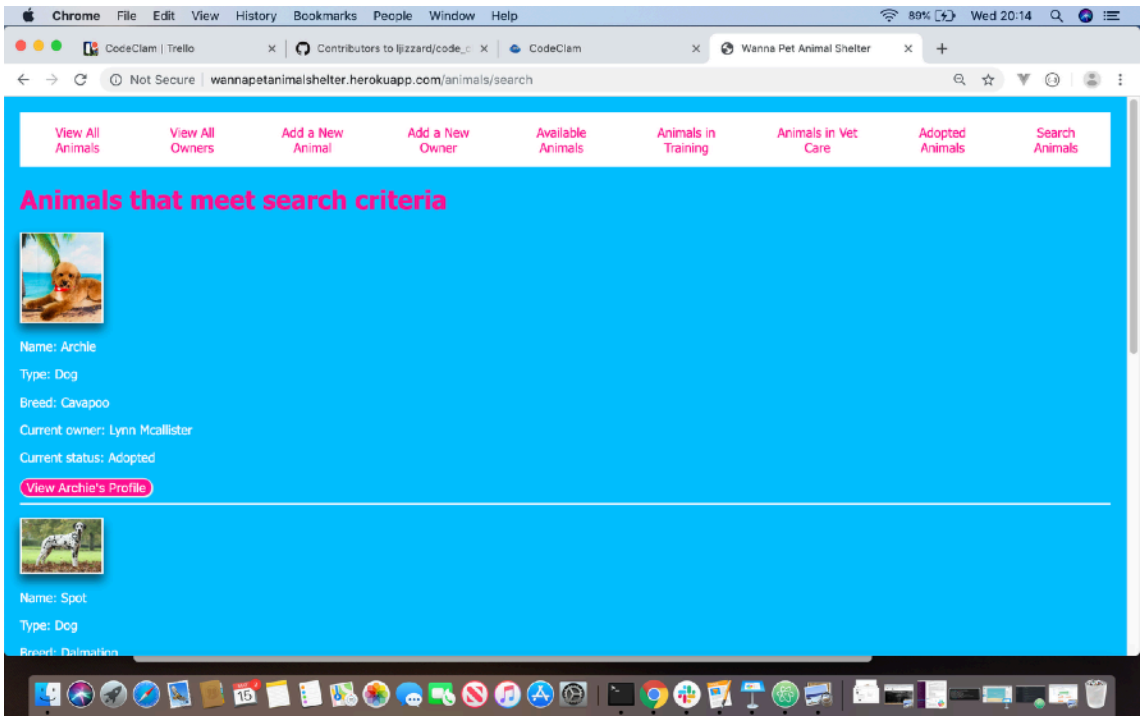
Unit	Ref	Evidence
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved
		Description:Below shows a user updating the details of one of the animals at the shelter. The dog has had its name changed from Archie to Douglas. The result of this can be seen in the bottom image.



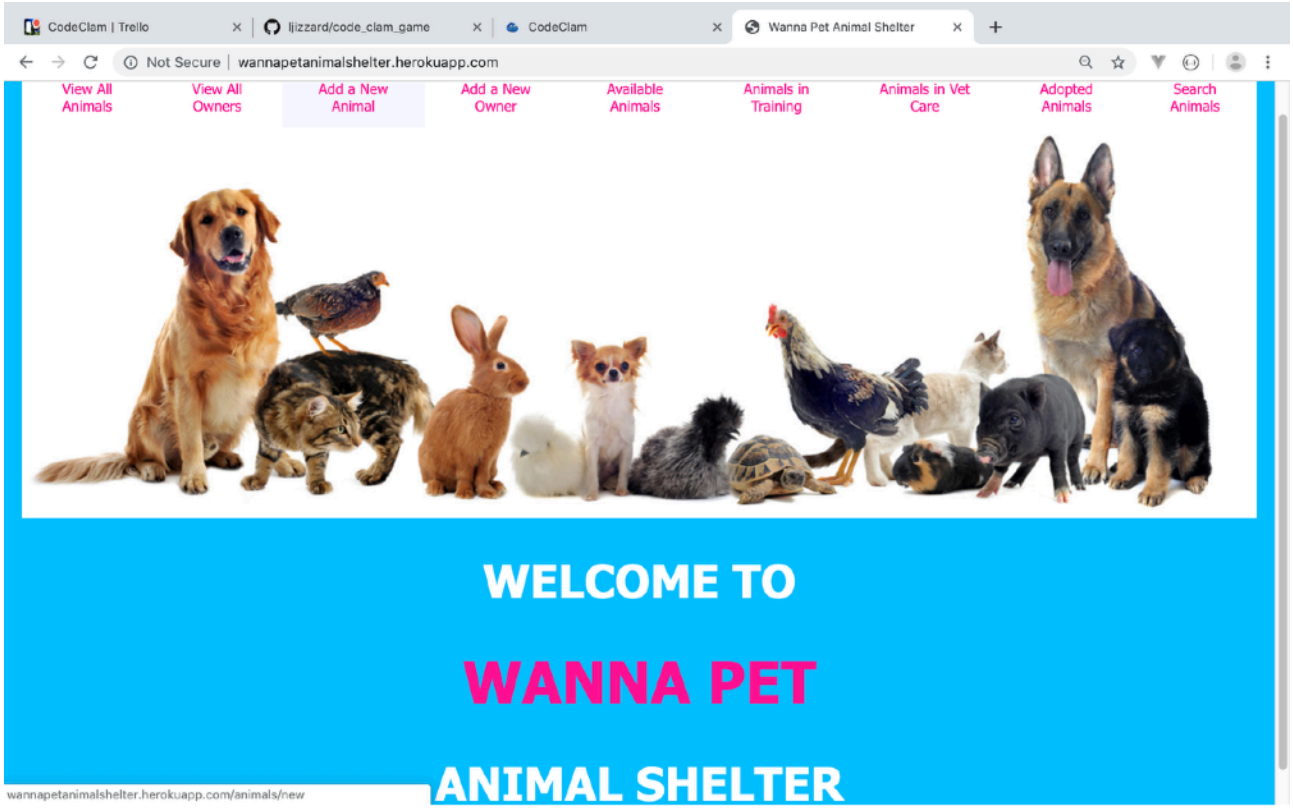


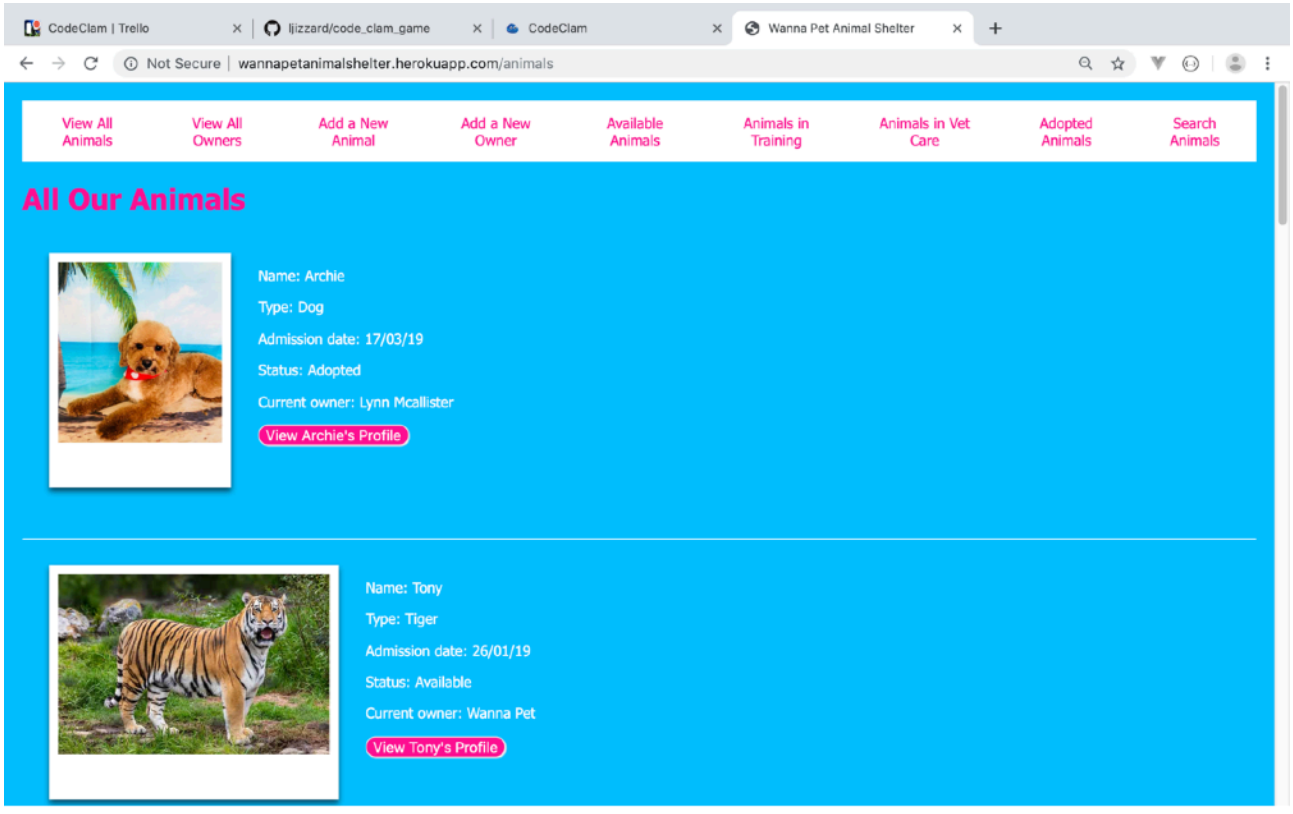
Unit	Ref	Evidence
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program
		Description: The images below show how the search functionality worked on the website. The user has searched for dog and the results page has returned the details of the four dogs the shelter has on its books.





