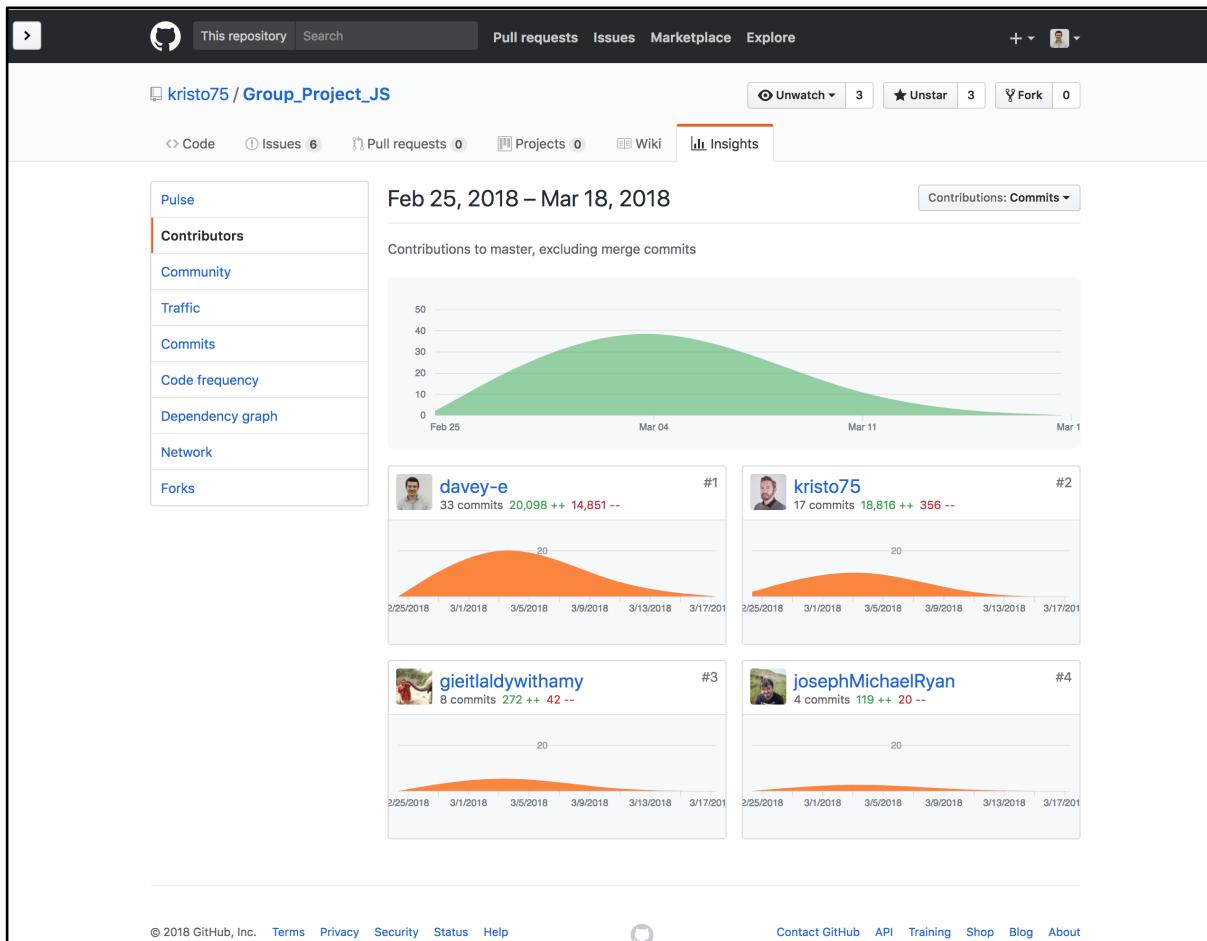


Project Unit – Evidence

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/1838mercator/10/24/>

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

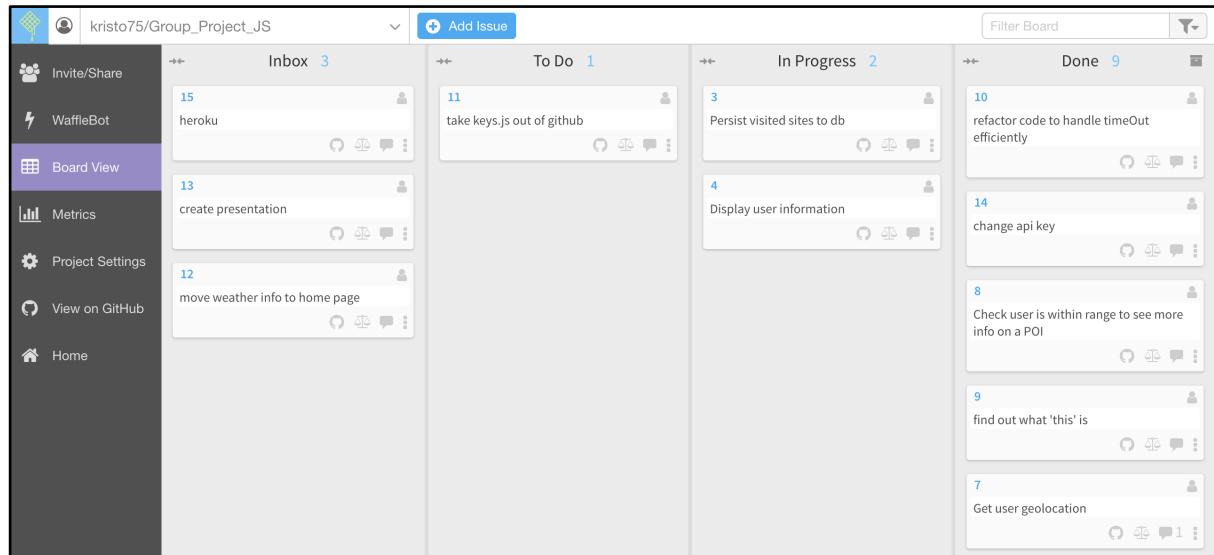
Text to speech

Suggested route between P.O.I.

