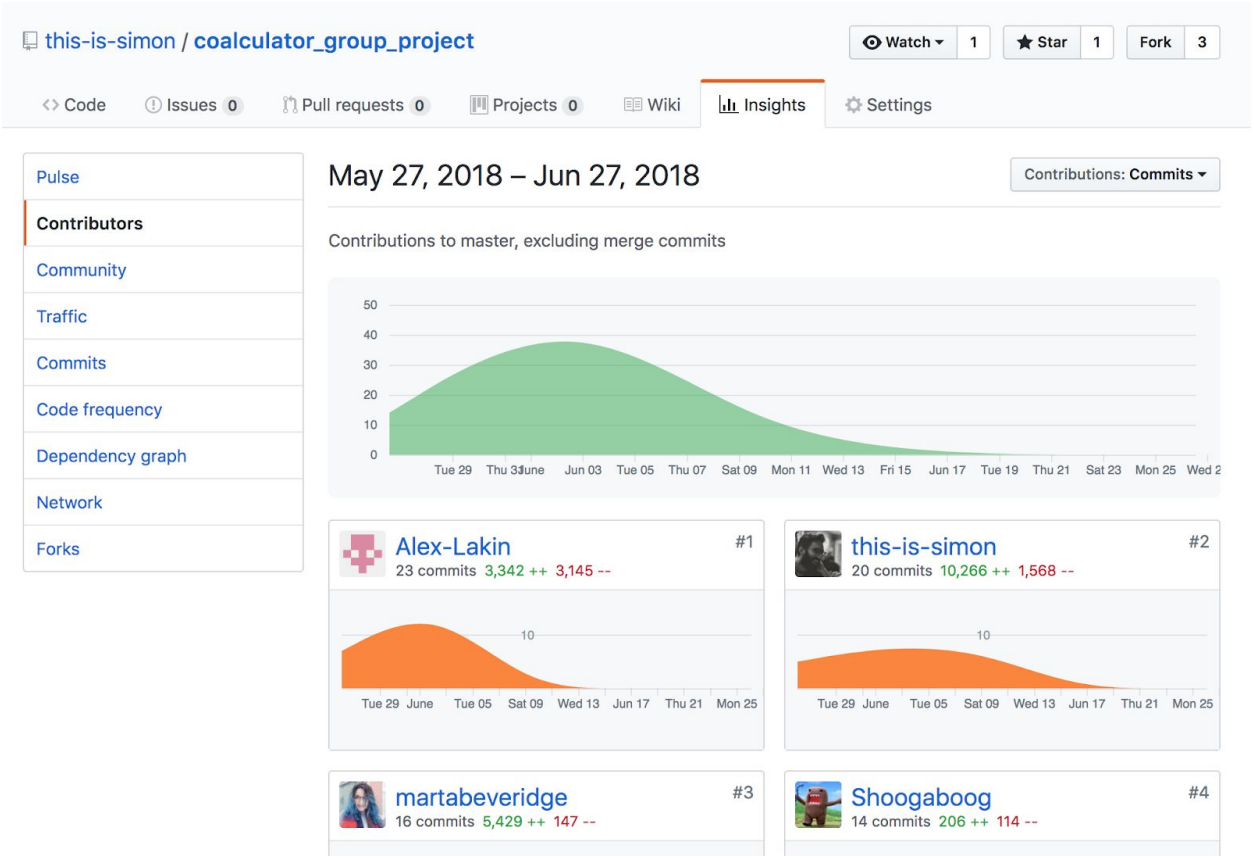
PDA Project Unit

Simon Atkins
Cohort e20

P.1 Github Contributors Page



P.2 Project Brief

Coalculator: The Carbon Footprint Calculator

The Coalculator is a week-long group project completed by Simon Atkins, Marta Beveridge, Lewis McLean and Alex Lakin in Week 14 of Codeclan's 16-week intensive Software Programming Bootcamp.

Brief

We created an MVP for our project which included the following:

- A form in which to enter user data (e.g. Car Miles, Train Miles)
- A total carbon footprint calculated by the user input
- A graph showing the breakdown of carbon tonnes compared to the UK average
- Data persistence through using a database

Extensions

We would have liked to add an additional chart in terms of a piechart breakdown of the user's total carbon footprint but we ran out of time. In future the programme can include more inputs for carbon useage (such as shopping habits, as these make a big impact on an individual's carbon footprint). Another interface can be added to calculate the carbon footprint of a household.

Planning and Methods

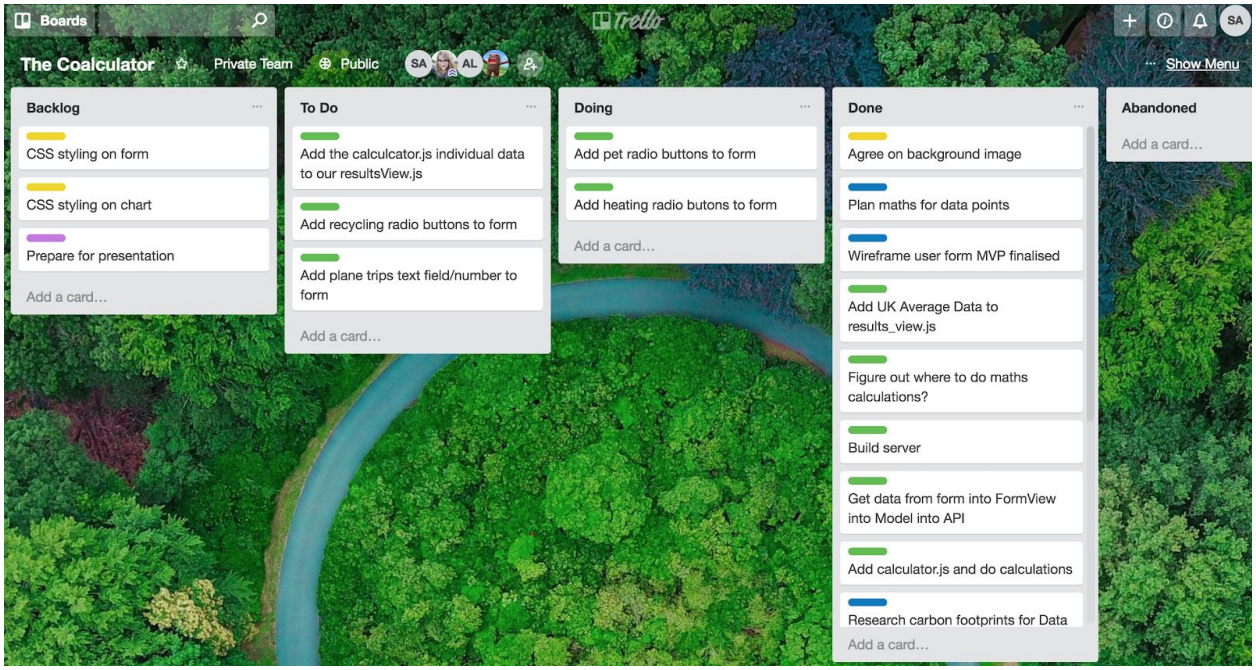
We adopted an Agile methodology with 1-1.5 hour sprints focusing on particular features and functionality, as well as doing daily stand-ups and retrospectives. We took turns in being Scrum Master each day and regularly took turns to drive.

After writing our MVP we created a few wireframes and user journeys so as to provide a decent UX. We drew UML diagrams in order to design and keep track of developments in our programme.

We built this programme from the back end to the front end, starting with our server and routers, using two models (co2.js and calculator.js) and finishing with three front-end views (form_view.js, results_view.js and graph_view.js). We used the PubSub method to send data around our app.

The form view holds the form for user input, the results view holds the users results (i.e. their calculated carbon footprint), and the graph view is nested within the form view.

P.3 Use of Trello



P.4 Acceptance Criteria

Acceptance Criteria:

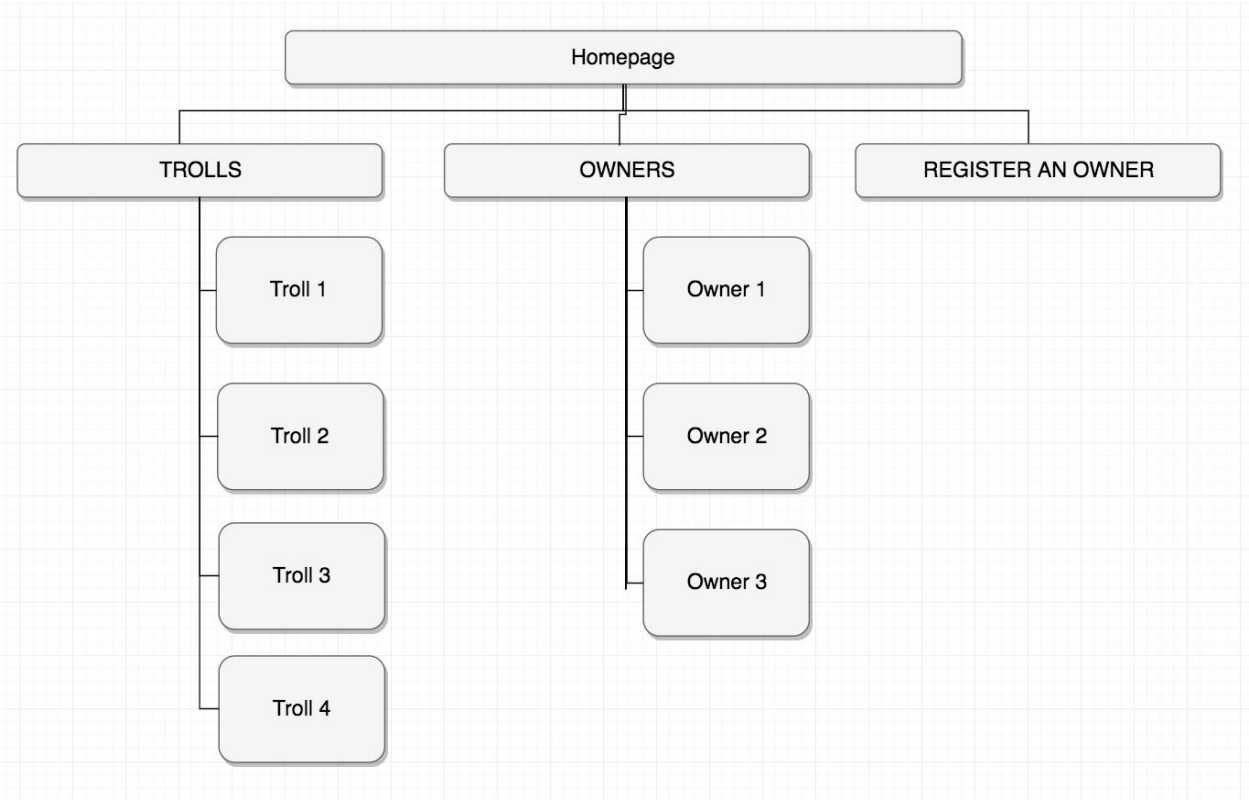
- User must be able to enter data into form
- User must be able to see text comparison of total carbon emissions compared with UK national average
- User must be able to compare individual targets against UK national average in graph
- Data must persist so that user can return to webpage without having to input data again

