STEPHEN HART – 14/12/18

Evidence Gathering Document for SQA Level 8 Professional Developer Award.

This document is designed for you to present your screenshots and diagrams relevant to the PDA and to also give a short description of what you are showing to clarify understanding for the assessor.

Each point that required details the Assessment Criteria (What you have to show) along with a brief description of the kind of things you should be showing.

Please fill in each point with screenshot or diagram and description of what you are showing.

Week 2

Unit	Ref	Evidence	
I&T	I.T.5	Demonstrate the use of an array in a program. Take screenshots of: *An array in a program *A function that uses the array *The result of the function running	
		Description:	

Paste Screenshot here

```
stops = [ "Croy", "Cumbernauld", "Falkirk High",
"Linlithgow", "Livingston", "Haymarket" ]
```

```ruby
def insert_stop
target=0
count=1
  for stop in stops
    if stop == "Falkirk High"
      target = count
    end
    count +=1
  end
stops.insert(target, "Polmont")
p stops
end
```

```
➜ Codeclan ruby test.rb
["Croy", "Cumbernauld", "Falkirk High", "Polmont", "Linlithgow", "Livingston", "Haymarket"]
```

Description here

"insert_stop" function inserts the stop "Polmont" at a point in the stops array after stop "Falkirk High"

Unit	Ref	Evidence	
I&T	I.T.6	Demonstrate the use of a hash in a program. Take screenshots of: *A hash in a program *A function that uses the hash *The result of the function running	
		Description:	

Paste Screenshot here

```ruby
@pet_shop = {
  admin: {
    total_cash: 1000,
    pets_sold: 0,
  },
  name: "Camelot of Pets"
},
```

```ruby
def total_cash(pet_shop)
  p @pet_shop[:admin][:total_cash]
end
```

```
[➜  Codeclan ruby test.rb
1000
```

Description here

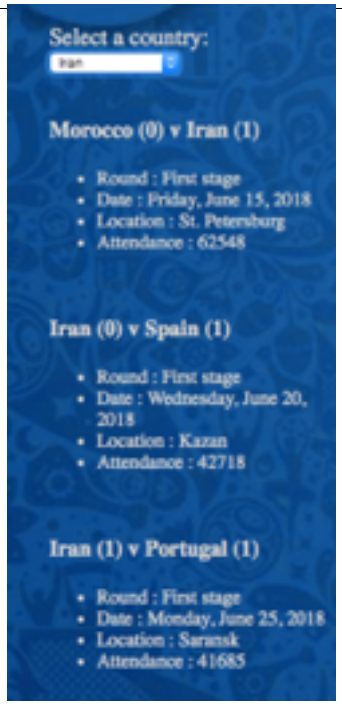
Function retrieves the value of ":total_cash" from the admin hash within the @pet_shop hash

Week 3

Unit	Ref	Evidence	
I&T	I.T.3	Demonstrate searching data in a program. Take screenshots of: *Function that searches data *The result of the function running	
		Description:	

Paste Screenshot here

```
// Filters all match data by team and calls team stats function
Matches.prototype.matchesByTeam = function (teamIndex) {
  const selectedTeam = this.home_team_country[teamIndex];
    // console.log(selectedTeam);
  const foundMatches = this.matches.filter((matches) => {
    return (matches.home_team_country === selectedTeam) ||
    (matches.away_team_country === selectedTeam);
  });
  this.teamStat(foundMatches, selectedTeam);
  return(foundMatches);
};
```



Description here

Function searches through an array of all matches at the recent football world cup then displays all matches (and various statistics) for the team selected by the user via a dropdown menu.

Unit	Ref	Evidence	
I&T	I.T.4	Demonstrate sorting data in a program. Take screenshots of: *Function that sorts data *The result of the function running	

		Description:

Paste Screenshot here

```
def self.current_month()
  sql = "SELECT * FROM transactions WHERE date_part('month',trans_date) =
      date_part('month',CURRENT_DATE) AND date_part('year',trans_date) =
      date_part('year',CURRENT_DATE) ORDER BY trans_date DESC"
  results = SqlRunner.run(sql)
  return results.map {|result|Transaction.new(result)}
end
```

All Transactions

Total spend (all months) : £2251

Date	Merchant	Category	Amount		
2018-10-04	Tesco	Shopping	£ 23.54		
2018-10-03	Majestic Wine	Wine	£ 40.99		
2018-10-03	Asda	Shopping	£ 198.37		
2018-10-03	The Black Bull	Going Out	£ 78.32		
2018-10-03	Newlands LTC	Sports	£ 60.52		
2018-10-03	Tesco	Petrol	£ 105.43		

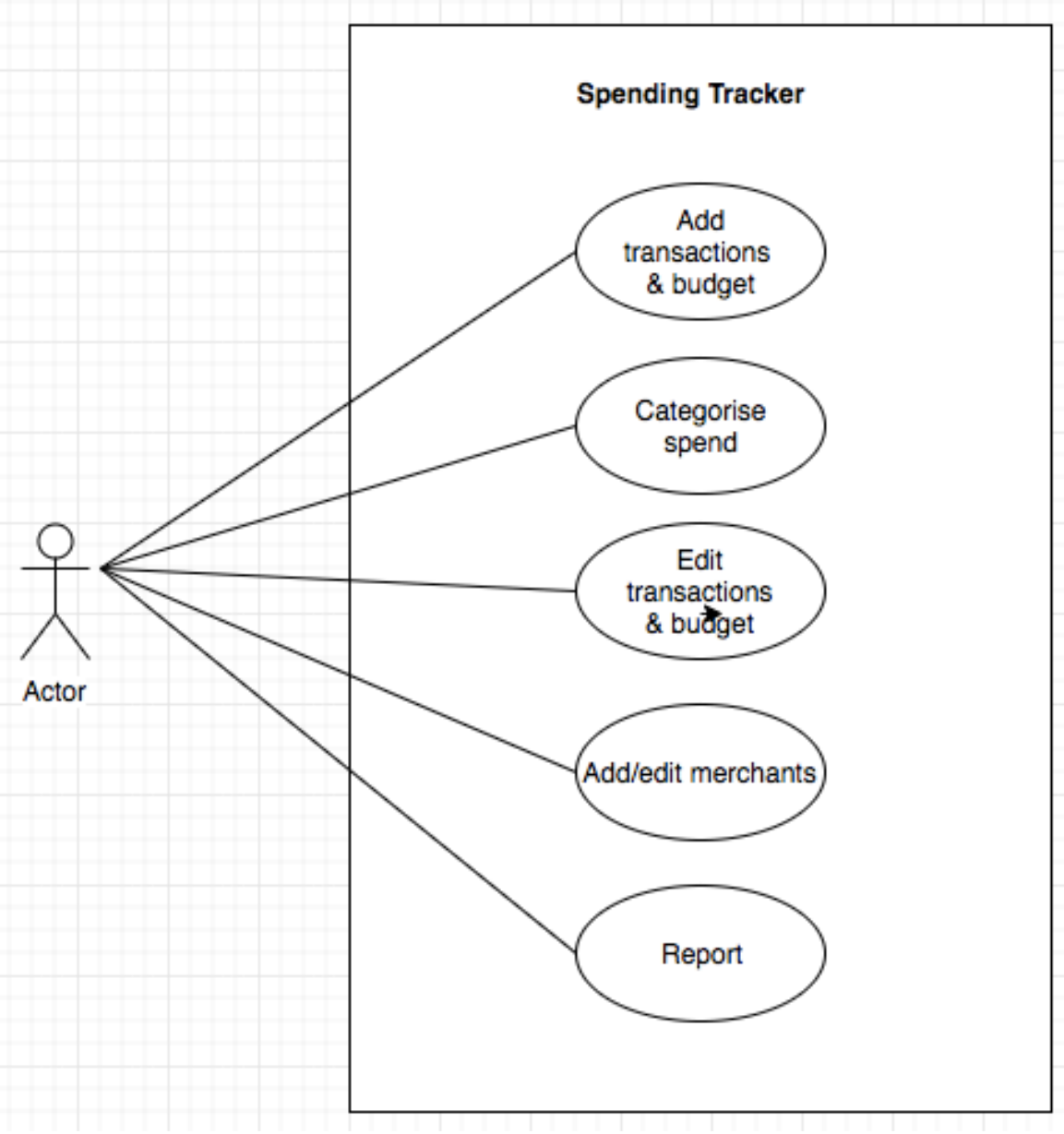
Description here

Function selects transactions where the transaction month equals the selected month then sorts the transactions in date ascending order before returning the transactions.

Week 5 and 6

Unit	Ref	Evidence	
A&D	A.D. 1	A Use Case Diagram	
		Description:	

Paste Screenshot here

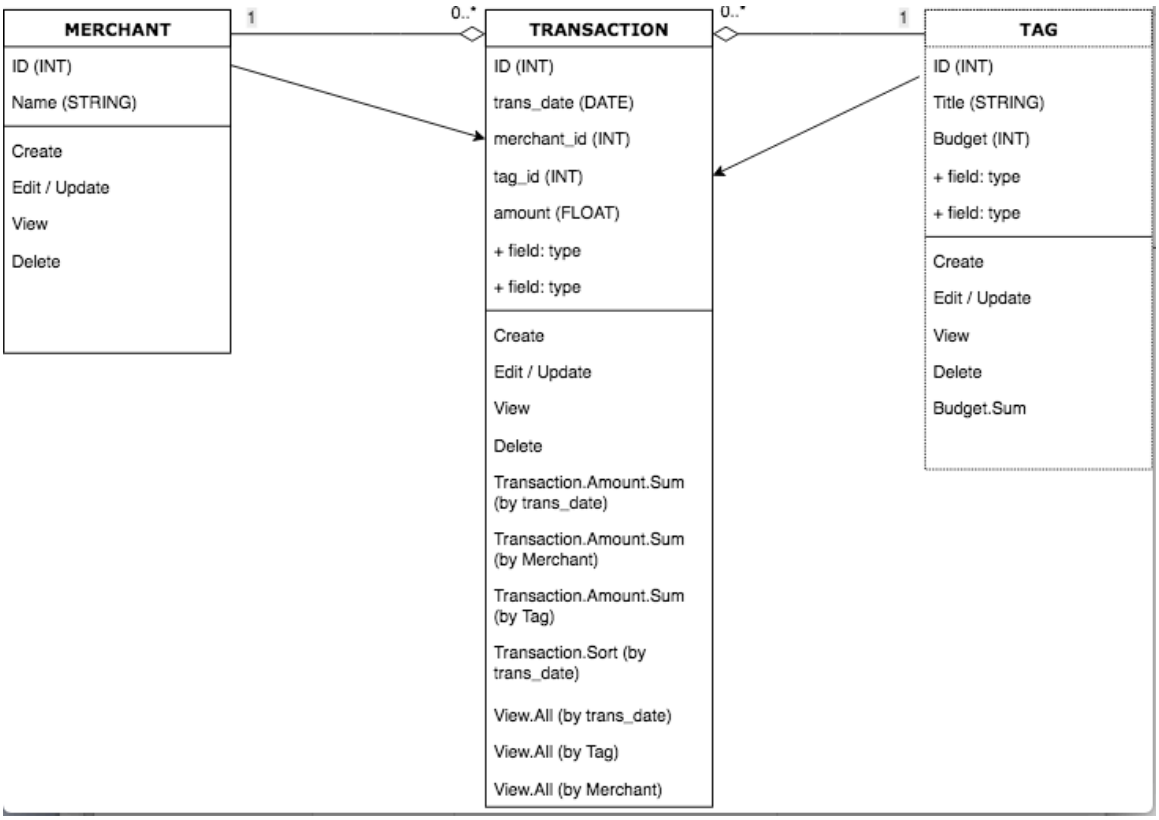


Description here

User Diagram shows a set of actions a user of my Week5 Spending Tracker can undertake.

Unit	Ref	Evidence	
A&D	A.D. 2	A Class Diagram	
		Description:	

Paste Screenshot her

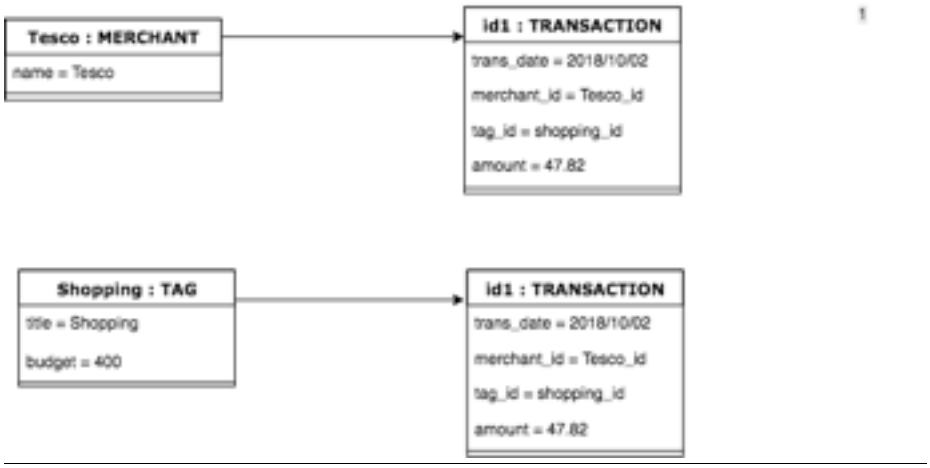


Description here

Class diagram of my Spending Tracker app showing the 3 classes – Merchant, Transaction and Tag (later changed name to budget in the app).

