WfEEK_02

An example of an array in a program

```ruby
# collectionOfNumbers is an array
collectionOfNumbers = [5, 10, 15, 20, 25]
```

A function that uses the array

```ruby
#printNumbers is a function that prints out the array to the console
def printNumbers(collectionOfNumbers)
  puts collectionOfNumbers
end

printNumbers(collectionOfNumbers)
```

The result of the function running

```
→  week_02 git:(master) ✗ ruby pda.rb
5
10
15
20
25
→  week_02 git:(master) ✗
```

An example of a hash in a program

```ruby
#listOfCountries is a hash of countries and their corresponding regions
listOfCountries = { "Croatia" => "Europe", "Brasil" => "South America", "Uganda" => "Africa"}
```

A function that uses the hash

```ruby
#listOfCountries is a hash of countries and their corresponding regions
listOfCountries = { "Croatia" => "Europe", "Brasil" => "South America", "Uganda" => "Africa"}

#displayListOfCountries is a function that prints out a string with the hash of listOfCountries
def displayListOfCountries(listOfCountries)
  listOfCountries.each do |key, value|
  puts "#{key} is located in #{value}"
end
end

displayListOfCountries(listOfCountries)
```

The result of the function running

```
[➜  week_02 git:(master) ✗ ruby pda.rb
Croatia is located in Europe
Brasil is located in South America
Uganda is located in Africa
➜  week_02 git:(master) ✗
```

WEEK_03

An example of a function that searches data

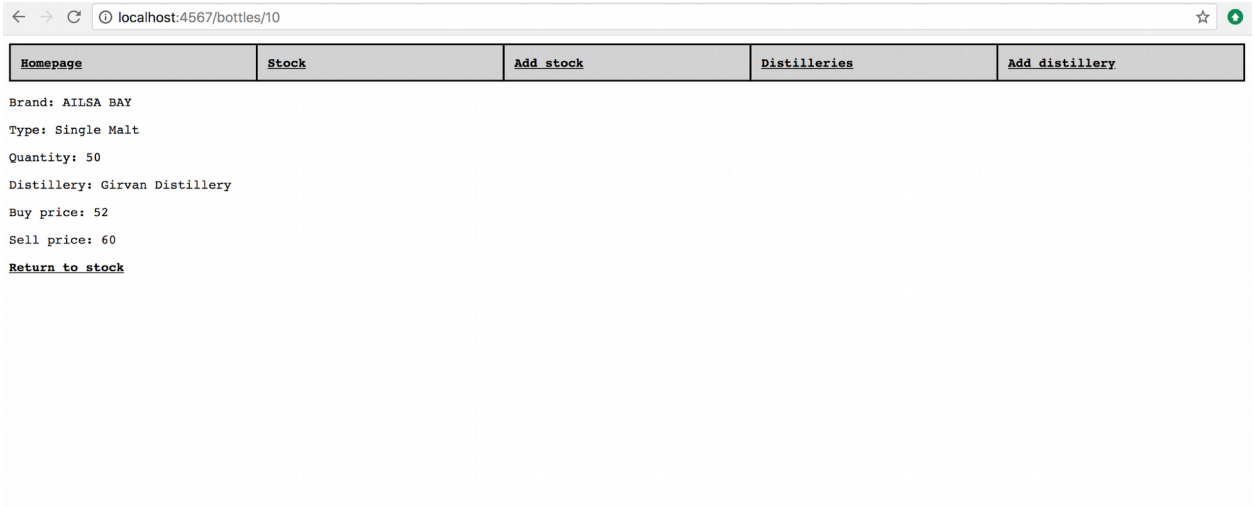
```ruby
#self.find function searches data to find a bottle of whisky by id in database
def self.find( id )
  sql = "SELECT * FROM bottles WHERE id = $1"
  values = [id]
  bottle = SqlRunner.run( sql, values )
  result = Bottle.new( bottle.first )
  return result
end
```

The result of the function running

In the console

```
[1] pry(main)> Bottle.find(10)
=> #<Bottle:0x007fa3408a9540 @buy_price=52, @distillery_id=10, @id=10, @name="AILSA BAY", @quantity=50, @sell_price=60, @type="Single Malt">
[2] pry(main)>
```

In the browser



Function that sorts data

```ruby
#self.all function sorts data to display bottles of whisky by quantity in ascending order
def self.all()
  sql = "SELECT * FROM bottles
  ORDER BY quantity ASC"
  bottle_data = SqlRunner.run(sql)
  bottles = Bottle.map_items(bottle_data)
  return bottles
end
```

The result of the function running

In the console #data is not sorted here

```
inventory=# SELECT * FROM bottles;
 id |           name           |    type     | distillery_id | quantity | buy_price | sell_price
----+--------------------------+-------------+---------------+----------+-----------+------------
 10 | AILSA BAY                | Single Malt |            10 |       50 |        52 |         60
 11 | ARDMORE 20 YEAR OLD 1996 | Single Malt |            11 |       55 |        45 |         65
 13 | CHIVAS REGAL 12 YEAR OLD | Blended     |            13 |       30 |        21 |         34
 14 | Highland park            | Single Malt |            12 |       55 |        20 |         40
 15 | AILSA BAY                | Single Malt |            14 |       50 |        52 |         60
 16 | ARDMORE 20 YEAR OLD 1996 | Single Malt |            15 |       55 |        45 |         65
 17 | DALMORE CIGAR MALT       | Single Malt |            16 |       15 |        63 |         85
 18 | CHIVAS REGAL 12 YEAR OLD | Blended     |            17 |       30 |        21 |         34
(8 rows)
```

In the browser #data is now sorted and ordered by quantity in ascending order with 15 being the smallest quantity in the dataset

SHOW ORDER

Brand: DALMORE CIGAR MALT

Type: Single Malt

Quantity: 15

Distillery: The Dalmore Distillery

Buy price: £63

Sell price: £85

Stock level: Low (below 20 bottles, order more stock soon!)

Mark up: £22

UPDATE ORDER
DELETE DISTILLERY

SHOW ORDER

Brand: CHIVAS REGAL 12 YEAR OLD

Type: Blended

Quantity: 30

Distillery: Strathisla distillery

Buy price: £21

Sell price: £34

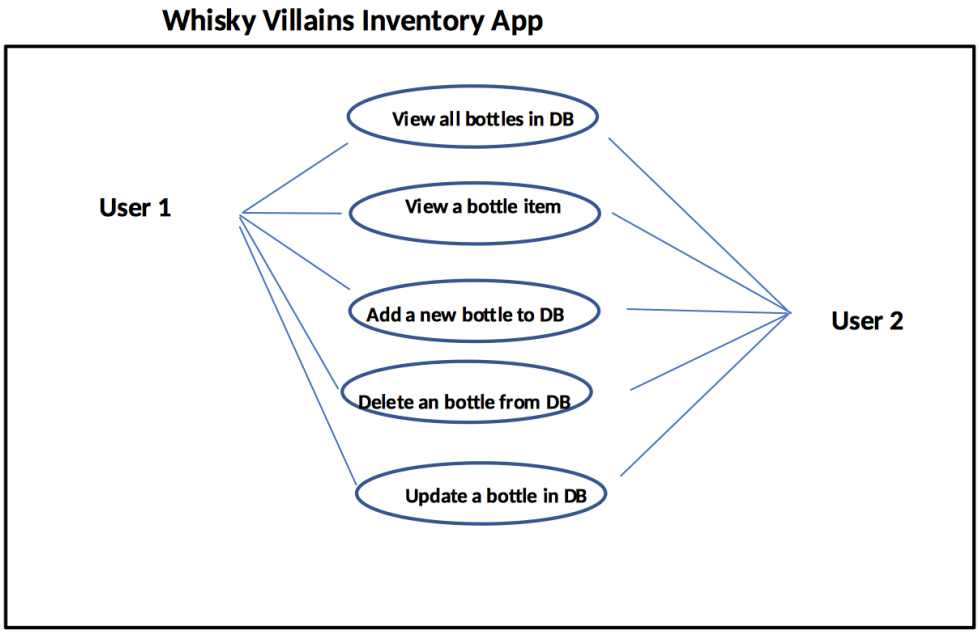
Stock level: Medium (between 20 and 50 bottles)

