Wormfare TokenSale Smart Contract

Project Overview

TokenSale is a smart contract that lets its owners presale an ERC20 token before the actual token is issued. The contract allows users to purchase tokens with USDT, incorporating discount and referral reward mechanisms. The contract works with a backend API that produces and signs the parameters for a user when they want to make a purchase. All received USDT except the referral reward part is transferred to a “treasury” wallet, specified during the contract deployment. After purchase, users do not receive actual tokens, the contract only tracks user token balances.

1. Functional Requirements

1.1. Roles

TokenSale contract has three roles:

- **Admin:** Full control over contract functions, including pausing and unpausing the contract, updating token price, and purchasing tokens manually for other users.
- **API Signer:** Signs token purchase parameters.
- **User:** Can purchase tokens with USDT and withdraw referral rewards in USDT received from their referral user purchases.

1.2. Features

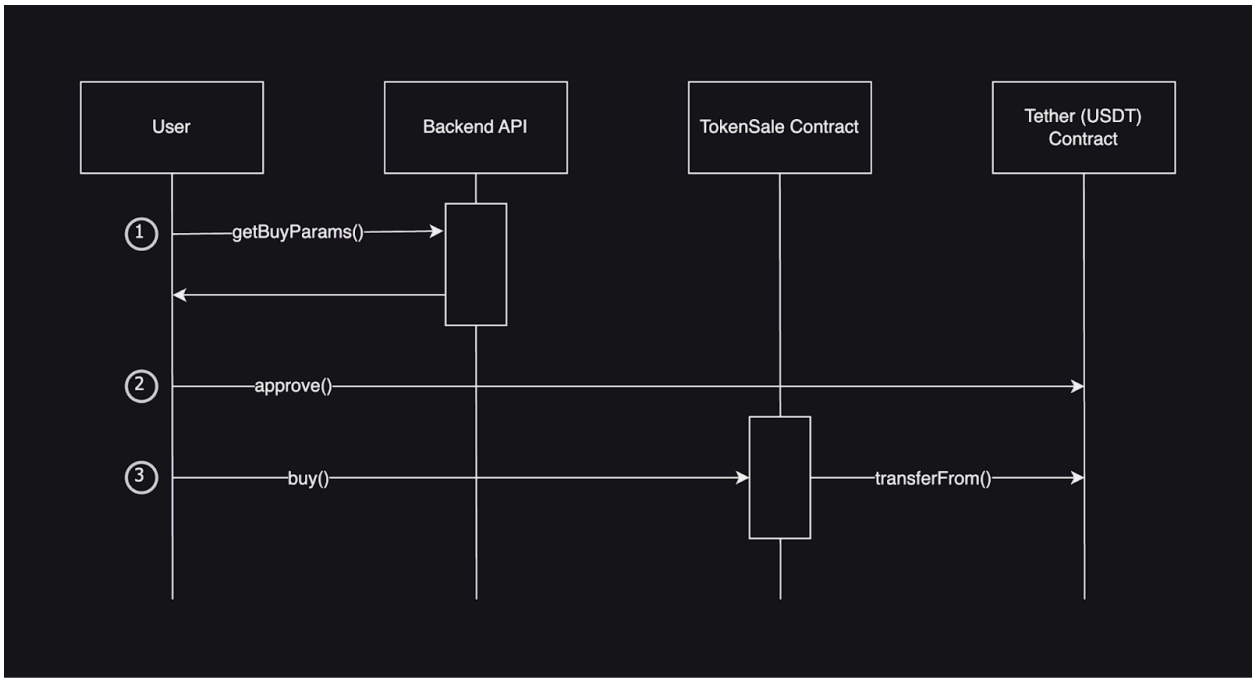
TokenSale contract has the following features:

- Purchase tokens with USDT. (User)
- Allocate USDT rewards according to the referral program.
- Withdraw USDT rewards from referral user purchases. (User)
- Purchase tokens on behalf of another user. (API Signer)
- Purchase tokens for someone else with a custom discount. (Admin)
- Pause/unpause the token purchase ability. (Admin)
- Update the token price. (Admin)
- Update the API Signer’s wallet address.

