

Week 2 / IT 5

```
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

```
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.

```
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

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

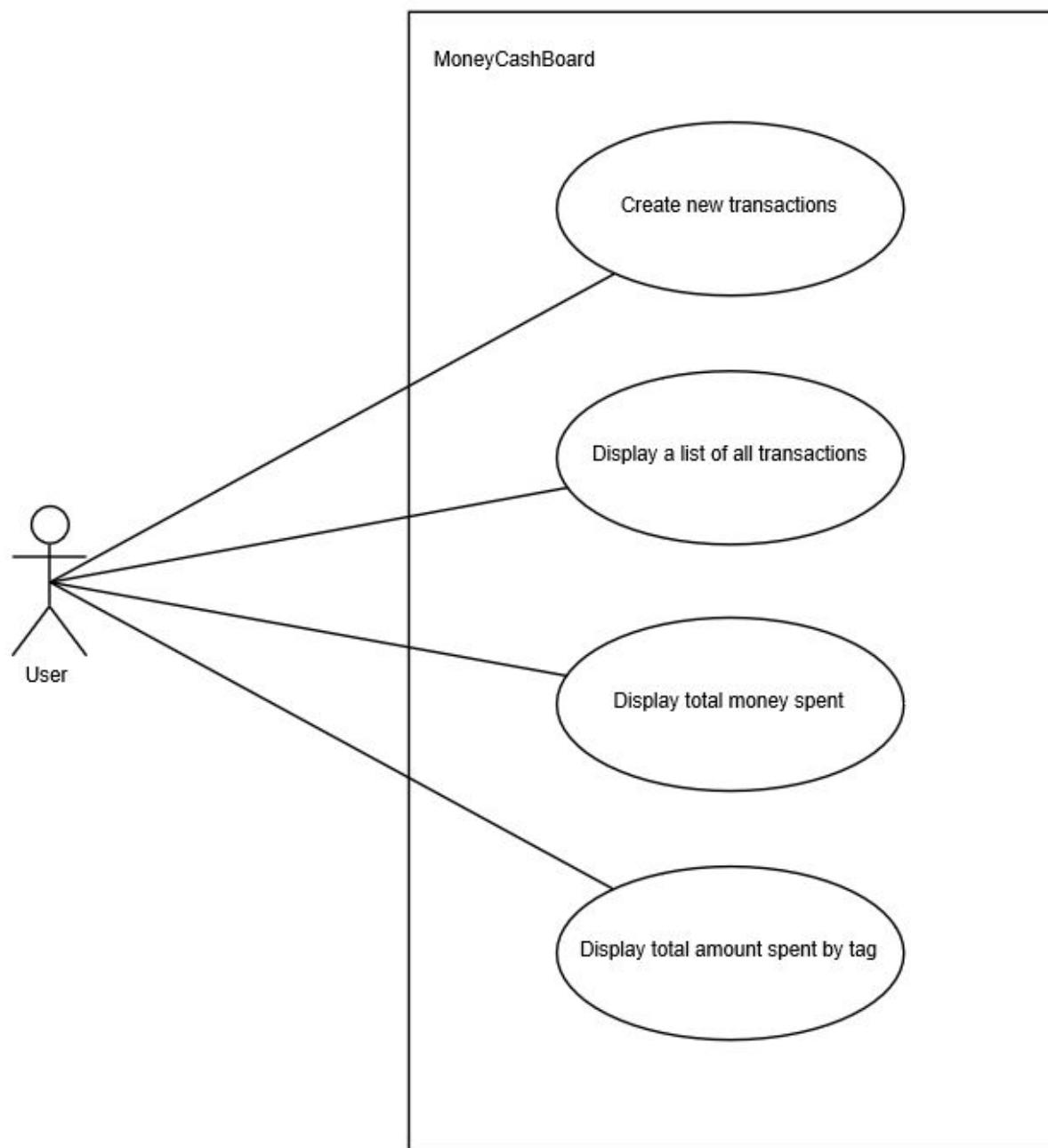
def self.all()
  sql = "SELECT * FROM customers ORDER BY name;"
  customers = SqlRunner.run(sql)
  return customers.map { |customer| Customer.new(customer) }
end
```

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.

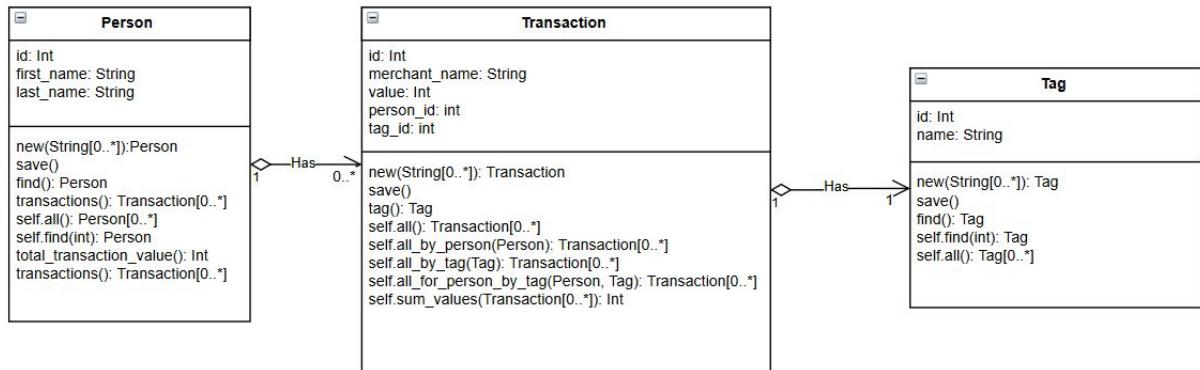
```
[1] pry(main)> Customer.all()
=> [#<Customer:0x007f91348a2100 @id=22, @name="Name 1">, 797 bytes
 #<Customer:0x007f91348a1f70 @id=23, @name="Name 2">]
[2] pry(main)>
```

This demonstrates the result of the function.

