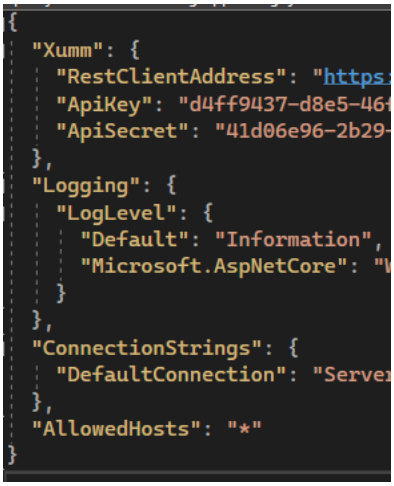
Ledger Engine Connect

POC Document

—

Introduction

In this document you will find a simple deployment guide of the current POC.



Application Settings

To change any of the XUMM credentials or the connection strings/passwords to the database, please open the appsettings.json file and then redeploy as explained above.

XUMM API Documentation

Signing in and Authentication

Methods

- The method “StartAuthentication” Preps the data that needs to be sent to XUMMAPI

```
/// <summary>
/// Starts the authentication process
/// </summary>
/// <returns>Player object with authentication packet</returns>
3 references
public Player StartAuthentication()
{
    // Create the Sign in Transaction and ad Custom Meta to it.
    var payload = new XummPayloadTransaction(XummTransactionType.SignIn).ToXummPostJsonPayload();
    payload.CustomMeta = new XummPayloadCustomMeta { Instruction = "Authenticate payload" };

    //Call the athenticate user method async and return a task.
    var result =  AuthenticateUser(payload);

    //returns the qrcode
    return result.Result;

}
```

- The method “AuthenticateUser” sends the data to XUMMAPI and returns player Data, which gets sent to Unity APP (Data consists of QR url)

```
/// <summary>
/// Async task for authentication of the user
/// </summary>
/// <param name="payload"></param>
/// <returns>Task to calling function</returns>
2 references
public async Task<Player> AuthenticateUser(XummPostJsonPayload payload)
{

    // Call XUMM API's create payload method (true flag there incase errors happen
    var result = await _sdk.Payload.CreateAsync(payload, true);

    // Check if the result is not null, then return the QR png
    if (result != null)
        return new Player
        {
            uuid = result.Uuid,
            qrurl = result.Refs.QrPng
        };

    //for testing
    /*  return result.Refs.QrPng;
    *///result.Uuid WVD  "254bc6e6-3c3f-47f4-8b93-7749c3fae076"

    //return the player
    return new Player();
}
```

Data Structures

All the player's authentication information is contained in the <player> object, which is constructed as follows:

```
/// <summary>
/// This shared library is so i can share Models with Unity directly.
/// </summary>
[Serializable]
11 references
public class Player
{
    public int playerGuid;          //identifier
    public string? playerName;      //players name
    public string? qrurl;           //url to the QR code for authentication
    public string? uuid;            //the player's XUMM UUID
}
```

This shared library dll must be copied to the unity project that is using the API (and must be updated if the code changes). There is a post build process configured for this, you just have to change the path to the correct one:

Getting Player NFTs

Methods

Here we get a list of all NFT's for an account. We call the XRP API endpoint "account_nfts" which returns metadata for all NFT's in a list format. We then make another call to the marketplace-api to get the DNA and image of the NFT before sending it back.

- The method GetPlayerNFTS gets the list

```
/// <summary>
/// Gets a list of the players NFT's
/// </summary>
/// <param name="account"></param>
/// <returns></returns>
2 references
public List<NftDnaResponse> GetPlayerNFTS(string account)
{
    // preparing API request
    Param[] paramsList = new Param[1];
    paramsList[0] = new Param
    {
        Account = account,
        LedgerIndex = "validated"
    };

    var nftObject = new NftRequestModel
    {
        Method = "account_nfts",
        Params = paramsList
    };

    //Serializing account object
    var serializationNftObject = JsonConvert.SerializeObject(nftObject);

    //Rest call to XRPcluster.com
    var client = new RestClient("https://xrplcluster.com");
    var request = new RestRequest("resource", Method.Post);
    request.RequestFormat = DataFormat.Json;
    request.AddHeader("Accept", "application/json");
    request.AddHeader("Content-Type", "application/json");
    request.AddParameter("application/json", serializationNftObject, ParameterType.RequestBody);

    //getting response
    var response = client.Execute(request);

    //Deserializing response into NftAccount Model
    var result = JsonConvert.DeserializeObject<NftAccount>(response.Content);

    // Retrieve the player's NFTs from NftAccount MOdel
    var nftList = new List<NftDnaResponse>();
    foreach (var nft in result.Result.AccountNfts)
    {
        var dna = GetNftMetaData(nft.NfTokenId);
        if (dna != null)
            nftList.Add(dna);
    }

    // return nftlist for account
    return nftList;
}
```

