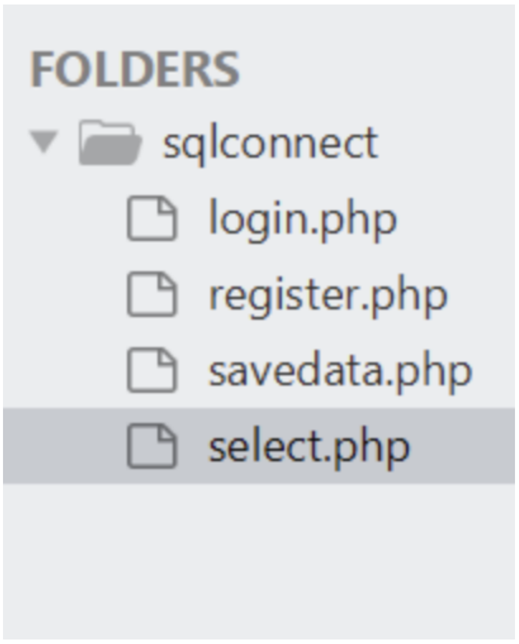
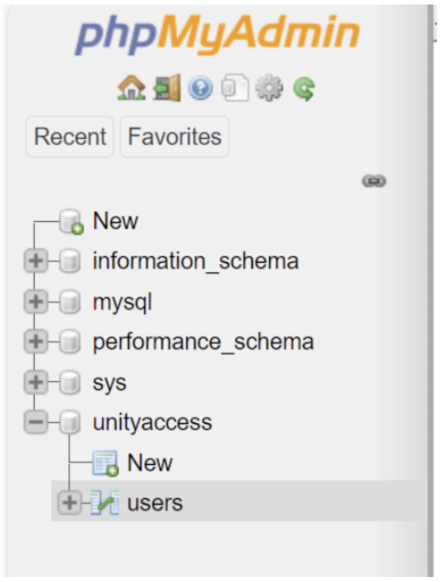
Criterion C: Development

Techniques used to develop solution

- SQL Database/PHP
    - Database connection
    - Database Register
    - Database Authentication
    - Database
- C#
    - Classes and Objects
    - Methods
    - Conditional Statements
    - Sorting
    - Exception handling
- Unity/Installed Libraries
    - Camera/QR Code Reader and Manager
    - Scene Manager
- Unity GUI Design
    - Unity Buttons
    - Unity Input boxes
    - Images

Technique 1: SQL Database/PHP

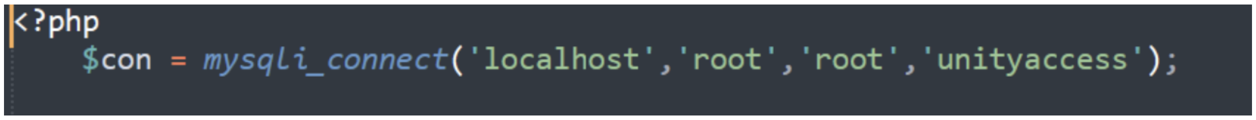




Above are the PHP files used to access the SQL data base (seen in the image on the right), these PHP files perform commands which build up the database used throughout the program.

Database connection

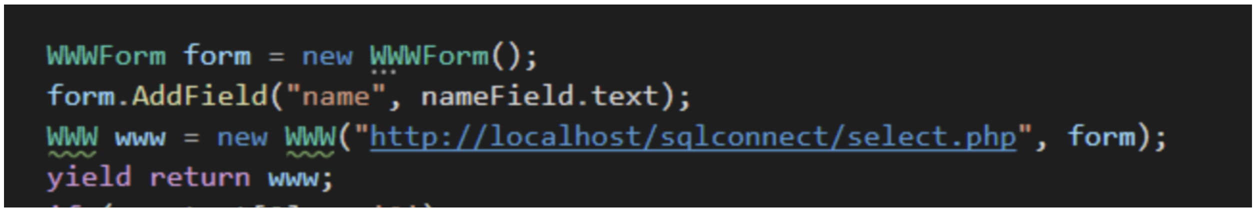
PHP Connection to Database



```
<?php
    $con = mysqli_connect('localhost','root','root','unityaccess');
```

The PHP file to access the database is programmed by using the function mysqli_connect(), connecting it to the variable $con. This mysqli_connect() method takes in four arguments, the location of a local host, the password root and root, along with the name of the primary database.

Unity Connection to PHP file

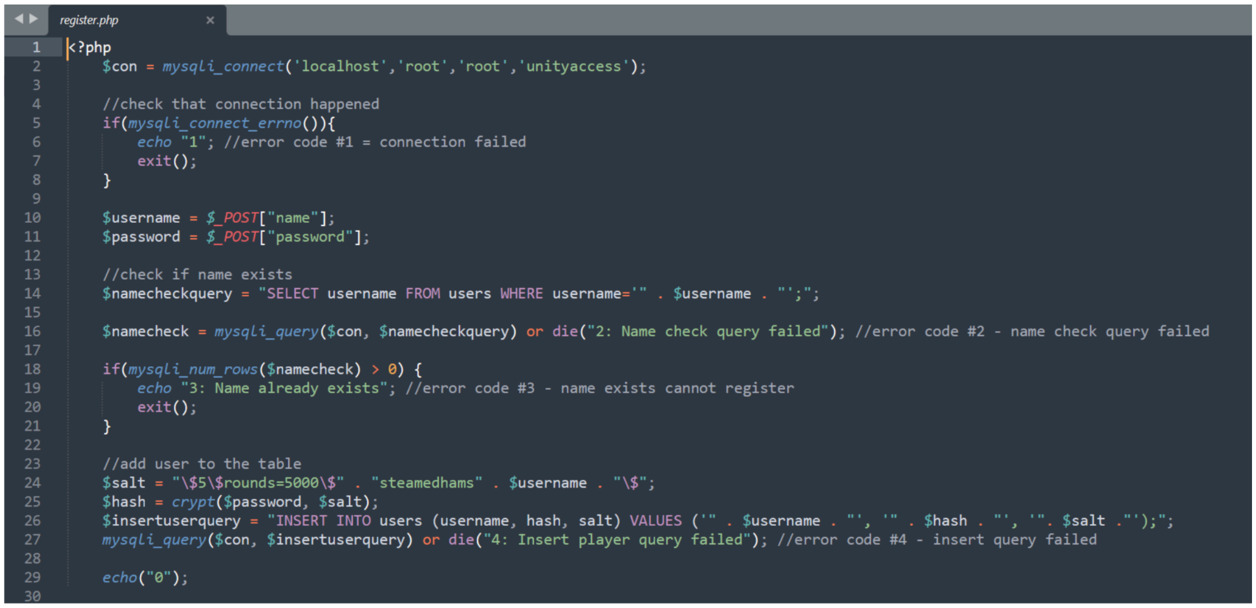


```
WWWForm form = new WWWForm();
form.AddField("name", nameField.text);
WWW www = new WWW("http://localhost/sqlconnect/select.php", form);
yield return www;
```

Now in the unity C# file we create an a form connection through utilizing the WWW object. By using the WWW object we take in the arguments like the file path taken as the first string, with the localhost of the database and then the PHP file path to the specific PHP file commands.

