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.

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

Each point requires details that cover each element of the Assessment Criteria, along with a brief description of the kind of things you should be showing.

Week 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:	

Paste Screenshot here

```ruby
def initialize(name, type)
  @name = name
  @type = type
  @fish_stored = []
  @stomach = []
end

def remove_fish_from_river(river)
  return "No fish to remove!" if river.contents.empty? == true
  @fish_stored << river.contents.last
  river.remove_fish
end
```

```ruby
def test_bear_take_fish_from_river
  @bear.remove_fish_from_river(@river)
  assert_equal([@fish_2], @bear.fish_stored)
  assert_equal([@fish_1], @river.contents)
end

def test_bear_take_fish_from_river__no_fish
  river_2 = River.new("Amazon")
  assert_equal("No fish to remove!",
  @bear.remove_fish_from_river(river_2))
end
```

```
➜ bear_river_fish git:(master) ruby specs/bear_spec.rb
Run options: --seed 35033

# Running:

.........

Finished in 0.001531s, 5878.5108 runs/s, 6531.6787 assertions/s.
```

Description here

For this program there is a Bear class which has an attribute named fish_stored which is an array. The method remove _fish_from_river manipulates this array. The method takes a parameter 'river'. In the example above an instance of a River class is passed into the method. The method checks if the river attribute 'contents', which is also an array, is empty and if so returns a string. If it is not empty then the last element in rivers 'contents' is pushed/shovelled into the fish_stored array. The method then asks the river to call upon its own method remove_fish to remove the element from it's own 'contents' array. The tests above check the outcome of passing a river with elements in the 'contents' array and the outcome of an empty 'contents' array. Both tests are passed.

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:	

Paste Screenshot here

```ruby
class GenreTest < MiniTest::Test

  def setup
    @genre = Genre.new({'id' => '1', 'genre_name' => 'Test_Genre'})
  end

  def test_genre_exists
    assert_equal(Genre, @genre.class)
  end

  def test_genre_can_have_id
    assert_equal(1, @genre.id)
  end

  def test_genre_has_name
    assert_equal('Test_Genre', @genre.genre_name)
  end

end
```

```ruby
def initialize(options)
  @id = options['id'].to_i if options['id']
  @genre_name = options['genre_name']
end
```

```
➜  record_shop_inventory git:(master) ruby specs/genre_spec.rb
Run options: --seed 20123

# Running:

...

Finished in 0.000864s, 3472.2222 runs/s, 3472.2222 assertions/s.

3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

Description here

The above example of a hash is used to pass information into the new method for a class. The hash contains two keys (id and genre_name). The initializer for the Genre class takes in an 'options' parameter. The hash is passed in and the values of the keys are used to set the attributes of the Genre instance. The tests check that these attributes are set successfully and all the tests pass.

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:	

Paste Screenshot here

```
Park.prototype.findByMostVisitors = function () {
  const mostVisitors = [];
  for (let dinosaur of this.dinosaurs) {
    if (mostVisitors.length === 0) {
      mostVisitors.push(dinosaur);
    }
    else if (dinosaur.guestsAttractedPerDay > mostVisitors[0].guestsAttractedPerDay) {
      mostVisitors.splice(0,1);
      mostVisitors.push(dinosaur);
    };
  };
  return mostVisitors;
};
```

```
it('should be able to find the dinosaur that attracts the most visitors', function () {
  park.addDinosaur(dinosaur1);
  park.addDinosaur(dinosaur2);
  assert.deepStrictEqual([dinosaur2], park.findByMostVisitors());
});
```

```
Park
  ✓ should have a name
  ✓ should have a ticket price
  ✓ should have a collection of dinosaurs
  ✓ should be able to add a dinosaur to its collection
  ✓ should be able to remove a dinosaur from its collection
  ✓ should be able to find the dinosaur that attracts the most visitors
  ✓ should be able to find all dinosaurs of a particular species
  ✓ should be able to remove all dinosaurs of a particular species
  ✓ should be able to calculate the total number of visitors per day
  ✓ should be able to calculate the total number of visitors per year
  ✓ should be able to calculate the total revenue from ticket sales for one year


14 passing (12ms)
```

Description here

The above code is a method belonging to a class named Park. An instance of a park object contains an array of instances of Dinosaur objects. Dinosaur objects contain an attribute named guestsAttractedPerDay. The method searches for and returns the Dinosaur object with the highest value of this attribute. The screenshots show the expected outcome and passing of the test.

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:	

Paste Screenshot here

```ruby
numbers = [3,7,3,82,4,2,8,9,55,4,3,5,7,8,4,39,20]

