

iPhone Assignment Group Effort



Group
Number:
24

Score range (10-7)	The group member made rigorous and regular contributions
Score (7-5)	The group member was mostly present and contributing, with minor lapses
Score (5-3)	The group member was average in terms of contribution, disappeared from time to time
Score (3-0)	This group member made minimal contribution and was disengaged for most of the project.

1st – Use the Self-assessment rubric above to come up with a final score for each of the group members in the team. The values should reflect the relative percentage of contribution.

The scoring should be done by group consensus and should be updated weekly through the period of the assessment. If any student is rating low then the group should put in a plan for the student increase their contribution.

Group Member (Name Student Number)	Score #
Student 1: Ken Wei Ooi s3693288	10
Student 2: Jeremy Kane s3700178	10
Student 3: Ruiyang Liang s3706515	10
Student 4: Xuan Ye s3586968	10

2nd – Discuss this amongst yourselves and rate the efficiency of your group dynamics. Enter a score from 1 to 4.

1 = awful; 2 = average; 3 = most of the times; 4 = always

Group dynamics	Score #
We were in complete sync with each other.	4
We communicated on a regular basis.	4
We had positive disagreements.	4
We were very productive in terms of outcomes.	4
We took initiative.	4

Contents

App Statement	4
Github Repository Link (Master Branch).....	4
1.1 Headline or Slogan:	4
1.2 Main purpose	4
1.3 Selling points:	4
1.4 A simple use case scenario for the app.....	4
1.5 A set of keywords that you expect users to use when searching for your app on the App store.	4
1.6 Identify a main use case scenario	4
Research Other Apps	5
1. Webjet	5
2. Skyscanner - travel deals.....	6
3. Flight Centre: Cheap Flights	7
4. Expedia: Hotel & Flight Deals.....	8
Comparison to other flight app.....	9
Summary of Comparison	9
UI/UX.....	9
REST based API.....	10
Skyscanner Travel APIs.....	10
OpenWeather API	12
Sketches	13
Low fidelity.....	13
Medium Fidelity	14
Workflow:	18
Design Theme	20
Contrast.....	20
Repetition.....	21
Alignment.....	21
Proximity	22
Design Theme (Apple).....	23
Clarity	23
Deference.....	24
Depth	25
Updated Wireframes for Assignment 2	26
Updated workflow	28

High Fidelity (Screenshots from Simulator)	30
Networking Approach Research	31
Networking (Updated REST API)	33
3.1 REST Skyscanner currency.....	33
3.2 REST Skyscanner airportCode	33
App Store Page Design	34
Assignment 2 Update Summary	35
Student Weekly Log	36
Week 2	36
Week 3	37
Week 4	38
Week 5	39
Week 6	40
Week 7	41
Week 8	42
Week 9	43
Week 10	45
Week 11	46

App Statement

Trip Planner

Github Repository Link (Master Branch) —

https://github.com/rmit-S2-2020-iPhone/a1-s3693288_s3706515_s3700178_s3586968.git

API – OpenWeatherAPI, SkyScanner

1.1 Headline or Slogan:

We will provide you the cheapest flight information that you need.

1.2 Main purpose

“Trip Planner” is an app that provides users with the ideal flight price and weather information for the flights

1.3 Selling points:

- “Trip Planner” can help users to get cheap flights
- “Trip Planner” provides users with all the details of the flights
- “Trip Planner” provide 24/7 live update of the flights’ price
- “Trip Planner” will remind the users the flight time
- “Trip Planner” also provides you the weather updates for your trip.

1.4 A simple use case scenario for the app.

Always paying extra for flights during your trip planning or having your trip ruined by bad weather. By using our application, you will be able to check for reasonably priced flights and check for weather conditions beforehand. This app will provide you with flight information and weather information that you need. These Flight details can also be saved for easier view access later.

1.5 A set of keywords that you expect users to use when searching for your app on the App store.

Trip, Planning, Flight, Flight Information, Cheap Flight, Weather Information, Free App

1.6 Identify a main use case scenario

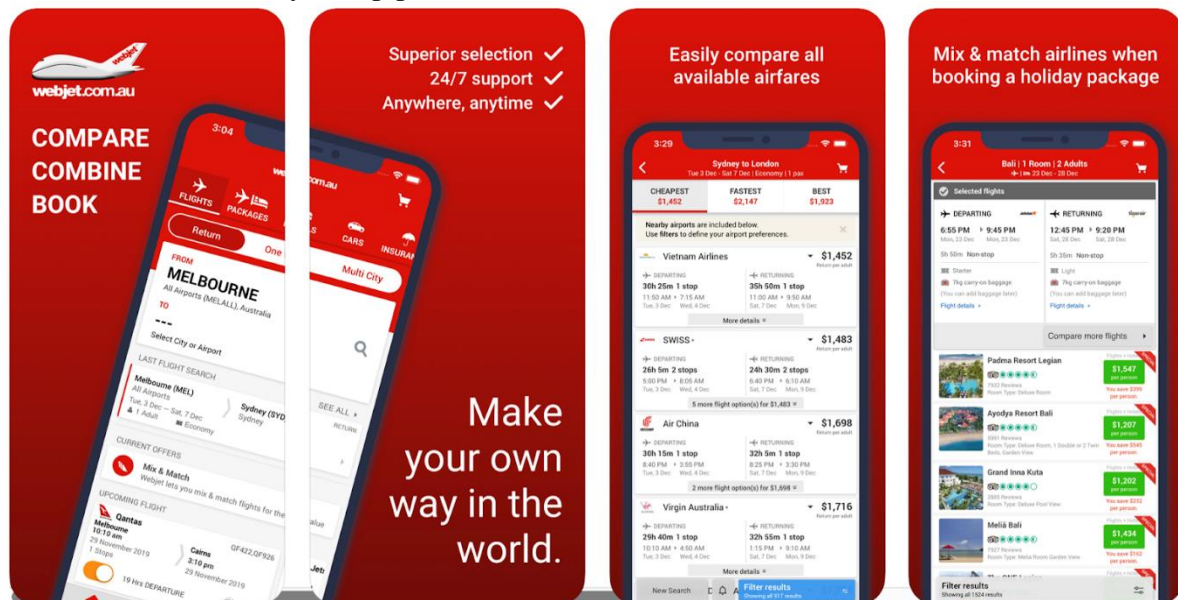
Ryan is planning to book a flight for returning to his home country. He downloads Trip Planner app from App Store. Then, he opens the app and select the departure city, arrival city, departure date and return date. A list of flights will be shown in the next page. Then, he scrolls through the list and find the flight that he is interested in. He then clicks on add to save the flight. Ryan then clicks on plan and able to check all previous saved flights. After finding the details he needs, he closes the app and will be able to check saved flights next time he returns to the app.

Research Other Apps

1. Webjet

URL: <https://apps.apple.com/au/app/webjet/id361122959>

Webjet is Australia & New Zealand's #1 online travel agent, it helps users to book flights and hotels with a reasonably cheap price.



Comparison:

WebJet	Trip Planner
<ul style="list-style-type: none"> Hotel Search Payment System Travel Insurance Flight Packages Rent a Car Price Alert Optional log in/sign up 	<ul style="list-style-type: none"> Flight Info Weather Info Weather and Flight notification No login required Favourite flight and weather details

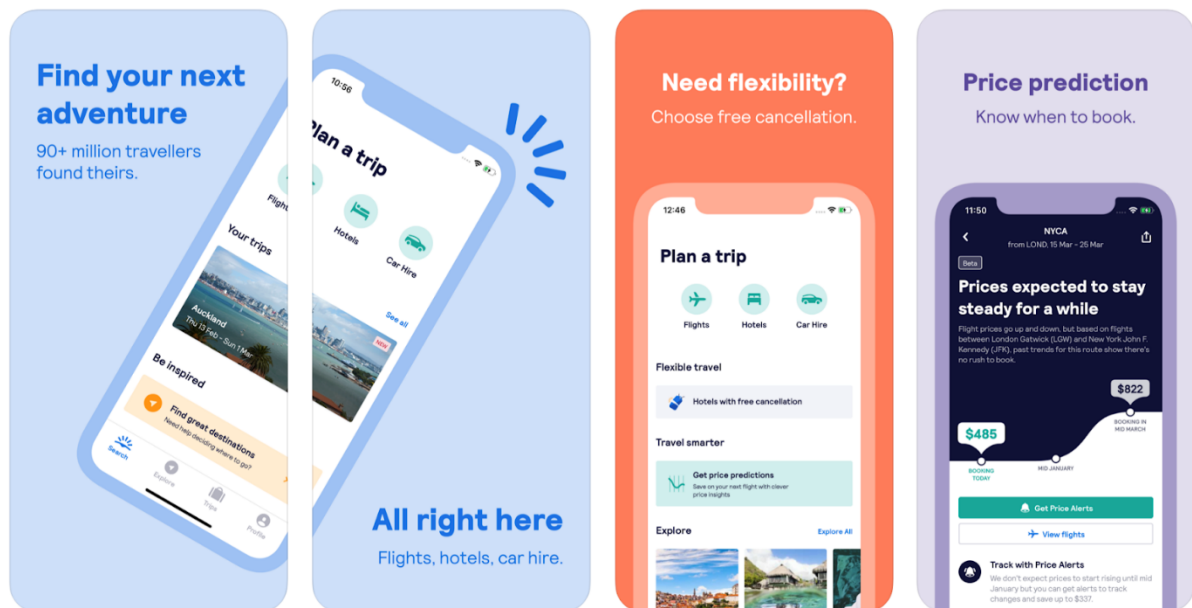
Similarity in features: Flight info search

Difference in features: No login is required in our app and our application allows users to retrieve weather information.

2. Skyscanner - travel deals

URL: <https://apps.apple.com/au/app/skyscanner-travel-deals/id415458524>

Skyscanner is to find and book great deals on flights, hotels, car hire. Your trip starts here.



