

metapro Package documentation

Version	Author	Date
0.1.0	Andrzej "RastaGrzywa"	06.01.2023

About metapro

What's new

Guides

Installation

Requirements

Package samples

Quick start

1. Editor

- Access tool
- Import app key
- Download resources
- App data object
- Change app key
- Use app data in script

2. Play mode

- Chose provider
- Connect wallet
- Login with Web3
- Display user data
- Show app tokens
- Use user and app data from script

Troubleshooting

Reference

MetaproAppSetup

Editor Window

PluginManager

Web3 connection providers

- metapro
- Wallet Connect
- MetaMask

WindowController

About metapro

Use the metapro package for an easy way to enter the web3 world containing integration with metapromarket and web3 users wallets.

Add support for developer:

- Integration with metapro marketplace
- Download assets stored behind NFT tokens
- Create game mechanics based on NFT ownership

Add support for gamers:

- Enable login with Web3 wallet
- Show player assets in game world
- Enable Web3 functionalities: send&sign transactions, manage tokens, check NFT ownership

Add Web3 integration to your projects using metapro package to take advantage of the security, decentralization, interoperability, and innovation offered by web3 and blockchain technology.

What's new in **0.1.0**

This section contains information about new features, improvements and fixed issues. For a complete list of changes made, refer to the [changelog](#).

This is the first release of the metapro package.

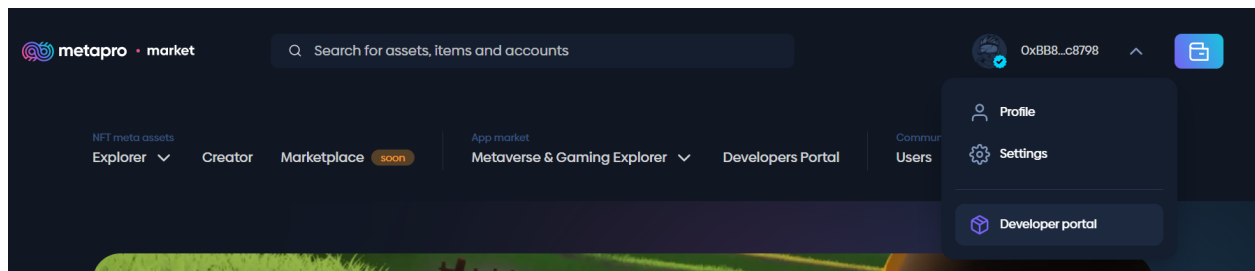
Contains:

- Setup development with app key
- Download resources connected to app from marketplace
- Player web3 wallet connection
- Web3 login
- Display and use player/app owned NFTs

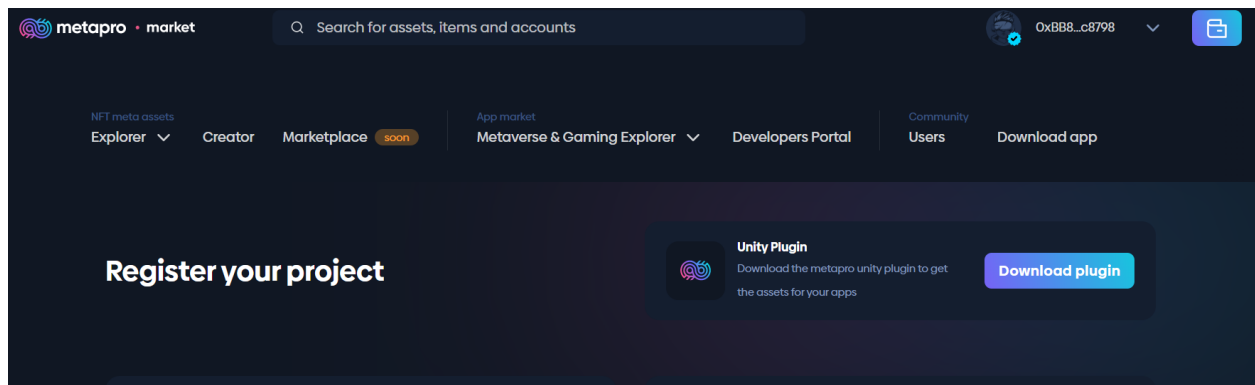
Installation guide

This package is currently visible only in the developer portal on metapromarket.com. To get the package go to metapromarket.com and follow guides to connect Your wallet and login with web3.

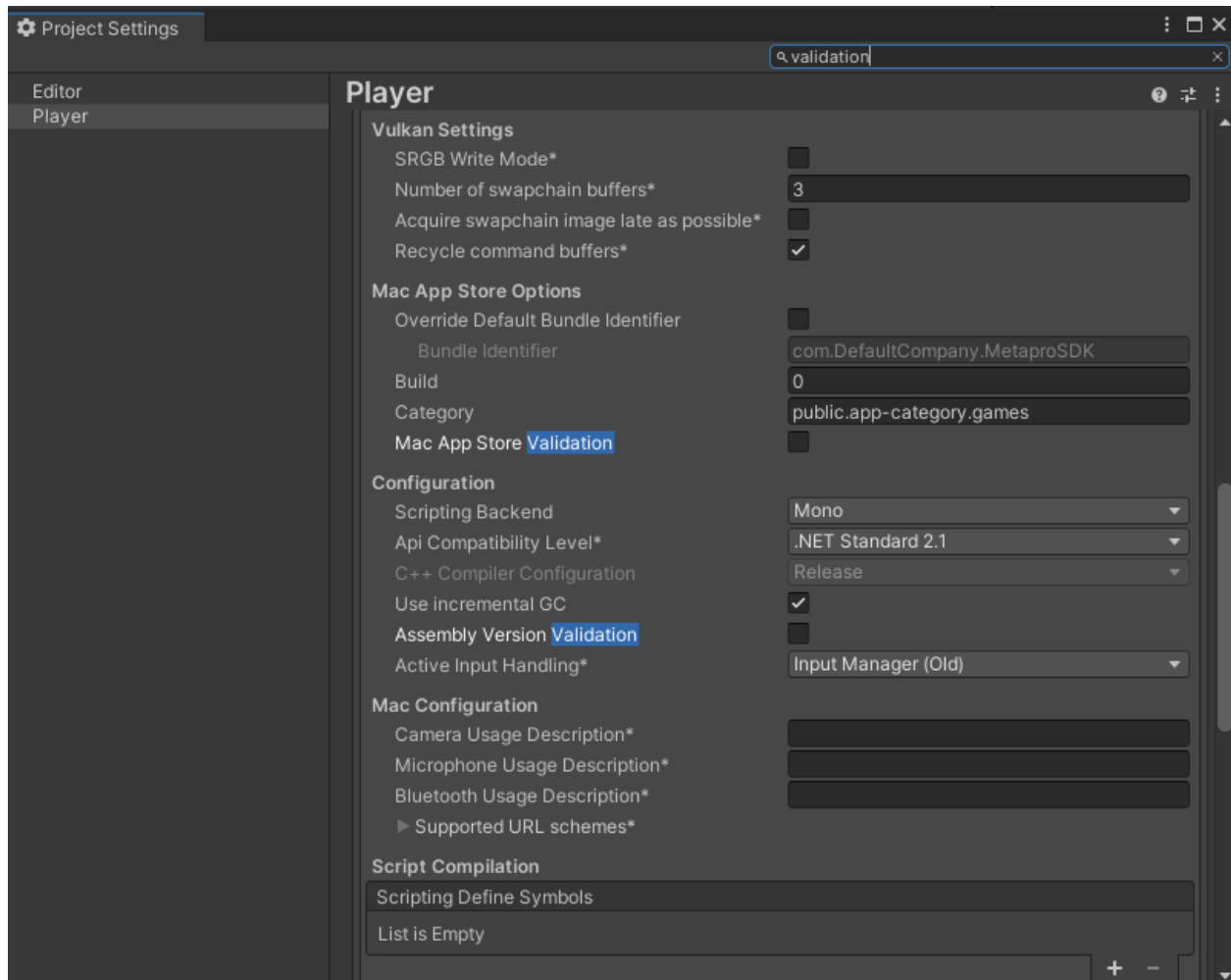
When Your account is logged You have access to developer portal. Click on dropdown with Your wallet address and click on **Developer portal** button.



To download Unity package click button **Download plugin** in Unity Plugin section.



WARNING: Before You can import metapro for Unity asset You need to disable **Assembly Version Validation** in Your project. You will find this under **Edit > Project Settings > Player > Other Settings > Configuration**.

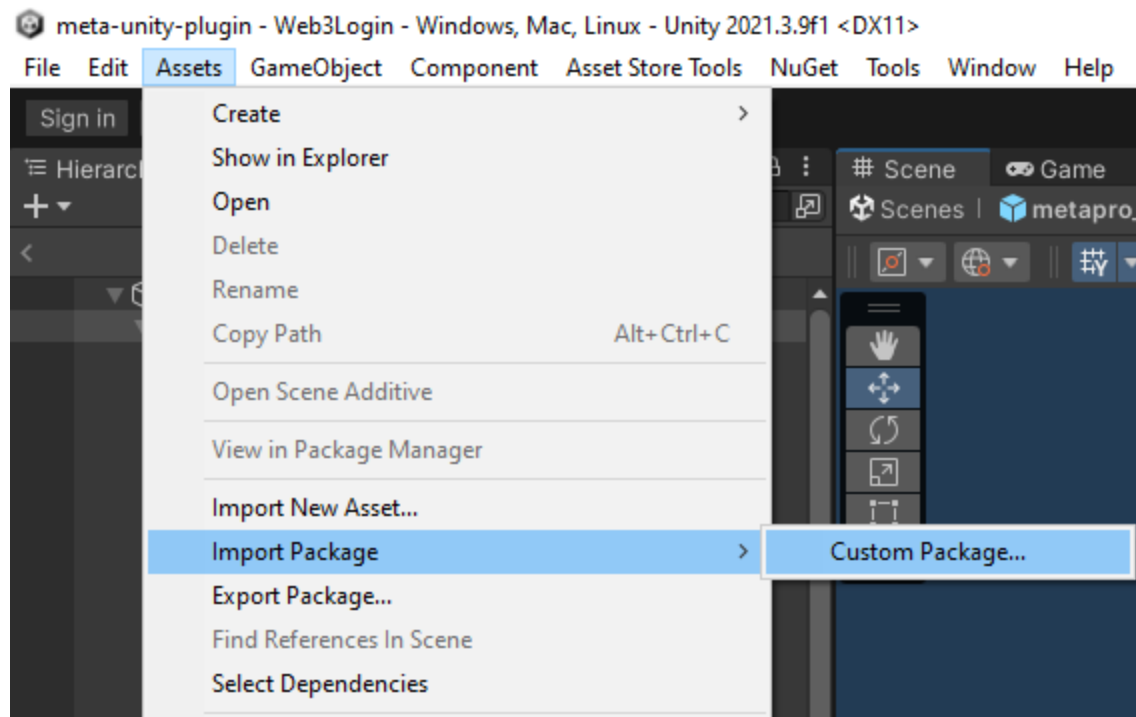


If You happen to import metapro for Unity package before disabling that option it'll be required to reimport assets. Go to **Assets > Reimport All** and in the popup window click the Reimport button.