1.3. Use Cases

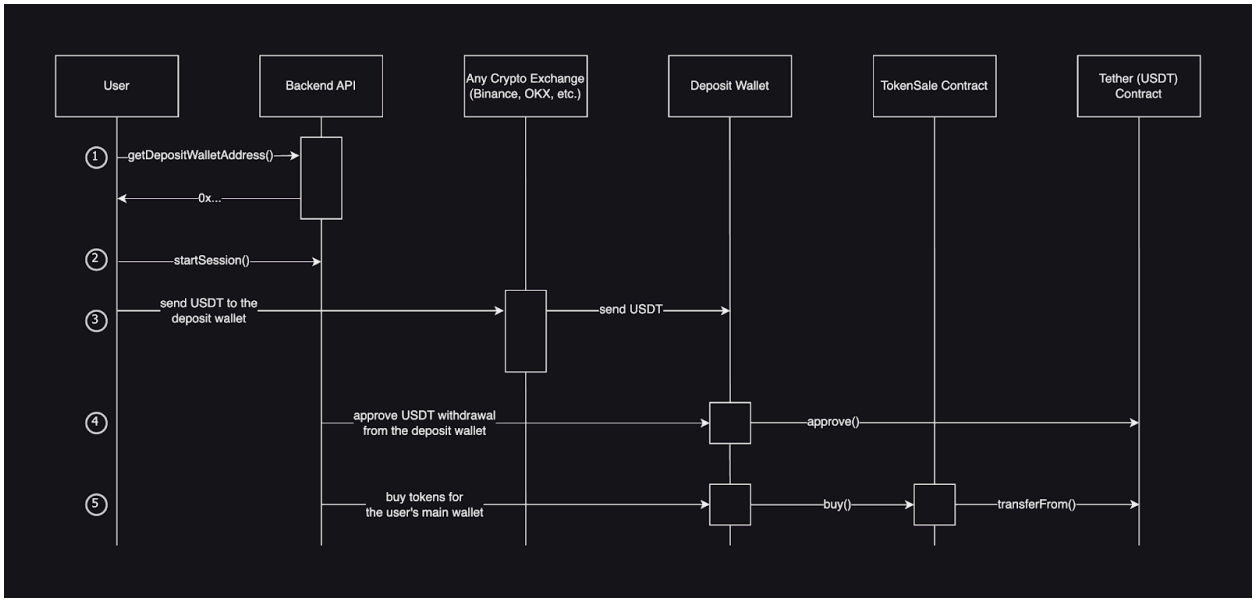
1.3.1. User purchases tokens on their own

A user purchases tokens for themselves onto their wallet and pays all the transaction fees. A discount and a referral reward may be applied. The purchase parameters are prepared and signed by an API Signer and the contract validates that the parameters have not been tampered with. In this case, the user has to sign two transactions (ERC20 **approve()** and **buy()** on the TokenSale contract) and pay the transaction fees.