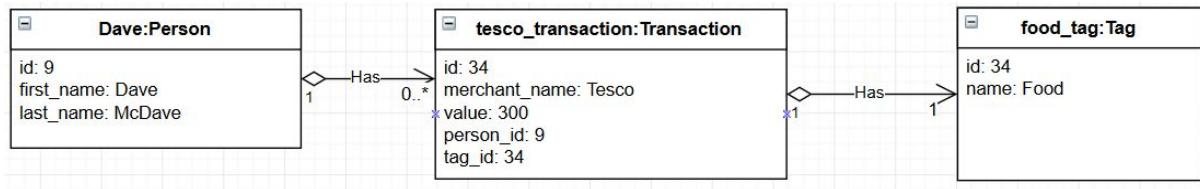
Week 5 / AD 1 / Use Case Diagram



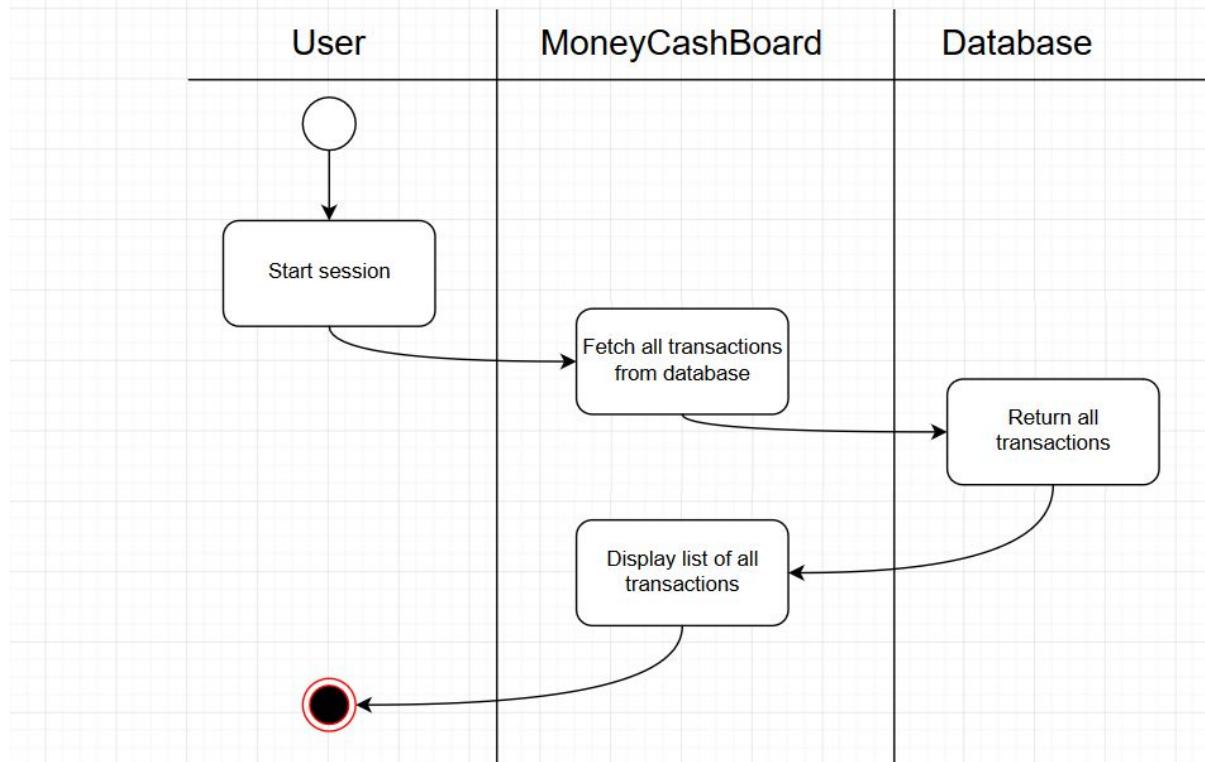
Week 5 / AD 2 / Class Diagram



Week 5 / AD 3 / Object Diagram



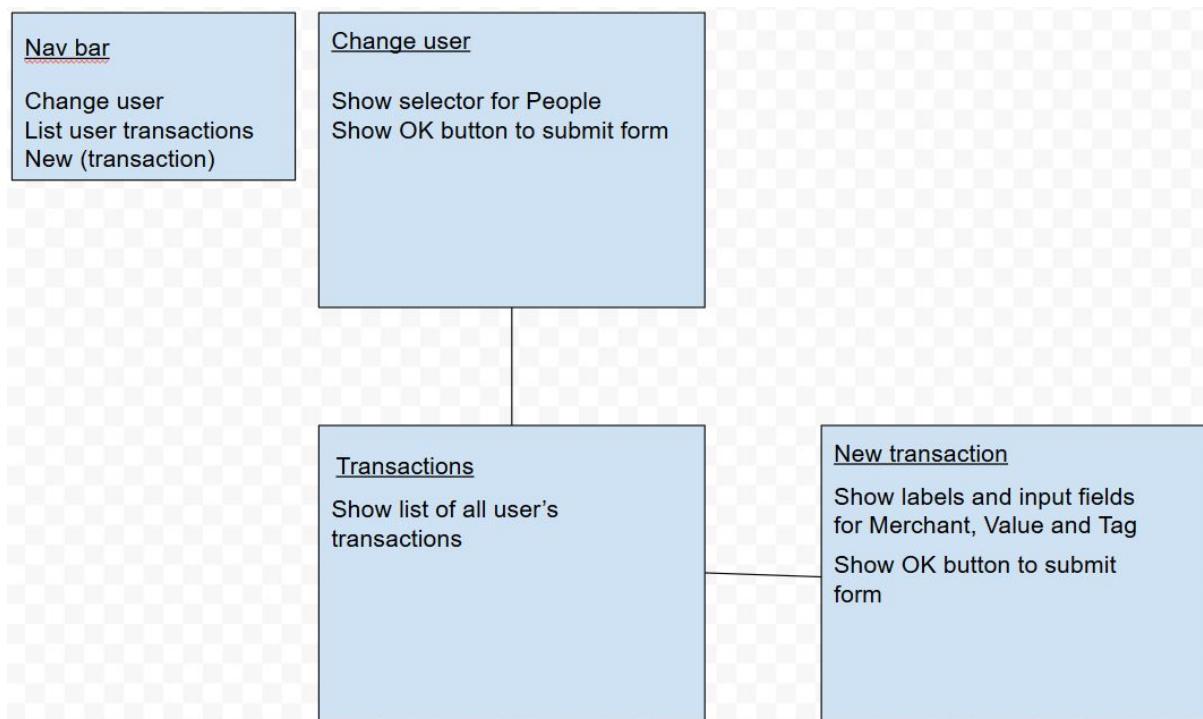
Display a list of all transactions



Week 5 / AD 6 / Implementations Constraints Plan

<u>Constraint</u>	<u>Possible Effect of Constraint on Product</u>	<u>Solution</u>
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)

HEADER TEXT		
<u>Home</u>	<u>New</u>	<u>Home</u>
		<u>New</u>
HEADER TEXT		
<u>Home</u>	<input type="text"/> Merchant name	<input type="text"/> Value
<u>New</u>	<input type="text"/> Tag	<input type="button" value="OK"/>
		<input type="text"/> Merchant name
		<input type="text"/> Value
		<input type="text"/> Tag
		<input type="button" value="OK"/>

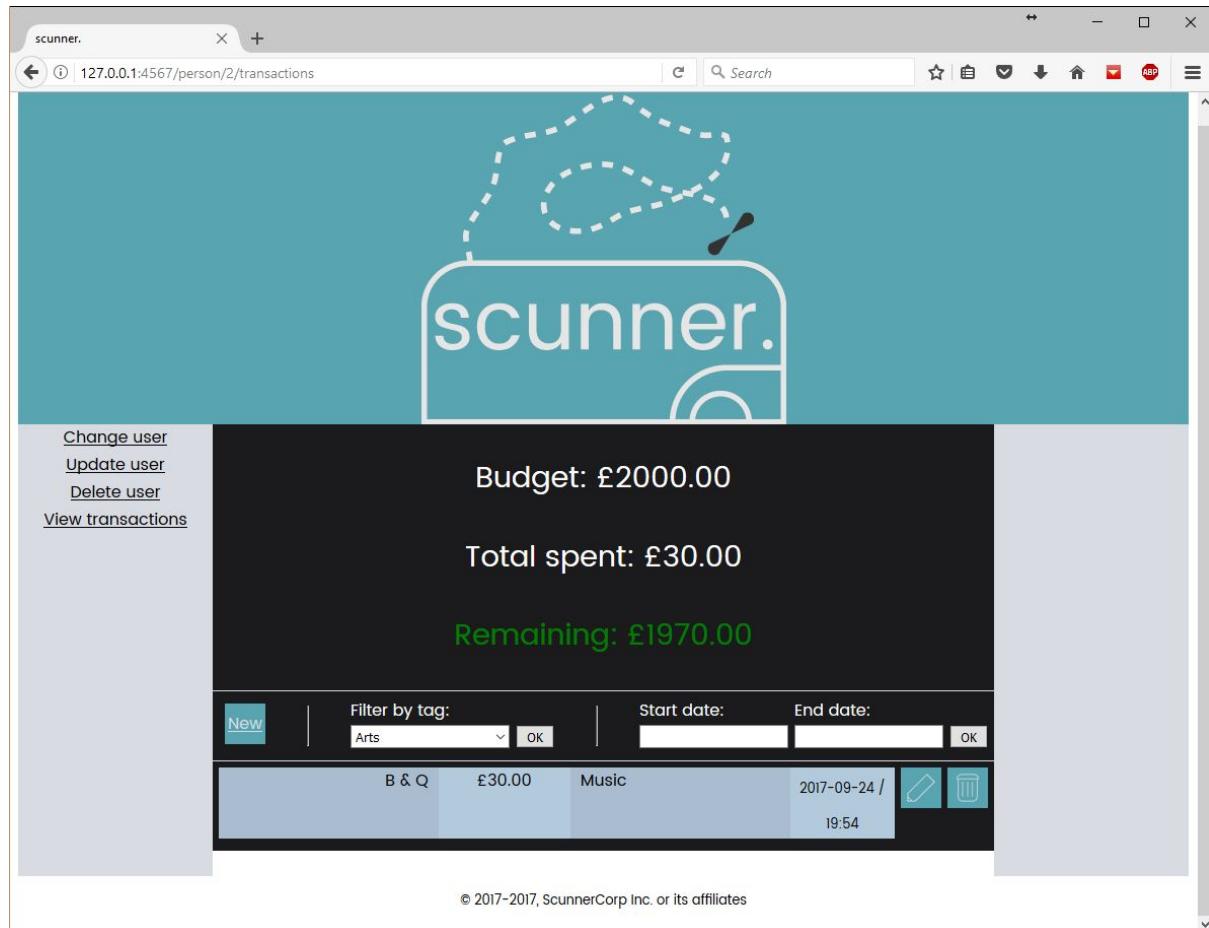
HEADER TEXT		
<u>Home</u>	<u>New</u>	<u>Home</u>
		<u>New</u>
HEADER TEXT		
<u>Home</u>	<u>New</u>	<u>Home</u>
		<u>New</u>
Running total: £839.87		
<input type="text"/> Merchant name 1	<input type="text"/> £18.97	<input type="text"/> Tag name 1
<input type="text"/> Merchant name 2	<input type="text"/> £18.97	<input type="text"/> Tag name 2
<input type="text"/> Merchant name 3	<input type="text"/> £18.97	<input type="text"/> Tag name 3
		<input type="text"/> Merchant 1
		<input type="text"/> £18.97
		<input type="text"/> Tag name 1
		<input type="text"/> Merchant 2
		<input type="text"/> £18.97
		<input type="text"/> Tag name 2
		<input type="text"/> Merchant 3
		<input type="text"/> £18.97
		<input type="text"/> Tag name 3

Week 5 / P 10 / Pseudocode for a function

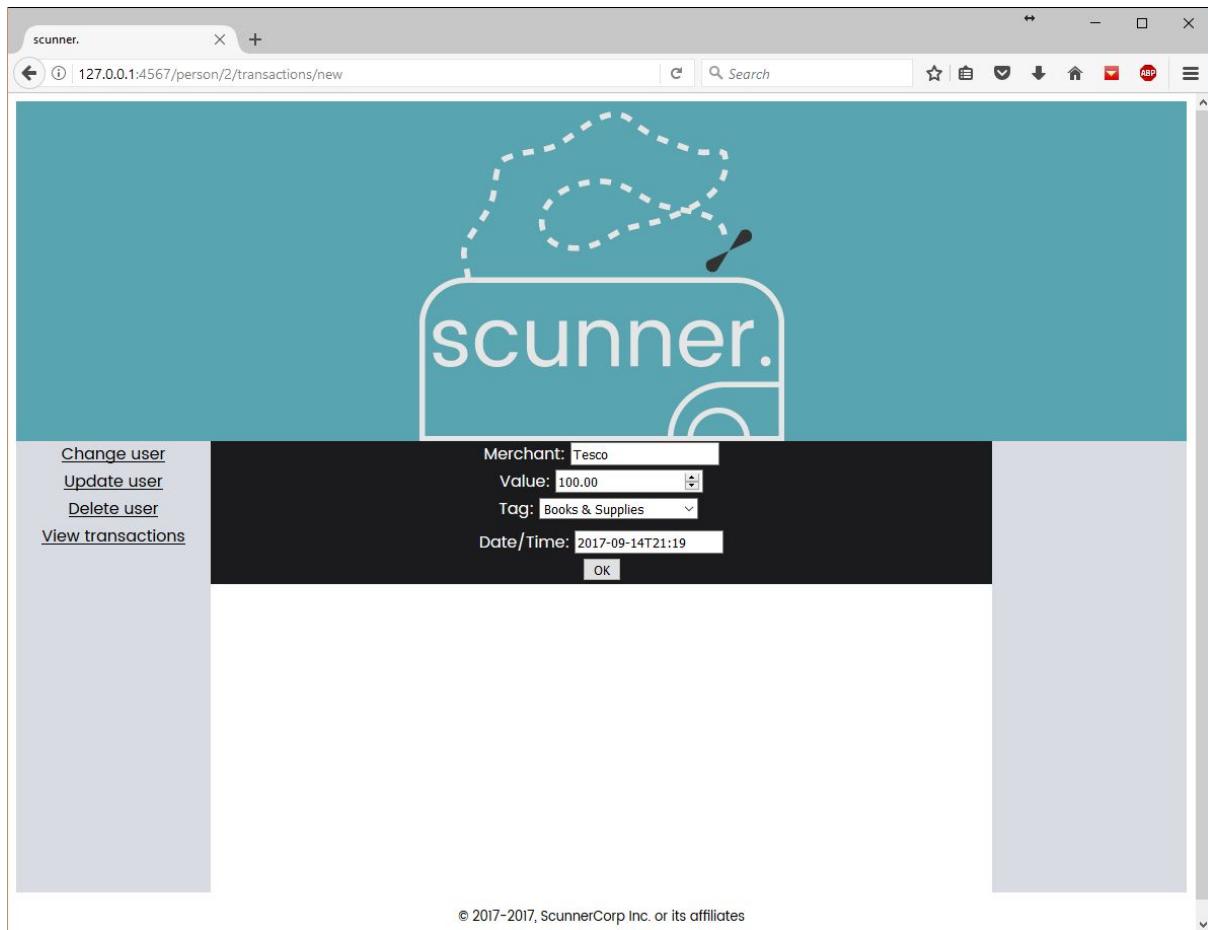
```
1  find_person(person_id_to_find)
2      myQuery = select all from people where person.person_id = person_id_to_find
3      result = run(myQuery)
4      return new Person(result.first)
5 end
```

