Camden View Mobile Application

Sheng-Wen Huang

Supervised by Dr. Graham Roberts

MSc Computer Science
September 18, 2019

This report is submitted as part requirement for the MSc Computer Science at UCL. It is substantially the result of my own work except where explicitly indicated in the text.

Department of Computer Science
University College London

Abstract

This project covers the development of an augmented reality mobile application which allows the design of new buildings to be viewed from every angle while standing close to the development site. The project is part of Industry Exchange Network (IXN) with Camden Council.

The application development was done in Unity and built on iOS platform. The programming language used was C#. A 3D model of a new building was constructed in Blender and imported into Unity. A connection between the mobile app and Camden Council open database was established via an API.

A mobile application was created successfully. It could fetch data from the database, populate the data in a list, allow users to select the planning application they were interested in, display the detail of the selected planning application, and show the 3D image of the new building in AR view.

Acknowledgements

I would like to express my very great appreciation to Professor Graham Roberts for his valuable suggestions and guidance during the development of this project. I would also like to offer my special thanks to Mr Richard Bond from Camden Council for his support and encouragement. My deep gratitude is also extended to Oliver Waterman, Sia Agarwal, and Akash Bhattacharya for their assistance and generosity.

Finally, I wish to thank my parents for their support throughout my study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project is about developing an Augmented Reality (AR) mobile application to allow the design for a new building to be viewed from different angles while standing close to the proposed location.

1.1 Problem Statement

This project is part of Industry Exchange Network (IXN) with Camden Council, which makes Camden Council the client of this project. Camden Council is the local authority of London Borough of Camden. Every planning and building development plans in London Borough of Camden have to be granted by Camden Council.

A formal request has to be submitted to the local authorities for permission of a new building or an extension of an existing one. The request is called planning application. Generally, the proposed drawings of plan and elevation will be submitted. All the applications are open to public online, and anyone is able to make comments to support or object to an application. Comments that are directly related to the proposal, which are called material considerations, are the only ones that will be considered. Material considerations include loss of sunlight or privacy, the design, size or height of the new buildings or extensions, the impact on the environment, etc. It is important that the authorities get feedback from the public before they approve or decline an application. However, it can be quite difficult for people to imagine the finished look and its impact on the surroundings based solely on the

2D drawings.

1.2 Aims and Goals

There are five main aims of this project:

- Learn about augmented reality (AR) technology and its application.
- Learn Unity scripting skill and to use Unity AR Foundation Plugin.
- Learn how to model AR objects.
- Learn about mobile GPS service.
- Learn to use the API (Application Programming Interface) of an existing database.

The goal of this project is to use augmented reality (AR) to solve the problem stated previously. An AR mobile application which shows the 3D model of the selected planning application can help the users to visualise the new buildings or extensions. Users can walk around the site and look at the the model from every angle. The visualisation gives them a better understanding of the planning application and how it affects the neighbourhood. The goals can be divided up into the following:

- The construction of a realistic 3D model of a planning application.
- The development of an iOS mobile application that connects to the Camden Council open database and displays a list of planning application.
- The implementation of mobile GPS service.

1.3 Approach

An iterative approach was adopted to develop this project. It started from the augmented reality part of the project. The major features required were plane detection, realistic AR model, and the ability to drag and drop the AR model on detected planes. The first iteration was an application that detected planes. The

second iteration added a feature that allowed users to drop numerous cubes on the detected planes. The dragging and dropping feature was later incremented. After the AR model was constructed and imported to Unity, the cube was replaced with the model. The next step was to connect the app to Camden Council's existing database and populate the list of planning applications. The list page was then integrated with GPS service and linked to the AR view.

1.4 Report Structure

In Chapter 1, the motivation is stated. The aims and goals of this project are also listed, followed by the approach that was used to solve the problem. Background research and related works are reviewed in Chapter 2. An introduction of the technologies implemented and the reasons for the decisions are also presented. A detailed problem statement will be given in Chapter 3, as well as a list of MosCoW requirements, use cases and the analysis of the requirements. Chapter 4 demonstrates the application architecture and explains the design and implementation of the application and decisions made. The testing and its results are mentioned in Chapter 5. At last, Chapter 6 evaluates the project and discusses about the potential and future plans of this project.

Chapter 2

Background Research

2.1 Background and Research Overview

Over the past few years, augmented reality (AR) technology has been growing rapidly. It has become very popular in the fields of gaming, retail industry, medicine, tourism, education and more. It is estimated that the global AR market will generate revenue of approximately USD 30 billion by 2023 [2].

AR is a technology that creates a digital layer to the real world. Unlike virtual reality (VR), AR does not create an entirely new virtual world. It expands the real world by adding computer-generated objects, such as 3D image, audio, or videos. There are four types of AR technologies: marker-based, markerless, projection-based, and superimposition-based [3] [4]. Marker-based AR recognises a marker, which can be a QR code or a picture, and initiates the digital generation [5]. On the other hand, markerless AR does not generate AR objects based on image recognition but location [6]. Therefore, markerless AR is also called location-based or position-based AR. It requires the GPS, compass, gyroscope and accelerator on the device to provide the location data of the user. The data is later used to determine the AR objects that the user sees. This type of applications are usually integrated with maps. Light projection technique is used in projection-based AR. Users can be allowed to interact with the projection on a physical object in the real world. For instance, hologram is an application of projection-based AR. Lastly, superimposition-base AR recognises a physical object and replaces it partially or

entirely with augmented objects. Therefore, object recognition plays a crucial part in superimposition-based AR.

Object recognition (superimposition-base AR) appeared to be a suitable way to position the AR models (the new buildings or the extension of existing buildings) in this project. The application recognises the adjacent buildings and calculates the location to place the AR model. ARKit Scanner app provided by Apple offers a simple way to scan a reference object using an iOS device. Before scanning, users have to define a bounding box to tell the app where in the real world to scan. After doing that, the users can start scanning by moving the device around and capturing the object from different angles. The result files can be exported and used by ARKit for detection [7]. However, the scanning app was not designed to scan large objects like buildings. Therefore, object recognition is not the appropriate technique to use.

Location-based AR was later considered. Since the longitude and latitude of every planning application were already stored in the Camden database, it is fairly simple to place the AR models on the right location. Nevertheless, this gave rise to another issue. Smartphone GPS is accurate only within a 4.9-meter radius under open sky [8]. The accuracy decreases when there are tall buildings, trees or bridges nearby that can block the signals. Needless to say, a higher accuracy is needed for the purpose of this project. The solution will be discussed in Chapter 4.

2.2 Related Works

One of the most successful AR mobile game is Pokémon Go. Pokémon Go is a location-based AR mobile game for both iOS and Android devices [9]. In the game, every player is a Pokémon Trainer whose mission is to catch wild Pokémon and evolve them so they can win battles. The locations of the player and nearby Pokémon are shown on a map. The player has to travel to the location in order to capture a wild Pokémon [10]. When the player encounters the wild Pokémon, it can be displayed in AR mode. In the AR mode, the Pokémon appears in the real world through the camera view, as if it is a real creature.

Another popular AR mobile game is Harry Potter: Wizards Unite. It is also a

geo-based AR game developed by WB Games San Francisco and Niantic, Int. [11]. Similar to Pokémon Go, players can view themselves and the ”Foundables” on a map. The player need to physically move to the location of Foundables to collect them. ”Cofoundable spells” are cast on the Foundables, the players have to break the spells to collect the Foundables. Exquisite AR images will show on the players' smartphones when they try to defeat the spells. Both Pokémon Go and Harry Potter: Wizards Unite display AR images according to the user's location. The AR images are placed on a detected plane to make it more realistic.

Mobile game is not the only application of location-based AR. StreetMuseum (currently unavailable [12]) produced by Museum of London takes users back to London in the 19th and 20th century. Users can click on the pin on a map to see the old image and its description. In 3D view, the camera view is partially overlaid with a historic image when the app recognises the user's location [13]. Once again, GPS data helps to achieve this task.

ARKi is an AR visualisation tool for architectural models [14]. Users can download existing models in the app or import new models (.fbx files) into the app, and view them in augmented reality view. The app detects surfaces which the users can place models on. The users can drag, scale and rotate the model to see it in detail. The purpose of this app is similar to that of Camden View. However, it allows the users to view the models wherever they want, and they can also scale and rotate the models freely.

2.3 Software Tools and Platforms

2.3.1 ARKit

Apple's AR development framework ARKit was introduced in 2017 with iOS 11. ARKit allows developers to build various AR applications using *”device motion tracking, camera scene capture, advanced scene processing, and display conveniences”* [15]. It is only supported on *”iOS 11.0 or later and an iOS device with an A9 or later processor”* [16]. The latest version ARKit 3 was released in 2019, with several groundbreaking features added. The major improvements include peo-

ple occlusion, motion capture, simultaneous front and back camera, multiple face tracking and collaborative sessions [17].

2.3.2 Unity and SceneKit

The two most commonly used tools for ARKit development are Unity and SceneKit. Unity is a cross-platform game engine that uses C# scripting API. It features powerful 3D content and animation, and therefore is used in 60 percent of AR or VR content [18]. ARKit can be used in Unity via Unity AR Foundation plugin [19]. SceneKit is Apple's framework for *"creating 3D games and adding 3D contents to apps using high-level scene descriptions"* [20]. Swift and Objective C are the primary languages. SceneKit is well-integrated with Mac OS and Xcode, it also provides Apple UI elements such as buttons, navigation bars and labels. It is particular suitable for experienced iOS developers or people who want to use iOS UI elements [21][22].

Unity was chosen to be the development toolkit in this project for three reasons. Firstly, Unity is a very mature tool for developing cross-platform 3D apps. There are numerous built-ins, assets and online resources that can help developers, while SceneKit is a fairly new product with less resources. Furthermore, as a powerful game engine, Unity is also able to deliver delicate 3D contents which is of paramount importance for this project. Unity supports many formats of 3D file, whereas SceneKit only takes Collada (.dae) and Wavefront (.obj) files. Last but not least, it is very simple to cross compile projects to many different platforms in Unity which can save a lot of time for developers. Although this project focuses on the iOS platform, it is important that the process of compiling to other platforms is simple and fast for client's future use.

