Week 2 / IT 5

```ruby
class River

def initialize(name)
  @name = name
  @fish = []
end

def name()
  return @name
end

def fish()
  return @fish
end

def get_num_fish()
  return @fish.count()
end

def add_fish(fish)
  @fish.push(fish)
end

def remove_fish(fish)
  if @fish.include?(fish)
    @fish.delete(fish)
  end
end

end
```

This exercise demonstrates the use of an array in a program. The *fish* array is defined in the *initialize* method. The *get_num_fish* function returns the result of the array.count() method as applied to the *fish* array.

```
irb(main):031:0> myRiver = River.new('testRiver')
=> #<River:0x00000001344d28 @name="testRiver", @fish=[]>
irb(main):032:0> myRiver.get_num_fish()
=> 0
irb(main):033:0>
```

This demonstrates the result of running the *get_all_fish()* method, which returns 0 as no fish have been added to the collection yet.

Week 2 / IT 6

```
def add_book_by_title(book_title)
  new_book = {
    title: book_title,
    rental_details: {
     student_name: "",
     date: ""
      }
    }
  @books.push(new_book)
end
```

This function demonstrates the use of a hash. In this case, the *new_book* hash is created and populated with the *book_title* parameter passed into the function, and a second hash called *rental_details*. The *new_book* hash is added to the end of the *books* collection in the last line of the function.

```
irb(main):062:0> myLibrary = Library.new([])
=> #<Library:0x000000017969d0 @books=[]>
irb(main):063:0> myLibrary.add_book_by_title('testTitle')
=> [{:title=>"testTitle", :rental_details=>{:student_name=>"", :date=>""}}]
irb(main):064:0>
```

This demonstrates the result of creating a new *Library* object, then running the *add_book_by_title()* function and passing in the string value ‘testTitle’.

Week 3 / IT 3

```ruby
def tickets()
  sql = "SELECT * FROM tickets WHERE tickets.customer_id = #{@id}"
  result = SqlRunner.run(sql)
  tickets = result.map { |ticket| Ticket.new(ticket) }
  return tickets
end
```

This function demonstrates searching data in a program. In this case, the function returns the result of searching a collection named *tickets* for all elements which contain a *customer_id* property which matches the search term.

```ruby
def screenings()
  sql = "SELECT * FROM tickets INNER JOIN screenings ON screenings.id = tickets.screening_id WHERE
    tickets.customer_id = #{@id};"
  result = SqlRunner.run(sql)

  if result == nil then return nil end

  return result.map { |screening| Screening.new( {
    'id' => screening['screening_id'],
    'film_id' => screening['film_id'],
    'start_time' => screening['start_time'],
    'capacity' => screening['capacity'] } ) }
end
```

This function demonstrates searching data in a program. In this case, the function returns a collection of *Screening* objects. The properties of each *Screening* object are populated using the results of searching the *tickets* table for all elements with a *customer_id* which matches the search term, then performing an inner join to retrieve further details from the *screenings* table.

```
[1] pry(main)> customer1.tickets()
=> [#<Ticket:0x007fc5a3b11c80 @customer_id=91, @id=175, @screening_id=259>,
 #<Ticket:0x007fc5a3b11b90 @customer_id=91, @id=176, @screening_id=259>,
 #<Ticket:0x007fc5a3b11ac8 @customer_id=91, @id=177, @screening_id=261>]
[2] pry(main)> customer1.screenings()
=> [#<Screening:0x007fc5a3a78c88
  @capacity=3,
  @film_id=91,
  @id=259,
  @start_time=#<DateTime: 2017-06-04T22:52:23+00:00 ((2457909j,82343s,0n),+0s,2299161j)>>,
 #<Screening:0x007fc5a3a78648
  @capacity=3,
  @film_id=91,
  @id=259,
  @start_time=#<DateTime: 2017-06-04T22:52:23+00:00 ((2457909j,82343s,0n),+0s,2299161j)>>,
 #<Screening:0x007fc5a3a780d0
  @capacity=5,
  @film_id=91,
  @id=261,
  @start_time=#<DateTime: 2017-06-06T22:52:23+00:00 ((2457911j,82343s,0n),+0s,2299161j)>>]
[3] pry(main)>
```

This demonstrates the result of the search functions running.

Week 3 / IT 4

customer.rb

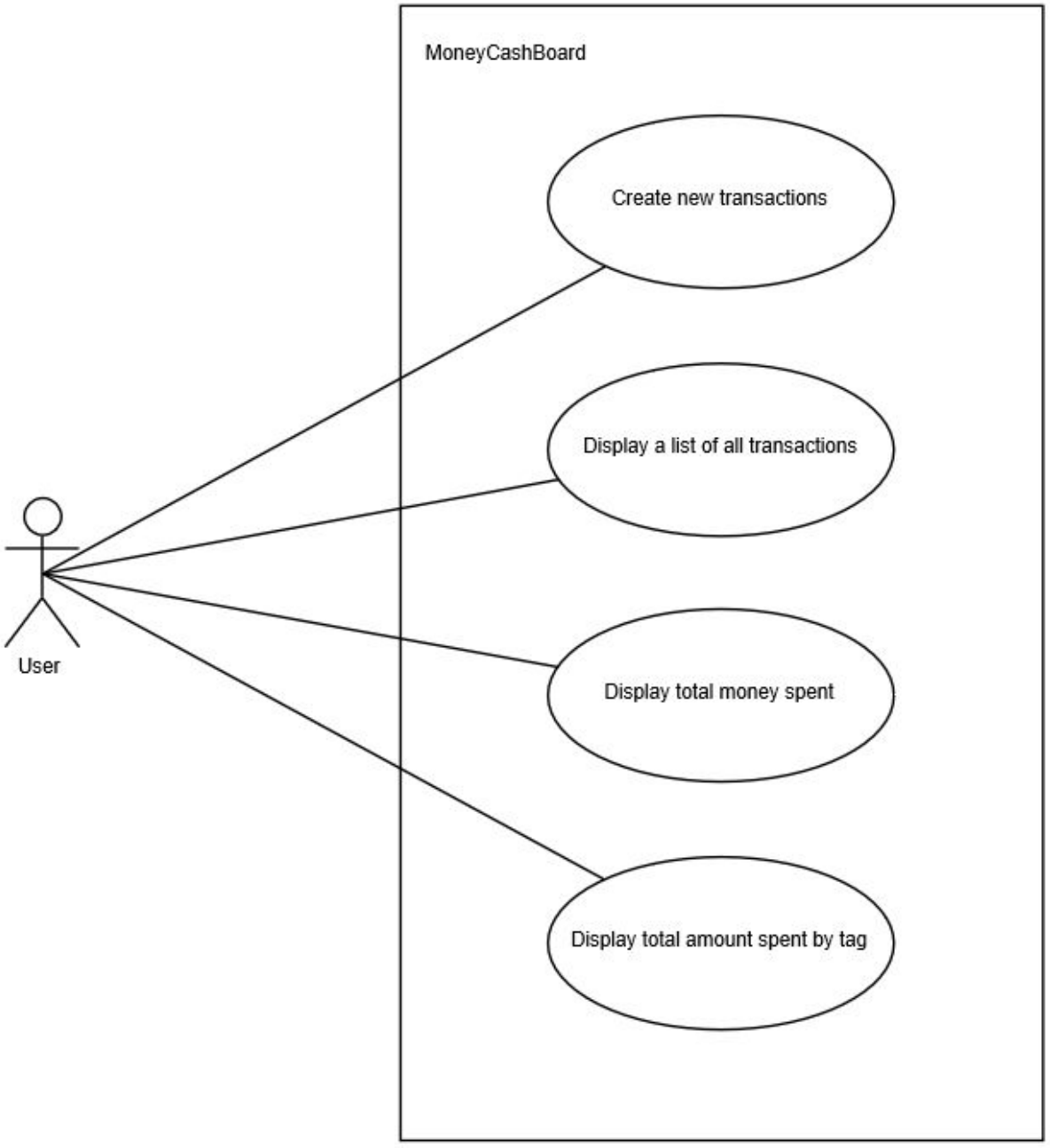
```ruby
require 'pg'
require_relative '../db/sql_runner'

class Customer

  attr_reader :id, :name

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

  def self.all()
    sql = "SELECT * FROM customers ORDER BY name;"
    customers = SqlRunner.run(sql)
    return customers.map { |customer| Customer.new(customer) }
  end
```

This demonstrates sorting in a program. A SQL query is used to retrieve all records from the *customer* table sorted into ascending order using the ORDER BY statement.

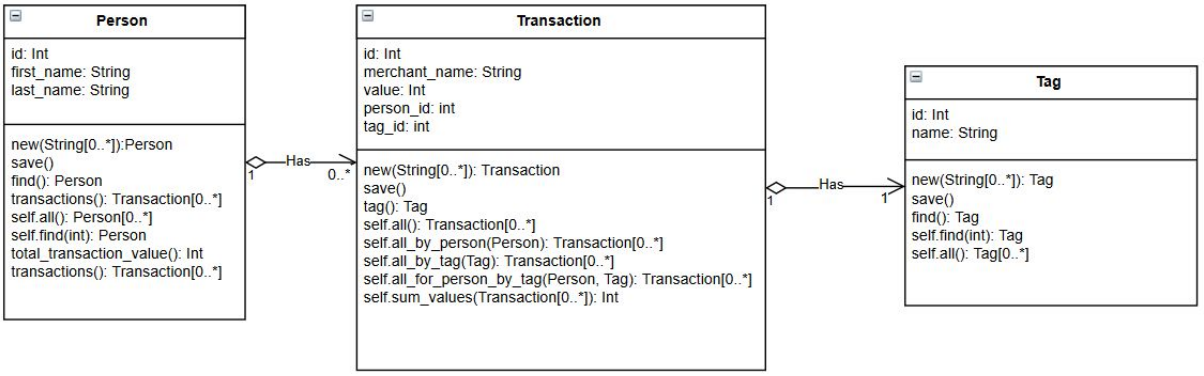
```
[1] pry(main)> Customer.all()
=> [#<Customer:0x007f91348a2100 @id=22, @name="Name 1">,
 #<Customer:0x007f91348a1f70 @id=23, @name="Name 2">]
[2] pry(main)>
```

This demonstrates the result of the function.