def sort_data(data)
  return data.sort()
end

puts sort_data(numbers)
```

```
[➜  PDA Code ruby sort.rb
2
3
3
3
4
4
4
5
7
7
8
8
9
20
39
55
82
```

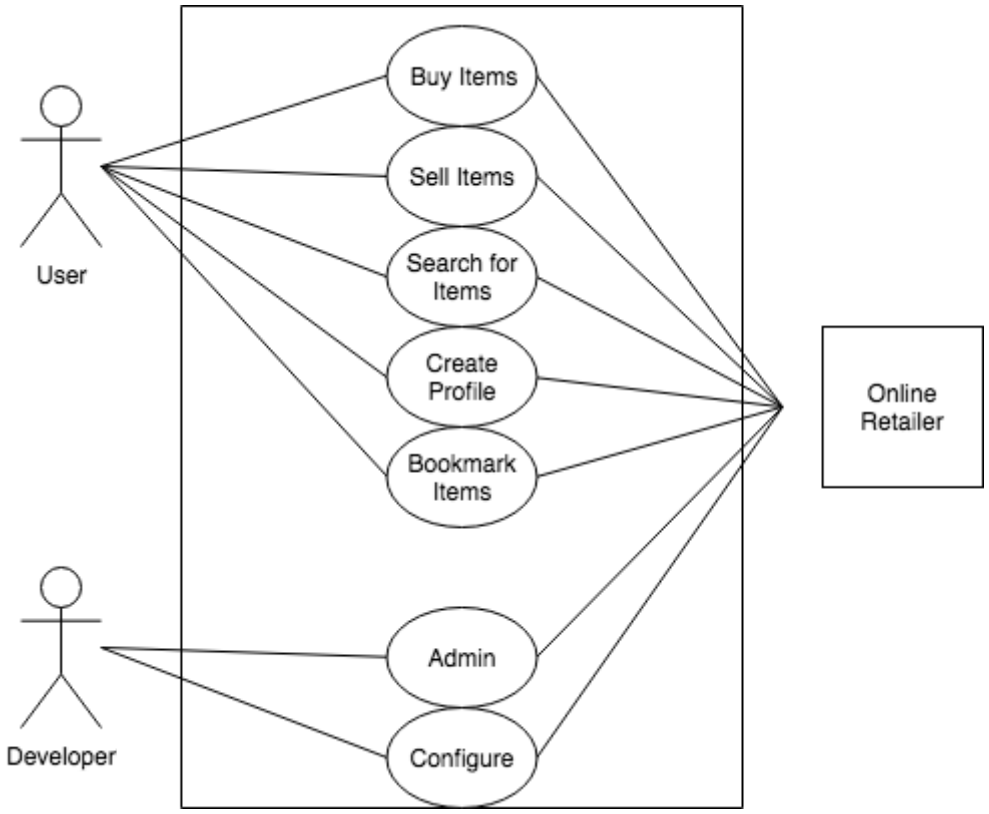
Description here

The sort_data function in the first screenshot above takes in an array of objects as an argument and returns a sorted array. The second screenshot shows the result of the method and the sorted value.

Week 5 and 6

Unit	Ref	Evidence	
A&D	A.D.1	A Use Case Diagram	
		Description:	

Paste Screenshot here

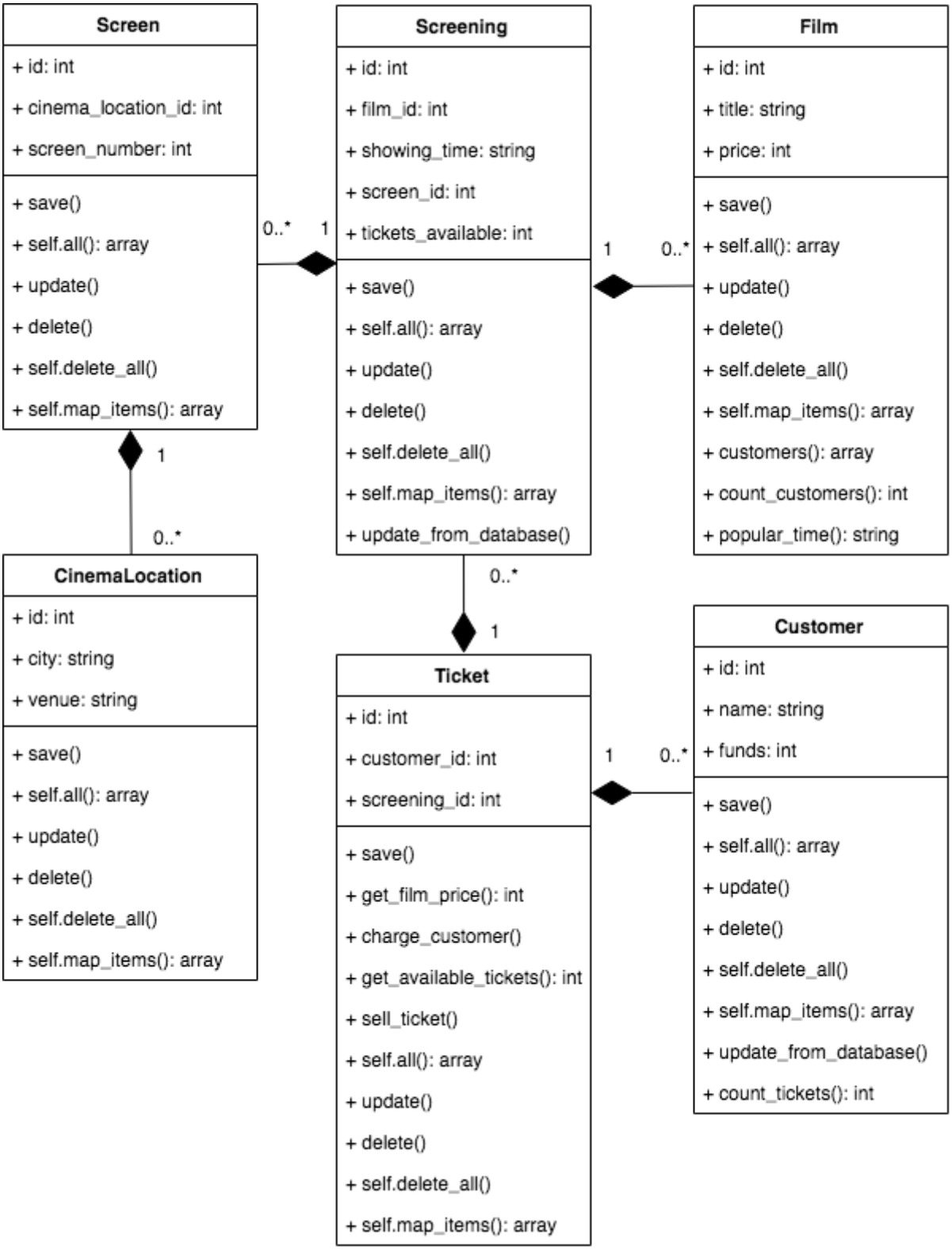


Description here

The above diagram displays the actions that both a user and a developer would need to perform using a system designed for an online retailer.

Unit	Ref	Evidence	
A&D	A.D.2	A Class Diagram	
		Description:	

Paste Screenshot here

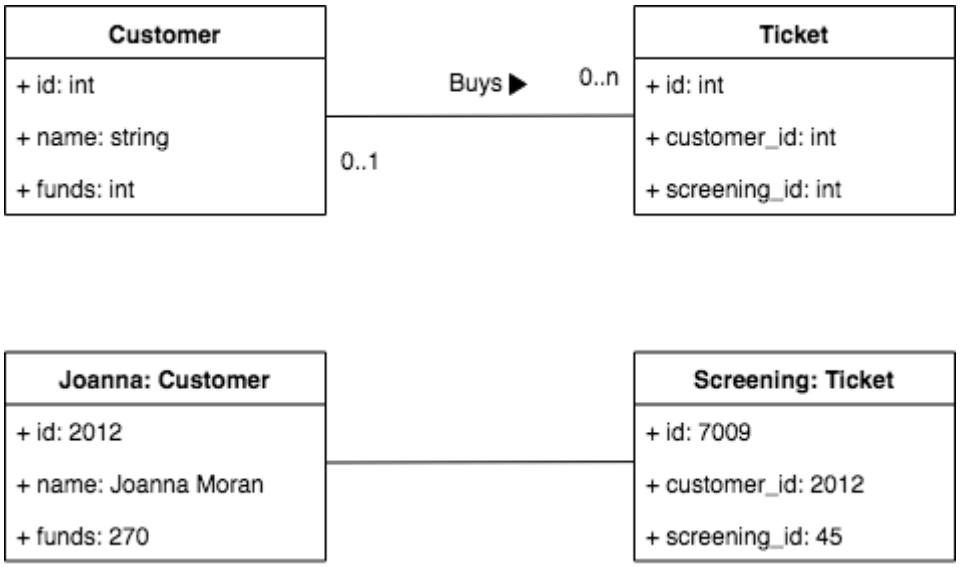


Description here

The above class diagram displays the classes for a cinema ticket ordering system including the methods and attributes of each class.

Unit	Ref	Evidence	
A&D	A.D.3	An Object Diagram	
		Description:	

Paste Screenshot here

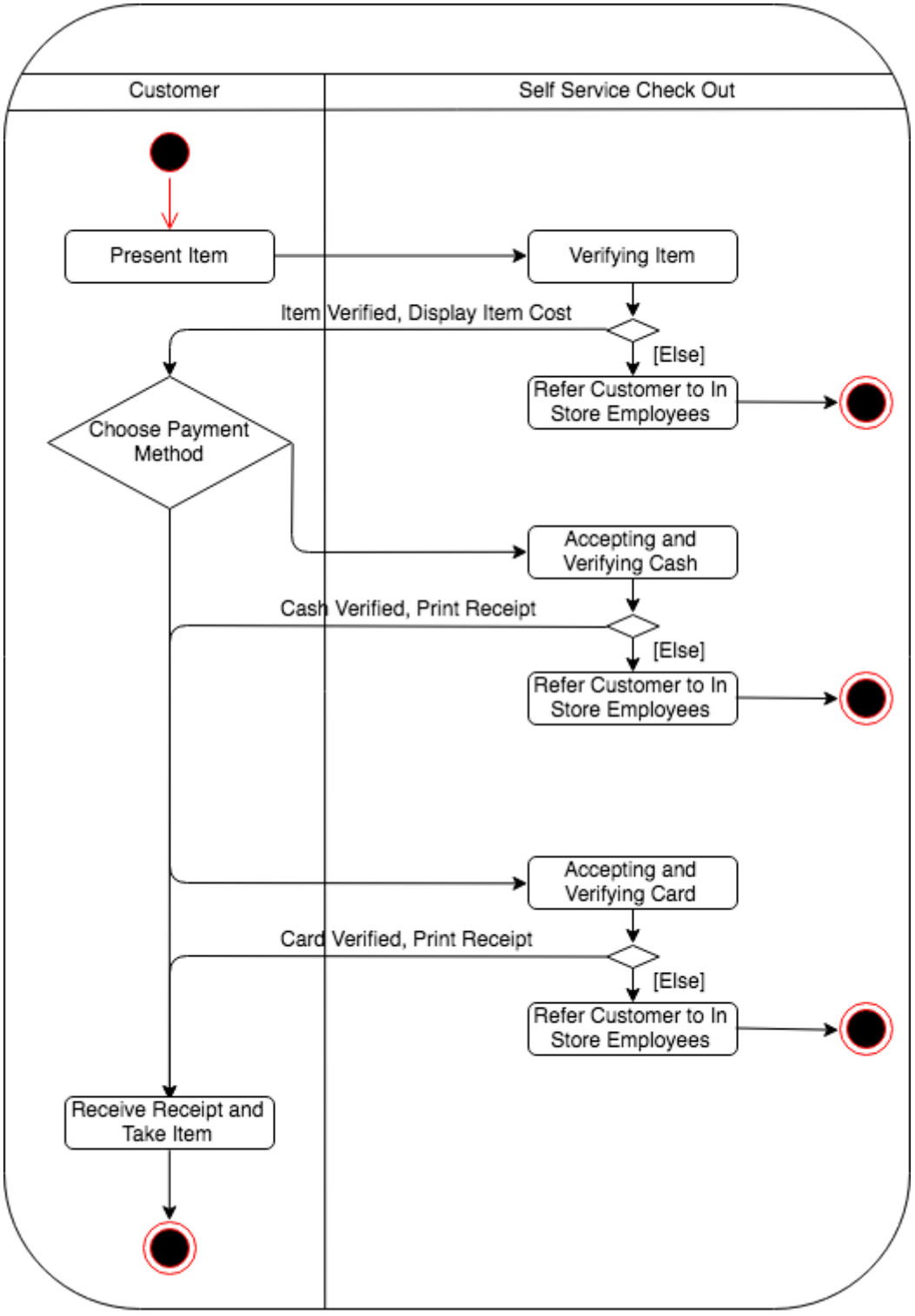


Description here

The above diagram displays object examples of a customer and a ticket for a cinema film ordering system.

Unit	Ref	Evidence	
A&D	A.D.4	An Activity Diagram	
		Description:	

Paste Screenshot here



Description here

The above diagram displays the actions available for a self service checkout in a supermarket.

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:	

Paste Screenshot here

Constraint Category	**Implementation Constraint**	**Solution**
Hardware and software platforms	Since the app is being implemented on a Mac then users of other systems may find it difficult to down and run the code.	Implement a login system and host the application on Heroku to allow for any user with browser access to use the app.
Performance requirements	The amount of data requested from the API may impact memory and slow down performance of the App.	Only request the required data for the specific route that the user is accessing, e.g. a single article for a specific articles page as opposed to all the articles and filtering the results.
Persistent storage and transactions	The app will need to store data and handle transactions.	The main dataset of the app will be sourced from an API, reducing the data which needs to be stored on the server. The login system will require storage of user details into a secure database. Forms will need to be implemented correctly to make sure that the user cannot enter incorrectly formatted data and prevent bad transactions.
Usability	The user will need access many different routes on demand using the app.	Implement a navigation bar that allows the user to travel quickly and easily between

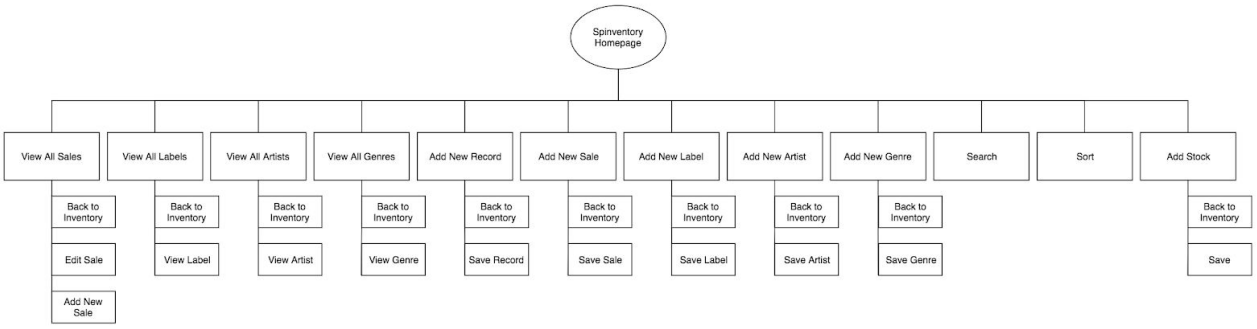
		routes on any given page.
Budgets	The budget has to be kept to a minimum in.	Research and agree upon only the most necessary technologies needed to allow the app to be developed successfully.
Time Limitations	The app must reach MVP within a week of commencing the project.	Using agile methodology implement short sprites that will allow for group retrospectives and important concise goals.

Description here

The above Implementation Constraints plan outlines the constraints of a project and the method/solution for overcoming those constraints.

Unit	**Ref**	**Evidence**	
P	P.5	User Site Map	
		Description:	

Paste Screenshot here

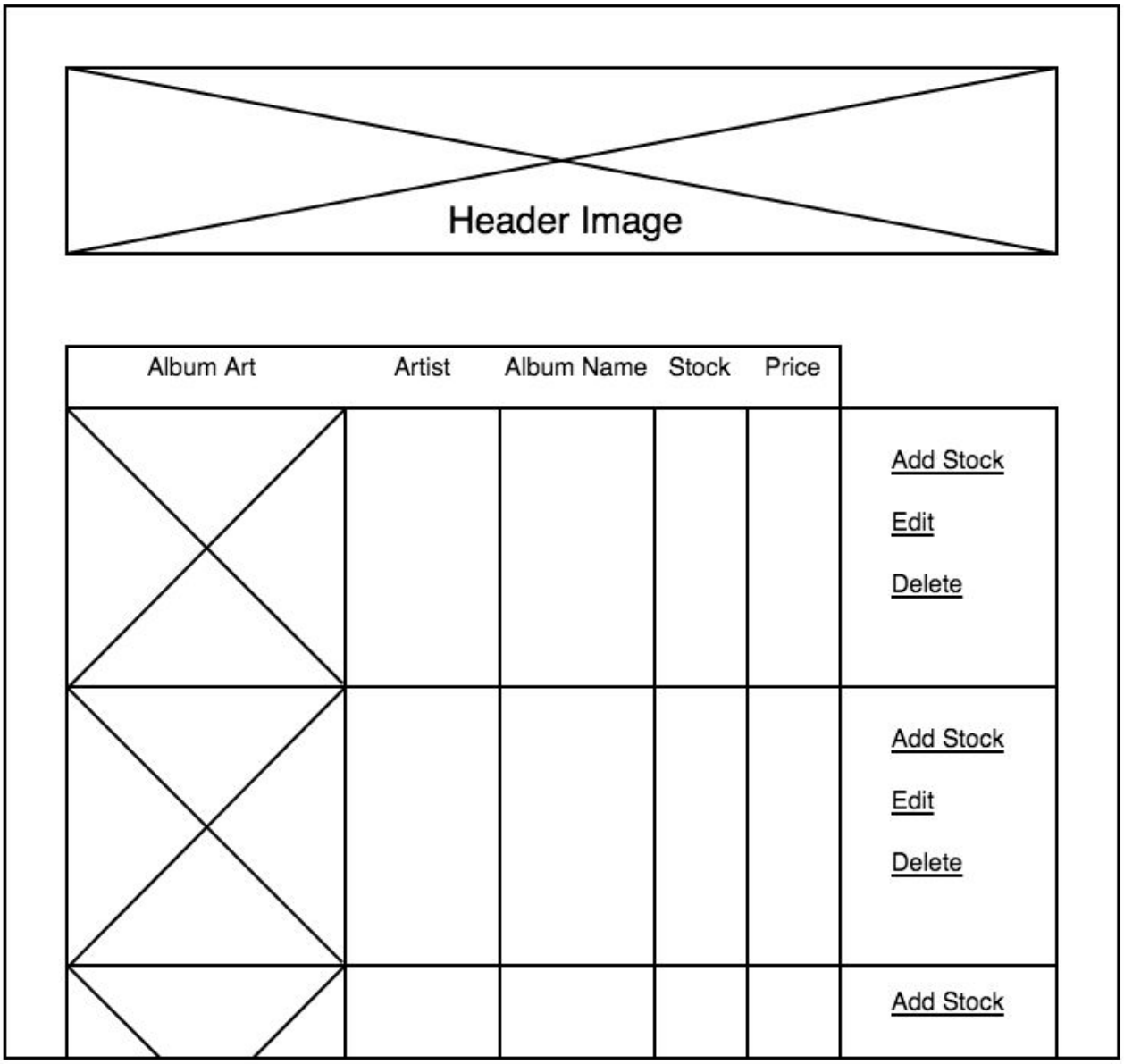


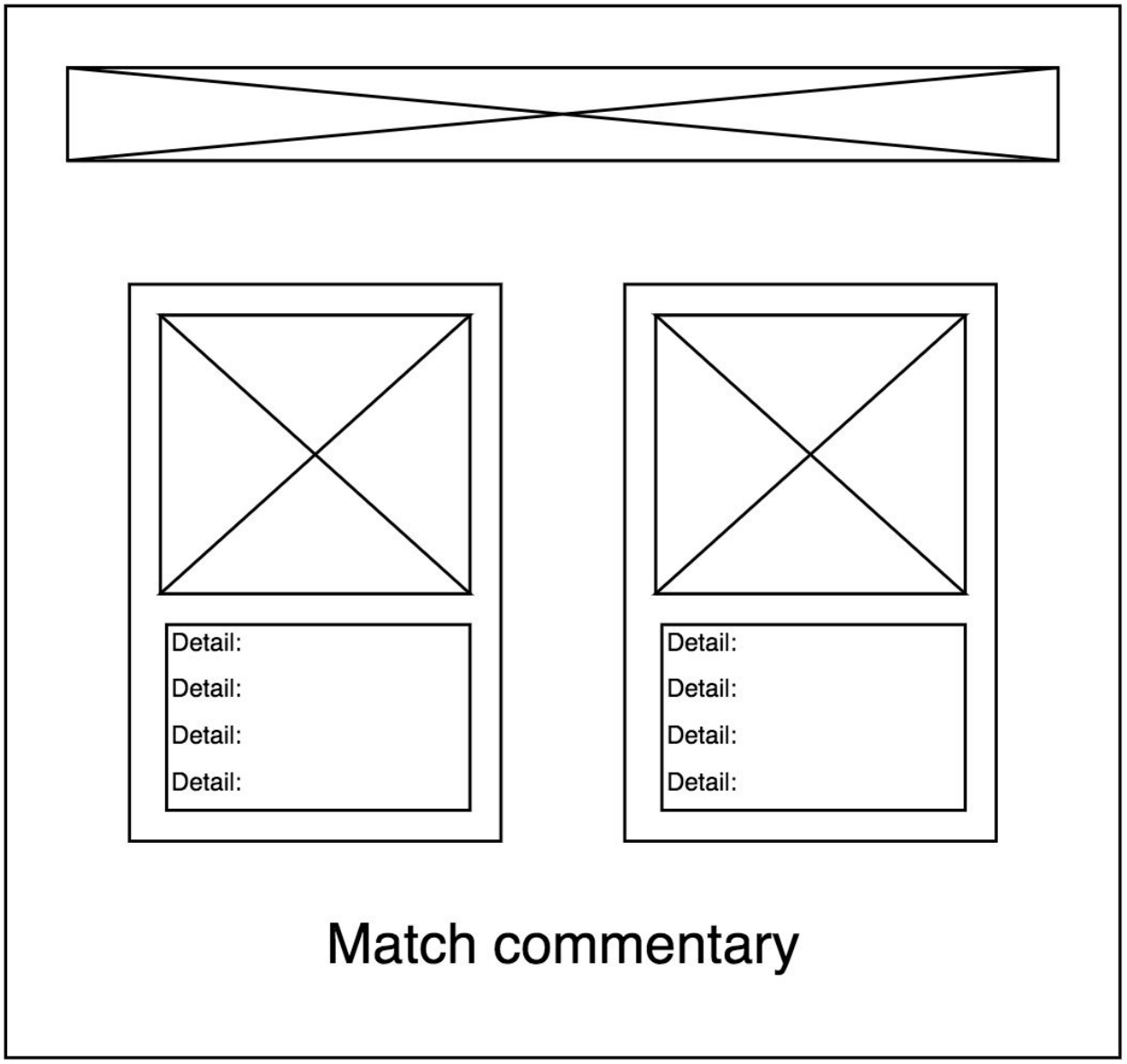
Description here

The above diagram shows a map of options available to a user of a record inventory browser app. It shows the hierarchy of the site in accordance with RESTful routes and the routes back to the front page of the app.

Unit	Ref	Evidence	
P	P.6	2 Wireframe Diagrams	
		Description:	

Paste Screenshot here





Description here

The first wireframe diagram above shows the front page layout of a record inventory app. It shows the layout of images, sections and options on the page.

The second wireframe diagram shows the layout of a top trumps style game. It shows the position of the header image as well as the cards and card details.

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

Paste Screenshot here

