Documentation (README)

General information

The demo supports iOS, Android, macOS, and Windows platforms. This document will help you to get started quickly with the MetaMask and Infura SDKs and build your own Web3 game!

Glossary

- **Blockchain** - a system in which a record of transactions, especially those made in a cryptocurrency, is maintained across computers that are linked in a peer-to-peer network
- **NFT** - a non-fungible token is a digital asset that establishes authenticity and ownership and can be verified on a blockchain network
- **SFT** - a semi-fungible token is a type of token that is relatively new and combines the characteristics of NFTs and FTs. These tokens initially function as fungible tokens, which means that you can exchange them for other tokens of a like nature. As soon as they stop having face value, they become NFTs
- **Typed structured data hashing and signing** - a procedure for hashing and signing of typed structured data as opposed to just byte strings
- **EIP-712** - a standard that describes how data is structured, hashed, and signed
- **Public address** - or public key is *a* cryptographic code that allows a user to receive cryptocurrencies into his or her account
- **Private key** - is an alphanumeric code used in cryptography, similar to a password
- **Minting an NFT** - creating a unique token on a blockchain. Only after minting an NFT is the digital collectible stored on the blockchain
- **NFT collection** - an assortment of digital assets released by an artist (or group of artists) containing a limited number of individual NFTs. In this demo we'll be referring to NFT collections made by us (or you), specifically for the game
- **NFT drop** - the release of a non-fungible token project. A drop refers to the exact date, time, and generally the minting price of the NFT
- **Smart contracts** - programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's involvement or time loss
- **Transaction hash** - a unique identifier, similar to a receipt, that serves as proof that a transaction was validated and added to the blockchain. In many cases, a transaction hash is needed in order to locate funds
- **Transaction gas fees** - a blockchain transaction fee, paid to network validators for their services to the blockchain. Without the fees, there would be no incentive for anyone to stake their ETH and help secure the network
- **ETH** - at its core, Ethereum is a decentralized global software platform powered by blockchain technology. Though it is most commonly known for its native cryptocurrency, ether (ETH)
- **Goerli ETH** - an Ethereum test network that allows for blockchain development testing before deployment on Mainnet, the main Ethereum network

SDKs showcased

- MetaMask SDK
- Infura SDK

3rd party libraries used

All the necessary libraries are already included within the necessary SDKs. They are:

- Nethereum
- Bouncy Castle
- Ecies GO
- Ecies JS
- Native Web Socket
- Socket.io Unity
- Zxing Unity
- IPFS Client
- External Dependency Manager

Supported Unity streams

- 2021.3

Build settings used per platform

Windows

Graphics API: Direct3D11
Scripting Backend: Mono / IL2CPP
API Compatibility Level: .NET Framework / .NET Standard

MacOS

Graphics API: Metal
Scripting Backend: Mono / IL2CPP
API Compatibility Level: .NET Framework / .NET Standard

Android

Graphics API: OpenGLES3 / Vulkan
Minimum API Level: Android 7.0 (level 24)
Scripting Backend: Mono / IL2CPP
API Compatibility Level: .NET Framework / .NET Standard

Target Architecture: ARMV7, ARM64

iOS

Graphics API: Metal
Target minimum iOS version: iOS 11.0 and up
Scripting Backend: IL2CPP
API Compatibility Level: .NET Framework / .NET Standard
Target Architecture: ARM64

Game's Pipeline

An image below showcases full interaction flow between the systems for this game specifically. While it looks daunting at first, we only need to understand the "Playing Web3 Game" and "Building Web3 Game" parts in order to build our game!