To install asset You can follow official Unity guide:

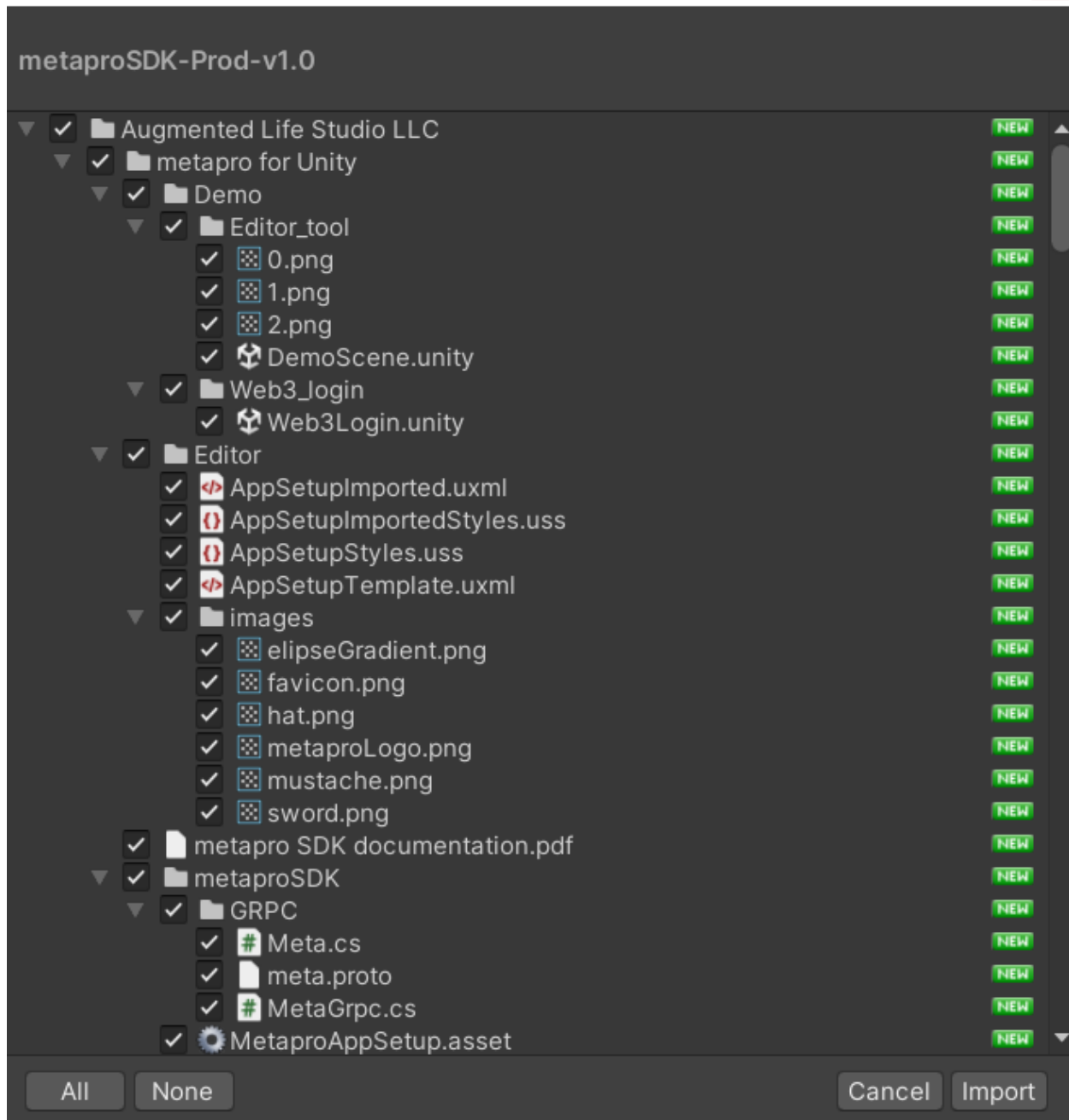
<https://docs.unity3d.com/Manual/AssetPackagesImport.html>

In the project You want to import an asset package choose: **Assets > Import Package > Custom Package**. In a file browser locate downloaded metapro asset .unitypackage file - select it and click **Open**.



The **Import Unity Package** window will appear. Make sure that every file has a checkmark and will be imported. Click the Import button and wait until all files are loaded and compiled.

Import Unity Package



To assure that the package is imported correctly check the existence of folders **Augumented Life Studio LLC** and **WC** under **Assets** folder in Your project. Second step to check is to look on the top bar of the Unity window and check if there is a tab called **Tools** that contains the Metapro **SDK Setup** button.

MetaproSDK - SampleScene - Windows, Mac, Linux - Unity 2021.3.6f1 Personal <DX11>

File Edit Assets GameObject Component Tools Window Help

Sign in



Metapro SDK Setup

Hierarchy



Scene



Game

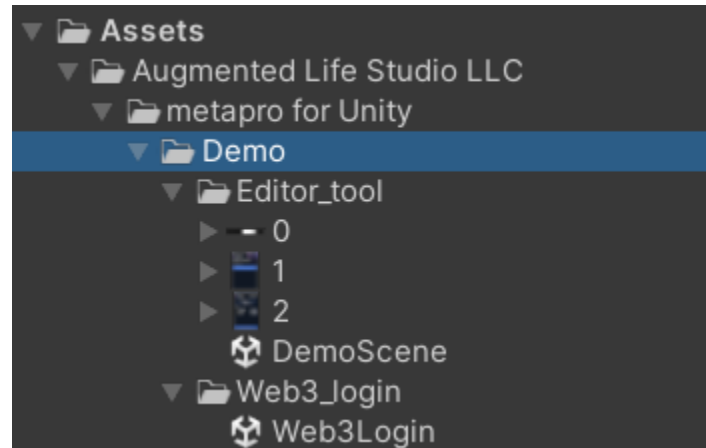
Requirements

Plugin was created and tested on **Unity 2021.3.9** and it's compatible with all further versions of Unity.

Package samples

The metapro package came with a simple example to help You get started.

To get sample scenes of use cases go to **Assets > Augmented Life Studio LLC > metapro for Unity > Demo**. You will find a scene that implements Web3 login and displays how to use the metapro editor plugin part.

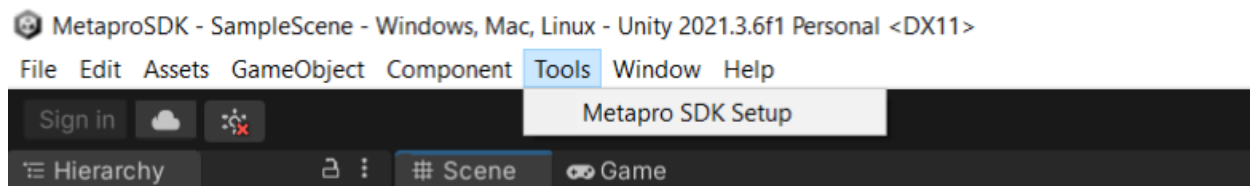


Quick start

1. Editor

- Access tool

To access the editor tool go into the top window bar and choose **Tools > Metapro SDK Setup**.

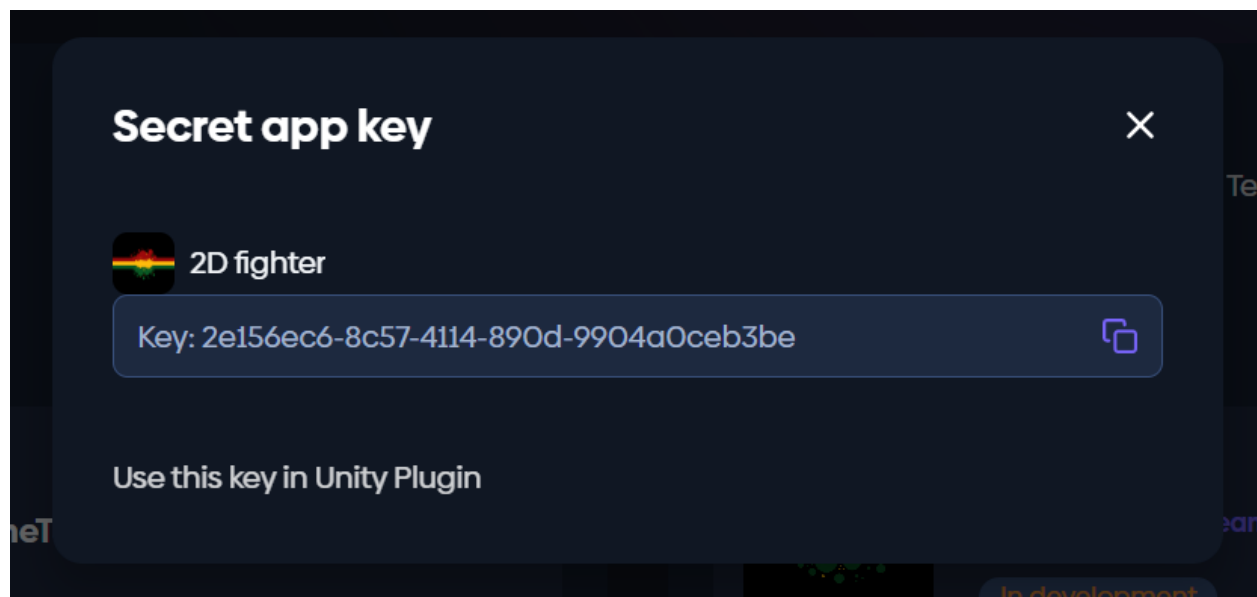
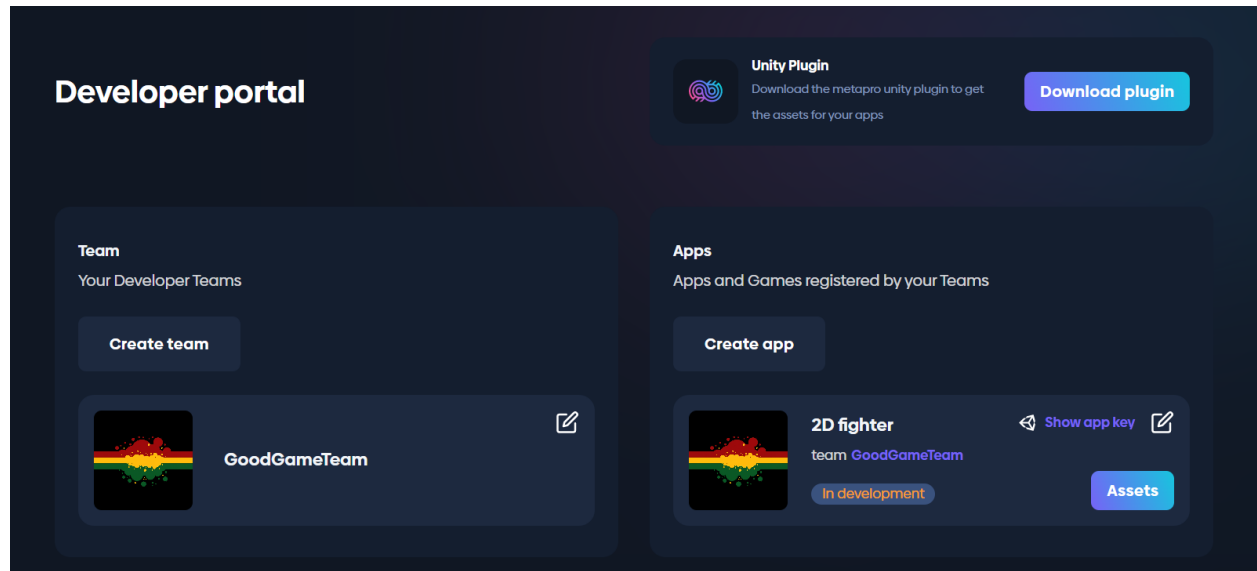