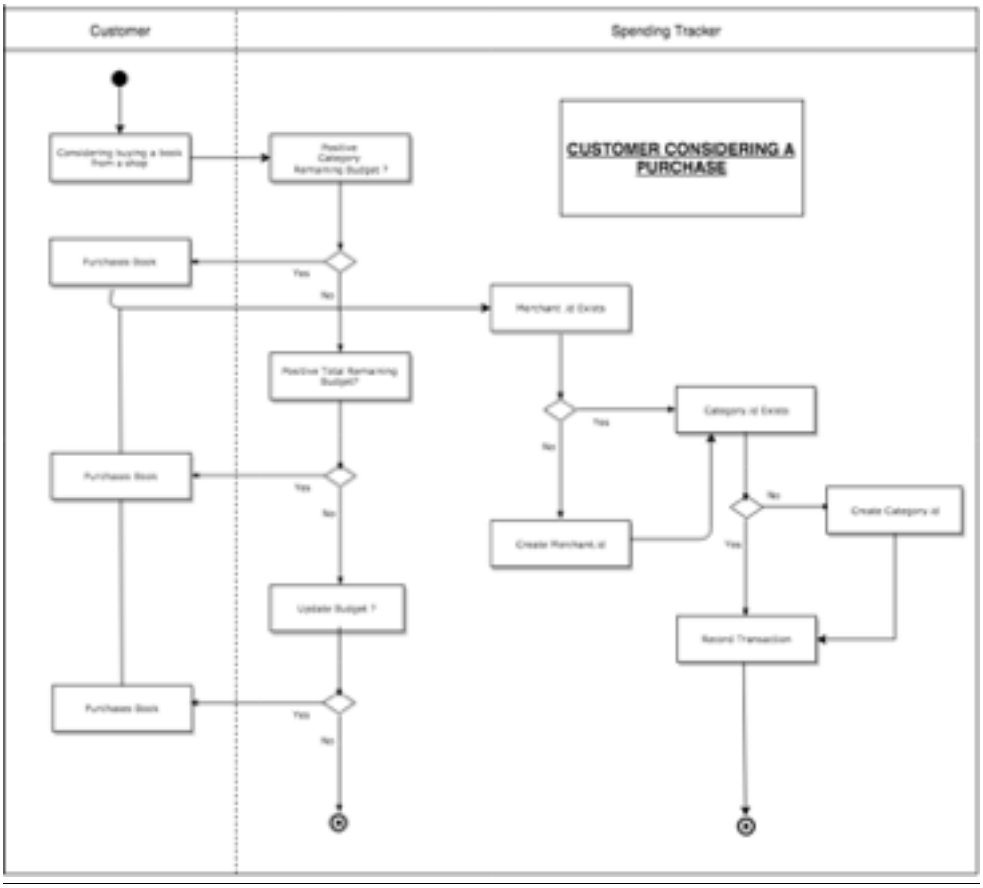
Unit	Ref	Evidence	
A&D	A.D. 3	An Object Diagram	
		Description:	

Paste Screenshot here



Unit	Ref	Evidence	
A&D	A.D. 4	An Activity Diagram	
		Description:	

Paste Screenshot here



Description here

Diagram show user interacting with the spending tracker app when contemplating and then executing a transaction.

Unit	Ref	Evidence	
A&D	A.D. 6	Produce an Implementations Constraints plan detailing the following factors: *Hardware and software platforms *Performance requirements *Persistent storage and transactions *Usability *Budgets *Time	
		Description:	

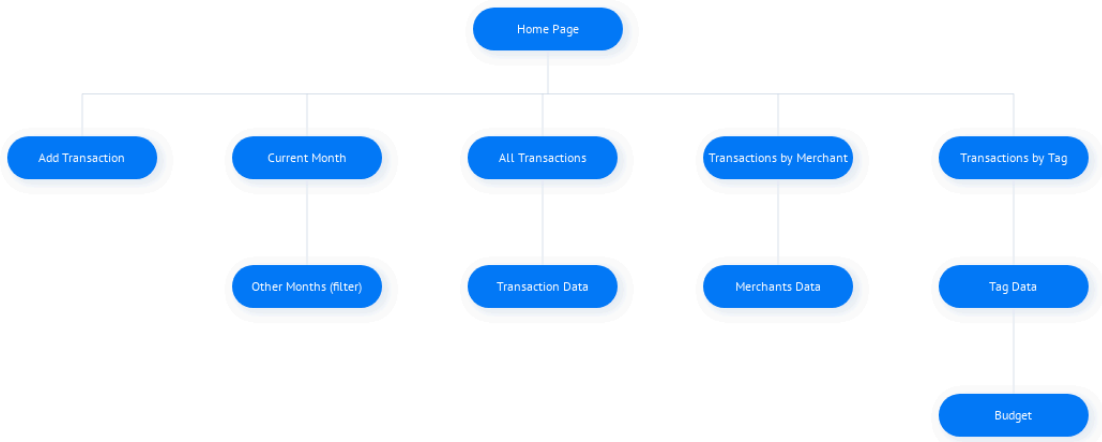
Paste Screenshot here

IMPLEMENTATIONS CONSTRAINTS PLAN

CONSTRAINT CATEGORY	IMPLEMENTATION CONSTRAINTS	SOLUTION
Hardware and Software Platforms	Continual access to internet required to access external API	Store temporary local copy of API on periodic basis as backup data if internet access not available
Performance Requirements	Sub 3 second load time	Ensure hosting company SLA meets requirement
Persistent Storage & Transactions	Relationship database required for data storage	Use MySQL
Usability	Require multi-lingual versions for users in Germany and France	Trial import of Google Translate
Budgets	Limited fixed budget (£5,000)	Agree long term fixed/capped cost maintenance/hosting contracts
Time Limitations	14 day development time limit	Limit features in initial release to "must haves"

Unit	Ref	Evidence	
P	P.5	User Site Map	
		Description:	

Paste Screenshot here

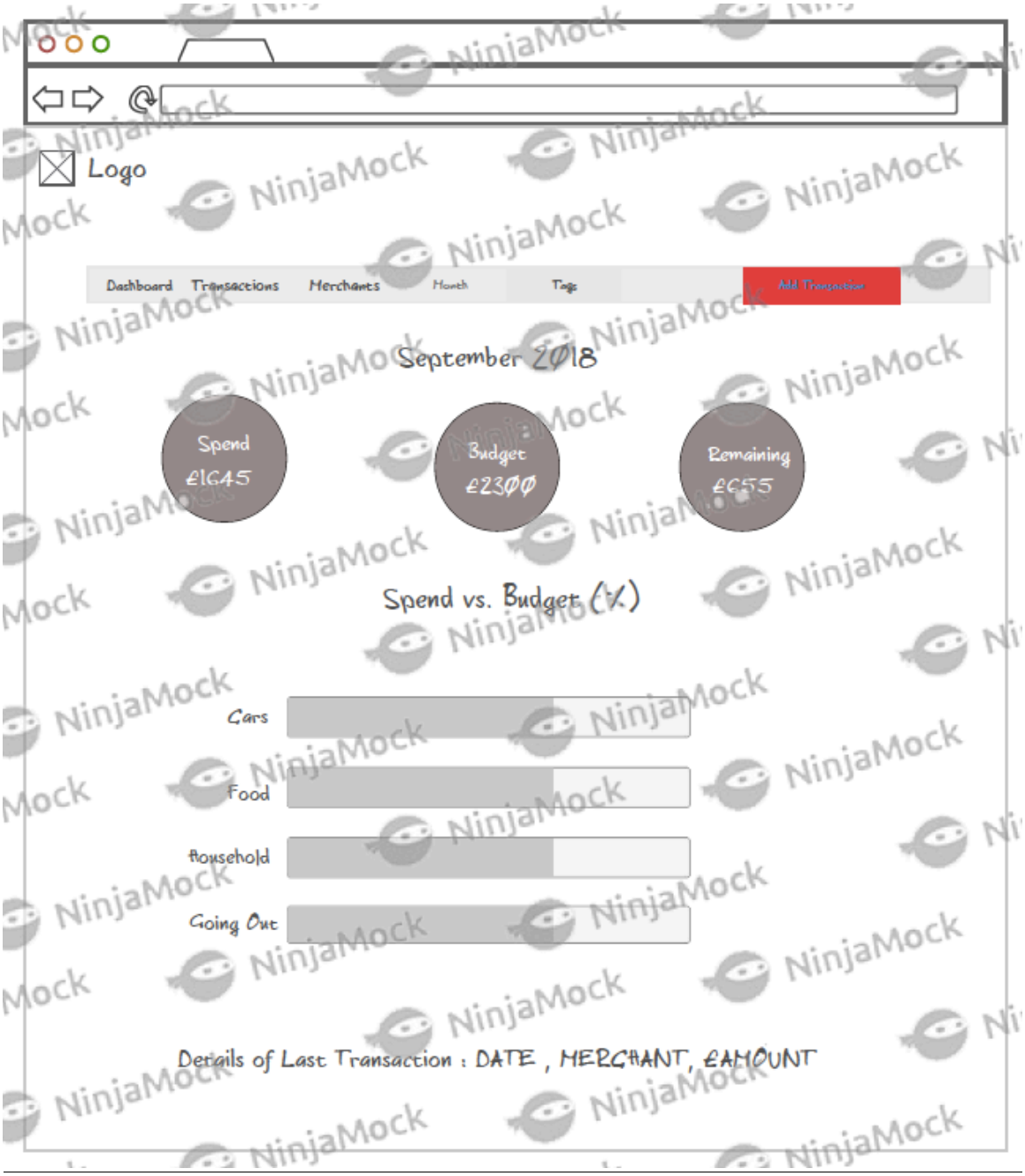


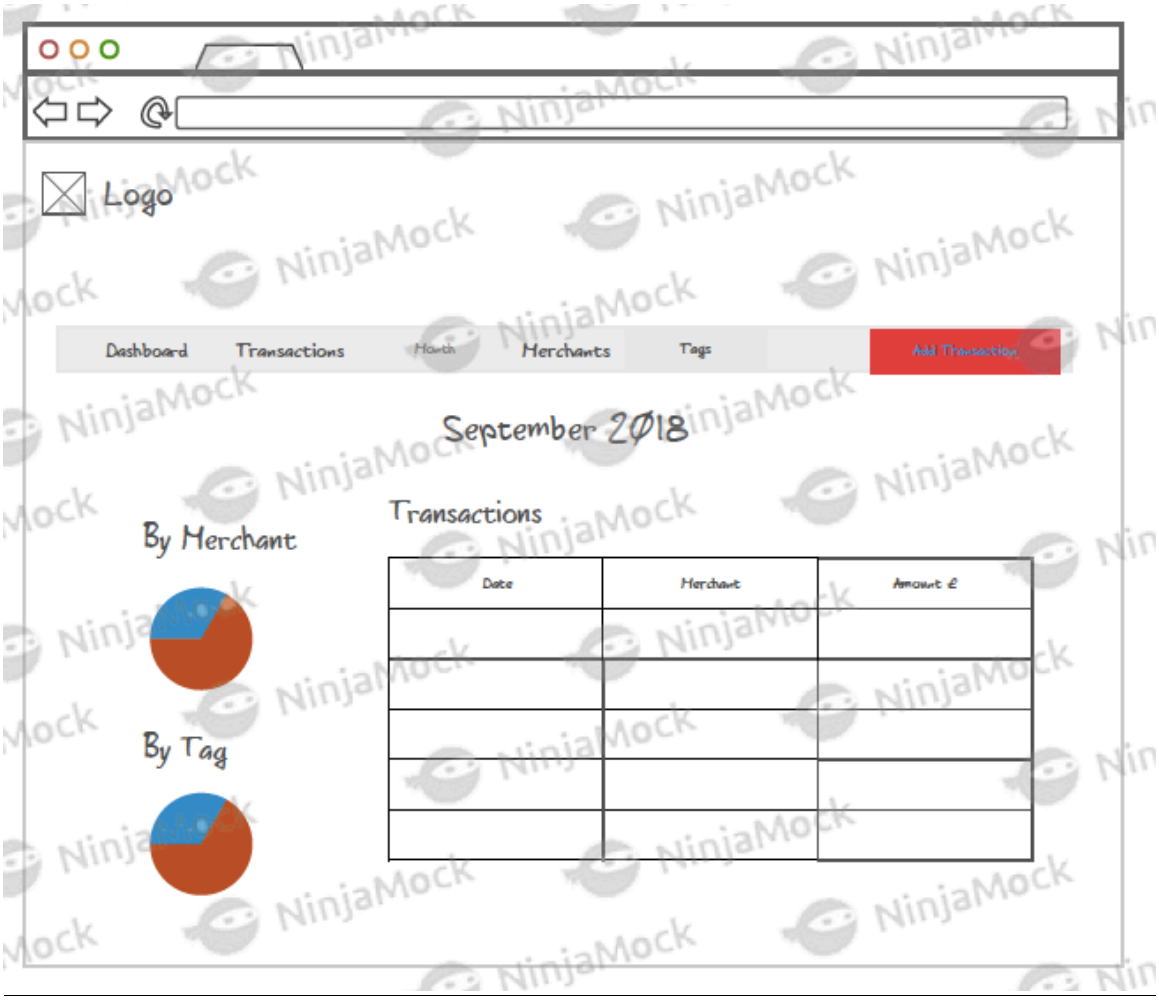
Description here

Site map of my spending tracker app showing the various view/pages.