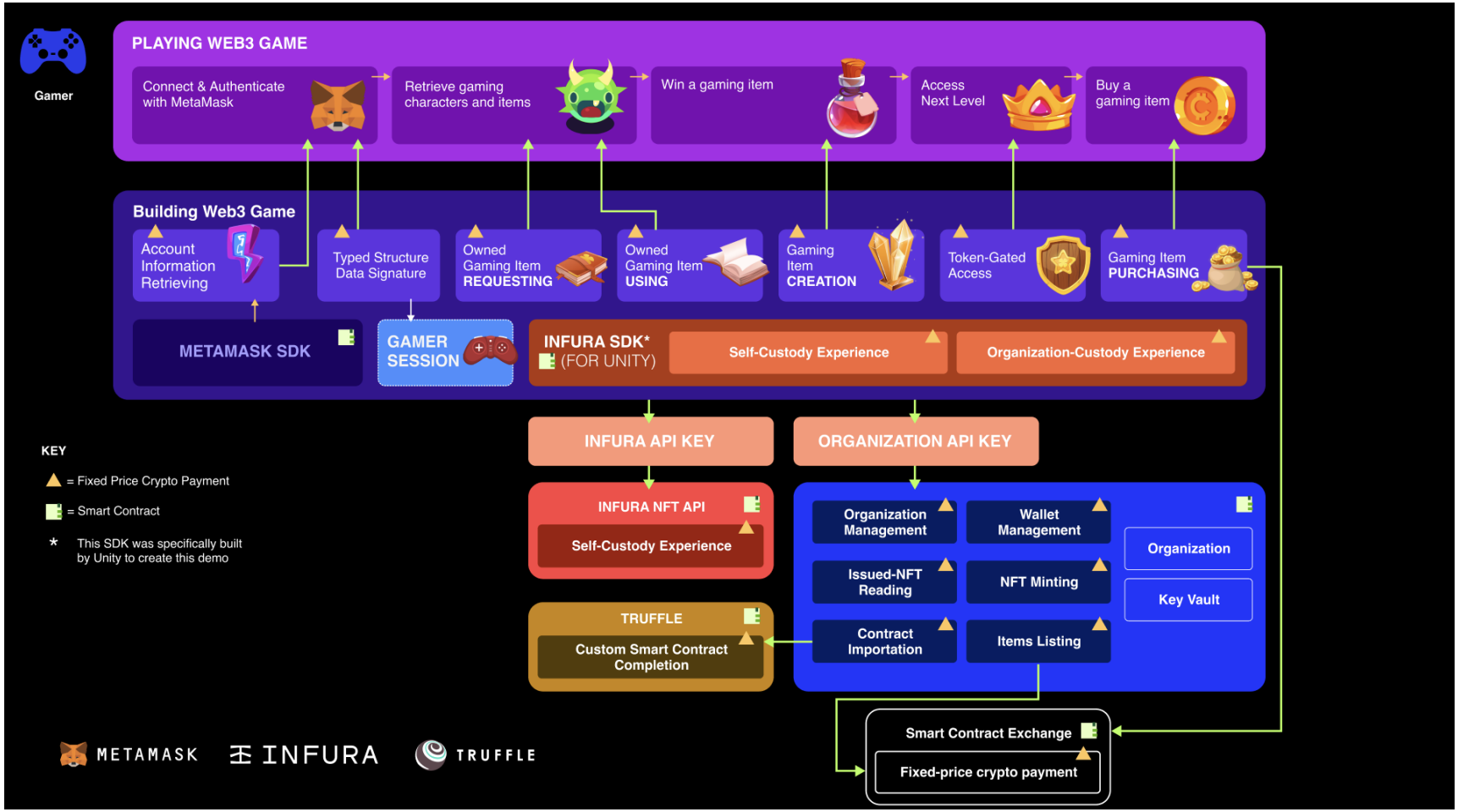
Before making your own game it's crucial to understand what your needs are. For instance, when designing a feature list for this demo started with what features would be available for an end user **playing** the game:

- Connecting & authentication
- Retrieving gaming characters and items
- Winning a gaming item
- Accessing other levels
- Buying a gaming item

Naturally, when **building** the game such features had to be implemented in some shape or form. Instead of building everything from scratch, we used already existing features that MetaMask and Infura SDKs provide to speed up the development process.

- Account information retrieving - MetaMask SDK
- Typed structure data signature/signing messages - MetaMask SDK
- Owned gaming item requesting - ConsenSys NFT platform (to make items), Infura SDK (to retrieve items)
- Owned gaming item using - the demo itself
- Gaming item creation - Infura SDK (NFT minting)
- Token-gated access - ConsenSys NFT platform (to make tokens) and Infura SDK (to retrieve tokens)
- Gaming item purchasing - ConsenSys NFT platform (to make purchasable items) and Infura SDK (to purchase items)

After figuring out what features are available to us and where - we can start building our game. 🙂


Gamer
PLAYING WEB3 GAME
Connect & Authenticate with MetaMask
Retrieve gaming characters and items
Win a gaming item
Access Next Level
Buy a gaming item
Building Web3 Game
Account Information Retrieving
Typed Structure Data Signature
Owned Gaming Item REQUESTING
Owned Gaming Item USING
Gaming Item CREATION
Token-Gated Access
Gaming Item PURCHASING
METAMASK SDK
GAMER SESSION
INFURA SDK* (FOR UNITY)
Self-Custody Experience
Organization-Custody Experience
INFURA API KEY
ORGANIZATION API KEY
INFURA NFT API
Self-Custody Experience
TRUFFLE
Custom Smart Contract Completion
Organization Management
Wallet Management
Issued-NFT Reading
NFT Minting
Contract Importation
Items Listing
Organization
Key Vault
Smart Contract Exchange
Fixed-price crypto payment
KEY
= Fixed Price Crypto Payment
= Smart Contract
* This SDK was specifically built by Unity to create this demo
METAMASK
INFURA
TRUFFLE

Connect & Authentication - MetaMask SDK

For the user's authentication, we have to build a message-signing-based authentication mechanism with a user's public address as their identifier. This consists of the following parts:

1. Sending a connect request
2. Sending a signed request

Sending a connect request

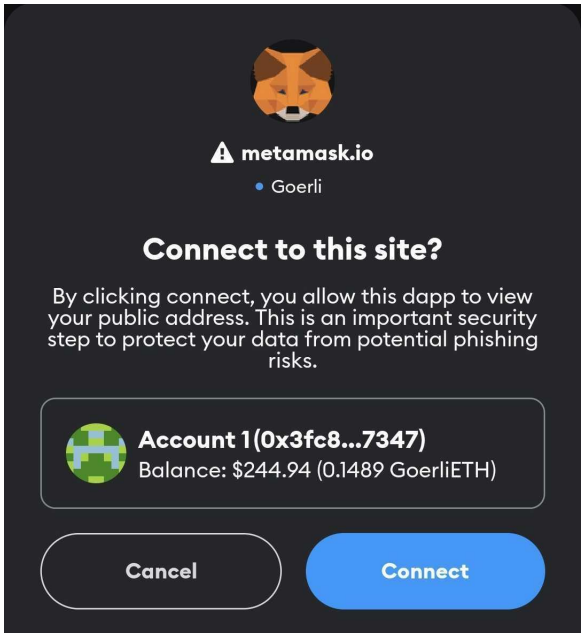
Windows / macOS

1. MetaMask SDK is instantiated
2. MetaMask SDK generates the QR code
3. The user scans the QR code
4. MetaMask SDK sends a connect request
5. User presses "Connect" in their MetaMask app

Android / iOS

1. MetaMask SDK is instantiated
2. MetaMask SDK initializes a deep link between the game and the MetaMask app
3. MetaMask SDK sends a connect request
4. User presses "Connect" in their MetaMask app

"Connect" message in the MetaMask app should look something like this.

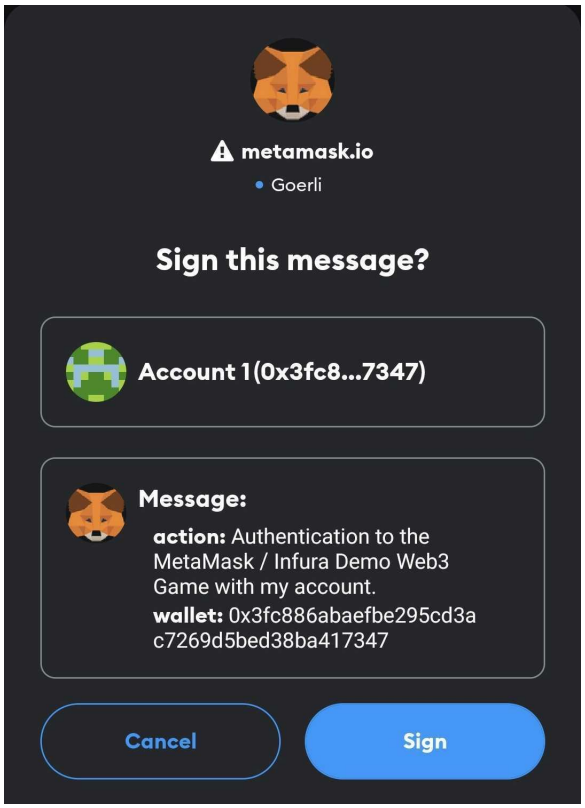


Sending a signed request

This step is more tricky because MetaMask SDK does not handle the sign request by default, though it gives the necessary functions in order to send a signing request. The message signing is crucial in order to prove ownership of an address without exposing its private key. To do so:

1. Wait for the "**WalletAuthorized**" event from the MetaMask SDK. This will be called after the user presses "Connect"
2. Send a signed request, see LoginManager.cs - **SendSignRequest()**
3. Validate a signed request, see LoginManager.cs - **ValidateSignTransaction()**

Sign request uses EIP-712 typed structured data. Within the demo, we are creating said structured data by hand and later convert it to a JSON. After sending a signed request, it should look something like this in the MetaMask app.



To validate the sign request, you need the original message and the signature. **Nethereum.Signer.EIP712** library contains multiple overloads for you to choose from, in order to recover the user's public address from the signature request.

The demo takes advantage of this one:
RecoverFromSignatureV4(string json, string signature)

Within the project, we are caching the signing message JSON before sending it out. The signature string is received from the MetaMask SDK "**TransactionResult**" event. After both of the variables are ready - we call Eip712TypedDataSigner.RecoverFromSignatureV4 and the resulting string is a public wallet address.

Retrieving gaming characters and items

The game utilizes the following types of gaming characters and items.

Type: Character

Type: Gear

Said items cannot appear in the game out of thin air, so to retrieve them, we are using the ConsenSys NFT platform in combination with the Infura SDK.

- Consensys NFT Platform is used to create NFT collections and items on the blockchain
- Infura SDK is used to retrieve said collections and items within your game

The ConsenSys NFT platform allows you to manage your NFT drops and secondary transactions using a feature-rich API that eliminates the frustrations of dealing with smart contracts. You can mint, list, display, and collect royalties and payments securely from non-fungible and semi-fungible assets with just a few simple API calls.

