

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree
of Bachelor of Science at the University of Cyprus

**HYBRID CLOUD-ENABLED OBJECT DESIGN WITH
AR VISUALIZATION OF COMPONENT
COMBINATIONS USING UNITY3D**

Saimon Kourra

University of Cyprus



Computer Science Department

MAY 2023

**UNIVERSITY OF CYORUS COMPUTER SCIENCE
DEPARTMENT**

APPROVAL PAGE

Bachelor of Science Thesis

**HYBRID CLOUD-ENABLED OBJECT DESIGN WITH AR VISUALIZATION OF
COMPONENT COMBINATIONS USING UNITY3D**

Presented by Saimon Kourra

Research Supervisor
Andreas Aristidou

University of Cyprus

MAY 2023

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to Dr. Andreas Aristidou for his invaluable guidance, support, and mentorship throughout the entirety of my thesis project. His expertise, patience, and insightful feedback have played a crucial role in shaping the direction and success of this research. I am truly grateful for his unwavering commitment and dedication to my academic and personal development.

I would also like to extend my appreciation to the Cyens Center of Excellence for providing me with technical support and the partial idea that formed the foundation of this thesis. Their contributions and collaboration have been instrumental in enhancing the depth and quality of my work. I am thankful for their willingness to share their knowledge and expertise, which has significantly enriched my research experience.

Furthermore, I would like to acknowledge the industry jobs that have contributed to my growth and learning in the field of API Hybrid-Clouding solutions and rational thinking. The hands-on experience and practical insights gained from these roles have been invaluable in shaping my understanding and problem-solving abilities. I extend my gratitude to the entire team for their guidance, patience, and support.

I am also grateful to all the professors, teachers, and colleagues who have provided their assistance, feedback, and encouragement throughout my academic journey. Their valuable contributions and stimulating discussions have greatly influenced the development of my ideas and the refinement of my work.

Finally, I would like to express my heartfelt gratitude to my family and friends for their steadfast support, compassion, and inspiration. They have been a consistent source of encouragement and motivation, always believing in my abilities and standing by me during challenging times.

ABSTRACT

As technologies continue to evolve, there is an increasing demand for modernization and automation to enhance the quality of life. This thesis focuses on the development of a hybrid cloud-enabled object design system that incorporates augmented reality (AR) visualization of component combinations using Unity3D. The objective of this research is to create a platform that allows users to design and visualize objects by combining different components in real-time. The system leverages the capabilities of hybrid cloud architecture to efficiently store and retrieve component data from cloud storage, enabling seamless collaboration and scalability. Unity3D is utilized as the development framework, providing the necessary tools for AR visualization and interaction. The research explores the challenges and considerations involved in implementing such a system, including data synchronization, network latency, and user experience optimization. The proposed system has the potential to revolutionize object design processes by providing a flexible, collaborative, and visually immersive environment for users to create and explore component combinations.

Table of Contents

List of Abbreviations and Acronyms	6
Chapter 1	7
Introduction.....	7
Motivation.....	7
Problem.....	7
Potential Applications	9
Solution in High Level.....	9
Contribution.....	9
Chapter 2	10
Literature and Related Work.....	10
Introduction	10
Augmented Reality.....	10
Hybrid – Clouding.....	11
Chapter 3	12
Methodology and Implementation	12
PostgreSQL.....	12
Spring Boot Web Service.....	13
Testing with Postman.....	14
Deploy on server using Docker.	15
Modeling	15
Using Web Service from Unity – C#	16
Merging Components	18
AR View	21
Chapter 4	24
Evaluation	24
Overview	24
Used Technologies	25
Discussion.....	28
Chapter 5	29
Conclusion	29
Limitations	29
Future Work.....	30
References.....	33
Appendix A	36
QR Scanner Implementation.....	36
Appendix B	40

List of Abbreviations and Acronyms

1. API: Application Processing Interface.
2. BYO: Build Your Own.
3. AR: Augmented Reality.
4. 3D: 3 Dimensions.
5. HC: Hybrid Clouding.
6. URL: Uniform Resource Locator.
7. HTTP: Hypertext Transfer Protocol.
8. JSON: JavaScript Object Notation.
9. DTO: Data Transfer Object.
10. DB: Database.
11. ID: Identification.
12. TIM: Tracked Image Manager.
13. TIP: Tracked Image Prefab.

Chapter 1

Introduction

Motivation.....	7
Problem.....	7
Potential Applications.....	9
Solution in High Level.....	9

Motivation

Many of us love to decorate our homes with furniture. Many industrial furniture companies like IKEA, have too much Build Your Own furniture. Some examples are chairs with different colors of body and legs and same or different design. With that said, if anyone goes to the store can get the components and then build the desired furniture. But what if the color, size, body design wasn't likeable by the buyer's aesthetics? Should the person turn down the model and return the parts to the store, so he/she can retry with something that he/she thinks would fit? This is too inconvenient for the 21st century. This is why we need to find a way to be able to construct our own models in a virtual world, and with AR being what it is today [1] we can easily display it in the real world. No need to assemble and disassemble the parts. We can simply do it conveniently in the digital space without getting tired or frustrated. However, this idea is, in part, brand new, and in the research library there is no such thing. The need to produce this technology is high for the industry as well, to give component-assemble based enterprises to evolve.

Problem

The process of assembling 3D components in runtime poses a significant challenge due to the massive volume of these components. Assembling 3D components dynamically at runtime

provides numerous advantages, such as increased flexibility, customization, and adaptability. However, when dealing with large volumes of 3D components, several issues arise that need to be addressed in order to achieve efficient and seamless runtime assembly.

One primary challenge is the sheer size and complexity of the 3D components involved. Assembling these components dynamically requires loading and manipulating a vast amount of data in real-time, often leading to performance bottlenecks and delays. The large volume of components can strain computational resources and introduce latency, hindering the smooth assembly process.

Additionally, the diverse nature of 3D components further compounds the problem. These components may come in various formats, sizes, and complexities, requiring extensive computational operations for proper alignment and integration. The variations in geometry, textures, and other attributes introduce additional computational overhead, making the runtime assembly process more demanding and time-consuming.

Furthermore, the scale of the problem amplifies the computational requirements. As the number of 3D components to be assembled increases, the computational complexity grows exponentially. This exponential growth poses significant challenges in terms of memory usage, processing power, and efficient utilization of resources.

Another crucial aspect to consider is the impact on user experience. In applications or systems where runtime assembly of 3D components is a key feature, users expect fast and responsive performance. Delays and lags in the assembly process can disrupt the user's workflow, leading to frustration and a suboptimal experience.

Addressing these challenges is essential for achieving efficient runtime assembly of 3D components, ensuring seamless performance, and enhancing user satisfaction. Solutions that optimize resource utilization, mitigate computational complexity, and streamline the assembly process are crucial to overcoming the limitations imposed by the massive volume of 3D components.

Potential Applications

This idea is not suited only for large enterprises as a flexible solution, it can also be applied to every artistic-like work that uses 3D modeling, such as Graphic Designer or Interior Designer. Referring to the HC solution, it can be used in combination with every small storage capacity device, because managing digital items can be intricate, requiring substantial storage capacity, fast transfer speeds, and containing substantial metadata that is interconnected and holds important semantic and provenance details [2]. This could also provide the possibility of allowing the user to store creations on a personal space in the cloud, with this method it can give ideas to the industry and other users that don't have artistic knowledge.

Solution in High Level

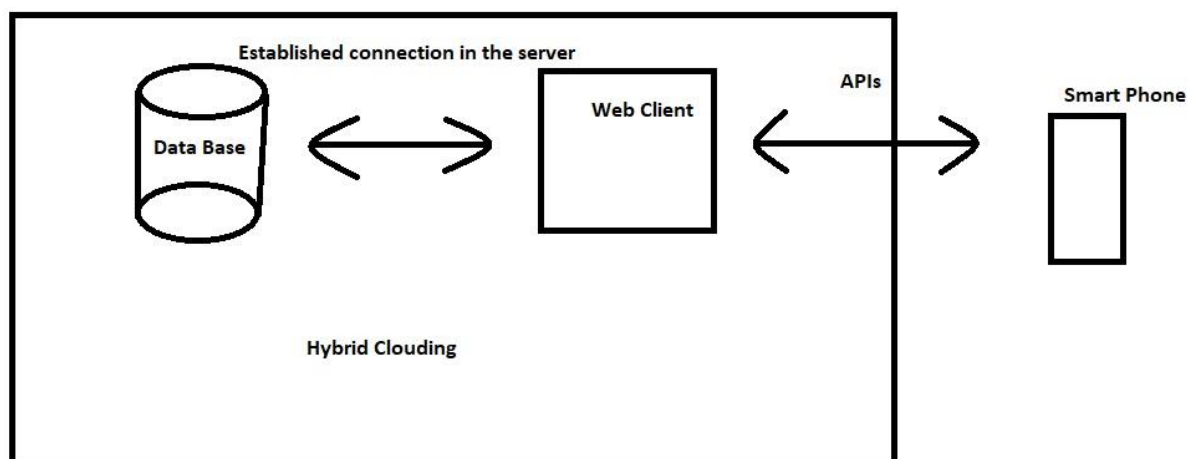


Figure 1: Solution Architecture.

Solution uses a web client service that serves APIs regarding storing and retrieving models to the connected database. The solution states that the application, when a user is building a new model, will fetch the selected component from the database, through the web client/service and import it to the scene, for the user to view and edit accordingly. Finally, to save the created model into the DB through the service after the AR view.

Contribution

This study's goal is not only to make the user's life easier and promote the industries products but to analyze the HC solutions combined with different technologies such as 3D Unity Engine. Additionally, analyzing this approach if is a good use case for reducing the

space on any device, check if we can fully customize a brand-new furniture from given components using this method. For exhibit purposes “Build your Own” mode, is also proposing a QR Scanning System [7] to help the user add the component to the Scene faster, as the user scan the corresponding code of the live-previewed model. Also, various features can be applied such as “View from Collection”, that allows users to view a premade model from another user or admin stored in the database.