Database Register

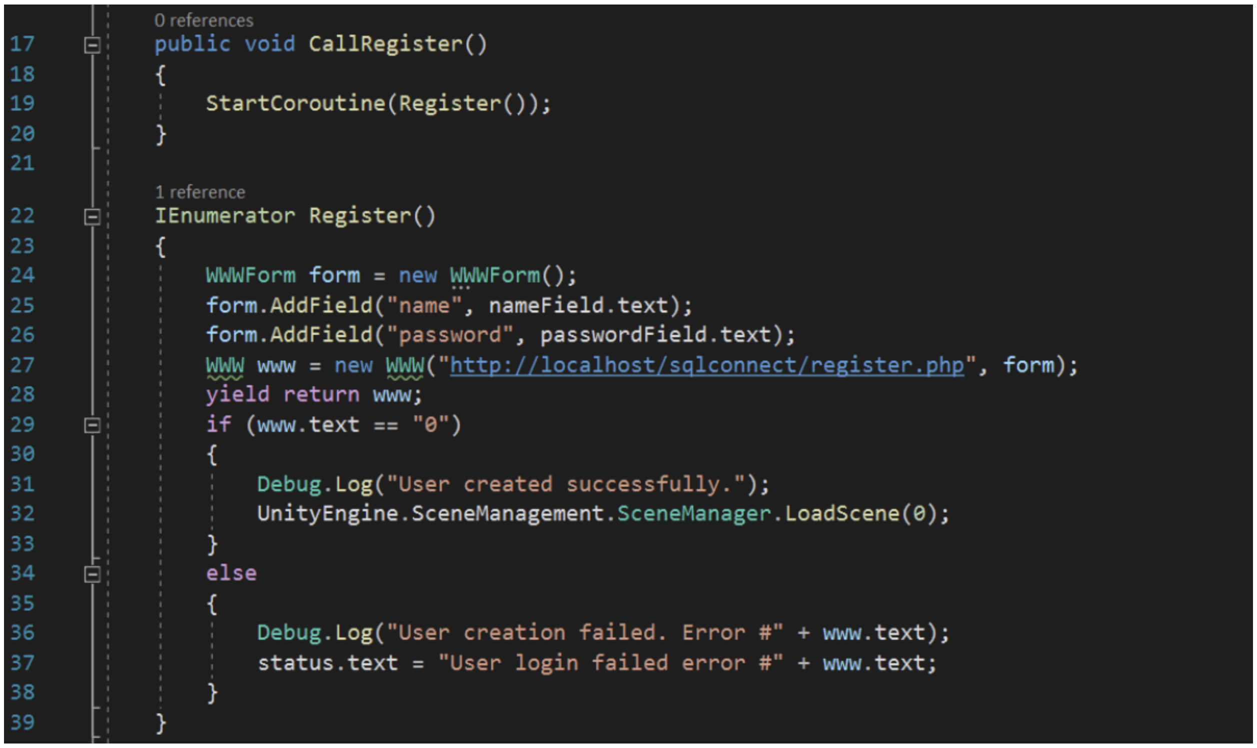
PHP Program



```php
<?php
    $con = mysqli_connect('localhost','root','root','unityaccess');

    //check that connection happened
    if(mysqli_connect_errno()){
        echo "1"; //error code #1 = connection failed
        exit();
    }

    $username = $_POST["name"];
    $password = $_POST["password"];

    //check if name exists
    $namecheckquery = "SELECT username FROM users WHERE username='" . $username . "';";

    $namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed"); //error code #2 - name check query failed

    if(mysqli_num_rows($namecheck) > 0) {
        echo "3: Name already exists"; //error code #3 - name exists cannot register
        exit();
    }

    //add user to the table
    $salt = "\$5\$rounds=5000\$" . "steamedhams" . $username . "\$";
    $hash = crypt($password, $salt);
    $insertuserquery = "INSERT INTO users (username, hash, salt) VALUES ('" . $username . "', '" . $hash . "', '". $salt ."');";
    mysqli_query($con, $insertuserquery) or die("4: Insert player query failed"); //error code #4 - insert query failed

    echo("0");

```

For the registration now with the database connected we create a variable $username and $password we equate these to the variables named “name” and “password” on the unity file using the $_POST method. From these newly created variables, we can encrypt the $password variable by creating a $salt and $hash. From that, we can create a query named $insertuserquery which inserts values in the user database username, hash, and salt with the new $username and encrypted password. With this query we can use the method mysqli_query() to take in the key (which has the path to the database) and the action to register the new username and password.

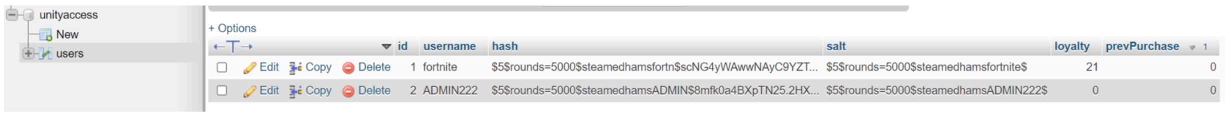
C# script



```
0 references
public void CallRegister()
{
    StartCoroutine(Register());
}

1 reference
IEnumerator Register()
{
    WWWForm form = new WWWForm();
    form.AddField("name", nameField.text);
    form.AddField("password", passwordField.text);
    WWW www = new WWW("http://localhost/sqlconnect/register.php", form);
    yield return www;
    if (www.text == "0")
    {
        Debug.Log("User created successfully.");
        UnityEngine.SceneManagement.SceneManager.LoadScene(0);
    }
    else
    {
        Debug.Log("User creation failed. Error #" + www.text);
        status.text = "User login failed error #" + www.text;
    }
}
```

Using the WWW object to get access to the PHP file, we use the form.AddField() method to create a form which are the username and password arguments. This method takes in the information taken from the username and password fields, while also creating a defined name for these variables which the PHP file searches for.