```
# Pseudocode

# Take in an array of numbers as an argument and a number to
remove from the array
# For every number in the numbers array
# Check if the number item is equal to the number to remove
# If true them remove the current item from the array
# Return the altered array
```

Description here

The above pseudocode describes a method which will remove a specified number from an array of numbers and the logic it will use to achieve this.

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:	

Paste Screenshot here

CodeClan_

Bridging the digital skills gap.

New Record

Label: Warp

Label Name: Only for new label

Label Location: Only for new label

Title: Geogaddi

Artist: Boards of Canada

Artist Name: Only for new artist

Genre(s):

- [x] Electronica
- [ ] Downtempo
- [ ] Trip Hop
- [ ] Electronic
- [ ] Plunderphonics
- [ ] Vaportrap
- [ ] Industrial Hip Hop
- [ ] Indie Rock
- [ ] Ambient Techno
- [ ] Shoegaze
- [ ] Psychedelic Pop

Genre Name: IDM

Release Date: 13/02/2002

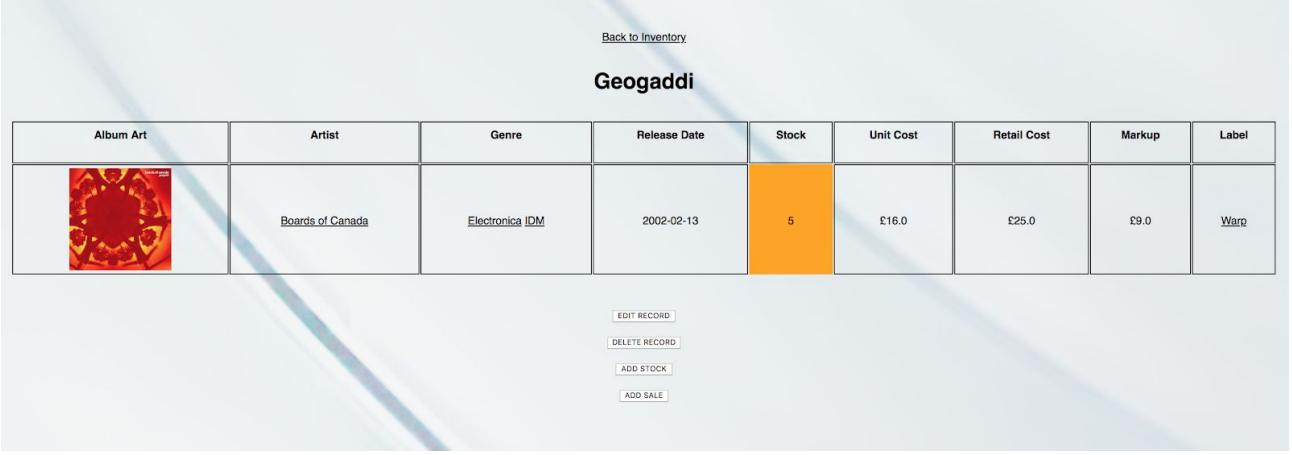
Stock Quantity: 5

Unit Price: 16

Retail Price: 25

Upload Album Artwork: Choose file boc-g.jpg

Save Record



Description here

The first screenshot above shows the user inputting details for a record to be stored in the program. The second screenshot shows the result of the inputted data being used in the program.

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:	

Paste Screenshot here

CodeClan_

Bridging the digital skills gap.

New Record

Label: Warp

Label Name: Only for new label

Label Location: Only for new label

Title: Geogaddi

Artist: Boards of Canada

Artist Name: Only for new artist

Genre(s):

- [x] Electronica
- [ ] Downtempo
- [ ] Trip Hop
- [ ] Electronic
- [ ] Plunderphonics
- [ ] Vaportrap
- [ ] Industrial Hip Hop
- [ ] Indie Rock
- [ ] Ambient Techno
- [ ] Shoegaze
- [ ] Psychedelic Pop

Genre Name: IDM

Release Date: 13/02/2002

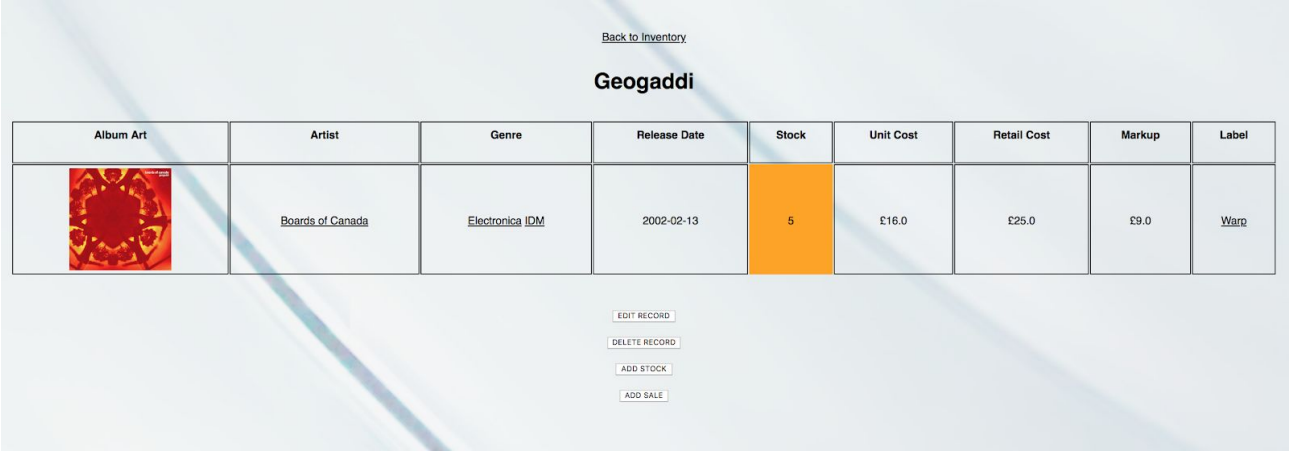
Stock Quantity: 5

Unit Price: 16

Retail Price: 25

Upload Album Artwork: Choose file boc-g.jpg

Save Record

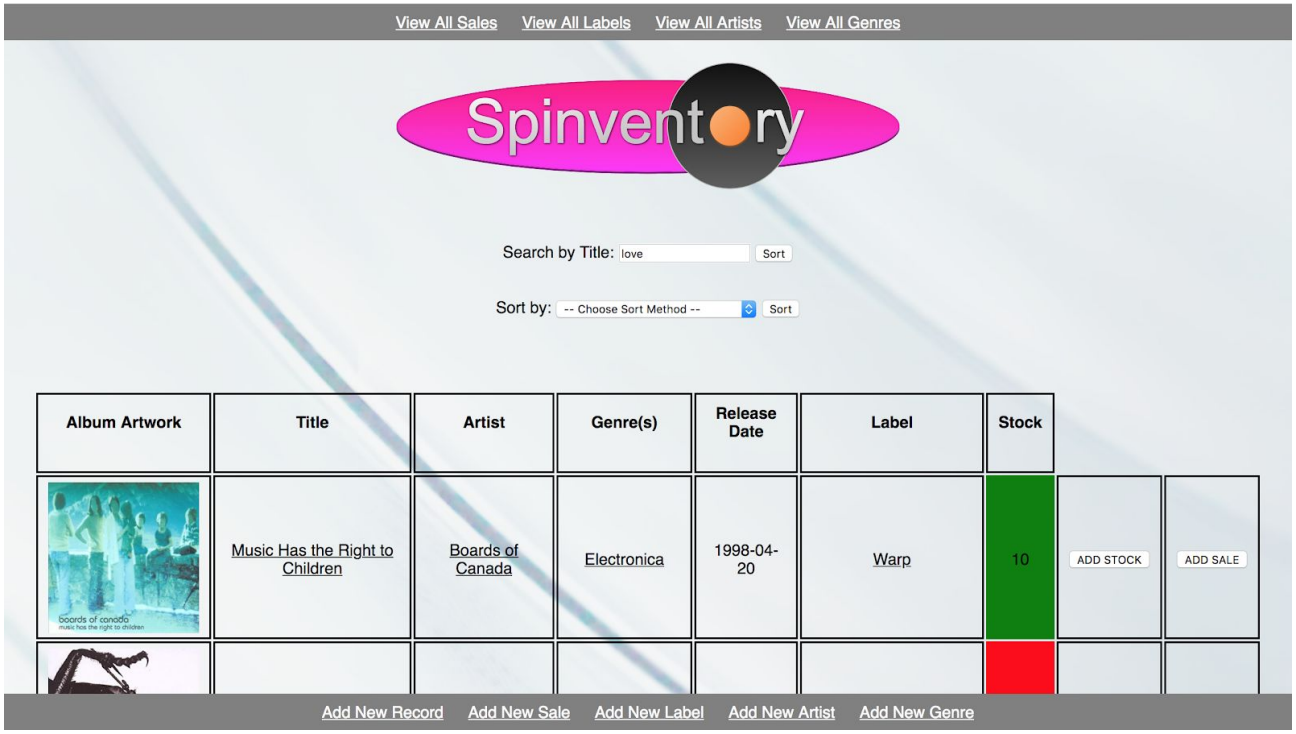


Description here

The first screenshot shows the data that has been inputted by the user. The second screenshot displays the inputted information in the correct format back to the user as confirmation that the data has been saved.

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:	

Paste Screenshot here

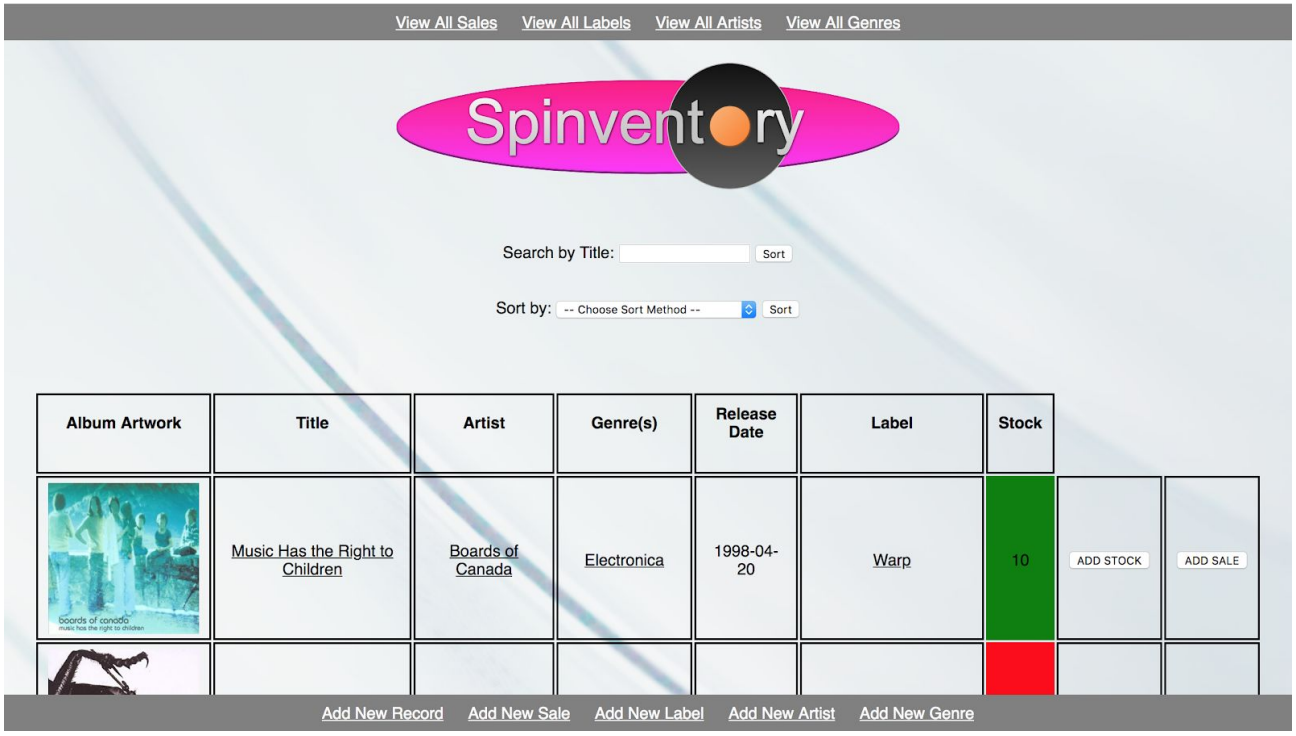


Description here

