AUGMENTED REALITY IN RETAIL

Final Year Project

COMP4801 Final Report
Supervisor – Professor HF Ting

Anchit Som 3035244265 | Rachaika Verma 3035244320
Abstract

Hong Kong- one of the world’s fastest economies today, provides endless opportunities for retailers all around the globe. However, this vertical city has plummeting rates that sometimes create an insurmountable barrier for several retailers and hawkers to establish a legitimate shopping environment, thus driving them to other cities with lesser costs of rental and infrastructure. With the applications of Augmented Reality being on the rise, especially in the retail world, the vision to perceive products before buying them can be actualised. On one hand, while online shopping provides convenience, minimal setup costs and limitless inventory for retailers, physical stores provide an interactive and intuitive environment for customers to view products. This report explores the opportunity present in the two markets and proposes an all new hybrid shopping experience enabled by Augmented Reality technology and Photogrammetry. The vision would be manifested as an iOS based application that scans objects in 3D and allows convenient viewing of the products in AR. The project aims to create a working e-commerce platform within the time frame of ten months that serves both customers and retailers thereby allowing retailers to upload various images of the product onto the platform which would be spawned as 3D models for the customers to help enhance the shopping experience.
Acknowledgement
I would like to thank our project supervisor Dr. HF Ting for his constant support and guidance through the course of this project. His valuable feedback and advise has guided us in the right direction. I would also like to thank the Faculty of Computer Science Engineering for providing us with the resources (iPad and the Mac Laboratory) needed for this project.

Abbreviations
• AR: Augmented Reality
• SDKs: Software Development Kits
• UI: User Interface
• VIO: Visual Inertial Odometry
• API: Application Programming Interface
# Table of Contents

List of Figures .................................................................................................................. 5
List of Tables ..................................................................................................................... 5
1 Introduction..................................................................................................................... 6
  1.1 Outline....................................................................................................................... 6
  1.2 What is Augmented Reality?..................................................................................... 6
  1.3 Background................................................................................................................ 6
  1.4 Scope........................................................................................................................ 7
  1.5 Deliverables.............................................................................................................. 8
2 Related Works................................................................................................................ 8
  2.1 Ikea.......................................................................................................................... 9
  2.2 Scann3D.................................................................................................................. 10
3 Features and Objectives............................................................................................... 10
  3.1 Model Creation......................................................................................................... 10
  3.2 Product Demo......................................................................................................... 10
  3.3 User Interface......................................................................................................... 10
4 Methodology.................................................................................................................. 10
  4.1 Frameworks used .................................................................................................... 11
    4.1.1 Apple’s ARKit 2............................................................................................... 11
    4.1.2 XCode 10.2 IDE............................................................................................. 11
    4.1.3 ReCAP API..................................................................................................... 12
  4.2 Application Design.................................................................................................. 13
  4.3 System Design......................................................................................................... 14
    4.3.1 Customer Journey............................................................................................ 14
    4.3.2 Retailer Journey.............................................................................................. 14
  4.4 Technical Implementation....................................................................................... 15
    4.4.1 Object Spawning.............................................................................................. 15
    4.4.1.1 WorldTracking Configuration.................................................................... 15
    4.4.1.2 SceneKit 3D.............................................................................................. 16
    4.4.2 Photo to 3D Model.......................................................................................... 19
    4.4.3 UI and Backend.............................................................................................. 24
      4.4.3.1 User Interface elements............................................................................ 24
      4.4.3.2 Database Logic......................................................................................... 26
5 Schedule......................................................................................................................... 31
6 Results and Discussion................................................................................................. 31
  6.1 From Unity to XCode.............................................................................................. 31
  5.2 Photo Processing Problems.................................................................................... 31
  5.3 User testing AR Spawning..................................................................................... 32
7 Conclusion..................................................................................................................... 33
8 References....................................................................................................................... 34
List of Figures