2.3.3 Working in Unity

There are several important components in a Unity gameplay: Scenes, GameObjects, Prefabs, and Cameras. Each scene is a different level in the game. The scenes define the environment and components in the game. Every object in the current scene is destroyed when a new scene is loaded. Every object in a game is

called a GameObject. Once a GameObject is created, it can be saved as a Prefab for future use. The Prefab system is particularly useful when a GameObject is used in multiple places or scenes as the Prefab system can automatically sync all the copies. Finally, a Camera in Unity is used to determine how players see the game world.

2.3.4 AR in Unity

As mentioned above, AR Foundation plugin allows developers to access ARKit 3 in Unity. A basic AR scene in Unity normally contains an AR session object and an AR session origin object [23]. The life cycle of an AR experience is controlled by AR session. The session can be started, paused, resumed or stopped. The final position, orientation and scale of the trackable features in the Unity scene are transformed by AR session origin. In other words, it transforms the session space data provided by AR devices into Unity space. The AR camera in the scene should be a child of AR session origin so that the camera and the detected tackables can move together.

2.3.5 Scripting in Unity

Unlike regular programming, programmers do not need to write the code that runs the application because Unity does that on its own. The scripts and components control the GameObject that they are attached to. Together, their behaviour and interaction with each other determine the gameplay. To be more specific, Unity engine runs in a big loop. In each loop, Unity updates the frame. In order to do that, Unity reads every element including scripts, lights, and meshes in the scene, and processes all the information. The scripts give instructions to Unity, and Unity executes them in each frame as fast as it could [24].

C# is the language used in Unity scripting. It is an object-oriented scripting language which has variables, functions and classes. From Unity 2018.1, Unity is integrated with Visual Studio for Unity Community or Visual Studio Code. They help developers to debug and take shortcuts.

2.3.6 Visual Studio Code

There are three reasons that Visual Studio Code was chosen to be the IDE (Integrated Development Environment) used for Unity scripting. First, Unity supports opening script in Visual Studio Code and there is debugging extension available [25]. Second, Visual Studio Code is lighter and faster than Visual Studio for Mac, which is also integrated with Unity. The last reason is that Visual Studio Code offers many plugins that make many everyday tasks faster, such as Git support.

2.3.7 Xcode

Xcode is an Apple IDE designed to produce apps for Apple products, such as iPhone, Mac, and iPad [26]. Even though Unity allows developers to build cross-platform applications, it is still necessary to use Xcode when building an iOS mobile application. The workflow is:

1. Develop the app in Unity, and script in Visual Studio Code.
2. Build the app to the iOS platform in Unity.
3. Open the project generated by Unity in Xcode.
4. Build the project in Xcode and run it on an iPhone.

It is worth noticing that although it is possible to run apps on simulator in Xcode, ARKit is not available in iOS simulator since it requires access to an camera. Therefore, an iOS device has to be connected to the computer when the project runs.

2.3.8 Blender and AutoCAD

There are many 3D modelling software, including AutoCAD, SketchUp, and Blender. SketchUp and Blender are easy to learn and very intuitive, especially SketchUp. On the other hand, AutoCAD has a steeper learning curve. Since precision modelling can be easily done, AutoCAD is the most common tool used to construct 3D building models. AutoCAD is widely used in engineering, construction and architecture fields and is perfect for large-scale projects, whereas Blender is used more in animation and visual art design and it is more suitable for small-scale

projects. However, Blender is designed for rendering high-quality images [27][28]. Its Cycle engine promises to deliver *"stunning ultra-realistic rendering"* [29]. The powerful rendering ability is something that AutoCAD nor SketchUp can offer. It is desired that the AR images are realistic and detailed in this project. As a result, it was decided that Blender was the 3D modelling software for this project.

2.3.9 Camden Open Data API

All the planning application data is open to public for research, analysis and development purposes on Open Data Camden [30] [31]. Everything on Camden's Open Data portal can be accessed via Socrata Open Data API (SODA) [32]. The data can be accessed via the API endpoint of the data set. By changing the endpoint URL, users can specify the data format to be JSON or CSV as illustrated below.

```
1 https://opendata.camden.gov.uk/resource/2eiu-s2cw.json
2 https://opendata.camden.gov.uk/resource/2eiu-s2cw.csv
```

The data can be filtered by using simple filters or Socrata Query Language queries. Simple filters applies the self-describing feature of SODA to filter data. The task can be done simply by appending the filter to the API endpoint URL. For example,

```
1 https://opendata.camden.gov.uk/resource/2eiu-s2cw.json?
    decision_type=Granted
```

fetches the planning application which has "Granted" decision type [33]. It is clear that simple filters are normally used when the filter condition is simple. As an alternative, SoQL can achieve more complicated queries. Socrata Query Language (SoQL) is similar to SQL (Structured Query Language) but was specifically designed to access data online. Apart from the basic clauses that are similar to SQL clauses, there are numerous functions and keywords in SoQL that can help developers [34] [35]. For instance, planning applications that are within 5-meter-radius of the location 51°30'05.6"N 0°08'30.8"W (51.501558, -0.141901) will be fetched by the following URL.

```
1 https://opendata.camden.gov.uk/resource/2eiu-s2cw.json?
```

```
$where=within_circle(location, 51.501558, -0.141901)
```

2.3.10 Map

Google Maps provide Maps Unity SDK [36] which allows developers to easily extend their Unity environment and combine real-world maps with their mobile games. However, Google charges developers for the full version.

There are two other free map SDKs for Unity: Mapbox [37] and WRLD [38], which can be easily imported to Unity. Both support AR location-based mobile applications, and provide pre-made assets and example scenes. Developers can create 3D worlds, take advantage of their powerful location finder or position objects on maps. Placing objects on a map perfectly suits the requirement of this project.

2.4 Summary

This chapter provided an introduction to the four different types of AR technology. It then proceeded into a review of related works, including location-based AR games and city guide app as well as architecture visualisation tool. After this, the software tools used in this project and the reasons to use them were presented.

Chapter 3

Requirements and Analysis

3.1 Requirements Identification

As explained in Chapter 1, the purpose of this project is to build a mobile application that allows the public to visualise planning applications with augmented reality. This chapter will discuss the steps taken to identify the functional requirements.

3.1.1 Problem Statement

A formal request called planning application has to be submitted to the local authority to ask for permission for building a new building or an extension to an existing one. All the planning applications are open to everyone online so that anyone can comment on or object to them. Only material considerations, the comments that are directly related to the proposal, will be taken into account. Material considerations include loss of sunlight or privacy, the design, size or height of the new buildings or extensions, the impact on the environment, etc. Generally, only proposed plan and elevation drawings are submitted with the application, 3D models are not required. However, it is difficult to imagine whether there will be a block of sunlight or an invasion of privacy after a building is built. Perhaps the new building will be too massive that it affects the skyline of the neighbourhood, or the design does not match the adjacent buildings. Hence, providing useful comments just by looking at the 2D drawings is challenging. AR technology is very useful under this circumstance. An AR mobile application that displays the 3D models of planning

applications enables users to view the buildings or extensions from different angles and get a good understanding of their impacts once they are built.

3.1.2 Functional Requirements

From the problem statement presented in Section 3.1.1, the functional requirements were obtained. The requirements were organised using MoSCoW method. The MoSCoW method is a way to prioritise tasks by categorise them into four types: must-have, should-have, could-have and won't have. The MoSCoW requirement list is provided in Table 3.1. It was established in the first month of the development, and had been used as a guide throughout the development process.

Table 3.1: MoSCoW requirements

MoSCoW	**Description**
Must-have	Access to Open Data Camden database.
	List view of planning applications.
	Detail page of a planning application.
	Display AR image of at least one application as a proof of concept.
	The app should be able to locate the user's position via GPS service.
	The dimensions of AR models should be accurate.

MoSCoW	Description
	Users should be able to drag and drop the AR model on a detected plane so that it can be position on the right place.
Should-have	Map view of planning applications. Keywords searching function. Users should be able to view the submitted documents (e.g. elevation and plan drawings, etc.). Materials should be shown realistically on AR models. The AR view should only be available when the users are close enough to the development site.
Could-have	Show supporting information on AR images (e.g. height of the building, the material of the wall, etc.). Users can sort the list of applications by location, decision date, etc. Users can filter the list of applications by decision level, system status, etc. Show lighting and shadows of the AR models.

MoSCoW	Description
Won't-have	A system that can automatically transform pdf drawing files into 3D models. AR models of all the existing applications.

3.1.3 Use Cases

Use cases help identify the expected behaviour of the app by showing the interaction between the user and the system. The use cases were discussed with the client at the initial meetings. It was established that the users must be able to view the list planning applications nearby. Map view should also be provided if possible. Users can be able to search planning applications by keywords. After selecting a planning application, users should be able to view the details of the application, including development address, development description, decision type, decision date, system status, etc. If users are within 20 meters of the development address, an option of AR view will be provided. The use case diagram is illustrated in Figure 3.1. The list of use case titles and description is provided in Table 3.2, whereas the full use cases are listed in Appendix C.1.

3.2 Analysis

The requirements were then analysed to decide how the app should be designed. The requirements of having a list and a map view of planning application indicated that there should be two scenes where they could jump to and from each other, displaying the list and map view. Moreover, there should be a scene that showed the details of a selected planning application. From this scene, users should be able to go to the AR view where they could drag and drop the building on the right position.

ID	Use Cases	Description
UC1	Display planning application list.	User browses the list of planning applications.
UC2	Display planning applications on map.	User browses the planning applications on map.
UC3	Show details of a planning application.	User reads the detail of a planning application.
UC4	Display 3D model in AR mode.	User views the 3D model in AR mode.
UC5	Search planning applications.	User searches planning applications.
UC6	Filter planning applications.	User filters planning applications.
UC7	Sort planning applications.	User sorts planning applications.
UC8	Display complete planning application documents.	User reads the documents submitted by the applicant.