Week 5 / P 13 & P 14 / User input

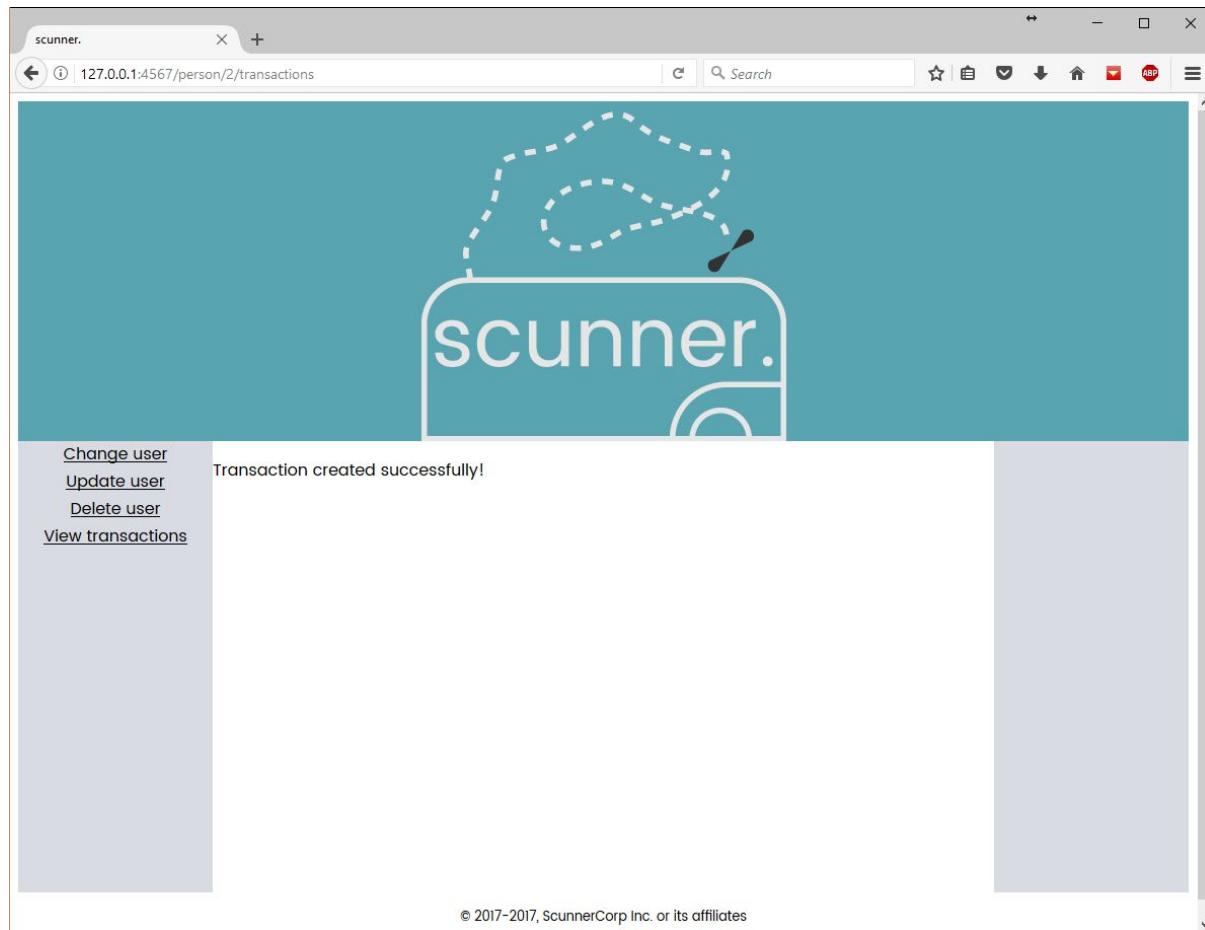
Screenshot of application status before user has created a new transaction



(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:

The screenshot shows a web browser window for the 'scunner.' application. The title bar says 'scunner.' and the address bar shows '127.0.0.1:4567/person/2/transactions'. The main content area features a teal header with the 'scunner.' logo. Below it, a black panel displays financial summary information: 'Budget: £2000.00', 'Total spent: £130.00', and 'Remaining: £1870.00'. To the left of this panel is a sidebar with links: 'Change user', 'Update user', 'Delete user', and 'View transactions'. The main content area also includes a 'New' button, a 'Filter by tag:' dropdown set to 'Arts' with an 'OK' button, and date selection fields for 'Start date:' and 'End date:'. A table lists two transactions: one from 'B & Q' for £30.00, Music, dated 2017-09-24 at 19:54, and another from 'Tesco' for £100.00, Books & Supplies, dated 2017-09-14 at 21:19. Each transaction row has edit and delete icons.

Category	Amount	Description	Date	Time	Actions
B & Q	£30.00	Music	2017-09-24	19:54	
Tesco	£100.00	Books & Supplies	2017-09-14	21:19	

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:

The screenshot shows a web browser window with the URL `127.0.0.1:4567/person/2/transactions`. The page displays a budget summary at the top: **Budget: £2000.00**, **Total spent: £255.00**, and **Remaining: £1745.00**. Below this, there is a filter bar with 'Filter by tag:' dropdown set to 'Music', 'Start date: 2017-01-01', 'End date: 2017-01-29', and an 'OK' button. The main area lists six transactions in a table format:

Merchant	Amount	Category	Date	Actions
B & Q	£30.00	Music	2017-09-24 / 19:54	
Tesco	£100.00	Books & Supplies	2017-09-14 / 21:19	
Sainsburys	£80.00	Music	2017-09-14 / 21:21	
Schu	£40.00	Clothing	2017-01-01 / 09:00	
Amazon	£2.00	Movies & DVDs	2017-01-08 / 10:00	
Ebay	£3.00	Electronics & Software	2017-01-19 / 11:00	

At the bottom of the page, a copyright notice reads: © 2017-2017, ScunnerCorp Inc. or its affiliates.

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:

The screenshot shows a web browser window with a dark-themed budgeting application. At the top left, there's a sidebar with links: 'Change user', 'Update user', 'Delete user', and 'View transactions'. The main area displays financial summary information: 'Budget: £2000.00', 'Total spent: £45.00', and 'Remaining: £1745.00'. Below this, a modal dialog is open with fields for 'New' transaction, category 'Arts', 'Start date: 2017-01-01', and 'End date: 2017-01-29'. The transaction list table has columns for Category, Amount, Description, Date, and Action icons (edit and delete). Three entries are listed: 1) Schu £40.00 Clothing 2017-01-01T09:00, 2) Amazon £2.00 Movies & DVDs 2017-01-08T10:00, and 3) Ebay £3.00 Electronics & Software 2017-01-19T11:00.

Category	Amount	Description	Date	Action
Schu	£40.00	Clothing	2017-01-01T09:00	
Amazon	£2.00	Movies & DVDs	2017-01-08T10:00	
Ebay	£3.00	Electronics & Software	2017-01-19T11:00	