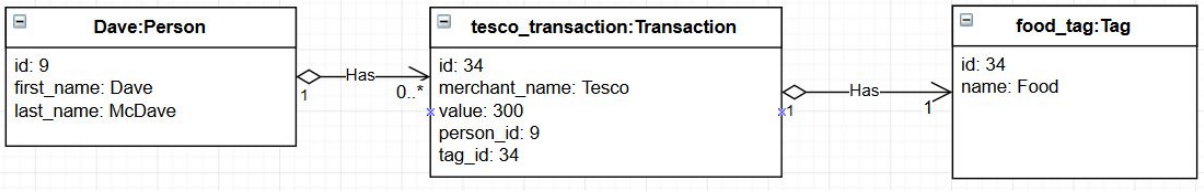
Week 5 / AD 1 / Use Case Diagram



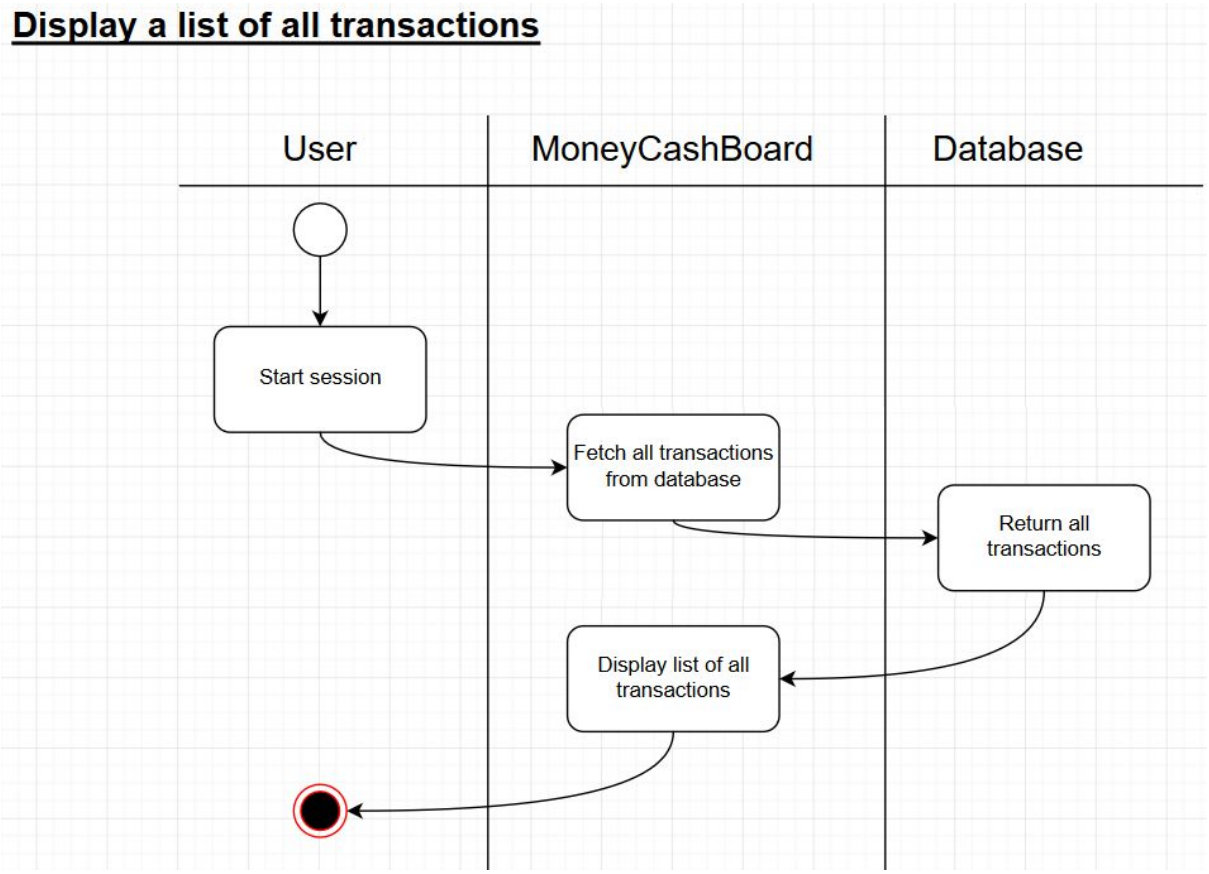
Week 5 / AD 2 / Class Diagram



Week 5 / AD 3 / Object Diagram



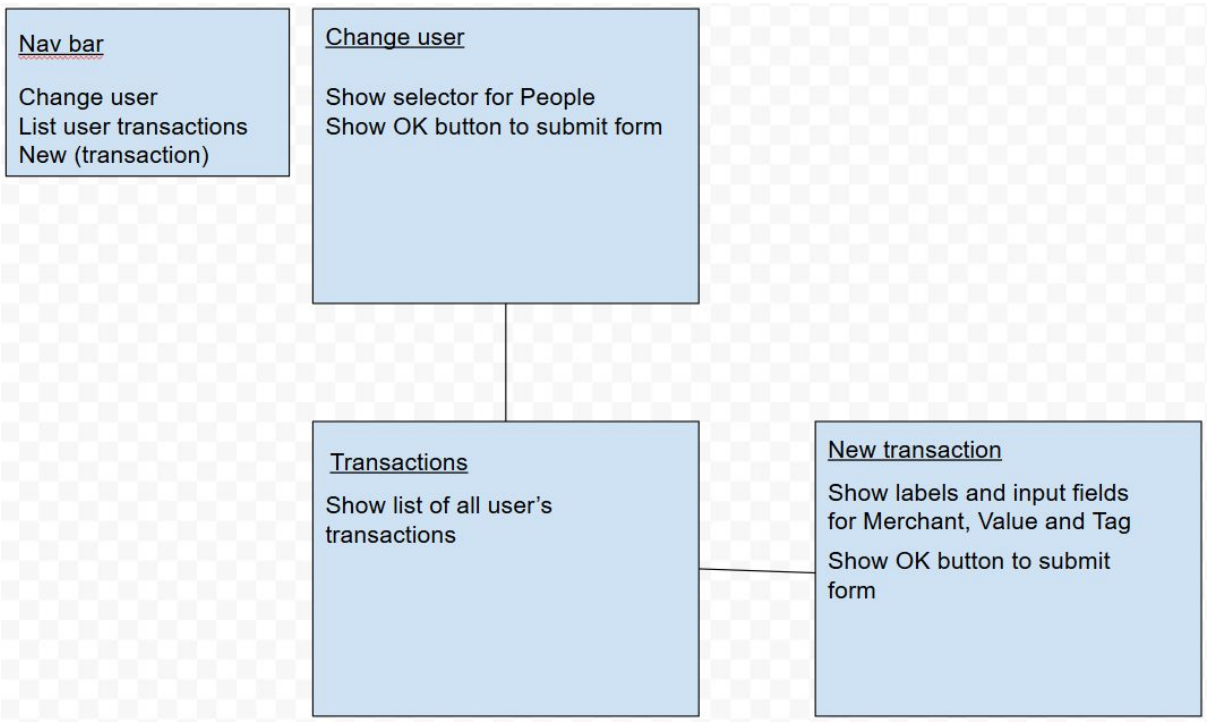
Week 5 / AD 4 / Activity Diagram



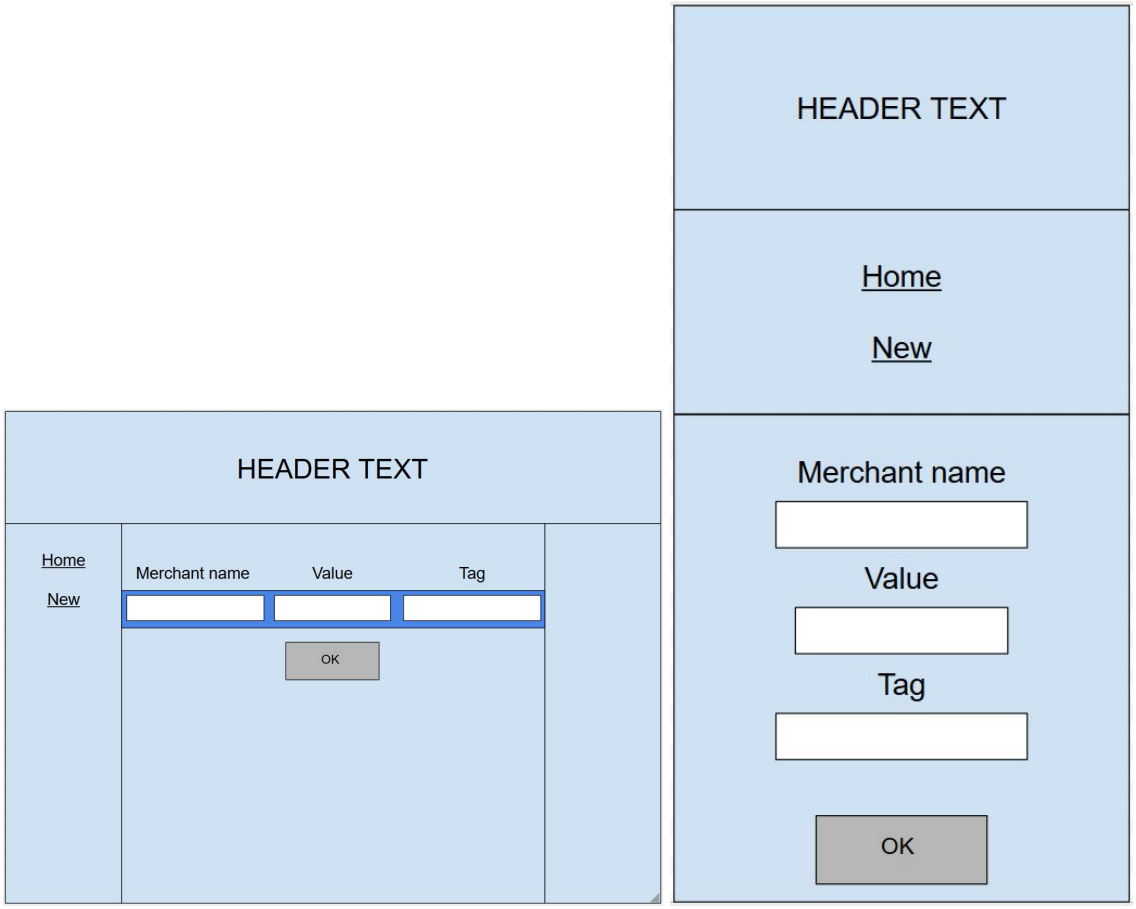
Week 5 / AD 6 / Implementations Constraints Plan

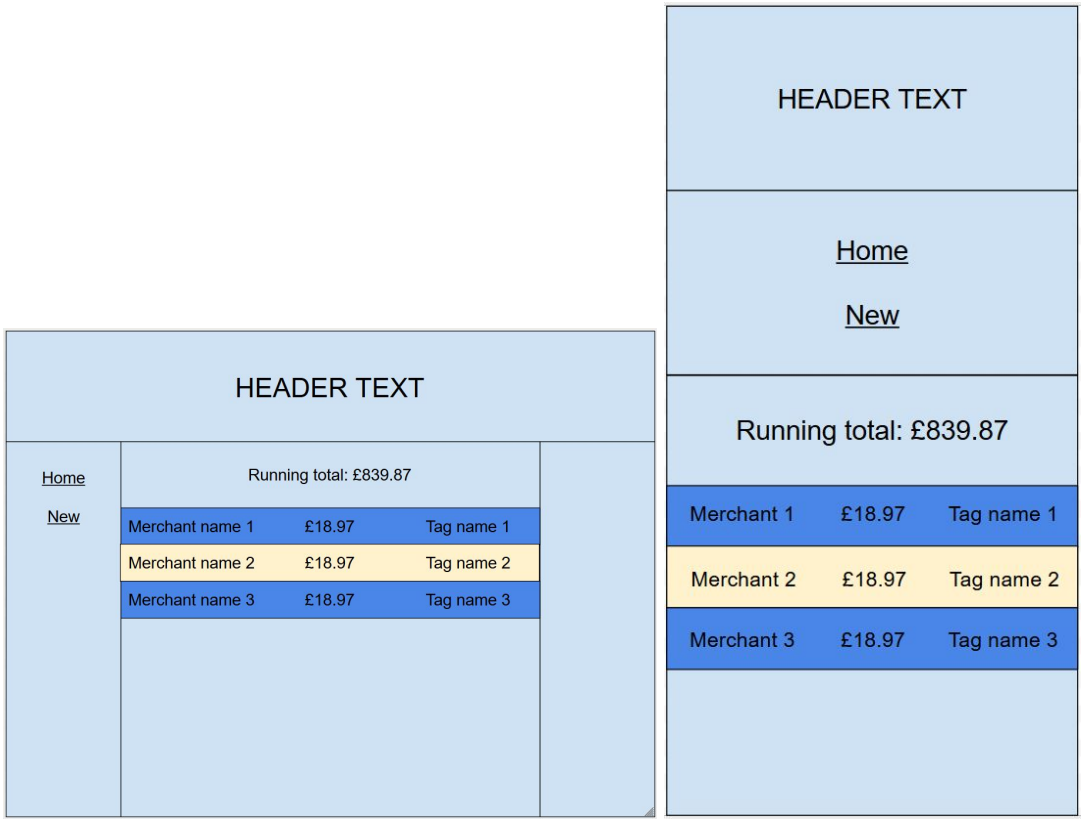
Constraint	**Possible Effect of Constraint on Product**	**Solution**
Hardware and software platforms	Poor browser compatibility may lead to reduced or unusable feature set.	Test cross-browser compatibility on all web pages.
	Technical restrictions (no authentication, no javascript, etc.) may lead to reduced /non-standard functionality.	Research and implement best available alternatives to current standard practices. Reduce system functionality where necessary.
Performance requirements	Under-performant client machine may negatively impact on user experience.	Measure system requirements and publish them in user documentation.
Persistent storage and transactions	Insufficient database storage will lead to failure.	Measure storage requirements based on limited data set and extrapolate future storage requirements accordingly.
	Insufficiently resourced (CPU/RAM/etc.) database server will underperform and/or fail under load.	Measure database server load under estimated normal conditions and extrapolate requirements accordingly.
	Large transaction volume will impact system responsiveness.	Estimate and/or test impact of simultaneous database transactions and ensure that database engine copes with above average load.
Usability	Lack of visual impairment support (high contrast, readable font sizes, screen reader cues, etc.) may negatively impact some users.	Design client-facing elements with W3C accessibility guidelines (https://www.w3.org/standards/webdesign/accessibility) in mind.
Budgets	N/A	N/A
Time limitations	Project must be completed and submitted before 12:00 on 14/06/2017.	Manage time effectively, ensure that all assets are completed to schedule.

Week 5 / P 5 / User Sitemap



Week 5 / P 6 / Wireframe Designs (desktop/mobile versions)



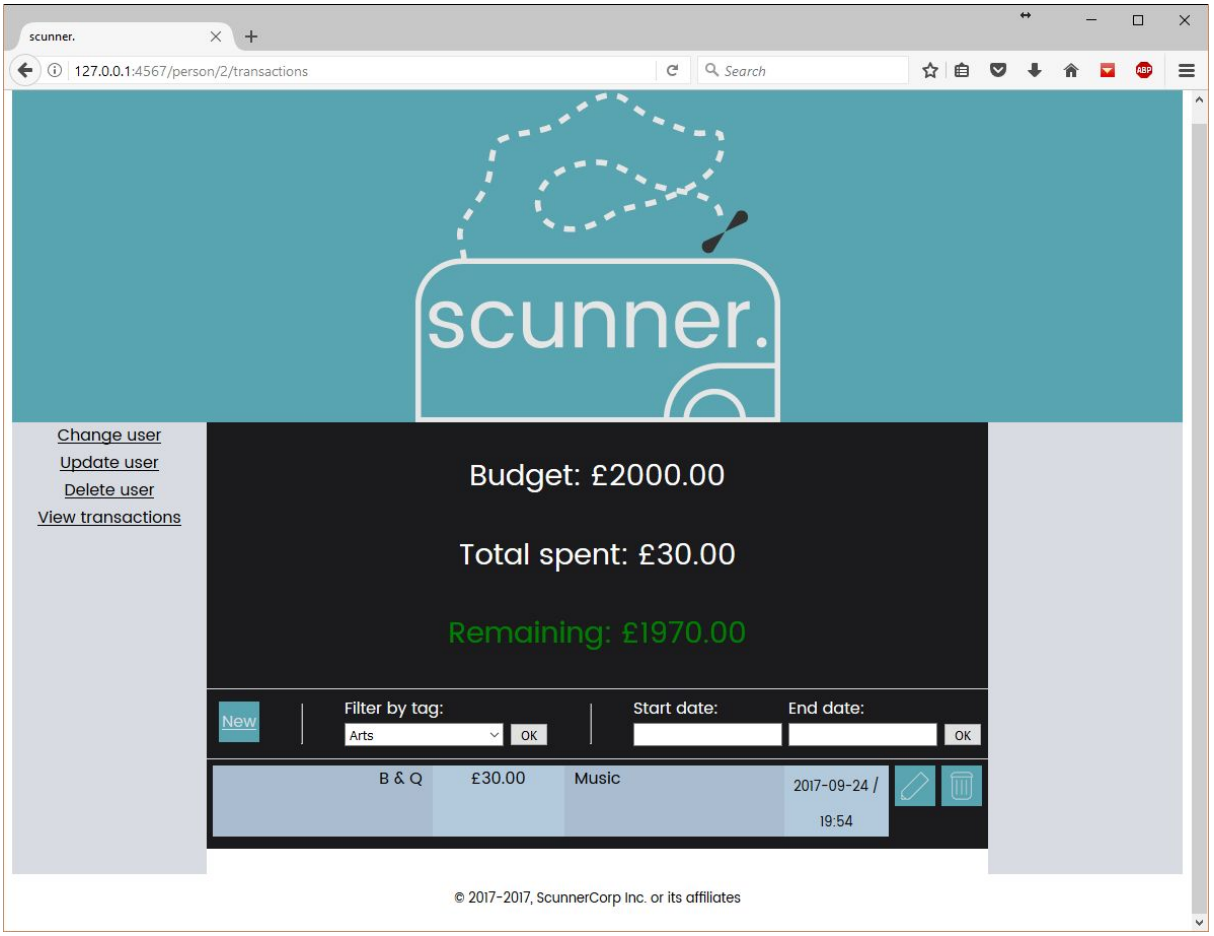


Week 5 / P 10 / Pseudocode for a function

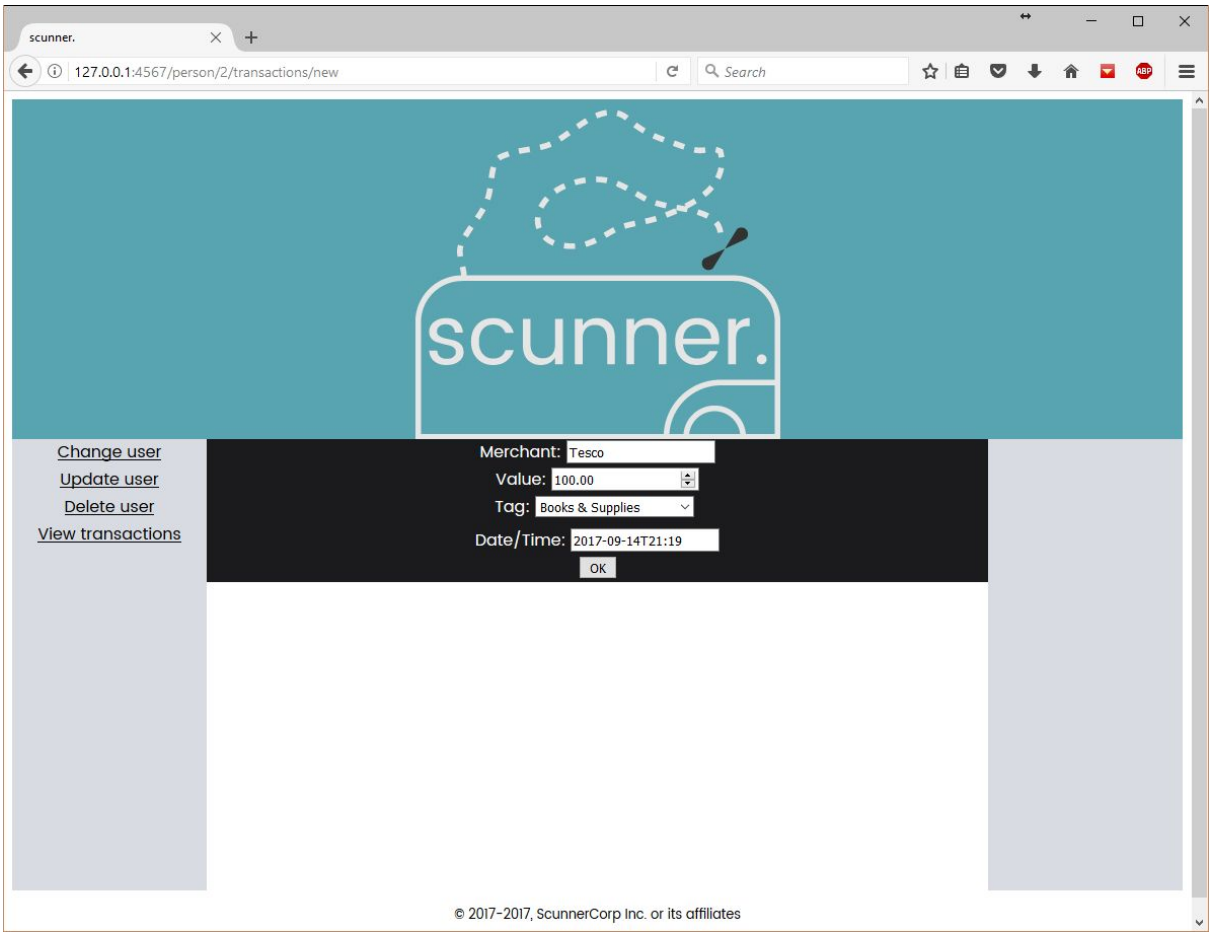
```
find_person(person_id_to_find)
    myQuery = select all from people where person.person_id = person_id_to_find
    result = run(myQuery)
    return new Person(result.first)
end
```