Comparison:

Skyscanner	Trip Planner
<ul style="list-style-type: none"> • Hotel Search • Payment System • Car Rent • Flight Info • Explore Spot 	<ul style="list-style-type: none"> • No Hotel Search • No Payment System • No Car Rent • Flight Info • No Explore Spot • No Car Rent • Weather Info • Weather & Flight notification

Similarity in features:

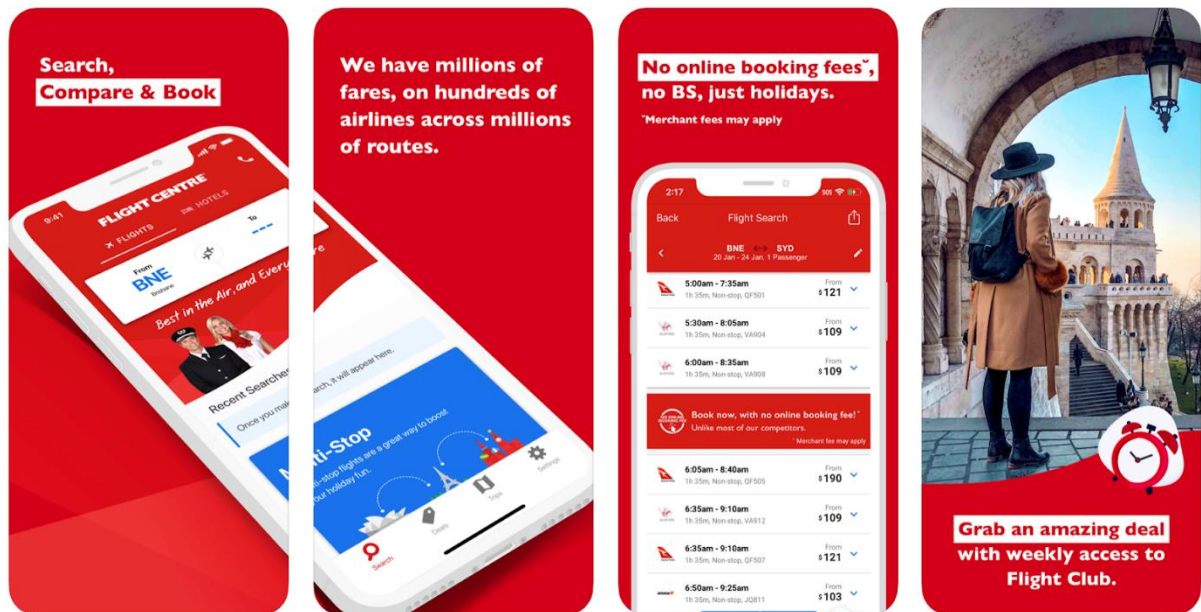
- Flight info

Different features

- For this App it combines hotel and flight offers together,
- Skyscanner has extra function is renting a car
- Skyscanner can explore the spot
- Skyscanner can also book the hotel
- Trip Planner has weather

3. Flight Centre: Cheap Flights

URL: <https://apps.apple.com/au/app/flight-centre-cheap-flights/id1152184991>



Flight Centre's official app for booking flights and researching great travel deals anytime, from wherever you are.

This app requires you to login for the application. And the app itself will save the trips that has been planned in the app.

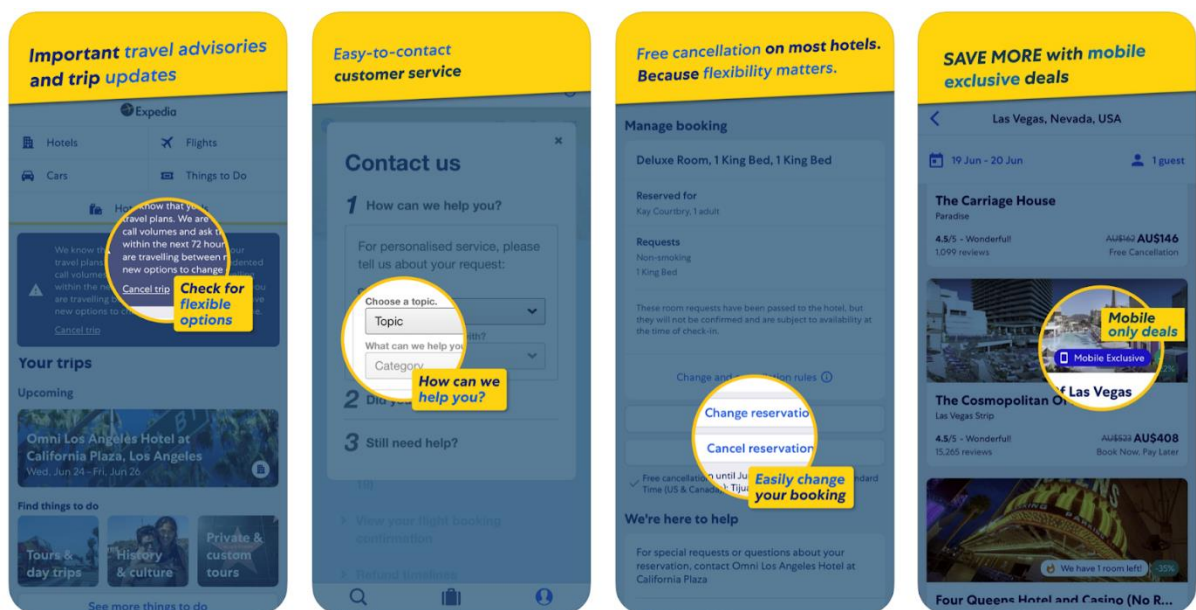
Flight Centre	Trip Planner
<ul style="list-style-type: none">• Payment System• Available Flight List• Accommodation Feature• Check In Feature	<ul style="list-style-type: none">• Flight Info• Weather Info• Weather and Flight notification• No login required• Favourite flight and weather details

Similarity in features: Available Flight info and saving trips.

Difference in features: No login is required in our app and our application allows users to retrieve weather information.

4. Expedia: Hotel & Flight Deals

URL: <https://apps.apple.com/au/app/expedia-hotel-flight-deals/id427916203>



The new Expedia App is your all-in-one travel companion. Save big on hotels, find the perfect flight, discover things to do, and get helpful trip reminders right when you need them. Plus, you will earn double Expedia Rewards points every time you book through the app.

Expedia	Trip Planner
<ul style="list-style-type: none"> Hotel Search Payment System Flight info Flight booking 	<ul style="list-style-type: none"> No Hotel Search No Payment System Flight Info Weather Info Weather & Flight notification

Similarity in features: Available Flight info and saving trips.

Difference in features: No login is required in our app and our application allows users to retrieve weather information, hotel search is not provided

Comparison to other flight app

	Other flight app	Our Trip Planner
Require registration	Yes	No
Easy initial set up	No	Yes
Lowest price search	No	Yes
Clean user interface	Yes	Yes
Accurate location	Yes	Yes
Weather reminder	No	Yes
Membership club	Yes	No
Hotel, Car rental booking	Yes	No

Summary of Comparison

Compared to several current flight booking Apps, we found that the common features of them are not easy for registration and much information shown on the app for the trip. From the users' standpoint, it is free for all users and there is no membership club to distinguish the normal and premium user, apart from providing different services and prices depending on how much the users had spent.

Inspired of other application, our Trip Planner will provide a clean interface that just show the search bar of the flight date and weather to the user, which reminds the user's what to wear and weather is good to trip that is to minimise the irrelevant advertisement and information such as membership club and latest recommended cheapest flight tickets to other places. After simple registration or just logging as a guest, the users can easily search the lowest flight tickets in ascending order compared to many flights company as the date and time they want.

For registration, the users do not need to register for login but can check their booking history and upcoming flights. As well as the reminder, it will remind the user the upcoming flights before the day of their trip, if any changes of the flights boarding in advance or delay will call attention to the user as well.

The app provides the cheapest flight tickets and clean design interface that is easy to discover.

UI/UX

The graphic design of the user interface is clean and chic to show the user the basic function to satisfy what they want, which is to increase the user's experience with fast and easy booking on their journey.

REST based API

Skyscanner Travel APIs

The SkyScanner Travel API will provide all available flight data available, from browsing the cheapest quotes, cheapest destinations (routes), and find the date with the lowest price in the period of 1 month or up to 12 months period. And all this data is returned from live prices from all the suppliers for the requested itinerary.

URL: <https://skyscanner.github.io/slate/>

Sample Get Request:

```
import Foundation

let headers = [
    "x-rapidapi-host": "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
    "x-rapidapi-key": "2b44f45586msh6138dce95c46efcp170543jsna3ba9defc8d2"
]

let request = NSMutableURLRequest(url: NSURL(string:
    "https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/browseroutes/v1
    .0/AU/AUD/en-us/MEL-sky/SYD-sky/2020-09-24/2020-09-25")! as URL,
    cachePolicy: .useProtocolCachePolicy,
    timeoutInterval: 10.0)

request.httpMethod = "GET"
request.allHTTPHeaderFields = headers

let session = URLSession.shared
let dataTask = session.dataTask(with: request as URLRequest, completionHandler: { (data, response, error) -> Void in
    if (error != nil) {
        print(error!)
    } else {
        let httpResponse = response as? HTTPURLResponse
        print(httpResponse!)
    }
})

dataTask.resume()
```

Sample Response:

```
{5 items
  "Routes":[]0 items
  "Quotes":262 items
  [100 items
    0:{6 items
      "QuoteId":1
      "MinPrice":201
      "Direct":true
      "OutboundLeg":{4 items
        "CarrierIds":[1 item
          0:1281
        ]
        "OriginId":67852
        "DestinationId":82628
        "DepartureDate":"2020-09-01T00:00:00"
      }
      "InboundLeg":{4 items
        "CarrierIds":[1 item
          0:1281
        ]
        "OriginId":82628
        "DestinationId":67852
        "DepartureDate":"2020-09-01T00:00:00"
      }
      "QuoteDateTime":"2020-08-23T06:26:00"
    }
  ]
}
```

```

1:{6 items
  "QuoteId":2
  "MinPrice":338
  "Direct":true
  "OutboundLeg":{4 items
    "CarrierIds":[...]1 item
    "OriginId":67852
    "DestinationId":82628
    "DepartureDate":"2020-09-01T00:00:00"
  }
  "InboundLeg":{4 items
    "CarrierIds":[...]1 item
    "OriginId":82628
    "DestinationId":67852
    "DepartureDate":"2020-09-02T00:00:00"
  }
  "QuoteDateTime":"2020-08-24T09:17:00"
}
Carriers:[4 items
0:{2 items
  "CarrierId":1281
  "Name":"Jetstar"
}
1:{2 items
  "CarrierId":1606
  "Name":"Qantas"
}
2:{2 items
  "CarrierId":1841
  "Name":"Virgin Australia"
}
3:{...}2 items
]
"Currencies":[1 item
0:{8 items
  "Code":"AUD"
  "Symbol":"$"
  "ThousandsSeparator":","
  "DecimalSeparator":"."
  "SymbolOnLeft":true
  "SpaceBetweenAmountAndSymbol":false
  "RoundingCoefficient":0
  "DecimalDigits":2
}
]
}

```

OpenWeather API

The OpenWeather API will provide us with the current weather of a location of 5 days forecast.