Add a weather update

Create P.O.I. API

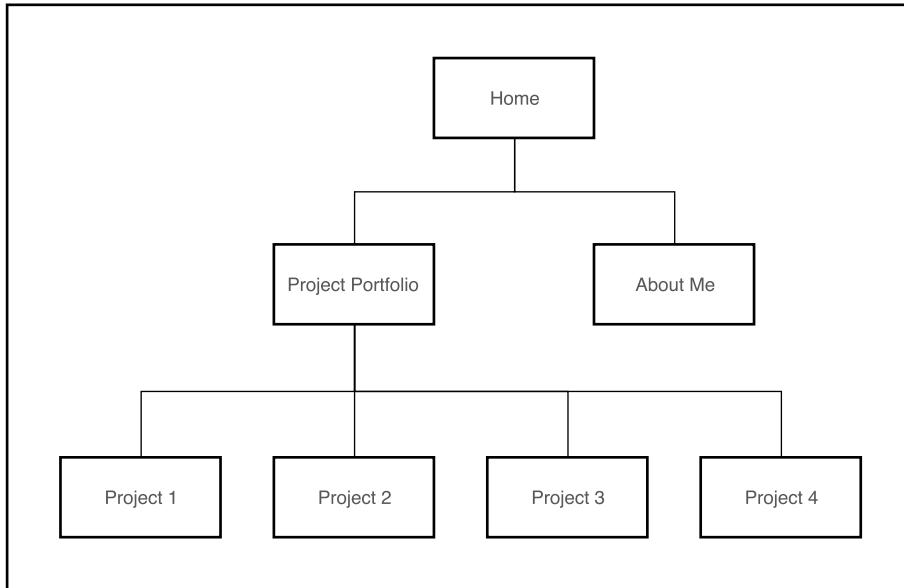
P3 – Group Project – Planning Screenshots



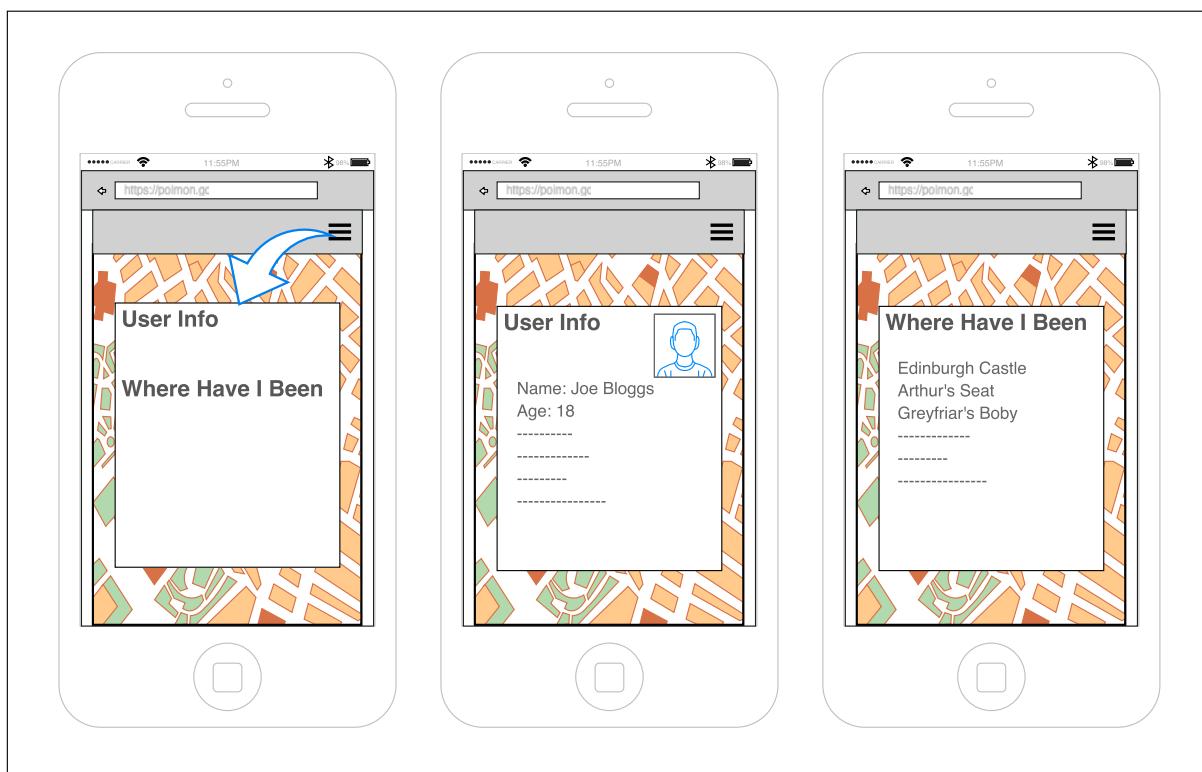
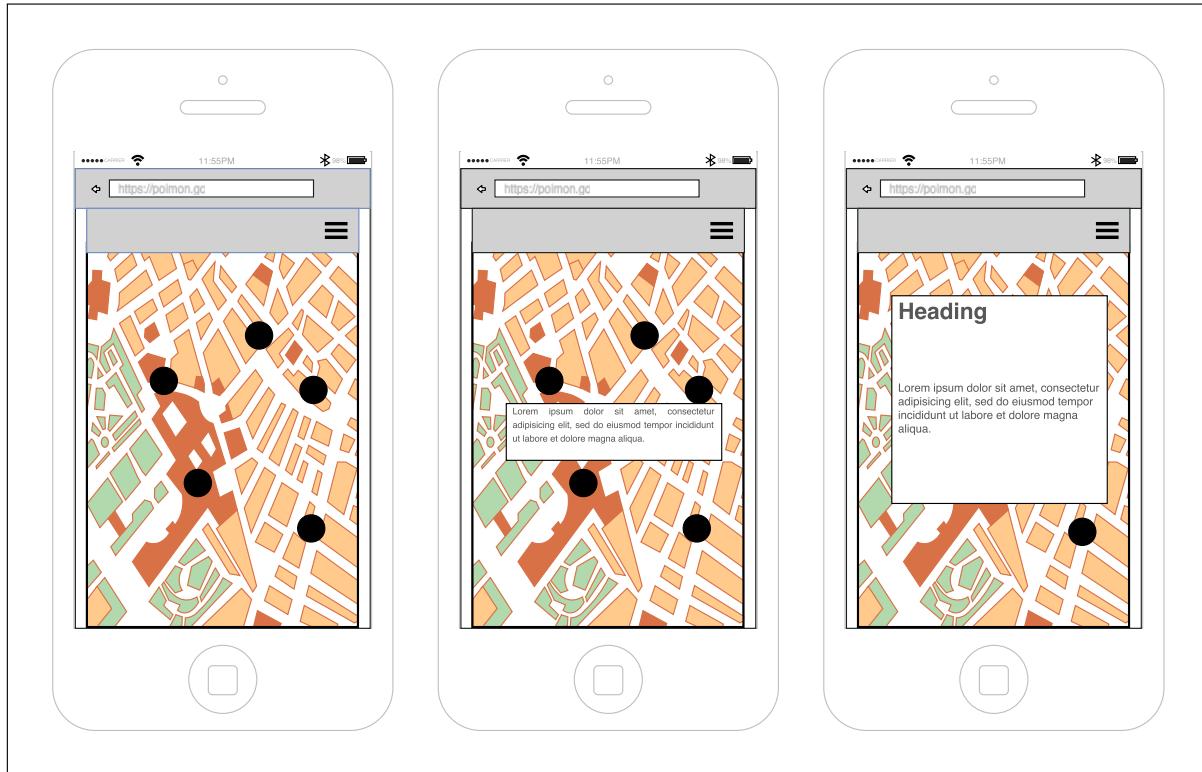
P4 – Acceptance Criteria and Test Plan

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 show 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 enter 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 enter 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 enter the new vendor	Pass

