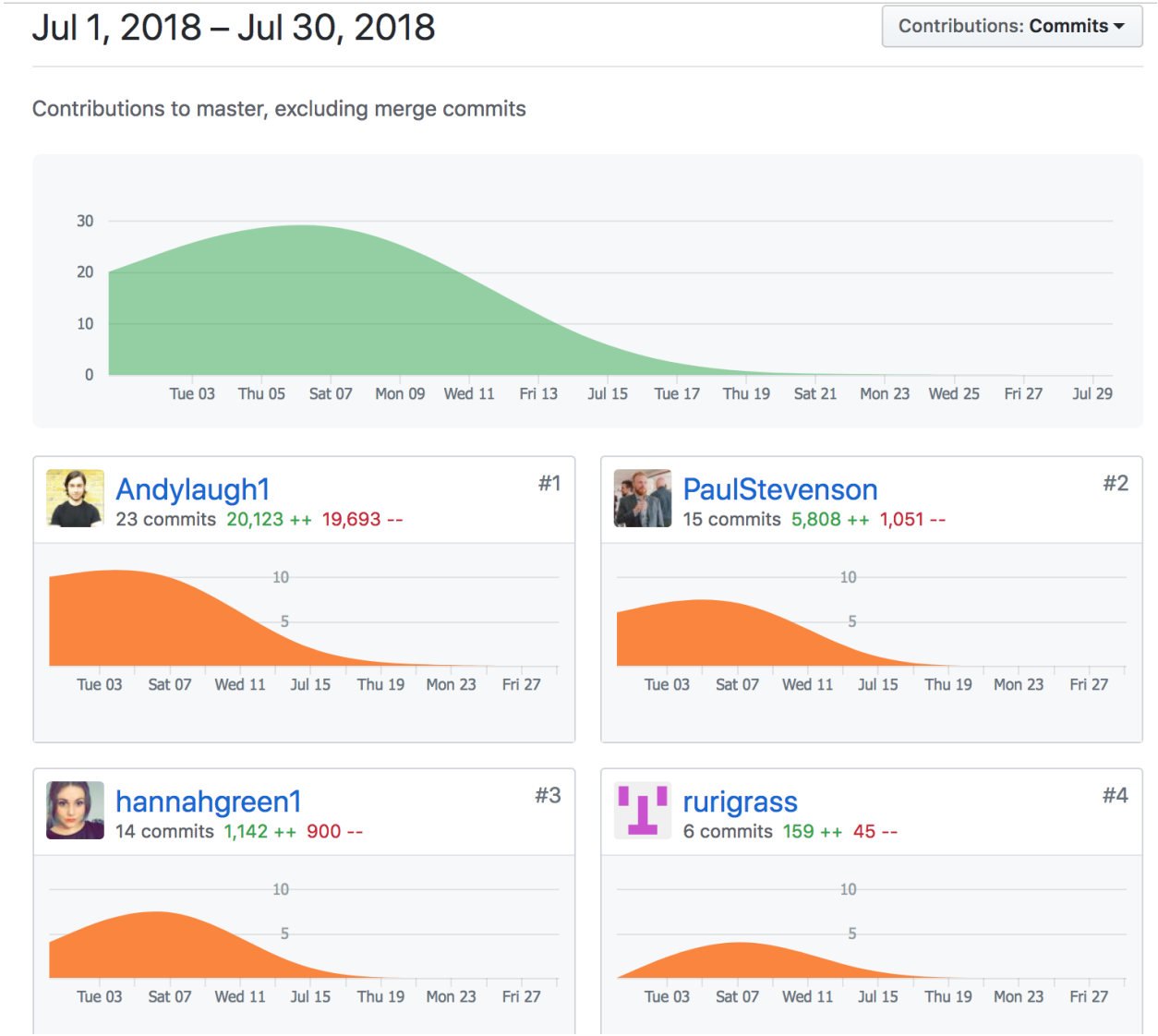
P 1.

Picture of contributors page on GitHub which shows the contributors on our team:



P 2.

Screenshot of project brief from our 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
- Explore the Solar System - navigate through planets and display information
- 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 interactivity that enables a user to move through different sections of content

Examples of further features

- Bring in data using an API or create your own
- Use charts or maps to display your information