Table 3.2: Use Case List

3.3 Summary

A complete problem statement was reiterated in this chapter and functional requirements were established. Detailed functional requirements were given using the MosCoW method. Use cases were then obtained from the requirements. The requirements were then analysed, and the fundamental layout of this application was decided.

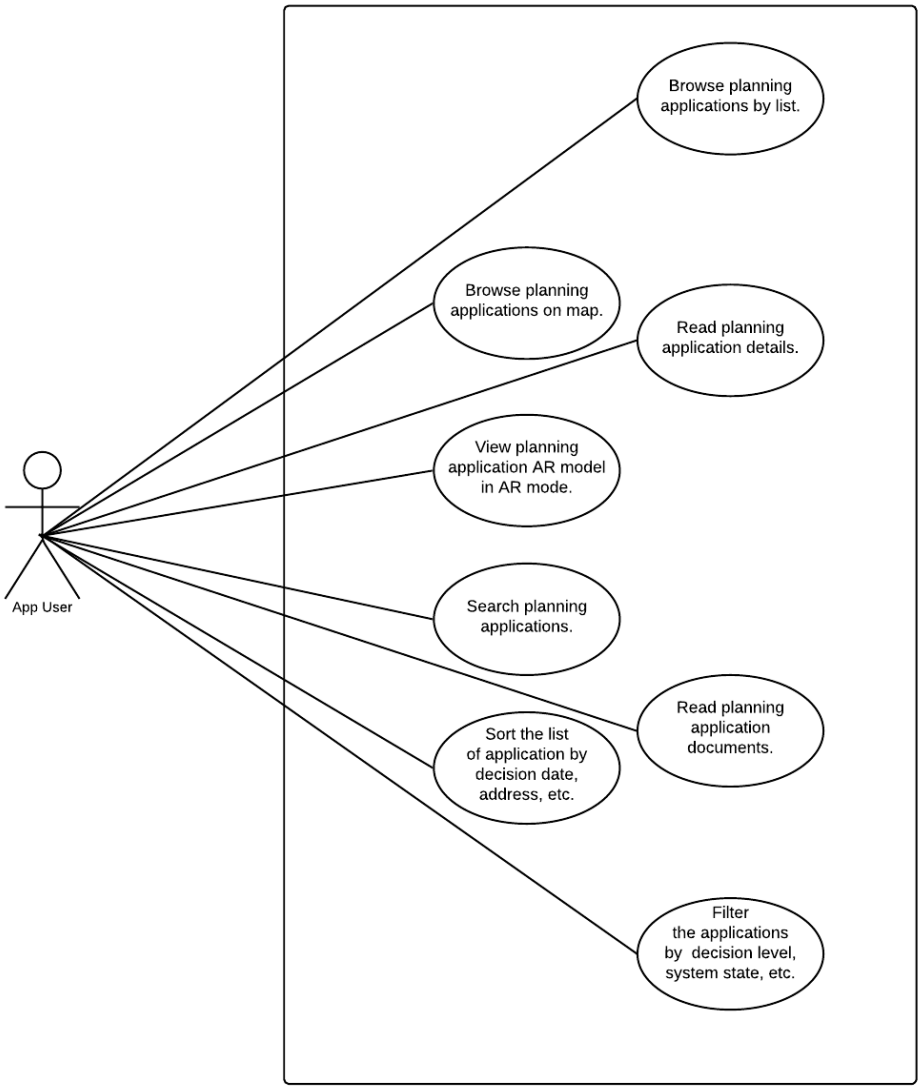


Figure 3.1: Use case diagram.

Chapter 4

Design and Implementation

This chapter introduces the design and implementation of Camden View app. The structure of the application is presented first, followed by the implementation of the main segments, augmented reality, 3D modelling and location service. The strategies and design decisions are also explained.

4.1 Application Structure

Figure 4.1 illustrates the application structure of Camden View app. The UI (User Interface) communicates to the database (Open Data Camden) through Socrata Open Data API. The list of planning applications are downloaded and displayed on the list and map view. The details of the selected planning application are shown on the detail page where AR mode can be activated. Only one AR model was built for this project as a proof of concept which was loaded straight from the scene. However, as presented in Figure 4.1, the plan is to construct an AR model generator and a cloud folder in the future (see Section 6.3). The AR model generator accesses the drawing files from Open Data Camden and generates a 3D model automatically, and then stores the model in a cloud folder and its cloud folder URL in the Camden database. When users activate the AR view, the system fetches the cloud folder URL of the model from Open Data Camden, downloads the model from the cloud folder, and loads it into the scene.

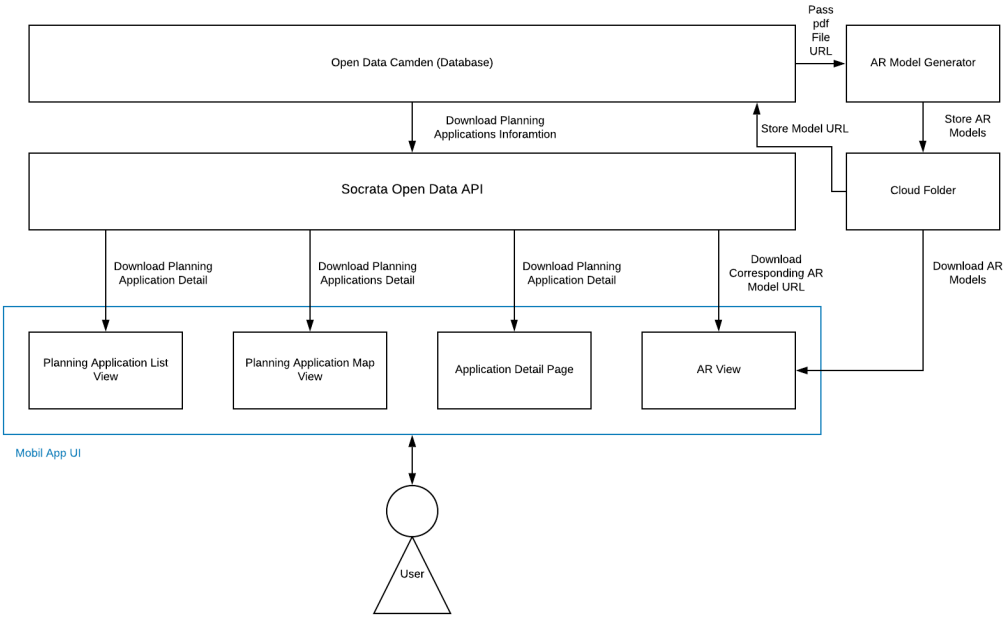


Figure 4.1: Application structure diagram.

4.2 Augmented Reality

As stated in Chapter 2.1, location-based AR suits the requirements of this project the most. Nevertheless, GPS data may not be accurate enough to satisfy the need. The solution was to allow users to drag and drop the AR image so that it can be positioned at the right location. The AR view cannot be activated unless the user is within 20 meters of the development site. After entering the AR view, the user can place the model on a detected horizontal plane, and rotate or drag it around on the plane. The following code [39] enabled the user to drop and drag the model. Unity provides an enum type *TouchPhase* that can detects the movement of finger touches [40]. *TouchPhase.Began* indicates a touch has been first detected, while *TouchPhase.Moved* determines whether the touch is moving. When a touch begins, and the model has not been placed, the model is placed at the touch location. When a touch moves, and its ray hits the previously placed model, the position of the model is updated. The rotation feature was implemented using Lean Touch on Unity Asset Store [41] by attaching *Lean Rotate Custom Axis* script to the 3D model prefab. The script allowed developers to specify the the required finger counts and the axis of rotation as illustrated in Figure 4.2.

```
if(Input.touchCount > 0)
{
    Touch touch = Input.GetTouch(0);
    touchPosition = touch.position;

    // the touch begins
    if(touch.phase == TouchPhase.Began)
    {
        Ray ray = arCamera.ScreenPointToRay(touch.
           position);
        RaycastHit hitObject;
        if(Physics.Raycast(ray, out hitObject))
        {
            placementObject = hitObject.transform.
               GetComponent<PlacementObject>();
        }
        if(arRaycastManager.Raycast(touchPosition, hits,
           UnityEngine.XR.ARSubsystems.TrackableType.
           PlaneWithinPolygon))
        {
            Pose hitPose = hits[0].pose;
            if(placementObject == null && !placedAlready)
            {
                // if not yet created, place the model
                placementObject = Instantiate(
                   placedPrefab, hitPose.position,
                   hitPose.rotation).GetComponent<
                   PlacementObject>();
                placedAlready = true;
            }
        }
    }
```

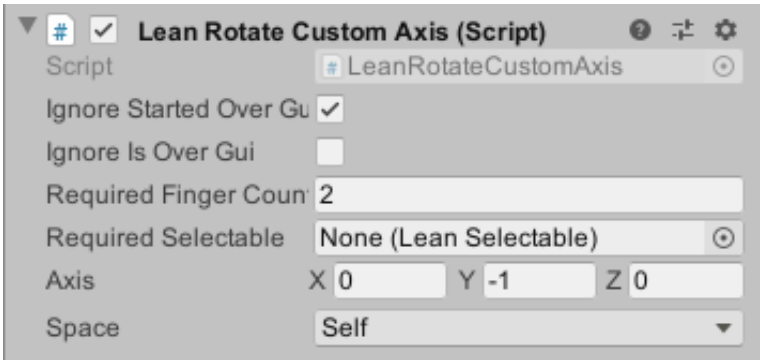


Figure 4.2: Lean Rotate Settings.

```
26      if(touch.phase == TouchPhase.Moved)
27      {
28          if(arRaycastManager.Raycast(touchPosition, hits,
                UnityEngine.XR.ARSubsystems.TrackableType.
                PlaneWithinPolygon))
29          {
30              // created and attemp to drag it
31              Pose hitPose = hits[0].pose;
32              if(placementObject != null)
33              {
34                  // update the postion of model
35                  placementObject.transform.position =
                        hitPose.position;
36                  placementObject.transform.rotation =
                        hitPose.rotation;
37              }
38          }
39      }
40  }
```

4.3 3D Modelling

A new residential development application which is located at 152 Royal College Street was chosen as a proof of concept. It was chosen because the location was convenient for site visits and the size of the building made it a great candidate for demonstration. The proposal was an *"erection of a 4 storey building including excavation of basement to provide one 3 bedroom flat (Class C3) and one 2 bedroom live/work unit"* [42]. The materials the developers planned to use were specified in a design and access statement. A model was constructed from the proposed floor plans and elevations in Blender. The textures of the building were rendered by using Blender's Cycle engine with the documents as a reference. Cycle is a *"physically-based path tracer for production rendering"* [29]. There are unlimited possibilities with Cycle node editor. Users have all the control they need over a material.