Chapter 2

Literature and Related Work

Introduction	10
Augmented Reality.....	10
Hybrid – Clouding.....	11

Introduction

There are plenty of examples of studies that use Augmented Reality or Hybrid Clouding individually. However, the combination of HC that enables model-building added to the AR, is a not popular solution [21] . Until now, popular solutions evolved around premade models that were built into the devices beforehand [4,5,6]. In this section of the paper, we will mention the previous studies for the separate technologies regarding AR and HC.

Augmented Reality

There are different libraries to use with unity for AR [8,9,10]. By using the tools for smartphones like ARKit for iOS [8] or ARCore for Android [8,9]. AR Foundation in general [12], you can easily use the augmented reality features. By doing so there are plenty of ways to calibrate the virtual item to the physical world. Taking advantage of the depth camera of the smartphone, the calculation for the actual size of the 3D Model is easily calculated from

the libraries, by using a pre-defined image for calibration, it calculates the size of the unique picture in the real world and displays the object to the corresponding size [11, 12].

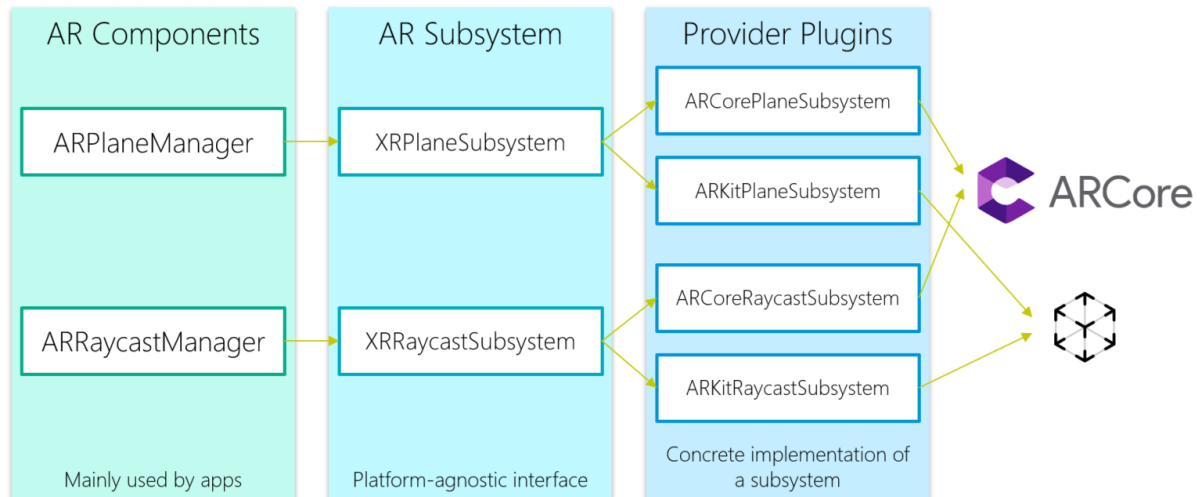


Figure 2: AR foundation architecture.

Hybrid – Clouding

Both private and public clouds are effective and adequate in their respective domains. However, when it comes to large-scale development processes and sharing objectives, there arises a need for more specific and widely accessible information and data sharing. To address this, a hybrid cloud computing model combines elements of both private and public clouds to create a more efficient cloud environment. In simple terms, the hybrid model primarily functions as a private cloud but allows organizations to utilize a public cloud when necessary for information sharing. This model enhances data and application security, especially for sensitive information and compliance with industry and financial regulations. It is widely used in business sectors due to its utility-oriented approach [13].

There are various methods to implement a hybrid cloud, such as selecting the appropriate applications and services to integrate with cloud technology and redistributing and sharing information among the different cloud environments [13].

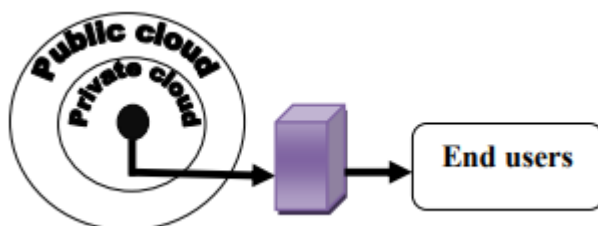


Figure 3: Hybrid-Cloud structure.

Chapter 3

Methodology and Implementation

PostgreSQL.....	12
Spring Boot Web Service.....	13
Testing with Postman.....	14
Deploy on server using Docker.	15
Modeling.....	15
Using Web Service from Unity – C#	16
Merging Components	18
AR View	21

PostgreSQL

Using PostgreSQL as my relational database to store the custom-made components inside. The schema for this database is a simple format using the models of the Spring Boot application as Entities to describe the tables of the following schema.

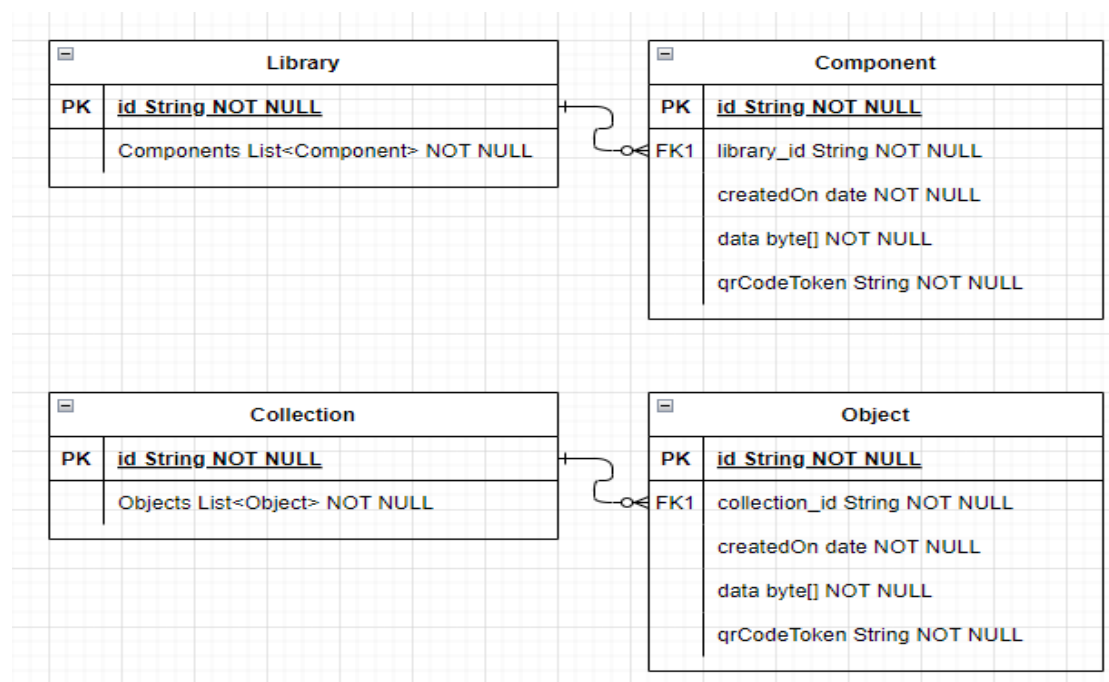


Figure 4: Database Schema for the Web Service App.

As we can see, the database distributes the Entities in 2 main categories with the same attributes. Library category is the one which is stated for the given components of the application. Therefore the components should be placed into the database before implementing the application in unity. Collection category however, is used to store and retrieve the custom-builds from the users or admins by the application at runtime. Despite that their attributes are the same, their whole purpose is different, so we need to implement those two in order to manipulate our data efficiently.

Spring Boot Web Service

Spring Boot Framework with java 17, has a simple layer structure for creating APIs. Controller, Service and Repository. These 3 layers are used to create a functional API using spring boot. In the Controller Layer we use the RequestMapping that indicates the base path of all the APIs in the current controller. Using annotations to create these mappings as well single Rest API operations such as POST for create/insert a model into the database or GET for retrieve that model with a given endpoint. In each API there is a call on the Service Layer; there lies all the business logic regarding the referred API. From the Service Layer we can now call the Repository Layer to implement the business logic of our implementation, such as storing the component into the repository. The Repository Layer is a representation of the schema mentioned in section 3.1, that is created from the Models/Entities that describe each table. There is no need for custom query the database because of the Hibernate Library used within the Sprig-boot application [15].

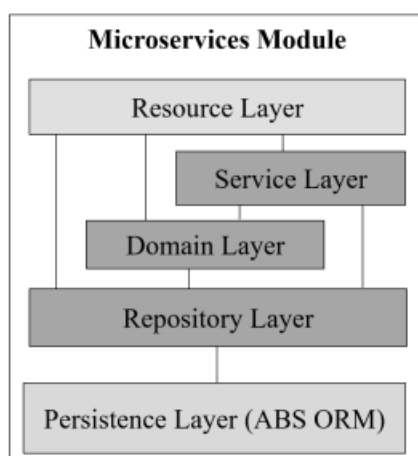


Figure 5: The anatomy of microservices module in Spring-Boot framework [16].

Setting up the configuration of the application is a mandatory task. We need to define the port that the service will run, for this app it is 8443. Add the datasource configurations to connect to the PostgreSQL that we have already set up on the server giving url path, username and password. Also applied the Spring-doc library to view the APIs through an implemented UI at the /swagger-ui.html endpoint. Can only be viewed after the service is up and running.

<http://localhost:8443/swagger-ui/index.html>

Testing with Postman

Postman is a tool that allows the user to make Restful API calls with ease. Using Swagger-UI as a guide, it was easy to create the APIs to test the service.