Unit	Ref	Evidence	
P	P.6	2 Wireframe Diagrams	
		Description:	

Paste Screenshot here


Logo
Dashboard
Transactions
Merchants
Month
Tags
Add Transaction
September 2018
Spend
£1645
Budget
£2300
Remaining
£655
Spend vs. Budget (%)
Cars
Food
Household
Going Out
Details of Last Transaction : DATE , MERCHANT, £AMOUNT



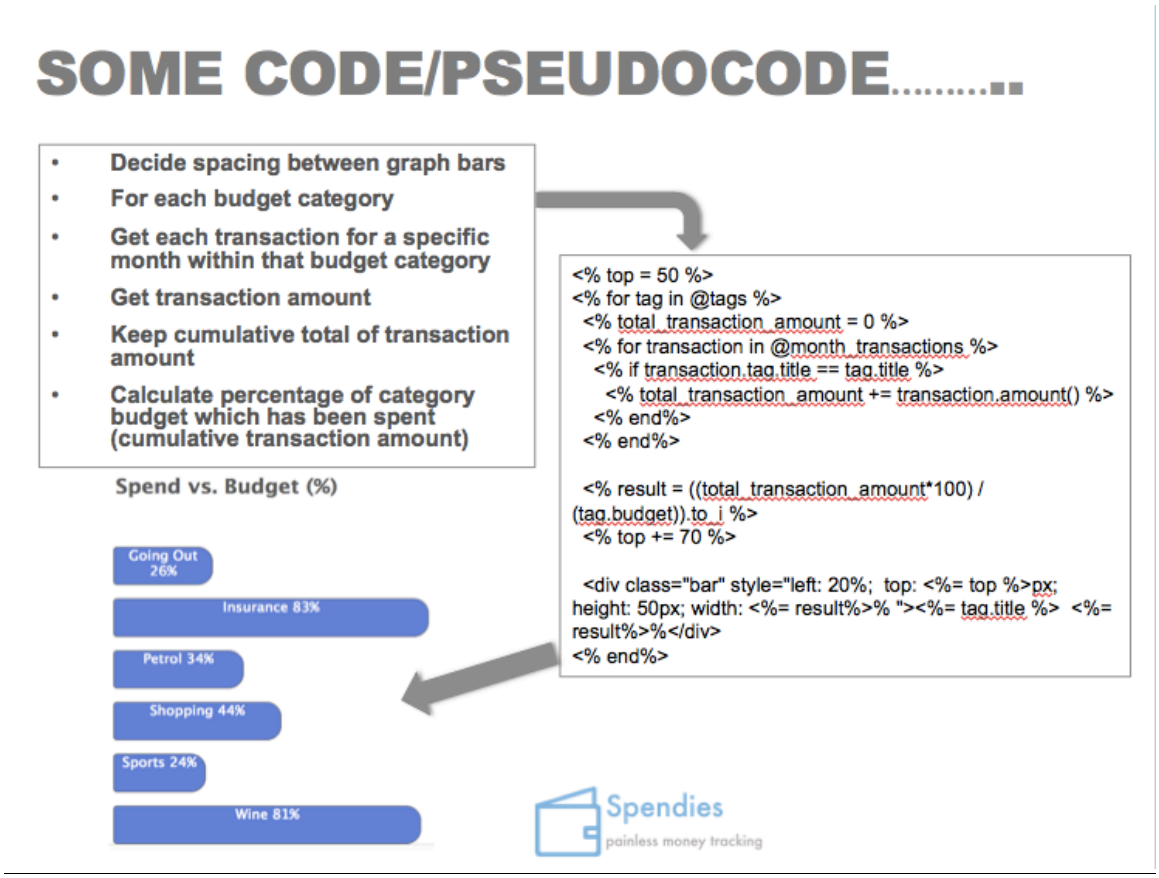
Description here

Wireframes of the spending tracker app showing 1) Home Page and 2) Page showing the spend for a given month

Unit	Ref	Evidence	
P	P.10	Example of Pseudocode used for a method	
		Description:	

Paste Screenshot here

```
PSEUDOCODE FOR BAR GRAPH CREATION (MONTH ACTUAL SPEND / MONTH BUDGET SPEND - ALL
CATEGORIES)

Decide spacing between graph bars

For every budget category

  Get every transaction for a specific month within each bugdet category

    Get transaction amount

Keep cumulative total of transaction amount

Calculate percentage of category budget which has been spent (cumulative transaction
amount)
```



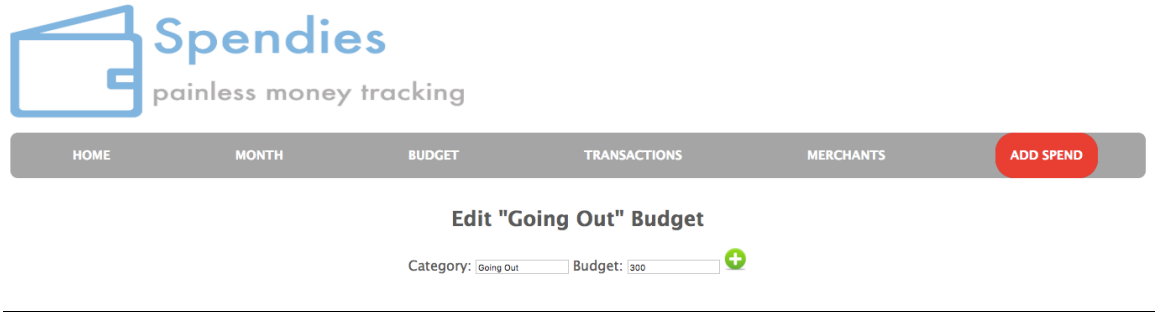
Description here

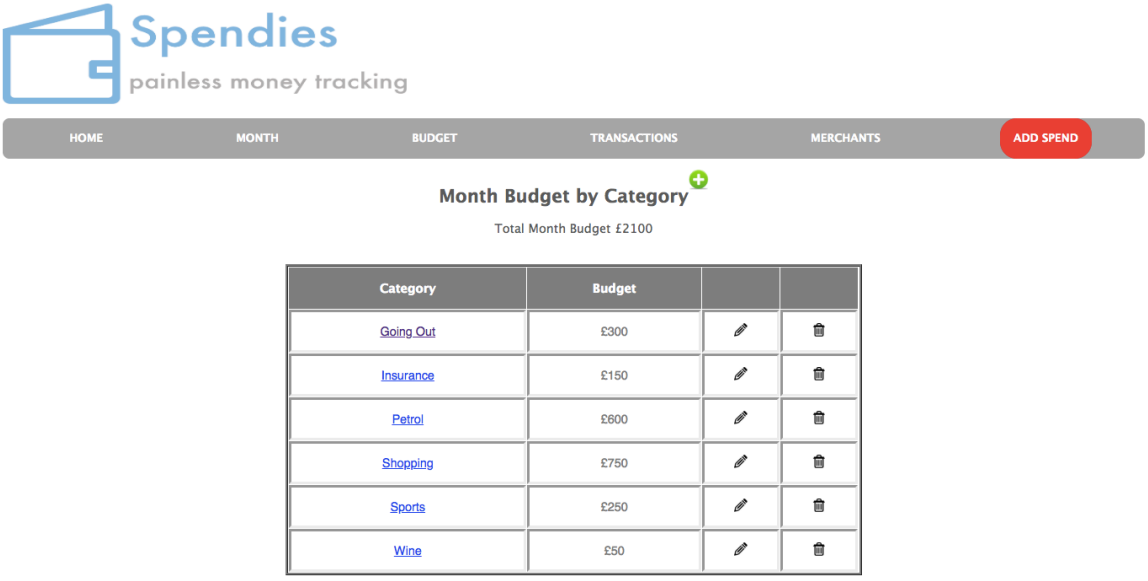
Pseudocode to create a bar graph in ruby which would then show the percentage of the month's budget for a given category which has actually been spent in the month to date.

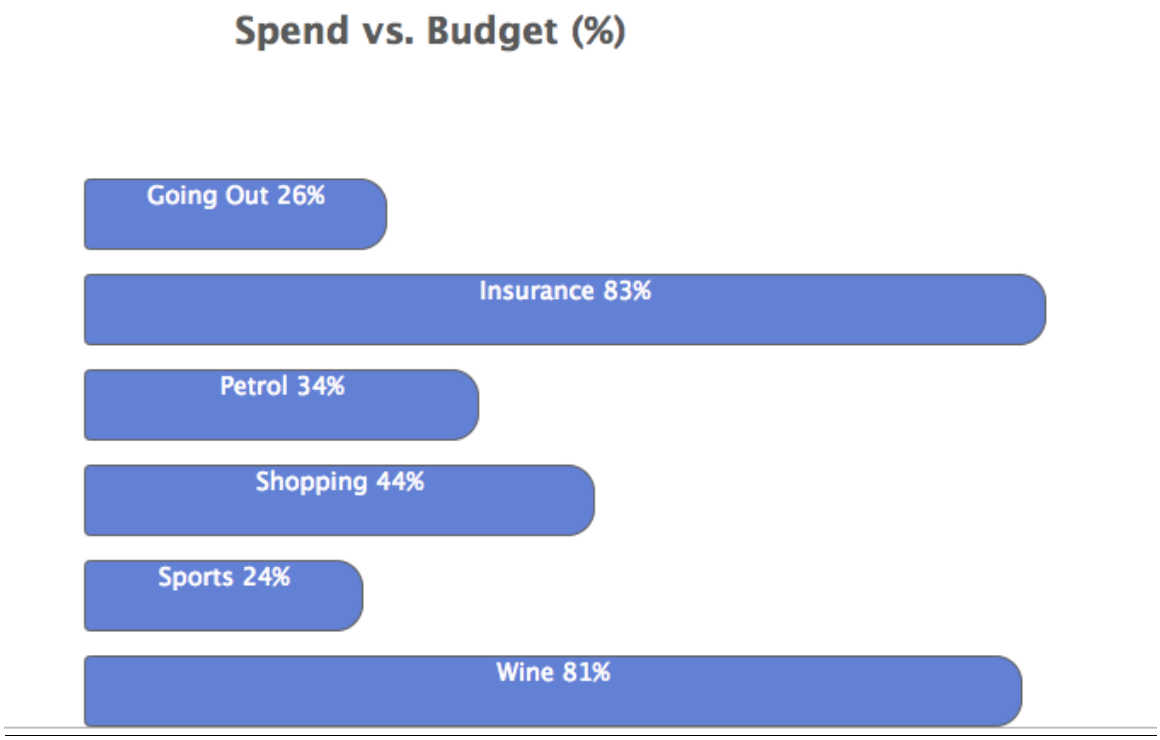
Unit	Ref	Evidence	
P	P.13	Show user input being processed according to design requirements. Take a screenshot of:	

		* The user inputting something into your program * The user input being saved or used in some way
		Description:

Paste Screenshot here







October 2018

Month Spend £845 Month Budget £2100 Remaining £1255

Choose Month : --------- ----

Date	Merchant	Category	Amount		
2018-10-04	Tesco	Shopping	£23.54		
2018-10-03	Tesco	Petrol	£105.43		
2018-10-03	Newlands LTC	Sports	£60.52		
2018-10-03	The Black Bull	Going Out	£78.32		
2018-10-03	Majestic Wine	Wine	£40.99		
2018-10-03	Asda	Shopping	£198.37		
2018-10-02	Asda	Shopping	£108.67		
2018-10-02	Hiscox	Insurance	£125.49		
2018-10-01	Tesco	Petrol	£103.78		