To create your digital asset, you need to:

1. Create an NFT collection
2. Create NFT items inside the collection

More information on how to get started with ConsenSys NFT can be found here: https://docs.consensys-nft.com/quickstarter

Requesting gaming items - Collection Items

In order to get the digital item collections, the game uses Infura SDK and its provided functionality.

1. The game links an Organization to the SDK. This will allow the SDK to interact with the organization's item collections. See NftManager.cs - **LinkNftOrganization()**
2. Using said organization object, we are calling **GetItemsFromCollection(COLLECTION_ID)**. See NftManager.cs **GetNftItemCollections()**
3. After the API call is complete, it returns a list of items that are assigned to that collection

Requesting gaming items - Owned Items

In order to get the already owned user's items, the game uses Infura SDK and its provided functionality.

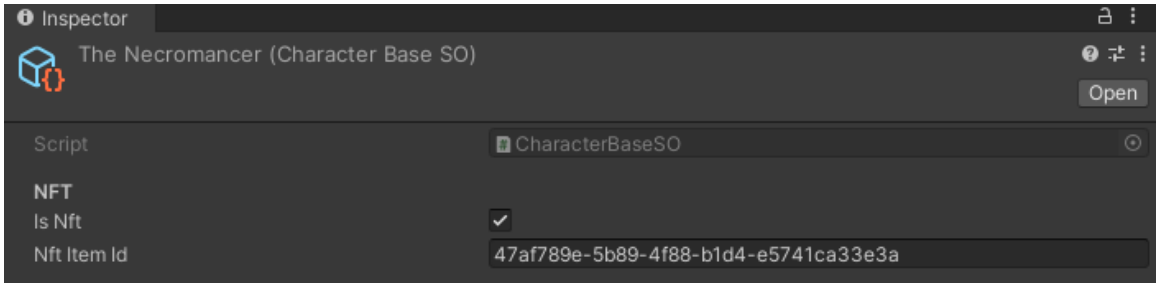
1. The game links an Organization to the SDK. This allows the SDK to interact with the Organization. See NftManager.cs - **LinkNftOrganization()**
2. Using the **Infura.API** object, we are calling **Infura.API.GetNfts(WALLET_ADDRESS)**
3. After the API call is complete, it returns a list of NFT items owned by the user

Using gaming items

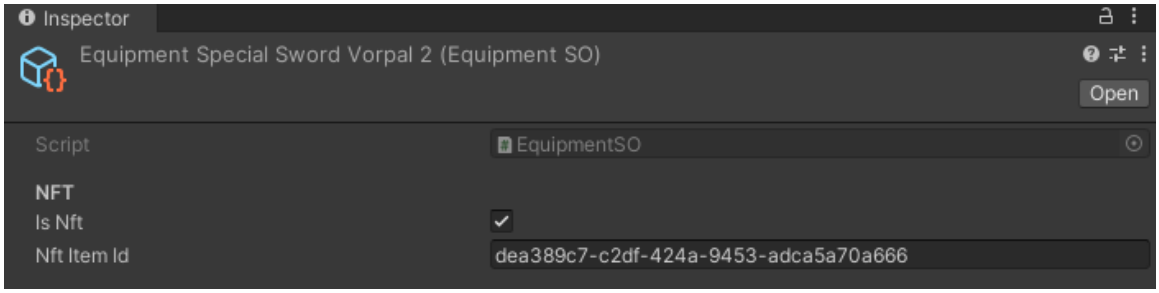
To use the items in the game, you need a mechanism to link virtual NFT items to the in-game ones. While you can totally pull everything off the blockchain (item attributes, media files, etc.) the sample game utilizes a more safe and more reliable approach. It stores all the available items in the game and enables them if they are found in the user's wallet.

All the available items can be found in the "Assets > Resources > GameData > Characters / Equipment".

Characters Scriptable Object



Equipment Scriptable Object



A ScriptableObject is a data container that you can use to save large amounts of data, independent of class instances. One of the main use cases for ScriptableObjects is to reduce your project's memory usage by avoiding copies of values.

In the Characters and Equipment ScriptableObjects, we are storing whether the item is an NFT, and its ID.

NftItem

When requesting the owner's NFT items **(Infura SDK - GetNfts())** - you are greeted with this. An array of **NftItem** type objects, containing the following information:

class NftItem (Infura SDK)
[JsonProperty("contract")] public string **Contract**; [JsonProperty("tokenId")] public BigInteger **TokenId**; [JsonProperty("supply")] public string **Supply**;

```
[JsonProperty("type")]
public TokenType Type;

[JsonProperty("tokenHash")]
public string TokenHash;

[JsonProperty("minterAddress")]
public string MinterAddress;

[JsonProperty("blockNumberMinted")]
public BigInteger? BlockNumberMinted;

[JsonProperty("transactionMinted")]
public string TransactionMinted;

[JsonProperty("createdAt")]
public DateTime? CreatedAt;

[JsonProperty("metadata")]
public Dictionary<string, object> Metadata;
```