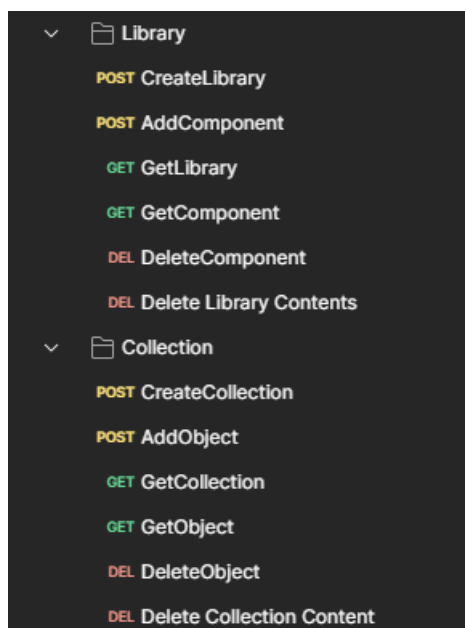


Figure 6: The Postman collection of the APIs that I created for this Spring-Boot App.

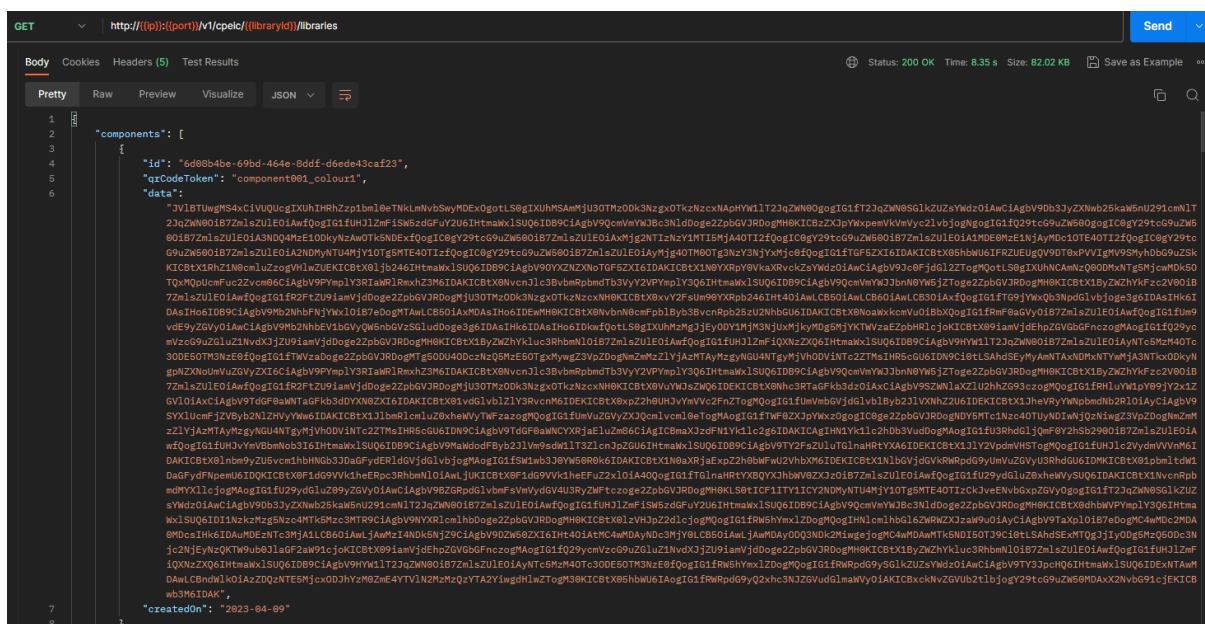


Figure 7: Test Get Library API from Postman.

In the above figure we can notice the components of the API. The current API uses GET operation, on the URL: <http://{{ip}}:{{port}}/v1/cpeic/{{libraryId}}/libraries> . {{}}* this indicates that the value is within an Environment variable. As expected from successful results, we got HTTP status 200 – OK, and the library with the given library Id, that contains a list with multiple components in JSON format. We can only view one element from lack of image capacity.

Deploy on server using Docker.

Docker is a Linux-based platform that permits the containerization of a running application that can run on a server within a container, that source is named image [17]. This application can be the database or the Spring-Boot application. For this paper we put both sources in docker images and deployed them to the server [17,18]. Finally testing the two individually we can now proceed with the implementation of the Unity Project.

Modeling

Using Blender, I was able to create a few basic virtual components [19] so I can use them for the mock Unity application. Created 3 different Types of pillows.



Figure 8: Type A



Figure 9: Type B



Figure 10: Type C

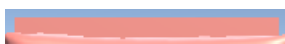


Figure 11: Connectors

From the above images, we can see that Type A has a bottom connector, Type B a top and a bottom connector, and Type C a top connector. Those same objects were also created using different materials and textures.

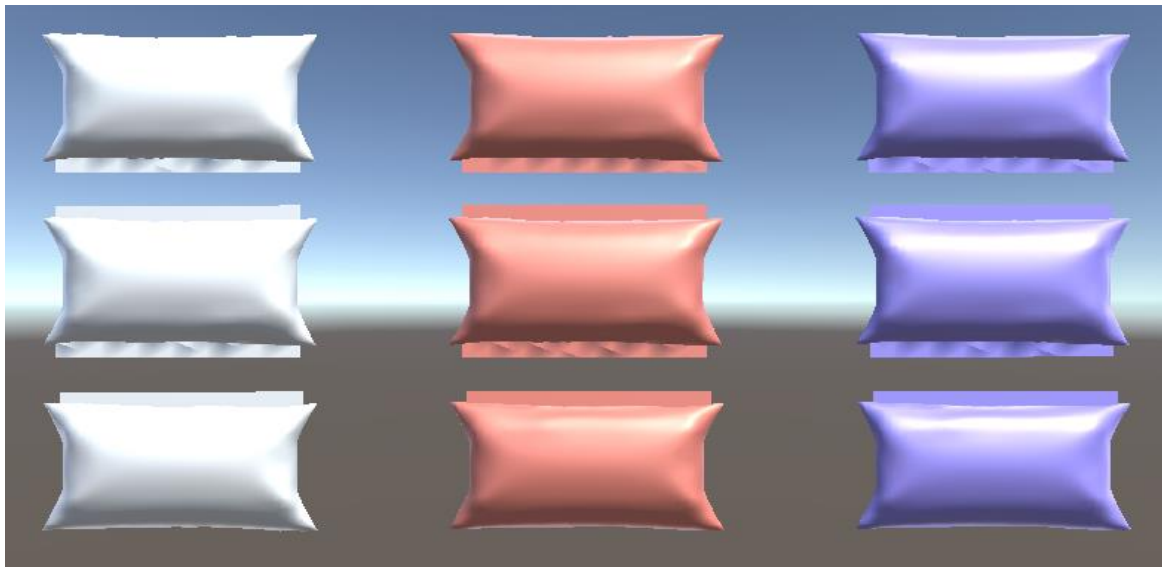


Figure 12: All the components created for the project.

These were exported as “.fbx” format from Blender and then imported to Unity’s Assets folder with a simple drag and drop.

Using Web Service from Unity – C#

We now have the Web Service and Database running in a Docker image within the Cloud. We can make API calls to store the components that are in our Assets folder in Unity. To do that, we need to create DTOs to represent the entities of the database into C# scripts [20]. For serialization purposes the Class should be serializable. Below is the code of the CreateComponentRequest class.

```
using System;

[Serializable]
public class CreateComponentRequest
{
    public string qrCodeToken;
    public byte[] data;
}
```

To create a component the API requires the qrCodeToken as String and the data as a byte array. The token is easy to fill because we used the name of the asset component in the Assets. As for the data, we need to find a way to serialize the whole GameObject with all the components attached to it (Scripts, Transform, Mesh, Mesh Renderer, Tags). I was able to make such a serialization with only 1 implementation. Using the PrefabUtility Library, I was able to break the GameObject to a byte[] type so I can parse it into json for the request body of the API.

POST /v1/cpeic/{libraryId}/components

Parameters

Name	Description
libraryId * required string (path)	libraryId

Request body required application/json

Example Value | Schema

```
{
  "data": [
    "string"
  ],
  "qrCodeToken": "string"
}
```

Responses

Code	Description	Links
200	OK	No links

Figure 13: Create component API example from Swagger-ui.

Making an API call to the cloud it was an easy task as we can see from the code.

```
public static string createComponent(CreateComponentRequest requestBody)
{
    string requestUrl = $"{BASEPATH}{LIBRARY_ID}/components";
    HttpRequest request =
(HttpWebRequest)WebRequest.Create(requestUrl);
    request.Method = "POST";
    request.ContentType = "application/json";

    string jsonRequest = JsonUtility.ToJson(requestBody);

    using (StreamWriter writer = new
StreamWriter(request.GetRequestStream()))
    {
        writer.Write(jsonRequest);
    }
    Debug.Log("json request: " + jsonRequest);
    HttpResponse response = (HttpResponse)request.GetResponse();
    using (StreamReader streamReader = new
StreamReader(response.GetResponseStream()))
    {
        string responseString = streamReader.ReadToEnd();
        CreateResponse component =
JsonUtility.FromJson<CreateResponse>(responseString);
        return component.id;
    }
}
```

While using a static method, we can apply the UnityEngine.JsonUtility for the request and response bodies, System.Net and UnityEngine.Networking Libraries to make simple API request to post the object to the cloud and get its referenceId.

After importing a component into the cloud the API returns an ID that is referencing on the componentID that is stored in the DB. This ID is saved to a unique Text document that is named after the component-name in the Assets Folder. With this we ensure the ease of the QR mode that we can retrieve the components from the Service the same way that Choose From Library mode uses.

How to get the component from the database. Same way we used for the insertion, we can use the exact same steps in reverse to retrieve it. Step 1: get the id stored in the Text with the name of the QR or Component-name that we want. Step 2: Make a GET API call to fetch the data from the server. Step 3: Deserialize the byte[] to GameObject type and finally instantiate the retrieved object to the Scene.

Merging Components

By using the retrieving method we discuss in section 3.4 we need a way to combine the models. Using a Static Class for representing the combined elements is useful, regarding that the navigation of the program is using Scene-Switching [22], which destroys all the references from the previous Scene. A ManageComponents Class was created for this purpose. We imported some rules to the class so we can restrict the users actions. Rule 1: Max merged pillows: 5 (configurable), Rule 2: At least 1 Type A and 1 Type C objects should be combined in order to proceed to the next scene. Rule 3: Models cannot be replaced before deleting the previous value.

