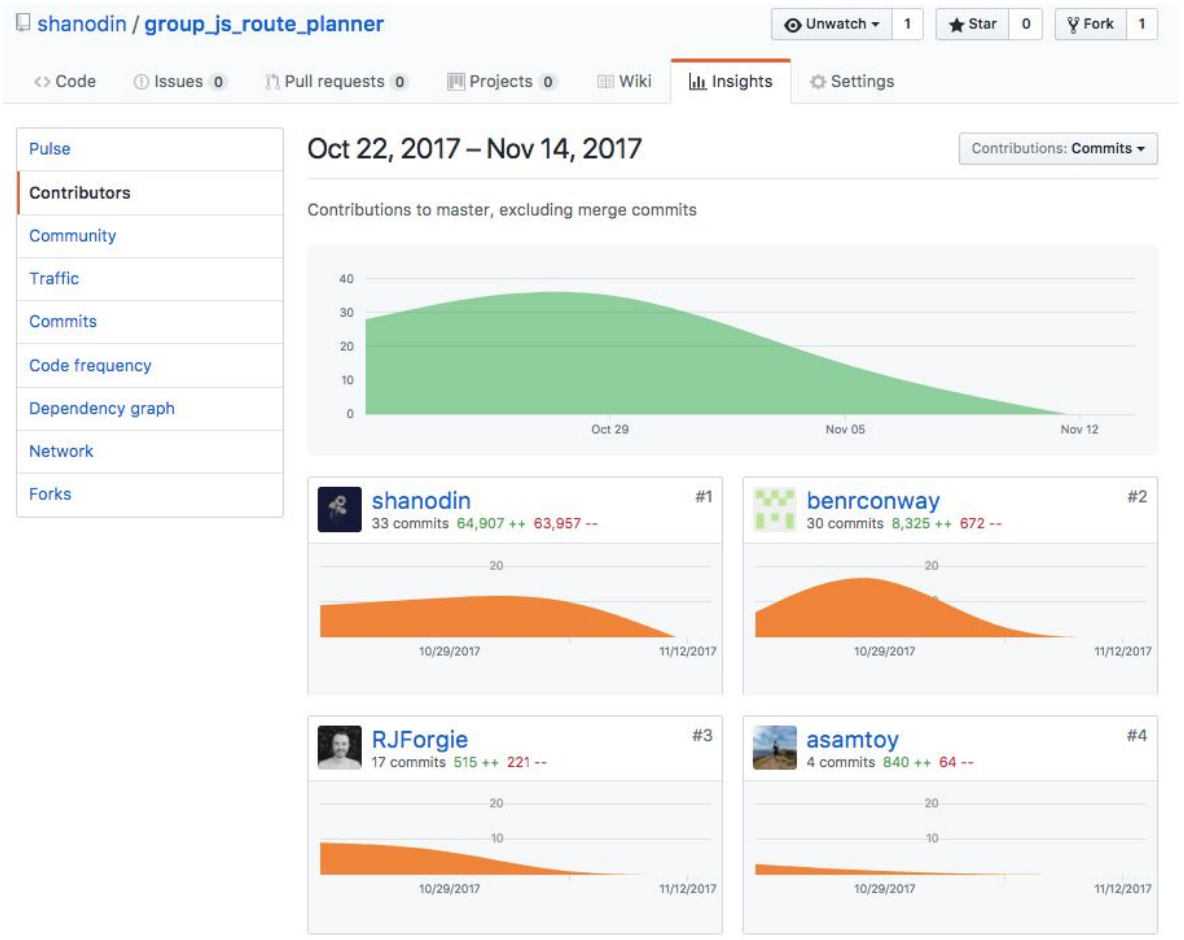
PDA Alice Rees - Project module

P 1 Screenshot of GitHub contributor page for group project



P 2 Project brief from group project

Route Planner

Visit Scotland are look for ways to encourage people to walk and cycle. Your task is to create an app that allows users to search for cycling and hiking routes, view routes on a map, save routes to a wishlist and mark a route done.

