Evidence for Project Unit

Ross Hill
Cohort E18

P1 – Contributors Page

P2 – Project Brief

Flightr

A new start-up Flightr, has spotted a niche in the Flight + Hotel comparison web app market. It realises it can simultaneously compare flights and hotels to offer the best package possible.

It wants a user to be able to reach our web app and input a departure date and return date and then display possible flight + hotel options. Once searched, it would calculate the available options of flights and represent the data analytically to show travel times or perhaps best value options. It will also display hotel options nearby and a total package price for return flights and hotel nights based on the user input.

They would like the MVP built as soon as possible along with any other features you can implement.

MVP

User can search by departure and return dates

User can view available flight and hotel options for their dates

User can sort flight & hotel packages by price

User can add packages to a favourites list

P3 – Project Planning

Flight/Hotel Comparison App | Private Team | Public

Must

- Search with origin, destination, departure/return dates, number of adults/children
- Display results as flight/hotel packages each with total price
- Sort results by total price
- Save package details to a favourites list
- Delete packages from favourites list

Should

- Filter results according to budget
- Sort results by hotel rating
- Display only top 10 results (though all results should be returned for filtering/sorting purposes)
- Autocomplete airport info in origin/destination fields

Could

- Modify packages within favourites
- Multiple result views: List, panel, map
- Background image that changes every 10(?) seconds
- Log a basic user profile to save packages under
- Detailed information on a package - Modal?
- Google map for hotel location
- 'Best deals today' search

Won't (for now)

- Have booking forwarding functionality
- Refresh packages saved to favourites according to updates in the API

P4 – Acceptance criteria/ Test Plan

Acceptance Criteria	Expected Result/Output	Pass/Fail
A user is able to view all animals available for adoption	List of all available pets are displayed when link is clicked	**Pass**
A staff member is able to view all pets assigned to owners	Pets are sorted into list with 'available to adopt' status on performing search	**Pass**
A staff member is able to assign an animal to a new owner	When staff member inputs a new owner for an animal this change is reflected on the main page	**Pass**
A staff member is able to remove pets from the database	Upon clicking the remove button, the animal is removed from the displayed list	**Pass**

P5 – User sitemap

P6 – Wireframe designs

P7 – System interaction diagrams

P8 – Object diagrams

P9 – Algorithms

```
sorted_array = [1,3,7,8,9,21,23,45,56]


def linear_search(search_number, array)
  for number in array
    return true if number == search_number
  end
  return false
end

p linear_search(23, sorted_array)
```

This is a linear search algorithm, designed to take in an array of numbers and a search item. The for loop iterates through each item in the array in sequence and returns the Boolean 'true' if the number is found. I have chosen it because it is simple to implement and understand, and effective at searching small amounts of data.

```
def binary_search(search_number, array)

  return false if array.empty?

  middle_index = array.length() / 2
  middle_element = array[middle_index]

  return true if search_number == middle_element

  new_search_area = search_number < middle_element ? array[0..middle_index -1] : array
  return binary_search(search_number, new_search_area)

end

p binary_search(0, sorted_array)
```

This is a binary search algorithm. It takes in a search number and an array and searches for the former within the latter. It compares the search number to the middle element of the array in order to then make its next search on the lower or higher half of the array where it continues this sequence until the number is found. I have chosen it because it is a more effective search with sparse datasets.

P10 – Pseudocode

```
public int spin() {
  # *  shuffle fruits ArrayList containing symbols
  # *  get symbol for reel 1
  # *  shuffle fruits ArrayList again
  # *  get symbol for reel 2
  # * shuffle fruits ArrayList again
  # * get symbol for reel 3

  # * run method to determine if a hold is given
  # * (if yes increment hold count, display message to user)
  # * run method to determine if nudge is given
  # * (if yes increment nudge count, display message to user)
#     return symbols to user (as new ArrayList?)
- IF no matching symbols
      # output string "Would you like to END game? HOLD a reel? NUDGE a reel? SPIN again?
case END
    # -terminate program
case NUDGE
*     IF nudge count > 0
# *   "Select reel to nudge"
# *   User input int (1, 2, 3)
# *   Shuffle ArrayList, output new fruit for selected reel
# *   Return symbols to user
* ELSE output string "You have no nudges available" -> return to prompt
case HOLD
* IF hold count > 0
# *   "Select reel to hold"
# *   User input (1, 2, 3)
# *   Shuffle ArrayList, output new fruit for the OTHER two reels
# *   Return symbols to user
* ELSE
#   output string "You have no holds available" -> return to prompt
case SPIN
……..
IF three matching symbols
calculate winnings
display win message to user "Congratulations, your winnings are" + winnings
(re-prompt or terminate game?)
```