Test Plan:

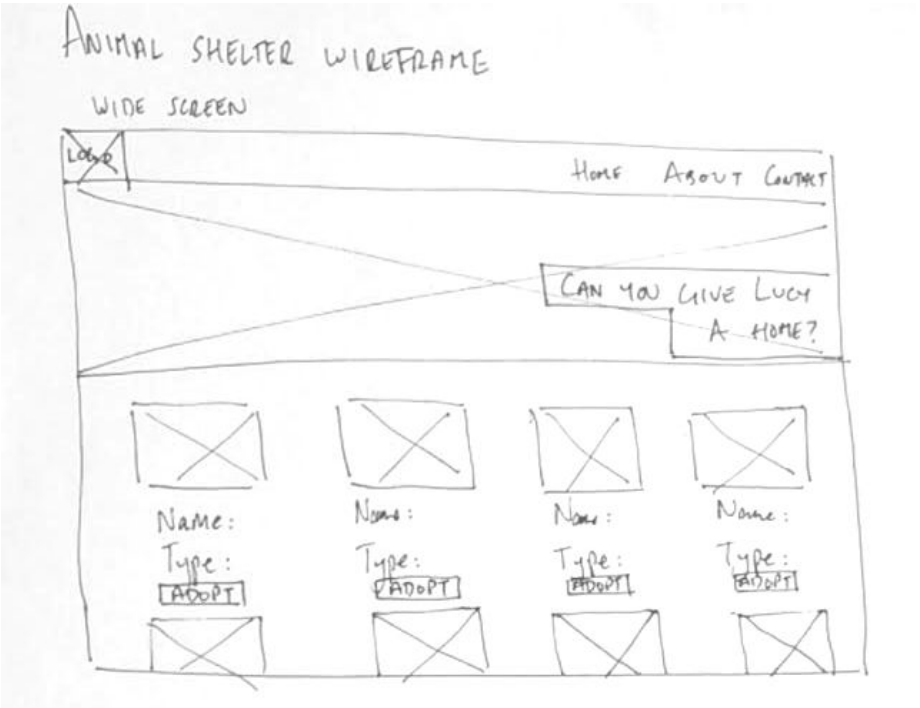
- Test that input fields in form work
- Test that data is sent to database
- Test that data is retrieved from database to populate dynamic fields
- Test that dynamic fields and graph are being correctly populated by database
- Test that data in dynamic fields and graph persist when user leaves webpage and returns