Week 5 / P 13 & P 14 / User input

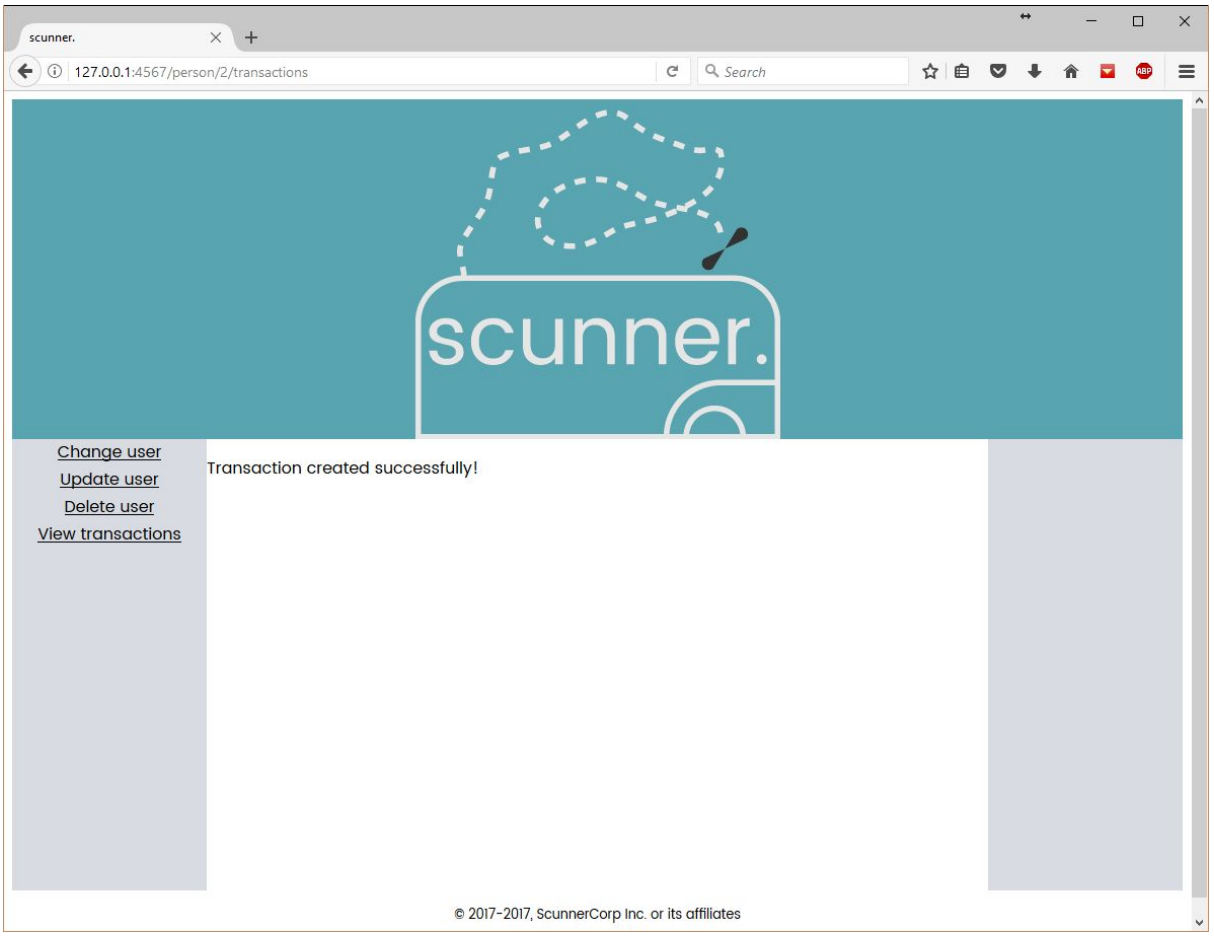
Screenshot of application status before user has created a new transaction



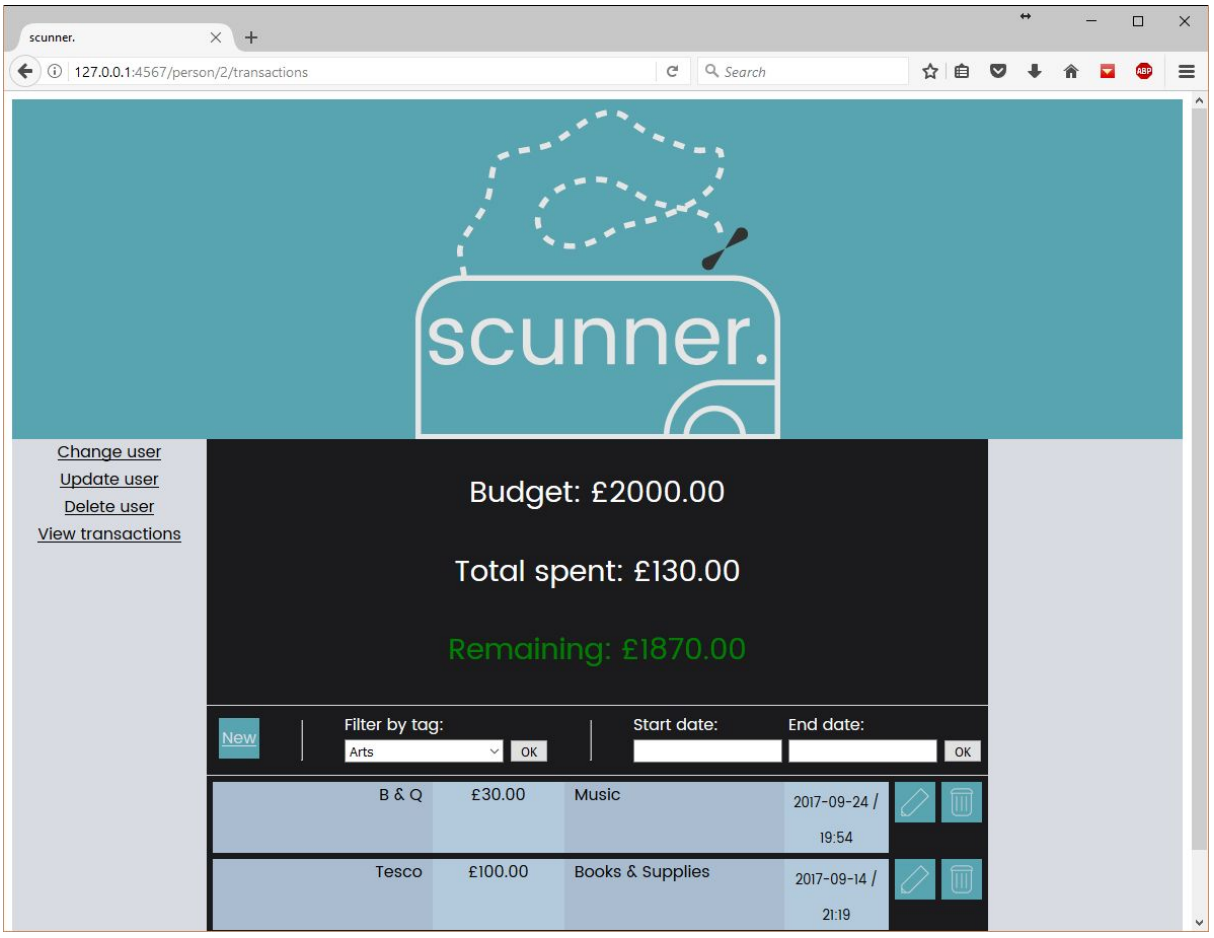
(P13 & P 14) Screenshot of user inputting new transaction data:



(P 14) Confirmation of user data being saved:

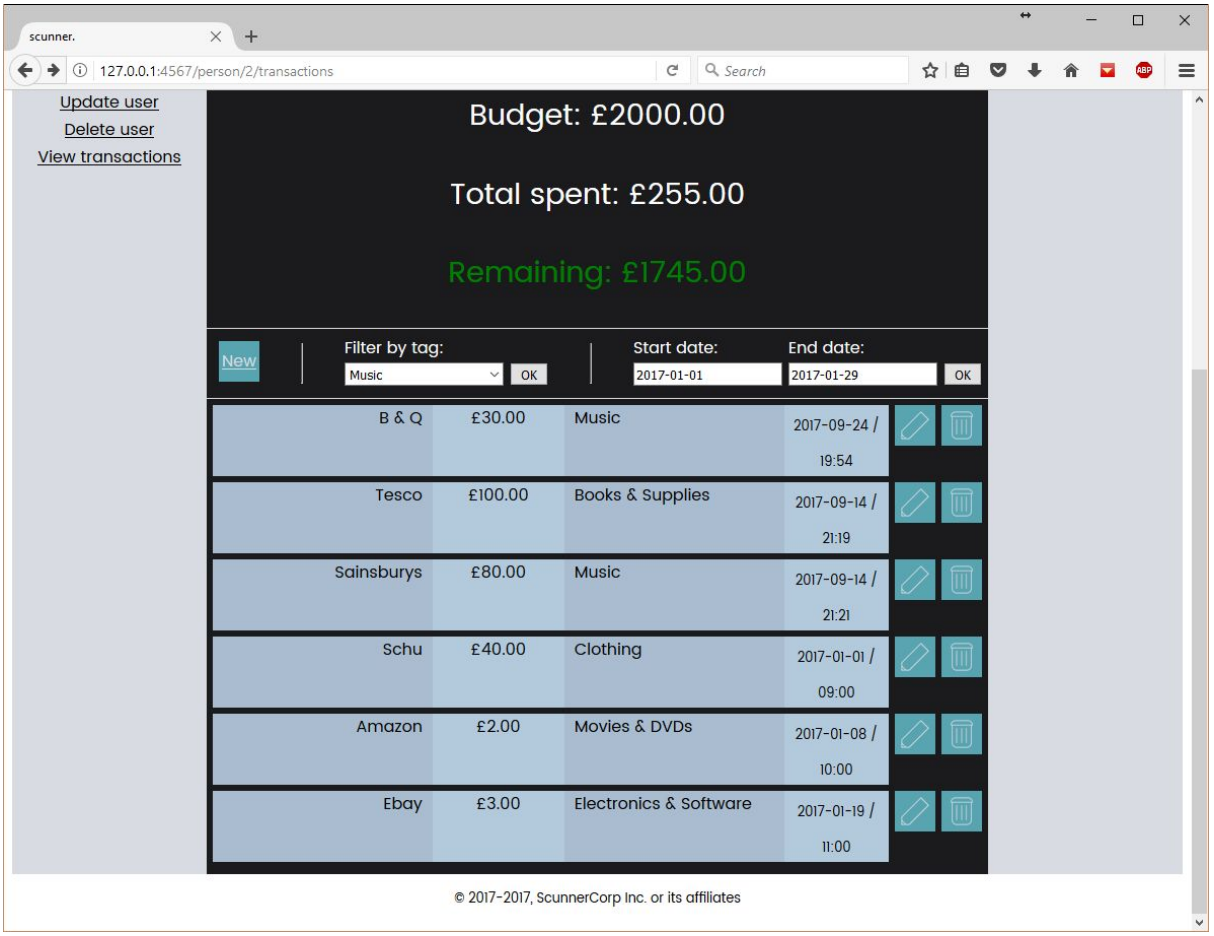


(P 13 & P 14) Screenshot of transactions after user has saved new transaction:

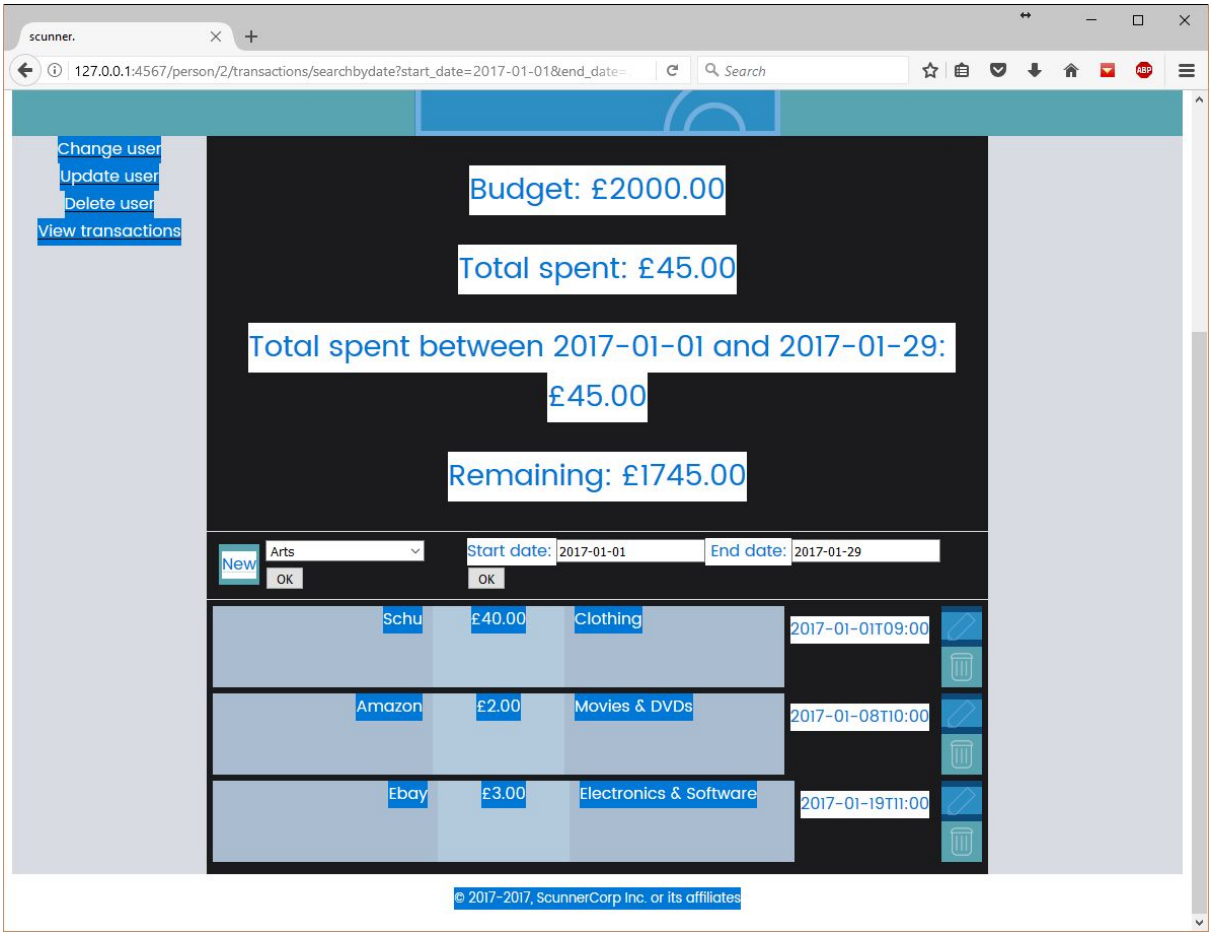


Week 5 / P 15 / Output of results / Feedback to user

Screenshot of transaction list before filtering by start/end date. Start and end dates for filtering have been input, but the adjacent ‘OK’ button has not yet been pressed:



Screenshot of transaction list after filtering by start/end date (list filtered to show only transactions occurring in January 2017). Text has been manually highlighted to improve readability:



Week 6 / IT 7 / Polymorphism

```
public interface IBasketDiscount
{
    int getValue(Basket basket, int currentValue);
}
```

The *IBasketDiscount* interface formalises the polymorphic *getValue* method.

