2019-2020 FYP Final Report

Title: Museum Experience Application

Name: John Cummins

ID: 16373731

Supervisor: Karen Young

Code: KY2

Table of Contents

TABLE OF FIGURES **2**

1. INTRODUCTION **3**

1.1. BACKGROUND RESEARCH 3

1.2. DOCUMENT OUTLINE 4

2. REQUIREMENTS **5**

2.1. FUNCTIONAL 5

2.2. NON-FUNCTIONAL 7

3. DESIGN **8**

3.1. THREE TIER CLIENT/SERVER ARCHITECTURE 8

3.2. TECHNOLOGIES 9

3.3. CHOOSING A MUSEUM 12

3.4. CHOOSING ARTEFACTS 12

3.5. PRESENTATION LAYER DESIGN: 14

3.6. APPLICATION 16

4. IMPLEMENTATION **17**

4.1. CREATING A PROJECT IN ANDROID STUDIO 17

4.2. ON CREATE METHOD 17

4.3. PROJECT FILES 18

4.4. IMAGE DETECTION 20

4.5. 3D MODELLING 21

4.6. SCENEFORM 23

4.7. ADDING MODELS TO THE SCENE 25

4.8. TRIVIA SECTION 27

4.9. ADDING VIDEO CONTENT: 29

4.10. ADDING AUDIO CONTENT: 30

4.11. OTHER LAYOUTS 31

5. TESTING **33**

5.1. USER TESTING 33

5.2. SYSTEM TESTING 36

6. PROJECT MANAGEMENT **38**

6.1. GANTT CHART 38

6.2. CODE MANAGEMENT 38

6.3. COMMUNICATIONS 39

7. CHALLENGES **40**

7.1. PROJECT MANAGEMENT 40

7.2. 3D MODELLING ISSUES 40

7.3. ANDROID EMULATOR ISSUES 41

8. FUTURE WORK **42**

9. ACKNOWLEDGMENTS **43**

10. REFERENCES **44**

Table of Figures

Figure 1: Use Case Diagram 6
Figure 2: High Level Project Overview 8
Figure 3: Screenshot of the Firebase Console 11
Figure 4: Photograph of Devon Chafing Dish 12
Figure 5: Photograph of Proclamation 12
Figure 6: Photograph of Eamon De Valera's head 13
Figure 7: Photograph of post box 13
Figure 9: Screenshot of actual home screen 14
Figure 8: Sketch of the home screen 14
Figure 11: Screenshot of Actual AR screen 15
Figure 10: Sketch of AR screen 15
Figure 12: Screenshot of Actual AR screen 16
Figure 13: Choice of Android activities 17
Figure 14: OnCreate method template 17
Figure 15: AndroidManifest.xml 18
Figure 16: Project build.gradle 19
Figure 17: App build.gradle 19
Figure 18: Image list file 20
Figure 19: Screenshot from running the Image quality tool 20
Figure 21: The finished product of the bowl 3D model 21
Figure 20: Creating the bowl 3D model on Smoothie 3D 21
Figure 22: The post box 3D model on the TurboSquid website 22
Figure 23: Code snippet from the ARFragment Java class 23
Figure 24: AugmenetedImgDb function 23
Figure 25: OnUpdateFrame function in the DisplayARCamera class 24
Figure 26 CreateModel function called in the post box if statement 25
Figure 27: CreateModel Function in the DisplayARCamera class 25
Figure 28: setLocationRotation function used to rotate the models 26
Figure 30: Screenshot of the postbox after the button is pressed 26
Figure 29 Screenshot of when the artefact is first detected 26
Figure 31: Screenshot of the top half of the "Questions" json file 27
Figure 32: Firebase security rules 28
Figure 33: UpdateQuestion function in the Trivia class 28
Figure 34: Easter Rising Quiz 29
Figure 35: Implementing the first video view in the Trivia class 29
Figure 36: Creating the media player for the audio file in the DisplayARCamera class 30
Figure 38: AR screen, when the proclamation is detected 30
Figure 37: Play and pause functions for the audio 30
Figure 39: Floor plan screen 31

FIGURE 40: GETITEM FUNCTION IN THE SECTIONSPAGEADAPTER CLASS 31
FIGURE 41: XML CODE FOR THE ABOUT TAB 32
FIGURE 42: SCREENSHOT OF THE TOP OF THE ABOUT TAB 32
FIGURE 43: FIGURE 44: SCREENSHOT OF THE ABOUT TAB AFTER SCROLLING DOWN 32
FIGURE 44: LIST OF VIRTUAL DEVICES TO CHOOSE FROM IN THE AVD MANAGER 37
FIGURE 45: PROJECT GANTT CHART 38
FIGURE 46: STACK OVERFLOW QUESTION I POSTED REGARDING EMULATOR ISSUE 41

1. Introduction

The project definition is as follows “This project will explore opportunities for exploiting new technologies that can enhance the visitor's experience of a museum (or possibly art gallery). It will involve conducting a critical review and evaluation of technologies currently employed in museums, and proposing a prototype system extending or integrating these technologies. This will be based on an understanding of the principles of inclusive design as well as strong interaction design skills”. At the beginning of this project, I set out a plan of how I was going to undertake the project. I decided to split the project up into 6 main steps. The first step was to decide what museum I wanted to base my project on. The second step was to review the technologies currently used in museums and decide which technology best suited my chosen museum. The third, fourth and fifth steps were to design, develop and test the application/system.

1.1. Background Research

On reviewing the technologies currently in museums across the world, one technology in particular stood out. This technology was augmented reality. Augmented reality can be used in many ways within museums. The Royal Ontario Museum is a museum of art, world culture and natural history which puts augmented reality to excellent use. It has an exhibition called Ultimate Dinosaurs: Giants from Gondwana, the exhibition uses an augmented reality application to layer virtual experiences over the real environment to bring the dinosaurs to life.[1] It works by users holding an iPad up to skeletons of dinosaurs throughout the museum and the app renders a 3D model on top of the skeleton. This allows museum visitors to bring these extinct animals to life on screen.

The Heroes and Legends exhibition in the Kennedy Space Centre Florida is dedicated to the men and women at the heart of America's space program. This exhibition uses augmented reality in the form of holograms, also known as holographic AR. This works with a hologram

and audio of the actual astronaut telling their stories of why they worked on the program and what the space experience was like. Gene Cernan an astronaut who tackled a terrifying spacewalk outside Gemini 9 capsule, can be seen in the form of a hologram above the actual historic space capsule accompanied by voice overs from her and fellow astronauts.[2] The use of holograms and audio makes visitors experience in the museum an extremely immersive one as it appeals to multiple senses.

The Detroit museum of arts is another museum which has been integrating AR into its exhibition, it has been premiering a mobile tour called Lumin. Lumin uses Google's tango technology to provide visitors with new ways to engage with the museum's collection.[3] This art museum was the first in the world to integrate 3d mapping and smartphone augmented reality technology into a public mobile tour.

1.2. Document Outline

The first section of this report provides a brief introduction to the project and the background research conducted. Section 2 deals with the functional and non-functional requirements of the project. Section 3 deals with the design of the project. The 4th section provides a detailed explanation of the implementation of the project. Section 5 deals with the testing conducted throughout the project. Section 6 deals with project management. Potential future work is discussed in section 7. In the final section, section 8 I discuss the challenges I overcame throughout the project.

2. Requirements

2.1. Functional

Use Cases

The augmented reality use cases I had in mind from the beginning of the project are listed below along with a brief description and basic flow of each.

Use case 1: Exploring an Artefact

Brief Description

Starts when the user holds the AR application up to artefact, additional information will pop up on screen around the artefact. The user can click the floating action button at the bottom of the screen. This use case ends when the user moves onto new artefact or user closes the application.

Basic Flow

1. User points camera to artefact in question
2. AR software in the application detects object
3. Application displays information on screen relevant to the detected artefact
4. User reads the additional information while holding the phone up to artefact
5. One of two options are displayed depending on the artefact:
    - User can click the **info** button at the bottom of the screen
    - User is brought to another view where additional information on the artefact can be found

      Or
    - User can click the **play** button at the bottom of the screen
    - User can choose to play a video clip or an audio file

Use case 2: Extending the View of an Artefact

Brief Description

Starts when the user holds the AR application up to artefact. A 3D model of the artefact in its original form is displayed on screen. This use case ends when the user moves onto the next artefact or closes the application.

Basic flow

1. User points phone camera to artefact in question
2. AR software in the application detects object
3. User gets option to view the 3D model by clicking a button
4. Application renders a 3D model of the artefact on screen
5. User views and interacts with the 3D model

Use case 3: Take a Quiz

Brief Description

Starts when the user clicks the button start quiz and ends when the user completes all the questions or exits the trivia tab.