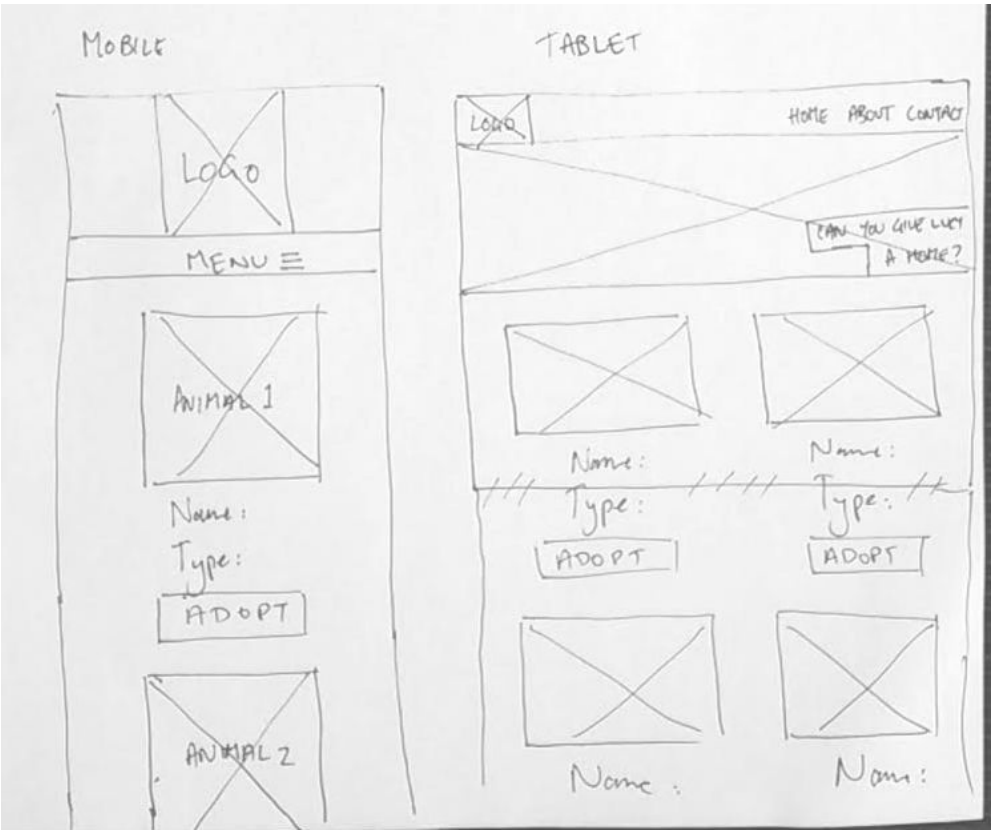
P.5 User Sitemap

User sitemap for Troll Shelter project

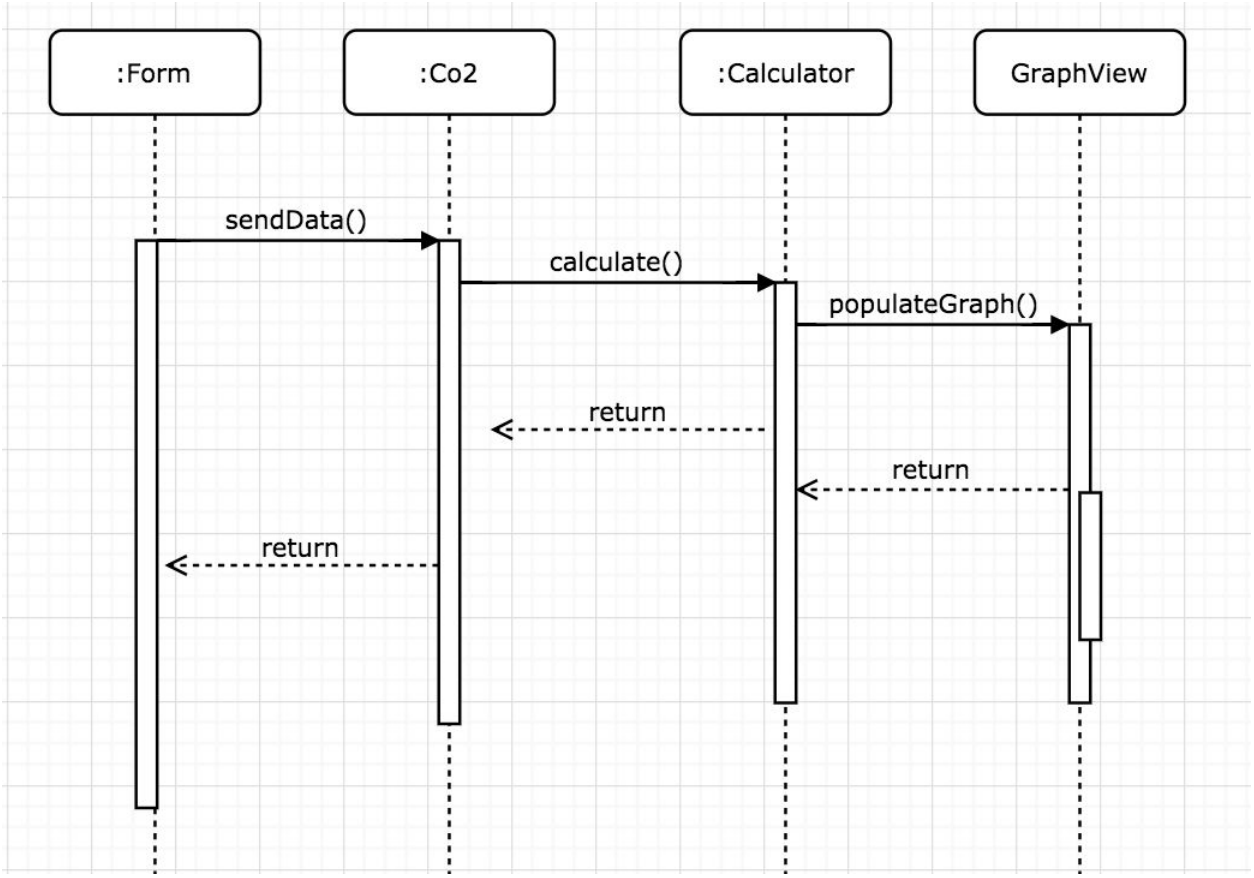


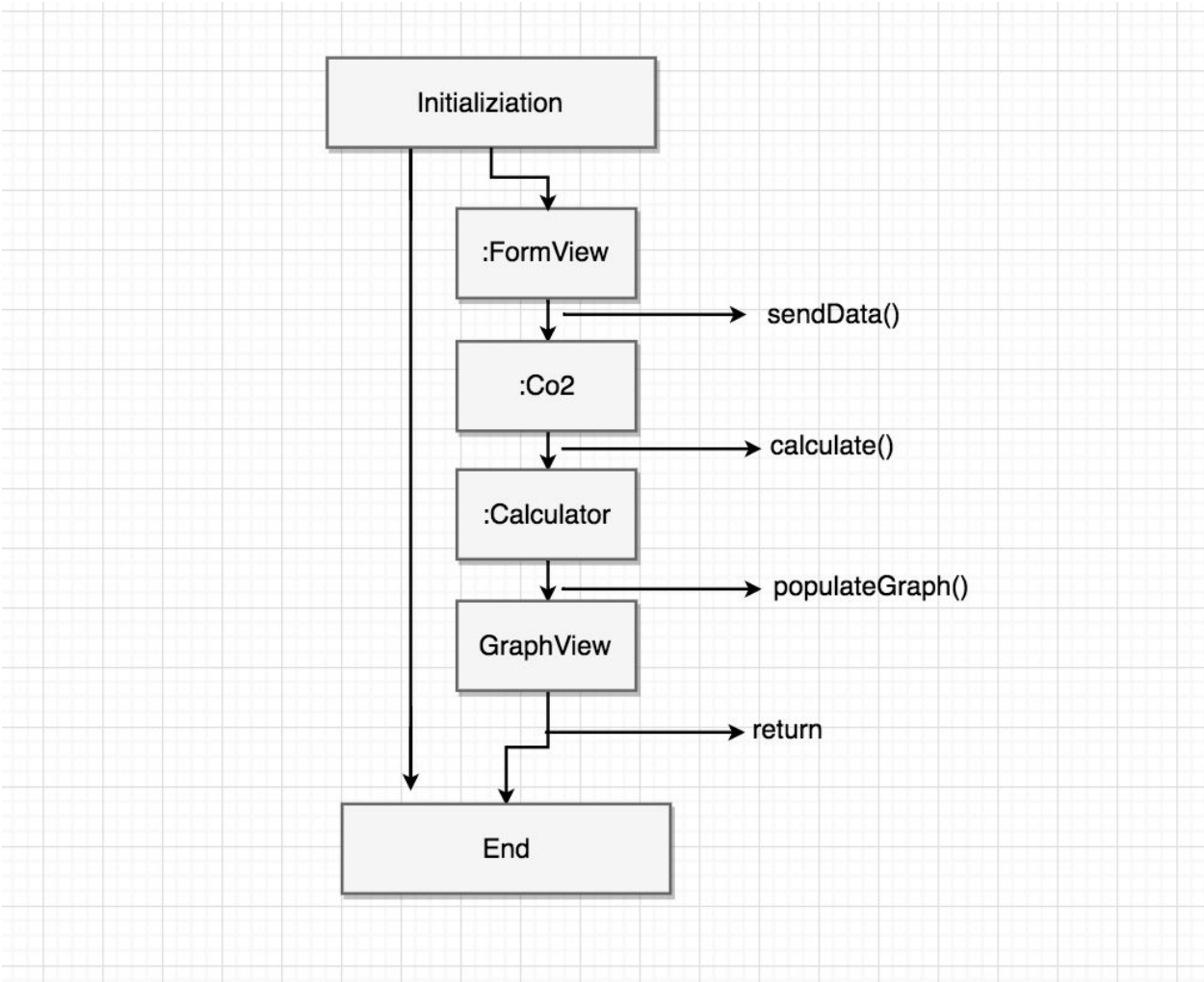
P.6 Wireframe Designs



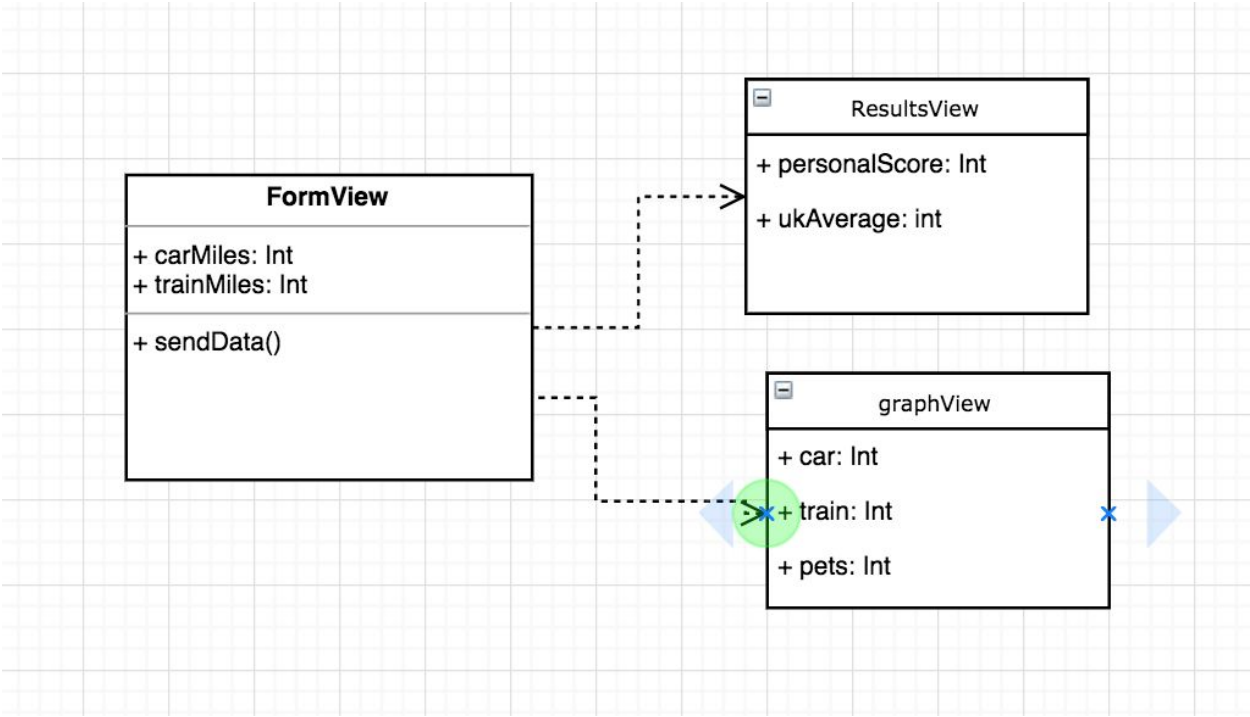

MOBILE
TABLET
LOGO
MENU ≡
ANIMAL 1
Name:
Type:
ADOPT
ANIMAL 2
LOGO
HOME ABOUT CONTACT
CAN YOU GIVE LUCY A HOME?
Name:
Type:
ADOPT
Name:
Type:
ADOPT
Name:
Name:

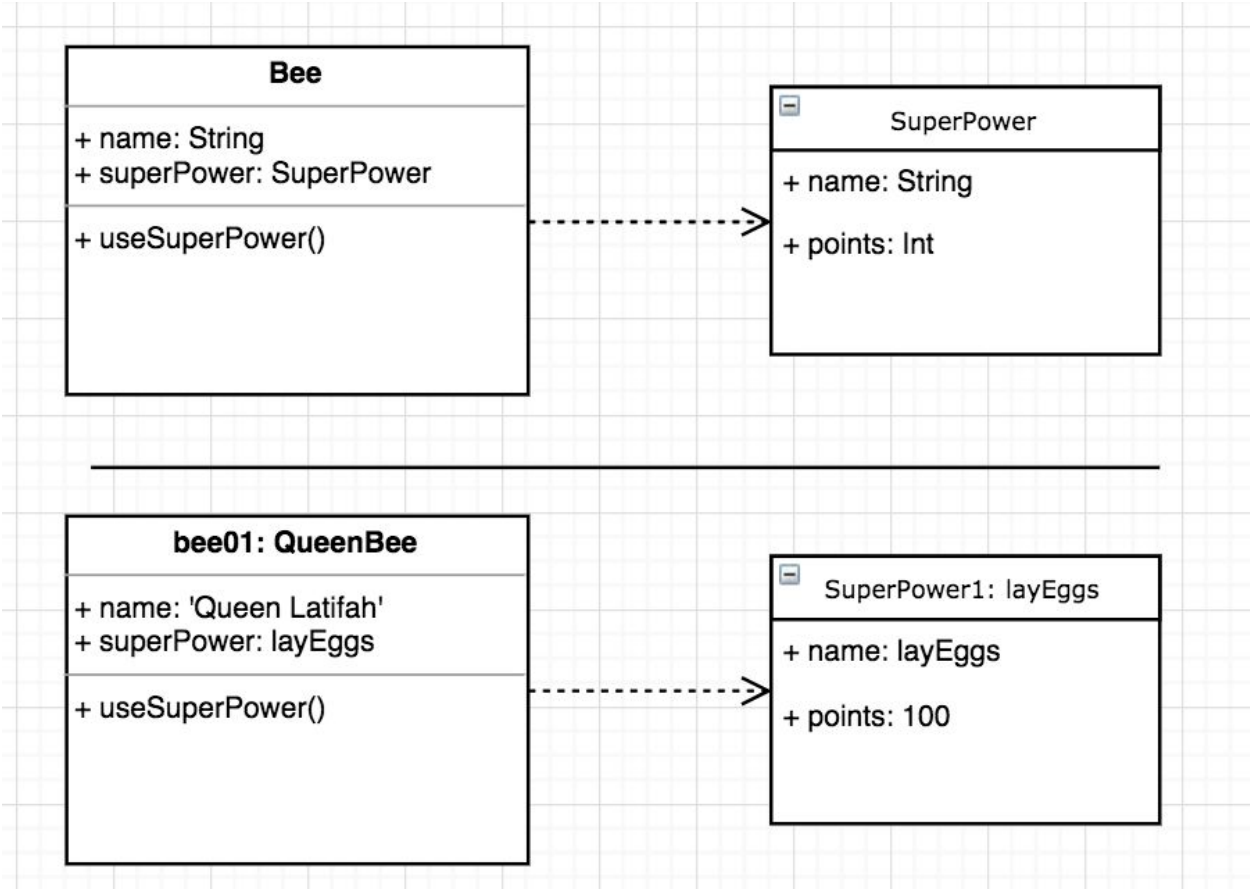
P.7 System Interactions Diagrams




Initializiation
:FormView
sendData()
:Co2
calculate()
:Calculator
populateGraph()
GraphView
return
End

P.8 Two Object Diagrams





P.9 Two Algorithms

On this example please take a screenshot and write what it is doing and why you have decided to use it

1. **Ticket-selling algorithm**

```java
    public Ticket sellTicket(Destination destination){
        for (int i = 0; i < ticketsForSale.size(); i++) {
            if (ticketsForSale.get(i).getDestination() == destination){
                return ticketsForSale.remove(i);
            }
        }
        return null;
    }
```

The algorithm sorts through each ticket in the ticketsForSale array and checks if the destination on the ticket is the destination that is being called in the sellTicket(Destination destination)

method. It then removes that ticket from the array. I used this algorithm to successfully sort through a collection of tickets and remove it so that a passenger can buy it.