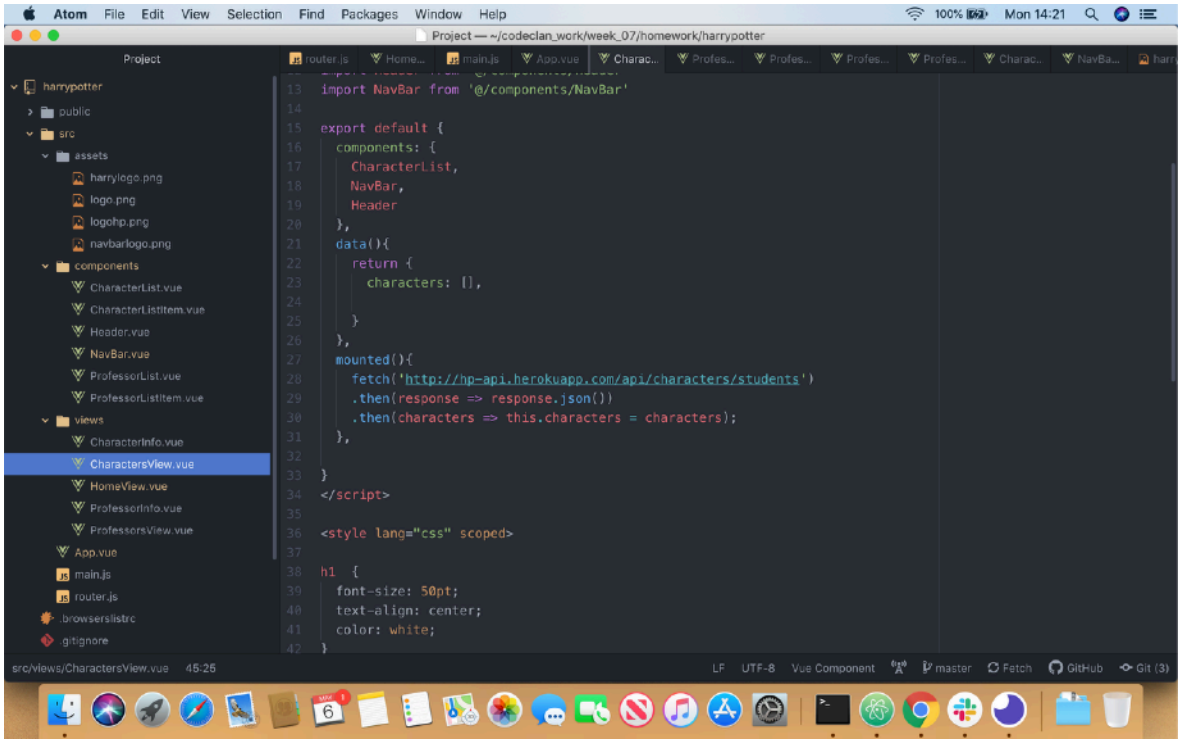
Unit	Ref	Evidence
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.
		Description: https://github.com/davemc85/animal_shelter





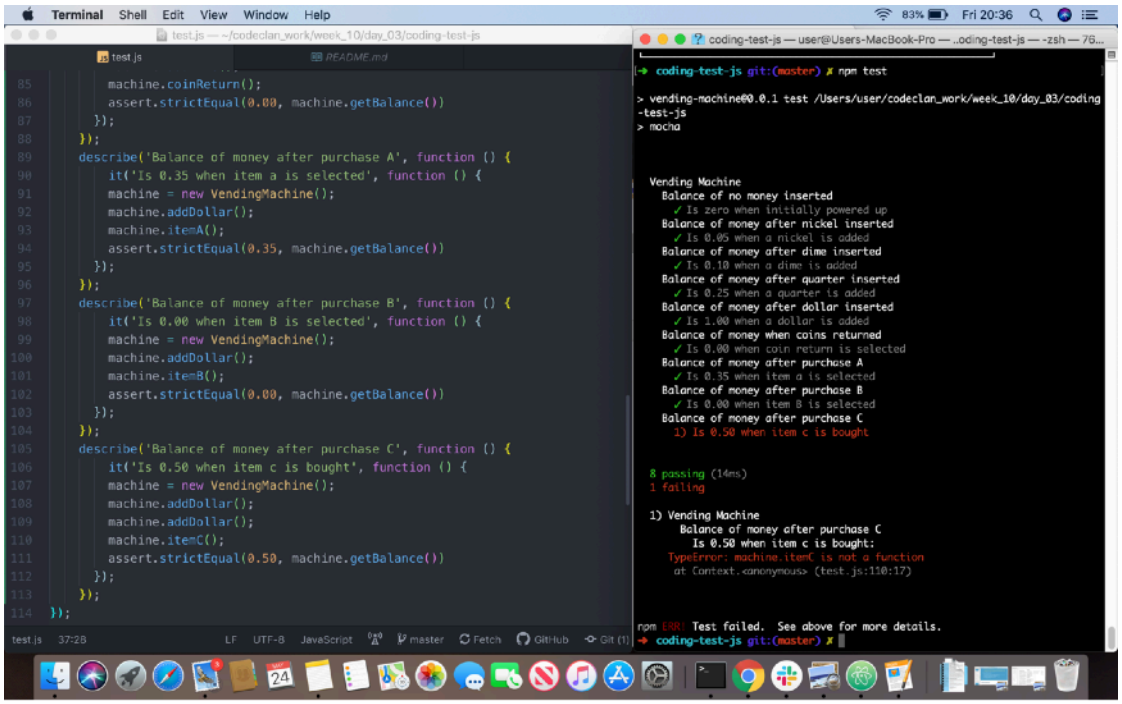
Week 7

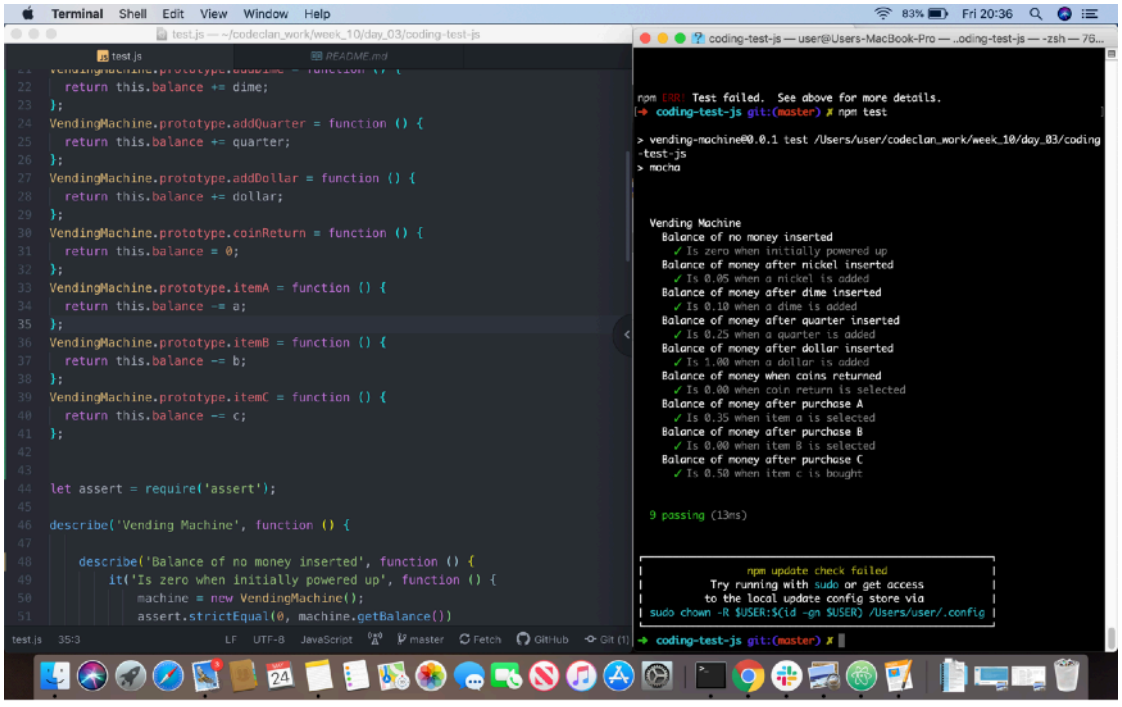
Unit	Ref	Evidence
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running
		Description: The API below has details of characters from the Harry Potter franchise. When a character name is selected, more information about that character is displayed.