Figure 1: Difference between VR and AR..................................................7
Figure 2: Impact of AR on Retail...............................................................9
Figure 3: Ikea Place App.............................................................. 9
Figure 4: ReCAP API Flow......................................................... 13
Figure 5: Add new Product............................................................ 15
Figure 6: Before plane and After Plane.......................................... 17
Figure 7: Tab bar Controller.......................................................... 25
Figure 8: Navigation Controller.................................................. 25
Figure 9: Collection View Controller........................................ 25
Figure 10: Table View and Page View Controller......................... 26
Figure 11: Data Class Diagram..................................................... 27
Figure 12: Poor Quality Rendering................................................ 32
Figure 13: Good Quality Rendering................................................ 32

List of Tables

Table 1 HTTP API Calls.......................................................14
Table 2 Project Schedule.....................................................31
1. **Introduction**

Augmented Reality today is a rather new technology with several implications in the modern world. However, as it is still very fresh, not much is known about its potential and implications. This section aims to implore several aspects of the technology with respect to the project at hand.

1.1 **Outline**

The report discusses the current trends in Augmented Reality and points out our motivation to engage in this project. This is followed by a list of objectives we aim to achieve and a detailed methodology for executing the project. The report then explores the work we have done till now and the results obtained in the process. It will also emphasize on the results of the several rounds of iterations and testing and will also highlight the project’s limitations. The next steps to the project are discussed along with a project schedule in the final section.

1.2 **What is Augmented Reality?**

Back in 2016 when Pokemon Go was one of the most downloaded game in the Appstore, Augmented Reality garnered the attention of various technology enthusiasts. Augmented Reality is best described as a technology “which superimposes a computer-generated image on the user’s view of the real world”. Augmented Reality is often confused with Virtual Reality as the latter has been around for much longer whereas AR is relatively new in the technology world. The primary factor that sets the two apart (refer to figure 1) is that unlike Virtual Reality, Augmented Reality is not a fully immersive computer stimulated technology instead it simply provides a composite view over the user’s real view. As seen in the image below, when using VR, the user’s environment is transformed into a new stimulated world by shutting out the real physical world while in AR the digital content is just superimposed onto the user’s view. This makes augmented reality more lifelike and intuitive.
1.3 **Background**
The current retail landscape can be broadly divided into two categories- offline shops and online stores. Online stores have become a norm in the past few years due to minimal setup costs and no inventory costs. However, the product viewing experience is not intuitive enough for buyers which often leads them to prefer Offline stores to make purchase decisions. Unfortunately for retailers, Offline stores still have infrastructure costs associated with them which makes them expensive to set up. Our project aims to bring the advantages of physical product viewing to online shopping using Augmented Reality and Photogrammetry. This hybrid shopping experience would enable users to interact with the products in virtually created environments that they can view from their phones.

1.4 **Scope**
The development of the retail application would be done on Apple’s ARKit2 platform thus the scope would be limited to iOS devices. The reason for choosing ARKit as our development platform has been discussed in the section 4.1.1. The application would be ready to deploy on the following ARKit Compatible devices.
- iPhone 6s and 6s Plus.
- iPhone 7 and 7 Plus.
- iPhone SE.
- iPad Pro (9.7, 10.5 or 12.9) – both first-gen and 2nd-gen.
- iPad (2017)
- iPhone 8 and 8 Plus.
- iPhone X, Xs and Xr

1.5 **Deliverables**

Our end deliverable would be an iOS based application that would be ready to deploy on the devices mentioned above. We would be using the ReCap API for photo to 3D model creation.

2. **Related Works**

AR has been used in mobile shopping applications by various firms like IKEA and Shopify to add competitive edge to their consumer-experience (Sheehan, 2018).

Photogrammetry has been used in architecture, industrial design and game development industries. This technique of modelling uses photos to make measurements between objects and create a subsequent geometric representation of the same. Recently companies like Shopify have shown interest in this space by launching services for retailers to convert their products to 3D scans.

The following section discusses a few of these use-cases and also furthers our case for making an using the 2 technologies in our application.
2.1 Ikea

Ikea Place is an application that helps the user visualize the furniture in their home space before making a purchase decision. (Chang, 2018).” This is particularly important in the home decor retail space as customers often spend countless hours trying to decide whether something will look good in their homes or not. The “try before you buy” concept by Ikea has been very well received by its customers and serves as a motivation for us to build our application for at-home product viewing.
2.2 **Scann3D**

This application requires the user to take approximately 20 to 30 photos of the object that they wish to 3D scan. This application available on iOS makes the device a standalone tool to turn images into 3D models for storing, sharing and editing in Sketchfab.