URL: <https://openweathermap.org/>

Sample Get Request:

```
import Foundation

let headers = [
    "x-rapidapi-host": "community-open-weather-map.p.rapidapi.com",
    "x-rapidapi-key": "2b44f45586msh6138dce95c46efcp170543jsna3ba9defc8d2"
]

let request = NSMutableURLRequest(url: NSURL(string:
    "https://community-open-weather-map.p.rapidapi.com/weather?lat=-37.813629&lon=144.963058&callback=test&id=2172797&units=%2522metric%2522&mode=xml%252C%20html&q=London%252Cuk")! as URL,
    cachePolicy: .useProtocolCachePolicy,
    timeoutInterval: 10.0)

request.httpMethod = "GET"
request.allHTTPHeaderFields = headers

let session = URLSession.shared
let dataTask = session.dataTask(with: request as URLRequest, completionHandler: { (data, response, error) -> Void in
    if (error != nil) {
        print(error!)
    } else {
        let httpResponse = response as? HTTPURLResponse
        print(httpResponse!)
    }
})

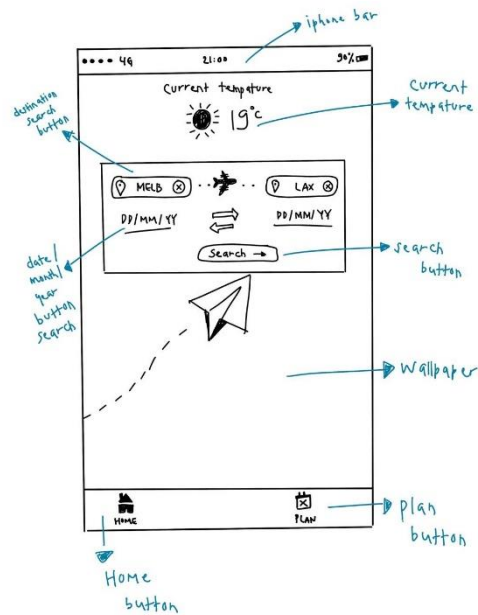
dataTask.resume()
```

Sample Response:

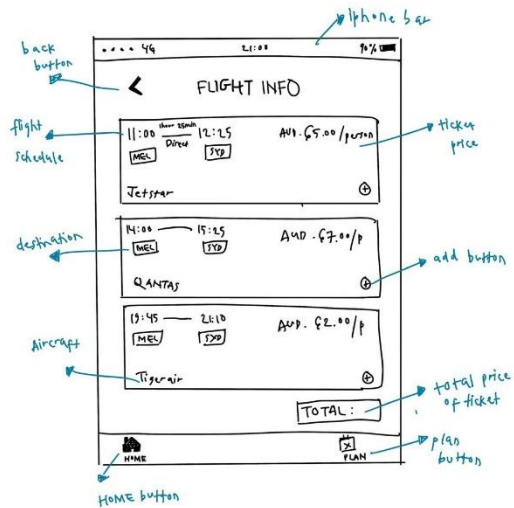
```
{
  "coord": {"lon":144.96,"lat":-37.81},
  "weather":[{"id":803,"main":"Clouds","description":"broken clouds","icon":"04n"}],
  "base":"stations",
  "main":{"temp":285.28,"feels_like":279.67,"temp_min":284.15,"temp_max":285.93,"pressure":1012,"humidity":66},
  "visibility":10000,
  "wind":{"speed":6.7,"deg":30},
  "clouds":{"all":75},
  "dt":1597154207,
  "sys":{"type":1,"id":9548,"country":"AU","sunrise":1597093791,"sunset":1597131662},
  "timezone":36000,
  "id":2158177,
  "name":"Melbourne","cod":200
}
```

Sketches

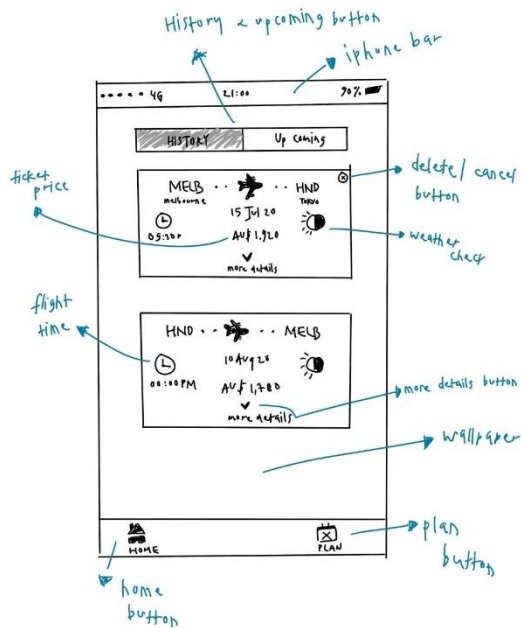
Low fidelity



Screen 1

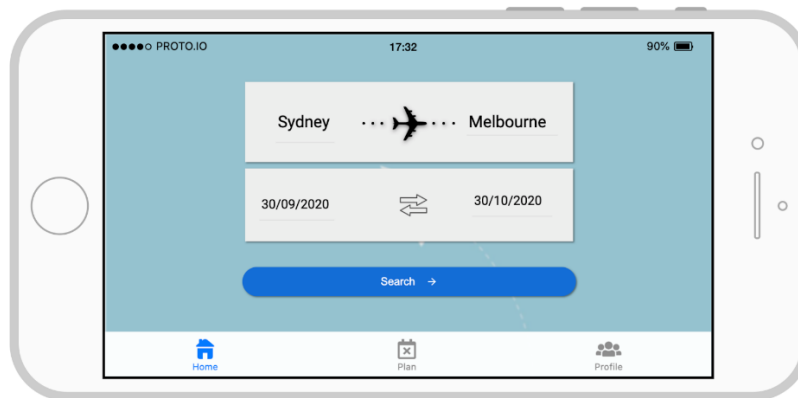


Screen 4



Screen 2 & 3 (same, just the history button change into up-coming).

Medium Fidelity
Adaptive Layout:

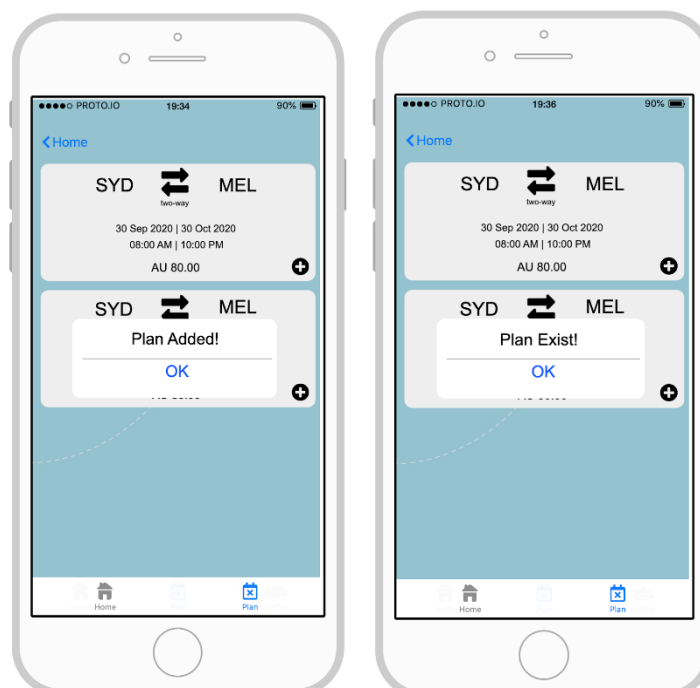


Home Page (Search for Flights)

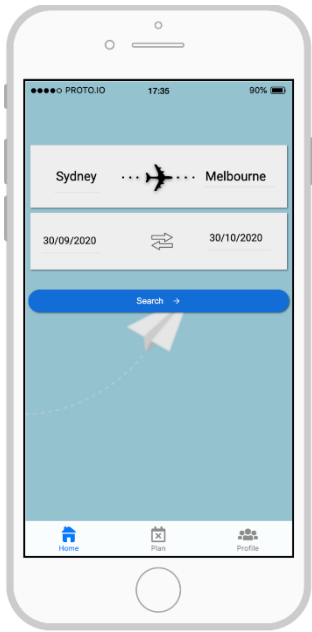


Plan (History) Page

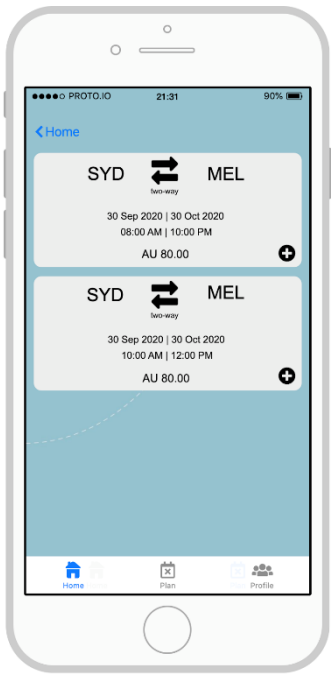
Popover Alert: (Showing Plan Added and Plan Exist after tapping on the add button)




Home

	<p>Home page shows 4 text boxes and 1 search button which can be interacted with. User can tap the departure and destination to select where they from and where they want to go. Below it, there's two date boxes the user can select when to start and when to end done selecting the above information user then can tap search to jump to the next page to get the flight information with the details they provided.</p>
---	---

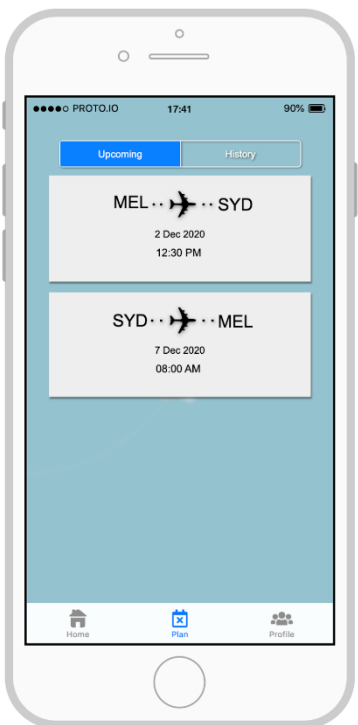
Home > Search

	<p>Lists all the flight information according to the user just provided the certain date and location on the home page. Then the users can select the plan they want and simply tap the add button to add the plan to the plan page.</p>
---	--

