Angry Teenagers NFT and sale contracts

Created	@November 22, 2021 4:38 PM
Lead	
Status	Completed
Type	Technical Spec
Related to Back End Work (Specification)	
Related to Front End Work (1) (Specification)	
Property	

Background

The goal of this document is to describe the Angry Teenager NFT implementation as well as the way to deploy it on the Tezos network. The Angry Teenager NFT implements the product & marketing requirements as show in the Angry Teenager NFT PRD.

Contents

Technical overview of the Angry Teenager NFT

Code can be found in our gitHub . Information on how to compile, test and deploy are stored in the main README.md file of this repo.

Sale contract

The sale contract is a contract that gives some tools to EcoMint to sell their NFTs. It is highly configurable and can handle several types of sales like public sale or pre-sale with allowlist. Moreover, this contract is the entrypoint of the mint. The user can only mint Angry Teenagers NFTs through this contract. The FA2 contract cannot be called directly to mint.

Some functionalities are highlighted below:

Mint and give NFT from the EcoMint admin (multisig):

EcoMint (us) can decide to mint (for free) and give the minted NFT to a chosen address. This is obviously optional and it is reserved to EcoMint admin (handle by a multisig). This can be used to give NFTs (to an investor for instance).

Create a whitelist (called allowlist)

There are 3 possible ways to fill this allowist:

1. Open a private registration event where people can register if they are approved by the EcoMint admin (the EcoMint admin must fill a pre-allowlist before this event). To register a configurable fee must be paid (it could be 0). This event can be configured to close automatically after a delay or it can be closed manually by the EcoMint admin. Only one space on the whitelist is available per address (wallet).
2. Open a public registration event to the allowlist with a maximum number of spaces available and a configurable fee to pay (it could be 0). First come, first serve. Only one space per wallet. Event can be configured to close automatically after a delay or it can be closed manually by the EcoMint admin.
3. EcoMint admin can fill the whitelist himself if he wants to.

Any of these 3 events can be used several times or none as long as no more than one event is opened at the same time.

Private Sale

The private sale used the allowlist. Only user in the allowlist can mint during this event. When the event is opened, the EcoMint admin defines the max number of NFTs for this event, the price per NFT (it could be 0) and a maximum number of NFT mintable per address (wallet). Users can mint several NFTs at the same time as long as they pay the right price and it is not more than the number allocated for the event. This event can be configured to close automatically after a delay or it can be closed manually by the EcoMint admin.

Public sale without whitelist

A public sale can be opened by the EcoMint admin. First come, first serve. When the event is opened, the EcoMint admin defines the max number of NFTs for this event, the price per NFT (it could be 0) and a maximum number of NFT mintable per address (wallet). Users can mint several NFTs at the same time as long as they pay the right price and it is not more than allowed by the event. This event can be configured to close automatically after a delay or it can be closed manually by the EcoMint admin.

Public sale with allowlist

A public sale can be opened by the EcoMint admin using the allowlist. It is basically the same as the previous event except that people in the whitelist can have one of the two following advantages (or both. It is up the EcoMint admin to decide how the event is configured):