3. **Features and Objectives**

This section outlines the different features of the application that we aim on building in the project

3.1 **Model creation**

The retailer will be guided through a process to take photos of the object they want to sell. After the photosession is complete, our application would convert these photos to 3D models using PGM. These 3D objects can be used by the retailer to display the items he has for sale along with the pictures he/she took.

3.2 **Product Demos**

These items for sale inside the storefront would form the basis of our AR experience. The user would be able to interact with the objects in AR in a manner similar to physical products before making a purchase. Our project focuses on four types of products that can be viewed at homes - Jewellery, Furniture, Handicrafts and Artwork.

3.3 **User Interface**

To guide the user through different modes of the application, an easy to use and intuitive UI will be designed. While the overall layout of the application is based on existing e-commerce platforms. Several unique design features are introduced to emphasise the hybrid experience of the product.

4. **Methodology**

This section discusses the technologies used by the team to implement the project. It lays out the application design as well as the system design in detail.
4.1 Frameworks used

4.1.1 ARKit SDK

4.1.1.1 Choosing the right SDK

The two major Mobile OS platforms Android and iOS both support Augmented Reality through their SDKs - ARKit and ARCore. Currently there are no hybrid solutions available in the market that allow us to create AR based applications for both platforms. Thus for our final year project, we had to choose between the two SDKs for development. While they provide similar features in motion tracking, light estimation and environmental understanding. We chose ARKit due to the following advantages:

- ARKit distinguishes between horizontal and vertical surfaces making tabletop detection easy for our application.
- ARKit has extensive developer support and has been released to all iPhone devices in the new update.
- ARkit2 allows for “Social AR” experiences over Wifi, enabling collaborative shopping scenarios.

4.1.1.2 How ARKit 2 works

ARKit is a Visual Inertial Odometry (VIO) system that tracks the user’s position by matching a point in the real world to a pixel on the camera sensor at each frame refresh. The Inertial Measurement Unit (IMU) also measures the user’s position using two inertial sensors - Gyroscope and accelerometer. The time measurements and accelerations are integrated backwards to calculate velocity and distance travelled between two frames. While error in such a system is upwards of 30% in most systems, ARKit’s error correction reduces the error to single digit. Every tiny muscle movement can be detected by the inertial system providing unparalleled accuracy. The output of these systems is combined using a Kaiman Filter to provide the best estimate of the actual position which is published to the ARKit SDK.

4.1.2 XCode 10 Integrated Development Environment

Since we chose ARKit as our development SDK, we decided to use XCode as our Development IDE as it is the standard development tool used for creating iOS
applications. Our application is built in Swift version 4.2.1 with the deployment target being iOS 12.2 devices. We used a popular dependency manager called cocoapods to handle dependencies for the project that would make our development effort quicker and more streamlined.

4.1.2.1 **SwiftyJSON** – Swift’s standard JSON library has explicit typing which requires JSON objects to be serialized and chained before usage, like this

```swift
if let JSONObject = try JSONSerialization.JSONObject(with: data, options: .allowFragments) as? [[String: Any]],
let username = (JSONObject[0]["user"] as? [String: Any])?["name"] as? String
```

Our application relies heavily on JSON objects that we receive from the RECAP API HTTP Server and the standard approach makes the code messy. SwiftyJSON abstracts a large part of this code and makes it possible to declare JSON objects implicitly and allows access in a manner similar to an ArrayObject, like this –

```swift
let json = JSON(data: dataFromNetworking)
if let userName = json[0]["user"]["name"].string {
    // Now you got your value
}
```

4.1.2.2 **Alamofire** – Swift traditionally uses NSURLSession and a Foundational URL Loading system to manage request-response cycles to HTTP Servers. Alamofire provides an easy to use interface that performs HTTP Networking asynchronously without having to write large chunks of the protocol manually.

4.1.2.3 **BSImagePicker** – Swift’s UIImagePickerController Class only allows for selection of one image from the photo library. Our application requires multiple selections to make 3D model creation possible thus use the BSImagePicker framework that provides functions for creating an Image Picker and converts the selected assets to JPEG objects ready for upload.