You could use GoogleMaps Directions API:

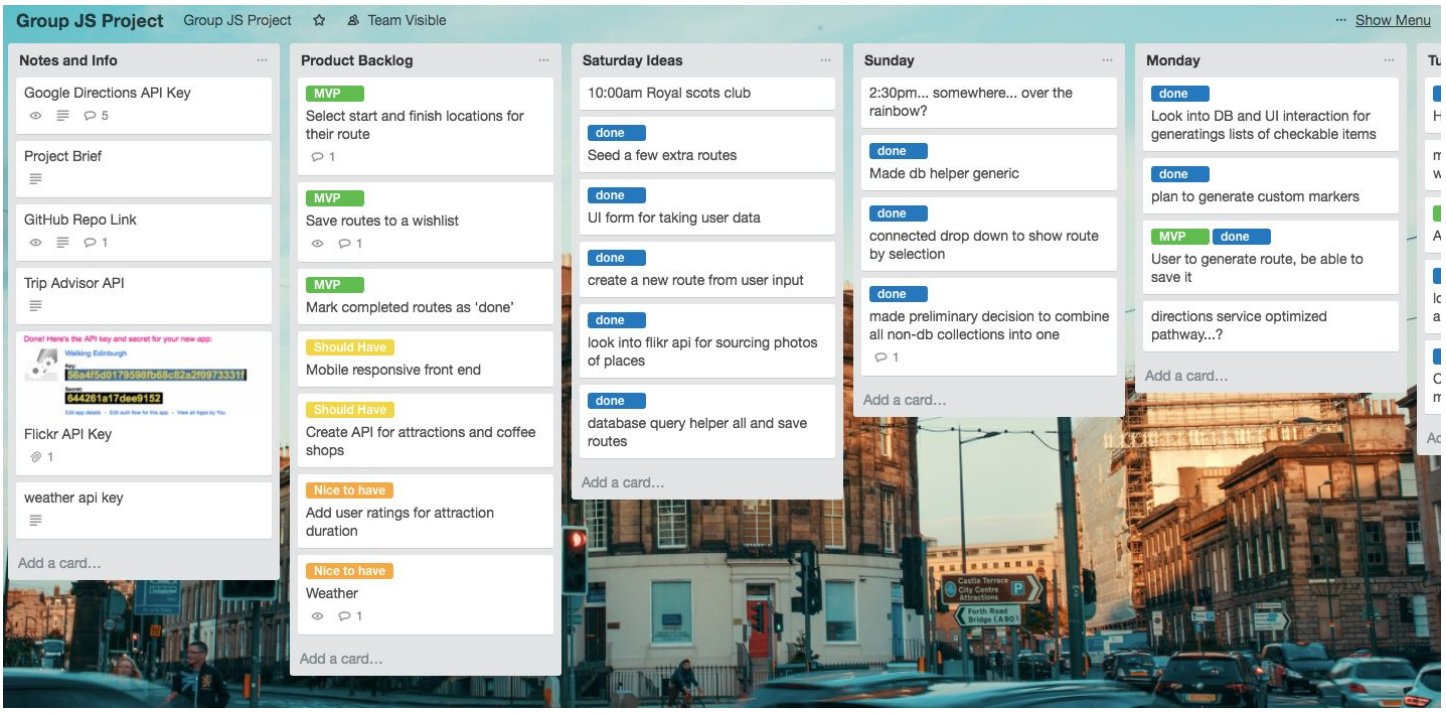
- https://developers.google.com/maps/documentation/directions/