The first screenshot shows the user typing a word into a search box. The second screenshot shows the result of the search, displaying the items that include the searched word.

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:	

Paste Screenshot here



Description here

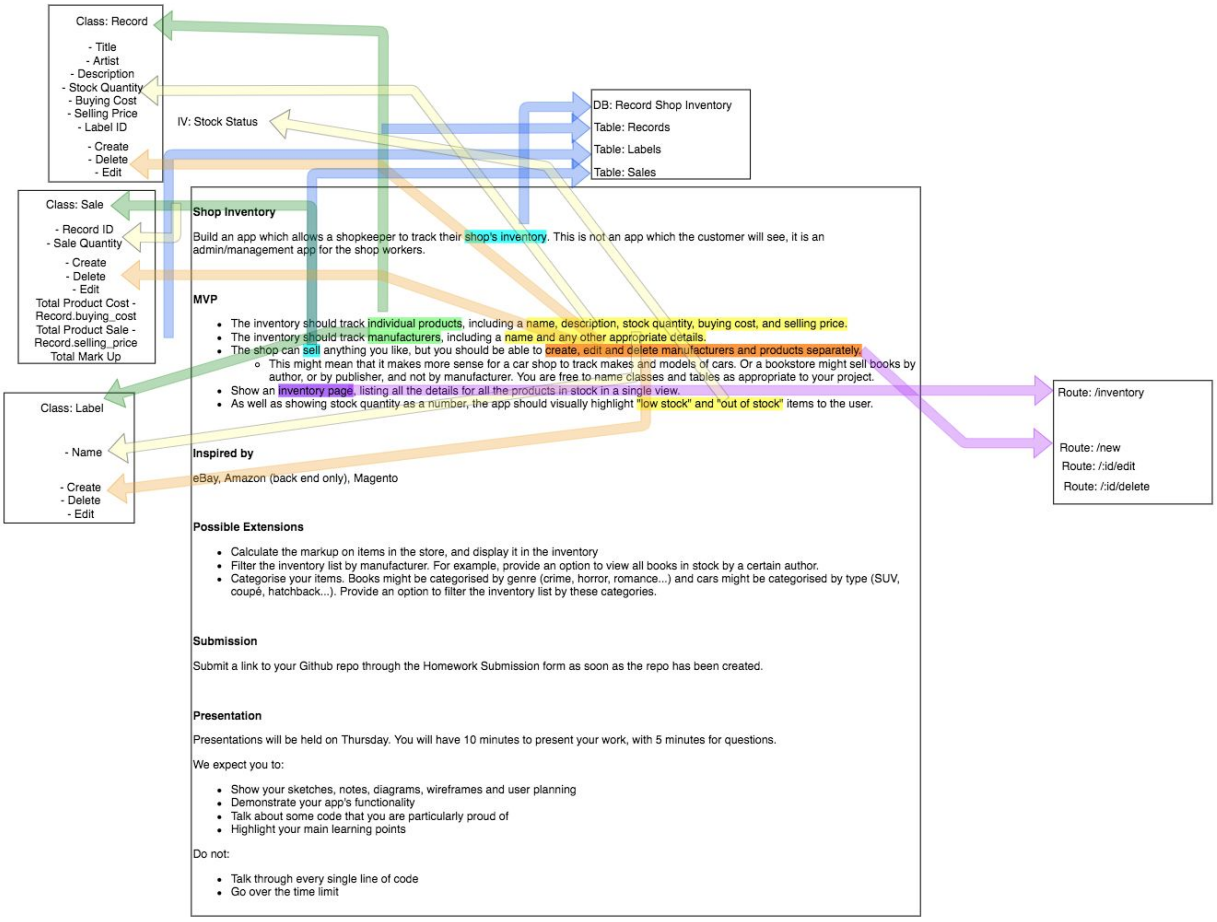
The above screenshot shows my record inventory app ‘Spinventory’. It is a Ruby, database driven app that allows the user to create, read, update and delete records, artists, genres and sales.

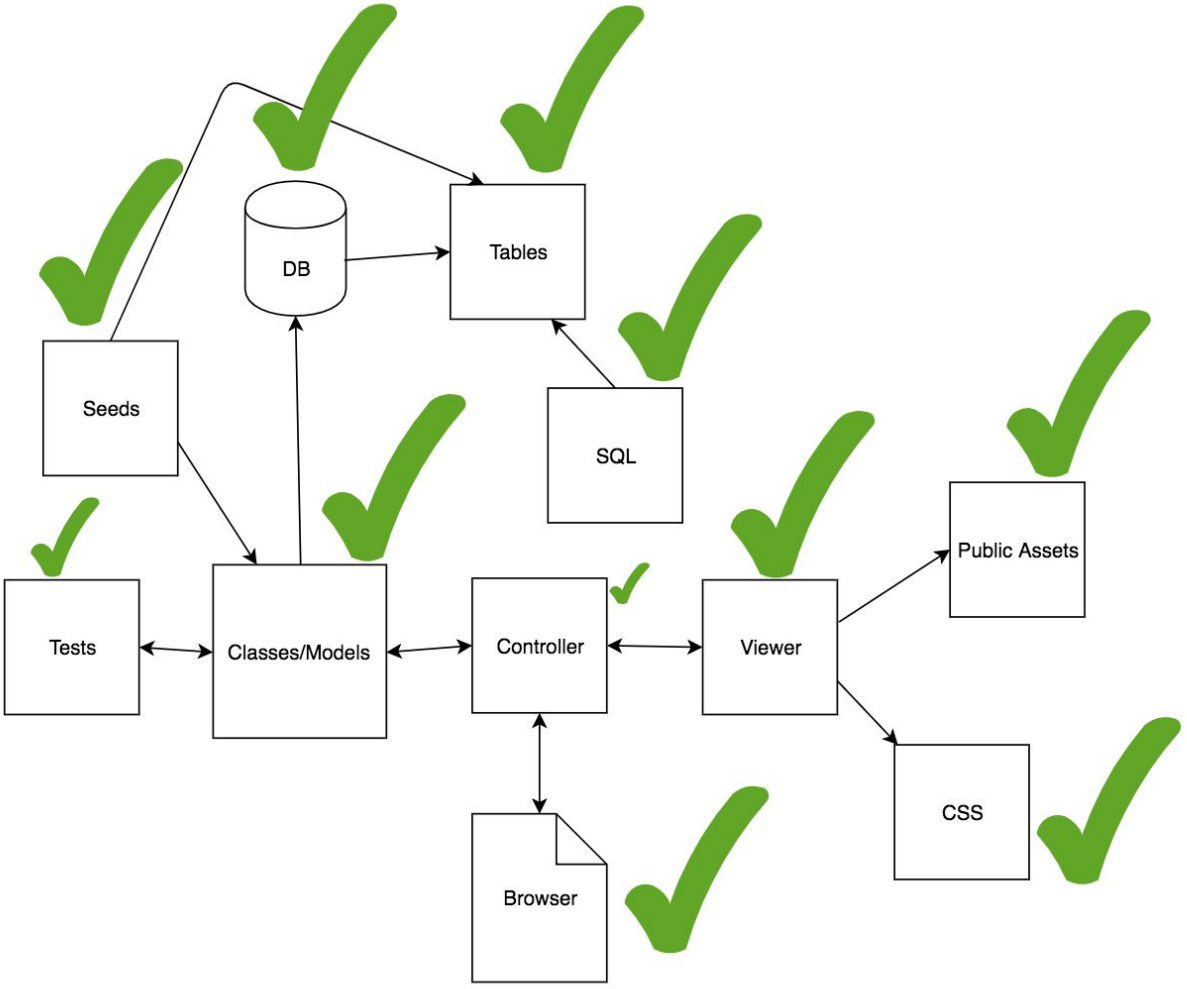
https://github.com/CMilligan26/project_record_shop_inventory

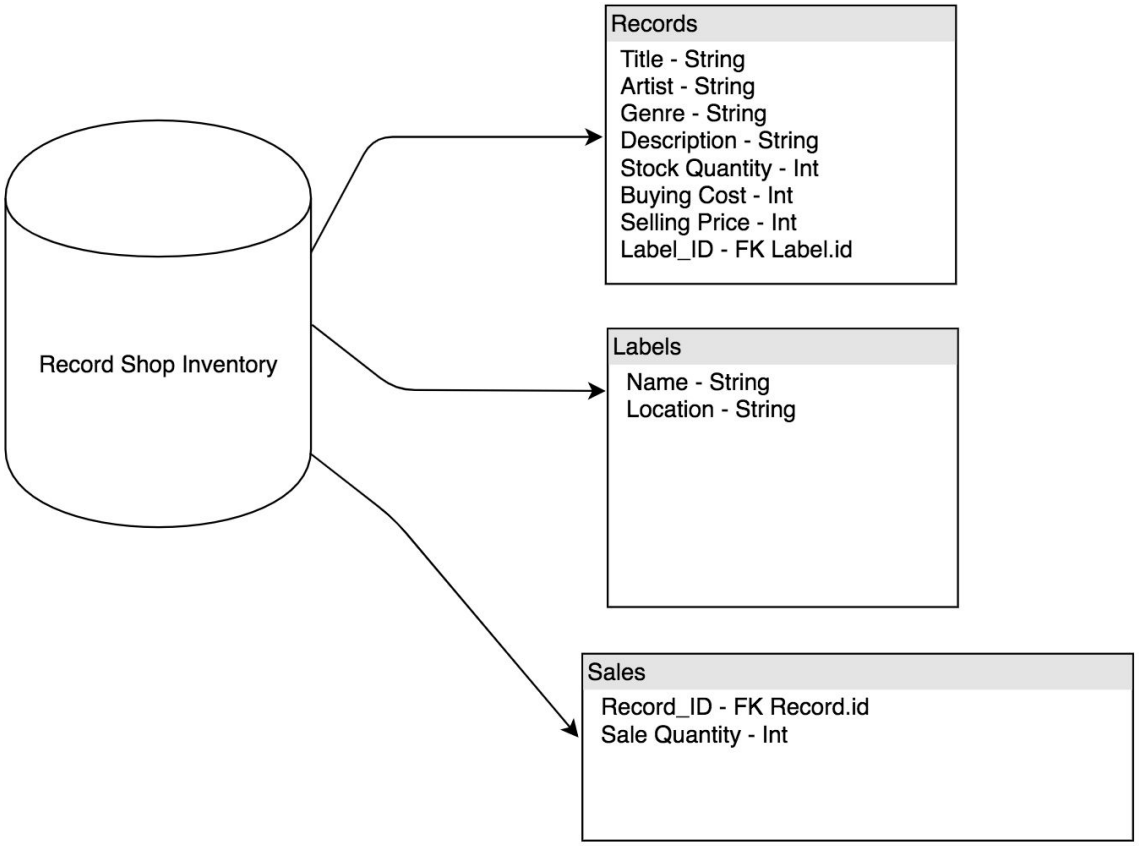
Unit	Ref	Evidence	

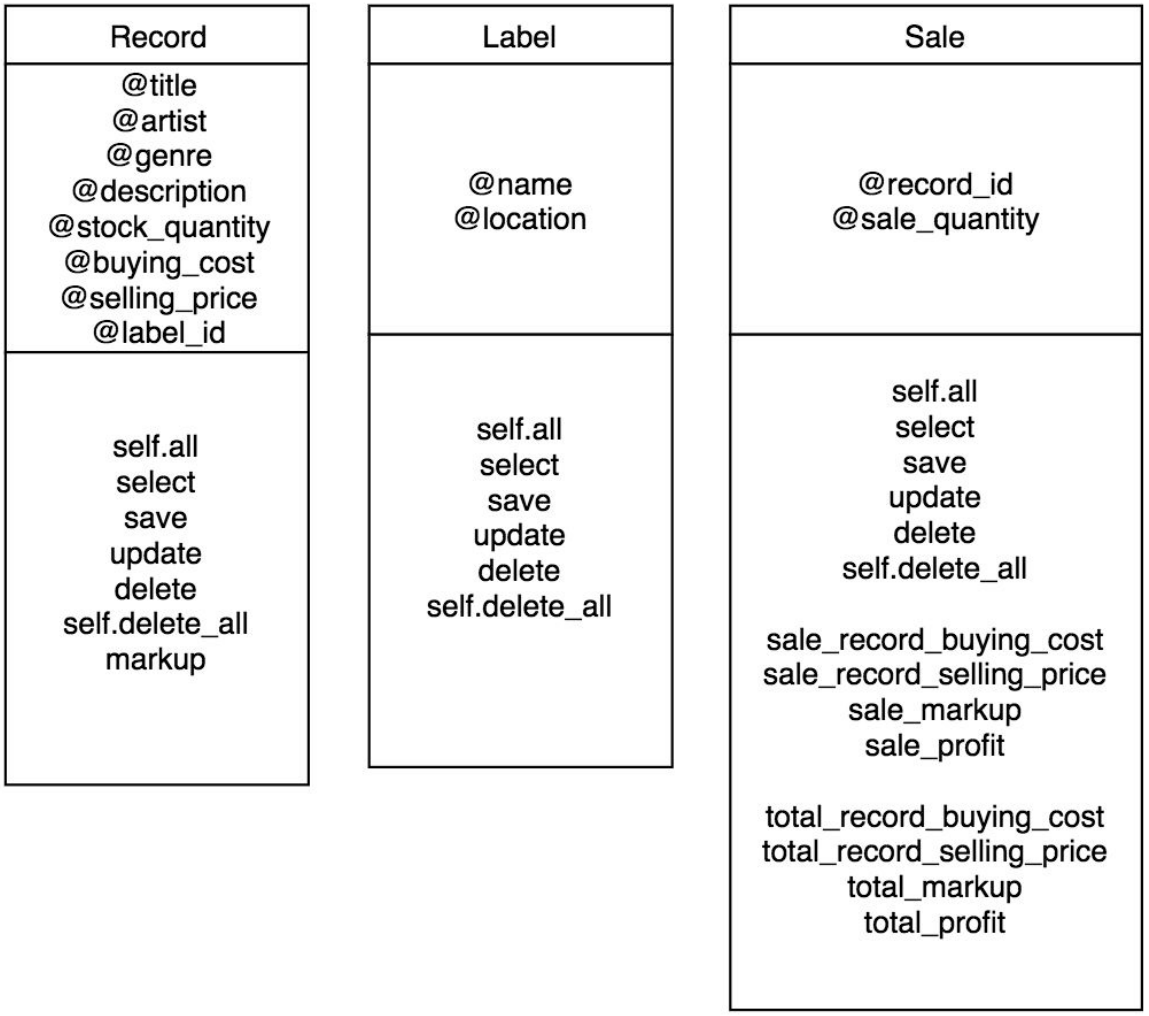
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.
		Description:

Paste Screenshot here




DB
Tables
Seeds
SQL
Tests
Classes/Models
Controller
Viewer
Public Assets
CSS
Browser


Records
Title - String
Artist - String
Genre - String
Description - String
Stock Quantity - Int
Buying Cost - Int
Selling Price - Int
Label_ID - FK Label.id
Record Shop Inventory
Labels
Name - String
Location - String
Sales
Record_ID - FK Record.id
Sale Quantity - Int


Record
@title
@artist
@genre
@description
@stock_quantity
@buying_cost
@selling_price
@label_id
self.all
select
save
update
delete
self.delete_all
markup
Label
@name
@location
self.all
select
save
update
delete
self.delete_all
Sale
@record_id
@sale_quantity
self.all
select
save
update
delete
self.delete_all
sale_record_buying_cost
sale_record_selling_price
sale_markup
sale_profit
total_record_buying_cost
total_record_selling_price
total_markup
total_profit

CodeClan_
Bridging the digital skills gap.

Name

Rob Gordon

Behaviours

Listens to music everyday.
Likes to categorise his own collection of music dependant on his mood.
Interested in many different genres of music.
Takes care of his records, he likes to have them all accounted for.
Likes to view album covers when looking for new records.
Regularly uses music streaming services.
Is very familiar with Discogs, eBay and Amazon.

Demographics

20 - 40 years old.
Has disposable income.
Business owner.

Needs and goals

Needs to store which records he sells.
Needs to store the labels that the records have been bought from.
Needs to record how many of a particular record he sells.
Needs to know the current stock of his record shop and be notified when he is low on stock for individual records.
Needs to sort the stored records by different categories.

His goal is to easily manage and record his stock.

User needs