2. Bubble Sorting algorithm

```
sort(array) {
    let swapped = false;

    for (let i = 1; i < array.length; i += 1) {
      swapped = false;

        if (this.comparator.lessThan(array[i + 1], array[i])) {
          const tmp = array[i + 1];
          array[i + 1] = array[i];
          array[i] = tmp;
          swapped = true;
        }
      }
      if (!swapped) {
        return array;
      }
    }
    return array;
  }
```

The algorithm takes in an array to sort, establishes a ‘swapped’ variable as false to begin with, looks at the first two items in the array, and then swaps them if the second is less than the first, before registering the swap as ‘true’ and moving on to the next pair. It then returns the array once all items have been swapped.

I used this algorithm in order to sort an array of numbers and return them from low to high as part of an exercise.

P.10 Example of Pseudocode

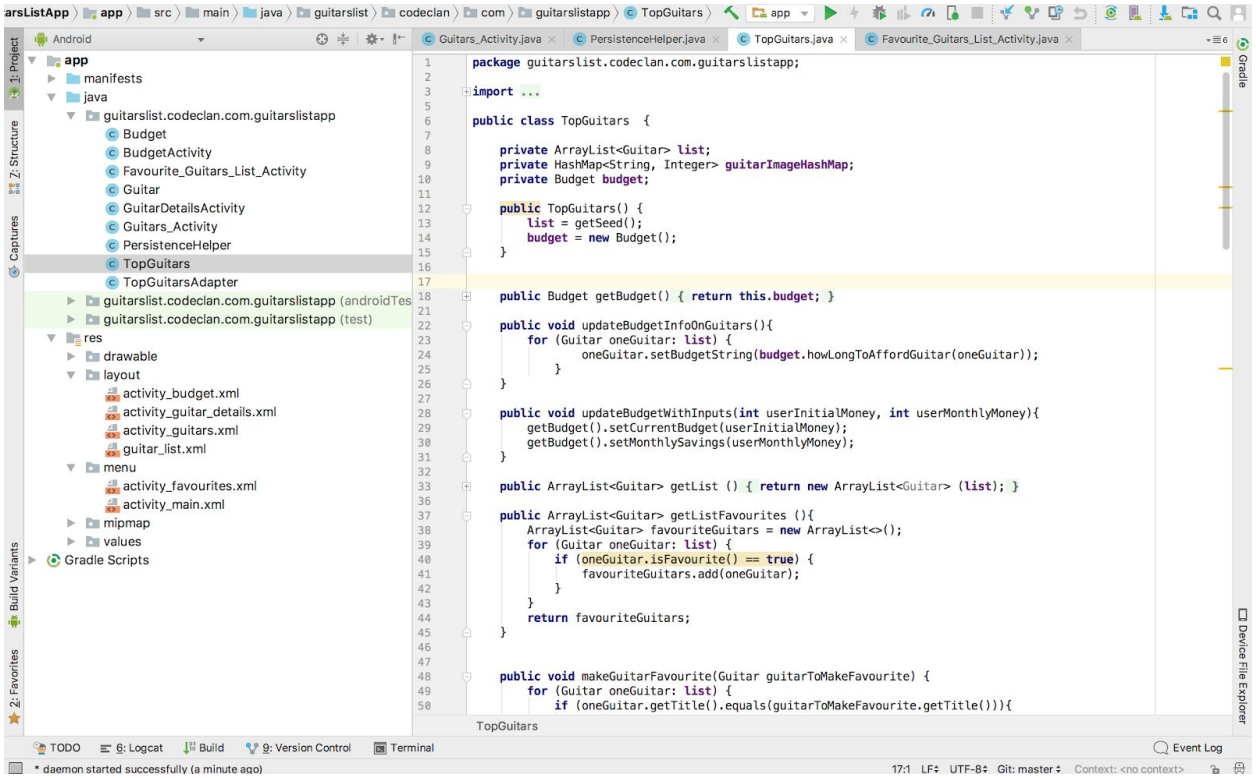
Pseudocode instructions for Ruby function to select data from SQL database

```ruby
def self.all

  #1. Request data from database
  sql = "SELECT * FROM trolls"

  #2. Run Postgres file on data to receive
  hashes
  troll_hashes = SqlRunner.run(sql)

  #3. Change hashes into objects through
  mapping
  troll_objects = troll_hashes.map {|troll|
  Troll.new (troll)  }

  #4. Return objects
  return troll_objects

end
```

P.11 Github link to one of your projects

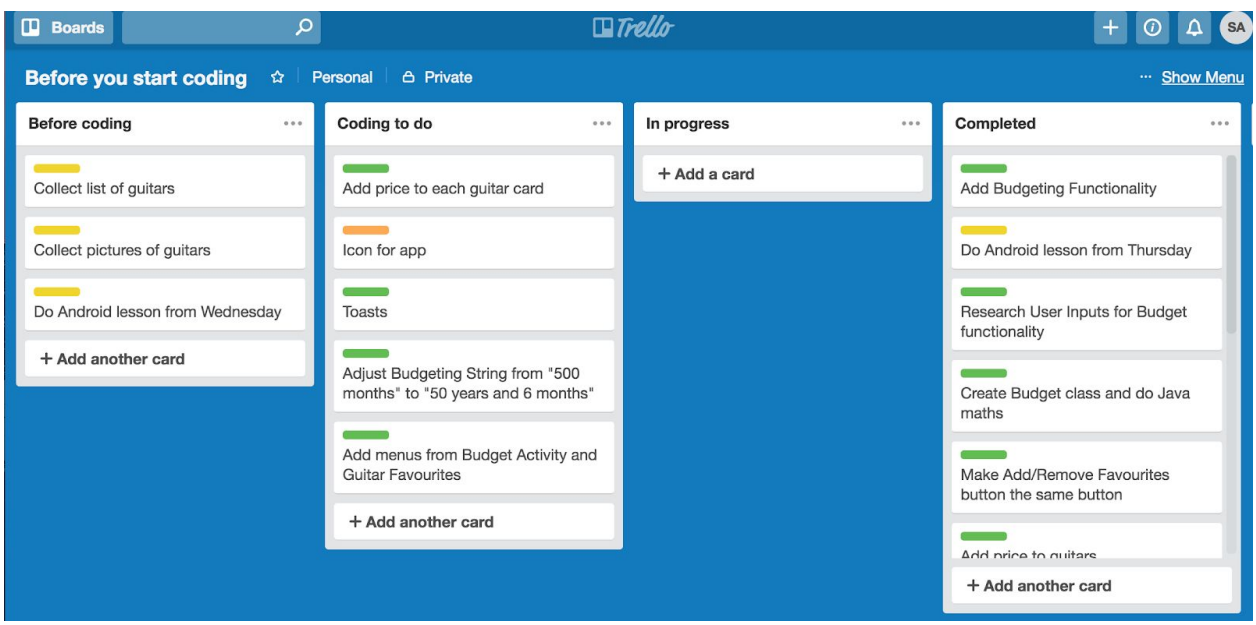
https://github.com/this-is-simon/cc_w8d3_android_TopGuitars

In this project I worked alone on an Android app to display Iconic Guitars.