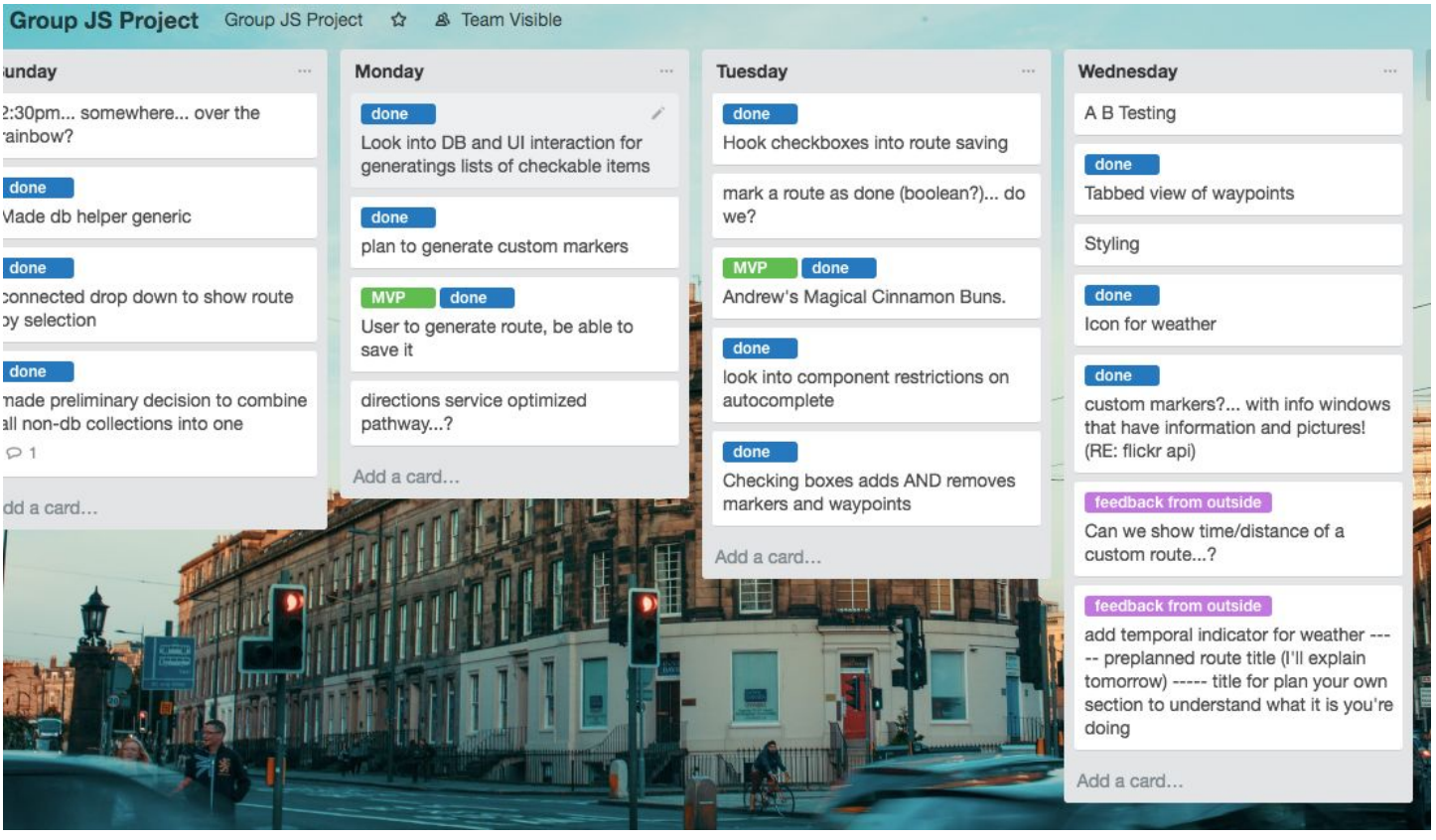
MVP

Users should be able to:

- Select start and finish locations for their route
- Save routes to a wishlist
- Mark completed routes as 'done'