Result on SQL site

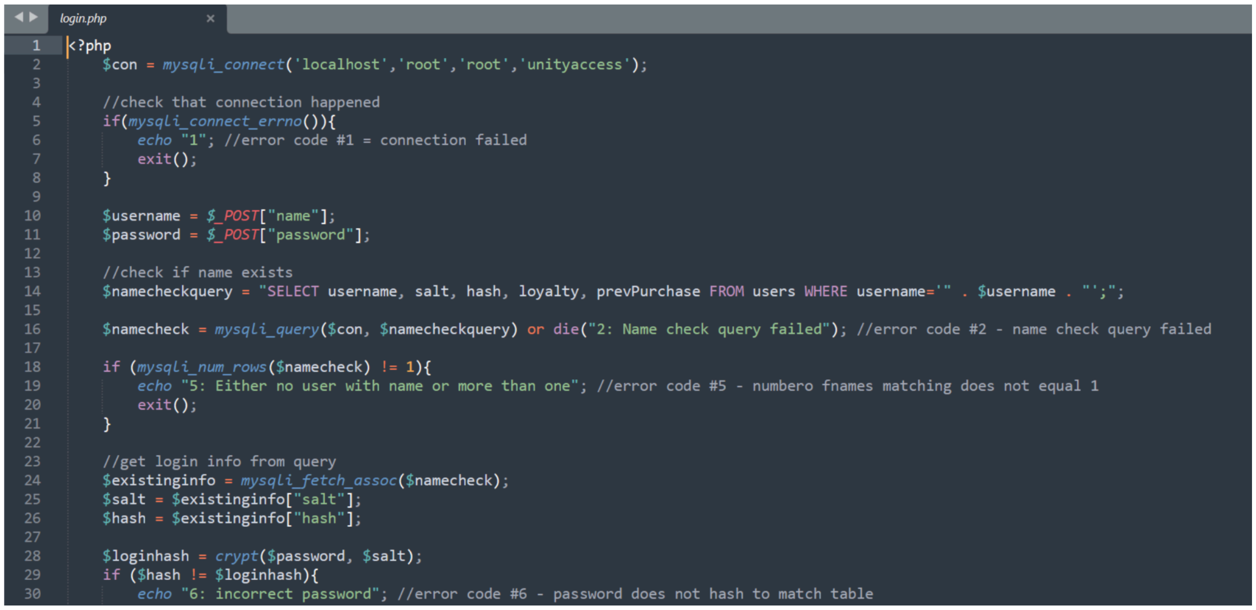


	id	username	hash	salt	loyalty	prevPurchase
Edit Copy Delete	1	fortnite	5rounds=5000$steamedhamsfortn$scNG4yWAwwNAyC9YZT...	5rounds=5000$steamedhamsfortnite$	21	0
Edit Copy Delete	2	ADMIN222	5rounds=5000$steamedhamsADMIN$8mfk0a4BXpTN25.2HX...	5rounds=5000$steamedhamsADMIN222$	0	0

The SQL site ends up looking like this when a new user is created.

Database Authentication

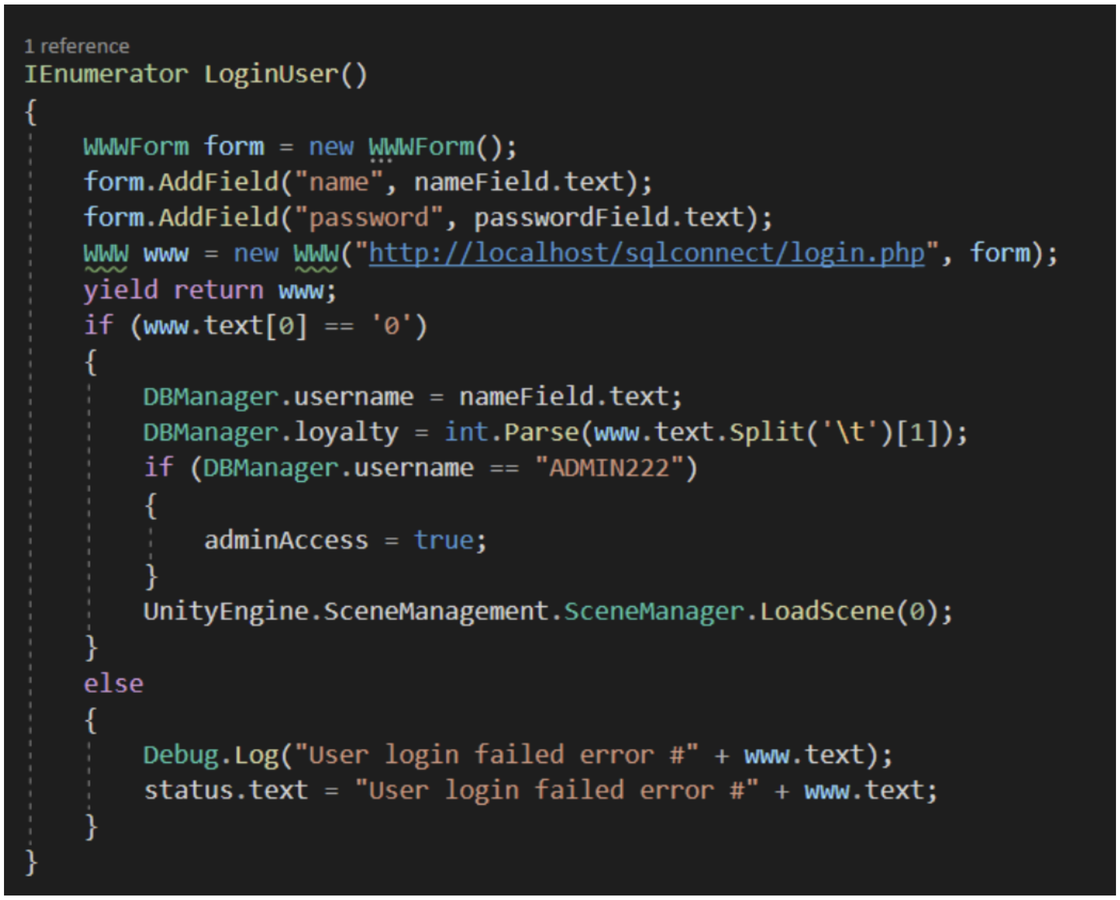
PHP Program



```php
<?php
    $con = mysqli_connect('localhost','root','root','unityaccess');

    //check that connection happened
    if(mysqli_connect_errno()){
        echo "1"; //error code #1 = connection failed
        exit();
    }

    $username = $_POST["name"];
    $password = $_POST["password"];

    //check if name exists
    $namecheckquery = "SELECT username, salt, hash, loyalty, prevPurchase FROM users WHERE username='" . $username . "';";

    $namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed"); //error code #2 - name check query failed

    if (mysqli_num_rows($namecheck) != 1){
        echo "5: Either no user with name or more than one"; //error code #5 - numbero fnames matching does not equal 1
        exit();
    }

    //get login info from query
    $existinginfo = mysqli_fetch_assoc($namecheck);
    $salt = $existinginfo["salt"];
    $hash = $existinginfo["hash"];

    $loginhash = crypt($password, $salt);
    if ($hash != $loginhash){
        echo "6: incorrect password"; //error code #6 - password does not hash to match table
```

The login system for the program’s PHP file is similar to the registration, we can get the information from the input username getting the array of information in the form of $existinginfo using the mysqli_fetch_assoc() method which takes in a verified username. The verified username’s encrypted password, with the encrypted password imputed, if all the information matches up, the user will be verified to log in.