- Import app key

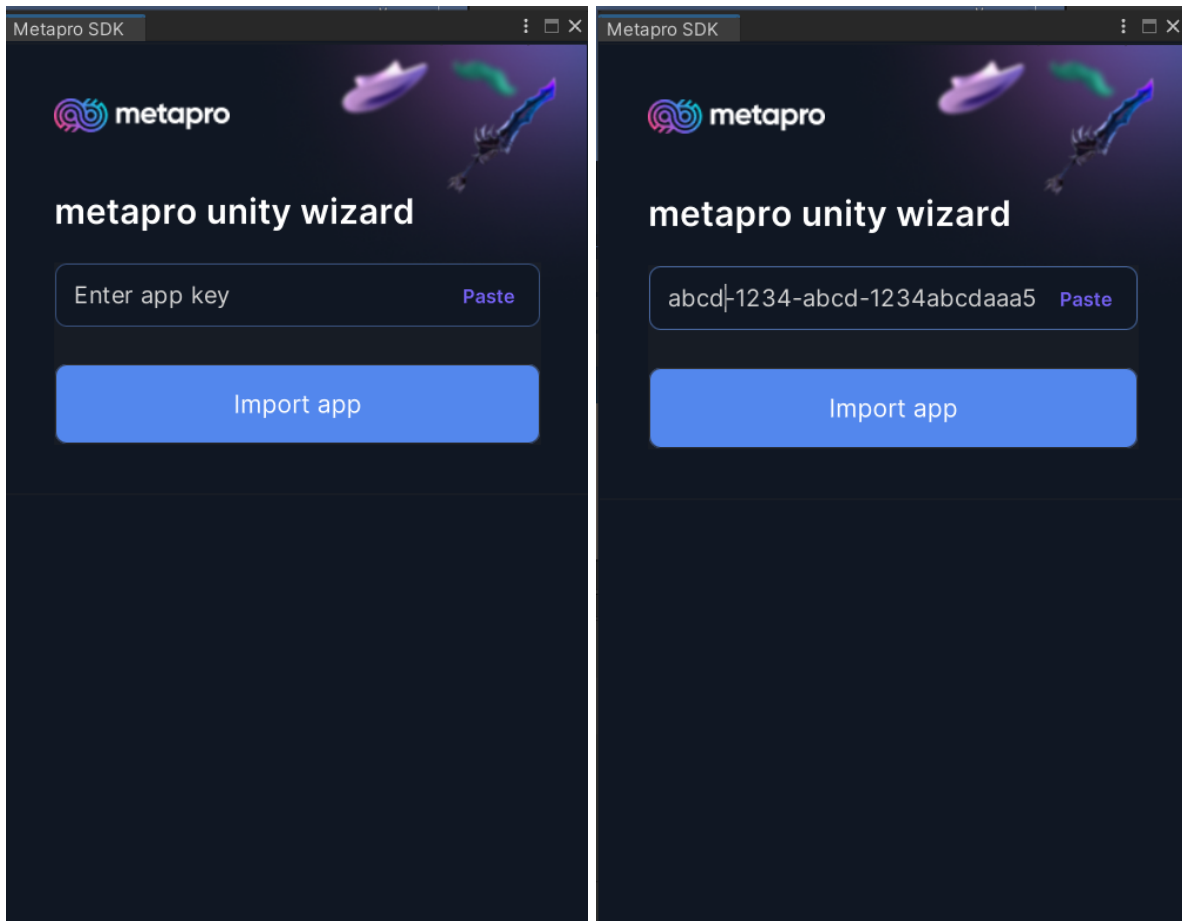
App key is necessary to be able to fully utilize the metapro tool. It will allow You to download assets and integrate Your project with the metapro marketplace.

To acquire an app key You need first have a team and application with assigned assets on metapromarket.com.

Go to the **Developer portal** and first **Create team**, second **Create app**. If You follow website guide You will be able to assign assets from market to Your application. When You assign any asset to an app button **Show app key** will show up on the game card. Click it and the popup will display secret key for an application.



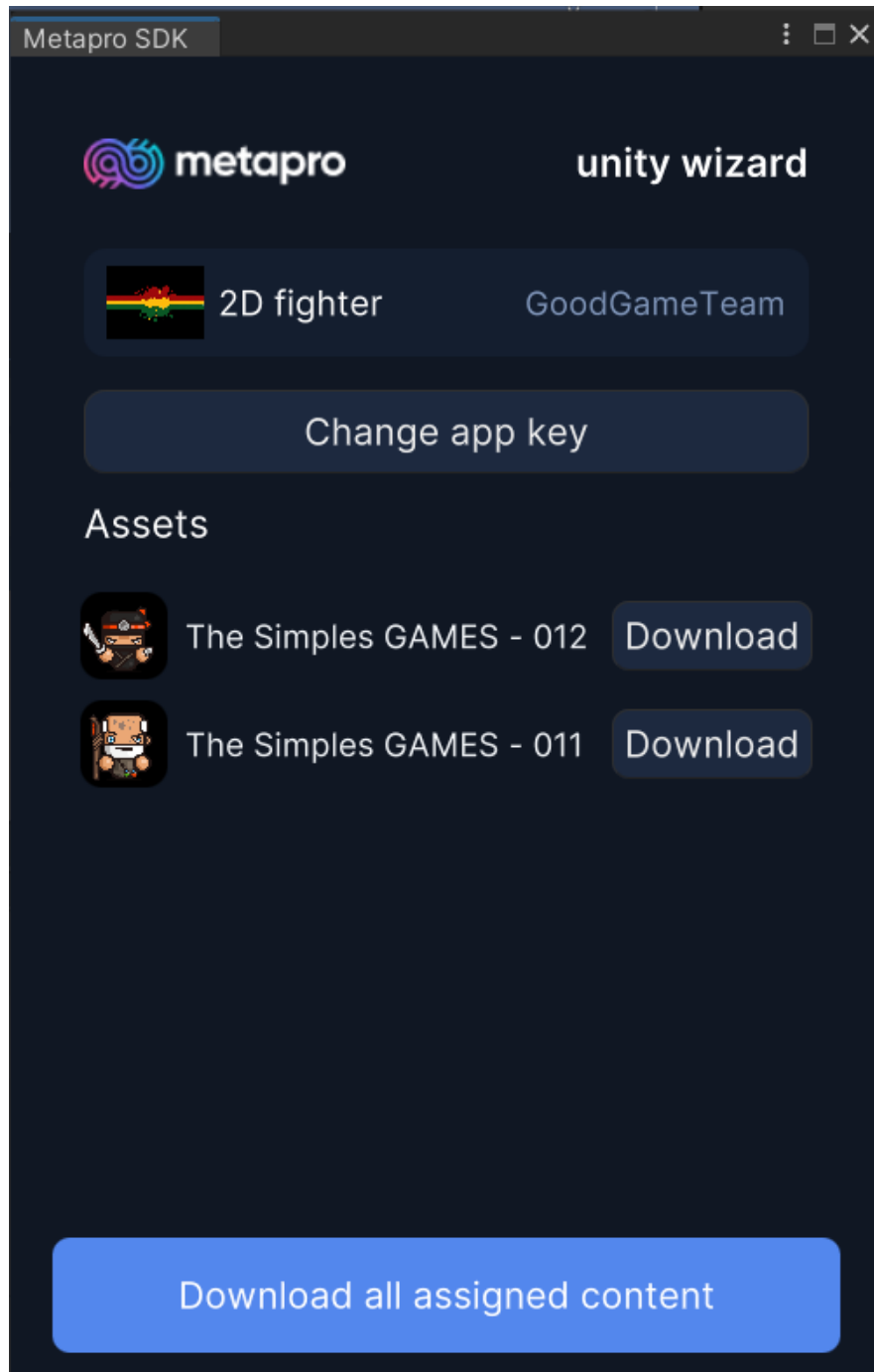
Copy the visible app key and paste it into the Metapro SDK window in Unity.



- Download resources

After the app key is imported, the Metapro SDK window will change and show all assigned NFTs to Your game on the metapro market.

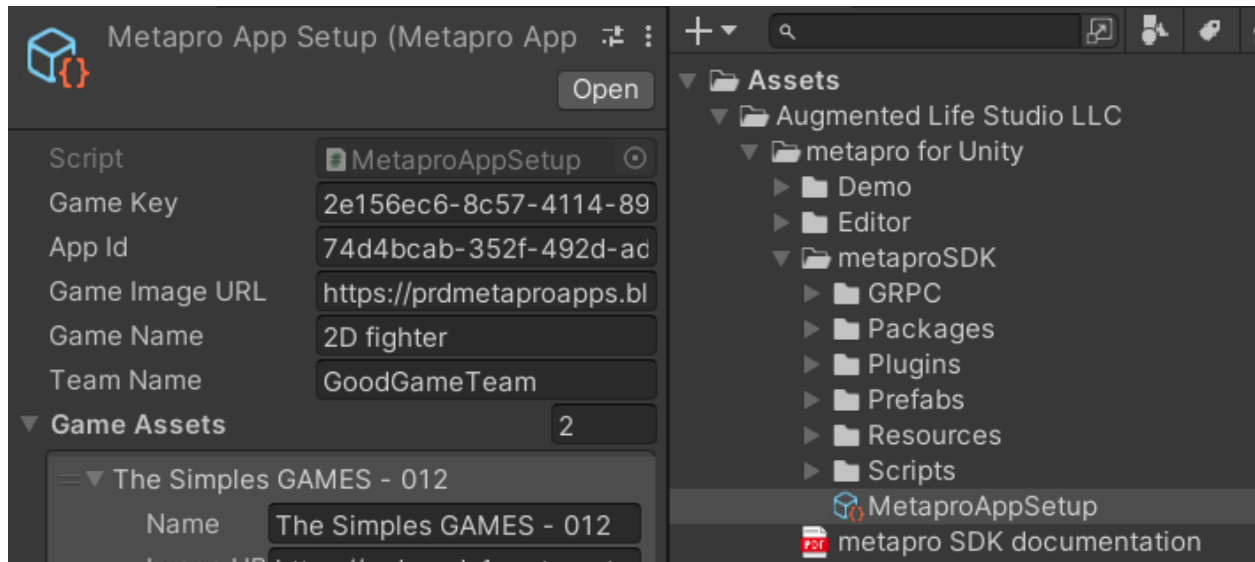
To download assets behind NFT click on **Download** button next to the token name. To download all assets click **Download all assigned content** at the bottom of the Metapro SDK window.



Every asset is downloaded to the **Assets > Resources > [token_id]** folder.

- App data object

Unity metapro SDK tool saves Your application data into Scriptable Object that is provided with package. You can find that object in **Assets > Augmented Life Studio LLC > metapro for Unity > metaproSDK > MetaproAppSetup**.