P11 – Solo Project

https://github.com/rossghill/Android-FruitMachine

P12 – Planning and Development

playfield = [[reel1 selection], [reel2 selection], [reel3 selection]]

spin()
win IF

reel1[1] == reel2[1] == reel3[1]

OR

reel1[0] == reel2[1] == reel3[2]

OR

reel1[2] == reel2[1] == reel3[0]

win IF

reel1A == reel2B == reel3C

OR

reel1B == reel2B == reel3B

reel1C == reel2B == reel3A

P13 – User input

P14/ P15 – Data persistence/ Results and feedback

Initial state:

Request:

Result:

P16 – API use

```javascript
// ------------------
const app = function() {
  const url = "https://restcountries.eu/rest/v2/all";
  makeRequest(url, requestComplete);

  const mapDiv = document.getElementById('main-map');
  const map = new MapWrapper(mapDiv, {lat:0, lng:0}, 1);
}
// ------------------

const makeRequest = function(url, callback) {
  const request = new XMLHttpRequest();
  request.open("GET", url);
  request.addEventListener("load", callback);
  request.send();
}

const requestComplete = function() {
  if (this.status != 200) return;
  const jsonString = this.responseText;
  const countries = JSON.parse(jsonString);
  populateList(countries);
}

const requestComplete2 = function() {
  if (this.status != 200) return;
  const jsonString = this.responseText;
  const gifsObject = JSON.parse(jsonString);
  const gif = gifsObject.data
  displayGIF(gif);
}
```

```javascript

const getGifInfo = function(countries) {
  const index = document.getElementById("countries-select").value;
  const country = countries[index];
  let gifURL = "https://api.giphy.com/v1/gifs/search?api_key=x51oU2WEGxWhQ4eytBg2WGQ05Dn54sGA&q=" + country.name

  makeRequest(gifURL, requestComplete2);
}

const displayGIF = function(gif) {
    const gifDisplay = document.getElementById("gif-container");
    const gifImg = document.getElementById("gif-img");

    gifImg.src = gif[0].images.fixed_height.url;
    gifDisplay.appendChild(gifImg);
  }
```

P17 – Bug tracking report

User should be able to search for a flight/hotel package			Pass
User can't select a date in the past	Fail	Added validation to check current date/ not allow input prior	Pass
User can't select a negative number of passengers	Fail	Added validation to prevent input below 1 adult and 0 children	Pass
API calls should run in sync instead of waiting for each other to finish	Fail	Refactoring to allow asynchronous API calls	Pass
MapView should display only nearest hotels to airport	Fail	Adjust zoom and radius in maps API call	Pass

P18 – Testing

```
beforeEach(function() {

flightDetails = {"origin": "LAX",
                 "destination": "LON",
                 "departureTime": "2018-03-16T12:50",
                 "arrivalTime": "2018-03-16T17:10"}
flight1 = new Flight(flightDetails);

});

  it('should have an origin', function(){
    const actual = flight1.origin;
    assert.strictEqual(actual, 'LAX');
  });

  it('should have a destination', function(){
    const actual = flight1.destination;
    assert.strictEqual(actual, 'LON');
  });

  it('should have a departure time', function(){
    const actual = flight1.departureTime;
    assert.strictEqual(actual, "2018-03-16T12:50");
  });

  it('should have an arrival time', function(){
    const actual = flight1.arrivalTime;
    assert.strictEqual(actual, "2018-03-16T17:10");
  });
});
```

```
Flight
  ✓ should have an origin
  ✓ should have a destination
  1) should have a departure time
  2) should have an arrival time
```

```
beforeEach(function() {

flightDetails = {"origin": "LAX",
                 "destination": "LON",
                 "departureTime": "2018-03-16 12:50",
                 "arrivalTime": "2018-03-16 17:10"}
flight1 = new Flight(flightDetails);

});

  it('should have an origin', function(){
    const actual = flight1.origin;
    assert.strictEqual(actual, 'LAX');
  });

  it('should have a destination', function(){
    const actual = flight1.destination;
    assert.strictEqual(actual, 'LON');
  });

  it('should have a departure time', function(){
    const actual = flight1.departureTime;
    assert.strictEqual(actual, "2018-03-16 12:50");
  });

  it('should have an arrival time', function(){
    const actual = flight1.arrivalTime;
    assert.strictEqual(actual, "2018-03-16 17:10");
  });
});
```

```
Flight
  ✓ should have an origin
  ✓ should have a destination
  ✓ should have a departure time
  ✓ should have an arrival time
```