C# script

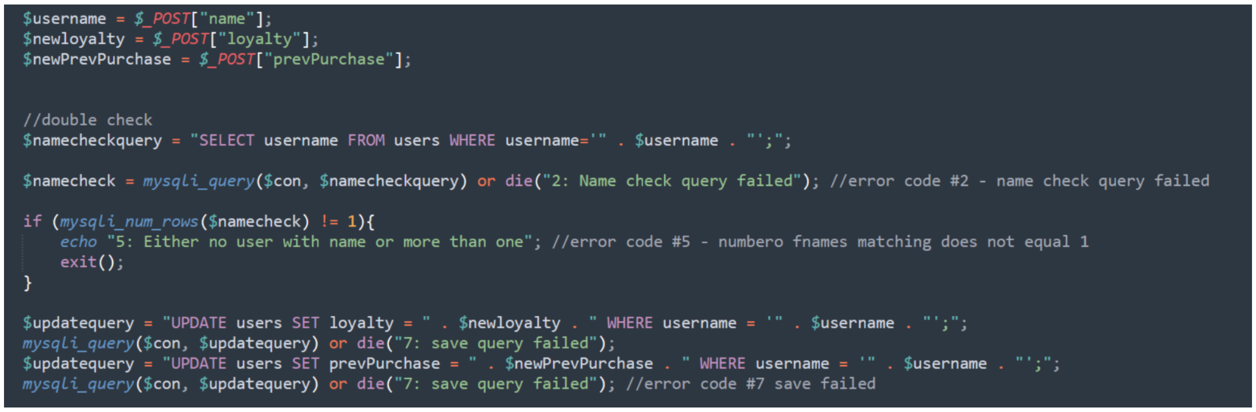


```
1 reference
IEnumerator LoginUser()
{
    WWWForm form = new WWWForm();
    form.AddField("name", nameField.text);
    form.AddField("password", passwordField.text);
    WWW www = new WWW("http://localhost/sqlconnect/login.php", form);
    yield return www;
    if (www.text[0] == '0')
    {
        DBManager.username = nameField.text;
        DBManager.loyalty = int.Parse(www.text.Split('\t')[1]);
        if (DBManager.username == "ADMIN222")
        {
            adminAccess = true;
        }
        UnityEngine.SceneManagement.SceneManager.LoadScene(0);
    }
    else
    {
        Debug.Log("User login failed error #" + www.text);
        status.text = "User login failed error #" + www.text;
    }
}
```

The same logic for the register is used for the LoginUser() method for the C# for the most part, however, now the program takes an if else statement, if verified, it will check whether or not the username imputed is the admin key, if the key is imputed the boolean adminAccess is true, however, either way the scene will go back to the main menu. The C# program will also contain the information obtained from the PHP file through the DBManager public variables.

Database (changing data)

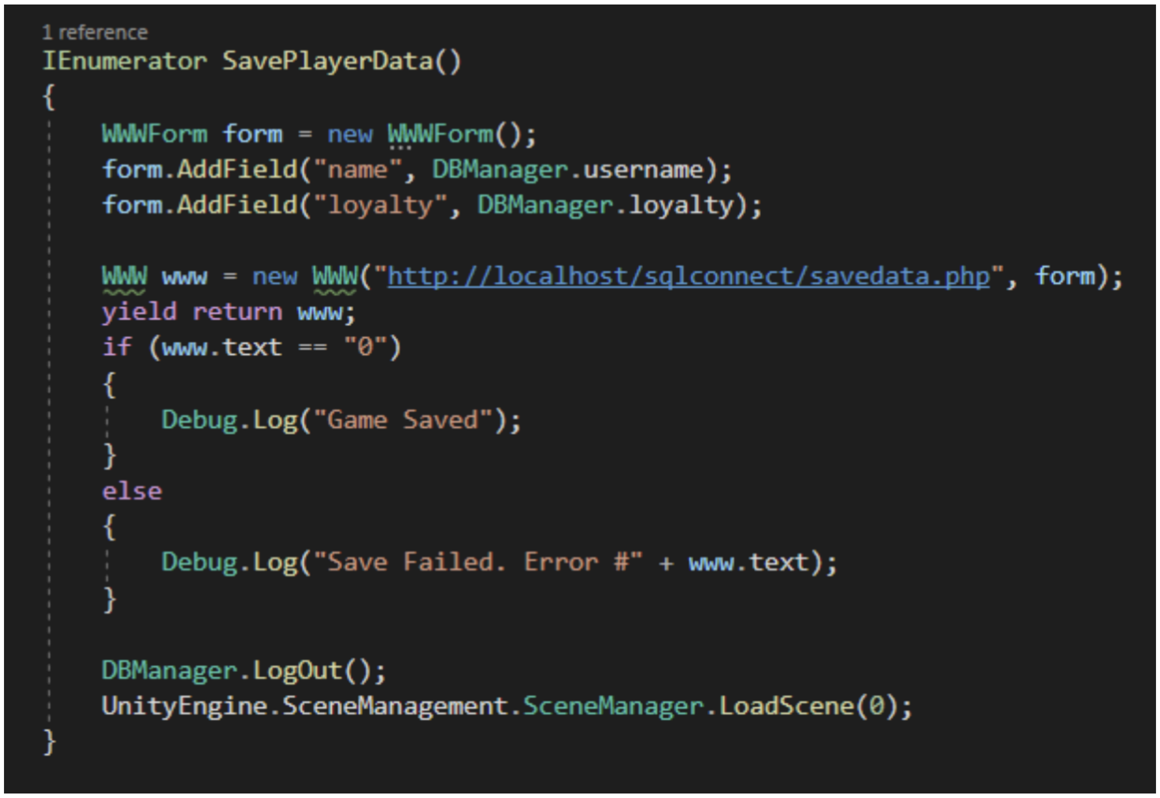
PHP Program



```
$username = $_POST["name"];
$newloyalty = $_POST["loyalty"];
$newPrevPurchase = $_POST["prevPurchase"];


//double check
$namecheckquery = "SELECT username FROM users WHERE username='" . $username . "';";

$namecheck = mysqli_query($con, $namecheckquery) or die("2: Name check query failed"); //error code #2 - name check query failed

if (mysqli_num_rows($namecheck) != 1){
    echo "5: Either no user with name or more than one"; //error code #5 - numbero fnames matching does not equal 1
    exit();
}

$updatequery = "UPDATE users SET loyalty = " . $newloyalty . " WHERE username = '" . $username . "';";
mysqli_query($con, $updatequery) or die("7: save query failed");
$updatequery = "UPDATE users SET prevPurchase = " . $newPrevPurchase . " WHERE username = '" . $username . "';";
mysqli_query($con, $updatequery) or die("7: save query failed"); //error code #7 save failed
```

The PHP program to save information will use the same starting systems, however, will use new variables for things that are changed like $newloyalty and $newPrevPurchase. A new command variable of $updatequery is used to update and equate specific sections of a database based on the $username like loyalty. The query is then changed using the previous mysqli_query with the action argument being the $updatequery.

