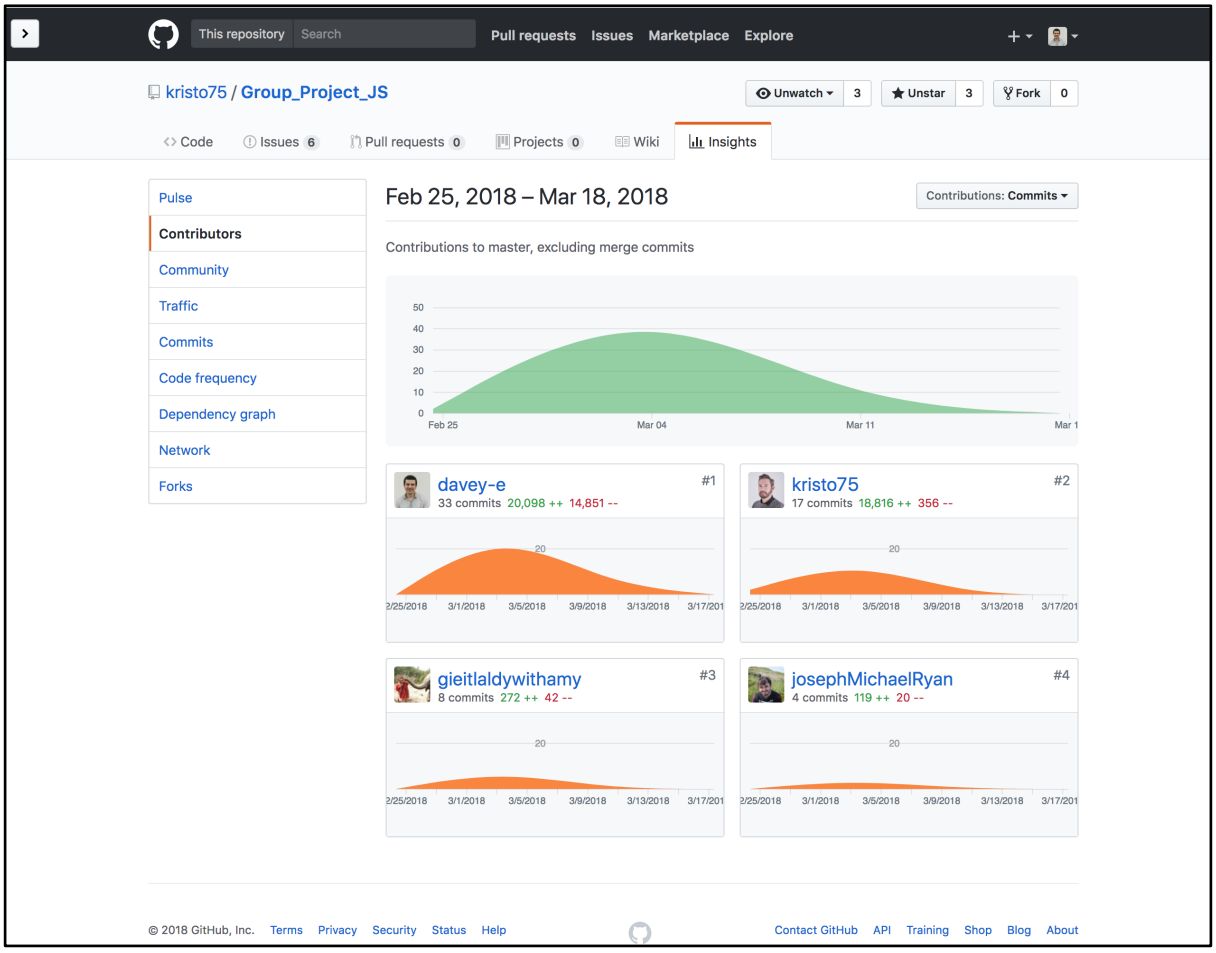
Project Unit – Evidence

David Ellis
Cohort E18

P1 – Group Project – GitHub Contributors Page Screenshot



P2 – Group Project – Project Brief Screenshots

Project Brief From Classnotes:

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

Some samples of existing apps for inspiration:

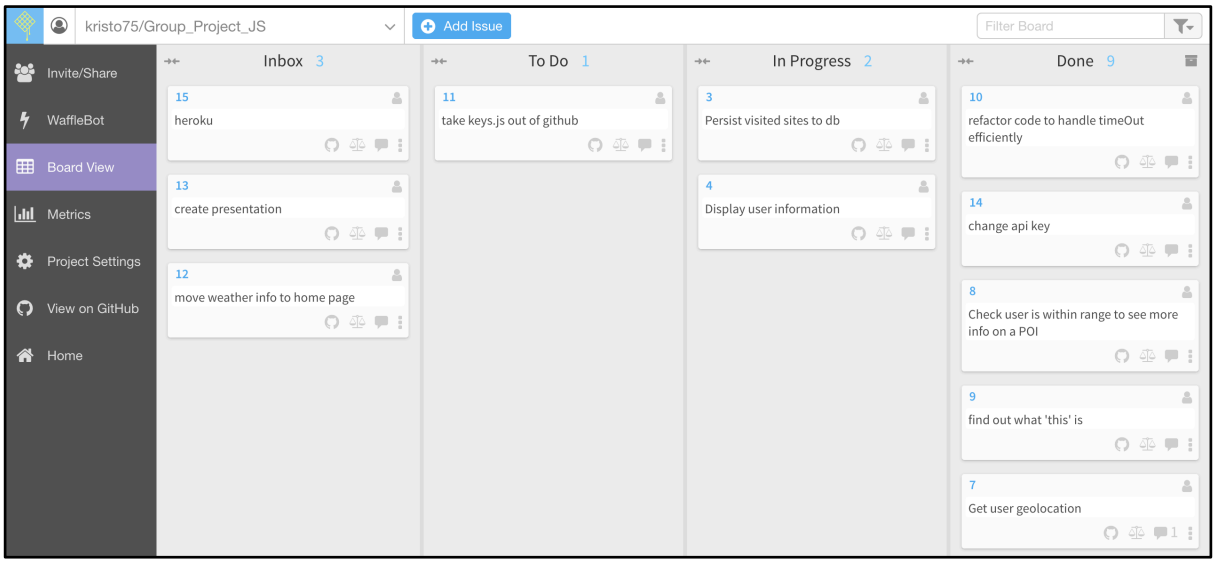
- http://chemistryset.chemheritage.org/#/
- http://www.royalmailheritage.com/main.php
- http://education.iceandsky.com/
- http://histography.io - may only work in Safari
- http://worldpopulationhistory.org/map/1838/mercator/1/0/24/

Project brief that we created for our specific app:

MVP:
Display a map with points of interest (P.O.I.)
Display user information
View basic information about P.O.I.
View detailed information about P.O.I.
Persist visited site to db

Extensions:
Display map of countries visited
Text to speech
Suggested route between P.O.I.
Add a weather update
Create P.O.I. API

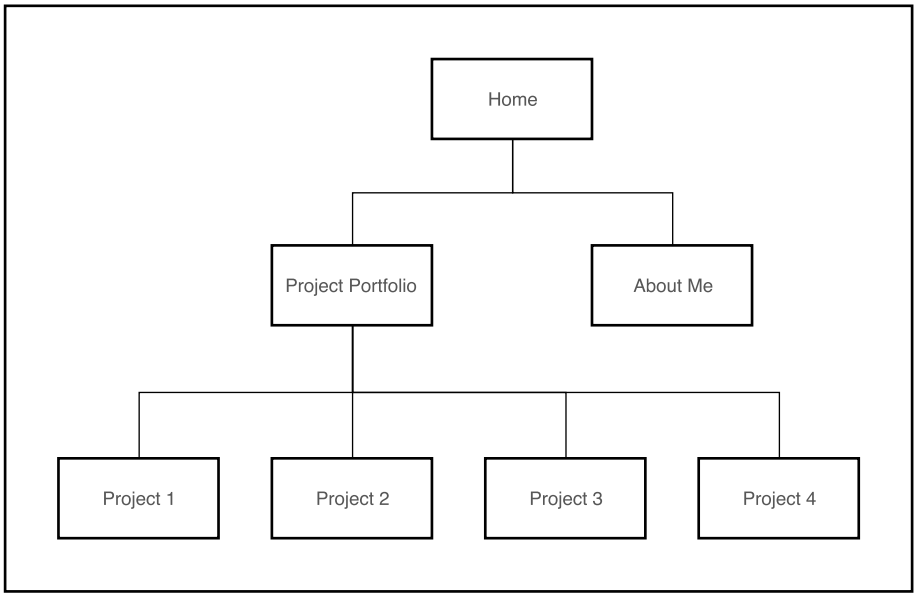
P3 – Group Project – Planning Screenshots