There were three main materials used in the model: concrete, bronze, and glass window. To create visual coherence and harmony with the adjacent buildings, the concrete used on the external finish was designed to be tinted with a rusty red. The effect was achieved in Blender by overlaying a rusty colour on a cement texture image as shown in Figure 4.3. Figure 4.4a is the original cement image and Figure 4.4b is the rendered result. Window frames and balustrade had bronze finish. To create the reddish-brown metal look, more complicated nodes were required. First, Voronoi texture generated a random Voronoi pattern, which was to partition a plane with n points into polygons [43]. It was specifically useful to create convincing metal effects [44]. The larger the scale parameter was, the more points were generated. Second, Musgrave texture was used to add an advanced procedural noise texture [45]. The Musgrave and Voronoi texture were then combined and used as the roughness of Glossy BSDF node which created reflective metal effect [46]. Furthermore, the Lambertian and Oren-Nayar diffuse reflection was added by the Diffuse BSDF node [47]. The settings of this texture is illustrated in Figure 4.5, while the final result is presented in Figure 4.6. Windows had a transparent and reflective texture which would be affected by the light. Therefore, the texture was not created in Blender but in Unity. The transparent rendering mode was selected

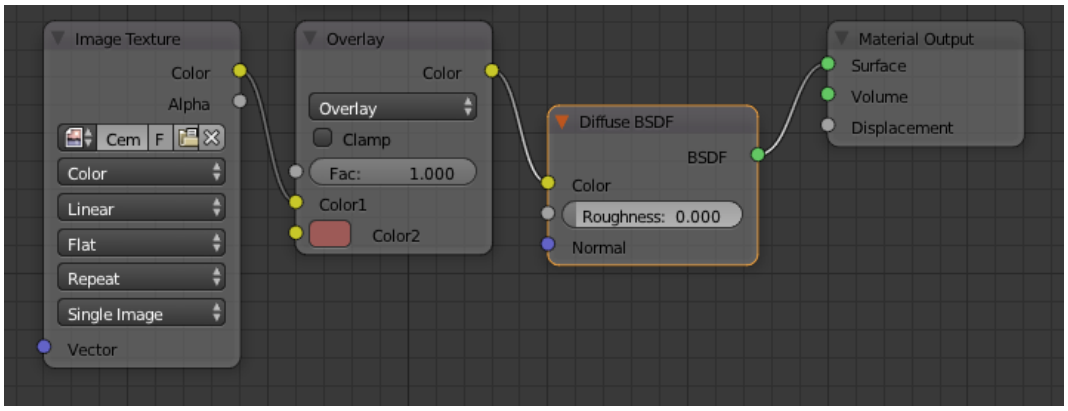
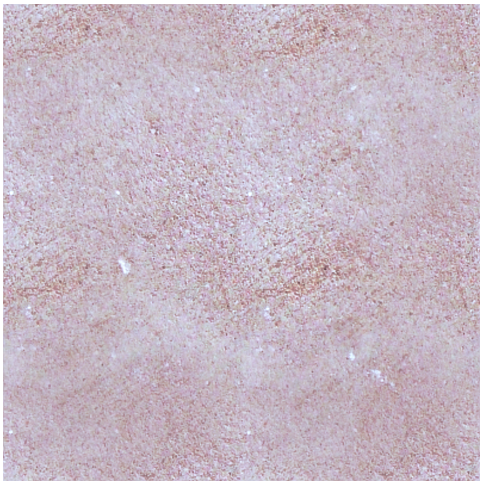


Figure 4.3: Rusty concrete Effect in cycle.

(a) Cement texture.



(b) Rendered rusty concrete.

Figure 4.4: Concrete wall effect.

as Figure 4.7 shows, and a reflection probe GameObject which captured the view of its surroundings was also added to the model [48].

In spite of all the hard work that was done to create the realistic materials, the materials made with Cycle engine could not be imported with the model into Unity. The solution was to *bake* the materials in Blender [49] first. Render baking is to render the shaders or lighting and generate an image that can be used later. Since the result is a pre-rendered texture image, the lighting and shadows are all fixed and static. Extra caution should be taken when baking effects that involves lighting. Baking image textures not only saved the rendering time and resources, but also allowed the Cycle shaders to be exported to Unity. The following steps were the process for using baked texture in Unity.

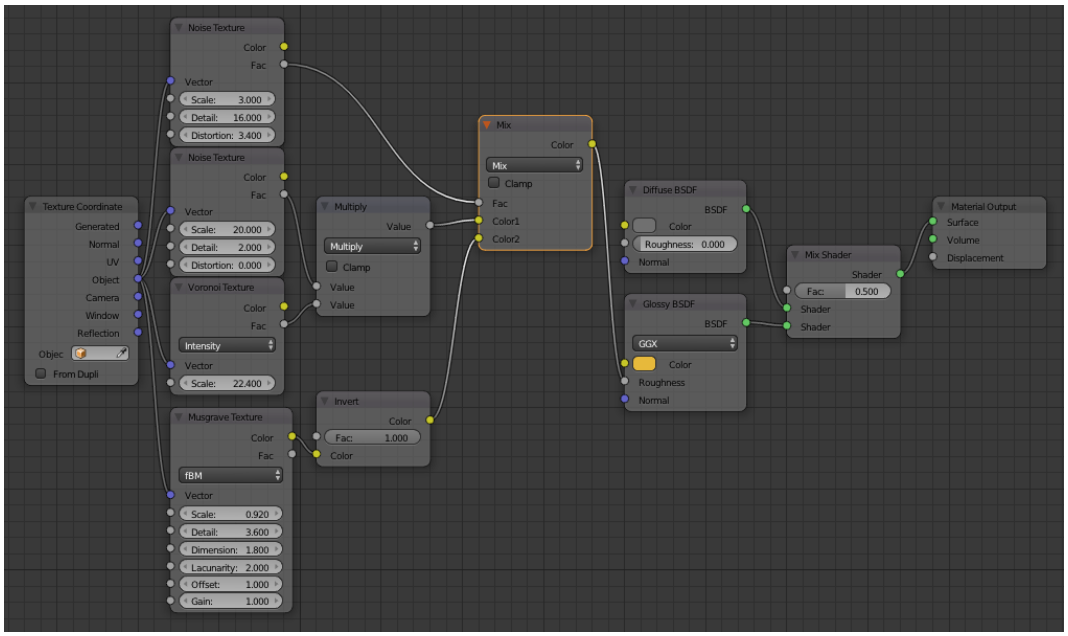


Figure 4.5: Bronze metal effect in cycle.

Figure 4.6: Bronze metal texture.

1. Bake the shader in Blender, and save the result image.

2. Create a new material in Unity.

3. Change the *Shader* to Mobile/Diffuse, and select the saved image texture generated by Blender as presented in Figure 4.8.

4. Apply the material to the desired components of the prefab model.

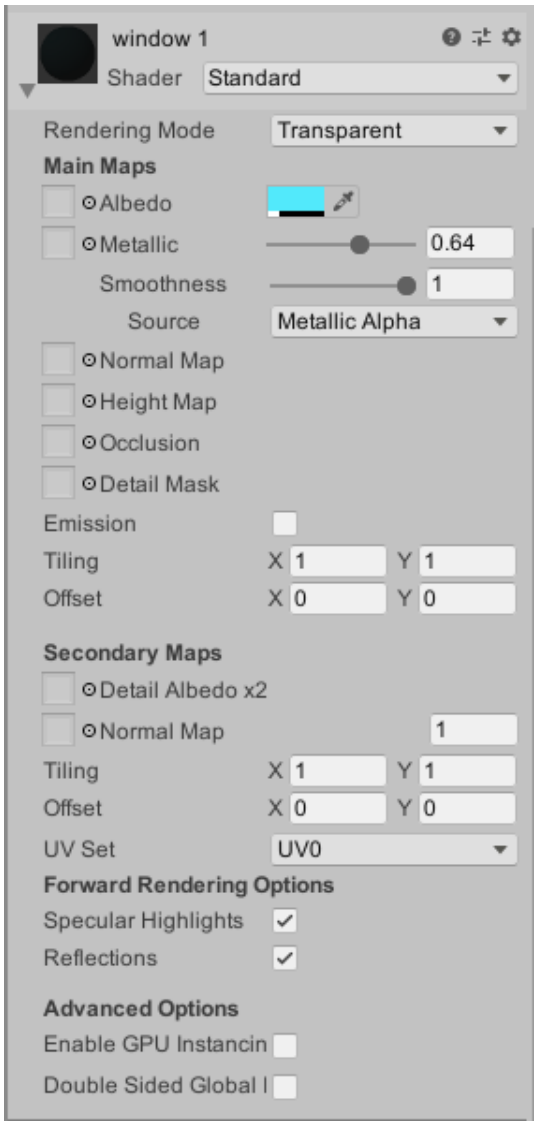


Figure 4.7: Glass window material setting in Unity.