Chrome File Edit View History Bookmarks People Window Help
Mon 14:23
localhost:8081/#/characterinfo
Home
View All Students
View All Professors
Harry Potter
House: Gryffindor
Date of Birth: 31-07-1980
Ancestry: half-blood
Patronus: stag
Played By: Daniel Radcliffe

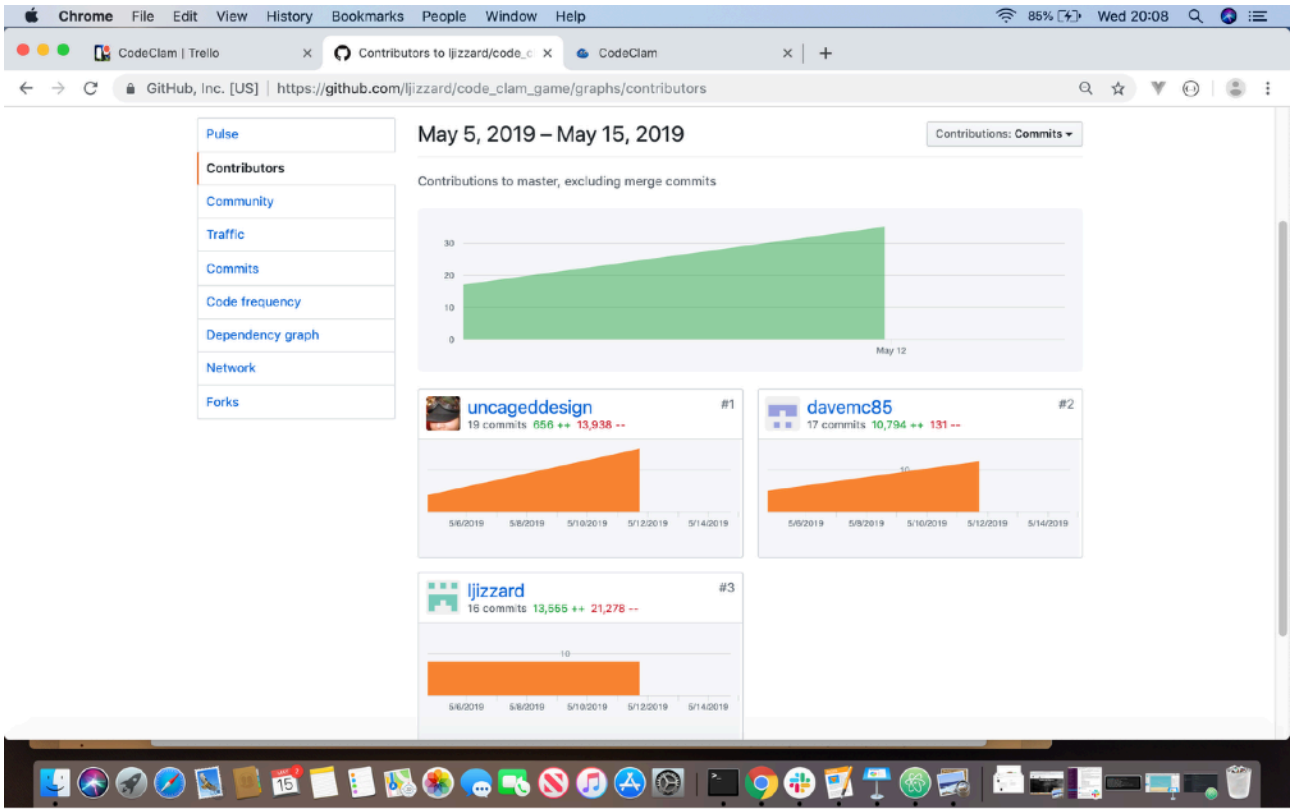
Unit	Ref	Evidence
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing
		Description: the below example shows a newly written test failing as the function which would allow the test to pass has not been written yet. Once the function has been written so that item C can be purchased, the test now can pass.



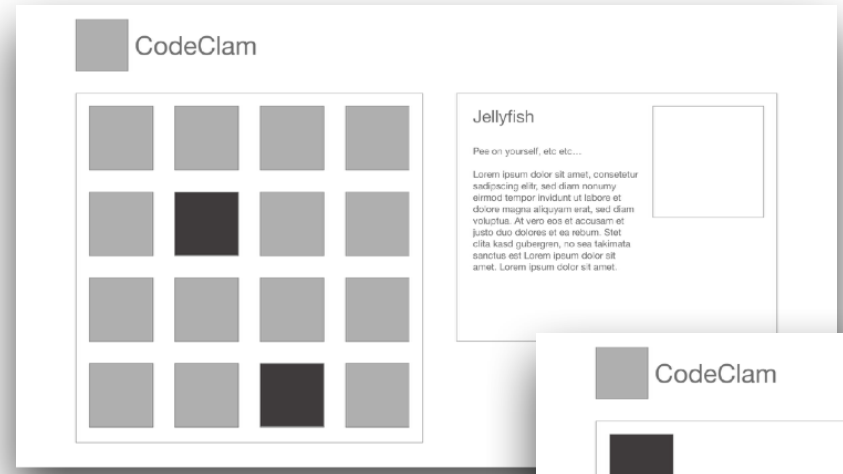


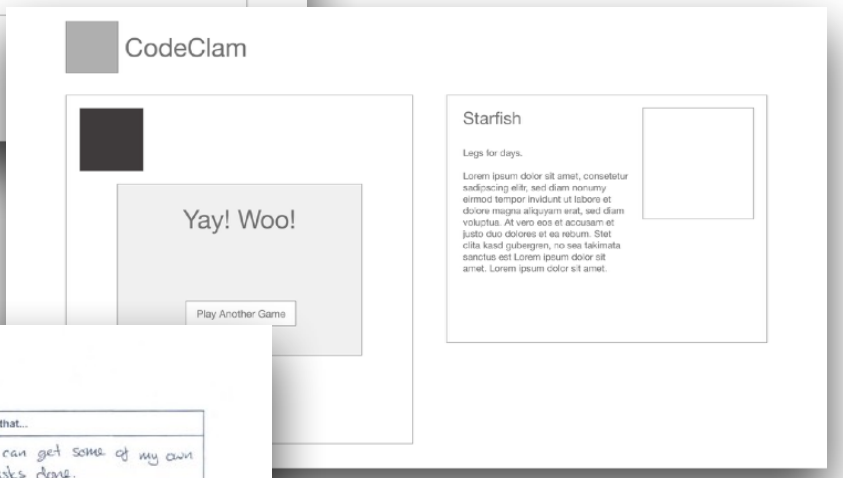
Week 9

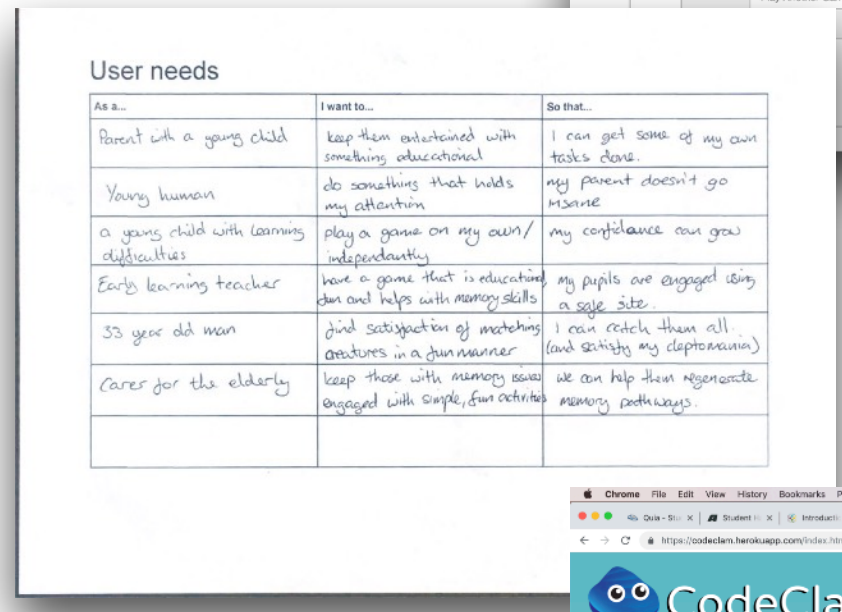
Unit	Ref	Evidence
P	P.1	Take a screenshot of the contributor's page on Github from your group project to show the team you worked with.
		Description: This screenshot below shows my group for the project. uncageddesign is Mhairi, ljizzard is Lindsey and davemc85 is myself



Unit	Ref	Evidence
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description: Here are some of the images from our planning for the Codeclam project. The initial wireframe diagrams show our initial layout idea and the bottom image shows how the final page looked. The user needs shows a sample of our users and what they would expect from the game.