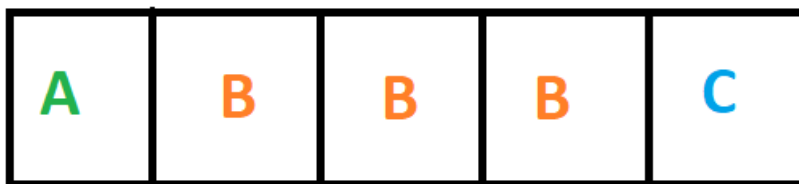


Figure 14: Representation of the one-dimensional array for the ManageComponents Class.

```
private const int MAX_TYPE_B_COMPONENTS = 3;
private const int MIN_TYPE_A_C_COMPONENTS = 1;

private static int componentsAdded = 0;
private static int typeBCounter = 0;

private static GameObject[] grid = new
GameObject[(MAX_TYPE_B_COMPONENTS+(MIN_TYPE_A_C_COMPONENTS*2))];
```

```

public static void addComponent(GameObject component)
{
    if(!isValidToAdd(component))
    {
        Debug.Log("Cannot add this component");
        return;
    }
    else if(component.CompareTag("TypeA"))
    {
        component.GetComponent<Item>().pos = 0;
        grid[0] = component;
    }
    else if(component.CompareTag("TypeB"))
    {
        insertTypeB(component);
    }
    else if(component.CompareTag("TypeC"))
    {
        component.GetComponent<Item>().pos = grid.Length-1;
        grid[grid.Length-1] = component;
    }
    componentsAdded++;
}

```

With this method we add the component to the grid table without being worried about how we can display them.

For the display part we can implement use a function from the caller class to Instantiate the prefabs according to its expectations/limitations. In this case we use the following functions.

```

private void displayComponents()
{
    editUI.SetActive(false);
    deleteAll();
    GameObject[] grid = ManageComponents.getGrid();
    float nextY = -(ManageComponents.getComponentsAdded()/2)+1;
    float nextZ = 0.001f;
    for(int i = grid.Length-1; i >= 0; i--)
    {
        if(grid[i]!=null)
        {
            displayComponent(grid[i], nextY, nextZ);
            nextY += nextYFactor;
            nextZ *= -1f;
        }
    }
}

```

```

    public void displayComponent(GameObject component,float nextY, float
nextZ)
    {
        Instantiate(component,new Vector3(-
0.5f,nextY,nextZ),Quaternion.Euler(0,0,90));
    }

```

As we can see, the expected display should be vertical. The connectors should not overlap so we use the nextZ variable to change the depth of each model by +/- 0.001 unity meters. The results are as follows.

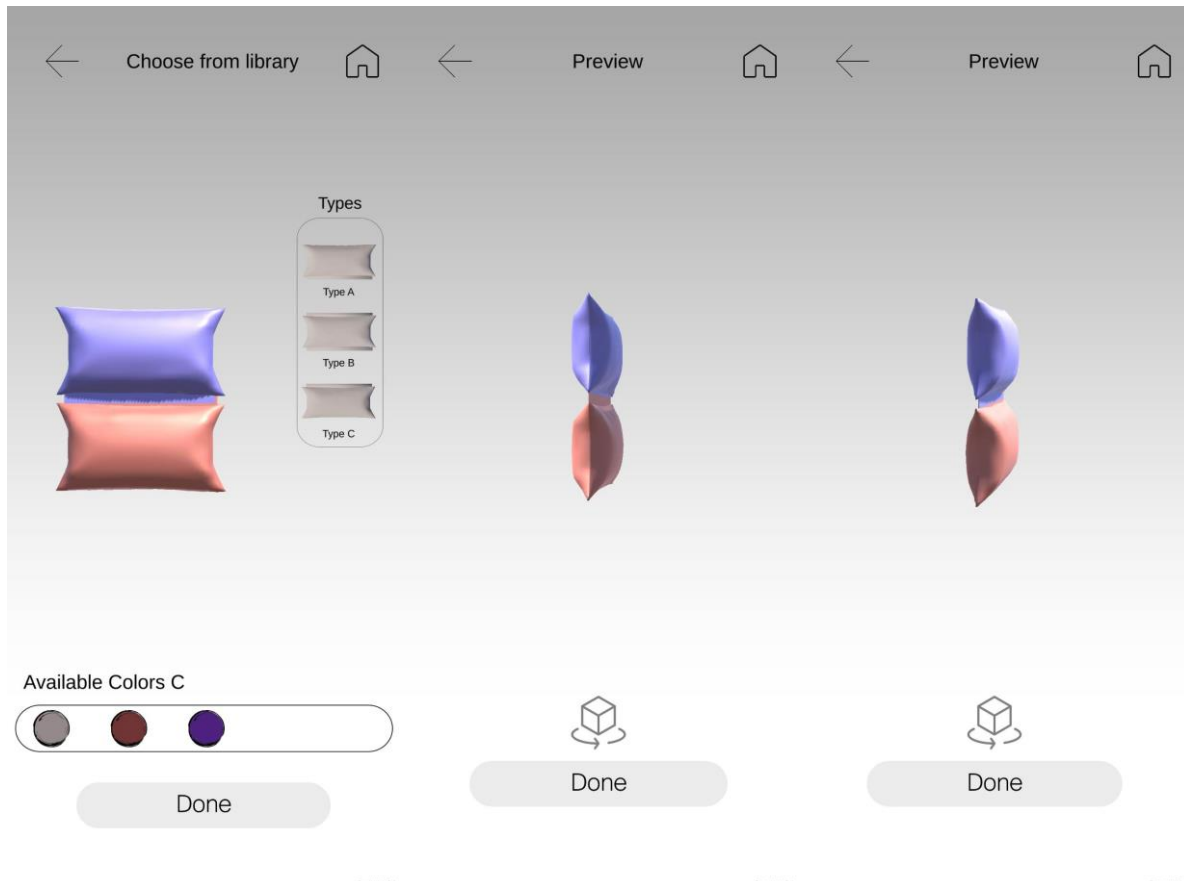


Figure 15: Merging Components. Figures 16,17: Preview Merge Components.

How do we proceed to view the assembled new model into Preview mode? Before changing the scene, we need to implement a logic that makes all the components as one GameObject. This can be accomplished simply by transferring all the objects into a new empty game object that we create when the done button is pressed. This way all the new components become Children of the Parent game object and we can easily track and edit 1 game object instead of >2 GameObjects.

```

public void onClickDone()
{
    //instantiate new gameobject - parent
    GameObject parent = setParent();
    //get all components in the scene
    List<GameObject> components = GetAllComponents();
    //insert all components into a new game object as their parents
    setChildrenToParent(components, parent);
    //change to Preview scene
    SceneLoader.SwitchScene(SceneLoader.Scene.View);
}

```

By creating the new game object as the parent, we need to set some components on it so we can take advantage of the singular object transformations. Therefore, we should initialize Parent with Tag name Parent, or any preferable unique tag name, set its position to the beginning of the axis (0, 0, 0). Finally, to not lose the object when the new scene awakening, we need to add a script to let it stay “alive” and not to be destroyed “DontDestroy”. Underneath is the example of the setParent() method.

```

private GameObject setParent()
{
    GameObject parent = new GameObject();
    parent.transform.position = Vector3.zero;
    parent.name = "parent";
    parent.tag = "Parent";
    parent.AddComponent<DontDestroy>();
    return parent;
}

```

AR View

Now that we have decided on the pattern of the new model we can proceed to the “View in Space” Scene. The AR scene uses 2 major components with AR core. The AR session origin that contains the AR camera, and the AR session that contains the AR Input Manager Component. To calibrate the AR mode, from virtual to real word we can use the AR Tracked Image Manager and attach it to the AR Session Origin as a new component.

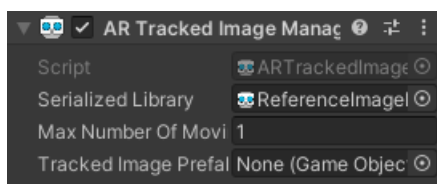


Figure 18: AR Tracked Image Manager Component.

To set up the TIM we must add a Serialized Library. This library must contain the unique image(s) that we will use for displaying our model on top of that.

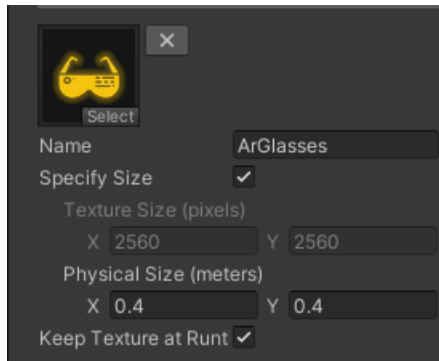


Figure 19: Reference Image Library.

Here is an example of a 1:1 image that we use for tracking. We also must activate the “Keep Texture at Runtime” Boolean. The reason to do that is because if we don’t the model won’t display at runtime as it should be. Physical size is an error and trial. Using a real item next to the virtual one to make sure that the measurement is correct by adjusting the X,Y parameters.

Tracked Image Prefab is the prefab we use to augment to the real world. In our case this will be the “Parent” prefab that has allocated all the components as Children. But because this is a new scene this cannot be added statically. To resolve this technical issue, we use a manager script that allows us to get reference on the TIP component and reference it as the Parent object.

```
private GameObject parent;
public ARTrackedImageManager trackedImageManager;

void Awake()
{
    parent = GameObject.FindWithTag("Parent");
    Destroy(parent.GetComponent<CameraMovement>());
    trackedImageManager.trackedImagePrefab = parent;
}
```



Figures 20, 21: AR View In Space Mode.

By adjusting the Unique Tracked Image we can view the model in different rotations. As we can see in figure 20 the image is at a vertical position, so the model is being displayed horizontally. Instead, in figure 21 the image is horizontally so the model is being shown in a vertical display.