4.4 Location Service

As stated in Section 2.3.10, Mapbox and WRLD SDKs both were powerful tools designed for location-based AR mobile app. They could be used to build the page where the location of planning applications and users was shown on a map. The first attempt was to use Mapbox which provided a location-based game template. The *LocationBasedGame* prefab was configured with the *LocationProvider*, Map, and *PlayerTarget* objects as illustrated in Figure 4.9. The *PlayerTarget* object indicated the user's location on the Map object, while the *LocationProvider* automatically accessed the GPS and positioned the user on the map. The prefab worked

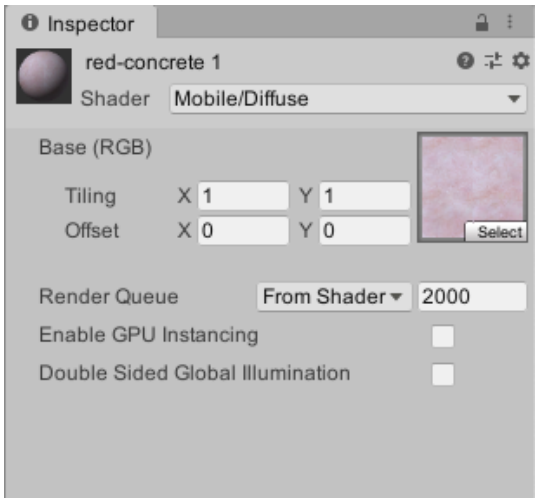


Figure 4.8: Using baked texture in Unity.

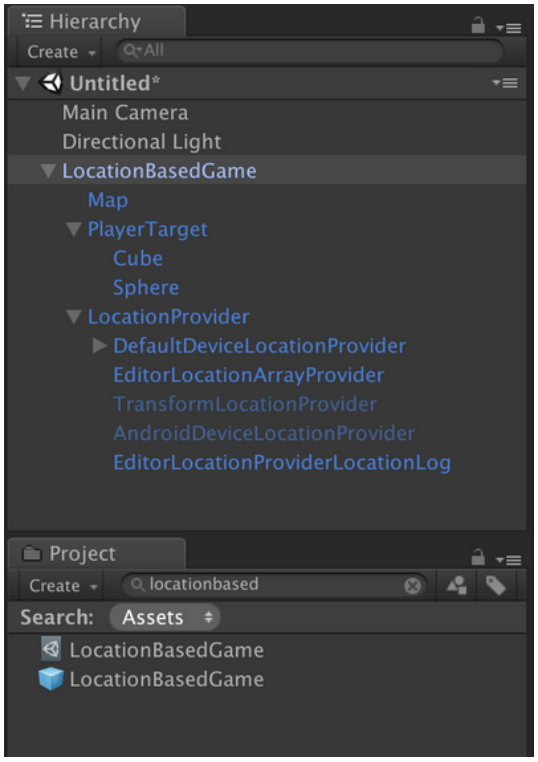


Figure 4.9: *LocationBasedGame* prefab provided by Mapbox [53].

well in Unity, however, the project had a building failure in Xcode. The error message was: *"'MapboxMobileEvents/MapboxMobileEvents.h' file not found"*, which indicated one of the header files could not be found. This issue was raised numerous times on Mapbox's GitHub repository [50] [51] [52]. Unfortunately, the solutions provided in the discussion did not work.

After failing to implement Mapbox, the focus was shifted to WRLD. Similar to Mapbox, template scenes and prefabs were included in the SDK. Everything worked in Unity, and it also built successfully. Nevertheless, an error message *"You must rebuild it with bitcode enabled (Xcode setting ENABLE_BITCODE), obtain an updated library from the vendor, or disable bitcode for this target. for architecture arm64"* showed in Xcode. The issue could be found on WRLD's GitHub [54], but once again, the solution provided did not solve the problem.

Several attempts were made to fix the bugs, however, these bugs seemed to be related to the way Unity produced Xcode projects [52]. Taking the limited time schedule into consideration, the implementation of the map view had to be put into future work.

Despite the unsuccessful attempts at creating a map view in the app, it was still necessary to implement the location service functionality. The *LocationService* API [55] in Unity allowed developers to retrieve GPS data. It is worth noting that *NSLocationWhenInUseUsageDescription* is required for GPS service to work on iPhones [56]. To provide the usage description, *Location Usage Description* in Unity player settings needed to be filled, as demonstrated in Figure 4.10. The sample code provided by Unity [55] was used to create a public coroutine *StartGPS()*. A coroutine is a special function that executes across frames [57]. *StartGPS()* was then called in another coroutine *StartRunningGPS()* where the Camden Open Data API was called. Since GPS data was used when connecting to the database, it is essential that the *StartGPS()* coroutine completed before database API was called. In order to do so, *yield return* was required. The *yield return* line was where the execution paused and be resumed the next frame. Therefore, the code below would not be executed until *StartGPS()* was finished.

```
1 IEnumerator StartRunningGPS()
2     {
3       // waits for GPS to terminate
4         yield return StartCoroutine(GPS.StartGPS());
5
```

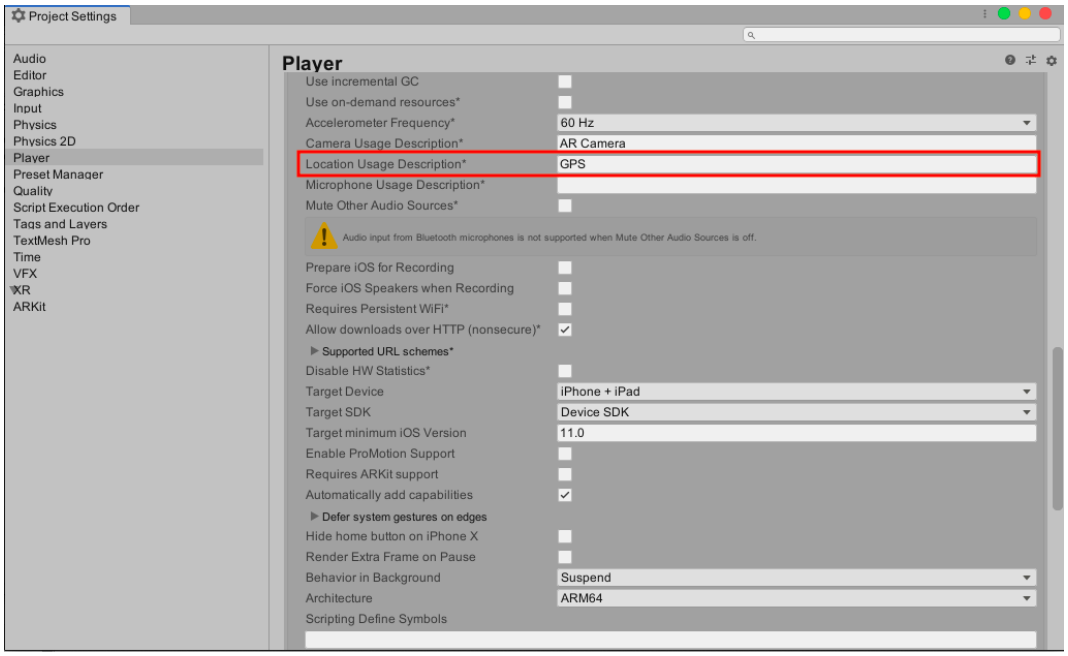


Figure 4.10: GPS settings in Unity.

```
6         // connect to database
7         string query = "?$where=within_circle(location, "
              + GPS.coordinate.x + ", " + GPS.coordinate.y
              + ", 2000)";
8         string url = "https://opendata.camden.gov.uk/
              resource/2eiu-s2cw.json" + query;
9         Debug.Log(url);
10        StartCoroutine(GetRequest(url));
11
12        // A non-existing page.
13        StartCoroutine(GetRequest("https://error.html"));
14    }
```

4.5 Population of Planning Applications

A list of planning applications were populated on the planning application list page. A script *ApplicationScrollList* was attached to a *Scroll View GameObject*. In the *ApplicationScrollList* script, GPS service and database API were called as de-

scribed in Section 4.4. The *GetRequest(url)* coroutine made a web request to the database API and got a response in JSON. The data was an array of objects. Nevertheless, the API in Unity *JsonUtility.FromJson* [58] that handled JSON data could not recognise objects in arrays. A class found on Unity Forum called *JsonHelper* was written to address this issue [59]. It wrapped the received JSON data in another object, and then created a wrapper class so that *JsonUtility.FromJson* could work properly.

```
1  public class JsonHelper
2  {
3      public static T[] getJsonArray<T>(string json)
4      {
5          string newJson = "{ \"array\": " + json + "}";
6          Debug.Log("newJson" + newJson);
7          Wrapper<T> wrapper = JsonUtility.FromJson<Wrapper
               <T>> (newJson);
8          return wrapper.array;
9      }
10
11     [Serializable]
12     private class Wrapper<T>
13     {
14         public T[] array;
15     }
16 }
```

To use *JsonHelper*, another class *ApplicationData* with all the planning application attributes was created.

```
1 ApplicationData[] jsonArray =
2     JsonHelper.getJsonArray<ApplicationData>(jsonResult);
```

JsonHelper.getJsonArray returned an array that contained *ApplicationData* objects

to *jsonArray*. Each object could be fetched by using a *foreach* loop. Inside the *foreach* loop, a function *AddButtons* was called.