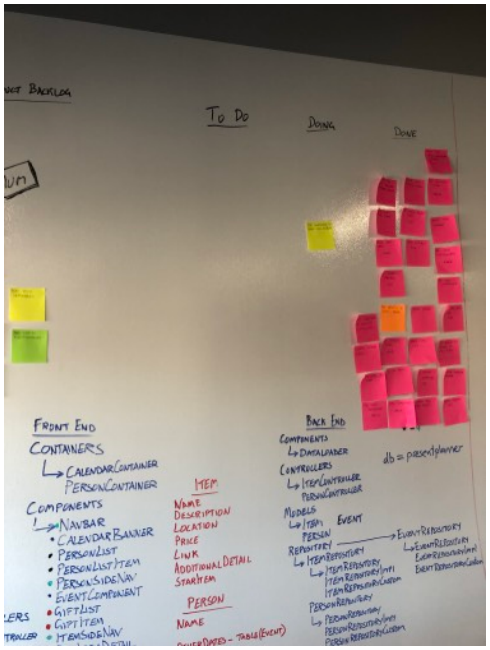


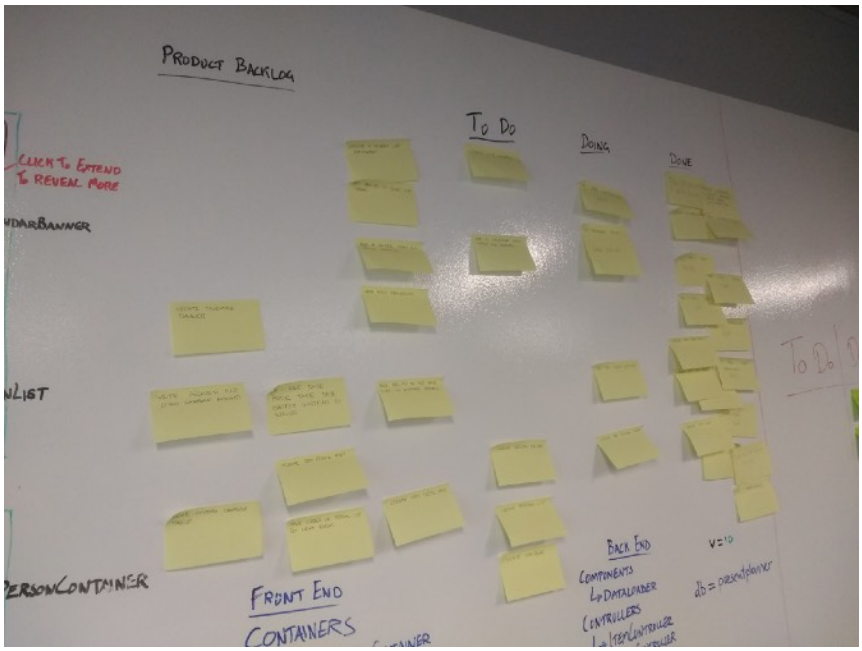




Unit	Ref	Evidence
P	P.2	Take a screenshot of the project brief from your group project.
		Description: The initial brief was to create a card or dice game so we had to create our own MVP and extensions. Below shows our final decision on what we should try and achieve in the week.

Unit	Ref	Evidence
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.
		Description: Below are photos of our agile board. You can see our initial ideas, our brief with MVP and extensions, wireframe diagrams, file structure, project backlog and job list. This was updated throughout the process to keep up to date with any changes.





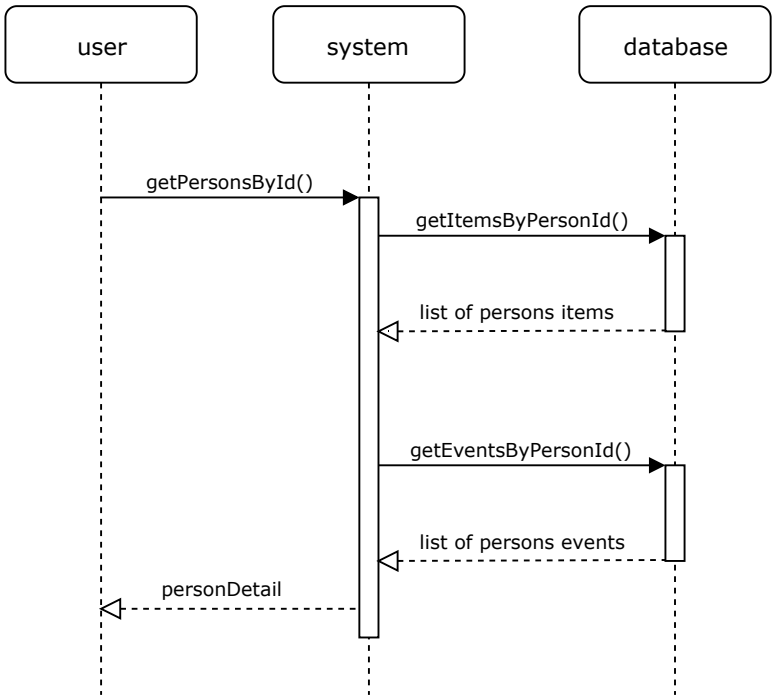
Unit	Ref	Evidence
P	P.4	Write an acceptance criteria and test plan.
		Description: Here is our group acceptance criteria and test plan. Each row has an element that fulfilled part of our Codeclam brief. As each part passed it meant our game was a step closer to completion.

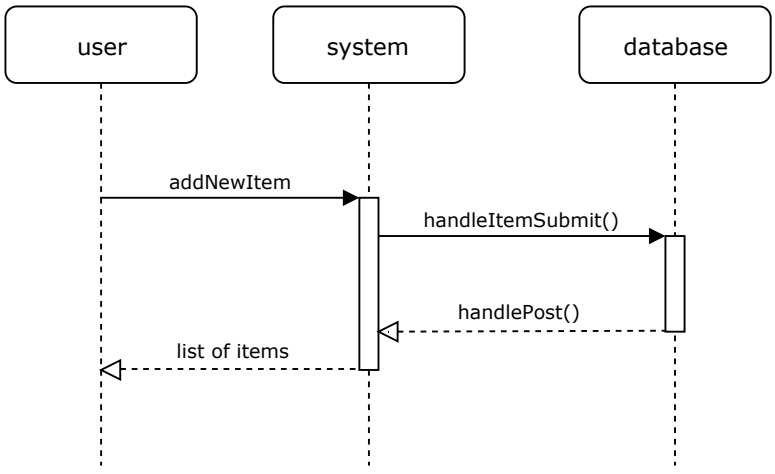
Acceptance Criteria

Stage: Initial Planning

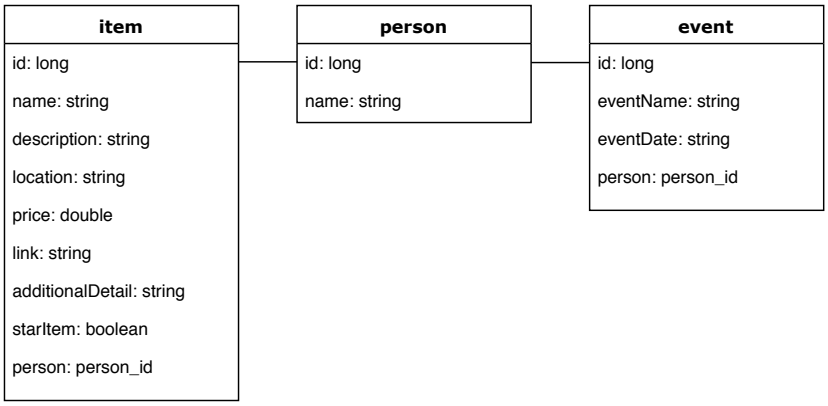
Acceptance Criteria	Expected Result/Output	Pass / Fail
Cards can be displayed in grid	Grid of 4 x 4	**Pass**
Card can be selected	Card changes class on click	**Pass**
Only two cards can be selected	Third click does nothing	**Pass**
If cards match they dissapear	Set opacity to 0	**Pass**
Cards reset if no match	Cards turn back over	**Pass**
Random fact generated	On page refresh	**Pass**
Game resets on completion	Auto refresh after timeout	**Pass**

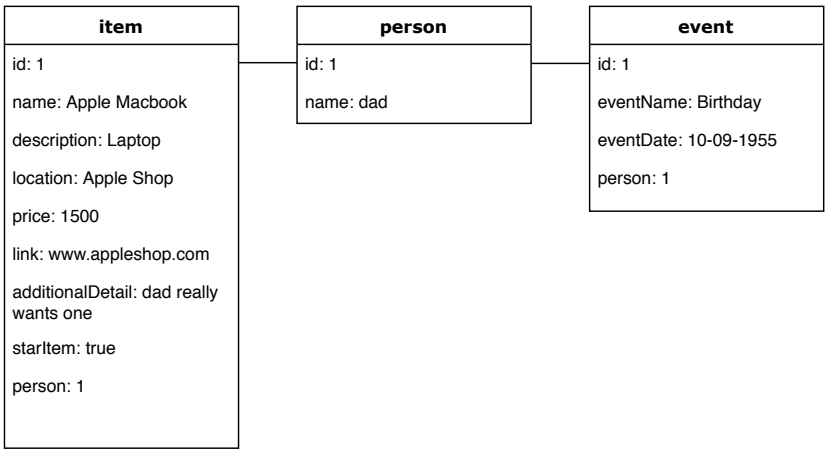
Unit	Ref	Evidence
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).
		Description: diagram1: this shows the interaction that occurs when the user clicks on a specific person. Behind the scenes the system pulls all the items and all the events assigned to that person and displays them in lists. **Diagram 2: this is a similar sequence diagram. It shows the interaction that would occur if the user wanted to add a new item to the list.**

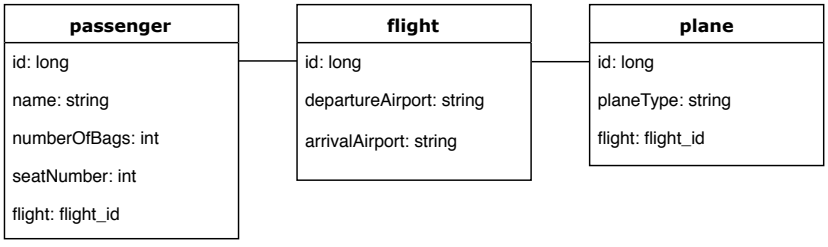




Unit	Ref	Evidence
P	P.8	Produce two object diagrams.
		Description: The object diagrams below show examples of the data structures for each class. The top example I for a present planner wish list where a person is assigned to an item and an event in a one to many relationship /9as in one person can have many items/events). The bottom example is of a similar set up but uses the airport example where a flight has a plane and many passengers.