This interface is subsequently implemented in the *LoyaltyCardDiscount* and *TenPercentOnTwentyPoundsDiscount* classes.

```
public class LoyaltyCardDiscount implements IBasketDiscount
{
    @Override
    public int getValue(Basket basket, int currentValue)
    {
        if(basket.getCustomer() == null || basket.getCustomer().getLoyaltyCard() == null)
        {
            return 0;
        }
        else
        {
            return (int)(currentValue * 0.02);
        }
    }
}
public class TenPercentOnTwentyPoundsDiscount implements IBasketDiscount
{
    @Override
    public int getValue(Basket basket, int currentValue)
    {
        if(currentValue > 2000)
        {
            return (int)(currentValue * 0.1);
        }
        else
        {
            return 0;
        }
    }
}
```

Instances of both of these classes are added to a collection of *IBasketDiscounts* (thus making use of polymorphism) in the *addBasketDiscount* method:

```
public void addBasketDiscount(IBasketDiscount basketDiscount)
{
    basketDiscounts.add(basketDiscount);
    updateCalculatedBasketValue();
}
```

Week 7 / AD 5 / Inheritance Diagram

This diagram shows the relationship between the abstract *Athlete* class and the concrete *Gymnast* and *Boxer* classes. The *Gymnast* and *Boxer* classes inherit all of the properties and methods of the *Athlete* class, and provide concrete implementations of the abstract *calculatePoints()* method.

Week 7 / IT 1 / Encapsulation

This screenshot demonstrates the use of encapsulation in a program. Here, the *distanceRunTotal*, *distanceRunLatest* and *name* fields have all been marked as private, restricting access to methods within the scope of the Athlete class. The *getName()* and *getDistanceRunLatest()* methods provide public access to the private fields.

```
public abstract class Athlete
{
    private int distanceRunTotal = 0;
    private int distanceRunLatest = 0;
    private String name;

    public Athlete(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

    public int getDistanceRunLatest()
    {
        return distanceRunLatest;
    }
```

Week 7 / IT 2 / Inheritance

Screenshot of the abstract *Athlete* class:

```
public abstract class Athlete
{
    private int distanceRunTotal = 0;
    private int distanceRunLatest = 0;
    private String name;

    public Athlete(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return name;
    }

    public int getDistanceRunLatest()
    {
        return distanceRunLatest;
    }

    public void setDistanceRunLatest(int distanceRunLatest)
    {
        this.distanceRunLatest = distanceRunLatest;
    }

    public void updateDistanceRunTotal()
    {
        distanceRunTotal += distanceRunLatest;
    }

    public int getDistanceRunTotal()
    {
        return distanceRunTotal;
    }

    public String prepare()
    {
        return "Prepared!";
    }
```

```
    public String compete()
    {
        return "Competed!";
    }

    public String awardMedal(int points)
    {
        String medal = "No medal!";

        if(5 <= points && points <= 9)
        {
            medal = "Bronze";
        }
        else if(10 <= points && points <= 14)
        {
            medal = "Silver";
        }
        else if(15 <= points)
        {
            medal = "Gold";
        }

        return medal;
    }

    public abstract int calculatePoints();
}
```

Screenshot of the concrete Boxer class:

```
public class Boxer extends Athlete
{
    int hitsGiven, hitsTaken;

    public Boxer(String name, int hitsGiven, int hitsTaken)
    {
        super(name);
        this.hitsGiven = hitsGiven;
        this.hitsTaken = hitsTaken;
    }

    public int calculatePoints()
    {
        return hitsGiven - hitsTaken;
    }
}
```

Screenshot showing the *hitsGiven* and *hitsTaken* integer objects used within the *Boxer* class.

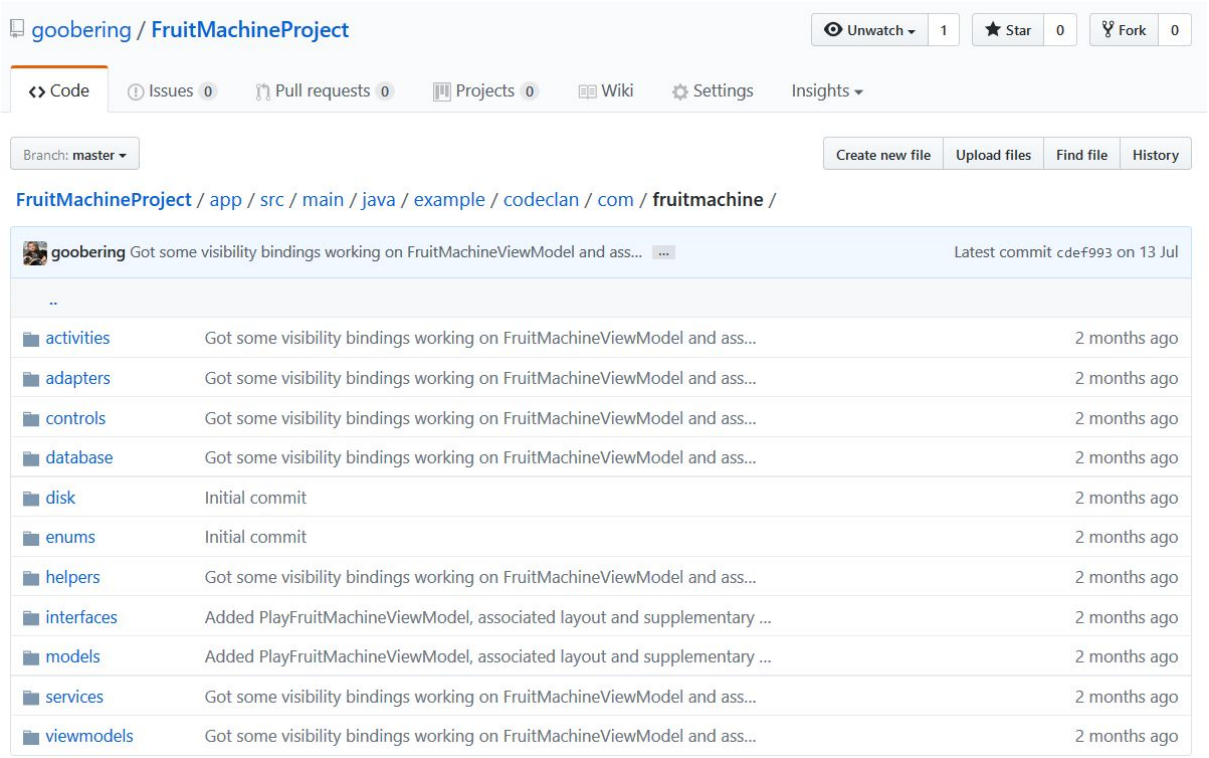
```
public class Boxer extends Athlete
{
    int hitsGiven, hitsTaken;
```

Screenshot showing the *Boxer* class's concrete implementation of the abstract *calculatePoints()* method defined in the *Athlete* class:

```
public int calculatePoints()
{
    return hitsGiven - hitsTaken;
}
```

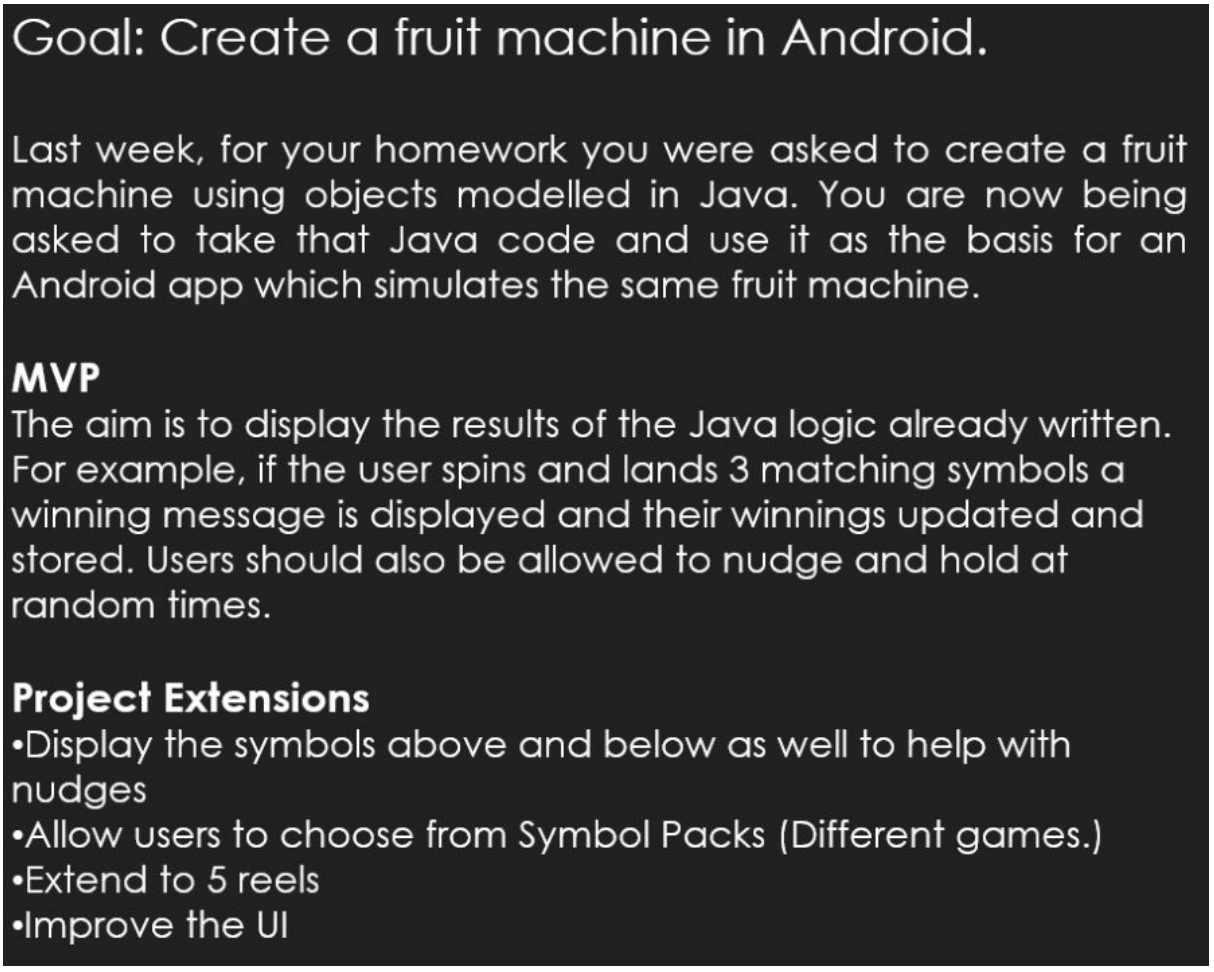
Week 7 / P 11 / Solo work Github

Github link:
https://github.com/goobering/FruitMachineProject

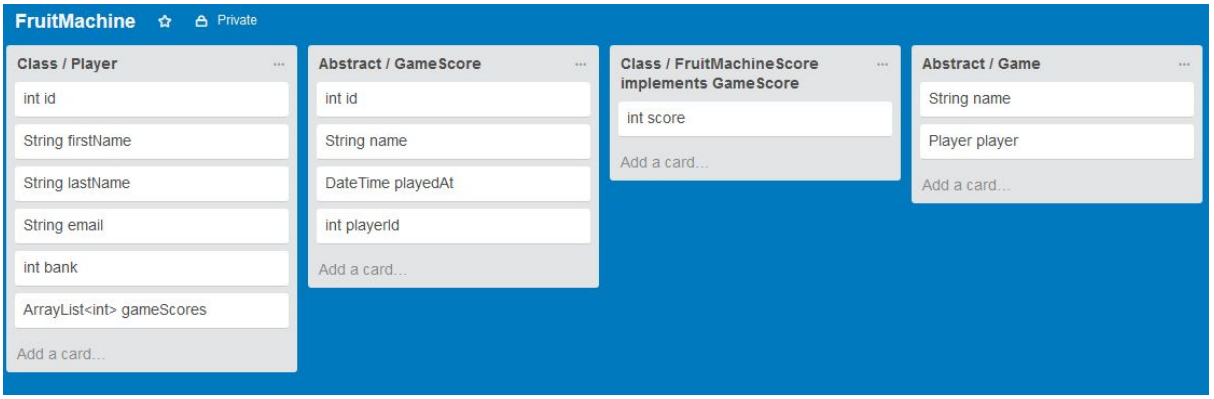


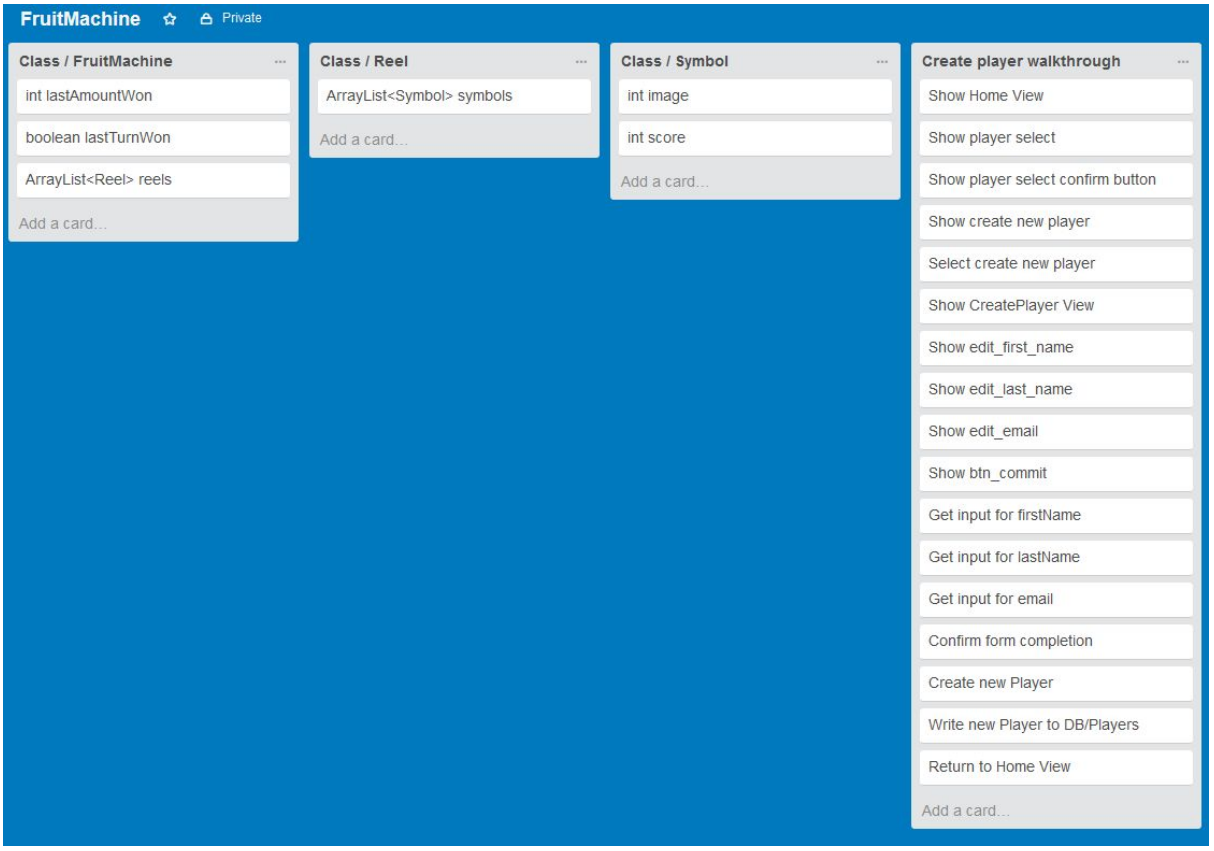
Week 7 / P 12 / Planning

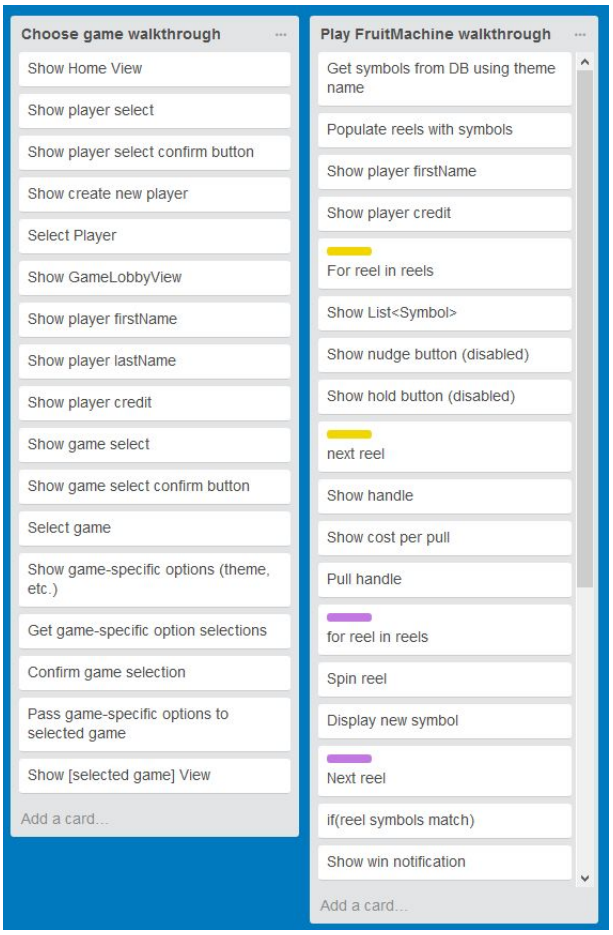
The brief for the project was as follows:



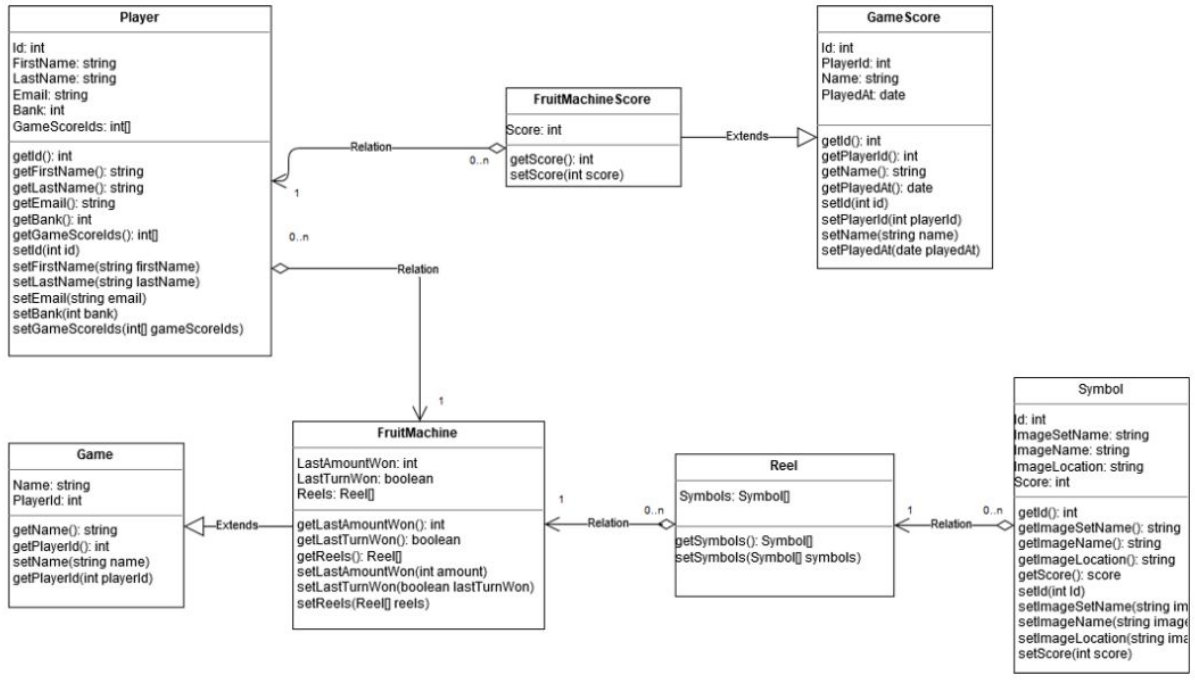
Screenshots of Trello board planning:





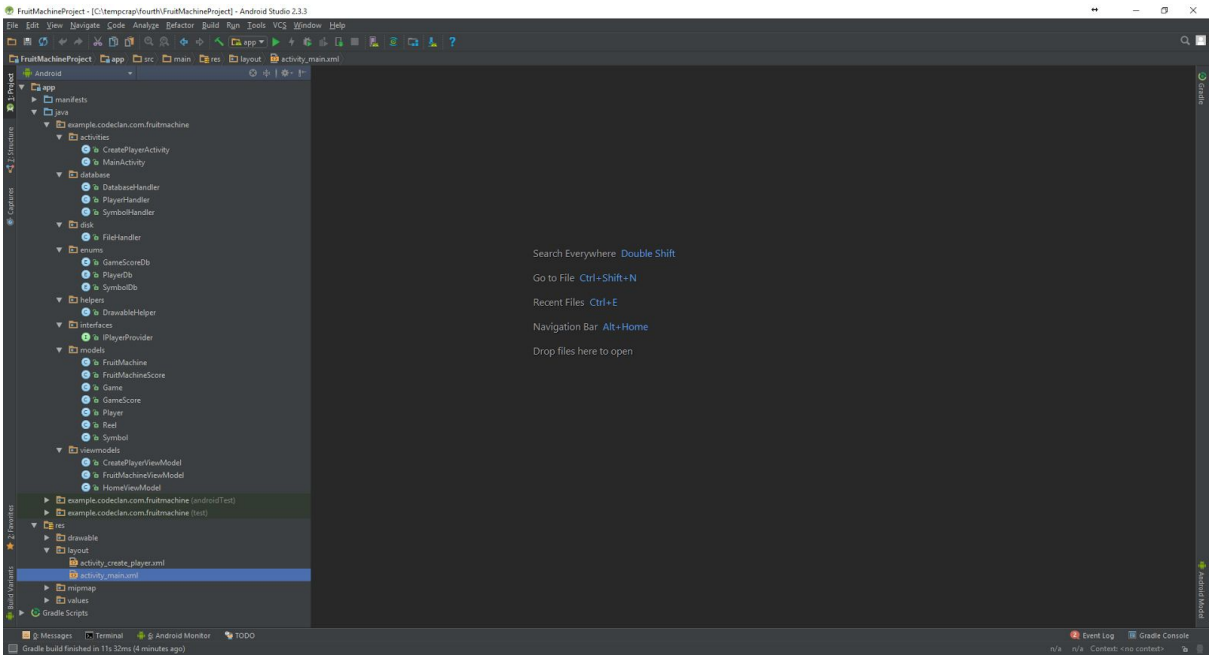


Class/relationship diagram:

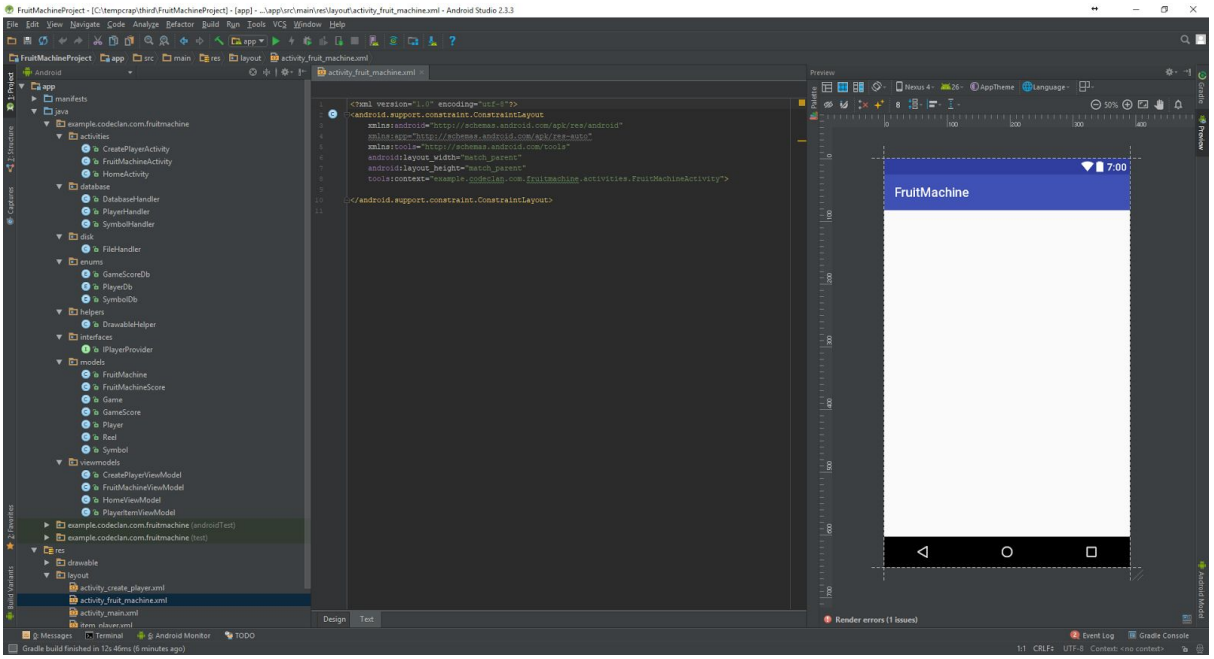


Progress:

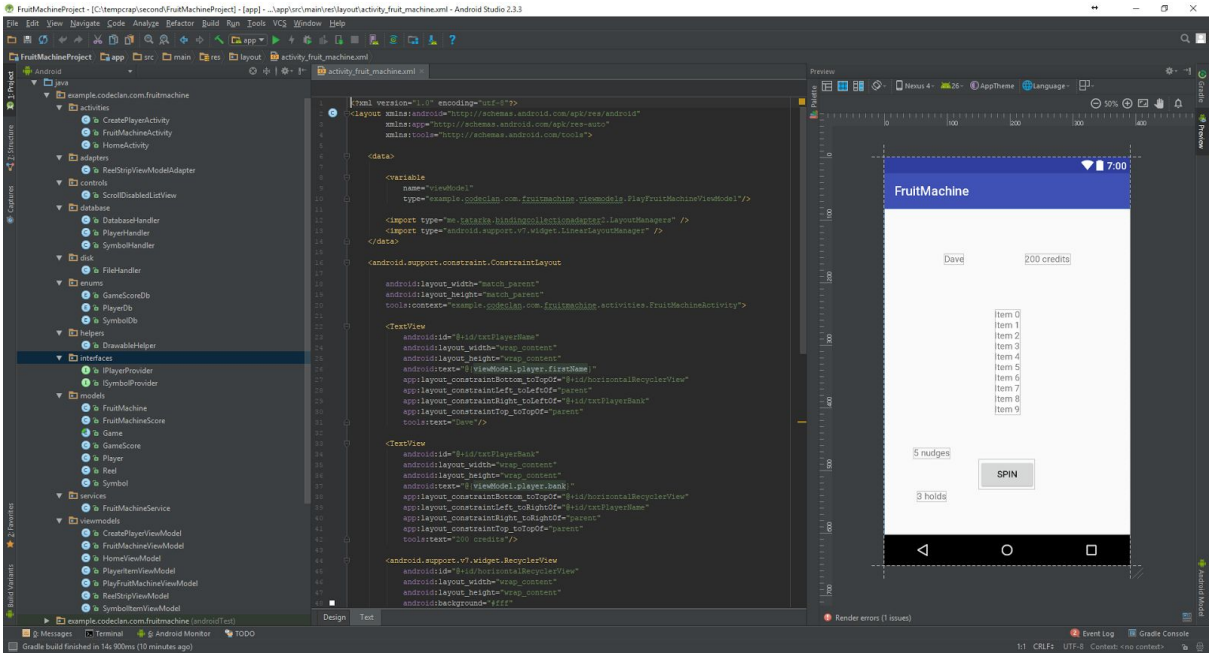
Second commit. Limited number of files in the explorer pane on the left, no activity_fruit_machine.xml file yet.



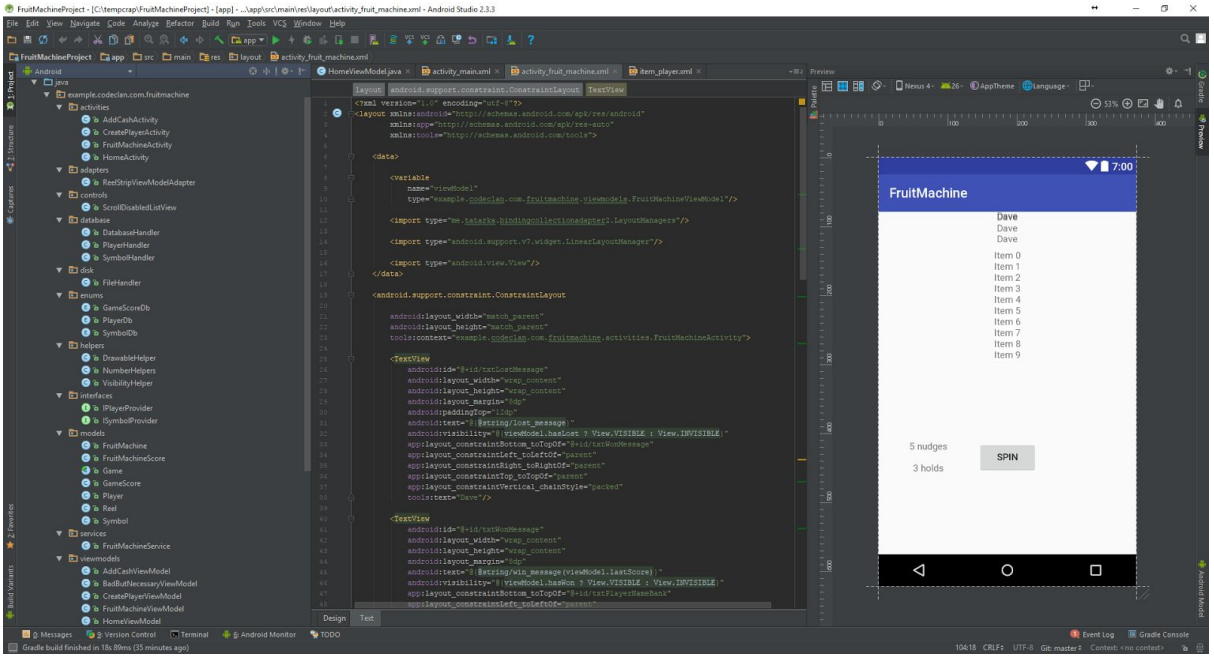
Third commit. Larger number of files created, and a blank activity_fruit_machine.xml interface.



Fourth commit. More code files added to the explorer pane, and a fleshed out activity_fruit_machine.xml interface.

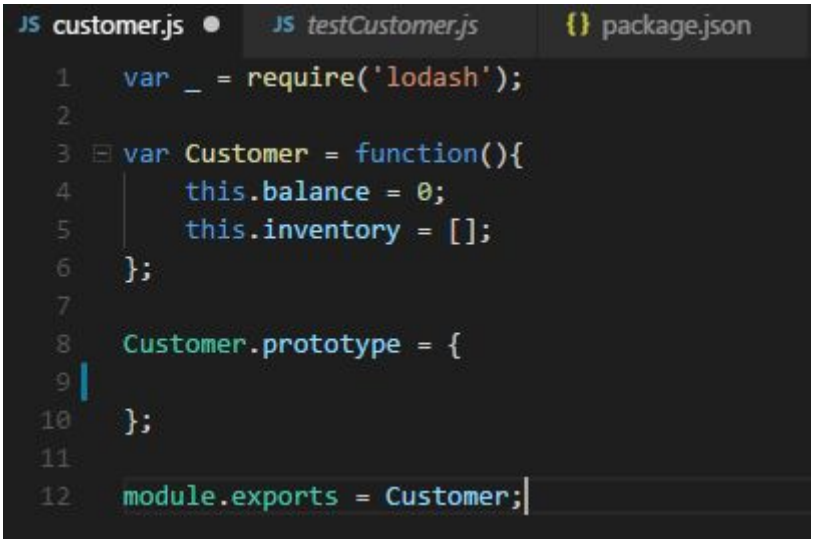


Fifth commit. The project is complete, as is the activity_fruit_machine.xml interface.