Basic flow

1. User clicks start quiz
2. Questions are displayed on screen, with four answer options
3. User picks an answer option
4. Chosen answer is displayed as right or wrong
5. Next question is displayed
6. Completed score and time is displayed once the quiz ends

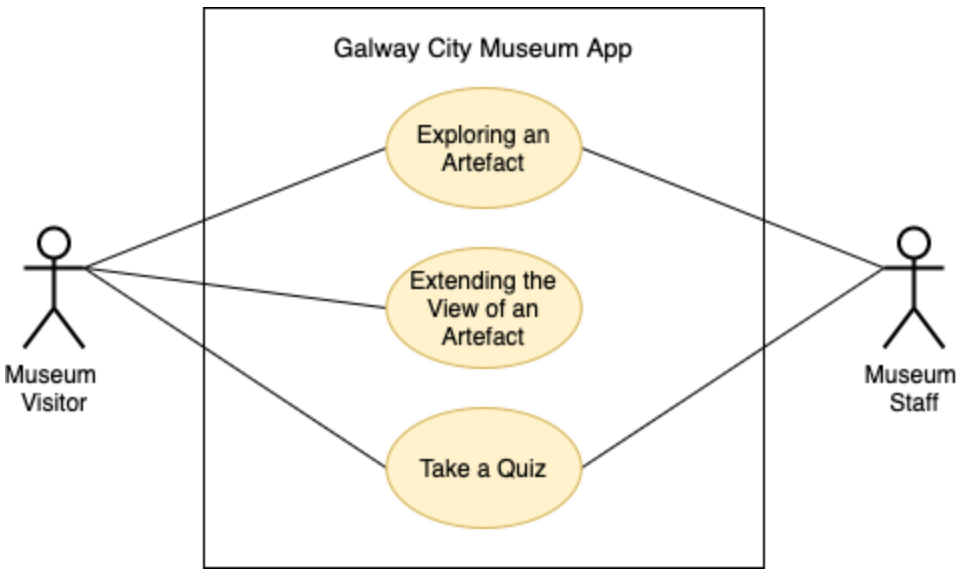

Figure 1: Use Case Diagram

2.2. Non-Functional

Usability: As my project is based around an application which will be used in a museum, the application has to be extremely user friendly as visitors of all ages will be using the application. The app has to have a simplistic layout so it is easy for the user to use.

Security:

The app can be used by anyone who downloads it. There is no login required as no data on the user needs to be stored. I have a set of security rules for the Firebase database.

Scalability:

The app will be somewhat scalable in the sense that I will start off with the capability of identifying a few artefacts and in the future will be able to scale out for more capabilities for other artefacts in the museum.

3. Design

3.1. Three Tier Client/Server Architecture

This project is based on a three layered architecture. A three layered architecture is a client-server architecture which is separated into three layers. The presentation layer, the application layer and the data layer. The presentation layer is the front end layer which consists of the user interface(UI) of the application, so essentially it's everything the user physically sees. The application layer contains the business logic of the application. It controls the application's functionality. The application layer receives input from users via the presentation layer, then processes the user's data with the help of the data layer.[4] The data layer houses database servers which store the data. Figure 2 shows a high level view of the project. The client technologies shown to the left and the server.

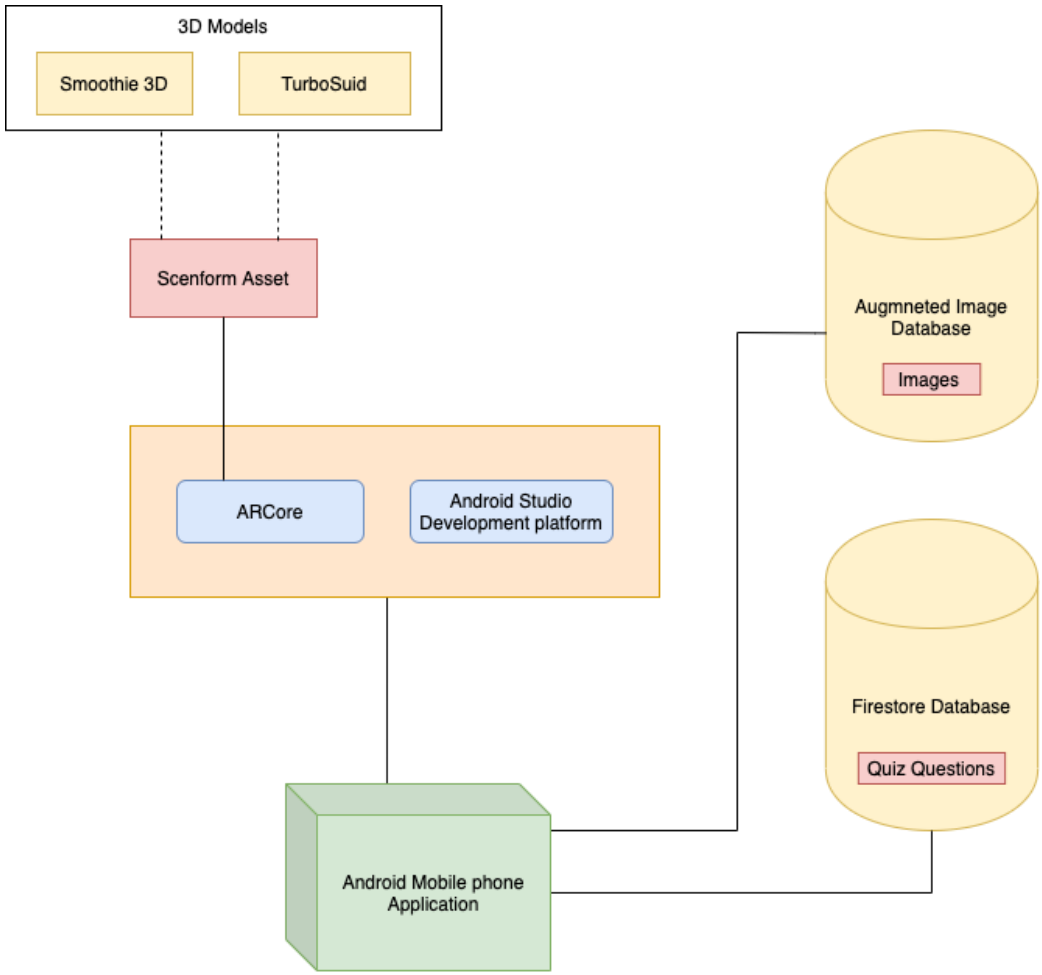

Figure 2: High Level Project Overview

3.2. Technologies

The first important technology decision was to decide which mobile operating system to develop the app for. The two options were Android or iOS. I conducted some research into the development process for both. The programming language of iOS is Objective-C and C. iOS devices are developed using the Xcode platform. Java is the language used to develop Android apps and then are developed using Android Studio. In the end I opted for Android. Mainly because I had previous experience working with Java on projects.

Client

Android Studio:

Android studio is an SDK especially designed for developing Android applications, it is built on JetBrains IntelliJ IDEA. This benefited me greatly as I had experience in developing using Intellij before. I began a Udemy course in order to gain knowledge on Android Studio. The course took me through the basics of developing a project on Android Studio.

Augmented Reality SDK:

It was essential to have a AR SDK to develop the AR feature in the project. One of the core features I needed in an SDK was having the ability to integrate with Android applications. Below are some of the AR SDK technologies I reviewed.

Smoothie 3D:

Smoothie 3D is a website that is used to construct basic 3D models online. I found this website while trying to figure out how to reconstruct a 3D model from a single 2D image. This was needed to create a 3D model from the Devon chafing dish photograph. I will talk about using this tool in more detail later.

Review of AR SDK Technologies

SDK	**Vuforia**	**ArToolKit**	**ARKit**	**ARCore**
Pricing/Open Source	Not Open Source - paid but has a free version	Free & Open Source	Free & Open Source	Free & Open Source
Supported platform	Android, iOS, UWP	Android, iOS, Windows, Linux	iOS 11/12	Android 7.0+ & iOS 11+
Made by	Vuforia	DAQRI	Apple	Google
Marker	Yes, adv. + VUMark	Basic	Yes	Yes
Unity 3D	Yes	Yes	No	Yes
3D Object tracking	Only box and cylinder	No	Yes	Yes
Geolocation	Yes	No	Yes	Yes
Features	2D and 3D recognition & scanning real objects for recognition.	2D recognition & emulate 3D tracking methods by using multiple images as markers.	They both make clever use of anchors, which prevent AR content from being lost if the device user wanders away from the area mapped by the app.	
Concerns	Free version is probably quite limited	No commercial support, outdated	IOS only - is a concern	

After reviewing the range of available technologies I opted for ARCore. It is made by Google, meaning it is compatible with Android applications. It also uses anchors which is a very useful AR feature to have.

Sceneform:

Sceneform makes it relatively straight forward to render realistic 3D scenes in AR and non-AR apps, without needing to learn OpenGL.[5] It has a high level API for rendering 3D models using Java. It also has an Android studio plugin which is used to import 3D models as sceneform assets. This plugin automatically adds the 3D models to the gradle file as well as providing a preview of the model in the viewer tab.

Augmented Image Database:

This is a database containing a list of images to be detected and tracked by ARCore [6].

Server

Firebase:

Firebase is a mobile and web app development platform that provides tools and services to help developers develop applications. I chose to use this database as it is easy to integrate with Android apps. It also has a user friendly dashboard, as shown in figure 3. Using this database meant I could build the app without the need of a server. It was necessary to setup a Firebase account in order to create a database. There was very little configuration involved. Another advantage of Firebase is it can easily be scaled out. This can be done by adding cloud volumes, so the database always has room to grow.