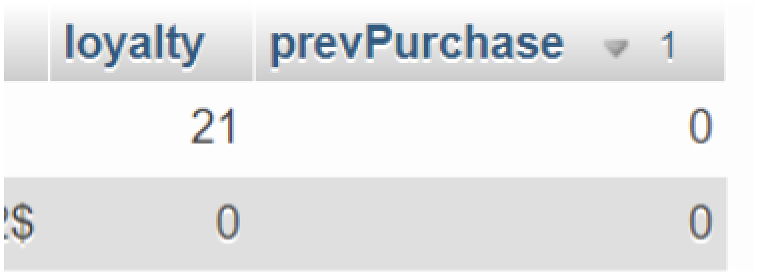
C# script



```
1 reference
IEnumerator SavePlayerData()
{
    WWWForm form = new WWWForm();
    form.AddField("name", DBManager.username);
    form.AddField("loyalty", DBManager.loyalty);

    WWW www = new WWW("http://localhost/sqlconnect/savedata.php", form);
    yield return www;
    if (www.text == "0")
    {
        Debug.Log("Game Saved");
    }
    else
    {
        Debug.Log("Save Failed. Error #" + www.text);
    }

    DBManager.LogOut();
    UnityEngine.SceneManagement.SceneManager.LoadScene(0);
}
```

This method is pretty much the same as previous, with the imputed data that we want saved being used as the form with us wanting to update the loyalty information with the variable DBManager.loyalty.

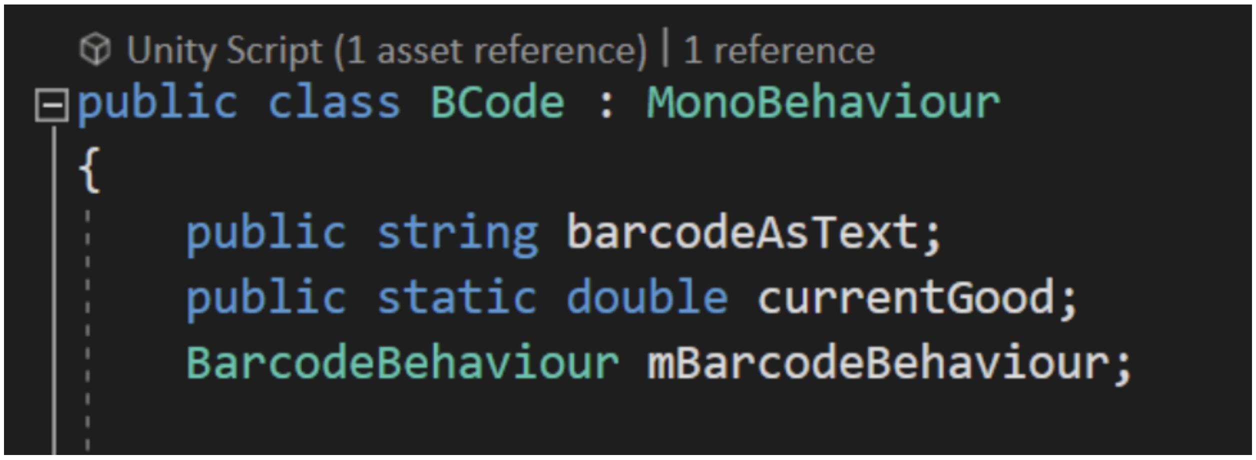
Result on SQL site



	loyalty	prevPurchase ▼ 1
	21	0
:$	0	0

The changes in data can be seen in the difference above.

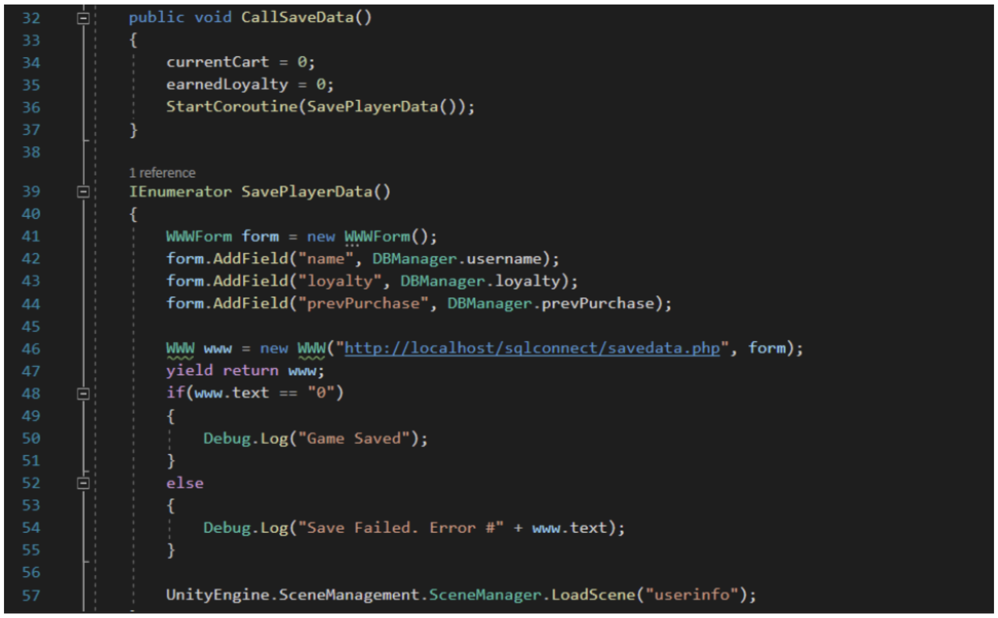
Technique 2: C#
Classes and Objects



```
Unity Script (1 asset reference) | 1 reference
public class BCode : MonoBehaviour
{
    public string barcodeAsText;
    public static double currentGood;
    BarcodeBehaviour mBarcodeBehaviour;
```

Objects and classes in Unity enable modular design, encapsulation, and code organization, enhancing maintainability and scalability for complex applications. This allows me in this case to deal with barcode data and text in singular parts and reference them in applications as large conglomerations.