- The method GetNftMetaData gets the metadata and images (We do a validation to filter entries to only have Monkee NFT's)

```
/// <summary>
/// Gets the NFT metadata and prepares response
/// </summary>
/// <param name="nftTokenId"></param>
/// <returns></returns>
1 reference
public NftDnaResponse GetNftMetaData(string nftTokenId)
{
    var client = new RestClient("https://marketplace-api.onxrp.com");
    var request = new RestRequest("/api/metadata/" + nftTokenId, Method.Get);
    request.AddHeader("accept", "application/json");
    var response = client.Get(request);
    if (response.StatusCode == System.Net.HttpStatusCode.OK)
    {
        var result = JsonConvert.DeserializeObject<NftDnaResponse>(response.Content.ToString());
        if (result.collection.name == "MONKEE MONKEE")
        {
            result.image = GetImage(result.image);
            return result;
        }
    }
    return new NftDnaResponse();
}
```

- The GetImage helper function for getting the image:

```
/// <summary>
/// Helper function to get the image from the interplanetary file system
/// </summary>
/// <param name="image"></param>
/// <returns>path to the image on ipfs</returns>
1 reference
public string GetImage(string image)
{
    //replace image url to make a proper 1
    var result = image.Replace("ipfs://", "https://ipfs.io/ipfs/");

    return result;
}
```

Data Structures

To facilitate the transfer of the metadata, we require the following data structures:

```
7 references
public class NftDnaResponse
{
    [JsonProperty("dna")]
    0 references
    public string dna { get; set; }
    [JsonProperty("name")]
    0 references
    public string name { get; set; }
    [JsonProperty("description")]
    0 references
    public string description { get; set; }
    [JsonProperty("image")]
    2 references
    public string image { get; set; }
    [JsonProperty("imageHash")]
    0 references
    public string imageHash { get; set; }
    [JsonProperty("edition")]
    0 references
    public int edition { get; set; }
    [JsonProperty("date")]
    0 references
    public long date { get; set; }
    [JsonProperty("attributes")]
    0 references
    public Attribute[] attributes { get; set; }
    [JsonProperty("compiler")]
    0 references
    public string compiler { get; set; }
    [JsonProperty("schema")]
    0 references
    public string schema { get; set; }
    [JsonProperty("nftType")]
    0 references
    public string nftType { get; set; }
    [JsonProperty("collection")]
    1 reference
    public Collection collection { get; set; }
}
```

```
1 reference
public class Collection
{
    [JsonProperty("name")]
    1 reference
    public string name { get; set; }
    [JsonProperty("description")]
    0 references
    public string description { get; set; }
}

1 reference
public class Attribute
{
    [JsonProperty("trait_type")]
    0 references
    public string trait_type { get; set; }
    [JsonProperty("value")]
    0 references
    public string value { get; set; }
}
```

Callbacks - Authentication

Here we set up an endpoint for the XUMM Api to call with the payloads we request.

Methods

- The GetPayloadByUUID call gets the payload for this user by UUID and saves the information into the database

```
/// <summary>
/// Gets the payload and stores it into the database.
/// </summary>
/// <param name="payloaduuid"></param>
2 references
public void  GetPayloadByUUID(string payloaduuid)
{

    //call the xumm endpoint with payloaduuidv4
    var client = new RestClient(_xummConnections.RestClientAddress);
    var request = new RestRequest("/platform/payload/"+payloaduuid,Method.Get);
    request.AddHeader("accept", "application/json");
    request.AddHeader("X-API-Key", _xummConnections.ApiKey);
    request.AddHeader("X-API-Secret", _xummConnections.ApiSecret);

    //get the response and parse the result
    var response = client.Get(request);
    var result = JsonConvert.DeserializeObject<PayloadDetails>(response.Content.ToString());

    //save status to DB
    var saveData = new PayloadStatus
    {
        Account = result.response.account,
        PayloadUUid = result.meta.uuid,
        Signed = result.meta.signed,
        IsActive = true,
        UpdatedOn = DateTime.Now,

    };
    _sqlRepository.Create(saveData);

}
```