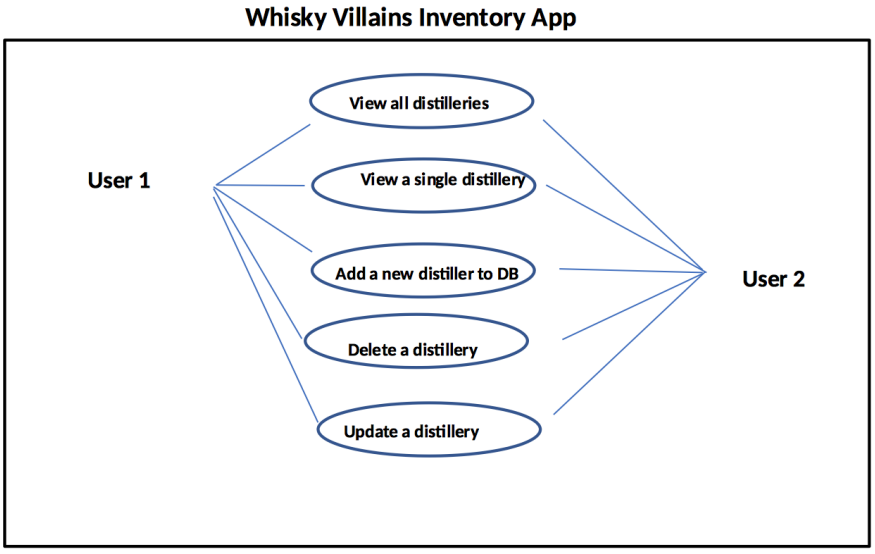
Mark up: £13

UPDATE ORDER
DELETE DISTILLERY

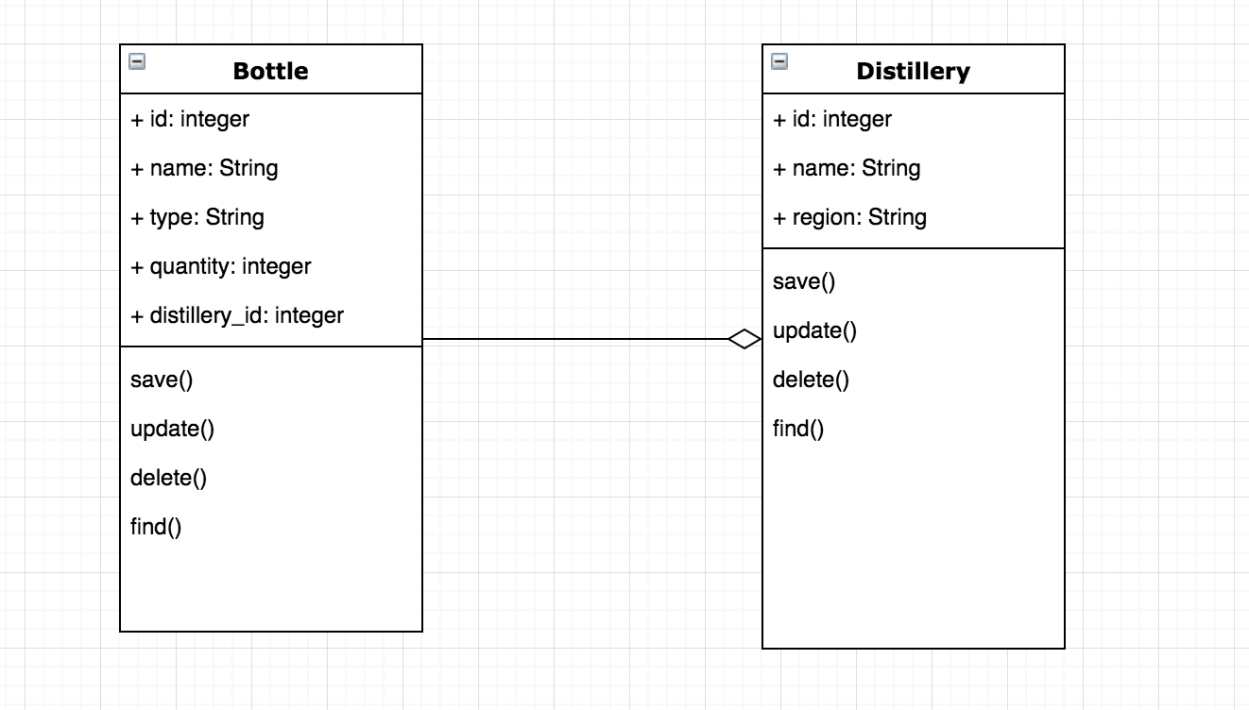
WEEK_05

A use case diagram

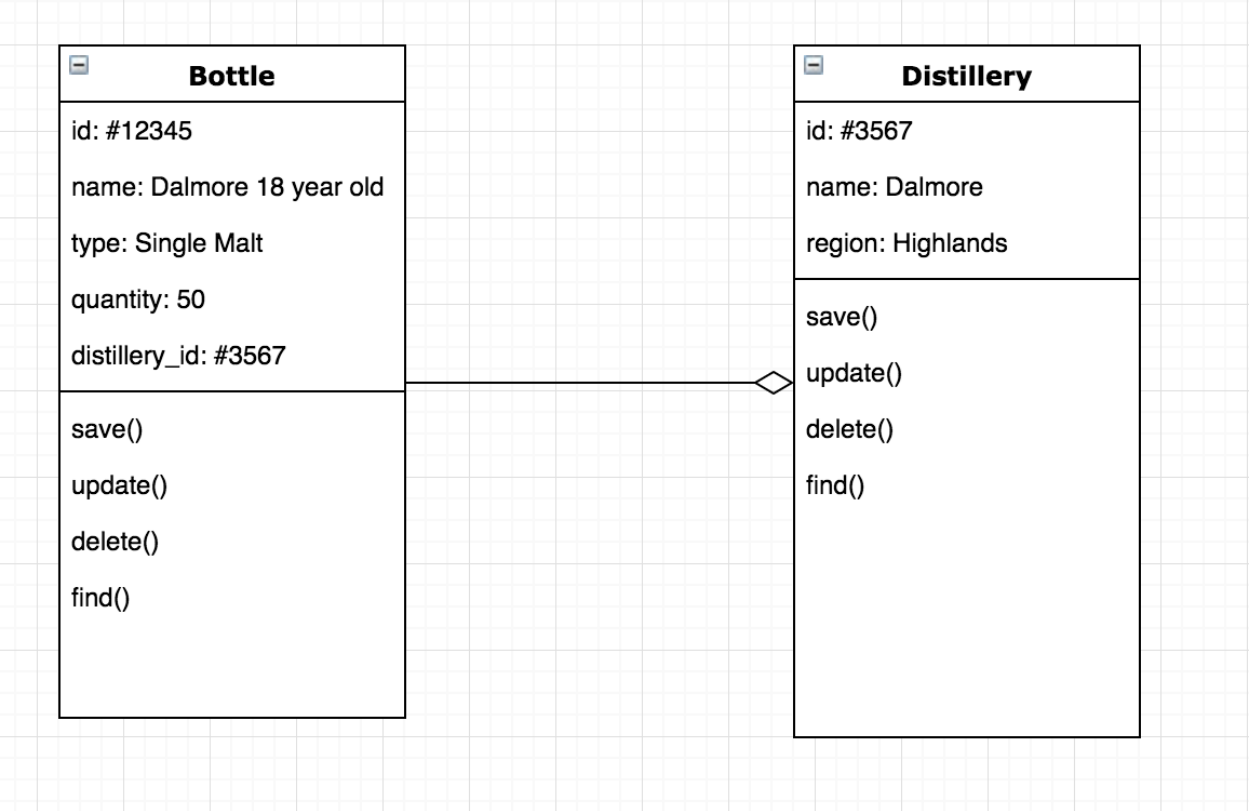




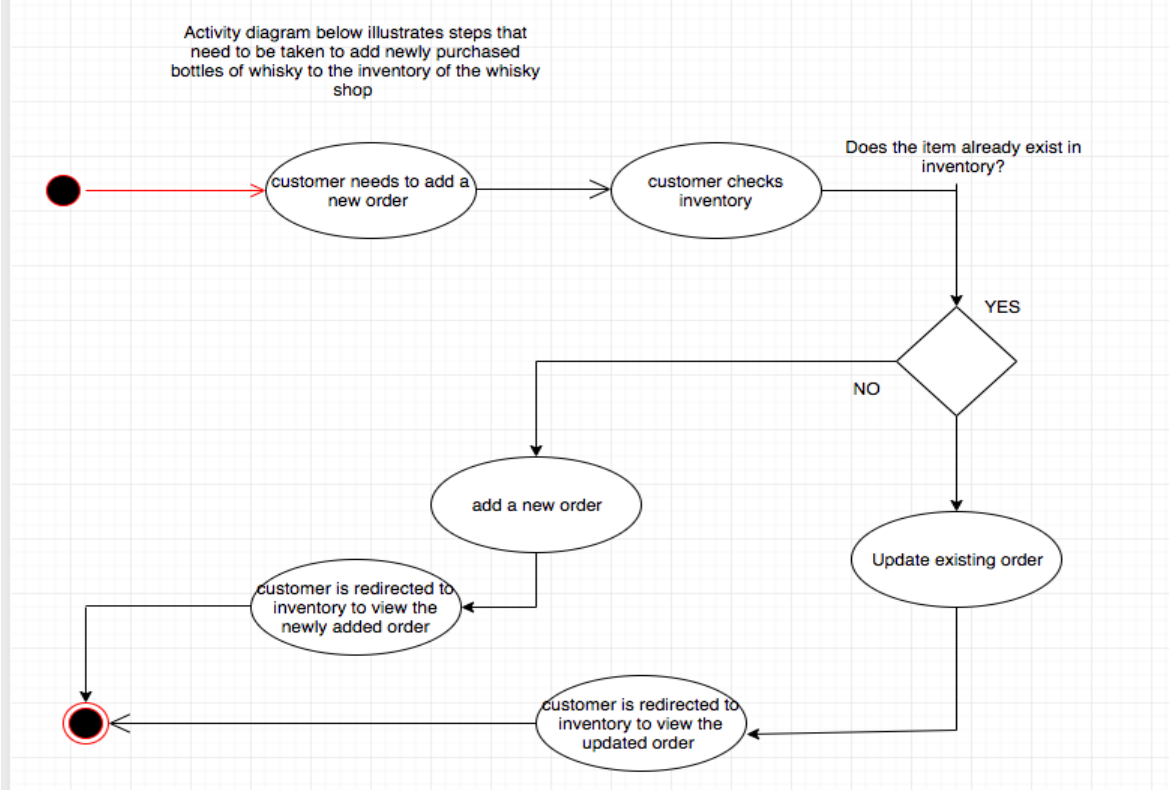
A class diagram



An object diagram



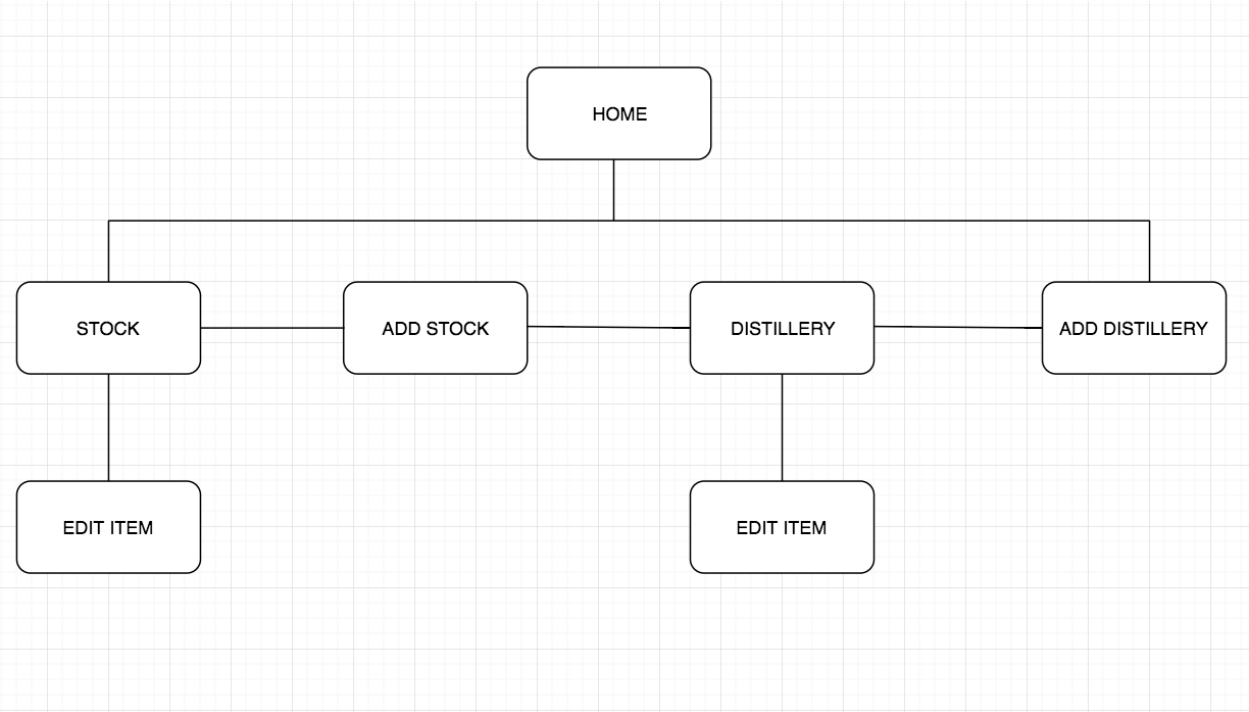
An activity diagram



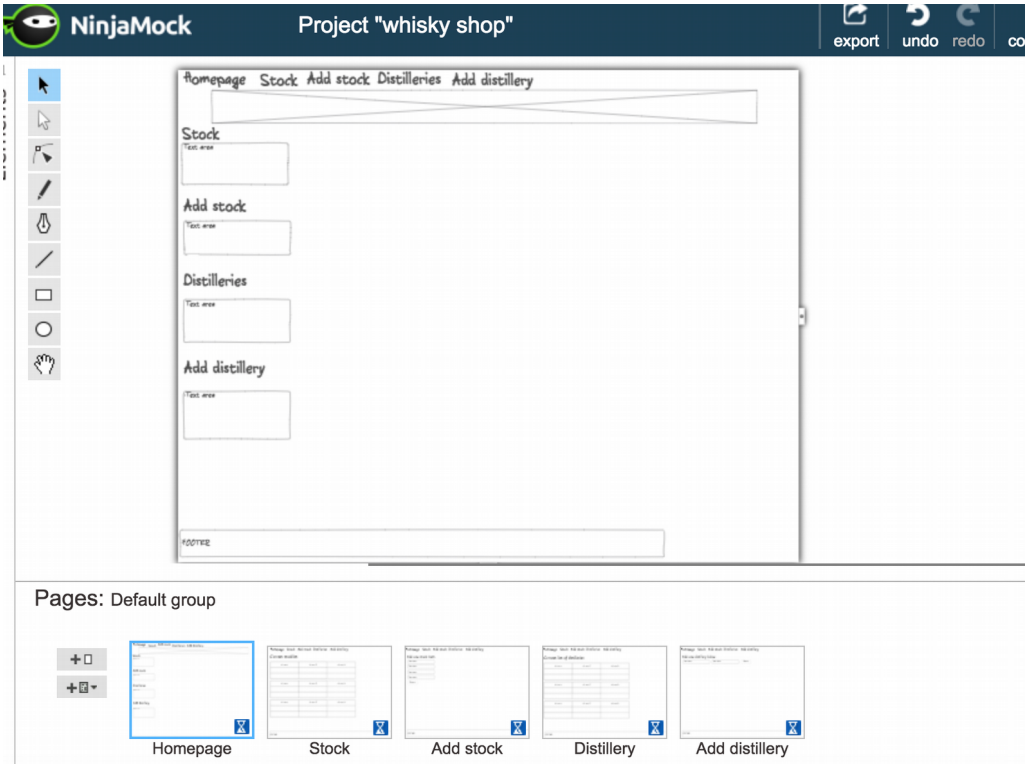
Implementation constraints plan

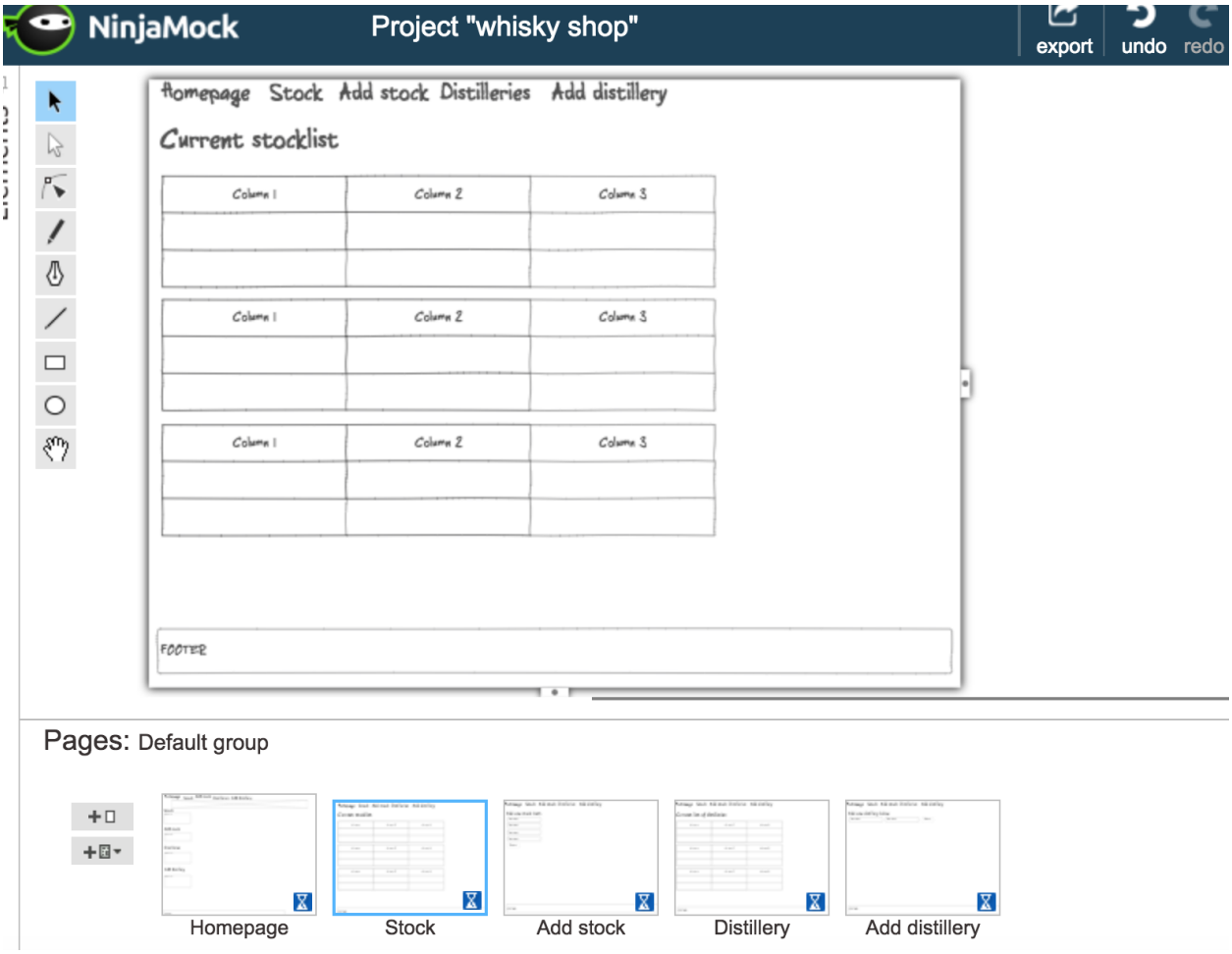
Topic	Possible effect of constraint on product	Solution
Hardware and software platforms	Users might not be able to use the whisky inventory app using different types of devices(PC, tablet, phone). This will restrict the usability of the product, limiting access options to the app.	Create a fully responsive app that scales to the viewport of the device the app is accessed on, the look and usability of the app is not restricted.
Performance requirements	The app does not load in every browser or app looks significantly different due to different browsers rendering html & css in different ways.	Use semantic HTML with semantic tags for example <p> for paragraph to make it clear to all browsers that enclosed text between <p> tags refer to paragraphs and must be displayed as such.
Persistent storage and transactions	When the user is using the whisky inventory app, they need to add new whisky bottles to the inventory and that information needs to be stored somewhere for them to review it later. This process of performing CRUD operations on the database must work flawlessly every time to avoid delays and bottlenecks in inventory	All the CRUD operations(CREATE, READ, UPDATE, DELETE) must be thoroughly before releasing the product to the client