- Change app key

Using metapro for Unity You can change the app key for Your project. To do this open the Metapro SDK window and click **Change app key** button. It will show the initial SDK window and clear data from Metapro App Setup Scriptable Object.

- Use app data in script

To use app data in Your scripts just simply add field of type MetaproAppSetup and assign scriptable object in inspector.

```
[SerializeField] private MetaproAppSetup metaproAppSetup;
```

If You cannot use that type of assignment You can use another approach of searching for asset with type MetaproAppSetup and extracting from returned list needed asset. **WARNING:** This approach can lead to unexpected results when You have more than one scriptable object in Your project. Use it carefully.


```
var guid:string[] = AssetDatabase.FindAssets( filter: "t:MetaproAppSetup");
if (guid.Length == 0)
{
    Debug.LogError( message: "MetaproAppSetup not found");
    yield break;
}

if (guid.Length > 1)
{
    Debug.LogError( message: "Only one instance of MetaproAppSetup can persist in project");
    yield break;
}

var setups = new MetaproAppSetup[guid.Length];
for (var i = 0; i < setups.Length; i++)
{
    var path:string = AssetDatabase.GUIDToAssetPath(guid[i]);
    setups[i] = AssetDatabase.LoadAssetAtPath<MetaproAppSetup>(path);
}

var metaproAppSetup = setups[0];
```

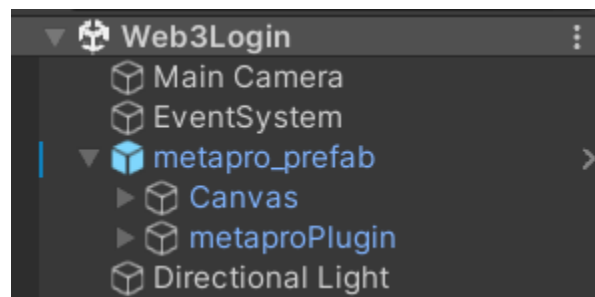
2. Play mode

This guide will show you how to work with players' login and its wallet data.

- Enable Web3 in Your game

To easily use Web3 features in Your application we provided a simple prefab to drag and drop into Your scene. You will find it in **Assets > Augmented Life Studio LLC > metapro for Unity > metaproSDK > Prefabs > metapro_plugin_prefab**.

Drag that object into the hierarchy of Your scene.



In prefab You will find Canvas that has implemented all screens needed in order to fully utilize web3 features.

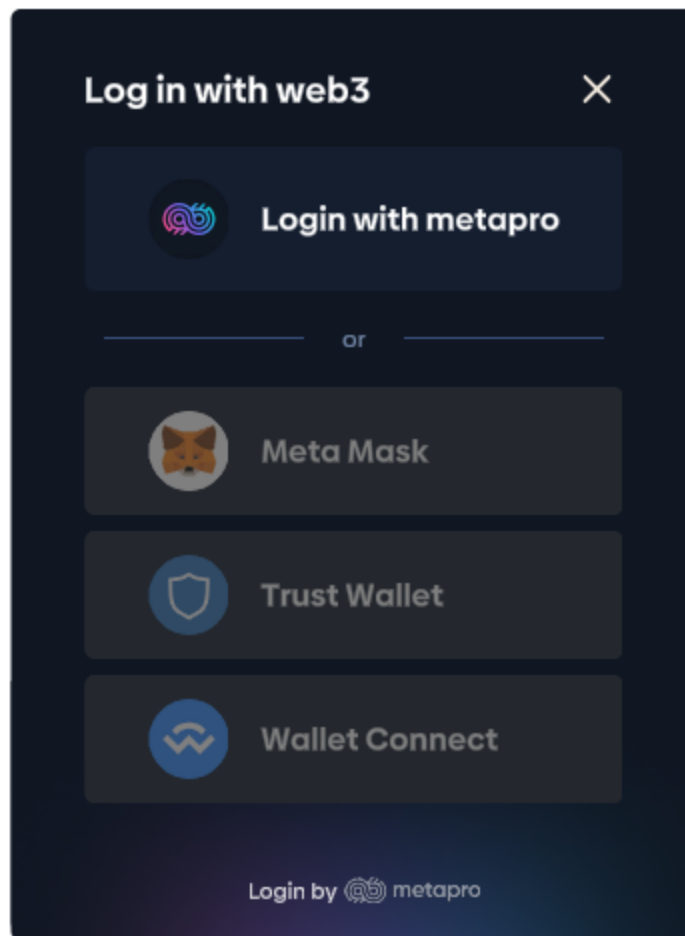
CAUTION: prefab doesn't contain **EventSystem**. You have to create it on Your own in Your scene.

All screens can be customized to match Your game style.

- Chose provider

Currently available provider to use is only the metapro wallet. Further development will extend that to use MetaMask, WalletConnect and Trust Wallet.

Players when a popup appears can choose a provider by clicking on its button.



- Connect wallet

To connect a wallet using the chosen provider, the user has to open his wallet app and in case of **metapro wallet** click to scan shown QR code.



After scanning, the user wallet application will show a confirmation screen to connect the wallet to Your application.

Wallet connect

Metapro is requesting to connect
to your wallet



Your wallet

0xbb851432edc19989d5c6cc54de34ecd7214c879
8

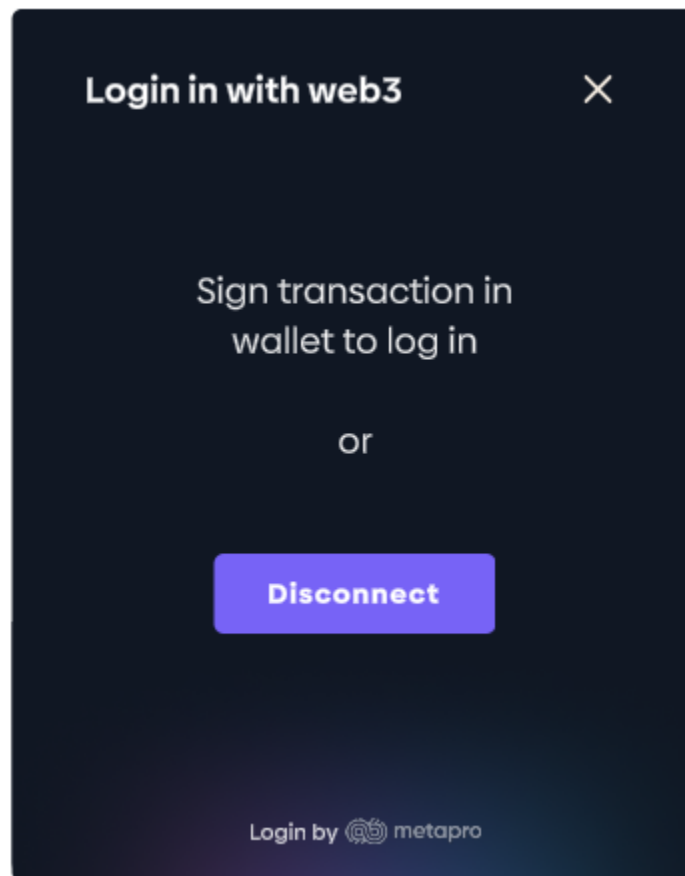
Connect

Reject

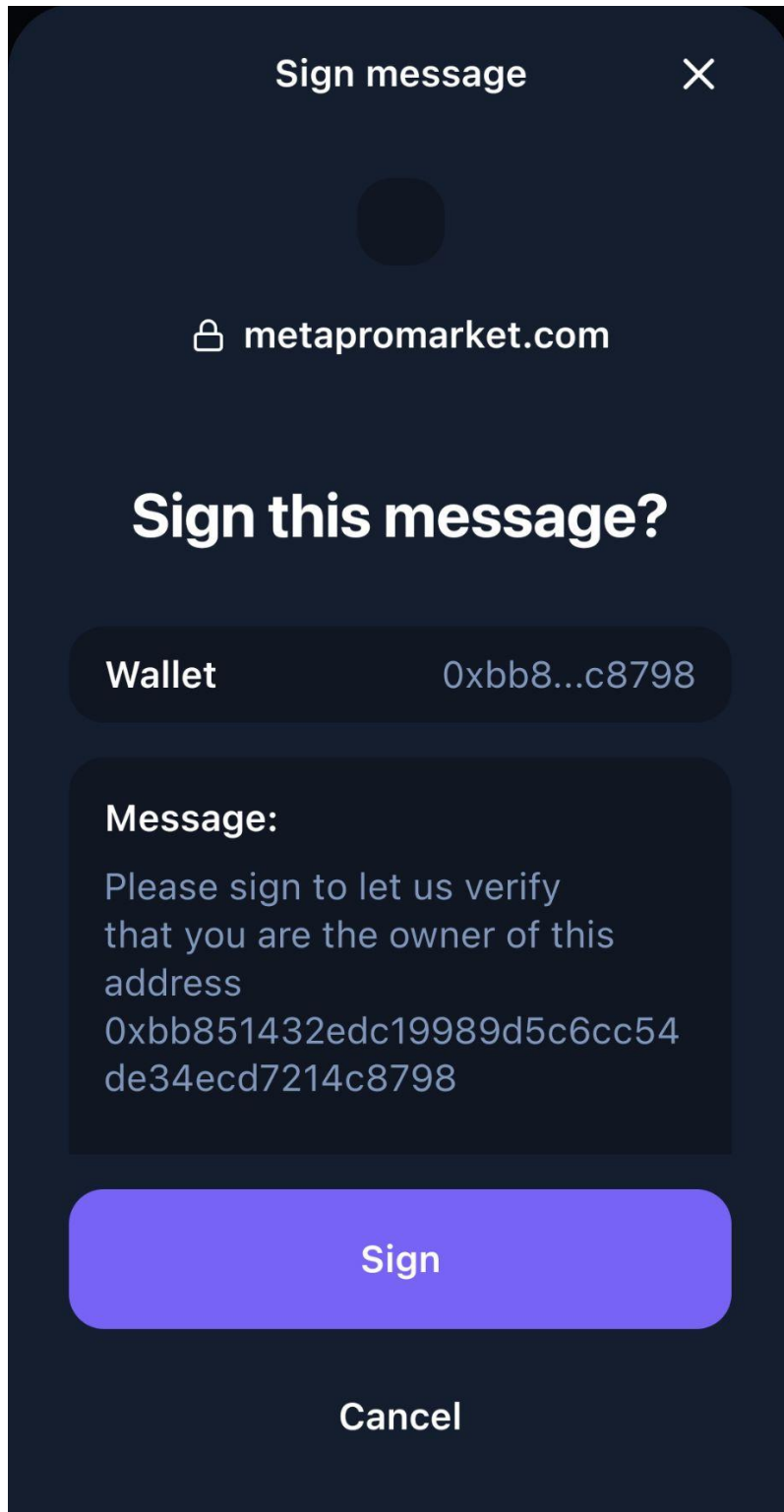
When a user clicks the **Connect** button on his wallet connection is established.

- Login with Web3

To provide all functionalities users need also log in using web3 signature transaction. Login will happen automatically after scanning QR code. While the application is waiting for the user to login SDK will display a popup that allows the player to disconnect the wallet. That window will always appear whenever the user wallet is connected but not logged in.