As a...	I want to...	So that...
record store owner who needs to know which records I have in stock	add new records i get in stock	I know which records I have in stock
record store owner who needs to know which records I have in stock	see all the records i have in stock	I know which records I have in stock
record store owner who needs to know which records I have in stock	update record details if they change or are incorrect	I know which records I have in stock
record store owner who needs to know which records I have in stock	delete records no longer being sold	I know which records I have in stock
record store owner who needs to keep track of sales	add sales	I can view the sales later
record store owner who needs to know which records I have in stock after sales	see all the records I have in stock after sales	I know which records I have still have in stock
record store owner who needs to know which labels I bought records from	see all the labels i bought records from	I know which labels i bought records from

CodeClan
Bridging the digital skills gap.

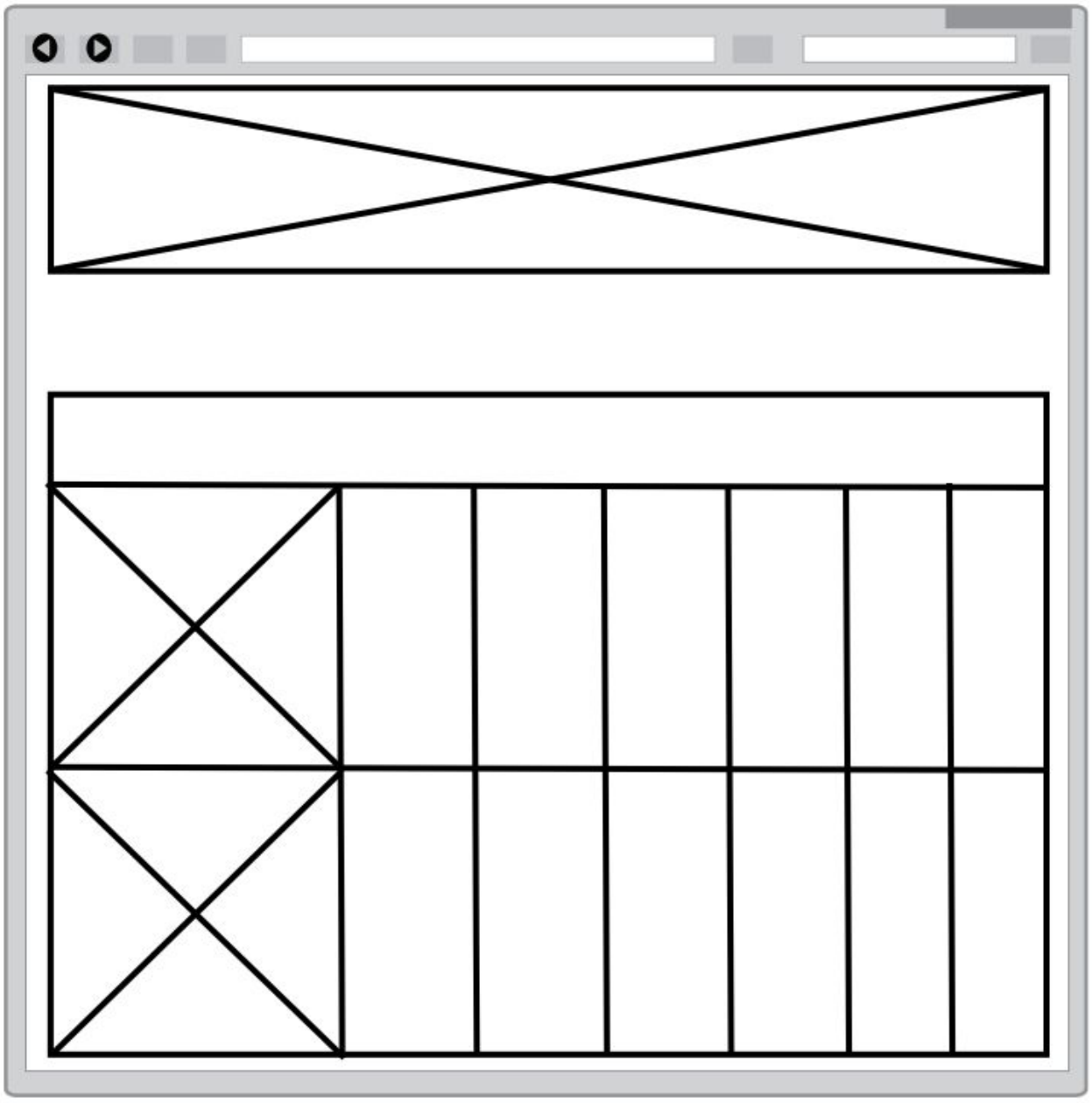
User journey

Your name _____ Date _____

CODECLAN

User action	User action	User action	User action
1 Start Rob opens the app	2 Rob wants to add a new record	3 Rob enters the record details	4 Rob is happy with the details he entered and wants to save it
System response	**System response**	**System response**	**System response**
get request for '/' connect with db select all records from the records table pull html respond with '/'	get request for '/new' pull html respond with '/new'		post request for '/' create record connect to db save to db redirect to '/:id'

User action	User action	User action	User action
5 Rob wants to know that he added the record	6	7	8 End
System response	**System response**	**System response**	**System response**
get request for '/:id' connect to db select record pull html respond '/:id'			



Description here

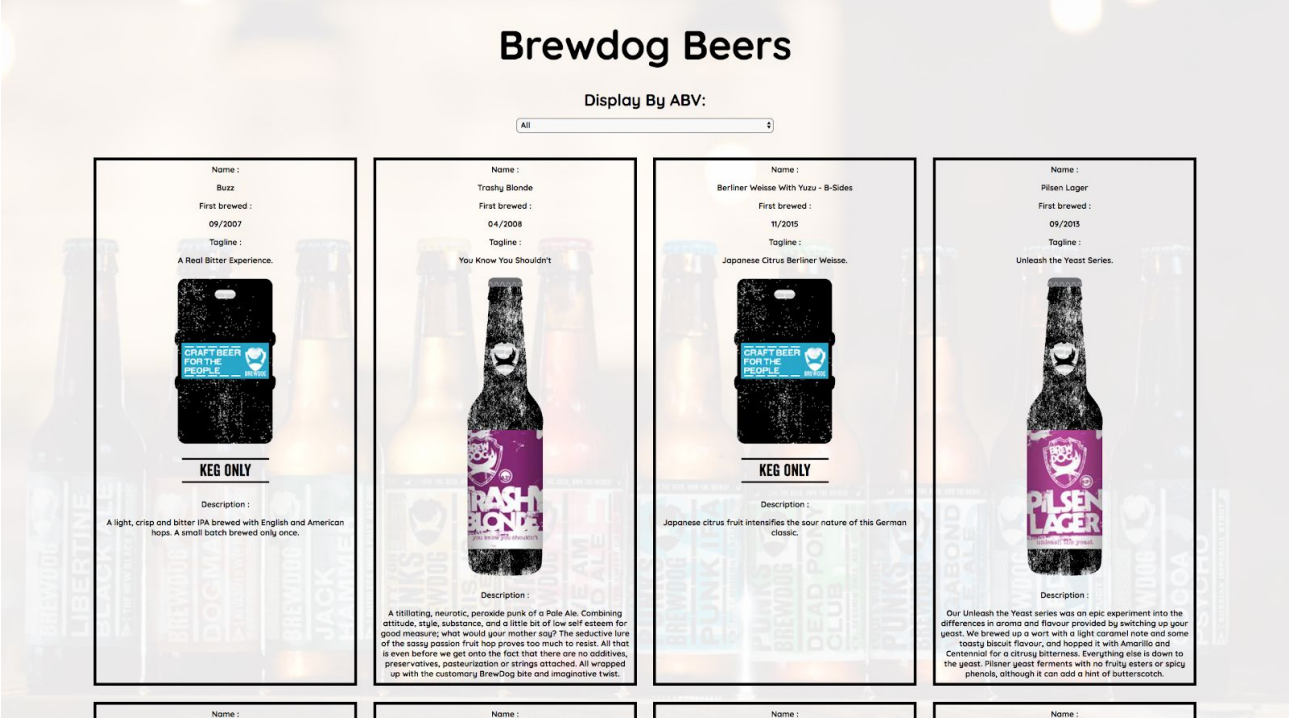
The above graphics and diagrams show the planning process that went into my ‘Spinventory’ record shop inventory app. In order there is a graphic breakdown of the brief which draws out ideas for classes, a graphic map of the app architecture, a graphic representation of the database, a graphic similar to a class diagram without the established relationships between the classes, a user profile, user needs, user journey and a wireframe diagram.

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:	

Paste Screenshot here