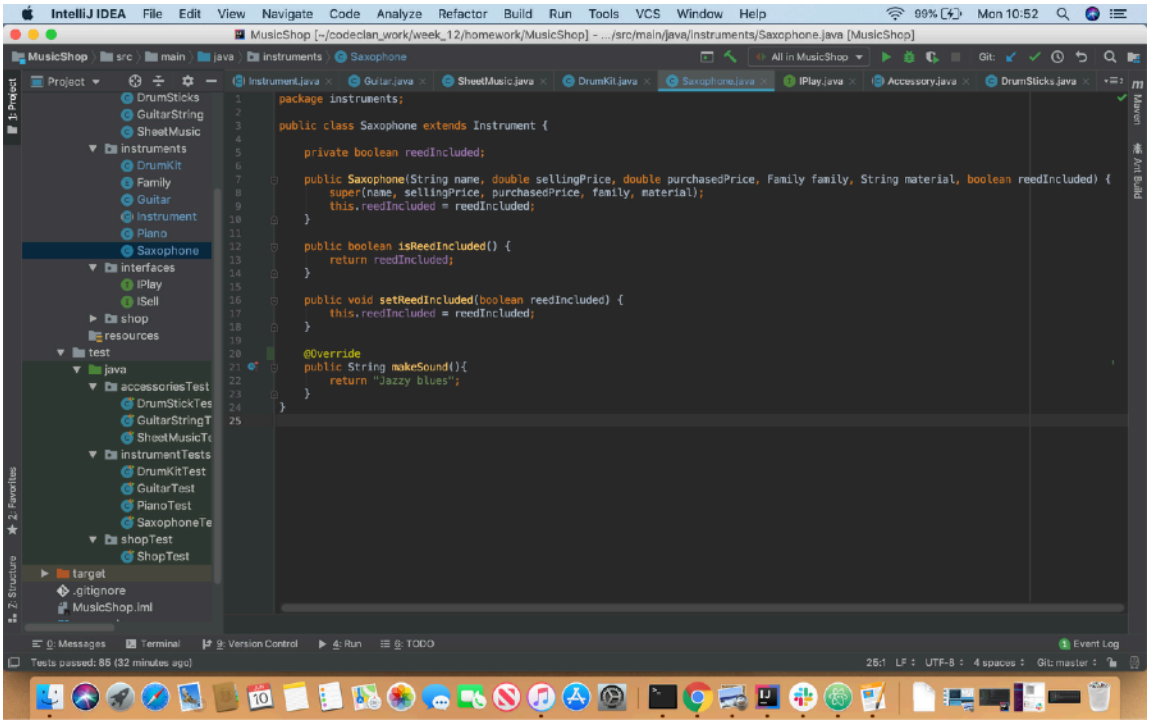


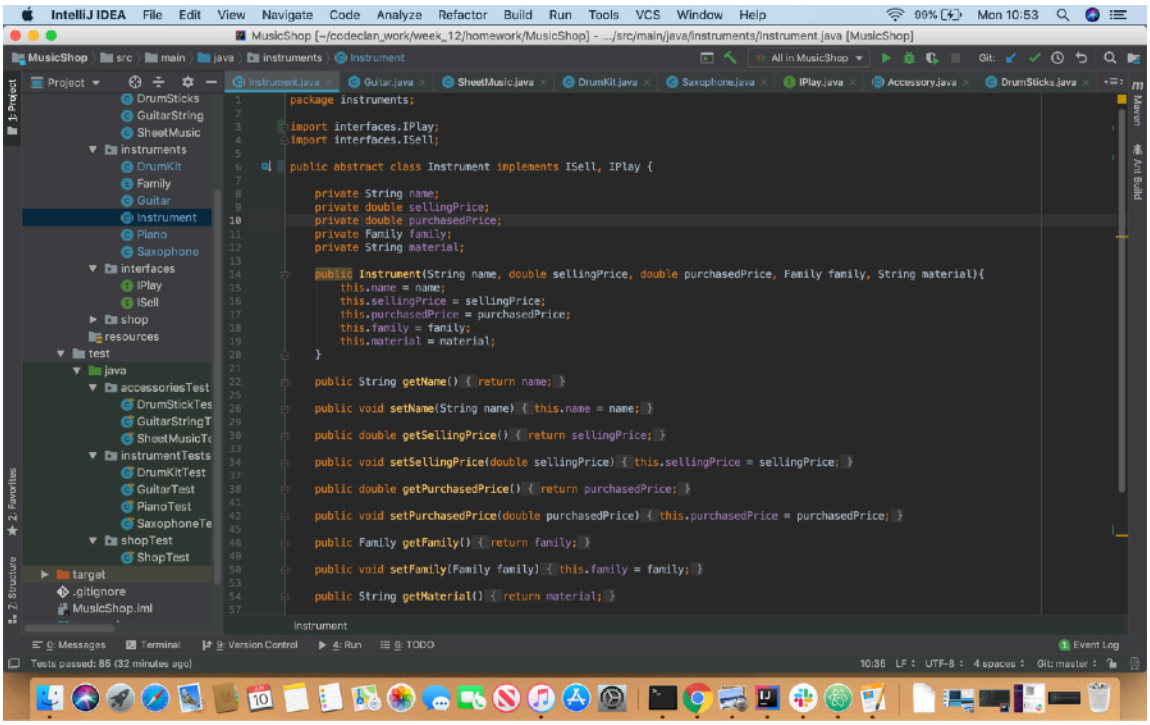
Unit	Ref	Evidence
P	P.17	Produce a bug tracking report
		Description: Here is sample of the bugs we encountered during the last group project. The vast majority of errors were syntax and spelling mistakes. The selection below shows a few of the more challenging issues.

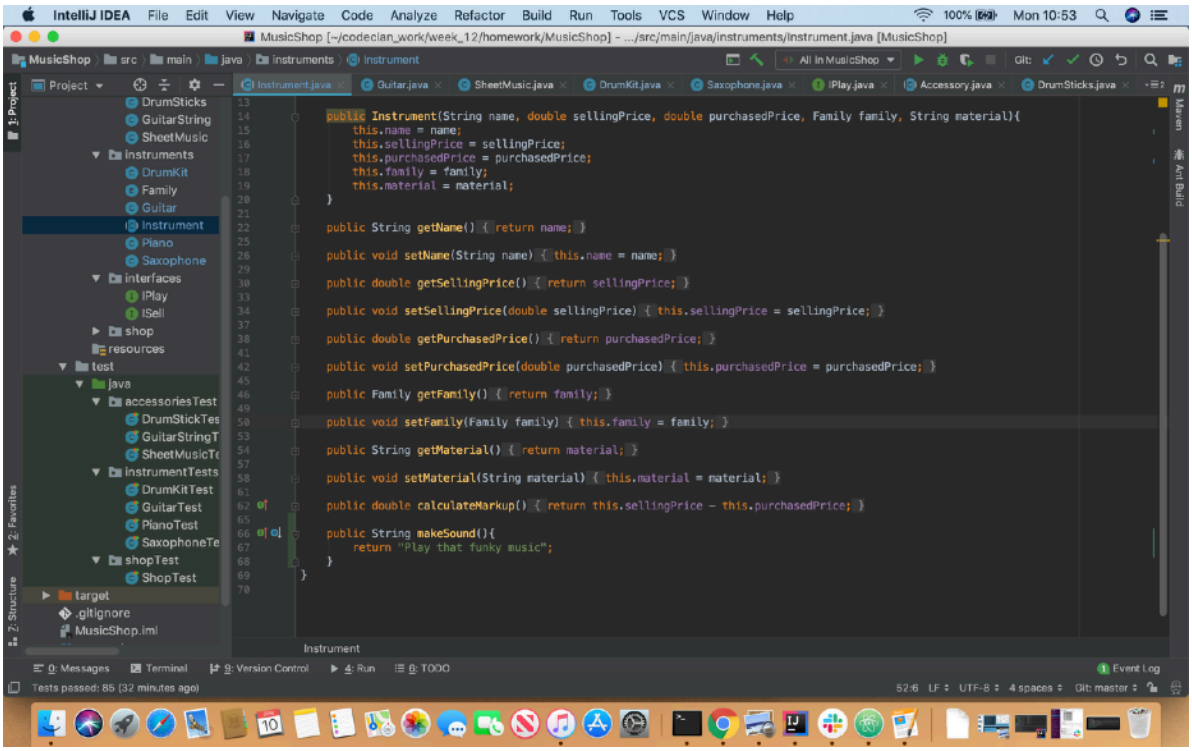
Bug/Error	Solution	Date
java.time.format.DateTimeParseException: Text could not be parsed at index 3	Changed the date format to match the input style used in db.	26-06-2019
Parsing error: Identifier 'event' has already been declared	We had declared ‘event’ twice. One was removed	25-06-2019
Uncaught TypeError: Cannot read property 'persons' of undefined	Had to use projections to assign an id to a person in the backend when you were looking at the “/events/1/persons”	26-06-2019
Star item function not working when new item is added	Changed event.target.value to event.target.checked as it has a checked function	26-06-2019