P5 – User Sitemap

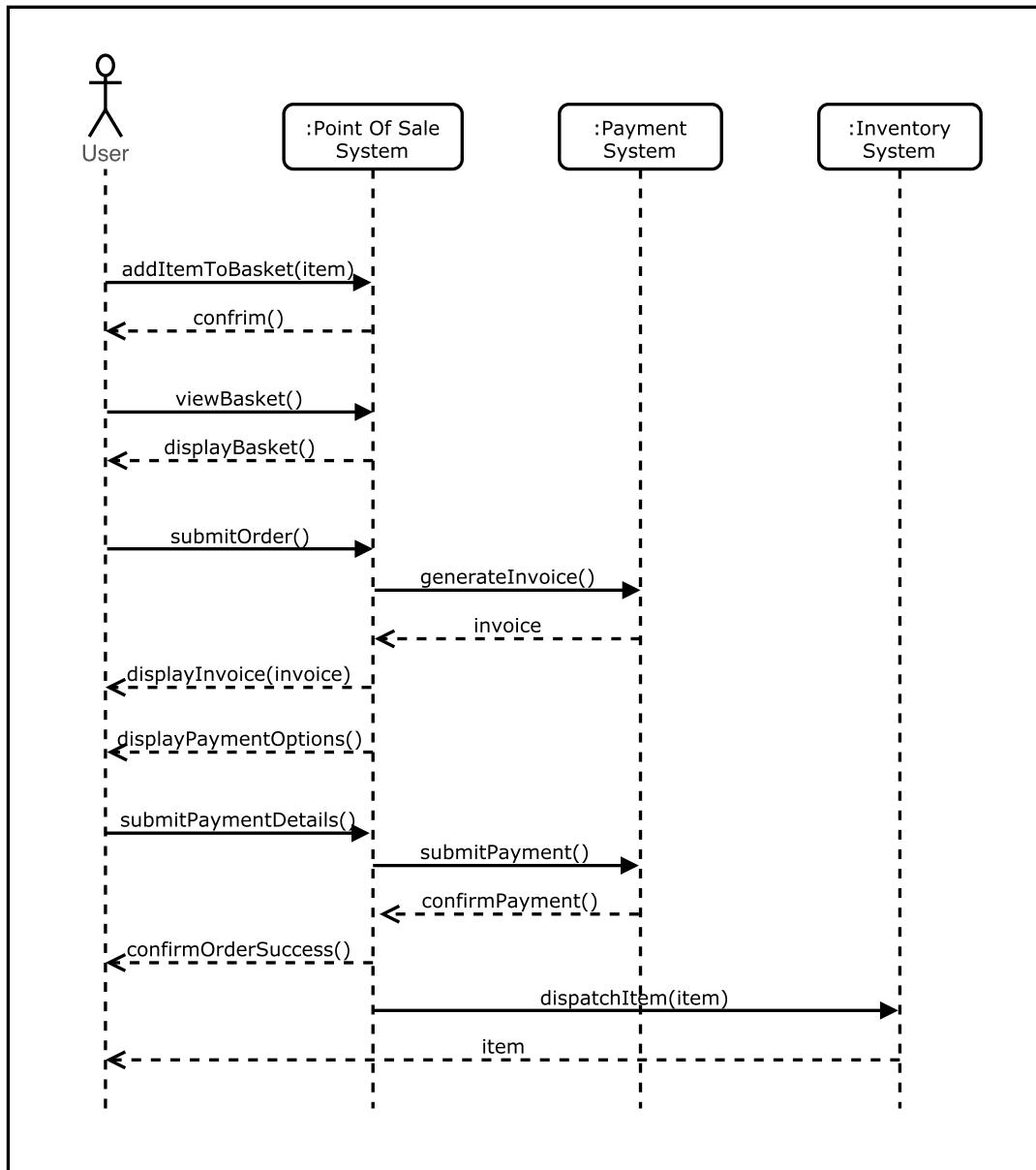


P6 – Wireframe Diagrams

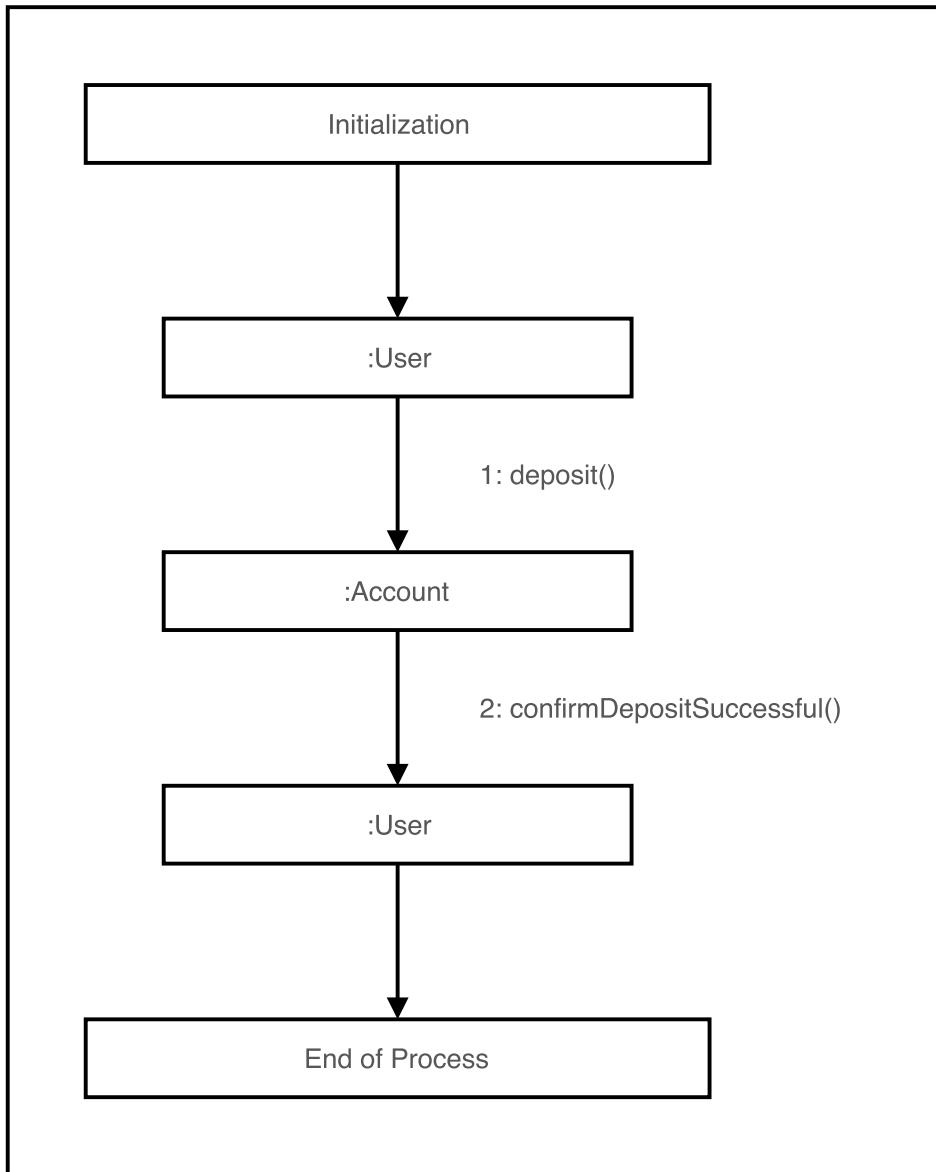


