Ethereum Tokens

Tokens are simply numbers assigned to addresses, stored and manipulated by Etherium transactions.

The mapping from wallet address to account balance is stored in smart contract state.

Anyone can create their own cryptocurrency in Etherium by deploying their own smart contract storing the tokens and giving access to them accurding to fixed logic.

Transactions manipulate tokens by calling smart contract's methods.

ERC20 Tokens

ERC20 is a specification of most common token smart contract API, it is compatible with Etherium wallet software and exchange services. Tokens can be approved to spent by third party.

ERC20 tokens most popular for ICO. Proposed by Fabian Vogelsteller in late 2015.

Specifiication:
https://eips.ethereum.org/EIPS/eip-20

Example implementations:

1. https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol (8k+ GitHub stars)

ERC20.sol – implementation
IERC20.sol – token interface

2.
https://github.com/ConsenSys/Tokens/blob/master/contracts/eip20/EIP20.sol (1k+ GitHub stars)

ERC20 Token Interface IERC20.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);
```

ERC20 Token Interface IERC20.sol

```
/**
    * @dev Moves `amount` tokens from the caller's account to `recipient`.
    *
    * Returns a boolean value indicating whether the operation succeeded.
    *
    * Emits a {Transfer} event.
    */
   function transfer(address recipient, uint256 amount) external returns (bool);

   /**
    * @dev Returns the remaining number of tokens that `spender` will be
    * allowed to spend on behalf of `owner` through {transferFrom}. This is
    * zero by default.
    *
    * This value changes when {approve} or {transferFrom} are called.
    */
   function allowance(address owner, address spender) external view returns
(uint256);
```

ERC20 Token Interface IERC20.sol

```
/**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);
```

ERC20 Token Interface IERC20.sol

```
/**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);
```

ERC20 Token Interface IERC20.sol

```
    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256
value);
}
```

ERC20 Token Implementation

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

import "../../GSN/Context.sol";
import "./IERC20.sol";
import "../../math/SafeMath.sol";
import "../../utils/Address.sol";

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20MinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
```

ERC20 Token Implementation

```
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;
```

ERC20 Token Implementation

```
/**
   * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
   * a default value of 18.
   *
   * To select a different value for {decimals}, use {_setupDecimals}.
   *
   * All three of these values are immutable: they can only be set once during
   * construction.
   */
  constructor (string memory name, string memory symbol) public {
      _name = name;
      _symbol = symbol;
      _decimals = 18; // Real stored value is multiplied by 10^18
  }
```

ERC20 Token Implementation

```
/**
   * @dev Returns the name of the token.
   */
  function name() public view returns (string memory) {
      return _name;
  }

  /**
   * @dev Returns the symbol of the token, usually a shorter version of the
   * name.
   */
  function symbol() public view returns (string memory) {
      return _symbol;
  }

  /**
   * @dev Returns the number of decimals used to get its user representation.
   * For example, if `decimals` equals `2`, a balance of `505` tokens should
   * be displayed to a user as `5,05` (`505 / 10 ** 2`).
   *
   * Tokens usually opt for a value of 18, imitating the relationship between
   * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
   * called.
   *
   * NOTE: This information is only used for _display_ purposes: it in
   * no way affects any of the arithmetic of the contract, including
   * {IERC20-balanceOf} and {IERC20-transfer}.
   */
  function decimals() public view returns (uint8) {
      return _decimals;
  }
```

ERC20 Token Implementation

```
/**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }
```

ERC20 Token Implementation

```
/**
   * @dev See {IERC20-transfer}.
   *
   * Requirements:
   *
   * - `recipient` cannot be the zero address.
   * - the caller must have a balance of at least `amount`.
   */
  function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
      _transfer(_msgSender(), recipient, amount);
      return true;
  }
```

ERC20 Token Implementation

```
/**
 * @dev See {IERC20-allowance}.
 */
function allowance(address owner, address spender) public view virtual override returns (uint256) {
    return _allowances[owner][spender];
}

/**
 * @dev See {IERC20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function approve(address spender, uint256 amount) public virtual override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

ERC20 Token Implementation

```
    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }
```

ERC20 Token Implementation

```
  /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }


/**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance
below zero"));
        return true;
    }
```

ERC20 Token Implementation

```
/**
 * @dev Moves tokens `amount` from `sender` to `recipient`.
 *
 * This is internal function is equivalent to {transfer}, and can be used to
 * e.g. implement automatic token fees, slashing mechanisms, etc.
 *
 * Emits a {Transfer} event.
 *
 * Requirements:
 *
 * - `sender` cannot be the zero address.
 * - `recipient` cannot be the zero address.
 * - `sender` must have a balance of at least `amount`.
 */
function _transfer(address sender, address recipient, uint256 amount) internal virtual {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    _beforeTokenTransfer(sender, recipient, amount);

    _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
    _balances[recipient] = _balances[recipient].add(amount);
    emit Transfer(sender, recipient, amount);
}
```

ERC20 Token Implementation