Usability	The inventory information must be displayed in a clear, organised and intuitive way in order to avoid user frustration with a non-intuitive confusing layout	Following best practice of UX (user experience) guidelines when designing the layout of the web app
Budgets	Failing to stick to the budget and running out of funds before project completion would lead to an app with limited functionality which will undermine the overall success of the app	Complete the MVP(minimally viable product) first so that even if you run out of budget at some point during the project, you will have a basic functioning app
Time limitations	Failing to stick to deadlines and running out of time before project completion would lead to a half finished app, with undetected bugs present in code due to an incomplete testing cycle	Prioritise planning and perform unit testing throughout the project all completed code is thoroughly tested. As above, prioritise MVP before moving onto extensions.

A site map



Produce two wireframes designs



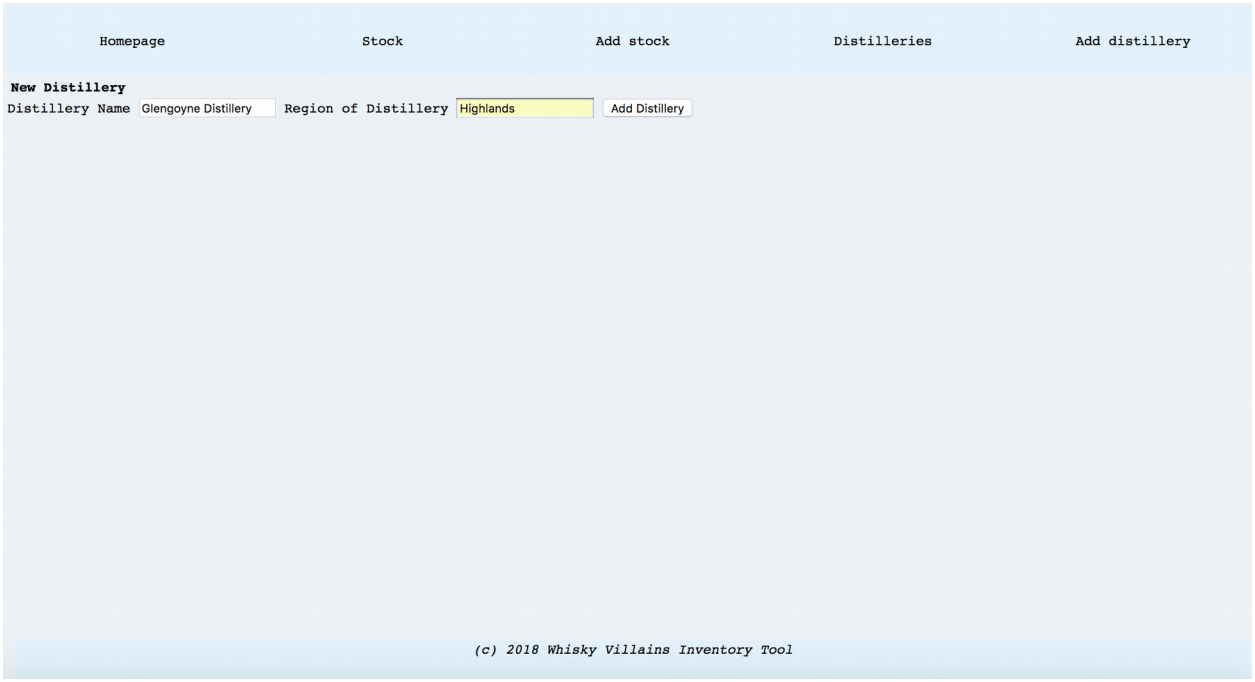


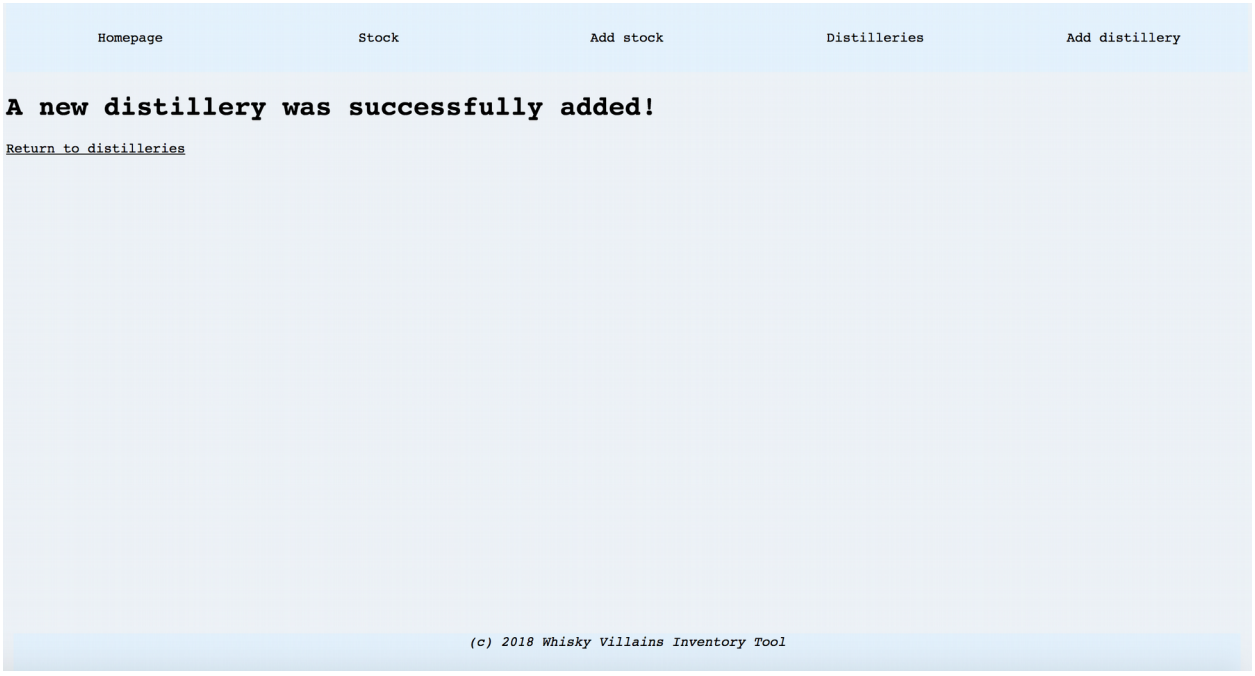
Take a screenshot of an example of pseudocode for a function

```
// transferDinosaur function needs to transfer a dinosaur between two paddocks

// pass in three arguments to this function: which dino needs to be moved,
// the paddock that the dino needs to vacate and the paddock that it needs to move to

// remove the dino from the paddock that it needs to vacate

// add the dino to the paddock that it needs to move to
```

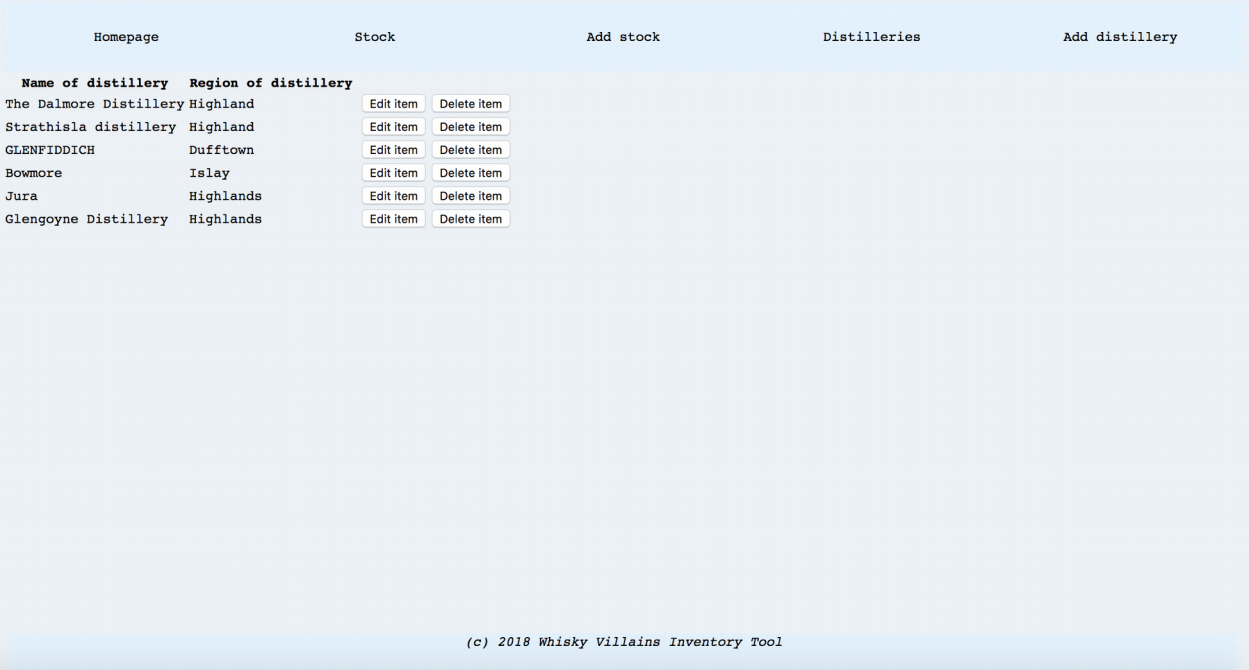
Show user input being processed according to design requirements. Take a screenshot of:

The user inputting something into your program





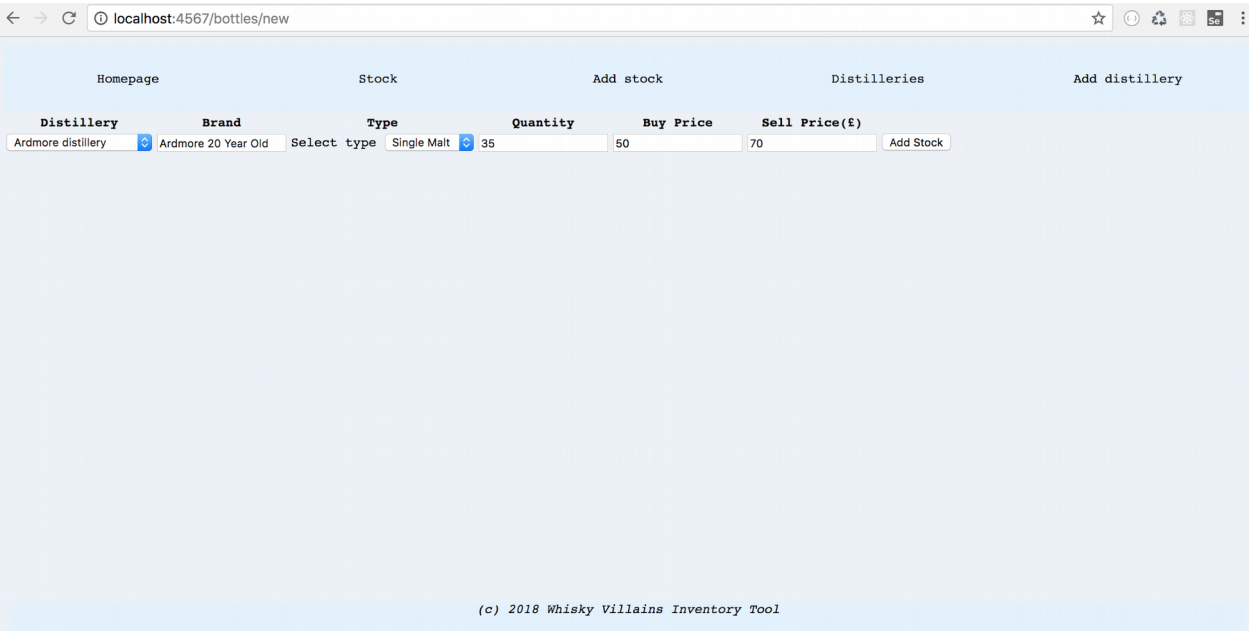
The user input being saved



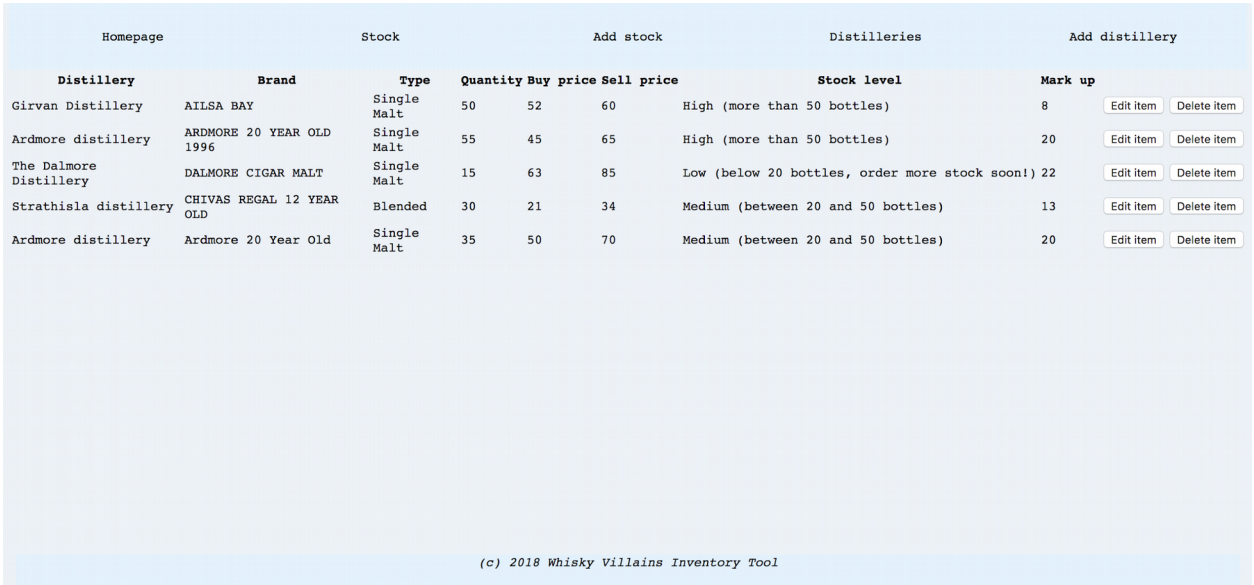
Glengoyne Distillery can be seen at the bottom of the list which corresponds to last item saved to database

Show an interaction with data persistence. Take a screenshot of:

Data being inputted into your program



Confirmation of the data being saved
In the browser



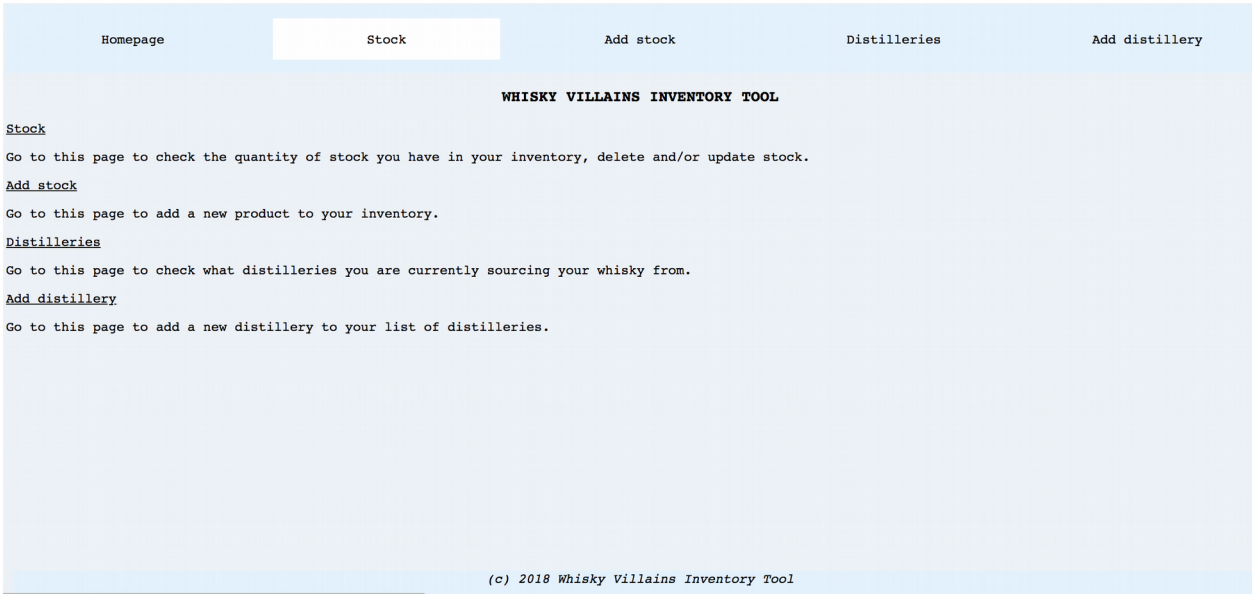
In the terminal

```
→  db git:(master) ✗ psql inventory
psql (10.0)
Type "help" for help.

inventory=# SELECT * FROM bottles;
 id |           name           |    type     | distillery_id | quantity | buy_price | sell_price
----+--------------------------+-------------+---------------+----------+-----------+------------
  1 | AILSA BAY                | Single Malt |             1 |       50 |        52 |         60
  2 | ARDMORE 20 YEAR OLD 1996 | Single Malt |             2 |       55 |        45 |         65
  3 | DALMORE CIGAR MALT       | Single Malt |             3 |       15 |        63 |         85
  4 | CHIVAS REGAL 12 YEAR OLD | Blended     |             4 |       30 |        21 |         34
  5 | Ardmore 20 Year Old      | Single Malt |             2 |       35 |        50 |         70
(5 rows)

inventory=#
```

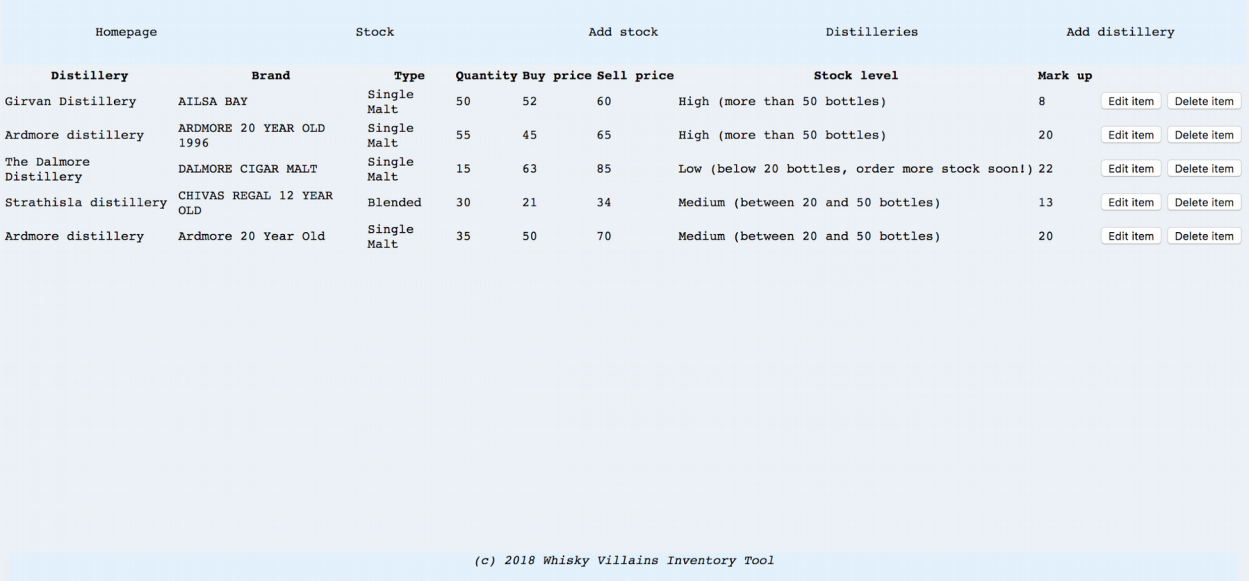
Show the correct output of results and feedback to user. Take a screenshot of:

The user requesting information or an action to be performed

User clicks on stock link in the navigation bar and expects to be taken to inventory page



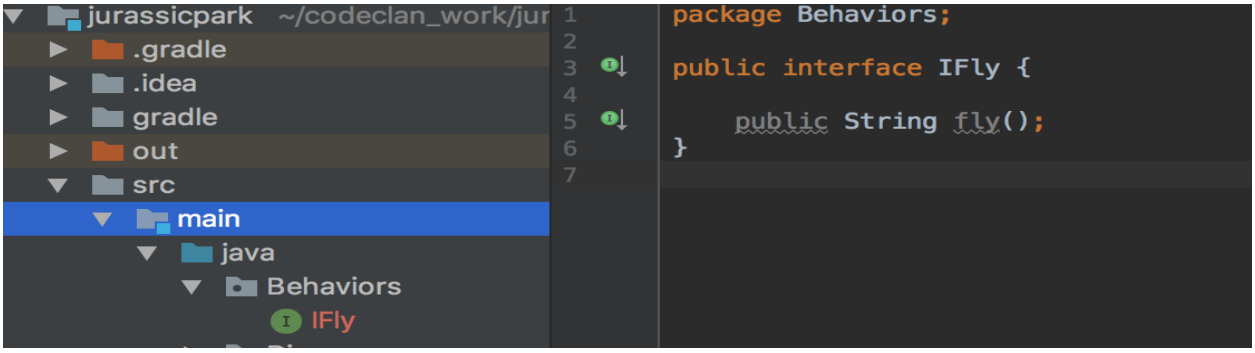
The user request being processed correctly and demonstrated in the program



Clicking on “stock” link in navigation bar takes the user to the inventory page displayed in screenshot above

WEEK_6

Demonstrate the use of Polymorphism in a program



Pterosaur(flying dinosaurs) abstract class implements IFly interface. As it is an abstract class all its subclasses implement the IFly interface too.

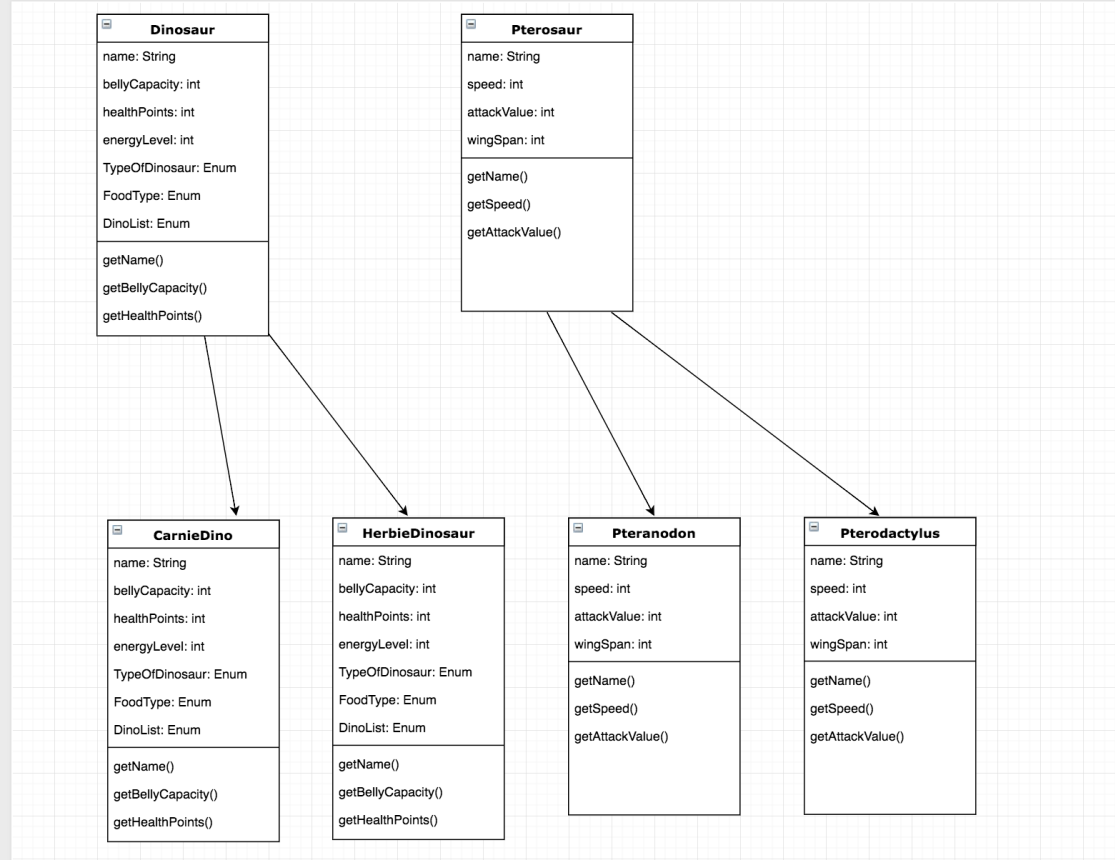
In the Park class, “IFly” objects are placed into an ArrayList called ‘predators’ to separate them from non flying dinosaurs who, in this scenario, are their prey.

```java
import Dinos.Dinosaur;
import Paddocks.Paddock;
import Paddocks.CarnivorePaddock;
import Paddocks.HerbivorePaddock;
import Behaviors.IFly;


import java.util.ArrayList;

public class  Park  {

    private int parkCapacity;
    private String name;
    private ArrayList<Paddock> paddocks;
    private ArrayList<Visitor> visitors;
    private HerbivorePaddock herbivorePaddock;
    private HerbivorePaddock herbivorePaddock2;
    private CarnivorePaddock carnivorePaddock;
    private Park park;
    private Visitor visitor;
    private ArrayList<IFly> predators;


    public Park(int parkCapacity) {
        this.parkCapacity = parkCapacity;
        this.name = "Ace Ventura";
        this.predators = new ArrayList<IFly>();
        this.paddocks = new ArrayList<Paddock>();
        this.visitors = new ArrayList<Visitor>();
        this.herbivorePaddock = new HerbivorePaddock( name: "HerbPaddock",  capacity: 10);
        this.carnivorePaddock = new CarnivorePaddock( name: "CarnPaddock",  capacity: 15);
    }

    public void addToPredatorWatchList(IFly predator) {
        this.predators.add(predator);
    }

    public void removeFromPredatorWatchList(IFly predator) {
        this.predators.remove(predator);
    }
```