AddButtons dynamically generated buttons [60]. There were two ways to achieve this. One was to instantiate and destroy objects every single time. However, creating and destroying objects was expensive, and would slow the application down. Therefore, it would be more ideal to use object pooling [61]. Instead of instantiating and destroying objects constantly, object pooling reused the objects. After pooling the object, a button was set up. The current *ApplicationData* was passed to *Setup* function so that it could later be passed to *ButtonClicked* function when the button was clicked. *ButtonClicked* function activated the panel that displayed the details of the application.

```
1  private void AddButtons(ApplicationData applicationItem)
2  {
3      ApplicationData item = applicationItem;
4      GameObject newButton = buttonObjectPool.GetObject();
5      newButton.transform.SetParent(contentPanel, false);
6
7      SampleButton sampleButton = newButton.GetComponent<
           SampleButton>();
8      sampleButton.Setup(item, this, detailPanel);
9  }
10
11 public void Setup(ApplicationData currentItem,
12     ApplicationScrollList currentScollList,
13     GameObject detailPanel)
14 {
15     item = currentItem;
16     title.text = item.development_address;
17     detail.text = item.development_description;
18
19     scrollList = currentScollList;
```

```
20      button.onClick.AddListener(()
21          => ButtonClicked(item, detailPanel));
22  }
23
24  public void ButtonClicked(ApplicationData currentItem,
25      GameObject detailPanel)
26  {
27      detailPanel.SetActive(true);
28      AppDetail detail
29        = detailPanel.GetComponentInChildren<AppDetail>();
30      detail.display(currentItem);
31  }
```

The page that displayed the application details was implemented in the same scene as the list of applications, and there were two reasons for that. The first reason was to avoid having to load the data from the database every time the user returned to the list page since everything would be destroyed when scenes changed. Secondly, putting them in the same scene could save the trouble of passing the current *ApplicationData* object between scenes.

4.6 Summary

In this chapter, the ideal structure of this application was elaborated. Due to the limited resources, the AR model generator was not built, and the 3D model was not stored on a cloud folder. A feature that allowed the user to drop, drag and rotate the 3D model in AR mode was added to compensate for the inaccuracy of GPS. Realistic textures of the 3D model were produced with Blender Cycle engine. The implementation of location service and data population were also described in this chapter.

Chapter 5

Testing

Software testing is an evaluation process that intend to discover if a software meets the requirements and if there is any fundamental defect. This chapter explains the test strategy used for testing the application. The steps of several test cases and results are also explored in detail.

5.1 Test Strategy

Test strategy is an outline of the testing approach adopted in the process of software development. The test strategy for this project was functional testing. Functional testing is a type of black box testing where code structure is not assessed. The functions of a software is tested by giving it an input and checking the output. Functional testing only analyses whether the requirements of the software are met, the process of the execution is not examined. There are two different ways to perform the tests, automation testing and manual testing. Due to the limited scope of this project, the application was tested manually. The testing procedure was:

1. Identify the function that the application was expected to fulfil and establish test cases.

2. Determine the input and output.

3. Execute the test.

4. Compare the actual output with the anticipated output.

5.2 Example Test Cases

A few fundamental test cases and the test steps are listed in this section, while the full test cases can be found in Appendix C.2.

5.2.1 Detecting Plane

The precondition of this test case was to have the planning application list page opened and one planning application selected. The user also had to be at least 20 meters away from the proposed site. The test was performed following these steps:

1. Open the app.
2. Select one planning application.
3. Click on the AR mode button.
4. Move the device around and see if horizontal planes are detected.

Horizontal planes should be detected and bounded by lines and meshes on the screen which is demonstrated in Figure 5.1.

5.2.2 Dropping Model in AR Mode

Having the AR mode activated and planes detected were the precondition of this test case. The test steps were:

1. Click on the AR mode button.
2. Drop the model on detected planes by tapping the screen with one finger.

The expected result was to have the model placed on the location where the user tapped. Once a model is dropped, no other models could be placed. The test result is presented in Figure 5.2

5.2.3 Moving Model in AR Mode

In order to test this function, a model had to be placed in the AR mode first. The test procedure was:

1. Drop the model on detected planes by tapping the screen with one finger.

Figure 5.1: Plane detection.

Figure 5.2: Dropping model.

Figure 5.3: Rotating model.

2. Drag the model with one finger.

It was anticipated that the user could drag the model anywhere on the detected planes with one finger.

5.2.4 Rotating Model in AR Mode

The precondition was having dropped an AR model on a detected plane.

1. Drop the model on detected plane by tapping the screen with one finger.

2. Rotate the model with two fingers.

The model should rotate along the y-axis (the axis perpendicular to the detected plane) on the detected plane and stay on the plane while rotating for this test to be considered successful. Figure 5.3 illustrates the test result.

5.3 Results

All the actual results matched the expected output and the same results could all be obtained repeatedly. This indicated that all tests were successful and the software could function properly.

5.4 Summary

This chapter went into detail on the testing of the software. Functional testing was the test strategy for this software. Only the functions of the application were evaluated, and how the code was executed was not taken into consideration. The function was identified first, and the input / output were determined. After that, the test was run, and the results were compared to the expected results. Several test cases were listed to give readers a better idea of how the tests were executed. All of the tests were passed, and the software was functioning well.

Chapter 6

Conclusion and Project Evaluation

This project proposed an AR visualisation tool which allows the design of a new building or extension to be viewed from all angles. This mobile application can help the public to get a better understanding of the new construction plan and make more constructive suggestions.

6.1 Achievements

The main goals set in Chapter 1 were mostly completed. An iOS mobile application that could connect to the Camden Council open database was created. The application could list the planning applications that were sorted according to the location of the user and allow the user to select and view the details. A realistic 3D model of a new building was built and could be viewed in the AR mode. As mentioned in Section 4.3, the glass material had to be generated in Unity to create the transparent and reflective effect. The effect was quite good under the sun. Features that enabled users to drop, drag and rotate the model so that it was placed at the right location were implemented. GPS service was also developed so that a customised service could be provided.

The aims were also accomplished. The basics of AR technology was learned. The skills at building an AR mobile application with GPS service in Unity were acquired. The knowledge of how to communicate with a database via APIs was also fully grasped. Finally, 3D modelling software and the workflow of exporting the complete model to Unity were both learned.

6.2 Critical Evaluation

In general, the application was implemented well. It had all the essential functions although more features could have been added. The application was built on iOS platform, but would be simple for the client to compile it to any other platforms in Unity in the future. The MoSCoW requirements were assessed to check how much was done. The list of all the requirements and their status were presented in Table 6.1. All of the must-haves and half of the should-haves were implemented.

There are three aspects that constitute this mobile application, the AR mode, the connection to the database, and the GPS service. This section also evaluates the project from those three aspects.

Table 6.1: MoSCoW requirements assessment

MoSCoW	Description	Status
Must-have	Access to Open Data Camden database.	Implemented
	List view of planning applications.	Implemented
	Detail page of a planning application.	Implemented
	Display AR image of at least one application as a proof of concept.	Implemented
	The app should be able to locate the user's position via GPS service.	Implemented
	The dimensions of AR models should be accurate.	Implemented

MoSCoW	Description	Status
	Users should be able to drag and drop the AR model on a detected plane so that it can be position on the right place.	Implemented
Should-have	Map view of planning applications.	Not implemented
	Keywords searching function.	Not implemented
	Users should be able to view the submitted documents (e.g. elevation and plan drawings, etc.).	Not implemented
	Materials should be shown realistically on AR models.	Implemented
	The AR view should only be available when the users are close enough to the development site.	Implemented
Could-have	Show supporting information on AR images (e.g. height of the building, the material of the wall, etc.).	Not implemented
	Users can sort the list of applications by location, decision date, etc.	Not implemented
	Users can filter the list of applications by decision level, system status, etc.	Not implemented

MoSCoW	Description	Status
	Show lighting and shadows of the AR models.	Not implemented
Won't-have	A system that can automatically transform pdf drawing files into 3D models.	Not implemented
	AR models of all the existing applications.	Not implemented

6.2.1 AR Mode

The dimensions of the 3D model were accurate. Most textures of the buildings were well simulated according to the design and access statement submitted by the developers. Moreover, the 3D model was not stored on a cloud folder but was implemented in the AR view scene. The same AR model would be shown in every AR view. Furthermore, there was no shadow effect in the AR view and no supporting information shown on the AR image. Finally, when users exited the AR mode, they were taken to the list page instead of the detail page. The ideal situation would be taking the users back to the detail page in case they wanted to read more about this planning application.

6.2.2 Connection to Database

The database was connected via Socrata Open Data API. The data was sorted according to the distance between the planning application and the user's location. However, custom filter and search features were not implemented. Fortunately, filter and search functions are highly connected and can be implemented together in the future.

6.2.3 GPS service

The application had access to the user's location, and the information was used to sort the list of planning applications displayed on the screen. If the users were

not within 20-meter-radius of the selected planning application site, AR mode could not be activated. Therefore, the GPS service helped the users to know the closest development site and if they were close enough to it to place the model at the right position. Nevertheless, since the map view was not developed successfully, the users could not see the development site location relative to theirs. Moreover, the application could not place the model automatically on its development site. The users had to place the model at the right address manually.

6.3 Future Work

If there were another six months, a system that could automatically generate 3D models could have been built. The models would have been stored on a cloud folder, and the URL of the cloud folder would have been added to Open Data Camden database. The shadow effect should have been implemented in Unity. Furthermore, keyword search, filter and order functions should have been added. A map which showed the location of the user and the planning applications should have been developed. The user could zoom and pan the map, and select certain planning application to enter the detail page. The possibility of displaying the model at the exact address automatically in the AR mode should have been explored. Another area that could have been done was to allow the users to make comments on planning applications via this mobile application.

6.4 Conclusion

Overall, this project was a success. The goals and aims set in Section 1.2 were largely achieved. AR technology, mobile application development skills, new software as well as programming languages were learned. The client was satisfied with the results and they were confident that they could easily extend the application in the future.

Bibliography

[1] gandalf3. How to make a bronze material in cycles? https://blender.stackexchange.com/a/18351.

[2] Market Research Future. Augmented reality market is expected to reach approximately usd 30 billion by 2023. https://www.marketresearchfuture.com/press-release/augmented-reality-market.

[3] Jyoti Gupta. How businesses are inclining towards types of augmented reality to enhance the consumers interaction. http://www.quytech.com/blog/type-of-augmented-reality-app/.

[4] ThinkMobiles. What is augmented reality (ar) and how does it work. https://thinkmobiles.com/blog/what-is-augmented-reality/.

