

PDA Evidence - Chris Burn, G2 Cohort

Week 2:

Demonstrate the use of an array in a program. Take screenshots of:

**An array in a program*

**A function that uses the array*

Two Arrays or strings, pulled into “add_length_of_arrays” method



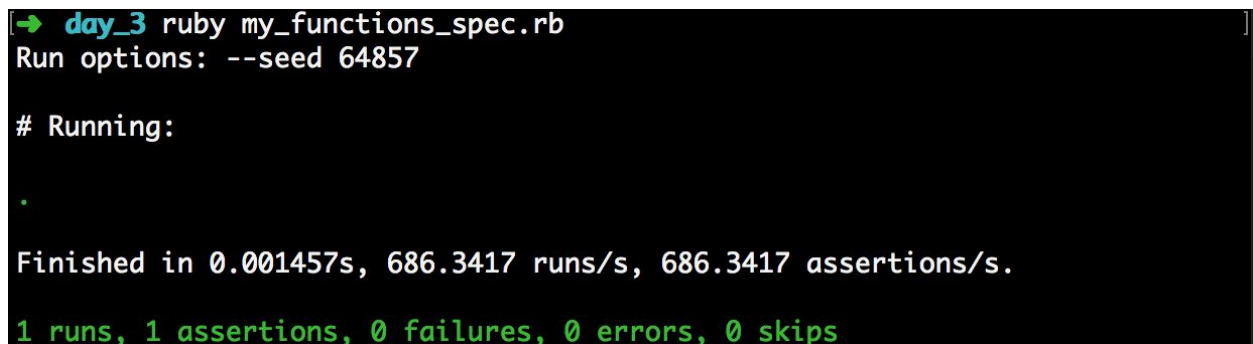
```
my_functions.rb
1 array_1 = [ "Ally", "John" ],
2 array_2 = [ "Steve", "Alan", "Adam" ]
3
4 def add_length_of_arrays(array_1, array_2)
5   add_length_of_arrays = array_1.length() + array_2.length()
6   return add_length_of_arrays
7 end
8
```

**The result of the function running*

Shows TDD testing of method through MiniTest and result of MiniTest in Terminal...



```
my_functions_spec.rb
1 require( 'minitest/autorun' )
2 require( 'minitest/rg' )
3 require_relative( 'my_functions' )
4
5 class My_Functions < MiniTest::Test
6
7
8   def test_add_length_of_arrays
9     result = add_length_of_arrays( [ "Ally", "John" ], [ "Steve", "Alan", "Adam" ] )
10    assert_equal( 5, result )
11  end
12
```



```
[→ day_3 ruby my_functions_spec.rb
Run options: --seed 64857

# Running:

.

Finished in 0.001457s, 686.3417 runs/s, 686.3417 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
```

Demonstrate the use of a hash in a program. Take screenshots of:

*A hash in a program

WITH

*The result of the function running

Hash and an Array with in-built Hashes, WITH testing of function (separate) via MiniTest

```
pet_shop_spec.rb  pet_shop.rb  x

def setup

  @customers = [ #Array with Hashes
    {
      name: "Craig",
      pets: [],
      cash: 1000
    },
    {
      name: "Zsolt",
      pets: [],
      cash: 50
    }
  ]

  @new_pet = { #Simple Hash
    name: "Bors the Younger",
    pet_type: :cat,
    breed: "Cornish Rex",
    price: 100
  }

end

def test_add_pet_to_customer
  customer = @customers[0]
  add_pet_to_customer(customer, @new_pet) #already defined cust[0]
  assert_equal(1, customer_pet_count(customer))
end
```

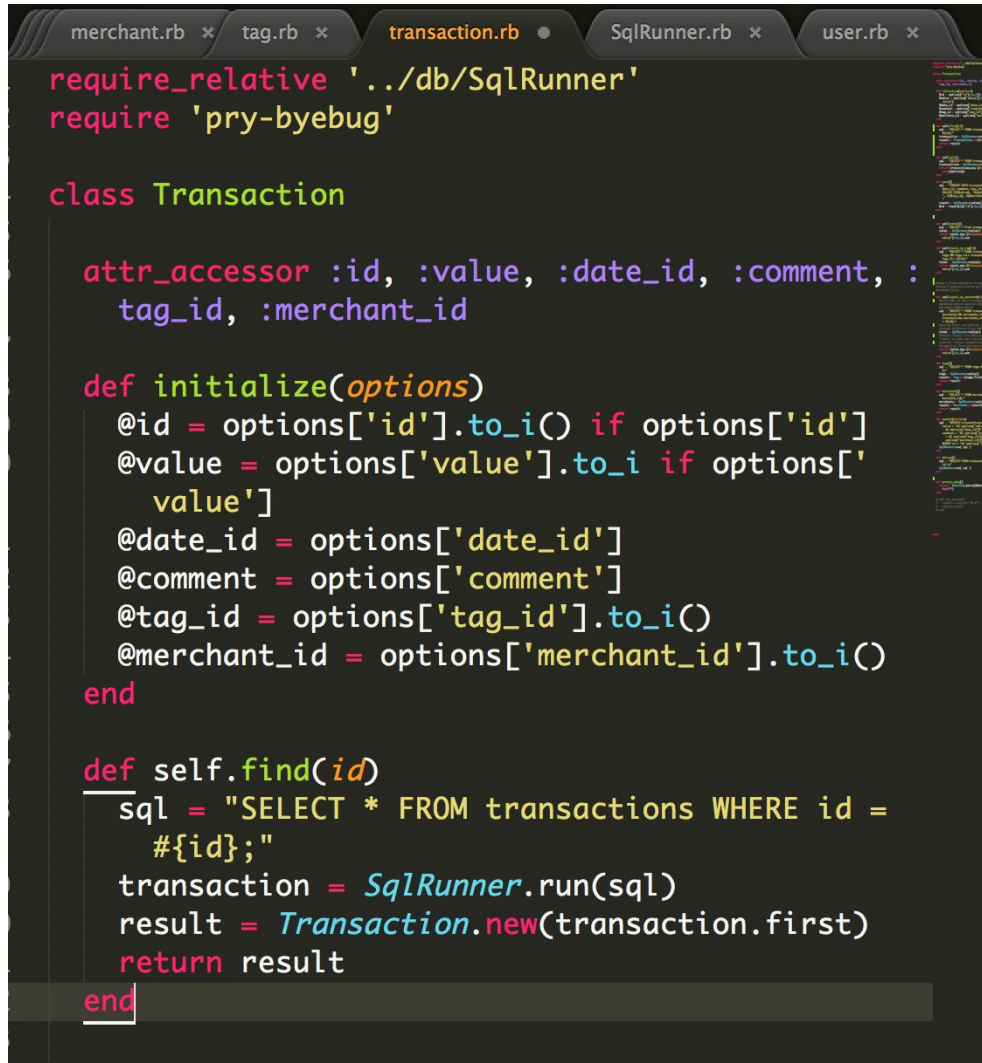
*A function that uses the hash

```
def add_pet_to_customer(customers, new_pet)
  customers[:pets] << new_pet
end
```

Week 3:

Demonstrate searching data in a program. Take screenshots of:

*Function that searches data

A screenshot of a code editor with several tabs at the top: 'merchant.rb', 'tag.rb', 'transaction.rb' (which is active), 'SqlRunner.rb', and 'user.rb'. The 'transaction.rb' file contains the following Ruby code:

```
require_relative '../db/SqlRunner'
require 'pry-byebug'

class Transaction

  attr_accessor :id, :value, :date_id, :comment, :
    tag_id, :merchant_id

  def initialize(options)
    @id = options['id'].to_i if options['id']
    @value = options['value'].to_i if options['value']
    @date_id = options['date_id']
    @comment = options['comment']
    @tag_id = options['tag_id'].to_i
    @merchant_id = options['merchant_id'].to_i
  end

  def self.find(id)
    sql = "SELECT * FROM transactions WHERE id = #{id};"
    transaction = SqlRunner.run(sql)
    result = Transaction.new(transaction.first)
    return result
  end
end
```

*The result of the function running

```
moneyapp=# SELECT * FROM transactions WHERE id = 1;
 id | value | date_id | comment | tag_id | merchant_id
-----+-----+-----+-----+-----+-----
  1 |   30 | 2017-06-04 | Weekly shop. | 1 | 2
(1 row)
```

Demonstrate sorting data in a program. Take screenshots of:

*Function that sorts data

```
require_relative '../db/SqlRunner'
require 'pry-byebug'

class Transaction

  attr_accessor :id, :value, :date_id, :comment, :
    tag_id, :merchant_id

  def initialize(options)
    @id = options['id'].to_i() if options['id']
    @value = options['value'].to_i if options['
      value']
    @date_id = options['date_id']
    @comment = options['comment']
    @tag_id = options['tag_id'].to_i()
    @merchant_id = options['merchant_id'].to_i()
  end

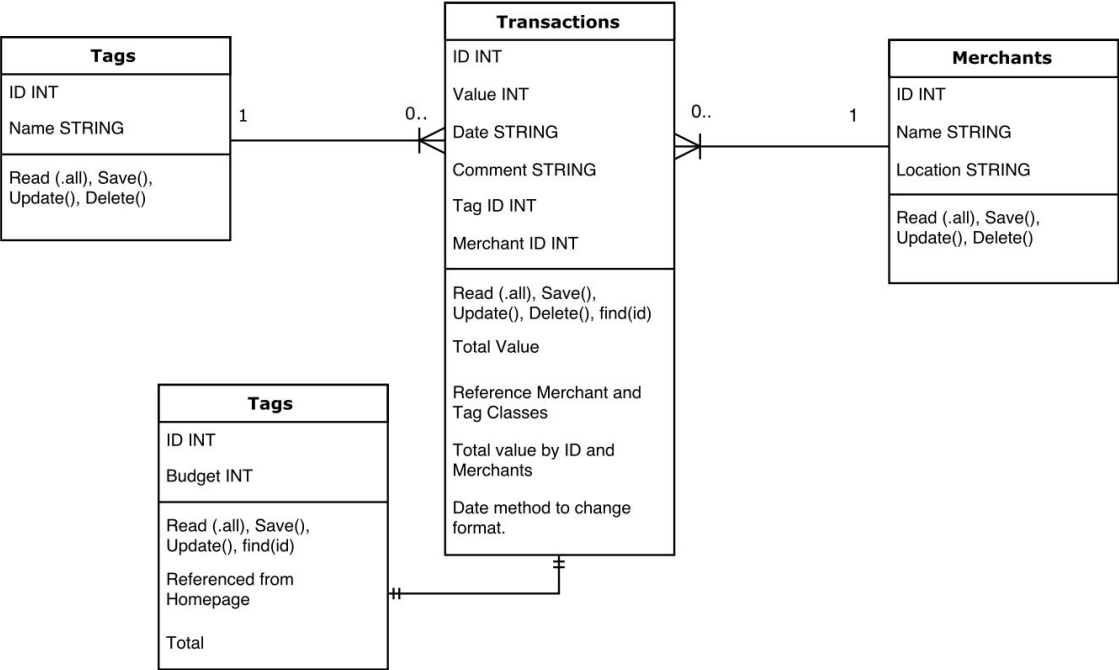
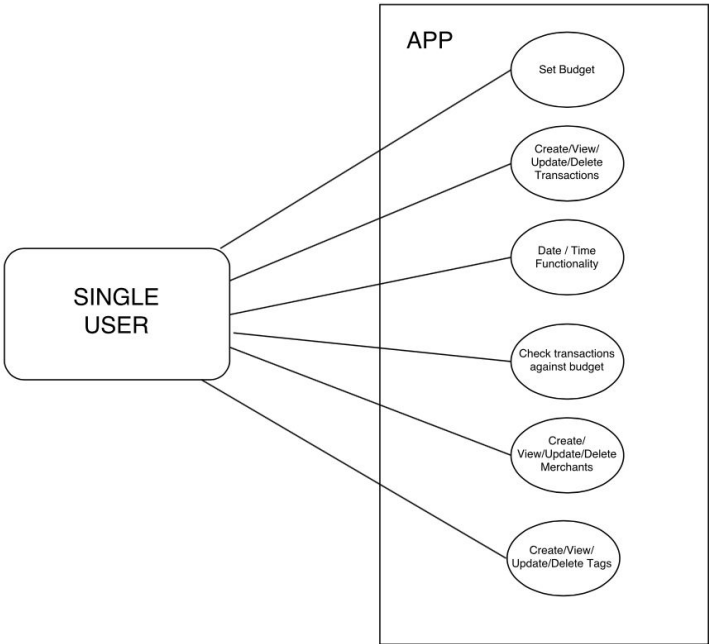
  def self.sort_by_date
    sql = "SELECT * FROM transactions ORDER BY
      date_id;"
  end
end
```

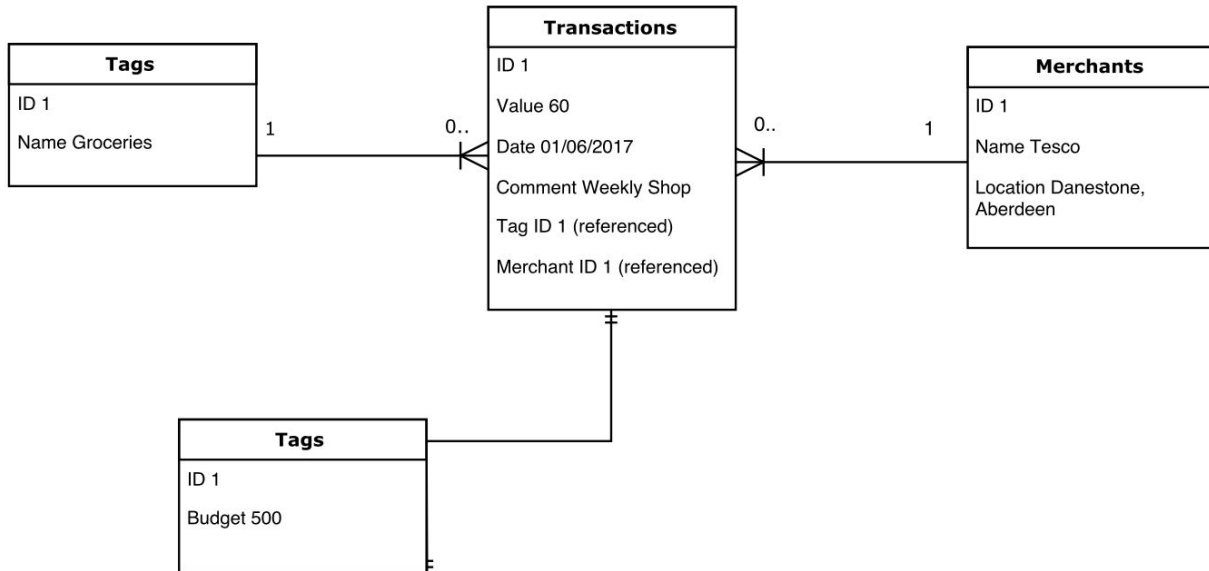
*The result of the function running

id	value	date_id	comment	tag_id	merchant_id
5	39	2017-05-20	Train tickets to Glasgow.	6	7
4	50	2017-05-21	Night out with mates.	7	5
3	15	2017-05-27	Mid-week takeaway.	4	4
7	40	2017-05-28	CDs and Blu-Ray for sisters birthday.	2	6
10	50	2017-05-29	Whisky for me!	9	9
6	45	2017-05-30	Weekly shop.	1	1
8	70	2017-06-01	Paint for home decoration.	3	3
1	30	2017-06-04	Weekly shop.	1	2
2	19	2017-06-05	Replacement window wipers for car.	5	8
11	25	2017-06-09	More pizza!	4	4
9	200	2017-06-11	Flights to Madrid	6	10