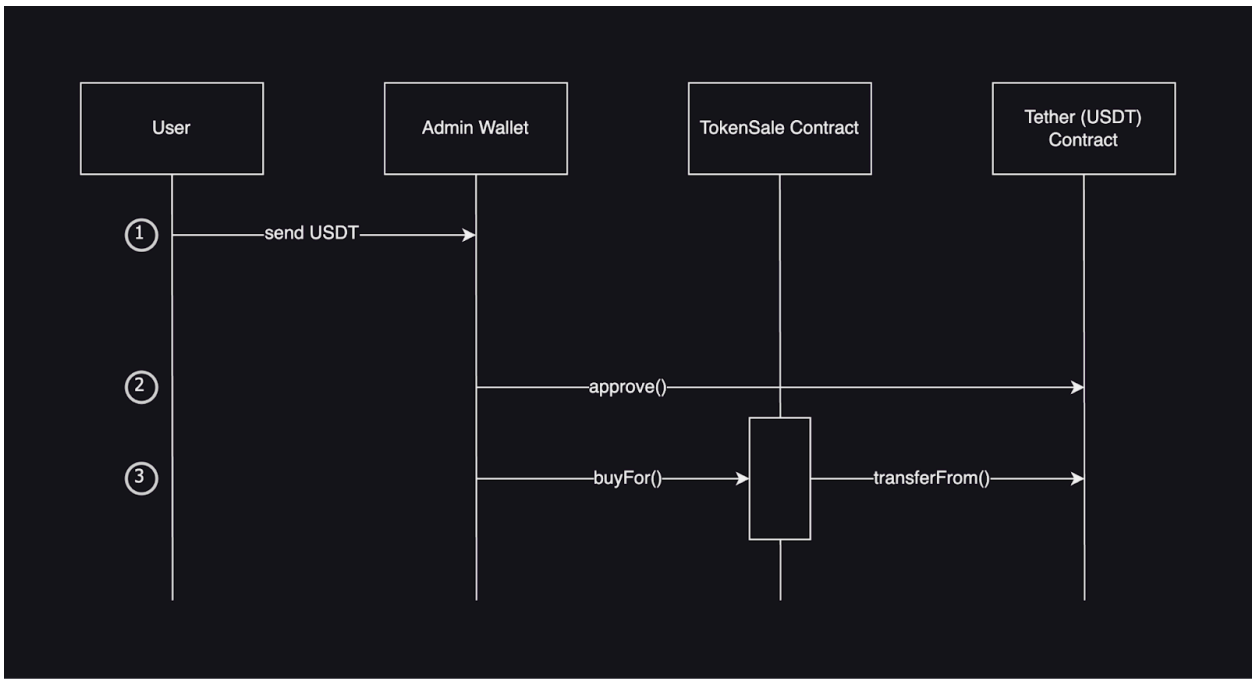
1.3.1. User purchases tokens by transferring USDT to a “deposit wallet”

A user purchases tokens for themselves by transferring the funds (typically using any centralized exchange) to a special “deposit wallet” provided by the backend API, and then the system purchases tokens on behalf of the user. The tokens are purchased for the user’s main wallet. In this case, the user does not pay any transaction fees and does not sign any transactions.



1.3.1. Admin purchases tokens for a user

An admin purchases tokens for another user. This case supposes a direct communication between a user and the admin. The admin can specify an arbitrary discount (up to 10%), but the referral program is unavailable.



2. Technical Requirements

This project has been developed with Solidity language, using Hardhat as a development environment. Typescript is the selected language for testing and scripting.

In addition, OpenZeppelin's libraries are used in the project. All information about the contracts library and how to install it can be found on their GitHub.