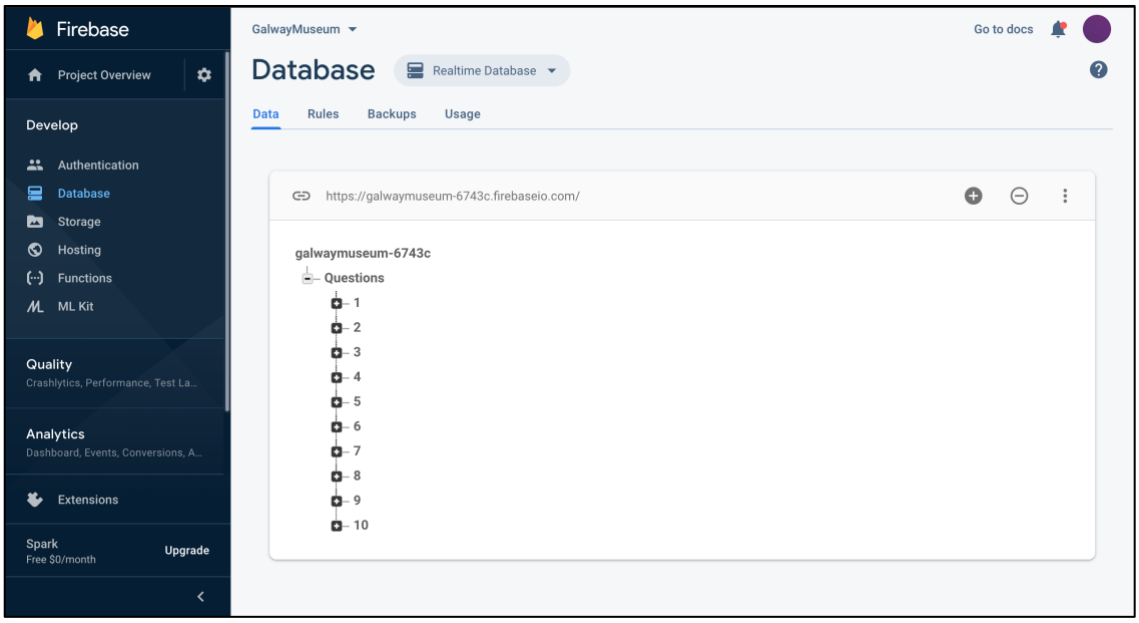

Figure 3: Screenshot of the Firebase Console

3.3. Choosing a Museum

My initial intention was to base my project on Collins Barracks museum in Dublin. I emailed the museum requesting information on some of their exhibitions but unfortunately I never received any correspondence back from them. My next choice was Galway City museum. It is a small museum with 7 exhibitions. I had a particular interest in two of them. They were the “Revolution in Galway” and the “Prehistoric Galway” exhibitions. I decided to concentrate on these two exhibitions for the augmented reality feature. Both exhibitions had artefacts that suited the use cases planned. During my visit to the museum I photographed some of the artefacts I thought could be incorporated into the AR feature.

3.4. Choosing Artefacts

Including all the artefacts from both exhibitions would have been too challenging. I decided to concentrate on 4. The Devon Chafing Dish was the first artefact I chose (figure 4). This artefact is in a glass display in the prehistoric Galway exhibitions. It is over 400 hundred years old and one of the oldest artefacts in the museum. The bowl was broken, with one of the handles missing. As a result, this artefact suited the second use case.

Figure 4: Photograph of Devon Chafing Dish

The next artefact is the proclamation (figure 5). This artefact speaks for itself with regards its significance. An audio file is played once this artefact is detected.

Figure 5: Photograph of Proclamation

The third artefact is a statue of Eamon De Valera's head (figure 6). This was also located in the "Revolution in Galway" exhibition. This was used for my first use case; whereby video clips are shown once this artefact is detected.

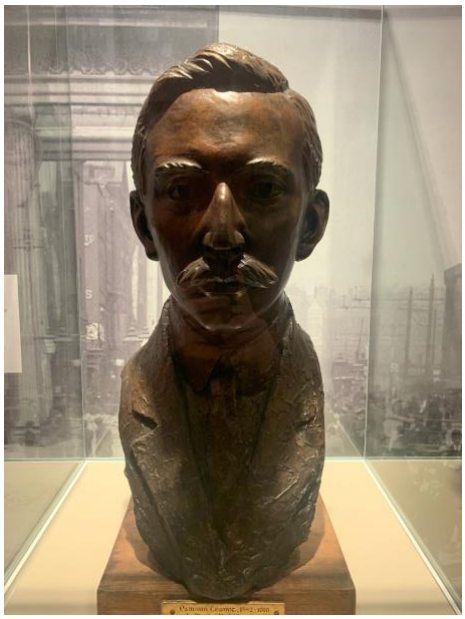

Figure 6: Photograph of Eamon De Valera's head

The final artefact (figure 7), a green post box is based around my first use case. This post box holds important significance. Before Ireland got its Independence post boxes were painted red. One of the first acts of the new Irish Government was to order that green would be the new colour. This artefact was used for my second use case.

Figure 7: Photograph of post box

3.5. Presentation layer Design:

Home Screen Design:

Designing the presentation layer involved creating a few rough sketches of each screen layout. For the home screen, I decided to include a tab view along the top. With three main tabs, the home tab, the about tab and the events tab. Tabs allow users to easily swipe from one view to the next, without clicking anything. A sketch of the home screen layout is shown in figure 8. There are three buttons on the bottom of the screen. There is a large central AR button as this is the main feature in the app, with a map button and a trivia button either side. The large central AR button will encourage users to use this feature first. In general I designed a simplistic home view with very few buttons to avoid confusion for users. I also included icons on the bottom buttons. A screenshot of how the layout actually looked is shown in figure 9.

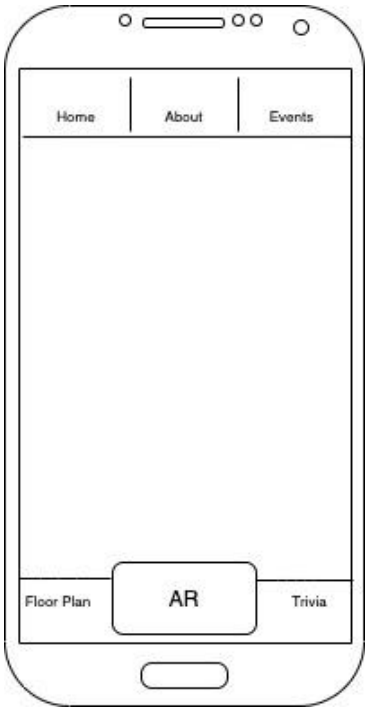

Figure 9: Sketch of the home screen

Figure 8: Screenshot of actual home screen

AR Screen Design:

In figure 10 I have a sketch of how I wanted the AR screen to look. This screen is the core of the app and is where users will be spending most of their time. It was essential to design this so it is as straightforward as possible for the user. To achieve this I added on screen prompts telling the user what to do. Once the user first enters AR mode, a set of basic instructions come up on screen. These instructions tell the user to move around, pointing the camera at artefacts in order to find out more about them. The sketch in figure 10 shows a text box where the name of the detected artefact can go. The bottom box displays extra information on the artefact or instructions on what to do next. The bottom box can turn into a button. If there is a 3D model of the artefact available, this button will render the model if it's clicked. In the bottom left hand corner there is a floating action button, which can be clicked to get in depth information on the artefact. The floating action button can also take the form of a play button depending on the detected artefact.

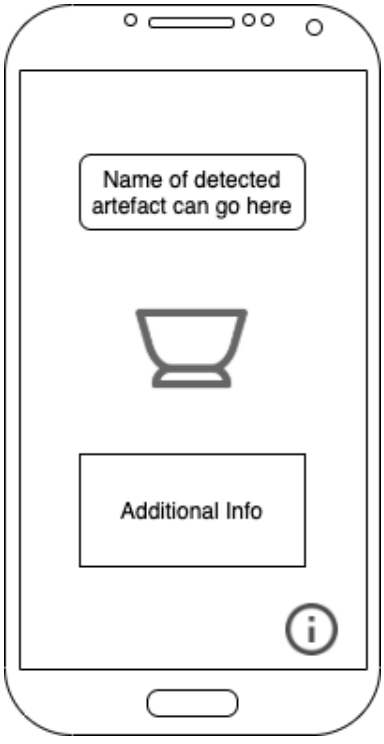

Figure 11: Sketch of AR screen

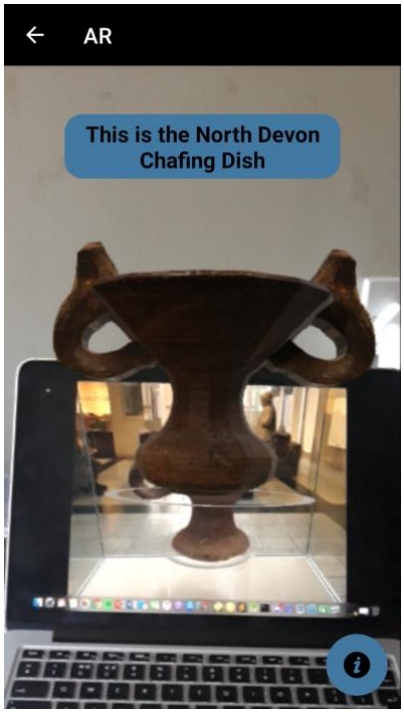

Figure 10: Screenshot of Actual AR screen

App colour scheme and icons:

User interface was very important in designing this application. I chose to maintain a consistent colour scheme throughout, to avoid complicating the application design or overpower users. I came across a blue/grey palette in an article on the Medium website. The palette is shown in figure 12. A lot of the buttons in the application have icons on them, most of which I sourced from FlatIcon.

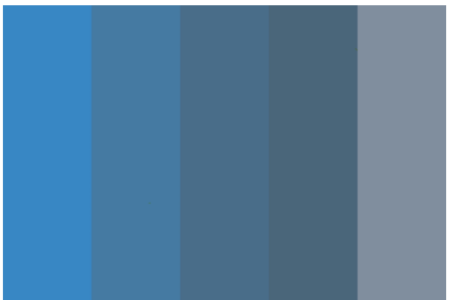

Figure 12: Screenshot of Actual AR screen

3.6. Application

The UI on android applications is handled through layouts. A layout is an xml file that defines the appearance of its corresponding activity and defines how all the controls are placed. These xml files are located in the res/layout folder. Activities and fragments make up the application layer of an android application. They are essentially java files responsible for handling interactions with the user. Each activity is defined by a layout.