P7 – System Interaction Diagrams

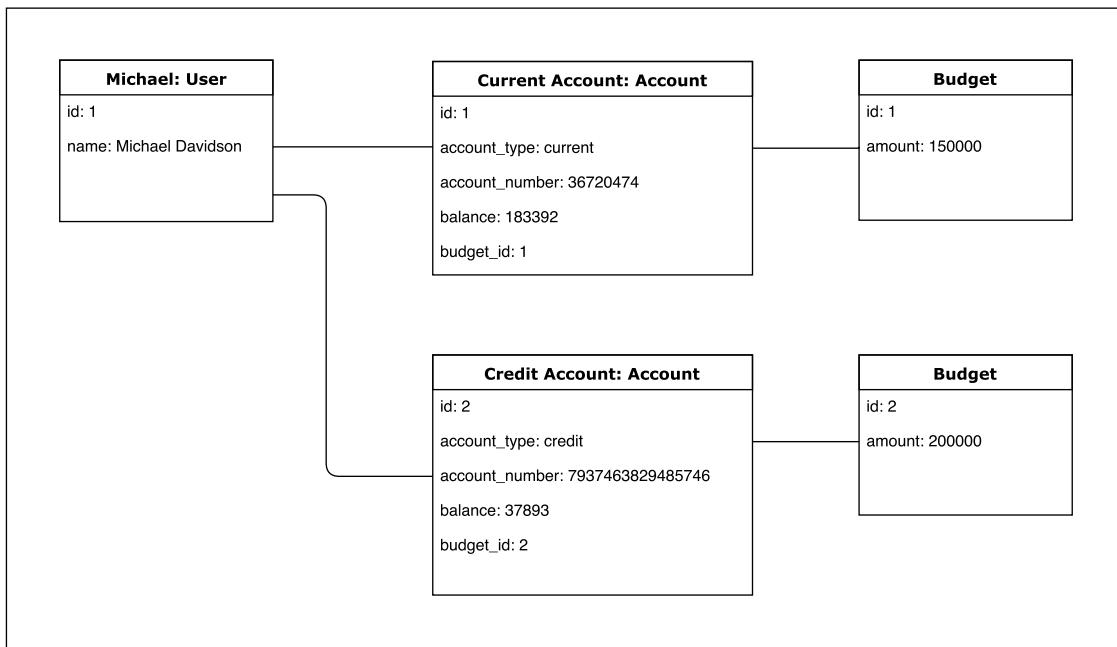
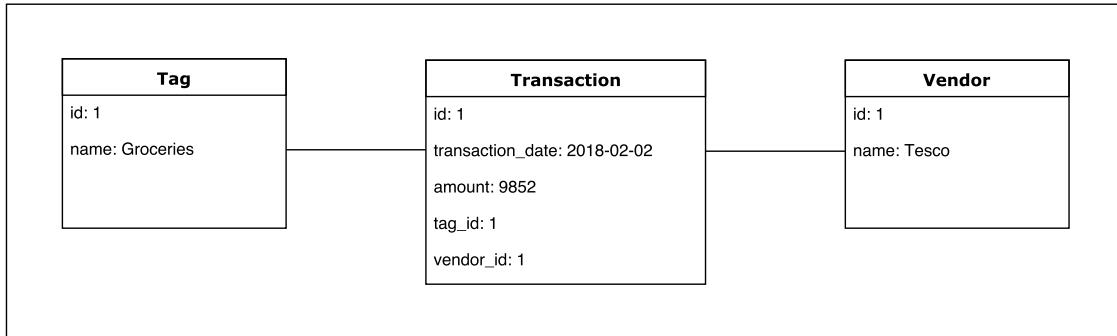
Sequence Diagram:



Collaboration Diagram:

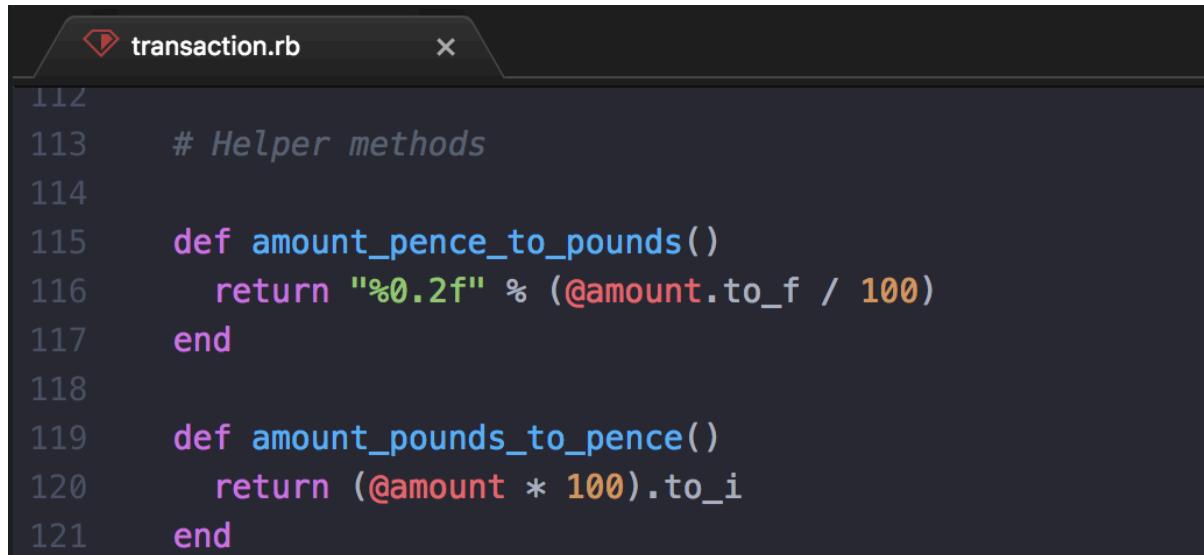


P8 – Object Diagrams



P9 – Algorithm Example Screenshots and Explanations

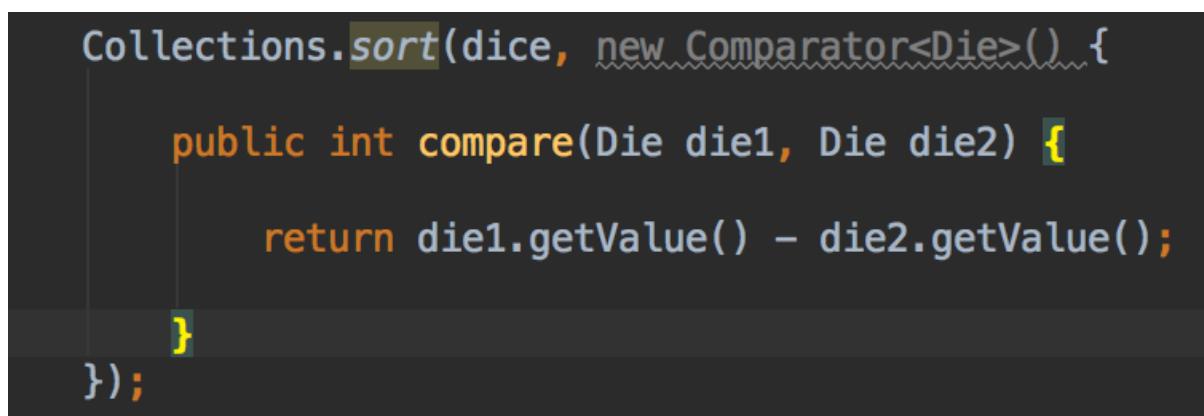
Algorithm Example 1:



```
transaction.rb
112
113      # Helper methods
114
115      def amount_pence_to_pounds()
116          return "%0.2f" % (@amount.to_f / 100)
117      end
118
119      def amount_pounds_to_pence()
120          return (@amount * 100).to_i
121      end
```

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:



```
Collections.sort(dice, new Comparator<Die>(){
    public int compare(Die die1, Die die2) {
        return die1.getValue() - die2.getValue();
    }
});
```

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

```
Player1 Turn # 1

Your scoresheet:

***** Upper Section *****
(1) Ones:           null
(2) Twos:           null
(3) Threes:         null
(4) Fours:          null
(5) Fives:          null
(6) Sixes:          null
***** Lower Section *****
(7) 3 of a Kind:   null
(8) 4 of a Kind:   null
(9) Full House:    null
(10) Small Straight: null
(11) Large Straight: null
(12) Yahtzee:       null
(13) Chance:        null
***** 

Roll # 1
You rolled:

4 2 5 4 3

Please indicate which dice you want to hold? e.g. type tftft or TFTFT to hold dice 1, 3 and 5
Just press Enter if you don't want to change the dice that are held
```

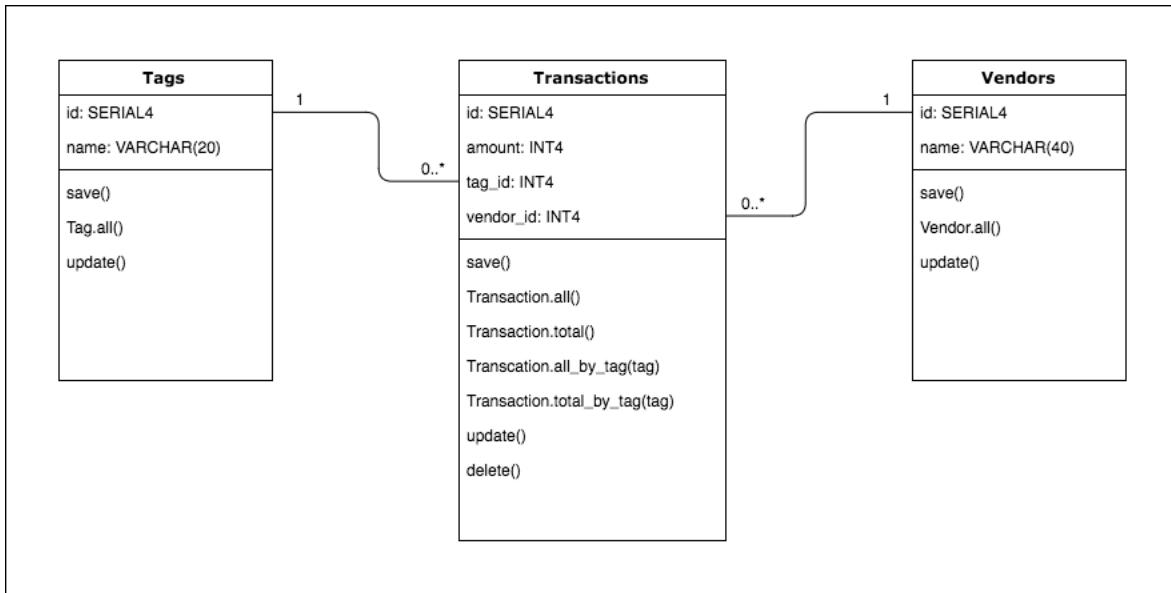
The screenshot shows a GitHub repository page for 'project2_week9_java_yahtzee_game'. The repository has 80 commits, 1 branch, 0 releases, and 1 contributor. The last commit was made 6 days ago by user 'davey-e'. The repository has 0 issues, 0 pull requests, 0 projects, and 0 wiki pages. There are buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'.

File	Description	Age
.gradle	Basic project structure created in IntelliJ	10 days ago
.idea	Basic project structure created in IntelliJ	10 days ago
gradle/wrapper	Basic project structure created in IntelliJ	10 days ago
src	Changes to GameRunner and ConsoleGameRunnerHelper to make things look...	6 days ago
.gitignore	Initial commit with .gitignore	10 days ago
Readme.md	Added Readme.md, which includes the project brief	10 days ago
build.gradle	Basic project structure created in IntelliJ	10 days ago
gradlew	Basic project structure created in IntelliJ	10 days ago
gradlew.bat	Basic project structure created in IntelliJ	10 days ago
settings.gradle	Basic project structure created in IntelliJ	10 days ago