P 3 Group project planning



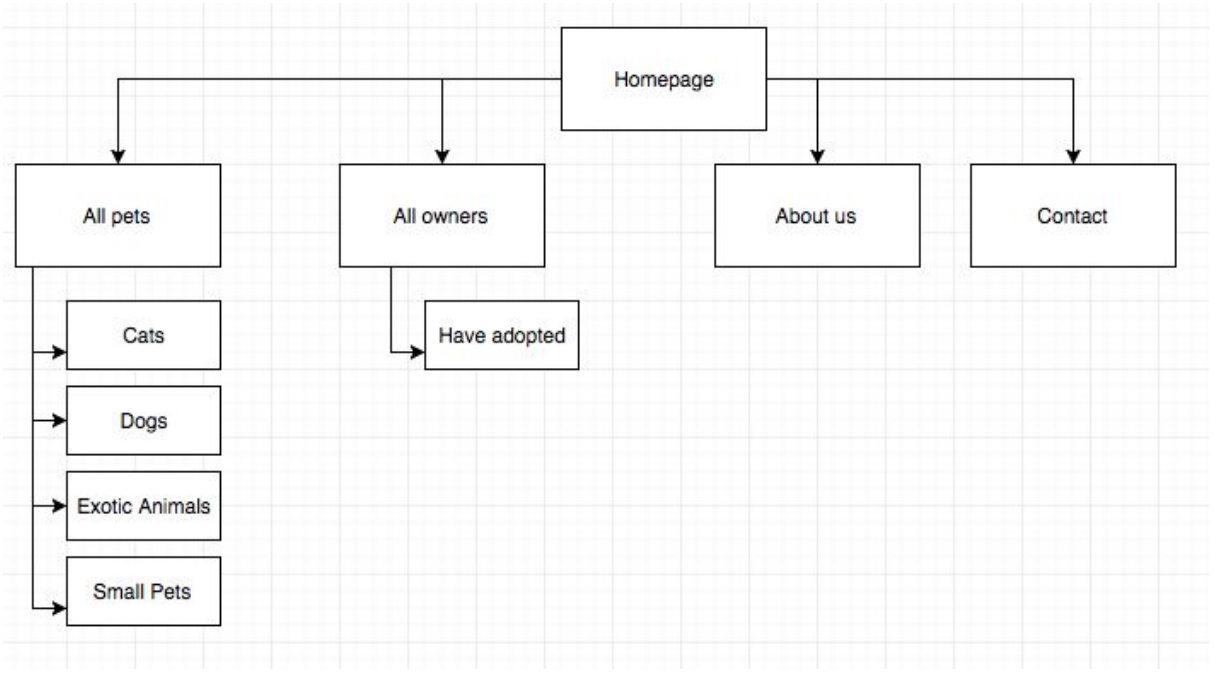


P 4 Acceptance and Criteria test plan

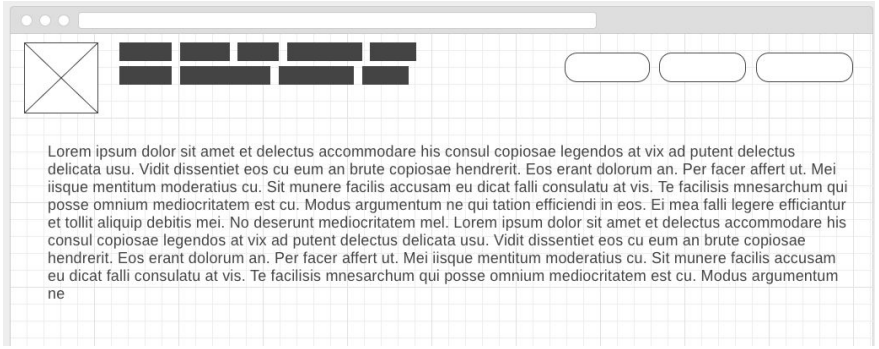
Acceptance Criteria	Expected Outcome/Result	Pass/Fail
A user can view a list of adoptable	List of pets marked as adoptable	Pass

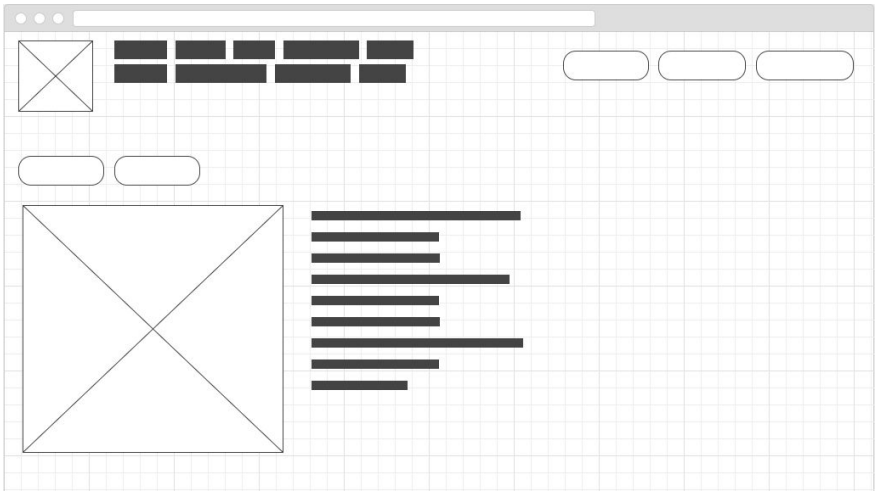
pets	is available when the link is clicked	
The list of adoptable pets can be filtered by type	De-slecting a type removes pets of that type from the list	Pass
The user should be able to add new owners to the database	When a new owner is added, it should be persisted to the database and show up in the list of owners.	Pass
The user should be able to make a pet ‘unadoptable’	When a pet is marked as unadoptable it should no longer appear in the adoptable pets list	Pass