```
page.setPageDetails();
const dataModel = new DataModel("https://api.punkapi.com/v2/beers", ['name', 'first_brewed',
'tagline', 'image_url', 'description'], 'abv');
dataModel.getData();
const dataList = new DataList('data_container');
dataList.bindEvents();
const selectView = new SelectView('data_select');
selectView.bindEvents();
});
```



Description here

The above code makes the call to the API and the screenshot shows the data from the API used in the running program.

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:	

Paste Screenshot here

```ruby
require('minitest/autorun')
require('minitest/rg')

require_relative('../models/artist')

class ArtistTest < MiniTest::Test

  def setup
    @artist = Artist.new({'id' => '1', 'artist_name' => 'Test_Artist'})
  end

  def test_artist_exists
    assert_equal(Artists, @artist.class)
  end

  def test_artist_can_have_id
    assert_equal(2, @artist.id)
  end

  def test_artist_has_name
    assert_equal('Test_rtist', @artist.artist_name)
  end

end
```

```
➜  record_shop_inventory git:(master) ✗ ruby specs/artist_spec.rb
Run options: --seed 1843

# Running:

FEF

Finished in 0.001107s, 2710.0270 runs/s, 1806.6847 assertions/s.

  1) Failure:
ArtistTest#test_artist_can_have_id [specs/artist_spec.rb:17]:
Expected: 2
  Actual: 1

  2) Error:
ArtistTest#test_artist_exists:
NameError: uninitialized constant ArtistTest::Artists
Did you mean?  Artist
    specs/artist_spec.rb:13:in `test_artist_exists'

  3) Failure:
ArtistTest#test_artist_has_name [specs/artist_spec.rb:21]:
Expected: "Test_rtist"
  Actual: "Test_Artist"

3 runs, 2 assertions, 2 failures, 1 errors, 0 skips
```

```ruby
require('minitest/autorun')
require('minitest/rg')

require_relative('../models/artist')

class ArtistTest < MiniTest::Test

  def setup
    @artist = Artist.new({'id' => '1', 'artist_name' => 'Test_Artist'})
  end

  def test_artist_exists
    assert_equal(Artist, @artist.class)
  end

  def test_artist_can_have_id
    assert_equal(1, @artist.id)
  end

  def test_artist_has_name
    assert_equal('Test_Artist', @artist.artist_name)
  end

end
```

```
➜  record_shop_inventory git:(master) ✗ ruby specs/artist_spec.rb
Run options: --seed 28739

# Running:

...

Finished in 0.000987s, 3039.5135 runs/s, 3039.5135 assertions/s.