© 2017-2017, ScunnerCorp Inc. or its affiliates

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);
        }
    }
}

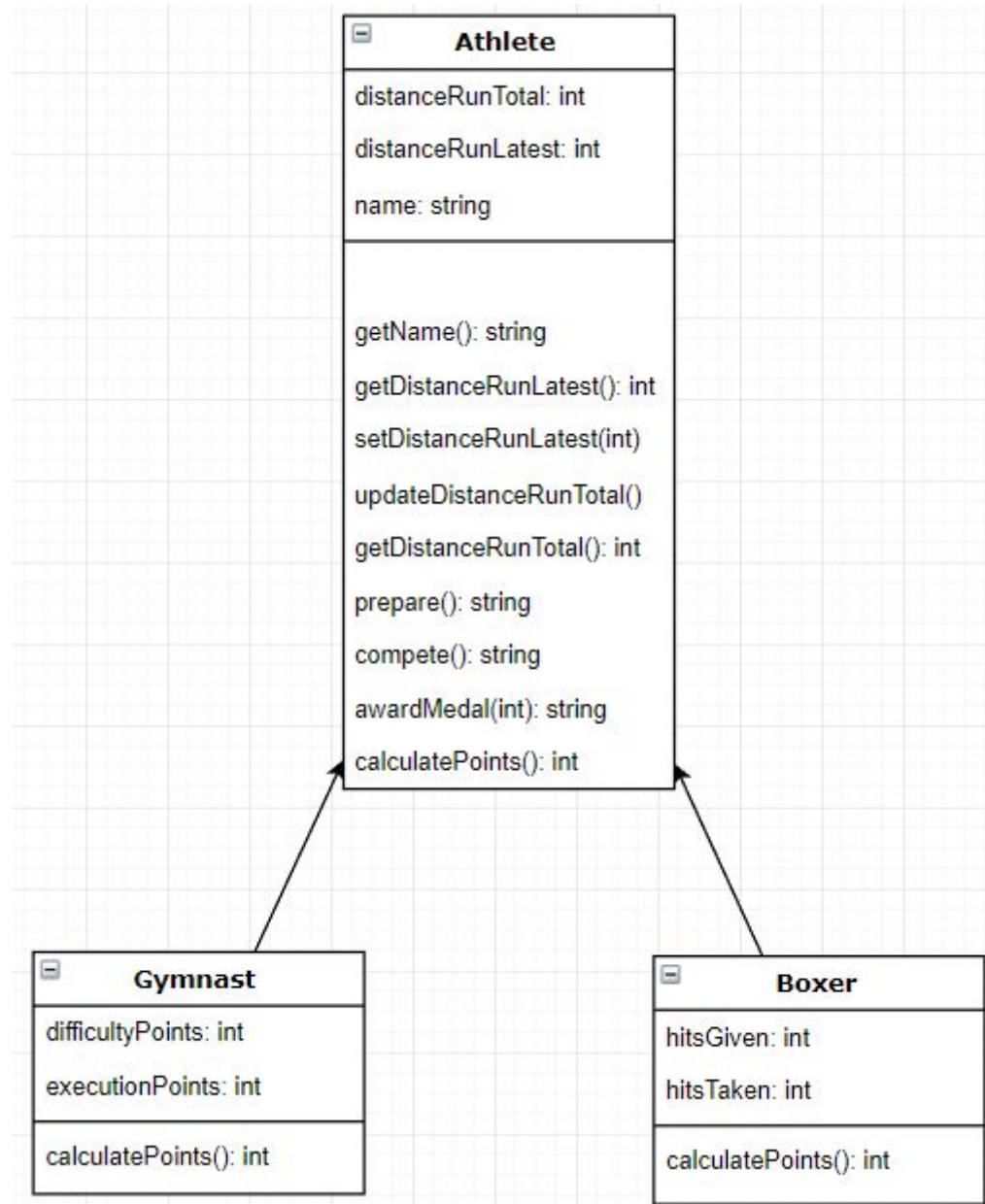
public class TenPercentOnTwentyPoundsDiscount implements IBasketDiscount
{
    @Override
    public int getValue(Basket basket, int currentValue)
    {
        if(currentValue > 2000)
        {
            return (int)(currentValue * 0.1);
        }
        else
        {
            return 0;
        }
    }
}
```

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>

The screenshot shows the GitHub repository page for 'FruitMachineProject'. At the top, there's a header with the repository name, a 'Unwatch' button (1 watch), a 'Star' button (0 stars), and a 'Fork' button (0 forks). Below the header, there are tabs for 'Code', 'Issues 0', 'Pull requests 0', 'Projects 0', 'Wiki', 'Settings', and 'Insights'. A dropdown menu shows the branch is 'master'. At the bottom of the header, there are buttons for 'Create new file', 'Upload files', 'Find file', and 'History'. The main content area shows a list of commits from user 'goobering'. The latest commit is 'cdef993 on 13 Jul' with the message 'Got some visibility bindings working on FruitMachineViewModel and ass...'. Below it is a list of commits for various subdirectories: 'activities', 'adapters', 'controls', 'database', 'disk', 'enums', 'helpers', 'interfaces', 'models', 'services', and 'viewmodels', all dated '2 months ago'.

Commit	Message	Date
goobering	Got some visibility bindings working on FruitMachineViewModel and ass...	Latest commit cdef993 on 13 Jul
..		
activities	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
adapters	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
controls	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
database	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
disk	Initial commit	2 months ago
enums	Initial commit	2 months ago
helpers	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
interfaces	Added PlayFruitMachineViewModel, associated layout and supplementary ...	2 months ago
models	Added PlayFruitMachineViewModel, associated layout and supplementary ...	2 months ago
services	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago
viewmodels	Got some visibility bindings working on FruitMachineViewModel and ass...	2 months ago

The brief for the project was as follows:

Goal: Create a fruit machine in Android.

Last week, for your homework you were asked to create a fruit machine using objects modelled in Java. You are now being asked to take that Java code and use it as the basis for an Android app which simulates the same fruit machine.

MVP

The aim is to display the results of the Java logic already written. For example, if the user spins and lands 3 matching symbols a winning message is displayed and their winnings updated and stored. Users should also be allowed to nudge and hold at random times.

Project Extensions

- Display the symbols above and below as well to help with nudges
- Allow users to choose from Symbol Packs (Different games.)
- Extend to 5 reels
- Improve the UI

Screenshots of Trello board planning:

The image displays two screenshots of a Trello board titled "FruitMachine".

Top Screenshot: This screenshot shows four columns of cards:

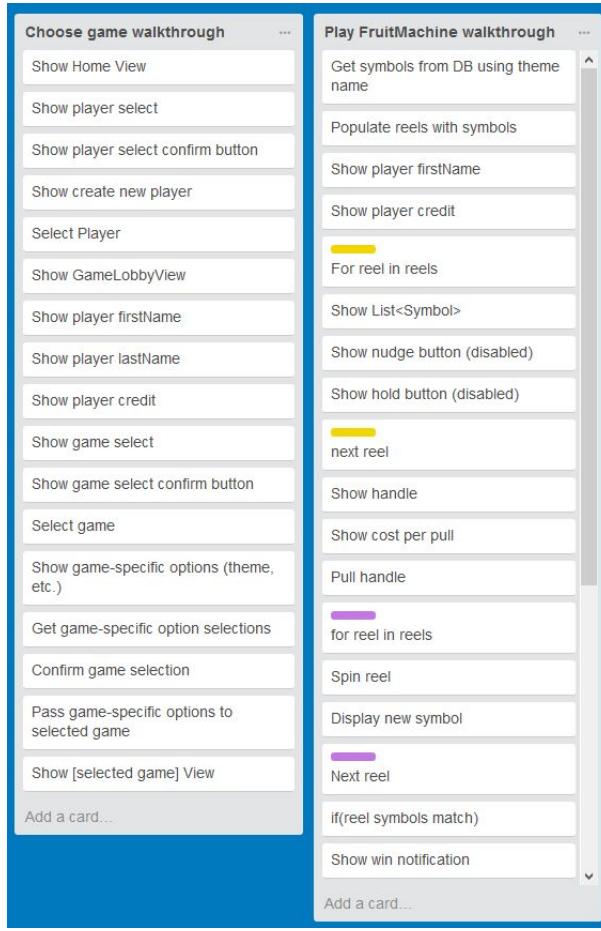
- Class / Player**: Contains fields for int id, String firstName, String lastName, String email, int bank, and ArrayList<int> gameScores.
- Abstract / GameScore**: Contains fields for int id, String name, DateTime playedAt, and int playerId.
- Class / FruitMachineScore implements GameScore**: Contains fields for int score.
- Abstract / Game**: Contains fields for String name and Player player.

Bottom Screenshot: This screenshot shows three columns of cards:

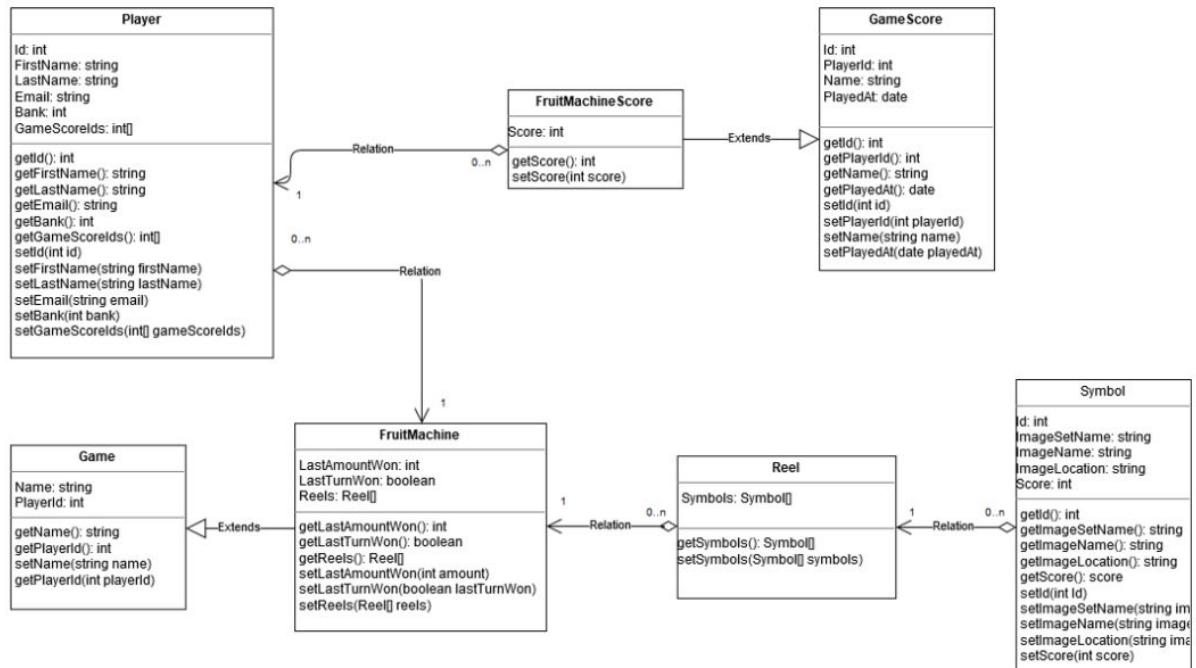
- Class / FruitMachine**: Contains fields for int lastAmountWon, boolean lastTurnWon, and ArrayList<Reel> reels.
- Class / Reel**: Contains field ArrayList<Symbol> symbols.
- Class / Symbol**: Contains fields for int image and int score.

On the right side of the bottom screenshot, there is a vertical column of cards titled "Create player walkthrough" which lists various steps:

- Show Home View
- Show player select
- Show player select confirm button
- Show create new player
- Select create new player
- Show CreatePlayer View
- Show edit_first_name
- Show edit_last_name
- Show edit_email
- Show btn_commit
- Get input for firstName
- Get input for lastName
- Get input for email
- Confirm form completion
- Create new Player
- Write new Player to DB/Players
- Return to Home View

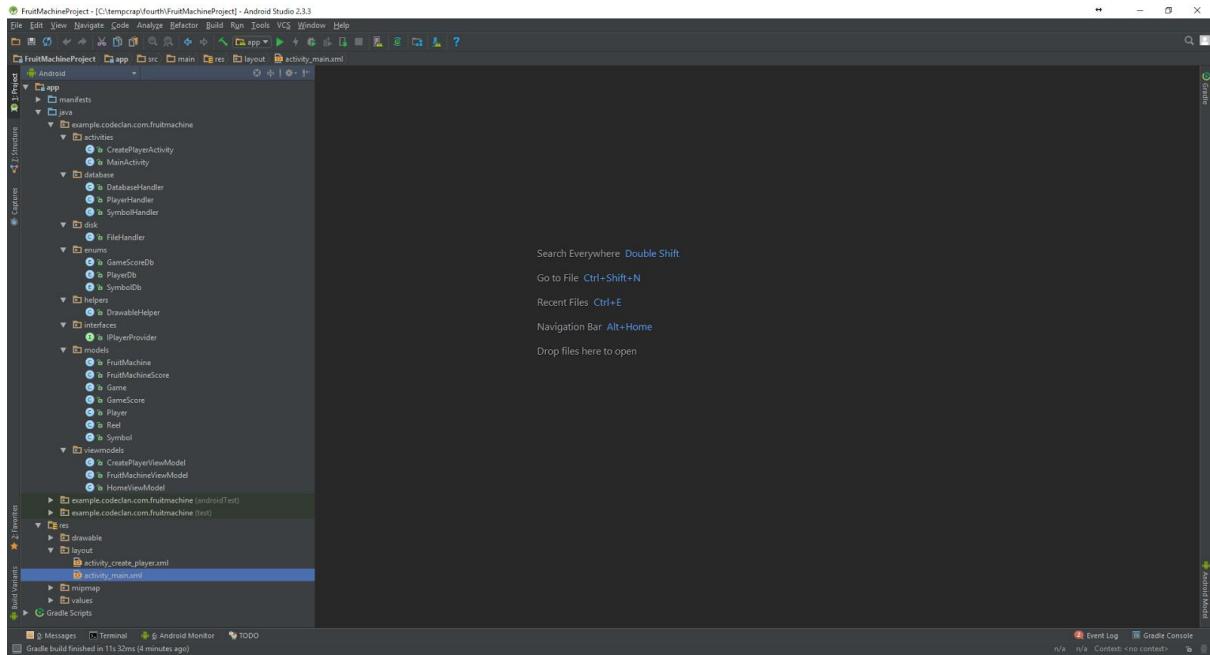


Class/relationship diagram:

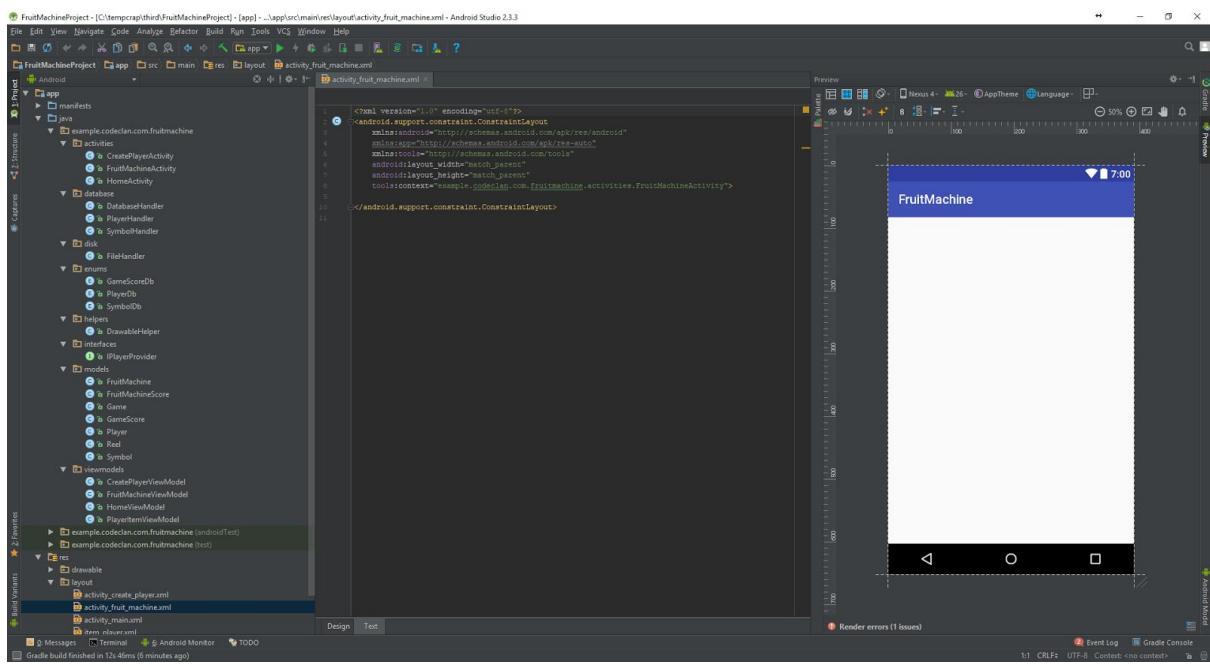


Progress:

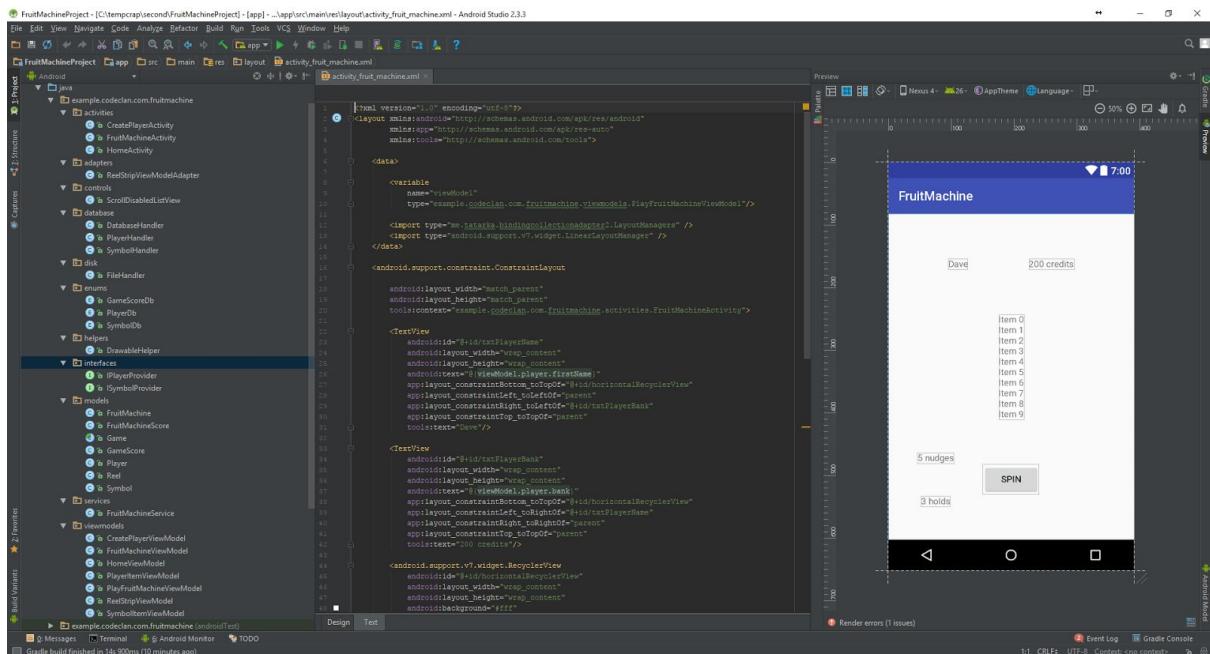
Second commit. Limited number of files in the explorer pane on the left, no activity_fruit_machine.xml file yet.



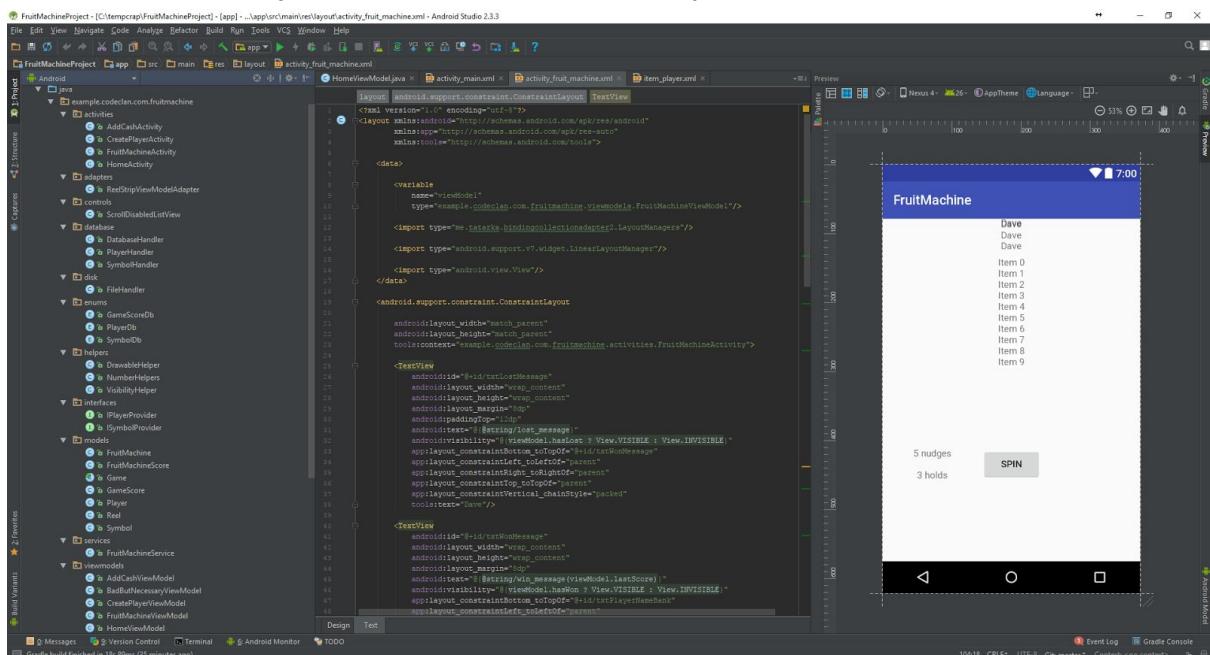
Third commit. Larger number of files created, and a blank activity_fruit_machine.xml interface.



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:

```
JS customer.js ● JS testCustomer.js {} package.json
 1  var _ = require('lodash');
 2
 3  var Customer = function(){
 4    this.balance = 0;
 5    this.inventory = [];
 6  };
 7
 8  Customer.prototype = {
 9
10  };
11
12  module.exports = Customer;
```

Testing class:

```
JS testCustomer.js ✘
1  var assert = require('assert');
2  var Customer = require('../customer.js');
3  var BookStore = require('../bookStore.js');
4  var Book = require('../book.js');
5
6  describe('Customer', function(){
7      var customer, testCustomer1;
8      var testBook1, testBook2, testBook3;
9      var testBookStore1;
10
11     beforeEach(function(){
12         testBook1 = new Book('testAuthor1', 'testTitle1', 'testGenre1', 10, 100);
13         testBook2 = new Book('testAuthor2', 'testTitle2', 'testGenre2', 20, 200);
14         testBook3 = new Book('testAuthor3', 'testTitle3', 'testGenre3', 30, 300);
15
16         customer = new Customer();
17         customer.balance = 200;
18         customer.inventory.push(testBook1);
19         customer.inventory.push(testBook2);
20         customer.inventory.push(testBook3);
21
22         testCustomer1 = new Customer();
23         testCustomer1.balance = 100;
24         testCustomer1.inventory.push(testBook1);
25         testCustomer1.inventory.push(testBook1);
26         testCustomer1.inventory.push(testBook1);
27
28         testBookStore1 = new BookStore('testName1', 'testCity1');
29
30         testBookStore1.addBook(testBook1);
31         testBookStore1.addBook(testBook2);
32         testBookStore1.addBook(testBook3);
33     });
34
35     it('has an inventory', function(){
36         assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3]);
37     });
});
```