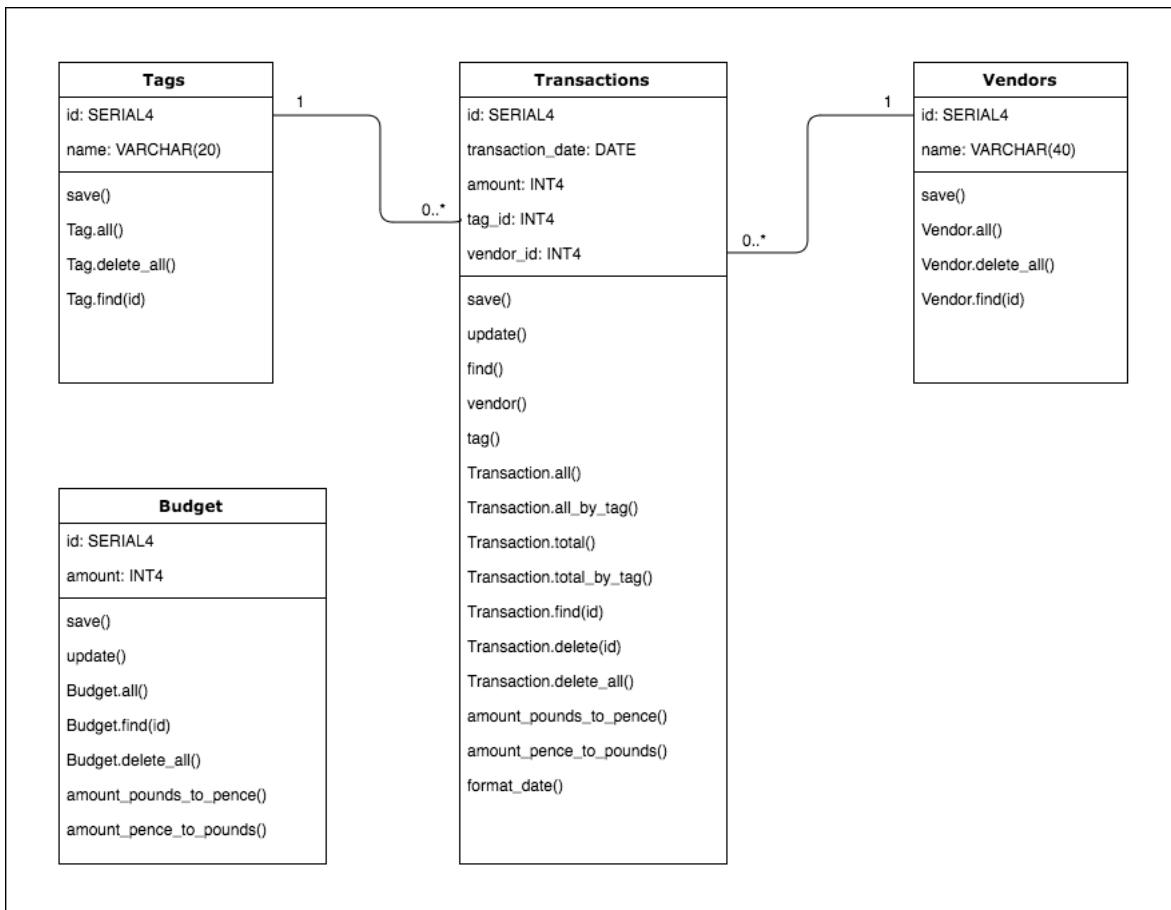
https://github.com/davey-e/project2_week9_java_yahtzee_game

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:

The wireframe shows a top navigation bar with a back/forward button and a URL field set to <http://localhost:4567>. Below this is a blue header bar with the title "Money Cashboard". The main area has a light blue background. On the left, there's a sidebar with "View Transactions" buttons for "All" and "By Tag". The main content area displays a table of recent transactions with columns for Date, Amount, Merchant, Type, and Tag.

Date	Amount	Merchant	Type	Tag
11/11/17	£12	Tesco	CC	Food
11/11/17	£3	Sainsburys	DC	Food
12/11/17	£5.99	MacDonalds	C	Food
13/11/17	£31	Sky	DD	TV

Below the sidebar, there are three buttons: "New Transaction", "New Account", and "New Tag".

Actual user interface at the end of the project:

The actual user interface has a dark header bar with the title "Money Cashboard" and a total amount of £1060.64. The sidebar on the left includes buttons for "All Transactions", "Transactions by Tag" (with categories like Bills, Clothes, Entertainment, Groceries, Presents), and "Edit Budget Amount". The main content area shows a table of transactions with columns for Date, Amount, Tag, Vendor, and "Transaction Details". The data in the table is more recent than in the wireframe.

Date	Amount	Tag	Vendor	Transaction Details
01/02/2018	£45.98	Groceries	Tesco	[...]
21/12/2017	£56.12	Presents	Amazon	[...]
20/12/2017	£190.45	Groceries	Tesco	[...]
17/12/2017	£33.10	Presents	Toys R US	[...]
17/12/2017	£44.56	Presents	Amazon	[...]
16/12/2017	£9.77	Groceries	Asda	[...]
15/12/2017	£23.45	Entertainment	Vue Cinema	[...]
15/12/2017	£34.00	Clothes	Next	[...]
11/12/2017	£33.78	Groceries	Tesco	[...]
10/12/2017	£109.23	Bills	Eon	[...]
10/12/2017	£14.98	Groceries	Coop	[...]
04/12/2017	£34.78	Presents	Toys R US	[...]
02/12/2017	£2.56	Groceries	Coop	[...]

Trello board at the start of the project:

The Trello board has the following structure:

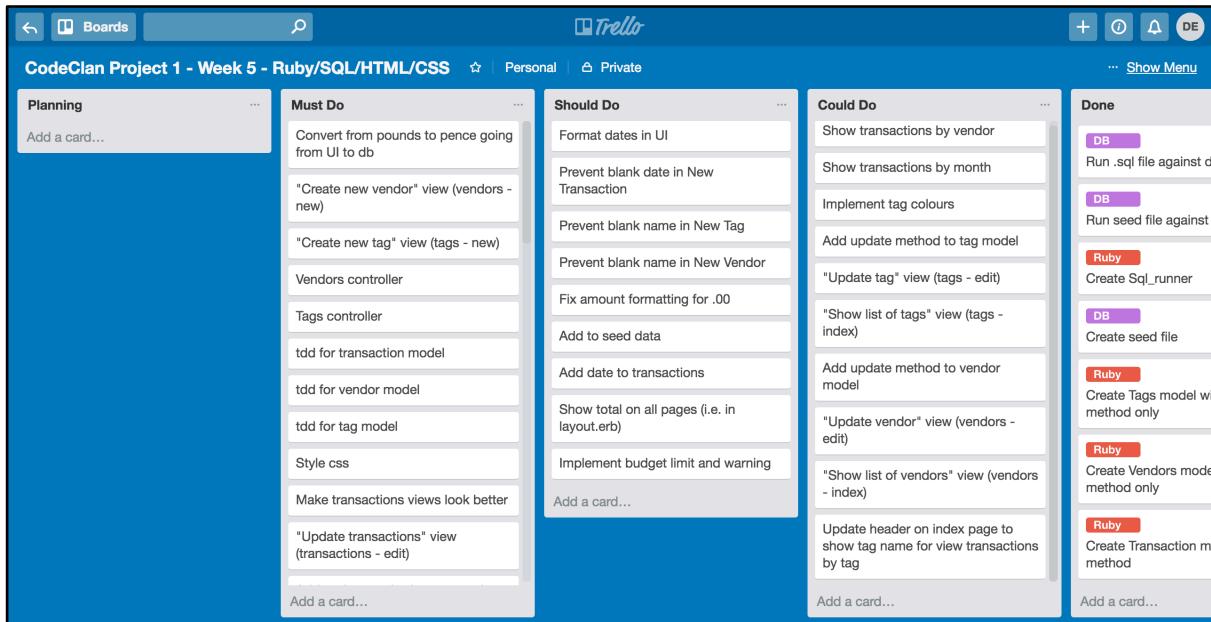
- Planning** column:
 - General: Document MVP and intended extensions (1 card)
 - General: Create Git repository in GitHub and submit link via homework form
 - Front End: Create proto-persona
 - Front End: Create user requirements
 - Front End: Create user journeys
 - Front End: Create UI sketches (wireframes)
 - Back End: Create class diagrams
- Must Do** column:
 - DB: Create database
 - DB: Create .sql file to generate db structure
 - DB: Run .sql file against db
 - DB: Run seed file against db
 - Ruby: Create Sql_runner
 - DB: Create seed file
 - Ruby: Create Tags model with save method only
 - Ruby: Add a card...
- Should Do** column:
 - Format dates in UI
 - Prevent blank date in New Transaction
 - Prevent blank name in New Tag
 - Prevent blank name in New Vendor
 - Fix amount formatting for .00
 - Add to seed data
 - Add date to transactions
 - Show total on all pages (i.e. in layout.erb)
 - Implement budget limit and warning
 - Add a card...
- Could Do** column:
 - Show transactions by vendor
 - Show transactions by month
 - Implement tag colours
 - Add update method to tag model
 - "Update tag" view (tags - edit)
 - "Show list of tags" view (tags - index)
 - Add update method to vendor model
 - "Update vendor" view (vendors - edit)
 - "Show list of vendors" view (vendors - index)
 - Update header on index page to show tag name for view transactions by tag
 - Add a card...
- Done** column:
 - General: Document MVP and intended extensions (1 card)
 - Front End: Create user journeys
 - Front End: Create UI sketches (wireframes)
 - Front End: Create user requirements
 - Front End: Create proto-persona
 - Back End: Create class diagrams
 - General: Create Git repository and submit link via homework form