(11 rows)

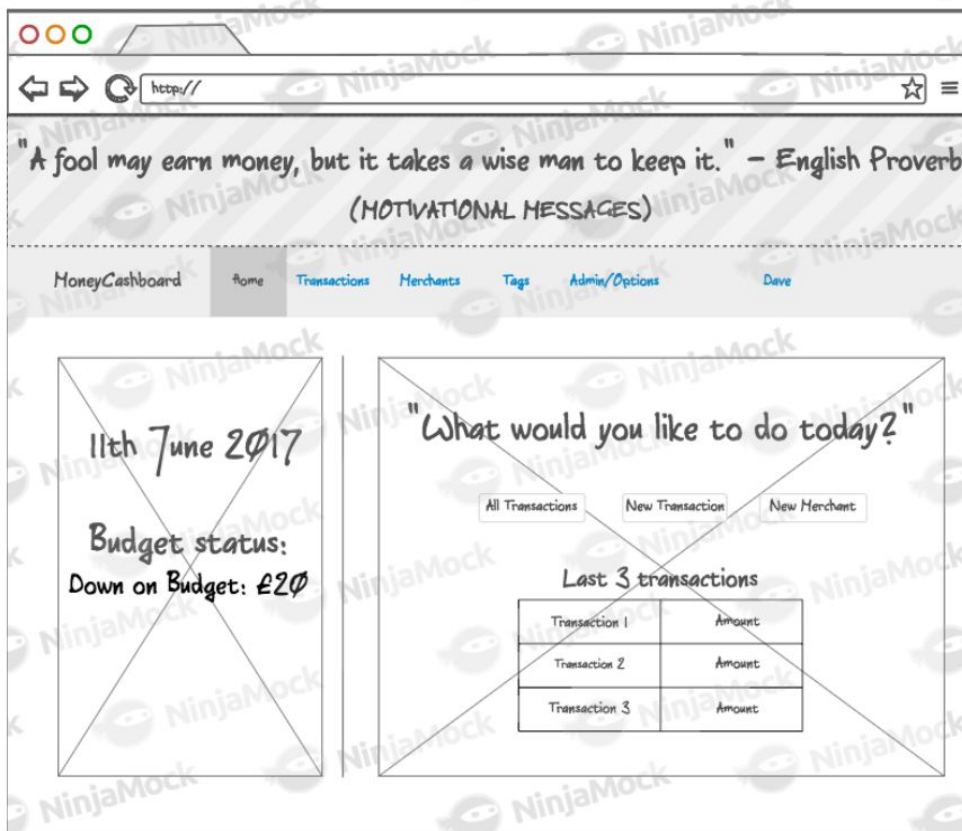
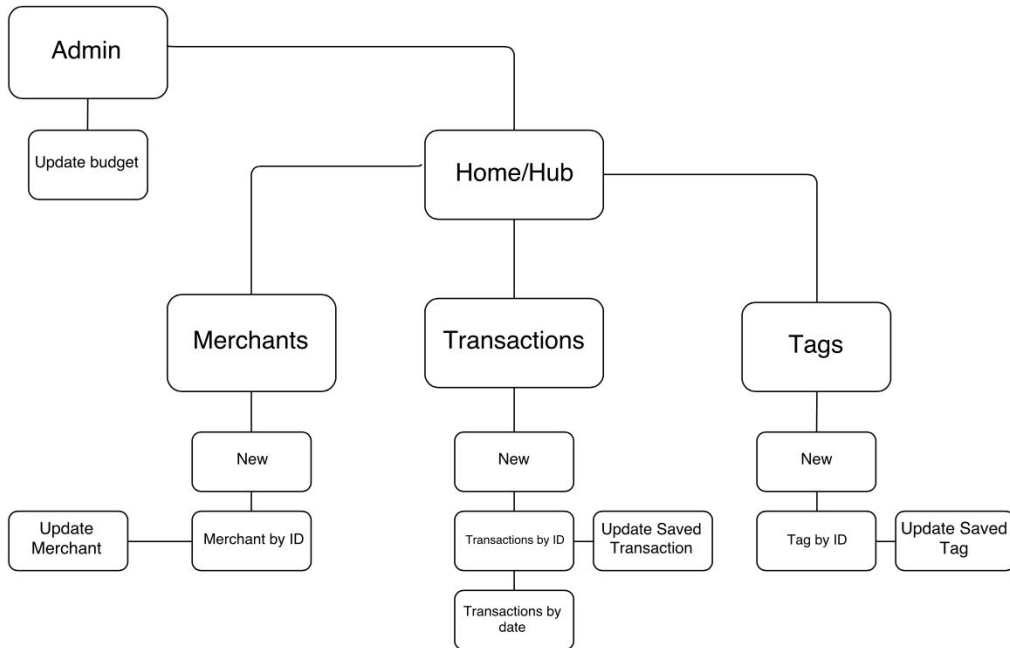
Week 5:

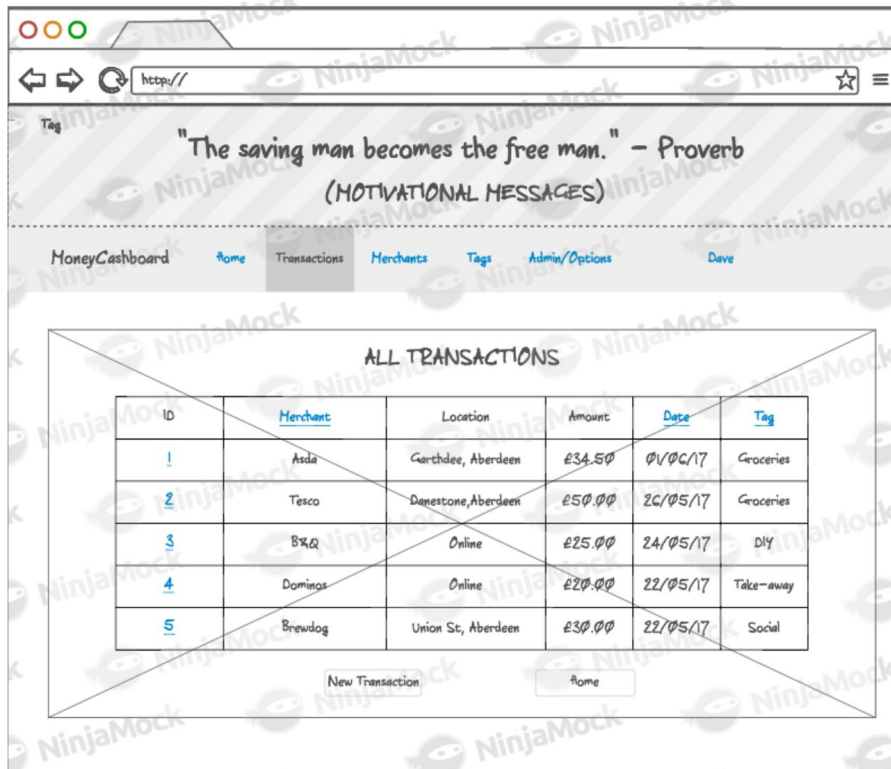




Money CashBoard Implementation Constraints Plan - Sheet1

IMPLEMENTATION CONSTRAINTS PLAN			
Constraint	Possible Effect on Constraint on Product	Solution	
Hardware and Software Problems	Reduced user-experience cross platforms, and on different user machines.	Program to compatible cross browsers and machines including mobile platforms. Use of dynamic displays using FlexBox should help to achieve this.	
Performance Requirements	Reduced performance when frequently calling on database data. Potential for reduced user experience.	Unlikely to have mass traffic as it's an application for personal use. Data being called from database is of relatively small size and is unlikely to cause an issue with performance.	
Persistent Storage and Transactions	Potential for future loss or reduction of free/cheap database capacity may curtail use of application going forward.	To get around this the website design will avoid the use of large media files / graphics / styling which also has potential for an upside on meeting budget requirements.	
Usability	User unable to navigate site with ease, reducing user experience and reduce likelihood of future use and roll out.	Personal use of application, not being used in public. Simple design with large menus/buttons and bold text and colours should reduce potential for navigation errors.	
Budgets	No budget considered but in theory a limited budget will curtail use of premium servers meaning potential for slower performance and risk of downtime. In addition, use of non premium graphics and design may limit the visual appeal of the finished product.	Use of free web tooling and server space will keep web app cost minimal for as long a period of time as application is required.	
Time Limitations	6 day turnaround for finished product will mean limited time to perfect usability and design/finish of product. Additional functionality above MVP may be limited.	Careful planning of project should reduce potential for road-blocks in execution and possible delays to project.	





Pseudo-code example showing a method calling on the result of an INNER JOIN on two database tables:

```
#make a Class method on Transaction Class to
display Transaction value by Merchant from
Merchant Class.

def self.total_by_merchant(id)
  #write SQL to join Transaction and Merchant
  database tables against ID(Transaction) and
  Merchant ID(Merchant)
  sql = "SELECT * FROM transactions INNER JOIN
  merchants ON merchants.id =
  transactions.merchant_id WHERE merchant_id
  = #{id};"
  #assign local variable to running of SQL code
  through SqlRunner Class Ruby file
  total = SqlRunner.run(sql)
  #return "total" variable and 'map'/loop
  'total' to pull out individual transactions
  #sum all 'value' properties (converted to
  Integer) of Transactions called
  return total.map {|transaction| transaction['
  value'].to_i}.sum
end
```


Show user input being processed according to design requirements. Take a screenshot of:

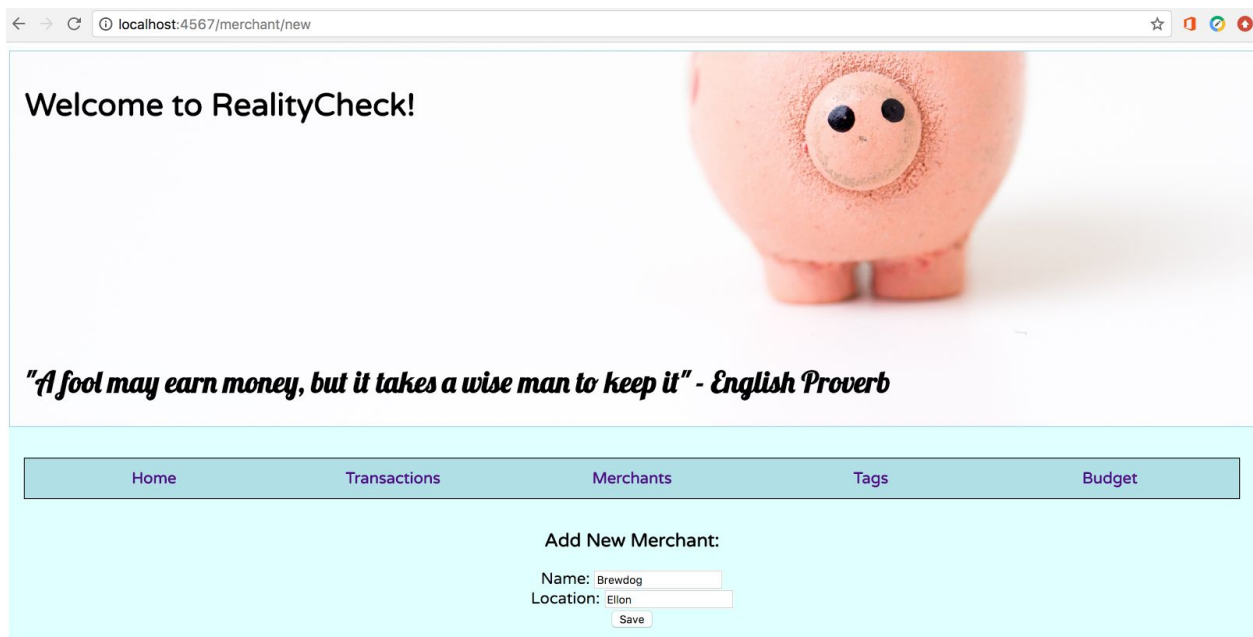
- * The user inputting something into your program
- * The user input being saved or used in some way

Show an interaction with data persistence. Take a screenshot of:

- * Data being inputted into your program
- * Confirmation of the data being saved

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



```
get '/merchant/new' do
  erb(:merchant_new)
end

post '/merchants' do
  @merchant = Merchant.new(params)
  @merchant.save()
  erb(:merchant_create)
end
```

```
controller.rb x merchant_new.erb x console.rb x styles.css x x
1 <div class="cell">
2
3 <h3>Add New Merchant:</h3>
4
5 <form action="/merchants" method="POST">
6
7 <div>
8 <label for="name">Name:</label>
9 <input type="text" id="name" name="name" required
10 >
11 </div>
12
13 <div>
14 <label for="location">Location:</label>
15 <input type="text" id="location" name="location"
16 required>
17 </div>
18
19 <input type="submit" value="Save">
20 </form>
21 </div>
```

```
controller.rb x merchant_create.erb x console.rb x styles.css x x
1 <div class="cell">
2 <h3>Merchant saved!</h3>
3
4 <h3><a href="/merchants">Back to Merchants</a></h3>
5
6 <h3><a href="/transaction/new">Back to New
7 Transaction</a></h3>
8 </div>
```

```

require_relative '../db/SqlRunner'
require 'pry-byebug'

class Merchant

  attr_accessor :id, :name, :location

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

  def self.all()
    sql = "SELECT * FROM merchants;"
    merchants = SqlRunner.run(sql)
    return merchants.map {|options| Merchant.new(
      options)}
  end

  def save()
    sql = "INSERT INTO merchants (name, location)
      VALUES ('#{@name}', ' #{@location}')"
    RETURNING *;"
    result = SqlRunner.run(sql)
    @id = result[0]['id'].to_i()
  end
end