```
38     it('has a balance', function(){
39         assert.strictEqual(customer.balance, 200);
40     });
41
42     it('can buy a Book', function(){
43         customer.buy(testBookStore1, testBook1);
44
45         assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3, testBook1]);
46         assert.strictEqual(customer.balance, 190);
47     });
48
49     it('can sell a book', function(){
50         testBookStore1.balance = 200;
51
52         var sold = customer.sell(testBookStore1, testBook1);
53
54         assert.strictEqual(sold, true);
55         assert.strictEqual(customer.balance, 210);
56         assert.deepEqual(customer.inventory, [testBook2, testBook3]);
57     });
58
59     it('can return the total value of their collection', function(){
60         assert.strictEqual(customer.getInventoryValue(), 60);
61     });
62
63     it('can view the total value of all books of a given Genre', function(){
64         assert.strictEqual(customer.getValueOfGenre('testGenre1'), 10);
65     });
66
67     it('can view their longest book', function(){
68         assert.strictEqual(customer.getValueOfLongestBook(), 30);
69     });
70
71     it('can sort their books by value (ascending)', function(){
72         customer.sortInventoryByValue(true);
73         assert.deepEqual(customer.inventory, [testBook1, testBook2, testBook3]);
74     });
75
76     it('can sort their books by value (descending)', function(){
77         customer.sortInventoryByValue(false);
78         assert.deepEqual(customer.inventory, [testBook3, testBook2, testBook1]);
79     });
80
81     it('should be able to compare the value of their collection with another BookWorm', function(){
82         assert.strictEqual(customer.compareWith(testCustomer1), 30);
83     });
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

```
js customer.js ✘  js testCustomer.js  {} package.json
 1  var _ = require('lodash');
 2
 3  var Customer = function(){
 4      this.balance = 0;
 5      this.inventory = [];
 6  };
 7
 8  Customer.prototype = {
 9      canBuy: function(book){
10          return this.balance >= book.price;
11      },
12      canSell: function(book){
13          return _.includes(this.inventory, book);
14      },
15      buy: function(bookStore, book){
16          if(bookStore.canSell(book) && this.canBuy(book)){
17              bookStore.sell(book, this);
18              this.inventory.push(book);
19              this.balance -= book.price;
20              return true;
21          } else {
22              return false;
23          };
24      },
25      sell: function(bookStore, book){
26          if(bookStore.canBuy(book) && this.canSell(book)){
27              bookStore.buy(book, this);
28              _.remove(this.inventory, book);
29              this.balance += book.price;
30              return true;
31          } else {
32              return false;
33          };
34      },
35      getInventoryValue: function(){
36          return _.sumBy(this.inventory, 'price');
37      },
38      getValueOfGenre: function(genre){
39          return _.sumBy(_.filter(this.inventory, {genre: genre}), 'price');
40      },
41      getValueOfLongestBook: function(){
42          return _.maxBy(this.inventory, 'pageCount').price;
43      },
44      sortInventoryByValue: function(isAscending){
45          if(isAscending){
46              this.inventory = _.sortBy(this.inventory, 'price');
47          } else {
48              this.inventory = _.sortBy(this.inventory, 'price').reverse();
49          }
50      },
51      compareWith: function(customer){
52          return this.getInventoryValue() - customer.getInventoryValue();
53      }
54  };
55
56  module.exports = Customer;
```

Test code passing

Customer

- ✓ has an inventory
- ✓ has a balance
- ✓ can buy a Book
- ✓ can sell a book
- ✓ can return the total value of their collection
- ✓ can view the total value of all books of a given Genre
- ✓ can view their longest book
- ✓ can sort their books by value (ascending)
- ✓ can sort their books by value (descending)
- ✓ should be able to compare the value of their collection with another BookWorm

10 passing (20ms)

Week 11 / P 16 / APIs

Code that uses an API:

(also stored at: https://github.com/goobering/api_calls/blob/master/public/index.js)

```
1  var windowLoaded = function(){
2      var SpeechRecognition = SpeechRecognition || webkitSpeechRecognition
3      var SpeechRecognitionEvent = SpeechRecognitionEvent || webkitSpeechRecognitionEvent
4
5      var recognition = new SpeechRecognition();
6      recognition.lang = 'en-GB';
7      recognition.interimResults = false;
8      recognition.maxAlternatives = 1;
9
10     var diagnostic = document.querySelector('#output');
11     var startTag = document.querySelector('#start-speech');
12     var startLabel = document.querySelector('#start-label');
13
14     startTag.onclick = function(){
15         // Clear out Wikipedia extract
16         var wikiDiv = document.querySelector('#wiki-extract');
17         wikiDiv.innerHTML = '';
18
19         // Start speech recognition
20         recognition.start();
21     }
22
23     // Called when the user has stopped speaking and parsed result returned
24     recognition.onresult = function(event){
25         var last = event.results.length - 1;
26         var phrase = event.results[last][0].transcript;
27
28         // Send the parsed string to an interface element
29         diagnostic.textContent = phrase;
30
31         var jsonPhrase = {"phrase": phrase};
32         getYoutubeResult(jsonPhrase);
33         getWikipediaResult(jsonPhrase);
34     };
35
36     recognition.onaudiostart = function() {
37         console.log('Audio capturing started');
38
39         startTag.style.backgroundColor = 'red';
40         startTag.style.color = 'white';
41         startLabel.textContent = 'Recording';
42     };
43 }
```

```
44  recognition.onaudioend = function() {
45    console.log('Audio capturing ended');
46
47    startTag.style.backgroundColor = 'gold';
48    startTag.style.color = 'lightslategrey';
49    startLabel.textContent = 'Start';
50  };
51
52  recognition.onspeechend = function(){
53    recognition.stop();
54  };
55
56  recognition.onerror = function(event){
57    diagnostic.textContent = 'Error occurred in recognition: ' + event.error;
58  };
59};
60
61 window.addEventListener('load', windowLoaded);
62
63 //searchTerm needs to be an object with a value representing the desired search
64 var getYoutubeResult = function(searchTerm){
65   var stringTerm = searchTerm.phrase;
66   var url = 'http://localhost:3000/youtube/' + stringTerm;
67   makeRequest(url, youtubeRequestComplete);
68 };
69
70 var getWikipediaResult = function(searchTerm){
71   var stringTerm = encodeURIComponent(searchTerm.phrase.trim());
72   var url = 'https://en.wikipedia.org/w/api.php?action=query&prop=extracts&format=json&exintro=&titles=' + stringTerm + '&origin=*';
73
74   var xhr = createCORSRequest('GET', url);
75   if (!xhr) {
76     alert('CORS not supported');
77     return;
78   }
79
80   // Response handlers.
81   xhr.onload = function(){
82     if(this.status !== 200)
83       return;
84
85     var responseText = xhr.responseText;
86     var jsonText = JSON.parse(responseText);
87     buildWiki(jsonText);
88   }
89 }
```

```
90     xhr.onerror = function() {
91         alert('Woops, there was an error making the request.');
92     };
93
94     xhr.send();
95 }
96
97 // Create the XHR object.
98 function createCORSRequest(method, url) {
99     var xhr = new XMLHttpRequest();
100    if ("withCredentials" in xhr) {
101        // XHR for Chrome/Firefox/Opera/Safari.
102        xhr.open(method, url, true);
103    } else if (typeof XDomainRequest != "undefined") {
104        // XDomainRequest for IE.
105        xhr = new XDomainRequest();
106        xhr.open(method, url);
107    } else {
108        // CORS not supported.
109        xhr = null;
110    }
111    return xhr;
112 }
113
114 var makeRequest = function(url, callback){
115     var request = new XMLHttpRequest();
116     request.open('GET', url);
117     request.addEventListener('load', callback);
118     request.send();
119 };
120
121 var youtubeRequestComplete = function(){
122     if(this.status !== 200)
123         return;
124
125     var jsonString = this.responseText;
126     var wtf = JSON.parse(jsonString);
127     var result = JSON.parse(wtf);
128
129     buildYoutube(result.items[0].id.videoId);
130 };
131
```

```

132 var buildYoutube = function(videoId){
133     console.log('videoId: ' + videoId);
134     console.log('this.YT: ' + this.YT);
135     if(this.YT){
136         this.YT.get('ytplayer').loadVideoById(videoId);
137         return;
138     }
139
140     // Load the IFrame Player API code asynchronously.
141     var tag = document.createElement('script');
142     tag.src = "https://www.youtube.com/iframe_api";
143     var firstScriptTag = document.getElementsByTagName('script')[0];
144     firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);
145
146     var player;
147
148     window.onYouTubeIframeAPIReady = function() {
149         player = new YT.Player('ytplayer', {
150             attr: { type: 'text/css', href: 'main.css', rel: 'stylesheet' },
151             videoId: videoId,
152             events: {
153                 'onReady': onPlayerReady,
154                 'onStateChange': onPlayerStateChange
155             }
156         });
157     };
158
159     function onPlayerReady(event) {
160         console.log('player: ' + player);
161         console.log('player.videoId: ' + player.b.c.videoId);
162         event.target.playVideo();
163     };
164
165     var done = false;
166     function onPlayerStateChange(event) {
167         if (event.data == YT.PlayerState.ENDED) {
168             //Get rid of the player
169             event.target.destroy();
170         };
171     };
172
173     function stopVideo() {
174         player.stopVideo();
175     };
176 }