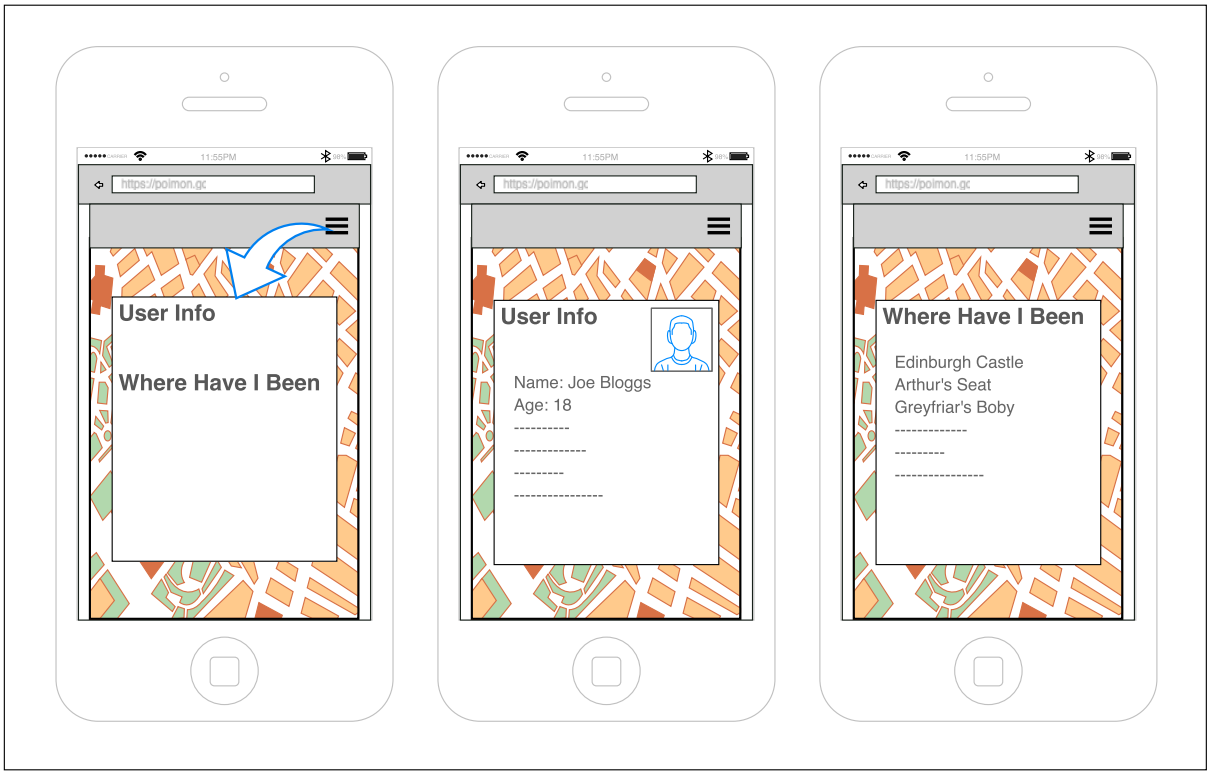
P4 – Acceptance Criteria and Test Plan

Acceptance Criteria	Expected Result/Output	Pass/Fail
A user is able to see a list of all transactions in the system for all tags	The main page of the app displays a table with all of the transactions for all tags fetched from the database shown. The table can be scrolled to shown transactions that are at the bottom of the table	Pass
A user is able to see the total amount spent for all transactions for all tags	The main page of the app displays the total amount of all transactions in the header bar	Pass
A user is able to see a list of the available tags that can be selected to filter the data	The main page of the app shows buttons for each of the tags that are in the system in the navigation bar on the left-hand side.	Pass
A user is able to filter the data by selecting one of the available tags	The user can click on each tag button in the navigation bar on the left-hand side to filter the data in the table by tag	Pass
A user is able to add a new transaction to the system	A New Transaction button is displayed on the navigation bar on the left-hand side of the main page of the app. When the user clicks on this button a form is displayed to allow the user to ender the transaction data	Pass
A user is able to select the date, amount, tag and vendor when adding a new transaction	The New Transaction form displays input elements for date, amount, tag and vendor and also displays a Submit button	Pass
A user is able to add a new tag to the system	A New Tag button is displayed on the navigation bar on the left-hand side of the main page of the app. When the user clicks on this button a form is displayed to allow the user to ender the new tag	Pass
A user is able to add a new vendor to the system	A New Vendor button is displayed on the navigation bar on the left-hand side of the main page of the app. When the user clicks on this button a form is displayed to allow the user to ender the new vendor	Pass