Data Structures

To facilitate the communication of this data, we need the below data structures. Whilst we didn't need all the fields, we implemented them for now incase you want to extend that later:

```
1 reference
public class PayloadDetails
{
    2 references
    public Meta meta { get; set; }
    0 references
    public Application application { get; set; }
    0 references
    public Payload payload { get; set; }
    1 reference
    public Response response { get; set; }
    0 references
    public Custom_Meta custom_meta { get; set; }
}
```

```
1 reference
public class Custom_Meta
{
    0 references
    public object identifier { get; set; }
    0 references
    public object blob { get; set; }
    0 references
    public string instruction { get; set; }
}
```

```
1 reference
public class Application
{
    0 references
    public string name { get; set; }
    0 references
    public string description { get; set; }
    0 references
    public int disabled { get; set; }
    0 references
    public string uuidv4 { get; set; }
    0 references
    public string icon_url { get; set; }
    0 references
    public string issued_user_token { get; set; }
}

1 reference
public class Payload
{
    0 references
    public string tx_type { get; set; }
    0 references
    public string tx_destination { get; set; }
    0 references
    public object tx_destination_tag { get; set; }
    0 references
    public Request_Json request_json { get; set; }
    0 references
    public string origintype { get; set; }
    0 references
    public string signmethod { get; set; }
    0 references
    public DateTime created_at { get; set; }
    0 references
    public DateTime expires_at { get; set; }
    0 references
    public int expires_in_seconds { get; set; }
}
```

```
1 reference
public class Request_Json
{
    0 references
    public string TransactionType { get; set; }
    0 references
    public bool SignIn { get; set; }
}

1 reference
public class Response
{
    0 references
    public string hex { get; set; }
    0 references
    public string txid { get; set; }
    0 references
    public DateTime resolved_at { get; set; }
    0 references
    public string dispatched_to { get; set; }
    0 references
    public string dispatched_nodetype { get; set; }
    0 references
    public string dispatched_result { get; set; }
    0 references
    public bool dispatched_to_node { get; set; }
    0 references
    public string environment_nodeuri { get; set; }
    0 references
    public string environment_nodetype { get; set; }
    0 references
    public string multisign_account { get; set; }
    1 reference
    public string account { get; set; }
    0 references
    public string signer { get; set; }
    0 references
    public string user { get; set; }
}
```

```
1 reference
public class Meta
{
    0 references
    public bool exists { get; set; }
    1 reference
    public string uuid { get; set; }
    0 references
    public bool multisign { get; set; }
    0 references
    public bool submit { get; set; }
    0 references
    public object pathfinding { get; set; }
    0 references
    public string destination { get; set; }
    0 references
    public string resolved_destination { get; set; }
    0 references
    public bool resolved { get; set; }
    1 reference
    public bool signed { get; set; }
    0 references
    public bool cancelled { get; set; }
    0 references
    public bool expired { get; set; }
    0 references
    public bool pushed { get; set; }
    0 references
    public bool app_opened { get; set; }
    0 references
    public bool opened_by_deeplink { get; set; }
    0 references
    public object return_url_app { get; set; }
    0 references
    public object return_url_web { get; set; }
    0 references
    public bool is_xapp { get; set; }
    0 references
    public object signers { get; set; }
}
```

Callbacks - Status

Here we check if a callback has been received and send back the account number associated with the payload. This pulls data from sql by payloadUUID

Methods

- The CheckStatus method checks the db to see if the payload has been signed, then returns the account

```
/// <summary>
/// Check the status for the specific  UUID
/// </summary>
/// <param name="payloadUUID"></param>
/// <returns></returns>
2 references
public string CheckStatus(string payloadUUID)
{
    //easy exit
    if (payloadUUID == null)
        return "" ;

    //check if there is an account stored with payloadUUID
    var statusModel = _sqlRepository.GetAll().Where(x => x.PayloadUUid == payloadUUID).FirstOrDefault();
    if (statusModel != null && statusModel.Signed && statusModel.IsActive)
    {
        return statusModel.Account;
    }

    //catch
    return "";
}
```

Accessing the API in Unity

There is a unity test project also in the Repo that shows the implementation. There is a file "XUMManager" which handles all the functionality. There is a "HttpClient" utility we also wrote to encapsulate Unity's web request functionality and make it easier to use.