```

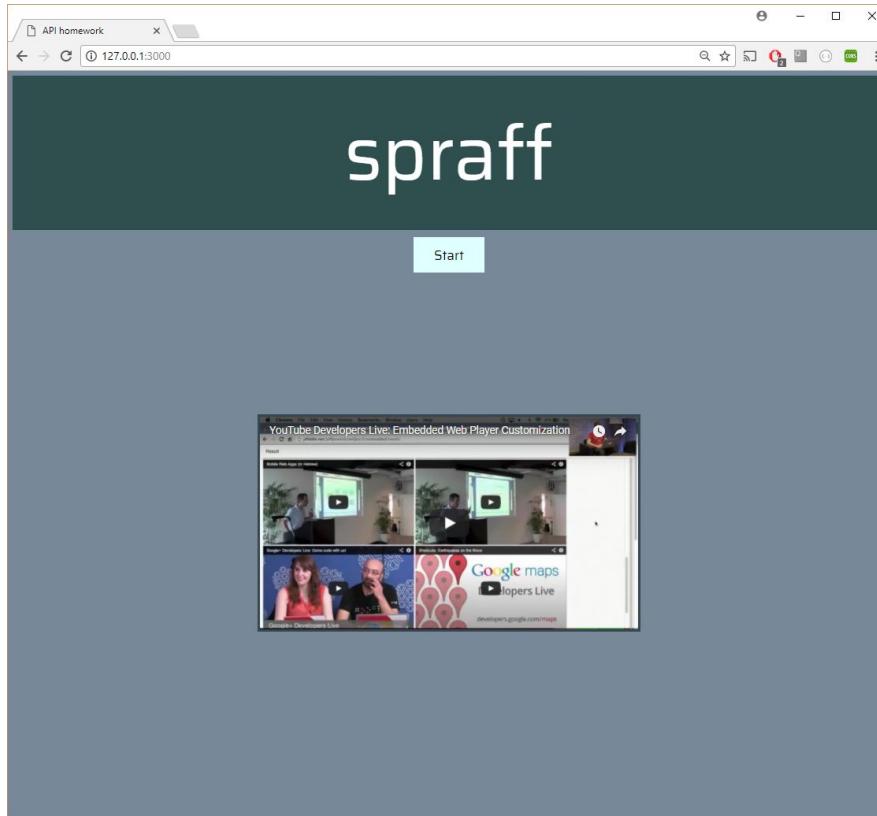
```

177
178 var buildWiki = function(wikiExtract){
179     if(Object.entries(wikiExtract.query.pages)[0][1].extract){
180         var wikiDiv = document.querySelector('#wiki-extract');
181
182         var wikiTag = document.createElement('p');
183         var text = Object.entries(wikiExtract.query.pages)[0][1].extract.replace(/<[^>]+>/gi,"");
184         wikiTag.innerText = text;
185
186         wikiDiv.appendChild(wikiTag);
187     };
188 };

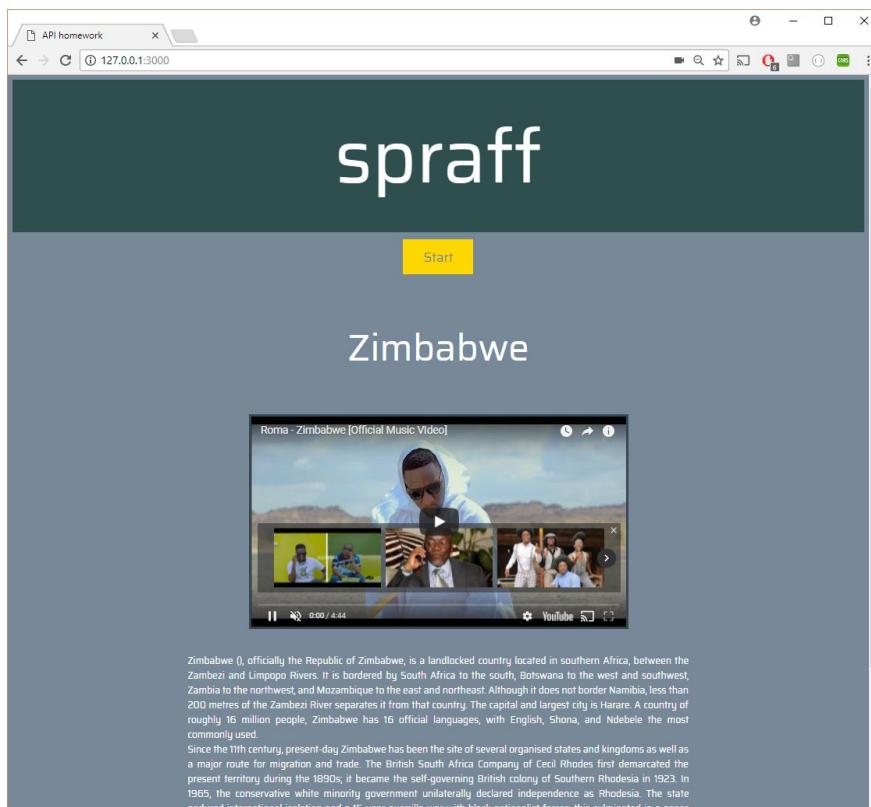
```

API being used by the program:

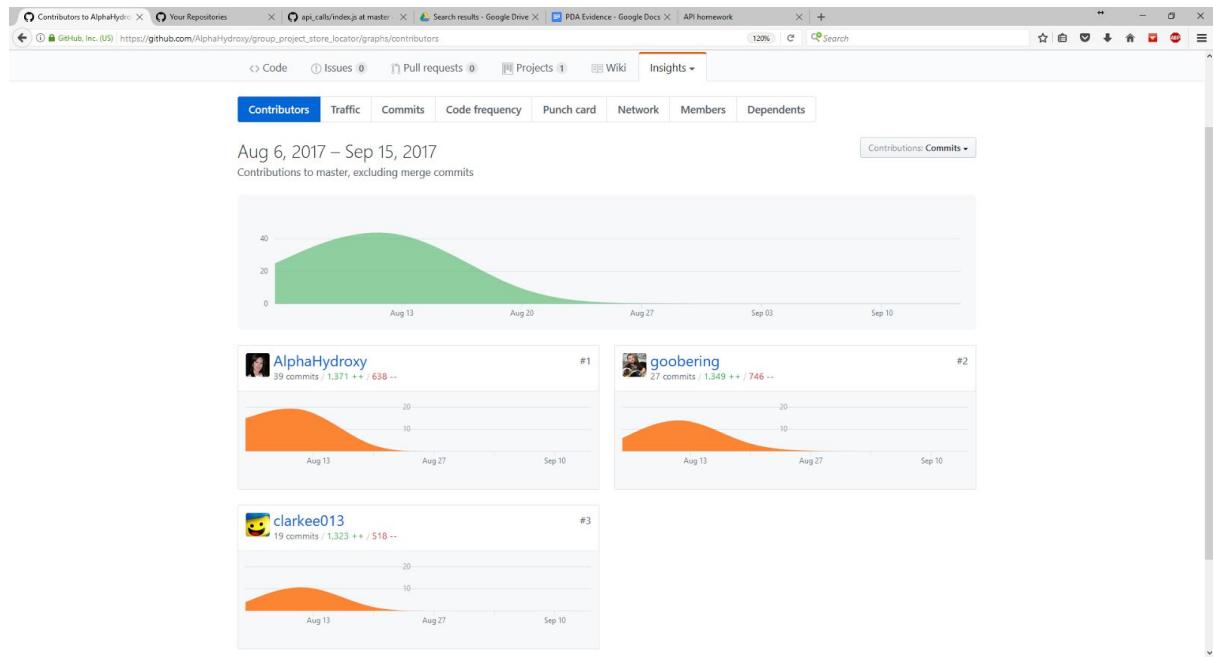
Before voice/phrase recognition:



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

Must

- User should be able to enter their postcode or town or street name and get a list of the top ten nearest stores
- Each store should have a distance, address, phone number, opening hours and a list of facilities and services (e.g. cash machine, car parking).
- Build your own API to persist the store data
- Use an external api to find addresses from postcodes
- Show a map of the stores
- Use Geolocation to allow users to "use current location" to find stores
- THINK OF A GOOD APP NAME

Could

- Add admin pages (password protected?) to manage stores in API
- DEPLOY! Host site externally (Amazon AWS/Microsoft Azure/etc. etc.)
- Custom map markers for stores with different facilities
- Sign in with social media (this is fraught with difficulty and much swearing)
- Disqus
- Create driving route that gets you to a store for opening time/before closing time
- Save favourite places
- Websockets????? live data?
- Event calendar for bar
- Feature: 'Where's open and closest right now?' / 'Where can I find [x] drink/meal?'
- Feature: 'I'm going to [x] pub at [y] time - here are directions' - send message to buddy
- advanced search...choice of radius'
- Uber/public transport/taxi link somewhere - 'HOW DO I GET

Would

- TDD!!!
- Add a card...

Maps project

Team Wheezy

Team Visible

MongoDB

Collection of Venues

1

Each Venue needs:

2

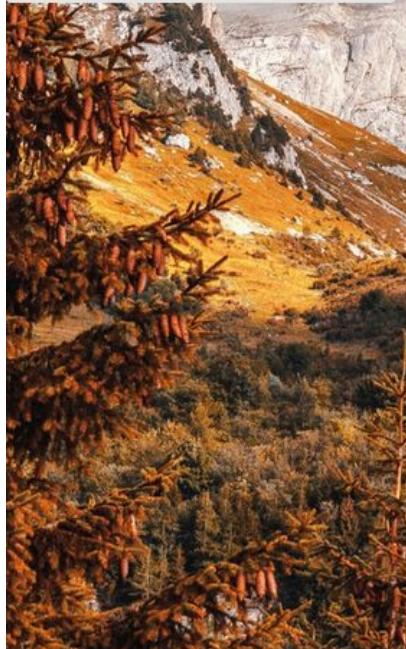
All these things should go in the Seeds.js file for initial population.