P5 – User Sitemap

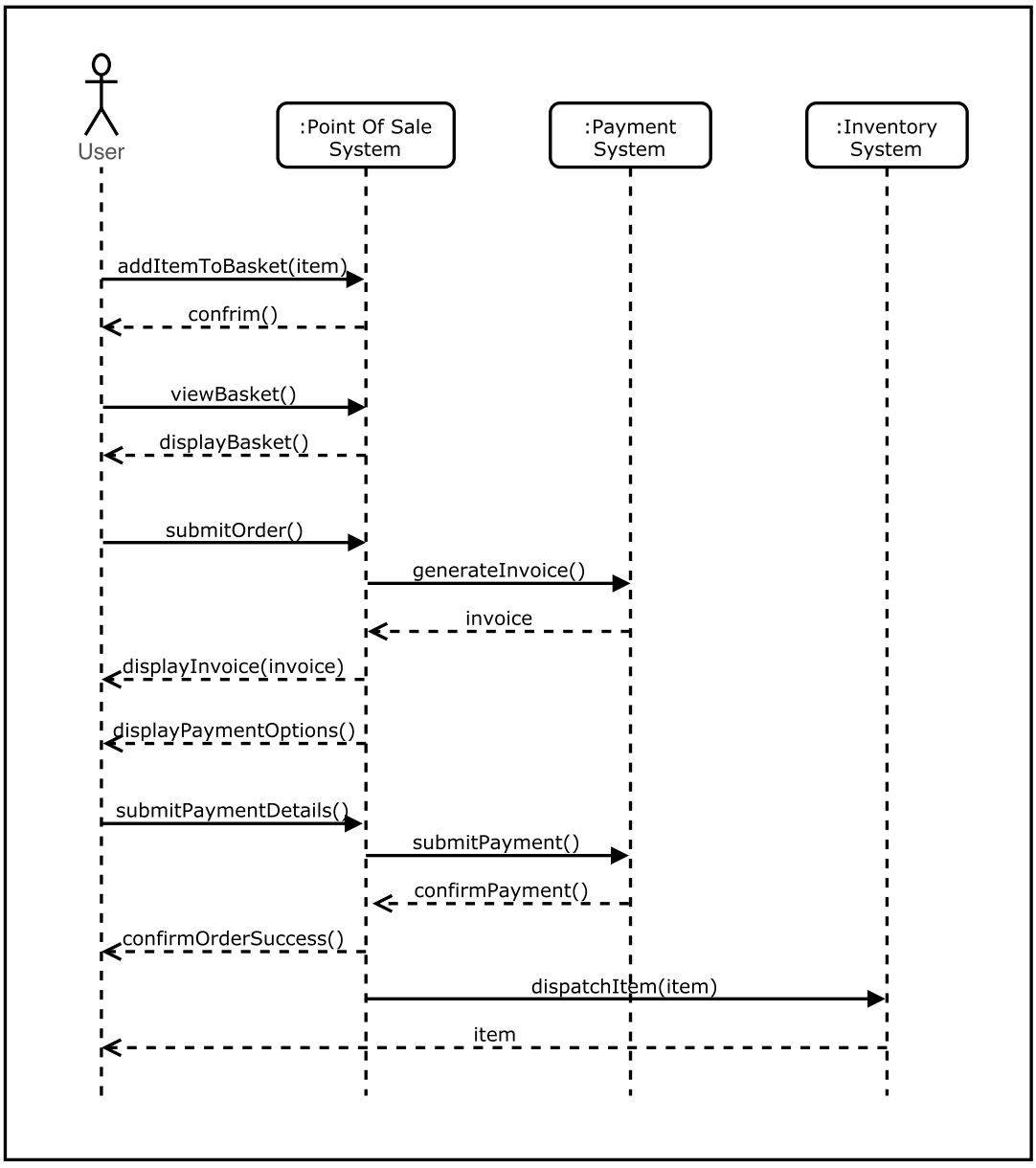


P6 – Wireframe Diagrams

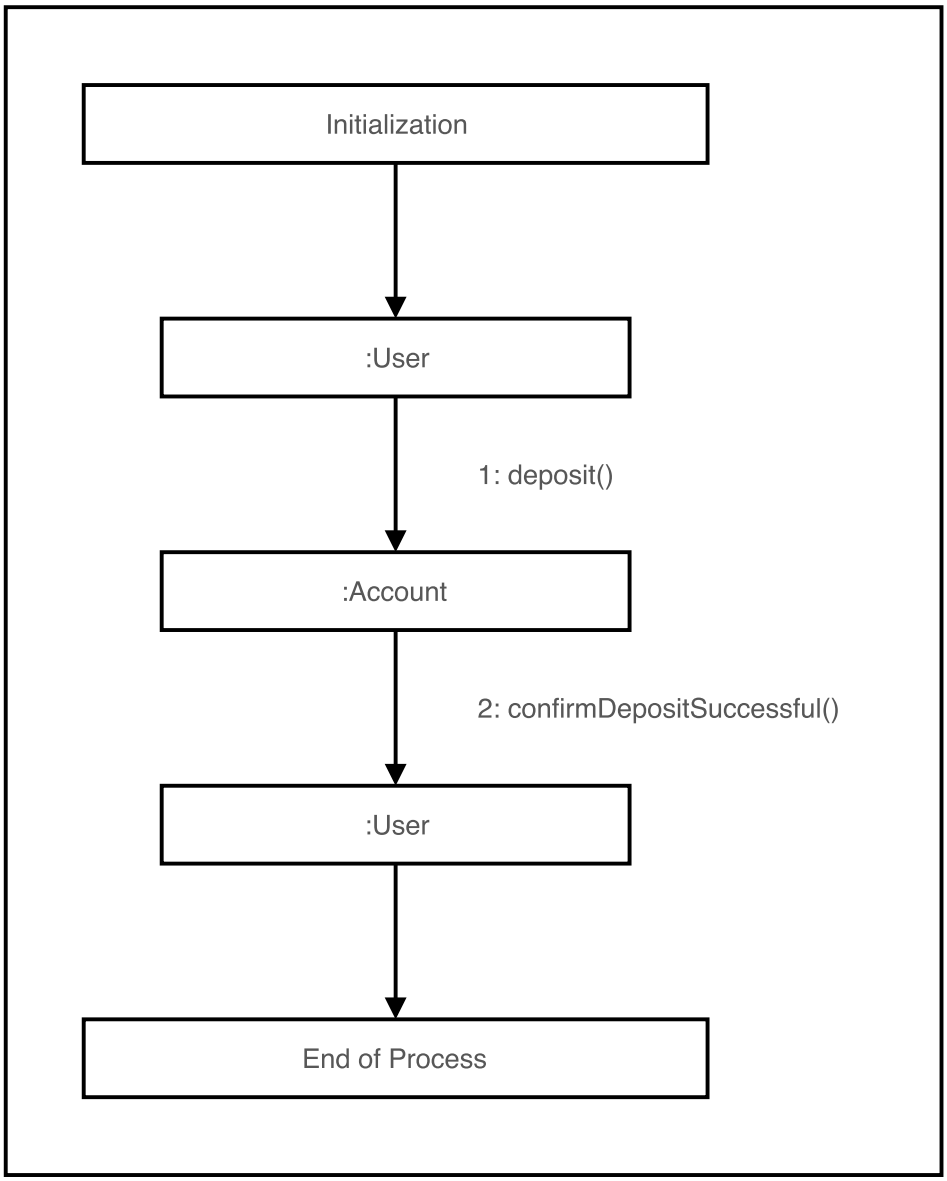


P7 – System Interaction Diagrams

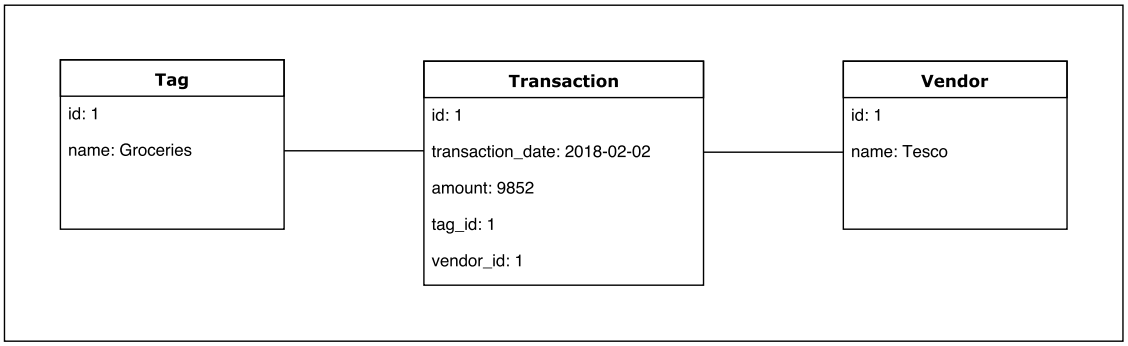
Sequence Diagram:

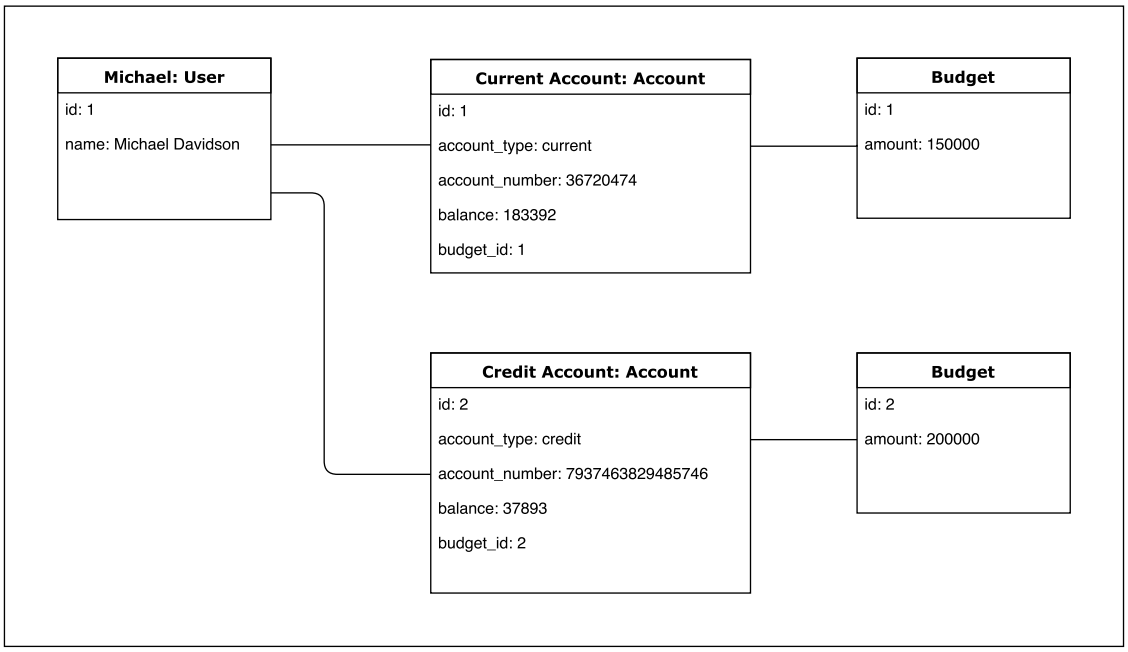


Collaboration Diagram:



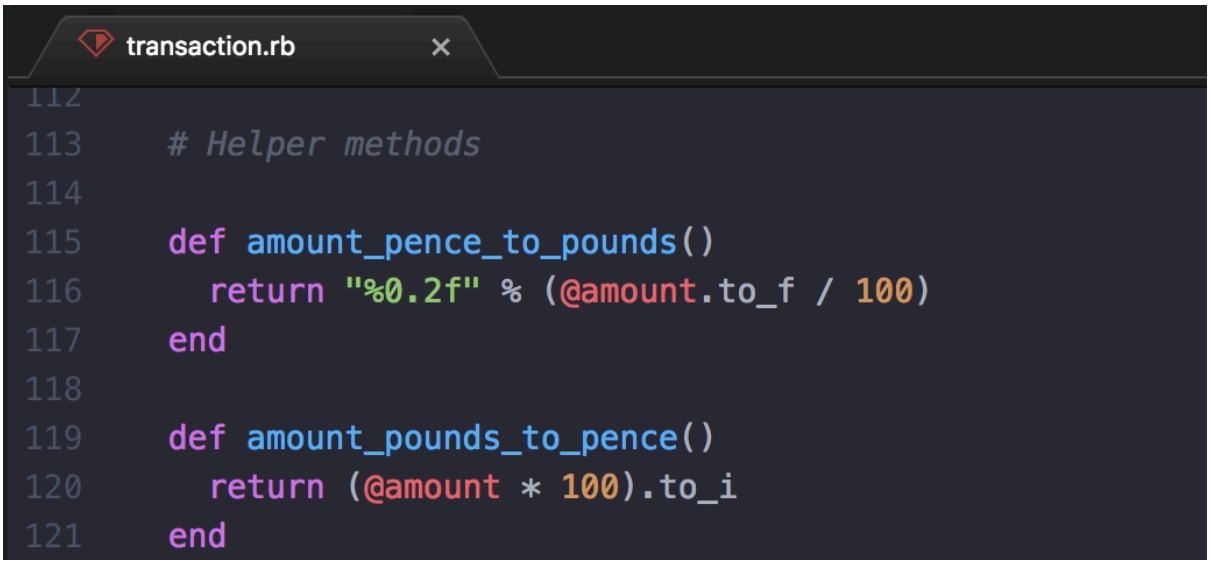
P8 – Object Diagrams





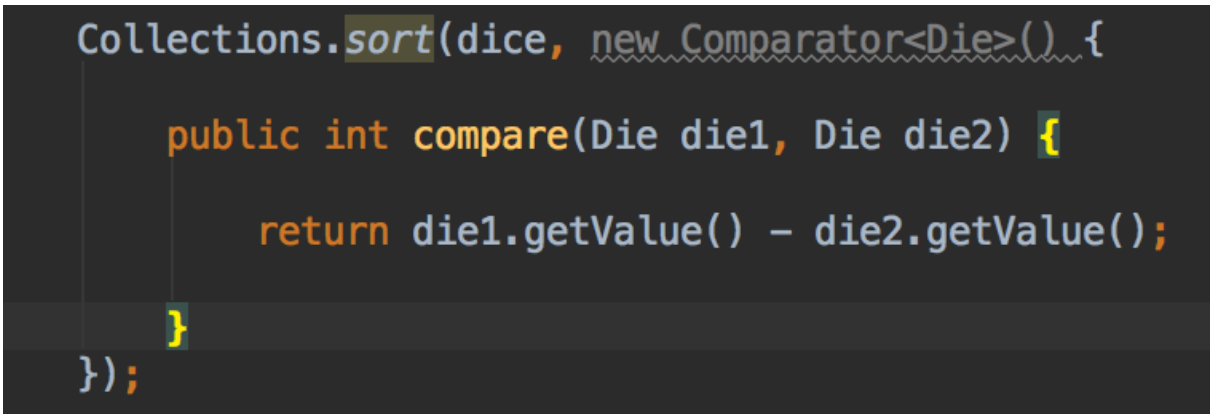
P9 – Algorithm Example Screenshots and Explanations

Algorithm Example 1:



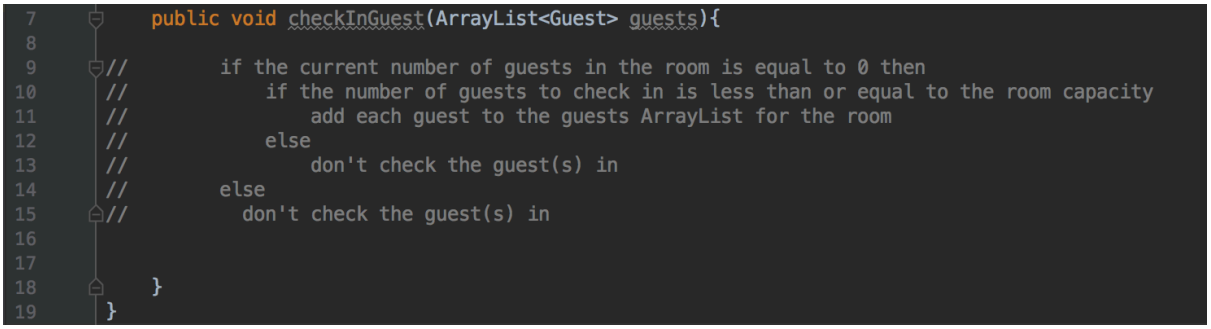
The two methods in the screenshot above were algorithms that I wrote as part of my Ruby project. The project involved displaying information about monetary transactions. The monetary value data was stored in the database in pence and the data needed to be displayed in pounds and pence in the user interface. The first algorithm above takes the value in pence from the database and divides by 100 and also converts the integer value into a floating point value with 2 decimal places. The second algorithm takes the value in pounds and pence from the user interface and multiplies by 100 to convert to pence and also converts the floating point value into an integer.