ItemData

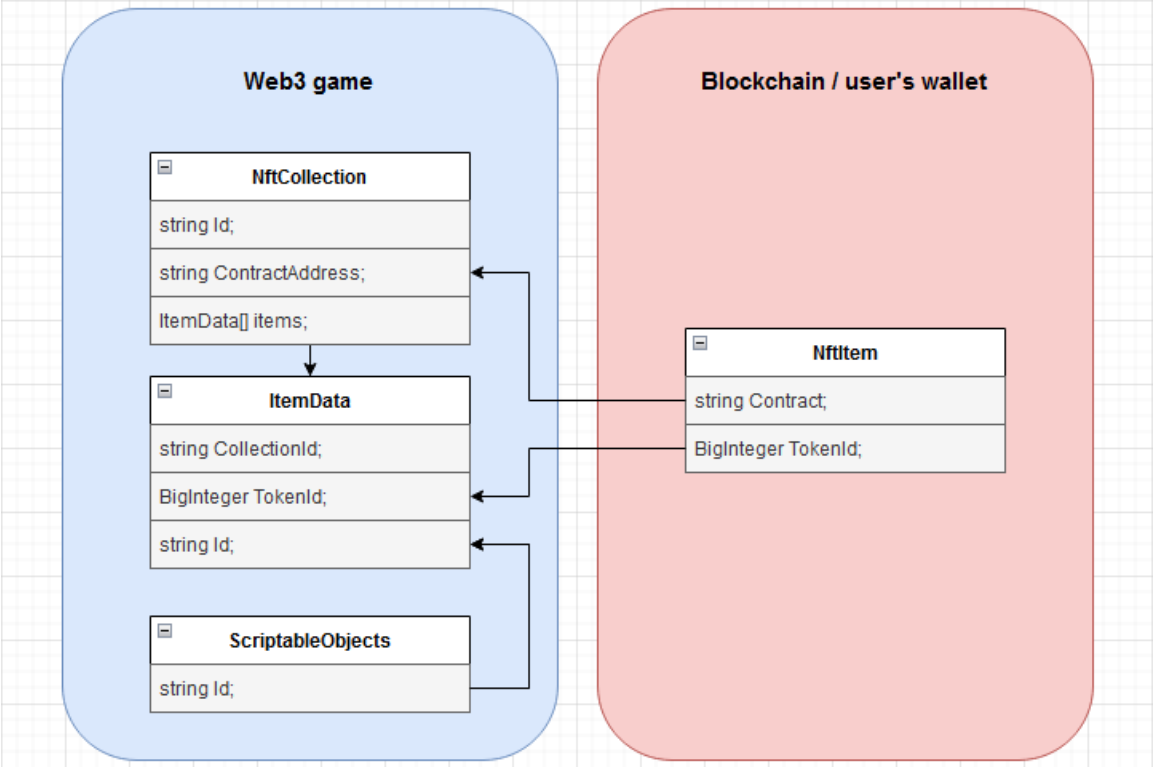
When requesting NFT items from a specific collection **(Infura SDK - GetItemsFromCollection())**, you get an array of **ItemData** type objects:

class ItemData (Infura SDK)

```
[JsonProperty("attributes")]
public Dictionary<string, string> Attributes;

[JsonProperty("collection_id")]
public string CollectionId;

[JsonProperty("token_id")]
public BigInteger  TokenId;

[JsonProperty("id")]
public string Id;

[JsonProperty("locked")]
public bool Locked;

[JsonProperty("media")]
public ItemDataMedia Media;
```

There isn't a simple common denominator between two types, thus we need to make the connection ourselves within the game.

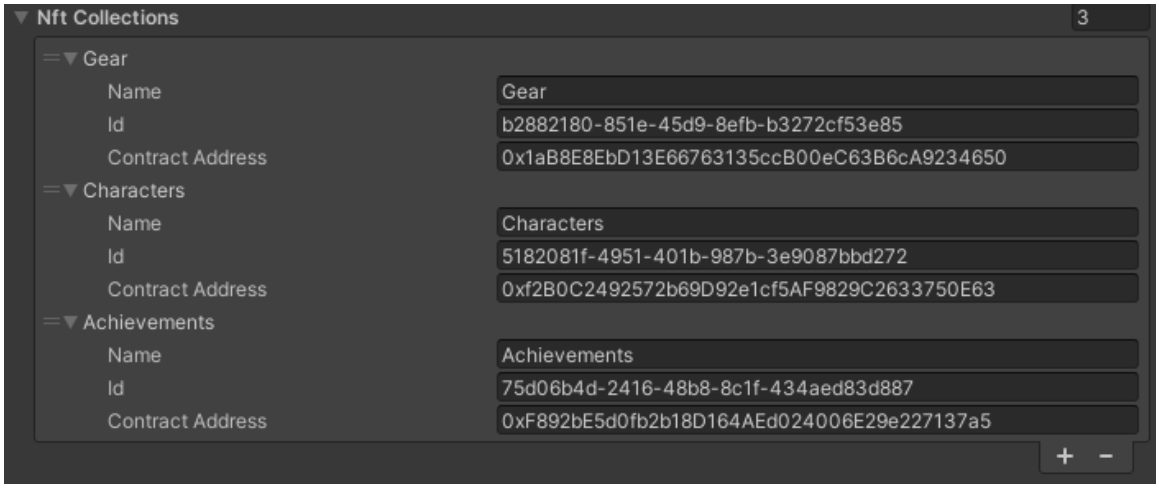
In the demo it looks something like this:



To elaborate, we are saving the collection ID, contract address plus the items it contains in a separate data object called **NftCollection**, like this:

class NftCollection (game-specific)
public string **Name**; public string **Id**; public string **ContractAddress**; private ItemData[] **items**;

NftManager GameObject (MainMenu scene)



After that everything becomes much easier. Once the **NftItem** is received from the user's wallet, we are asking if an item exists in the NFT collection's smart contract. We do so by comparing the contract address, as well as the token Id with the collection items:
NftItem.Contract <-> NftCollection.ContractAddress
NftItem.TokenId <-> ItemData.TokenId

If the item is found, it's stored in the list of owner's items (NftManager.cs - **List<OwnedItems>**) and can be easily referenced in the game.

Winning a gaming item

Once the user wins a gaming item and you'd like them to have it in a form of an NFT - the process is called minting. Minting an NFT means converting digital data into crypto collections or digital assets recorded on the blockchain.

To mint gaming items, we take advantage of Infura's SDK-provided functionality. When the user is unlocking a free item, our path to minting is this:

1. Received a request to unlock an item (i.e., the user pressed a button)
2. Finding **ItemData** equivalent of said item. See NftManager.cs - **GetItemById()**
3. Calling **ItemData.Mint(WALLET_ADDRESS)** on the ItemData type class object
4. Wait for the transaction hash - the result of the **Mint()** function

The same concept would apply to any item the customer has received in the game for free. After the transaction, the hash is returned, and the NFT is already in the user's wallet. When using test ETH (the game uses Goerli), you can access the transaction on the blockchain using this link:

https://goerli.etherscan.io/tx/TX_HASH

That said, minting can sometimes take quite a long time, up to a minute or two. In this case, it would be best to leave the item in the “waiting” state until the transaction is fully done and let the user play in the meantime.

Access the next level

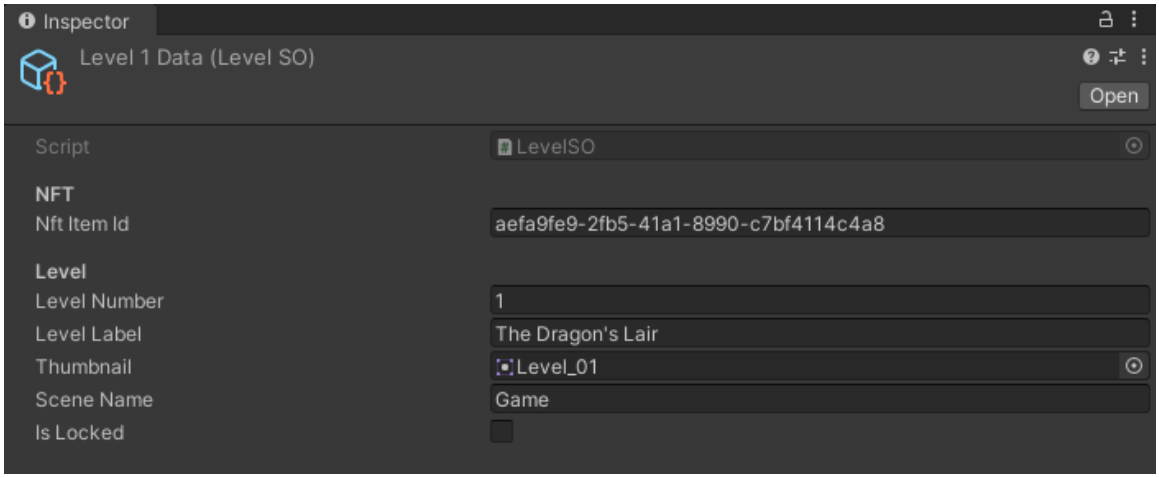
The game contains the following achievements to access the next level.

Type: Achievement

From the NFT side of things, the implementation is exactly as any other item - you need to create a collection and create an NFT item within that collection. See the section **Retrieving gaming characters and items**. The difference is the implementation within the game, i.e., how you’re consuming that NFT item.

In the game, we have created “**LevelSO**” - a scriptable object containing level information. The topmost field is an “Nft Item Id” which contains an NFT item ID.

The scriptable object can be found in “Assets > Resources > GameData > Levels”.