```

```

    result = SqlRunner.run(sql)
    @id = result[0]['id'].to_i()
  end

  def self.find(id)
    sql = "SELECT * FROM merchants WHERE id = #{id};"
    merchant = SqlRunner.run(sql)
    result = Merchant.new(merchant.first)
    return result
  end

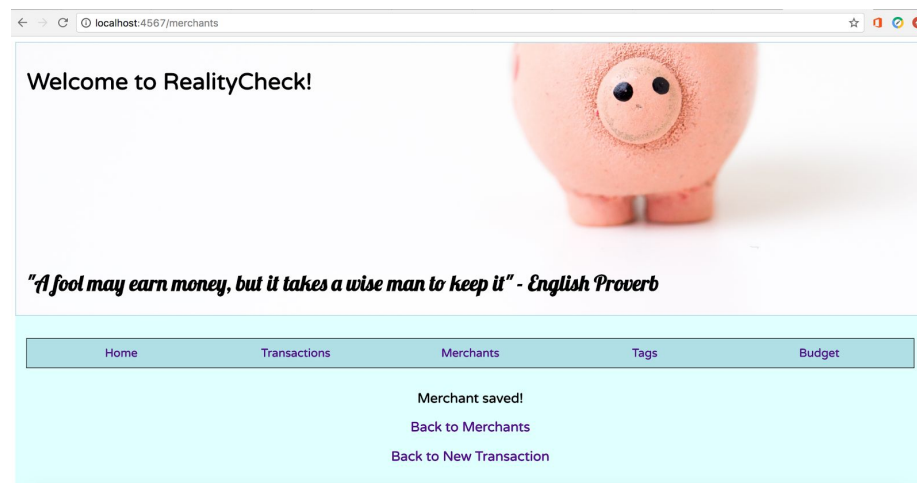
  def update(options)
    sql = "UPDATE merchants SET
      name = '#{ options['name'] }',
      location = '#{ options['location'] }'
      WHERE id = '#{ options['id'] }';"
    SqlRunner.run( sql )
  end

  def delete()
    sql = "DELETE FROM merchants WHERE id=#{ @id };"
    SqlRunner.run( sql )
  end
end
end

```

```
17 | Amazon UK      | Online
18 | Scotrail       | Train Station, Aberdeen
19 | Halfords       | Berryden, Aberdeen
20 | Whisky Exchange | Online
21 | British Airways | Online
22 | Falkirk Council | West Bridge Street, Falkirk
23 | Asda           | Garthdee, Aberdeen
24 | Tesco          | Danestone, Aberdeen
25 | B&Q            | Online
26 | Dominos        | Online
27 | Brewdog        | Merchant City, Glasgow
28 | Amazon UK      | Online
29 | Scotrail       | Train Station, Aberdeen
30 | Halfords       | Berryden, Aberdeen
31 | Whisky Exchange | Online
32 | British Airways | Online
33 | Falkirk Council | West Bridge Street, Falkirk
34 | Brewdog        | Ellon
(34 rows)

(END)
```



Week 6:

Demonstrate the use of Polymorphism in a program:

POLYMORPHISM

- Capacity to take on different forms.
- A language's ability to process objects of various types and classes through a single, uniform interface.
- Example, how a parent class refers to a child class object
- Any object that satisfies more than one IS-A relationship is polymorphic in nature.
- Fair to say that every Object in Java is polymorphic in nature, each on passes IS-A test for itself and for Object class.

E.g. Animal class, Cat IS sub-class of Animal. Cat satisfies IS-A relationship of own type and its Super-class, Animal.

```
package com.example.user.multipleclasshomework;

import java.util.IllegalFormatException;

/**
 * Created by user on 20/06/2017.
 */

public class Employee {

    private int empId;
    private String name;
    private String ssn;
    private double salary;

    public Employee(int empId, String name, String ssn, double salary) {
        this.empId = empId;
        this.name = name;
        this.ssn = ssn;
        this.salary = salary;
    }

    public int getEmpId() {
        return empId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        if(name != null && name != "") {
```

```
package com.example.user.multipleclasshomework;

/**
 * Created by user on 20/06/2017.
 */

public class Manager extends Employee {

    private String deptName;

    public Manager(int empId, String name, String ssn, double salary, String deptName) {
        super(empId, name, ssn, salary);
        this.deptName = deptName;
    }

    public String getDeptName() {
        return this.deptName;
    }

}
```

```

package com.example.user.multipleclasshomework;

/**
 * Created by user on 20/06/2017.
 */

public class Director extends Manager {

    private double budget;

    public Director(int empId, String name, String ssn, double salary, String deptName, double budget) {
        super(empId, name, ssn, salary, deptName);
        this.budget = budget;
    }

    public double getBudget(){
        return this.budget;
    }

}

```

```

import org.junit.Test;

import static org.junit.Assert.assertEquals;

/**
 * Created by user on 20/06/2017.
 */

public class TestDirector {

    Director director;

    @Before
    public void before(){
        director = new Director(20, "Sean Jones", "FGHJ", 50000.00, "Sausages", 500000.00);
    }

    @Test
    public void canRaiseSalary(){
        director.raiseSalary(80000.00);
        assertEquals(80000.00, director.getSalary(), 0.01);
    }

    @Test
    public void canSetName(){
        director.setName("Brent Jones");
        assertEquals("Brent Jones", director.getName());
    }

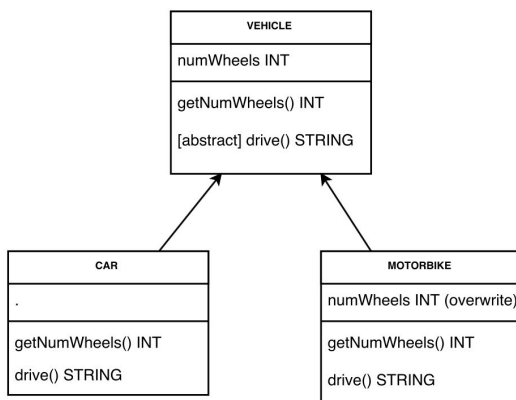
}

```

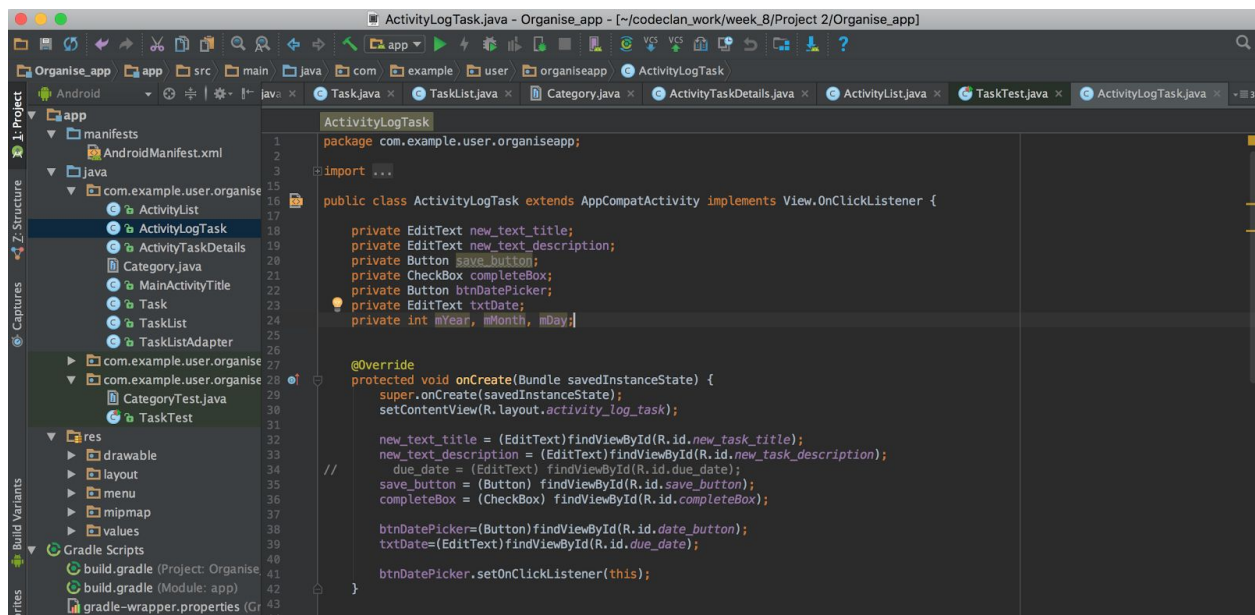
>> All 2 tests passed - 5ms

Week 7:

An Inheritance Diagram



Take a screenshot of an example of encapsulation in a program



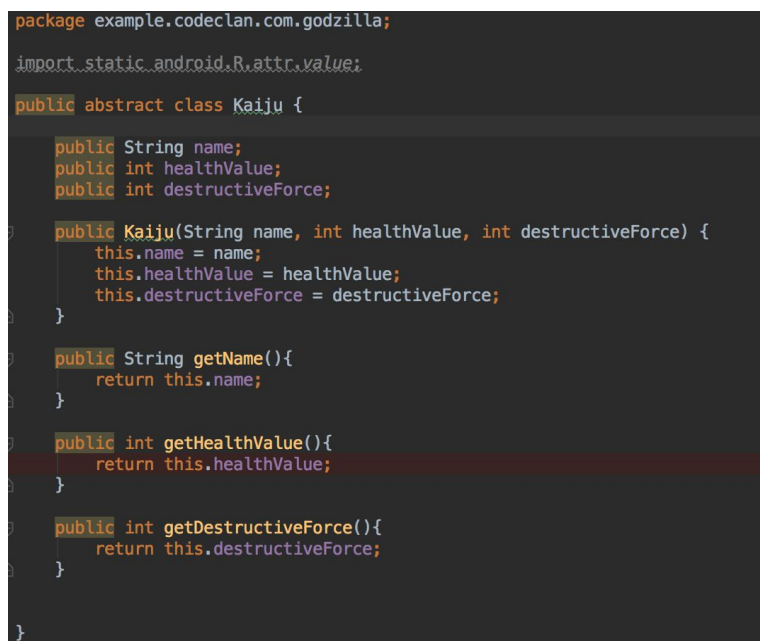
Take a screenshot of the use of Inheritance in a program. Take screenshots of:

*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.



```

package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Godzilla extends Kaiju implements Attack {

    public Godzilla(String name, int healthValue, int destructiveForce) {
        super(name, healthValue, destructiveForce);
    }

    //...

    public String roar(){
        return "ROAR!";
    }

}

```

```

package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Pikachu extends Kaiju implements Attack {

    public Pikachu(String name, int healthValue, int destructiveForce){
        super(name, healthValue, destructiveForce);
    }

    public String attack(){
        return "Kablammo!!";
    }

    public String roar(){
        return "PIKA! PIKA!";
    }

}

```

```

package example.codeclan.com.godzilla;

/**
 * Created by user on 21/06/2017.
 */

public class Runner {

    public static void main(String[] args){

        Godzilla godzilla = new Godzilla("Zilla", 120, 50);
        Pikachu pikachu = new Pikachu("Pika", 90, 35);
        SkyScraper skyScraper = new SkyScraper("largebuilding", 30);
        Townhall townhall = new Townhall("smallbuilding", 20);

        System.out.println("Zilla enters the City, looks like trouble!");
        System.out.println("Zilla looks pretty healthy with health " + godzilla.getHealthValue());

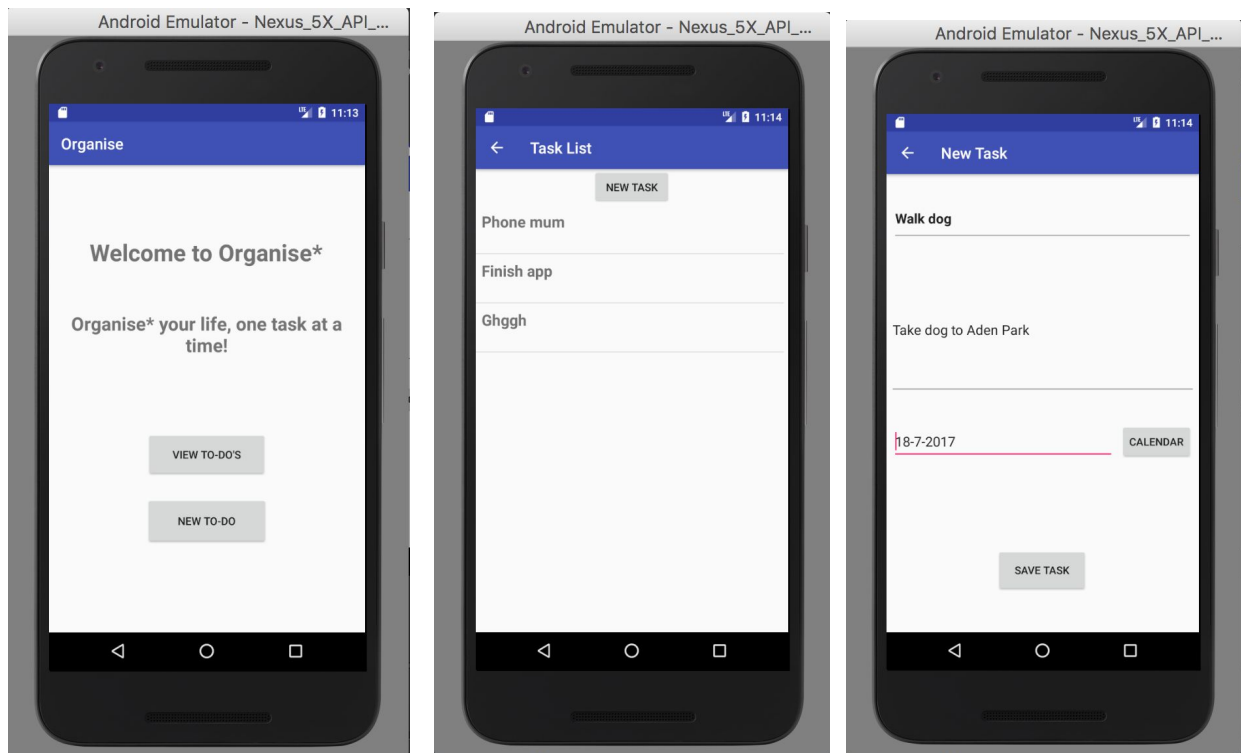
    }

}