Algorithm Example 2:



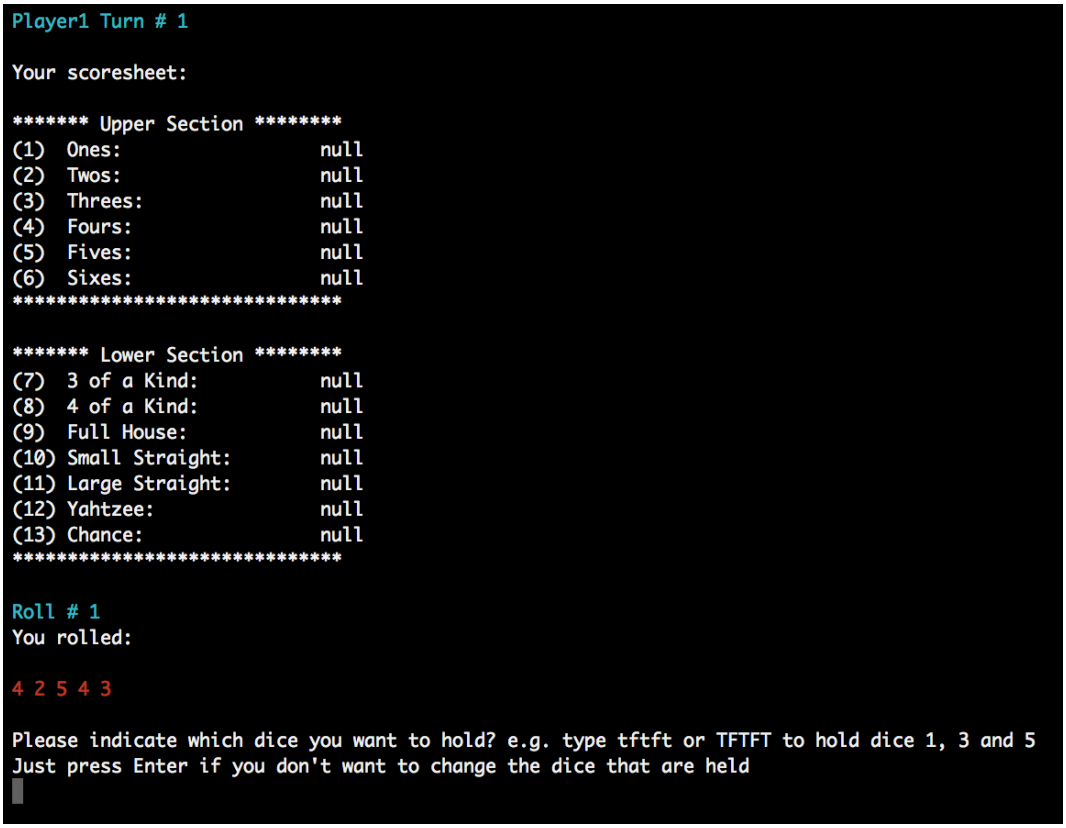
This is an algorithm I wrote as part of my Java project, which was a Yahtzee game. As part of the logic for the game I needed to sort the dice, which consisted of 5 Die objects. The die objects each have a value (an integer number between 1 and 6) and I needed to sort the 5 dice based on the value, so I created an algorithm to use a comparator to allow the sort to work based on the die values.

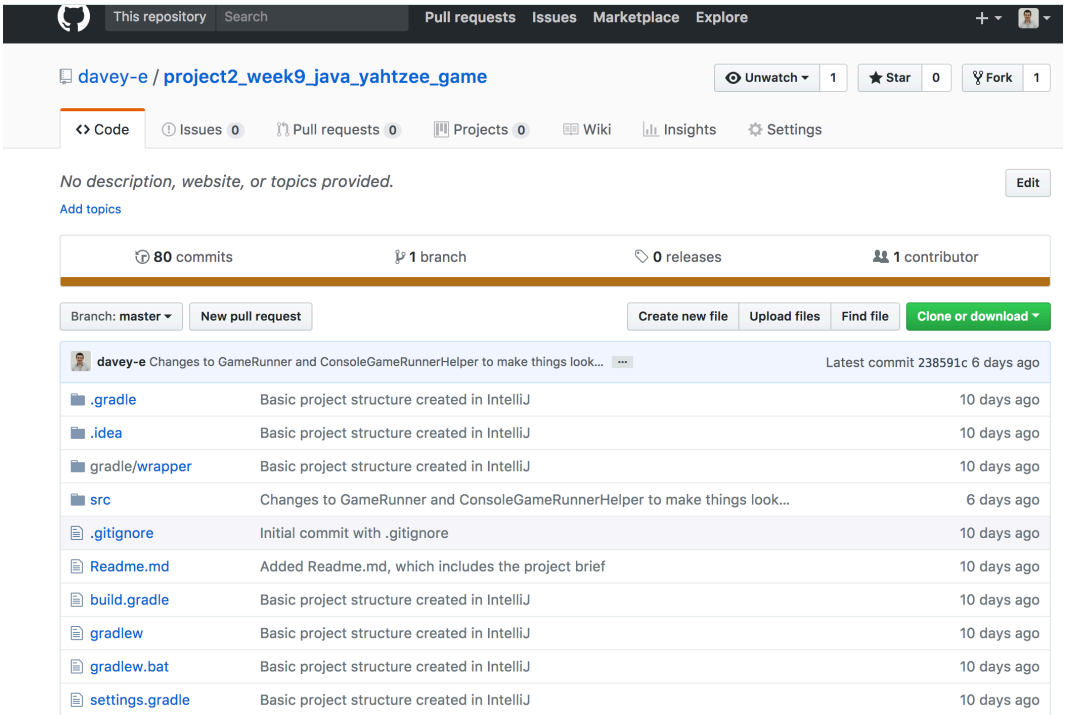
P10 – Pseudocode Example Screenshot



```
7       public void checkInGuest(ArrayList<Guest> guests){
8
9   //        if the current number of guests in the room is equal to 0 then
10  //            if the number of guests to check in is less than or equal to the room capacity
11  //                add each guest to the guests ArrayList for the room
12  //            else
13  //                don't check the guest(s) in
14  //        else
15  //          don't check the guest(s) in
16
17
18      }
19  }
```