[5] AnyMotion. Marker augmented reality. https://anymotion.com/en/wissensgrundlagen/augmented-reality-marker.

[6] Sonia Schechter. What is markerless augmented reality? https://www.marxentlabs.com/what-is-markerless-augmented-reality-dead-reckoning/.

[7] Apple. Scanning and detecting 3d objects. https://developer.apple.com/documentation/arkit/scanning_and_detecting_3d_objects.

[8] Navigation U.S. National Coordination Office for Space-Based Positioning and Timing. Gps accuracy. https://www.gps.gov/systems/gps/performance/accuracy/.

[9] Inc. Niantic. Pokmon official website. https://www.pokemongo.com/en-gb/.

[10] Inc. Niantic. Finding and catching wild pokmon. https://niantic.helpshift.com/a/pokemon-go/?p=web&s=getting-started&f=finding-and-catching-wild-pokemon&l=en.

[11] Inc. Niantic. Harry potter: Wizards unite official website. https://www.harrypotterwizardsunite.com.

[12] Museum of London. Museum of london apps. https://www.museumoflondon.org.uk/discover/museum-london-apps.

[13] Ellie Zolfagharifard. Streets of london now... and then: Stand still and picture yourself in history with app that creates hybrid images of present and past. https://www.dailymail.co.uk/sciencetech/article-2567739/Streetmuseum-app-creates-hybrid-images-London.html.

[14] Lidija Grozdanic. The top 5 virtual reality and augmented reality apps for architects. https://www.archdaily.com/878408/the-top-5-virtual-reality-and-augmented-reality-apps-for-archite

[15] Apple. Arkit framework. https://developer.apple.com/documentation/arkit.

[16] Apple. Verifying device support and user permission. https://developer.apple.com/documentation/arkit/verifying_device_support_and_user_permission.

[17] Apple. Get ready for arkit 3. https://developer.apple.com/augmented-reality/arkit/.

[18] Unity. Public relations. https://unity3d.com/public-relations.

[19] William Todd Stinson. Ar foundation support for arkit 3. https://blogs.unity3d.com/2019/06/06/ar-foundation-support-for-arkit-3/.

[20] Apple. Scenekit framework. https://developer.apple.com/documentation/scenekit.

[21] Neil Mathew. Unity vs scenekit: which tool you should use to build your arkit app. https://www.freecodecamp.org/news/unity-vs-scenekit-which-tool-you-should-use-to-build-your-arkit

[22] Dan Wyszynski. The match-up: Scenekit or unity for arkit? https://hackernoon.com/scenekit-or-unity-for-arkit-3fa3566d4d32.

[23] Unity. About ar foundation. https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@1.0/manual/index.html.

[24] Unity. Coding in c# in unity for beginners. https://unity3d.com/learning-c-sharp-in-unity-for-beginners.

[25] Lukasz Paczkowski. Replacing monodevelop-unity with visual studio community starting in unity 2018.1. https://blogs.unity3d.com/2018/01/05/discontinuing-support-for-monodevelop-unity-starting-in-unity-20

[26] Apple. Xcode help. https://help.apple.com/xcode/mac/current/.

[27] ibrahimk. Autocad vs blender: Head to head comparison. https://www.computeraideddesignguide.com/autocad-vs-blender/.

[28] Andrew Simon Thomas. Cad vs. modeling: Which 3d software to choose? https://www.shapeways.com/blog/archives/27653-cad-vs-modeling-which-3d-software-to-choose.html.

[29] Blender. Rendering and beyond. https://www.blender.org/features/rendering/.

[30] Camden Council. Open data camden. https://opendata.camden.gov.uk.

[31] Camden Council. Open data camden planning application. https://opendata.camden.gov.uk/Environment/Planning-Applications/2eiu-s2cw/data.

[32] Camden Council. Camden open data api. https://opendata.camden.gov.uk/stories/s/Camden-Open-Data-API/tf35-tpy4.

[33] Socrata. Simple filtering. https://dev.socrata.com/docs/filtering.html.

[34] Socrata. Queries using soda. https://dev.socrata.com/docs/queries/.

[35] Socrata. Soql function and keyword listing. https://dev.socrata.com/docs/functions/.

[36] Google. Maps unity sdk overview. https://developers.google.com/maps/documentation/gaming/overview_musk.

[37] Mapbox. Maps for unity. https://www.mapbox.com/unity/.

[38] WRLD. Getting started with the wrld unity sdk. https://www.wrld3d.com/unity/latest/docs/examples/.

[39] Dilmer Valecillos. Various ar foundation examples created with arkit3. https://github.com/dilmerv/UnityARFoundationEssentials.

[40] Unity. Touchphase. https://docs.unity3d.com/ScriptReference/TouchPhase.html.

[41] Carlos Wilkes. Lean touch document. http://carloswilkes.com/Documentation/LeanTouch.

[42] Camden Council. Details page for planning application - 2017/6978/p. https://planningrecords.camden.gov.uk/Northgate/Redirection/redirect.aspx?linkid=EXDC&PARAM0=459318.

[43] F. Aurenhammer and R. Klein. Voronoi diagrams. In *Handbook of Computational Geometry*, pages 201–290. North-Holland, 2000.

[44] Blender. Voronoi texture node. https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/voronoi.html.

[45] Blender. Musgrave texture node. https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/musgrave.html.

[46] Blender. Glossy bsdf. https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/glossy.html.

[47] Blender. Diffuse bsdf. https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/diffuse.html.

[48] Unity. Reflection probe. https://docs.unity3d.com/Manual/class-ReflectionProbe.html.

[49] Blender. Render baking. https://docs.blender.org/manual/en/latest/render/cycles/baking.html.

[50] campeezjesse. 'mapboxmobileevents/mapboxmobileevents.h' file not found. https://github.com/mapbox/mapbox-unity-sdk/issues/644.

[51] truekit. #import ¡mapboxmobileevents/mapboxmobileevents.h¿ 'mapboxmobileevents/mapboxmobileevents.h' file not found. https://github.com/mapbox/mapbox-unity-sdk/issues/566.

[52] agent reed. ios build failure - 'mapboxmobileevents/mapboxmobileevents.h' file not found. https://github.com/mapbox/mapbox-unity-sdk/issues/301.

[53] Mapbox. Location based games. https://docs.mapbox.com/unity/maps/overview/location-based-games/.

[54] FenderThinks. libstreamalpha dependency does not support compilation with bitcode enabled on ios. https://github.com/wrld3d/unity-api/issues/2.

[55] Unity. Locationservice.start. https://docs.unity3d.com/ScriptReference/LocationService.Start.html.

[56] Apple. Nslocationwheninuseusagedescription. https://developer.apple.com/documentation/bundleresources/information_property_list/nslocationwheninuseusagedescription.

[57] Unity. Coroutines. https://docs.unity3d.com/Manual/Coroutines.html.

[58] Unity. Jsonutility.fromjson. https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html.

[59] ffleurey. How to load an array with jsonutility? https://forum.unity.com/threads/how-to-load-an-array-with-jsonutility.375735/#post-2585129.

[60] Unity. Recorded video training: Shop ui with runtime scroll lists. https://learn.unity.com/tutorial/live-training-shop-ui-with-runtime-scroll-lists#5c7f8528edbc2a002053b4cb.

[61] Unity. Object pooling. https://learn.unity.com/tutorial/object-pooling.

Appendix A

System Manual

Specification

- Name of the application: Camden View
- Platform: iOS
- Minimum iOS version: iOS 11
- Testing device: iPhone 8
- Development tool:
  - Unity 2019.3.0a4
  - Xcode 10.3
  - Visual Studio Code 1.36
  - Blender 2.78
- Unity packages:
  - AR Foundation 2.1.0
  - ARKit XR Plugin 2.1.0
- SDK used: Lean Touch
- Development environment: macOS 10.14.6

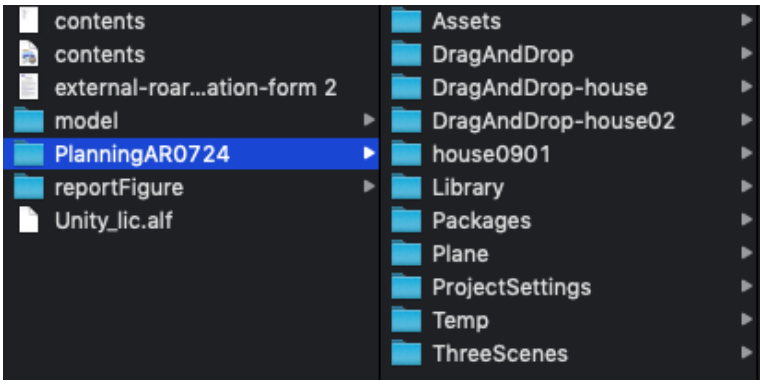


Figure A.1: File structure.

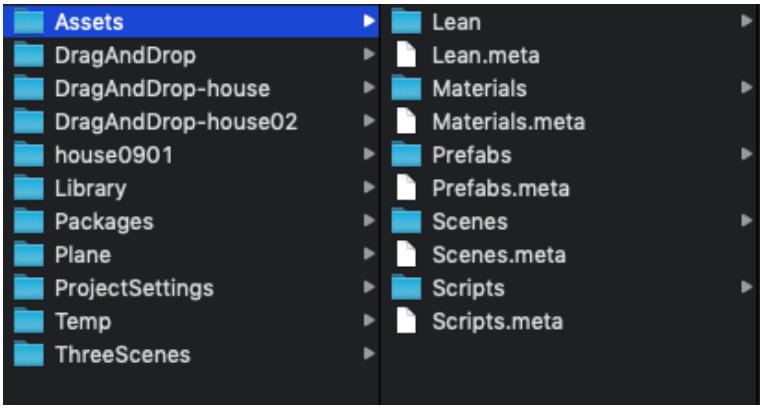


Figure A.2: Assets folder.

Project File

As illustrated in Figure A.1, *Assets*, *Library*, *Packages*, and *Temp* folders are Unity default folders. The other folders are project that were built. *ThreeScenes* is the final version of this project. This is the folder that Xcode reads and runs.