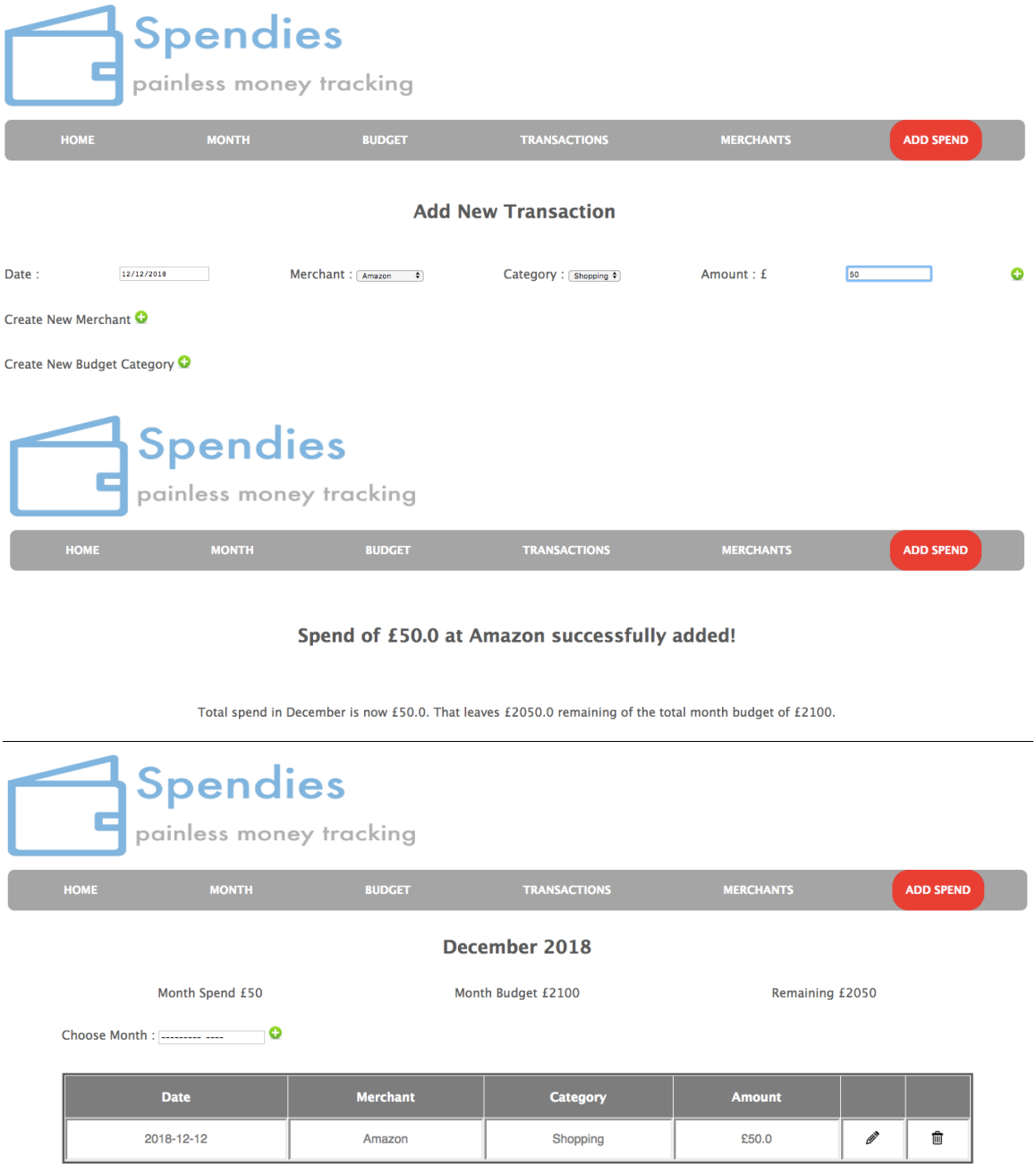
Description here

User inputs £300 as the budget limit for the category “Going out” (Screenshot 1). This is then saved and displayed in the total monthly budget table (screenshot 2) and is also used in the spend as a percentage of budget graph in Screenshot 3 (26% calculated as 78.32 / 300 where £78.32 is the total monthly spend for “going out” as shown in

Screenshot4).).

Unit	Ref	Evidence	
P	P.14	Show an interaction with data persistence. Take a screenshot of: * Data being inputted into your program * Confirmation of the data being saved	
		Description:	

Paste Screenshot here

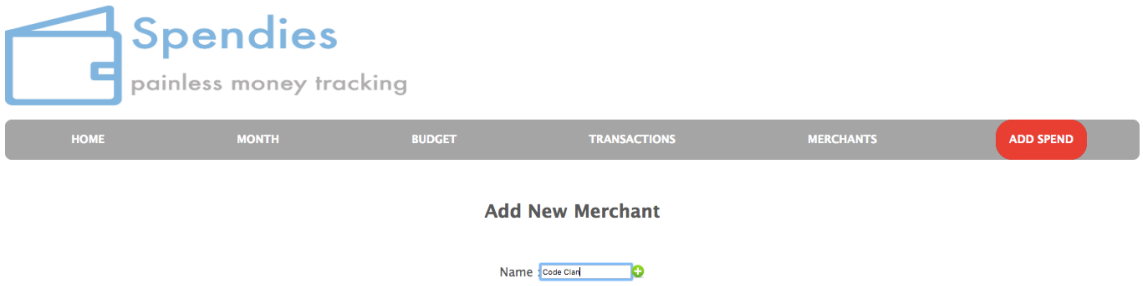


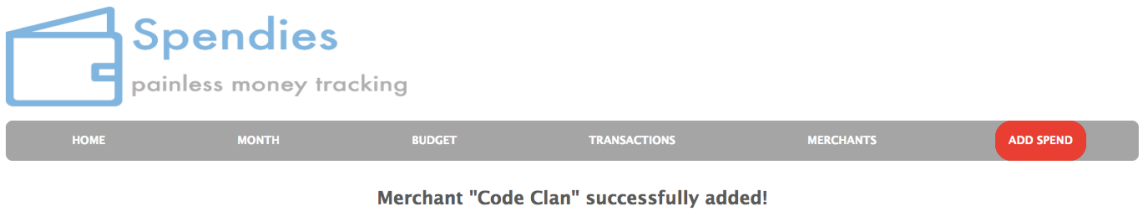
Description here

User enters a £50 transaction in screenshot 1. Screenshot 2 advises user that the transaction was successfully added. Screenshot 3 shows the list of the month's transactions and the £50 transaction can be seen in the list (only transaction for month).

Unit	Ref	Evidence	
P	P.15	Show the correct output of results and feedback to user. Take a screenshot of: * The user requesting information or an action to be performed * The user request being processed correctly and demonstrated in the program	
		Description:	

Paste Screenshot here





Merchants

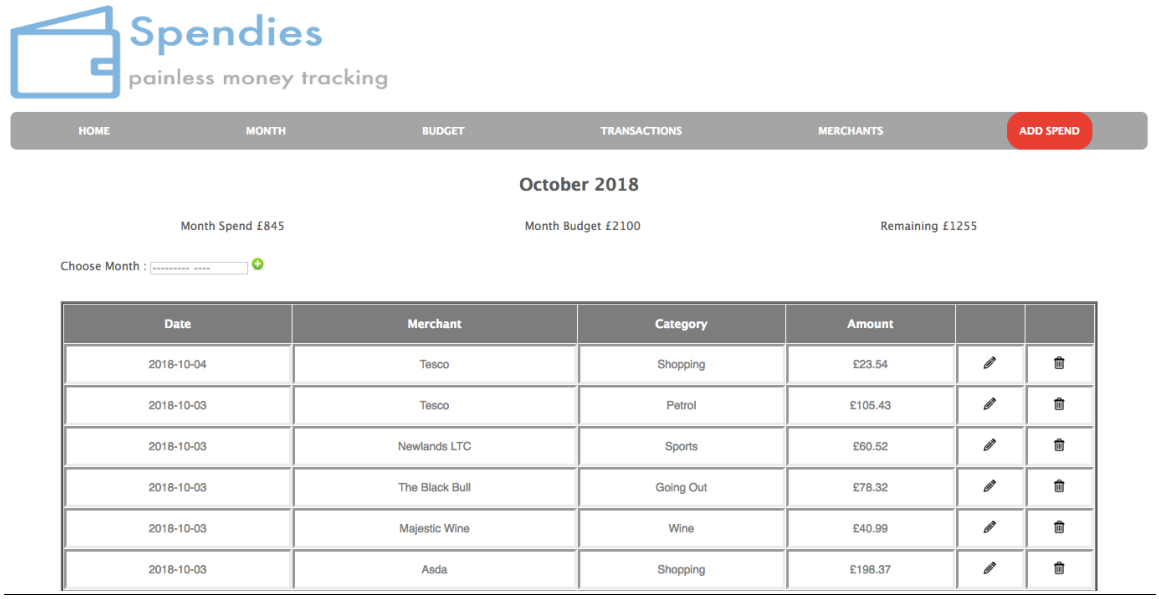
Name		
Amazon		
Asda		
Boots		
Code Clan		
Hiscox		
Majestic Wine		
Newlands LTC		
Tesco		
The Black Bull		

Description here

User requests that the merchant “Code Clan” is added to the merchant list (Screenshot1). This is confirmed in Screenshot2 and Screenshot3 shows the list of all merchants which now includes “Code Clan”.

Unit	Ref	Evidence	
P	P.11	Take a screenshot of one of your projects where you have worked alone and attach the Github link.	
		Description:	

Paste Screenshot here



Description here

Spending Tracker project undertaken in week4 at the end of the Ruby module.
https://github.com/stephenh71/Week4_Project_v2

Unit	Ref	Evidence	
P	P.12	Take screenshots or photos of your planning and the different stages of development to show changes.	
		Description:	

Paste Screenshot here

Commits on Oct 3, 2018

Final Submission stephenh71 committed on 3 Oct	8c3bda8

Commits on Oct 2, 2018

Various formatting changes stephenh71 committed on 2 Oct	a52db1d
Date filter added stephenh71 committed on 2 Oct	a3252a3
CSS almost complete stephenh71 committed on 2 Oct	de6e1c0

Commits on Oct 1, 2018

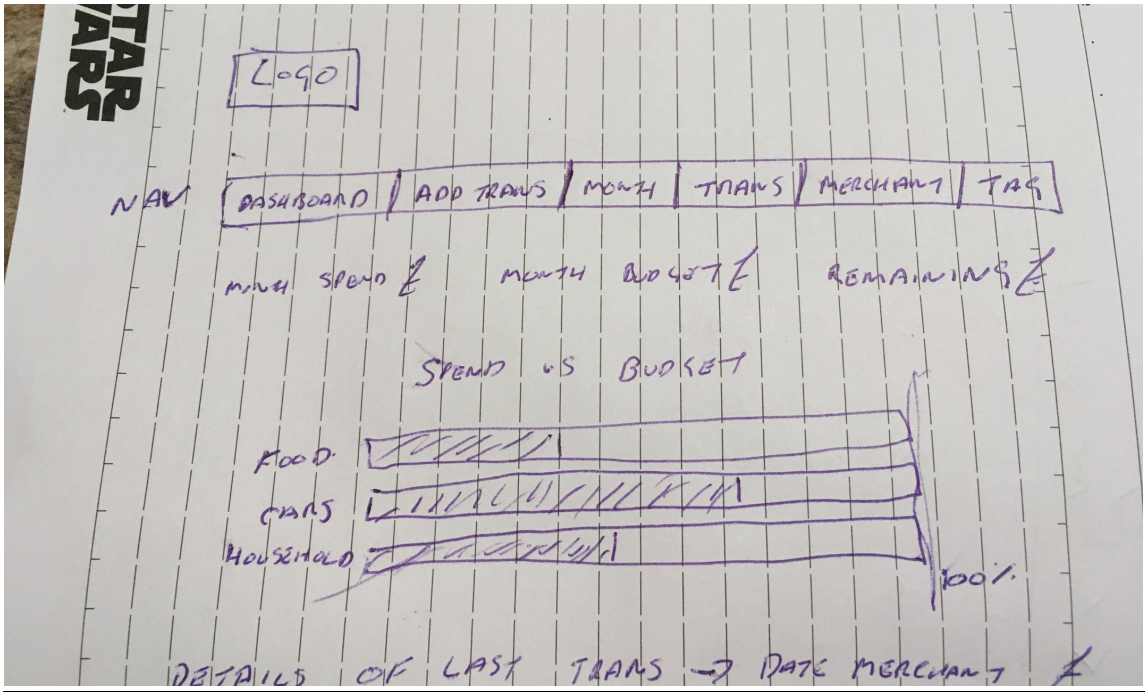
Dashboard Actual vs Budget added stephenh71 committed on 1 Oct	b70467b
Tag and Merchant view added stephenh71 committed on 1 Oct	9e19476