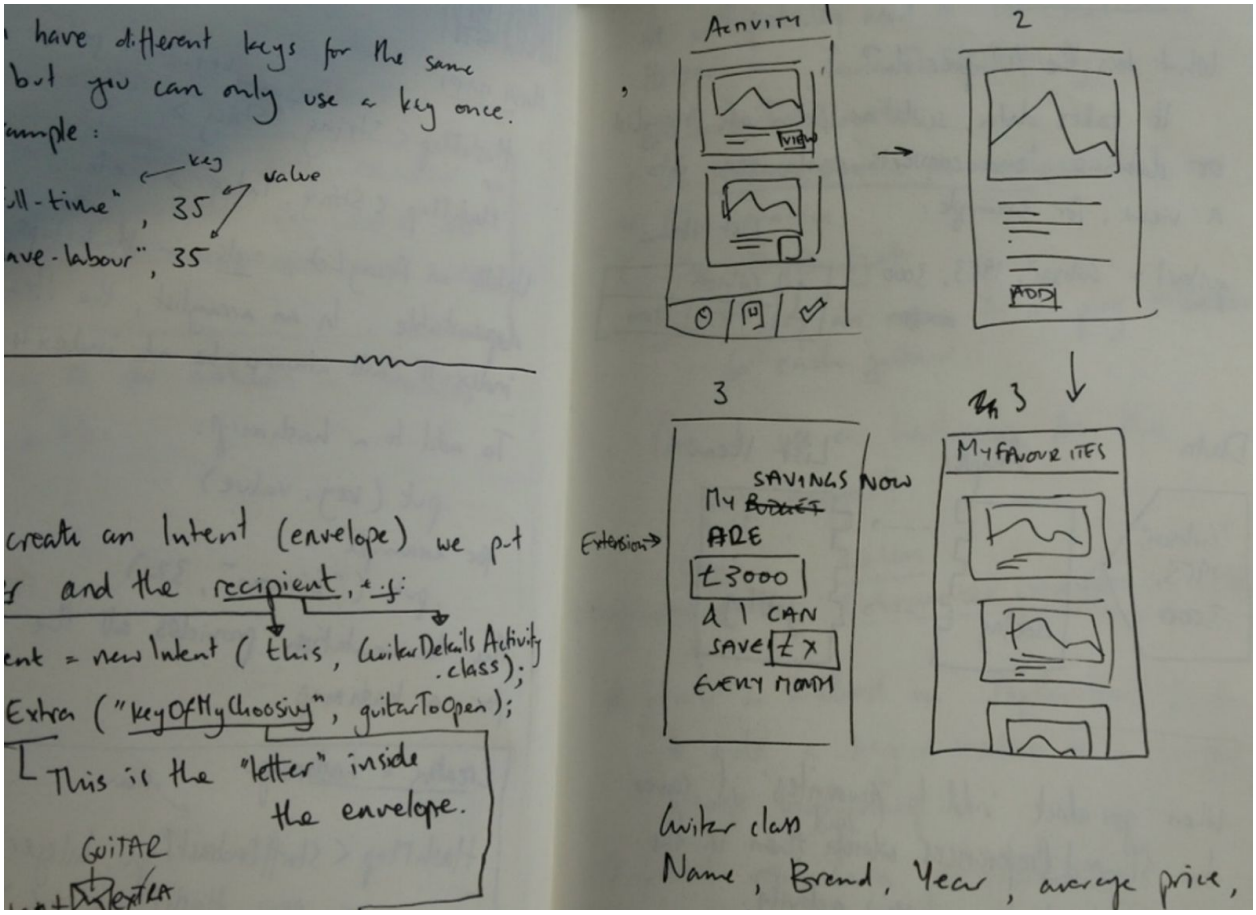


P.12 Screenshot of your planning and different stages of development to show changes

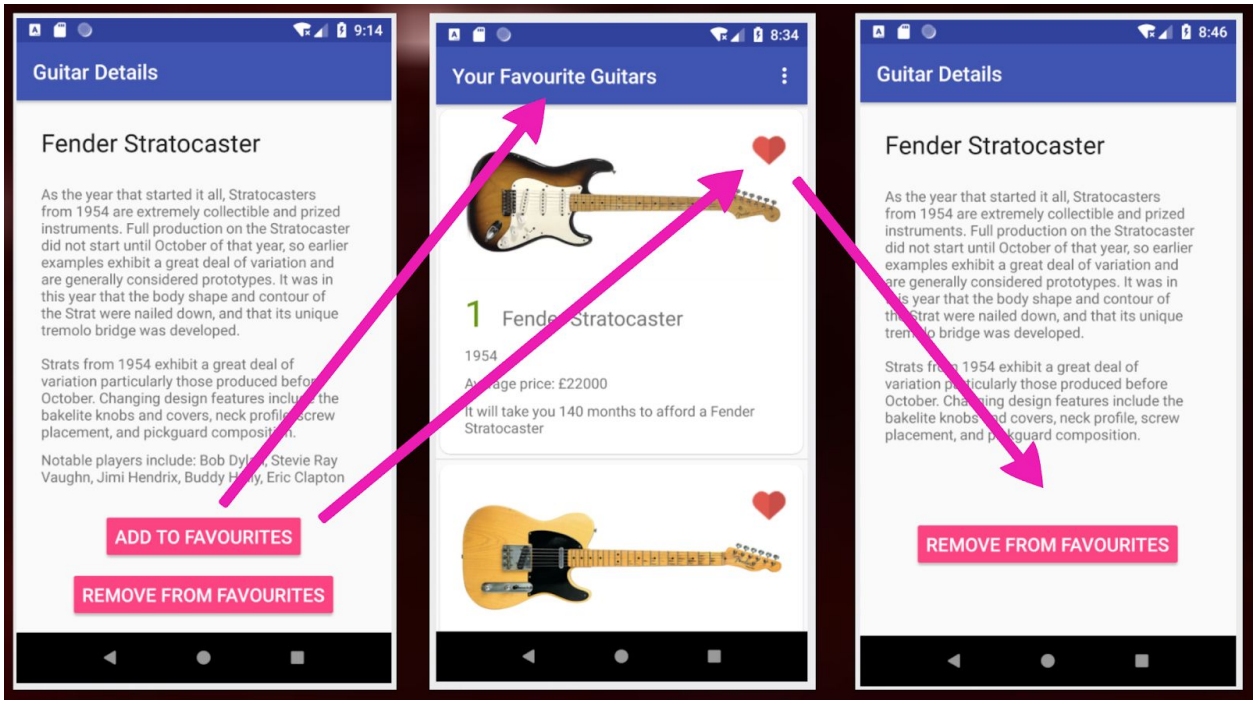
1. Planning project using Trello:



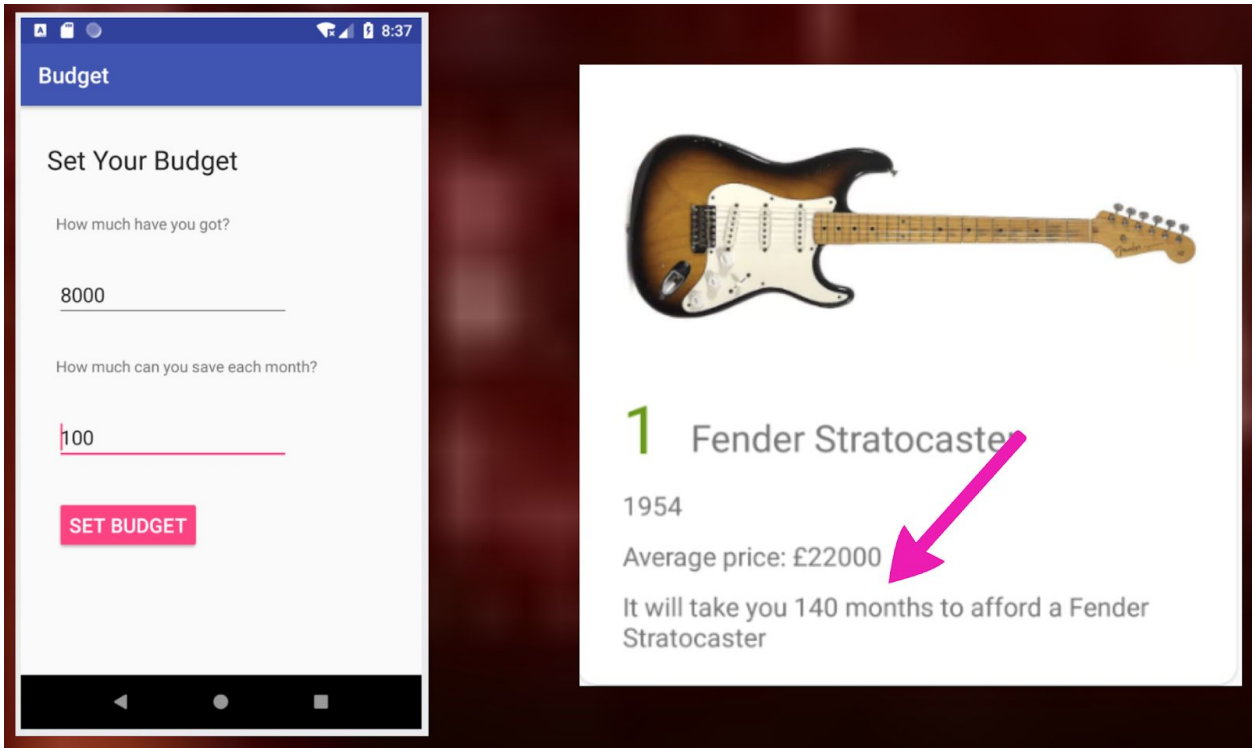
2. Proposed wireframes for 3 app activities plus additional fourth extension