4.1.3 **ReCAP API by Autodesk**

ReCap Photo is an Autodesk 360 service that creates 3D Scans from photos such that users can view and share the 3D data. Users upload images on Autodesk 360 and the Cloud-based service creates a 3D mesh model reading for importing. The API works on REST Calls and
allows developers to integrate this service for into their applications and products. The figure below gives a brief overview about how the service works.

Figure : ReCAP API Flow

4.2 Application Design

Our application, BazAR, is designed as a P2P marketplace designed to support two use cases- Add a new Product, View Products for Sale

After loading, the various shopfronts and featured products are displayed on the home screen along with an add product button. On clicking the button, the user is directed to an input screen where he/she adds details of the product and uploads images that are converted to a 3D Model. After model creation is complete, the product item is displayed in the listing for other customers to view in the application.

The Application has been designed in accordance to existing e-commerce applications like amazon, Taobao such that user’s feel the familiarity of a marketplace. On the home screen, the customer is provided a list of shopfronts. After the user selects a shopfront, he sees a product list and can click on a product to view details. When the user views a specific item, a “view in AR” button appears and on clicking, the camera opens up for the user to view the 3D model in their camera view.
4.3 System Design

4.3.1 Customer Journey

Step 1: The user starts the application
Step 2: User browses through the list of shops which appear from the data entries
Step 3: User selects and enters the particular shopfront, and views the product list
Step 4: User selects a product from the list
Step 5: The respective 3D model is called from the database and the camera is started again
Step 6: Repeat steps 1,2,3,4,5 till user exits.

4.3.2 Retailer Journey

When the retailer starts the application, the home screen has a button called add new product. This pulls up a form (as seen in the figure below) for the retailer to enter the product details and select images to upload to the server for them to be converted into a 3D model of the retailer’s product. When the retailer clicks on the create 3D model button, the following calls are made to the ReCap API for processing the mesh:

<table>
<thead>
<tr>
<th><strong>HTTP API Calls</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST photoscene</td>
<td>This creates a Photoscene that will perform calibration and meshing.</td>
</tr>
<tr>
<td>POST file</td>
<td>The reference for each image file is provided in this call for uploading to ReCAP360.</td>
</tr>
<tr>
<td>POST photoscene/:photosceneid</td>
<td>Initiates processing of the Photoscene created above.</td>
</tr>
<tr>
<td>GET photoscene/:photosceneid/progress</td>
<td>This returns the current processing progress percentage and status of the Photoscene.</td>
</tr>
<tr>
<td>GET photoscene/:photosceneid</td>
<td>Returns a time-limited HTTPS link to the .obj output file that can be downloaded for display as 3D object.</td>
</tr>
</tbody>
</table>

Table 1 HTTP API Calls
4.4 **Technical Implementation**

4.4.1 **Object Spawning**

After clicking on the “View in AR” button on the product listing, the selected product is spawned as a 3D object on the AR Scene.

4.4.1.1 **WorldTracking Configuration**

WorldTrackingConfiguration is used by ARKit to track the user’s real-world space and map it to the device. Provides the basis for this as it helps to map and track the user’s real-world space. It creates a WorldTracking Scene on which the image is spawned and the object increases in size when you movie towards it.

4.4.1.2 **SceneKit**

The SceneKit Library is used as a rendering engine for import, manipulation and rendering of 3D assets. We create a SceneKit for object spawning as it requires only descriptions of the scene’s contents and actions that the user might perform.
4.4.1.2.1 ARSCNView

An ARSCNView is created by SceneKit in the main storyboard for object Spawning. This scene consists of structural elements called SCN Nodes to which various elements like geometry, cameras, lights etc. can be attached.