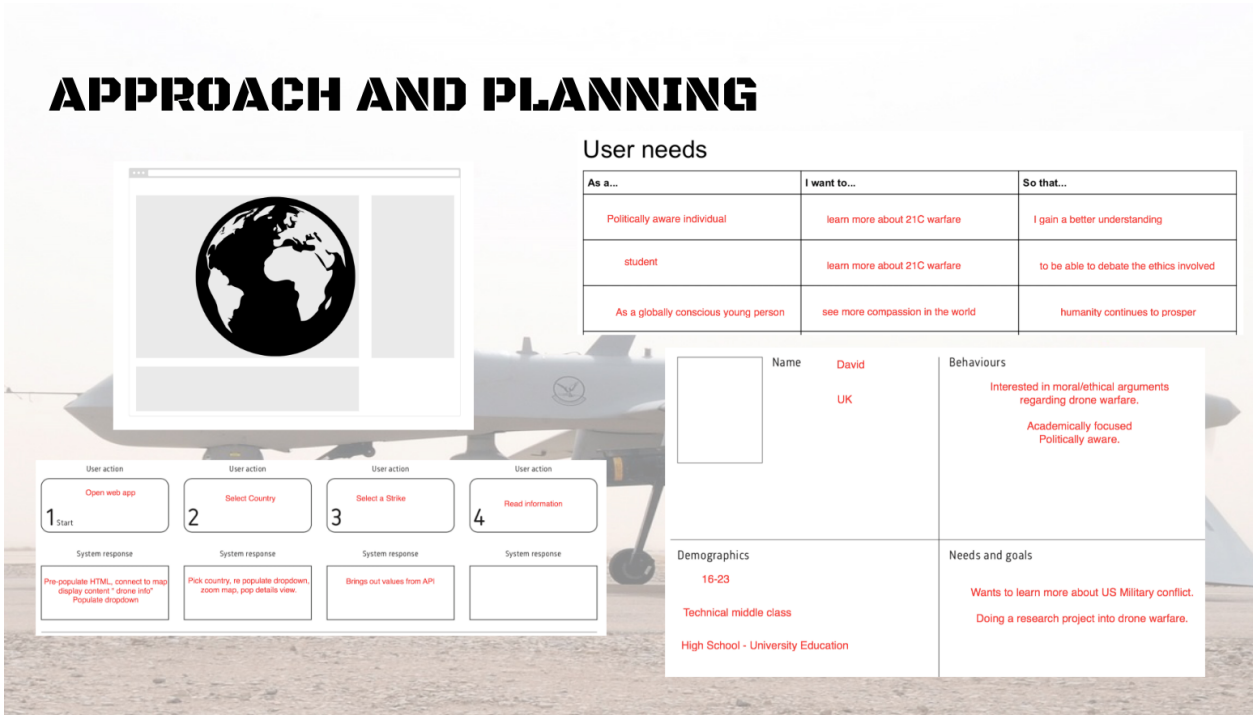
API, Libraries, Resources

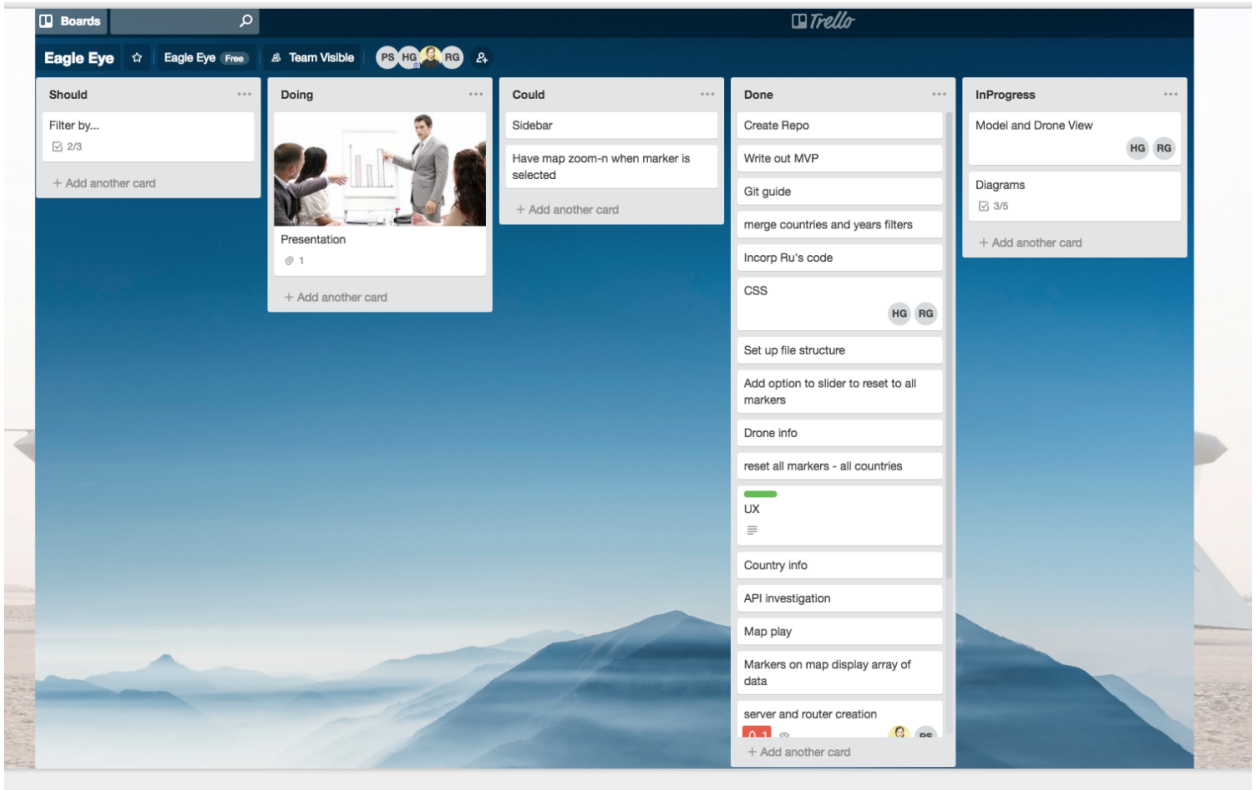
- https://www.highcharts.com/ HighCharts is an open-source library for rendering responsive charts.
- https://leafletjs.com/ Leaflet is an open-source library for rendering maps and map functionality.

```
Educational App- Project Brief
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.

MVP- Designed by team
Build an app to allow users to view information regarding covert drone strikes since 2002.
Users should be able to click on map and find out the drone data from that area.

* Build Full Stack JavaScript application
   * NodeJS
   * Webpack
   * HTML
   * CSS
* Plan project using:
   * UX
   * Trello
* Behaviour Driven Development
* Make use of API's
   * https://api.dronestre.am/data
* Have some interactivity that enables a user to move through different sections of content

```

P 3.

Screenshot of planning completed by group during project:



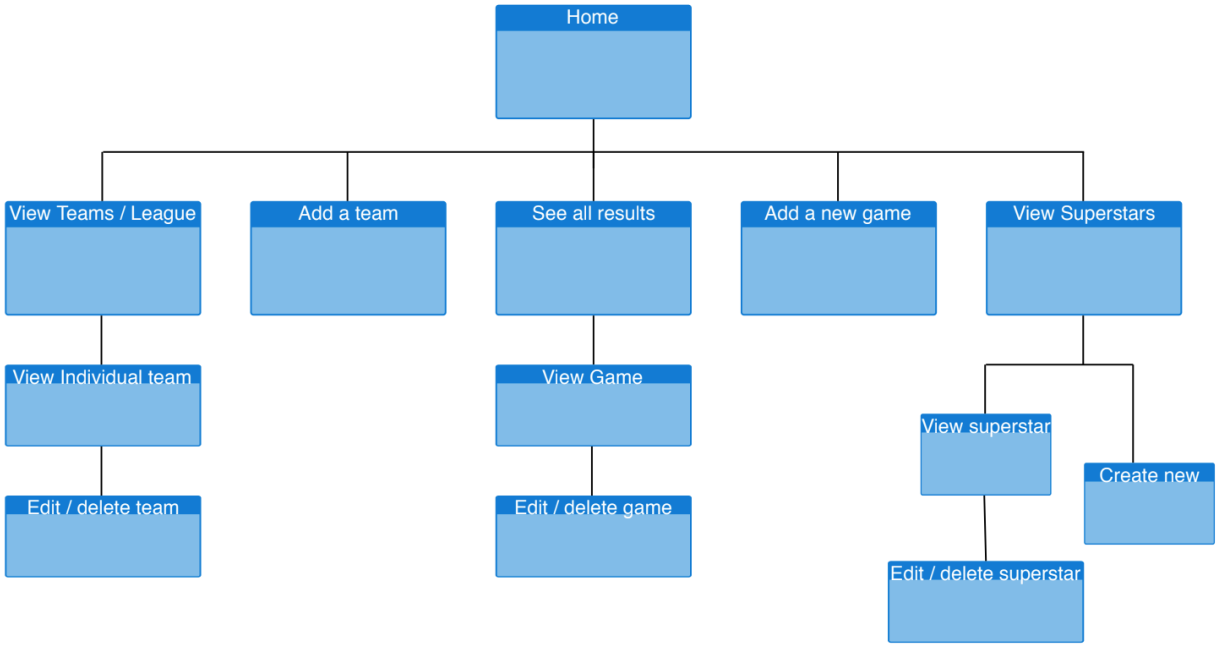


P 4.
Acceptance criteria and test plan:

As a user of the Collections Application, I want to be able to add or remove items to/from different collections, and I want to be able to manage the items in the collection.
I want to be able to mark items as sellable, donatable or swapable.
I need to be able to keep a record of the potential profit of all of my for sale items

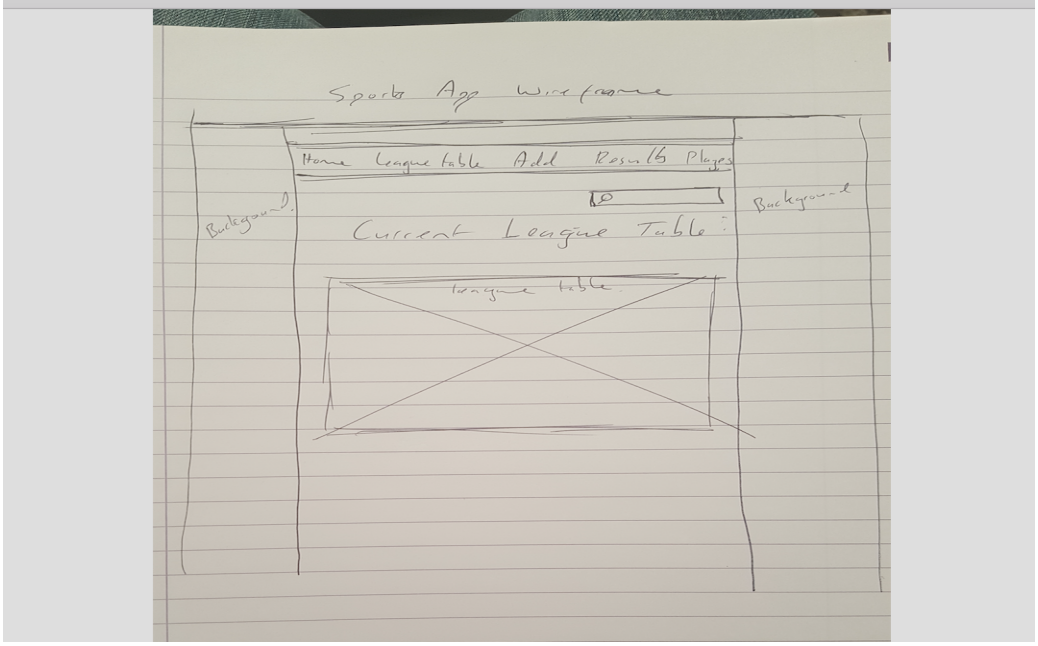
Number	Description	Steps	Expected Result	Actual Result
1	Ability to add a new item	Create a new 'item' and assign it a type	New item is created with all relevant properties: - name - type - purchase price - shipping price - current market value	Tests passing
2	Ability to mark items as sellable	Call method to alter 'for sale' status to 'true'	Item is now listed as sellable and getter test passes to show this is the case	Tests passing
3	Ability to add item to collection	Item is added to a collection of those items	Relevant collection now shows the item added. Test should pass which counts the items in the collection	Tests passing
4	Ability to calculate total cost of items in collection	Calculate the total cost (purchase + shipping) of each item individually and then calculate the total cost of all items in collection	Collection should return a total cost of all items put together	Tests passing
5	Ability to calculate potential profit of sellable items	Add all the total costs of all Sellable items in a collection, and subtract from the current market value for each. This provides the potential profit for each item, and these should all be added together	Collections manager returns the total potential profit of all sellable items	Tests passing