P 5 Sitemap

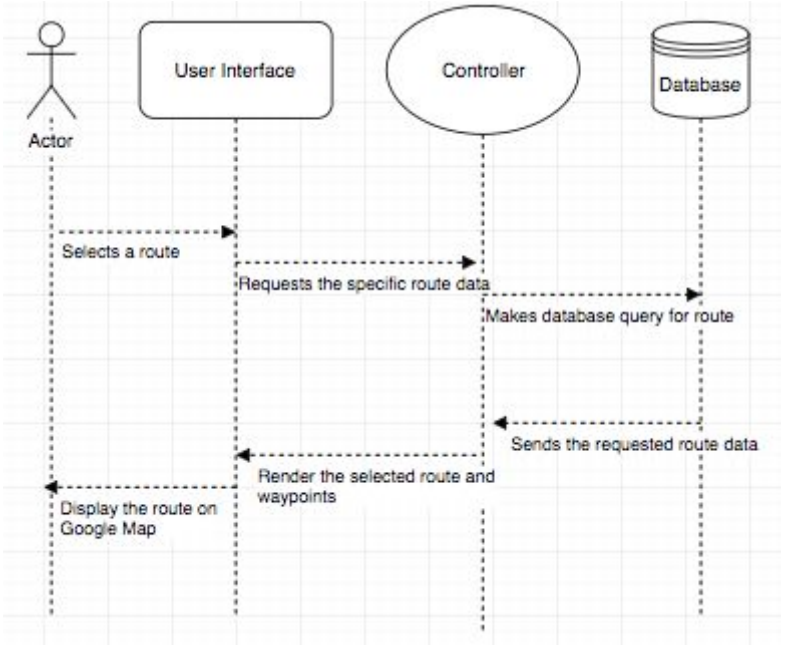


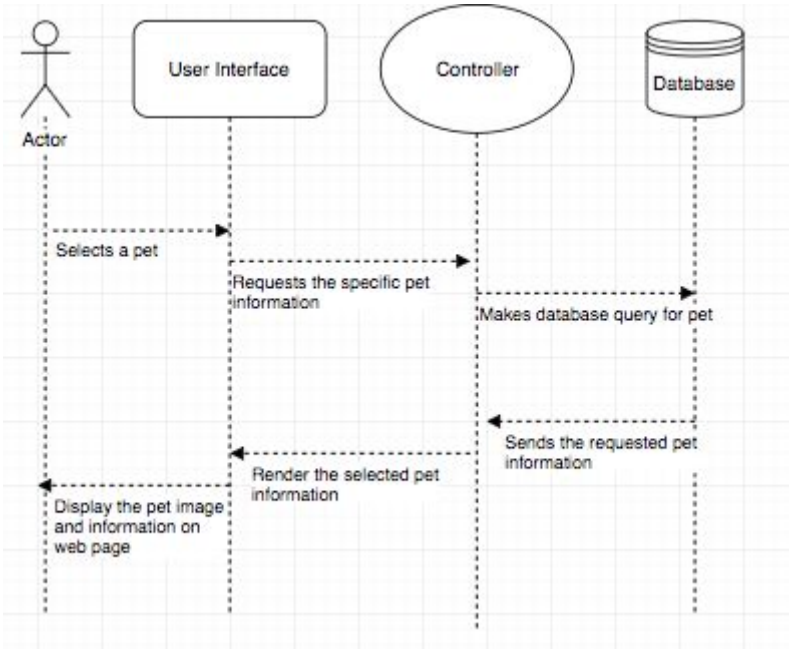
P 6 Two Wireframes



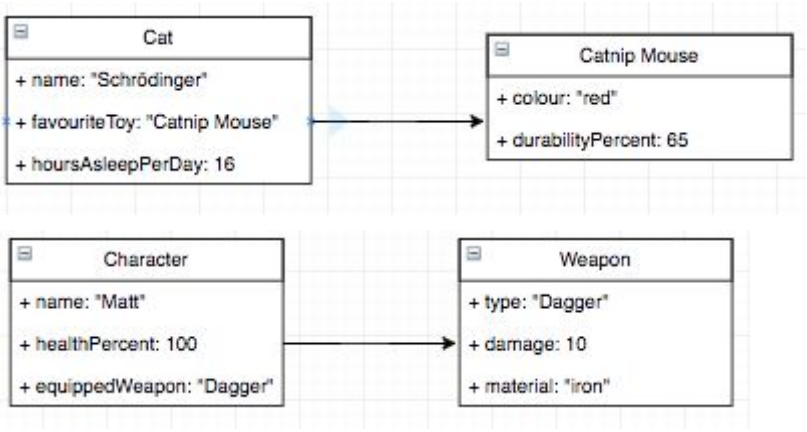


P 7 Two system interaction diagrams (sequence or collaboration)





P 8 Produce two object diagrams



P 9 Screenshots of two algorithms you have written & statement about why

This algorithm checks whether a player's cards are over 21 in total. It also accounts for the rule in blackjack where an Ace can be either 1 or 11 depending on a player's score. This algorithm is required because once a player goes above 21 they are out and no longer participate in the round.

```
public void checkForBust() {
    this.outParticipants = new ArrayList<>();
    HashMap<Playable, Integer> cardValues =  this.cardsCount();
    for (Map.Entry<Playable, Integer> entry : cardValues.entrySet()) {
        if (entry.getValue() > 21) {
            Playable player = entry.getKey();
            int aces = 0;
            for (Card card : player.getCards() ){
                if (card.getFace() .equals(CardFace.ACE)) {
                    aces++;
                }
            }
            int adjustedScore = entry.getValue() - (aces * 10);
            if(adjustedScore > 21) {
                outParticipants.add(entry.getKey());
                int index = this.participants.indexOf(entry.getKey());
                this.participants.remove(index);
            }
        }
    }
}
```