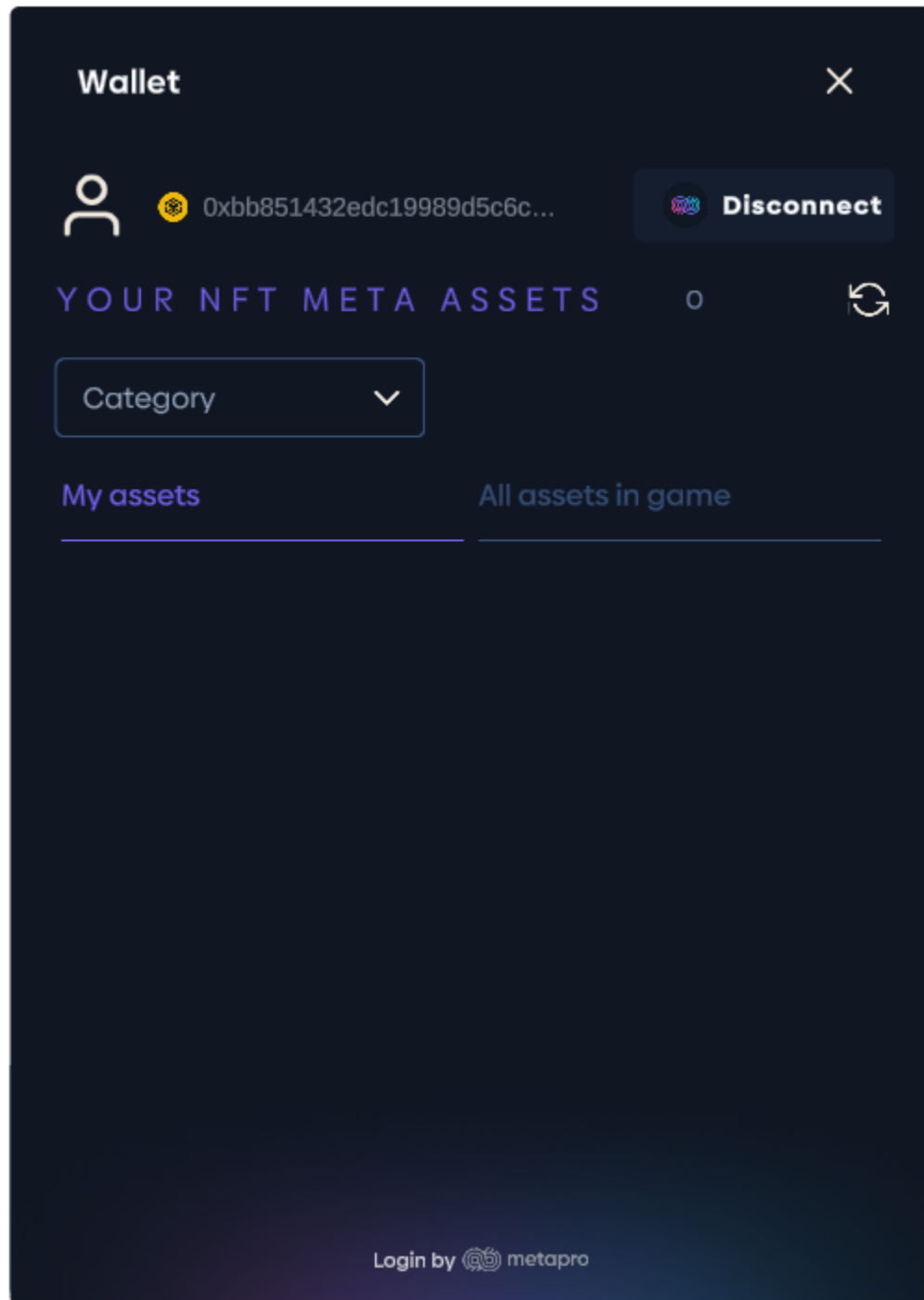


On the user wallet, transaction information will be displayed and requested to sign the login transaction. After clicking on the **Sign** button player will be signed into Your game.



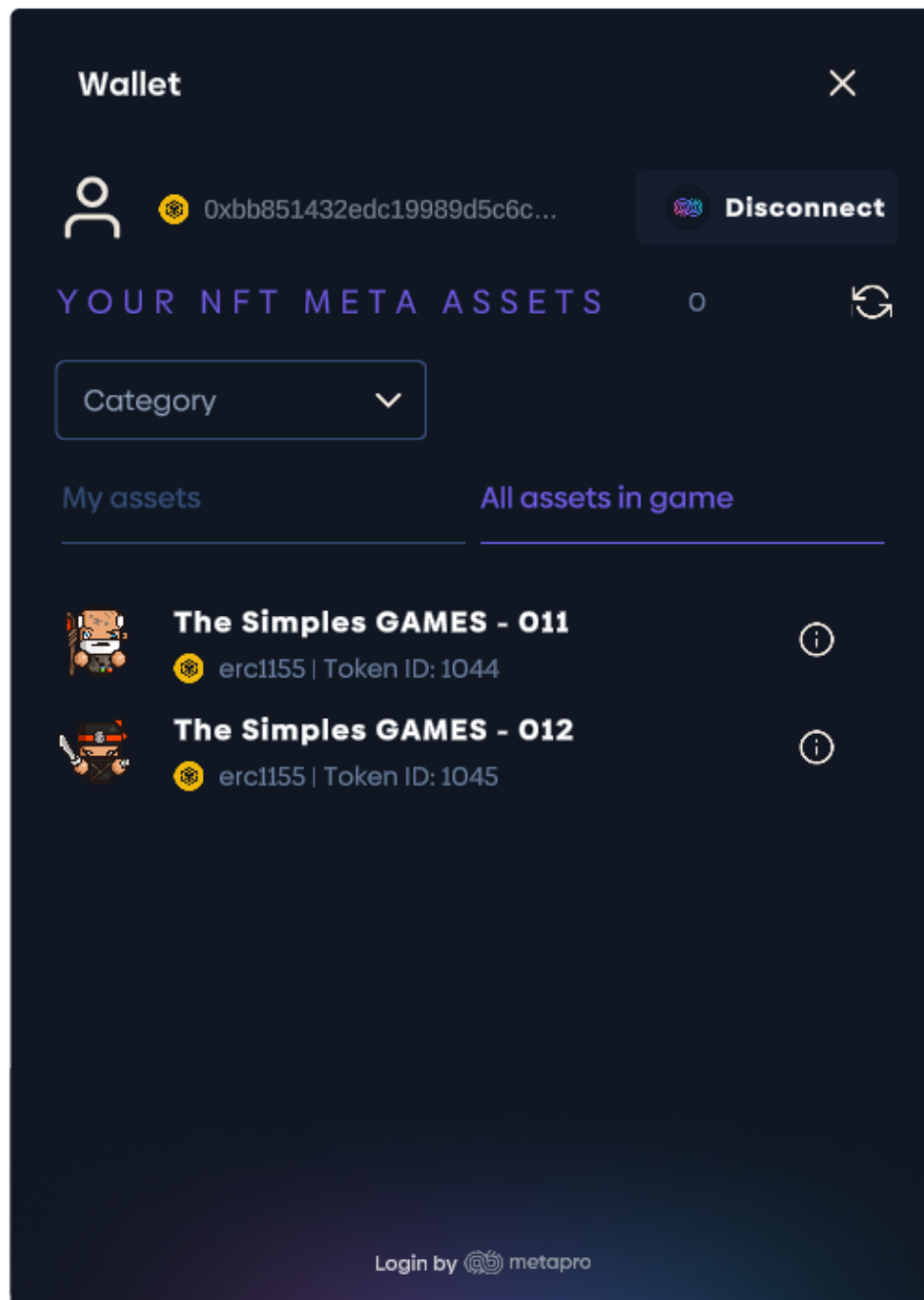
- Display user data


When the player properly connects the wallet and login to Your application a popup is displayed that shows user information at the top of screen. User information contains: wallet address, users NFTs also name and avatar if they're provided in metapro market user profile page.





- Show app tokens


To switch between user and app tokens, the player has to click respectively **My assets** or **All assets in game**. Note that **My assets** section will display only NFTs that are assigned and implemented in that project on the metapro market.




More information about particular token can be found after clicking on the  button next to the token name. Displayed screen will show information about the token and whether the user has that token in his wallet or not.

Wallet


 0xbb851432edc19989d5c6c...

 **Disconnect**




Not on wallet


The Simple GAMES - 011

 **erc1155** | Token ID: 1044

Item experience **soon**

NFT Details

Token ID	1044
Supply	1
Token standard	erc1155
Chain	 BNB Chain
Contract address	0xa293d6868...

Login by  metapro

- Use user and app data from script

To access user and app tokens from any other script You can get it from Singleton class called **PluginManager**. You will find there lists of user and application tokens and also user data. To use it type: **PluginManager.Instance.[public_field]**.

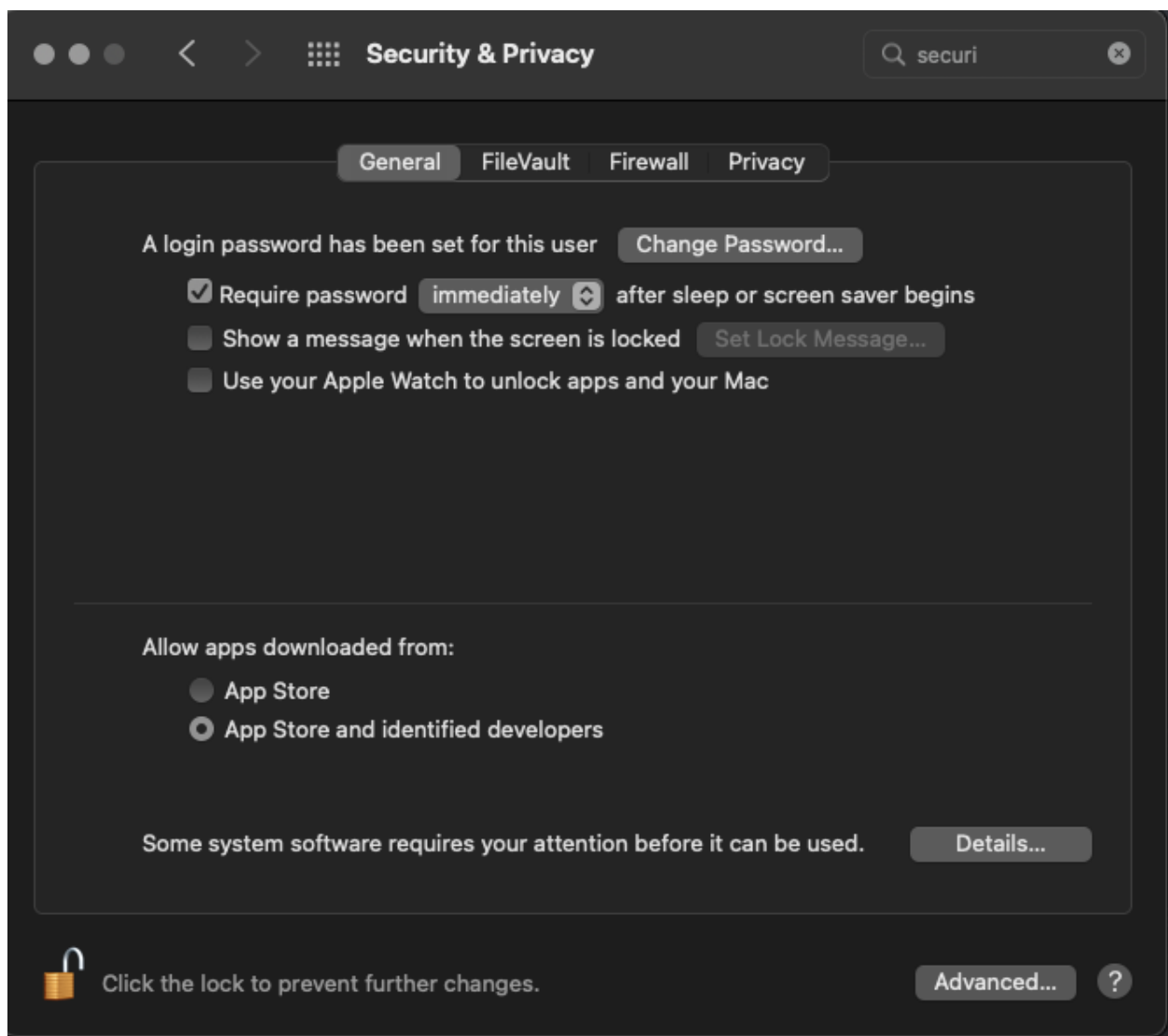
```
public List<NftTokenData> userNfts;  
public List<NftTokenData> applicationNfts;  
private UserData _userData;  
1 usage  Andrzej Fejklowicz  
public UserData UserData => _userData;
```