```

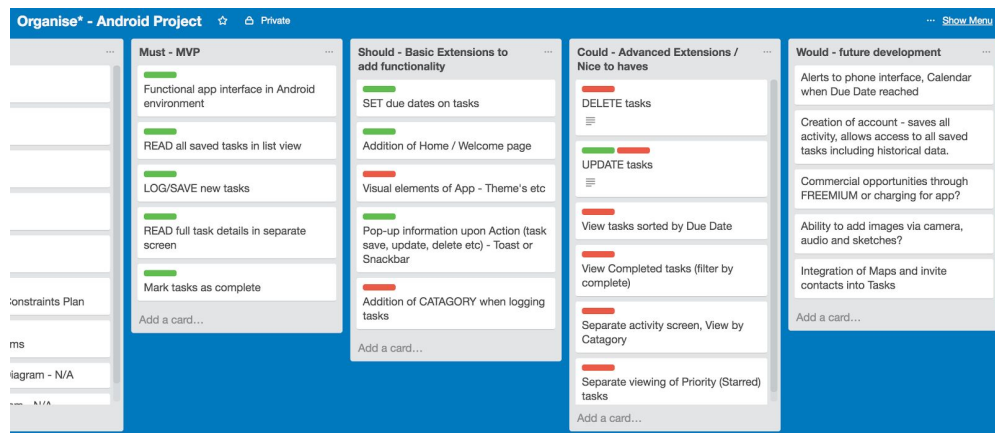
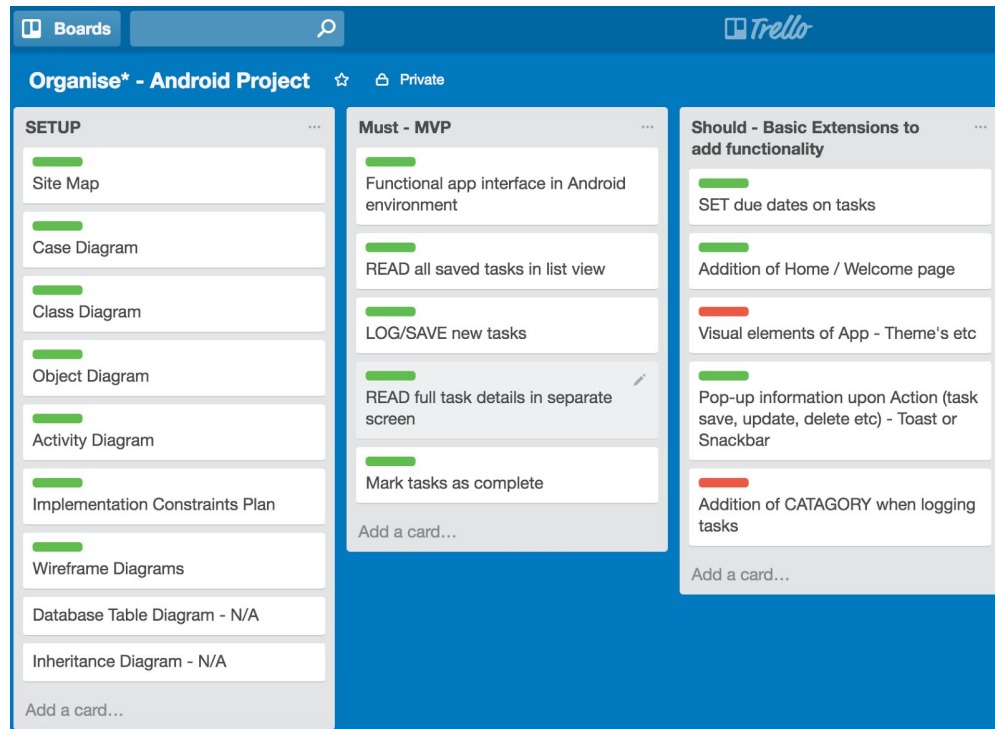
Take a screenshot of one of your projects where you have worked alone and attach the Github link.

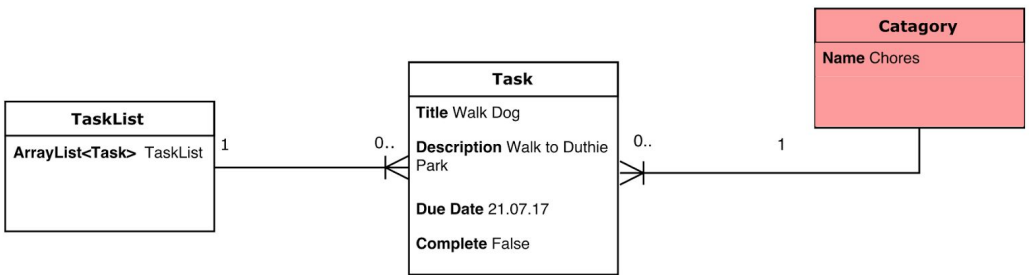
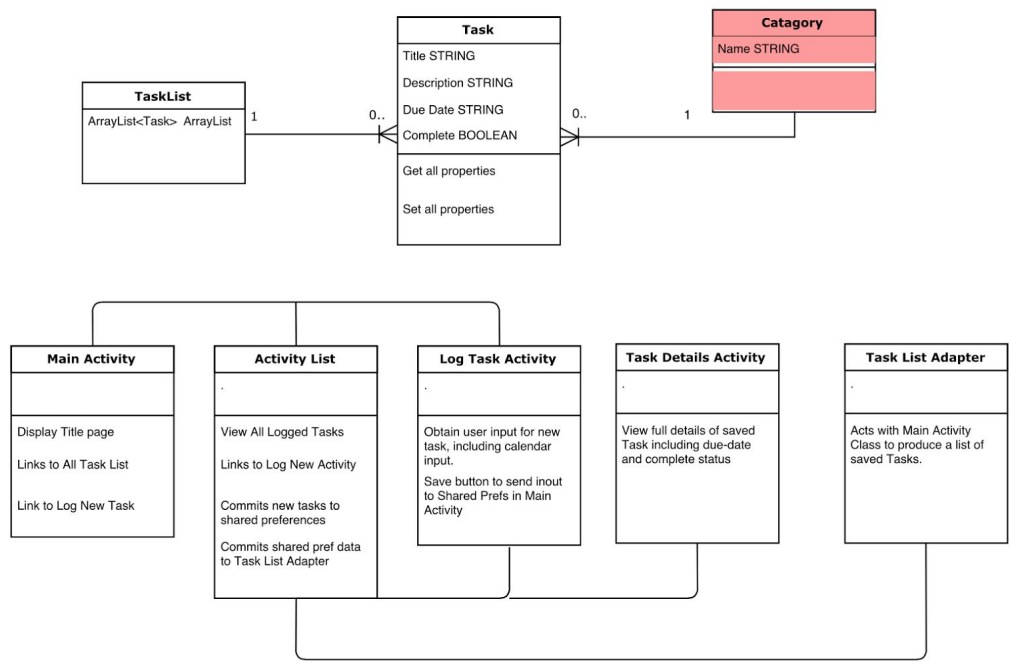
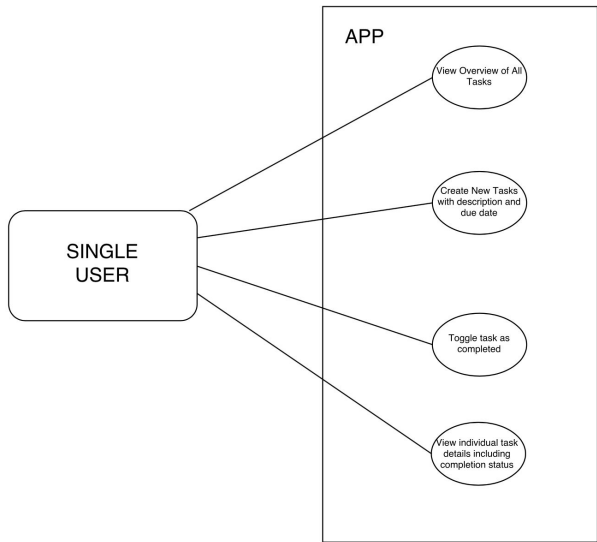
https://github.com/chris-burn/To-Do_list_project