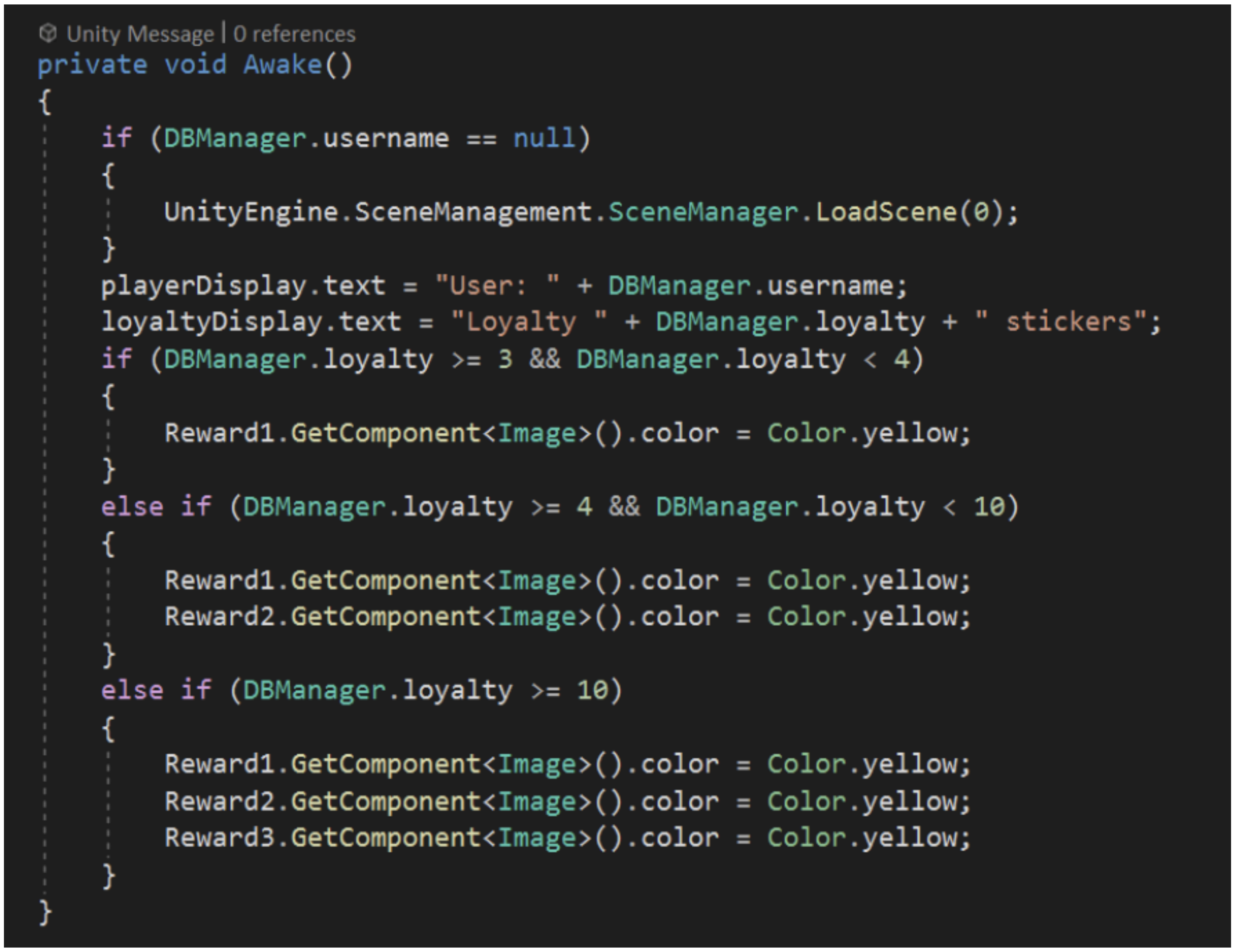
Methods



```
public void CallSaveData()
{
    currentCart = 0;
    earnedLoyalty = 0;
    StartCoroutine(SavePlayerData());
}

1 reference
IEnumerator SavePlayerData()
{
    WWWForm form = new WWWForm();
    form.AddField("name", DBManager.username);
    form.AddField("loyalty", DBManager.loyalty);
    form.AddField("prevPurchase", DBManager.prevPurchase);

    WWW www = new WWW("http://localhost/sqlconnect/savedata.php", form);
    yield return www;
    if(www.text == "0")
    {
        Debug.Log("Game Saved");
    }
    else
    {
        Debug.Log("Save Failed. Error #" + www.text);
    }

    UnityEngine.SceneManagement.SceneManager.LoadScene("userinfo");
```

In the same sense, through using methods we simplify the readability of the code, only changing the general variable the methods run through rather than changing every appearance of the variable in the entire method itself.

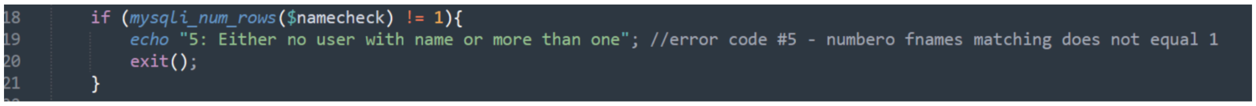
Conditional Statements



```
Unity Message | 0 references
private void Awake()
{
    if (DBManager.username == null)
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene(0);
    }
    playerDisplay.text = "User: " + DBManager.username;
    loyaltyDisplay.text = "Loyalty " + DBManager.loyalty + " stickers";
    if (DBManager.loyalty >= 3 && DBManager.loyalty < 4)
    {
        Reward1.GetComponent<Image>().color = Color.yellow;
    }
    else if (DBManager.loyalty >= 4 && DBManager.loyalty < 10)
    {
        Reward1.GetComponent<Image>().color = Color.yellow;
        Reward2.GetComponent<Image>().color = Color.yellow;
    }
    else if (DBManager.loyalty >= 10)
    {
        Reward1.GetComponent<Image>().color = Color.yellow;
        Reward2.GetComponent<Image>().color = Color.yellow;
        Reward3.GetComponent<Image>().color = Color.yellow;
    }
}
```

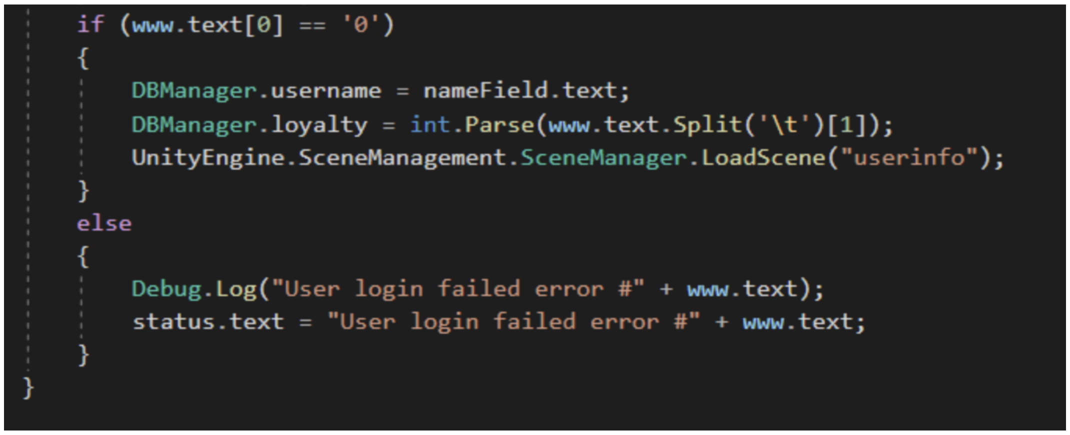
This section of code demonstrates how the reward images will switch from normal to the color yellow depending on the amount of loyalty points someone earns.

Exception handling



```
    if (mysqli_num_rows($namecheck) != 1){
        echo "5: Either no user with name or more than one"; //error code #5 - numbero fnames matching does not equal 1
        exit();
    }
```

Here is a singular example of the eror handling in which the username imputed does not return with a result. There are many other error cases, however, it will output a none “0” result which is read as failure by the C# program.