Troubleshooting

Known issues that could appear when using metapro SDK tool:

1. OSX developers Security & Privacy settings

When try to download assets from Metapro SDK Window Apple system can show popup about security issues with application. It does that because plugin uses GRPC library to download and create files. When that occur go to Your Security & privacy settings and enable it. It should work properly from now on.



Reference

MetaproAppSetup - *MetaproAppSetup.cs*

This class holds a Scriptable Object that saves data gathered from the metapro server.

Contains fields:

GameKey - Key provided on application setup

AppId - Application id stored in metapro database

GameImageURL - URL to uploaded preview image of game

GameName - Name of game

TeamName - Name of the team that is assigned to that application

GameAssets - List of all assets assign to the application on metapro market

```
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "MetaproAppSetup", menuName = "Metapro/AppSetup")]
public class MetaproAppSetup : ScriptableObject
{
    public string GameKey;
    public string AppId;

    public string GameImageURL;
    public string GameName;
    public string TeamName;

    public List<AvailableAsset> GameAssets;
}
```

Editor Window - *AppSetupWindow.cs*

AppSetupWindow extends *EditorWindow* class and provides all algorithms necessary to display metaproSDK setup editor window and manage its operations.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using Grpc.Core;
using MetaPod;
using metaproSDK.Scripts.Serialization;
using Newtonsoft.Json;
using Unity.EditorCoroutines.Editor;
using UnityEditor;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UIElements;

public class AppSetupWindow : EditorWindow
{
    private bool _hasKeyAssigned;
    private TemplateContainer baseContainer;
    private StyleSheet baseStyleSheet;
    private TemplateContainer importedContainer;
    private StyleSheet importedStyleSheet;

    private static string PROD_URL = "";

    private string _requestURL = PROD_URL;

    private MetaproAppSetup _metaproAppSetup;

    [MenuItem("Tools/Metapro SDK Setup")]
    public static void ShowWindow()
    {
        var window = GetWindow<AppSetupWindow>();
        window.titleContent = new GUIContent("Metapro SDK");
        window.minSize = new Vector2(400, 600);
        window.maxSize = new Vector2(400, 600);
    }

    private void OnEnable()
    {
        var editorAssetsPath = "Assets/Augmented Life Studio LLC/metapro for
Unity/Editor";
        VisualTreeAsset baseAssetTree =
AssetDatabase.LoadAssetAtPath<VisualTreeAsset>(editorAssetsPath +
"/AppSetupTemplate.uxml");
        baseContainer = baseAssetTree.CloneTree();
        baseStyleSheet =
AssetDatabase.LoadAssetAtPath<StyleSheet>(editorAssetsPath +
"/AppSetupStyles.uss");
```

```

        VisualTreeAsset importedAssetTree =
AssetDatabase.LoadAssetAtPath<VisualTreeAsset>(editorAssetsPath +
"/AppSetupImported.uxml");
        importedContainer = importedAssetTree.CloneTree();
        importedStyleSheet =
AssetDatabase.LoadAssetAtPath<StyleSheet>(editorAssetsPath +
"/AppSetupImportedStyles.uss");

        var guid = AssetDatabase.FindAssets("t:MetaproAppSetup");
        var setups = new MetaproAppSetup[guid.Length];
        for (int i = 0; i < setups.Length; i++)
        {
            var path = AssetDatabase.GUIDToAssetPath(guid[i]);
            setups[i] = AssetDatabase.LoadAssetAtPath<MetaproAppSetup>(path);
        }

        _metaproAppSetup = setups[0];

        CheckData();
        UpdateVisuals();
    }

    private void CheckData()
    {
        EditorUtility.SetDirty(_metaproAppSetup);
        AssetDatabase.SaveAssets();
        if (_metaproAppSetup.GameKey != "")
        {
            _hasKeyAssigned = true;
        }
        else
        {
            _hasKeyAssigned = false;
        }
    }

    public void UpdateVisuals()
    {
        if (_hasKeyAssigned)
        {
            EditorCoroutineUtility.StartCoroutine>ShowItemsView(), this);
        }
        else
        {
            EditorCoroutineUtility.StartCoroutine>ShowBaseView(), this);
        }
    }

    private IEnumerator ShowBaseView()
    {
        importedContainer.RemoveFromHierarchy();
        rootVisualElement.Add(baseContainer);
        rootVisualElement.styleSheets.Add(baseStyleSheet);

        yield return new WaitForSeconds(1f);
    }

```

```

        var textField =
rootVisualElement.Query<TextField>("styled_input").First();
        textField.value = "Enter app key";

        var setupButton = rootVisualElement.Query<Button>("button").First();
        setupButton.clicked -= SetupButtonOnClicked;
        setupButton.clicked += SetupButtonOnClicked;

        var pasteButton =
rootVisualElement.Query<Button>("paste_input").First();
        pasteButton.clicked -= PasteButtonOnClicked;
        pasteButton.clicked += PasteButtonOnClicked;

    }

    private IEnumerator ShowItemsView()
    {
        baseContainer.RemoveFromHierarchy();

        rootVisualElement.Add(importedContainer);
        rootVisualElement.styleSheets.Add(importedStyleSheet);

        yield return new WaitForSeconds(1f);

        var appName = rootVisualElement.Query<Label>("app_name").First();
        appName.text = _metaproAppSetup.GameName;

        var teamName = rootVisualElement.Query<Label>("team_name").First();
        teamName.text = _metaproAppSetup.TeamName;

        var appImage = rootVisualElement.Query<Image>("app_image").First();

EditorCoroutineUtility.StartCoroutine(GetTexture(_metaproAppSetup.GameImageURL
, appImage), this);

        var listView =
rootVisualElement.Query<ScrollView>("test_view").First();

        var changeKeyButton =
rootVisualElement.Query<Button>("change_app").First();
        changeKeyButton.clicked -= ClearImportedView;
        changeKeyButton.clicked += ClearImportedView;

        var downloadAllButton =
rootVisualElement.Query<Button>("download_all_button").First();
        downloadAllButton.clicked -= DownloadAllItems;
        downloadAllButton.clicked += DownloadAllItems;

        foreach (var gameAsset in _metaproAppSetup.GameAssets)
        {

            Box itemBox = new Box();
            itemBox.AddToClassList("item_element");

```



```

        Image itemImage = new Image();
        itemImage.AddToClassList("item_image");

EditorCoroutineUtility.StartCoroutine(GetTexture(gameAsset.ImageURL,
itemImage), this);
        Label itemName = new Label();
        itemName.AddToClassList("item_name");
        itemName.text = gameAsset.Name;
        Button itemButton = new Button();
        itemButton.AddToClassList("item_button");
        itemButton.text = "Download";
        itemButton.clicked += () =>
        {
            DownloadFile(gameAsset.BucketHash, gameAsset.TokenId,
gameAsset.ItemId);
        };
        itemBox.Add(itemImage);
        itemBox.Add(itemName);
        itemBox.Add(itemButton);
        listView.Add(itemBox);
    }

}

private IEnumerator GetTexture(string url, Image imageToAssign)
{
    UnityWebRequest www = UnityWebRequestTexture.GetTexture(url);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        Debug.Log(www.downloadHandler.error);
        imageToAssign.visible = false;
        imageToAssign.SetEnabled(false);
        EditorCoroutineUtility.StartCoroutine(GetGifData(url,
imageToAssign), this);
    }
    else
    {
        imageToAssign.SetEnabled(true);
        imageToAssign.visible = true;
        Texture texture =
((DownloadHandlerTexture)www.downloadHandler).texture;
        imageToAssign.image = texture;
    }
}

private IEnumerator GetGifData(string url, Image imageToAssign)
{
    UnityWebRequest www = UnityWebRequest.Get(url);
    yield return www.SendWebRequest();
    Debug.Log("Downloading preview GIF for image: " + url);
    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        Debug.Log(www.downloadHandler.error);
    }
}

```

```

    }
    else
    {
        Debug.Log("Downloaded GIF texture");
        var fileId = url.Split('/')[3];
        var relativePath = "Assets/Resources/downloadedGif_" + fileId +
".gif";

        ByteArrayToFile(relativePath, www.downloadHandler.data);
        AssetDatabase.ImportAsset(relativePath);
        AssetDatabase.Refresh();
        TextureImporter importer =
(TextureImporter) TextureImporter.GetAtPath(relativePath);

        importer.isReadable = true;
        importer.textureType = TextureImporterType.Sprite;

        TextureImporterSettings importerSettings = new
TextureImporterSettings();
        importer.ReadTextureSettings(importerSettings);
        importerSettings.textureType = TextureImporterType.Sprite;
        importerSettings.spriteExtrude = 0;
        importerSettings.spriteGenerateFallbackPhysicsShape = false;
        importerSettings.spriteMeshType = SpriteMeshType.FullRect;
        importerSettings.spriteMode = (int)SpriteImportMode.Single;
        importer.SetTextureSettings(importerSettings);
        importer.spriteImportMode = SpriteImportMode.Single;
        importer.maxTextureSize = 1024;
        importer.alphaIsTransparency = true;
        importer.textureCompression =
TextureImporterCompression.Uncompressed;
        importer.alphaSource = TextureImporterAlphaSource.FromInput;
        EditorUtility.SetDirty(importer);

        importer.SaveAndReimport();
        Sprite spriteImage =
(Sprite) AssetDatabase.LoadAssetAtPath(relativePath, typeof(Sprite));
        imageToAssign.image = spriteImage.texture;
        imageToAssign.SetEnabled(true);
        imageToAssign.visible = true;
    }
}

private void DownloadAllItems()
{
    foreach (var gameAsset in _metaproAppSetup.GameAssets)
    {
        DownloadFile(gameAsset.BucketHash, gameAsset.TokenId,
gameAsset.ItemId);
    }
}

private void ClearImportedView()
{
    var listView =
rootVisualElement.Query<ScrollView>("test_view").First();
    _metaproAppSetup.GameKey = "";
}

```