Week 12

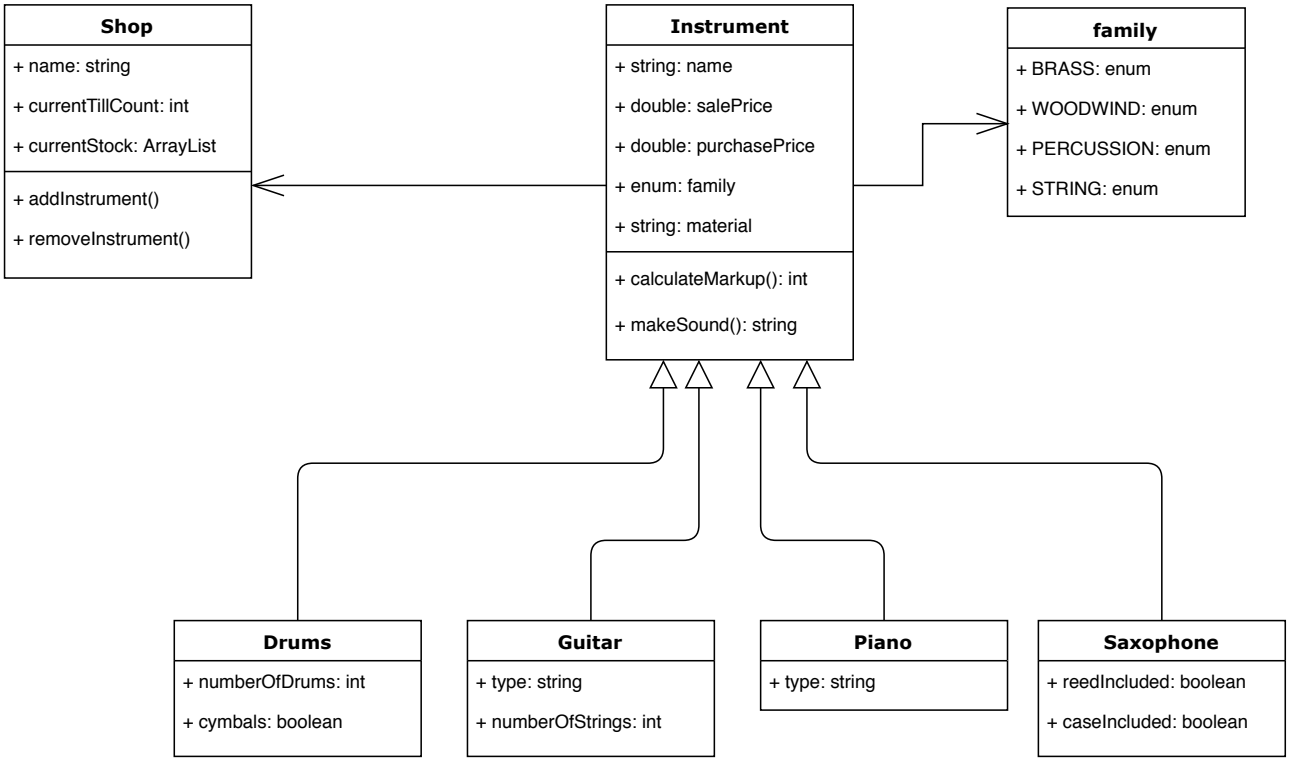
Unit	Ref	Evidence
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.
		Description: Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. The superclass Instrument here has a makeSound() method. The subclasses of Instrument (guitar, piano etc) all have their own implementation of makeSound() when it is called.



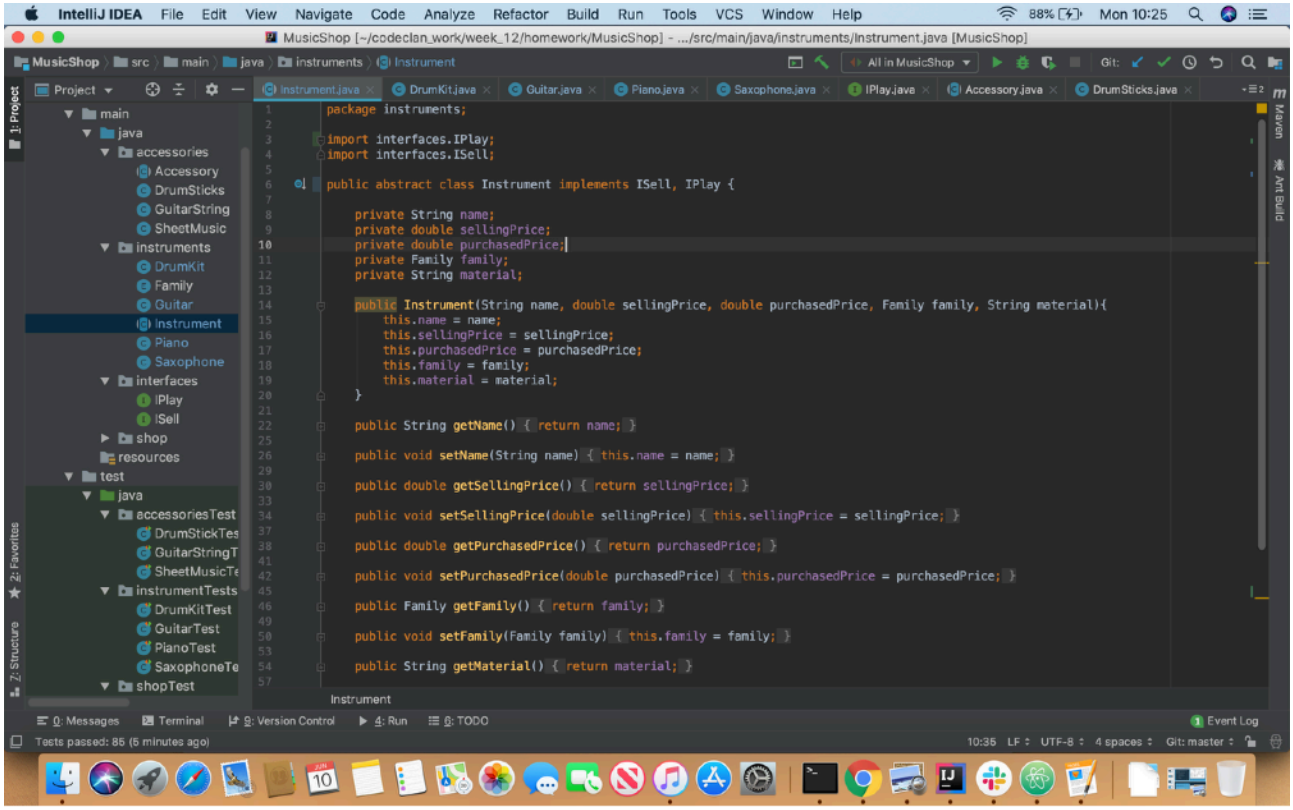



IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
MusicShop [~/codeclan_work/week_12/homework/MusicShop] - .../src/main/java/instruments/Instrument.java [MusicShop]
Project
DrumSticks
GuitarString
SheetMusic
instruments
DrumKit
Family
Guitar
Instrument
Piano
Saxophone
interfaces
IPlay
ISell
shop
resources
test
java
accessoriesTest
instrumentTests
DrumKitTest
GuitarTest
PianoTest
SaxophoneTe
shopTest
ShopTest
target
.gitignore
MusicShop.iml
Instrument.java
Guitar.java
SheetMusic.java
DrumKit.java
Saxophone.java
IPlay.java
Accessory.java
DrumSticks.java
public Instrument(String name, double sellingPrice, double purchasedPrice, Family family, String material){
this.name = name;
this.sellingPrice = sellingPrice;
this.purchasedPrice = purchasedPrice;
this.family = family;
this.material = material;
}
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public double getSellingPrice() { return sellingPrice; }
public void setSellingPrice(double sellingPrice) { this.sellingPrice = sellingPrice; }
public double getPurchasedPrice() { return purchasedPrice; }
public void setPurchasedPrice(double purchasedPrice) { this.purchasedPrice = purchasedPrice; }
public Family getFamily() { return family; }
public void setFamily(Family family) { this.family = family; }
public String getMaterial() { return material; }
public void setMaterial(String material) { this.material = material; }
public double calculateMarkup() { return this.sellingPrice - this.purchasedPrice; }
public String makeSound(){
return "Play that funky music";
}
}
Instrument
0: Messages
Terminal
9: Version Control
4: Run
6: TODO
Event Log
Tests passed: 85 (32 minutes ago)
52:6 LF UTF-8 4 spaces Git: master