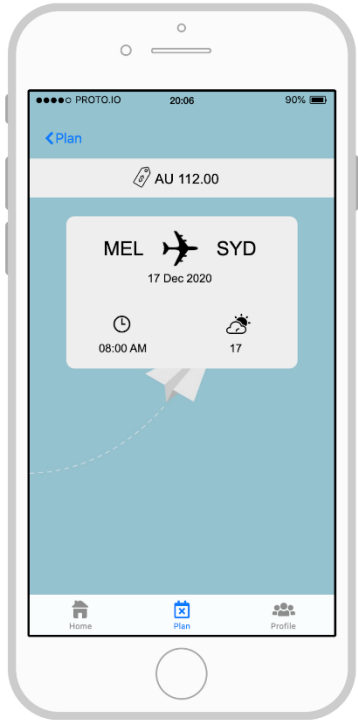
Plan > History

	<p>Lists all the plans the user has already set up before as history, the users can also delete the history by left scroll the screen.</p>
---	--


Plan > Up coming

	<p>Lists all upcoming flight info plans. The user can tap into the plan to see the details and also delete the upcoming plan by left scroll the upcoming plan.</p>
---	--

Plan > History > Details Cell

	<p>Show more details for the selected cell just by simply tap the certain plan inside the plan page.</p>
--	--

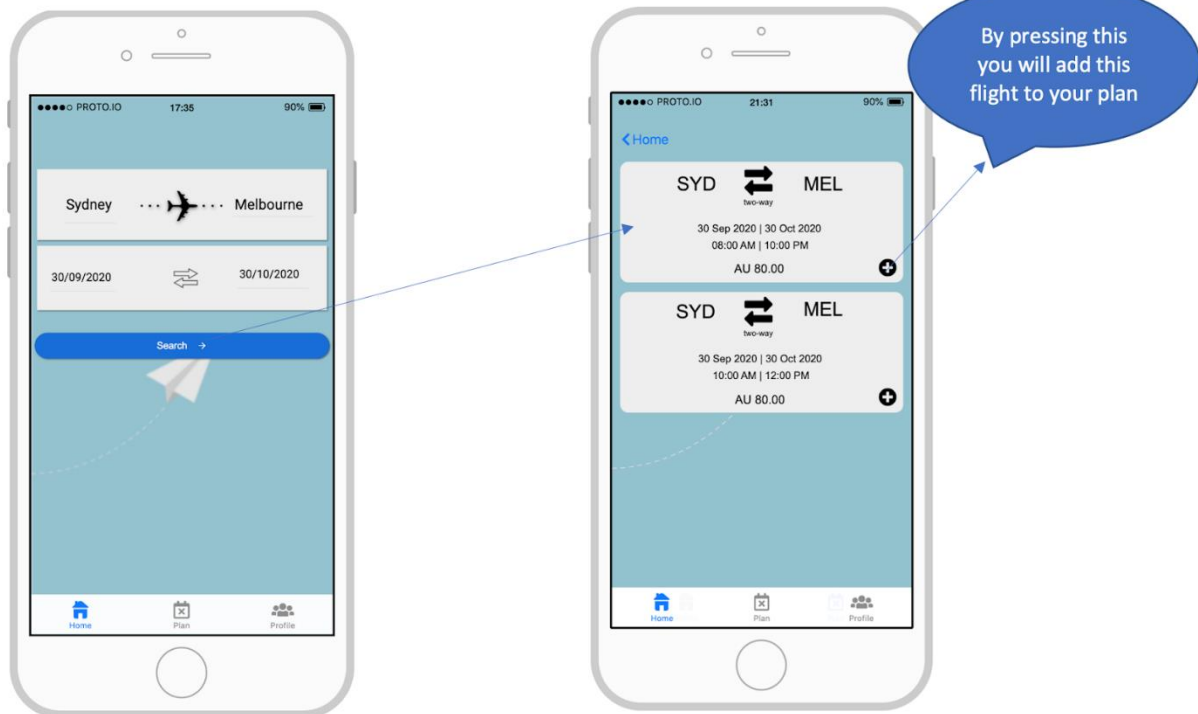
Profile

	<p>The page will show our group details.</p>
---	--

Workflow:

Searching for a Flight Plan Use Case

Home Screen → Plan



Flight Result > Adding Plan (Alert)



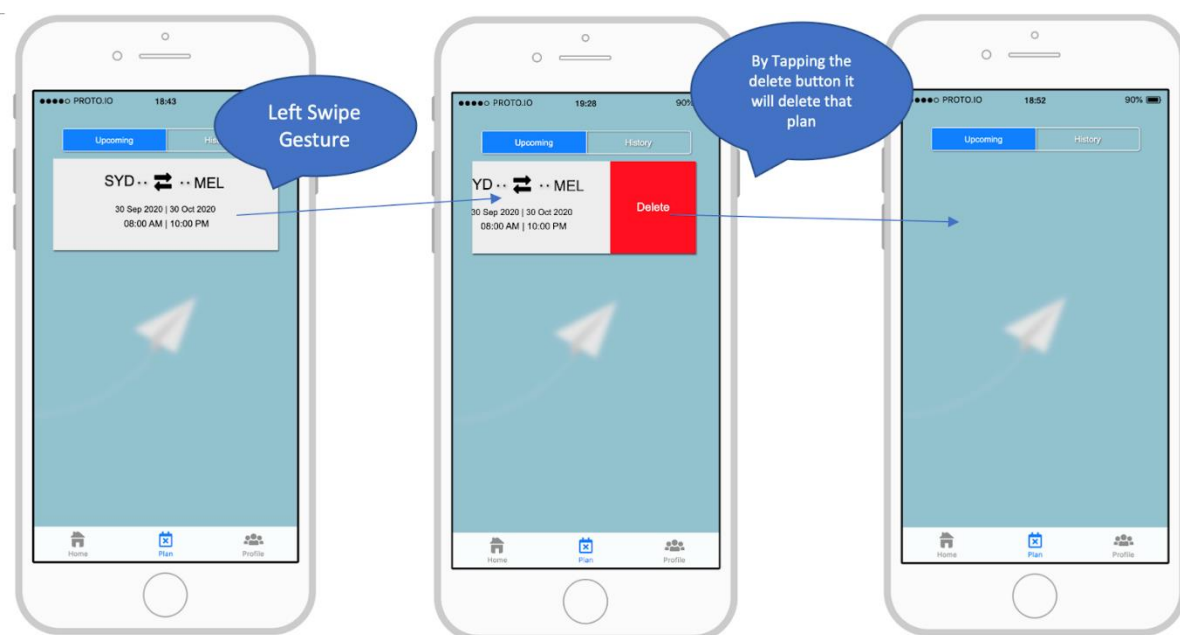
To view details about an upcoming flight plan Use Case

Plan → Show Detail



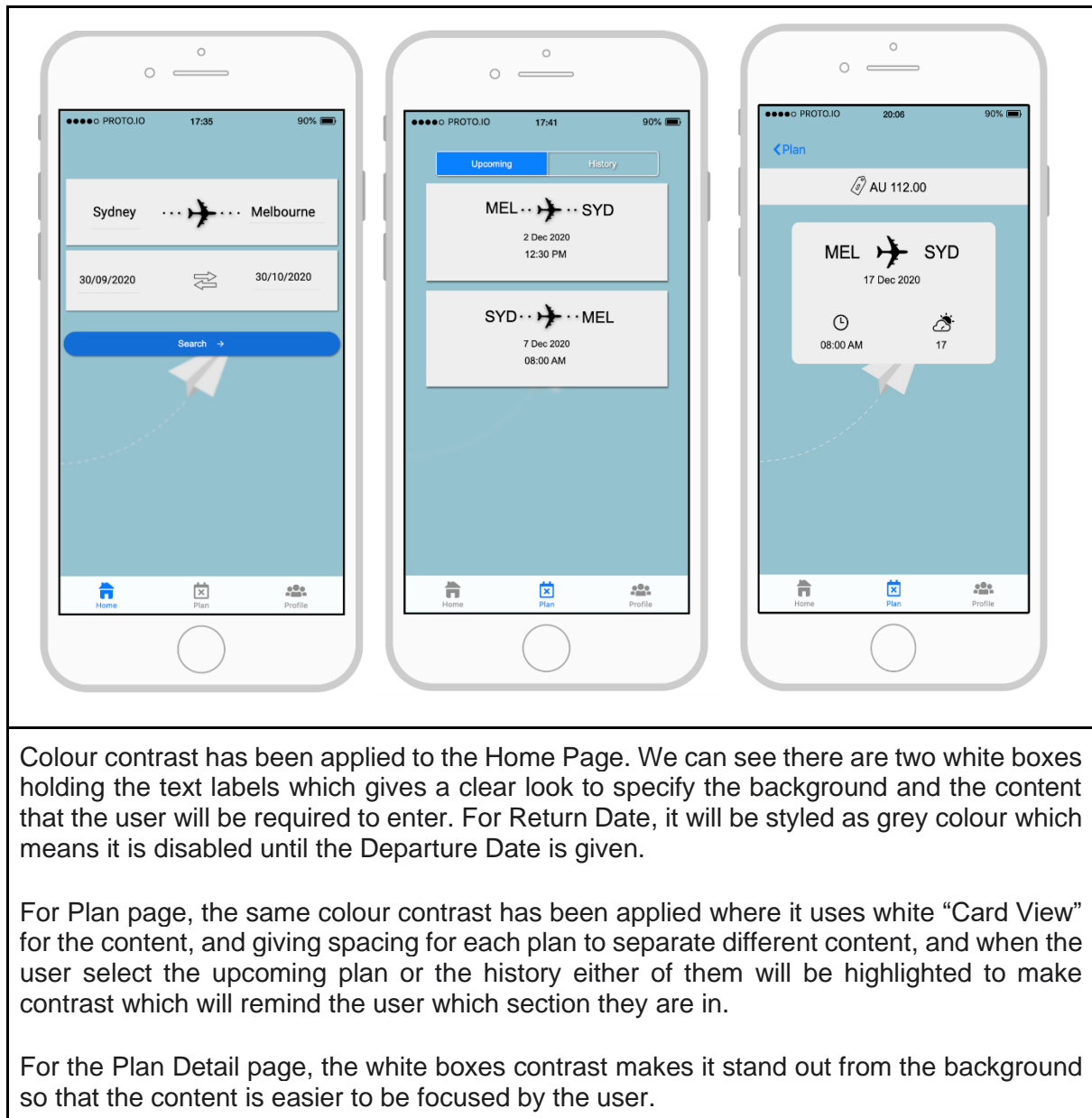
Deleting a plan from upcoming plan Use Case