Chapter 4

Evaluation

4.1 Overview	23
4.2 Used Technologies	27
4.3 Discussion	26

Overview

This section describes a comprehensive analysis of the Hybrid-Clouding Solution, evaluating its advantages and disadvantages, as well as assessing the feasibility and challenges associated with implementing the 3D Model-Assembly and Augmented Reality (AR) display for newly created objects. By examining these aspects, this study aims to shed light on the potential benefits and limitations of adopting a Hybrid-Clouding Solution and the ease of integrating the 3D Model-Assembly and AR display technology.

The primary objective of this thesis is to evaluate the Hybrid-Clouding Solution, considering its impact on various aspects such as performance, scalability, security, and cost-effectiveness. The analysis will involve examining the benefits and drawbacks of this approach, comparing it to other cloud deployment models, and identifying its suitability for different organizational needs and requirements.

In addition to evaluating the Hybrid-Clouding Solution, this study will also delve into the implementation challenges and ease associated with the 3D Model-Assembly and AR display technology. The focus will be on assessing the technical requirements, training needs, and potential difficulties that may arise when integrating these technologies into existing workflows. Furthermore, the study will explore the potential benefits and productivity gains that can be achieved through the adoption of 3D Model-Assembly and AR display technology for visualizing and interacting with newly created objects.

To achieve the objectives outlined above, a mixed-methods approach will be employed. A comprehensive review of existing literature will be conducted to gather insights into the Hybrid-Clouding Solution, 3D Model-Assembly, and AR display technology. This literature review will serve as a foundation for understanding the current state of these technologies, their benefits, and their limitations.

Furthermore, empirical research will be conducted to collect primary data from professionals who have experience with Hybrid-Clouding Solutions and the implementation of 3D Model-Assembly with AR display technology. Surveys and interviews will be utilized to gather qualitative and quantitative data, providing real-world perspectives on the challenges and opportunities associated with these technologies.

The collected data will be analyzed using appropriate statistical methods and qualitative analysis techniques. The evaluation of the Hybrid-Clouding Solution will focus on assessing its advantages and disadvantages in terms of performance, scalability, security, and cost-effectiveness. The analysis of the 3D Model-Assembly and AR display implementation will explore the technical requirements, training needs, and potential productivity gains that can be achieved through these technologies.

By the end of this study, it is anticipated that a comprehensive understanding of the benefits, limitations, and implementation challenges of the Hybrid-Clouding Solution and the 3D Model-Assembly with AR display technology will be obtained. The findings and recommendations derived from this analysis will provide valuable insights to organizations considering the adoption of a Hybrid-Clouding Solution and the implementation of 3D Model-Assembly and AR display technology for visualizing and interacting with newly created objects.

Used Technologies

This section outlines the technologies utilized in the development of the project. The chosen technologies were carefully selected to meet the requirements and objectives of the research. The following technologies were employed:



Figure 22: Spring Boot framework logo.



Figure 23: Java logo.



Figure 24: PostgreSQL logo.

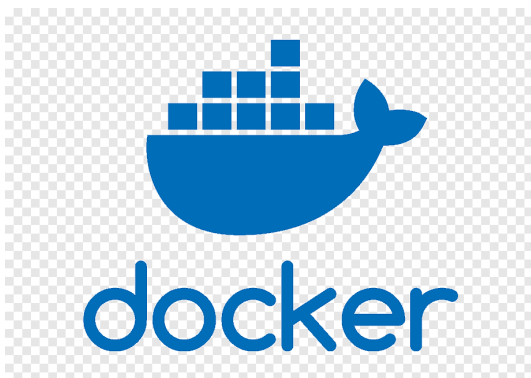


Figure 25: Docker logo.



Figure 26: Postman logo.

The aforementioned technologies played a vital role in supporting the HC architecture of the project. They were utilized for various tasks, including designing the database schemas, developing the APIs, and conducting thorough testing. Each of these technologies was indispensable for ensuring the successful implementation and optimal functioning of the final outcome.



Figure 27: Blender logo.

Blender was the only modeling technology that was used in the implementation of this project. It was easy to create and export the models through its useful and enhanced UI.



Figure 28: Unity logo

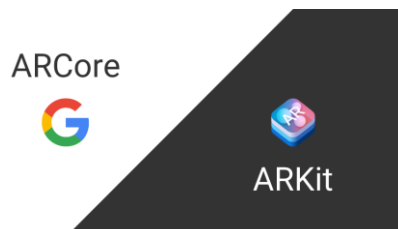


Figure 29: ARCore and ARKit Logos

Unity's settings for Android build greatly facilitate the process of creating applications for the Android platform. The user-friendly interface and intuitive controls make it easy to handle 3D models and components, providing a seamless experience for developers. Scripting in C# further enhances the capabilities of Unity, allowing for complex interactions and behavior customization. Moreover, the integration of libraries such as ARCore and ARKit expands the possibilities of augmented reality development, enabling the creation of immersive and interactive experiences.

Discussion

The Evaluation chapter thoroughly examined the effectiveness of the proposed Hybrid Cloud architect solution. This section provides a summary of the evaluation, highlighting the benefits and drawbacks of the solution.

The Hybrid Cloud architect solution offers several advantages. One notable benefit is its efficient utilization of space on smaller devices. By leveraging cloud resources, the solution effectively manages data and computational requirements, making optimal use of limited storage capacity. This feature is particularly advantageous in situations where storage limitations are a concern.

Another strength of the solution lies in its seamless integration with Rest API-driven systems. This compatibility allows the solution to be applied across a wide range of platforms, including websites and mobile applications. It provides flexibility and scalability to accommodate diverse architectures.

However, certain limitations were identified during the evaluation process. One significant drawback is the increased response time compared to the static prefab solution. Retrieving the byte array and deserializing it to the GameObject takes longer in the Hybrid Cloud architect solution. This delay can impact real-time interactions and the overall user experience, especially in scenarios where quick updates are crucial.

Additionally, the implementation of serialization/deserialization at runtime presents challenges during deployment. The inability to build the app into smartphones using the PrefabUtility Library limits testing to the Unity Editor, hindering a comprehensive assessment on target devices. This limitation creates uncertainty when transitioning from development to production.

Furthermore, it is crucial to consider the cost and complexity associated with the Hybrid Cloud architect solution. Integrating multiple resources, such as cloud infrastructure, server setups, and API configurations, requires in-depth knowledge of various technologies. The implementation process may incur higher expenses and demand additional development efforts.

In conclusion, the evaluation of the Hybrid Cloud architect solution highlights its benefits in terms of space optimization on small devices and compatibility with Rest API-driven systems. However, the solution's drawbacks, including increased response time and challenges in runtime deployment, should be carefully evaluated. Moreover, the cost and implementation complexity must be considered to ensure practicality within the project's scope and constraints.

Chapter 5

Conclusion

Limitations	29
Future Work.....	30

Limitations

While the research thesis on time reduction through API calls for retrieving and desterilizing a 3D model at runtime demonstrates several advancements, it is important to acknowledge certain limitations that were encountered during the course of the study. These limitations include:

Incompatibility with Smart-phone Devices: One significant limitation of the research is the inability to achieve the desired outcome on smart-phone devices. The implementation of the proposed solution is currently restricted to Unity's Editor, which limits its practical applicability. Further investigation and development are required to address this limitation and explore potential solutions for integrating the proposed approach into smart-phone devices.

High Technological Requirements: The successful implementation of the project required expertise in multiple technologies, including the Spring Boot framework, APIs, Java, Docker, Linux terminal, Unity, and C#. The diverse knowledge base and skill set required may limit

the accessibility and feasibility of the proposed solution for researchers or developers who are not familiar with these technologies.

Time Consumption: The implementation of the project was found to be time-consuming. This time investment stemmed from various factors, such as the complexity of the problem domain, the need to integrate multiple technologies, and the intricacies of working with 3D models and APIs. The considerable time required for implementation might pose challenges for researchers or developers with limited resources or time constraints.

External Dependencies: The research heavily relied on external APIs and libraries for retrieving and desterilizing 3D models at runtime. Any changes or limitations imposed by these external dependencies could impact the functionality and performance of the proposed solution. The research is subject to potential constraints imposed by these third-party components, which may introduce uncertainties and affect the overall reliability of the system.

Generalizability: The research thesis primarily focuses on the proposed solution's implementation in a specific context. The findings and conclusions drawn from this study might not be easily generalizable to other domains or scenarios. The limitations associated with generalizability should be acknowledged, and further research is required to assess the scalability and applicability of the proposed solution in different contexts.

Addressing these limitations would provide valuable insights and opportunities for further research and development, ultimately enhancing the practicality and effectiveness of the proposed approach for reducing time consumption through API calls for retrieving and desterilizing 3D models at runtime.

Future Work

Based on the research findings and identified limitations, there are several potential areas for future work and development. These areas include:

Dynamic Button Creation: One area of improvement is the implementation of a fast and effective dynamic button creation system after the application's start. This enhancement would enable administrators to add new items to the database without requiring a patch or

update to the application itself. By implementing such a feature, the system would gain flexibility and adaptability, allowing for easier management and expansion of the application's content.

Website Integration for Unity's Prefab Files: An important future direction is to explore and develop methods to adjust existing websites to support Unity's prefab files. This integration would enable the loading of Unity prefabs dynamically using a web service. By achieving this compatibility, users would have the ability to interact with 3D models and other Unity elements seamlessly within a web environment. This expansion would significantly broaden the accessibility and reach of the application.

"Share" Mode for social media: To enhance user engagement and satisfaction, the implementation of a "share" mode for social media platforms is a promising avenue for future work. Enabling users to share their thoughts, experiences, and 3D models with friends and family through social media channels would facilitate communication and foster a sense of community. This feature would provide users with the opportunity to gather opinions and feedback, thereby enhancing their overall experience and promoting the application's growth.

Performance Optimization: Further research and development can be conducted to optimize the performance of the proposed solution. This includes refining the API calls, streamlining the desterilization process, and identifying potential bottlenecks or areas for improvement in the runtime loading of 3D models. By addressing performance-related challenges, the application can deliver a smoother and more efficient user experience.