Step 1 - Authentication

Here we call the API to start the sign-in process and get the login QR Code, as well as setting up the coroutine for the callback

```
/// <summary>
/// calls our custom API and gets a login QR code, as well as setting up the callback coroutine.
/// </summary>
0 references
public async void StartSignin()
{
    _qrContainer.SetActive(true);

    //call API
    var player = await HttpClient.Get<Player>("https://unityxummapi.azurewebsites.net/player/signin/500");
    //var player = await HttpClient.Get<Player>("https://localhost:7042/player/signin/500");

    //catch
    if (player == null)
    {
        Debug.LogError("Service not running or internet connection not working - please try again");
        return;
    }

    // need to assign to individual field
    Debug.Log("QR URL : " + player.qrurl.ToString());
    Debug.Log("player : " + player.ToString());

    //load that qr code and show it
    StartCoroutine(LoadQRCode(player.qrurl.ToString()));
    StartCoroutine(CheckForCallback(player.uuid));
}
```

```
/// <summary>
/// iEnum for loading the QR code
/// </summary>
/// <param name="url"></param>
/// <returns></returns>
1 reference
IEnumerator LoadQRCode(string url) {

    //send the request
    UnityWebRequest request = UnityWebRequestTexture.GetTexture(url);
    yield return request.SendWebRequest();

    //handle connection errors
    if (request.result == UnityWebRequest.Result.ConnectionError || request.result == UnityWebRequest.Result.ProtocolError)
        Debug.Log(request.error);
    else
    {
        //get hte texture
        Texture2D qrtex = ((DownloadHandlerTexture)request.downloadHandler).texture;
        if (qrtex != null)
        {
            qrImage.sprite = Sprite.Create(qrtex, new Rect(0, 0, qrtex.width, qrtex.height), new Vector2(0, 0));
        }
        //set texture
        qrImage.material.mainTexture = qrtex;

        Debug.Log(qrtex.dimension);
    }
}
```

Step 2 - The Response

Here we wait for the API to send back the data after the user has authenticated with their mobile app

```
/// <summary>
/// Checks for any callbacks for the specified UUID
/// </summary>
/// <param name="uuid"></param>
/// <returns></returns>
1 reference
private IEnumerator CheckForCallback(string uuid)
{
    //check for a specified amount of times before timing out.
    var isWaiting = true;
    while (isWaiting && _retryCounter<_maxRetriesForXUMM)
    {
        //keep track of retries
        _retryCounter++;

        //send the request
        UnityWebRequest request = UnityWebRequest.Get($"https://unityxummapi.azurewebsites.net/player/getstatus/{uuid}");
        //UnityWebRequest request = UnityWebRequest.Get($"https://localhost:7042/player/getstatus/{uuid}");
        yield return request.SendWebRequest();

        Debug.Log("Request was :" + request);

        //catch connection errors
        if (request.result==UnityWebRequest.Result.ConnectionError)
        {
            Debug.LogError("Network error: " + request.error);
        }
        else
        {
            //check response code is valid and that there's data
            if (request.responseCode == 200 && request.downloadHandler.data != null)
            {
                Debug.Log("Callback received!");

                //check for acc details
                var account = request.downloadHandler.text.ToString();
                if (!string.IsNullOrEmpty(account))
                {
                    //stop checking for validated account cause we found onee
                    isWaiting = false;

                    //account found, lets get the NFT's
                    UnityWebRequest nftRequest = UnityWebRequest.Get($"https://unityxummapi.azurewebsites.net/player/nfts/{account}");
                    //UnityWebRequest nftRequest = UnityWebRequest.Get($"https://localhost:7042/player/nfts/{account}");
                    yield return nftRequest.SendWebRequest();

                    Debug.Log("NFT Request data was :" + nftRequest);

                    //handle connection errors again
                    if (nftRequest.result== UnityWebRequest.Result.ConnectionError)
                    {
                        Debug.LogError("Network error: " + nftRequest.error);
                    }
                    else
```

```
                    else
                    {
                        //check if a response was received and that there's data.
                        if (nftRequest.responseCode == 200 && nftRequest.downloadHandler.data != null)
                        {
                            //deserialize
                            var nftList = JsonConvert.DeserializeObject<List<NftDnaResponse>>(nftRequest.downloadHandler.text);

                            //get all the images of the nft's
                            foreach (var nft in nftList)
                            {
                                UnityWebRequest imageResult = UnityWebRequestTexture.GetTexture(nft.image);
                                yield return imageResult.SendWebRequest();

                                //handle errors
                                if (imageResult.result == UnityWebRequest.Result.ConnectionError)
                                {
                                    Debug.LogError("Network error: " + nftRequest.error);
                                }
                                else
                                {
                                    //get the image texture from the web
                                    Texture2D imtex = ((DownloadHandlerTexture)imageResult.downloadHandler).texture;
                                    if (imtex != null)
                                    {
                                        qrImage.sprite = Sprite.Create(imtex, new Rect(0, 0, imtex.width, imtex.height), new Vector2(0, 0));
                                    }

                                    qrImage.material.mainTexture = imtex;

                                    //adding each of the images to a list
                                    imageList.Add(qrImage);

                                }
                            }
                        }

                    }
                }
            }
            else
            {
                Debug.Log("No callback received.");
            }
        }
        yield return new WaitForSeconds(15f);
    }

    if (_retryCounter >= _maxRetriesForXUMM)
    {
        isWaiting = false;
        Debug.Log("Max retries for checking - aborting");
    }
}
```