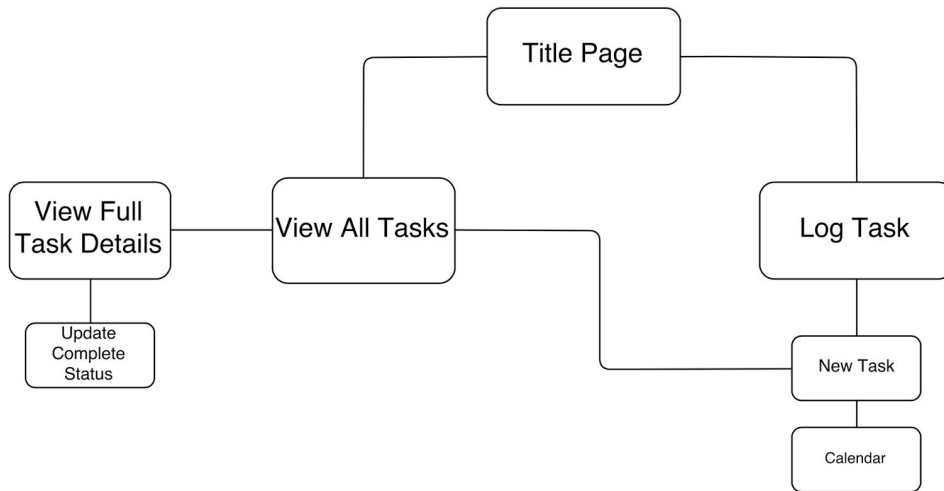


Take screenshots or photos of your planning and the different stages of development to show changes.

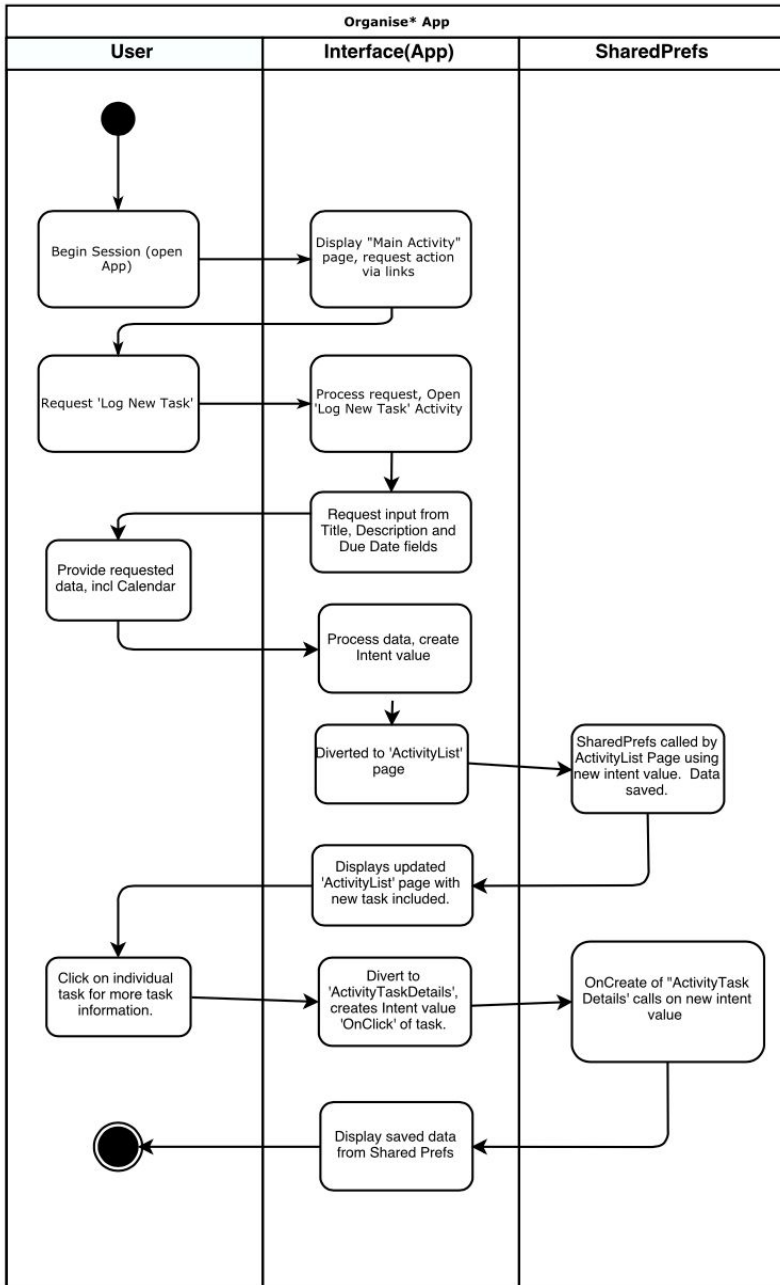
Android App, PROJECT PLANNING:







IMPLEMENTATION CONSTAINTS PLAN		
Constraint	Possible Effect on Constraint on Product	Solution
Hardware and Software Problems	Limited use - Android App only	Multiple versions of apps would be required. No instant solution.
Performance Requirements	Potential limit to performance especially as the data size increases. Minimal visuals in app at present should limit issues with regards app performance.	Unlikely to have mass traffic as it's an application for personal use. Data being used for App use is of relatively small size and is unlikely to cause an issue with performance. Use of SQL database would solve any data volume concerns.
Persistant Storage and Transactions	Data persistence achieved through SharedPreferences internal to app only, no database use which may limit app potential.	To get around this the website design will avoid the use of large media files / graphics /styling which also has potential for an upside on meeting budget requirements.
Usability	User unable to navigate site with ease, reducing user experience and reduce likelihood of future use and roll out.	Simple design layout with large menus/buttons and bold text and colours should reduce potential for navigation errors. Personal use of application, not for public distribution in current state.
Budgets	No budget considered but in theory a limited budget will curtail use of premium servers meaning potential for slower performance and risk of downtime. In addition, use of non premium graphics and design may limit the visual appeal of the finished product.	Use of free web tooling and server space will keep web app cost minimal for as long a period of time as application is required.
Time Limitations	6 day turnaround for finished product means limited design/finish of product. Additional functionality above MVP may be limited.	Careful planning of project should reduce potential for road-blocks in execution and possible delays to project.



<

Add New Task

⋮

Title: Buy Groceries

Description:
Shopping to be done at Supermarkets:
To buy –
Bread,
Milk,
Eggs,
Ham and Pineapple pizza.

Due date: 15/07/17

Mark as Priority ☒

Add Task

TASK ADDED

<

My Tasks

⋮

Buy Groceries
Shopping ☒

Walk Dog
Dog ☐

Wash Dishes
kitchen ☐

Prepare Presentation
Work ☐

Pick up Significance Other
Family ☐

Pay Council Tax
Bills ☐

Cut Grass
Garden ☐

Phone Mum
Family ☐

<

Task Details

⋮

Buy Groceries ☒

Shopping to be done at Supermarkets:
To buy –
Bread,
Milk,
Eggs,
Ham and Pineapple pizza.

Due date: 15/07/17

Mark as Complete ☐

Delete Task Edit Task