Figure A.2 shows the content of the assets folder. *Lean* was generated by Lean Touch to store its data. All of the scenes are stored in the *Scenes* folder, while *Scripts* holds all the scripts created by the developer.

Appendix B

User Manual

When the application is opened, users will see the list page as illustrated in Figure B.1. On the bottom of the screen, there is a *List* and a *Map* button. Click on the map button to go to map view, which will be implemented in the future (Figure B.2). The users can scroll down the list and select the application they are interested in. After selecting one application, they will be taken to the detail page where the development address, description, registered date, etc. are displayed as illustrated in Figure B.3. If they are close enough to the development site (within 20 meters), they can activate the AR view by clicking the AR mode button.

In the AR mode, the application would detect horizontal surfaces, and overlay them with mesh. The effect was shown in Figure B.4. To drop the model, tap on the detected plane with one finger. After dropping the model the users can drag the model around with one finger or rotate it with two fingers. Click the back button on the left-top corner to go back to the list page. The screenshots were provided in Figure B.5.

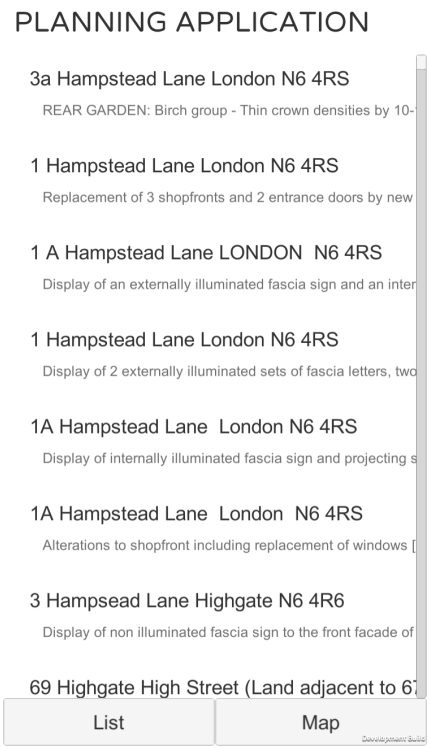
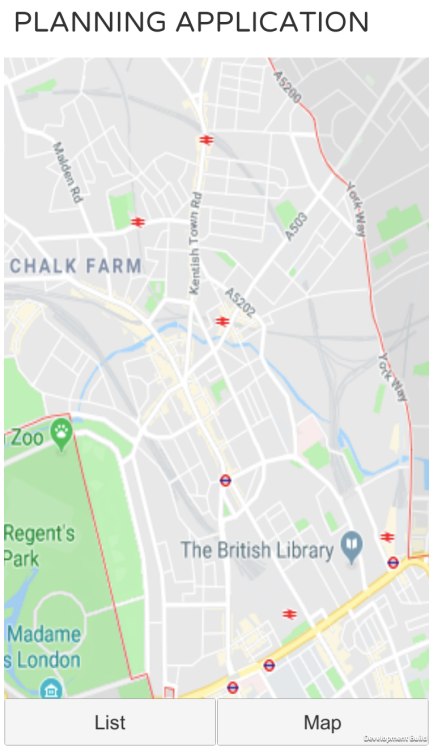


Figure B.1: List page.



(a) First map view.

(b) Second map view.

Figure B.2: Map view.

< Back

2017/1313/T

DEVELOPMENT ADDRESS

3a Hampstead Lane London N6 4RS

DEVELOPMENT DESCRIPTION

REAR GARDEN: Birch group - Thin crown densities by 10-15% 1 x Acer - Thin crown density by 15%. 1 x Cercis - Crown thin by 10-15% FRONT GARDEN: 1 x Tulip Tree - Thin crown density by 20%, reduce back from neighbouring property to allow for 0.50m clearance. Remove the lowest branch protruding over pavement 1 x Magnolia - Reduce back from building to allow 0.50m clearance. Thin crown density by

REGISTERED DATE

2017-03-08T00:00:00.000

DECISION DATE

2017-04-11T00:00:00.000

APPLICATION TYPE

Notification of Intended Works to Tree(s) in a Conservation Area

SYSTEM STATUS

Final Decision

FULL APPLICATION

https://planningrecords.camden.gov.uk/Northgate/Redirection/redirect.aspx?linkid=EXDC&PARAM0=449140

AR view

Figure B.3: Detail page.

(a) Plane detection position 1.

(b) Plane detection position 2.

Figure B.4: Plane detection in action.

(a) Drop the model.

(b) Rotate and view the building from different angles.

(c) Rotate and view the building from different angles.

Figure B.5: Placing the model.

(d) Rotate and view the building from different angles.

(e) Rotate and view the building from different angles.

(f) Rotate and view the building from different angles.

Figure B.5: Placing the model. (cont.)

Appendix C

Supporting Documentation and diagrams

C.1 Detailed Use Cases

The detailed use cases are presented in Table C.1 - Table C.8.

C.2 Test Cases

The detailed test cases are listed in Table C.9 - Table C.15.

Use Cases	Display planning application list.
ID	UC1
Brief Description	User browses the list of planning applications.
Primary Actors	App user
Preconditions	The app is not launched.
Main Flow	1. User opens the app. 2. The app fetches data from Open Data Camden database. 2. User scrolls down the list of planning applications.
Postconditions	The app is booted, and the list is loaded.

Table C.1: Use Cases: UC1

Use Cases	Display planning applications on map.
ID	UC2
Brief Description	User browses the planning applications on map.
Primary Actors	App user
Preconditions	The app is opened and on the list view.
Main Flow	1. User selects the map view. 2. User sees his or her location on the map as well as the location of the planning applications. 3. User pans and zooms the map.
Postconditions	The map is loaded.

Table C.2: Use Cases: UC2

Use Cases	Show details of a planning application.
ID	UC3
Brief Description	User reads the detail of a planning application.
Primary Actors	App user
Preconditions	User selects a planning application from the list or the map.
Main Flow	1. User selects a planning application either on the list or on the map. 2. User reads the detail of the planning application on the detail page.
Postconditions	The detail of the selected planning application is displayed.

Table C.3: Use Cases: UC3

Use Cases	Display 3D model in AR mode.
ID	UC4
Brief Description	User views the 3D model in AR mode.
Primary Actors	App user
Preconditions	User is on the detail page and is within 20-meter range of the development site.
Main Flow	1. User clicks on the ”AR view” button. 2. User moves the devices so that planes can be detected. 3. User tap on the screen to place the model on a detected plane. 4. User drags the model on the planes. 5. User rotates the model.
Postconditions	The model can be viewed from different angles.

Table C.4: Use Cases: UC4

Use Cases	Search planning applications.
ID	UC5
Brief Description	User searches planning applications.
Primary Actors	App user
Preconditions	The list or map view is loaded.
Main Flow	1. User types in keywords in the search bar and clicks ”search”. 2. Planning applications that contains the keywords are shown.
Postconditions	Planning applications that contains the keywords are displayed on the page.

Table C.5: Use Cases: UC5

Use Cases	Filter planning applications.
ID	UC6
Brief Description	User filters planning applications.
Primary Actors	App user
Preconditions	The list or map view is loaded.
Main Flow	1. User sets the filter conditions. 2. Planning applications that meets the conditions are displayed.
Postconditions	The app filters the planning applications.

Table C.6: Use Cases: UC6

Use Cases	Sort planning applications.
ID	UC7
Brief Description	User sorts planning applications.
Primary Actors	App user
Preconditions	The list view is loaded.
Main Flow	1. User selects the attribute to sort the planning applications. 2. Planning applications are sorted.
Postconditions	The sorted planning application list is shown.

Table C.7: Use Cases: UC7

Use Cases	Display complete planning application documents.
ID	UC8
Brief Description	User reads the documents submitted by the applicant.
Primary Actors	App user
Preconditions	User is on the detail page.
Main Flow	1. User clicks on the ”Full Application” URL. 2. The web page of the full application is opened. 3. User clicks on the documents.
Postconditions	The documents are displayed.

Table C.8: Use Cases: UC8

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC1	Populate planning applications.	1. Open the app. 2. List of planning applications are loaded.	A list of planning applications should be displayed.	As expected.	Pass

Table C.9: Test Cases: TC1

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC2	Scroll the list.	1. The list of planning applications are loaded. 2. Scroll down the list.	The scroll view should only be able to move vertically.	As expected.	Pass

Table C.10: Test Cases: TC2

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC3	Read details of a planning application.	1. Scroll down the list. 2. Select one planning application. 3. Enter detail page.	The detail page should display the address, description, registered date, decision date, application type, system status, etc.	As expected.	Pass

Table C.11: Test Cases: TC3

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC4	Detect planes.	1. Open the app. 2. Select one planning application. 3. Click on AR mode button. 4. Move the device around and see if horizontal planes are detected.	Horizontal planes should be detected and bounded by lines and mesh on the screen.	As expected.	Pass

Table C.12: Test Cases: TC4

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC5	Drop the model in AR mode.	1. Click on AR mode button. 2. Drop the model on detected plane by tapping the screen with one finger.	The model should be placed on the position specified by the user.	As expected.	Pass

Table C.13: Test Cases: TC5

Test Case ID	Test Scenario	Test Steps	Expected Results	Actual Results	Pass / Fail
TC6	Move the model in AR mode.	1. Drop the model on detected plane by tapping the screen with one finger. 2. Drag the model with one finger.	Users should be able to drag the model on detected plane.	As expected.	Pass

Table C.14: Test Cases: TC6

Test Case ID	**Test Scenario**	**Test Steps**	**Expected Results**	**Actual Results**	**Pass / Fail**
TC7	Rotate the model in AR mode.	1. Drop the model on detected plane by tapping the screen with one finger. 2. Rotate the model with two fingers.	Users should be able to rotate the model on detected plane.	As expected.	Pass

Table C.15: Test Cases: TC7