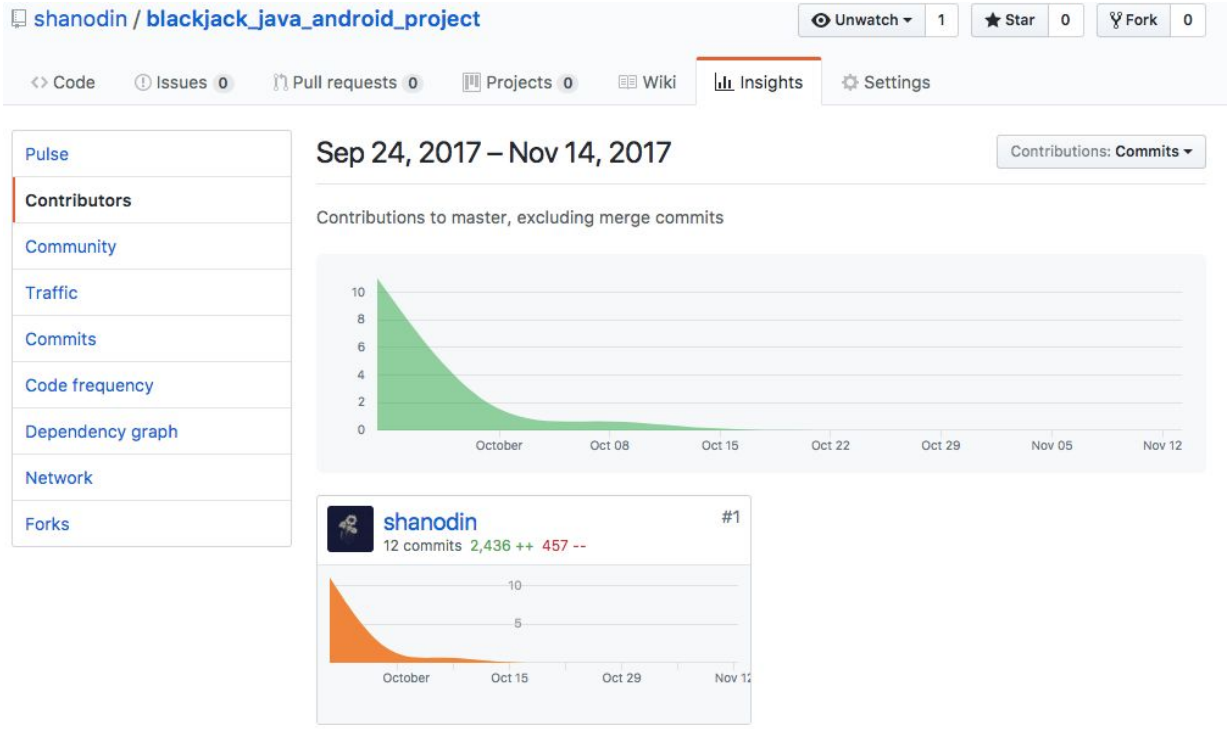
This algorithm checks through players to see if they have 'Blackjack' i.e. a score of exactly 21. It also accounts for players who have a 'true' Blackjack, i.e. a score of 21 with only 2 cards. This is because players with a score of 21 don't receive any more cards, and a player with a 'true' Blackjack automatically wins their round.

```java
public void checkForBlackjack() {
    this.blackjackParticipants = new ArrayList<>();
    HashMap<Playable, Integer> cardValues =  this.cardsCount();
    for (Map.Entry<Playable, Integer> entry : cardValues.entrySet()) {
        if (entry.getValue() == 21) {
            Playable player = entry.getKey();
            if (player.getCards().size() == 2) {
                player.setBlackjack(true);
            }
            blackjackParticipants.add(entry.getKey());
            int index = this.participants.indexOf(entry.getKey());
            this.participants.remove(index);
        }
    }
}
```

P10 An example of Pseudocode

```
// function to find the players who have won the hand
// find the score of the dealer
// make an empty array list to add the winning players to
// if the dealer has true blackjack, no players added to winners array & return
// if the dealer is out, all players who are not also out get added to winners array
// for each player in the participants array, check their total score
// compare their score to dealer
// if player score is greater than dealer score, add player to winners array
// if the loop completes and there are no winners yet, the dealer is the only winner
```

P11 A project where I worked alone + github link



https://github.com/shanodin/blackjack_java_android_project

P12 Planning screenshots

P13 User input being processed

Table of owners before user adding a new owner:

Name	Phone Number	Details
Matthew Addison	01234567890	Details
Alice Rees	01987654321	Details
Jonny Lockhart	03966484659	Details
Seamus Macleod	0482047526	Details