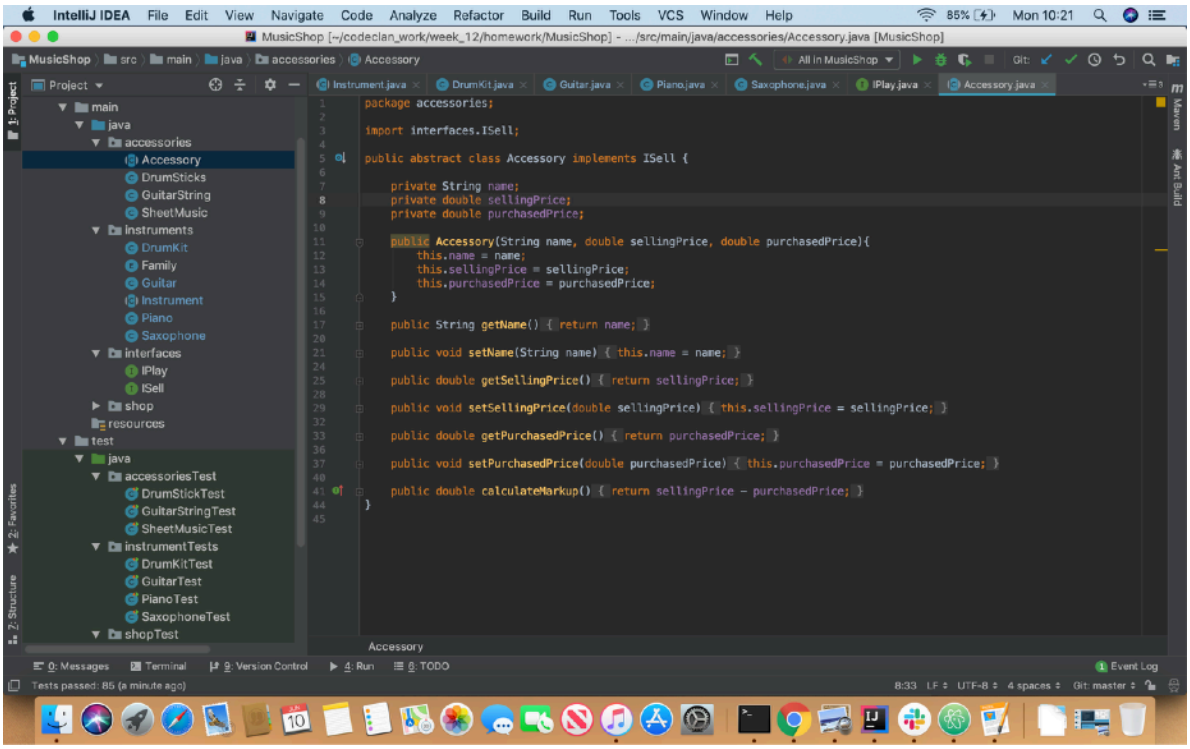
Unit	Ref	Evidence
A&D	A.D.5	An Inheritance Diagram
		Description: This is an inheritance diagram for a music shop. The 4 different instruments (Drums, guitar, piano and saxophone) all inherit properties and methods from the instrument class which is abstract. The Shop class has the functions auto add and remove instruments in an ArrayList for the current stock.

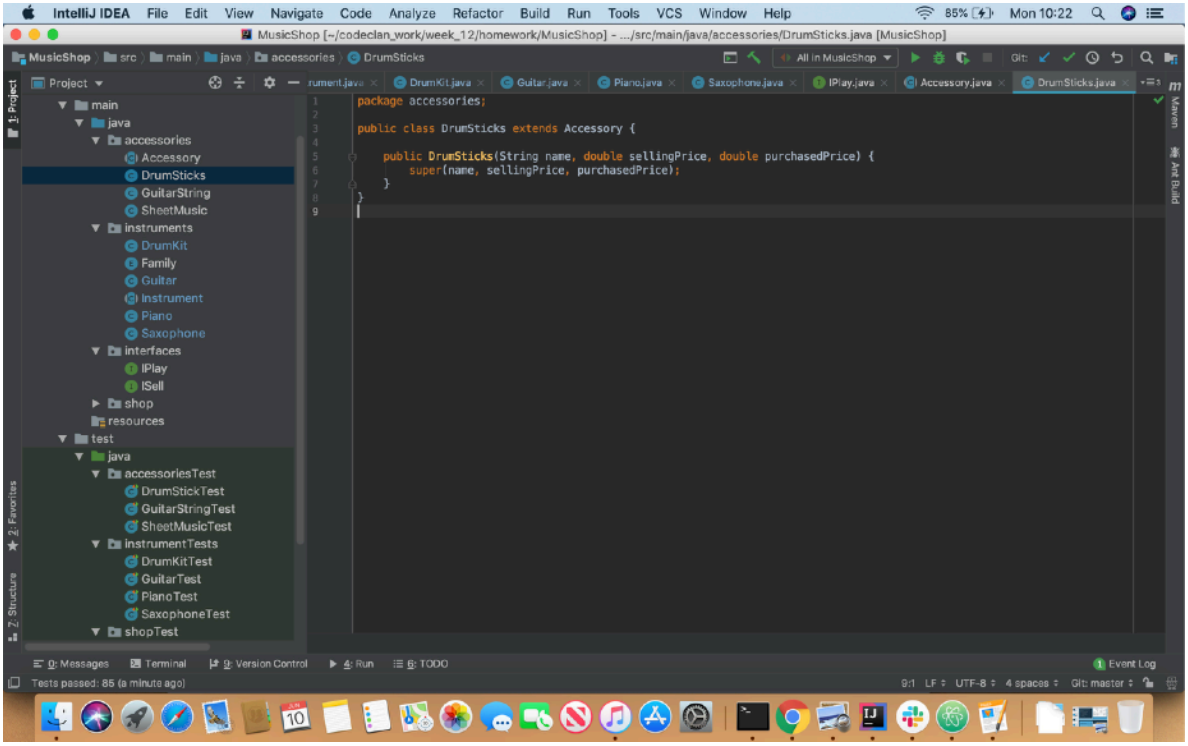


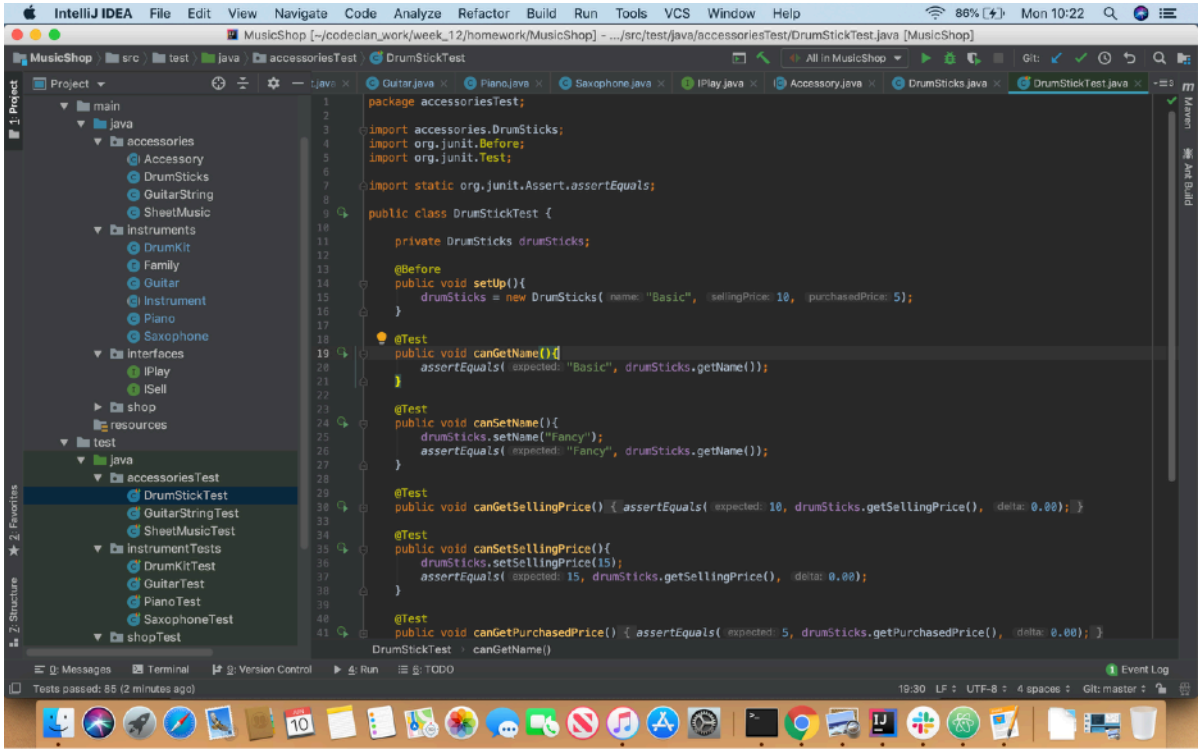
Unit	Ref	Evidence
I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description: The code below is for an instrument abstract class. The instance variables are private so that they cannot be accessed directly from outside the class. You can only set and get values of these variables through the methods of the class using getter and setters.