If this item is retrieved from the user's wallet during the login process - the user has already passed level 1 and the game needs to allow them to access level 2, and so on.

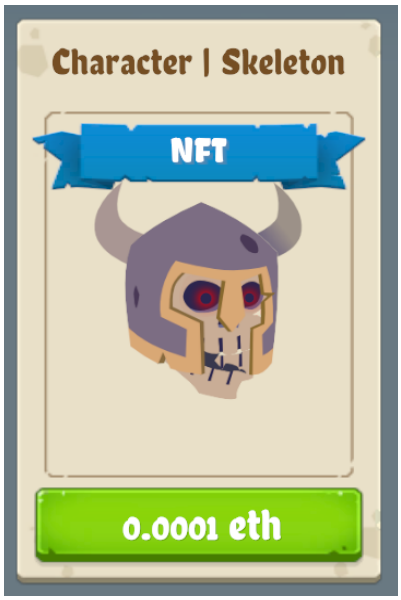
To link the "**Nft Item Id**" with its respective NFT collection, the demo uses the same linking process as it was explained in the "**Using gaming items**" section.

Buying a gaming item

Visiting the demo's store you'll find the following items. Each and every item can be bought for a specific amount of testing ETH + transaction gas fee.

Equipment

Characters



To implement the purchasing capabilities in the game, we used **Infura SDK**. To implement it, you need to:

1. Create listings for the item using the ConsenSys NFT API

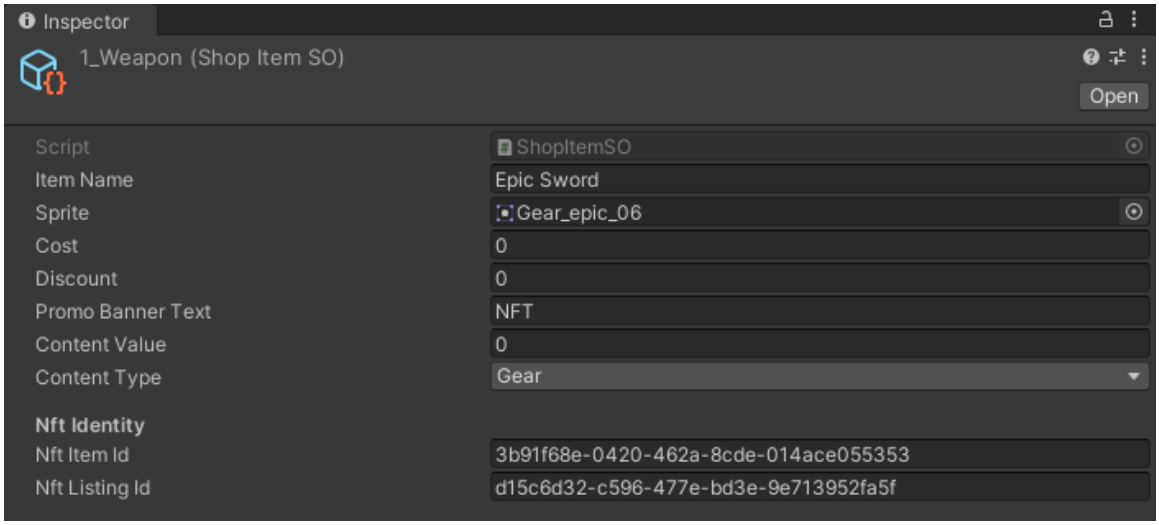
2. Create listed items within the game
3. Link items within the game with the items from ConsenSys NFT API
4. Create a purchase intent using the Infura SDK
5. Send a purchase intent using the Infura SDK

Diving deeper into the implementation, to create listings and retrieve them in the game, you need to:

1. Create a listing for the item using the ConsenSys NFT API
2. Get NFT listings using Infura SDK. The listings are organization-specific, so to get them you need to call **OrgApiClient** organization.GetListings(). See NftManager.cs - **GetNftItemListings()**

ShopItemSO

To create shop items, once again, we used ScriptableObjects. Here you can see the shop item's ScriptableObject:



It contains many fields used for the shop. But most importantly, it contains a place to host "**Nft Item Id**" and "**Nft Listing Id**", so that these items can be bought and tracked within the lifecycle of the game.

ListingOutput

When retrieving the listings within the game (**OrgApiClient** organization.**GetListings()**), the returned result is an array of type **ListingOutput**. It contains the following fields:

class ListingOutput
public Guid **CollectionId**; public string **Currency**; public DateTime **EndTime**; public Guid **Id**; public Guid **ItemId**; public int **MaxQuantityPerTx**; public List<string> **PaymentProviders**; public Object **Policy**; public string **Price**; public int **QuantityListed**; public int **QuantityRemaining**; public string **SaleType**; public DateTime **StartTime**; public string **Status**;

The most notable field in the **ListingOutput** class is “ItemId”, as it is used to determine which shop item the listing corresponds to. “**ItemId**” (Listing) should equal “**Nft Item Id**” (ShopItemSO).