4.4.1.2.2 SCN Scene Renderer

This component manages the Scene Kit’s rendering and animation of the scene’s contents. The code snippet below automatically adds light to the scene which is ambient and adjusting to the camera’s environment.

```swift
super.viewDidLoad()
self scnView.debugOptions = [ARSCNDebugOptions.showFeaturePoints]
scnView.delegate = self
scnView.autoEnablesDefaultLighting = true
```

4.4.1.2.3 Touches Component

Touches are instances of the UI Touch class which are used to trigger the object spawning. Scene Kit calls the touchesBegan function whenever a new user touch is detected on the screen. The first touch location is detected and saved locally in the form of a 2D space with only x and y components.

```swift
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    if let touch = touches.first {
        let touchLocation = touch.location(in: scnView)
        let results = scnView.hitTest(touchLocation, types: .existingPlaneUsingExtent) //converting 2d to 3d touch has x and y component, so its a 2d location, hit a point in 3d space
    }
}
```

In the snippet below, the coordinates of the touch component were- x: -0.026806; z: -0.087583. Since the touch is in a 2D plane, it is missing a Y component. However, after the hitTest function is called, the 2D coordinates are mapped to a 3D planes as shown in the highlighted blue section.
4.4.1.2.4 Plane Detection

Plane detection is added to help the user visualize the object on a tabletop for instead of spawning in air. The two images show the difference in the spawning of a dice before and after the addition of plane detection.

Figure 6: Before Plane and After Plane
The plane detection component is set as horizontal this adds ARPlane Anchor objects to the WorldTracking Configuration. It also notifies the ARSCNView if the camera detected a horizontal flat surface or not.

A blue grid is displayed when an ideal horizontal surface is detected by the rear camera. Feature points (the yellow dots in the figure above) are added to fix spawning position in the plane and lock location regardless of where the user moves. The renderer function below checks if the anchors added were used for plane detection or not and then places the plane on the scene view.

```swift
func renderer(_: renderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    if anchor is ARPlaneAnchor {
        let planeAnchor = anchor as! ARPlaneAnchor
        let plane = SCNPlane(width: CGFloat(planeAnchor.extent.x), height: CGFloat(planeAnchor.extent.z))

        let planeNode = SCNNode()
        planeNode.position = SCNVector3(x: planeAnchor.center.x, y: 0, z: planeAnchor.center.z)

        planeNode.transform = SCNMatrix4MakeRotation(-Float.pi/2, 1, 0, 0)

        let gridMaterial = SCNMaterial()
        gridMaterial.diffuse.contents = UIImage(named: "art.scnassets/grid.png")
        plane.materials = [gridMaterial]
        planeNode.geometry = plane
        node.addChildNode(planeNode)
    } else {
    }
}
```

After all these components of SceneKit have been setup, the object is finally spawned onto a plane by adding the 3D model (.obj) file as a scene and appending it to the main scene view.

The function below shows the appending of the “bagNode” as a scene in the scene view using the hit result’s coordinates.

```swift
if let bagNode = scene.rootNode.childNodes(withName: "Dice", recursively: true) {
    bagNode.position = SCNVector3(x: hitResult.worldTransform.columns.3.x, y: hitResult.worldTransform.columns.3.y + bagNode.boundingSphere.radius, z: hitResult.worldTransform.columns.3.z)

    sceneView.scene.rootNode.addChildNode(bagNode)
}```
UI touch coordinates provide the object coordinates where the object has been successfully spawned as per user touch input.

### 4.4.2 Photo to 3D Model Creation

The 3D model creation process is triggered when the user clicks on the create 3D model button in the “Add Product Screen”. This button is tied to a create model function which runs the request-response cycles from authentication to 3D Model creation in the following steps.

1. **Authentication**: After registering our application with the ReCAP service, we receive a Client ID and Client Secret that is used to authenticate the application and generate a unique access token for further calls.

   **Request**

   ```swift
   @IBAction func createModel(_ sender: Any) {
   //Authorization
   
   let headers: HTTPHeaders = [
   "Content-Type": "application/x-www-form-urlencoded"
   ]
   
   let params = [
   "client_id": "bkGhU3g7XHkU4amG5pMAF6rPuLOVayYt",
   "client_secret": "eDudkJFZ3iUB5nvd",
   "grant_type": "client_credentials",
   "scope": "data:read data:write"
   ]
   