4. Implementation

4.1. Creating a project in Android Studio

The starting point of my project in Android studio was creating a new project with a basic Activity as the Home Activity. On creating an activity, you can choose from a list of predefined activities, as shown in the figure 13. When a new Activity is created, two files are then generated, a java file and an xml file. The java file is known as the activity file and the xml is the layout file. For the home activity the two file names were HomeActivity and activity_home. I kept this naming convention, of the name of the activity followed by Activity and then activity_ followed by the name of the activity for each new activity created.

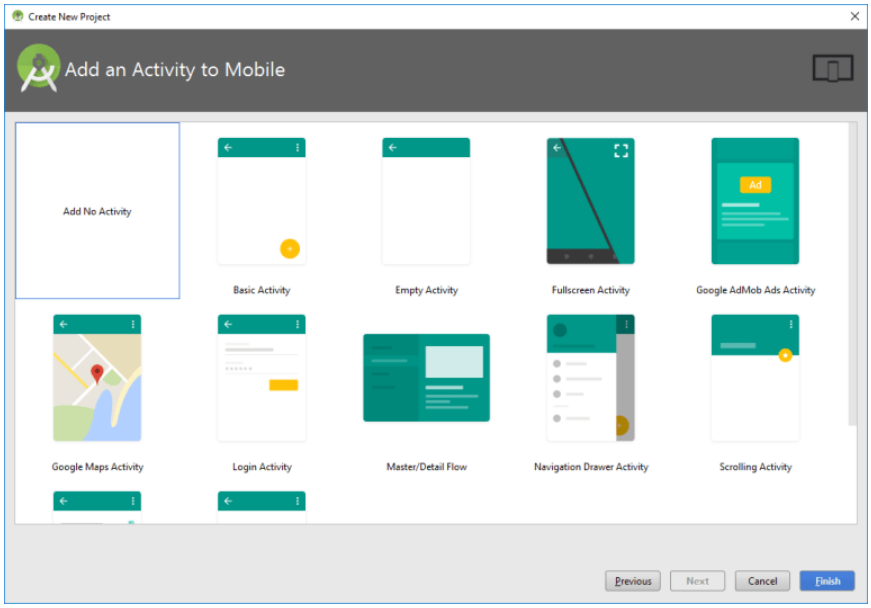

Figure 13: Choice of Android activities

4.2. On Create Method

Inside every Activity java class there is an OnCreate method as shown in the code snippet below. This function is called once the activity is launched. The onCreate method is where the corresponding layout for that activity is specified using setContentView. Views that are specified in the layout class such as text views, image views, video views and buttons are assigned to java variables in the onCreate method. The toolbar is also referenced here.

```
public class Home_Activity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home_);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
```

Figure 14: OnCreate method template

4.3. Project files

AndroidManifest.xml

The Android Manifest file is an xml file that is essentially the backbone of an Android application. Every activity is specified here as well as permissions for the application to use features on the phone. If an activity is not specified in the Android manifest file, then the application does not know the activity exists. Figure 15 displays the top half of the android.manifest.xml file. Each component is separated by tags. The permissions for the application are specified at the top of the file. As you can see I have requested permission for the application to use both the camera and vibrate feature on the phone. The application tag includes all the activities as well as the app icon, name, label and theme. Each activity is then specified in the activity tags. The parent activity is also specified inside the activity tag. This tells the application where to go once the back arrow is pressed at the top of an activity. In the video activity case below the parent activity is the DisplayArCamera.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.fyp">

    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.VIBRATE" />

    <uses-feature
        android:name="android.hardware.camera.ar"
        android:required="true" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/galway_logo"
        android:label="Galway City Museum"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".VideoActivity"
            android:parentActivityName=".DisplayARCameraActivity">
        </activity>
        <activity
            android:name=".PostboxInfoActivity"
            android:label="Penfold Pillar Box"
            android:parentActivityName=".DisplayARCameraActivity"
            android:theme="@style/AppTheme.NoActionBar">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.example.fyp.DisplayARCameraActivity" />
        </activity>

        <meta-data
            android:name="com.google.android.actions"
            android:resource="@drawable/gradient_2" />

        <activity
            android:name=".BowlInfoActivity"
            android:label="North Devon Chafing Dish"
            android:parentActivityName=".DisplayARCameraActivity"
            android:theme="@style/AppTheme.NoActionBar" />
        <activity android:name=".EndQuizActivity" />
```

Figure 15: AndroidManifest.xml

Build.gradle

Gradle is a build tool that builds and configures app directories and files. There are two build.gradle files in android applications, one for the overall project and one for the app. The project build.gradle is known as the top-level gradle file and is located in the root folder of the project. The project build.gradle file configurations are globally accessible, so whatever is included here will affect all modules. Figure 16 displays the top half of the project build.gradle file.

```
// Top-level build file where you can add configuration
// options common to all sub-projects/modules.

buildscript {
    repositories {
        google()
        jcenter()

    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.5.3'
        classpath 'com.google.ar.sceneform:plugin:1.4.0'
        classpath 'com.google.gms:google-services:4.3.3'

        // NOTE: Do not place your application dependencie
        // they belong in the individual module build.gra
    }
}

allprojects {
    repositories {
        google()
        jcenter()
        maven{
            url "https://maven.google.com" //Google's Mav
        }

    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Figure 16: Project build.gradle

App build.gradle

The app build.gradle is located in the app folder. Whatever is located in this file, will only affect the module it's located in. Dependencies, build types and flavours are specified in here. Figure 17 is a screenshot from the app build.gradle file. This is where the bulk of the dependencies are declared, as they are only needed for this module. I have my sceneform, firebase and ARCore dependencies included in here. The configurations for each sceneform asset is included underneath the dependencies. Sceneform will be discussed in more detail in a later section.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.legacy:legacy-support-v4:1.0.0'
    implementation 'androidx.lifecycle:lifecycle-extensions:2.0.0'
    testImplementation 'junit:junit:4.13'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'com.google.android.material:material:1.0.0'
    implementation 'com.google.ar:core:1.4.0'
    //implementation 'com.google.ar.sceneform.ux:sceneform-ux:1.15.0'
    implementation 'com.google.ar.sceneform.ux:sceneform-ux:1.4.0'
    implementation 'com.google.firebase:firebase-analytics:17.2.2'
    //implementaion 'com.google.android.gms:play-services-auth:17.0.0'
    implementation 'com.google.firebase:firebase-firestore:21.4.0'
    implementation 'com.google.firebase:firebase-database:19.2.1'

}


sceneform.asset('sampledata/Smoothie3DBowlModel/model_bowl.obj',
        'default',
        'sampledata/Smoothie3DBowlModel/model_bowl.sfa',
```

Figure 17: App build.gradle

4.4. Image Detection

The AR feature in my project uses image detection as the trigger. ARCore provides the ability to detect 2D images. In order for ARCore to know what images to detect, it first has to be supplied with a database of images. Images that need to be detected are called reference images. I created this database using arcoreimg tool. This tool is a command line tool that takes a set of reference images and generates an image database. [7] Images can be supplied from your directory or from an image list file. I chose to add the images from an image list file. Using an image list file allows you to assign a name to each image as well as setting the images physical width in metres. The image list file I used is shown in figure 18.
Each line in the image list file contains a name, the image file path and the width of the image. The database is created using the build-db command, followed by the input image list path and the output database path. The database file is saved to the specified output database path. Once created, I then moved the imagedb.imgdb file to the raw folder in my android studio project.

```
bowl|/Users/johncummins/Downloads/FYP stuff/Images_for_DB/bowlimage.jpeg|0.2
postbox|/Users/johncummins/Downloads/FYP stuff/Images_for_DB/postbox.jpeg|0.5
proclamation|/Users/johncummins/Downloads/FYP stuff/Images_for_DB/proclamation.jpeg|0.3
eamon|/Users/johncummins/Downloads/FYP stuff/Images_for_DB/eamon.jpeg|0.3
```

Figure 18: Image list file

Checking Image Quality:

Arcoreimg tool also supplies a command line tool for evaluating image quality. This tool is run by using the eval-img command followed by the path to the image, as shown in figure 19. This returns an image quality score of between 0 and 100. I ran this command with my reference images and I received the following scores. The more distinct the features in an image are, the higher the score. The higher the score the more detectable an image is. My images averaged around 50, which could have been better.

```
(base) Johns-MacBook-Pro-2:macos johncummins$ ./arcoreimg eval-img --input_image_path=/Users/johncummins/Downloads/Images_for_DB/proclamation.jpeg
40
(base) Johns-MacBook-Pro-2:macos johncummins$ ./arcoreimg eval-img --input_image_path=/Users/johncummins/Downloads/Images_for_DB/bowlimage.jpeg
30
(base) Johns-MacBook-Pro-2:macos johncummins$ ./arcoreimg eval-img --input_image_path=/Users/johncummins/Downloads/Images_for_DB/eamon.jpeg
60
(base) Johns-MacBook-Pro-2:macos johncummins$ ./arcoreimg eval-img --input_image_path=/Users/johncummins/Downloads/Images_for_DB/postbox.jpeg
35
(base) Johns-MacBook-Pro-2:macos johncummins$
```

Figure 19: Screenshot from running the Image quality tool

In the early days of the project I added each reference image individually to the database, this was implemented in the OnCreate method of the ARDisplayCamera. This meant a new local database was created every time the onCreate method was called. This worked fine for adding just one reference image, but as time moved on I wanted to detect more artefacts, which required the addition of more reference images. Loading the AR activity with anything over two reference images led to the AR activity becoming very slow to start. Once I moved over to the augmented image database I cut the loading time for the AR feature from 15 seconds down to just 5 seconds.

4.5. 3D Modelling

Bowl model

I used smoothie 3D to develop a 3D model of the Devon Chaffing Dish. This was done by creating a new project on the website. The image of the bowl was then added to project. The image was displayed on screen with modelling tools displayed around it. I divided the image of the bowl into separate components. Each component was then individually expanded to the width of the artefact. One of the modelling tools included in this website was symmetries. This tool allows you to reflect an image though the x, y or z axis. A second handle was created on the bowl using this tool. This was done by selecting the handle component as shown in the circle in figure 20. I then reflected this component through the y-axis, to make a handle appear on the other side of the bowl. The finished model was exported as an obj file. Obj is the file format that Android Studio uses to import sceneform objects.

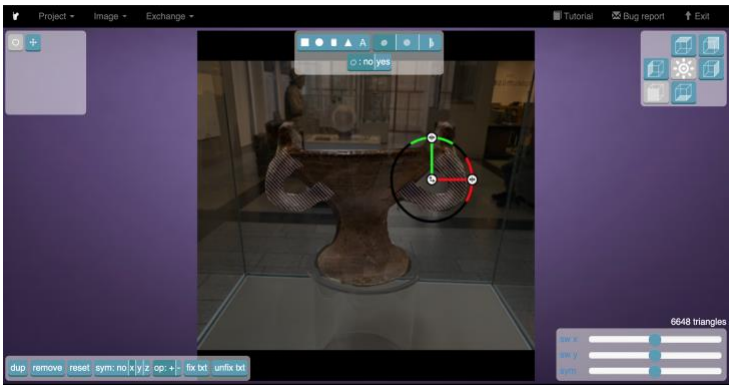

Figure 21: Creating the bowl 3D model on Smoothie 3D

Figure 20: The finished product of the bowl 3D model

Post box Model

The other 3D model that needed to be created was of the post box that was on display in the Galway museum. As discussed previously this post box was a symbol of the Irish independence. My plan was to render a 3D model of the green post box and allow the user to tap on it to see it change back to the colour it was before Ireland got its independence. I found a website called TurboSquid, that supplies 3D models which can be downloaded in obj file format. There was a 3D model of an imperial red post box on the website, which is shown in figure 22. This was suitable image for the original postbox. I downloaded the 3D model in the obj format and then converted it to a green colour using gimp photo editor tool.

Figure 22: The post box 3D model on the TurboSquid website

Importing models to Android Studio

Importing the 3D model to Android Studio firstly involved making a sample directory in my Android Studio project. The two obj files as well as the mtl files were copied to the sample directory. The obj file is the 3D version of the model and the mtl file declares the textures of the model. Once copied to the sample directory I imported each model to sceneform. Sceneform allows you to render the 3D models in augmented reality applications. To import the obj as a sceneform asset I clicked on the obj file in android studio and chose import sceneform object. An option came up to include the mtl file, I included the mtl file path and clicked import. This then created two files, a sfa (sceneform asset) file and a sfb(sceneform binary) file. The sfa file is the human readable version and the sfb file is the model that is physical inputted into the application.

4.6. Sceneform

CustomARFragment

The CustomARFragment handles ARCore session creation, requesting camera permission, as well as provide providing UX elements. It sets up a new session once the DisplayARCamera activity is run. This class extends the pre-created ArFragment class that is included with ARCore. Figure 23 shows a code snippet from the CustomARFragment class. This class calls the AugmenteImgDB which is in the DisplayARCamera class. I added the CustomARFramgent class to the DisplayARCamera xml file, which essentially gives the user the camera view when they open this Activity.

```
package com.example.fyp;

import ...

public class CustomARFragment extends ArFragment {
    @Override
    protected Config getSessionConfiguration(Session session) {

        //gets rid of promt to tell user to move phone around
        getPlaneDiscoveryController().setInstructionView(null);

        Config config = new Config(session);
        config.setUpdateMode(Config.UpdateMode.LATEST_CAMERA_IMAGE);
        //session.configure(config);

        DisplayARCameraActivity arActivity = (DisplayARCameraActivity) getActivity();
        arActivity.AugmentedImgDb(config, session);
        this.getArSceneView().setupSession(session);
```

Figure 23: Code snippet from the ARFragment Java class

DisplayARCameraActivity

The DisplayARCamera class is where the bulk of the code for the AR feature is run. This class is called once the user clicks the AR button. Figure 24 displays the AugmentedImgDB function which is in the DisplayARCamera class. This function adds the augmented image database to the project. Firstly, it creates an Input Stream to the reference image database which was talked about previously. A new object of the AugmentedImageDatabase is made from this Input Stream. This object is then added to the config.

```
//This function sets up an augmented image database
//This function is called in custom ARfragment class
public void AugmentedImgDb(Config config, Session session){
    InputStream inS = getResources().openRawResource(R.raw.imagedb);
    try {
        AugmentedImageDatabase augmentedImgDB = AugmentedImageDatabase.deserialize(session, inS);
        config.setAugmentedImageDatabase(augmentedImgDB);
    } catch (IOException e) {
        e.printStackTrace();
```

Figure 24: AugmenetedImgDb function

OnUpdateFrame

OnUpdate is the core function in the DisplayARCamera class, it is called every time the scene is updated. This function gets a screenshot of the camera screen and compares against each reference image in the augmented Image database, as shown on line 118 of figure 25. Once an image is being tracked, it enters a series of if statements. Each if statement compares the name of the image being tracked to the names of 4 chosen artefacts. Within each if statement is the code to execute different features depending on the artefact. These features will be discussed in more detail in alter sections.

```
108     private void onUpdateFrame(FrameTime frameTime){
109         Frame frame = arFragment.getArSceneView().getArFrame(); //gets the current frame (basically a s
110
111         //this gets all the augmented images that have been added to the database
112         Collection<AugmentedImage> augmentedImages = frame.getUpdatedTrackables(AugmentedImage.class);
113
114         // Get instance of Vibrator from current Context
115         Vibrator v = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
116
117         //goes through entire image DB and check if any images have bee detected
118         for (AugmentedImage augmentedImage : augmentedImages){ //for each image aug image in augDB
119             if (augmentedImage.getTrackingState() == TrackingState.TRACKING){ //if aug img is being tra
120
121         //BOWL
122                 if (augmentedImage.getName().equals("bowl") && addBowlModel == true){ //if img being tr
123                     // Vibrate for 400 milliseconds
124                     v.vibrate( milliseconds: 300);
```

Figure 25: OnUpdateFrame function in the DisplayARCamera class

4.7. Adding Models to the Scene

CreateModel Function

Create model essentially builds the 3D model. This function takes in 5 parameters.
The ARfragment, an anchor, the model sfb file, and an x and z value. The anchor is a position on which objects can be placed. The x value and z value are not used in this function but are passed onto the addNodeToScene function, which will be discussed in the next section. ModelRenderable.builder renders the 3D model that is set in set source, which is passed into this function through model parameter, as shown in figure 26.

```
private void createModel(ArFragment fragment, Anchor anchor, Uri model, int x, int z){ //builds the model
    ModelRenderable.builder() Builder
            .setSource(fragment.getContext(), model) Builder
            .build() CompletableFuture<ModelRenderable>
            .thenAccept(renderable -> addNodeToScene(fragment, anchor, renderable, x, z)) CompletableFuture<Void>
```

Figure 26 CreateModel function called in the post box if statement

The CreateModel function is called in the postbox if statements, as shown in figure 27 It is also called for the bowl if statement but the bowl.sfb model is passed through instead of "green_postbox.sfb". Inside the ModelRenderable.builder function, the model gets built then the addNodetoscene function is called.

```
createModel(arFragment, augmentedImage.createAnchor //then place 3D model ontop of image
                (augmentedImage.getCenterPose()), //creates anchor in centre of detected image
        Uri.parse("green_postbox.01.sfb"),  x: 0, z: 90); // model of green_postbox.sfb)
```

Figure 27: CreateModel Function in the DisplayARCamera class

Adding the Node to the Scene

This function is the final step in adding the 3D model to the scene. This function essentially creates the object in the camera view, so the user can view it in the real world. A node is a place where an object can be attached, there are two types of nodes, anchor nodes and transformable nodes. Anchor nodes are positioned on the ARCore anchor, they stay in the same position relative to the real world, position and orientation cannot be changed. A transformable nodes position on the other hand can be interacted with, we can rotate, move and scale this node. First an anchor node is made, that takes the input of the anchor. Next a transformable node is made, this node takes the input of the renderable from the createModel function. The anchor node is set as the parent of the transformable node. The transformable node is then added as a child to the fragment. Also passed into this function are two integers, x and z. These are passed into the addNodeToScene function with the model. The z value

was set to 90 for the postbox, as I needed the transformable node to rotate 90 degrees in the z axis when it was added to the scene in order to have the correct orientation. The x value was then needed to rotate the bowl model 90 degrees on its x-axis, in order to have a correct orientation once added to the scene. This was done with the setLocalRotation function, as shown in figure 28. Node.Select() is the last line of code in this function, this selects the transformable node (3D model) so it can be interacted with.

```
node.setLocalRotation(Quaternion.axisAngle(new Vector3( v: 1f,  v1: 0,  v2: 0), xVal));
```

Figure 28: setLocationRotation function used to rotate the models

Once the artefact is detected the name of the artefact pops up above the 3D model. A button gets displayed below the 3D model as shown in figure 29. The user can press this button and the post box changes to what it looked like before Ireland got its independence, as shown in figure 30. The floating action button at the bottom of the screen allows the user to retrieve more information on the artefact.

Figure 30 Screenshot of when the artefact is first detected

Figure 29: Screenshot of the postbox after the button is pressed

4.8. Trivia Section

The trivia section contains one quiz on the Easter Rising. It consists of 10 questions, where the user is presented with 4 answer buttons for each question. The questions and answers are pulled from a real-time Firebase database.

Adding Firebase to my app:

There were two ways to connect Firebase to an app. The first and the recommended way was using the Firebase console, the second was using Android Studio Firebase assistant. I chose to use the Firebase console. I set up a Firebase account and then created a new project. The project was named “Galway Museum”. Registering the app with Firebase was done by adding a firebase configuration file to my android studio project. The config file is a json file that was generated once the Firebase project was created. I added a plugin for google service and the Firebase dependency to the app build.gradle files.

Populating the database:

The next step was populating the database with questions. I gathered 10 questions some of which I found online and some of which I came up with myself. A json file containing the questions was created. The json contains each individual question as a string, as well as 4 answer options and the correct answer. Figure 31 shows the first 3 questions from the json file. Once created I imported the json file to my project in Firebase, using the import option in the Firebase console.

```
{
    "Questions" : {
    "1" : {
    "question" : "On what date did the Easter Rising start?",
    "option1" : "April 21st",
    "option2" : "April 24th",
    "option3" : "April 29th",
    "option4" : "April 26th",
    "answer" : "April 24th"
    },
    "2" : {
    "question" : "Which leader declared the Republic by reading the 1916 Proclamation?",
    "option1" : "Padraig Pearse",
    "option2" : "Thomas Clarke",
    "option3" : "James Connolly",
    "option4" : "Michael Collins",
    "answer" : "Padraig Pearse"
    },
    "3" : {
    "question" : "What does IRB stand for?",
    "option1" : "Irish Republican Brother",
    "option2" : "Irish Republic Brotherhood",
    "option3" : "Irish Republican Brotherhood",
    "option4" : "Irish Republic Band",
    "answer" : "Irish Republican Brotherhood"
    },
```

Figure 31: Screenshot of the top half of the "Questions" json file

Firebase rules:

The security rules I created for my firebase project are shown in figure 32. The rules are fairly basic; they allow anyone using the app to read the data but only authenticated users can write to the database.

```
// These rules grant access to a node matching
// user's ID from the Firebase auth token
{
  "rules": {
    ".read": true,
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Figure 32: Firebase security rules

Implementing the Quiz

Once the database was set up and populated with the questions, I moved back to android studio, where I created a new activity called Trivia. One text box and 4 buttons were added to the activity_trivia.xml file. Below is a screenshot of a section of the UpdateQuestion function from the TriviaActivity java file. As shown in figure 33, a root reference to the database was created. A reference to the child node called "Questions" was also created. This node contains the list of questions, each question uniquely identified by a number 1 to 10.

```
//database reference pointing to root of database
rootRef = FirebaseDatabase.getInstance().getReference();

//database reference pointing to child node questions
childRef = rootRef.child("Questions");

        //question
        newRef = childRef.child(Integer.toString(qNum));
        newRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {

                String qName = dataSnapshot.child("question").getValue().toString();
                questionBox.setText(qName);
```

Figure 33: UpdateQuestion function in the Trivia class

A third reference called newRef was then created. This is a reference to a child node of the "Questions" node, this reference points to the variable qNum. This is essentially a reference to each individual question item. The qNum variable gets updated each time the UpdateQuestion function is called, this loops through each item in the database. There is a .addValueEventListner attached to the reference of the question item, this listens to the database reference it's attached to. Inside the onDataChange function, the string value of the question is assigned to a string variable called qName. qName is then assigned to the questionBox variable, which is a text box where the questions are displayed. The same is then

done for each answer option, adding the answer option string value to each button. The answer to the question is also assigned to a variable answer.

Quiz Layout:

A onClickListner is set to each of the 4 buttons and if the answer is equal to the button the user presses, the button goes green and the "correct" counter gets incremented. On the other hand, if the user presses the button not equal to the answer, the button goes red and the correct counter is not incremented. Once an answer button is pressed the updateQuestion function gets called and the next question is displayed. The layout of the finished quiz is shown in figure 34. Including a text view for the current question number, a timer and a score.

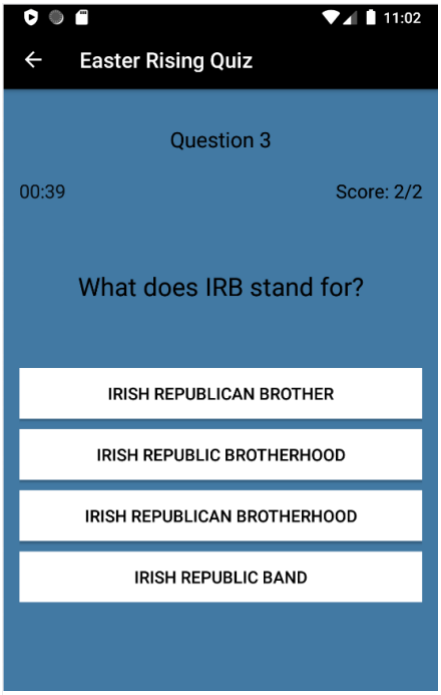

Figure 34: Easter Rising Quiz

4.9. Adding Video Content:

Adding video content started with creating a new basic activity called video. A relative layout was added to the activity_video.xml , as well as two video view's. One video view was located underneath the other. A text view was also added right under each video view to display the caption of the video. The video views were then added to the java files using findViewById. I downloaded video clips of Eamon De Valera and stored them as mp4 videos in the raw directory of the project. The path of each video is referenced as a string. The string is parsed to a URI, then the URI is set to the video view. A media controller was added to each video. The media controller displays the play/pause controls for the video. Figure 35 shows the code for the creation of the first video view.

```
videoView1 = findViewById(R.id.videoView1);
String videoPath = "android.resource://" + getPackageName() + "/" + R.raw.eamon_video;
Uri uri = Uri.parse(videoPath);
videoView1.setVideoURI(uri);

MediaController  mediaController = new MediaController( context: this);
videoView1.setMediaController(mediaController);
//knows where it has to be positioned on screen
mediaController.setAnchorView(videoView1);
```

Figure 35: Implementing the first video view in the Trivia class

4.10. Adding Audio Content:

Audio content is played once the proclamation is detected. I found a YouTube clip of relatives of the rising reading the proclamation aloud. I thought this would be suitable for inclusion. A new Activity was not created for this, instead the play button was added as a floating action button at the bottom of the AR screen. The audio file implementation is done in the DisplayARCamera class. Figure 36 shows how the source of the audio clip is attached to the media player.

```
mediaPlayer = MediaPlayer.create( context: this, R.raw.proclamation_audio_1);
```

Figure 36: Creating the media player for the audio file in the DisplayARCamera class

Two separate functions for the play and pause were created, as shown in figure 37. The play function is called from the “Eamon Dev” section in the OnUpdateFrame function. This function os called when the play floating action button is pressed. Once the play button is pressed the play button changes to the pause button, so the user can pause the audio. The pause function is then called when the pause button is pressed. This is all done while looking at the proclamation though the camera in the background, as shown in figure 38.

```
public void playAudio(View view){
    mediaPlayer.start();
    pauseButton.show();
    playButton.hide();
}

public void pauseAudio(View view){
    mediaPlayer.pause();
    pauseButton.hide();
    playButton.show();
}
```

Figure 38: Play and pause functions for the audio

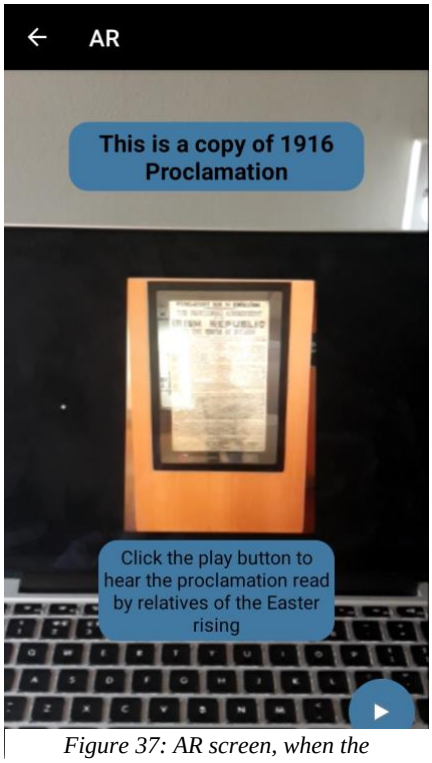

Figure 37: AR screen, when the proclamation is detected

4.11. Other Layouts

Tabbed Activity

A tabbed activity is an activity that allows the user to swipe between sibling screens (fragments) while keeping the parent screen. I used it in the home view, with the three tabs at the top; home, about and events. I also used the tab view in the floor plan section, this allows users to swipe between the ground floor, first floor and second floor. I will go through the implementation of the floor plan layout in detail as both implementations follow the same pattern. The floor plan layout is shown in figure 39.

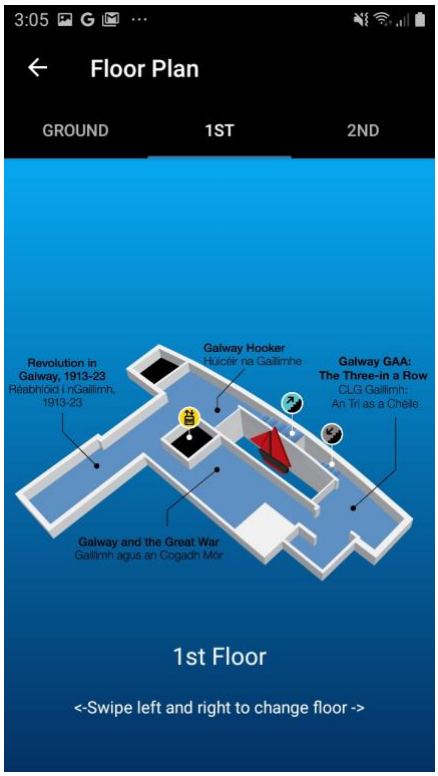

Figure 39: Floor plan screen

Floor plan

A new tabbed activity was created. This generated four files. An activity_floor_plan.xml, one fragment.xml and then two separate java files, a PagViewModel and a SectionsPageAdapter.

Three tabs were needed in the tab view. I had to create, two more separate fragment.xml tabs to hold the layout of each tab. I also created 2 more java files, each pointing to the fragment layout. The code from the sectionsPageAdapter is shown in figure 40. The getItem function is essentially a switch statement that switches between each of the specified fragments.

```
@Override
public Fragment getItem(int position) {
    // getItem is called to instantiate the fragment for the given page.
    // Return a PlaceholderFragment (defined as a static inner class below).

    Fragment fragment = null;
    switch(position) {
        case 0:
            fragment = new GroundFloorFragment();
            break;
        case 1:
            fragment = new FirstFloorFragment();
            break;
        case 2:
            fragment = new SecondFloorFragment();
            break;
    }
            return fragment;
```

Figure 40: GetItem function in the SectionsPageAdapter class

Scroll View

Scroll view allows the user to scroll up and down within a view. This feature is used in views where all the text doesn't fit onto the screen. Scroll view was needed in both the "more info" pages. It is also needed in the about tab as shown in figure 42. This involved a considerable amount of xml code in the respective layout files. To the right is a screenshot of the xml file for the about tab. The root tag is a scroll view, everything inside the scroll view tags is included in the scroll view.

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="80dp">

<RelativeLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">


    <TextView
        android:id="@+id/welcome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        android:text="About the Museum"
        android:textColor="#FFC107"
        android:textSize="24sp">
```

Figure 41: Xml code for the about tab

Figure 42: Screenshot of the top of the about tab

Scroll ↓

Figure 43: Figure 44: Screenshot of the about tab after scrolling down

5. Testing

5.1. User Testing

Due to unforeseen circumstances regarding coronavirus, I was unable to conduct as much user tests as I had originally planned. I did however conduct user testing with two family members. I conducted both tests individually. The questions I asked and both of the user's answers are displayed below. In order to make it easier to read I have displayed the first user's answers in blue and the user 2's answers in green. I split the questions up into 5 different sections. Each section is numbered and labelled in bold below. Sections 1 to 3 were asked before the testing began. The section 4 questions were conducted during the test. Then the questions in Section 5 were asked once the test was over.

1.User demographics:

To get started, can you tell me briefly about yourself?

What is your current occupation?

User 1: I'm a secondary school teacher

User 2: I'm a vet student

On a scale of 1 to 5 (1=not at all confident, 5=very confident), how would you rate your level of confidence in using your mobile phone for augmented reality applications?

User 1: 4

User 2: 3

2. Existing Approaches

What augmented reality(AR) applications have you used before?

User 1: Expeditions (AR teaching tool)

User 2: Snapchat filters

When was the last time you used it?

User 1: Last month

User 2: Yesterday, probably

Please describe your experience with this tool.

User 1: I can use it to visualise the cell structure for my students

User 2: Don't have to do much really, once Snapchat detects my face I can use the filters

3. Prior Knowledge as a Baseline

Which of the following indicates how much you know about the use of augmented reality applications in a museum?

[Option 1:] I don't know anything about that.

[Option 2:] I know a little, but I could learn more

[Option 3:] I am an expert.

User 1: Option 2

User 2: Option 2

4. Observing User Behaviours

What are you thinking as you view the home screen?

User 1: The background is nice

User 2: There are a lot of buttons

If you were looking for opening hours where would you expect to find it?

User 1: Events

User 2: About

If you were looking for a map of the museum where would you expect to find it?

User 1: Floor plan section

User 2: Floor plan

If you were looking for the AR feature where would you expect to find it?

User 1: AR button

User 2: AR

Tasks:

- Load the events page
- Go to the about tab,
- Click the floor plan button
- Point the camera to the bowl, Eamon De Valera, post box and the proclamation
- Take the quiz

How was the experience of using the product to complete these tasks?

User 1: Good

User 2: Fine

[Probe:] What are your thoughts on the language used?

User 1: Good, easy to comprehend

User 2: Good

[Probe:] How easy or difficult was it to navigate?

User 1: Wasn't hard, instructions when the camera came up explained a lot

User 2: Easy

[Probe:] What are your thoughts on the design and layout?

User 1: Design is nice, layout good

User 2: Design is nice

[Internal] How long did it take the user to complete these tasks?

User 1: 3 minutes 30 seconds

User 2: 3 minutes

5. End-to-End User Experience

How would you describe your overall experience with the product?

User 1: Enjoyable

User 2: Good

What did you like the most about using this product?

User 1: It was easy to use

User 2: The AR feature is very cool

What did you like the least?

User 1: Nothing really

User 2: The quiz, didn't get some of the easy ones

What, if anything, surprised you about the experience?

User 1: The 3D models looked impressive

User 2: The phone detected the objects really quickly

What, if anything, caused you frustration?

User 1: The instructions came up every time I went into the AR

User 2: Nothing in particular

Overall the feedback was good from both users. They both had no issues conducting the tasks set out for them. They both were able to easily navigate between screens in the app. The app itself ran very smoothly with no glitches.

5.2. System Testing

With regards to system testing for the app there were two choices, automated testing and manual testing. I chose to focus mainly on manual testing throughout the development process as in my opinion manual testing has more benefits. One of the benefits being getting hands on experience using the app. The system testing consisted mainly of debugging using log statements. The app was run on the emulator after adding each new UI component and each new function. For example, if a new button was added it was tested by adding a log statement to the button code. The app was then run on the emulator. I clicked the button to ensure the log statement came up in the console. This ensured the button was working before writing any code behind the button.

Target Operating System:

From the beginning of the project I set the target API level to 29 and the min API level is 24 (Nougat). These are specified in the app build.gradle file. This means the app can run on any device running from API level 24 to API level 29. I chose API level 24 as the min API level as most phones in circulation at the moment are at least API level 24. The device also has to have ARCore and sceneform to use the AR feature.

Testing on the Android Emulator:

An Android emulator is an Android Virtual Device(AVD) that represents a specific Android device. AVD are created in the AVD manager on Android Studio. To create a new device, you click create virtual device in the AVD manager. A list of virtual devices to choose from are then displayed (as shown in figure 44). The majority of my tests were carried out on a Pixel, API level 27. I also ran tests using the Nexus one and the Pixel XL as AVD's, both Oreo (27) and Pie(28). They both ran as expected.

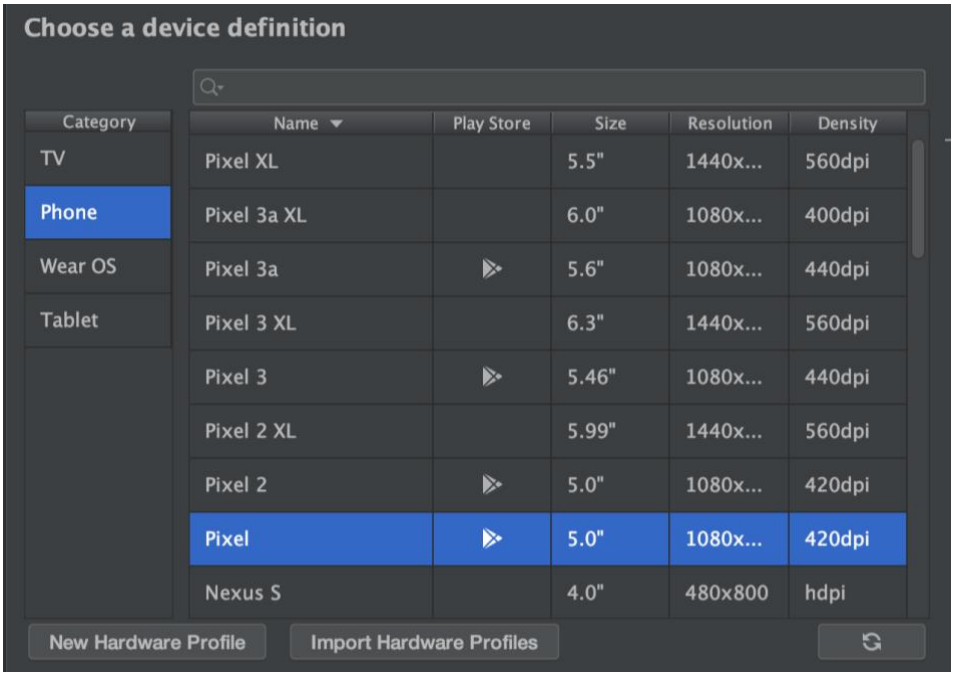

Figure 44: List of virtual devices to choose from in the AVD manager

Testing on the Physical Device:

As mentioned previously I only had an emulator to test the app in the early days but near the end of the development I got a physical android device. I felt it was important to carry out tests on both the emulator and physical devices. The physical device was an Android J3(API 28). The app worked exactly as expected on the physical device. I tested the app out over Wi-Fi connection and over 4G. There was very little difference in the running of the app between the two different connections. However, in the Trivia section the questions did load slightly faster over the 4G network, this was probably due to the mobile network connection being better than the Wi-Fi in my house. I also tested running the app on the physical device and the emulator concurrently. I went into the Trivia section on both devices and took the quiz, the questions loaded without any interference.

6. Project Management

6.1. Gantt chart

The tasks and completion dates associated with the development of my project are outlined below the Gantt chart (Figure 45).

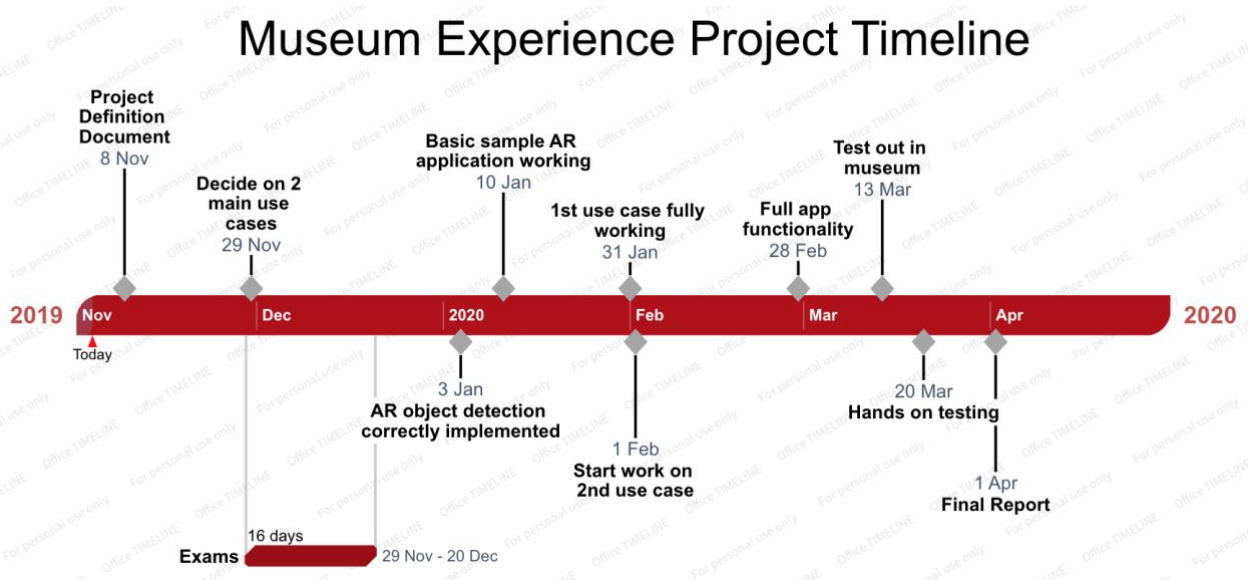

Figure 45: Project Gantt chart

6.2. Code management

A GitHub repository was set up at the beginning of the project to manage the project code. At the end of each day developing I pushed the code I had worked on that day along with a commit message describing the code to GitHub. Using GitHub everyday enabled me to keep track of different commits throughout the course of the project. This proved to be very useful in the later stages of the project. If I was working on a new feature and other parts of the app stopped working, I could simply look to the GitHub commits and revert back to previous days commit. This meant I would instantly have an older version of the app back up and running.

A link to the GitHub repository can be found here: https://github.com/johncummins/FYP

6.3. Communications

Over the course of the project my supervisor and I held weekly/bi-weekly meetings. If there was a prolonged period of time where we did not meet, I would send an email with an update on my progress to my supervisor. A google doc was set up where I use to keep minutes of each of our meetings and noted any research completed. A to-do list was maintained over the course of the project. This was updated every day, where completed tasks were crossed off and future tasks added in. The to-do list was also stored on the Google doc which was shared with my supervisor to keep both parties informed.

7. Challenges

7.1. Project Management

As a result of the android emulator giving me trouble, I was unable to meet the majority of the project deadlines I set out. I added two weeks on to each deadline to make new ones.

7.2. 3D Modelling Issues

One of my use cases extending the view of artefacts was a particularly difficult use case. I had to develop 3D models and I was completely new to 3D modelling. One of the models I had to create was a model of a bowl. The main issue developing this model was the artefact only had one handle left on it. I wanted to recreate a model of how it would have looked in its original form. I needed to find a way to add the second handle to the 3D model. I found Smoothie 3D, which allowed me to reflect the handle through the y-axis to make it appear as if it was on the other side of the bowl.

I tried to develop a model of the postbox from the postbox image using Smoothie 3D. However, it did not come out as expected. The postbox is a hexagon but the model appeared as a cylinder. Some of the colours from the background of the image were present on the postbox. Once I found the 3D model of the imperial postbox I then needed to make the exact same model in green which I achieved by tinting the mtl file from green to red. I also had some difficulty with the mtl files with both of the models. When I downloaded the post box obj, there was no mtl file with it. I contacted the website where I got it off to see if they could source the mtl file for me. They replied saying the textures were included in a tga file format not in the mtl format. The texture file had to be in the mtl format in order for it be imported as a sceneform object. As a solution I downloaded Blender, which is a 3D modelling tool. I imported the obj and added the tga file format to the model and then exported it including the texture as a mtl format.

7.3. Android Emulator Issues

The main feature in my project is the AR feature, so naturally it was my starting point in the implementation of the overall project. Before beginning work on my own AR feature, I downloaded a sample AR project from the Android developers page. It was a simple app that rendered 3D models of android widgets on layer on top of the real world. At the time, I didn't have an Android phone to test the app on, so I was using an Android emulator on my laptop. The emulator itself was functioning fine but when I ran the sample AR project, the app would crash. The crash would occur and I would receive an error relating to the OpenGL version. Open Graphics Library(OpenGL) is a cross platform, cross language API foo rendering 2D and 3D vector graphics.[8] I could not find a fix for the error anywhere online. I spent days looking for solutions and eventually asked a question on stack overflow about the error. A screenshot of the question I posted on Stack overflow is shown in figure 46. I got no reply from the Stack overflow question.

Figure 46: Stack overflow question I posted regarding emulator issue

I soon found out that I had OpenGL version 3.0 on my laptop and the required OpenGL version for an ARCore app to run on an emulator was OpenGL 3.1. I tried running the app on a desktop in a computer room in the college but the same issue arose as the desktops only had OpenGL 3.0. Eventually I came across a solution on Reddit, which suggested changing the dependency for ar sceneform dependency to 1.4, instead of the latest 1.15, I did this and I got it working.

8. Future work

For future work it would be useful to expand the reference image database. As I mentioned previously I did not manage to test the AR feature in the museum, instead I tested it showing images of the artefact on my laptop. The images being displayed on the laptop were the exact same as the reference images I uploaded to the augmented image database. This meant the artefact was correctly detected very quickly. To make the AR feature more realistic for the real world, I would need to take multiple images of each artefact and upload them all to the database. This would involve getting images from different angles and in different lighting environments. The bowl artefact is displayed in a standalone glass cabinet in the Galway city museum. This means it can be viewed from every angle, which in theory means I would need reference images from every angle to be included in the database. With more images in the augmented images database the app becomes more consistent when using it in an actual museum. I would have liked to put more time into developing 3D models. It was an aspect I quite enjoyed. I seen improvements in myself using these technologies as time progressed. Another week working on creating the 3D models could have proved beneficial but time was a limiting factor.

If someone else were to pick up this project and build on it, I would advise doing all the manual testing on a physical Android device. I conducted most of my testing on an emulator, the emulator worked perfectly but it was unreliable. As mentioned previously I had trouble getting the emulator up and running for Augmented reality apps. Some days it would just stop working for no reason. I would then have to delete that emulator and create a new one from scratch. This proved to be very frustrating. This project could be based in different museum in Ireland or indeed the world. It might prove very interesting and beneficial in art or science museum.

9. Acknowledgments

I would like to thank my supervisor Karen Young for her support and guidance throughout this final year project. In addition I would like to thank my lecturers who equipped me with the required skills and knowledge to complete this project successfully.

10. References

1 Ultimate Dinosaurs on Royal Ontario Museum n.d., accessed the of October, < https://www.rom.on.ca/en/exhibitions-galleries/exhibitions/past-exhibitions/ultimate-dinos/augmented-reality>.

2 Jennifer Billock 2017, Five Augmented Reality Experiences That Bring Museum Exhibits to Life, Smithsonian Magazine, accessed on the 14th of October, < https://www.smithsonianmag.com/travel/expanding-exhibits-augmented-reality-180963810/>.

3 DIA Lumin B-roll F on Detroit institute of arts 2017, accessed on the 12th October 2019, <https://www.youtube.com/embed/joam01lrsJU>.

4 Three tier architecture – complete overview on jReport 2019, accessed on 20th January 2020, < https://www.jinfonet.com/resources/bi-defined/3-tier-architecture-complete-overview/>.

5 Sceneform overview on Google developers 2020, accessed 20th January 2020, <https://developers.google.com/sceneform/develop>.

6 Augmented Image Database on Google developers 2019, accessed on 15th February, <https://developers.google.com/ar/reference/c/group/augmented-image-database>.

7 Arcoreimg tool on Google developers 2019, accessed on 15th February, <https://developers.google.com/ar/develop/java/augmented-images/arcoreimg>.

8 OpenGL 4.0 specification 2017, accessed on 1st February, <https://en.wikipedia.org/wiki/OpenGL>.