3 runs, 3 assertions, 0 failures, 0 errors, 0 skips
```

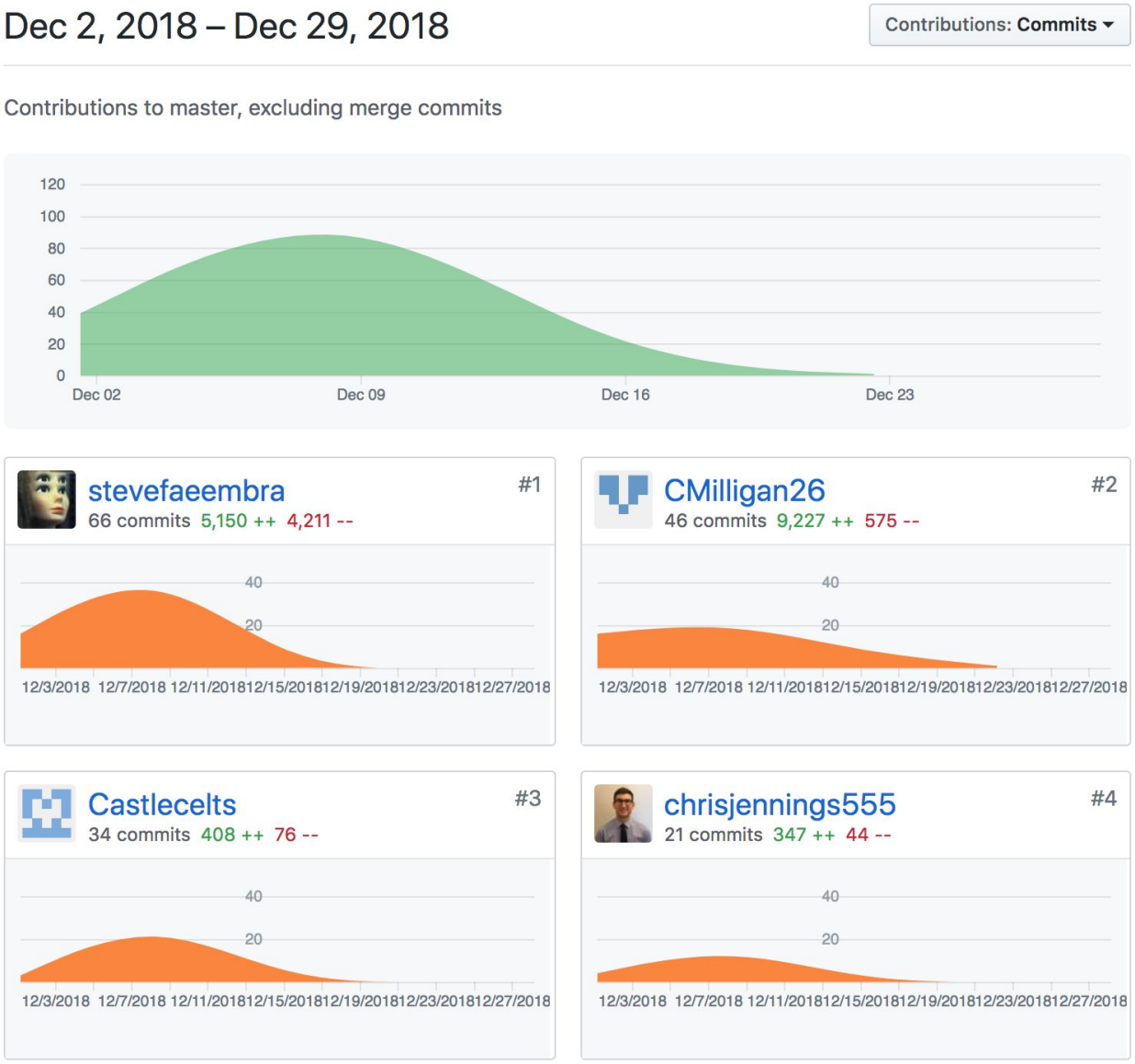
Description here

The above screenshots show three failing tests. The tests are then corrected and the tests are shown to pass. The first test failed because the incorrect class name Artists was expected instead of the correct class name Artist. The second test failed because the incorrect id 2 was expected instead of the correct id 1. The last test failed because of a syntax error where the expected name was Test_rtist instead of the correct Test_Artist.

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:	

Paste Screenshot here



Description here

The above screenshot shows the team members of my Javascript group project.

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

		Description:

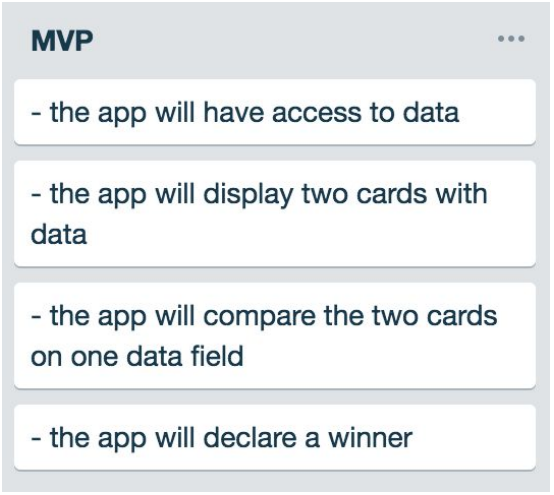
Paste Screenshot here

Browser Game

Create a browser game based on an existing card or dice game. Model and test the game logic and then display it in the browser for a user to interact with.

Write your own MVP with some specific goals to be achieved based on the game you choose to model.

You might use persistence to keep track of the state of the game or track scores/wins. Other extended features will depend on the game you choose.

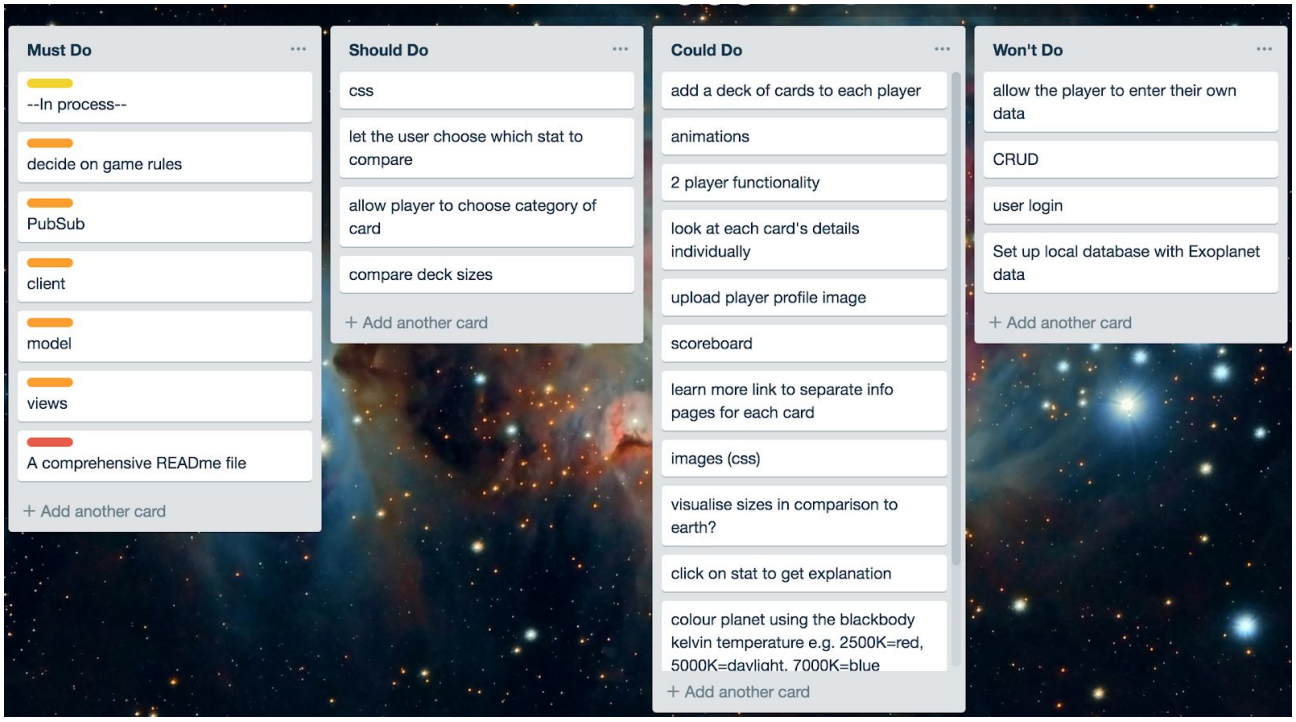


Description here

The above screen shots show the brief provided to us by our instructors and the MVP that we wrote as a group and presented to the instructors and was approved.

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

Paste Screenshot here



Description here

The above screenshot shows the MOSCOW planning created on trello during my group project.

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

Paste Screenshot here

Acceptance Criteria	Expected Result	Pass/Fail
A user is able to enter record details and save the record details.	The data should be saved and the resulting record displayed to the user when they click the save button.	Pass
A user is able to enter a word into the search bar.	Data should be presented that matches the criteria entered by the user when they click search.	Pass
A user is able to enter artist	The data should be saved and	Pass

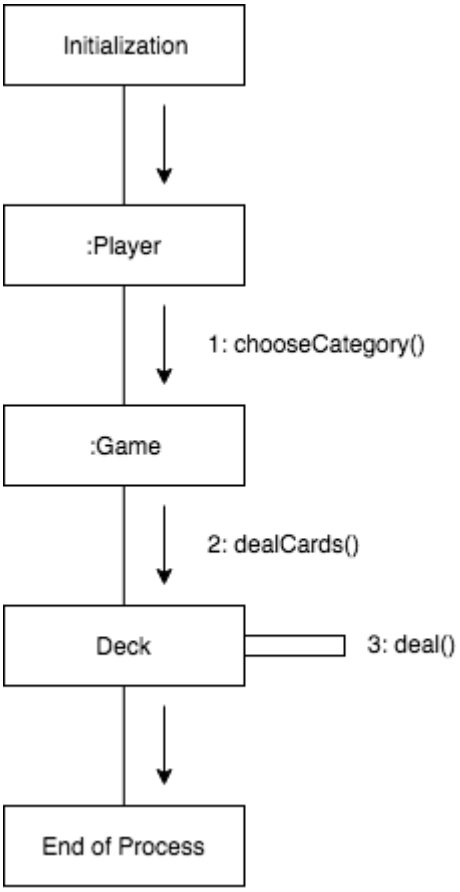
details and save the artist details.	the resulting artist displayed to the user when they click the save button.	
A user is able to enter sales details and save the sales details.	The data should be saved and all sales displayed to the user when they click the save button.	Pass

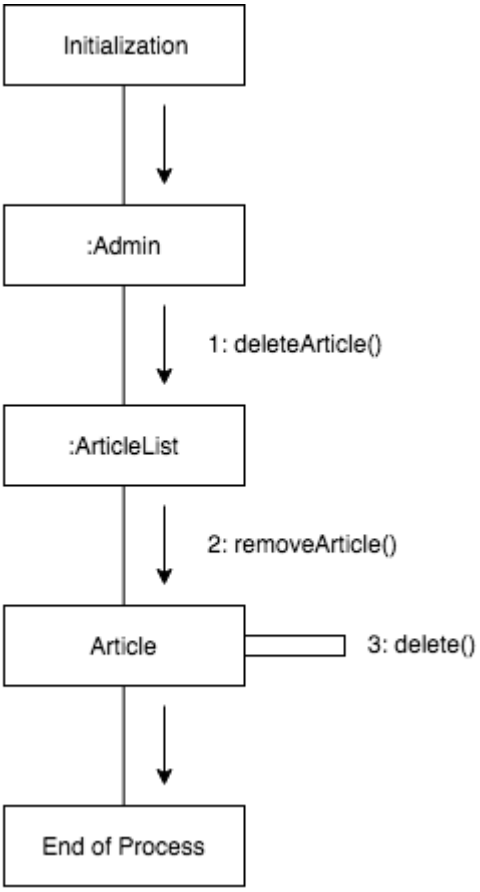
Description here

The above Acceptance Criteria and Test Plan describes what actions the user should be able to perform on my record inventory management app, what the expected result should be once they have performed these actions and whether the app successfully performed the actions and passed the test.

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

Paste Screenshot here



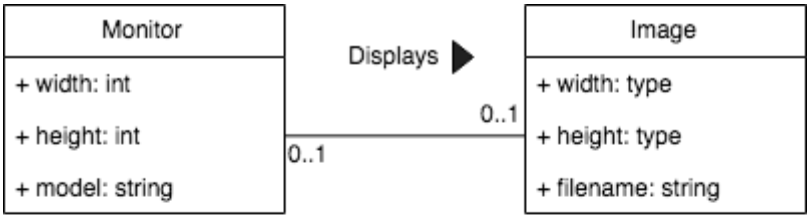


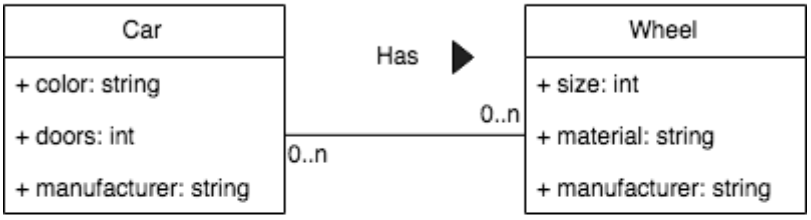
Description here

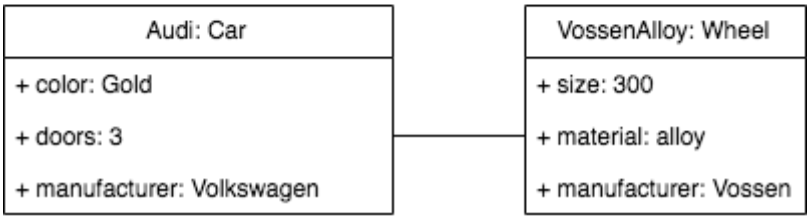
The two collaboration diagrams above show the sequence of events that occur in each process and the objects and methods involved in these sequence steps.

Unit	Ref	Evidence	
P	P.8	Produce two object diagrams.	
		Description:	

Paste Screenshot here







Description here

The above diagrams show two examples of class diagrams and accompanying object diagrams. The object diagrams assign example data to the properties of the class diagrams and provide context to the use of the objects.

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

Paste Screenshot here

Bug/Error	**Solution**	**Date**
Unhandled Rejection (SyntaxError): Unexpected token P in JSON at position 0	Restart Java application to refresh the API.	06/02/2019
User able to click card category and change outcome after initial click.	Prevent interaction with the categories before next match starts.	12/12/2018
Computer turn outcome too quick for the player to follow.	Implemented PubSub to allow for communication between the elements and provide feedback between turn stages.	10/12/2018
Computer player plays the same card they just won from the player on the next turn.	Unshifted the card into the front of the computer players hand array to prevent this.	08/12/2018
Total profits from record sales not accurate after changing record retail price.	Recorded record price at time of sale and number of sales in order to accurately calculate the profits.	10/11/2018

Description here

The above bug tracking report shows recorded bugs/errors that I have encountered and the solutions to these issues as well as the date the bugs were encountered on.

Week 12

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

Paste Screenshot here