addToPredatorWatchList() adds IFly objects to an ArrayList of predators.
removeFromPredatorWatchList() removes IFly objects from the ArrayList of predators.

WEEK_9

Below is an inheritance diagram showing two abstract classes(Dinosaur & Pterosaur) and their respective subclasses



Take an example of encapsulation in a program

```
public abstract class Pterosaur implements IMove {
    private String name;
    private int speed;
    private int attackValue;
    private int wingSpan;
    private String kind;

    public Pterosaur(String name, int speed, int attackValue, int wingSpan, String kind) {
        this.name = name;
        this.speed = speed;
        this.attackValue = attackValue;
        this.wingSpan = wingSpan;
        this.kind = kind;
    }

    public String move(){
        return "I can fly!";
    }

    public String getName() {
        System.out.println("I am " + this.name + ", a flying reptile of type " + this.kind + " and I am here to kick s
        return name;
    }

    public void setName(String name) { this.name = name; }

    public String getKind() { return kind; }

    public int getSpeed() { return speed; }

    public int getAttackValue() {
        System.out.println("I am packing heat and my attack value is " + attackValue + " health points that my enemies
        return attackValue;
    }
```

Take a screenshot of the use of Inheritance in a program. Take screenshots of:

Abstract class

```java
package Dinos;

import Behaviors.IMove;

import java.util.ArrayList;

public abstract class Dinosaur implements IMove {

    private String name;
    private TypeOfDinosaur type;
    private ArrayList<Food> belly;
    private int bellyCapacity;
    private int healthPoints;
    private int energyLevel;
    private Food food;
    private FoodType foodType;
    private DinoList dinoKind;

    public Dinosaur(String name, TypeOfDinosaur type, int bellyCapacity, int healthPoints, int energyLevel, DinoList dinoKind) {
        this.name = name;
        this.belly = new ArrayList<>();
        this.type = type;
        this.bellyCapacity = bellyCapacity;
        this.healthPoints = healthPoints;
        this.energyLevel = energyLevel;
        this.dinoKind = dinoKind;
    }

    public String move(){
        return "I am a fast walker but not much of a runner";
    }

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

    public void setName(String name) { this.name = name; }

    public int getHealthPoints() { return healthPoints; }

    public void setHealthPoints(int healthPoints) { this.healthPoints = healthPoints; }

    public int getEnergyLevel() { return this.energyLevel; }

    public void setEnergyLevel() { this.energyLevel = energyLevel; }

    public TypeOfDinosaur getType() { return this.type; }
```

Subclass that inherit from abstract class “Dinosaur”

```java
package Dinos;

public class HerbieDino extends Dinosaur {
    public HerbieDino(String name, TypeOfDinosaur type, int bellyCapacity, int healthPoints, int energyLevel, DinoList dinoKind) {
        super(name, type, bellyCapacity, healthPoints, energyLevel, dinoKind);
    }
}
```

An object in the inherited class

```java
package DinoTests;

import ...

public class TestHerbivoreDino {

    HerbieDino herbieDino;
    TypeOfDinosaur typeOfDinosaur;
    Food food;
    FoodType foodType;
    DinoList dinoKind;


    @Before
    public void before() {
        food = new Food(foodType.PLANTS);
        herbieDino = new HerbieDino( name: "Steve", typeOfDinosaur.Herbivore,  bellyCapacity: 10,  healthPoints: 15,  energyLevel: 10, dinoKind.Spinosaurus);
    }
```

A method that uses the information inherited from another class

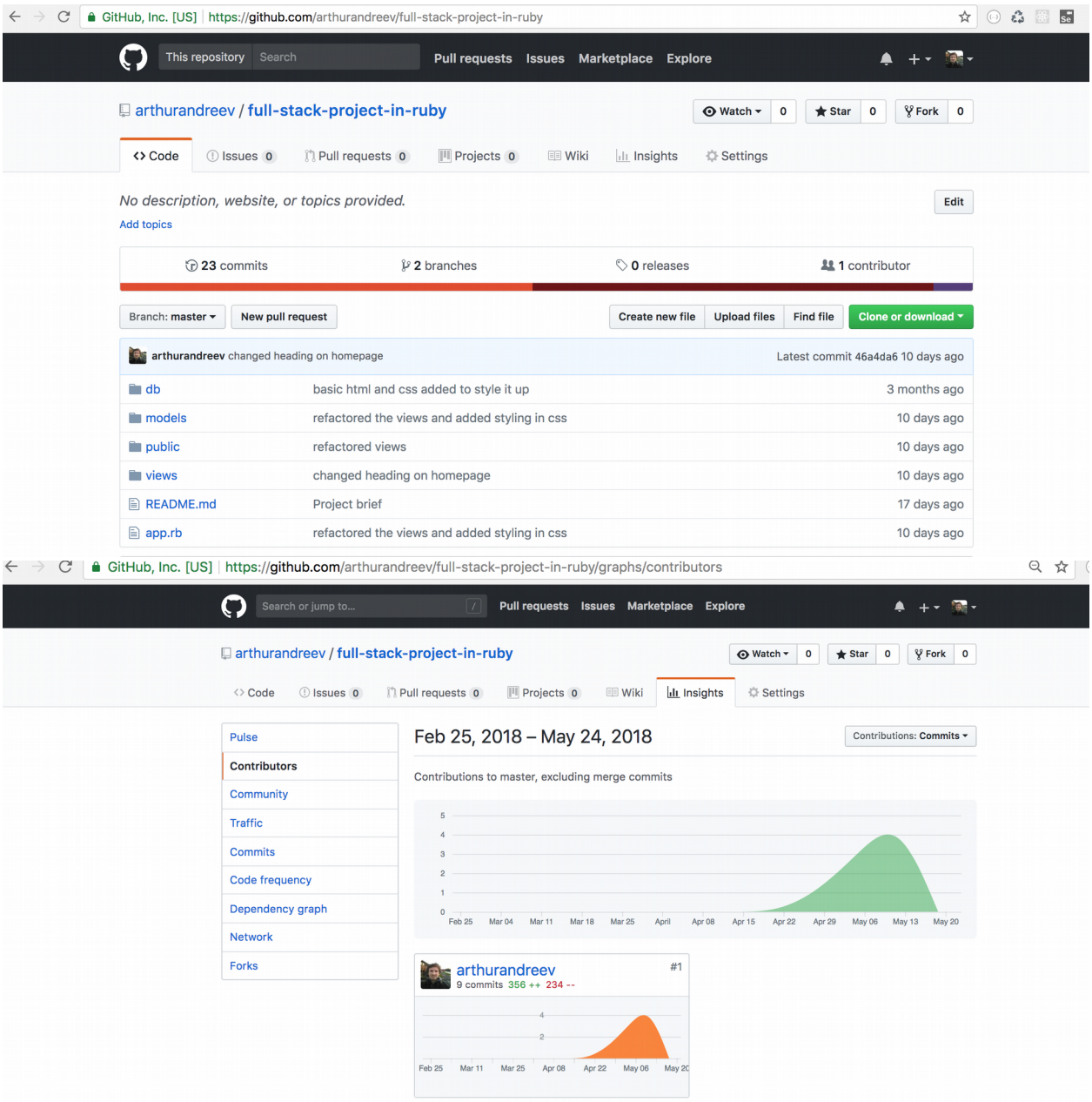
```java
public abstract class Dinosaur implements IMove {

    private String name;
    private TypeOfDinosaur type;
    private ArrayList<Food> belly;
    private int bellyCapacity;
    private int healthPoints;
    private int energyLevel;
    private Food food;
    private FoodType foodType;
    private DinoList dinoKind;

    public Dinosaur(String name, TypeOfDinosaur type, int bellyCapacity, int healthPoints, int energyLevel, DinoList dinoKind) {
        this.name = name;
        this.belly = new ArrayList<>();
        this.type = type;
        this.bellyCapacity = bellyCapacity;
        this.healthPoints = healthPoints;
        this.energyLevel = energyLevel;
        this.dinoKind = dinoKind;
    }

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

    public void setName(String name) { this.name = name; }
```

getName and setName are both methods on Dinosaur abstract class

```java
public class TestHerbivoreDino {

    HerbieDino herbieDino;
    TypeOfDinosaur typeOfDinosaur;
    Food food;
    FoodType foodType;
    DinoList dinoKind;


    @Before
    public void before() {
        food = new Food(foodType.PLANTS);
        herbieDino = new HerbieDino( name: "Steve", typeOfDinosaur.Herbivore,  bellyCapacity: 10,  healthPoints: 15,  energyLevel: 10, dinoKind.Spinosaurus);
    }

    @Test
    public void canGetName() { assertEquals( expected: "Steve", herbieDino.getName()); }

    @Test
    public void canSetName(){
        herbieDino.setName("Dave");
        assertEquals( expected: "Dave", herbieDino.getName());
    }
```

getName and setName are methods inherited from Dinosaur abstract class

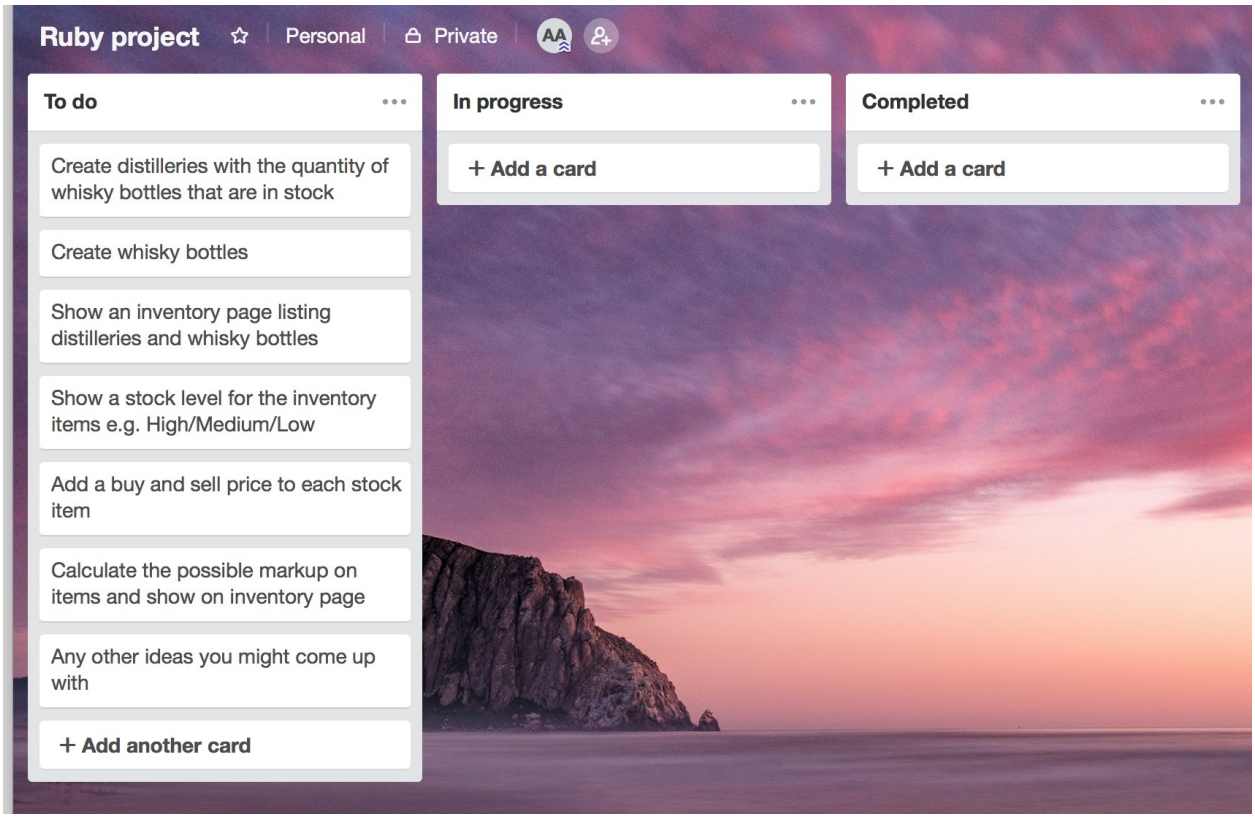
Take a screenshot of one of your projects where you worked alone and attach the Github link