3. Completed app with ‘Favouriting’ function, now to plan ‘Budgeting’ function...

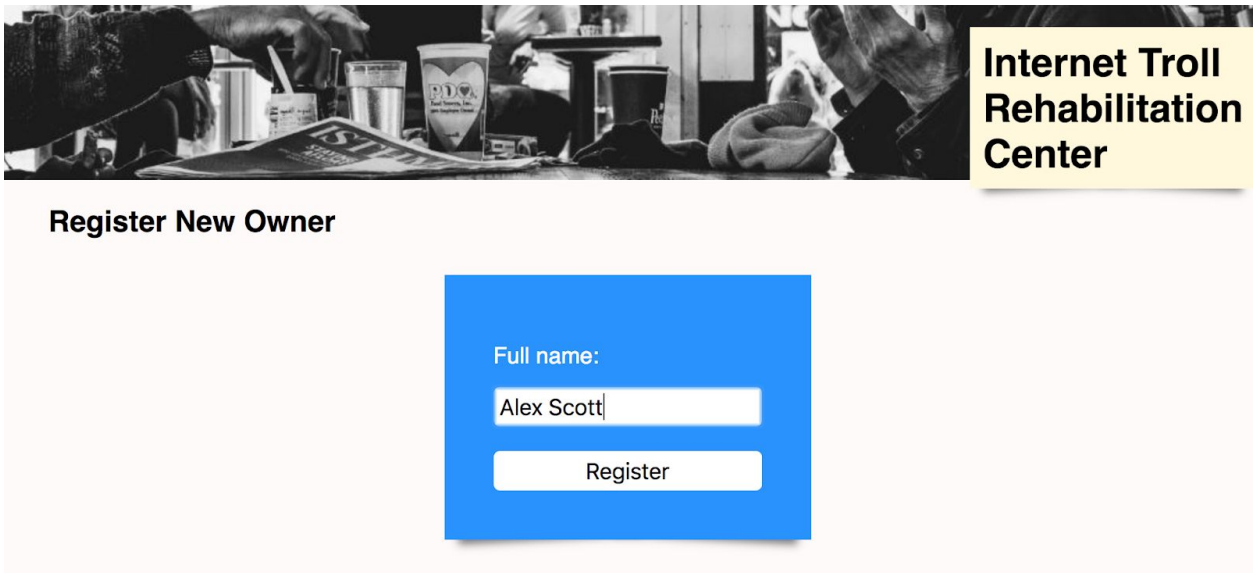


4. Extension completed with ‘Budgeting’ function

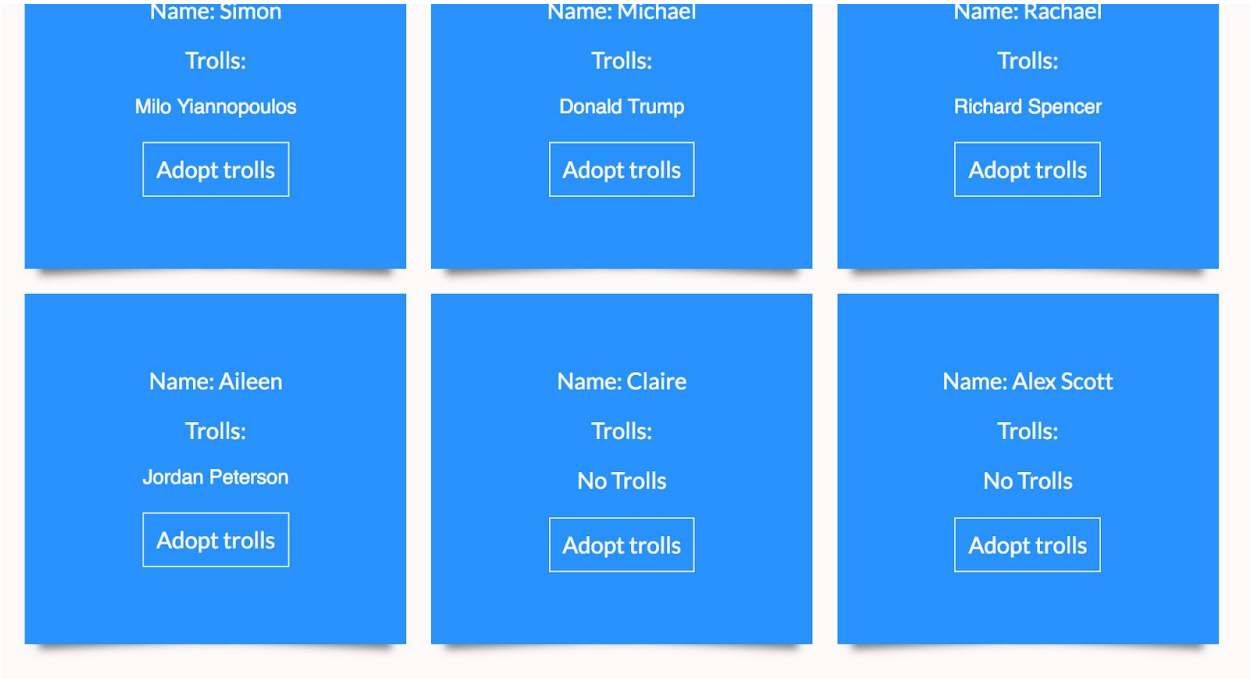


P.13 User Input

User input being added



User input being saved as a new Troll Owner, Alex Scott bottom right

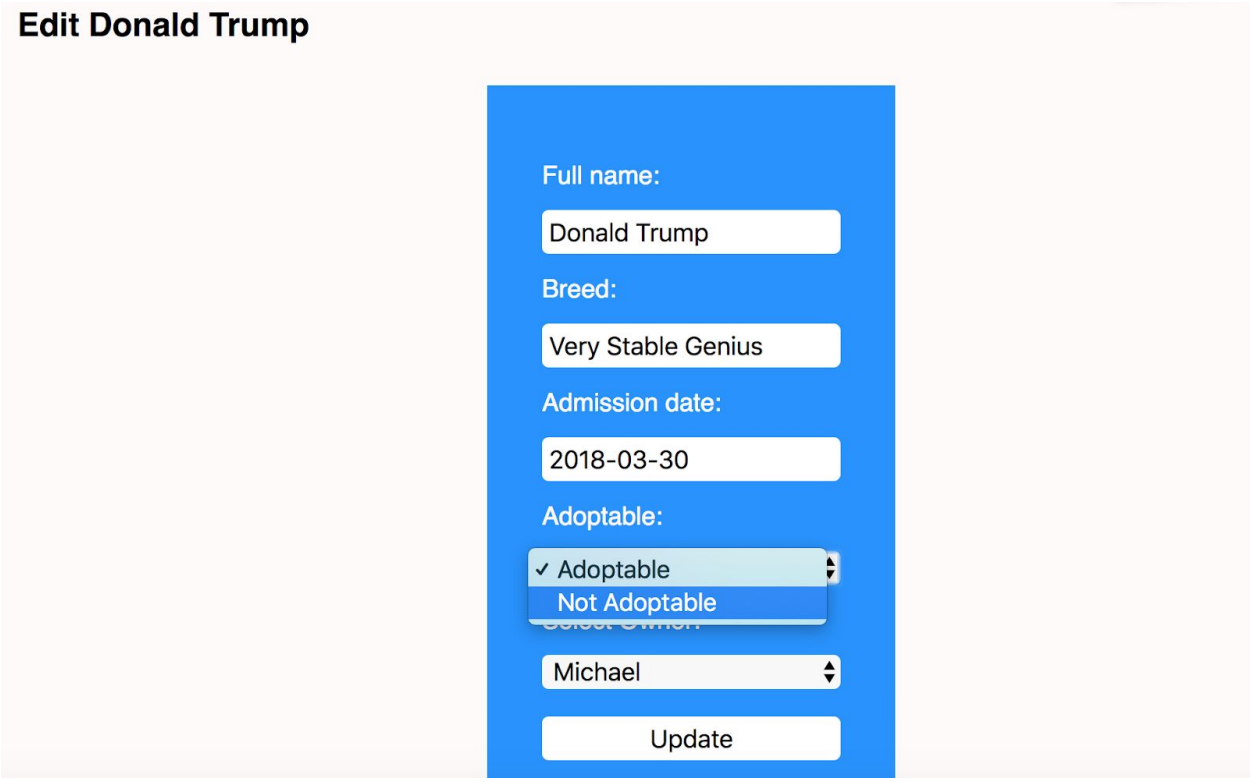


P.14 Interaction with data persistence

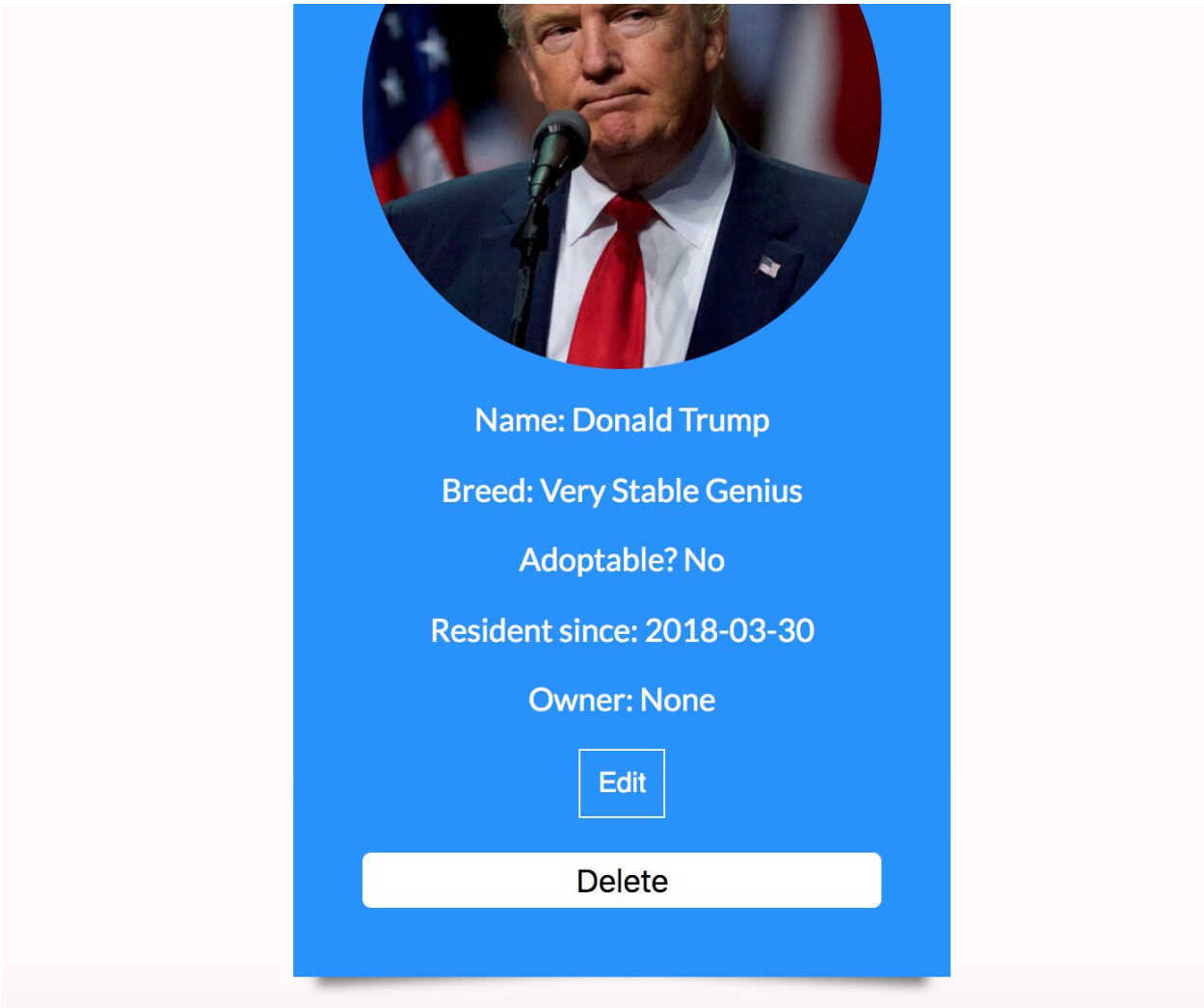
Database before being edited and saved - See Donald Trump, Adoptable 'Yes'

