CA400 Airport Assistant

Testing Manual

Name: Shauna Moran

Student Number: 15381166

Supervisor: Ray Walshe

Table of Contents

Unit Testing

Unit testing is where individual components of the software are tested. While testing this application both standard unit tests and instrumented unit tests were carried out. The latter will be dealt with in further detail later in this document. These tests were carried out using Junit4 and Espresso. Here we will delve into how the AR functionality of the application was unit tested. In the ARUtilTest file each of the functions in the ARUtil file are tested.

GetCubeVolumeTest

Firstly, we will look at the getCubeVolume test. Here we ensure that the algorithm being used to calculate the cuboid liquid's container dimensions is working correctly. We pass four values into the function which represent the length, width and height values that the AR scan has gathered and the correct volume value. A margin of error of 0.01 is also been passed to the test.

```
@Test
public void getCubeVolume() {
    assertEquals(1.0, arHelper.getCubeVolume(1.0, 1.0, 1.0), 0.01);
    assertEquals(24.0, arHelper.getCubeVolume(2.0, 3.0, 4.0), 0.01);
    assertNotEquals(24.0, arHelper.getCubeVolume(-2.0, 3.0, 4.0));
}
```

Two different scenarios are checked by this function. AssertEquals checks that the three numbers we have passed in to the function return the correct volume which has been passed in. An assertNotEquals test is also carried out. This checks that the length, width and height values being passed to the do not equal the result value which we are passing to it. This test returns a positive result.

GetCylinderVolumeTest

A similar test, but one that proved very important was the getCylinderVolume test. Again, this use of this test is to ensure that the algorithm returns the correct volume. Unlike the last test it is passed three values. The volume result, the width of the container and its height. This time a margin or error of 0.5 is passed to the function. This is due to the increased complexity of the algorithm used in this calculation.

```
@Test
public void getCylinderVolume(){
    assertEquals(12.57, arHelper.getCylinderVolume(4.0, 1.0), 0.5);
    assertEquals(307.88, arHelper.getCylinderVolume(14.0, 2.0), 0.5);
    assertNotEquals(21.5, arHelper.getCylinderVolume(2.0, 9.3));
}
```

When I first ran this test I was unsure why the test kept failing. Once I looked into the issue I finally realised that my application was calculating the area of a cylinder rather than the volume. Thankfully, this test made me spot this issue so I could resolve it. This test, similarly to the test above has an assertEquals and an assertNotEquals. Once the function was corrected, the test now returns a successful result.

LuggageResultTest

Unlike the above tests, the luggageResult test must be passed six values. This is because this function checks the user's luggage dimensions (the first three values) against the airline's dimensions (the last three values). The test uses an assertTrue and assertFalse.

```
    @Test
    public void luggageResult(){
        assertTrue(arHelper.luggageResult(30, 20, 10, 40, 30, 20));
        assertFalse(arHelper.luggageResult(55, 40, 30, 20, 10,5));
    }
}
```

AssertTrue checks that the function returns true, that the user's luggage is of the correct dimensions and assertFalse checks that the function returns false, that the user's bags do not pass the luggage check. A successful result is returned from this test.

PassFailResultTest

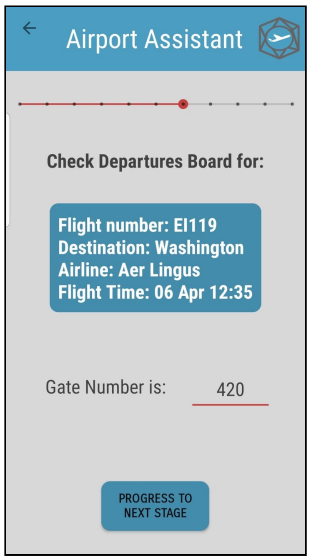
Finally, we must check the passFailResult function. The purpose of this function is to check if the values passed to it by the cylinder and cuboid volume functions are less than 200ml. They then return a pass or fail result. Similarly to the last test, this is carried out using an assertTrue and an assertFalse.

```
@Test
public void passFailResult() {
    assertTrue(arHelper.passFailResult(100));
    assertFalse(arHelper.passFailResult(250));
}
```

The assertTrue checks that the value entered is below the max container volume size and assertFalse checks that the value entered is above this size. This test returned a successful result.

Instrumented Unit Testing

Instrumented unit tests are unit tests that run on devices and emulators. As my application contains a vast amount of user interfaces I had to ensure that each of these worked well together. Instrumented unit tests have been written for each of the screens of the application. Button clicks, radio button selections and text entered are all tested to ensure that they are operating correctly. In total there are 27 instrumented unit tests. These tests were carried out using Junit4 and Espresso.



Some examples of such tests include the DirectionsToGate1.java. As you can see to the left, on this screen the user must enter their gate number and click the progress button. In this instrumented test we check that this functionality works correctly.

DirectionsToGate1Test

```
  @Test
public void testStart() {
    assertNotNull(directionsToGate1.btProgress);
}
@Test
public void enterGateInfo() {
    onView(withId(R.id.gateNumberEnter)).perform(typeText("408"));
    closeSoftKeyboard();
    onView(withId(R.id.btProgress)).perform(click());
}
```

In this test I firstly check that the button “Begin Journey” exists using an assertNotNull. We then check that the page as a whole works correctly. This test fills in the “Gate Number” and then clicks the “Progress” button. This test returns a successful result.

Another example of such instrumented unit testing is the SecurityARShapeCheckTest. In this screen the user must select whether the they want to continue with a cylinder scan or a cuboid scan.

SecurityARShapeCheckTest

```
@Test
public void testStart() {
    assertNotNull(securityARShapeCheck.btCyclinder);
    assertNotNull(securityARShapeCheck.btCuboid);
}
@Test
public void cylinderClick() {
    onView(withId(R.id.btCyclinder)).perform(click());
}
@Test
public void cuboidClick() {
    onView(withId(R.id.btCuboid)).perform(click());
}
```

In this test we firstly check that the the two buttons are present again using an assertNotNull. Following on from this we have two tests. One checks the cylinder click button and the other checks the cuboid click button. Once we have completed both of these checks we know that the activity works correctly. This test also returns a successful result.

Use Case Testing

Use Case testing allows us to step through the application inspecting each use case and each possible variation within each of these use cases. This manual testing technique tests each aspect of the applications functionality. One then marks whether this functionality works as expected or whether an issues were thrown in the process.

We can see below an issues that arose during this testing and the solutions to such issues.

Reference Number	***Scenario***	***Result***	***Developers comments/proposed solution***
001	Login to the application	Success - user logged in	
002	Logout of the application	Success - user logged out	
003	Upload a journey	Crashes if space left after flight number	Will add a check for this and remove it
004a	Yes checking in luggage - no values filled in	Success - correct warning message appears	
004b	Yes checking in luggage - one value not filled in	Success - correct warning message appears	
004c	Yes checking in luggage - all values filled in	Success - values added to Firebase	
004d	Not checking in luggage	Success - stage is skipped	
005a	Confirm Journey	Success - journey begins	
005b	Cancel Journey	Success - user returned to home page successfully	
006a	Use AR functionality to measure hand luggage	Application crashes if click clear points before any points entered	Need to add an if statement for this case
006b	Skip AR functionality	Success - AR stage is skipped	
006c	Quit AR functionality during scan	Success - user returned to AR start screen	
006d	Repeat AR Scan	Success - scan is repeated successfully	
007a	Get directions to airport- car	Success - correct directions returned	

007b	Get directions to airport- public transport	Success - correct directions returned	
007c	Get directions to airport- cycling	Success - correct directions returned	
007d	Get directions to airport- walking	Success - correct directions returned	
007e	Get directions to airport- no values filled	Success- correct error message is displayed	
007f	Skip directions to airport	Success - user brought to next screen	
008a	Complete check-in phase	Success - information displayed correctly. Link working.	
008b	Do not check-in luggage and skip check-in phase	Success - user brought straight to security screen	
009a	Skip Security AR Liquid Scan	Success - user brought directly to security screen	
009b	Complete Security AR Cuboid Scan	Measurements not accurate	Look into how the application maps the planes. Then look how this affects the measurements.
009c	Complete Security AR Cylinder Scan	Measurements not accurate	Look into how the application maps the planes. Then look how this affects the measurements.
009d	Skip Security AR Cuboid Scan	Success - user brought to security screen	
009e	Skip Security AR Cylinder Scan	Success - user brought to security screen	
009f	Repeat Security AR Cuboid Scan	Success - Scan repeated successfully	
009g	Repeat Security AR Cylinder Scan	Success - Scan repeated successfully	
010	Complete security phase	Success - user brought to duty free screen	
011	Complete duty free stage	Success - map displayed correctly	

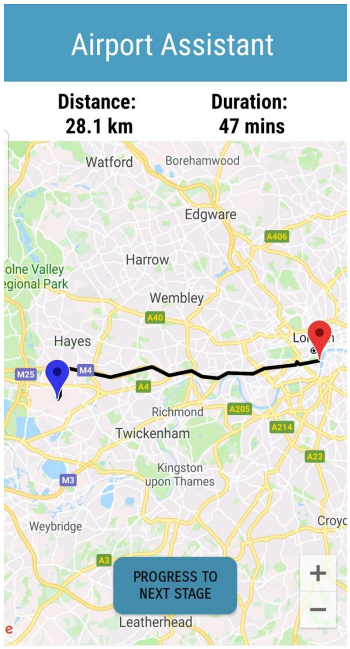
012a	Enter gate number	Success - directions displayed on next screen	
012b	Do not enter gate number	Success - correct warning message displayed	
012c	Get directions to gate	Issue with certain gate numbers. Directions not brought directly to the gate.	Appears to be issue with the floor in the airport. Will look into this in further detail.
013	Complete in-flight stage	Success - correct information displayed. Link working correctly.	
013a	Enter destination, select car and click get directions to destination	Success - correct directions displayed	
013b	Enter destination, select public transport and click get directions to destination	Success - correct directions displayed	
013c	Enter destination, select cycling and click get directions to destination	Success - correct directions displayed	
013d	Enter destination, select walking and click get directions to destination	Success - correct directions displayed	Need to look into why this is happening
013e	Do not enter destination	Success - correct warning message displayed	
013f	Do not select mode of transport	Success - correct warning message displayed	
013g	Skip directions to destination	Success - works correctly. Journey completed.	
013h	Complete directions to destination stage	Success - journey completed successfully	

Integration Testing

Integration testing is where a number of the units to be tested in the software are combined and tested as a whole. The purpose of this testing is that although we have already tested each unit we must test the interactions between these units. A number of integration tests were carried out during the testing of this application. They were used to test how certain elements such as the AR SDK, Maps APIs and web scraping integrate with the application.

ArrivalsMap test

```
@Test
public void testStart() {
    assertNotNull(arrivalsMap.btProgress);
}
@Test
public void progressButtonClick() {
    onView(withId(R.id.btProgress)).perform(click());
}
```



In this test we are testing how this activity integrates with the number of Maps APIs and the map fragment which are used in this screen. Firstly there is a testStart test which checks that the progress button is present using an assertNotNull. We then check the clicking of the button. Both of these tests return a successful result.

ARCubeScan test

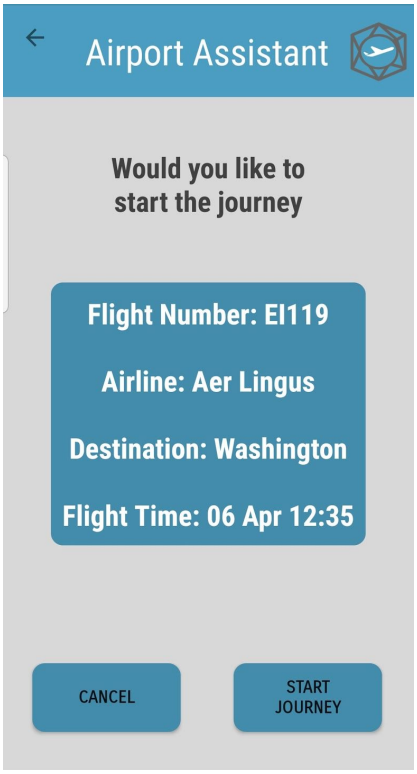
```
@Test
public void testStart() {
   assertNotNull(securityARCubeScan.btComplete);
   assertNotNull(securityARCubeScan.btQuit);
   assertNotNull(securityARCubeScan.btClearPoints);
}
@Test
public void completeClick() {
   onView(withId(R.id.btCompleteScan)).perform(click());
}
@Test
public void quitClick() {
   onView(withId(R.id.btQuitScan)).perform(click());
}
@Test
public void clearPointsClick() {
   onView(withId(R.id.btClearPoints)).perform(click());
}
```

Unlike the map screen, this screen uses an AR fragment and the ARCore SDK. As there are three buttons on this screen we have four tests. The first test checks that the three buttons are present on the screen. The following three tests check that each of the buttons can be clicked. Each of these three tests return successful results.

ConfirmJourney Test

```
@Test
public void testStart() {
    assertNotNull(confirmJourney.btstartJourney);
    assertNotNull(confirmJourney.btCancel);
}
@Test
public void startButton() {

onView(ViewMatchers.withId(R.id.btstartJourney)).perform(click());
}
@Test
public void cancelButton() {
    onView(withId(R.id.btCancel)).perform(click());
}
```

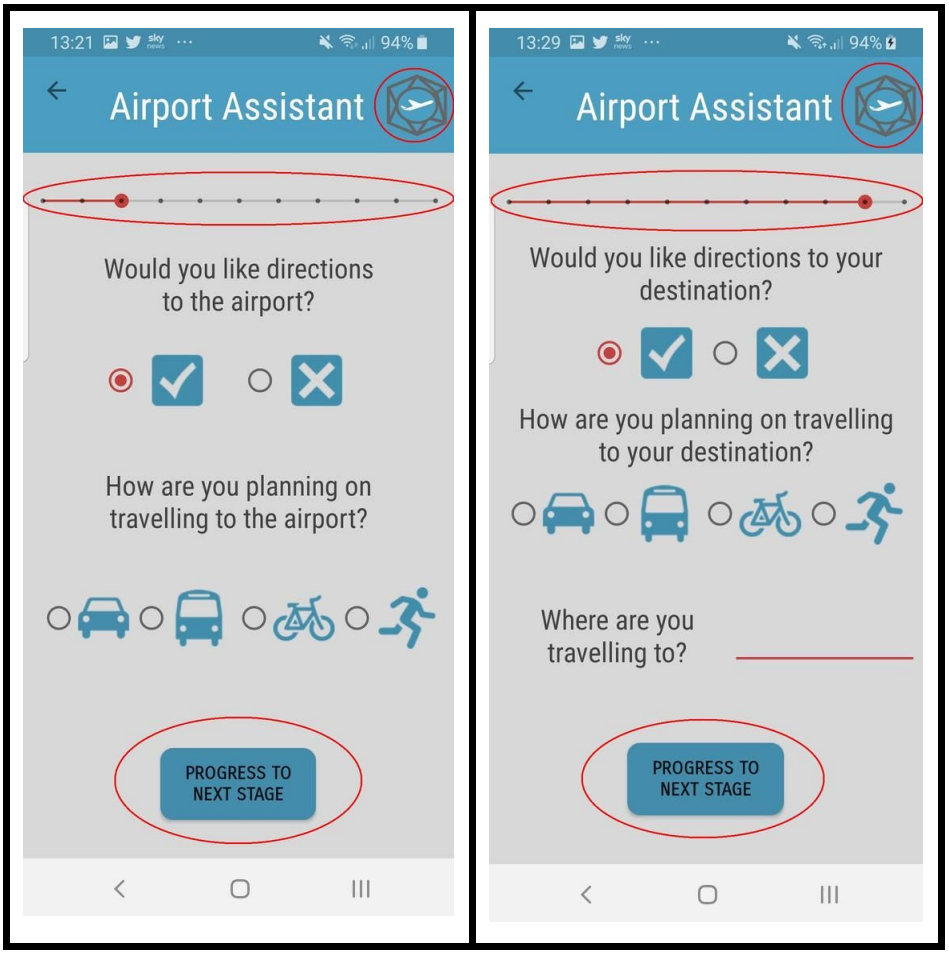


The Confirm Journey screen integrates web scraping with the Airport Assistant application. Here the journey information is displayed to allow the user to check that the details which have been scraped from their flight number are correct. As there are two buttons on this screen there are three tests. One to check that two buttons are present on the screen. The remaining two tests check that the two buttons can be clicked.

Heuristic Testing

Schneiderman's Eight Golden Rules

- **Strive for consistency**
  Buttons, Menus, Icons and User Flows must remain consistent throughout the application. This is to allow users to easily become familiar with the application. This can be seen in the "Progress to next stage" button, back button, progress bar and "Skip this stage" button. Similarly, workflows are designed in this manner. Functionality such as completing AR Scan is performed in the same manner throughout the application. This is also the case for selecting whether a user would like directions or not. Despite being presented at the very beginning and very end stages of the application. It is completed the same way.



- **Enable frequent users to use shortcuts**

  Allowing the user to skip certain stages that they do not deem relevant to them, causes them to pass through the application more swiftly. This also removes a vast amount of the effort involved in using the application as a user must not complete any stages they do not wish to take part in.

- **Offer informative feedback.**

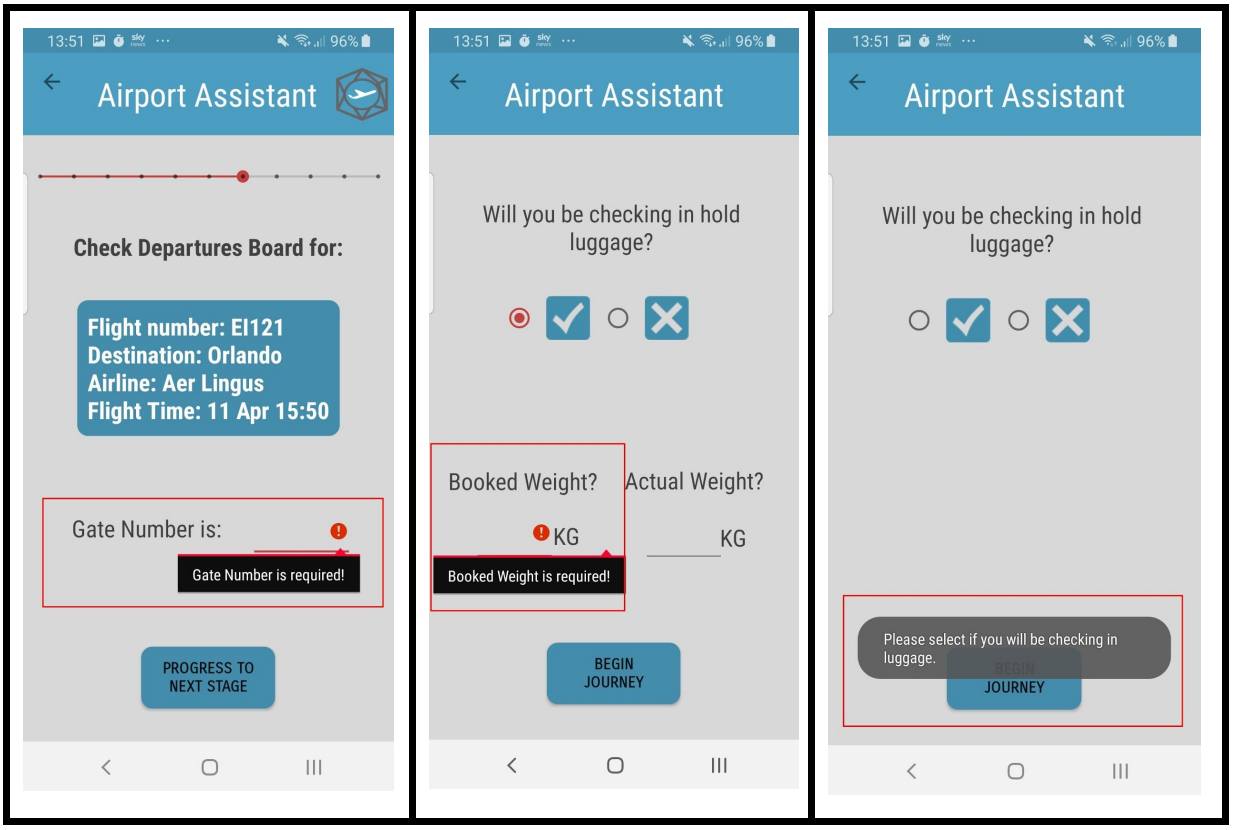
  Throughout their journey, users are presented a progress bar which indicates where is the process they are currently at. This keeps users informed about their progress and does not leave them in the dark at any stage.

- **Design dialogue to yield closure.**

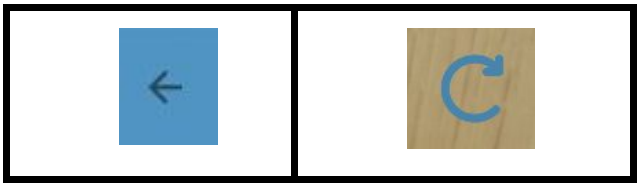
  Users are guided through the process stage by stage and must select themselves to complete a step and move onto the next. This keeps users in the know of exactly what their processes has yielded and exactly what stage they are at.

- **Offer simple error handling.**

  All text fields and radio buttons are flagged to ensure that a user inputs all required fields before progressing. This ensures that all required fields are entered, that no errors occur and that users are kept informed of what is going on in the application.

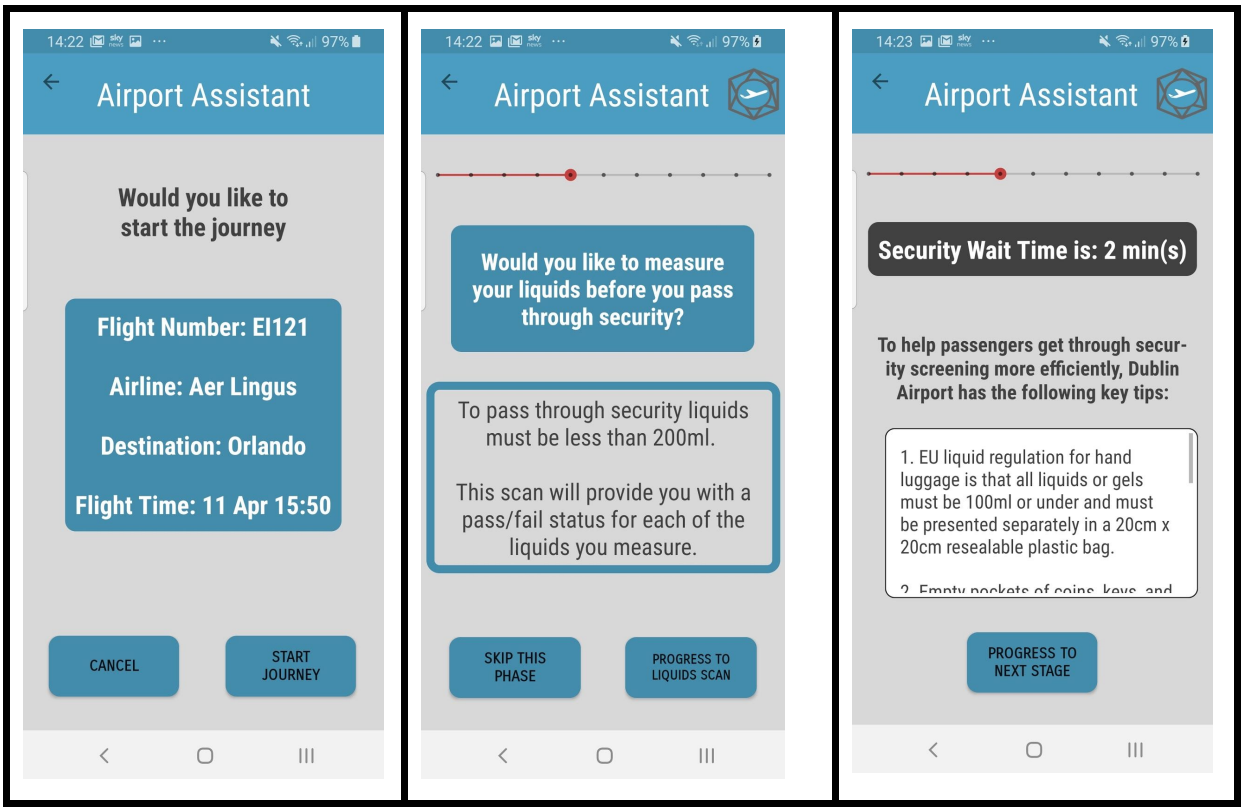


- **Permit easy reversal of actions.**
  Back buttons are a key feature of the application which allow users to return to a previous stage if they so wish. Similarly, on the AR Screens, users are provided with the ability to click the refresh button which wipes all the points the user has just placed and allows them to map out new points.



- **Support internal locus of control.**
  Users are given control of which processes they wish to complete throughout the application. They can decide to complete or skip a phase. This allows users to feel that they are in control of their own journey.

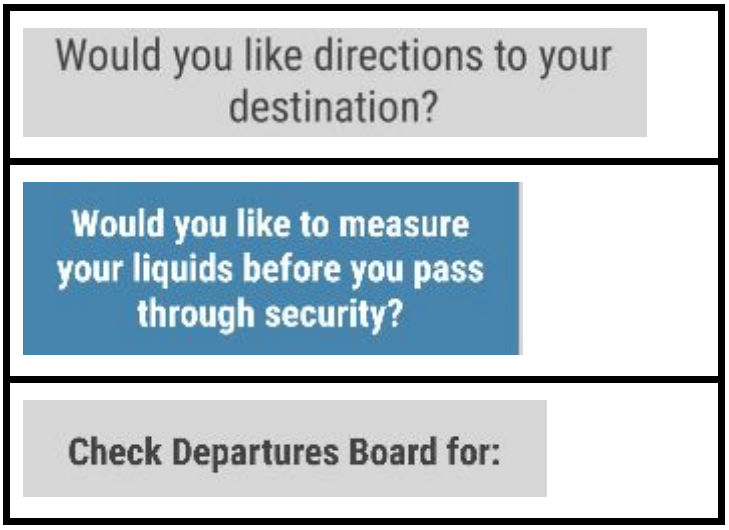
- **Reduce short-term memory load.**
  Interfaces throughout the application are simple and easy to use. They are intuitive rather than expecting users to learn off how to use the application. Users are not overloaded with information at any stage during the application.



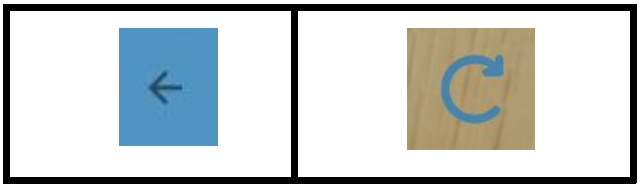
Nielsen's Heuristics

- **Visibility of system status**
  As users themselves are given the power to make use of or skip a certain stage, they are in control of the process. This, alongside the use of a progress bar, allow users to see what stage they are at in the application and keeps them informed throughout.

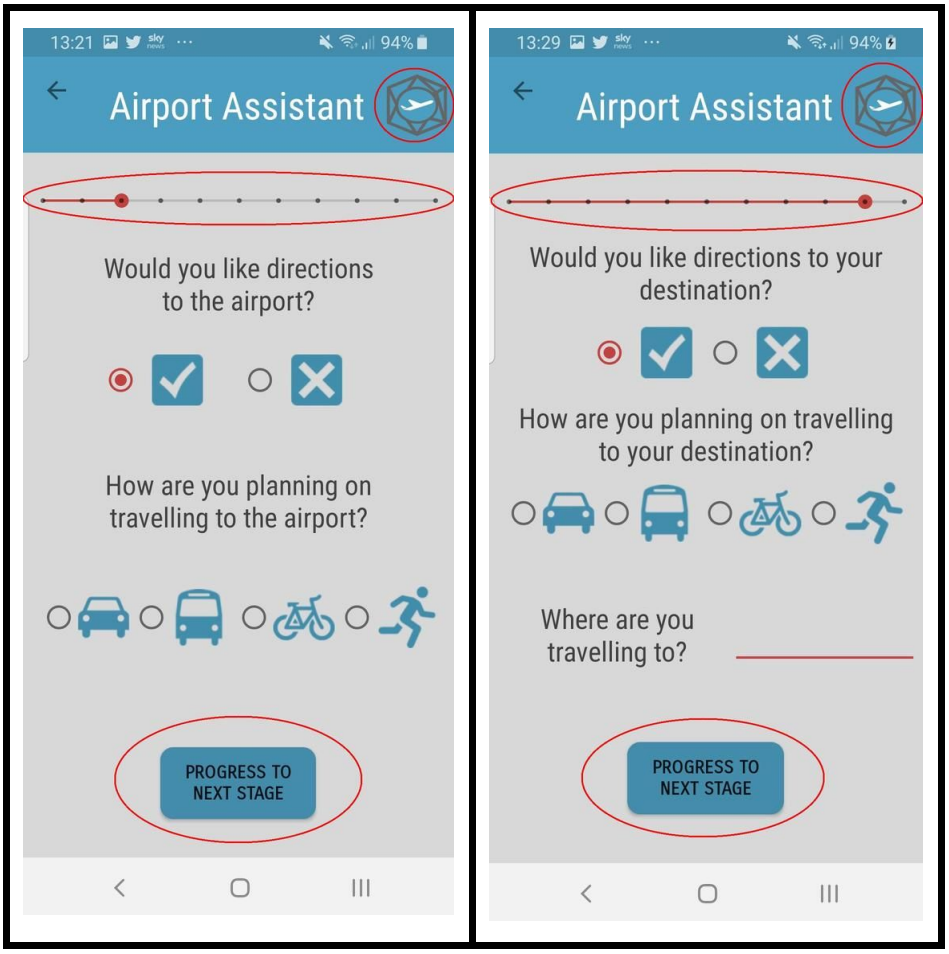
- **Match between system and the real world**
  Language used throughout the application is simple and easily understood. No technical terms are used, instead the language of the user is used. This is very important in areas such as the AR guide. This often difficult to understand concept is explained in plain terms which makes it more appealing to the users.



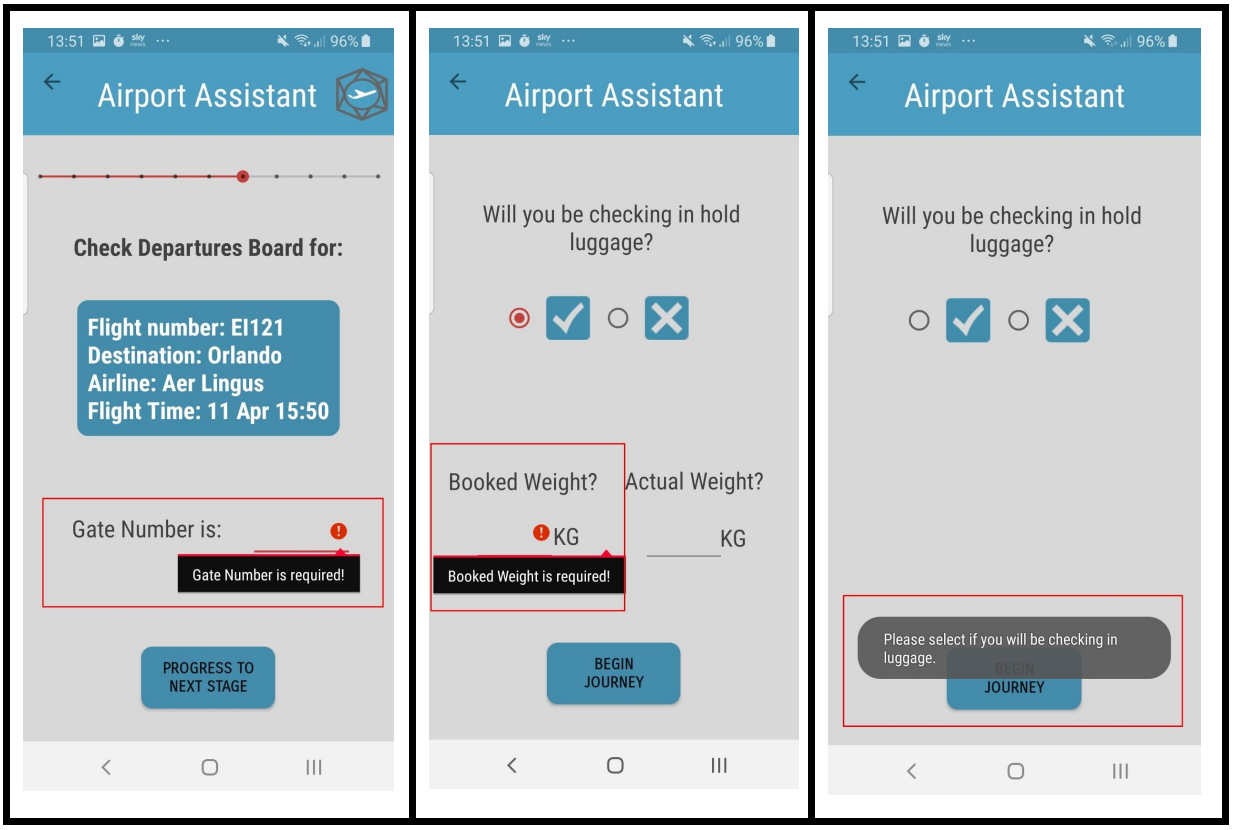
- **User control and freedom**
  Back buttons are prevalent throughout the application. This allows users to reverse actions or exit screens if they wish. Similarly, on the AR Screens, users are provided with the ability to click the refresh button which wipes all the points the user has just placed and allows them to map out new points.



- **Consistency and standards**
  Icons, buttons, menus and processes are consistent in all stages of the application. Due to this users become familiar with the application yet do not have to know all about the application due to its intuitiveness. These consistencies include back buttons, progress buttons, progress bar and language used.



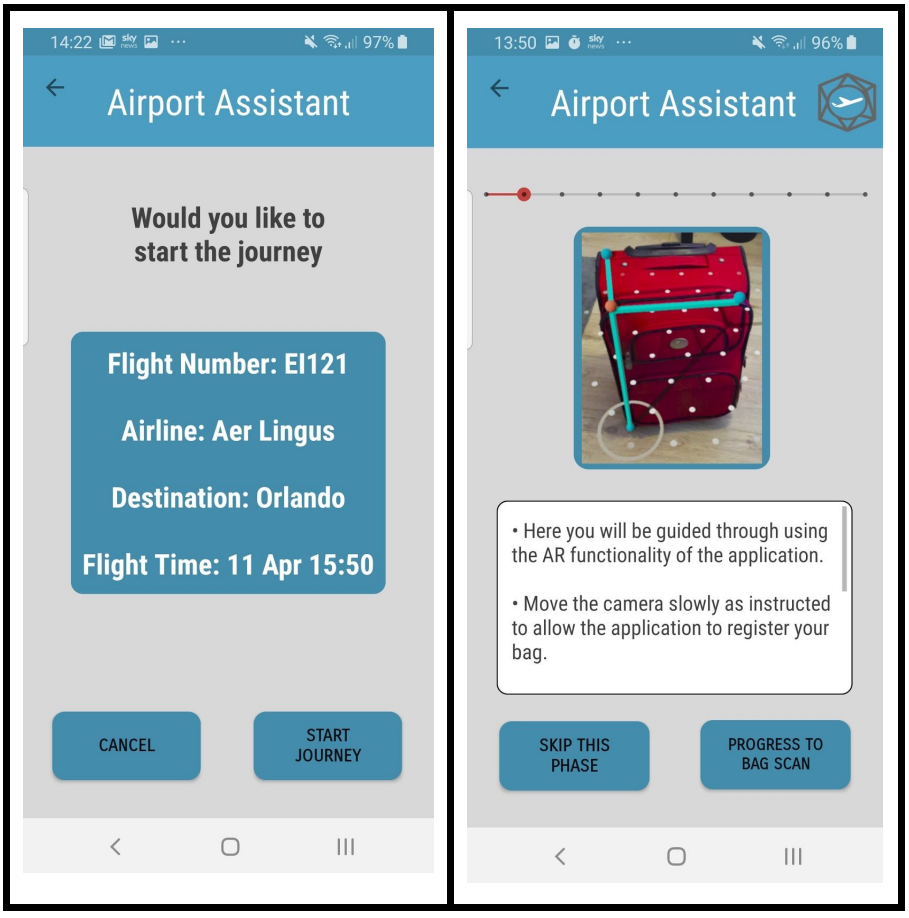
- **Error prevention**
  Radio buttons and text fields throughout the application are flagged with warnings that alert users that a text field or a radio option or a text field respectively, have not been filled. Items such as this allow errors to be prevented, users to be kept informed and all fields to be entered.



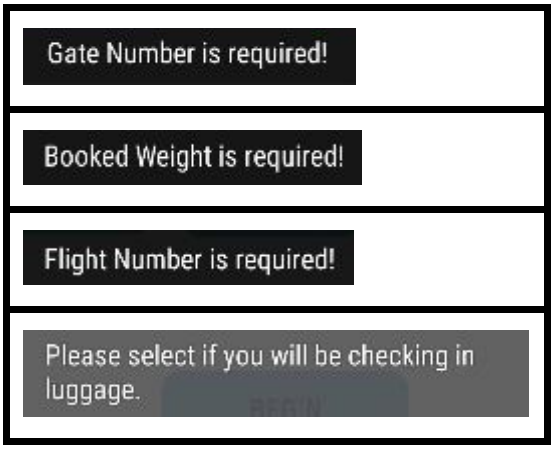
- **Recognition rather than recall**
  Consistency throughout the application makes it easier for users to become familiar with the application rather than learning exactly how the application works.

- **Flexibility and efficiency of use**
  As users decide what phases they wish to complete, they can skip any stages they do not wish to take part in. This means that the time necessary to pass through the application is dependant on the functionality the user wishes to make use of.

- **Aesthetic and minimalist design**
  Only necessary information is provided to the user. The user is not overloaded or expected to deal with vast amounts of data at one time. They are simply informed on what they need to know.



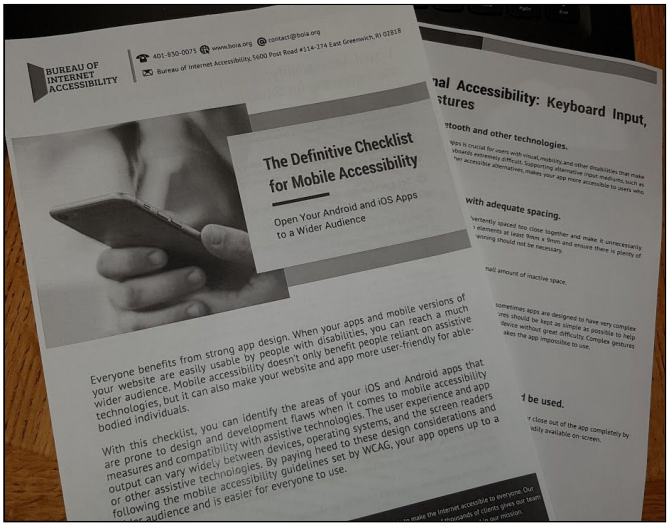
- **Help users recognize, diagnose, and recover from errors**
  Error messages throughout the application are written in plain simple english to make them easy for users to understand and follow. They also clearly explain possible solutions for the user.



- **Help and documentation**
  User documentation that has been written using IEEE User Documentation Standards will be provided to users.

Accessibility Testing

The Definitive Checklist for Mobile Accessibility



To complete my user testing I used “The Definitive Checklist for Mobile Accessibility” which was written by the Bureau of Internet Accessibility. This guide has a number of sections which include two major sections “Visual Accessibility” and “Tactile and Operational Accessibility”.

The “Visual Accessibility” checks are:

Designing for small screens	
Minimise the amount of information that is put on each page	Pass
Provide a reasonable default size for content	Pass
Position form fields below their labels	Pass

Enable Zooming within the app	
Magnify entire screen	Fail
Resize text 200% without using assistive technology	Fail
Ensure that the browser pinch zoom is not blocked	N/A
Provide on-page controls to change the text size	Fail
Contrast	
Minimum contrast ratio should be 4:5:1 and 3:1 for large text	Pass

The “Tactile and Operational Accessibility” checks are:

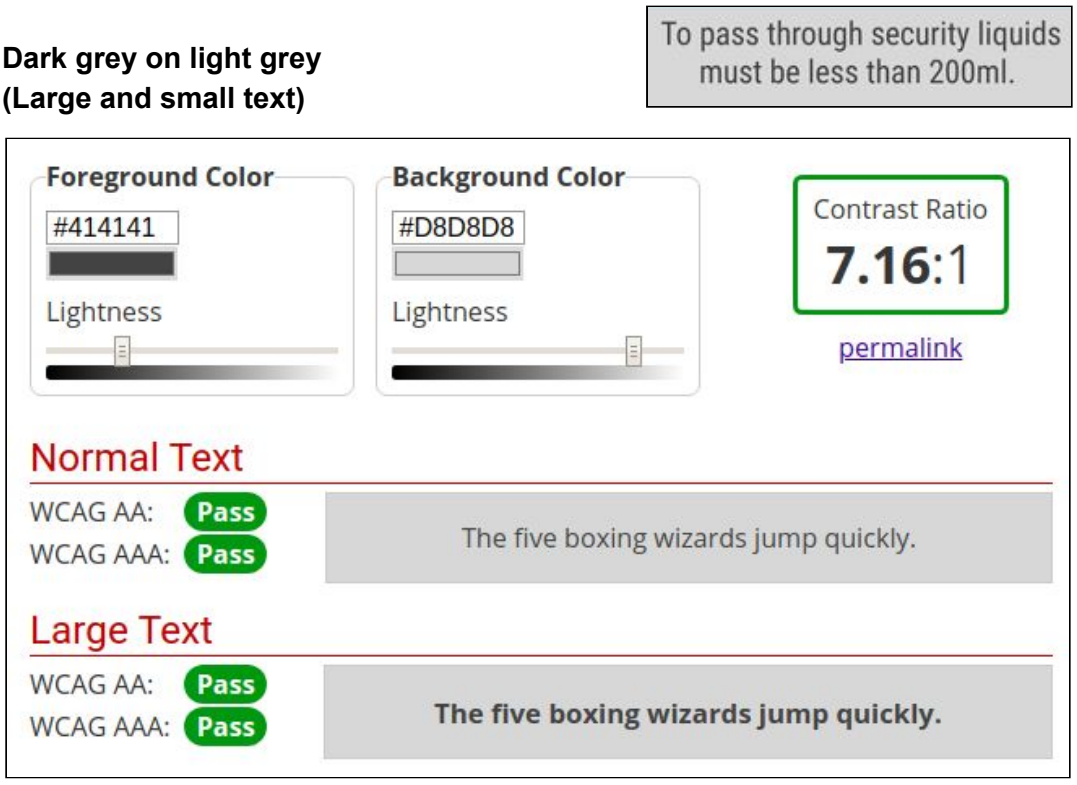
Keyboard Support	
Keyboard control for touchscreen devices	Pass
Touch targets	
Confirm that touch targets are 9mm high by 9mm wide	Pass
Confirm that touch targets close to minimum size are surrounded by a small amount of inactive space	Pass
Gestures	
Touchscreen gestures are simple to implement	Pass
Buttons are easy to access	Pass
On-Screen indicators	
Make sure page layout is consistent	Pass
Important page elements are positioned before the page scroll	Pass
Group actionable elements that perform the same action together	Pass
Provide clear indication that elements are actionable	Pass
Provide instructions for actions that require complex interaction	Pass
Set the virtual keyboard to the type of data entry required	Pass
Provide easy methods for data entry	Pass
Support the characteristic properties of the platform	Pass

After completing this testing I can see that there are a number of areas where I could further develop the accessibility of my application. These areas primarily focus on zooming. I will now look into developing these areas before I finish the development of my application.

Colour Contrast Checks

As part of the above accessibility testing I completed a number of colour contrast checks. Throughout these tests I had to ensure that the colour contrast ratio was above 4.5:1 for small text and 3:1 for large text.

Dark grey on light grey
(Large and small text)



Light blue on white
(Large text only)

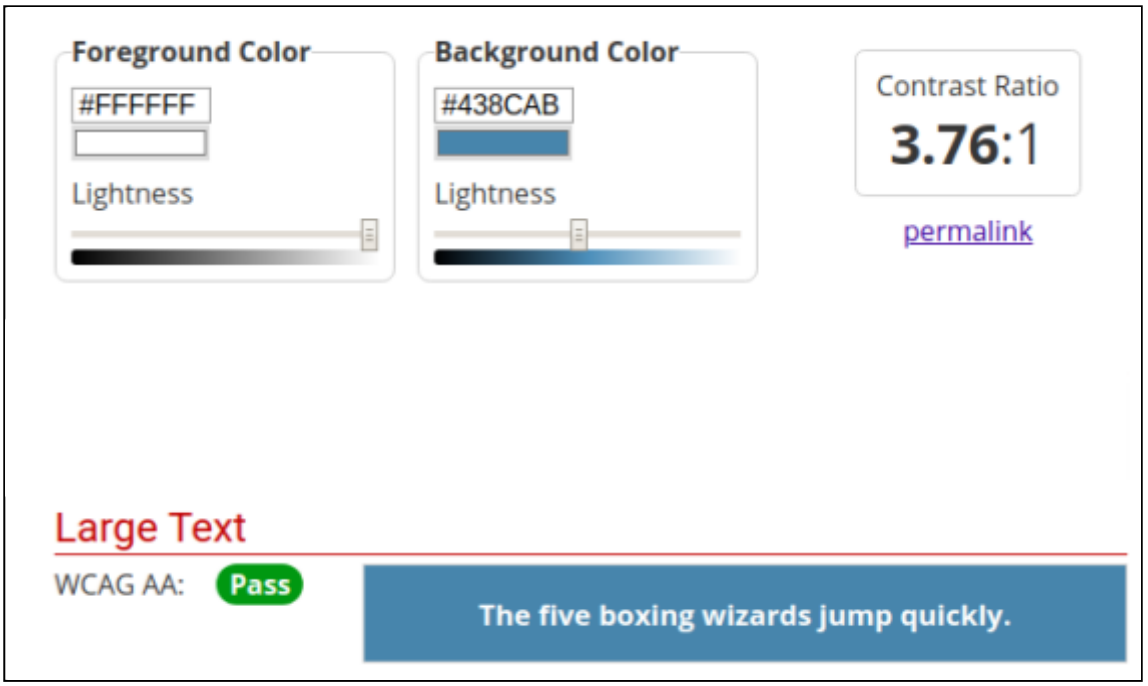
AIRPORT

Black on light blue
(Large and small text)

PROGRESS TO NEXT STAGE

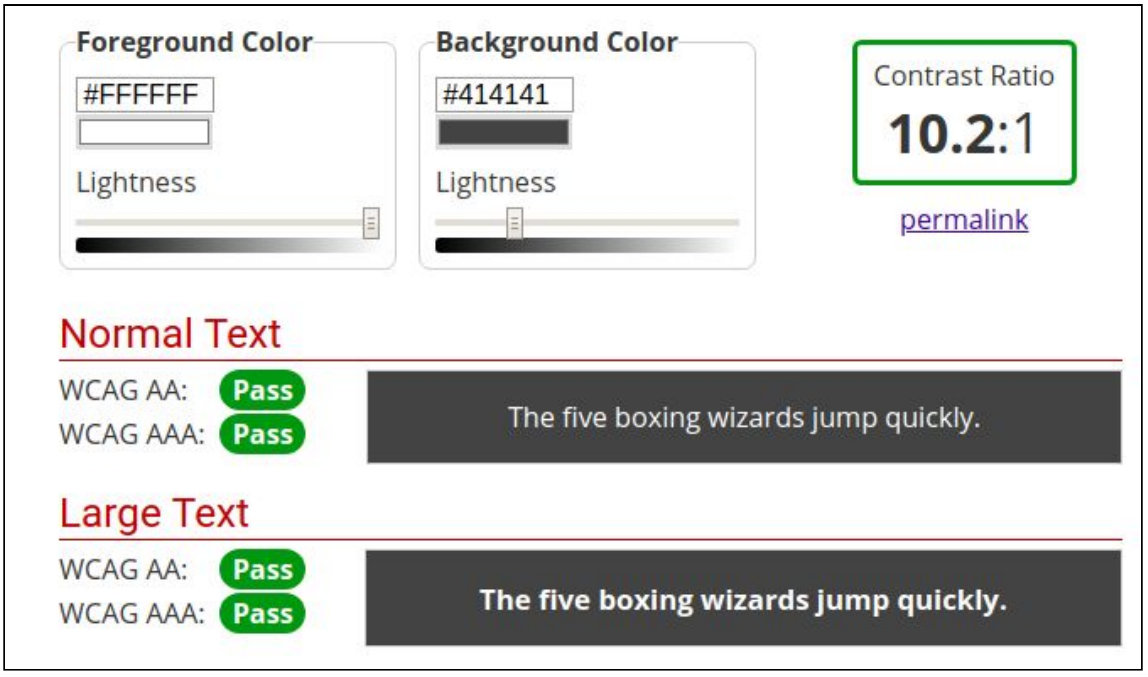
White on light blue
(Large text only)

Flight Number?



White on dark grey
(Large and small text)

Security Wait Time is: 15 min(s)

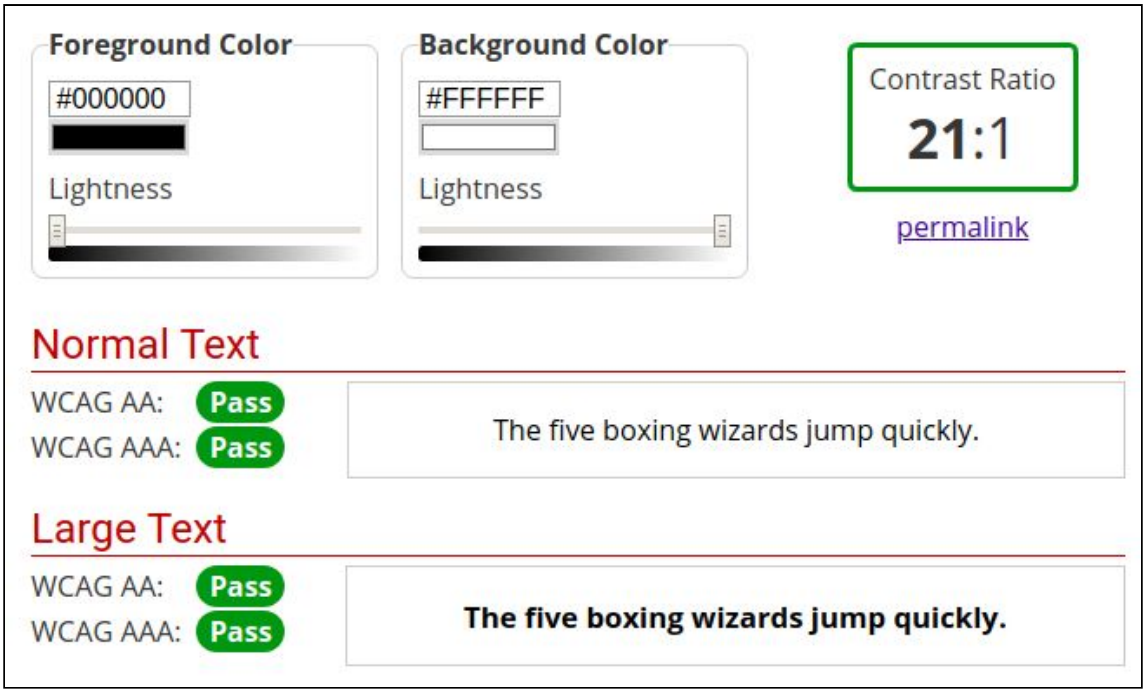


Light grey on light blue
(Large text only)

https://www.aerlingus.com

Black text on white background
(Small and large text)

Distance: 16.6 km



Hearing Difficulties

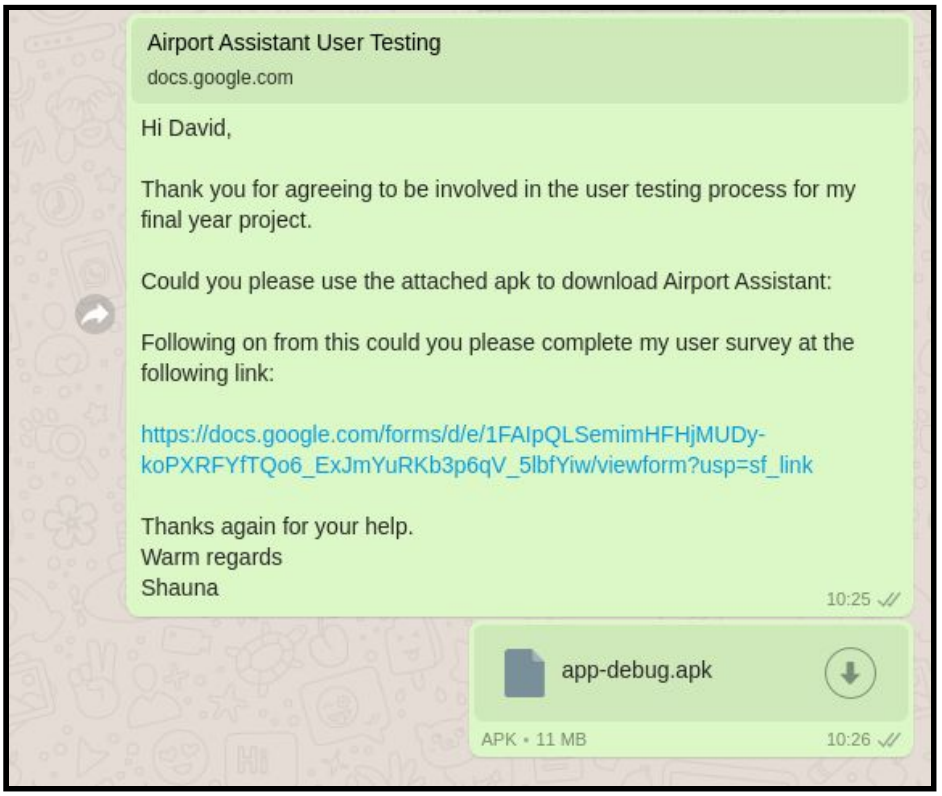
As there no audio within the application their will be no difficulties present for users with hearing difficulties.

Motor Skills

Buttons throughout the application are large and easy for users to use. Items which are to be used in sequence are also placed close together which benefits these users.

User Testing

To complete my user testing I decided to send my apk to a number of participants and then ask them to take part in a user survey. I attempted to send out emails for this purpose but unfortunately Gmail does not allow the sending of .apk files. I sent out Whatsapp messages to a number of people with varying levels of technical experience. Here is an example of the message which was sent to participants

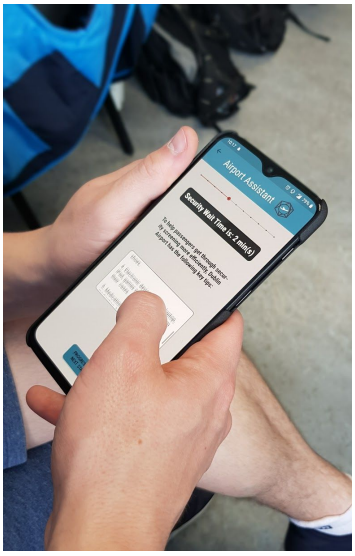
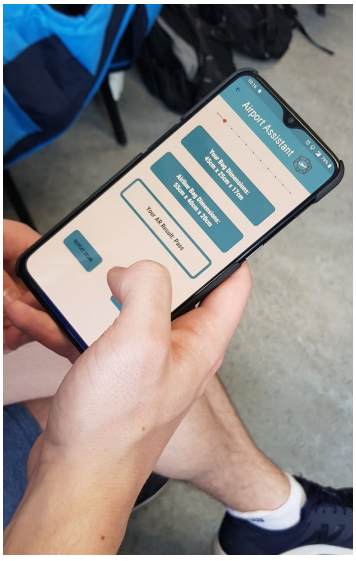
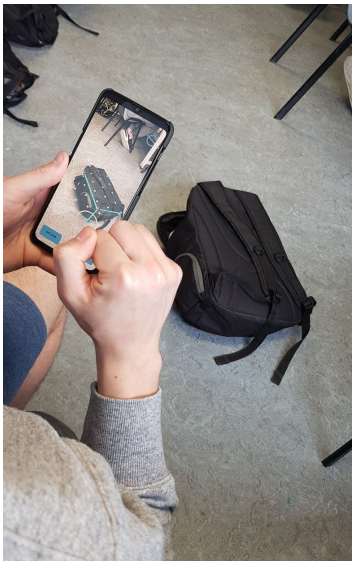


Users could then test the application wherever they wished. A number of these tests were carried out by participants who could access the DCU computing building so that further user testing could be carried out on these participants. This testing including timing the users using the application.

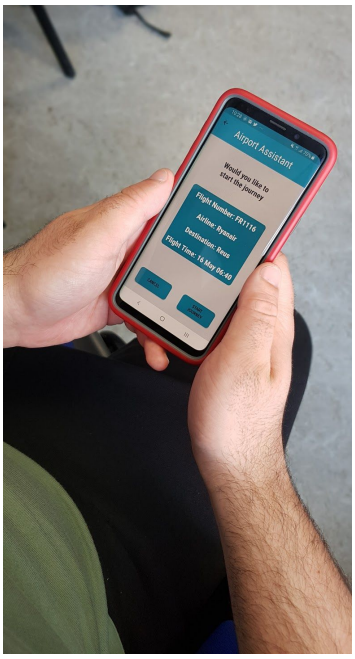
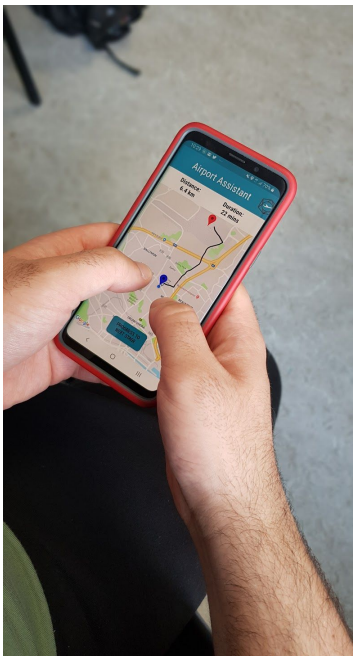
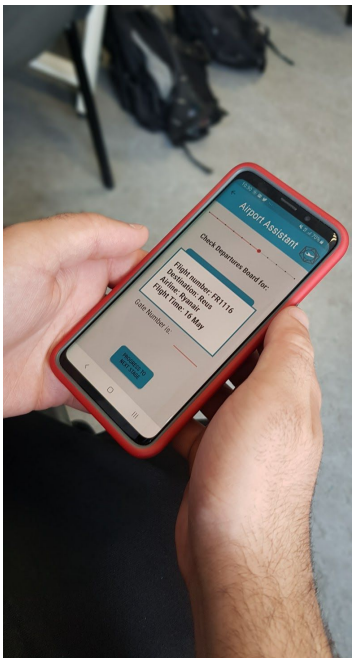
As this was carried out on the users own device, the application was tested on a number of different Android devices including a Samsung S8 and S9, a Motorola G6 and a OnePlus 6T. Any users who had agreed to take part in testing but were iPhone users or did not have the adequate level of Android completed the testing using my device.

Here we can see these tests being carried out in the DCU Computing labs on the users' own devices.

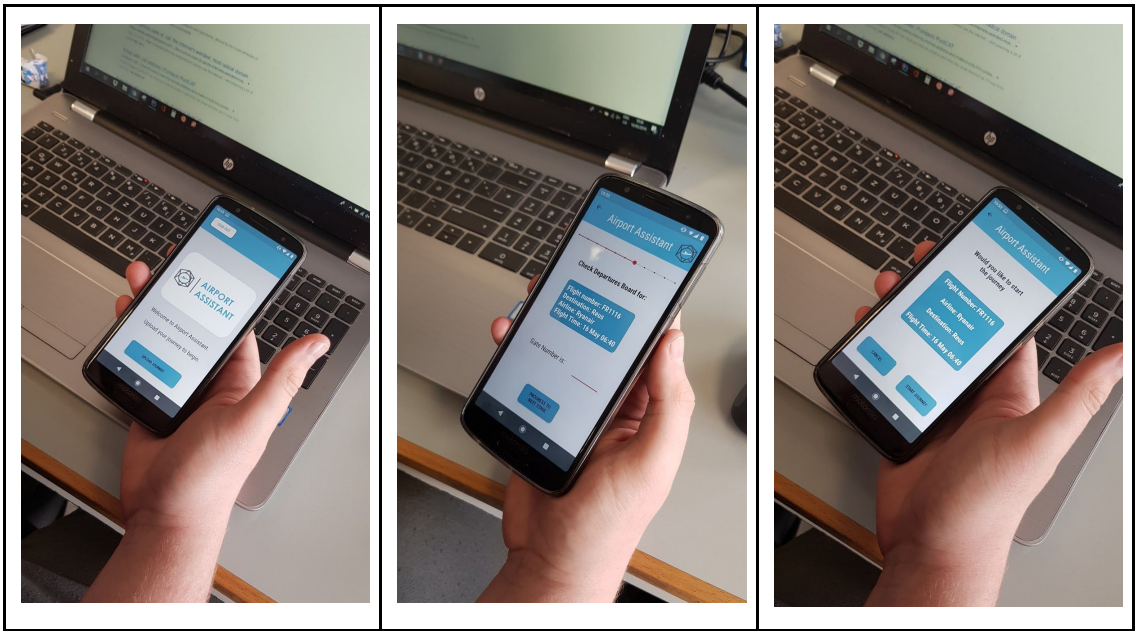
David using a OnePlus 6T

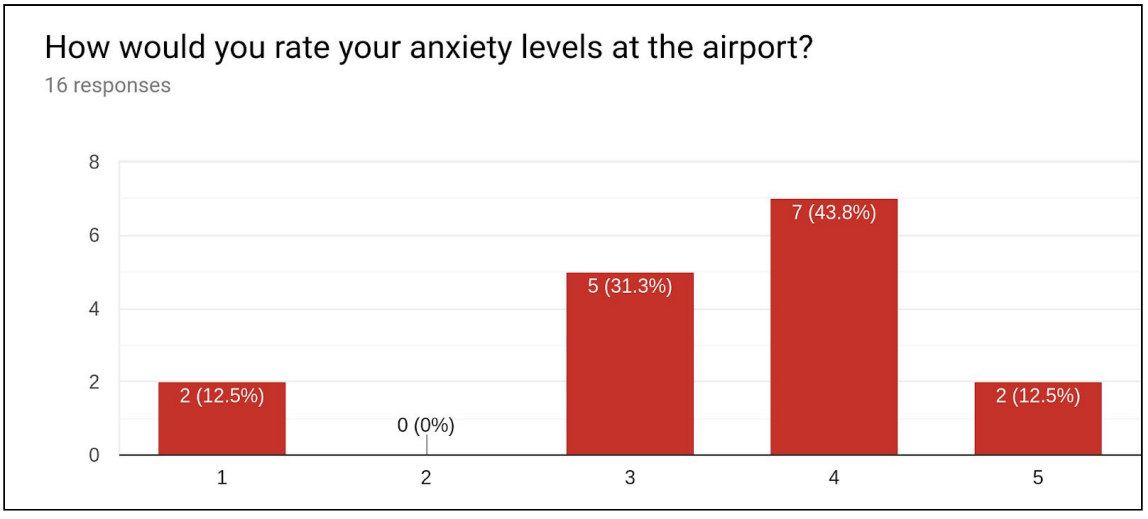
Conor using a Samsung S9

Sean using a Motorola G6

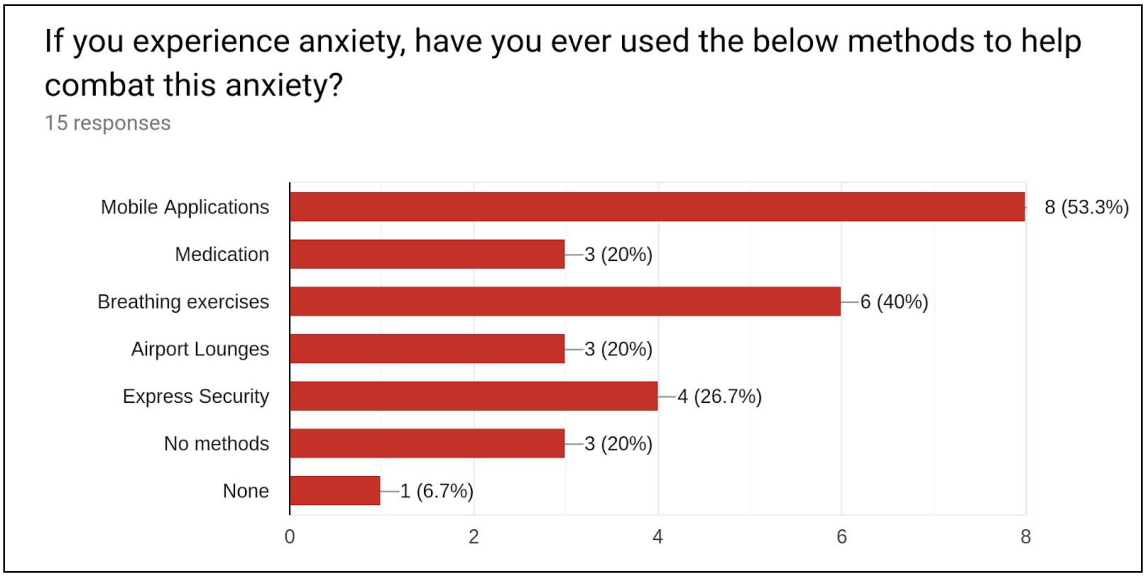


Following on from using the application participants were asked to complete a survey using Google Forms. Here they were asked a number of questions related to the application including general questions about their experience with anxiety in the airport and how often they travel. The remainder of the survey is concerned with how the user found using the application. The results of this survey are as follows:

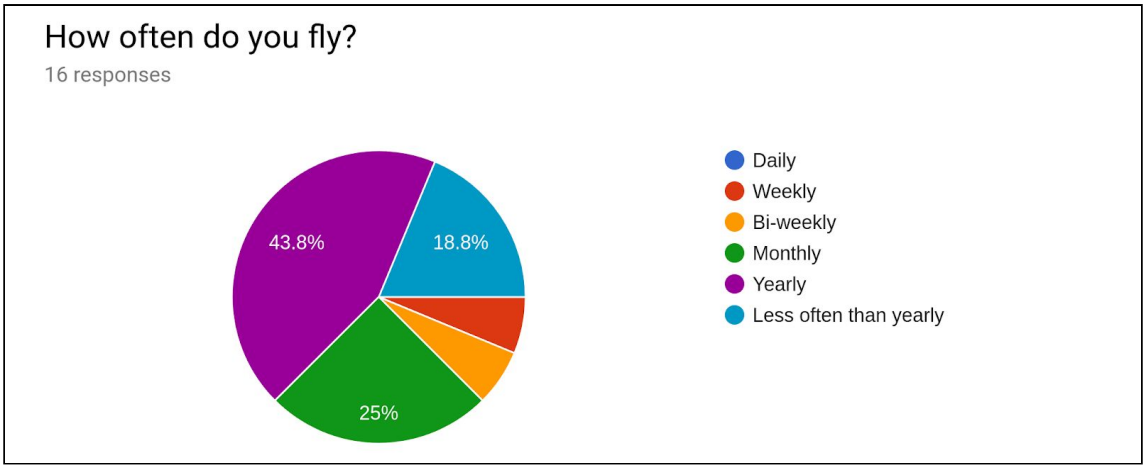


The survey began by users being asked how would rate their anxiety levels at the airport on a scale from one to five. This allowed me to gather some information on the users taking part in the survey and some context to their results. If a user said later that they would not use the application, I could return and check if they feel anxiety in the

airport. This also provided some validation for my idea as the vast majority of the users who took part in the survey ranked their anxiety in the region of three to five.

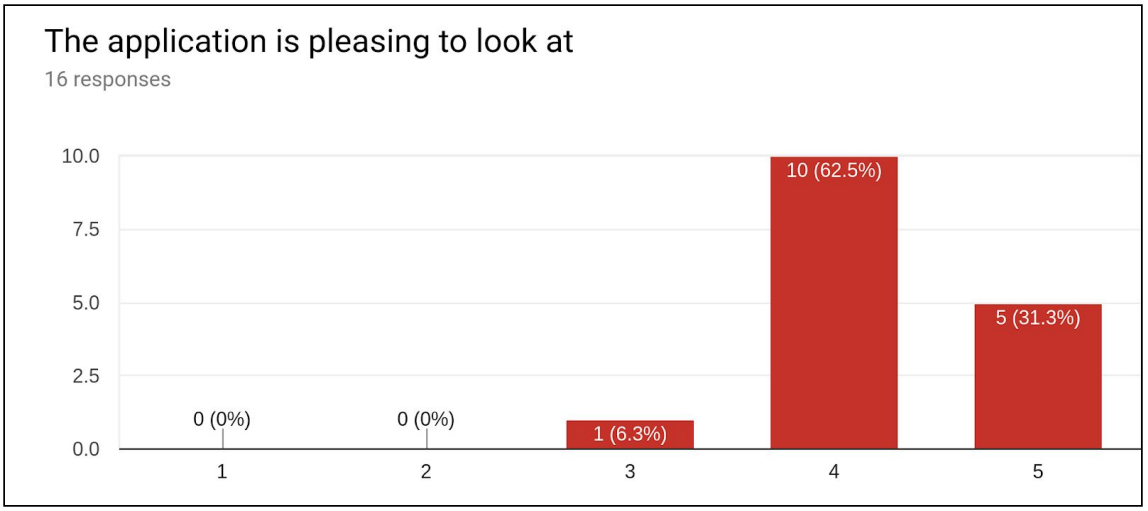


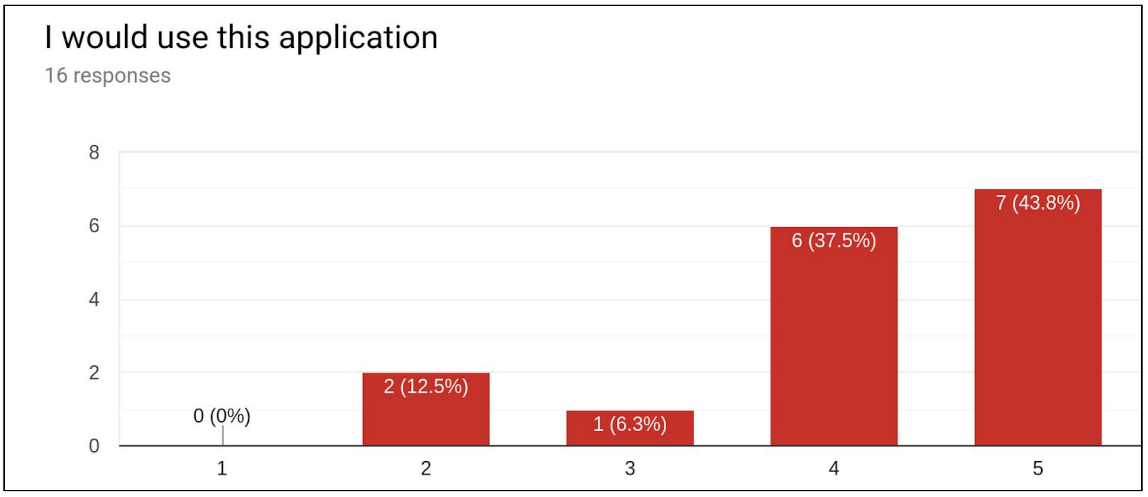
This question was asked to gather some information on how users are currently combatting this stress in airports. Over 50% of those surveyed are already making use of mobile applications which allowed me to see that there would be a place for this application in their current routine. I was also quite surprised that 20% of the applicants make use of medication or airport add ons such as the use of lounges or express security.

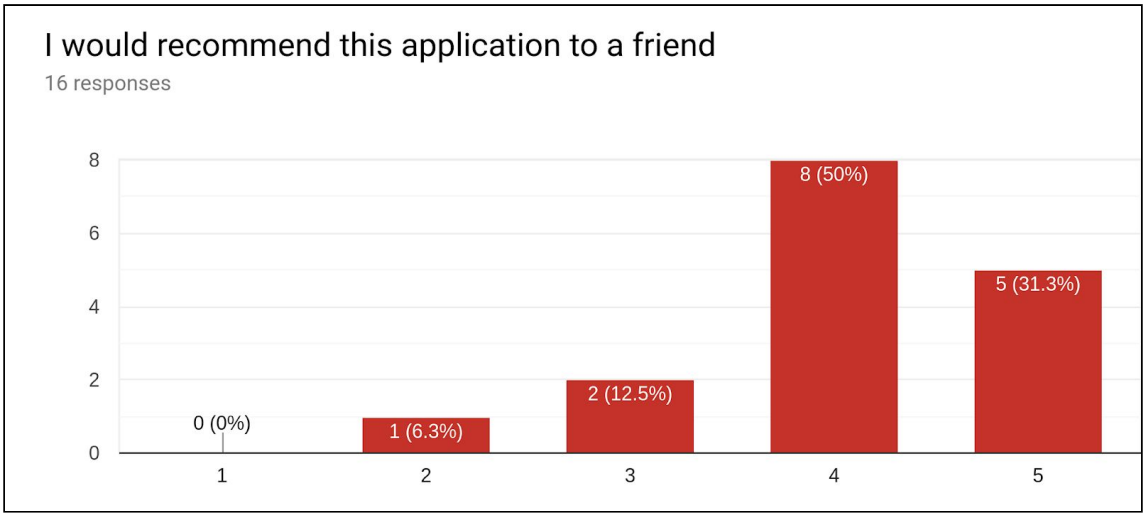


This was the last of the questions which related directly to the user. I wanted to gather information on how often the people I am surveying travel. Unsurprisingly, the majority of them fly yearly.

I then began asking a number of questions about the application.

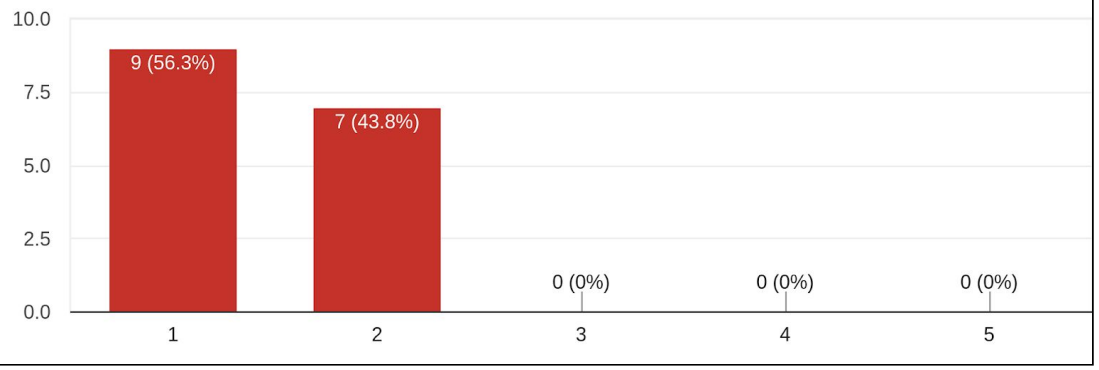






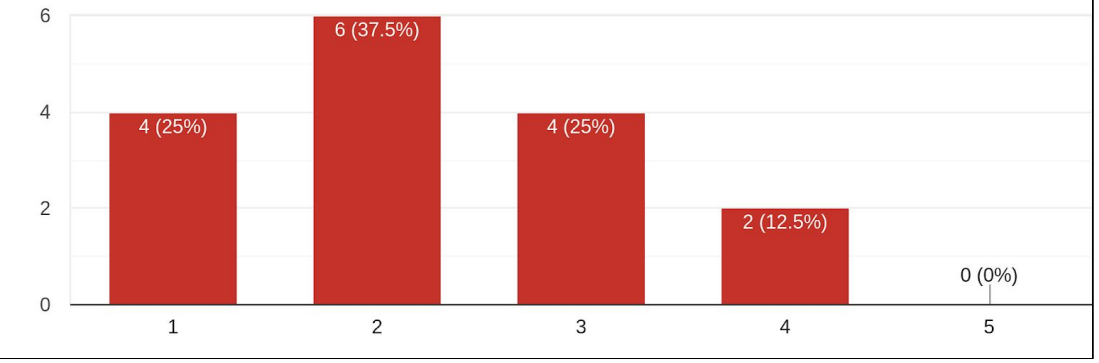
I found this application frustrating to use

16 responses



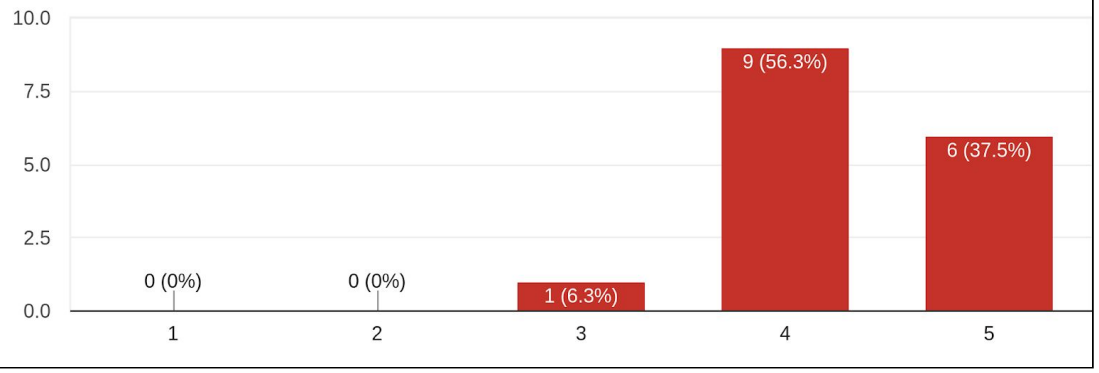
This application could be improved

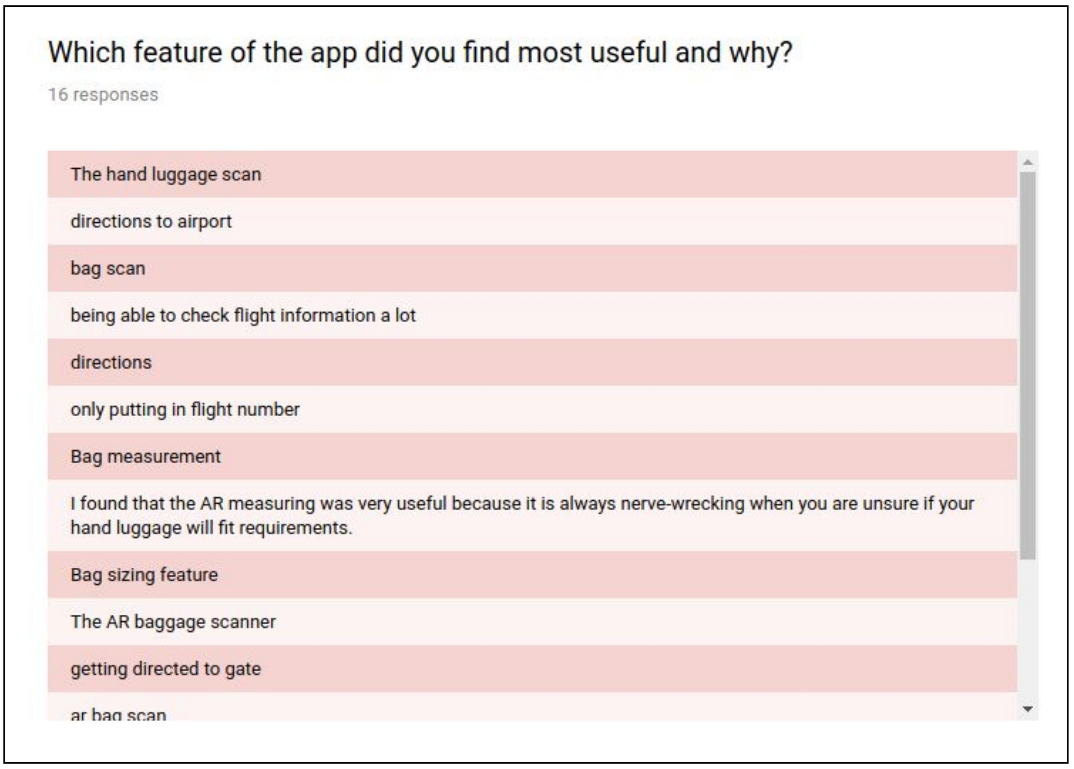
16 responses



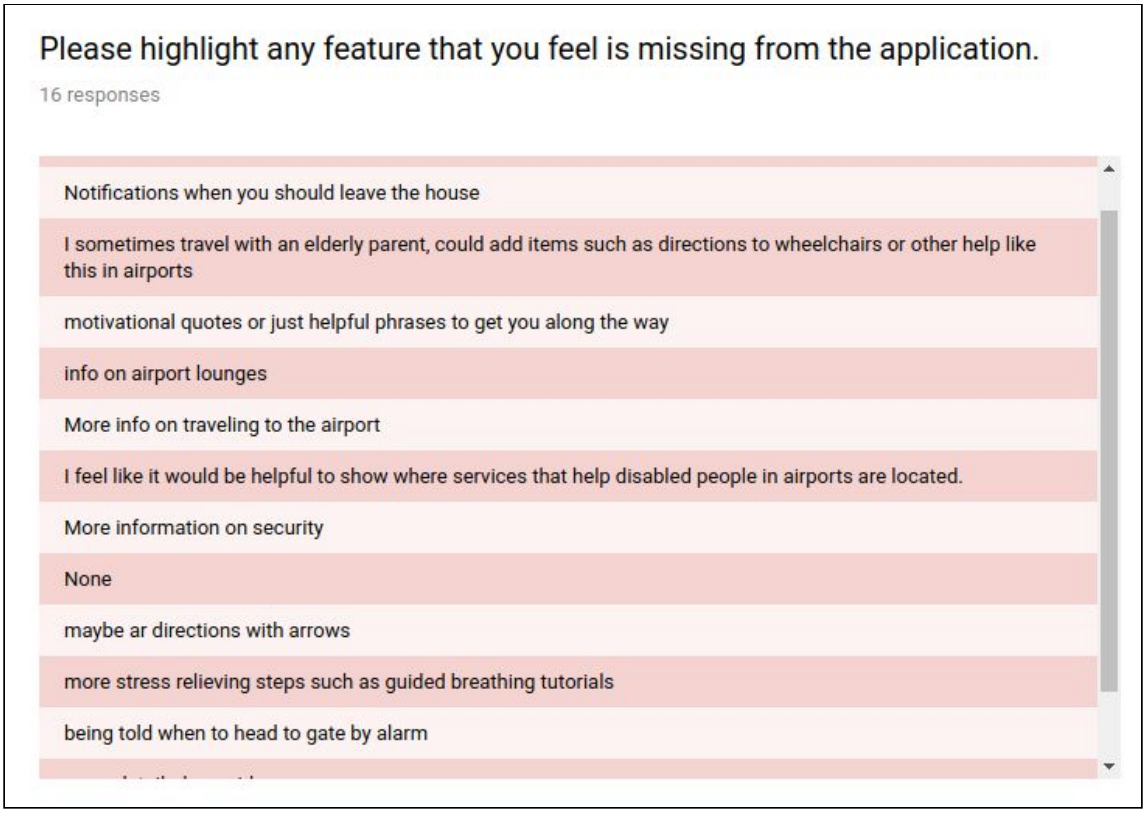
I was happy with the pace I learned to use the application

16 responses

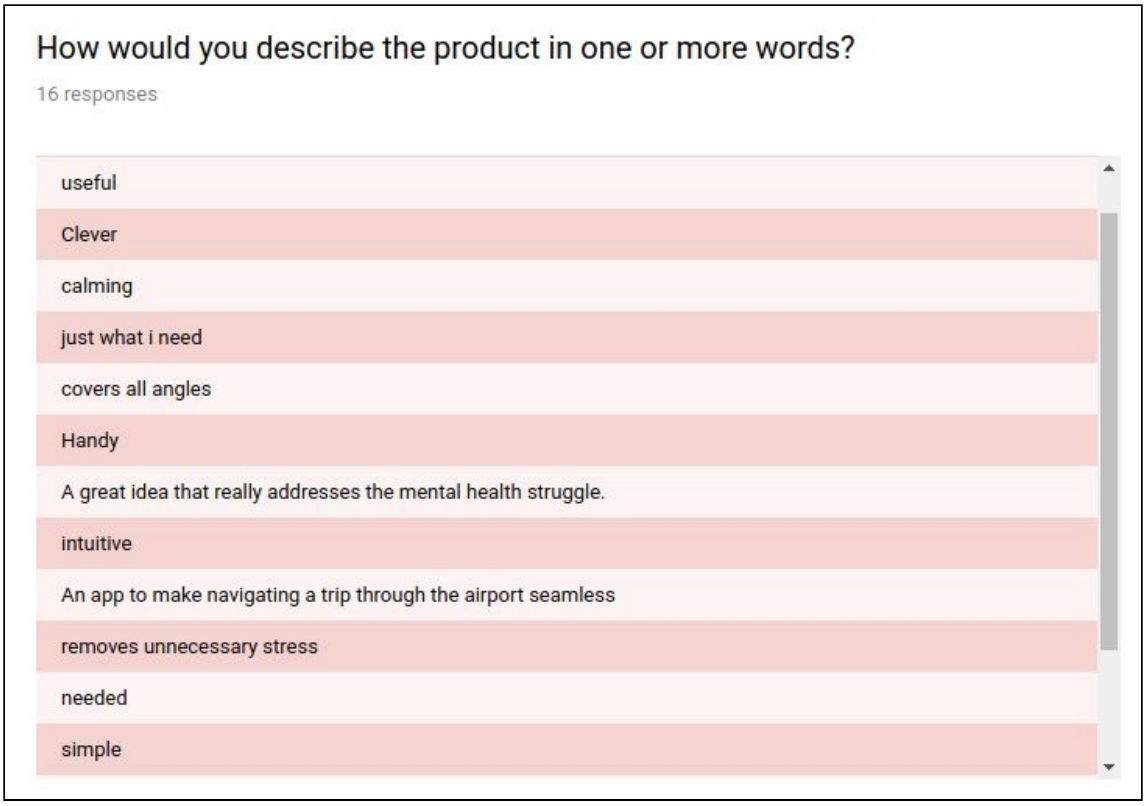


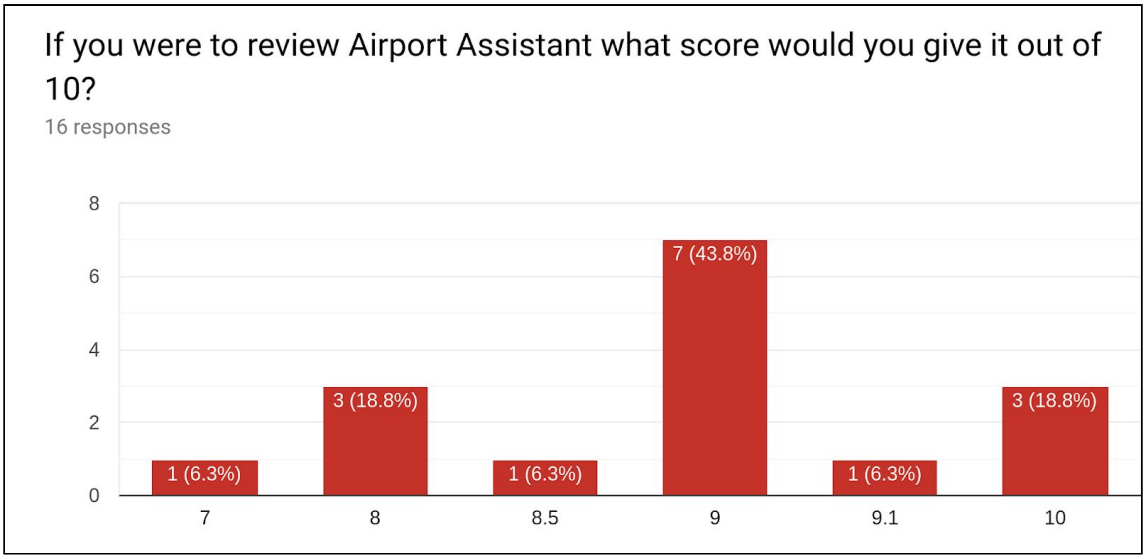


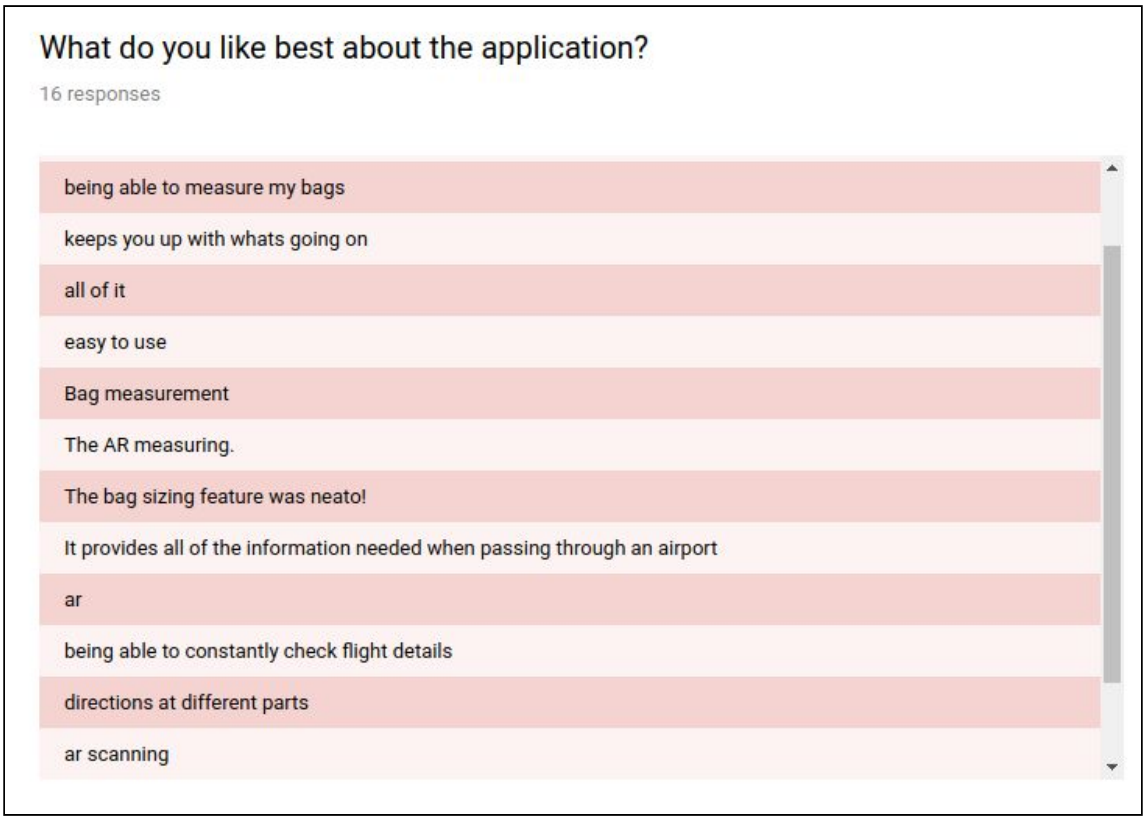
As I expected, the feature which users found most useful was the AR functionality of the application, primarily the baggage scanning. A number of users also brought up the directions functionality and the ability to check their flight information frequently.



I also wanted to see what future work users would like to see completed for airport assistant. These primarily included adding additional stress relieving functionality, notifications alerting users when to begin travelling either to their gate or to the airport and specific information on certain aspects of the airport.







Timed User Tests

Users who completed the testing in the Computing labs were asked to complete timed tests. This involved users being timed while they were using the application so that it can be determine how the found using the application without asking them.

The following table shows the results of testing:

Instructions	***Time Needed***	***T***	***i***	***m***	***e***	***T***	***a***	***k***	***e***	***n***
	(sec)	***u1***	***u2***	***u3***	***u4***	***u5***	***u6***	***u7***	***u8***	***u9***
Login to the application	**10**	8	7	7	11	8	8	11	8	10
Logout of the application	**5**	4	2	3	4	6	5	3	4	3
Upload a journey	**10**	7	9	7	8	11	7	9	8	8

Enter check-in luggage values	**12**	10	11	9	11	13	10	9	12	11
Do not check-in luggage	**5**	4	3	6	4	3	3	4	5	7
Complete AR functionality to scan hand luggage as a whole	**50**	35	59	32	47	38	36	52	46	40
Skip AR functionality	**10**	7	5	4	8	9	11	6	7	8
Quit AR functionality during scan	**15**	11	8	10	13	12	17	14	15	11
Repeat AR Scan	**5**	3	4	3	4	4	6	5	3	4
Get directions to airport	**20**	13	11	12	18	17	20	23	16	14
Skip directions to airport	**10**	8	6	7	9	8	11	7	12	6
Complete check-in phase	**10**	9	7	8	8	7	11	5	9	10
Complete security phase	**12**	11	11	9	10	8	13	8	9	12
Complete duty free stage	**20**	17	20	18	17	15	21	19	17	18

Enter gate number	**10**	8	9	8	10	8	9	12	9	11
Get directions to gate	**15**	13	12	14	13	17	13	12	15	14
Complete in-flight stage	**20**	17	18	16	17	18	22	19	20	18
Enter destination, and click get directions to destination	**30**	26	27	28	25	22	34	29	28	29
Skip directions to destination	**10**	5	4	6	7	3	8	7	9	11

From this we can see that most people completed the tasks in the amount of time that was required to complete them with only a small number of outliers. This allowed me to see that users did find the application intuitive and were able to use the application with ease.