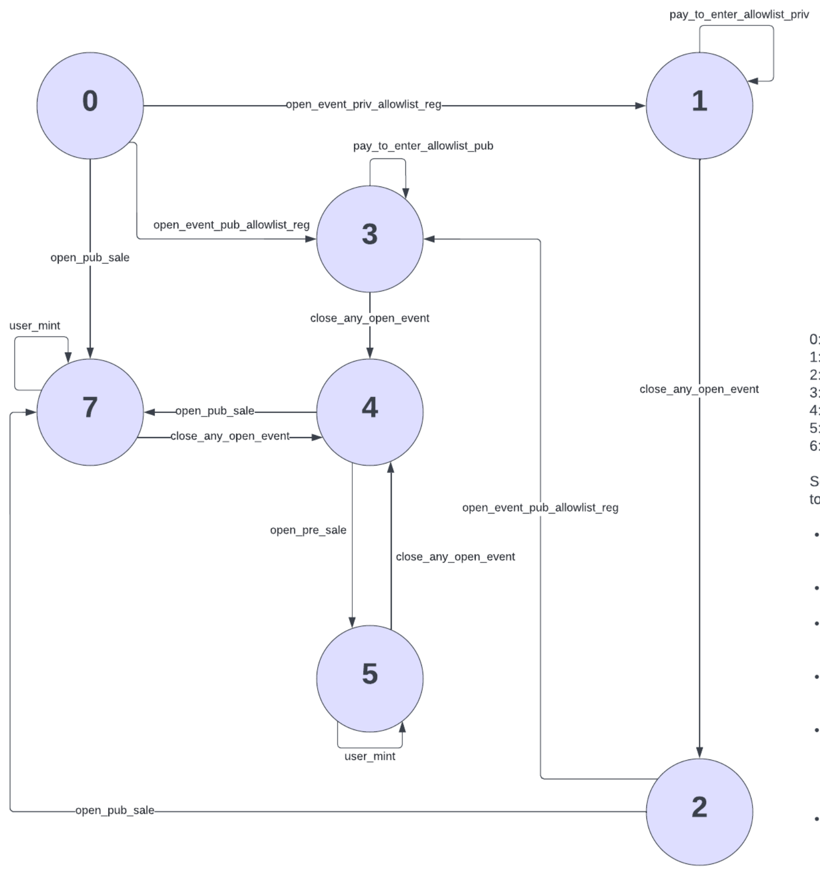
- **Discount**: The user in the allowlist will pay the price defined for the event minus the configured discount
- **Mint rights**: The people in the whitelist will always be able to mint all the configured NFTs per user in this event as long as the event is still opened and even if the total supply of NFTs defined for this event has been minted already.

Clear allowlist

The allowlist can be cleared to start a new sale process.

Any of the event defined in this email can be opened/closed several times or none depending on what we want to do (as long as it makes sense).

State machine



0: STATE_NO_EVENT_OPEN
1: STATE_EVENT_PRIV_ALLOWLIST_REG
2: STATE_NO_EVENT_WITH_PRIV_ALLOWLIST_READY
3: STATE_EVENT_PUB_ALLOWLIST_REG
4: STATE_NO_EVENT_WITH_PUB_ALLOWLIST_READY
5: STATE_EVENT_PRESALE
6: STATE_EVENT_PUB_SALE

Some information are not displayed on the chart to simplify the reading:

- Entrypoint admin_fill_allowlist does not change the state and can only be called from state 0, 2 or 4 (when no events are opened).
- Entrypoint admin_fill_pre_allowlist does not change the state and can only be called from state 0.
- Entrypoint mint_and_give does not change the state and can only be called from state 0, 2 or 4 (when no events are opened).
- Entrypoint clear_allowlist returns to state 0 when successfully called. It can only be called from state 0, 2 or 4 (when no events are opened).
- Arrows with entrypoint close_any_open_event are a simplification. Events can be configuted to have a maximum duration. In this case they can be closed with a call to close_any_open_event or because the duration of the event elapsed.
- Arrows with entrypoint open_pub_sale are a simplification. The entrypoint can be either open_pub_sale or open_pub_sale_with_allowlist.

Process presale contracts

The sale contracts can process presale contracts. EcoMint creates presale contracts to sale tokens before the main project is opened.

The entrypoint `admin_process_presale` can be called using the address of a presale contract. The entrypoint will then:

- Read all the tokens and the associated owner address which are minted but not burned in the presale contracts
- Mint a token in the main FA2 contract for this token using the associated owner address

- Burn the token in the presale contract.

Presale contracts shall follow the following interface to be injected successfully into the sale contract:

- FA2 interface
- Extra required entrypoint(s):
    - `burn`: (admin) Token are burned

  ```
  sp.TList(sp.TNat) # List of tokens to burn
  ```

- Onchain views
    - `all_tokens`: Return all the tokens minted in a list

      ```
      sp.TUnit
      ```

      Return:

      ```
      sp.TList(sp.TNat)
      ```

    - `get_token_owner`: Get the address that owns the token

      ```
      sp.TNat # Token Id
      ```

      Return:

      ```
      sp.TAddress
      ```

    - is_token_burned: Return whether a token is burned or not

      ```
      sp.TNat # Token Id
      ```

      Return:

      ```
      sp.TBool
      ```

Interface

Contract entrypoints:

- `admin_fill_allowlist`: (admin) The admin fill the allowlist.
    - This is an optional operation that can only happen when the allowlist is not used yet (i.e no sale open).

  ```
  sp.TSet(sp.TAddress)
  ```

- `admin_fill_pre_allowlist`: (admin) The admin fill the pre-allowlist.
    - The pre-allowlist is used to give a ticket to a user for getting a space in the allowlist at the condition that he pays a fee for it.

    - This fee will have to be paid by the user during the even called `event_private_allowlist_registration`.

```
sp.TSet(sp.TAddress)
```

- `open_event_priv_allowlist_reg`: (admin) This event allows addresses on the pre-allow list to register to the allowlist for a fee by calling the user function `pay_to_enter_allowlist_priv`.
    - If the `use_deadline` parameter is `True`, then the event will close automatically when the `deadline` is reached. The price parameter defines the price to be paid to enter the allowlist.
    - This event is optional but needs to happen first if chosen by the admin.

```
sp.TRecord(price=sp.TMutez, use_deadline=sp.TBool, deadline=sp.TTimestamp)
```

- `pay_to_enter_allowlist_priv`: (all in pre-allowlist) User transfers Tez to register to the allowlist while calling this function.
    - This is only possible when the associated event has been opened by the admin (`open_event_priv_allowlist_reg`).
        - Only addresses is the pre-allowlist (filled by the admin using `admin_fill_pre_allowlist`) are allowed to call this function.

```
sp.TUnit
```

- `open_event_pub_allowlist_reg`: (admin) This event allows public addresses to register to the allow list for a fee by calling the function `pay_to_enter_allowlist_pub`.
    - A maximum number of space is given by the parameter `max_space`.
    - If `use_deadline` parameter is `True`, then the event will close automatically when the `deadline` is reached. The price parameter defines the price to be paid to enter the allowlist.
    - This event is optional but needs to happen before the allowlist is used if chosen by the admin.

```
sp.TRecord(max_space=sp.TNat, price=sp.TMutez, use_deadline=sp.TBool, deadline=sp.TTimestamp)
```

- `pay_to_enter_allowlist_pub`: (all) User transfers Tez to register to the allowlist while calling this function. This is only possible when the associated event has been opened by the admin (`open_event_pub_allowlist_reg`).
    - This event is opened to any address (as long as it is not already in the allowlist) but there is a limited number of space.

```
sp.TUnit
```

- `open_pre_sale`: (admin) The admin open the pre-sale. Pre-sale are optional. Pre-sale uses the allowlist.
    - Only addresses in the allowlist can mint token (by calling the `user_mint` function). Not more than max_supply token can be minted in total during the event.
    - Not more than `max_per_user` token can be minted per user. Price of token during the pre-sale is defined by the parameter price.
    - If `use_deadline` parameter is `True`, then the event will close automatically when the `deadline` is reached.

```
sp.TRecord(max_supply=sp.TNat, max_per_user=sp.TNat, price=sp.TMutez, use_deadline=sp.TBool, deadline=sp.TTimestamp)
```

- `open_pub_sale`: (admin) The admin open the public sale. Public sale is optional. Public sale could or could not use the allowlist. When calling this entry point the allowlist is not used. To use the allowlist call `open_pub_sale_with_allowlist`.
    - Not more than `max_supply` token can be minted in total during the event.

    - Not more than `max_per_user` token can be minted per user. Price of token during this public sale is defined by the parameter price.
    - If `use_deadline` parameter is `True`, then the event will close automatically when the `deadline` is reached.

    ```
    sp.TRecord(max_supply=sp.TNat, max_per_user=sp.TNat, price=sp.TMutez, use_deadline=sp.TBool, deadline=sp.TTimestamp)
    ```

- `open_pub_sale_with_allowlist`: (admin) The admin open the public sale with the allowlist. Public sale is optional.
    - Everybody can mint token during public sale, however people in the allowlist can have the following advantage depending on the configuration (params `mint_right` and `mint_discount`):
        1. `mint_right`: User can always mint token when the public sale is opened even when max_supply is reached
        2. `discount`: User pays a discounted price (price - discount) to mint token.
    - Not more than `max_per_user` token can be minted per user even for user in the allowlist.
    - Price of token during this public sale is defined by the parameter price (with a possible discount for allowlist address).
    - If `use_deadline` parameter is `True`, then the event will close automatically when the `deadline` is reached.

    ```
    sp.TRecord(max_supply=sp.TNat, max_per_user=sp.TNat, price=sp.TMutez, use_deadline=sp.TBool, deadline=sp.TTimestamp, mint_right=sp.TBoo
    ```

- `set_metadata` (admin): Set the contract metadata with the off-chain views.

    ```
    sp.TUnit
    ```

- `user_mint` (all): User mints token during pre-sale (if he is on the allowlist) or public sale. Expected amount of Tez shall be transferred when calling this function.

    ```
    sp.TRecord(amount=sp.TNat, address=sp.TAddress)
    ```

- `close_any_open_event` (admin): The admin can closed any open event. This is necessary to close event that have no duration.

    ```
    sp.TUnit
    ```

- `mint_and_give` (admin): The admin can mint and give a given number of token to an address.
    - This can only happen when no event are opened.

    ```
    sp.TRecord(amount=sp.TNat, address=sp.TAddress)
    ```

- `set_next_administrator: ` (who: admin) Set the next admin. The next admin is not admin yet. It will be come admin by calling the entrypoint `validate_new_admininistrator`.

    ```
    sp.TAddress
    ```

- `validate_next_administrator` (who: “next_admin”) Validate the new admin. `set_next_administrator` shall be called by the old admin prior to this function.

    ```
    sp.TUnit
    ```

- `set_transfer_address` (admin): Set the transfer address. The transfer address is the address where the fund are sent when token are minted by a user.

  ```
  sp.TAddress
  ```

- `register_fa2` (admin): Register the FA2 contract. In this case, this is the **AngryTeenager NFT contract**.

  ```
  sp.TAddress
  ```

- `clear_allowlist` (admin): Clear the allowlist (and the pre-allowlist) and return to state 0.
  This can be called anytime if not event are opened.

  ```
  sp.TUnit
  ```

- `mutez_transfer`(admin): Protection to allow an extraction of the fund in the contract. No fund are supposed to be held by this contract but if it happens we want to be able to extract them.

  ```
  sp.TAddress
  ```

- `admin_process_presale` (admin): See https://www.notion.so/ecomint/Angry-Teenagers-NFT-and-sale-contracts-b1efe7fb4fad48c89777a07ccd356ad2#72ba364f4e5045bdb8f1fb285cd0d4e0

  ```
  sp.TAddress # Presale contract address
  ```

Off-chain views

- `get_mint_token_available`: Return the number of tokens an address can mint. If no sale are open, this function will return 0.

  ```
  sp.TAddress
  ```

  Return:

  ```
  sp.TNat
  ```

FA2 contract

The FA2 standard defines a common interface for multi-asset token particularly suitable for NFTs. The FA2 contract of the AngryTeenagers is not based on any template. It is implemented internally for the following reasons:

- Simplicity of the code
- Readability of the code
- Performance of the code
- Extensibility of the code
- Implement specific features of the Angry Teenager code.

However, this contract implements both the **TZIP-012** and **TZIP-016**:

- `balance_of`: From TZIP-012. Function must be kept as is.
- `transfer`: From TZIP-012. Function must be kept as is.
- `update_operators`: From TZIP-012. Function must be kept as is.

The Angry Teenagers contract contains a snapshot mechanism. This allow to retrieve via an onchain view any user balance at any point in time in the past using the Tezos block as a time reference. This is useful for DAOs that requires to get the voting power of one user not at the time of the user's vote but at the time where the vote is opened.