1	Richard Spencer	Neo Nazi	Yes	2018-03-15	3
2	Jordan Peterson	Alt-Right Misogynist	Yes	2018-01-14	4
3	Trevor Noah	Angry Snowflake	Yes	2017-10-21	NULL
4	Milo Yiannopoulos	Free Speech Scumbag	Yes	2017-03-10	1
5	Donald Trump	Very Stable Genius	Yes	2018-03-30	NULL
6	Alex Jones	Flat Earther	No	2016-07-18	NULL
7	Katie Hopkins	Career Troll	No	2016-07-18	NULL

User view of data being inputted and edited for internet troll Donald Trump



Confirmation of data having been saved on app (see Adoptable - No)



Confirmation of data having been saved in database (see Adoptable - No)

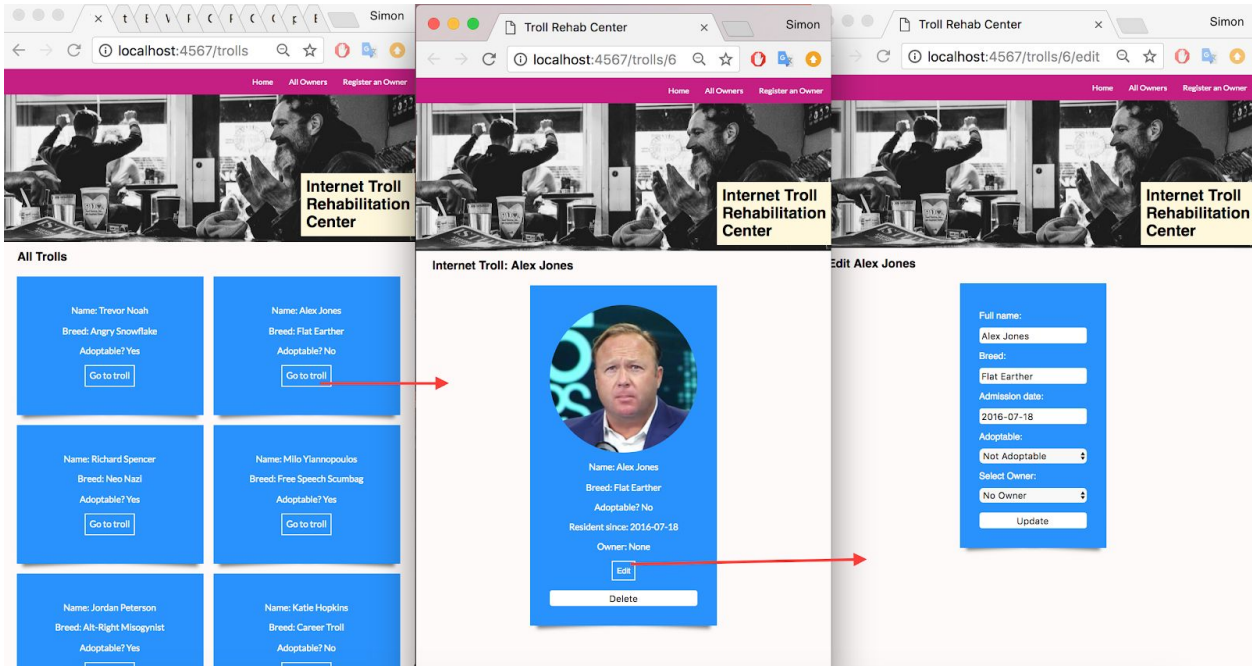
this-is-simon | troll_shelter | trolls | Connected. | PostgreSQL 9.6.3 LOCAL

SQL Query
owners
trolls
pg_catalog
information_schema

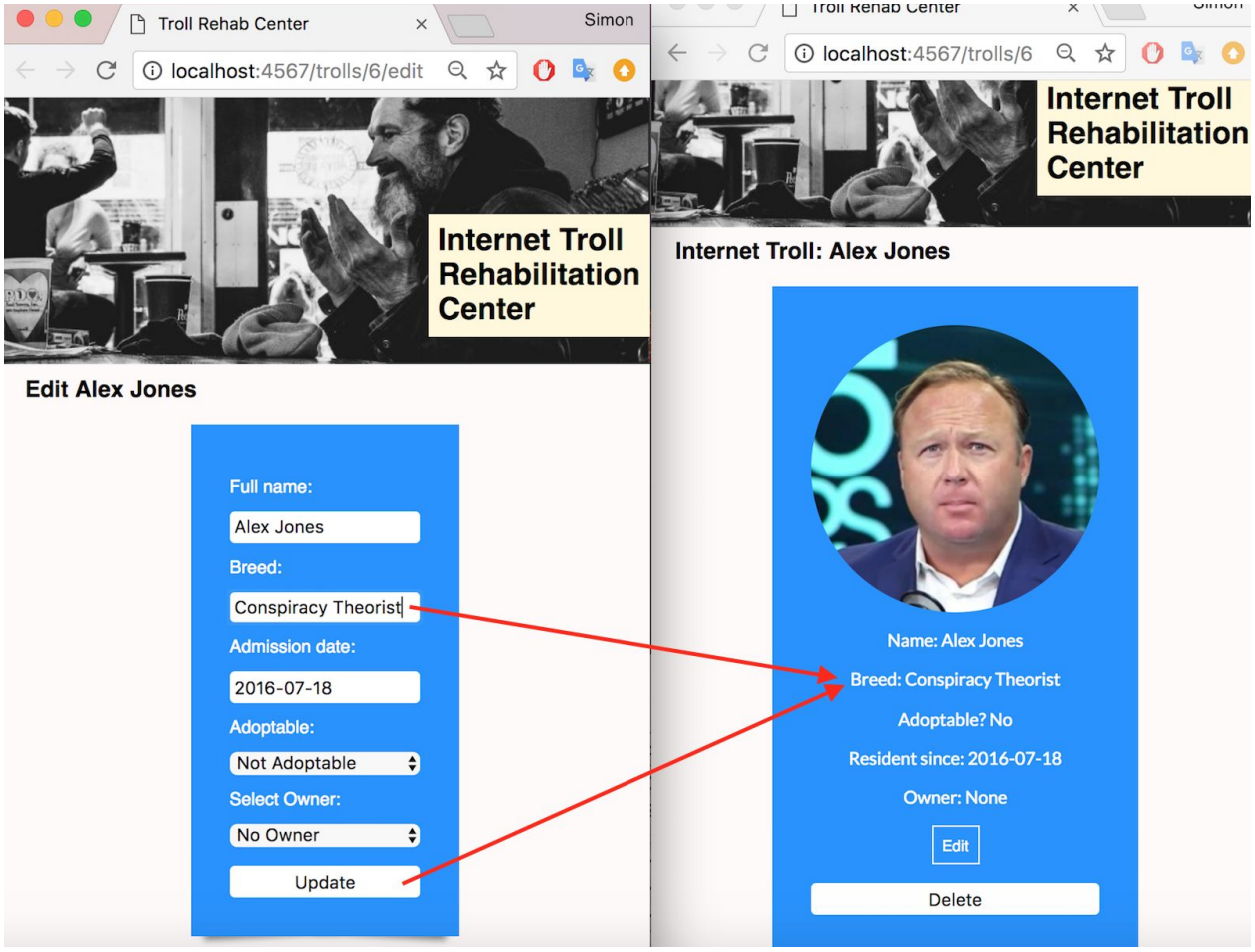
id	name	breed	adoptable	admission_date	owner_id
1	Richard Spencer	Neo Nazi	Yes	2018-03-15	3
2	Jordan Peterson	Alt-Right Misogynist	Yes	2018-01-14	4
3	Trevor Noah	Angry Snowflake	Yes	2017-10-21	NULL
4	Milo Yiannopoulos	Free Speech Scumbag	Yes	2017-03-10	1
5	Donald Trump	Very Stable Genius	No	2018-03-30	NULL
6	Alex Jones	Flat Earther	No	2016-07-18	NULL
7	Katie Hopkins	Career Troll	No	2016-07-18	NULL

P.15 User output result

User requesting action to be performed (updating Alex Jones' breed from 'Flat Earther' to 'Conspiracy Theorist')



User input being processed and displayed in the programme



P.16 Show an API being used in your programme

Show the code that uses the API, and the API being used by the programme while running

Project

- coalculator_group_project
  - .git
  - client
  - node_modules
  - random_resources
  - server
    - db
      - seeds.js
    - routers
      - co2_router.js
      - index_router.js
    - server.js
  - .gitignore
  - package-lock.json
  - package.json
  - readme.md
  - webpack.config.js

index_router.js

```javascript
const express = require('express');
const router = express.Router();
const MongoClient = require('mongodb').MongoClient;
const co2Router = require('./co2_router.js')

MongoClient.connect('mongodb://localhost:27017', (err, client)
  const db = client.db('co2Database');
  const co2Collection = db.collection('co2Collection');
  router.use('/api/co2', co2Router(co2Collection));
});

module.exports = router;
```

seeds.js

```javascript
use co2Database;
db.dropDatabase();

db.co2Collection.insertOne({
  car: '0',
  train: '0',
  plane: '0',
  recycle: '0.6',
  heating: '8.1',
  pets: '0',
  meat: '0'
});
```

co2_router.js

```javascript
const express = require('express');
const router = express.Router();
const ObjectID = require('mongodb').ObjectID;

const co2Router = function(co2Collection) {

  router.get('/', (req, res) => {
    co2Collection
      .find()
      .toArray()
      .then((docs) => res.json(docs));
  });

  router.put('/:id', (req, res) => {
    const id = req.params.id;
    const updatedCo2 = req.body;
    co2Collection
      .updateOne(
        { _id: ObjectID(id) },
        { $set: updatedCo2 }
      )
      .then(() => {
        co2Collection
          .find()
          .toArray()
          .then((docs) => res.json(docs));
      });
  });

  return router;
}

module.exports = co2Router;
```