Plan → Deleting Plan

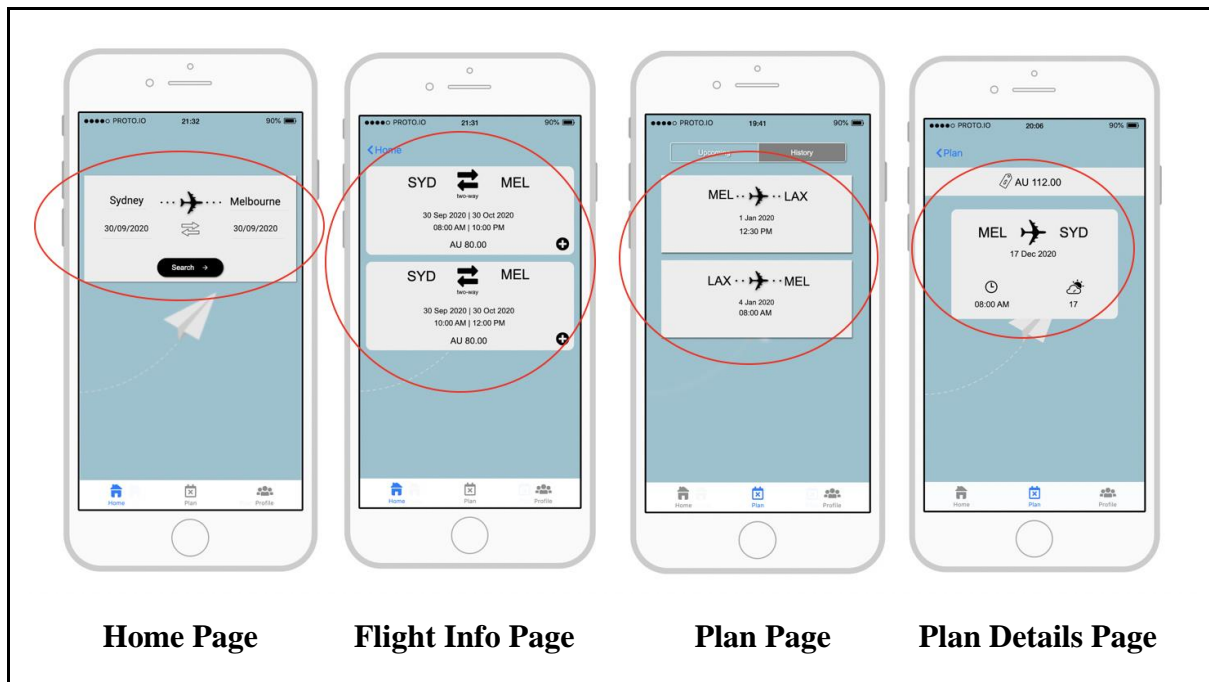


Design Theme

Contrast

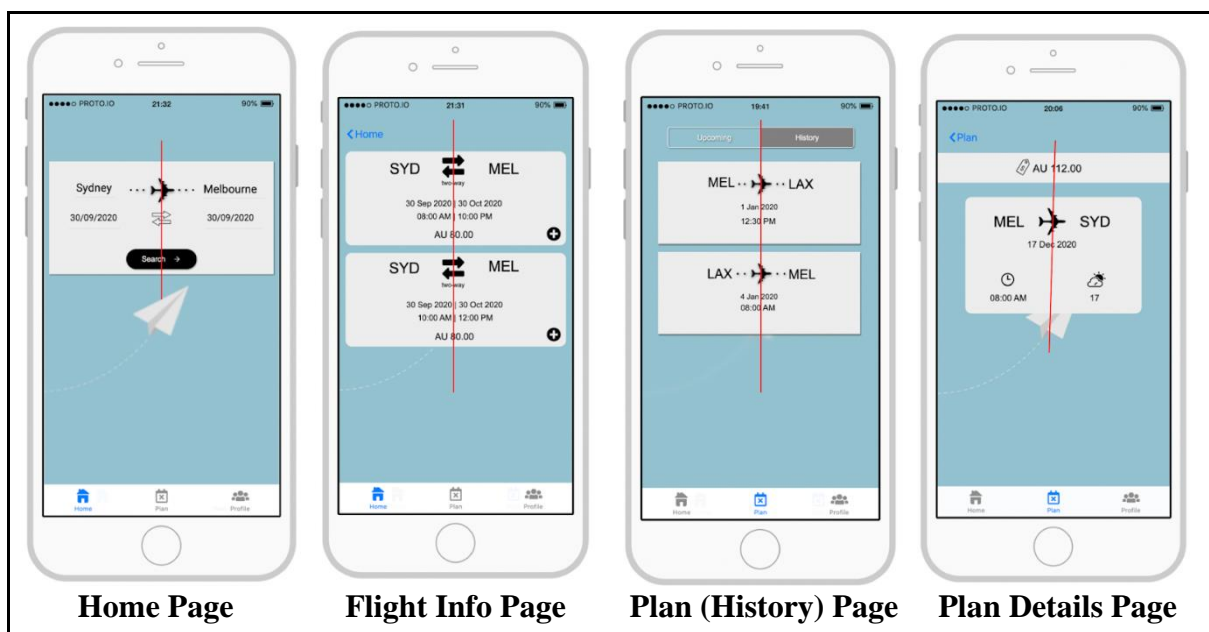


Repetition



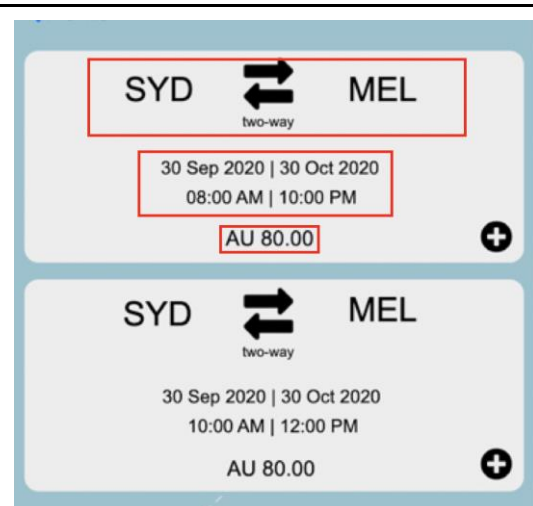
We applied the repetition design principle in our app. By applying uniform colours and themes throughout the pages as we use the same background, the blue background with the airplane logo in the middle. We apply the repetition principle by showing all the details or data to the users with a squared panel that is shaped like a “card” throughout all the screens. We uniformly use the white coloured “card” panel in all the screens, from the search panel, search results, plans, and even plan details. And if the data is more than one, we will show it with more than one “card” panel with a table having the cell designed like the “card” panel. By this way, the app will be comprehensible and will make the user feel familiar with the app and especially it will make the app look consistent on how it displays the data.

Alignment

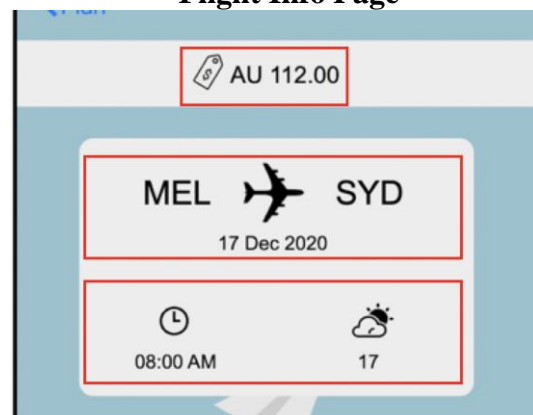


We apply the alignment design by picking the centred aligned text and all the data are all centred align as we do not have many data to shown we decided that it would be the best for us to pick the centred alignment for our data as it will be easier for the user to read and understand the data and we created a spacing between each data and it is very clear because each data will have its own “card” panel with their own white background so it is very clear to see the negative spacing between the different data (card).

Proximity



Flight Info Page



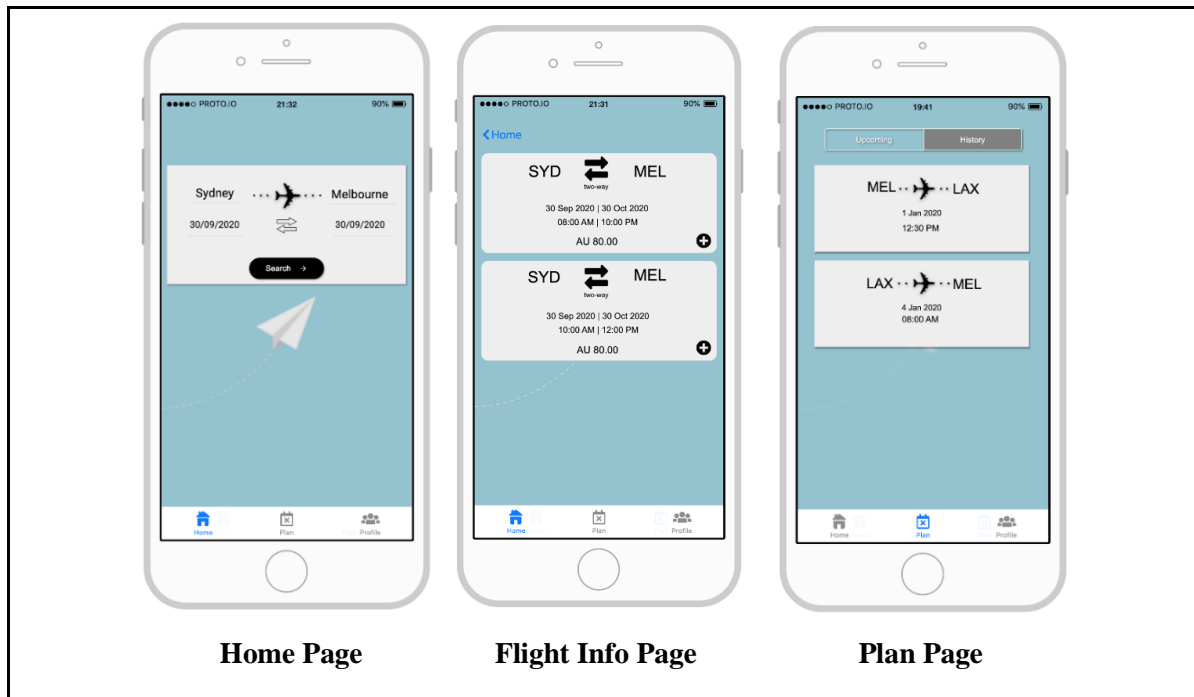
Plan's Details Page

We apply the proximity design by making a difference in the font size to the most important data (title) and the description we make it clear to read by making the title much bigger (SYD - MEL) is the main title and the description below it is smaller like the dates, time, and the price. Besides that, the description of the logo we make is the smallest. and the spacing between the data is not consistent for each data e.g. The time and date are close to each other as those 2 are related but the spacing is not the same with the price as it is not that related with the date and time. The city name of the source and destination is close to each other with the log as those things are related to each other and there is extra spacing to the description. It will make it easier to read as we divide the data by their relationship with each other as it is easier for the users' eye to see the group of data on which is related to each other.