```

        _metaproAppSetup.AppId = "";
        _metaproAppSetup.GameName = "";
        _metaproAppSetup.GameImageURL = "";
        _metaproAppSetup.TeamName = "";
        _metaproAppSetup.GameAssets = new List<AvailableAsset>();
        listView.contentContainer.Clear();
        CheckData();
        UpdateVisuals();
    }

    private void PasteButtonOnClicked()
    {
        TextEditor textEditor = new TextEditor();
        textEditor.Paste();
        var textField =
rootVisualElement.Query<TextField>("styled_input").First();
        textField.value = textEditor.text;
    }
    private void SetupButtonOnClicked()
    {
        EditorCoroutineUtility.StartCoroutine(GetAppData(), this);
    }

    private string DownloadFile(string bucketHash, int tokenId, string itemId)
    {
        var channel = new Channel("prd-pod-1.metaprotocol.one:8181",
ChannelCredentials.Insecure);

        var client = new Storage.StorageClient(channel);

        var bucketResponse = client.GetBucket(new GetBucketRequest {
BucketIdentifier = bucketHash });

        var fileURI = "";
        foreach (var bucketResponseFile in bucketResponse.Files)
        {
            var fileResponse = client.GetFile(new GetFileRequest {
FileIdentifier = bucketResponseFile });

            var fileBytes = Array.Empty<byte>();
            SortedDictionary<ulong, byte[]> fileChunksDictionary = new
SortedDictionary<ulong, byte[]>();

            foreach (var fileResponseChunk in fileResponse.Chunks)
            {
                var fileChunkResponse = client.GetChunk(new GetChunkRequest {
ChunkIdentifier = fileResponseChunk.ChunkIdentifier });
                fileChunksDictionary.Add(fileChunkResponse.ChunkIndex,
fileChunkResponse.Chunk.ToByteArray());
            }

            foreach (var (key, value) in fileChunksDictionary)
            {
                fileBytes = CombineBytes(fileBytes, value);
            }
        }
    }

```

```

    }

    Directory.CreateDirectory("Assets/Resources/" + tokenId);
    ByteArrayToFile("Assets/Resources/" + tokenId + "/" +
fileResponse.FileName, fileBytes);
    Debug.Log("Asset download completed");
    fileURI = "Assets/Resources/" + fileResponse.FileName;

EditorCoroutineUtility.StartCoroutine(SendDownloadConfirmation(_metaproAppSetu
p.AppId, bucketHash, tokenId, itemId, bucketResponseFile), this);

        AssetDatabase.ImportAsset("Assets/Resources/" + tokenId,
ImportAssetOptions.ImportRecursive);
    }

    return fileURI;
}

public static byte[] CombineBytes(byte[] first, byte[] second)
{
    byte[] bytes = new byte[first.Length + second.Length];
    Buffer.BlockCopy(first, 0, bytes, 0, first.Length);
    Buffer.BlockCopy(second, 0, bytes, first.Length, second.Length);
    return bytes;
}

public bool ByteArrayToFile(string fileName, byte[] byteArray)
{
    try
    {
        using (var fs = new FileStream(fileName, FileMode.Create,
FileAccess.Write))
        {
            fs.Write(byteArray, 0, byteArray.Length);
            return true;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception caught in process: {0}", ex);
        return false;
    }
}

private IEnumerator SendDownloadConfirmation(
    string appId,
    string bucketHash,
    int tokenId,
    string itemId,
    string fileId)
{
    WWWForm form = new WWWForm();
    form.AddField("appId", appId);
    form.AddField("_bucketHash", bucketHash);
    form.AddField("_tokenId", tokenId);
    form.AddField("itemId", itemId);

```

```

        form.AddField("fileId", fileId);

        using (UnityWebRequest www = UnityWebRequest.Post("https://" +
            _requestURL + "api.metaproprotocol.com/ms/teams/v1/collect", form))
        {
            yield return www.SendWebRequest();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.LogError(www.error);
                Debug.LogError(www.downloadHandler.text);
            }
            else
            {
                Debug.Log("Form upload complete!");
            }
        }
    }

    protected IEnumerator GetAppData()
    {
        var appKey =
            rootVisualElement.Query<TextField>("styled_input").First();
        var appKeyValue = appKey.value;

        using (UnityWebRequest www = UnityWebRequest.Get("https://" +
            _requestURL + "api.metaproprotocol.com/ms/apps/v1/apps/appkey/" +
            appKeyValue))
        {
            yield return www.SendWebRequest();

            if (www.isNetworkError || www.isHttpError)
            {
                Debug.LogError(www.error);
            }
            else
            {
                _metaproAppSetup.GameKey = appKeyValue;

                var appItemsResult =
                    JsonConvert.DeserializeObject<AppItemsResult>(www.downloadHandler.text);

                _metaproAppSetup.AppId = appItemsResult.appId;
                _metaproAppSetup.GameImageURL = appItemsResult.gallery[0];
                _metaproAppSetup.GameName = appItemsResult.name;
                _metaproAppSetup.TeamName = appItemsResult.team;

                _metaproAppSetup.GameAssets = new List<AvailableAsset>();

                foreach (var itemResult in appItemsResult.items)
                {
                    AvailableAsset asset = new AvailableAsset();
                    asset.Name = itemResult.tokenName;
                    asset.BucketHash = itemResult._bucketHash;
                    asset.ImageURL = itemResult.image;
                    asset.ItemId = itemResult.itemId;
                }
            }
        }
    }

```

```
        asset.TokenId = itemResult._tokenId;
        _metaproAppSetup.GameAssets.Add(asset);
    }

    CheckData();
    UpdateVisuals();
}

}
```

Plugin Manager - *PluginManager.cs*

This class is the brain of metapro plugin for Unity operation. This Singleton stores all data and needed functions for easy access during game Runtime.

Important public fields:

UserData - stores downloaded user data (username, bio, avatar etc.) provided on profile page on metapro market

userNfts/applicationNfts - lists containing NFT token data

selectedNft - NFT token data that users select to view in modal

Class provides methods to request data from metapro server for userData, user/application NFTs.

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using metaproSDK.Scripts.Controllers;
using metaproSDK.Scripts.Serialization;
using metaproSDK.Scripts.Utils;
using Newtonsoft.Json;
using Serialization;
using UnityEngine;
using UnityEngine.Networking;

namespace metaproSDK.Scripts
{
    public class PluginManager : Singleton<PluginManager>
    {
        [SerializeField] private MetaproAppSetup metaproAppSetup;
        [SerializeField] private UserWindowController userWindowController;

        [SerializeField] private List<ProviderController> providerPrefabs;

        [Serializable]
        public struct ChainSprite {
            public ChainType type;
            public Sprite sprite;
        }
        public List<ChainSprite> chainsSprites;

        [Serializable]
        public struct WalletSprite {
            public WalletProviderType type;
            public Sprite sprite;
        }
        public List<WalletSprite> walletsSprites;
    }
}
```

```

        public Sprite GetSelectedWalletSprite => walletsSprites.First(p =>
p.type == selectedWalletProvider).sprite;

        public Color activeBadgeColor;
        public Color inActiveBadgeColor;

        public List<NftTokenData> userNfts;
        public List<NftTokenData> applicationNfts;
        private UserData _userData;
        public UserData UserData => _userData;

        public NftTokenData selectedNft;
        private WalletProviderType selectedWalletProvider;
        public WalletProviderType SelectedWalletProvider =>
selectedWalletProvider;
        private ProviderController _providerController;

        private void Start()
        {
            userNfts = new List<NftTokenData>();
            applicationNfts = new List<NftTokenData>();
            StartCoroutine(GetApplicationNFT());
        }

        public void OnWalletConnected()
        {
            userWindowController.ShowLoginWeb3Screen();
            _providerController.RequestSign();
        }

        public void ClearCurrentProvider()
        {
            Destroy(_providerController.gameObject);
            _providerController = null;
        }

        public void SetupUserData(UserData userData)
        {
            _userData = new UserData();
            _userData.userId = userData.userId;
            _userData.wallet = userData.wallet;
            _userData.userName = userData.userName;
            _userData.userBio = userData.userBio;
            _userData.userAvatarURL = userData.userAvatarURL;
            _userData.accessToken = userData.accessToken;
            _userData.tokenType = userData.tokenType;

            var url = "https://api.metaproprotocol.com/ms/nft/v1/user/" +
            _userData.wallet +
            "/tokens?_items=true&sort%5Btoken.creationBlock%5D=desc";
            StartCoroutine(GetUserNFT(url));
        }

```



```

        public IEnumerator GetApplicationNFT()
        {
            var getAppAssets =
UnityWebRequest.Get("https://api.metaproprotocol.com/ms/teams/v1/items?appId="
+ metaproAppSetup.AppId);

            yield return getAppAssets.SendWebRequest();

            if (getAppAssets.isNetworkError || getAppAssets.isHttpError)
            {
                Debug.LogError(getAppAssets.error);
                Debug.LogError(getAppAssets.downloadHandler.text);
                yield break;
            }

            var appAssets =
JsonConvert.DeserializeObject<Results<ItemResult>>(getAppAssets.downloadHandle
r.text);

            var nftParams = String.Join('&', appAssets.results.Select(a =>
"tokenIds=" + a._tokenId).ToList());
            var requestUrl =
"https://api.metaproprotocol.com/ms/nft/v1/tokens?" + nftParams;

            UnityWebRequest getNfts = UnityWebRequest.Get(requestUrl);

            yield return getNfts.SendWebRequest();

            if (getNfts.isNetworkError || getNfts.isHttpError)
            {
                Debug.LogError(getNfts.error);
                yield break;
            }
            var appNftResults =
JsonConvert.DeserializeObject<Results<NftUserTokensResult>>(getNfts.downloadHa
ndler.text);

            foreach (var nftTokensResult in appNftResults.results)
            {
                var nftTokenData = new NftTokenData();
                nftTokenData.tokenId = nftTokensResult.token._tokenId;
                nftTokenData.quantity = nftTokensResult.token._quantity;
                nftTokenData.imageUrl = nftTokensResult.token.image;
                nftTokenData.tokenName = nftTokensResult.token.tokenName;
                nftTokenData.standard = nftTokensResult.standard;
                nftTokenData.contract = nftTokensResult.token.address;
                nftTokenData.supply = nftTokensResult.token._quantity;
                nftTokenData.chain =
ChainTypeExtension.GetChainById(nftTokensResult.chainId);
                foreach (var tokenProperty in
nftTokensResult.token.properties)
                {
                    if (tokenProperty.key == "asset_category")
                    {
                        nftTokenData.category = (string)tokenProperty.value;
                    }
                }
            }
        }
    }