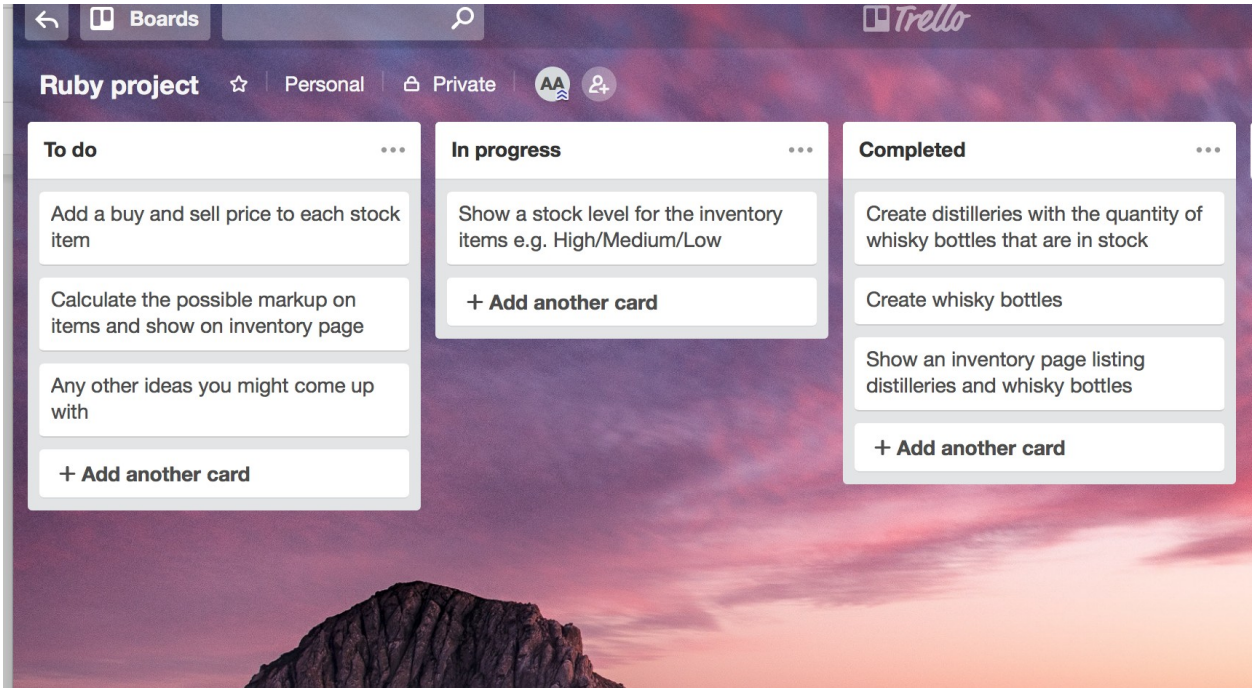
https://github.com/arthurandreev/full-stack-project-in-ruby

Take screenshots or photos of your planning and the different stages of development to show changes

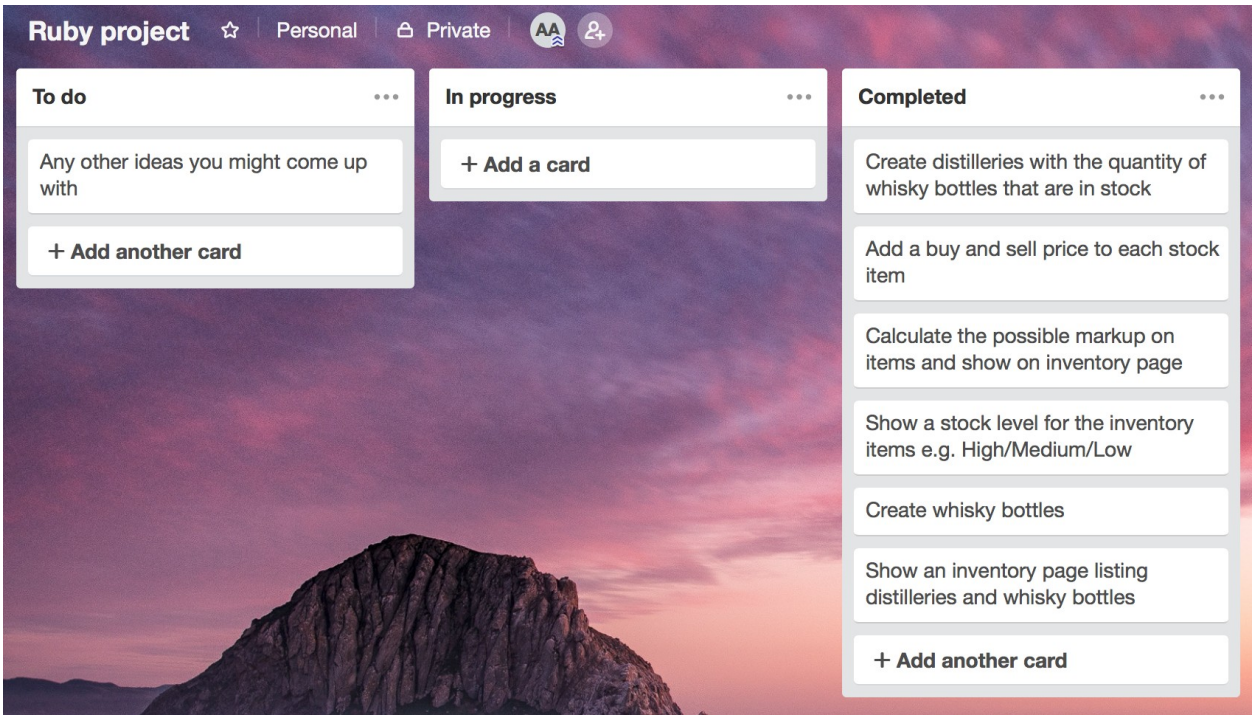
Start of project



Halfway point



Completed project



WEEK_11
Demonstrate testing in your program. Take screenshots of:

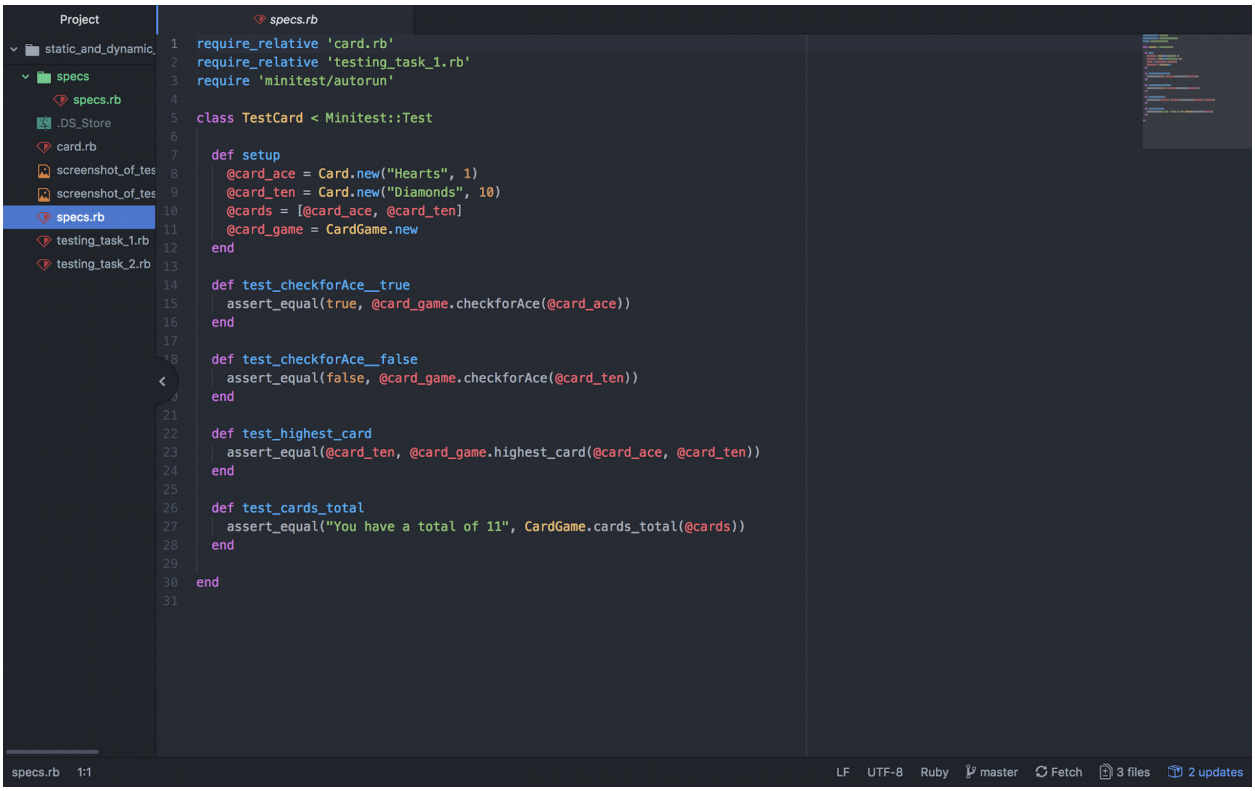
Example of test code

card.rb

```ruby
class Card
  attr_reader :suit, :value

def initialize(suit, value)
  @suit = suit
  @value = value;
end
end

```

```ruby
require_relative('card.rb')
class CardGame


  def checkforAce(card)
    if card.value = 1 #card.value should be followed by a double equal sign
      return true
    else
      return false
    end
  end

#dif should say def as def is the required keyword used to define a method
#card1 and card2 must be separated with a comma
  dif highest_card(card1 card2)
  if card1.value > card2.value
    return card.name #there is no variable called 'name' in this class, it should say 'card1' instead
  else
    card2
  end
end
end

def self.cards_total(cards)
  total #total must be set to 0 here as it will be set to null by default
  for card in cards
    total += card.value
    ##return statement should be outside of the loop after the first end keyword. There should be a space between "of" and
    return "You have a total of" + total
  end
end
#missing an end keyword here to end the class
```

Specs file



The test code failing to pass

```
→ static_and_dynamic_tasks git:(master) ✗ ruby specs.rb
specs.rb:2:in `require_relative': /Users/user/codeclan_work/my_pda_folder/static_and_dynamic_tasks/testing_task_1.rb:27: syntax error, unexpected keyword_end, expecting end-of-inp
ut (SyntaxError)
        from specs.rb:2:in `<main>'
→ static_and_dynamic_tasks git:(master) ✗
```

Example of the test code once errors have been corrected

```ruby
require_relative('card.rb')
class CardGame



  def checkforAce(card)
    if card.value == 1
      return true
    else
      return false
    end
  end

  def highest_card(card1, card2)
  if card1.value > card2.value
    return card1
  else
    card2
end
end

def self.cards_total(cards)
  total = 0
  for card in cards
    total += card.value
  end
  return "You have a total of #{total}"
end
end

```

The test code passing

```
→  static_and_dynamic_tasks git:(master) ✗ ruby specs.rb
Run options: --seed 62710

# Running:

....

Finished in 0.001419s, 2819.6416 runs/s, 2819.6416 assertions/s.

4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
→  static_and_dynamic_tasks git:(master) ✗
```

WEEK_12
Show an API being used within your program. Take a screenshot of:

The code that uses or implements the API

```
const apiBeerArray = [];

document.addEventListener('DOMContentLoaded', () => {
  const url = 'https://api.punkapi.com/v2/beers';
  makeRequest(url, requestComplete);

  const selectBeerDropDown = document.querySelector('#beer-selector');
  selectBeerDropDown.addEventListener('change', handleSelectBeer);
});

const makeRequest = function (url, callback) {
  const request = new XMLHttpRequest();
  request.open('GET', url);
  request.send();
  request.addEventListener('load', callback);
};

const requestComplete = function () {
  if (this.status !==200) return;
  const jsonString = this.responseText;
  const beers = JSON.parse(jsonString);
  beers.forEach((beer) => {
    apiBeerArray.push(beer);
  });
  populateBeerDropDown();
};
```