User inputting something:

User's input being stored:

Name	Phone Number	Details
Matthew Addison	01234567890	Details
Alice Rees	01987654321	Details
Jonny Lockhart	03966484659	Details
Seamus Macleod	0482047526	Details
Simon Anger	07462934647	Details

P14 Data persistence

Data being inputted
Please see above example from P13

Confirmation data is saved
Please see above example from P13

P15 Correct output of results and feedback to a user

User requesting information/an action being performed
Please see above example from P13

Request being processed correctly and demonstrated
Please see above example from P13

P 16 Show an API being used within your program

The code that uses the API

```
var init = function() {
  var url = 'https://api.punkapi.com/v2/beers'

  var makeRequest = function(url) {
    var request = new XMLHttpRequest();
    request.open('GET', url);
    request.addEventListener('load', function() {
      var beers = JSON.parse(this.responseText);
      render(beers);
    });
    request.send();
  }

  var addBeersToList = function(beers) {
    var dropdown = document.querySelector('#select');
    beers.forEach(function(beer, index) {
      var option = document.createElement("option");
      option.innerText = beer.name;
      option.value = index;
      select.appendChild(option);
    })
  }

  var render = function(beers) {
    addBeersToList(beers);
    var beerDropDown = document.querySelector('#select');

    var handleBeerSelect = function() {
      var selectedBeer = beers[this.value];
      populate(selectedBeer);
    }
    beerDropDown.addEventListener('change', handleBeerSelect);
  }
```

```
  var populate = function(beer) {
    var name = document.querySelector('#name');
    var selectedBeerName = beer.name;
    name.innerText = selectedBeerName;

    var tagline = document.querySelector('#tagline');
    var selectedBeerTagline = beer.tagline;
    tagline.innerText = selectedBeerTagline;

    var abv = document.querySelector('#abv');
    var selectedBeerAbv = beer.abv;
    abv.innerText = selectedBeerAbv + "% ABV";

    var image = document.querySelector('#beer-pic');
    image.src = beer.image_url;
  }

  makeRequest(url);
}

window.addEventListener('load', init)
```

The API being used while the program is running

Brewdog Beers

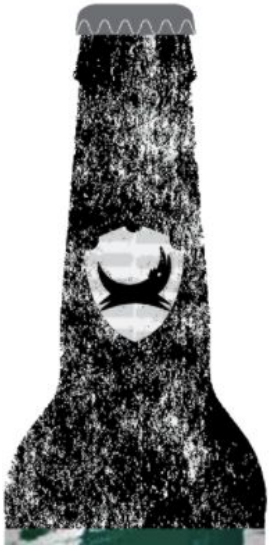
Fake Lager

Beer Info

Fake Lager

Bohemian Pilsner.

4.7% ABV



P 17 Produce a bug tracking report

User can choose from the pre-populated routes	Fail	Create link between dropdownlist and routes database	Pass
User plan their own route within Edinburgh	Fail	Use Latitude/Longitude bounding to restrict to Edinburgh	Pass
Route start point and endpoint autocomplete when within Edinburgh	Fail	Use Google Places API	
User can save their route	Fail	Write a ‘create’ action for routes database	Fail
Add save route button and link to creation action	Pass		
Selecting an attraction displays a waypoint icon depending on category	Fail	Add a switch statement to determine attraction category before generating waypoint	Pass

P 18 Demonstrate testing

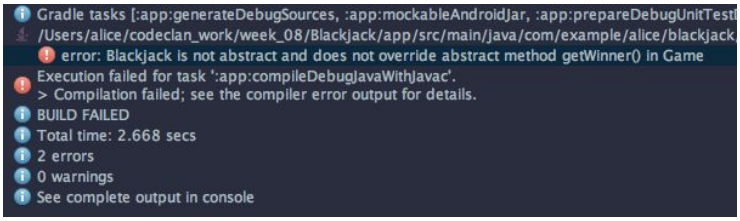
Test code

```java
@Test
public void testGetDealerWinner(){
    dealer.addCards(
            new Card(CardSuit.DIAMONDS, CardFace.KING),
            new Card(CardSuit.DIAMONDS, CardFace.EIGHT));
    player1.addCards(
            new Card(CardSuit.CLUBS, CardFace.FIVE),
            new Card(CardSuit.HEARTS, CardFace.TEN),
            new Card(CardSuit.CLUBS, CardFace.TWO) );
    blackjack.checkForBlackjackOrBust();
    assertEquals(dealer, blackjack.getWinner().get(0));
}
```