```

```

        }
        applicationNfts.Add(nftTokenData);
    }
}

private IEnumerator GetUserNFT(string url)
{
    yield return new WaitForSeconds(0.5f);
    UnityWebRequest www = UnityWebRequest.Get(url);
    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        Debug.Log(www.error);
        Debug.Log(www.downloadHandler.error);
    }
    else
    {
        var nftTokenResult =
JsonConvert.DeserializeObject<Results<NftUserTokensResult>>(www.downloadHandle
r.text);

        foreach (var nftUserTokensResult in nftTokenResult.results)
        {
            var nftTokenData = new NftTokenData();
            nftTokenData.tokenId = nftUserTokensResult.token._tokenId;
            if (nftUserTokensResult.owners.Length != 0)
            {
                nftTokenData.quantity =
nftUserTokensResult.owners[0]._quantity;
            }

            nftTokenData.imageUrl = nftUserTokensResult.token.image;
            nftTokenData.tokenName =
nftUserTokensResult.token.tokenName;
            nftTokenData.standard = nftUserTokensResult.standard;
            nftTokenData.contract = nftUserTokensResult.token.address;
            nftTokenData.supply = nftUserTokensResult.token._quantity;
            nftTokenData.chain =
ChainTypeExtension.GetChainById(nftUserTokensResult.chainId);

            foreach (var tokenProperty in
nftUserTokensResult.token.properties)
            {
                if (tokenProperty.key == "asset_category")
                {
                    nftTokenData.category =
(string)tokenProperty.value;
                }
            }

            userNfts.Add(nftTokenData);
        }
    }
    userWindowController.ShowAssetsScreen();
}

```

```

    public void ShowAssetCard(NftTokenData nftTokenData)
    {
        selectedNft = nftTokenData;
        userWindowController.ShowAssetCardScreen();
    }

    public void ShowProviderLogin(WalletProviderType walletProviderType)
    {
        selectedWalletProvider = walletProviderType;
        _providerController = Instantiate(providerPrefabs.First(p =>
p.ProviderType == selectedWalletProvider), transform);
        _providerController.ShowConnection();
    }

    public void ShowQRCodeScreen(Sprite qrCodeSprite)
    {
        userWindowController.ShowQRCodeScreen(qrCodeSprite);
    }

    public void DisconnectWallet()
    {
        _providerController.DisconnectWallet();
        userWindowController.ShowProviderScreen();
    }
}

```

Web3 connection providers:

- Metapro : currently uses WalletConnect provider standard
- Wallet Connect - *WalletConnectProviderController.cs*

Class contains setup for Wallet Connect type connection. Showing QR code and requesting personal sign login.

```
using System.Collections;
using System.Threading.Tasks;
using metaproSDK.Scripts.Serialization;
using metaproSDK.Scripts.Utills;
using UnityEngine;
using WalletConnectSharp.Core.Models;
using WalletConnectSharp.Unity;

namespace metaproSDK.Scripts.Controllers
{
    public class WalletConnectProviderController : ProviderController
    {
        private WalletConnect _walletConnect;

        public override void Initialize()
        {
            _walletConnect = GetComponent<WalletConnect>();

            _walletConnect.ConnectedEventSession.AddListener(OnConnectionStarted);
            _walletConnect.DisconnectedEvent.AddListener(OnDisconnect);
        }

        private void OnConnectionStarted(WCSessionData arg0)
        {
            PluginManager.Instance.OnWalletConnected();
        }

        private void OnDisconnect(WalletConnectUnitySession arg0)
        {
            Debug.LogWarning("User disconnected");
            PluginManager.Instance.DisconnectWallet();
        }

        public override void ShowConnection()
        {
            StartCoroutine(DelayConnectionStart());
        }

        private IEnumerator DelayConnectionStart()
        {
            yield return new WaitUntil(() => _walletConnect != null &&
            _walletConnect.isConnectionStarted);
        }
    }
}
```

```

        var qrCodeSprite =
QRCodeImageHandler.GenerateQRCode(_walletConnect.Session.URI);
        PluginManager.Instance.ShowQRCodeScreen(qrCodeSprite);
    }

    public override void RequestSign()
    {
        StartCoroutine(SignLogin());
    }

    private IEnumerator SignLogin()
    {
        yield return new WaitForSeconds(1f);
        var wallet =
WalletConnect.ActiveSession.Accounts[0].ToLower();

        var hash = "";
        yield return
StartCoroutine(MetaproServerRequests.GetHashss(wallet, value => hash =
value));

        var verifyMessage = $"Please sign to let us verify\nthat you
are the owner of this address\n{wallet}\n\nRequest ID {hash}";

        var task = Task.Run(async () => await
WalletConnect.ActiveSession.EthPersonalSign(wallet, verifyMessage));
        yield return new WaitUntil(() => task.IsCompleted);
        var loginSignature = task.Result;

        var userData = new UserData();
        yield return
StartCoroutine(MetaproServerRequests.LoginWallet(wallet, loginSignature,
hash, value => userData = value));

        PluginManager.Instance.SetupUserData(userData);
    }

    public override void DisconnectWallet()
    {
        WalletConnect.ActiveSession.Disconnect();
        Destroy(gameObject, 1f);
    }
}

```

- MetaMask - *MetaMaskProviderController.cs* & *MetaMaskConnectionListener.cs*

MetaMaskProviderController class manages requests for connection with QR code personal sign to MetaMask wallet.

MetaMaskConnectionListener class listens to MetaMask plugin responses and based on them calls an action.

```

using System;
using System.Collections;
using System.Threading.Tasks;
using MetaMask.Models;
using MetaMask.Unity;
using metaproSDK.Scripts.Serialization;
using metaproSDK.Scripts.Utills;
using UnityEngine;

namespace metaproSDK.Scripts.Controllers
{
    public class MetaMaskProviderController : ProviderController
    {
        private bool _disconnected = false;
        public override void Initialize()
        {
            MetaMaskUnity.Instance.Initialize();
            MetaMaskUnity.Instance.Wallet.WalletAuthorized +=
OnWalletConnected;
            MetaMaskUnity.Instance.Wallet.WalletDisconnected +=
OnWalletDisconnected;
        }

        private void OnDestroy()
        {
            MetaMaskUnity.Instance.Wallet.WalletAuthorized -=
OnWalletConnected;
            MetaMaskUnity.Instance.Wallet.WalletDisconnected -=
OnWalletDisconnected;
        }

        private void OnWalletConnected(object sender, EventArgs e)
        {
            PluginManager.Instance.OnWalletConnected();
        }

        private void OnWalletDisconnected(object sender, EventArgs e)
        {
            _disconnected = true;
            PluginManager.Instance.DisconnectWallet();
        }

        public override void ShowConnection()
        {
            MetaMaskUnity.Instance.Wallet.Connect();
        }

        public override void RequestSign()
        {
            StartCoroutine(SignLogin());
        }

        private IEnumerator SignLogin()
        {
            yield return new WaitForSeconds(1f);
            var wallet =

```

```

MetaMaskUnity.Instance.Wallet.SelectedAddress.ToLower();

        var hash = "";
        yield return
StartCoroutine(MetaproServerRequests.GetHashss(wallet, value => hash =
value));

        var verifyMessage = $"Please sign to let us verify\nthat you
are the owner of this address\n{wallet}\n\nRequest ID {hash}";

        var paramsArray = new[] { wallet, verifyMessage };

        var request = new MetaMaskEthereumRequest
        {
            Method = "personal_sign",
            Parameters = paramsArray
        };

        var task = Task.Run(async () => await
MetaMaskUnity.Instance.Wallet.Request(request));
        yield return new WaitUntil(() => task.IsCompleted);
        var loginSignature = task.Result;

        var userData = new UserData();
        yield return
StartCoroutine(MetaproServerRequests.LoginWallet(wallet,
loginSignature.ToString(), hash, value => userData = value));

        PluginManager.Instance.SetupUserData(userData);
    }

    public override void DisconnectWallet()
    {
        if (_disconnected == false)
        {
            MetaMaskUnity.Instance.Wallet.Disconnect();
        }
        Destroy(gameObject, 1f);
    }
}
}

```

```

using System;
using MetaMask.Models;
using MetaMask.Transports.Unity;
using metaproSDK.Scripts.Utills;
using UnityEngine;

namespace metaproSDK.Scripts.Controllers.ConnectionListeners
{
    public class MetaMaskConnectionListener : MonoBehaviour,

```

```

IMetaMaskUnityTransportListener
{
    public void OnMetaMaskConnectRequest(string url)
    {
        var qrCodeSprite = QRCodeImageHandler.GenerateQRCode(url);
        PluginManager.Instance.ShowQRCodeScreen(qrCodeSprite);
    }

    public void OnMetaMaskRequest(string id, MetaMaskEthereumRequest request)
    {
    }

    public void OnMetaMaskFailure(Exception error)
    {
    }

    public void OnMetaMaskSuccess()
    {
    }
}

```

WindowController - *UserWindowController.cs*

This class handles switching between modal windows and holds their current state.

```

using System;
using metaproSDK.Scripts.Utills;
using UnityEngine;
using UnityEngine.UI;

namespace metaproSDK.Scripts.Controllers
{
    public class UserWindowController : MonoBehaviour
    {
        [SerializeField] private GameObject providerWindow;
        [SerializeField] private GameObject qrCodeWindow;
        [SerializeField] private GameObject waitLoginWeb3Window;
        [SerializeField] private AssetsWindowController assetsWindow;

        [SerializeField] private Image qrCodeImage;

        private ScreenType _currentScreenType;
        private bool _isScreenOpened;

        private void Start()
        {
            _currentScreenType = ScreenType.Providers;
        }
    }
}

```



```

    }

    public void HideAllScreens()
    {
        providerWindow.SetActive(false);
        qrCodeWindow.SetActive(false);
        waitLoginWeb3Window.SetActive(false);
        assetsWindow.gameObject.SetActive(false);
        _isScreenOpened = false;
    }

    public void ShowProviderScreen()
    {
        HideAllScreens();
        providerWindow.SetActive(true);
        _currentScreenType = ScreenType.Providers;
        _isScreenOpened = true;
    }

    public void ShowQRCodeScreen(Sprite qrCodeSprite)
    {
        HideAllScreens();
        qrCodeWindow.SetActive(true);
        _currentScreenType = ScreenType.QRCode;
        _isScreenOpened = true;
        qrCodeImage.sprite = qrCodeSprite;
    }

    public void ShowLoginWeb3Screen()
    {
        HideAllScreens();
        waitLoginWeb3Window.SetActive(true);
        _currentScreenType = ScreenType.LoginWeb3;
        _isScreenOpened = true;
    }

    public void ShowAssetsScreen()
    {
        HideAllScreens();
        assetsWindow.gameObject.SetActive(true);
        assetsWindow.ShowAssetsList();
        _currentScreenType = ScreenType.Assets;
        _isScreenOpened = true;
    }

    public void ShowAssetCardScreen()
    {
        HideAllScreens();
        assetsWindow.gameObject.SetActive(true);
        assetsWindow.ShowNftCard();
        _currentScreenType = ScreenType.NFTCard;
        _isScreenOpened = true;
    }

    public void EnableCurrentScreen(bool enable)
    {

```

```

        if (enable == false)
        {
            HideAllScreens();
            return;
        }
        switch (_currentScreenType)
        {
            case ScreenType.Providers:
                ShowProviderScreen();
                break;
            case ScreenType.QRCode:
                PluginManager.Instance.ClearCurrentProvider();
                ShowProviderScreen();
                break;
            case ScreenType.LoginWeb3:
                ShowLoginWeb3Screen();
                break;
            case ScreenType.Assets:
                ShowAssetsScreen();
                break;
            case ScreenType.NFTCard:
                ShowAssetCardScreen();
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
    }

    public void ToggleScreen()
    {
        if (!_isScreenOpened)
        {
            EnableCurrentScreen(false);
            return;
        }
        EnableCurrentScreen(true);
    }
}

```