Usability Testing and User Feedback: Conducting usability testing and gathering user feedback is essential to understanding the strengths and weaknesses of the application. Incorporating user perspectives and preferences can provide valuable insights for future enhancements and refinements. Iterative user testing and feedback collection should be conducted to ensure that the application meets user expectations and aligns with their needs.

Scalability through Load Balancers and Kubernetes: To handle the increasing demands of assembling 3D components at runtime, a scalable infrastructure is essential. Load balancers can distribute the incoming requests across multiple servers, preventing any single server from becoming overwhelmed. Additionally, adopting container orchestration frameworks like Kubernetes can facilitate efficient scaling of the application by automating the deployment, scaling, and management of containerized services.

Resource Management and Consumption: As the volume of 3D components increases, resource management becomes crucial. Future work should focus on optimizing resource consumption and utilization to ensure efficient and cost-effective operations. This could involve strategies such as dynamic resource allocation, load-aware scheduling, and auto-scaling based on workload demands. By effectively managing resources, the solution can handle the growing demands of runtime assembly without compromising performance or incurring unnecessary expenses.

Implementation of Queues for Asynchronous Data Exchange: As the complexity and volume of 3D components increase, there may be a need for asynchronous data exchange, especially for handling metadata associated with the components. Implementing queues can facilitate the decoupling of components, enabling asynchronous communication and improving system responsiveness. Queues can be used to buffer and process metadata in the background, reducing the impact on real-time assembly operations and ensuring smoother performance.

Robust Error Handling and Fault Tolerance: In a large-scale system, failures and errors are inevitable. Future work should focus on enhancing the solution's fault tolerance by implementing robust error handling mechanisms. This can involve techniques such as error detection, graceful degradation, and automated recovery. By minimizing the impact of failures and providing seamless error recovery, the system can maintain high availability and minimize disruptions during runtime assembly operations.

Continuous Monitoring and Performance Optimization: To ensure the ongoing efficiency and effectiveness of the solution, continuous monitoring and performance optimization are vital. Implementing monitoring tools and techniques can provide insights into system behavior, performance bottlenecks, and resource utilization. This information can then be used to identify areas for optimization and fine-tuning, resulting in improved overall system performance and reliability.

By pursuing these avenues for future work, the proposed solution can be further refined and expanded, offering enhanced functionality, flexibility, and user engagement.

References

- [1] Tang, J.K., Lau, W.M., Chan, K.K. and To, K.H., 2014, December. AR interior designer: Automatic furniture arrangement using spatial and functional relationships. In 2014 International Conference on Virtual Systems & Multimedia (VSMM) (pp. 345-352). IEEE.
- [2] Doerr, M., Tzompanaki, K., Theodoridou, M., Georgis, C., Axaridou, A. and Havemann, S., 2010, September. A Repository for 3D Model Production and Interpretation in Culture and Beyond. In VAST (Vol. 2010, p. 11th).
- [3] Satre, S.M., Mangla, H., Sambhare, S. and Kshatriya, S., 2020. Furniture Organizer Application Using Augmented Reality. In: Journal of Emerging Technologies and Innovative Research (JETIR), 7(4).
- [4] Viyanon, W., Songsuittipong, T., Piyapaisarn, P. and Sudchid, S., 2017, July. AR furniture: Integrating augmented reality technology to enhance interior design using marker and markerless tracking. In Proceedings of the 2nd International Conference on Intelligent Information Processing (pp. 1-7).
- [5] Shetty, V., Samirasimha, R. and Bedere, S., 2018, September. Performance evaluation of augmented reality based 3D modelling furniture application. In 2018 international conference on advances in computing, communications and informatics (ICACCI) (pp. 2426-2431). IEEE.
- [6] Ali, A., Zain, S.M. and Yasin, S.N.T.M., 2022. Utilizing Marker to Integrate Augmented Reality in Furniture Using Unity 3D. International Journal of Advanced Research in Technology and Innovation, 4(2), pp.99-108.
- [7] Guedes, L.S., Marques, L.A. and Vitório, G., 2020. Enhancing interaction and accessibility in museums and exhibitions with augmented reality and screen readers. In Computers Helping People with Special Needs: 17th International Conference, ICCHP 2020, Lecco, Italy, September 9–11, 2020, Proceedings, Part I 17 (pp. 157-163). Springer International Publishing.







- [8] Linowes, J. and Babilinski, K., 2017. Augmented reality for developers: Build practical augmented reality applications with unity, ARCore, ARKit, and Vuforia. Packt Publishing Ltd.
- [9] Lanham, M., 2018. Learn ARCore-Fundamentals of Google ARCore: Learn to build augmented reality apps for Android, Unity, and the web with Google ARCore 1.0. Packt Publishing Ltd.
- [10] Duda, J. and Oleszek, S., 2020. Realization of PLM application integration with AR technology. *International Journal of Industrial Engineering and Management*, 11(4), p.263.
- [11] Desierto, A.J.R., Recaña, A.S.A., Arroyo, J.C.T. and Delima, A.J.P., 2020. GoonAR: A bilingual children storybook through augmented reality technology using unity with Vuforia framework. *Int. J. Adv. Trends Comput. Sci. Eng*, 9.
- [12] Chaudhry, T., Juneja, A. and Rastogi, S., 2021. AR foundation for augmented reality in unity. *Int. J. Adv. Eng. Manage.*, 3(1), pp.1-7.
- [13] Srinivasan, A., Quadir, M.A. and Vijayakumar, V., 2015. Era of cloud computing: A new insight to hybrid cloud. *Procedia Computer Science*, 50, pp.42-51.
- [14] Gajewski, M. and Zabierowski, W., 2019, May. Analysis and comparison of the Spring framework and play framework performance, used to create web applications in Java. In *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)* (pp. 170-173). IEEE.
- [15] Duarte, A., 2021. Spring Boot. In *Practical Vaadin: Developing Web Applications in Java* (pp. 289-303). Berkeley, CA: Apress
- [16] Naily, M.A., Setyautami, M.R.A., Muschevici, R. and Azurat, A., 2018. A framework for modelling variable microservices as software product lines. In *Software Engineering and Formal Methods: SEFM 2017 Collocated Workshops: DataMod, FAACS, MSE, CoSim-CPS, and FOCLASA, Trento, Italy, September 4-5, 2017, Revised Selected Papers 15* (pp. 246-261). Springer International Publishing.

- [17] Docker, I., 2020. Docker. linea].[Junio de 2017]. Disponible en: <https://www.docker.com/what-docker>.
- [18] You, L. and Sun, H., 2022. Research and Design of Docker Technology Based Authority Management System. Computational Intelligence and Neuroscience, 2022.
- [19] Nguyen, V.T. and Dang, T., 2017, October. Setting up virtual reality and augmented reality learning environment in unity. In 2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct) (pp. 315-320). IEEE.
- [20] Kelly, S. and Kumar, K., 2021. Unity Networking Fundamentals. Apress.
- [21] Chikaraddi, A.K., Kanakaraddi, S.G., Seeri, S.V., Naragund, J.G. and Giraddi, S., 2022. ARFA-QR Code Based Furniture Assembly Using Augmented Reality. In Sustainable Communication Networks and Application: Proceedings of ICSCN 2021 (pp. 321-334). Singapore: Springer Nature Singapore.
- [22] Halpern, J. and Halpern, J., 2019. Introduction to unity. Developing 2D Games with Unity: Independent Game Programming with C#, pp.13-30.

Appendix A

QR Scanner Implementation

In Appendix A, we provide a detailed description of the QR scan implementation used in the research project. The QR scan implementation served as a key component for identification of 3D components in the study. The section outlines the technical aspects of the QR scan implementation, including the software and hardware requirements, the development environment, and the integration with the existing system. Additionally, the step-by-step process for scanning QR codes and retrieving relevant data is provided, along with any necessary configuration settings. Screenshots are included to illustrate the user interface and workflow of the QR scan implementation. This comprehensive documentation of the QR scan implementation allows readers to gain a deeper understanding of the data collection methodology employed in the research project.

		
component001 _colour1	component001 _colour2	component001 _colour3
		
component002 _colour1	component002 _colour2	component002 _colour3

		
component003 _colour1	component003 _colour2	component003 _colour3

Figure 30: All the QR codes with their decoded messages.

For the implementation I used the ZXing.NET open-source library that was available through the web. This Library allows the developers to decode the QR code. Firstly, I got a reference for the device camera and rotated it accordingly to view it vertically. The Scene starts by using the Scan mode to import a component and each time a successful scan is made by using the Snapshot button to scan, the camera mode switches off and the components are revealed. When the button is pressed, by using the ZXing library we are able to decode the QR code and get the corresponding component from the Web using its ID that its stored inside of a text in assets.

```
[SerializeField] private TextMeshProUGUI resultText;
[SerializeField] private RawImage cameraDisplay;

[SerializeField] private GameObject scanNextBtn;
[SerializeField] private GameObject scanBtn;

private PrefabManagerQR prefabManager;

private WebCamTexture camTexture;

void Awake()
{
    prefabManager = GetComponent<PrefabManagerQR>();
    // Start camera feed
    camTexture = new WebCamTexture();
    camTexture.requestedWidth = 1280; // Set the desired width
    camTexture.requestedHeight = 620; // Set the desired height
    cameraDisplay.texture = camTexture;

    // settle it to vertical display
    cameraDisplay.rectTransform.Rotate(new Vector3(0, 0, 0));
    cameraDisplay.rectTransform.Rotate(new Vector3(0, 0, 270));
}
```

```

        RectTransform cameraDisplayRectTransform =
cameraDisplay.rectTransform;
        cameraDisplayRectTransform.sizeDelta = new Vector2(Screen.width *
1.5f, Screen.height * 0.5f);

        cameraDisplay.gameObject.SetActive(true);
        camTexture.Play();

        scanNextBtn.SetActive(false);
        scanBtn.SetActive(true);
    }

    public void OnClickScan()
    {
        // Set up barcode reader
        IBarcodeReader reader = new BarcodeReader();

        // Scan for QR code
        Result result = reader.Decode(camTexture.GetPixels32(),
                                     camTexture.width,
                                     camTexture.height);

        if (result != null)
        {
            // QR code decoded
            resultText.text = result.Text;

            camTexture.Stop();
            scanBtn.SetActive(false);
            scanNextBtn.SetActive(true);
            cameraDisplay.gameObject.SetActive(false);
            //display scanned component
            displayOnPrefabManagerQR(result.Text);
        }
        else
        {
            // QR code not found
            resultText.text = "No QR code found";
        }
    }
}