P11 – Individual Project Screenshot and GitHub Link

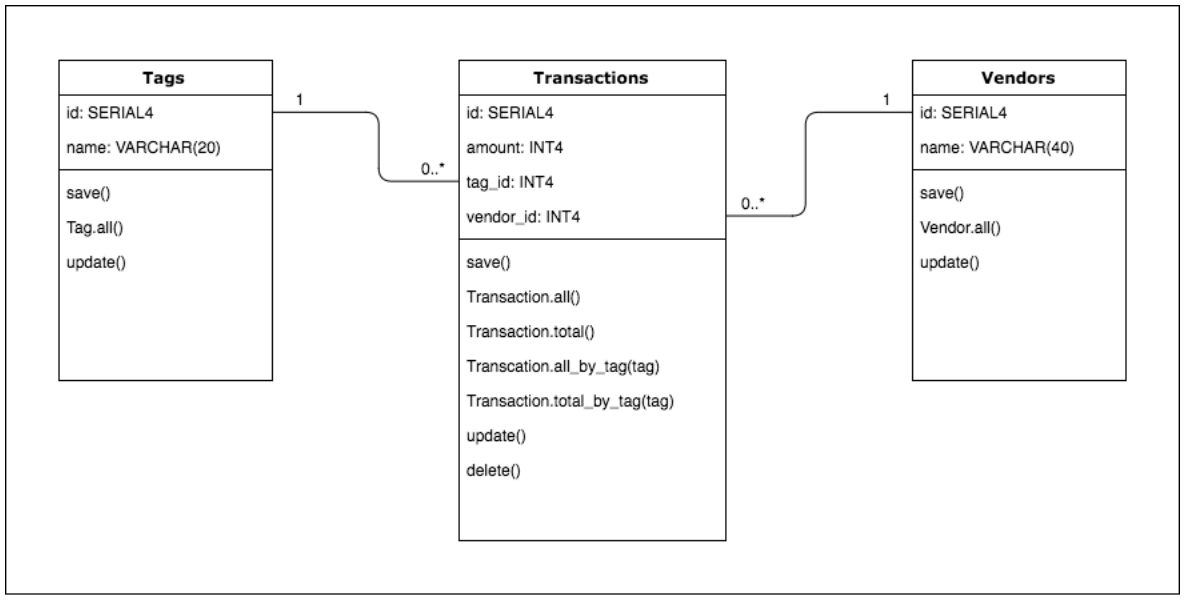




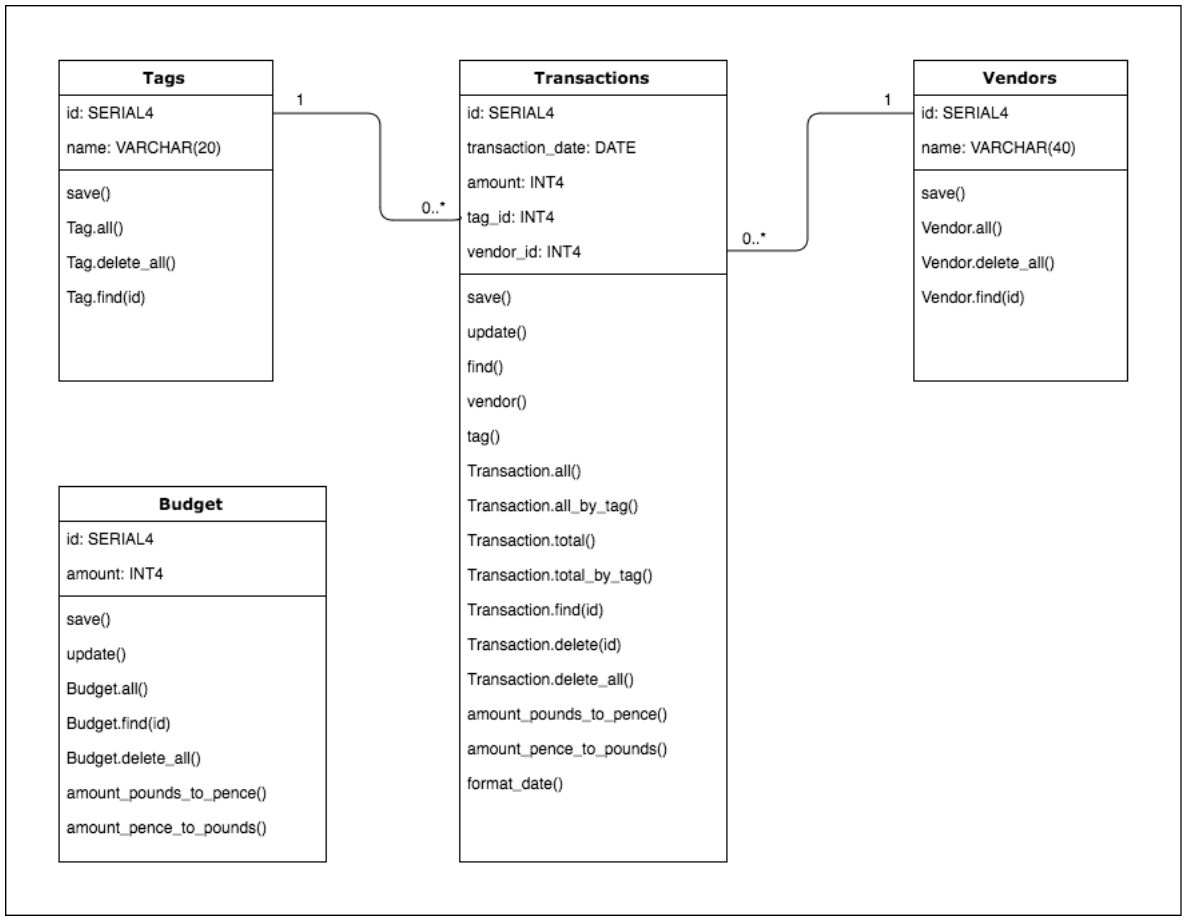
https://github.com/davey-e/project2_week9_java_yahtzee_game

P12 – Project Planning Screenshots/Photos at Different Stages

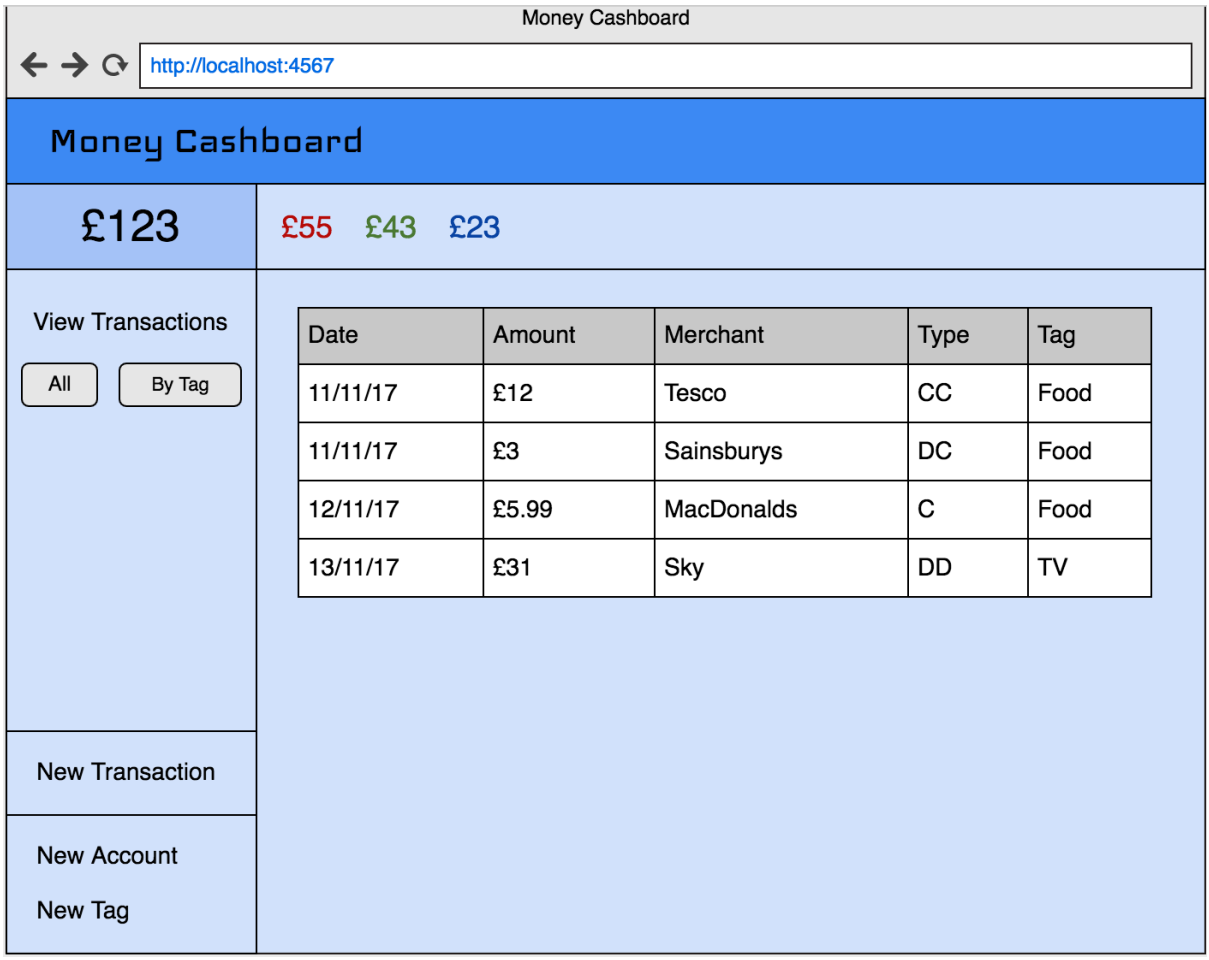
Class diagram as planned at the start of the project:



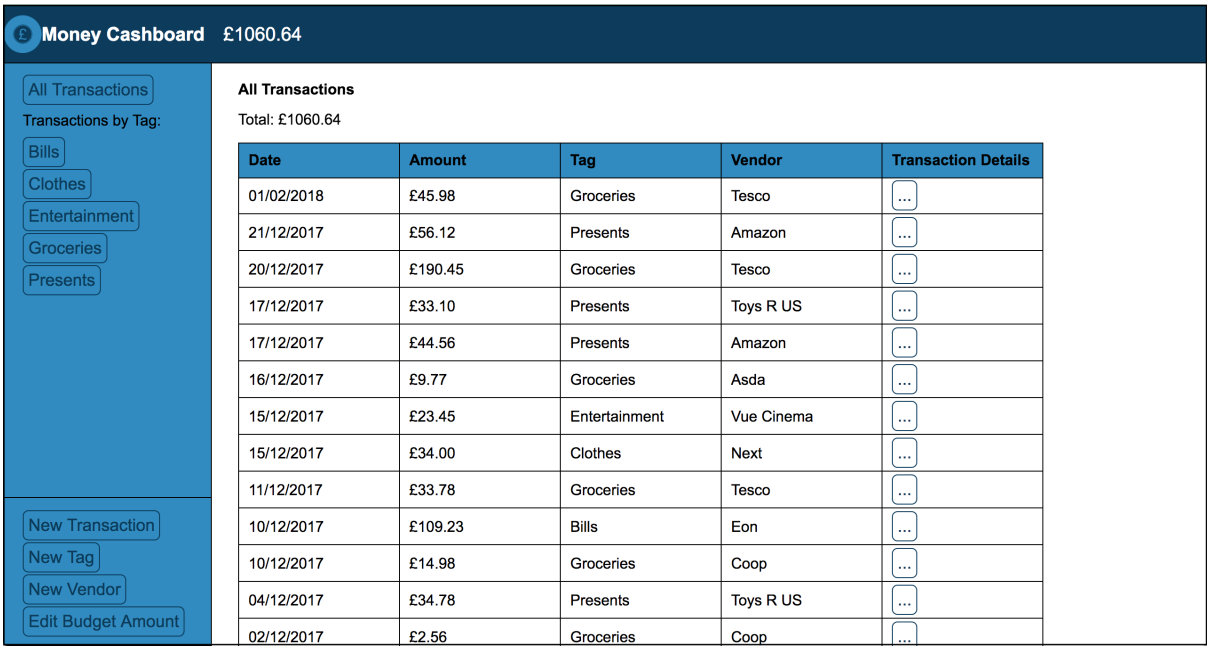
Class diagram at the end of the project:



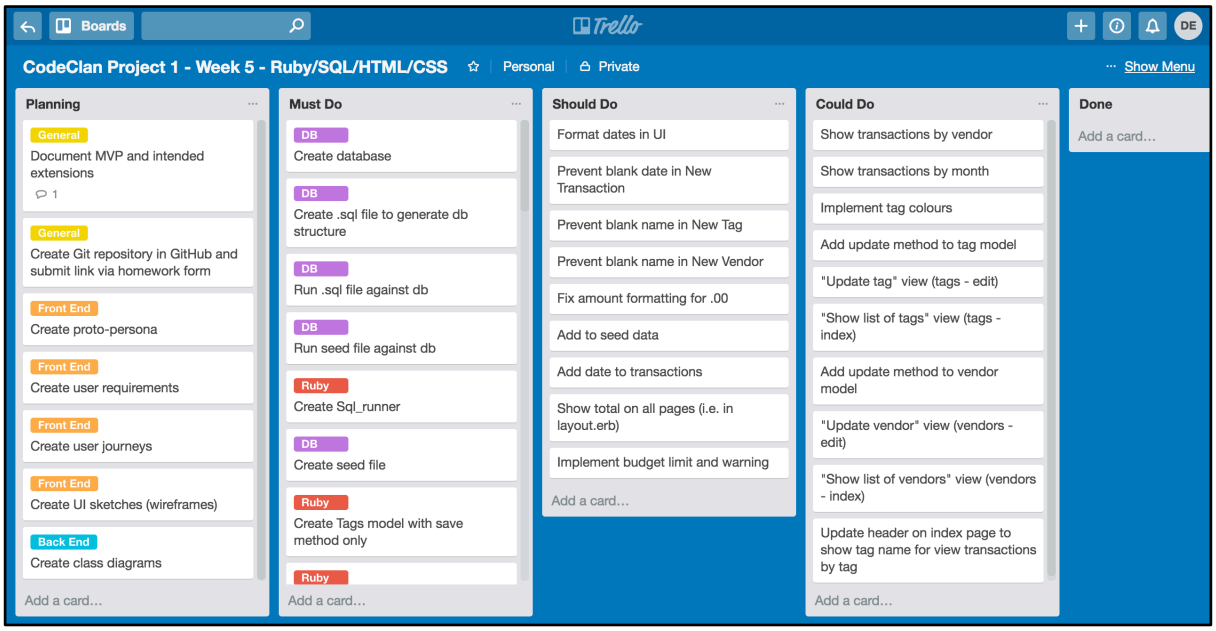
Wireframe of the user interface at the start of the project:



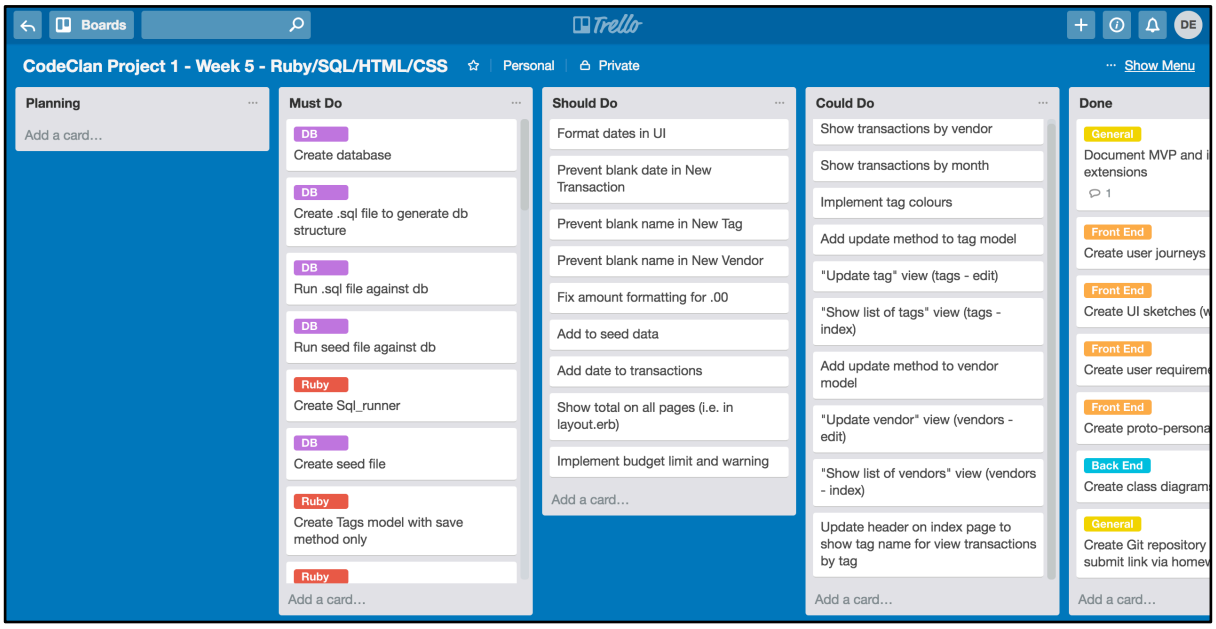
Actual user interface at the end of the project:



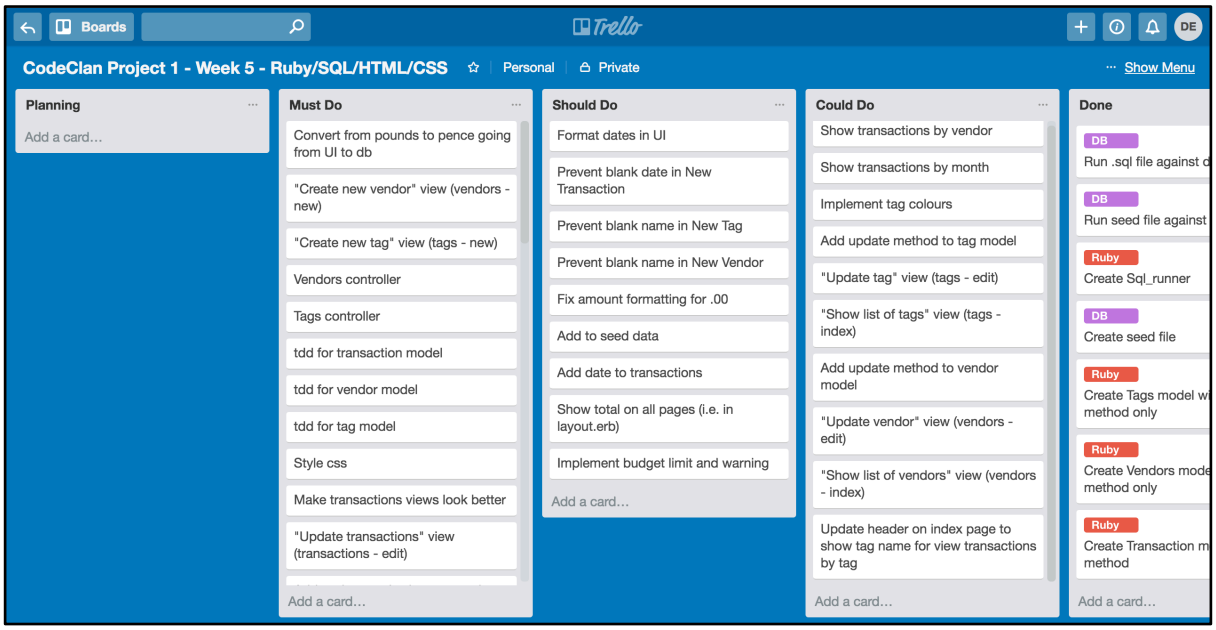
Trello board at the start of the project:



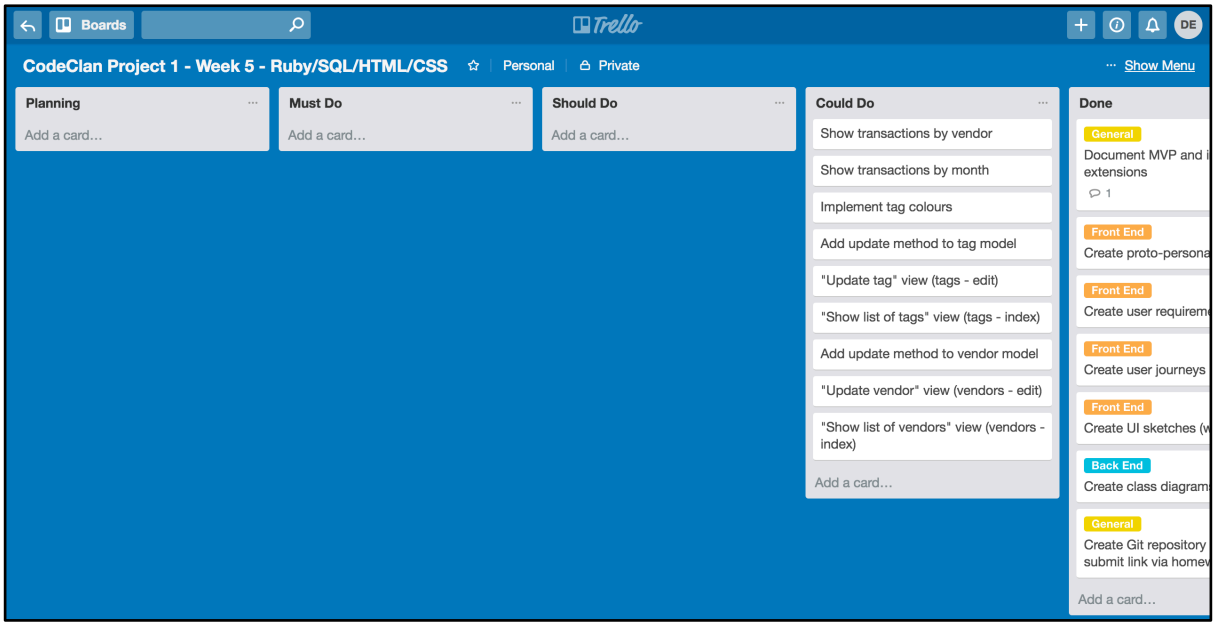
Trello board after planning items complete:



Trello board after basic code structure items complete:

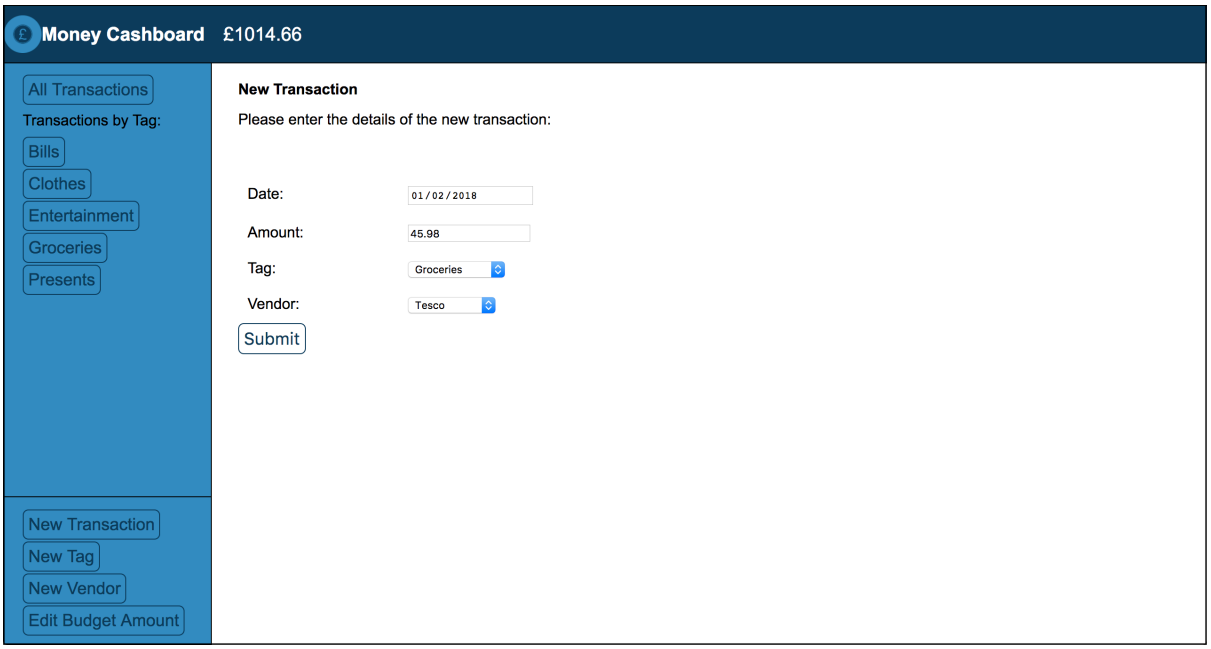


Trello board at the end of the project:

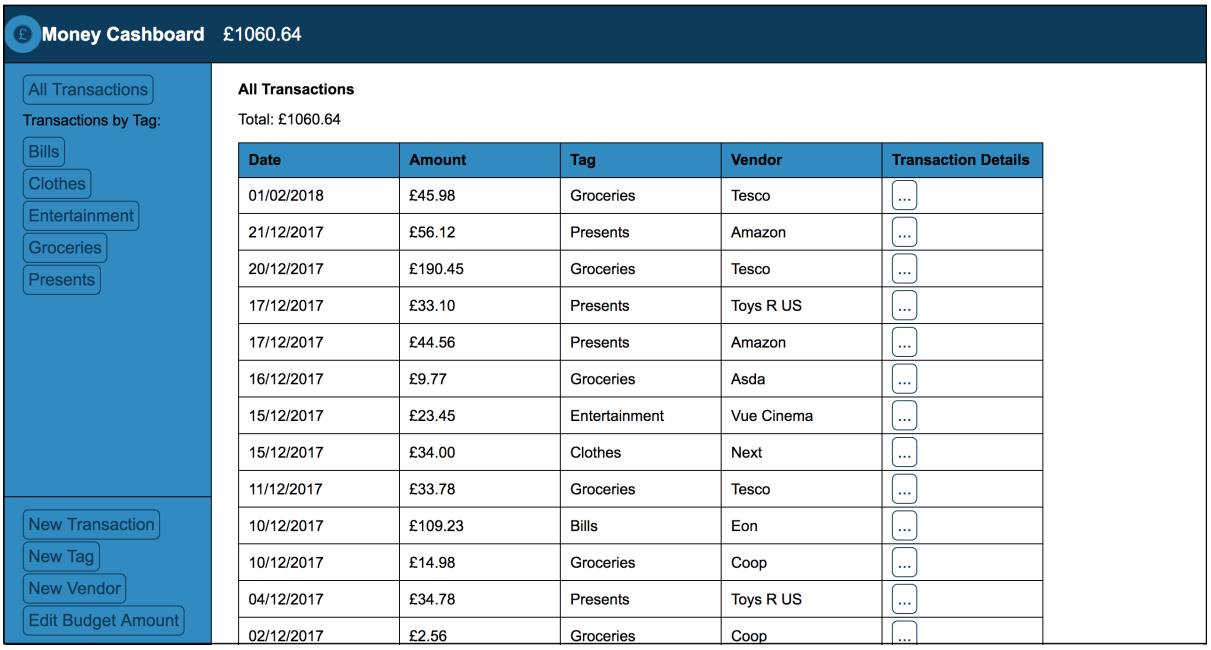


P13 – User Input Processing Screenshots

User input:

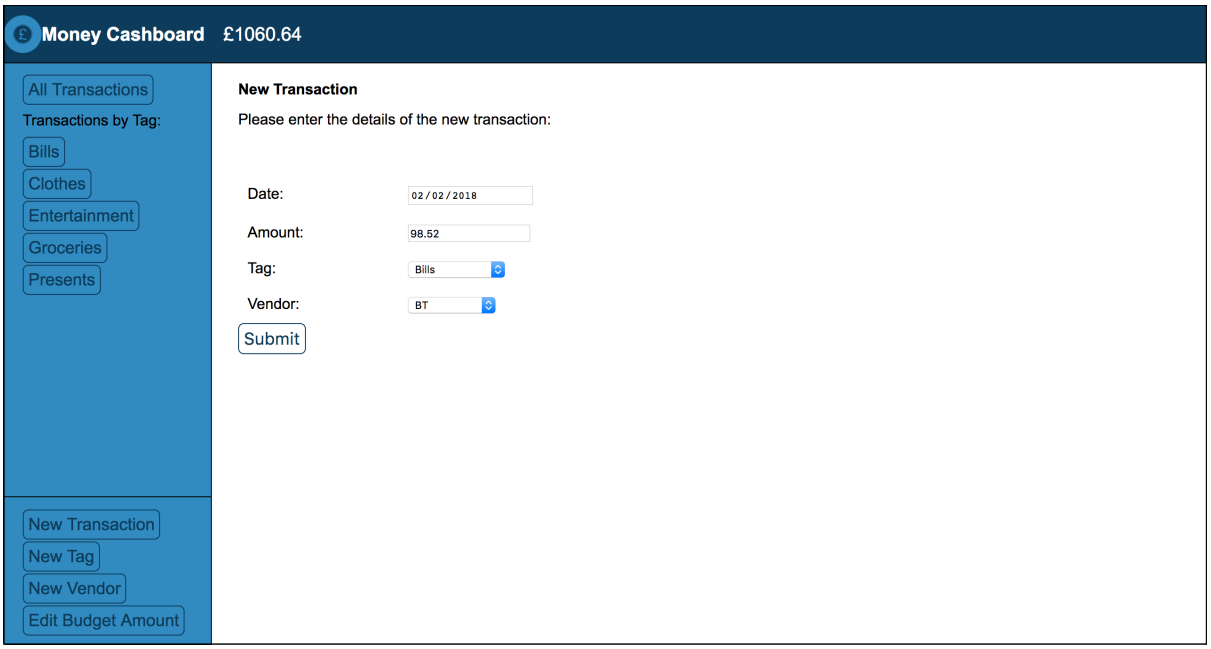


User input saved:



P14 – Data Interaction and Persistence Screenshots

User Input:

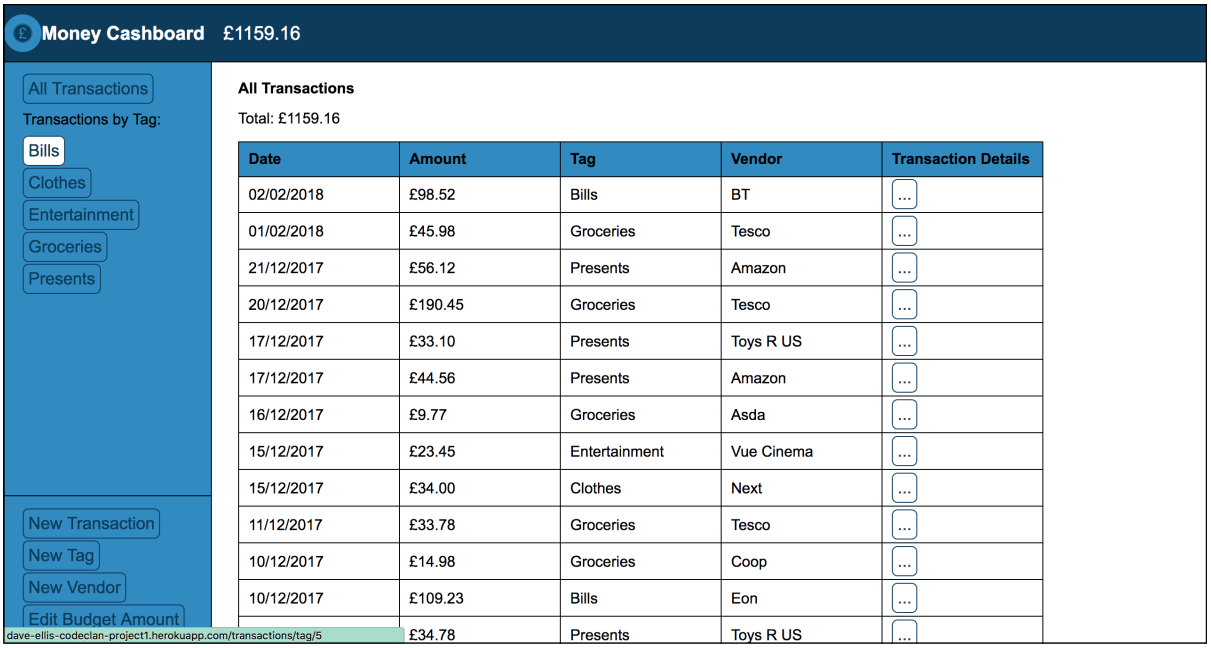


Data saved in database:

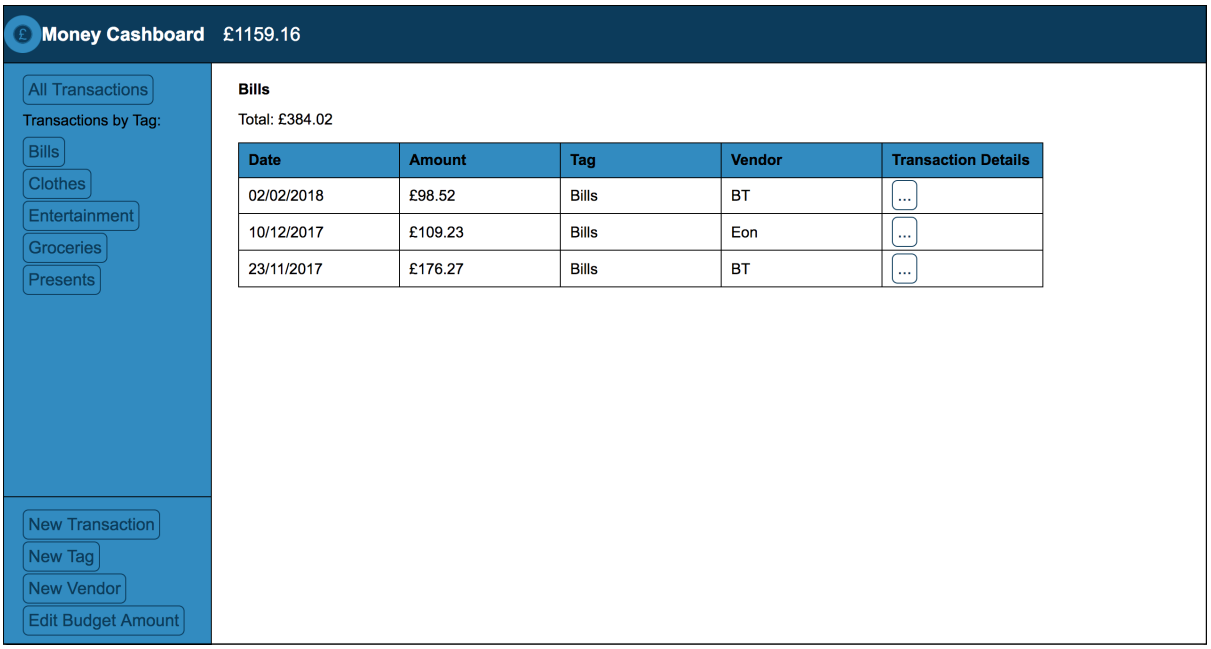
```
money_cashboard=# select transaction_date, amount, tags.name as tag, vendors.name as vendor
from transactions, tags, vendors
where transactions.tag_id = tags.id
and transactions.vendor_id = vendors.id
order by transaction_date desc;
 transaction_date | amount |      tag      |   vendor
------------------+--------+---------------+------------
 2018-02-02       |   9852 | Bills         | BT
```

P15 – User Feedback/Output Screenshots

The user wants to see only the transactions with the “Bills” tag, so they click on the Bills button under Transactions by Tag:



The program responds by showing only the transactions with the Bills tag in the table:



P16 – API Usage Screenshots

index.js

```
const app = function(){
    url = "https://restcountries.eu/rest/v2/all";

    // display last selected country
    const countryObject = getCountryFromLocalStorage();
    if(countryObject !== null){
        createCountryDetailList(countryObject);
    }

    // get data from api
    makeRequest(url, requestComplete);
}

// get data from api
const makeRequest = function(url, callback){
    const request = new XMLHttpRequest();
    request.open("GET", url);
    request.addEventListener("load", callback);
    request.send();
}

// callback used in makeRequest to convert api data into array of country objects
const requestComplete = function(){
    if(this.status !== 200) return;
    const jsonString = this.responseText; //This is a string
    const countries = JSON.parse(jsonString); //This parses the string into a js object
    populateList(countries);
}
```

P17 – Bug Tracking Report

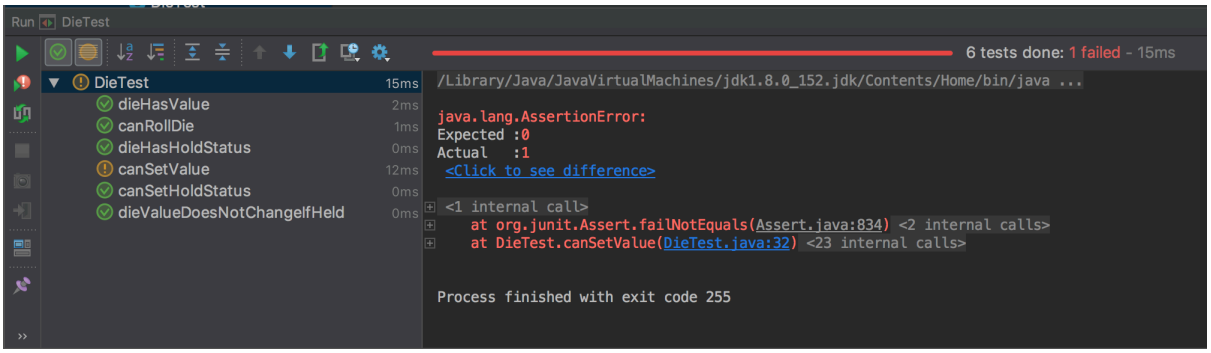
Issue	Status	Bug Fix	Status After Fix
User can see 5 random points of interest from a 500m radius of their current location	Failed	Added a callback to the javascript code to make sure that points of interest are retrieved from the Sygic Travel api before randomly selecting and displaying 5 points of interest	Passed
User can click on the marker for a point of interest to see basic information about the selected place	Failed	Fixed a bug which was causing the information for the wrong point of interest to be displayed due to the incorrect POI id being passed to the Sygic Travel api	Passed
User can see detailed information for a point of interest once they are within 50m of the location	Failed	Added code to calculate the distance between the users current position and the point of interest position and then if the distance is less than 50m detailed information is retrieved from the Sygic Travel api and displayed in a modal	Passed
User can save a point of interest to their list of visited places once they are within 50m of the location	Failed	Added code to allow locations to be added to a Mongo database, but saving the locations only happens once the user is within 50m of the location and is triggered when they click on the marker for the location	Passed
User can see a list of the points of interest that they have visited	Failed	Fixed a bug which was causing none of the points of interest to by displayed due to the fact that the id of the database entries was not being passed in the call to get data from the database	Passed
User can see the weather for their current location	Failed	Fixed a bug which was causing the weather to be displayed multiple times due to the fact that the area where the weather data is displayed was not being cleared before adding the weather data to the DOM	Passed

P18 – Testing Example Screenshots

Example of test code:

```java
public class DieTest {

    Die die1;

    @Before
    public void before(){

        die1 = new Die();
    }

    @Test
    public void dieHasValue(){
        assertEquals( expected: 0, die1.getValue());
    }

    @Test
    public void dieHasHoldStatus(){
        assertFalse(die1.getHoldStatus());
    }

    @Test
    public void canSetValue(){
        die1.setValue(1);
        assertEquals( expected: 0, die1.getValue());
    }

    @Test
    public void canSetHoldStatus(){
        die1.setHoldStatus(true);
        assertTrue(die1.getHoldStatus());
    }

    @Test
    public void canRollDie(){
        die1.rollDie();
        assertFalse( condition: die1.getValue() > 6);
        assertFalse( condition: die1.getValue() < 1);
    }
```

Test failing to pass:



Test code with errors corrected:

```
9   public class DieTest {
10
11      Die die1;
12
13      @Before
14      public void before(){
15
16          die1 = new Die();
17      }
18
19      @Test
20      public void dieHasValue(){
21          assertEquals( expected: 0, die1.getValue());
22      }
23
24      @Test
25      public void dieHasHoldStatus(){
26          assertFalse(die1.getHoldStatus());
27      }
28
29      @Test
30      public void canSetValue(){
31          die1.setValue(1);
32          assertEquals( expected: 1, die1.getValue());
33      }
34
35      @Test
36      public void canSetHoldStatus(){
37          die1.setHoldStatus(true);
38          assertTrue(die1.getHoldStatus());
39      }
40
41      @Test
42      public void canRollDie(){
43          die1.rollDie();
44          assertFalse( condition: die1.getValue() > 6);
45          assertFalse( condition: die1.getValue() < 1);
46      }
```

Tests passing: