Software Specification Document: BRAGGART

A social marketplace where users can login/signup with ZilPay to mint and auction NFTs, create and interact with posts showcasing external NFTs from multiple other blockchains, follow each-other and display galleries of posts in 3D WebXR views.

The (dApp) should fun, social and gamified with proper incentive mechanisms in place to keep user returning to the site. The user interface should be bright and layman-friendly with a hint of technicality in mono-spaced labels and buttons' designs.

FEATURES OF BRAGGART:

Platform to Showcase—https://gitcoin.co/issue/Zilliqa/Zilliqa/2546/100025673

- Social networking that allows creators to showcase NFTs from other blockchains as ‘posts’ on the platform that others can like, re-post, comment on their work and follow their profile!
- Creators can organize multiple ‘posts’ as ‘galleries’ which are shareable/viewable with their own unique link!
- Creators can add multiple blockchain account/addresses to their user profile as external links mapped by type & viewable by their respective Blockchain explorers.
- User profile is shareable via a URL link

Patreon Powered—https://gitcoin.co/issue/Zilliqa/Zilliqa/2544/100025671

- Unlockable content, Patreon-style for NFTs where creators can create tiers containing hidden exclusive content accessible by purchase of tier. The duration of access for this exclusive content varies, from days to weeks to forever.

Innovate: Build Anything You Want—https://gitcoin.co/issue/Zilliqa/Zilliqa/2547/100025674

- Built-in *Art Maker Wizard* modal to the dApp that allows creators to dynamically combine content like images, gif and text on the fly into an image blob that imports into the NFT listing creation form and is uploaded to IPFS when minted.

    - PyTorch-supported AI art generator service **(coming soon!)**
- Built-in *Open World Showcase* – When NFTs are minted or ‘Posts/Galleries’ are published on the platform, they are viewable in 3D view space by WebXR-supported libraries like three.js, aframe.js and assets from Google Blocks.

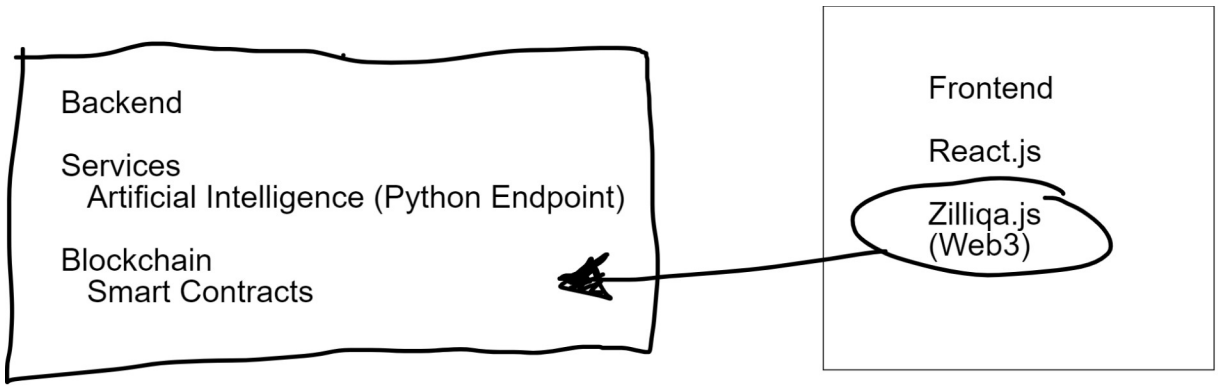
Customizable Auction Parameters—https://gitcoin.co/issue/Zilliqa/Zilliqa/2545/100025672

- Easily create different type of listings to mint your NFT, with no extra programming of smart contract code required.
- Supported listings are: fixed price, declining-price (dutch auction), highest-bid (english auction), increasing-price (bond curve), bundled, and open-for-offers.
    - Bundled listing: NFT batches with different prices: bundle existing assets together with one price for the bundle.
    - Increased-price listing: price increase linked to a bonding curve, increases over time based on total token supply.
        - *Bond Curve Formula*: y = m((1+.75)^log^2x) + b
    - Highest-bid listing: eBay-style auctions that allow you to pick a reserve price, set a time duration, and whoever puts in the highest bid will win the item as long as it’s above your reserve.
    - Declining-price listing: choose a start price, an end price, and a duration. The price will decline over time until it reaches the end, allowing you to appeal to eager buyers and sell for above-market prices.
    - Fixed price listing: Pick a price, that's it.
    - Open for offers, even if not on sell
- Auction parameters are easily configurable in the dApp’s UI as mutable fields in the corresponding ZRC-1 NFT smart contract.

SUBMISSION GRADING RUBRIC:

Product Features, User Experience [UI/UX], Incentive Mechanisms, Video Walk-through & Technical Architecture Documentation

TECHNICALS:



Zilliqa Blockchain, Scilla Smart Contract Language, Zilliqa.js JSON-RPC / React.js / Three.js / Aframe.js / GunDB / IPFS & Pinata.cloud

📎 = https://scilla.readthedocs.io/en/latest/index.html

📎 = https://ide.zilliqa.com/#/

📎 = https://bubble.io/blog/build-patreon-clone-no-code/

📎 = https://en.wiktionary.org/wiki/{WORD}#English

📎 = ZRC-1 Standard https://github.com/Zilliqa/ZRC/blob/master/zrcs/zrc-1.md

INSPIRATIONAL REFERENCES:

NFTS

0. OpenSea
1. Rarible
2. Foundation.app
3. Zilliqa.Mintable.app
4. Davinci.gallery
5. Zora.co
6. YellowHeart
7. Async.art
8. SuperRare
9. Mintbase
10. Tokentrove
11. SoRare
12. Enjin.io

Marketplaces & Galleries

0. Eldorado.gg
1. Sellfy
2. Gumroad
3. MetaGrail
4. Nifty Gateway
5. MakersPlace
6. KnownOrigin
7. Cargo Build
8. BlockParty
9. Ark.gallery
10. Ello.co
11. Lazy.com
12. TryShowtime.com
13. Cent.co ☆

Payment Methods

0. ZilPay
1. MetaMask
2. WalletConnect

Membership Sites

0. Patreon
1. Collab.land
2. BitClout
3. Ko-Fi
4. Memberful
5. SubscribeStar

Data

0. Non-fungible
1. NFTBank
2. Forefront.news

Sign In

0. Bitski
1. Fortmatic
2. Flow

3D

0. Hologram.cool
1. Spline.design
2. Decentraland
3. Cryptovoxels
4. SketchFab
5. Google Blocks

INFORMATION ON NFTS:

1. Can only have one official owner at a time and are not interchangeable for other assets because they have unique properties, making them non-fungible.

2. Creators of the NFT can retain ownership rights and claim resale percentage directly.
3. Each token minted has a unique identifier recorded by a smart contract
4. Creator of the NFT can determine how many replicas exist, and how many are minted to make them rare or collectible.
5. Royalties are manual and lack accuracy. If they we're programmed by the smart contract and automatic, they'd make things easier for the creators.
0.The creator's address is part of the token's metadata.
1.Tokens have different standards, and NFTs must be built on a specific standard. For instance, some NFTs on Ethereum are built with the ERC-721 standard for art and the ERC-1155 standard for gaming.
2.Social tokens are used as community tokens to have ownership and membership in a Creator's creation. These token holders get access to exclusive content.
3.Can create collections (mapped arrays of Items) for individual NFTs.
4.Token metadata can be stored off-chain by a method called `tokenURI` that tells applications where to find the metadata for a given item/smart contract - IPFS.

ORDER OF BOUNTY COMPLETED

::= Create/Auction Parameters & Patreon -> Showcase -> Innovate

dApp UI Features

• Navbar:

Create Button: clicking this allows the user to begin an interface wizard for creating a new collectible on the platform, uploading their content (through supported extension handling) and customize their digital asset offering [NFT] with properties, unlockable content, subscriptions and

stats. Afterwards, a price and type of listing can be set [auctions, fixed price, declining-price, etc].

•Types of NFTs:
•Collectibles
•Virtual World Items
•Utility (Membership Subscriptions)
•Music
•Art
•Sports and Trading Cards
•Connect Wallet: navigates a user to a splash page that allows a user to connect their wallet provider to their platform profile or create a new wallet. *}= /connect
•Supported Wallets: MetaMask, WalletConnect?
•Marketplace: a grid-based card-style view of all the NFTs on the platform *}= /explore
•Actions
•Like property allows you to favorite a digital asset
•Comment property allows you to add a text field to the text thread of a digital asset
•Subscribe as a member to a digital asset (NFT)
•Item View Page: view an artwork or digital asset with a unique UUID code. This displays:
•Price in USD fiat / Blockchain Value
•Seller Info [name, profile link]
•Seller Rating
•Item Description
•Item Name Label
•Purchase button
•Property Tabs:
•Copyright Transferred?
•Downloadable file?
•Resellable? [Secondary Trading Mechanism*]
•In Collection? # of # in Collection?
•Claimable?
•Item Metadata: each digital asset/NFT is it's own smart contract, with it's own metadata, including the link to the original image storage link, original uploaded wallet address, the contract address of the digital asset/NFT, name of the token, token ID, filekey, category, and other fields which are either URLs, text fields or boolean values.
•User Profile:
•User Settings are directly associated with the connected wallet's Zilliqa address via ZilPay, which is required to login into the platform.
•User Interface
•Header Image, user-uploaded and displayed as 'Header-Img'
•Profile Picture, user-uploaded and displayed as 'Pfp-Img'
•Name Field, user-created and displayed as 'Full Name Label'

•Wallet Address Field, taken from wallet and displayed as 'Wallet Address Label' under 'Full Name Label'
•Biography Field, user-created and displayed as 'Bio Text'
•Follow Button, database-association 'many-to-many' relation.
•Metadata tabs for user profile: 'Following', 'Followers', 'Created', 'Owned', 'Subscriptions', 'Liked',
•Activity Portal: showcase what you and other users are doing on Braggart in real-time.
•Notification
•Title of Asset
•Amount purchased, by who, for #, from who
•Timestamp date
•Link to blockchain explorer's transaction record
•Notifications Center
•Pop-in Alert Notifications, located in the top-left, top-right, bottom-left and bottom-right corners of the screen.
•Search Bar: powered by Javascript that sifts through public blockchain data for metadata that associates NFTs to the platform, this allows users to search for items, accounts/wallet addresses and subscriptions.
•Stats Leaderboard: the gamification of the platform that allows NFTs to be ranked by stats like price, # owners, # of memberships, # of views, # of likes, etc.

Software Specification

0.GENERAL SEGMENT

> Basic functionalities of the web app platform that enhance the experience, related to but not directly correlated to any particular bounty's required smart contract definition

0.Flows
•ONBOARDING: User receives a popover modal that prompts them to enter their chosen username to sign up, and connects directly to ZilPay in a one-click flow process.
1.X
.
::

0.PATREON SEGMENT

> Leveled, purchasable tiers/layers of access to NFTs. Creators can detail unlockable and assign a price to each group/collection/tier of unlockables. Each person that purchases a unlockable tier is considered a 'champion/member'.

0.Requirements
•Users can create tiers when uploading a NFT.
•Relative URL example: `/{@username:wallet_address}/nft/{UUID}#tiers`

•When a tier is purchased by a wallet address, that address is added to a map of addresses that have access to that tier's exclusive content. Otherwise, it should be hidden to the general public.
•Exclusive content could be: URL to a private Discord Server, one-to-one chats, unlocking private videos, advanced notifications on the Creator's drops, etc.
•Each tier should be presented in a card view list, containing the `name of the tier + price of the tier + detail of exclusive content listed in bullet points + big call-to-action JOIN button` displayed on the card view.
•Access to the tier duration can last for a week, month or forever.
1.Smart Contract Definition

```
(** SUPPORT TIERS FOR NFTS CONTRACT **)
```

3.Inspirational References
•Patreon.com
•Sellfy.store
.
::
1.SHOWCASE SEGMENT

> Social platform resembling an art gallery that allows people to showcase their NFTs across the Web2.0 with a sharable profile and corresponding URL link. Requires account creation.

0.Requirements
•Each existing NFT owned by a User can be uploaded to the platform by adding the NFT's tokenAddress to an empty map array that belongs to the User, called a "post"
•A "Post" represents the abstraction of a NFT's token address.
•Post can be favorited, reposted ("bragg'd"), commented on by other Users (wallet addresses) and the amount of views on a post are recorded by the web app client and sent as a message to the smart contract.
•A Post can contain tags, used to explore categories and genres of different NFTs as well as the mimetype recorded from IPFS
•Relative URL example: `/tags/{tag_name}`
•A Post will also feature an option to input a string as 'bid auction URL || Bid On' for any external NFTs that aren't on the Zilliqa blockchain.
•Relative URL example: `/{@username::wallet_address}/post/{UUID}`
•Users have the ability to create collections of existing on-chain and off-chain NFTs recorded on the platform called "galleries"
•A "Gallery" is a collection of "Posts". It has a name label, a description and a shareable link.
•It is displayed in a horizontal card view array.
•Relative URL example: `/discover/galleries/{UUID}/view`
•To create a gallery, Users will go on their profile and press the curate a gallery button.

•A NFT uploaded directly to the platform, on the Zilliqa blockchain, is different than a post. A "Post" is a social object, whereas a "NFT" object is a transactional object able to be purchased with ZIL. Some "Posts" can be "NFTs" but not all "Posts" are "NFTs".
•Relative URL example: `/{@username::wallet_address}/nft/{UUID}`
•"NFTs" objects are alternatively called "Listing" objects.
•All "Posts", "Galleries", "Favorites" [relational record of recorded likes] and "NFTs" of a User are available via their public, shareable profile.
•Relative URL example: `/{@username::wallet_address}`
•User profiles can followed and also be rated with `0/5.0 ratings value` by the mean of a range of uInt128 inputted by people who have purchased NFTs at auction from said User.
•Guests to the website platform can explore uploaded content via the relative URL `/discover` and `/discover/{post_type}` which either list galleries, posts and NFTs together or organizes them by type.
1.Smart Contract Definition

```
(** SHOWCASE NFTS CONTRACT **)
```

3.Inspirational References
•TryShowtime.com
•Ello.co
•Davinci.gallery
•OpenSea.io
.
::
2.AUCTION SEGMENT

> Easily configurable and customizable no-code parameters for auctioning NFTs.

0.Requirements
•Users can either upload the NFT at a fixed price or create a different type of auction on creation. Each auction is considered a parameter of the NFT
•Relative URL example: `/create`
•Initial customizable auction parameters:
•Increased-price listing type
•Highest-bid-by-duration-time listing type
•Fixed-price listing type
•Declining-price listing type
•Open for offers type
•NFT bundles with one set price, different items,
•Relative URL: `/bundles/{UUID}`
•Others...to be added later
•random NFT drop amongst the buyers,

•All auction parameters are configurable on the `/create` page and correspond to a smart contract that allows wallet_addresses/Users to create NFTs.
1.Smart Contract Definition

```
(** CREATE AND AUCTION NFTS CONTRACT **)
```

3.Inspirational References
•Mintbase
•Mintable.app
•https://opensea.io/blog/guides/welcome-to-opensea/
•https://billyrennekamp.medium.com/converting-between-bancor-and-bonding-curve-price-formulas-9c11309062f5
•https://hackernoon.com/more-price-functions-for-token-bonding-curves-d42b325ca14b
.
::
3.INNOVATE SEGMENT

> Baked in meme and AI generator wizard like GIPHY for creating NFTs dynamically. WebVR-based 3D viewer for NFTs.

0.Requirements
•Meme generator's user interface would that be of a modal wizard that allows Users to upload media with the mimetype `png`, `jpg` and decorate it with captions, filters and stickers to be imported to the creation form for NFTs to create & mint for the User
•Relative URL example: `/maker#memes`
•Reference: https://giphy.com/create/gifmaker
•AI ART generator would use an existing neural network model like PyTorch to generate a piece of art with a regular mimetype in real-time and import to the creation form for NFTs to create & mint for the User
•Relative URL example: `/maker#ai`
•Reference: https://creator.nightcafe.studio/create
•3D viewer of NFT Art ("Posts", "Galleries") for a User. Supported by WebXR/WebGL libraries like Three.js, glTF and Google model-viewer.
•Relative URL example: `/{@username::wallet_address}/{type}/{UUID/3d#view}` for Posts & NFTs or `/discover/galleries/{UUID}/view/3d` for Galleries.
•Reference: https://modelviewer.dev/ || https://www.cryptovoxels.com/
1.Smart Contract Definitions

```
(** RETRIEVE NFTS/POSTS/GALLERIES FROM A USER CONTRACT **)
```

3.Inspirational References
•https://www.cryptovoxels.com/
•https://www.giphy.com

•https://thispersondoesnotexist.com/

SMART CONTRACT DEFINITIONS:

LEGEND:

```
☑ - Field Implemented



✅ - Transitions & Procedures Implemented
```

::.

Example ::= `Crowdfunding Smart Contract`

```
For an intuition of a contract layout, consider a Crowdfunding smart contract. The
goal of the

contract is, as the name implies, to collect donations aiming for a certain goal
by a specified deadline,

given as a łmaximalž block number in the underlying blockchain.4

It should then allow potential

backers to donate certain amounts of funds, making records of those donations. If
the goal is

reached by the deadline, the owner of the contract, specified upfront via its
account address, should

be able to extract the funds, at which point the fulfillment of their obligations
to the backers is no

longer a concern that could be addressed via the blockchain. If the goal is not
reached before the

deadline, each backer should be able to claim their donation back.
```

:::

`CREATE/AUCTION Smart Contract` && `PATREON Smart Contract`

> Easily create NFTs with customizable auction parameters and associate detail properties.

The goal of this smart contract, is to create/mint non-fungible tokens containing content and metadata for a specified wallet address as well as make that non-fungible token auctionable by providing customizable parameters an end client application can easily configure.

It should allow potential users to upload their media to IPFS and associate said media to the metadata of the NFT's tokenId via tokenURI.

Auction Parameter Types

Auction bids should be ordered from highest-to-lowest, with each higher offer pushed to the array of bids, invalidating the lower-priced bids. Only one bid can be active at a time, and the way to push a bid down or make the current isActive boolean field value of a bid => False, is to input a higher offer field in a newly submitted bid represented by a uInt128 value.

For the English-style auction parameter, Creators of the NFT can set a reserve price value that must be met in order for the NFT to be sold to the highest bid in the auction. If the reserve is not met, the bid ends and the NFT is marked as notListed => True, which is a boolean value that allows the NFT to be sold or not.

This will let the Creator retain ownership of the NFT if it is not sold.

When purchased, proceeding funds from the NFT purchase should be sent to the Creator's wallet address, with no current escrow in place by the platform. A 2.5% fee will be taken by the Braggart platform and a royalty parameter should exist for Creators' wallet address to receive royalties (default: 10%) from the future sales of the NFT by the new owner address.

Contract Parameters {Immutable: cannot be changed once deployed}

= Creator will determine the wallet address of the Creator of the NFT, this is the initial account that mints the NFT and lists it on the platform. ☑

The immutable Creator value will be stored in the contract definition as a `ByStr20` when inputted on creation, and the value will represent the creator account's wallet address:

```
// ByStr20 - zil wallet address

(

    creator: ByStr20

)
```

= Symbol will be the symbol for the ZRC-1 token that the NFT becomes when minted/permanently published to the Zilliqa blockchain. ☑

The immutable Symbol value will be stored in the contract definition as a `String` when inputted on creation, and the value will represent the NFT's token symbol:

```
// String - alphanumeric value used to represent the NFT's token

(

    symbol: String

)
```

= Auction Type will determine the type of auction parameter the NFT will have, which is further customizable via fields & transitions. ☑

The immutable Auction Type value will be stored in the contract definition as a `Uint32` when inputted on creation, and the values will be commented in the form of a enum for the end clients to display:

```
// 0: Fixed Price, 1: English Bid, 2: Fixed Price, 3: Declining Price, 4: Open for Offers, 5: Batches With Different Prices; "Bundles"

(

    auctionType: Uint32

)
```

= Platform Address will be the address of the platform the NFT is minted on. In this case, it will be Braggart's platform smart contract that associates all newly-minted NFTs on the platform to Braggart. ☑

The immutable Platform Address value will be stored in the contract definition as a `ByStr20` when inputted on creation, and the value will represent the Braggart platform's smart contract address:

```
// ByStr20 - zil smart contract address for Braggart

(
```

```
    platformAddress: ByStr20

)
```

:::

Fields {Mutable: can be changed once deployed via transition}

= NFTs

NFT Item Properties

•Name ☑
•Description ☑
•Unlockable Content? (Patreon-Style Tiers) [Boolean] ☑
•Add Tier to Item Transition: Key (Name), Value (Content), Unlock Price -> {Map Type}

```
// Tier -> Map tk tv = Map String (Pair (String) (Uint128))



struct NFT_Item {

  String item_name ✅ // Field implemented in Contract

  String description ✅ // Field implemented in Contract

  Bool unlockableContent? ✅ // Field implemented in Contract

  struct Tier {

    Uint32 id;

    String name;

    String content;

    Uint128 unlock_price;

    Uint128 access_time_duration; // how long purchaser can see
```

```
  } ☑

  Uint256 total_supply ✅ // Field implemented in Contract

  String token_uri ❓ // dont know how this will be handled in
contract

  Uint128 initial_fixed_price ✅ // Field implemented in Contract

  struct Auction {

    //...

   }

}
```

•Total Supply Amount: number of NFT items to sold
•TokenURI (Metadata)
•Image URL
•Attributes
•Fixed Price
•= Auction Configuration
•increased-price listing: choose a start price, price increases gradually and is based on a bonding curve
•highest-bid: eBay-style auctions that allow you to pick a reserve price (new!), minimum bid price, set a time duration (7d range), and whoever puts in the highest bid will win the item as long as it's above your reserve. This is called a English auction.
•declining-price listing: choose a start price, an end price, and a duration. The price will decline over time until it reaches the end, allowing you to appeal to eager buyers and sell for above-market prices. This is called a Dutch auction.
•fixed price listing: Pick a price that never changes. This is Amazon style.
•open for offers, even if not on sell. This is buying-and-selling-group style.

```
// Based on a contract's auctionType, certain features of an Auction
map are disabled and enabled based on the auctionType's
currentValue. This allows the NFT item's auction parameters to be
customizable without code. This is aided by well-implemented
procedures and transitions
```

```
/****** CUSTOMIZABLE AUCTION PARAMETER LOGIC *******/



/***** Fields hold values, transitions handle implementation of
listing types based on mutable fields' "custom auction parameters"
****/



If auctionType = open for offers, then all mutuable fields/auction
parameters in Auction map are disabled except the boolean value
"lockedToOffers?", and initial fixed price is hidden from viewer. A
'offer' button is shown instead, and the transition for addNewOffer
is available.



If auctionType = fixed price listing, only the mutable field for
initial_fixed_price is shown and all other auction parameters in the
Auction map are disabled/hidden.



If auctionType = highest bid, mutable fields like reserve price,
minimum price, current_price and time duration are enabled in the
Auction map as well as boolean values like "includeMinimumPrice?",
"currentPriceIsShown?", "biddingIsAllowed?" "includeReservePrice?"
are enabled from Option None [False] to Some [True]. This allows for
bidding and a transition manages the bid map based on this
auctionType. The highest bid value of the map is then shown as the
current price.



If auctionType = declining price listing, mutable fields like start
price, end price, current price and time duration are enabled in the
Auction map as well as boolean values like "includeEndingPrice?",
"includeStartPrice?" and "currentPriceIsShown?" are enabled. This
does not allow for bidding as the price is set high then declines
```

```
automatically over time. currentPrice mutable field displays the
current price.



If auctionType = increased price listing, mutable fields like start
price and current price are enabled in the Auction Map, as well as
boolean values like "includeStartPrice?", "currentPriceIsShown?" &
"increasePriceOnBondCurve?" are enabled. A procedure and transition
handle the gradual increase of price over time using a bonding curve
formula and continually check/update price. The number of
total_supply of the NFT


(amount sold) factors in with the startPrice to reach a gradual
increase in price as time passes. This updates the currentPrice
mutable field with the new calculated value.



>> Bonding Curve Formula __ square root curve :: Continous Price =
(startPrice ^ 1/2) * Total Token Supply


OR new_total = ((start_price ^ 1/2) * total_supply) / DECIMALS



^>> Adds division of given DECIMALS notation to update price.



🔗 Reference: https://medium.com/linum-labs/intro-to-bonding-curves-
and-shapes-bf326bc4e11a



*****/



// Scilla equivalent: Map tk tv = Map
```

```
struct Auction { // Customizable Auction Parameters


  Uint128 access_time_duration;


  Uint128 start_price;

  Uint128 reserve_price;

  Uint128 minimum_price;

  Uint128 current_price;


    Bool lockedToOffers?;

  Bool biddingIsAllowed?;


  Bool includeMinimumPrice?;

  Bool includeReservePrice?;


  Bool includeStartPrice?;

  Bool includeEndingPrice?;


  Bool currentPriceIsShown?;
```

```
  Bool increasePriceOnBondCurve?;

}
```

-

:::

= Name needs to display the name of the NFT item.

The mutable Name field will store a alphanumeric string value to represent the name of the NFT item:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field item_name : String = String ""



struct Name {

    String item_name;

}
```

`field item_name : String = String ""`

= Description needs to describe the content of the NFT item.

The mutable Description field will store a long text value to represent the description of the NFT item:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field description : String = String value



struct Description {

  Description String
```

```
}
```

`field description : String = String value`

= Bids need to have the offer_price, offer_address, creation_timestamp. The "NFT" the Bid belongs to will contain a timeout value represented by a Uint128 that determines the Bid's expiration. The Bid's creation timestamp is represented by the block number the bid's transaction Id belongs to.

The mutable bids field will store a Map of Bids, and should then resemble this:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field bids : Map keyType (tk) valueType (tv)



struct Bid {

    ByStr20 bidder_address;

    Uint128 bid_price; // unsigned integers can't be negative.

  BNum created_at;

  Uint128 duration_time;

}
```

`field bids : Map ByStr20 (Pair (Uint128) (BNum))`

= Members need to have purchased the unlockable tier from the NFT to be added to the map array of members, which belongs to the NFT item. They can see unlockable content based on the Tier[id] they've purchased. A transition handles the access control of members for each NFT item that has unlockableContent as True.

The mutable Members field will store a Map of Members, and should then resemble this:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field members : Map keyType (tk) valueType (tv)
```

```
struct Members {

    ByStr20 member_address;

    Uint32 tier_purchased; // correlates to Tier[id].

  BNum joined_at;

}
```

`field members : Map ByStr20 (Pair (Uint32) (BNum))`

= Favorites keeps track on what addresses have 'loved' the NFT item. This creates a map array of addresses that have performed the transition/procedure FavoriteNFTItem and returns a integer value to represent the count.

The mutable Favorites field will store a Map of wallet addresses, and should then resemble this:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field favorites : Map keyType (tk) valueType (tv)



struct Favorites {

    ByStr20 favoriter_address;

   String username;

}
```

`field favorites : Map ByStr20 String = Emp ByStr20 String`

= Offers is extremely similar to the Bids map for English-style auctions. It needs to have the offer_price, offer_address, creation_timestamp. The "NFT" the Offer belongs to will contain a timeout value represented by a Uint128 that determines the Offer's expiration. The Offer's creation timestamp is represented by the block number the bid's transaction Id belongs to.

Transition logic for Offers should not be limited to a time duration expiry like Bids!

The mutable Offers field will store a Map of Offers, and should then resemble this:

```
// Equivalent data written in C++ struct

// Scilla Equivalent = field bids : Map keyType (tk) valueType (tv)



struct Offer {

    ByStr20 offer_address;

    Uint128 offer_price; // unsigned integers can't be negative.

  BNum created_at;

}
```

`field offers : Map ByStr20 (Pair (Uint128) (BNum))`

:::

`Create Bundled Auction Contract`

"bundles": existing NFTs bundled together into one collection, one price

Contract Definitions: (creator: ByStr20, symbol: String, auction_type: Uint32, platform_contract_address: ByStr20)

```
// auction_type = 0: Fixed Price, 1: English Auction
```

Mutable Fields: Name, Description, Fixed Priced Value, A map array of NFT Items, Auction Configuration Map,

```
// field bundle_items : Map ByStr20 Uint64



struct itemsInBundle {
```

```
  ByStr20 item_address


  Uint64 bundleOrderId


}
```

```
If auctionType = fixed price listing, only the mutable field for
initial_fixed_price is shown and all other auction parameters in the Auction map
are disabled/hidden.
```

```
If auctionType = highest bid, mutable fields like reserve price, minimum price,
current_price and time duration are enabled in the Auction map as well as boolean
values like "includeMinimumPrice?", "currentPriceIsShown?", "biddingIsAllowed?"
"includeReservePrice?" are enabled from Option None [False] to Some [True]. This
allows for bidding and a transition manages the bid map based on this auctionType.
The highest bid value of the map is then shown as the current price.
```

Transitions: AddItemToBundle, configureAuction

The configureAuction transition read's contract definition's auctionType then configures required booleans parameters, inputs end client-provided values into Uint128 parameters and the imperative fragment implements the style of the auction based on auctionType

::.

::.

`SHOWCASE Smart Contract`

> Social utilities that allow users to upload their NFTs from different blockchains to showcase across the 2.0 web via a shareable profile URL link as well as interact with posts, comment, favorite, and organize collections. All of these will be viewable from the user profile, and explore page of the Braggart platform.

The goal of this smart contract is to bring minimal social networking to the Braggart platform in the style of Ello.co and TryShowtime.com. Existing NFTs can be published as 'Posts' recorded on the blockchain, containing existing NFTs queried by external blockchain metadata about the NFT's token id/address and replicable mutable fields like 'comment' maps, 'favorite' maps.

To create a Post showcasing an externally-uploaded NFT, specify which blockchain the NFT is from, the token address and create the Post. In order to display the Post, a transition will exist to retrieve the tokenURI's image metadata from the tokenAddress of a Post.

Requirements

•User..Profile Scaffolding

> Transition: find wallet_address by username

•IMMUTABLE: Wallet_address {ByStr20} ☑
•HeaderImage {String} [handled by IPFS] ☑
•ProfilePfpImage {String} [handled by IPFS] ☑
•Name {String} ☑
•IMMUTABLE: @Username {String} ☑
•Bio {String} ☑
•Following / Followers [two maps, who wallet is following, who is following wallet] {Map} ☑
•Following Map {ByStr20 address String username BNum followed_at_timestamp}
•Following Transition: `followUser() - adds wallet_address to following map`, `unfollowUser() - adds wallet_address to following map`
•Followers Transitions
•External URL Link {String} ☑
•Social Twitter URL Link {String} ☑
•Segued Tab of User Collections
•Map of Favorites ("Liked") ☑ - queryAllPostsFavoritedByWalletAddress()
•Map of Posts ("Owned") ☑- queryAllPostWithWalletAddressAsOwner()
•Map of NFTs ("Created") ☑ - queryAllNFTsWithWalletAddressAsCreator()
•Post Scaffolding
•NFTs

> NFTs created on the Braggart platform are differentiated from Posts by smart contract and user interface which displays NFTs as 'Created' and Posts as 'Owned'

•IMMUTABLE: Blockchain Type `0: Ethereum, 1: Near, 2: Harmony, 3: Matic` ☑
•IMMUTABLE: tokenAddress - address of the minted NFT token ☑
•tokenURI metadata retrieved by Address via Transition: `returnMetadataFromTokenAddress()`
•IMAGE: Image URL
•NAME: Name Label
•CURRENT_OWNER: Owner
•MUTABLE: Caption [Description for the post by the User/wallet_address] {String} ☑
•Favorited [Array of all who favorited by a User/wallet_address] {Map} ☑
•Favorite Transition [[AddPostToFavorites]]
•Bragg ("Repost") [Array of all reposts by a User/wallet_address] {Map} ☑
•Views [Integer count of times viewed] {Uint64} ☑
•Comments [Array of strings posted by other Users/wallet_addresses on a Post] {Map} ☑

•Timestamp [Epoch retrieved from BlockNumber] {BNum} ☑
•IMMUTABLE: Owner [User/Wallet address that posted the post] {ByStr20} ☑
•IMMUTABLE: UUID [Platform's post ID that makes it shareable] {Uint128} ☑
•Tags [Explorable metatag properties that describe the Post] {Map Uint32 String `tagId Name`} ☑
•External URL Link [Link to the external blockchain's dApp to bid on NFT] {String} ☑
•Gallery Scaffolding

> Galleries are collections of post displayed in a horizontal card view with a name, description and special UUID link to make shareable

•MUTABLE: Name [Name of the gallery] {String} ☑
•IMMUTABLE: Curator [User/wallet address that combined the posts] {ByStr20} ☑
•MUTABLE: Description [Text describing the gallery] {String} ☑
•IMMUTABLE: UUID [Platform's post ID that makes it shareable] {Uint128} ☑
•MUTABLE: Featured Posts [All the Posts added to the gallery] {Map} ☑

```
•struct FeaturedPosts {
   Uint32 postUUID
  ByStr20 post_contract_address
}
```

•Explore
•Query Posts by Wallet Address (Post.owner)
::.

`INNOVATE Smart Contract`

END CLIENT OBJECTIVE: Dynamically create user-customized art that can be exported as blob with mime-type 'png' or 'jpg' to be uploaded directly to IPFS as media for NFT Item Creation

CONTRACT OBJECTIVE: Retrieve values and metadata like tokenURI's image_url field to be displayed in a 3D space via client-side three.js `javascript` library.

.::