The API being used by the program whilst running

localhost:3000

BrewDog collection

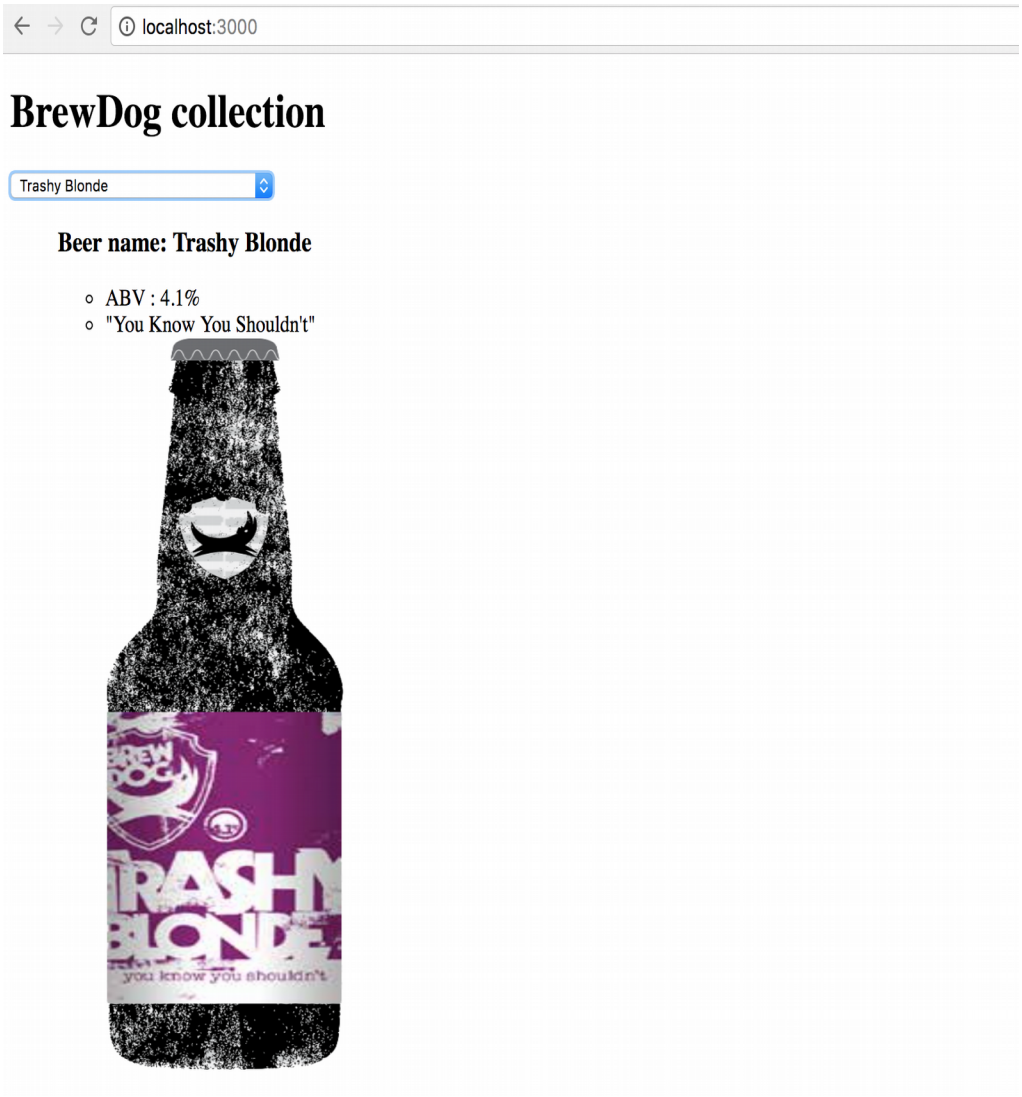
Select a beer to view:

BrewDog collection

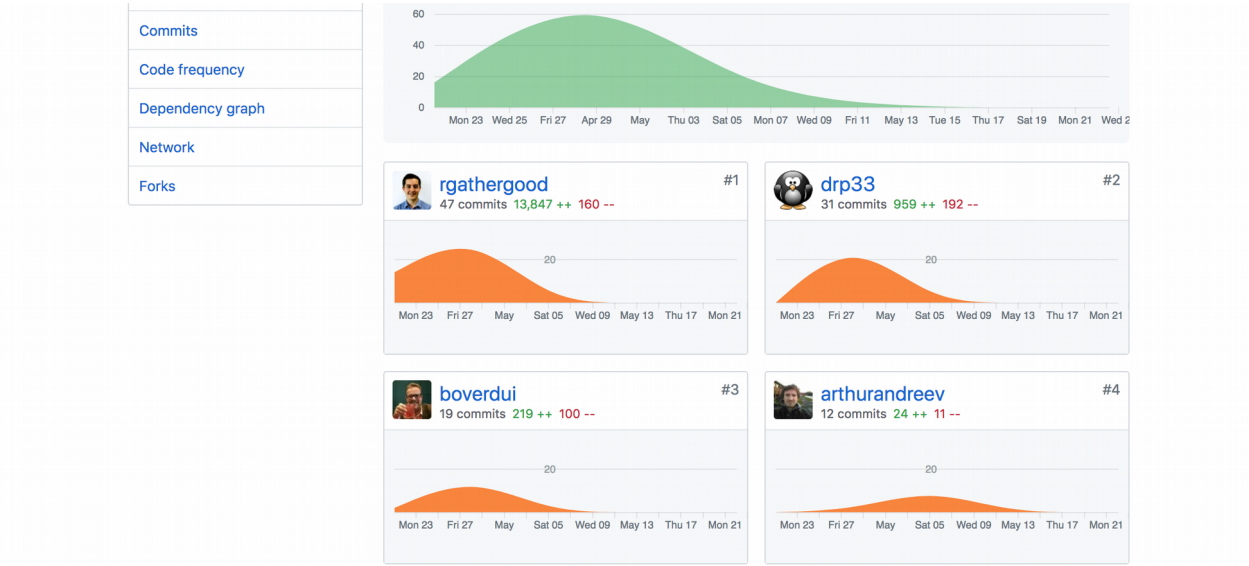
Devine Rebel (w/ Mikkeller)

Beer name: Devine Rebel (w/ Mikkeller)

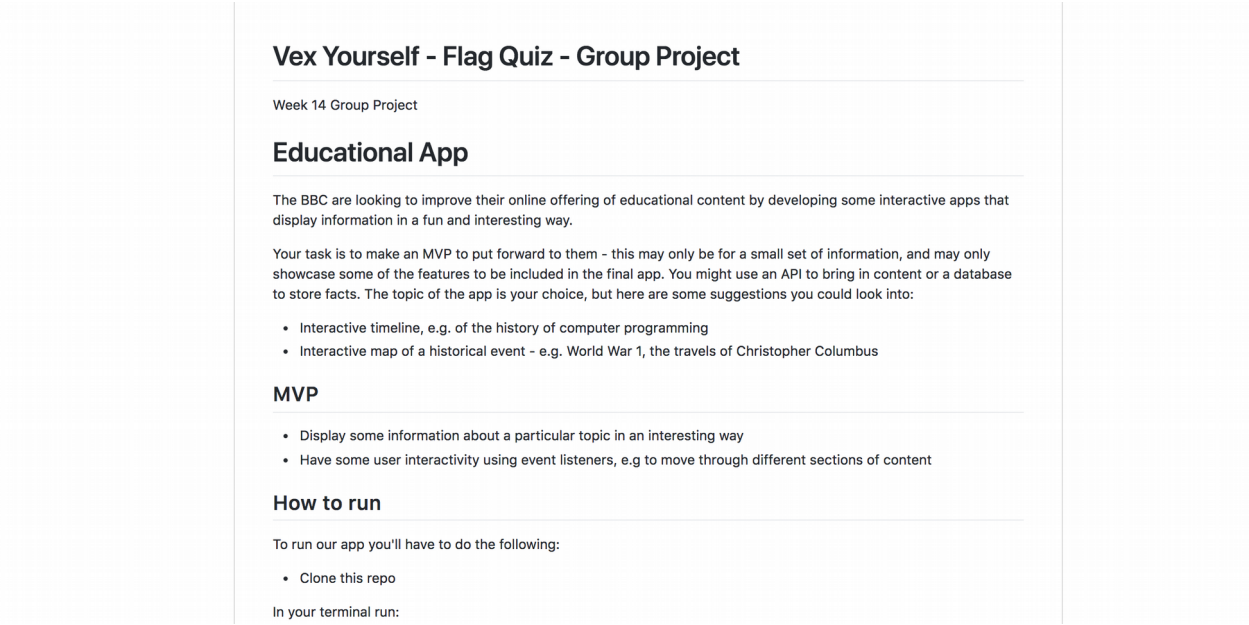
- ABV : 12.5%
- "Oak-aged Barley Wine."


localhost:3000
BrewDog collection
Trashy Blonde
Beer name: Trashy Blonde
ABV : 4.1%
"You Know You Shouldn't"
you know you shouldn't

Take a screenshot of contributor’s page on GitHub from your group project to show the team you worked with



Take a screenshot of the project brief from your group project



Vex Yourself - Flag Quiz - Group Project

Week 14 Group Project

Educational App

The BBC are looking to improve their online offering of educational content by developing some interactive apps that display information in a fun and interesting way.

Your task is to make an MVP to put forward to them - this may only be for a small set of information, and may only showcase some of the features to be included in the final app. You might use an API to bring in content or a database to store facts. The topic of the app is your choice, but here are some suggestions you could look into:

- Interactive timeline, e.g. of the history of computer programming
- Interactive map of a historical event - e.g. World War 1, the travels of Christopher Columbus

MVP

- Display some information about a particular topic in an interesting way
- Have some user interactivity using event listeners, e.g to move through different sections of content

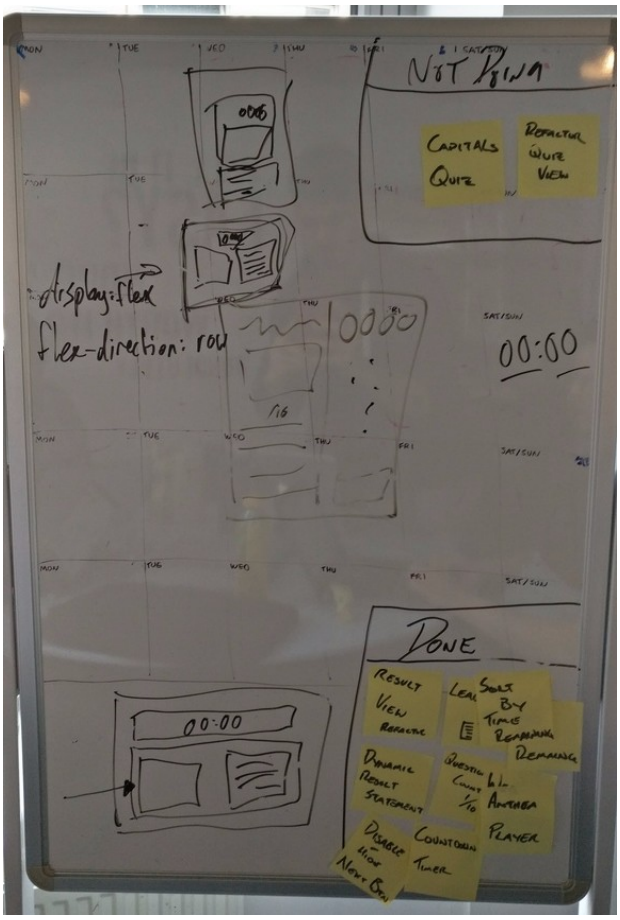
How to run

To run our app you'll have to do the following:

- Clone this repo

In your terminal run:

Provide a screenshot of the planning you completed during your group project



Write an acceptance criteria and test plan

Acceptance criteria	**Expected result/output**	**Pass/Fail**
A user is able to access all stock items in inventory	A full stock-list is displayed when link to 'Stock' is clicked	Pass
A user is able to edit a stock item in inventory	A new page is displayed which allows the user to edit the selected stock item when "Edit item" button is clicked	Pass
A user is able to delete a stock item in inventory	A selected item is deleted when the user clicks "Delete item" button	Pass
A user is able to add a new item to inventory	A new page is displayed which allows the user to add a new stock item when "Add stock" link is clicked	Pass
A user is able to add a new distillery to the list of distilleries	A new page is displayed which allows the user to add a new distillery when "Add distillery" link is clicked	Pass

Produce two system interaction diagrams(sequence and/or collaboration diagrams)