In the project folder, the following structure is found:

```
├── Dockerfile
├── README.md
├── contracts
│   ├── TokenSale.sol
│   ├── nft
│   │   └── WormfareGenesis.sol
│   └── testing
│       └── Tether.sol
├── deploy
│   └── contracts
│       ├── 01_TokenSale.ts
│       ├── nft
│       │   └── 01_WormfareGenesis.ts
│       ├── testing
│       │   └── 01_Tether.ts
│       └── utils
│           └── deployment-utils.ts
├── deployments
│   ├── mainnet
│   │   ├── DefaultProxyAdmin.json
│   │   ├── TokenSale.json
│   │   ├── TokenSale_Implementation.json
│   │   ├── TokenSale_Proxy.json
│   │   └── solcInputs
```

```
│   │   ├── 004ddf6428d8a7eea646b5552aa16639.json
│   │   ├── 0e89febeebc7444140de8e67c9067d2c.json
│   │   ├── 6687bec7aa5695e2252aa64569d80c00.json
│   │   ├── a92ddfc6f1849afdec66a100cd0559e6.json
│   │   ├── b24801de1e013d9c81d53b67db672452.json
│   │   └── c2b3b48738bda74354ae333500e7a316.json
│   ├── stage
│   │   ├── DefaultProxyAdmin.json
│   │   ├── TokenSale.json
│   │   ├── TokenSale_Implementation.json
│   │   ├── TokenSale_Proxy.json
│   │   └── solcInputs
│   │       ├── 004ddf6428d8a7eea646b5552aa16639.json
│   │       ├── 0e89febeebc7444140de8e67c9067d2c.json
│   │       └── 6687bec7aa5695e2252aa64569d80c00.json
│   ├── stage2
│   │   ├── DefaultProxyAdmin.json
│   │   ├── Tether.json
│   │   ├── TokenSale.json
│   │   ├── TokenSale_Implementation.json
│   │   ├── TokenSale_Proxy.json
│   │   ├── WormfareGenesis.json
│   │   ├── WormfareGenesis_Implementation.json
│   │   ├── WormfareGenesis_Proxy.json
│   │   └── solcInputs
│   │       ├── 004ddf6428d8a7eea646b5552aa16639.json
│   │       ├── 0e89febeebc7444140de8e67c9067d2c.json
│   │       ├── 0ee8391a58d9f1c4ad5f1a336746d13e.json
│   │       ├── 6687bec7aa5695e2252aa64569d80c00.json
│   │       ├── b24801de1e013d9c81d53b67db672452.json
│   │       └── c2b3b48738bda74354ae333500e7a316.json
│   └── testnet
│       ├── DefaultProxyAdmin.json
│       ├── Tether.json
│       ├── TokenSale.json
```

```
│       ├── TokenSale_Implementation.json
│       ├── TokenSale_Proxy.json
│       └── solcInputs
│           ├── 0e89febeebc7444140de8e67c9067d2c.json
│           ├── 7df759dd576d5baa5bebdad2a350c290.json
│           ├── a3bd2cda489ed4d9522ba17a54bff995.json
│           ├── a92ddfc6f1849afdec66a100cd0559e6.json
│           └── b24801de1e013d9c81d53b67db672452.json
├── docs
│   ├── TokenSale.md
│   ├── Wormfare TokenSale Smart Contract.pdf
│   ├── WormfareGenesis.md
│   ├── api
│   │   ├── TokenSale.md
│   │   ├── nft
│   │   │   └── WormfareGenesis.md
│   │   └── testing
│   │       └── Tether.md
│   └── assets
│       ├── TokenSale_case1.png
│       ├── TokenSale_case2.png
│       └── TokenSale_case3.png
├── hardhat.config.ts
├── hardhat.validate.ts
├── package.json
├── src
│   ├── enums
│   │   └── network.enum.ts
│   ├── utils.ts
│   └── zod
│       ├── zod-helpers.ts
│       └── zod-rules.ts
├── templates
│   ├── contract.hbs
│   ├── event.hbs
```

```
│   ├── function.hbs
│   ├── helpers.js
│   └── page.hbs
├── test
│   ├── deploy
│   │   ├── tether.deploy.ts
│   │   ├── token-sale.deploy.ts
│   │   └── wormfare-genesis.deploy.ts
│   ├── token-sale.test.ts
│   ├── utils
│   │   ├── common.ts
│   │   └── constants.ts
│   └── wormfare-genesis.test.ts
├── tsconfig.json
└── yarn.lock
```

Start with **README.md** to find all the basic information about the project structure and scripts that are required to test and deploy the contracts.

Inside the **./contracts** folder, **TokenSale.sol** contains the smart contract with the token sale functionality explained in section 2.2 of this document. The additional solidity file **Tether.sol** is provided in the testing directory in order to run the tests for the project.

In the **./tests** folder, **token-sale.test.ts** provides the tests of the different methods of the main contract, in Typescript. The **utils** directory contains various test helpers, while the **deploy** directory houses scripts for contract deployment used during testing.

The TokenSale contract can be deployed using the **01_TokenSale.ts** script in **./deploy**. In order to do so, **.env.example** must be renamed to **.env**, and all required data must be provided.

The project configuration is found in **hardhat.config.ts**, where dependencies are indicated. Mind the relationship of this file with **.env**.