Trello board after planning items complete:

The Trello board has the following structure after planning items are completed:

- Planning** column:
 - Add a card...
- Must Do** column:
 - DB: Create database
 - DB: Create .sql file to generate db structure
 - DB: Run .sql file against db
 - DB: Run seed file against db
 - Ruby: Create Sql_runner
 - DB: Create seed file
 - Ruby: Create Tags model with save method only
 - Ruby: Add a card...
- Should Do** column:
 - Format dates in UI
 - Prevent blank date in New Transaction
 - Prevent blank name in New Tag
 - Prevent blank name in New Vendor
 - Fix amount formatting for .00
 - Add to seed data
 - Add date to transactions
 - Show total on all pages (i.e. in layout.erb)
 - Implement budget limit and warning
 - Add a card...
- Could Do** column:
 - Show transactions by vendor
 - Show transactions by month
 - Implement tag colours
 - Add update method to tag model
 - "Update tag" view (tags - edit)
 - "Show list of tags" view (tags - index)
 - Add update method to vendor model
 - "Update vendor" view (vendors - edit)
 - "Show list of vendors" view (vendors - index)
 - Update header on index page to show tag name for view transactions by tag
 - Add a card...
- Done** column:
 - General: Document MVP and intended extensions (1 card)
 - Front End: Create user journeys
 - Front End: Create UI sketches (wireframes)
 - Front End: Create user requirements
 - Front End: Create proto-persona
 - Back End: Create class diagrams
 - General: Create Git repository and submit link via homework form

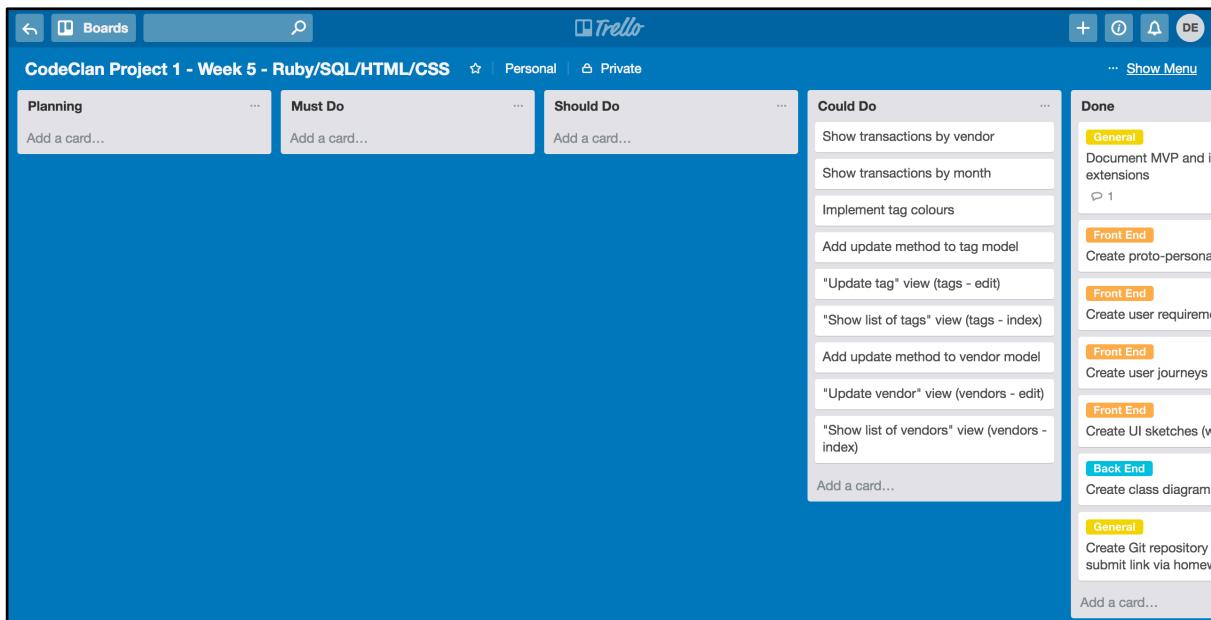
Trello board after basic code structure items complete:



A screenshot of a Trello board titled "CodeClan Project 1 - Week 5 - Ruby/SQL/HTML/CSS". The board has four main sections: Planning, Must Do, Should Do, Could Do, and Done.

- Planning:** Add a card...
- Must Do:**
 - Convert from pounds to pence going from UI to db
 - "Create new vendor" view (vendors - new)
 - "Create new tag" view (tags - new)
 - Vendors controller
 - Tags controller
 - tdd for transaction model
 - tdd for vendor model
 - tdd for tag model
 - Style css
 - Make transactions views look better
 - "Update transactions" view (transactions - edit)
- Should Do:**
 - Add a card...
- Could Do:**
 - Show transactions by vendor
 - Show transactions by month
 - Implement tag colours
 - Add update method to tag model
 - "Update tag" view (tags - edit)
 - "Show list of tags" view (tags - index)
 - Add update method to vendor model
 - "Update vendor" view (vendors - edit)
 - "Show list of vendors" view (vendors - index)
 - Update header on index page to show tag name for view transactions by tag
- Done:**
 - DB**
 - Run .sql file against db
 - DB**
 - Run seed file against db
 - Ruby**
 - Create Sql_runner
 - DB**
 - Create seed file
 - Ruby**
 - Create Tags model w/ method only
 - Ruby**
 - Create Vendors model w/ method only
 - Ruby**
 - Create Transaction model w/ method

Trello board at the end of the project:



A screenshot of the same Trello board at the end of the project. The board has the same sections: Planning, Must Do, Should Do, Could Do, and Done.