```
@Test
public void canBuyItem(){
    customer.buyItem(didgeridoo);
    assertEquals(didgeridoo, customer.getBoughtItems().get(0));
}
```

```
CustomerTest 7 ms
canBuyItem 7 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

```
public class Customer {
    private String name;

    private ArrayList<ISell> boughtItems;

    public Customer(String name) {
        this.name = name;
        this.boughtItems = new ArrayList<>();
    }

    public String getName() { return name; }

    public void buyItem(ISell itemToBuy) { this.boughtItems.add(itemToBuy); }

    public ArrayList<ISell> getBoughtItems() { return boughtItems; }

}
```

```
public abstract class StockItem implements ISell {

    private String name;
    private String description;
    private double retailPrice;
    private double wholesalePrice;
```

```java
package ShopStuff;

import Interfaces.ISell;

public class Lesson implements ISell {
    private String relatedInstrument;
    private String instructor;
    private String dateAndTime;


    public Lesson(String relatedInstrument, String instructor, String dateAndTime) {
        this.relatedInstrument = relatedInstrument;
        this.instructor = instructor;
        this.dateAndTime = dateAndTime;
    }

    public String getRelatedInstrument() {
        return relatedInstrument;
    }

    public String getInstructor() {
        return instructor;
    }

    public String getDateAndTime() {
        return dateAndTime;
    }
}
```

```java
@Test
public void canBuyLesson(){
    customer.buyItem(lesson);
    assertEquals(lesson, customer.getBoughtItems().get(0));
}
```

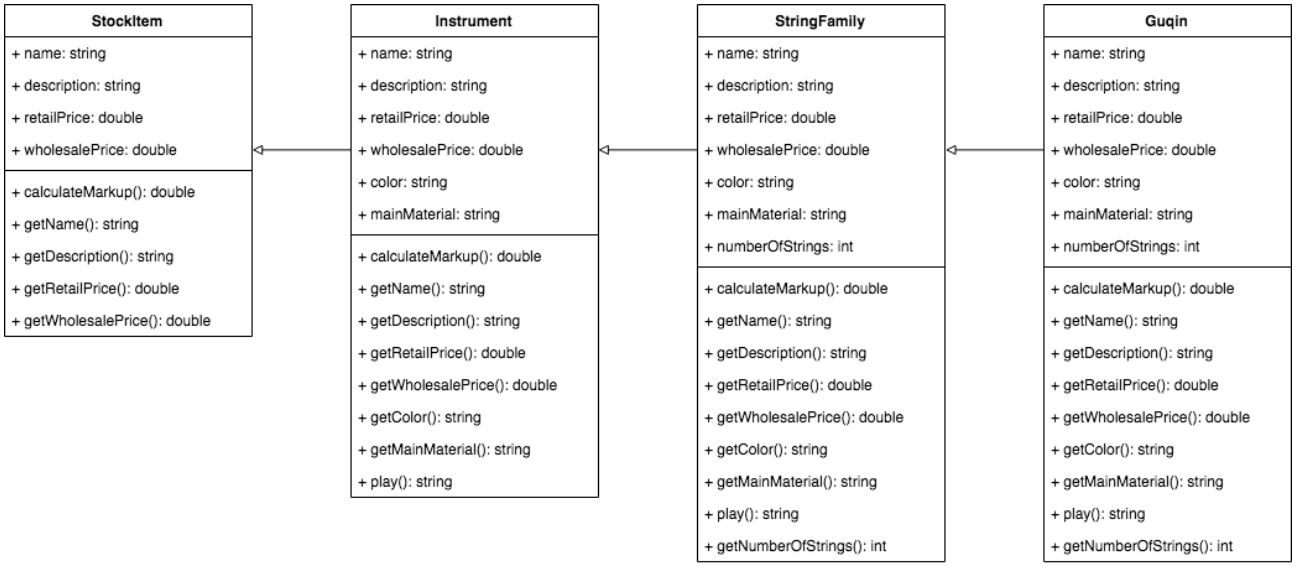
```
CustomerTest 2 ms
  canBuyLesson 2 ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_181.jdk/Contents/Home/bin/java ...

Process finished with exit code 0
```

Description here

The above screenshots show a Customer object implementing a method buyItem and passing a Didgeridoo object as an argument. This is then added to the customers list of bought items as shown by the test which passes. The polymorphism comes from the fact that StockItem implements the interface ISell and the Didgeridoo object inherits from the StockItem class. The customer is able to pass any ISell item into it's bought array. Therefore the customer is able to treat the StockItem/Didgeridoo as an ISell object. The final screenshots show another class Lesson implementing the ISell interface and another test showing that this can also be bought by the customer, and a test shows that this passes.

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

Paste Screenshot here



Description here

The above diagram shows the inheritance chain of classes StockItem, Instrument, StringFamily and Guqin. It also explicitly shows the properties and methods which are inherited from the parent classes.

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

Paste Screenshot here

```
package ShopStuff;

public abstract class StockItem {

    private String name;
    private String description;
    private double retailPrice;
    private double wholesalePrice;

    public StockItem(String name, String description, double retailPrice, double wholesalePrice) {
        this.name = name;
        this.description = description;
        this.retailPrice = retailPrice;
        this.wholesalePrice = wholesalePrice;
    }

    public double calculateMarkup() { return this.retailPrice - this.wholesalePrice; }

    public String getName() { return name; }

    public String getDescription() { return description; }

    public double getRetailPrice() { return retailPrice; }

    public double getWholesalePrice() { return wholesalePrice; }
}
```

Description here

The above screenshot shows the use of encapsulation in a class named StockItem. The encapsulation aspects means that the class is self contained and properties cannot be accessed or changed directly from outside of the class. This is achieved through private properties and methods that return information which have to be accessed through an instance of the 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.	
		Description:	

Paste Screenshot here

```java
package ShopStuff;

public abstract class StockItem {

    private String name;
    private String description;
    private double retailPrice;
    private double wholesalePrice;

    public StockItem(String name, String description, double retailPrice, double wholesalePrice) {
        this.name = name;
        this.description = description;
        this.retailPrice = retailPrice;
        this.wholesalePrice = wholesalePrice;
    }

    public double calculateMarkup() { return this.retailPrice - this.wholesalePrice; }

    public String getName() { return name; }

    public String getDescription() { return description; }

    public double getRetailPrice() { return retailPrice; }

    public double getWholesalePrice() { return wholesalePrice; }
}
```

```java
package InstrumentStuff.Abstracts;

import CustomerStuff.Customer;
import ShopStuff.StockItem;

public abstract class Instrument extends StockItem {
    private String color;
    private String mainMaterial;


    public Instrument(String name, String description, double retailPrice, double wholesalePrice, String color,
                      String mainMaterial) {
        super(name, description, retailPrice, wholesalePrice);
        this.color = color;
        this.mainMaterial = mainMaterial;
    }

    public String getColor() {
        return this.color;
    }

    public String getMainMaterial() {
        return this.mainMaterial;
    }

    public String play(Customer customer) {
        return customer.getName() + " is playing the " + this.getName();
    }
}
```

```java
package InstrumentStuff.Abstracts;

public abstract class StringFamily extends Instrument{
    private int numberOfStrings;

    public StringFamily(String name, String description, double retailPrice, double wholesalePrice, String color,
                        String mainMaterial, int numberOfStrings) {
        super(name, description, retailPrice, wholesalePrice, color, mainMaterial);
        this.numberOfStrings = numberOfStrings;
    }

    public int getNumberOfStrings() {
        return this.numberOfStrings;
    }
}
```

```java
package InstrumentStuff;

import InstrumentStuff.Abstracts.StringFamily;

public class Guqin extends StringFamily {

    public Guqin(String name, String description, double retailPrice, double wholesalePrice, String color,
                 String mainMaterial, int numberOfStrings) {
        super(name, description, retailPrice, wholesalePrice, color, mainMaterial, numberOfStrings);
    }
}
```

```java
guqin = new Guqin(
        name: "Paulownia Wood Guqin, Chinese 7-string Zither, for Beginner",
        description: "Great for beginners!",
        retailPrice: 180,
        wholesalePrice: 130,
        color: "Brown",
        mainMaterial: "Wood",
        numberOfStrings: 7);
```

```java
@Test
public void canCalculateMarkup() { assertEquals( expected: 50, guqin.calculateMarkup(), delta: 0.001); }
```

Description here

In the screenshots above you can see that there is the top parent class named StockItem which contains methods and instance variables common to all stock items. The next class is named Instrument and this inherits from StockItem and contains additional methods and instance variables common to instruments. The next class is named StringFamily and this inherits from Instrument and again contains additional methods and instance variables common to members of the string family of instruments. Finally the child class Guqin inherits from the StringFamily class. The second to last screenshot shows the initialisation of a new Guqin object containing the properties of it's parents and the last screenshot shows the Guqin object calling a method which is inherited from the top of the inheritance chain, StockItem.

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:	

Paste Screenshot here

```
public StockItem getMostExpensiveProduct(){
    StockItem mostExpensive = this.stock.get(0);
    for (int i = 1; i < this.stock.size(); i++) {
        if (this.stock.get(i).getRetailPrice() > mostExpensive.getRetailPrice()) {
            mostExpensive = this.stock.get(i);
        }
    }
    return mostExpensive;
}
```

```
public void sortStockByMarkUp(){
    ArrayList<StockItem> stock = new ArrayList<>();
    for (StockItem stockItem : this.stock) {
        stock.add(stockItem);
    }

    for (int i = 0; i < stock.size(); i++) {
        StockItem temp = stock.get(i);
        for (int o = i+1; o < stock.size(); o++) {
            if (stock.get(o).calculateMarkup() < temp.calculateMarkup()) {
                temp = stock.get(o);
            }
        }
        stock.remove(temp);
        stock.add(i, temp);
    }

    this.stock = stock;
}
```

Description here

The aim of the first algorithm is to return the StockItem with the most expensive retail price from an array of StockItems named stock. It achieves this by first setting the variable mostExpensive to the first item in the array. The for loop then compares the retail price of the mostExpensive variable to the retail price of the current item in the array. If the current items retail price is greater than the retail price of mostExpensive then mostExpensive is set to the current item. This is iterated over the whole array and then the method returns the variable mostExpensive.
The aim of the second algorithm is to sort the same stock array by the markup of each stockItem. The algorithm achieves this by creating a copy of the current stock array so this can be manipulated by the method. The method checks every item in the copied array using a for loop and it creates a temporary variable to store the current item. The current item is compared to every other item in the array using another for loop and sets the temporary variable to the current item if its markup is less than the temporary variable markup. It then removes the item from the stock array and adds it back into the current index. The result is a sorted array which then replaces the original array belonging to the class.