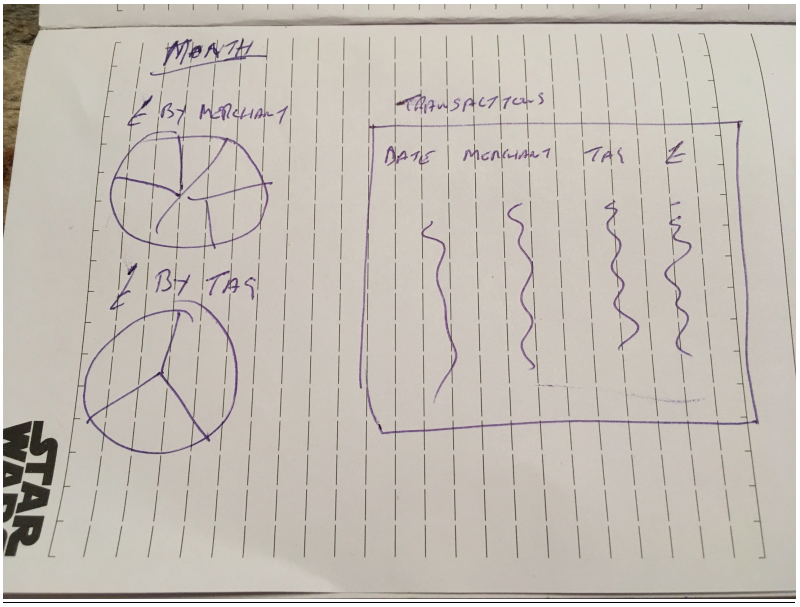
Commits on Sep 30, 2018

Selected Month created - not working stephenh71 committed on 30 Sep	083cfa6
Month view added stephenh71 committed on 30 Sep	c89b97d
Sunday start point, month filter still WIP stephenh71 committed on 30 Sep	77b9f0c
Budget total added and minor formatting stephenh71 committed on 30 Sep	a819a34

Commits on Sep 29, 2018

Display order and Transaction total function added stephenh71 committed on 29 Sep	448230b
MVP tidied stephenh71 committed on 29 Sep	126c891
MVP stephenh71 committed on 29 Sep	933174c
Transactions and Merchants MVP working stephenh71 committed on 29 Sep	8763d49


LOGO
NAV
DASHBOARD
ADD TRANS
MONTH
TRANS
MERCHANT
TAG
MONTH SPEND £
MONTH BUDGET £
REMAINING £
SPEND VS BUDGET
FOOD
CARS
HOUSEHOLD
100%
DETAILS OF LAST TRANS → DATE MERCHANT £


MONTH
£ BY MERCHANT
£ BY TAG
TRANSACTIONS
DATE
MERCHANT
TAG
£

Views :

1. Layout - buttons :
    - Dashboard RETURN

    - Transaction CREATE

2. Home page/Dashboard
    - Buttons :
        - Tags - VIEW/EDIT
        - Merchants - VIEW/EDIT (current_month_amount.sum)
        - Transaction - VIEW/EDIT ORDER BY date - VIEW BY month

      View :
        - Total Actual Spend current_month
        - Total Budget Spend current_month
        - Pie chart - Actual Spend current_month BY category

3. Tag - title, current_month_amount.sum, budget_amount.sum - EDIT button, month filter

4. Merchant - name, current_month_amount.sum - EDIT button, month filter

5. Transaction - date, merchant.name, category.name, amount SORTED BY date - EDIT button, month filter, tag filter, merchant filter

---

Data Structure :

Class Merchant

- Id INT
- Name STRING

Methods:
CREATE
EDIT / UPDATE
DELETE
VIEW
merchant.transaction.amount.sum FOR date(month)

Class Tag

- Id INT
- Title STRING
- Budget INT

Methods:
CREATE

EDIT / UPDATE
DELETE
VIEW
tag.transaction.amount.sum FOR date(month)
tag.transaction.budget.sum FOR date(month)

Class Transaction

- Id INT
- trans_date DATE
- merchant_id REFERENCES merchant.id
- tag_id REFERENCES tag.id
- amount INT (FLOAT?)

Methods:
CREATE
EDIT / UPDATE
DELETE
VIEW

Description here

Screenshot 1 -Github commit history showing stages of development
Screenshots 2&3 – Early stage rough wireframe diagrams
Screenshot 4 – initial planning on views/site map
Screenshot 5 – early class diagram

Later stage planning docs already appended previously above.

Week 7

Unit	Ref	Evidence	
P	P.16	Show an API being used within your program. Take a screenshot of: * The code that uses or implements the API * The API being used by the program whilst running	
		Description:	

Paste Screenshot here

```
Matches.prototype.getData = function () {
  const request = new Request("https://worldcup.sfg.io/matches");
  request.get((data) => {
    this.matches = data
    PubSub.publish('Matches:all-matches-ready', data)
    // console.log(data);

    PubSub.subscribe('SelectView:change', (event) => {
      const teamIndex = event.detail;
      // console.log(teamIndex);
      this.publishMatchesByTeam(teamIndex);
    });
    this.publishTeams();
    this.totalStat();
  });
};
```

2018 World Cup Results

Select a country

- Argentina
- Australia
- Belgium
- Brazil
- Colombia
- Costa Rica
- Croatia
- Denmark
- Egypt
- England
- France
- Germany
- Iceland
- Iran
- Japan
- Korea Republic
- Mexico
- Morocco
- Nigeria
- Panama
- Peru
- Poland
- Portugal
- Russia
- Saudi Arabia
- Senegal
- Serbia
- Spain
- Sweden
- Switzerland
- Tunisia
- Uruguay

2018 World Cup Results

Select a country: Germany

Germany (0) v Mexico (1)

- Round : First stage
- Date : Sunday, June 17, 2018
- Location : Moscow
- Attendance : 78011

Germany (2) v Sweden (1)

- Round : First stage
- Date : Saturday, June 23, 2018
- Location : Sochi
- Attendance : 44287

Korea Republic (2) v Germany (0)

- Round : First stage
- Date : Wednesday, June 27, 2018
- Location : Kazan
- Attendance : 41835

Germany reached the First stage

Stats - average per game :

Attempts on goal : 22.3

Attempts on target : 6.7

Goals : 0.7

Fouls : 9.7

Yellow cards : 0.7

Posession (%) : 67

Pass Accuracy (%) : 87

Description here

API requests details of all matches at the recent world cup from the FIFA website (Screenshot1). These are then used to populate a drop-down menu with the name of each participating team (Screenshot2) which when selected displays details of all their matches at the world cup and some statistics (Screenshot3).

Unit	Ref	Evidence	
P	P.18	Demonstrate testing in your program. Take screenshots of: * Example of test code * The test code failing to pass * Example of the test code once errors have been corrected * The test code passing	
		Description:	

Paste Screenshot here

```
beforeEach(function() {
  journey1 = new Journey('linlithgow', 'stirling', 'train', 30);
  journey2 = new Journey('paris', 'frankfurt', 'train', 400);
  journey3 = new Journey('sydney', 'new york', 'aeroplane', 10000);
  journey4 = new Journey('london', 'rome', 'car', 1200);
  journey5 = new Journey('lancaster', 'isle of man', 'ferry', 80);
  journeys = [journey1, journey2, journey3, journey4, journey5];
  traveller = new Traveller(journeys);
});

it('should have a collection of journeys', function() {
  const actual = traveller.journeys;
  assert.deepStrictEqual(actual, journeys);
});
```

```
it('should be able to get a unique list of the modes of transport', function() {
  const actual = [ 'train', 'aeroplane', 'car', 'ferry' ];
  assert.strictEqual(actual, traveller.getUniqueModesOfTransport());
});
```

```
> mocha specs


  Journey
    ✓ should have a start location
    ✓ should have a end location
    ✓ should have a mode of transport
    ✓ should have a distance

  Traveller
    ✓ should have a collection of journeys
    ✓ should be able to get the journeys start locations
    ✓ should be able to get the journeys end locations
    ✓ should be able to get a list of the modes of transport
    ✓ should be able to get journeys by transport
    ✓ should be able to get journeys over a certain distance
    ✓ should be able to calculate total distance travelled
    1) should be able to get a unique list of the modes of transport


  11 passing (21ms)
  1 failing

  1) Traveller
       should be able to get a unique list of the modes of transport:

      AssertionError [ERR_ASSERTION]: Input objects identical but not reference equal:

[
  'train',
  'aeroplane',
  'car',
  'ferry'
]

      + expected - actual


      at Context.<anonymous> (specs/traveller_spec.js:80:12)
```

```
  it('should be able to get a unique list of the modes of transport', function() {
    const actual = [ 'train', 'aeroplane', 'car', 'ferry' ];
    assert.deepStrictEqual(actual, traveller.getUniqueModesOfTransport());
  });
```

```
> mocha specs


  Journey
    ✓ should have a start location
    ✓ should have a end location
    ✓ should have a mode of transport
    ✓ should have a distance

  Traveller
    ✓ should have a collection of journeys
    ✓ should be able to get the journeys start locations
    ✓ should be able to get the journeys end locations
    ✓ should be able to get a list of the modes of transport
    ✓ should be able to get journeys by transport
    ✓ should be able to get journeys over a certain distance
    ✓ should be able to calculate total distance travelled
    ✓ should be able to get a unique list of the modes of transport


  12 passing (29ms)
```

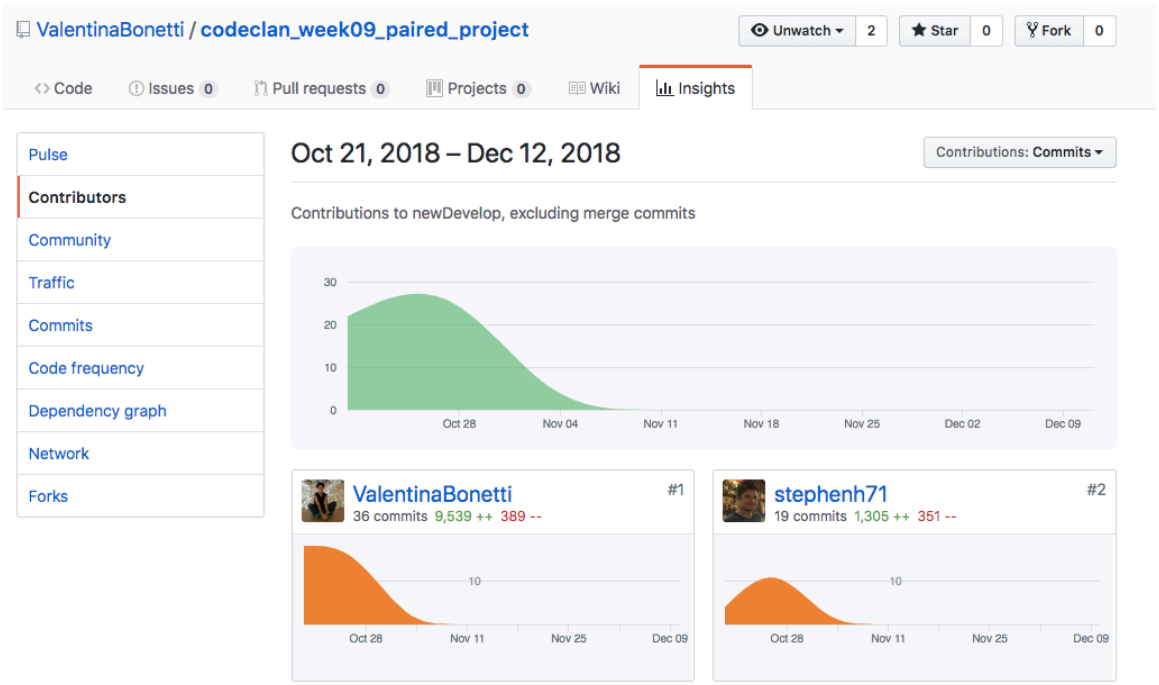
Description here

Screenshot 1 & 2 – test code example including test 12 which fails in Screenshot 3 because the test assertion was “strictEqual” whereas a “DeepStrictEqual” assertion was required. Screenshot 4 shows the test code after the correction and Screenshot 5 shows all tests passing after the correction.

Week 9

Unit	Ref	Evidence	
P	P.1	Take a screenshot of the contributor’s page on Github from your group project to show the team you worked with.	
		Description:	

Paste Screenshot here



Description here

Worked with Valentina Bonetti on a share price/portfolio tracker - https://github.com/ValentinaBonetti/codeclan_week09_paired_project

Unit	Ref	Evidence	
P	P.2	Take a screenshot of the project brief from your group project.	
		Description:	

Paste Screenshot here

Shares Portfolio Application

A local trader has come to you with a portfolio of shares. She wants to be able to analyse it more effectively. She has a small sample data set to give you and would like you to build a Minimum Viable Product that uses the data to display her portfolio so that she can make better decisions.