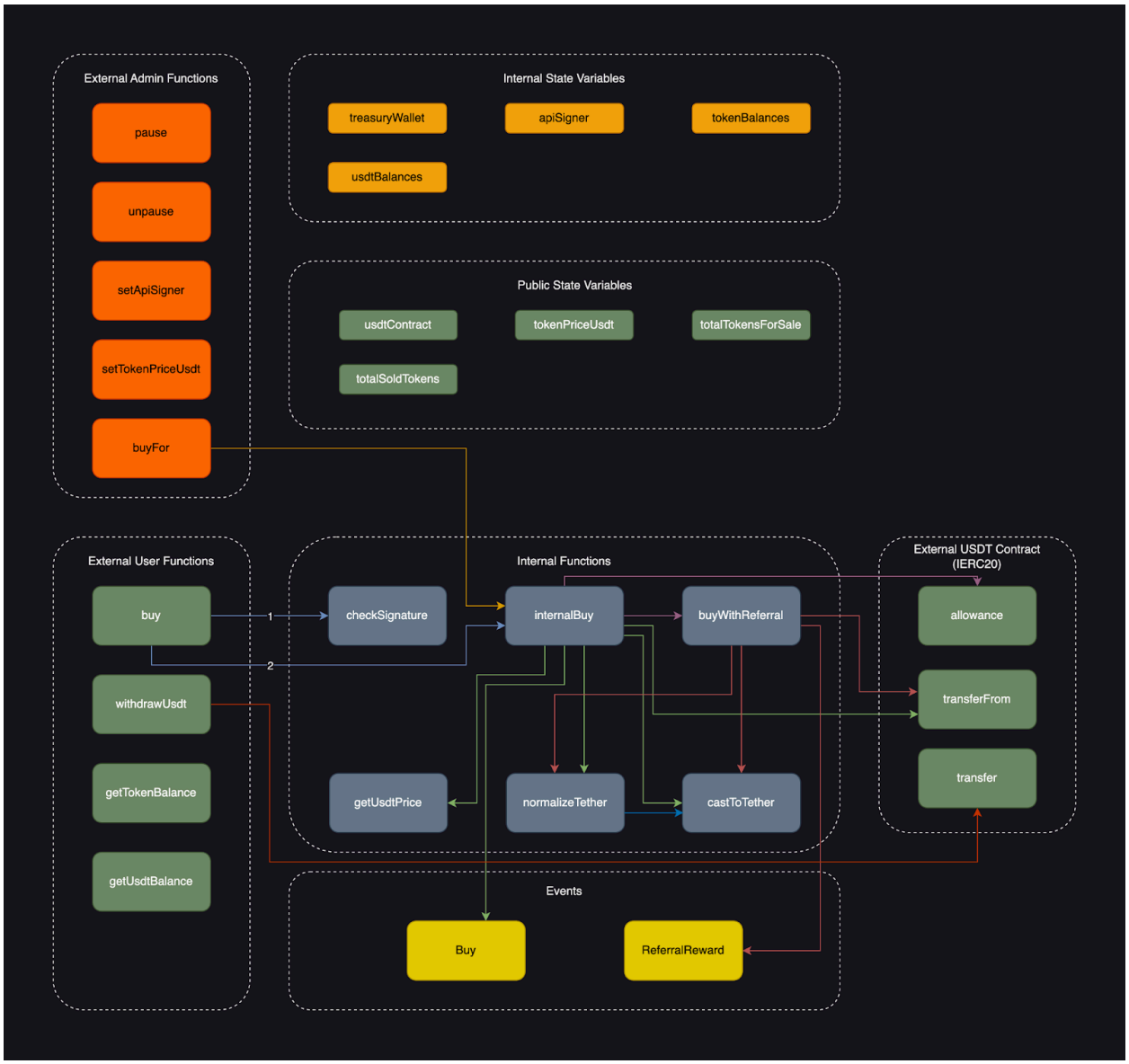
More information about this file's configuration can be found in the Hardhat Documentation.

The **hardhat.validate.ts** file contains the validation rules for the variables found in the **.env** file.

Finally, the **./docs** folder contains the project documentation, including this file. Autogenerated documentation can be found in the **api** folder.

2.1. Architecture Overview

The following chart provides a general view of the TokenSale contract structure and interactions between different functions.



2.2.1. TokenSale.sol

A token sale contract that accepts payments in USDT tokens in exchange for project tokens, which users will receive later when they are issued. The contract supports discounts and referral rewards in USDT that can be

immediately withdrawn. The contract is designed to support upgradeability via the transparent proxy pattern.

A typical flow looks as follows:

Contract deployment

1. The contract is deployed, specifying the following parameters:
    - **_admin** - An address of an admin wallet. This wallet will receive the **DEFAULT_ADMIN_ROLE** role and will be able to execute all the admin functions shown in the diagram above.
    - **_usdtContract** - An address of a USDT (Tether ERC20) contract.
    - **_treasuryWallet** - An address of a treasury wallet that USDT tokens will be transferred to after each purchase.
    - **_apiSigner** - An address of a wallet responsible for signing purchase parameters.
    - **_totalTokensForSale** - Total amount of tokens the contract may sell.
    - **_tokenPriceUsdt** - Token price in USDT. 18 decimals should be used here.

Purchase tokens

2. A user asks a backend service to produce and sign parameters for the **buy()** contract function, approves the contract to withdraw USDT from their balance, and calls the **buy()** contract function to make a token purchase. If a user is eligible for a purchase discount, or if a user was invited by another user and, therefore, participates in the referral program, the backend script will take all of this into account and include it in the parameters. The parameters and their signature are then validated by the contract according to the "EIP-712: Typed structured data hashing and signing" standard. If the user changes any of the parameters they got from the backend service, the **buy()** contract function will revert.

Check token balance

3. A user can check their token balance using the **getTokenBalance()** contract function. Also, a user can check their USDT reward balance received from their referrals by calling the **getUsdtBalance()** contract function.

Withdraw USDT rewards from referrals

4. A user can withdraw USDT tokens received as rewards from their referral purchases by calling the **withdrawUsdt()** contract function.

2.2.1.1. Assets

The contract has the following structs:

- **BUY_PARAMS_TYPEHASH** - This constant contains the hash of the parameter types, checked when a user purchases tokens.
- **PERCENT_MULTIPLIER** - This constant contains the multiplier used for all percentage values in all function call arguments.
- **treasuryWallet** - Wallet that will receive all incoming USDT from token purchases.
- **apiSigner** - Backend service wallet address that signs the purchase parameters.
- **usdtContract** - USDT (Tether ERC20) contract address.
- **tokenPriceUsdt** - Token price in USDT. 18 decimals is used here.
- **totalTokensForSale** - The total amount of tokens the contract can sell.
- **totalSoldTokens** - The total amount of sold tokens.
- **tokenBalances** - A mapping that contains token balances for each user.
- **usdtBalances** - A mapping that contains USDT referral reward balances for each user.

2.2.1.2. Events

The contract has the following events:

- **Buy** - Emitted when tokens are purchased by a user.

- **ReferralReward** - Emitted when a user receives a USDT reward from their referral user purchases.
- **WithdrawUsdt** - Emitted when a user withdraws a USDT reward earned from their referrals.
- **ApiSignerUpdate** - Emitted when the **API Signer** address is updated by the admin.
- **TokenPriceUsdtUpdate** - Emitted when the USDT token price is updated by the admin.

2.2.1.3. Modifiers

The contract has the following modifiers:

- **whenNotPaused** - reverts a call to the function it was applied to if an admin has paused the contract.

2.2.1.3. Functions

- **initialize** - Initialization function.
- **pause** - Turns off the **buy()** and **withdrawUsdt()** functions. (Admin)
- **unpause** - Reverts the effects from the **pause()** function. (Admin)
- **setApiSigner** - Update the address of the **API Signer** wallet. (Admin)
- **setTokenPriceUsdt** - Update token price in USDT. 18 decimals should be used here. (Admin)
- **buyFor** - Buy tokens for someone else. A discount of up to 10% can be specified here, but the referral program is unavailable. (Admin)
- **buy** - Purchase tokens. Call arguments must be provided and signed by the **API Signer**.
- **checkSignature** - (internal) Checks that the arguments passed to the **buy()** function were signed by the **API Signer** and not tampered with.
- **buyWithReferral** - (internal) Calculates a referral reward and adds it to the referral user balance.
- **withdrawUsdt** - Withdraw USDT rewards received from referrals.
- **getTokenBalance** - Returns caller's token balance.
- **getUsdtBalance** - Returns caller's USDT reward balance. Uses 18 decimals.

- **getUsdtPrice** - (internal) Returns token price in USDT. Uses 18 decimals.
- **normalizeTether** - Floors the given amount to the 6th decimal. Accepts an 18-decimal number and returns an 18-decimal number.
- **castToTether** - Cast the given 18-decimal number to a 6-decimal number. Returns a number with 6 decimals.