Week 10 / P 18 / Testing
Test code

Class being tested:



Testing class:

JS testCustomer.js

```
var assert = require('assert');
var Customer = require('../customer.js');
var BookStore = require('../bookStore.js');
var Book = require('../book.js');

describe('Customer', function(){
    var customer, testCustomer1;
    var testBook1, testBook2, testBook3;
    var testBookStore1;

    beforeEach(function(){
        testBook1 = new Book('testAuthor1', 'testTitle1', 'testGenre1', 10, 100);
        testBook2 = new Book('testAuthor2', 'testTitle2', 'testGenre2', 20, 200);
        testBook3 = new Book('testAuthor3', 'testTitle3', 'testGenre3', 30, 300);

        customer = new Customer();
        customer.balance = 200;
        customer.inventory.push(testBook1);
        customer.inventory.push(testBook2);
        customer.inventory.push(testBook3);

        testCustomer1 = new Customer();
        testCustomer1.balance = 100;
        testCustomer1.inventory.push(testBook1);
        testCustomer1.inventory.push(testBook1);
        testCustomer1.inventory.push(testBook1);

        testBookStore1 = new BookStore('testName1', 'testCity1');

        testBookStore1.addBook(testBook1);
        testBookStore1.addBook(testBook2);
        testBookStore1.addBook(testBook3);
    });

    it('has an inventory', function(){
        assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3]);
    });
```

```
38
39      it('has a balance', function(){
40          assert.strictEqual(customer.balance, 200);
41      });
42
43      it('can buy a Book', function(){
44          customer.buy(testBookStore1, testBook1);
45
46          assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3, testBook1]);
47          assert.strictEqual(customer.balance, 190);
48      });
49
50      it('can sell a book', function(){
51          testBookStore1.balance = 200;
52
53          var sold = customer.sell(testBookStore1, testBook1);
54
55          assert.strictEqual(sold, true);
56          assert.strictEqual(customer.balance, 210);
57          assert.deepEqual(customer.inventory, [testBook2, testBook3]);
58      });
59
60      it('can return the total value of their collection', function(){
61          assert.strictEqual(customer.getInventoryValue(), 60);
62      });
63
64      it('can view the total value of all books of a given Genre', function(){
65          assert.strictEqual(customer.getValueOfGenre('testGenre1'), 10);
66      });
67
68      it('can view their longest book', function(){
69          assert.strictEqual(customer.getValueOfLongestBook(), 30);
70      });
71
72      it('can sort their books by value (ascending)', function(){
73          customer.sortInventoryByValue(true);
74          assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3]);
75      });
```

```
76
77      it('can sort their books by value (descending)', function(){
78          customer.sortInventoryByValue(false);
79          assert.deepEqual(customer.inventory, [testBook3, testBook2, testBook1]);
80      });
81
82      it('should be able to compare the value of their collection with another BookWorm', function(){
83          assert.strictEqual(customer.compareWith(testCustomer1), 30);
84      })
85  });
```

Test code failing. The code which causes ‘has an inventory’ and ‘has a balance’ to pass is required in order to properly run the test code.

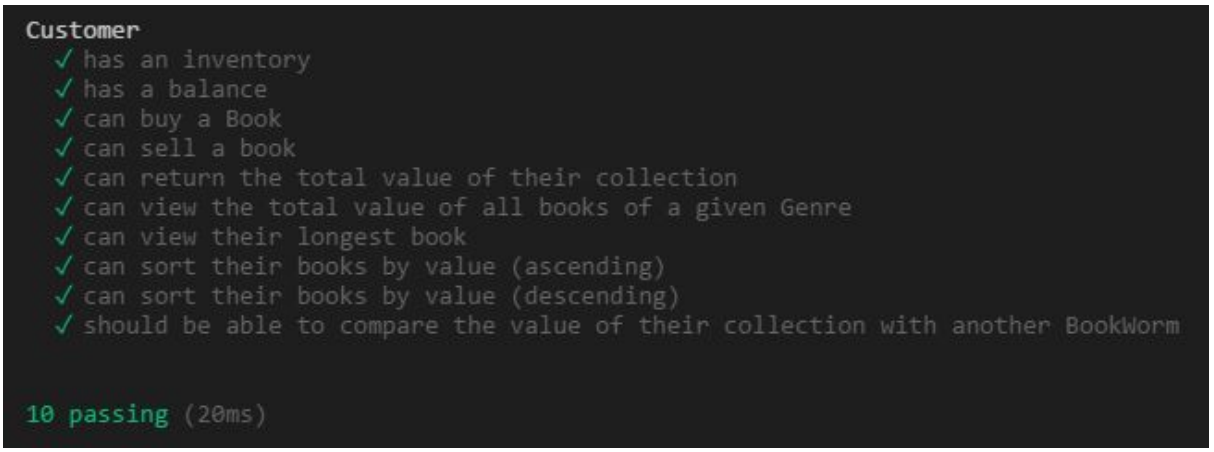
```
Customer
  ✓ has an inventory
  ✓ has a balance
  1) can buy a Book
  2) can sell a book
  3) can return the total value of their collection
  4) can view the total value of all books of a given Genre
  5) can view their longest book
  6) can sort their books by value (ascending)
  7) can sort their books by value (descending)
  8) should be able to compare the value of their collection with another BookWorm


2 passing (21ms)
8 failing

1) Customer can buy a Book:
   TypeError: customer.buy is not a function
    at Context.<anonymous> (specs/testCustomer.js:44:18)

2) Customer can sell a book:
   TypeError: customer.sell is not a function
    at Context.<anonymous> (specs/testCustomer.js:53:29)

3) Customer can return the total value of their collection:
   TypeError: customer.getInventoryValue is not a function
    at Context.<anonymous> (specs/testCustomer.js:61:37)

4) Customer can view the total value of all books of a given Genre:
   TypeError: customer.getValueOfGenre is not a function
    at Context.<anonymous> (specs/testCustomer.js:65:37)

5) Customer can view their longest book:
   TypeError: customer.getValueOfLongestBook is not a function
    at Context.<anonymous> (specs/testCustomer.js:69:37)

6) Customer can sort their books by value (ascending):
   TypeError: customer.sortInventoryByValue is not a function
    at Context.<anonymous> (specs/testCustomer.js:73:18)

7) Customer can sort their books by value (descending):
   TypeError: customer.sortInventoryByValue is not a function
    at Context.<anonymous> (specs/testCustomer.js:78:18)

8) Customer should be able to compare the value of their collection with another BookWorm:
   TypeError: customer.compareWith is not a function
    at Context.<anonymous> (specs/testCustomer.js:83:37)
```

Test code after corrections

customer.js | testCustomer.js | package.json

```
var _ = require('lodash');

var Customer = function(){
    this.balance = 0;
    this.inventory = [];
};

Customer.prototype = {
    canBuy: function(book){
        return this.balance >= book.price;
    },
    canSell: function(book){
        return _.includes(this.inventory, book);
    },
    buy: function(bookStore, book){
        if(bookStore.canSell(book) && this.canBuy(book)){
            bookStore.sell(book, this);
            this.inventory.push(book);
            this.balance -= book.price;
            return true;
        } else {
            return false;
        };
    },
    sell: function(bookStore, book){
        if(bookStore.canBuy(book) && this.canSell(book)){
            bookStore.buy(book, this);
            _.remove(this.inventory, book);
            this.balance += book.price;
            return true;
        } else {
            return false;
        };
    },
    getInventoryValue: function(){
        return _.sumBy(this.inventory, 'price');
    },
    getValueOfGenre: function(genre){
        return _.sumBy(_.filter(this.inventory, {genre: genre}), 'price');
    },
    getValueOfLongestBook: function(){
        return _.maxBy(this.inventory, 'pageCount').price;
    },
    sortInventoryByValue: function(isAscending){
        if(isAscending){
            this.inventory = _.sortBy(this.inventory, 'price');
        } else {
            this.inventory = _.sortBy(this.inventory, 'price').reverse();
        }
    },
    compareWith: function(customer){
        return this.getInventoryValue() - customer.getInventoryValue();
    }
};

module.exports = Customer;
```

Test code passing



Week 11 / P 16 / APIs

Code that uses an API:
(also stored at: https://github.com/goobering/api_calls/blob/master/public/index.js)

```
var windowLoaded = function(){
    var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition
    var SpeechRecognitionEvent = SpeechRecognitionEvent || webkitSpeechRecognitionEvent

    var recognition = new SpeechRecognition();
    recognition.lang = 'en-GB';
    recognition.interimResults = false;
    recognition.maxAlternatives = 1;

    var diagnostic = document.querySelector('#output');
    var startTag = document.querySelector('#start-speech');
    var startLabel = document.querySelector('#start-label');

    startTag.onclick = function(){
        // Clear out Wikipedia extract
        var wikiDiv = document.querySelector('#wiki-extract');
        wikiDiv.innerHTML = '';

        // Start speech recognition
        recognition.start();
    }

    // Called when the user has stopped speaking and parsed result returned
    recognition.onresult = function(event){
        var last = event.results.length - 1;
        var phrase = event.results[last][0].transcript;

        // Send the parsed string to an interface element
        diagnostic.textContent = phrase;

        var jsonPhrase = {"phrase": phrase};
        getYoutubeResult(jsonPhrase);
        getWikipediaResult(jsonPhrase);
    };

    recognition.onaudiostart = function() {
        console.log('Audio capturing started');

        startTag.style.backgroundColor = 'red';
        startTag.style.color = 'white';
        startLabel.textContent  = 'Recording';
    };

```

```
    recognition.onaudioend = function() {
        console.log('Audio capturing ended');

        startTag.style.backgroundColor = 'gold';
        startTag.style.color = 'lightslategrey';
        startLabel.textContent  = 'Start';
    };

    recognition.onspeechend = function(){
        recognition.stop();
    };

    recognition.onerror = function(event){
        diagnostic.textContent = 'Error occurred in recognition: ' + event.error;
    };
};

window.addEventListener('load', windowLoaded);

//searchTerm needs to be an object with a value representing the desired search
var getYoutubeResult = function(searchTerm){
    var stringTerm = searchTerm.phrase;
    var url = 'http://localhost:3000/youtube/' + stringTerm;
    makeRequest(url, youtubeRequestComplete);
};

var getWikipediaResult = function(searchTerm){
    var stringTerm = encodeURIComponent(searchTerm.phrase.trim());
    var url = 'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=' + stringTerm + '&origin=*';

    var xhr = createCORSRequest('GET', url);
    if (!xhr) {
        alert('CORS not supported');
        return;
    }

    // Response handlers.
    xhr.onload = function(){
        if(this.status !== 200)
        return;

        var responseText = xhr.responseText;
        var jsonText = JSON.parse(responseText);
        buildWiki(jsonText);
    }

```

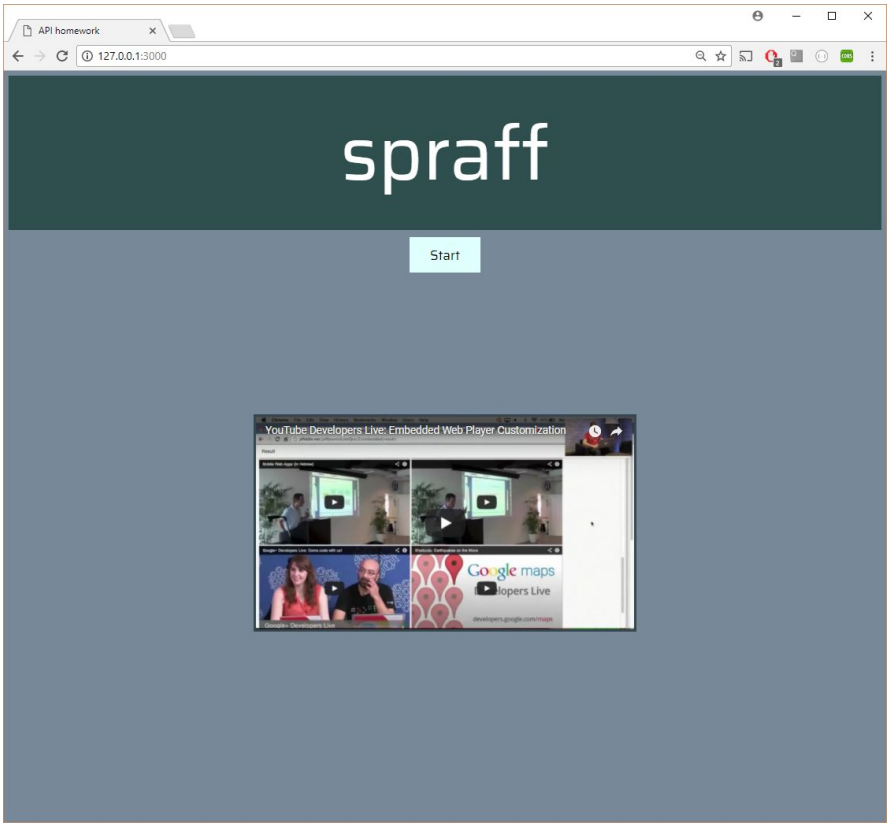
```
    xhr.onerror = function() {
        alert('Woops, there was an error making the request.');
    };

    xhr.send();
}

// Create the XHR object.
function createCORSRequest(method, url) {
  var xhr = new XMLHttpRequest();
  if ("withCredentials" in xhr) {
    // XHR for Chrome/Firefox/Opera/Safari.
    xhr.open(method, url, true);
  } else if (typeof XDomainRequest != "undefined") {
    // XDomainRequest for IE.
    xhr = new XDomainRequest();
    xhr.open(method, url);
  } else {
    // CORS not supported.
    xhr = null;
  }
  return xhr;
}

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

var youtubeRequestComplete = function(){
    if(this.status !== 200)
        return;

    var jsonString = this.responseText;
    var wtf = JSON.parse(jsonString);
    var result = JSON.parse(wtf);

    buildYoutube(result.items[0].id.videoId);
};

```

```
var buildYoutube = function(videoId){
    console.log('videoId: ' + videoId);
    console.log('this.YT: ' + this.YT);
    if(this.YT){
        this.YT.get('ytplayer').loadVideoById(videoId);
        return;
    }

    // Load the IFrame Player API code asynchronously.
    var tag = document.createElement('script');
    tag.src = "https://www.youtube.com/iframe_api";
    var firstScriptTag = document.getElementsByTagName('script')[0];
    firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

    var player;

    window.onYouTubeIframeAPIReady = function() {
        player = new YT.Player('ytplayer', {
            attr: { type: 'text/css', href: 'main.css', rel: 'stylesheet' },
            videoId: videoId,
            events: {
            'onReady': onPlayerReady,
            'onStateChange': onPlayerStateChange
            }
        });
    };

    function onPlayerReady(event) {
        console.log('player: ' + player);
        console.log('player.videoId: ' + player.b.c.videoId);
        event.target.playVideo();
    };

    var done = false;
    function onPlayerStateChange(event) {
        if (event.data == YT.PlayerState.ENDED) {
            //Get rid of the player
            event.target.destroy();
        };
    };

    function stopVideo() {
        player.stopVideo();
    };
}
```

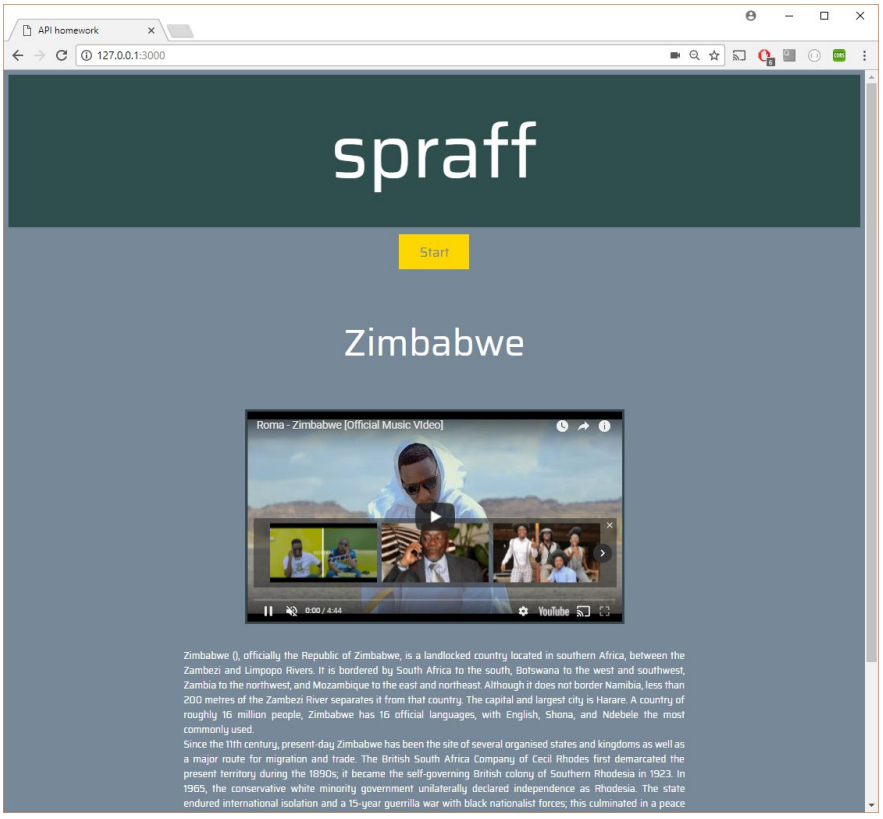
```

var buildWiki = function(wikiExtract){
    if(Object.entries(wikiExtract.query.pages)[0][1].extract){
        var wikiDiv = document.querySelector('#wiki-extract');

        var wikiTag = document.createElement('p');
        var text = Object.entries(wikiExtract.query.pages)[0][1].extract.replace(/<[^>]+>/gi,"");
        wikiTag.innerText = text;

        wikiDiv.appendChild(wikiTag);
    };
};
```