P 5.
Example of user sitemap



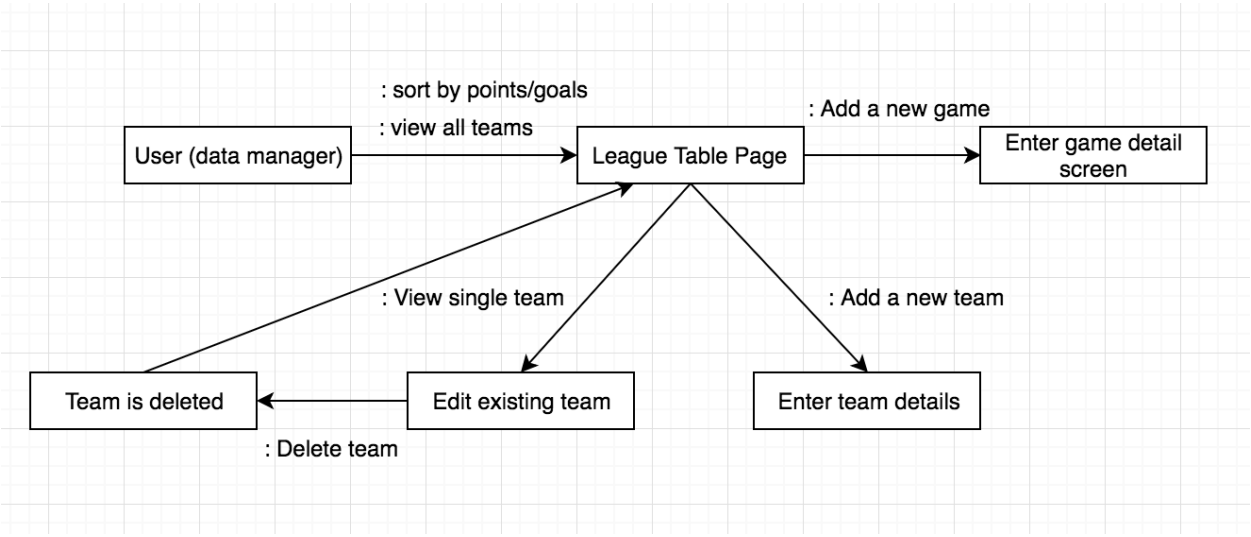
P 6.

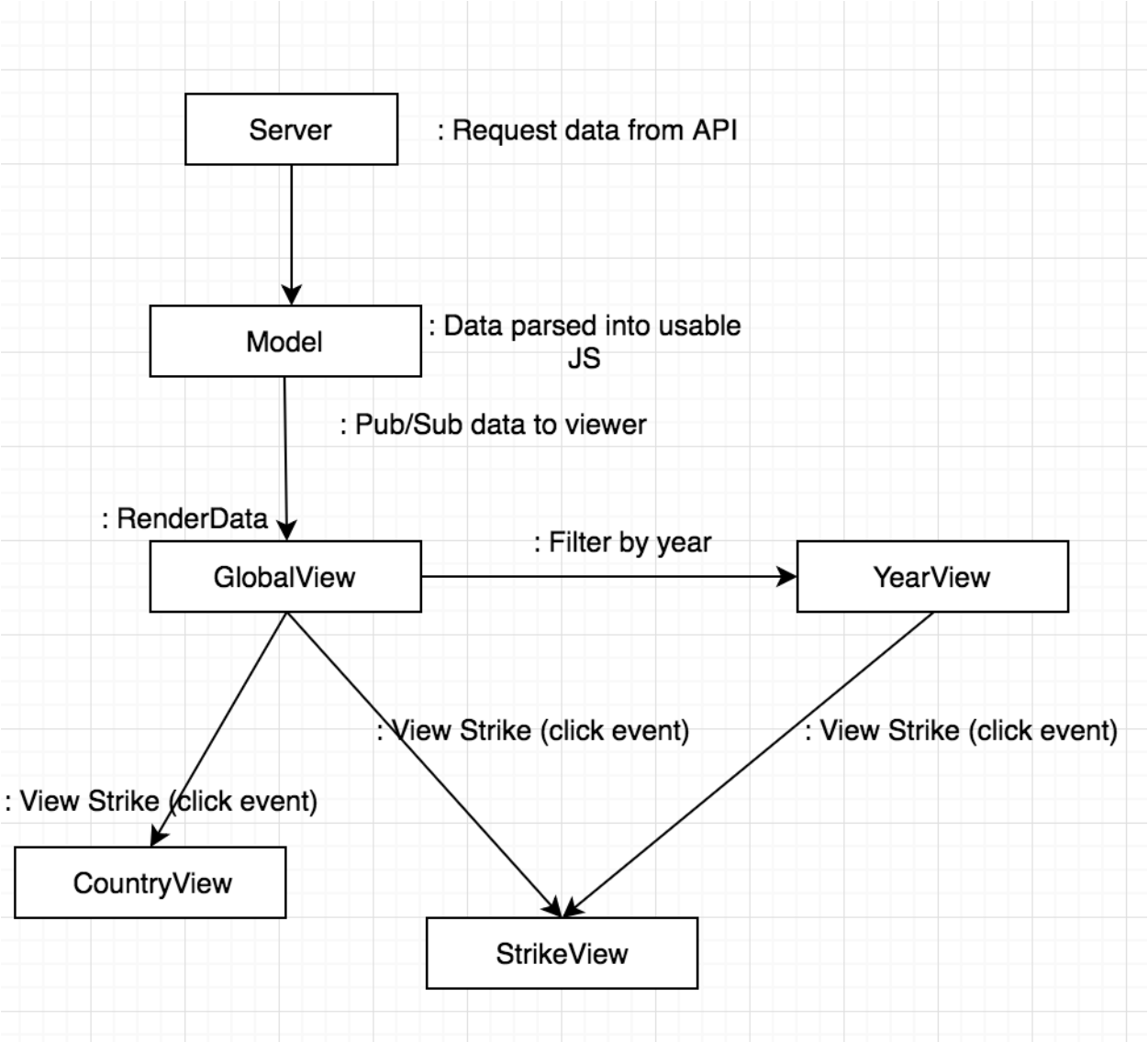
Two examples of wireframe designs:



P7

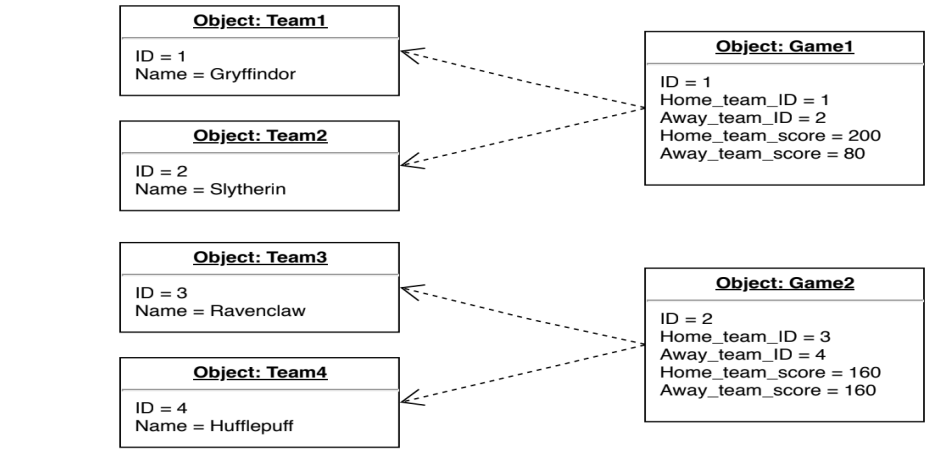
Two examples of system interaction diagrams:

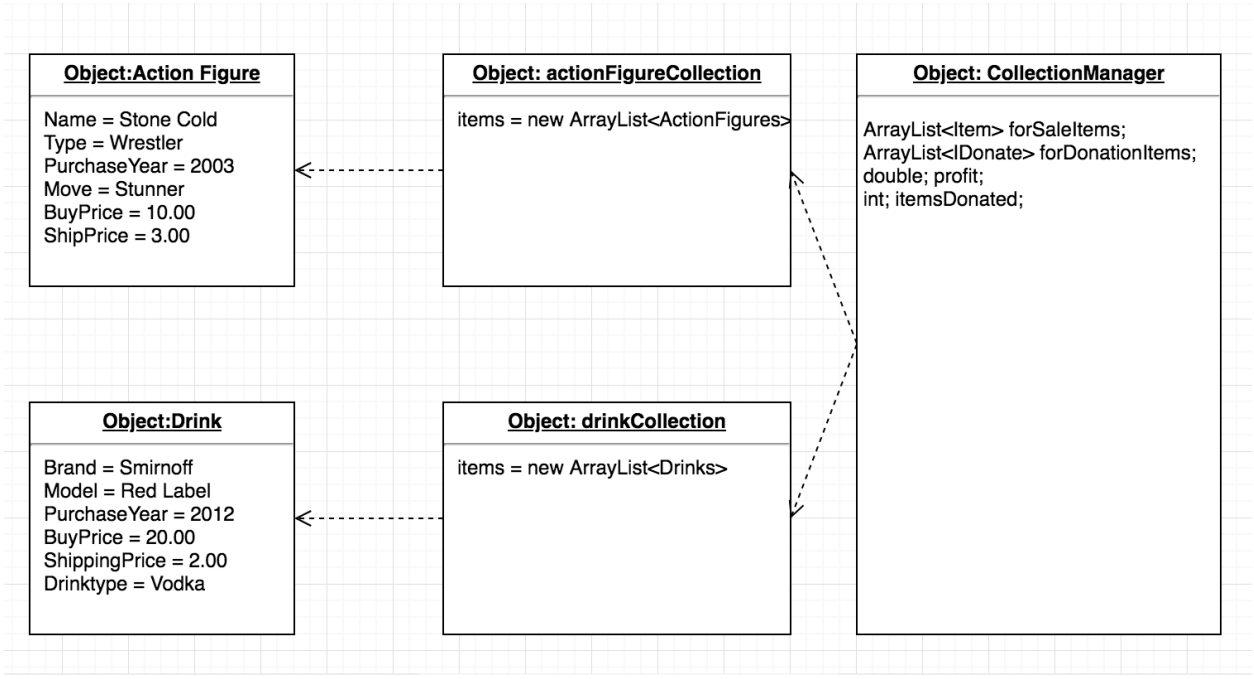

: sort by points/goals
: view all teams
: Add a new game
User (data manager)
League Table Page
Enter game detail screen
: View single team
: Add a new team
Team is deleted
Edit existing team
Enter team details
: Delete team



P8

Two examples of object diagrams:





P 9.

Example of two algorithms I have written and why I chose to use them:

Example 1:

```java
import java.util.Comparator;

public class SortByProfit implements Comparator<Item> {
    public int compare(Item a, Item b) {
        if ( a.calculateProfitIfSold() < b.calculateProfitIfSold() ) return 1;
        else if ( a.calculateProfitIfSold() == b.calculateProfitIfSold() ) return 0;
        else return -1;
    }

}
```

```java
    }

    public void sortByProfit(){
        Collections.sort(forSaleItems, new SortByProfit());
    }

    public void sortByPurchaseYear() {
```

First is an example of a comparator which I built to use on my Collections Manager class, and which sorted my forSale Items by the amount of profit they would make. The first screenshot shows the comparator being built as a separate class of it's own, and the second screenshot shows the comparator algorithm being called in a sort function inside the Collections Manager class.
I chose this algorithm as my user case showed that the user would wish to sort their items listed for sale by the potential profit they could earn - as this would help in deciding what to sell.

Example 2:

```
public double calculateTotalItemPricePaid() {
    double totalPricePaid = 0;
    for (Item item : items) {
        totalPricePaid += item.getBuyPrice();
    }
    return totalPricePaid;
}

public double calculateTotalShippingPricePaid() {
    double totalShippingPrice = 0;
    for (Item item : items) {
        totalShippingPrice += item.getShippingPrice();
    }
    return totalShippingPrice;
}

public double calculateTotalPricePaid() {
    double totalPricePaid = calculateTotalItemPricePaid() + calculateTotalShippingPricePaid();
    return totalPricePaid;
}
```

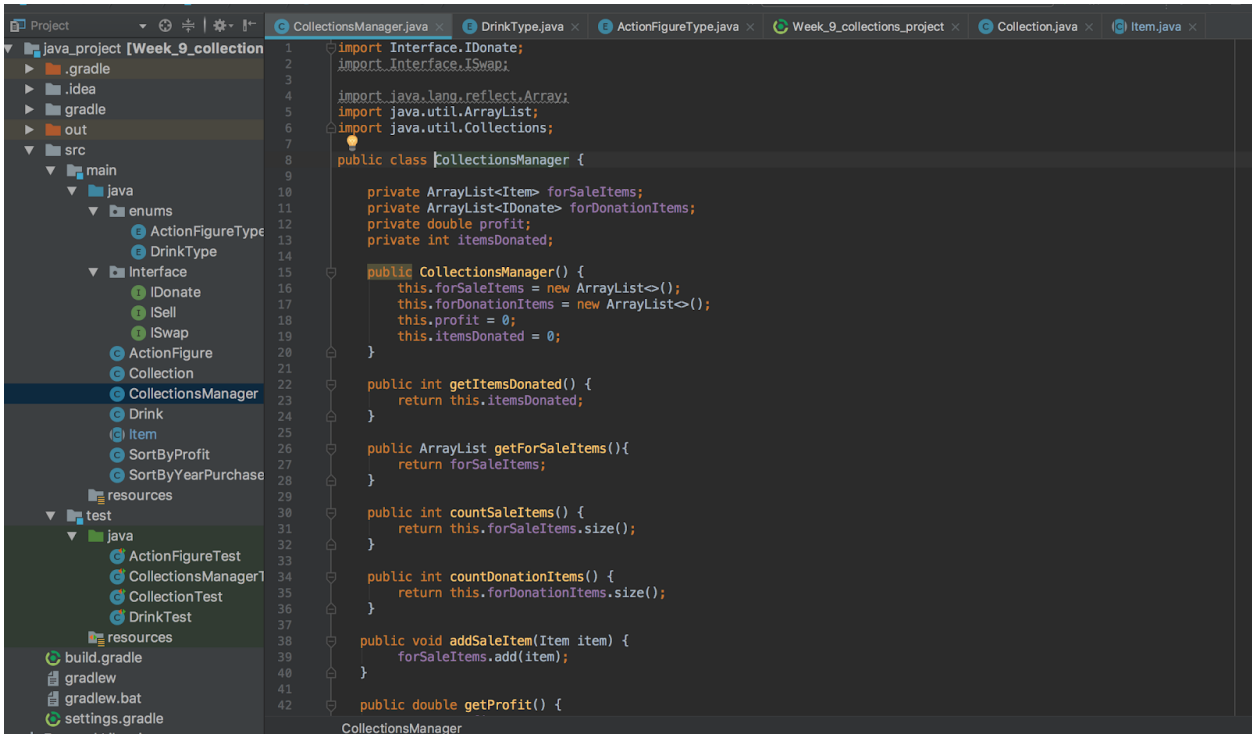
The second example shows three methods which are all related to each other. The first calculates the total price paid for all the items in a collection. The second algorithm calculates the total shipping cost for all the items in the collection. And the third method calculates the total price of the collection based on the totals from the first two algorithms.
I chose these algorithms as they were necessary for the user to manage the cost and eventually the profit of their collection (after selling some items)

P 10.

Example of pseudocode used in method to sort teams in league by points (highest to lowest) and then by goal difference (highest to lowest)