- Planning:** Add a card...
- Must Do:** Add a card...
- Should Do:** Add a card...
- Could Do:**
 - Show transactions by vendor
 - Show transactions by month
 - Implement tag colours
 - Add update method to tag model
 - "Update tag" view (tags - edit)
 - "Show list of tags" view (tags - index)
 - Add update method to vendor model
 - "Update vendor" view (vendors - edit)
 - "Show list of vendors" view (vendors - index)
 - Add a card...
- Done:**
 - General**
 - Document MVP and its extensions
 - Front End**
 - Create proto-personas
 - Front End**
 - Create user requirements
 - Front End**
 - Create user journeys
 - Front End**
 - Create UI sketches (Wireframes)
 - Back End**
 - Create class diagram
 - General**
 - Create Git repository submit link via homeview

P13 – User Input Processing Screenshots

User input:

£ Money Cashboard £1014.66

New Transaction
Please enter the details of the new transaction:

Date:

Amount:

Tag:

Vendor:

Submit

All Transactions
Transactions by Tag:

User input saved:

£ Money Cashboard £1060.64

All Transactions
Total: £1060.64

Date	Amount	Tag	Vendor	Transaction Details
01/02/2018	£45.98	Groceries	Tesco	<input type="button" value="..."/>
21/12/2017	£56.12	Presents	Amazon	<input type="button" value="..."/>
20/12/2017	£190.45	Groceries	Tesco	<input type="button" value="..."/>
17/12/2017	£33.10	Presents	Toys R US	<input type="button" value="..."/>
17/12/2017	£44.56	Presents	Amazon	<input type="button" value="..."/>
16/12/2017	£9.77	Groceries	Asda	<input type="button" value="..."/>
15/12/2017	£23.45	Entertainment	Vue Cinema	<input type="button" value="..."/>
15/12/2017	£34.00	Clothes	Next	<input type="button" value="..."/>
11/12/2017	£33.78	Groceries	Tesco	<input type="button" value="..."/>
10/12/2017	£109.23	Bills	Eon	<input type="button" value="..."/>
10/12/2017	£14.98	Groceries	Coop	<input type="button" value="..."/>
04/12/2017	£34.78	Presents	Toys R US	<input type="button" value="..."/>
02/12/2017	£2.56	Groceries	Coop	<input type="button" value="..."/>

All Transactions
Transactions by Tag:

P14 – Data Interaction and Persistence Screenshots

User Input:

The screenshot shows the Money Cashboard application interface. At the top, it displays a balance of £1060.64. On the left, there's a sidebar with buttons for 'All Transactions', 'Transactions by Tag' (with sub-options like Bills, Clothes, Entertainment, Groceries, Presents), and links for 'New Transaction', 'New Tag', 'New Vendor', and 'Edit Budget Amount'. The main area is titled 'New Transaction' and contains fields for Date (02/02/2018), Amount (98.52), Tag (Bills), and Vendor (BT). A 'Submit' button is at the bottom of the form.

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:

All Transactions				
Transactions by Tag:				
<button>Bills</button> <button>Clothes</button> <button>Entertainment</button> <button>Groceries</button> <button>Presents</button>				
Total: £1159.16				
Date	Amount	Tag	Vendor	Transaction Details
02/02/2018	£98.52	Bills	BT	[...]
01/02/2018	£45.98	Groceries	Tesco	[...]
21/12/2017	£56.12	Presents	Amazon	[...]
20/12/2017	£190.45	Groceries	Tesco	[...]
17/12/2017	£33.10	Presents	Toys R US	[...]
17/12/2017	£44.56	Presents	Amazon	[...]
16/12/2017	£9.77	Groceries	Asda	[...]
15/12/2017	£23.45	Entertainment	Vue Cinema	[...]
15/12/2017	£34.00	Clothes	Next	[...]
11/12/2017	£33.78	Groceries	Tesco	[...]
10/12/2017	£14.98	Groceries	Coop	[...]
10/12/2017	£109.23	Bills	Eon	[...]
	£34.78	Presents	Toys R US	[...]

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

All Transactions				
Transactions by Tag:				
<button>Bills</button> <button>Clothes</button> <button>Entertainment</button> <button>Groceries</button> <button>Presents</button>				
Total: £384.02				
Date	Amount	Tag	Vendor	Transaction Details
02/02/2018	£98.52	Bills	BT	[...]
10/12/2017	£109.23	Bills	Eon	[...]
23/11/2017	£176.27	Bills	BT	[...]

P16 – API Usage Screenshots

```
JSX index.js x
1 const app = function(){
2     url = "https://restcountries.eu/rest/v2/all";
3
4     // display last selected country
5     const countryObject = getCountryFromLocalStorage();
6     if(countryObject !== null){
7         createCountryDetailList(countryObject);
8     }
9
10    // get data from api
11    makeRequest(url, requestComplete);
12 }
13
14 // get data from api
15 const makeRequest = function(url, callback){
16     const request = new XMLHttpRequest();
17     request.open("GET", url);
18     request.addEventListener("load", callback);
19     request.send();
20 }
21
22 // callback used in makeRequest to convert api data into array of country objects
23 const requestComplete = function(){
24     if(this.status !== 200) return;
25     const jsonString = this.responseText; //This is a string
26     const countries = JSON.parse(jsonString); //This parses the string into a js object
27     populateList(countries);
28 }
```

Countries

Afghanistan



- Country Name: Afghanistan
- Population: 27657145
- Capital City: Kabul

P17 – Bug Tracking Report

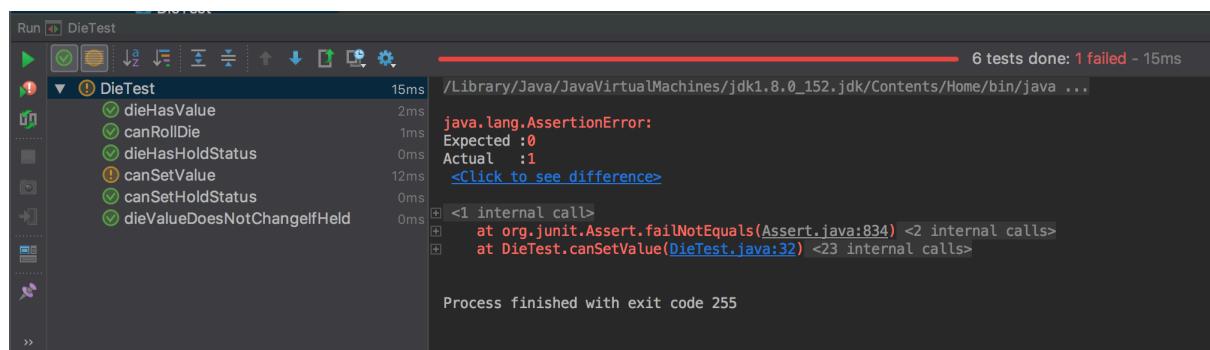
Issue	Status	Bug Fix	Status After Fix
User can see 5 random points of interest from a 500m radius of their current location	Failed	Added a callback to the javascript code to make sure that points of interest are retrieved from the Sygic Travel api before randomly selecting and displaying 5 points of 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 be 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:

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

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

Tests passing:

Test	Time
dieHasValue	24ms
canRollDie	42ms
dieHasHoldStatus	0ms
canSetValue	0ms
canSetHoldStatus	1ms
dieValueDoesNotChangeIfHeld	0ms