The following custom entry-points of the contract are implemented:

- `set_pause`: Set the FA2 contract in pause. Mint or transfer calls are then not possible.

```
sp.TUnit
```

- `set_administrator`: Change the main administrator.

```
sp.TAddress
```

- `set_next_administrator:` (who: admin) Set the next admin. The next admin is not admin yet. It will be come admin by calling the entrypoint `validate_new_admininistrator`.

```
sp.TUnit
```

- `set_sale_administrator`: Change the sale administrator.
  - Only the main administrator can call this entry-point.
  - The sale administrator is allowed to mint tokens. All the operations that can perform the sale administrator can be performed by the main administrator too.

```
sp.TAddress
```

- `set_artwork_administrator`: Change the artwork administrator.
  - Only the main administrator can call this entry-point.
  - The sale administrator is allowed to update the artwork of tokens (this is for instance useful after the first mint when the server updates the artwork of the token). All the operations that can perform the artwork administrator can be performed by the main administrator too.

```
sp.TAddress
```

- `mint`: Mint token. Only main and sale administrator can call this entry-point. This function will create a new token with a unique geo located tag and generic artwork.

```
sp.TAddress
```

- `update_artwork_data`: Update the artwork of a given token.
  - Only the main and artwork administrators can call this entry-point. Token needs to be all non-revealed. If not, the call will fail.

```
sp.TList(sp.TPair(TOKEN_ID, sp.TRecord(artifact_uri=sp.TBytes, display_uri=sp.TBytes, thumbnail_uri=sp.TBytes, attributes=sp.TBytes, ar
```

- `admin_add_what3words`: Only the main admin can call this entry-point.
  - Add **what3words** square (land) to the contract. They cannot be removed anymore. Mint is not possible if there are no remaining words.

```
sp.TSet(sp.TBytes)
```

- `mutez_transfer`: Only the main admin can call this entry-point.
    - This can be used to remove fund that are stuck in the contract (this should not happen however as this contract is not supposed to receive funds).

  ```
  sp.TAddress
  ```

- `set_royalties:` Change the royalties field of the token metadata (the minted ones as well as the ones to mint).
    - Only the main administrator can call this function.

Some on-chain views are implemented to get information about the snapshot:

- `get_total_voting_power`: Returns the current voting power of the FA2 contract i.e the number of NFTs minted so far.

  ```
  sp.TUnit
  ```

  Return :

  ```
  sp.TNat
  ```

- `get_voting_power`: Returns the voting power (i.e the balance) of one user at any point of time now or in the past using the Tezos block Id.

  ```
  sp.TPair(sp.TAddress, sp.TNat)
  Address: Address of the user
  Nat: Tezos block Id
  ```

  Return:

  ```
  sp.TNat
  ```

Some off-chain views are implemented:

- `token_metadata`: Returns the token metadata of one token. Standard function.
    - Angry Teenager raw metadata are not stored in the contract storage but are generated on the fly when this function is called.

  ```
  sp.TNat
  ```

Return:

```
sp.TPair(sp.TNat, sp.TMap(sp.TString, sp.TBytes))
```

Returned the metadata as described in https://www.notion.so/ecomint/Angry-Teenager-NFT-b1efe7fb4fad48c89777a07ccd356ad2#174f452163bf491aa99d0db71669d08e

- `get_project_oracles_stream`: Return a ceramic link pointing on the the project oracles

  ```
  sp.TUnit
  ```

  Return:

```
sp.TBytes
```

- `count_tokens`: Return the total number of token minted in the contract

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TNat
  ```

  Return:

- `does_token_exist`: Return whether a token (by receiving its token_id) exists or not

```
sp.TBool
```

- `all_tokens`: Return a list of all token_ids minted inside the contract

  ```
  sp.TUnit
  ```

  Return:

```
  sp.TList(sp.TNat)
```

- `get_user_tokens`: Return a list of all token_ids owned by a user

  ```
  sp.TAddress
  ```

  Return:

  ```
  sp.TList(sp.TNat)
  ```

- `get_all_non_revealed_tokens`: Return the set of all `token_ids` minted but not revealed yet (artwork have not been updated by the artwork admin yet).