Creating a purchase intent

After creating listings within the ConsenSys NFT system, respective item representations in the game, and linking them together, the next step would be to create a purchase intent. To do so:

1. Find a **ListingItem** for the specific item the user wants to buy (**ListingOutput** section)
2. Create a mint voucher purchase intent using the Infura SDK. To do so, we need to call **OrgApiClient** organization.**CreateMintVoucherPurchaseIntent(LISTING, WALLET_ADDRESS)**, providing the previously found listing and the user’s wallet address. See NftManager.cs - **PurchaseItem()**

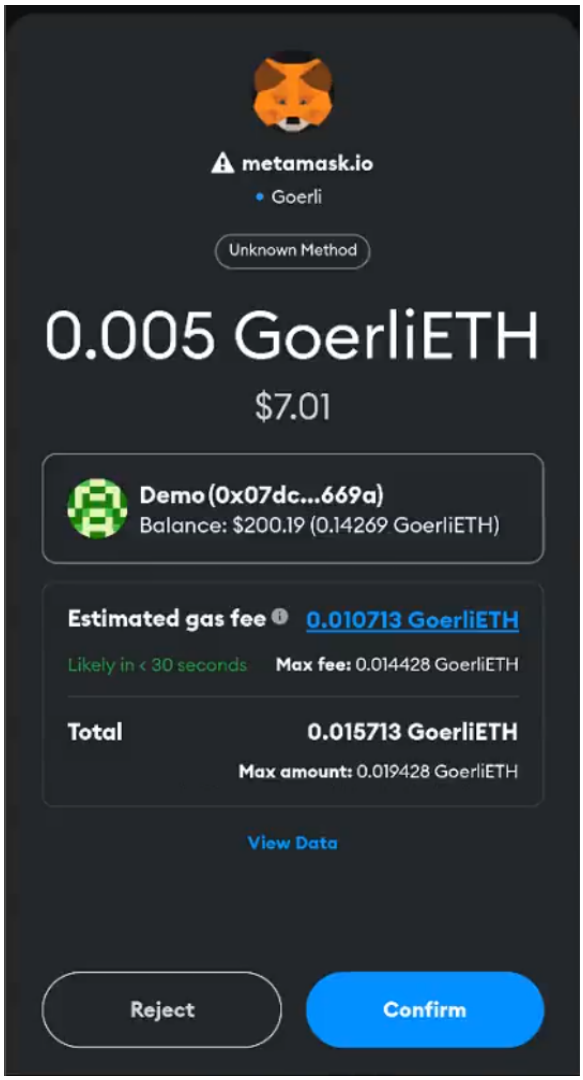
The demo, to be specific, is creating a mint voucher, but you can read more about different payment providers here.

Sending a purchase intent

Finally, when the purchase intent is created, it’s ready to be sent. To do so:

1. Create a new **Web3** object. You can do so by calling **MetaMask.CreateWeb3()**.
2. Send a purchase intent using the Infura SDK. To do so, we need to call **OrgApiClient** organization.**SendPurchaseIntent(WEB3, PURCHASE_INTENT)**, providing the previously created purchase intent and just created **Web3** object. See NftManager.cs - **PurchaseItem()**

If everything is done successfully, there shouldn't be any errors from the Infura SDK. But most importantly - the user should be able to see the purchase request in their MetaMask application.



As you can see, we were trying to buy the "Epic Axe" for **0.005** Goerli ETH (testing ETH). After adding gas fees, the final price goes up to **0.015713** Goerli ETH. At this point, the user can confirm or reject the purchase. If the user presses "Confirm" – we have to mint an item for them (see the "**Minting gaming items**" section), otherwise, cancel the purchase.

That's it!

By now, you should have a general understanding of how to:

- Authenticate users for your game using the Metamask SDK
- Create NFTs for items, levels, characters, etc. on the blockchain using the ConsenSys NFT API
- Retrieve NFTs created on the blockchain and link them to objects used within the game using the Infura SDK
- Issue NFTs to players within the game using the Infura SDK
- Implement transactions for purchasable NFTs within the game using the Infura SDK

Known Issues

1. **[Mobile] Hard crash when sending purchase intent messages to the MetaMask app**

   No known workarounds.

2. **[Unity Editor] An error in build when using IL2CPP and IL2CPP Code Generation “Faster Runtime” build settings**

   This is a Unity bug. Instead, you should build with the “Faster (smaller) builds” setting until it’s resolved on the Unity Editor’s side.

The sample used

Dragon Crashers - 2D Sample Project

https://assetstore.unity.com/packages/essentials/tutorial-projects/dragon-crashers-2d-sample-project-190721

https://assetstore.unity.com/packages/essentials/tutorial-projects/ui-toolkit-sample-dragon-crashers-231178

Dragon Crashers is an official sample project that showcases Unity's native suite of 2D tools and graphics technology, 2D Renderer. The gameplay is a vertical slice of a side-scrolling Idle RPG.