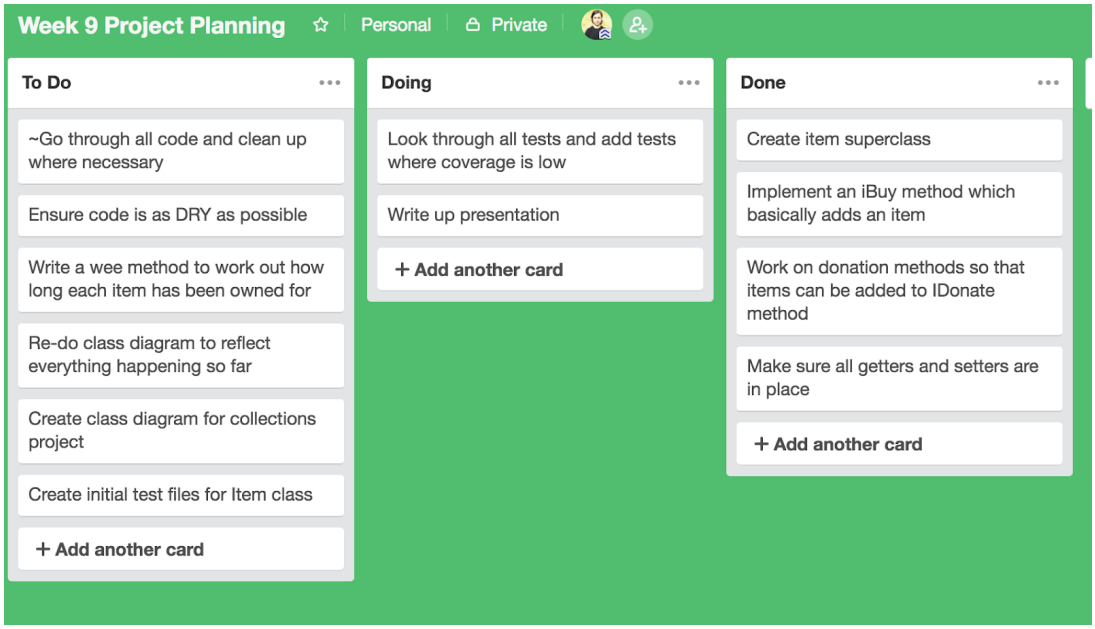
```
Pseudocode for method which sorts league by points difference and then by goal-difference

Getting the total points for a team:
total_points = 0
def add total points for team
  for game in games_won_by_team, add 3 points (points for win),
    and add all the points together and add to total_points
  for game in games_drawn_by_team add 1 point (points for draw),
    and add all the points together and add to total_points
  end

  return total_points

Getting the total goals for team:
work out total goals scored while home team
work out the total goals scored while away team
  add these two values together and place in a variable(1)
work out the total goals conceded while home team
work out the total goals conceded while away team
  add these two values together and place in a variable(2)
Subtract variable 2 from variable 1 and return the result as total_goal_difference


  Sort_by method which sorts league table by points and goal difference:
  having obtained methods to work out total points and total goal difference for each team,
  we need a method to sort the league table first by points(highest to lowest), then by goal difference.

  First find Teams.all
  assign Team.all to a variable 'teams'
  For each team in teams, sort_by total points first, and by goal difference second.
  return teams, reversed so that the team with most points/highest goal difference is first
```

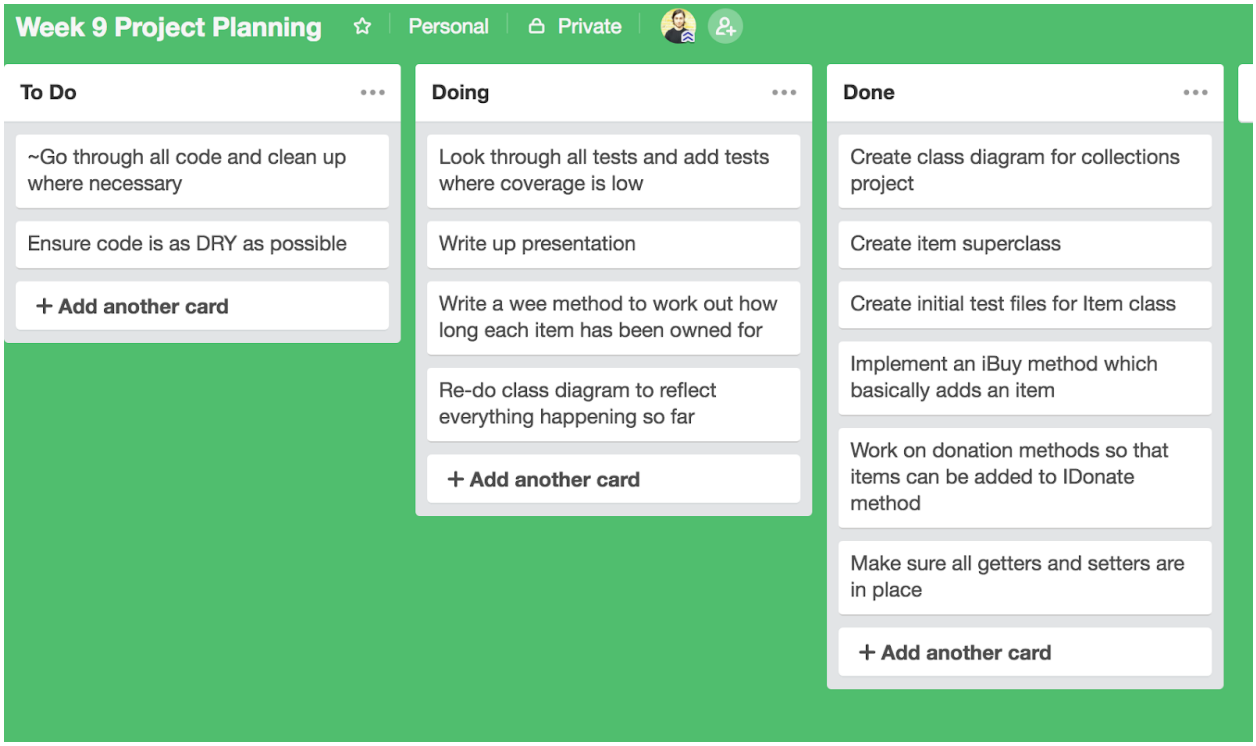
P.11. An example screenshot from a project on which I have worked alone, and the Github link:

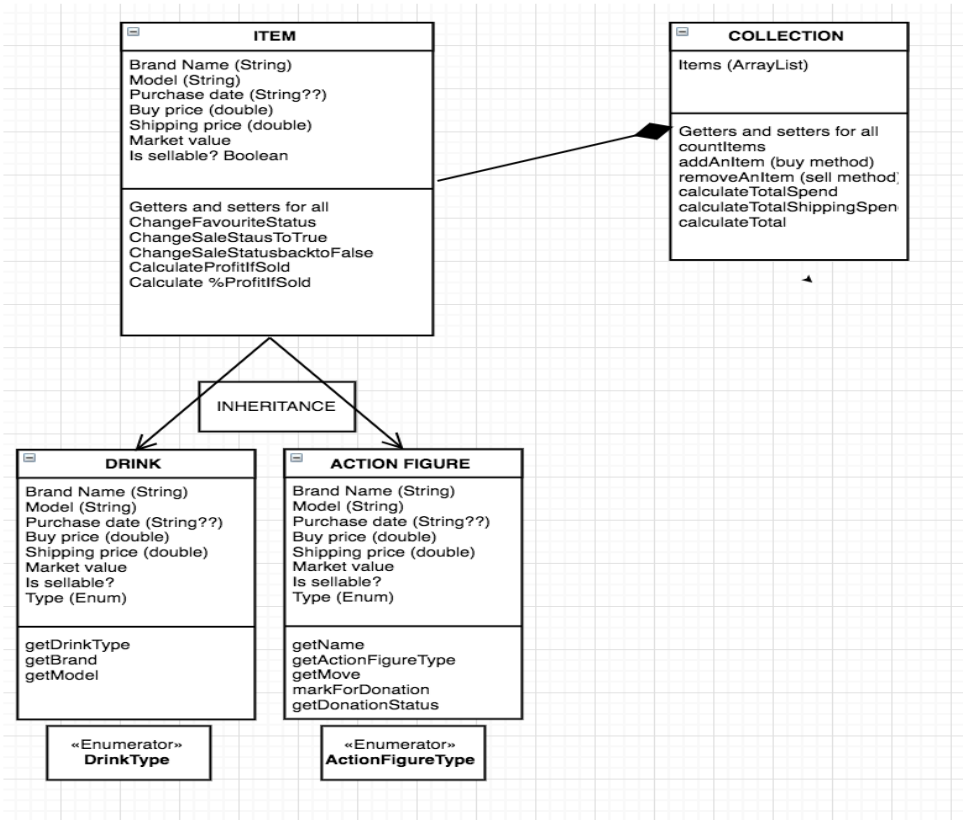


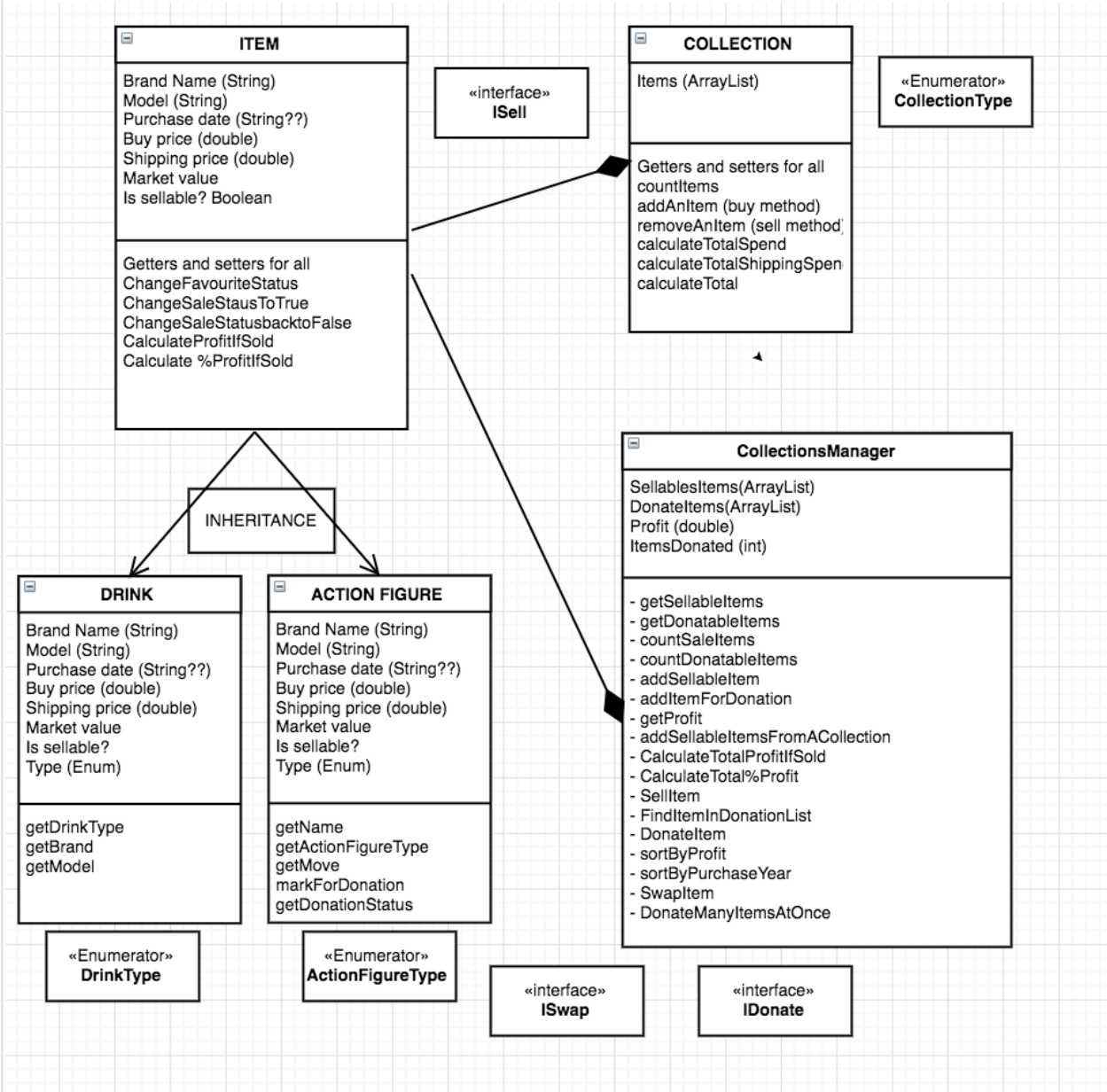
https://github.com/Andylaugh1/week9_java_collections_project

P.12. Screenshots and photos of the project planning process showing changes and updates


Week 9 Project Planning
Personal
Private
To Do
~Go through all code and clean up where necessary
Ensure code is as DRY as possible
Write a wee method to work out how long each item has been owned for
Re-do class diagram to reflect everything happening so far
Create class diagram for collections project
Create initial test files for Item class
+ Add another card
Doing
Look through all tests and add tests where coverage is low
Write up presentation
+ Add another card
Done
Create item superclass
Implement an iBuy method which basically adds an item
Work on donation methods so that items can be added to IDonate method
Make sure all getters and setters are in place
+ Add another card


Week 9 Project Planning
Personal
Private
To Do
~Go through all code and clean up where necessary
Ensure code is as DRY as possible
+ Add another card
Doing
Look through all tests and add tests where coverage is low
Write up presentation
Write a wee method to work out how long each item has been owned for
Re-do class diagram to reflect everything happening so far
+ Add another card
Done
Create class diagram for collections project
Create item superclass
Create initial test files for Item class
Implement an iBuy method which basically adds an item
Work on donation methods so that items can be added to IDonate method
Make sure all getters and setters are in place
+ Add another card


ITEM
Brand Name (String)
Model (String)
Purchase date (String??)
Buy price (double)
Shipping price (double)
Market value
Is sellable? Boolean
Getters and setters for all
ChangeFavouriteStatus
ChangeSaleStausToTrue
ChangeSaleStatusbacktoFalse
CalculateProfitIfSold
Calculate %ProfitIfSold
COLLECTION
Items (ArrayList)
Getters and setters for all
countItems
addAnItem (buy method)
removeAnItem (sell method)
calculateTotalSpend
calculateTotalShippingSpen
calculateTotal
INHERITANCE
DRINK
Brand Name (String)
Model (String)
Purchase date (String??)
Buy price (double)
Shipping price (double)
Market value
Is sellable?
Type (Enum)
getDrinkType
getBrand
getModel
«Enumerator»
DrinkType
ACTION FIGURE
Brand Name (String)
Model (String)
Purchase date (String??)
Buy price (double)
Shipping price (double)
Market value
Is sellable?
Type (Enum)
getName
getActionFigureType
getMove
markForDonation
getDonationStatus
«Enumerator»
ActionFigureType



P.13. Example of user input being saved or used

Screenshot 1: player clicks to add a new game

Home | View League Table | Add a New Team | See All Results | Add a New Game | View Superstars

Current League Table

Name	Played	Games Won	Games Drawn	Games Lost	Goal Difference	Poi
Gryffindor	2	2	0	0	410	6
ieston United	1	1	0	0	170	3
Ravenclaw	2	1	0	1	170	3
Slytherin	2	1	0	1	-160	3
eal Sosobad	1	0	1	0	0	1
idley Cannons	1	0	1	0	0	1
mont Dynamos	1	0	0	1	-170	0
Hufflepuff	2	0	0	2	-420	0

Screenshot 2: user enters new game details:

Add the Most Recent Results

Select Home Team: Murieston United

Select Away Team: Slytherin

Home Team Score: 200

Away Team Score: 150

Add Result to Championship

Screenshot 3: User can see the new game which has been added, under the see all results tab:

Home: Chudley Cannons 200

Away: Real Sosobad 200

The game was a draw

Show Game

Home: Murieston United 310

Away: Dechmont Dynamos 140

Murieston United won

Show Game

Home: Murieston United 200

Away: Slytherin 150

Murieston United won

Show Game

Screenshot 4: Updated league table following the processing of the new result:

Current League Table

Pos	Name	Played	Games Won	Games Drawn	Games Lost	Goal Difference	Points
1	Gryffindor	2	2	0	0	410	6
2	Murieston United	2	2	0	0	220	6
3	Ravenclaw	2	1	0	1	170	3
4	Slytherin	3	1	0	2	-210	3
5	Real Sosobad	1	0	1	0	0	1
6	Chudley Cannons	1	0	1	0	0	1
7	Dechmont Dynamos	1	0	0	1	-170	0
8	Hufflepuff	2	0	0	2	-420	0

P.14: Interaction with database persistence:

Picture 1: Seed file which adds new teams to the database and saves them:

```ruby
require('pry')
require_relative('../models/game.rb')
require_relative('../models/team.rb')
require_relative('../models/player.rb')

Game.delete_all
Player.delete_all
Team.delete_all

team1 = Team.new ({
  "name" => "Gryffindor",
  "transfer_funds" => 200000
  })
team1.save

team2 = Team.new ({
  "name" => "Slytherin",
  "transfer_funds" => 400000
  })
team2.save

team3 = Team.new ({
  "name" => "Hufflepuff",
  "transfer_funds" => 150000
  })
team3.save

team4 = Team.new ({
  "name" => "Ravenclaw",
  "transfer_funds" => 250000
  })
team4.save
```

Screenshot 2: Showing Team class file, initialised, requiring SQL runner and showing the .save function.

```ruby
require_relative('../db/sql_runner.rb')

class Team

  attr_reader :id
  attr_accessor :name, :transfer_funds

  def initialize(options)
    @id = options['id'].to_i
    @name = options['name']
    @transfer_funds = options['transfer_funds'].to_i
  end

  def save()
    sql = "INSERT INTO teams (name, transfer_funds) VALUES ($1, $2) RETURNING *"
    values = [@name, @transfer_funds]
    team_data = SqlRunner.run(sql, values)
    @id = team_data.first()['id'].to_i
  end
```