MVP

A user should be able to:

- view total current value.
- view individual and total performance trends.
- retrieve a list of share prices from an external API and allow the user to add shares to her portfolio.
- View a chart of the current values in her portfolio.

Example Extensions

- Speculation based on trends and further financial modelling using projections.

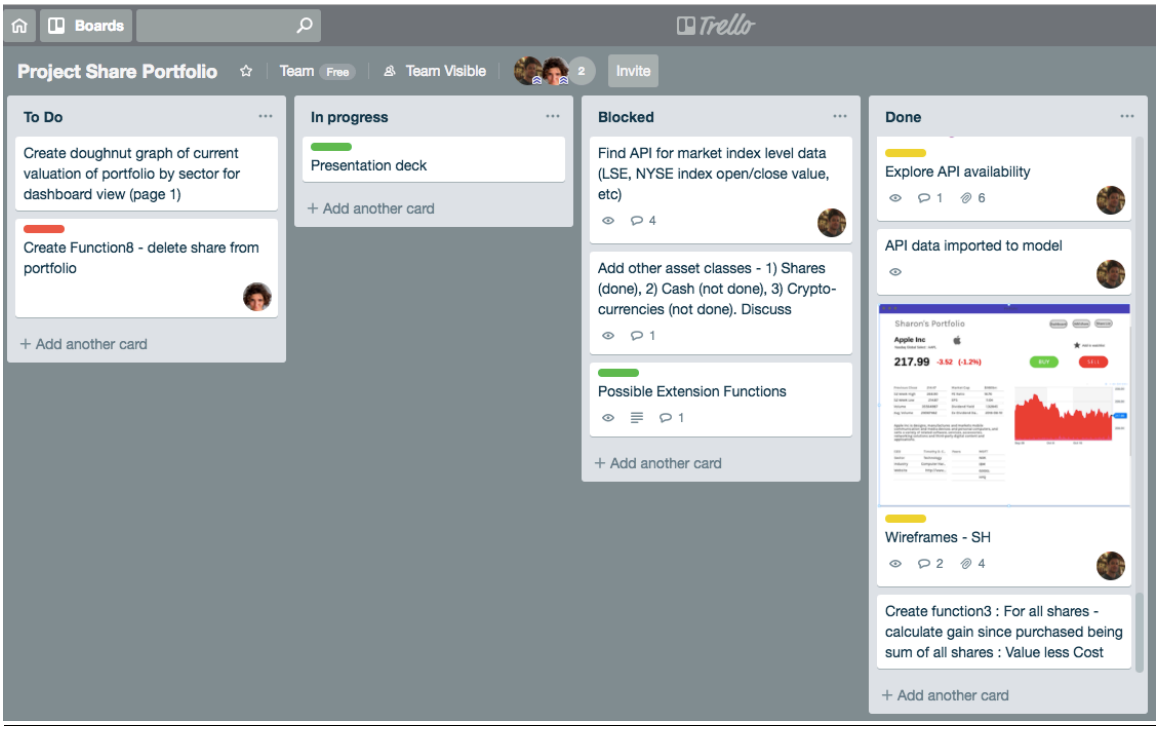
API, Libraries, Resources

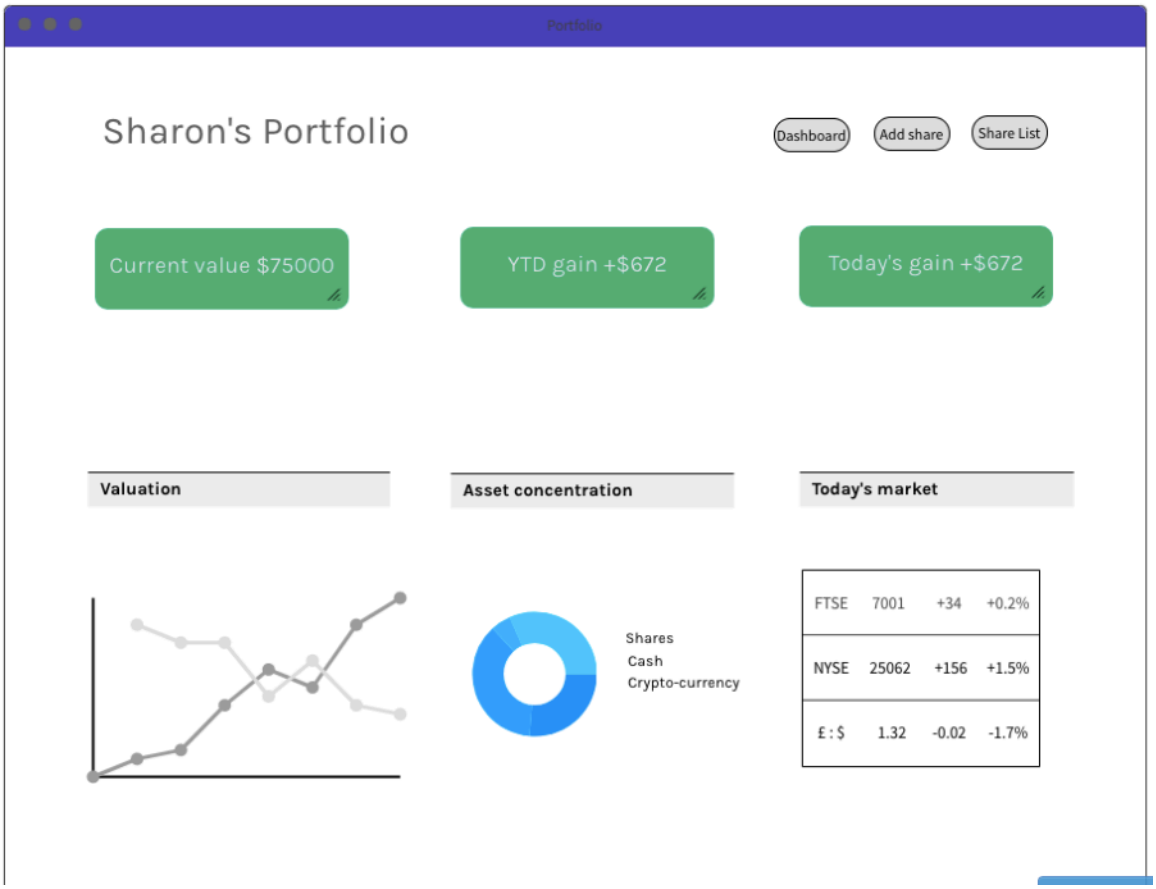
- https://www.alphavantage.co/ (Requires sign up)
- https://developers.google.com/chart/

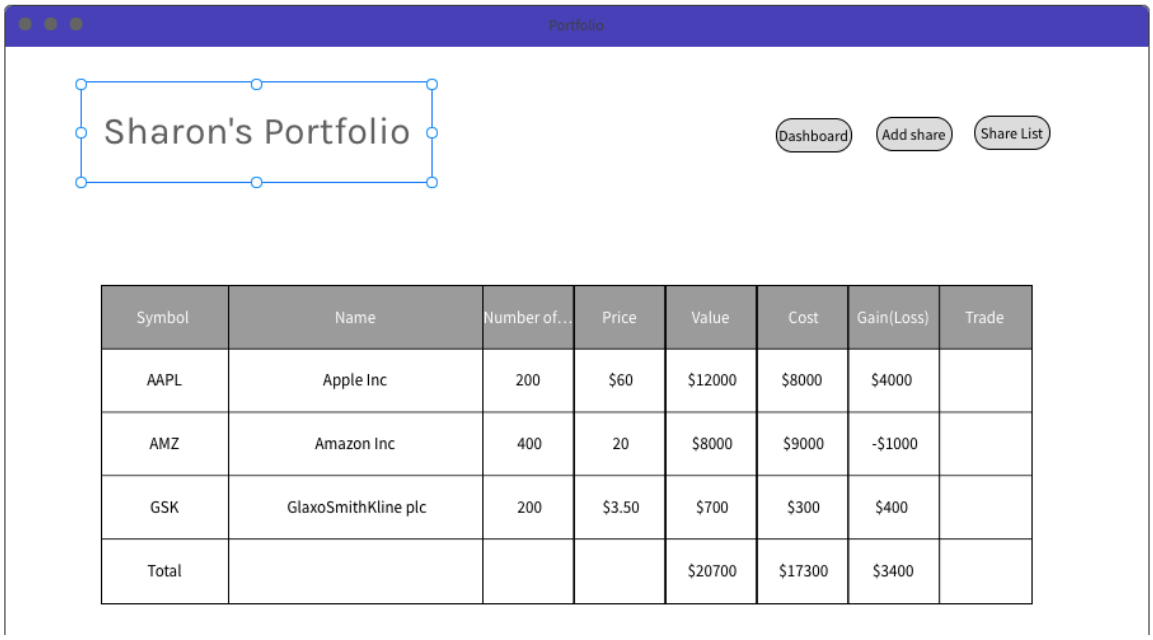
Description here

Unit	Ref	Evidence	
P	P.3	Provide a screenshot of the planning you completed during your group project, e.g. Trello MOSCOW board.	
		Description:	