```

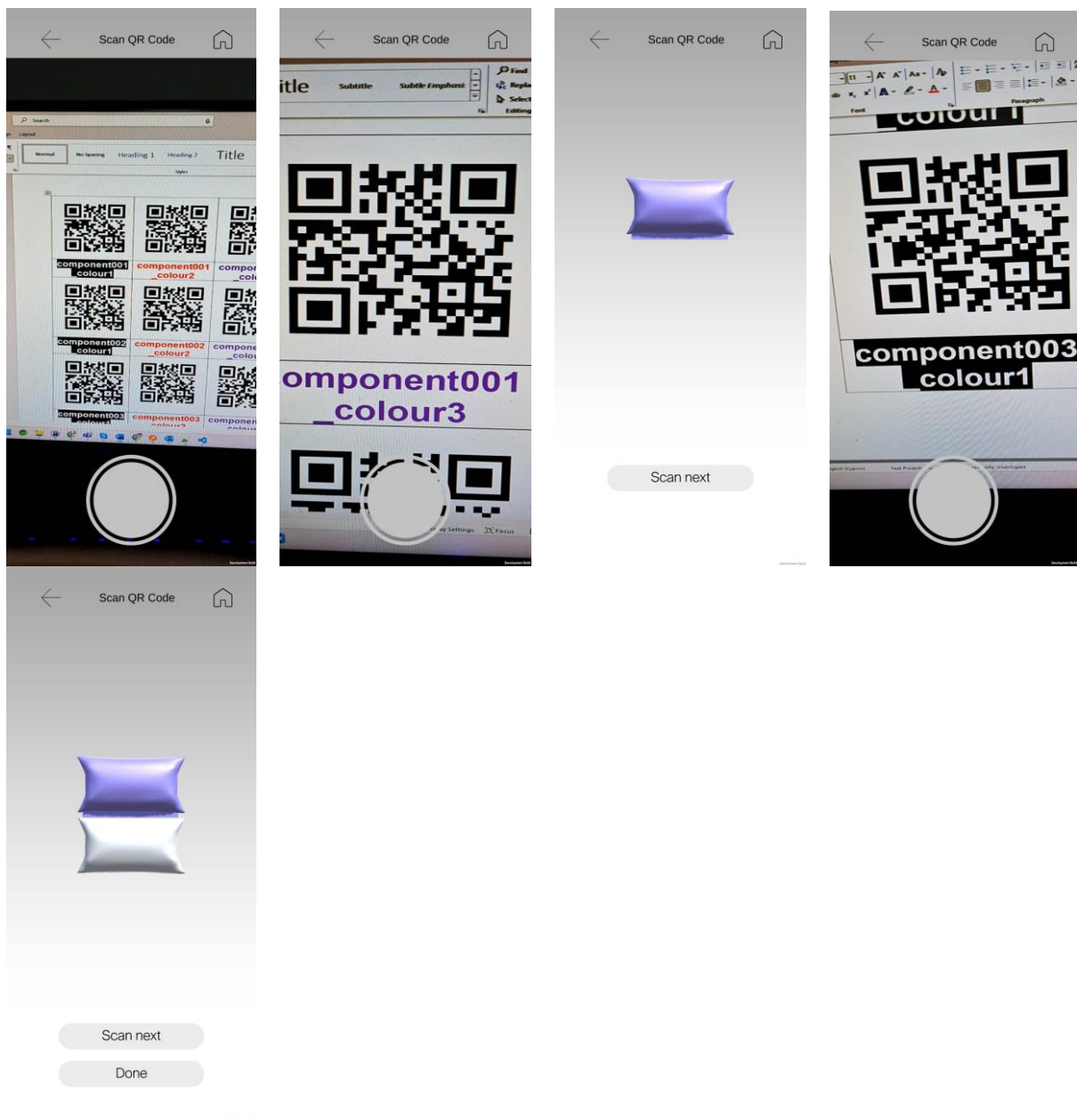


Figure 31 - 35: the Workflow of the whole procedure.

Appendix B

Navigation System in Unity between Scenes

Appendix B provides a concise description of the navigation system implemented in Unity for seamless transitions between scenes in the research project. The appendix covers the technical aspects, development environment, and integration within the project architecture. It outlines the core functionalities such as scene loading, switching, and data transfer. Step-by-step instructions and code snippets are included, addressing challenges encountered and offering potential solutions. Appendix B equips readers with the necessary information to implement efficient scene navigation in their own Unity projects.

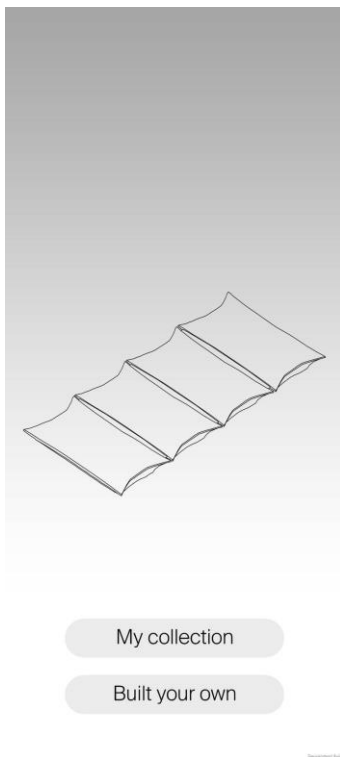
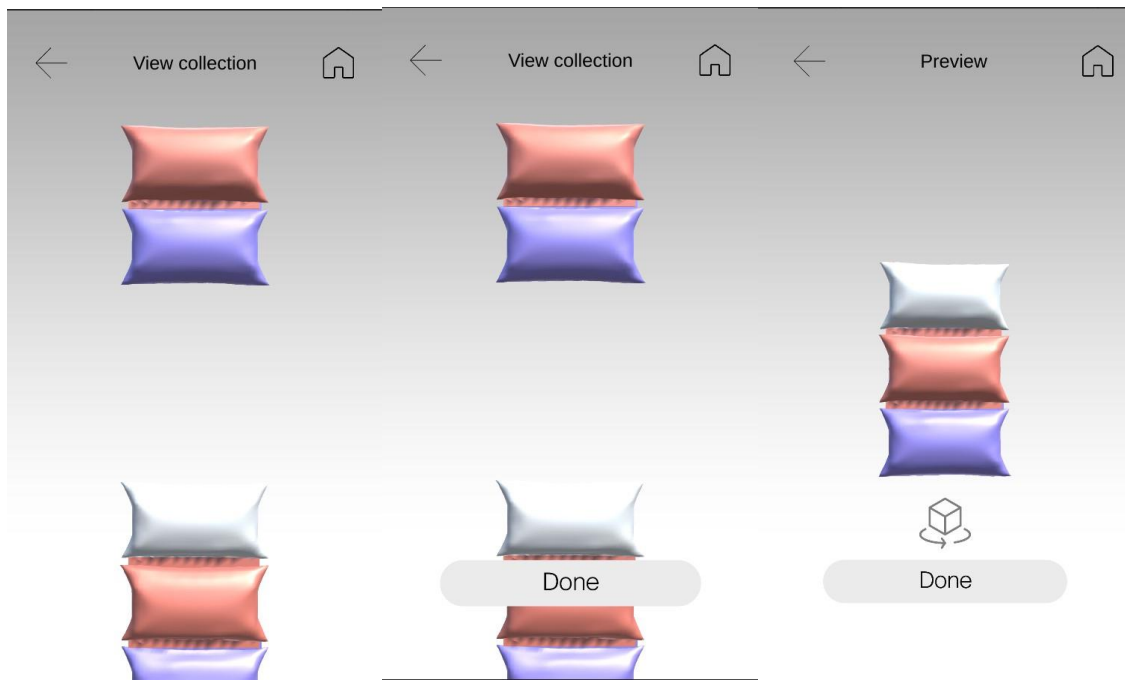


Figure 36: Home Screen



Figures 37 – 39: View Collection Scene to Preview scene.