   let swiftyJsonvar = JSON(responseData.result.value)
   }
   
   **Response**

   ```json
   {
   "token_type": "Bearer",
   "expires_in": 1799,
   "access_token": "eyJhbGciOiJIUzI1NiIsImtpZCI6ImtpZCI6MjM4NjYzNDAyMjE1MjIwMjA3Iiwiymmetric: eyJhbiI6NiwiZGF0YSI6MTA3MjU4Nzg4McIsImNvbnRlbnRfdXNlcCI6MjA1MjU1NTg1ODQ5NjUzODMwMcIsImF1ZCI6MjA1MjU1NTg1ODQ5NjUzODMwMcIsInZlcnNvbnRfdXNlcCI6MjA1MjU1NTg1ODQ5NjUzODMwMcIsImh0bW9yZXIiOiJ1c2VyLWJhY2tncm91c0xhcmQgJyIsImFwaV91c2VyX2lkIjoxfQ.uzNexXCeu4efGPKGGHdKxoJDXHAzLb2S2nSjrq_yS"
   }
   ```
2. **Create PhotoScene**: The subsequent call is nested in the response of the previous call and the access token is added to the header of the new call. The new call requests creation of a unique photosceneid in which our photos will be uploaded and model will be stored.

**Request**

```swift
let token = swiftyJsonVar["access_token"].rawString()!
let access_token = "Bearer " + token
let headers2: HTTPHeaders = [
    "Authorization": access_token,
    "Content-Type": "application/x-www-form-urlencoded"
]
let params2 = [
    "scenename": "testscene",
    "format": "obj",
    "scenetype": "object"
]
 Alamofire.request("https://developer.api.autodesk.com/photo-to-3d/v1/photoscene", method: .post,
parameters: params2, headers: headers2).responseJSON { (responseData) -> Void in
    if ((responseData.result.value) != nil) {
        let swiftyJsonVar = JSON(responseData.result.value)
    }
}
```

**Response**

```swift
{
    "msg": "No error",
    "Photoscene": {
        "photosceneid": "hcYJcrnHUsNSPII9glhVe8lRF6lFXs4NHzGqJ3zdWMU"
    }
}
```

3. **Upload Files**: The unique photoscene id is passed as a parameter to the new call and an image gallery is opened up using BSImagePicker. The selected photos are added in a for loop as a part of the POST call and uploaded to the API. A progress bar is shown which tracks the status of the upload.

**Request**
Response

```json
{
    "photosceneid": "AtAuFsedTdqWdhF9VzHepp5oM9fTIuizI4xdMbZ",
    "Files": {
        // create an instance
        let vc = BSSamplePickerViewController()
        // display picture gallery
        self.bs_presentImagePickerController(vc, animated: true,
            select: { (asset: PHAsset) -> Void in
                // User deselected an assets.
                },
            cancel: { (assets: [PHAsset]) -> Void in
                // User cancelled. And this where the assets currently selected.
                },
            finish: { (assets: [PHAsset]) -> Void in
                // User finished with these assets
                Alamofire.upload(multipartFormData: { (multipartFormData) in
                    for (key, value) in params3 {
                        multipartFormData.append((value).data(using: String.Encoding.utf8),
                            name: key as String)
                    }
                }
                for i in 0..<assets.count{
                    autoreleasepool {
                        let manager = PHImageManager.default()
                        let option = PHImageRequestOptions()
                        var thumbnail = UIImage()
                        option.isSynchronous = true
                        manager.requestImage(for: assets[i]).targetSize: PHImageManagerMaximumSize, contentMode: .aspectFill, options: option,
                        completionHandler: {(result, info) -> Void in
                            thumbnail = result!
                        }
                    }
                }
                }, usingThreshold: UInt64.init(), to:
                "https://developer.api.autodesk.com/photo-to-3d/v1/file", method: .post,
                headers: headers3) { (result) in
                    switch result{
                        case .success(let upload, _, _):
                            upload.responseJSON { response in
                                print(response.result.value!)}
                            Alamofire.request("https://developer.api.autodesk.com/photo-to-3d/v1/photoscene/"+photosceneid, method: .post,
                                headers: headers2).responseJSON { (responseData) -> Void in
                                if (responseData.result.isSuccess){
                                    // The photo scene was successfully uploaded.
                                }
                                }}
    }
```
4. **Retrieve Progress**: After Upload is completed, a GET Call is made repeatedly inside a for loop to check the progress of the model creation. This progress is shown on a progress bar and the for loop exits when the progress shown reaches 100%.