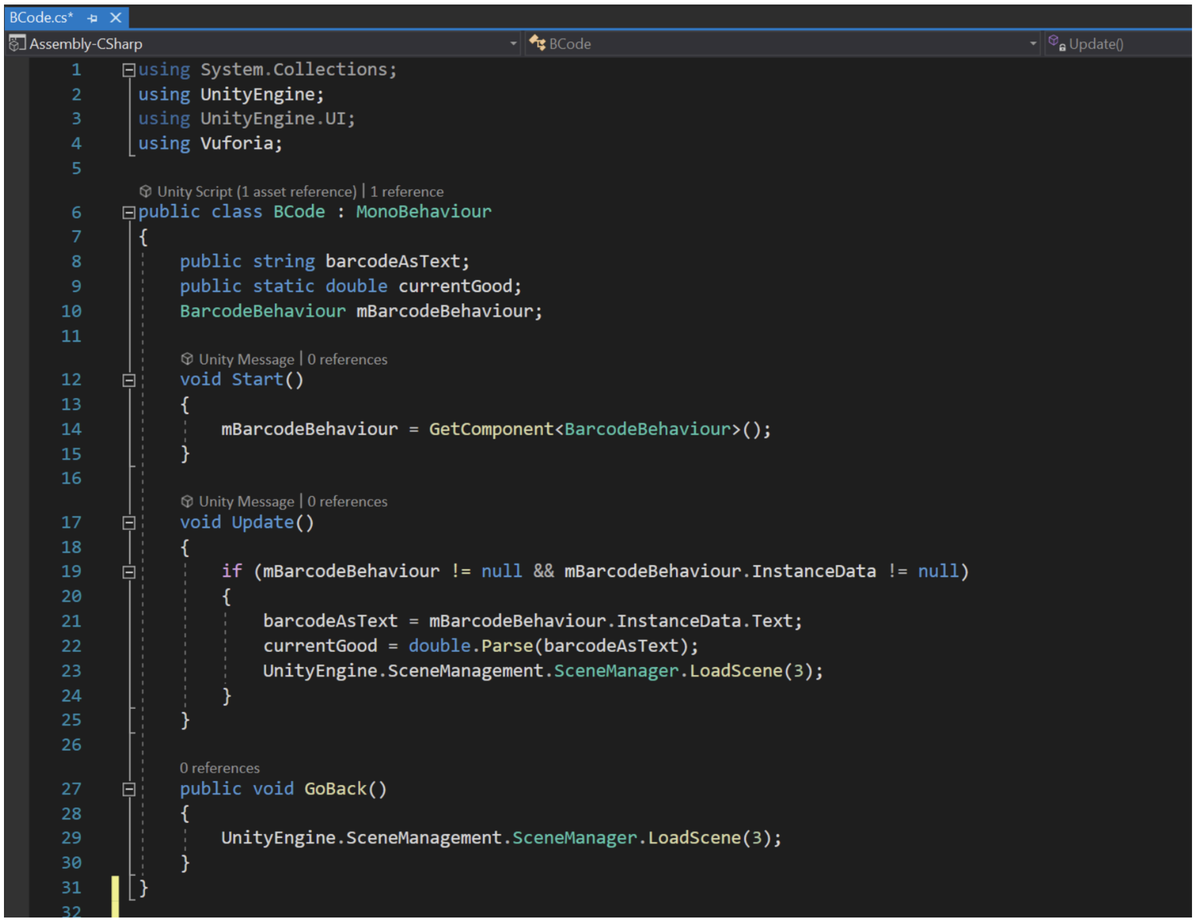


```
    if (www.text[0] == '0')
    {
        DBManager.username = nameField.text;
        DBManager.loyalty = int.Parse(www.text.Split('\t')[1]);
        UnityEngine.SceneManagement.SceneManager.LoadScene("userinfo");
    }
    else
    {
        Debug.Log("User login failed error #" + www.text);
        status.text = "User login failed error #" + www.text;
    }
}
```

As the output of the PHP file is read as not “0”, there will be a an error which prints out the issue the PHP file outputs.

Technique 3: Unity/Installed Libraries

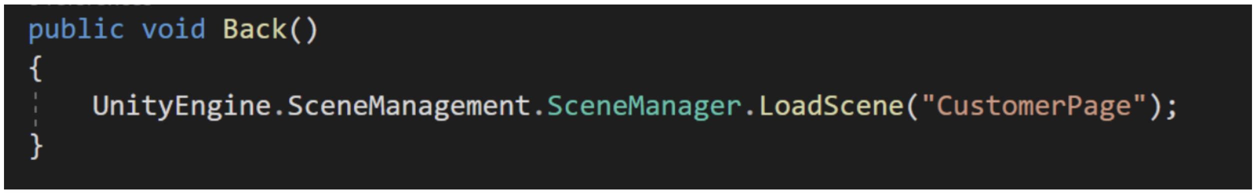
Camera/QR Code Reader and Manager (Vuforia)

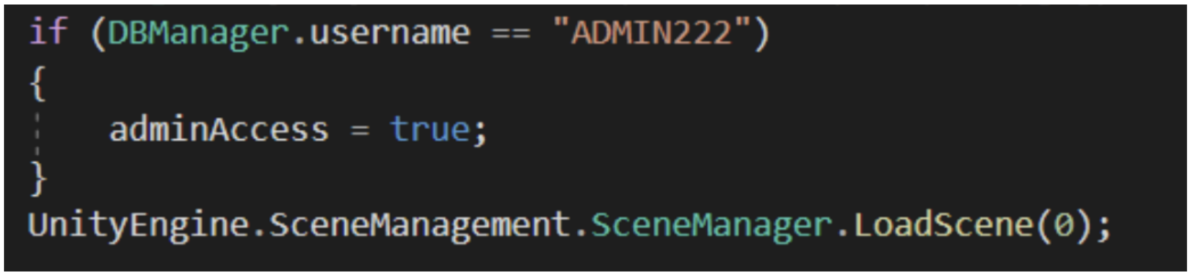


```
using System.Collections;
using UnityEngine;
using UnityEngine.UI;
using Vuforia;

Unity Script (1 asset reference) | 1 reference
public class BCode : MonoBehaviour
{
    public string barcodeAsText;
    public static double currentGood;
    BarcodeBehaviour mBarcodeBehaviour;

    Unity Message | 0 references
    void Start()
    {
        mBarcodeBehaviour = GetComponent<BarcodeBehaviour>();
    }

    Unity Message | 0 references
    void Update()
    {
        if (mBarcodeBehaviour != null && mBarcodeBehaviour.InstanceData != null)
        {
            barcodeAsText = mBarcodeBehaviour.InstanceData.Text;
            currentGood = double.Parse(barcodeAsText);
            UnityEngine.SceneManagement.SceneManager.LoadScene(3);
        }
    }

    0 references
    public void GoBack()
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene(3);
    }
}
```

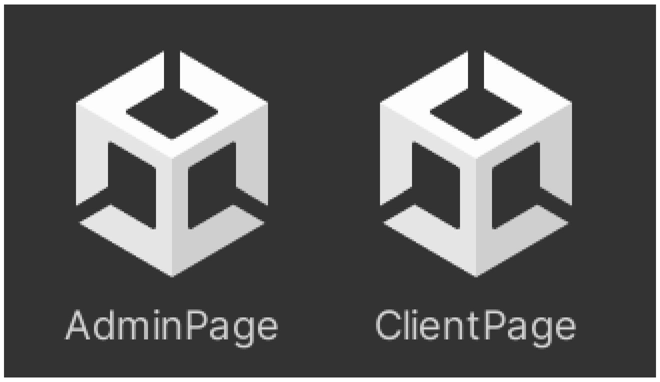
Through using the Vuforia import library (QR/Barcode reading software), using the built in AR camera we are able to open a camera through the usage of BarcodeBehavior object and read the information through the instance data from the read object built into the library.