Diagram 1

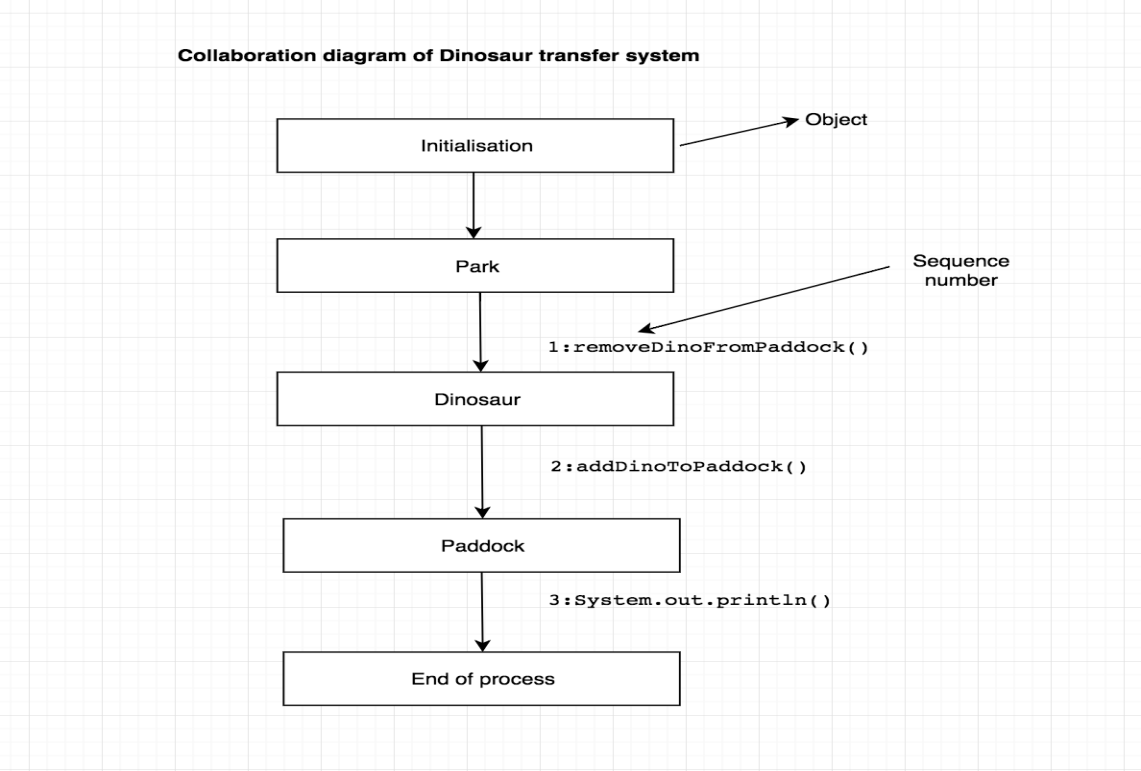
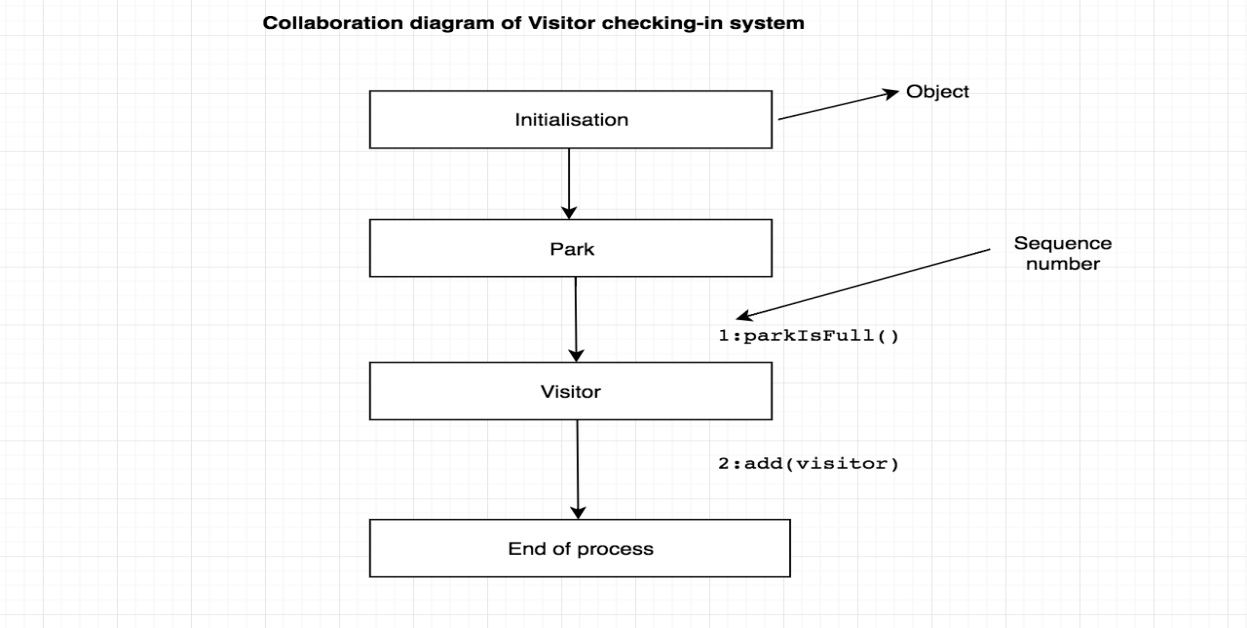
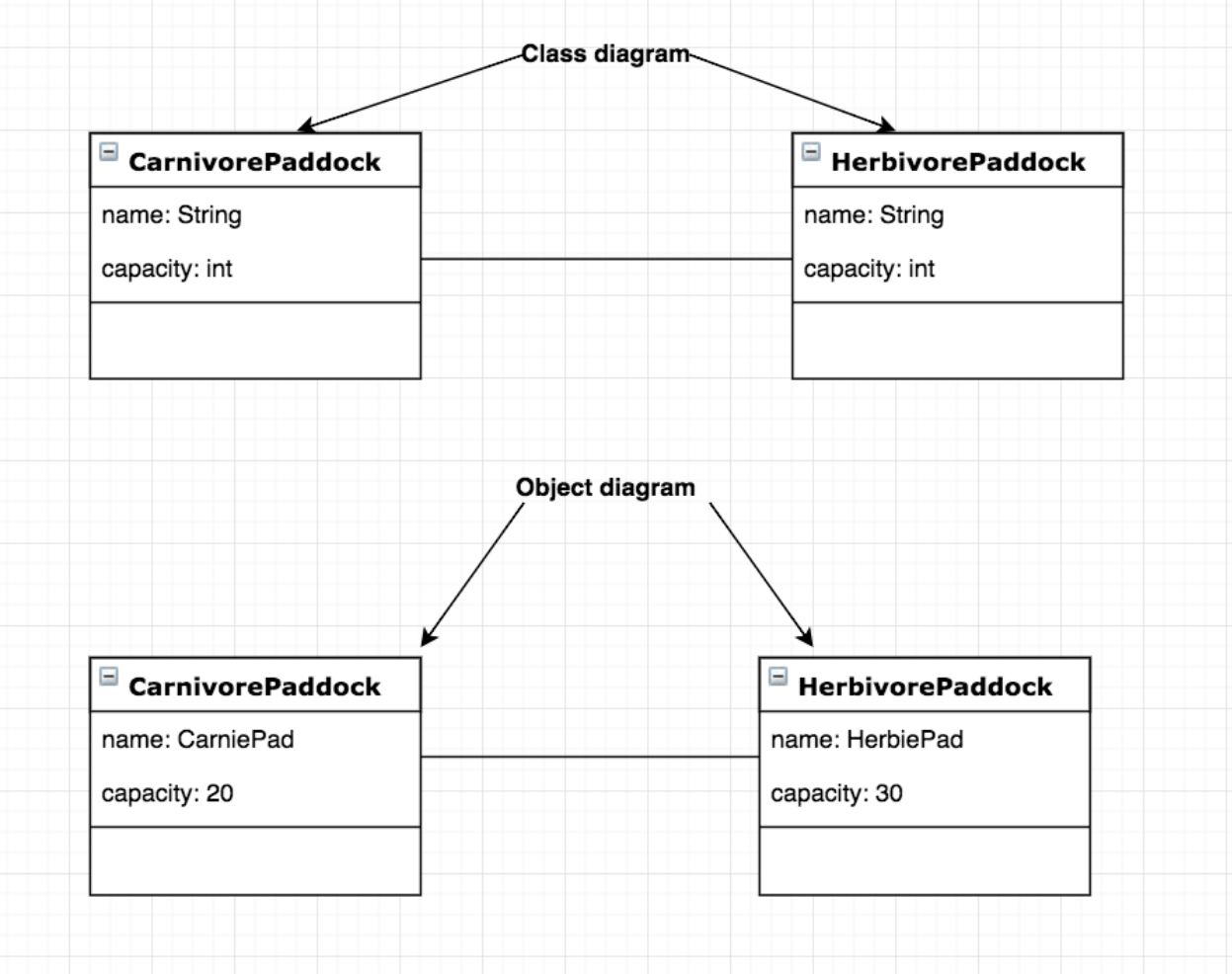


Diagram 2

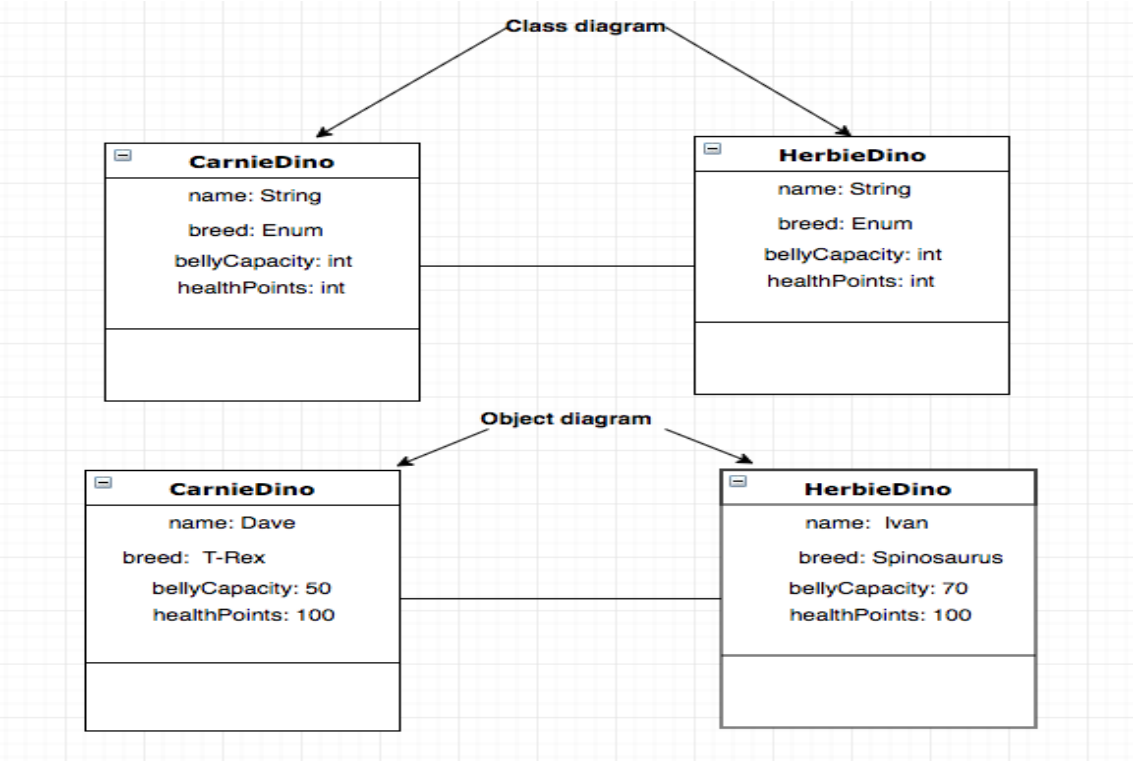


Produce two object diagrams

Object diagram 1



Object diagram 2



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

1) **addDinosaurToPaddock()**

```
    public void addDinosaurToPaddock(Dinosaur dinosaur) {
//        switch case statement that calls getType method on dinosaur object to access TypeOfDinosaur Enum
        switch (dinosaur.getType()) {
            case Herbivore:
                herbivorePaddock.addDinoToPaddock(dinosaur);

            case Carnivore:
                carnivorePaddock.addDinoToPaddock(dinosaur);
        }
    }
```

This method adds different types of dinosaurs (herbivore/carnivore) in separate paddocks as they can't reside in the same paddock. I chose this method because I needed to write an algorithm for checking what type of dinosaur has been passed in to my method and then add this dinosaur object to the correct paddock. I used a switch/case statement here which is passed in the getType() method that returns the type of dinosaur object passed in (herbivore or carnivore which is stored in an Enum). These are the only two types that exist in the Enum hence there are two corresponding cases for Herbivore and Carnivore. addDinoToPaddock() method adds the dinosaur object to the correct paddock depending on whether dinosaur is herbivore or carnivore.

2) **transferDinosaur()**

```
public void transferDinosaur(Dinosaur dinoToMove, Paddock paddockToMoveFrom, Paddock paddockToMoveTo){
    paddockToMoveFrom.removeDinoFromPaddock(dinoToMove);
    paddockToMoveTo.addDinoToPaddock(dinoToMove);
    System.out.println("Dino has been transferred to a new paddock!");
}
```

I chose this method because I needed to write an algorithm that would allow me to transfer dinosaurs between paddocks.

This method takes in three parameters: dinosaur, paddockToMoveFrom and paddockToMoveTo.
First step in this algorithm is calling removeDinoFromPaddock method on paddockToMoveFrom and passing it our dinosaur object. This will remove our dinosaur from paddock.
Second step is calling addDinoToPaddock method on paddockToMoveTo and passing it our dinosaur object. This will add our dinosaur to a new paddock.
Finally a message will be printed out to the console that will read "Dino has been transferred to a new paddock".

Produce a bug tracking report

Expected result	1st testing attempt		2nd testing attempt
The user must be able to start a new quiz by clicking on "start new quiz" button			Passed
The countdown timer must activate as soon as "play" button is clicked			Passed
The user must be able to enter their name which will be saved alongside his quiz score	Failed	An empty array that was created outside of newQuiz() function was initially set to 'const' which prevented it from being modified. Changed the empty array to 'let' which rectified this error and allowed the empty array to take in result values such as scores	Passed
The user must be able to see countdown timer in the following format- 00:00	Failed	Couldn't get the time left to display with ":" in between minutes and seconds as was required. Added split(":") method to timer.display which separated time strings with ":" that worked perfectly	Passed