Project — ~/codeclan/e20/workfiles/week_13/coalculator_group_project

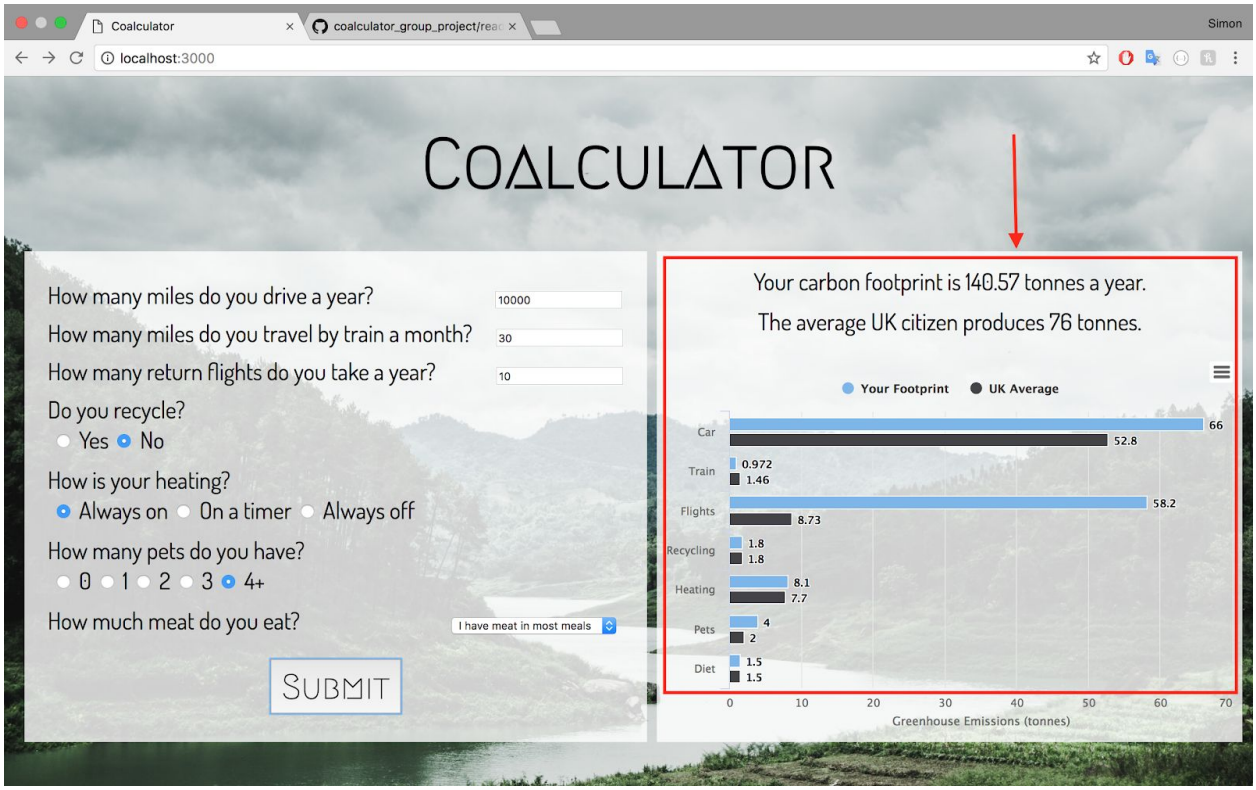
Project

- coalculator_group_project
  - .git
  - client
    - public
    - src
      - helpers
        - pub_sub.js
        - request.js
      - models
        - calculator.js
        - co2.js
      - views
        - form_view.js
        - graph_view.js
        - results_view.js
      - app.js
  - node_modules
  - random_resources
  - server
    - db
      - seeds.js
    - routers
      - co2_router.js
      - index_router.js
    - server.js
  - .gitignore

co2.js | calculator.js

```javascript
const Request = require('../helpers/request.js');
const PubSub = require('../helpers/pub_sub.js');

const Co2Data = function(url){
  this.url = url;
  this.id = null;
}

Co2Data.prototype.getData = function () {
  const request = new Request(this.url);
  request.get()
  .then((co2Collection) => {
    PubSub.publish('co2Collection:data-loaded', co2Collection);
    this.id = co2Collection[0]._id;
  })
  .catch(console.error);
};

Co2Data.prototype.formSubmitListener = function () {
  PubSub.subscribe('FormView:updated-data-ready', (evt)=>{
    this.updateData(evt.detail)
  });
};

Co2Data.prototype.updateData = function (evt) {
  const request = new Request(this.url);
  request.put(evt, this.id)
};

module.exports = Co2Data;
```

When user submits form in Coalculator, data is sent to the database and retrieved using an API which populates the graphs.



P.17 Produce a bug tracking report

Bug tracking report			
User must be able to input data	Pass		
Data must populate Form from API	Fail	Router fixed to allow data to pass to form	Pass
User data must update API	Pass		
GraphView must call data from API and populate visuals	Fail	Method being called too early. Fixed by binding functions.	Pass

P.18 Testing your program

Show the test code, the test not passing, and then the text fixed.

AirportTest.java

```java
    }

    @Test
    public void checkForName() { assertEquals( expected: "JFK", airport1.getName()); }

    @Test
    public void checkNumberOfPlanesInFleet() { assertEquals( expected: 3, airport1.getNumberOfPlanesInFleet()); }

    @Test
    public void planeCanLeaveAirport(){
        airport1.planeLeavesAirport(plane1);
        assertEquals( expected: 2, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void planeCanArriveAtAirport(){
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void airportIsFull(){
        airport1.planeArrivesAtAirport(plane5);
        airport1.planeArrivesAtAirport(plane6);
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

}
```

Airport.java

```java
    private String name;
    private ArrayList<Plane> fleet;
    private int maxPlanes;
    private ArrayList<Passenger> tourists;

    public Airport(String name, ArrayList<Plane> fleet, int maxPlanes){
    this.name =name;
    this.fleet = fleet;
    this.maxPlanes = maxPlanes;
    this.tourists = new ArrayList<>();
    }

    public String getName() { return this.name; }

    public int getNumberOfPlanesInFleet() { return fleet.size(); }

    public void planeLeavesAirport(Plane plane) { fleet.remove(plane); }

    public void planeArrivesAtAirport(Plane plane){
        if (fleet.size() < maxPlanes) {
        fleet.add(plane);} else {
            System.out.println("Your airport is full!");
        }
    }

}
```

AirportTest.java

```java
    @Test
    public void planeCanLeaveAirport(){
        airport1.planeLeavesAirport(plane1);
        assertEquals( expected: 2, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void planeCanArriveAtAirport(){
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void airportIsFull(){
        airport1.planeArrivesAtAirport(plane5);
        airport1.planeArrivesAtAirport(plane6);
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

}
```

AirportTest › planeCanArriveAtAirport()

Airport.java

```java
    public Airport(String name, ArrayList<Plane> fleet, int maxPlanes){
    this.name =name;
    this.fleet = fleet;
    this.maxPlanes = maxPlanes;
    this.tourists = new ArrayList<>();
    }

    public String getName() { return this.name; }

    public int getNumberOfPlanesInFleet() { return fleet.size(); }

    public void planeLeavesAirport(Plane plane) { fleet.remove(plane); }

    public void planeArrivesAtAirport(Plane plane){
        if (fleet.size() < maxPlanes) {
        fleet.add(plane);} else {
            System.out.println("Your airport is full!");
        }
    }
```

Airport › planeLeavesAirport()

AirportTest

Tests failed: 1, passed: 4 of 5 tests – 61 ms

AirportTest	61 ms
airportIsFull	32 ms
checkForName	0 ms
planeCanLeaveAirport	0 ms
planeCanArriveAtAirport	29 ms
checkNumberOfPlanesInF	0 ms

```
junit.framework.AssertionFailedError:
Expected :4
Actual   :3
 <Click to see difference>

<1 internal call>
    at junit.framework.Assert.failNotEquals(Assert.java:329) <3 internal calls>
    at junit.framework.TestCase.assertEquals(TestCase.java:409)
    at AirportTest.planeCanArriveAtAirport(AirportTest.java:55) <23 internal calls>

Process finished with exit code 255
```

AirportTest.java

```java
    @Test
    public void planeCanLeaveAirport(){
        airport1.planeLeavesAirport(plane1);
        assertEquals( expected: 2, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void planeCanArriveAtAirport(){
        airport1.planeArrivesAtAirport(plane4);
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

    @Test
    public void airportIsFull(){
        airport1.planeArrivesAtAirport(plane5);
        airport1.planeArrivesAtAirport(plane6);
        assertEquals( expected: 4, airport1.getNumberOfPlanesInFleet());
    }

}
```

AirportTest › planeCanArriveAtAirport()

Airport.java

```java
    public Airport(String name, ArrayList<Plane> fleet, int maxPlanes){
    this.name =name;
    this.fleet = fleet;
    this.maxPlanes = maxPlanes;
    this.tourists = new ArrayList<>();
    }

    public String getName() { return this.name; }

    public int getNumberOfPlanesInFleet() { return fleet.size(); }

    public void planeLeavesAirport(Plane plane) { fleet.remove(plane); }

    public void planeArrivesAtAirport(Plane plane){
        if (fleet.size() < maxPlanes) {
        fleet.add(plane);} else {
            System.out.println("Your airport is full!");
        }
    }
```

Airport › planeLeavesAirport()

AirportTest

Tests passed: 5 of 5 tests – 9 ms

AirportTest	9 ms
airportIsFull	9 ms
checkForName	0 ms
planeCanLeaveAirport	0 ms
planeCanArriveAtAirport	0 ms
checkNumberOfPlanesInF	0 ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_141.jdk/Contents/Home/bin/java ...
objc[819]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMachines/jdk1.8.0_141.jdk/Contents/Home/bin/java (0x10
Your airport is full!

Process finished with exit code 0
```