Scene Manager


```
public void Back()
{
    UnityEngine.SceneManagement.SceneManager.LoadScene("CustomerPage");
}
```

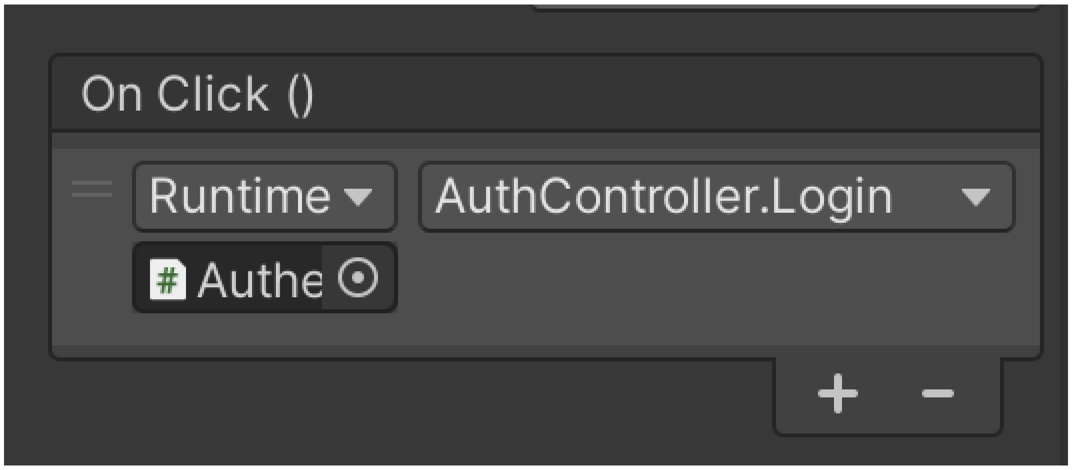

```
if (DBManager.username == "ADMIN222")
{
    adminAccess = true;
}
UnityEngine.SceneManagement.SceneManager.LoadScene(0);
```

By using the built-in scene manager offered by Unity, we can simply swap to different windows. By using Back() the method will search for the exact name or index of a window and swap to it. Different scenes can be seen below.

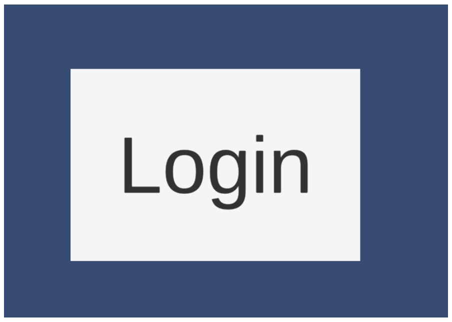


Technique 4: Unity GUI Design

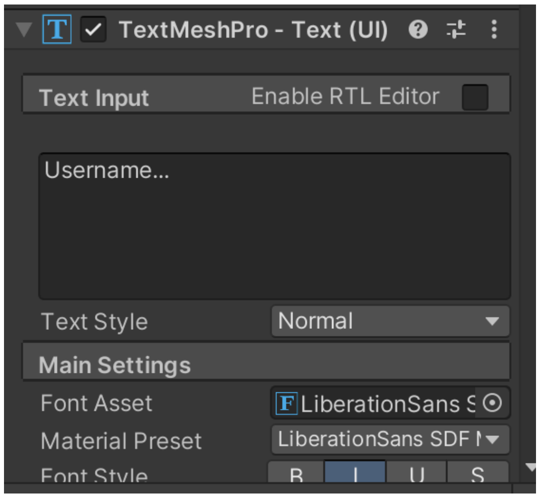
Unity Buttons



In using the prebuilt GUI and UI of Unity we can utilize the simple designs of the buttons below, and also simply call upon any method by using the Unity UI to associate scripts to the button.



Unity Input boxes



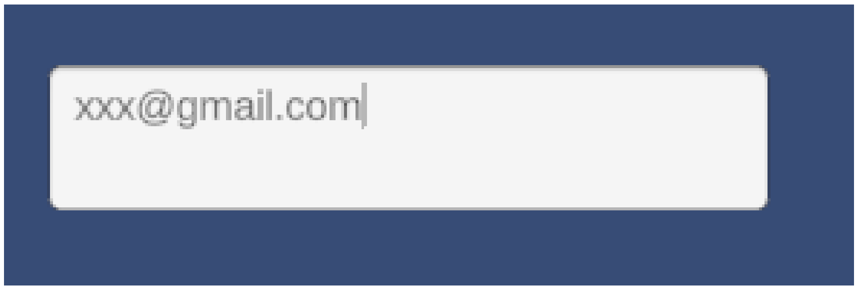
Username...

Text Style: Normal

Font Asset: LiberationSans S

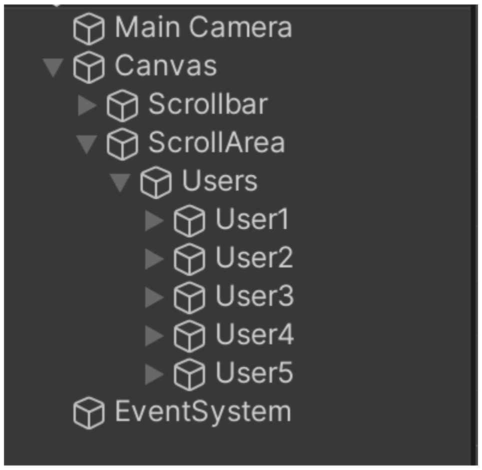
Material Preset: LiberationSans SDF M

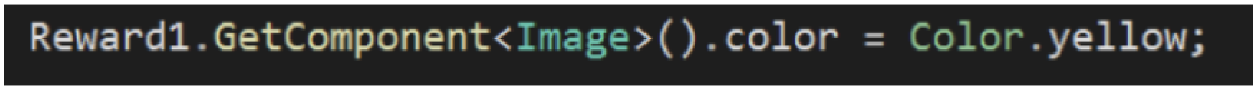
Input Boxes allow us to replace the text while in the application to the desired information we want to input. Associating input to a variable using a text object.



Images

Through using the image object to render, we are able to control different aspects of the image based on simple commands.





```
Reward1.GetComponent<Image>().color = Color.yellow;
```

Libraries
Vuforia
https://developer.vuforia.com/
PHP my admin
https://www.phpmyadmin.net/
Unity
https://unity.co**Criterion_C_Development**m/