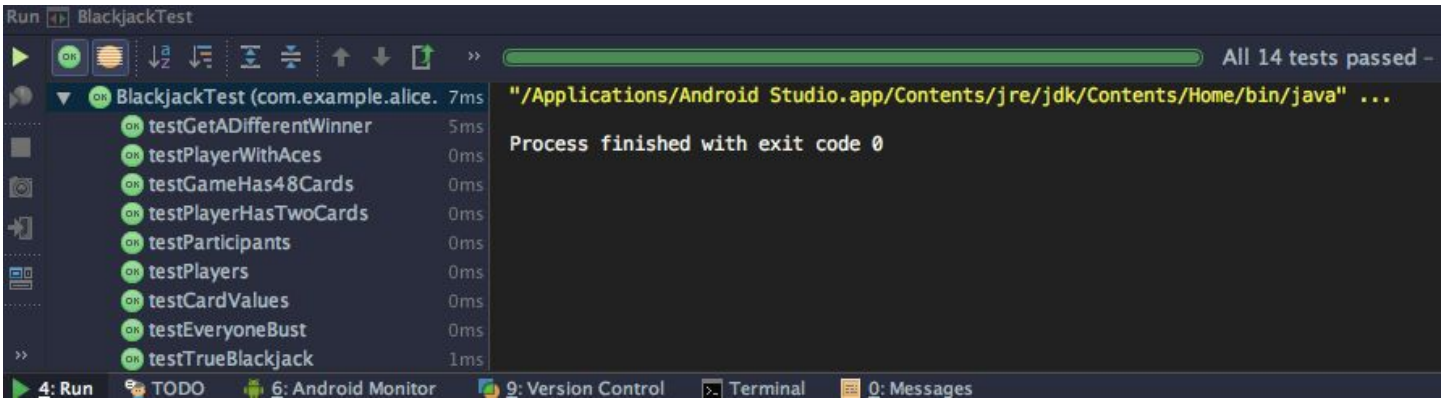
```java
@Test
public void testGetWinner() {
    dealer.addCards(
            new Card(CardSuit.HEARTS, CardFace.KING),
            new Card(CardSuit.DIAMONDS, CardFace.KING),
            new Card(CardSuit.DIAMONDS, CardFace.EIGHT));
    player1.addCards(
            new Card(CardSuit.CLUBS, CardFace.ACE),
            new Card(CardSuit.CLUBS, CardFace.KING) );
    player2.addCards(
            new Card(CardSuit.SPADES, CardFace.EIGHT),
            new Card(CardSuit.CLUBS, CardFace.FOUR) );
    player3.addCards(
            new Card(CardSuit.SPADES, CardFace.TWO),
            new Card(CardSuit.DIAMONDS, CardFace.THREE));
    blackjack2.checkForBlackjackOrBust();
    assertEquals(3, blackjack2.getWinner().size());
}
```

```java
@Test
public void testEveryoneBust() {
    dealer.addCards(
            new Card(CardSuit.HEARTS, CardFace.KING),
            new Card(CardSuit.DIAMONDS, CardFace.KING),
            new Card(CardSuit.DIAMONDS, CardFace.EIGHT));
    player1.addCards(
            new Card(CardSuit.CLUBS, CardFace.TEN),
            new Card(CardSuit.HEARTS, CardFace.TEN),
            new Card(CardSuit.CLUBS, CardFace.TWO) );
    player2.addCards(
            new Card(CardSuit.SPADES, CardFace.EIGHT),
            new Card(CardSuit.SPADES, CardFace.EIGHT),
            new Card(CardSuit.CLUBS, CardFace.SEVEN) );
    player3.addCards(
            new Card(CardSuit.SPADES, CardFace.TWO),
            new Card(CardSuit.SPADES, CardFace.EIGHT),
            new Card(CardSuit.SPADES, CardFace.EIGHT),
            new Card(CardSuit.DIAMONDS, CardFace.FOUR));
    blackjack2.checkForBlackjackOrBust();
    assertEquals(0, blackjack2.getWinner().size());
```

Test failing (no ‘getWinner method written)



Test passing



Code which makes the test pass

```
public ArrayList<Playable> getWinner() {
    int dealerScore = this.dealer.checkTotal();
    ArrayList<Playable> winners = new ArrayList<>();
    if (this.dealer.getBlackjack() == true) {
        winners.add(dealer);
        return winners;
    }
    if (outParticipants.contains(dealer)){
        winners.addAll(blackjackParticipants);
        winners.addAll(participants);
        return winners;
    }
    for (Playable player : participants) {
        int playerScore = player.checkTotal();
        if (playerScore > dealerScore) {
            winners.add(player);
        }
        if (winners.size() == 0){
            winners.add(dealer);
        }
        return winners;
    }
    return winners;
}
```