As you can see the screenshots on the left I made a square the group all of the data that is related to each other and you can see the spacing is not uniform for each individual data it is only uniform for each that related and more spacing between related and unrelated data.

Design Theme (Apple)

Clarity



We organised our application content in such a way that it shows the desired message and allows users to act easily. We applied clarity in all our scenes. As you can see from the top, the texts are well spaced and not too complicated per line. Obvious icons have been used, for example the home, plan and profile button in the tab bar are not ambiguous, they clearly indicate what the symbol means. We are also using meaningful colors to indicate active states for example, blue indicates a tab bar is active and grey means inactive.

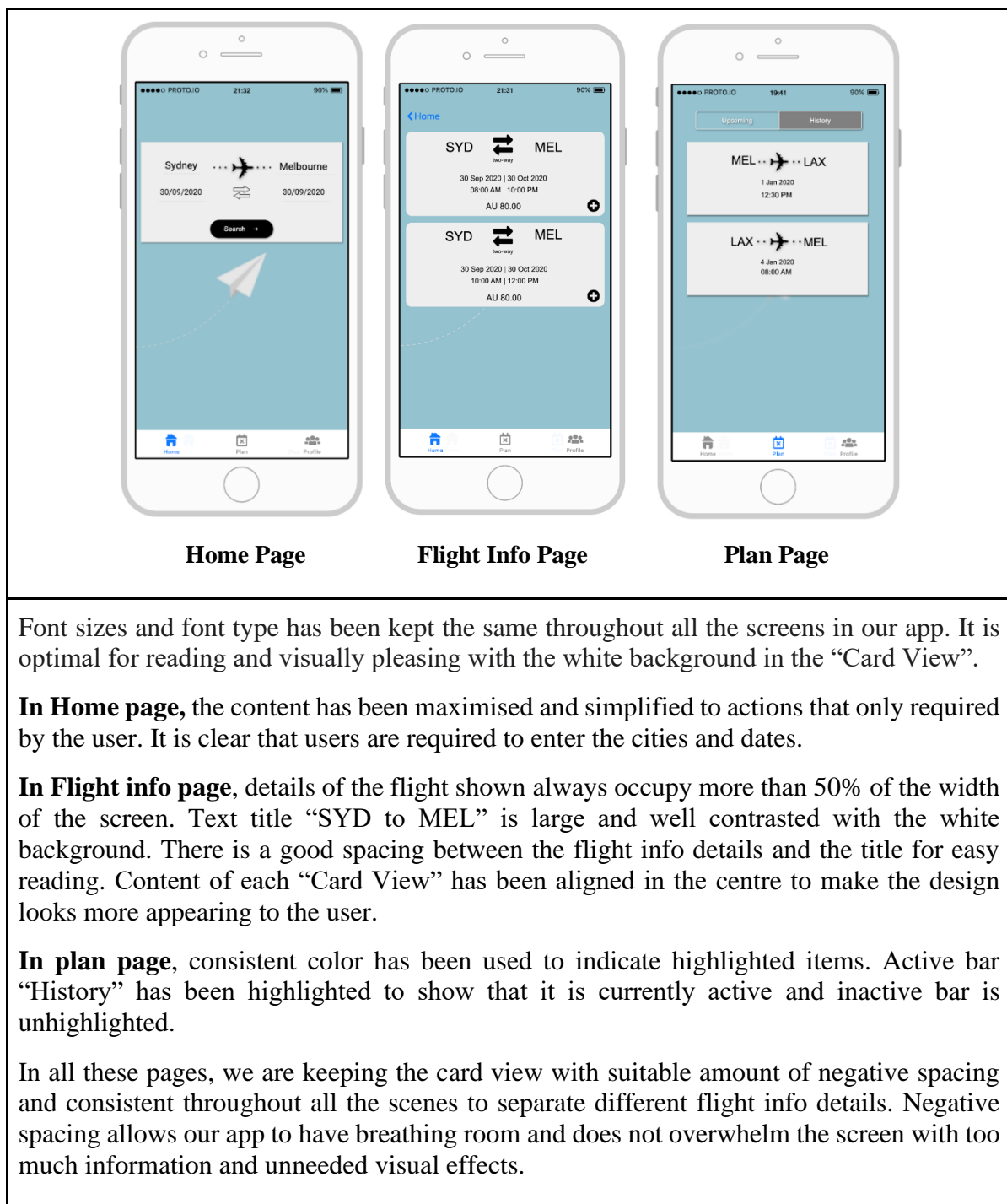
Home page looks simple and it is clear about what actions users can make with selecting cities, dates. After filling in the details, users will need to click on search button for flight results. Buttons and Text Fields in this page are setup in an optimal size where it can be easily tappable.

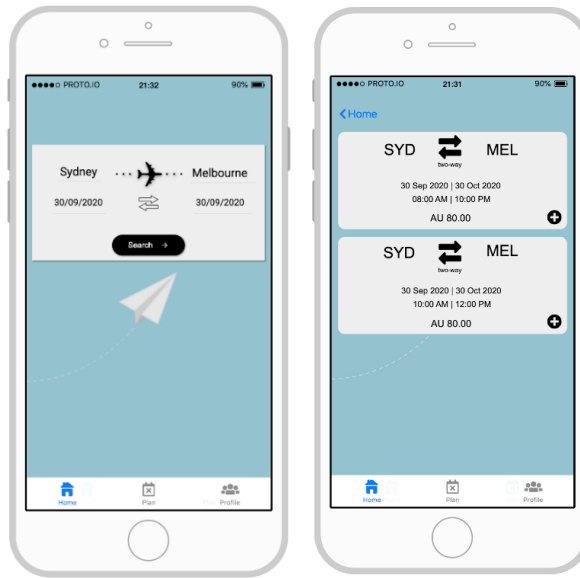
In Flight info page, details of the flight shown is clear and summarised in this page. It contains all the necessary information and a button on the right side which is an add icon. User will be able to save these plans by tapping the button. We are using the “< Home” instead of “< Back” to be clear about where user will be redirected.

In plan page, information is kept simple without making the user confused with unnecessary information. Flight info title “MEL to LAX” is larger in size so that it is easier for users to scan through the page.

In all these pages, we are keeping the card view with white background and consistent throughout all the scenes. Texts included are all minimal and enough to convey the information that users need.

Deference





Home Page

Flight Info Page



Plan Page

Plan Details Page

Depth theme has been applied through all the scenes. We used a blue background with our app. But we have setup the “Cards view” inside our app to have a white background. It allows the app to bring focus to the foreground and also keeping its natural colours.

Buttons in Home Page and Flight Info Page is coloured, so that it is obvious that user can interact with it by tapping search when they have filled in all the details or add plan by tapping the add button.

In Plan Details Page, users are allowed to delete a flight plan by swiping to the left.

Updated Wireframes for Assignment 2

Medium Fidelity:

Adaptive Layout:



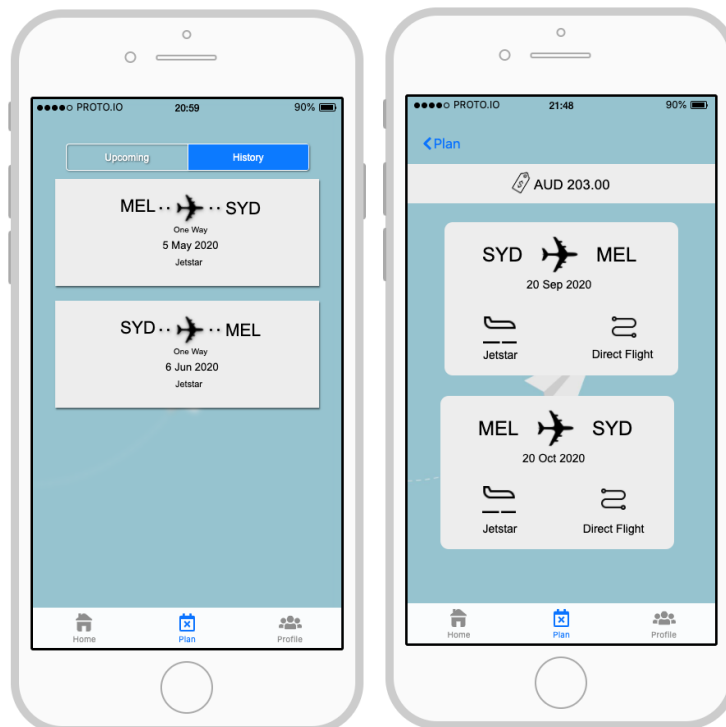
Home Page (Search for Flights)



Plan (History) Page



Flight Results Page



Plan Page

Updated workflow

Home Screen -> plan



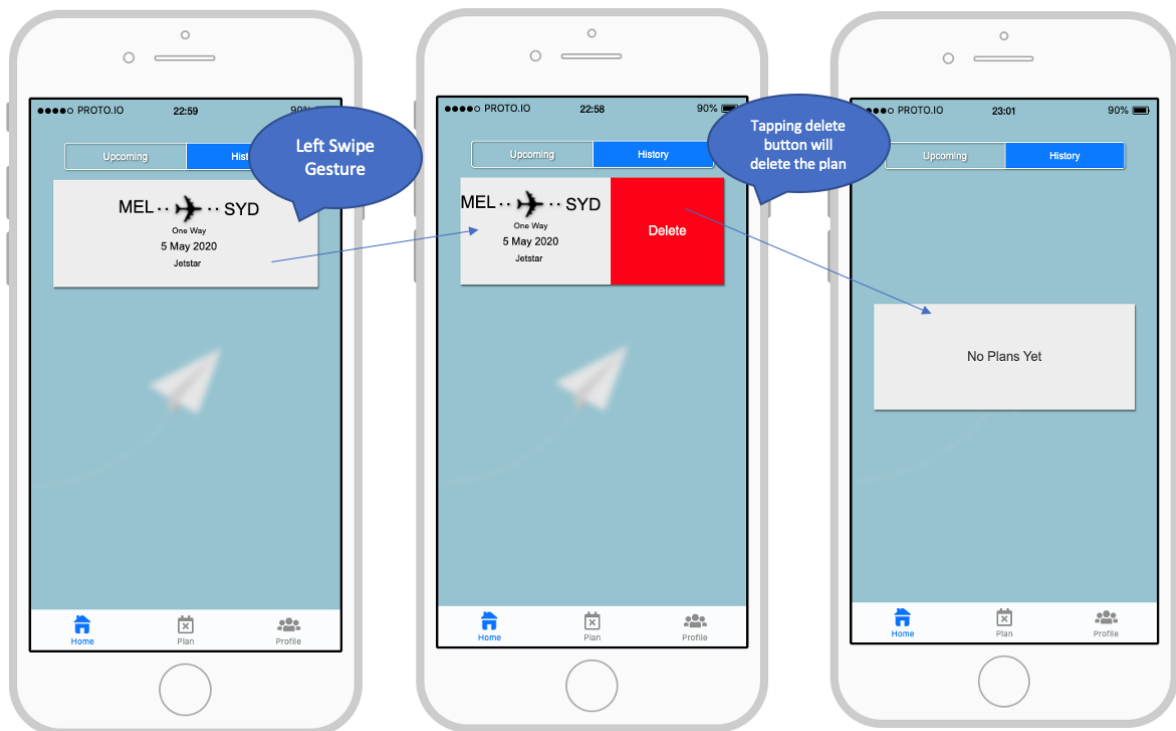
Flight Result > Adding Plan (Alert)