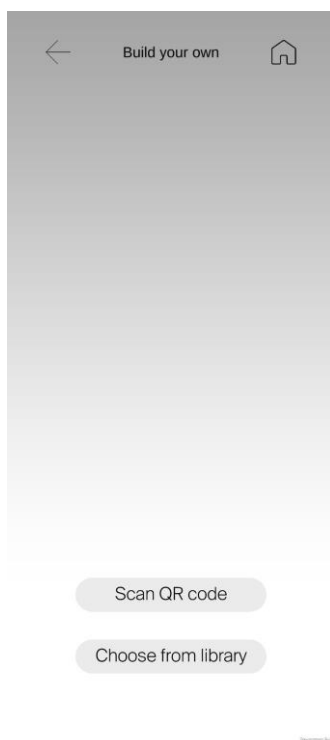
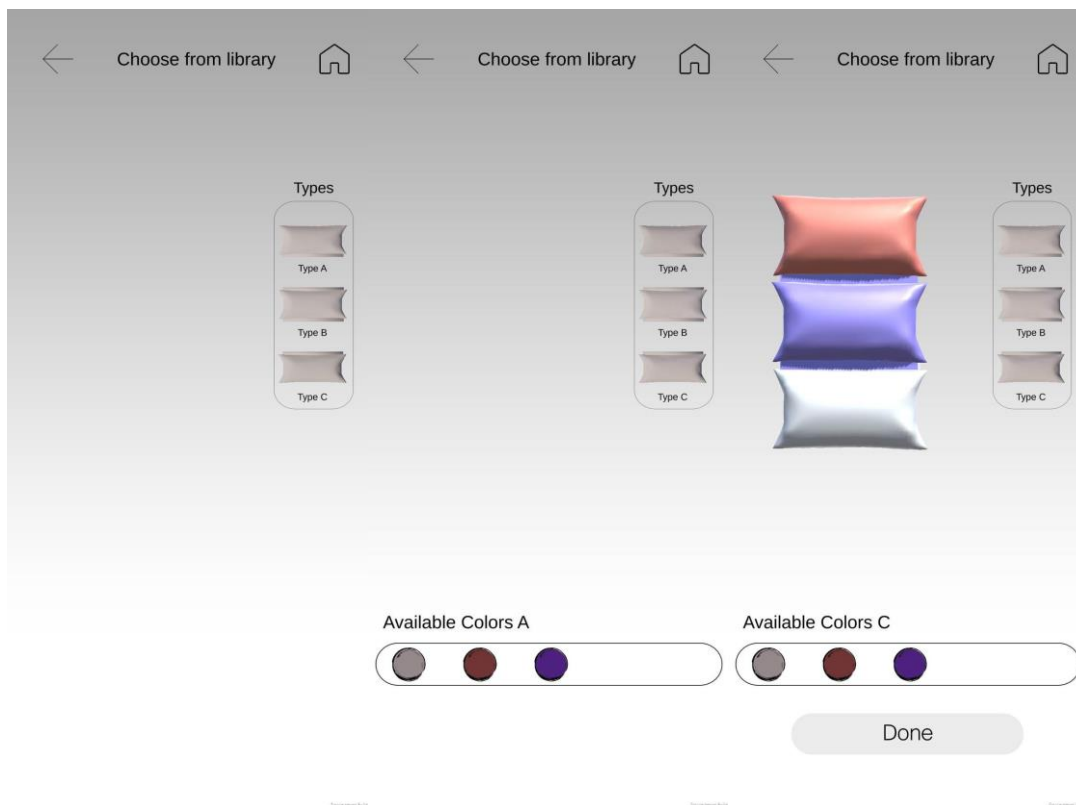
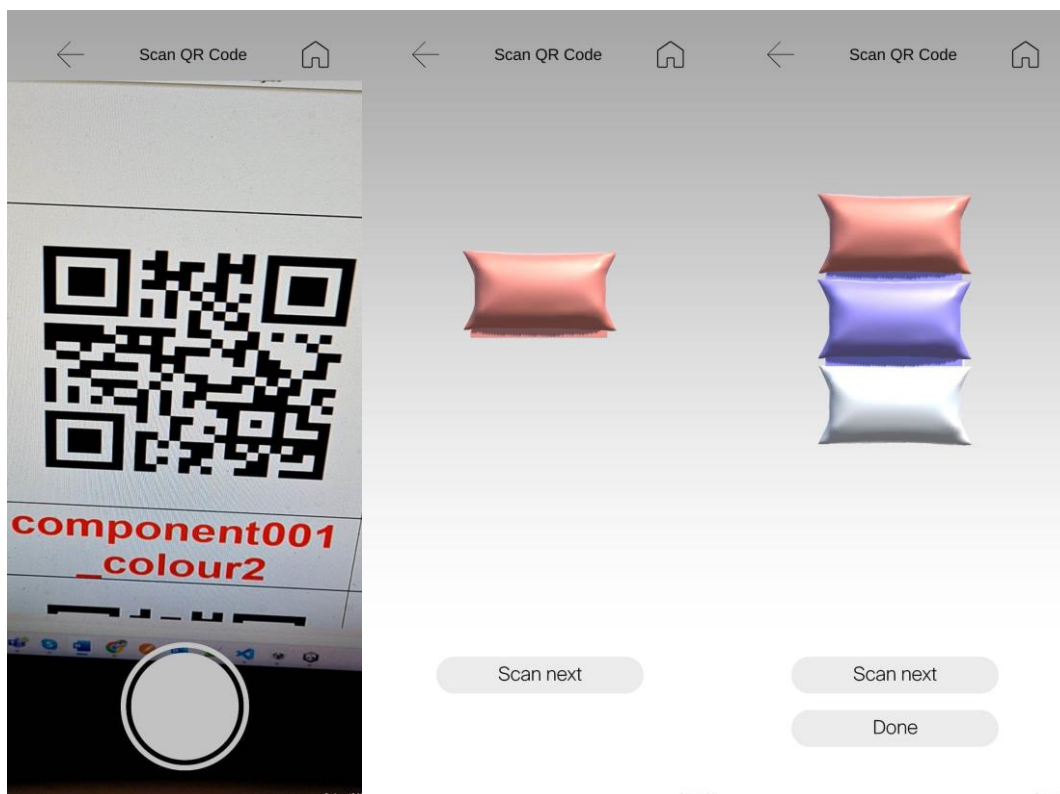


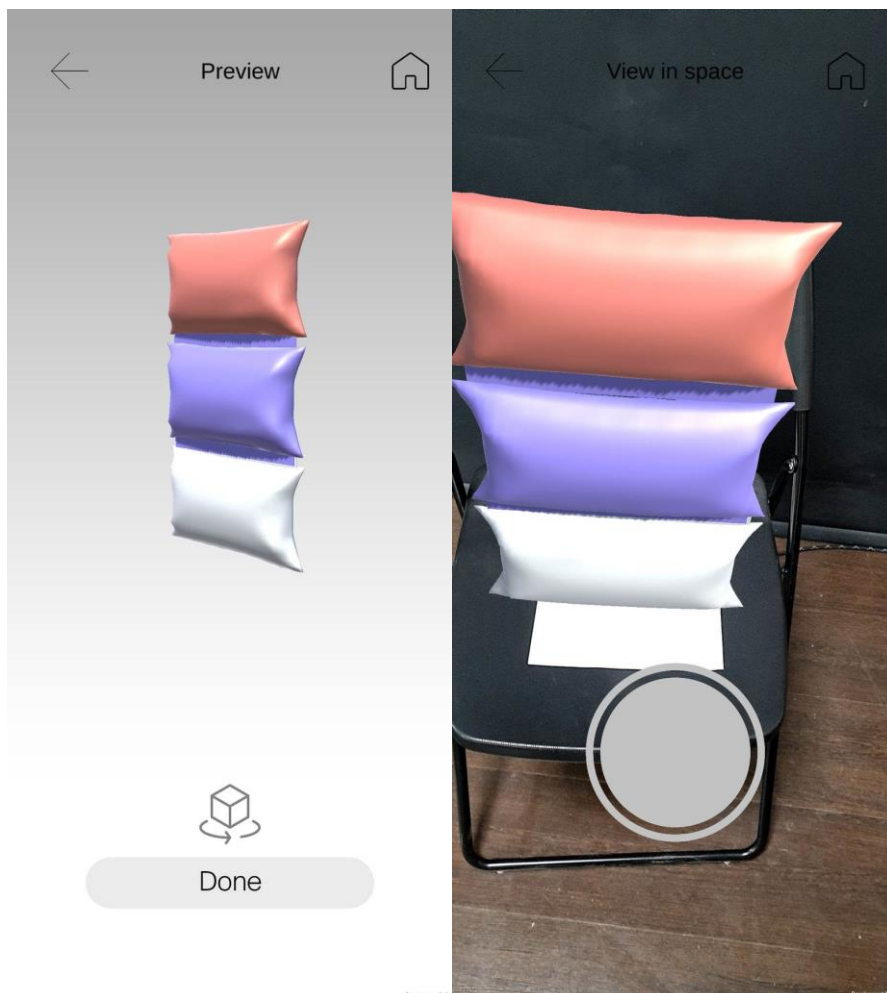
Figure 40: Build Your Own Scene.



Figures 41 – 43: Choos from Library Scene.



Figures: 44 – 46: Scan QR Scene.



Figures: 47 – 48: Preview Scene and View in Space Scene (AR).

The scenes Choose from Library and Scan QR, when done button is pressed, they navigate to Preview Scene, and from there to View in Space Scene.

The following code is the one that I used in my implementation so I could navigate using scenes.

```
private static SceneLoader.Scene previous;

public void OnClickToBuildYourOwn()
{
    emptyGrid();
    SceneLoader.SwitchScene(SceneLoader.Scene.BuildYourOwn);
}

public void OnClickToViewCollection()
{
    emptyGrid();
    previous = SceneLoader.Scene.ViewCollection;
    SceneLoader.SwitchScene(SceneLoader.Scene.ViewCollection);
}
```

```

public void OnClickToChooseFromLibrary()
{
    deleteCameraMovement();
    previous = SceneLoader.Scene.BuildYourOwn;
    SceneLoader.SwitchScene(SceneLoader.Scene.ChooseFromLibrary);
}

public void OnClickToScanQR()
{
    Debug.Log("to Scan QR");
    deleteCameraMovement();
    previous = SceneLoader.Scene.BuildYourOwn;
    SceneLoader.SwitchScene(SceneLoader.Scene.ScanQR);
}

public void OnClickToMainMenu()
{
    Debug.Log("to Home");
    emptyGrid();
    SceneLoader.SwitchScene(SceneLoader.Scene.Home);
}

public void OnClickToView()
{
    Debug.Log("to View");

    SceneLoader.SwitchScene(SceneLoader.Scene.View);
}

public void OnClickToBack()
{
    Debug.Log("back to " + previous);
    deleteCameraMovement();
    SceneLoader.SwitchScene(previous);
}

public void OnClickToAR()
{
    setParent();
    SceneLoader.SwitchScene(SceneLoader.Scene.AR);
}

```

SceneLoader.Scene is an enumeration.

```

using UnityEngine;
using UnityEngine.SceneManagement;

public class SceneLoader : MonoBehaviour
{
    public enum Scene { Home, BuildYourOwn, ViewCollection, View,
        ChooseFromLibrary, ScanQR, AR}
}

```

```
public static void SwitchScene(Scene scene)
{
    SceneManager.LoadScene(scene.ToString());
}
}
```