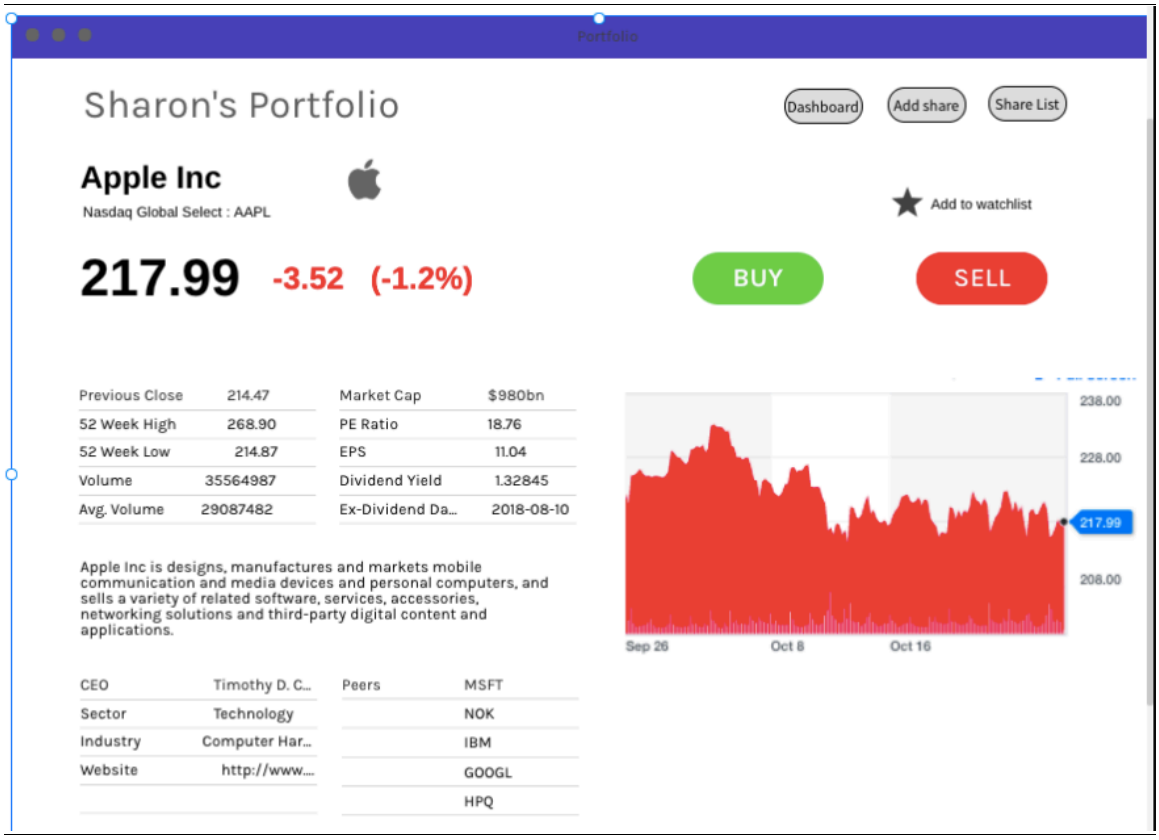
Paste Screenshot here

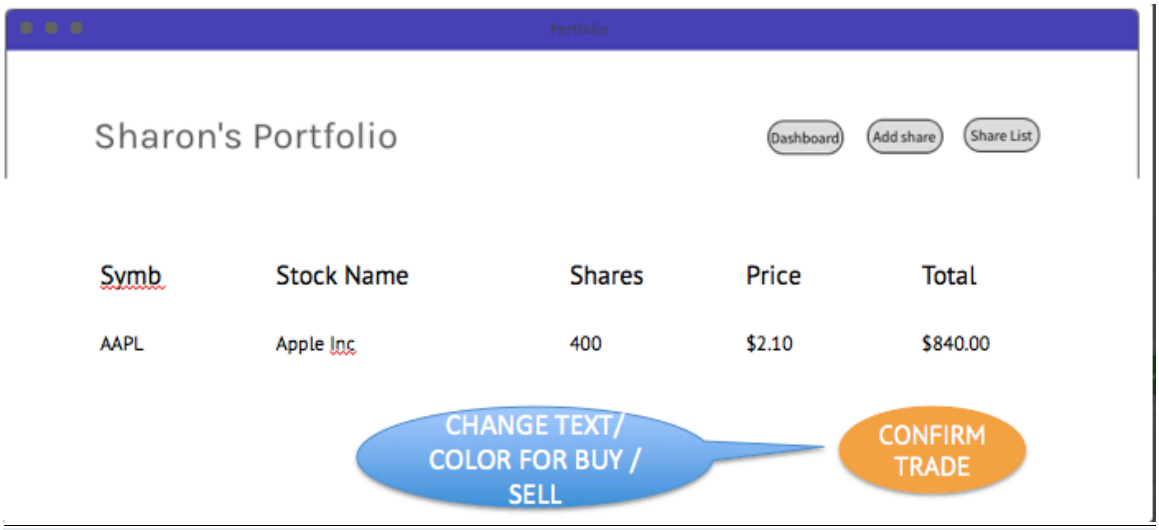

Boards
Trello
Project Share Portfolio
Team Free
Team Visible
2
Invite
To Do
Create doughnut graph of current valuation of portfolio by sector for dashboard view (page 1)
Create Function8 - delete share from portfolio
+ Add another card
In progress
Presentation deck
+ Add another card
Blocked
Find API for market index level data (LSE, NYSE index open/close value, etc)
Add other asset classes - 1) Shares (done), 2) Cash (not done), 3) Crypto-currencies (not done). Discuss
Possible Extension Functions
+ Add another card
Done
Explore API availability
API data imported to model
Wireframes - SH
Create function3 : For all shares - calculate gain since purchased being sum of all shares : Value less Cost
+ Add another card

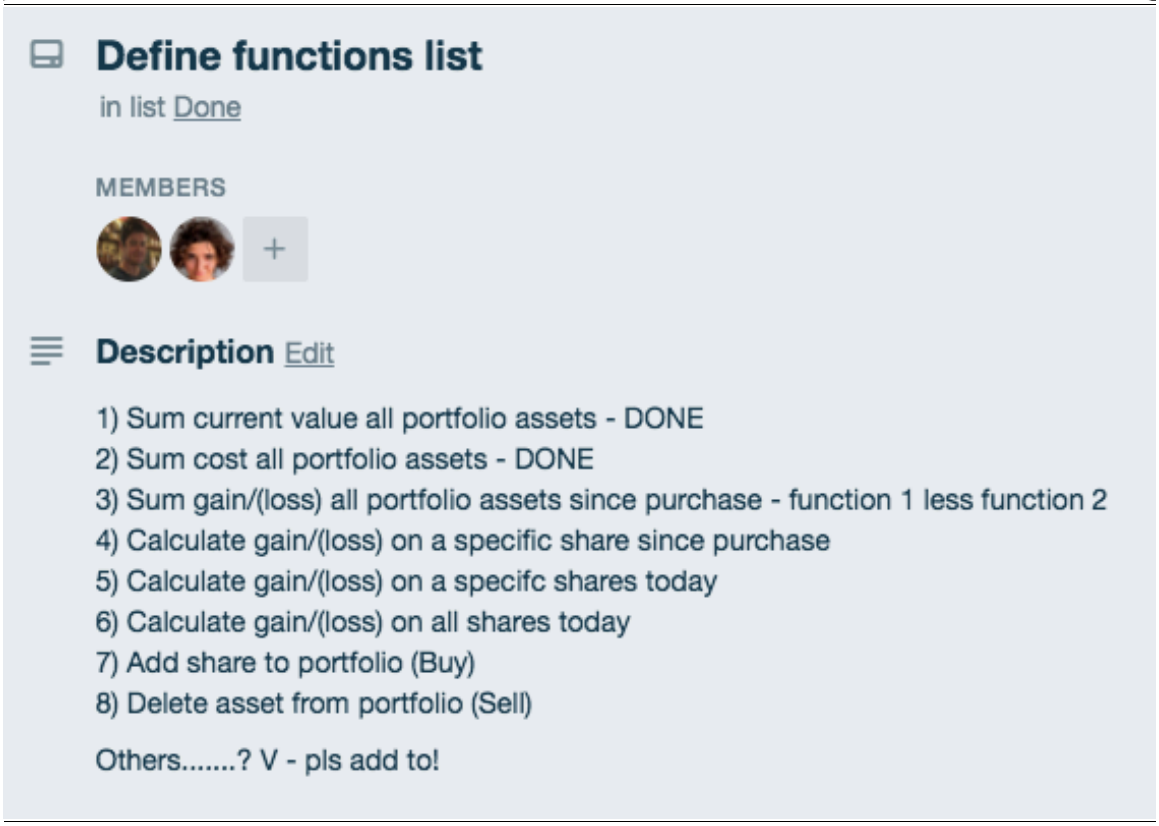

Portfolio
Sharon's Portfolio
Dashboard
Add share
Share List
Current value $75000
YTD gain +$672
Today's gain +$672
Valuation
Asset concentration
Today's market
Shares
Cash
Crypto-currency
FTSE 7001 +34 +0.2%
NYSE 25062 +156 +1.5%
£:$ 1.32 -0.02 -1.7%



Symbol	Name	Number of...	Price	Value	Cost	Gain(Loss)	Trade
AAPL	Apple Inc	200	$60	$12000	$8000	$4000	
AMZ	Amazon Inc	400	20	$8000	$9000	-$1000	
GSK	GlaxoSmithKline plc	200	$3.50	$700	$300	$400	
Total				$20700	$17300	$3400	







Description here

Screenshot1 – Trello Board
Screenshts2-5 – various wireframes
Screenshot 6 – functionality/features list

Unit	Ref	Evidence	

P	P.4	Write an acceptance criteria and test plan.

Paste Screenshot here

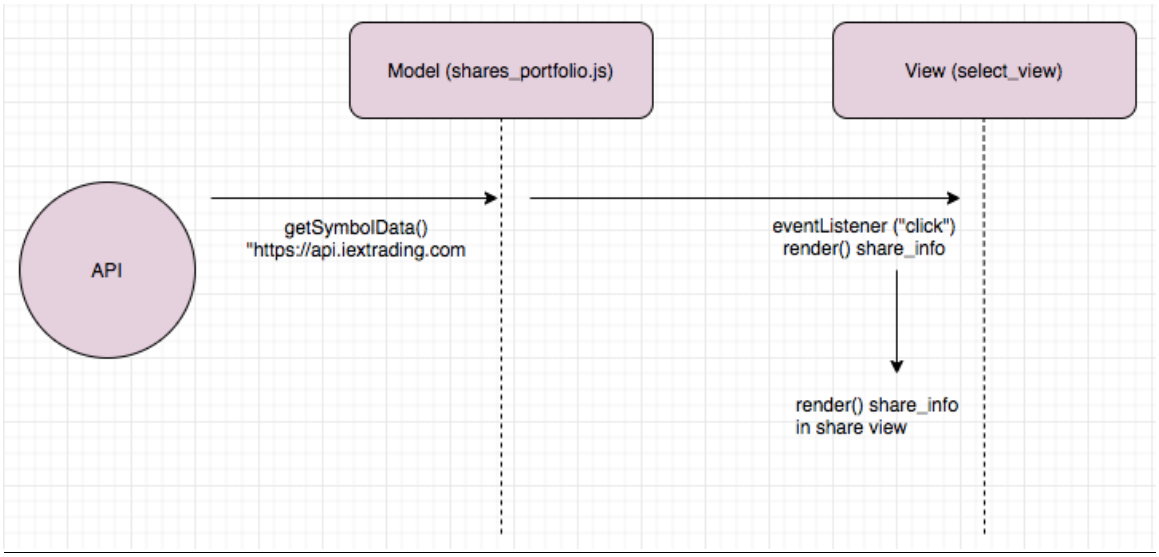
ACCEPTANCE CRITERIA (a user is able to……)	EXPECTED RESULT/OUTPUT	PASS/FAIL
Book a table for a specific date and time	Booking object created in database with specific customer name, table number and date/time	Pass/Fail
Update a booking	Booking object in database reflects amended attribute	Pass/Fail
Delete a booking	Booking object removed from database	Pass/Fail
View all bookings for a specific day	Booking List component accurately and completely renders all bookings for any specified day which has at least one booking	Pass/Fail
View a list of customers ordered by booking numbers (most bookings first)	Customer list is initially rendered chronologically by date of creation but can be sorted by descending visit frequency by clicking on the sort button	Pass/Fail
Only book available tables	Table select drop-down menu only includes available tables and dynamically changes as additional criteria (party size, date/time) are specified.	Pass/Fail
Add a customers receipt to their booking	Booking object created in database includes receipt total attribute.	Pass/Fail
Calculate how much a customer has spent over a specific time period	Single customer view includes spend total which equals the sum of all receipts appended to bookings over same period.	Pass/Fail
Award a discount to frequent customers	Customer record updated to reflect discount level (gold, silver, bronze).	Pass/Fail

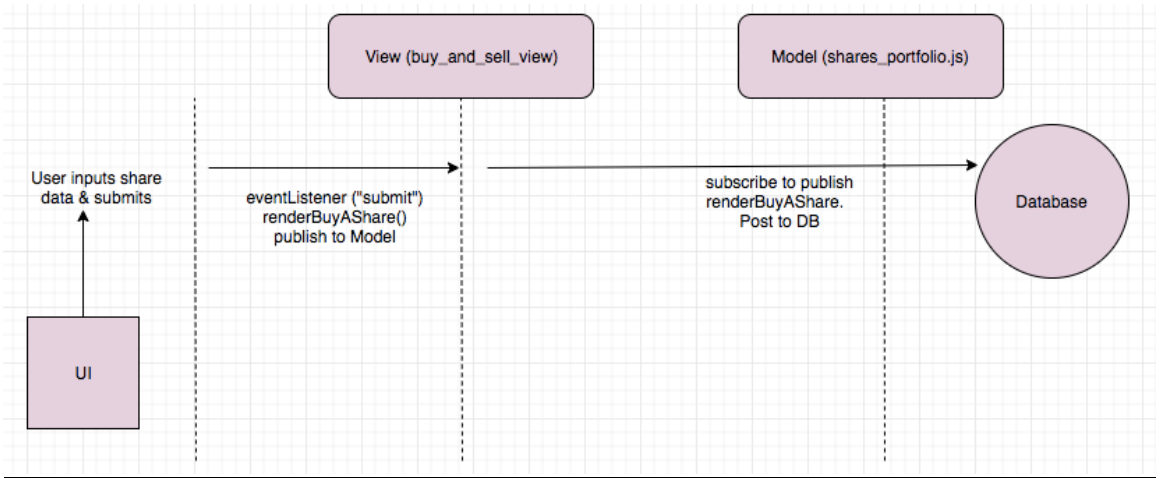
Description here

Acceptance criteria from group project no.2 (restaurant booking system).