**Request**

```swift
var checkprog: Double = 0.0
let group = DispatchGroup()

while (checkprog < 100) {
    if (response.data.result.isSuccess) {
        let newjsonvar = JSON(response.result.value)
        print(newjsonvar["Photoscene"]["progress"].doubleValue.description)
        group.leave()

        checkprog = newjsonvar["Photoscene"]["progress"].doubleValue
        self.progressViewModel.setProgress(Float(checkprog), animated: true)
        self.progressLabelModel.text = "\(Int(checkprog)) %"
    }
}
```

**Response**
5. **Download Model**: After the progress reaches 100, the program exits the for loop and sends a request to download the image and the resulting URL is stored in the database.

**Request**

```javascript
Alamofire.request("https://developer.api.autodesk.com/photo-to-3d/v1/photoscene/"+photosceneid, method: .get, headers: headers2).responseJSON { (responseData) -> Void in
    if (responseData.result.isSuccess)(
        // Process the response
    )
}
```

**Response**

```json
{
    "Photoscene": {
        "photosceneid": "AtAuFsedTdqWdhF9VzHepp5oM9PITiuizI4xdMbz",
        "progressmsg": "DONE",
        "progress": "100",
        "scenelink": "https://s3.amazonaws.com/com.autodesk.storage...",
        "filesize": "41640100",
        "resultmsg": {}
    }
}
```
4.4.3 **User Interface and Backend of the E-commerce application**

4.4.3.1 **User Interface Elements**

We followed the Model View Controller architectural pattern to separate the application’s backend logic from the front end.

Each Graphical User Interface is called a View Controller and it resides on the main storyboard. The storyboard shows the connections between different ViewControllers and the sequence of execution. In our application, we use segues to connect the viewcontrollers and handle the transition between them.

The storyboard in our application contains the following view controllers-

1. **Navigation Controller**: This controller acts as a parent interface that embeds the content of child view controllers. It defines an ordered array called the navigation stack which shows navigation hierarchical content. It manages the navigation bar at the top and the toolbar at the bottom of the interface.

2. **Page View Controller**: This controller manages the transition between pages of content wherein each page is managed by a child view controller.

3. **Tab Bar Controller**: This view controller shows a tab bar and on clicking a specific tab, it displays the root view of the corresponding view controller and replaces any previous views.

4. **Collection View Controller**: This controller manages a collection of data items and presents the contents using an instance of the UI Collection View Cell class.

5. **Table View Controller**: This view controller helps to manage the product list in the browse product view which appears in a table view. It consists of a scroll view that aids in scrolling in case of multiple products.
**Figure 7:** Tab Bar Controller of the BazaAR Application.

This controller has two tabs for user view: Home and Browse, which is displayed on user selection.

**Figure 8:** Navigation Controller of the BazaAR Application.

Contains a navigation stack for the home screen which has multiple UI views and Collection Views.

**Figure 9:** Collection View Controller

This has multiple cells to display the shopfronts and the products. The image is rendered using UI View as a part of the collection view cell.
4.4.3.2 Database Logic

We will now discuss how the database was populated and parsed for the application.

1. Data Models

For the application’s model layer, we use the Core Data Stack. It provides a set of classes that corroboratively support the data models. We make use of the NS Persistent Container Class which handles the creation of the managed object model, persistent store coordinator and the managed object context thereby simplifying the creation and management of the core data stack.

For our application, we created 4 main data objects/entities:

Figure 10: Table View Controller and Page View Controller
The Product list uses the table view controller to display all the products in the browse section. The Detailed view uses page view to render the product details.
1. **Product**: This data entity stores the product details like id, name, sale price, description etc.

2. **Manufacturer**: This data entity stores the details of the manufacturer like id and name.

3. **Product Image**: This stores the images associated with a product which will be used to create the 3D models.

4. **Product Info**: This data entity stores the detailed information of a product which will be displayed on the product detail page of the application.

The class diagram below gives a clear picture of the data models and their relationship with one another.