  ```
  sp.TUnit
  ```

  Return:

  ```
  sp.TSet(sp.TNat)
  ```

- `total_supply`: Get the total supply for one given `token_id`. Because this contract contains NFTs only, the total can be either 0, either 1 but not more.

  ```
  sp.TNat
  ```

  Return:

```
sp.TNat
```

- `is_operator`: Return whether an address is the operator of a specific token.

  ```
  sp.TRecord(token_id=sp.TNat, owner=sp.TAddress, operator=sp.TAddress)
  ```

  Return:

  ```
  sp.TBool
  ```

- `get_balance`: Return the balance for a specific `token_id` for one address. Because this contract contains only NFTs, the balance is only 0, only 1.

  ```
  sp.TRecord(owner=sp.TAddress, token_id=sp.TNat)
  ```

  Return:

  ```
  sp.TNat
  ```

Angry Teenager NFT characteristics

As we saw, the Angry Teenager NFT follows pretty much the standard but it has some specific characteristics.

Contract Metadata

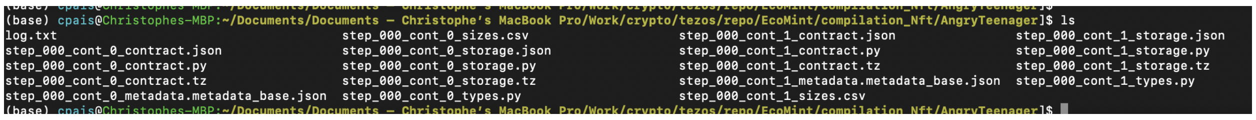
Contract metadata are needed to at least communicate the off-chain views to the users. This is defined in **TZIP-016**. One way to create this metadata (and the associated Michelson code) is to use the `metadata_base` technics.

```
def __init__(self, administrator, metadata):
        self.operator_set = Operator_set()
        self.error_message = Error_message()

        self.init(
            ledger = sp.big_map(tkey=TOKEN_ID, tvalue=sp.TAddress),
            operators=self.operator_set.make(),

            # Administrator
            administrator=administrator,
            sale_contract_administrator=administrator,

            # Pause
            paused=sp.bool(False),

            # Minted tokens
            minted_tokens = sp.nat(0),

            # What3words set
            squares_set = sp.big_map(tkey=TOKEN_ID, tvalue=sp.TBytes),
            squares_set_next_empty = sp.nat(0),

            # Token metadata
            avatars_metadata = sp.big_map(tkey=TOKEN_ID, tvalue=AVATARS_CONTAINER_TYPE),
            generic_token_info = sp.big_map(tkey=TOKEN_ID, tvalue=sp.TRecord(date=sp.TTimestamp, minter=sp.TAddress)),

            metadata = metadata
        )

        list_of_views = [
             self.get_balance
             , self.does_token_exist
             , self.count_tokens
```

```
              , self.all_tokens
              , self.is_operator
              , self.total_supply
              , self.token_metadata
         ]

         metadata_base = {
              "version": "1.0"
              , "description": (
                      "Angry Teenagers... on the Tezos blockchain."
         )
              , "interfaces": ["TZIP-012", "TZIP-016", "TZIP-021"]
              , "authors": [
                  "EcoMint LTD"
             ]
              , "homepage": "https://www.angryteenagers.xyz"
              , "views": list_of_views
              , "permissions": {
                  "operator":
                      "owner-or-operator-transfer"
                  , "receiver": "owner-no-hook"
                  , "sender": "owner-no-hook"
             }
         }
         self.init_metadata("metadata_base", metadata_base)
```

The metadata_base is compiled using SmartPy that will generate a file containing this metadata.

See:



```
(base) cpais@Christophes-MBP:~/Documents/Documents — Christophe's MacBook Pro/Work/crypto/tezos/repo/EcoMint/compilation_Nft/AngryTeenager]$ ls
log.txt                                   step_000_cont_0_sizes.csv                 step_000_cont_1_contract.json             step_000_cont_1_storage.json
step_000_cont_0_contract.json             step_000_cont_0_storage.json              step_000_cont_1_contract.py               step_000_cont_1_storage.py
step_000_cont_0_contract.py               step_000_cont_0_storage.py                step_000_cont_1_contract.tz               step_000_cont_1_storage.tz
step_000_cont_0_contract.tz               step_000_cont_0_storage.tz                step_000_cont_1_metadata.metadata_base.json  step_000_cont_1_types.py
step_000_cont_0_metadata.metadata_base.json  step_000_cont_0_types.py               step_000_cont_1_sizes.csv
(base) cpais@Christophes-MBP:~/Documents/Documents — Christophe's MacBook Pro/Work/crypto/tezos/repo/EcoMint/compilation_Nft/AngryTeenager]$
```

File `step_000_cont_1_metadata.metadata_base.json`. This file can be sent to IPFS and the CID can be used to be link to the contract:

```
sp.add_compilation_target("AngryTeenager",
                          AngryTeenager(
                              administrator=sp.address("tz1b7np4aXmF8mVXvoa9Pz68ZRRUzK9qHUf5"),
                              metadata=sp.utils.metadata_of_url("ipfs://QmfRFoyk8exK4zjUvfNaxmP36HuacLaRC2aK3fNf7XvPak")))
```

Another way is to use the SmartPy online editor.

It sounds better to store the contract metadata inside IPFS as it takes a lot of space when we have off-chain view.

Moreover, a function:

```
@sp.entry_point
def set_metadata(self, k, v):
    sp.verify(self.is_administrator(sp.sender), message = self.error_message.not_admin())
    self.data.metadata[k] = v
```

It allows to replace the metadata and then create new off-chain functions.

Token Metadata

The contract contains both a token_metadata array and an off-chain function called token_metadata which will return the metadata of a given `token_id`. This is standard as per the **TZIP—12** (we use both the Basic and **Custom** technic):

> Token Metadata Storage & Access

> A contract can use two methods to provide access to the token-metadata.

> **Basic**: Store the values in a big-map annotated `%token_metadata` of type `(big_map nat (pair (nat %token_id) (map %token_info string bytes)))`.

> **Custom**: Provide a `token_metadata` off-chain-view which takes as parameter the `nat` token-id and returns the `(pair (nat %token_id) (map %token_info string bytes))` value.

> In both cases the “key” is the token-id (of type `nat`) and one MUST store or return a value of type `(pair nat (map string bytes))`: the token-id and the metadata defined above.

> If both options are present, it is recommended to give precedence to the the off-chain-view (custom).

Token metadata characteristics

- The set of **what3words** squares is not mutable
- Only the owner of a token can transfer this token. Even admins can’t do it. The owner can use operator though.

Token metadata fields

1. “name”:
    a. Mandatory (TZIP-021).
    b. Not mutable
    c. Ex: “Angry Teenager #1”.
2. “symbol”:
    a. Mandatory (TZIP-021).
    b. Not mutable
    c. “ANGRY”
3. “decimals”:
    a. Mandatory (TZIP-021).
    b. Not mutable
    c. Ex: “0
4. “artifactUri”: A URI to the asset
    a. Recommended (TZIP-021)
    b. Immutable after the reveal (the main admin or the artwork admin calls the entrypoint `update_artwork_data` to reveal the token and update the artifactUri field).
    c. Ex: It is an IPFS link
5. “displayUri”: A URI to an image of the asset
    a. Recommended (TZIP-021)
    b. Immutable after the reveal (the main admin or the artwork admin calls the entrypoint `update_artwork_data` to reveal the token and update the displayUri field).
    c. Ex: It is an IPFS link
6. “thumbnailUri”: a URI to an image of the asset for wallets and client applications to have a scaled down image to present to end-users. Recommend maximum size of 350x350px.
    a. Recommended (TZIP-021)
    b. Immutable after the reveal (the main admin or the artwork admin calls the entrypoint `update_artwork_data` to reveal the token and update the thumbnailUri field).

   c. Ex: It is an ipfs link

7. “externalUri”: a URI to a ceramic stream.
   a. Recommended (TZIP-021)
   b. Immutable after the reveal (the main admin or the artwork admin calls the entrypoint `update_artwork_data` to reveal the token and update the externalUri field).
   c. Ex: It is a ceramic link

8. “attributes”: a json string representing the attributes of the NFT artwork
   a. Recommended (TZIP-021)
   b. Immutable after the reveal (the main admin or the artwork admin calls the entrypoint `update_artwork_data` to reveal the token and update the attributes field).
   c. Ex:

```
[
    {
      "name": "project",
      "value": "project 1"
    }
    {
      "name": "Background",
        "value": "Blue"
    },
    {
        "name": "Face",
        "value": "Oval"
    },
    {
        "name": "Mouth",
        "value": "Braces"
    },
    {
        "name": "Headware",
        "value": "Hat"
    },
    {
        "name": "Anger",
        "value": "1.4",
        "type": "number"
    }
  ]
```

9. “description”: General notes, abstracts, or summaries about the contents of an asset.
   a. Recommended (TZIP-021)
   b. Not mutable
   c. “Angry Teenagers... on the Tezos blockchain”

10. “rights”: Copyright and rights of the company.
    a. Not mutable
    b. “`"© 2022 EcoMint. All rights reserved."`”

11. “creators”: The primary person, people, or organization(s) responsible for creating the intellectual content of the asset. The field is an array with all elements of the type string. Each of the elements in the array must be unique.
    a. Recommended (TZIP-021)
    b. Not mutable
    c. “`["EcoMint LTD. https://www.angryteenagers.xyz"]`”

12. “isBooleanAmount”: Describes whether an account can have an amount of exactly 0 or 1. (The purpose of this field is for wallets to determine whether or not to display balance information and an amount field when transferring.)

    a. Recommended (TZIP-021)
    b. Not mutable
    c. “true”

13. “isTransferable”: All tokens will be transferable by default to allow end-users to send them to other end-users. However, this field exists to serve in special cases where owners will not be able to transfer the token.
    a. Not mutable
    b. “true”
14. “shouldPreferSymbol”: Allows wallets to decide whether or not a symbol should be displayed in place of a name.
    a. Not mutable
    b. “false”
15. “formats”: The object is an array with all elements of the type `format` (see TZIP-021)
    a. Not mutable

```
[{"mimeType": "image/png","uri":"')
formats_bytes_suffix = sp.utils.bytes_of_string('"}]
```

16. “what3wordsFile”: The file link in IPFS containing the what3words square of the project. This is common to all tokens inside the contract.
    a. Not mutable
17. “what3wordsId”: The index pointing to the what3words zone belonging to this token in the what3words file in IPFS defined by what3wordsFile
    a. Not mutable
    b. “101”
18. “revealed”: Indicates whether the NFT has been revealed by the server or not (i.e whether the artworks have been updated after the initial mint by the user).
    a. Mutable but only one time. A call to the entry point `update_artwork_data` will make this field returning “false” and being immutable.
    b. Return a boolean string
19. “royalties”: A json string defining the EcoMint strategy of royalties when a token is transferred. Can be changed by the main admin using the entry point `set_royalties`
    a. Mutable
    b. Ex: “`'{"decimals": 2, "shares": { "tz1b7np4aXmF8mVXvoa9Pz68ZRRUzK9qHUf5": 10}}'`”
20. “projectOraclesUri”: a URI to a ceramic stream representing the project oracles. All NFTs returns the same link.
    a. Not mutable
    b. It is a ceramic link

Reference Metadata

1. Ziggurats

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/c0b7e307-a47e-4e98-ba26-f9784c37a9be/ziggurats-md.json

2. Neonz

```
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/cdd26108-e8c3-48f3-90c5-be2554f3cabc/neonz-md.json
```

3. Tezzards

```
https://s3-us-west-2.amazonaws.com/secure.notion-static.com/8965d774-72ca-42b8-9179-a69f357fe441/tezzards-md.json
```