Unit	Ref	Evidence	
P	P.7	Produce two system interaction diagrams (sequence and/or collaboration diagrams).	
		Description:	

Paste Screenshot here





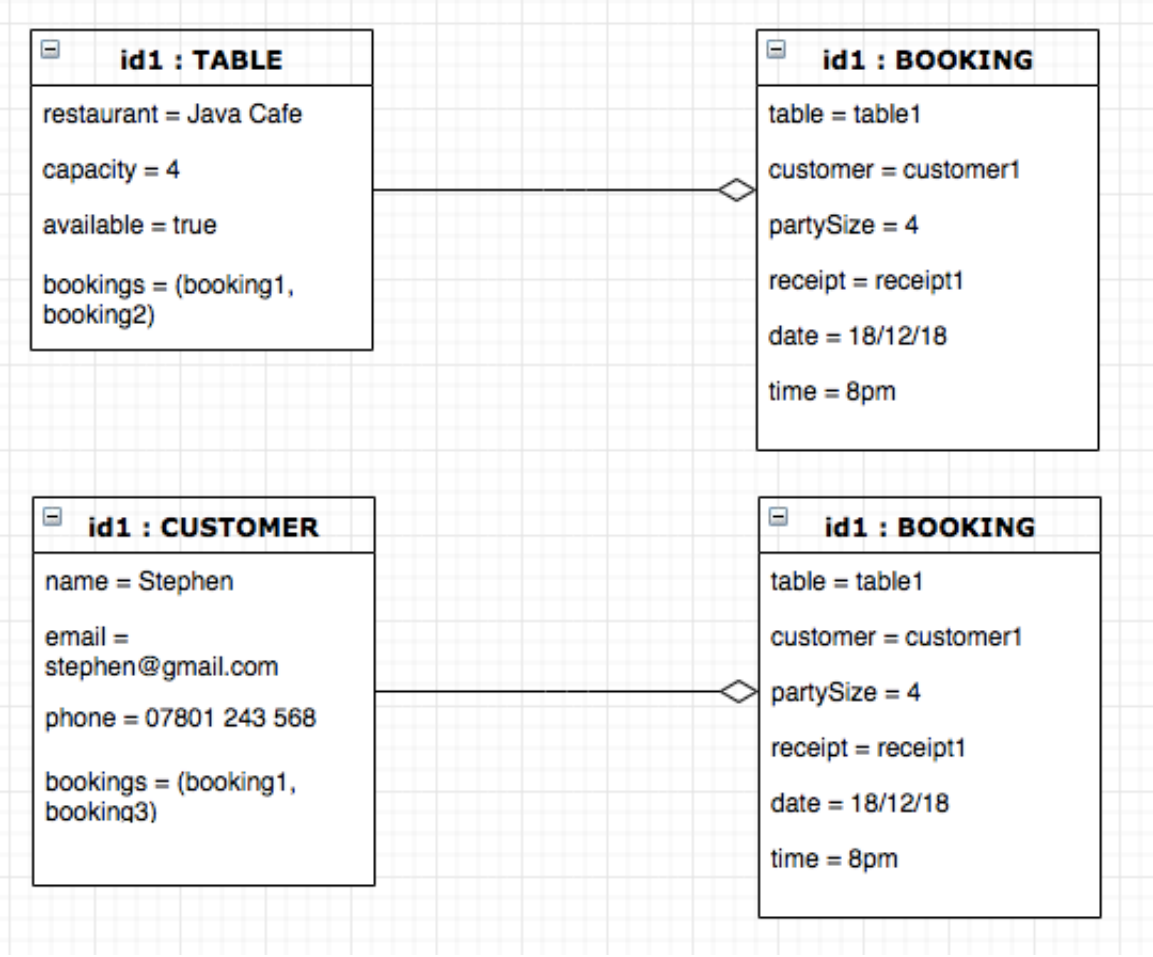
Description here

Screenshot 1 – model gets data from API, user selects which share item to render

Screenshot 2 – user completes “buy shares” form which is published tomodel and share item is added to database.

Unit	Ref	Evidence	
P	P.8	Produce two object diagrams.	
		Description:	

Paste Screenshot here



Description here

From restaurant booking diagram – Diagram 1 : Table & Booking, Diagram 2 : Customer & Booking

Unit	Ref	Evidence	
P	P.17	Produce a bug tracking report	
		Description:	

Paste Screenshot here

BUG LIST

DATE	BUG/ERROR	SOLUTION
7/12/18	Null Object being passed after database fetch request	Use Promise syntax to ensure fetch is complete before next function called.
8/12/18	Table colour doesn't change from green (available) to red (unavailable) when booked.	Changed approach. Dropdown table list now filtered to exclude booked tables thereby restricting customer choice only to available tables.
9/12/18	Edit customer details doesn't return to correct view after submission	Incorrect path used - corrected

---

Description here

Week 12

Unit	Ref	Evidence	
I&T	I.T.7	The use of Polymorphism in a program and what it is doing.	
		Description:	

Paste Screenshot here

```
public class Shop {

    private ArrayList<ISell> stock;

    public Shop() {
        this.stock = new ArrayList<ISell>();
    }
```

```java
public abstract class Goods implements ISell {

    private String description;
    private double cost;
    private double price;

    public Goods(String description, double cost, double price) {
        this.description = description;
        this.cost = cost;
        this.price = price;
    }
```

```java
import Behaviours.IPlay;
import Behaviours.ISell;

public abstract class Instrument extends Goods implements IPlay, ISell {

    private String sound;

    public Instrument(String description, double cost, double price, String sound) {
        super(description, cost, price);
        this.sound = sound;
    }
```

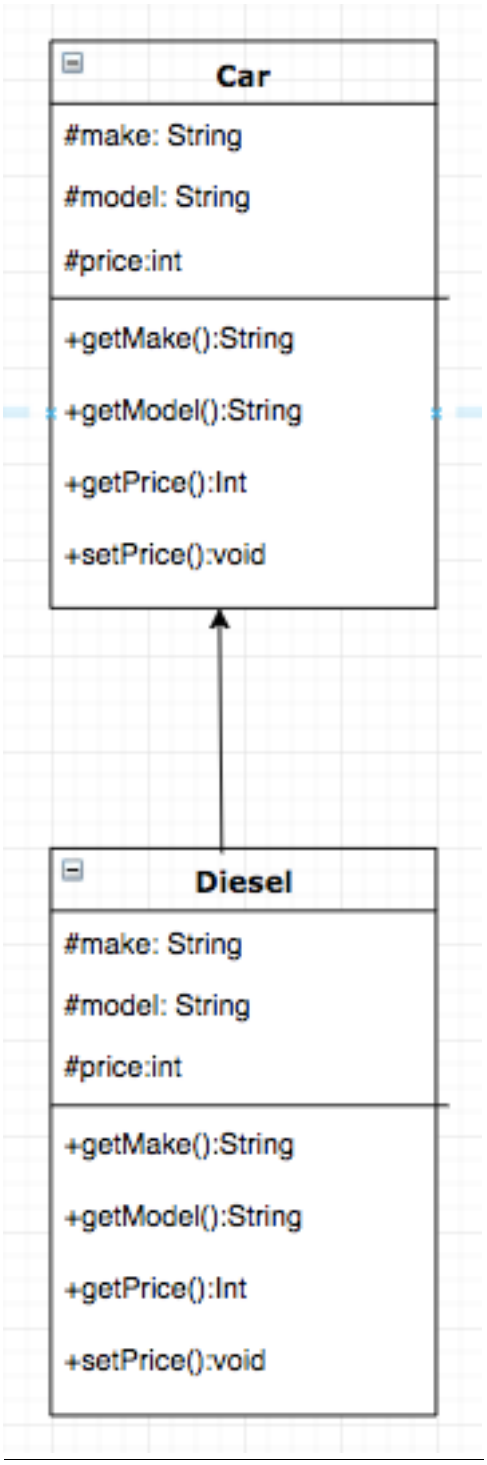
Description here

Shop Class has an ArrayList called stock containing ISell items (Goods and Instruments) which all implement the ISell interface. The ISell items are of different classes but can go into the ArrayList as they share the ISell interface.

Unit	Ref	Evidence	
A&D	A.D. 5	An Inheritance Diagram	
		Description:	

Paste Screenshot here

```java
public class Diesel extends Car {

    public Diesel(String make, String model, String colour, int price) {
        super(make, model, colour, price);
    }

}
```



Description here

The diesel class extends (and inherits from) the car class.

Unit	Ref	Evidence	

I&T	I.T.1	The use of Encapsulation in a program and what it is doing.
		Description:

Paste Screenshot here

```
public abstract class Car implements IComponent {
    private String make;
    private String model;
    private String colour;
    private int price;

    public Car(String make, String model, String colour, int price) {
        this.make = make;
        this.model = model;
        this.colour = colour;
        this.price = price;
    }

    public String getMake() {
        return make;
    }
```

Description here

The properties of the Car class are private and therefore are inaccessible outwith the class. A “getter” is used /called to access them out with and in example above the method getMake() returns the make of car.

Unit	Ref	Evidence	
I&T	I.T.2	Take a screenshot of the use of Inheritance in a program. Take screenshots of: *A Class *A Class that inherits from the previous class *An Object in the inherited class *A Method that uses the information inherited from another class.	
		Description:	

Paste Screenshot here

```java
public abstract class Kaiju implements IAttack, IDamage {

    private String name;
    private int healthValue;
    private int attackValue;

    public Kaiju(String name, int healthValue, int attackValue) {
        this.name = name;
        this.healthValue = healthValue;
        this.attackValue = attackValue;
    }

    public String getName() {
        return name;
    }
```

```java
public String roar(){
    return "Roooooar...";
}
```

```java
public class BigLizard extends Kaiju {

    public BigLizard(String name, int healthValue, int attackValue) {
        super(name, healthValue, attackValue);
    }

}
```

```java
public class BigLizardTest {

    BigLizard bigLizard;
    Tank tank;

    @Before
    public void setUp() throws Exception {
        bigLizard = new BigLizard("Godzilla", 100, 15);
        tank = new Tank("Chieftan", 20, 2);
    }

    @Test
    public void getName() {
        assertEquals("Godzilla", bigLizard.getName());
    }
```

```java
@Test
public void canRoar() {
    assertEquals("Roooooar...", bigLizard.roar());
}
```

Description here

"Kaiju" is the superclass, and each Kaiju has the following properties - name, healthValue and attackvalue. "BigLizard" is a class which extends the Kaiju class and inherits its properties. "Godzilla" is an object in the BigLizard class and inherits all of the properties of the "Kaiju" class and has all of the "Kaiju" class methods available to it. The last 2 screenshots demonstrates the testing of the use of 2 of these methods "getName" and "canRoar" by the Godzilla object "bigLizard)

Unit	Ref	Evidence	
P	P.9	Select two algorithms you have written (NOT the group project). Take a screenshot of each and write a short statement on why you have chosen to use those algorithms.	
		Description:	

Paste Screenshot here

```
Shares.prototype.prepareAllChartData = function (shareItems,price) {
    var dayTotalGain = 0
    var chartArray2 = [["Date","Gain/(loss)"]];
    var carrierArray=[];
    for(var i = 0; i< 253; i++){
        shareItems.forEach(item => {
            dayTotalGain += (((price[item.symbol].chart[i].close)*item.n_of_shares)-
            (item.cost_per_share*item.n_of_shares));
        });
        carrierArray.push(new Date(price.A.chart[i].date));
        carrierArray.push(parseFloat(dayTotalGain));
        chartArray2.push(carrierArray)
        carrierArray=[];
        var dayTotalGain = 0
    };
    PubSub.publish('SharesPortfolio:chart-gain-data-ready', chartArray2);
```

Description here

Chosen this algorithm because I thought it was relatively complicated (for my standard) and involved a loop within a loop to take data originally sourced from a relatively unfriendly data structure (taken initially from an external API) to then format an extracted data set in the form of a new array for use in Google graphs.

```
Shares.prototype.getIndividualApiData = function (symbol) {
    const apiObject = {};
    const request1 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/price`);
    const request2 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/quote`);
    const request3 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/stats`);
    const request4 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/company`);
    const request5 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/relevant`);
    const request6 = new Request(`https://api.iextrading.com/1.0/stock/${symbol}/logo`);
    return Promise.all([request1.get(), request2.get(), request3.get(), request4.get(), request5.get(),request6.get()]).then((values)
    => {
      apiObject.price = values[0];
      apiObject.avgTotalVolume = values[1].avgTotalVolume;
      apiObject.change = values[1].change;
      apiObject.changePercent = values[1].changePercent;
      apiObject.close = values[1].close;
      apiObject.companyName = values[1].companyName;
      apiObject.latestVolume = values[1].latestVolume;
      apiObject.marketCap = values[1].marketCap;
      apiObject.peRatio = values[1].peRatio;
      apiObject.previousClose = values[1].previousClose;
      apiObject.primaryExchange = values[1].primaryExchange;
      apiObject.symbol = values[1].symbol
      apiObject.week52High = values[1].week52High
      apiObject.week52Low = values[1].week52Low
      apiObject.priceToSales = values[2].priceToSales;
      apiObject.priceToBook = values[2].priceToBook;
      apiObject.consensusEPS = values[2].consensusEPS
      apiObject.dividendYield = values[2].dividendYield
      apiObject.exDividendDate = values[2].exDividendDate
      apiObject.CEO = values[3].CEO
      apiObject.companyName = values[3].companyName
      apiObject.description = values[3].description
      apiObject.industry = values[3].industry
      apiObject.sector = values[3].sector
      apiObject.website = values[3].website
      apiObject.peers = values[4].symbols.slice(0,5)
      apiObject.logo = values[5].url
      this.getChartData(symbol);
      PubSub.publish('Shares:api-data-ready',apiObject);
      return apiObject;
    });
```

Chosen this algorithm because it pulls on requests from multiple API's to form data set which is then used in the rest of the app. I couldn't find one API to provide all my data needs and this was the first time I'd used multiple API's. It was also the first time I'd come across "asynchronicity"(?) and the need to use Promises to wait for the API data to be returned before proceeding with the rest of the code.