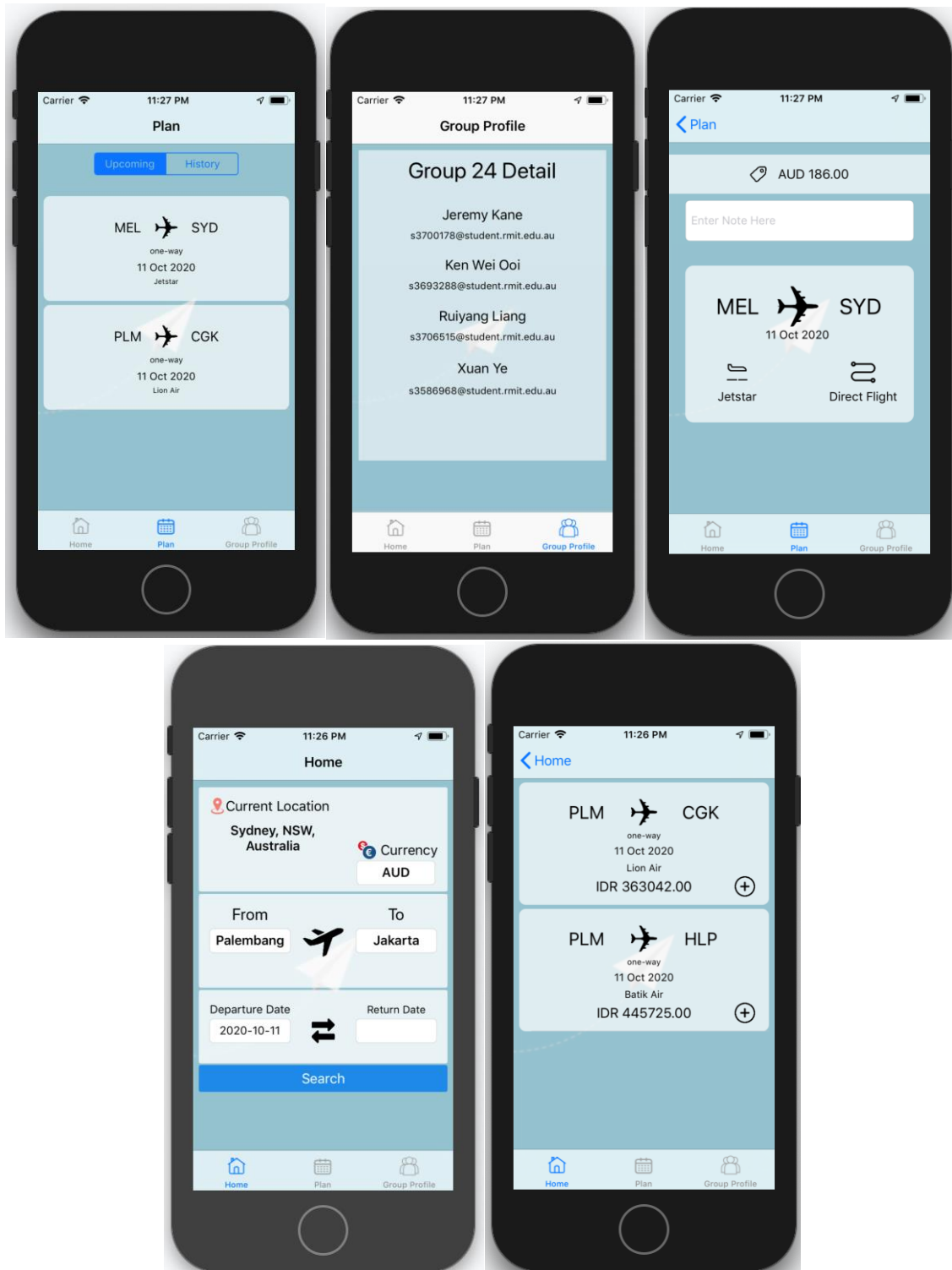
Plan > Show Detail

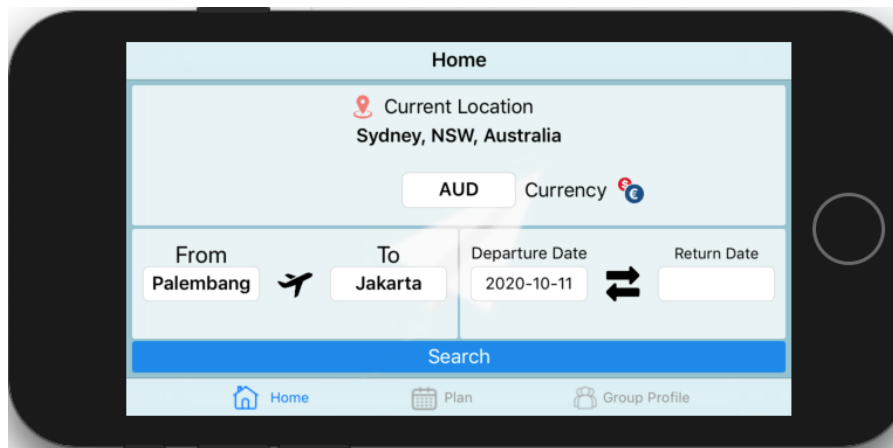


Plan > Deleting Plan



High Fidelity (Screenshots from Simulator)





Adaptive Layout on Home Page on iPhone SE with size classes

Networking Approach Research

Architecting a robust networking layer with protocols. Designing a robust networking layer of the app. Firstly, we will have a view model class and declare a service and the object we want, then we initiate the service, then we will need to create an easy way to use service which can be injected into our view model then the object can be fetched. After that we can then start to working at the service level and work to the networking layer, if we get a new object then the protocol of that object will be defined and created but that's not enough, the next thing we want to do is implementation of the function getObject which is implemented as an extension of protocol method itself which can give them default behaviours and functionality. So, if objects want to conform to the protocol can implement their get method. The get object method here will use a networking object, the fetch method will also be called after the get object method called which we will pass an Enum value, but now we still don't know what it will be at this moment, just informing the network something it has to fetch. The completion closure which is passed to the get object method which is passed on to the fetch(_:completion:) method. Above operations will get the data of the object and decode it when the network call is successfully done. Above operations make the view model as data source agnostic. Because of the protocol of object, we still need to get more goals done:

1. The networking object should be able to get the endpoint or request configuration of the object.
2. The networking fetchObject(_:completion:) will need to be decoded into the model.
3. Objects that have object getting protocol requires to have the object that is networking.

To meet the 1 and 2 goals, a new protocol called networking will be needed. In order to make the fetch function with (_:completion:) generic to be a decodable object T, high level of flexibility will be achieved. The service layer then can let the networking object decode the data. The endpoint will be accepted by the fetch method. The data we can assign it a struct for we need. Finally, add the network to the get object protocol that we declared before. The fetch object with (_:completion) still needs to add more stuff to it. The endpoint will be asked for the object for the URL request object which the request will be configured by the endpoint. Final modifications of the code are still needed, to do that the requestProviding protocol will be added, update the networking protocol and the extension of it, for the fetch object function should also be renamed to execute(_:completion:) since we don't know the network call will be GET or POST. The final thing to do is to update the getObject method which we added to

get object protocol, the method has changed the name from fetch to execute since the type of the argument `execute(_:completion)` wants the `requestProviding`.

The advantage of the approach is networking is highly testable, easily testing the data we get from the sent from the network and flexible due to everything we can get we can get from and check with the `URLRequest` which is abstracted behind the protocol.

Another approach which is where we are dealing with a lot of data from the app. There will be so many lines of codes which might repeat some of the codes, then we can just make them in just a few lines of codes, an example is that. `URLRequest.login(name:nameTF.text!, pass:passTF.text!).sendRequest { response in ...}`. To achieve that we first need to build a network layer, we build a request with its builders, the Enums then send the request to the alamofire. The request will be created by it and will just do it by matching the Enum and match it to the `URLConvertible` which is a protocol which will have a function that creates a URL request. Inside the Enum `TodaoRouter: URLRequestConvertible`, we will have the `baseURL` the case to match with the data, function `asURLRequest()` throws -> `URLRequest` which containing implemented the Enum value before we set like get, create and delete which will return the kind of API call, below it let the params to another Enum if is get, delete then return nil, if create then return object name, after it we start to set the URL based on the based URL we had, after getting the full URL we can then get the data, if we want to create many Enums in our app in different parts, we will be using many configurations for the requests which will be matching the Enum. The things like headers, main URL and the other information we need to create our request, the `URLRequestConvertible` is also needed to the Enum which is used for conformance to them, use it once rather than rewriting it in every Enum. After the above operations, a new Enum we will create, append the previous protocol to it then the request is done, and we can then use it.

For example, we will create a user Enum with login and register cases to match with the Enum we created before, we first to confirm which URL we need for the login path or register path, after that we just need to add and match the information for the user and return the params. Then it is time to create a server path Enum for use, which contains the function the user needs and the path of the URL the API needs. The good thing about using this approach is we do not need to write many repeating codes and simplify the work for the developer.

A case of using:

The robust networking layer with protocol this approach using the view model which is in a good design structure and it compares with the second approach, the second one using Enum to match the data we need might be messy sometimes even though it can reduce the workload and the repeating of codes. Secondly, the sometimes it's easier to check the value that we passed in the struct, for the first approach is 100% testable and mockable due to its feature which has everything down to the `URLRequest` no need to check every Enum one by one, and also for the first approach which is also very flexible just add more method to get the object unlike the second one approach which need to add one more Enum for it.

Networking (Updated REST API)

In A2, we have used Skyscanner to get all the currencies, airport codes and flight quotes. Which is total of 3 different API Calls for the prototype. As we are using Basic Free Plan, 50 requests/minute is the limit for our API calls.