API being used by the program:

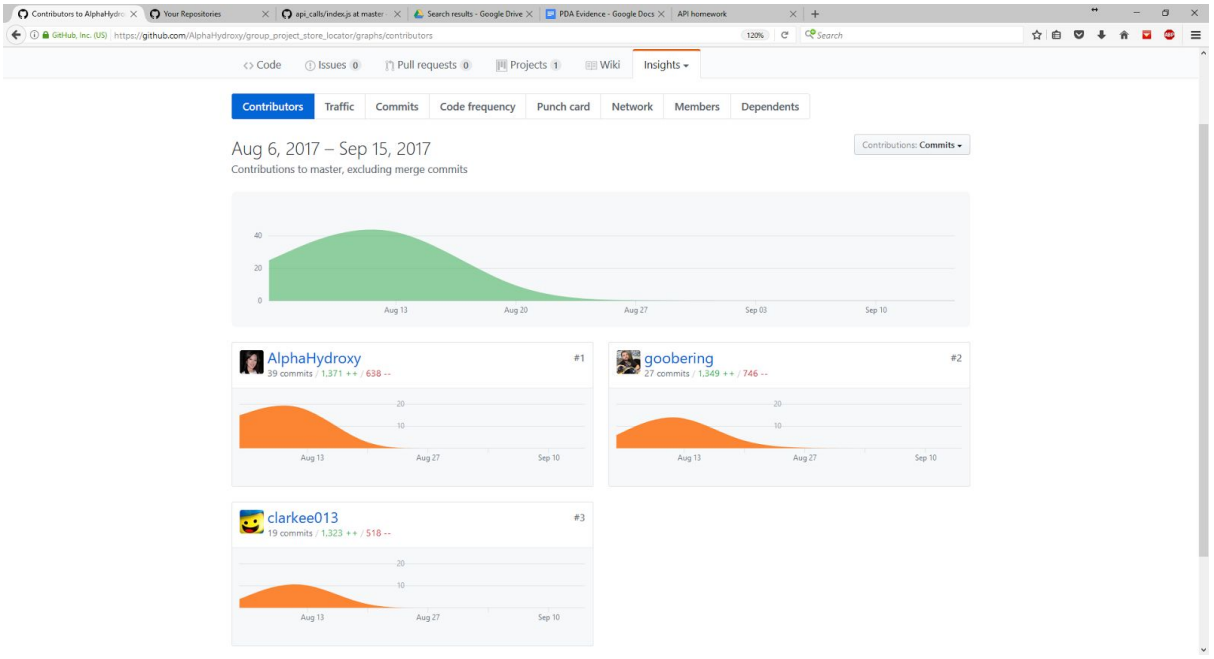
Before voice/phrase recognition:



After successful voice/phrase recognition (‘Zimbabwe’). The first Youtube result and the first Wikipedia result matching the phrase are displayed on-screen:



Week 13 / P 1 / Group project Github contributors page



Week 13 / P 2 / Group project brief

Store Finder

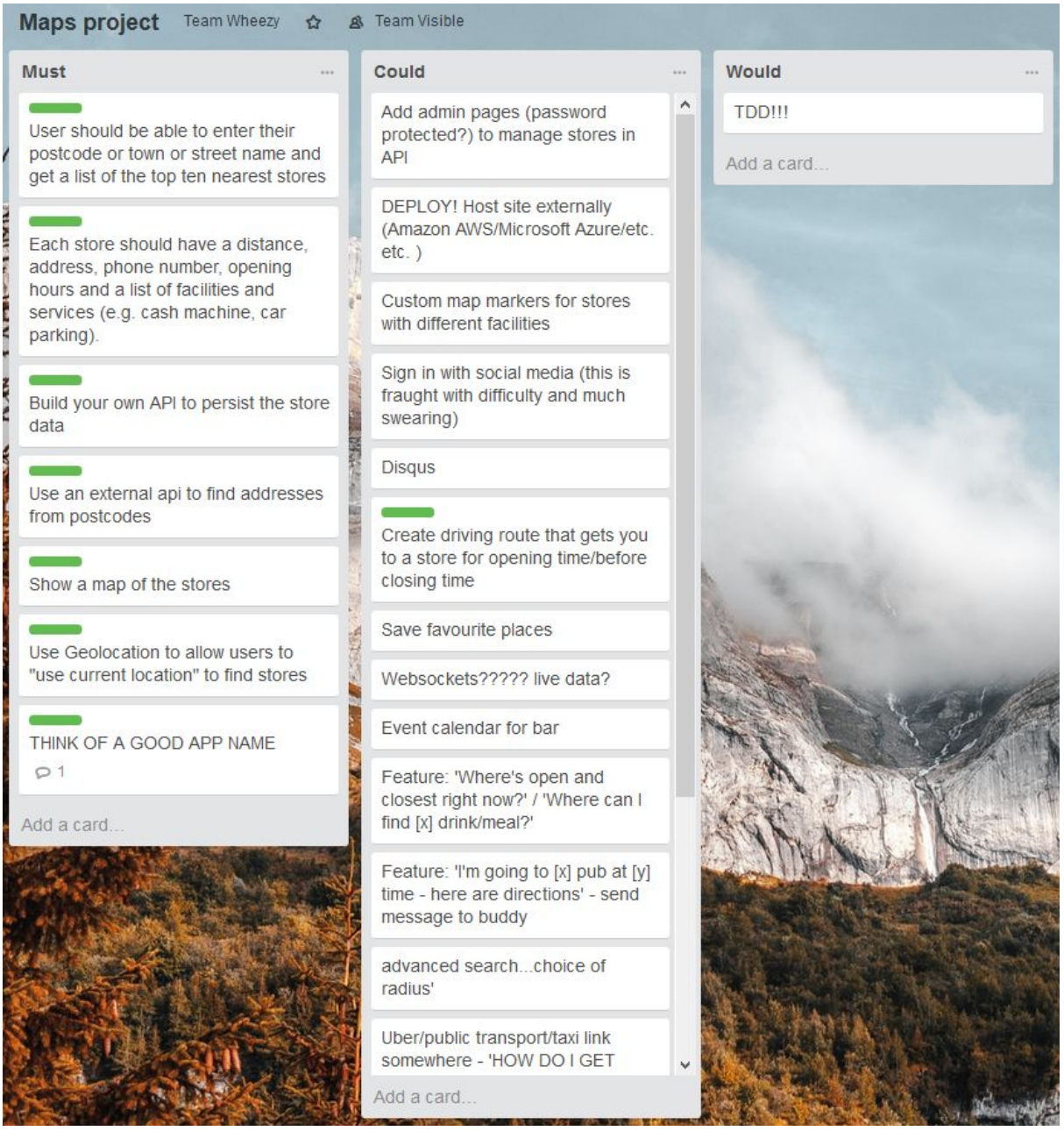
Global pub giant Withoutaspoon, wants a pub finder that allows users to "Find your nearest store" for their website.

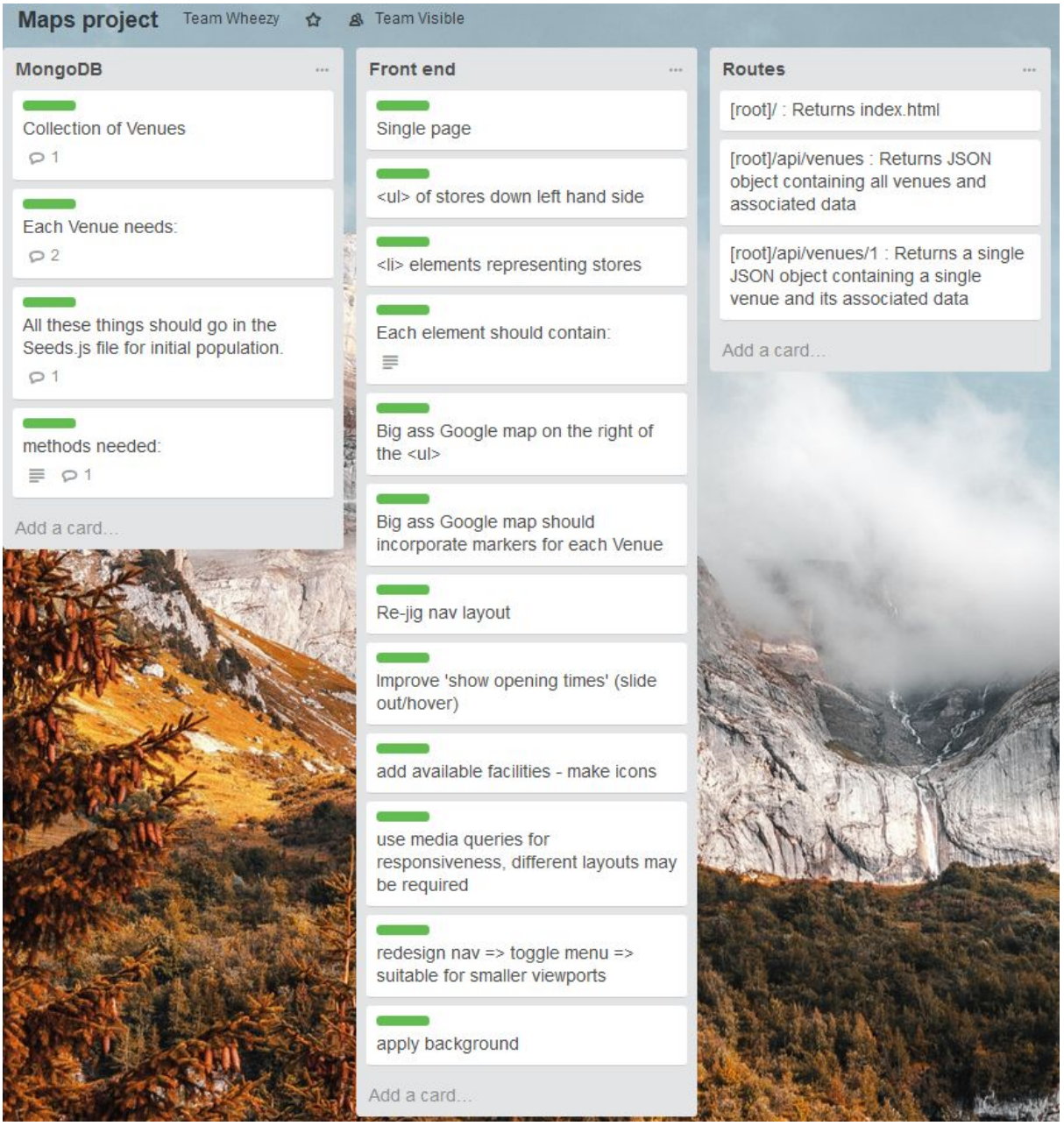
User can enter their postcode or town or street name and get a list of the venues within 3 miles, and a map showing all of the venues. Each has a distance, address, phone number, opening hours and a list of facilities and services (e.g. cash machine, car parking).

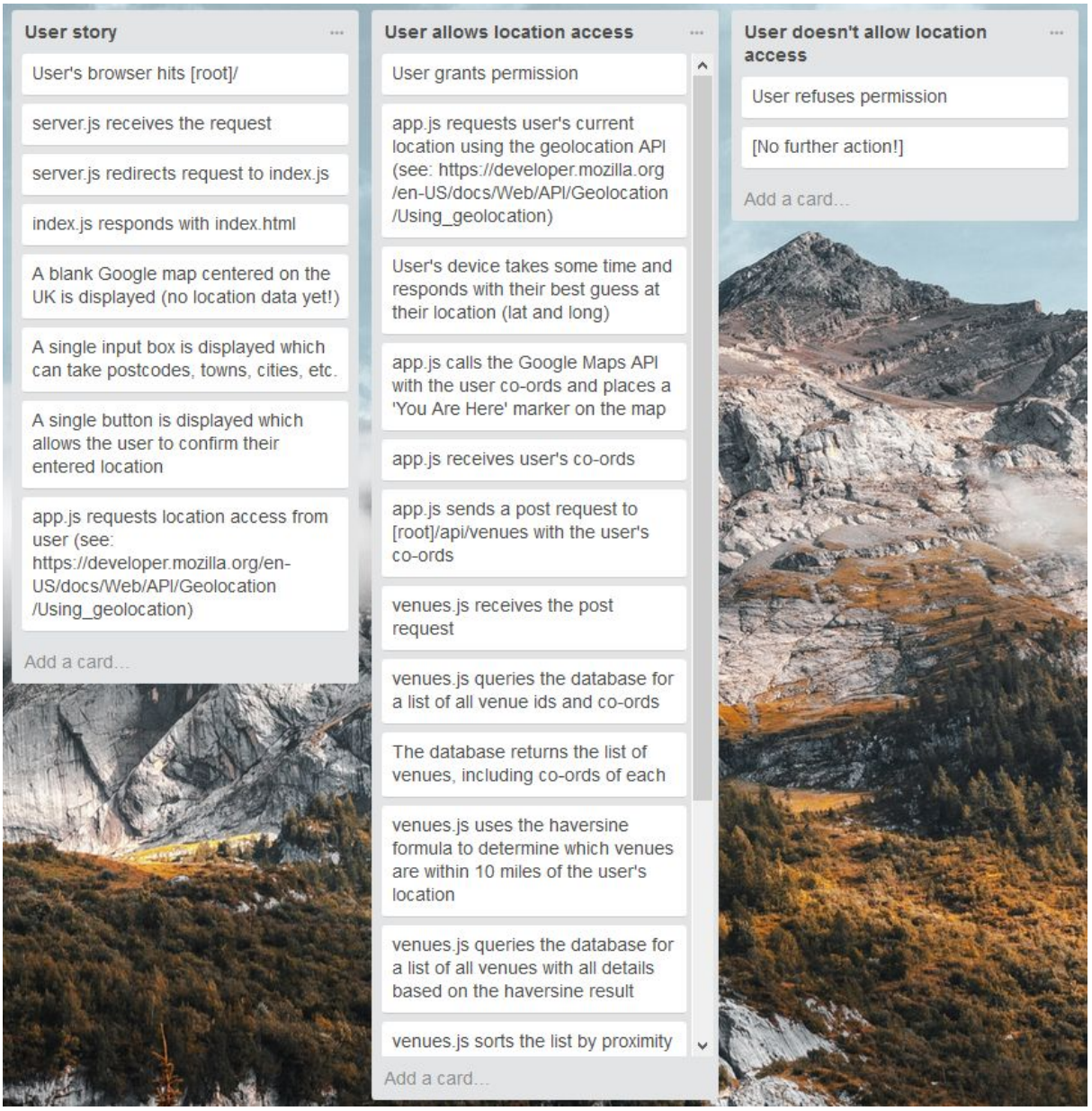
To get started:

- Clone the project
- npm install
- npm run build
- npm start

Week 13 / P 3 / Planning (Trello, etc.)