1

methods needed:

1 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

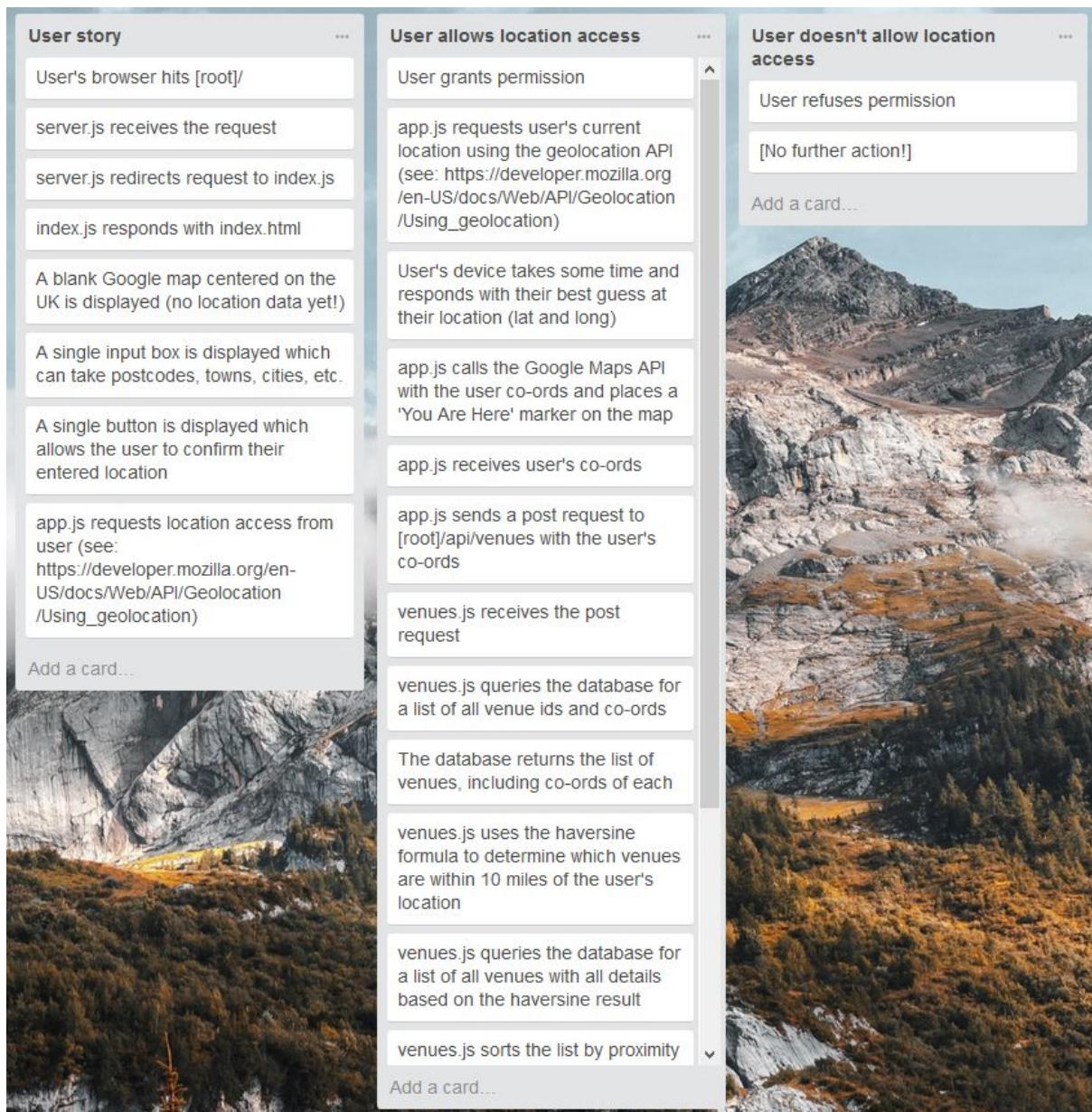
[root]/ : Returns index.html

[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...





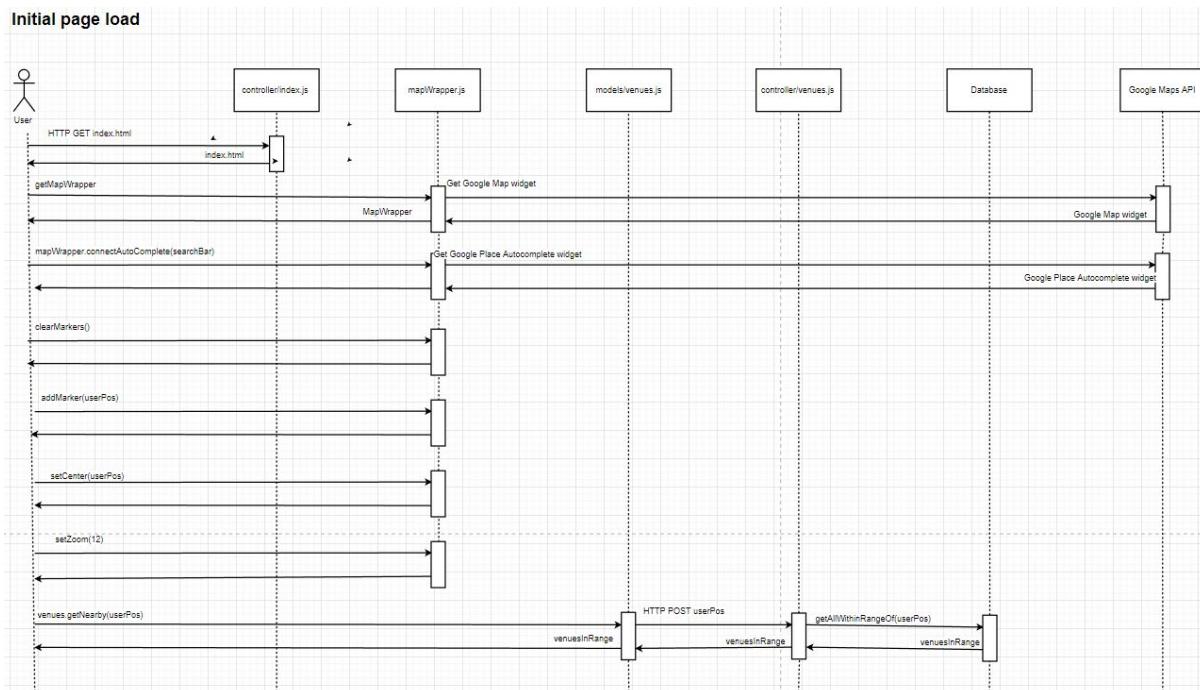
Week 13 / P 4 / Acceptance criteria and test plan

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

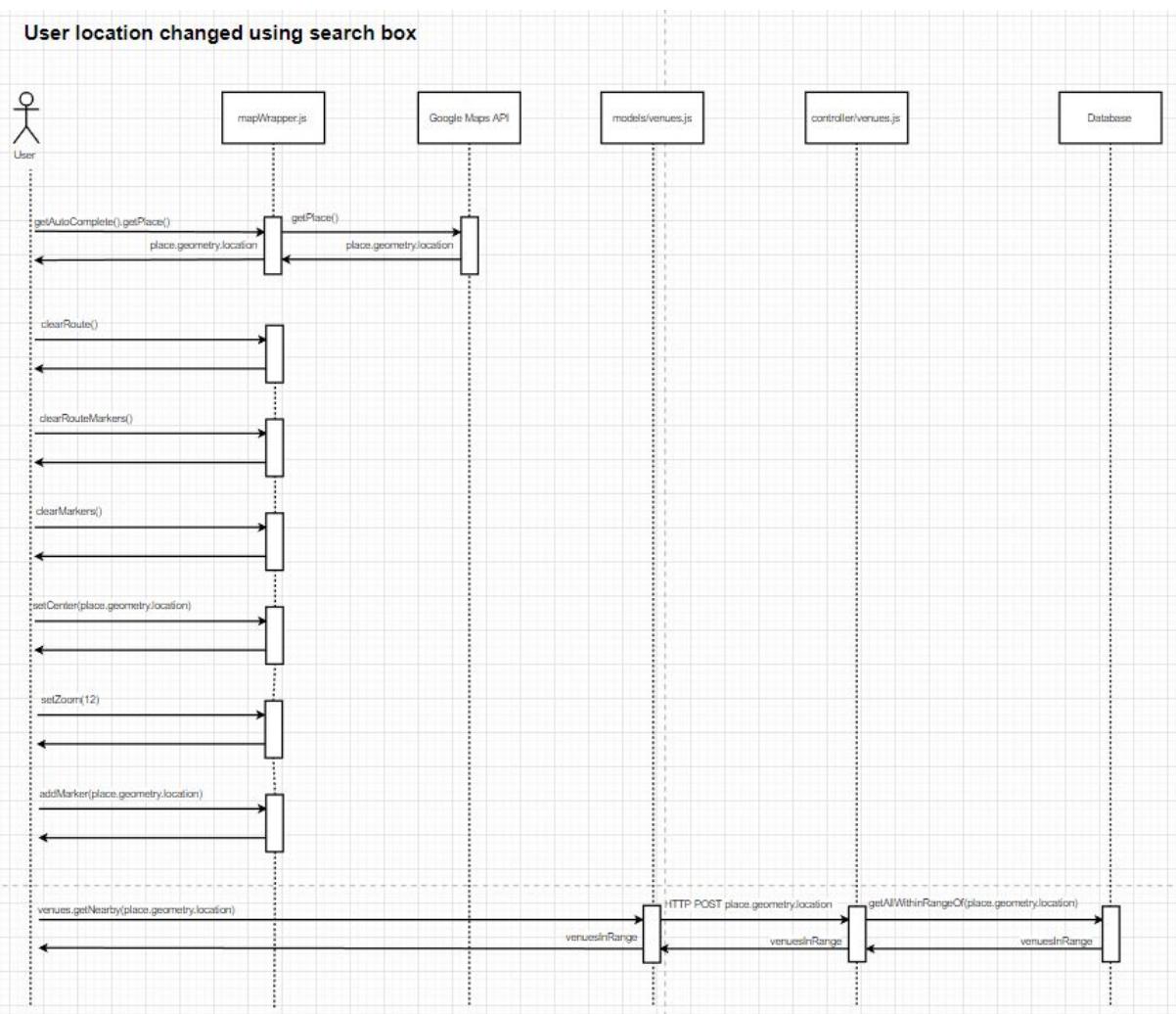
Week 13 / P 7 / System interaction diagrams

Sequence diagrams

Initial page load



User location changed using search box



Week 13 / P 8 / Object diagrams

Venue
<pre>_id = 507f1f77bcf86cd799439011 name = "The Counting House" addressLine1 = "2 St Vincent Place" addressLine2 = "" town = "Glasgow" region = "Lanarkshire" postCode = "G1 2DH" phone = "0141 225 0160" email = "" facilities = ["Children Welcome", "Room Hire", "WiFi", "Baby Changing", "Meeting Facilities", "Step Free Access"] openingTimes = { monday: ["07:00", "00:00"], tuesday: ["07:00", "00:00"], wednesday: ["07:00", "00:00"], thursday: ["07:00", "00:00"], friday: ["07:00", "00:00"], saturday: ["07:00", "00:00"], sunday: ["07:00", "00:00"]} coords = {long:-4.251891372088039, lat:55.86107943371641}</pre>

Comment
<pre>name: "Dale Johnston" email: "username@gmail.com" title: "Comment Title" comment: "This is a short comment."</pre>

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 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