![Data class Diagram](image-url)

**Figure 11: Data class Diagram**
2. **Data Parsing**

We parse the data from the JSON into our data model through the load products function in the AppDelegate class. This function performs data extraction and JSON serialization to get the data from the database and display it in the respective container view cell. For our application, the AppDelegate class performs the following functions:

- It holds the application’s startup code.
- It instantiates and calls the core data stack.
- It stores the application’s central data objects which may not have their own view controllers.
- It responds to transitions in the applications and maintains the state of the application.
- Resets the database when necessary by using NS Batch Delete Request.
- Loads products from the database, by performing necessary serializations and then displaying it in the correct container view cell as shown in the code snippet below.
2. **Data Updating**

Since this is a Proof-of-concept app, we have used a JSON file to store information of the products created locally such that we can mimic how the application would behave when tied to a real database.

Each time the user adds a new product and clicks on submit. The submit function is called which creates a JSON object from the text input and URLs of the image and model.
The existing JSON from products.json is retrieved, the new JSON is added and the file is rewritten to update the list of entries.

```swift
func submit(_ sender: UIButton) {
    let newdata = [String: Any] {
        "type": type.text!,
        "salePrice": price.text!,
        "name": "",
        "mainimg": imagesURL,
        "Model": modelURL!,
        "manufacturerName": manuname.text!,
        "description": summary.text!
    }

    var jsonObj = JSON()
    if let path = Bundle.main.path(forResource: "products", ofType: "json") {
        do {
            let existingdata = try Data(contentsOf: URL(fileURLWithPath: path), options: .alwaysMapped)
            let jsonObj = try JSON(data: existingdata)
            jsonObj.append(newdata)
        } catch let error {
            print("parse error: \(error.localizedDescription)"
        } else {
            print("Invalid filename/path."
        }
    }

    let str = jsonObj.description
    let finaldata = str.data(using: String.Encoding)! 

    if let file = FileManager.default.fileHandle(forWritingAtPath:"products.json") {
        file.write(finaldata)
    }
}
5. **Schedule**

<table>
<thead>
<tr>
<th>Timeframe</th>
<th>Task</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>September-October 2018</td>
<td>Feasibility Study</td>
<td>Completed</td>
</tr>
<tr>
<td>November-December 2018</td>
<td>Functionality Testing</td>
<td>Completed</td>
</tr>
<tr>
<td>January-February 2019</td>
<td>System Design and Backend Development</td>
<td>Completed</td>
</tr>
<tr>
<td>March-April 2019</td>
<td>UI and Testing</td>
<td>Completed</td>
</tr>
</tbody>
</table>

Table 2 Project Schedule

6. **Results and Discussion**

6.1 **From Unity to XCode**
We started initial development on Unity however we encountered several issues regarding camera management and app development. Due to Unity being a game engine we were severely restricted by the UI class and had to wrap each gameobject in .cs scripts for development. This made it hard to implement our database logic and create tasks that run asynchronously. Thus we shifted to xCode to develop our application as it supports MVC architecture and allows for traditional mobile development functions.

6.2 **Photoprocessing Problems**
While conducting Photoprocessing, we had several issues with the API ranging from internal errors on the server side to errors in processing of the model. Even when our models got processed, we got poor quality models like the one below.
After debugging and trying various hacks, we discovered that the quality and the number of image uploads were creating a problem. To solve the issue we followed a more informed approach to our phototaking by making sure each picture was in focus and every angle was covered. We started getting better results when we uploaded greater than 70 pictures of the object however the processing time of the model increased to 40 mins.

**6.3 User Testing AR Spawning**

We ran several tests to measure quality of the spawned objects and asked for feedback from different user groups. The spawning performed well in well-lit and minimally textured environments and the accuracy decreased in environments that were either too bright or too dim.
7. Conclusion

Augmented Reality is still in its infancy and has a lot of potential to grow in the future. With companies like Apple and Google developing dedicated software bundles for AR, developers are creating world-class AR applications. After successfully creating our proof of concept application, we hope to build further on the features by reducing time for model creation and increasing model quality. We hope to implement the payment portals and login at a later stage for commercial viability of the application. We believe that the application has the potential of changing the retail industry and how people perceive AR applications.
8. References