Maps project
Team Wheezy
Team Visible
MongoDB
Collection of Venues
1
Each Venue needs:
2
All these things should go in the Seeds.js file for initial population.
1
methods needed:
1
Add a card...
Front end
Single page
 of stores down left hand side
 elements representing stores
Each element should contain:
Big ass Google map on the right of the
Big ass Google map should incorporate markers for each Venue
Re-jig nav layout
Improve 'show opening times' (slide out/hover)
add available facilities - make icons
use media queries for responsiveness, different layouts may be required
redesign nav => toggle menu => suitable for smaller viewports
apply background
Add a card...
Routes
[root]/ : Returns index.html
[root]/api/venues : Returns JSON object containing all venues and associated data
[root]/api/venues/1 : Returns a single JSON object containing a single venue and its associated data
Add a card...

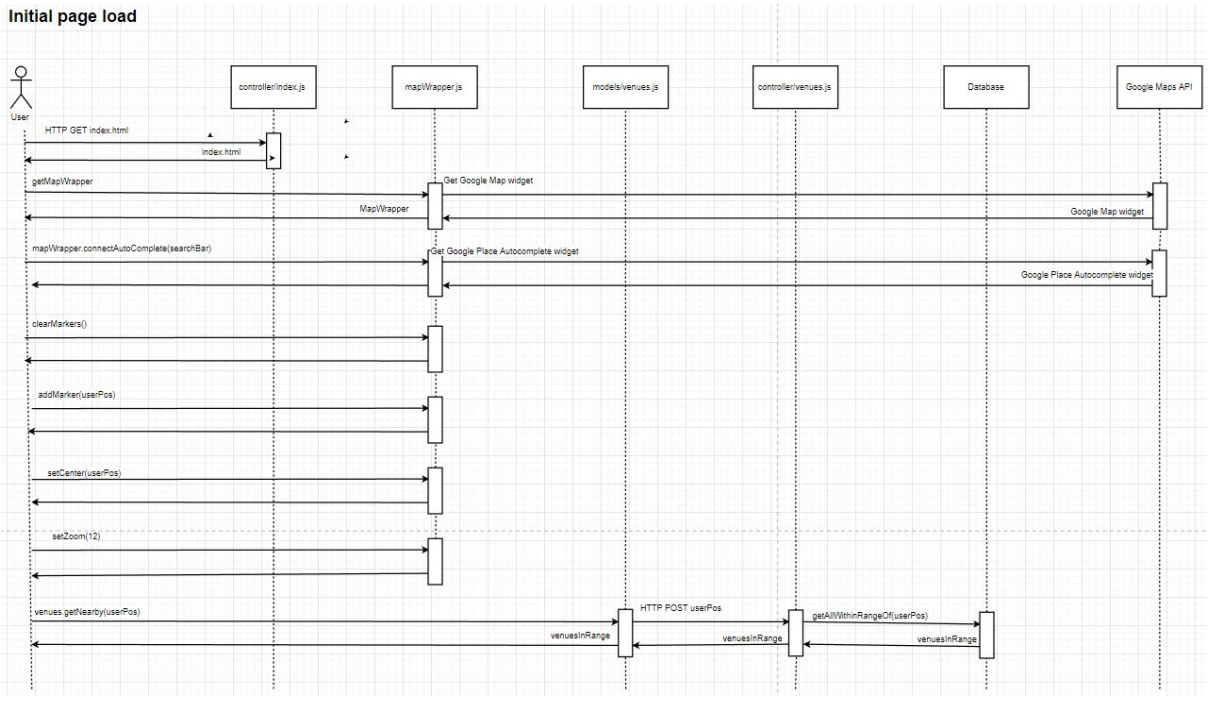

User story
User's browser hits [root]/
server.js receives the request
server.js redirects request to index.js
index.js responds with index.html
A blank Google map centered on the UK is displayed (no location data yet!)
A single input box is displayed which can take postcodes, towns, cities, etc.
A single button is displayed which allows the user to confirm their entered location
app.js requests location access from user (see: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation)
Add a card...
User allows location access
User grants permission
app.js requests user's current location using the geolocation API (see: https://developer.mozilla.org/en-US/docs/Web/API/Geolocation/Using_geolocation)
User's device takes some time and responds with their best guess at their location (lat and long)
app.js calls the Google Maps API with the user co-ords and places a 'You Are Here' marker on the map
app.js receives user's co-ords
app.js sends a post request to [root]/api/venues with the user's co-ords
venues.js receives the post request
venues.js queries the database for a list of all venue ids and co-ords
The database returns the list of venues, including co-ords of each
venues.js uses the haversine formula to determine which venues are within 10 miles of the user's location
venues.js queries the database for a list of all venues with all details based on the haversine result
venues.js sorts the list by proximity
Add a card...
User doesn't allow location access
User refuses permission
[No further action!]
Add a card...

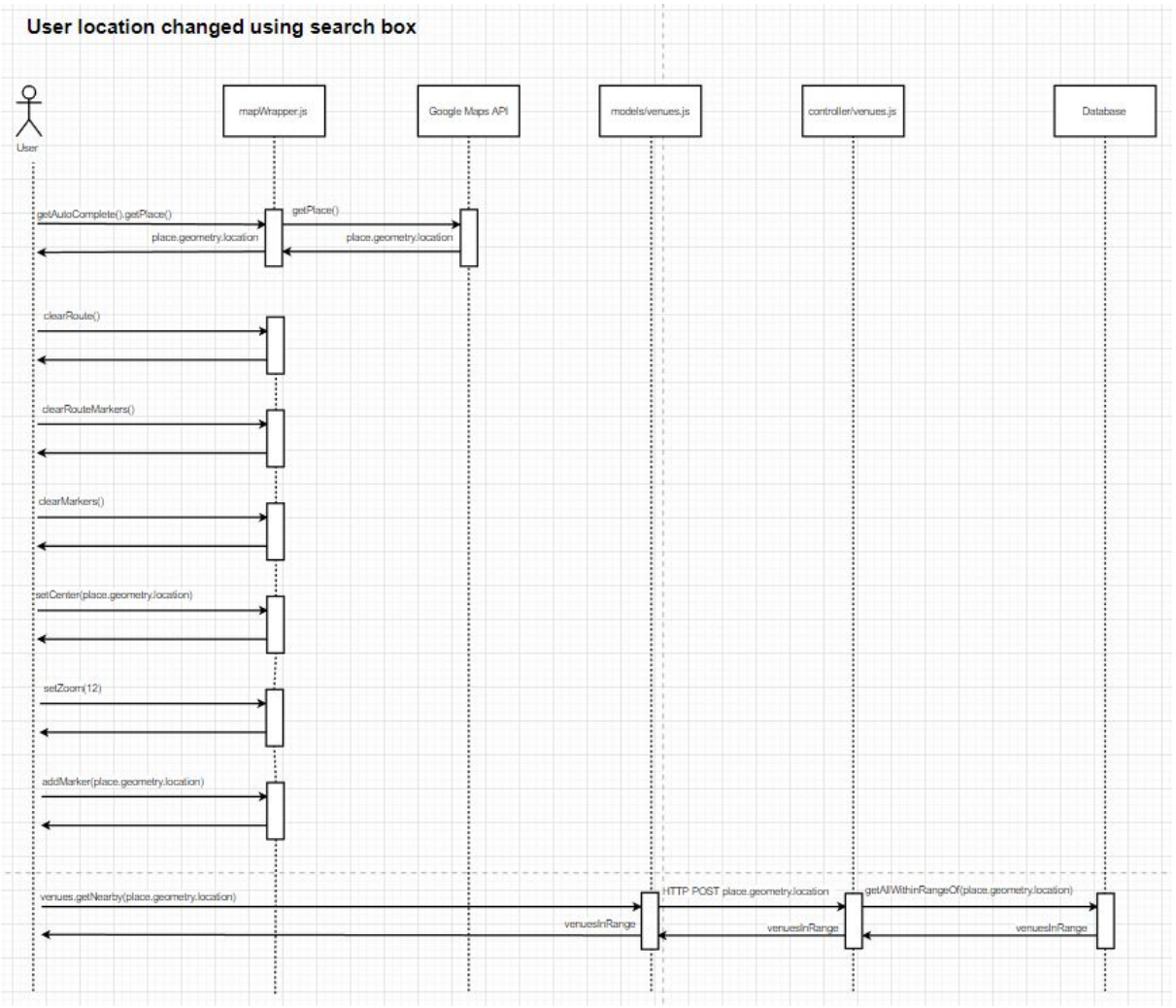
Week 13 / P 4 / Acceptance criteria and test plan

Acceptance Criteria	Expected Result/Output	Pass /Fail
User should be able to enter postcode/address get list of nearest pubs	List of pubs to show on homepage	Pass
Each pub should have distance, address, phone number, opening hours & other facilities	list view on homepage for the details & facilities should be seen	Pass
Facilities to show to load from internal API storage	The pubs and facilities should show correctly on the list on the list view of the webpage	Pass
External API to be used for address look up and populates to map view	Markers for pubs saved in internal API showing on map	Pass
Geolocation to allow user to see pubs around their location	Pubs shown should be within distance of the user	Pass
User should be able to click on pub markers to display walking route	Walking route displayed on map from current location	Pass
User should be able to save favourite pub	not implemented yet	Fail

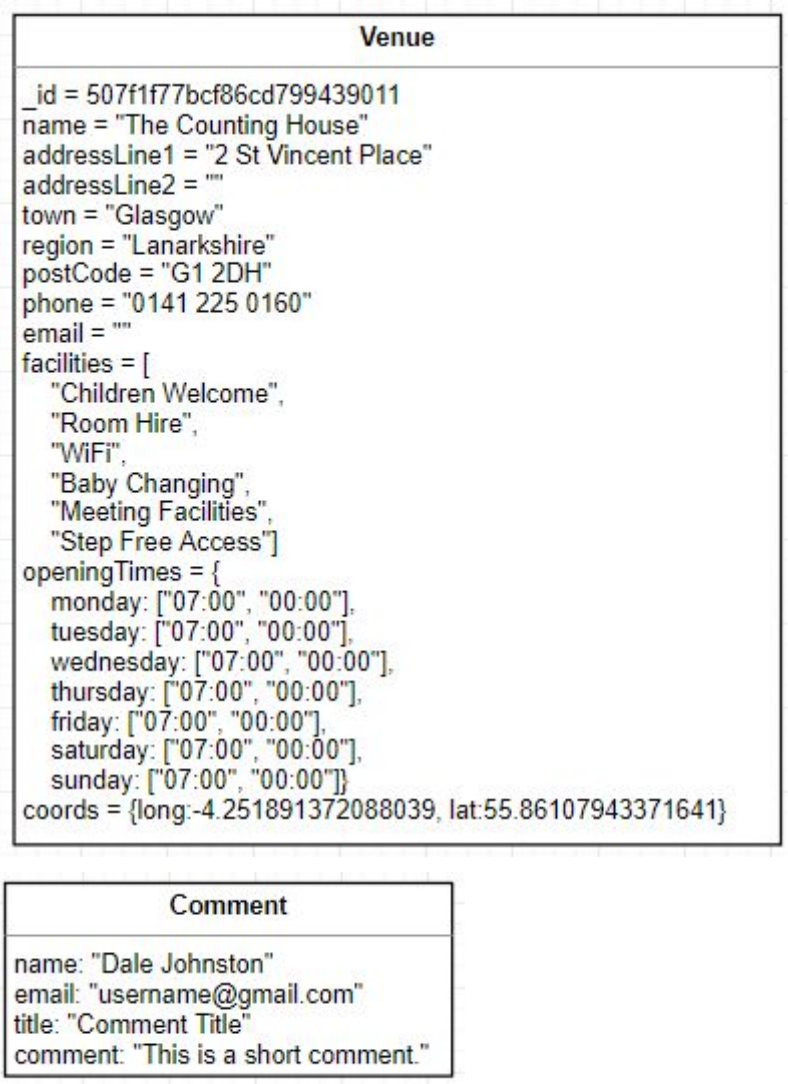
Week 13 / P 7 / System interaction diagrams

Sequence diagrams




User location changed using search box
User
mapWrapper.js
Google Maps API
models/venues.js
controller/venues.js
Database
getAutoComplete().getPlace()
getPlace()
place.geometry.location
place.geometry.location
clearRoute()
clearRouteMarkers()
clearMarkers()
setCenter(place.geometry.location)
setZoom(12)
addMarker(place.geometry.location)
venues.getNearby(place.geometry.location)
HTTP POST place.geometry.location
getAllWithinRangeOf(place.geometry.location)
venuesInRange
venuesInRange
venuesInRange

Week 13 / P 8 / Object diagrams



Week 13 / P9 / Algorithms

Linear search algorithm (FruitMachineProject/FruitMachineViewModel.java):

```
private boolean symbolsMatch(ObservableList<ReelStripViewModel> reelStrips)
{
    boolean match = true;
    SymbolItemViewModel firstSymbol = reelStrips.get(0).getSymbols().get(1);

    for (ReelStripViewModel reelStrip : reelStrips)
    {
        if (reelStrip.getSymbols().get(1).getScore() != firstSymbol.getScore())
        {
            match = false;
        }
    }

    return match;
}
```

This code illustrates a simple usage of a linear search.

Each *ReelStripViewModel* object contains an *ArrayList<SymbolItemViewModel>* which contains a set of fruit machine symbol objects.

Just prior to running this search the *ArrayList<SymbolItemViewModel>* collection on each of 5 *ReelStripViewModel* objects had been rotated a random number of times using the *Collections.rotate()* method to simulate a fruit machine spinning its reels. I used the above algorithm to determine whether the scores of each symbol at index 1 in each *ReelStripViewModel's ArrayList<SymbolItemViewModel>* matched. This would indicate that, for example, the machine was displaying 5 lemons and would result in a win for the player.

Rock/Paper/Scissors move comparator

```
public static int compareMoves(String playerMove, String computerMove)
{
    if(playerMove.equals(computerMove))
    {
        return 0;
    }

    switch(playerMove)
    {
        case "Rock":
            return(computerMove.equals("Scissors") ? 1 : -1);
        case "Paper":
            return(computerMove.equals("Rock") ? 1 : -1);
        case "Scissors":
            return(computerMove.equals("Paper") ? 1 : -1);
        default:
            return 0;
    }
}
```

This function takes in two strings, one representing the player's move and the other representing the computer's move. The moves may be one of 'Rock', 'Paper' or 'Scissors'.

The function returns -1, 0 or 1:

- -1 represents a player loss
- 0 represents a draw
- 1 represents a player win

The function first evaluates if both moves are identical, which produces a draw and returns 0.
The function then uses a switch/case statement to evaluate which of the 'Rock'/'Paper'/ 'Scissors' options the player has used.
Each case statement uses a ternary operator to compare the player's move with the computer's move. In each case, if the computer plays the option which beats the player's move the function returns -1, and if the computer plays the option which loses to the player's move the function returns 1.

A default case is included which returns 0. Barring exceptional circumstances this should never be hit, as the draw condition is evaluated at the beginning of the function.

This is a reasonably concise approach which covers all possible game outcomes. It would be reasonable to amend the default case of the switch statement to return an integer which doesn't represent a game state, e.g. -100.

Week 13 / P 17 / Bug tracking report

User must be able to update their location using the search box.	Failed	Added code to prevent return keypress from submitting entire page as form.	Passed
Application must respond appropriately when user enters unfindable location in search box.	Failed	Added check on location to ensure it exists, and displayed an error alert on unfindable location input.	Passed
User must be able to to create a walking route between venues by holding the Ctrl key while clicking venues on the map.	Failed	Added global cross-platform onkeydown/onkeyup handlers to detect Ctrl keypress.	Passed
User must be able to create a walking (not driving/public transport/etc.) route between venues	Failed	Added travelMode: ‘WALKING’ to MapWrapper.prototype.showRoute method. Google’s map route default is driving.	Passed
Application should correctly retrieve a list of facilities available at each venue from the database.	Failed	Corrected seeds file to populate the database with ‘facilities’, rather than ‘facilites’.	Passed
User must be able to clear a previously created walking route	Failed	First attempt tried to modify the marker collection while looping over it. Corrected this using a while loop to monitor the collection’s length and remove an element on each iteration.	Passed