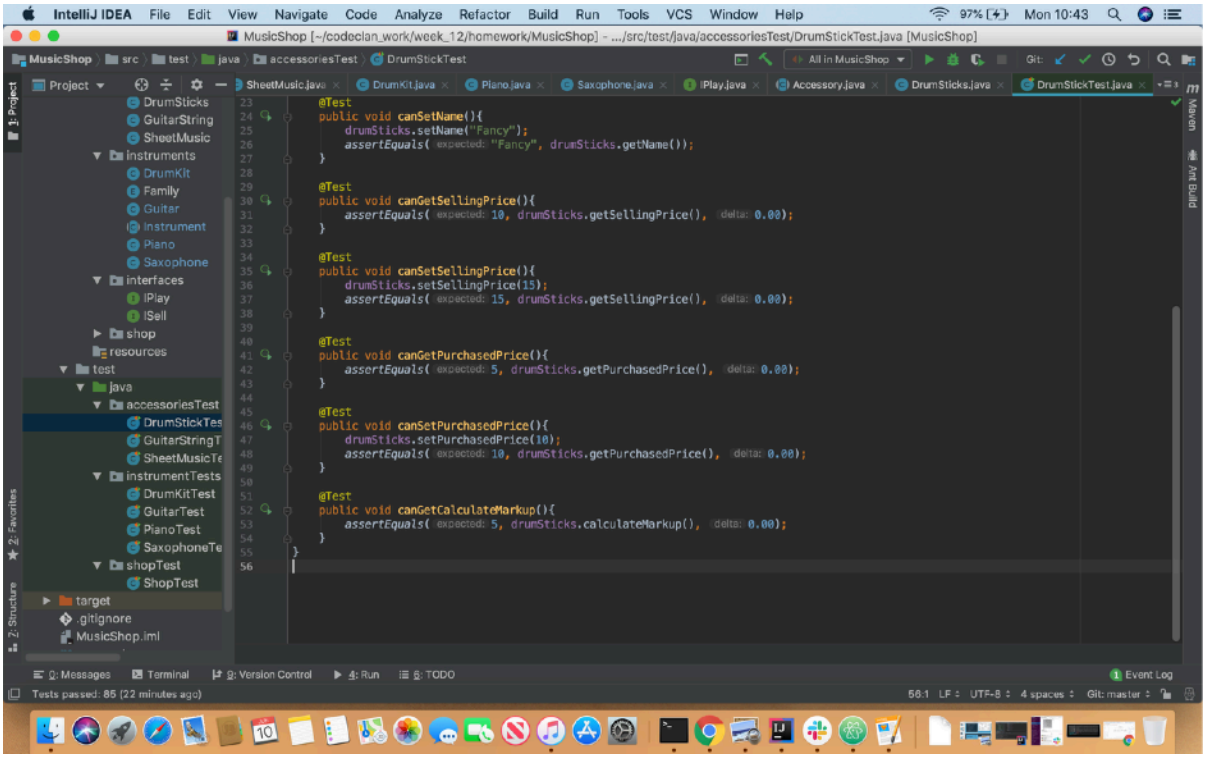


Unit	Ref	Evidence
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.
		Description: This example uses the music shop. The DrumSticks class inherits from the Accessory class. The calculateMarkup() method is shown calculating the markup of the DrumSticks with the result shown for the drum sticks in the last image.






IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Mon 10:22
MusicShop [~/codeclan_work/week_12/homework/MusicShop] - .../src/test/java/accessoriesTest/DrumStickTest.java [MusicShop]
Project
main
java
accessories
Accessory
DrumSticks
GuitarString
SheetMusic
instruments
DrumKit
Family
Guitar
Instrument
Piano
Saxophone
interfaces
IPlay
ISell
shop
resources
test
java
accessoriesTest
DrumStickTest
GuitarStringTest
SheetMusicTest
instrumentTests
DrumKitTest
GuitarTest
PianoTest
SaxophoneTest
shopTest
package accessoriesTest;
import accessories.DrumSticks;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.assertEquals;
public class DrumStickTest {
private DrumSticks drumSticks;
@Before
public void setUp(){
drumSticks = new DrumSticks(name: "Basic", sellingPrice: 10, purchasedPrice: 5);
}
@Test
public void canGetName(){
assertEquals(expected: "Basic", drumSticks.getName());
}
@Test
public void canSetName(){
drumSticks.setName("Fancy");
assertEquals(expected: "Fancy", drumSticks.getName());
}
@Test
public void canGetSellingPrice() { assertEquals(expected: 10, drumSticks.getSellingPrice(), delta: 0.00); }
@Test
public void canSetSellingPrice(){
drumSticks.setSellingPrice(15);
assertEquals(expected: 15, drumSticks.getSellingPrice(), delta: 0.00);
}
@Test
public void canGetPurchasedPrice() { assertEquals(expected: 5, drumSticks.getPurchasedPrice(), delta: 0.00); }
DrumStickTest › canGetName()
0: Messages
Terminal
9: Version Control
4: Run
6: TODO
Event Log
Tests passed: 85 (2 minutes ago)
19:30 LF UTF-8 4 spaces Git: master


IntelliJ IDEA File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Mon 10:43
MusicShop [~/codeclan_work/week_12/homework/MusicShop] - .../src/test/java/accessoriesTest/DrumStickTest.java [MusicShop]
DrumSticks
GuitarString
SheetMusic
instruments
DrumKit
Family
Guitar
Instrument
Piano
Saxophone
interfaces
IPlay
ISell
shop
resources
test
java
accessoriesTest
DrumStickTes
GuitarStringT
SheetMusicTe
instrumentTests
DrumKitTest
GuitarTest
PianoTest
SaxophoneTe
shopTest
ShopTest
target
.gitignore
MusicShop.iml
@Test
public void canSetName(){
drumSticks.setName("Fancy");
assertEquals(expected: "Fancy", drumSticks.getName());
}
@Test
public void canGetSellingPrice(){
assertEquals(expected: 10, drumSticks.getSellingPrice(), delta: 0.00);
}
@Test
public void canSetSellingPrice(){
drumSticks.setSellingPrice(15);
assertEquals(expected: 15, drumSticks.getSellingPrice(), delta: 0.00);
}
@Test
public void canGetPurchasedPrice(){
assertEquals(expected: 5, drumSticks.getPurchasedPrice(), delta: 0.00);
}
@Test
public void canSetPurchasedPrice(){
drumSticks.setPurchasedPrice(10);
assertEquals(expected: 10, drumSticks.getPurchasedPrice(), delta: 0.00);
}
@Test
public void canGetCalculateMarkup(){
assertEquals(expected: 5, drumSticks.calculateMarkup(), delta: 0.00);
}
}
0: Messages
Terminal
9: Version Control
4: Run
6: TODO
Event Log
Tests passed: 85 (22 minutes ago)
56:1 LF UTF-8 4 spaces Git: master

Unit	Ref	Evidence
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.
		Description: The first image shows a change calculator for a vending machine. The function takes in a number input (change) and works out the best way to issue that amount of change using the least amount of coins. **The second image is of a bubble sort used to order the passengers by the seat number they were randomly assigned when a ticket was purchased. Both of these algorithms, although complex to look at, provide a very clear process of working out the correct result.**

```
// this method is used to calculate what coins would be returned
// for a given amount of change

public void workOutChange(int change){
     ArrayList<Coin> returnedCoins = new ArrayList<Coin>();
    if(change >= 200){ // if change is more than 200 a two pound coin can be issued
        this.returnedCoins.add(twoPound);
        change -= 200;
    } if(change >= 100){ // if change is more than 100 a pound coin can be issued
        this.returnedCoins.add(pound);
        change -= 100;
    } if (change >= 50){ // if change is more than 50 a 50p coin can be issued
        this.returnedCoins.add(fifty);
        change -= 50;
    } if (change >= 20){// if change is more than 20 a 20p coin can be issued
        this.returnedCoins.add(twenty);
        change -= 20;
        if (change >= 20){ // if change is still more than 20 a 20p coin can be issued ie 40p change
            this.returnedCoins.add(twenty);
            change -= 20;
        }
    } if (change >= 10){ // if change is more than 10 a 10p coin can be issued
        this.returnedCoins.add(ten);
        change -= 10;
    } if (change >= 5){ //// if change is more than 5 a 5p coin can be issued
        this.returnedCoins.add(five);
        change -= 5;
    }
}
```

```
public ArrayList<Passenger> sortPassengersBySeatNumber(Flight flight) {
    ArrayList<Passenger> sortedPassengers = flight.getPassengers();
    Passenger temp;
    for (int i = sortedPassengers.size() -1; i > 0; i--){

        boolean isSorted = true;

        for (int j = 0; j < i; j++){
            if (sortedPassengers.get(j + 1).getSeatNumber() < sortedPassengers.get(j).getSeatNumber()){
                isSorted = false;
                temp = sortedPassengers.get(j + 1);
                sortedPassengers.set(j + 1, sortedPassengers.get(j));
                sortedPassengers.set(j, temp);
            }
        }
        if (isSorted){
            break;
        }
    }
    return sortedPassengers;
}
```