3.1 REST Skyscanner currency

Sample Request:

```
class REST_Skyscanner_currency {

    // Array of all currency from the request
    private var _currencies:[String] = []
    var delegate:Refresh?

    let headers = [
        "x-rapidapi-host": "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
        "x-rapidapi-key": "76a5a70cb6msh2275a6e85d3ba0dp1ed0a2jsn2d38f62c123f"
    ]

    private let session = URLSession.shared

    private let base_url:String = "https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/reference/v1.0/currencies"

    var currencies:[String] {
        return _currencies
    }

    func getAllCurrency() {
        guard let escapedAddress = base_url.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)
        else {return}
        if let url = URL(string: escapedAddress) {
            var request = URLRequest(url: url)
            request.allHTTPHeaderFields = headers
            getData(request, element: "Currencies")
        }
    }

    private func getData(_ request: URLRequest, element: String) {
        let task = session.dataTask(with: request,
            completionHandler: {
                data, reponse, downloadError in
                    if let error = downloadError {
```

3.2 REST Skyscanner airportCode

Sample Request:

```
class REST_Skyscanner_AirportCode {
    var places: [Places] = []
    var delegate: Refresh?

    let headers = [
        "x-rapidapi-host": "skyscanner-skyscanner-flight-search-v1.p.rapidapi.com",
        "x-rapidapi-key": "b5160ea580msh7ae96c9d60b5bc2p1e35b4jsn4850892b52cd"
    ]

    private let session = URLSession.shared

    private let base_url:String = "https://skyscanner-skyscanner-flight-search-v1.p.rapidapi.com/apiservices/autosuggest/v1.0"

    func getAirportCode(countryName:String, currency:String, place:String, location:String) {
        let url = base_url + "/" + countryName + "/" + currency + "/" + place + "/" + "?query=" + location

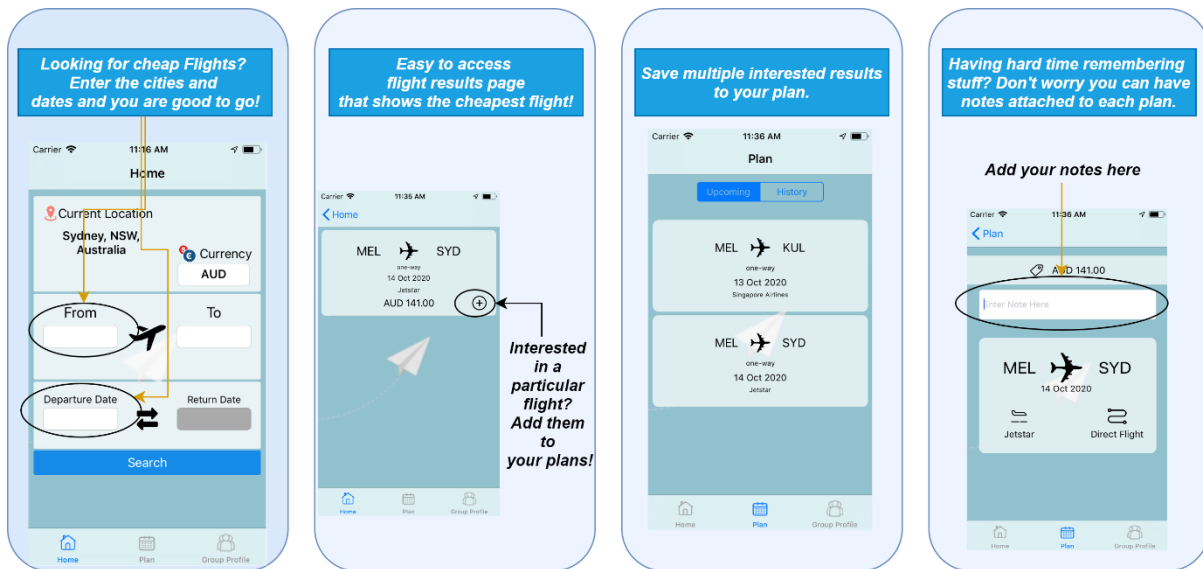
        guard let escapedAddress = url.addingPercentEncoding(withAllowedCharacters: CharacterSet.urlQueryAllowed) else{return}

        if let url = URL(string: escapedAddress) {
            var request = URLRequest(url: url)
            request.allHTTPHeaderFields = headers
            getData(request, element: "results")
        }
    }

    private func getData(_ request: URLRequest, element: String) {
        let task = session.dataTask(with: request, completionHandler: {
            data, response, downloadError in

                if let error = downloadError
                {
                    print(error)
                }
            }
        )
    }
}
```

App Store Page Design



We will include these scenes into our app store page. Besides that, we will add a short description for our app listing the features of our app.

Description of our APP:

Features:

- Trip Planner allows you to search for cheap flights anywhere you want to go.
- Trip Planner includes a flight result page that display all the cheapest flight you could find for your desired destination.
- Trip Planner includes a Plan's Page for you to view all saved flight plans.
- Trip Planner includes a Note box in each plan, so you could add some simple notes for that plan.

Assignment 2 Update Summary

In Assignment 2, there are several parts where we have improved based on the feedback of A1. We have removed Weather API and used multiple SkyScanner API calls instead to get currencies in the home page, get airport codes and using the airport codes to search for the flight quotes from SkyScanner.

Updates that have been made:

1. Cocoa Framework:
Core location – Our prototype uses core location to track user's location if they allowed the app to do so. Then, we will update the currency based on the user's location.
2. Core Data:
Our Plan's page contains a newly update adding notes feature, so notes added by user will be saved in core data.
3. Adaptive Layout:
On our home page, adaptive layout and auto layout have been applied with size classes on iPhone SE.
4. Home Page:
Users are now able to choose their choice of currency on the home page, and the flight results will display the price in the chosen currency.
5. Plan Page:
Notes feature has been added. Users can add some notes on each plan that they add from the flight results page and will be able to update it if they need to.
6. Cocoa Pods:
CocoaPods have been installed in this project and we have included Google Places SDK to use the places autocomplete for searching for cities. CocoaPods helps to keep the whole Swift project dependent instead of typing unnecessary file in the form of frameworks and libraries.

Student Weekly Log

Week 2

Week 2	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Schedule Weekly Meeting
Pick App idea or development
Research on available APIs

Task List (Coming Week)	Completed
Pick Project Idea	Y
Create Google Docs for the report	Y
Create Facebook Messenger for communication	Y

Week 3

Week 3	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Research 4 Similar Apps
Define Project Statement
Design Low Fidelity

Task List	Completed
Define Slogan	Y
Define purpose	Y
Define selling points	Y
Define use case scenario	Y
Define app advantages and disadvantage	Y

Sketch Low Fidelity	Y
Research about 4 Other similar apps	Y

Week 4

Week 4	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Create the project structure
Create develop branch
Designing Low Fidelity
Designing Medium Fidelity

Task List (Coming Week)	Completed
Continue Sketch Low Fidelity	Y

Continue Sketch Medium Fidelity	Y
Start creating the project structure	Y
Create develop branch	Y

Week 5

Week 5	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Separate task for the prototype coding part.
Separate the task for report (Summary of difference between research app).
Decide the design pattern to be used in the app.
Research about how to do custom table view cell.
Research about how to do pickview (Date, Custom Item).
Try out adaptive layout and auto layout

Task List (Coming Week)	Completed
Task separation Report	Y
Task separation Code	Y
Choosing Design Pattern	Y
Research about custom table view cell	Y
Testing with auto layout and adaptive layout (Setting up constraints)	Y
Research regarding UIPickerView	Y

Week 6

Week 6	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Finalising the report

Complete design themes and principle based on screenshots of medium fidelity
Applying design pattern to the code.
Applying constraints for auto layout and adaptive layout
Update the UI of our prototype
Finalising the prototype

Task List (Coming Week)	Completed
Finalising the report	Y
Complete design themes and principle based on screenshots of medium fidelity	Y
Applying design pattern to the code.	Y
Applying constraints for auto layout and adaptive layout	Y
Update the UI of our prototype	Y
Finalising the prototype	Y

Week 7

Week 7	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes

Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Formatting the report for submission
Refactoring the code for submission
Commenting the code for submission
Fixing crashing issue when tab bar button is clicked twice
Finalised the UI of our prototype

Task List (Coming Week)	Completed
Formatting the report for submission	Y
Refactoring the code for submission	Y
Commenting the code for submission	Y
Fixing crashing issue when tab bar button is clicked twice	Y
Finalised the UI of our prototype	Y

Week 8

Week 8	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment	Yes	Yes	Yes	Yes

(25%)				
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Discuss on what could be improved on our application
Discussing regarding the approach of REST api call

Task List (Coming Week)	Completed
Research regarding approaches of REST API in swift	Y
List out what could be improved	Y

Week 9

Week 9	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes

Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Based on the released feedback, discussion about what could be further improved on design and structure.
Separate tasks for team members (Home Page, Results page, Plan page, REST API structure, UI testing, Unit Testing)
Discuss which cocoa framework we could use.
Research about how core data could be implemented

Task List (Coming Week)	Completed
Update Home Page to use core location	Y
Research about core data.	Y
Decide on which cocoa framework we could use.	Y
Tasks separated among team members.	Y
Decide on how adaptive layout should look like for updated view.	Y

Week 10

Week 10	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Team members show progress of the updated project.
Discuss what can be improved on the current progress.

Task List (Coming Week)	Completed
Complete home page with core location, currency API call.	Y
Complete home page with adaptive layout with iPhone SE.	Y
Use core data for Plan page.	Y
Implement UI testing.	Y
Implement Unit Testing.	Y
Update Flight Results Page with API calls from Skyscanner	Y

Week 11

Week 11	s3693288	s3700178	s3706515	s3586968
Attended Meeting/Discussion (25%)	Yes	Yes	Yes	Yes
Contribute to Week Task Assignment (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Task List (25%)	Yes	Yes	Yes	Yes
Completed Previous Week Tutorial Exercises and or research learning (25%)	Yes	Yes	Yes	Yes
Total	100%	100%	100%	100%

Agenda
Finalise the application.
Update report with the updated prototype
Check for any bugs present inside the application.

Task List (Coming Week)	Completed
Finished Adaptive layout on the home page.	Y
Fixed crashing bugs by adding conditions for accessing API.	Y
Updated report with updated prototype.	Y
Finalised the application with all required functions.	Y
Finish testing on the application.	Y

Reference

1. <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>
2. Protocol-Oriented Network in swift.
<https://www.linkedin.com/pulse/protocol-oriented-network-swift-abdulrahman-eaita>
3. Architecting a robust networking layer with protocols.
<https://www.donnywals.com/architecting-a-robust-networking-layer-with-protocols/>