Screenshot 3: The seeds file being run to populate the database with the team data:

```
➜  sports_app_project git:(master) ✗ ruby db/seeds.rb

     /Users/user/codeclan_work/week_05/sports_app_project/db/seeds.rb @ line 182 :

    177:   "team_id" => team2.id,
    178:   "transfer_value" => 1000
    179:   })
    180: player10.save
    181:
 => 182: binding.pry
    183: nil

[1] pry(main)>
```

Screenshot 4: Database run showing all of the saved team data:

```
sports_league=# \q
➜  sports_app_project git:(master) ✗ psql -d sports_league -f db/sports_league.sql
DROP TABLE
DROP TABLE
DROP TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
➜  sports_app_project git:(master) ✗ psql -d sports_league
psql (10.2)
Type "help" for help.

sports_league=# SELECT * FROM teams;
 id |       name        | transfer_funds
----+-------------------+----------------
  1 | Gryffindor        |         200000
  2 | Slytherin         |         400000
  3 | Hufflepuff        |         150000
  4 | Ravenclaw         |         250000
  5 | Chudley Cannons   |         500000
  6 | Dechmont Dynamos  |          40000
  7 | Murieston United  |           6000
  8 | Real Sosobad      |          60000
(8 rows)

sports_league=#
```

P. 15: Example of the current output of results and feedback to the user

Screen1 showing all players in list

Click on a player's last name to see more details

Last Name	First Name	Position	Team	V
Potter	Harry	Seeker	Gryffindor	1
Malfoy	Draco	Seeker	Slytherin	7
Gonzalez	Alicia	Goalkeeper	Ravenclaw	5
Laughlin	Kirsty	Chaser	Hufflepuff	4
Ramson	Richard	Chaser	Hufflepuff	1
Marjoribanks	Duncan	Chaser	Ravenclaw	1
Stafford	Joe	Beater	Slytherin	
Laughlin	Andy	Beater	Gryffindor	
Gerrard	Steven	Goalkeeper	Slytherin	

Add a New Superstar

Screenshot 2 shows Harry Potter having been clicked on:

Name: Harry Potter

Position: Seeker

Current Team: Gryffindor

Current Market Value: 100000

Edit Player

Transfer Player

Delete Player

Screen 3 shows the player list after clicking the delete button on Harry Potter; the player has been deleted from the database:

Click on a player's last name to see more details

Last Name	First Name	Position	Team	Valu
Malfoy	Draco	Seeker	Slytherin	7000(
Gonzalez	Alicia	Goalkeeper	Ravenclaw	5000(
Laughlin	Kirsty	Chaser	Hufflepuff	4000(
Ramson	Richard	Chaser	Hufflepuff	1000(
Marjoribanks	Duncan	Chaser	Ravenclaw	1000(
Stafford	Joe	Beater	Slytherin	9000
Laughlin	Andy	Beater	Gryffindor	8500
Gerrard	Steven	Goalkeeper	Slytherin	1000

Add a New Superstar

P. 16: Evidence of an API being used within a program:

First screenshots show the API being called and returned as a promise in JSON format:

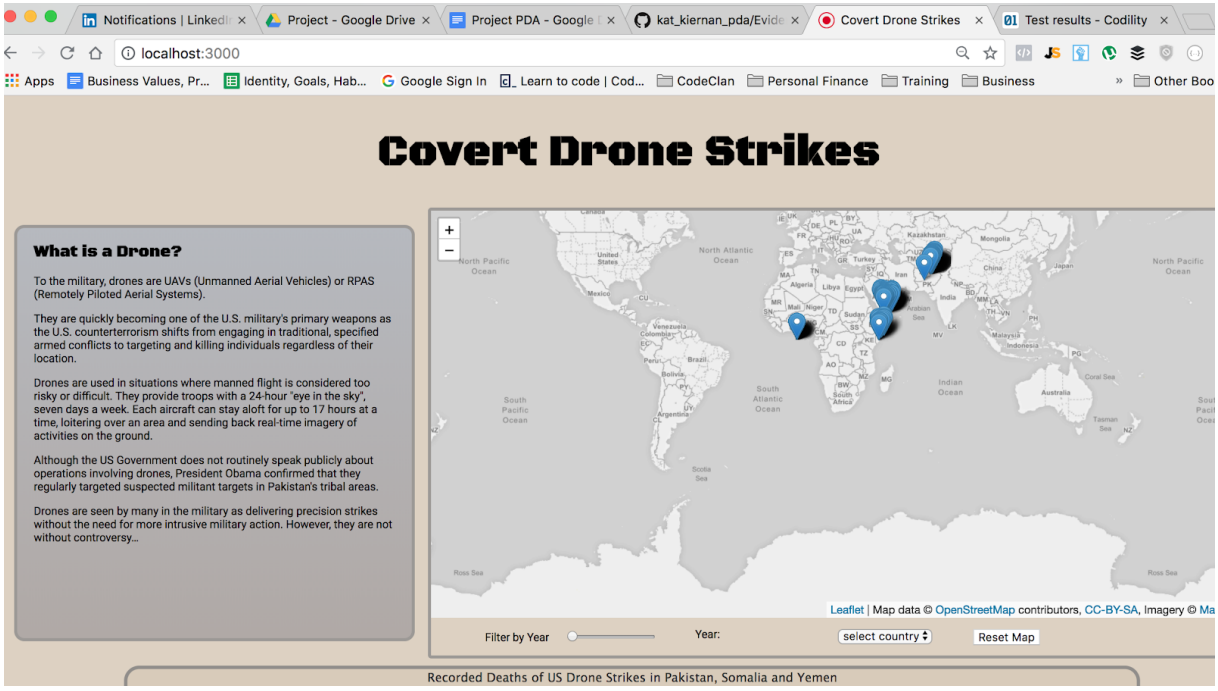
```
const express = require('express');
const app = express();
const path = require('path');
const parser = require('body-parser');
const MongoClient = require('mongodb').MongoClient;
const createRouter = require('./helpers/create_router.js');
const fetch = require('node-fetch');

// MongoClient.connect('mongodb://localhost:27017', (err, client) => {
// const db = client.db("drone_strikes_app");
// const droneStrikes = db.collection('drone_strikes_db');
// const droneStrikesRouter = createRouter(droneStrikes);
// app.use('/api/drone_strikes_db', droneStrikesRouter);
// });

const publicPath = path.join(__dirname, '../client/public');
app.use(express.static(publicPath));

app.use(parser.json());

app.listen(3000, function(){
  console.log(`listening on port ${ this.address().port}`);
});

app.get('/api/drones', (req, res) => {
  const url = 'https://api.dronestre.am/data';

  fetch(url)
    .then(jsonData => jsonData.json())
    .then(data => res.json(data)); // MODIFIED
});
```

```
Drones.prototype.getData = function () {
  const url = this.url;
  const request = new Request(url);
  const handleRequest = (responseData) => {
    this.dronesData = responseData;
    PubSub.publish('Drones:data-ready', this.dronesData);

  }

  request.get()
    .then(handleRequest)
    .catch(error => console.error(error));
};
```

And used in App.js (localHost 3000):

```
document.addEventListener('DOMContentLoaded', ()=> {
  console.log("hello");

const dronesUrl ='http://localhost:3000/api/drones'
const drones = new Drones(dronesUrl);
drones.getData();
```

The API being used by the program while running on LocalHost 3000:

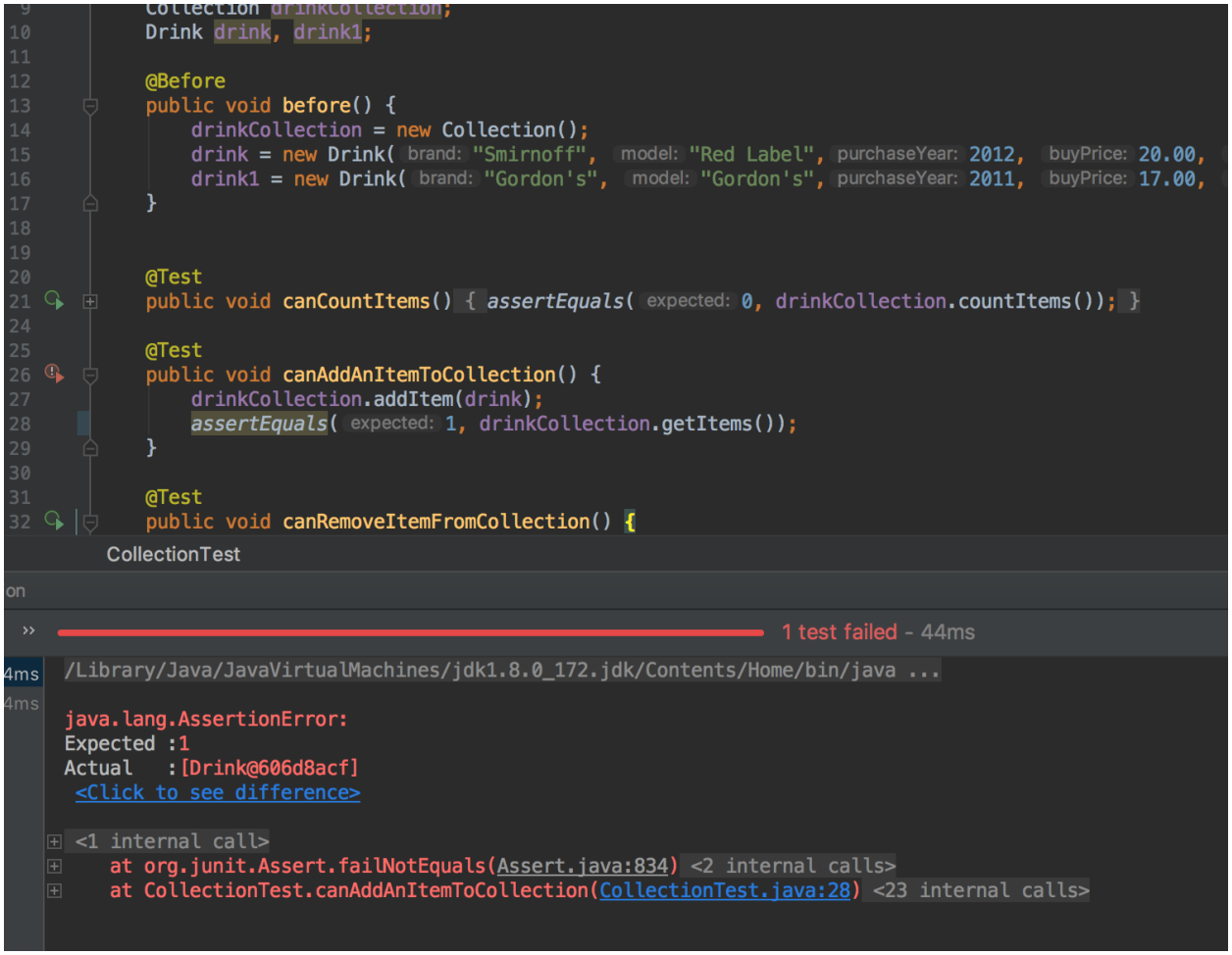


P. 17: Bug tracking report from Javascript Drone project:

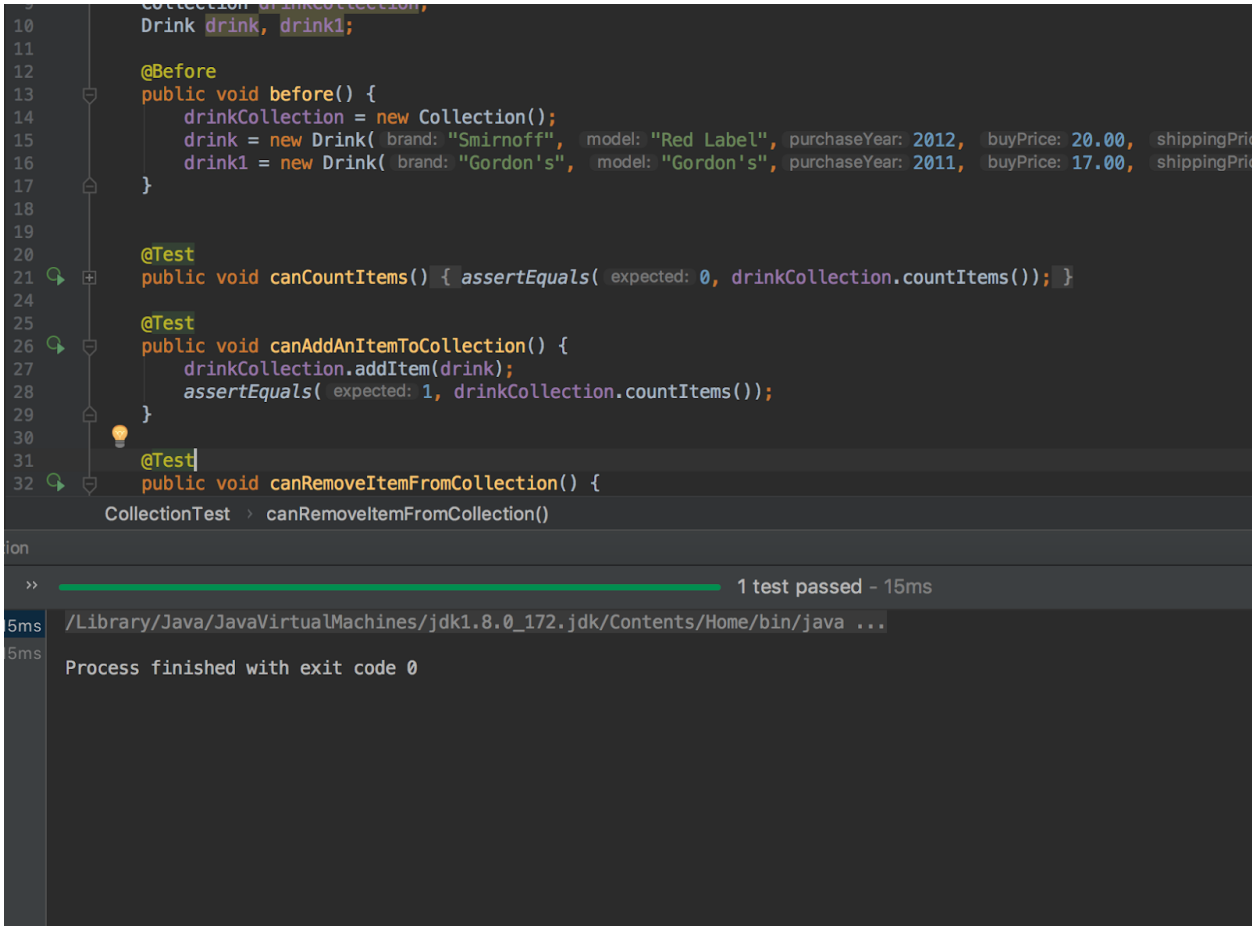
Category	Label	Value
Bug ID	ID number	#1
	Name	COUNTRY VIEW - unable to re-render map after selecting a country
	Reporter	Andy L
	Submit Date	09/07/18
Bug overview	Summary	When selecting a country to view the drone strikes, the map does not re-render the markers to show only that country
	URL	http://localhost:3000/
	Screenshot	
Environment	Platform	Macintosh
	Operating System	OS X 10.12.0
	Browser	Chrome 53
Bug details	Steps to reproduce	Go to select a country dropdown, select country, map does not re-render the markers
	Expected result	The markers on the map should re-render, showing only those markers relevant to the selected country
	Actual result	Map does not change
	Description	/
Bug tracking	Severity	Major
	Assigned to	Andy
	Priority	High
Notes	Notes	/

P. 18: Demonstrate testing in a program:

A. Example of test code for test 'canAddAnItemToCollection' and the test failing due to calling the wrong function being called (getItems instead of countItems):



B. Example of the test passing once the test code has been fixed - ‘countItems’ method called instead of ‘getItems’


Drink drink, drink1;
@Before
public void before() {
drinkCollection = new Collection();
drink = new Drink(brand: "Smirnoff", model: "Red Label", purchaseYear: 2012, buyPrice: 20.00,
drink1 = new Drink(brand: "Gordon's", model: "Gordon's", purchaseYear: 2011, buyPrice: 17.00,
}
@Test
public void canCountItems() { assertEquals(expected: 0, drinkCollection.countItems()); }
@Test
public void canAddAnItemToCollection() {
drinkCollection.addItem(drink);
assertEquals(expected: 1, drinkCollection.countItems());
}
@Test
public void canRemoveItemFromCollection() {
CollectionTest › canRemoveItemFromCollection()
1 test passed - 15ms
/Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Contents/Home/bin/java ...
Process finished with exit code 0