```
/** @dev Creates `amount` tokens and assigns them to `account`, increasing
 * the total supply.
 *
 * Emits a {Transfer} event with `from` set to the zero address.
 *
 * Requirements
 *
 * - `to` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

    _beforeTokenTransfer(address(0), account, amount);

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
}
```

ERC20 Token Implementation

```
/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * Requirements
 *
 * - `account` cannot be the zero address.
 * - `account` must have at least `amount` tokens.
 */
function _burn(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: burn from the zero address");

    _beforeTokenTransfer(account, address(0), amount);

    _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}
```

ERC20 Token Implementation

```
/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
 *
 * This is internal function is equivalent to `approve`, and can be used to
 * e.g. set automatic allowances for certain subsystems, etc.
 *
 * Emits an {Approval} event.
 *
 * Requirements:
 *
 * - `owner` cannot be the zero address.
 * - `spender` cannot be the zero address.
 */
function _approve(address owner, address spender, uint256 amount) internal virtual {
    require(owner != address(0), "ERC20: approve from the zero address");
    require(spender != address(0), "ERC20: approve to the zero address");

    _allowances[owner][spender] = amount;
    emit Approval(owner, spender, amount);
}
```

ERC20 Token Implementation

```
    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }
}
```

ERC20 Token: How to Use

```
pragma solidity ^0.6.0;

import "openzeppelin-contracts/contracts/token/ERC20/ERC20.sol";

contract GLDToken is ERC20 {
    constructor(uint256 initialSupply) ERC20("Gold", "GLD") public {
        _mint(msg.sender, initialSupply*10**uint256(18));
    }
}
```

See https://docs.openzeppelin.com/contracts/3.x/erc20

ERC20 Token: Complie

```
snap refresh solc --channel=latest/beta

git clone https://github.com/OpenZeppelin/openzeppelin-contracts.git
cd openzeppelin-contracts
git checkout remotes/origin/release-v3.0.0
cd ..
solc GLDToken.sol --combined-json abi,bin --pretty-json > solc-output-token.json
```

ERC20 Token: Deploy

```
const fs = require("fs");
const Web3 = require('web3');

// Create a web3 connection to a running geth node over JSON-RPC running at
// http://localhost:3334
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));

// Read the compiled contract code
// Compile with
// solc GLDToken.sol --combined-json abi,bin --pretty-json > solc-output-token.json
let source = fs.readFileSync("solc-output-token.json");
let contracts = JSON.parse(source)["contracts"];

// ABI description as JSON structure
let abi = JSON.parse(contracts["GLDToken.sol:GLDToken"].abi);

// Smart contract EVM bytecode as hex
let code = '0x' + contracts["GLDToken.sol:GLDToken"].bin;

var myContract = new web3.eth.Contract(abi);

myContract.deploy({
    data: code,
    arguments: [1000000] // Token initialSupply
})
.send({
    from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230',
    gas: 1500000,
    gasPrice: '30000000000000'
}, function(error, transactionHash){ })
.on('error', function(error){ })
.on('transactionHash', function(transactionHash){ })
.on('receipt', function(receipt){
   console.log(receipt.contractAddress) // contains the new contract address
})
.on('confirmation', function(confirmationNumber, receipt){  })
.then(function(newContractInstance){
    console.log(newContractInstance.options.address) // instance with the new contract address
});
```

ERC20 Token: Deploy Result

Command:
node ./deploy-contract-token.js

Node JS Output:
0x86191EFF97E95706C630C251E44D17533b4C221E
0x86191EFF97E95706C630C251E44D17533b4C221E

Geth Output:
INFO [05-20|13:52:26.084] Submitted contract creation fullhash=0x8f62a461713684d8c18c6245b7eb6df902e31d044fddbdf77c2e5c490f8ce888 contract=0x86191EFF97E95706C630C251E44D17533b4C221E
INFO [05-20|13:52:26.084] Commit new mining work number=19 sealhash=1498c3…4e45ee uncles=0 txs=1 gas=1110580 fees=33.3174 elapsed=712.247µs
INFO [05-20|13:52:26.086] Successfully sealed new block number=19 sealhash=1498c3…4e45ee hash=a5d361…8e9a59 elapsed=1.568ms
INFO [05-20|13:52:26.086] mined potential block number=19 hash=a5d361…8e9a59
INFO [05-20|13:52:26.087] Commit new mining work number=20 sealhash=dfc1f6…a1d878 uncles=0 txs=0 gas=0 fees=0 elapsed=871.644µs
INFO [05-20|13:52:26.087] Sealing paused, waiting for transactions

ERC20 Token: Call to get Balance

```
const fs = require("fs");
const Web3 = require('web3');

// Create a web3 connection to a running geth node over JSON-RPC running at
// http://localhost:3334
const web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:3334"));

// Read the compiled contract code
// Compile with
// solc GLDToken.sol --combined-json abi,bin --pretty-json > solc-output-token.json
let source = fs.readFileSync("solc-output-token.json");
let contracts = JSON.parse(source)["contracts"];

// ABI description as JSON structure
let abi = JSON.parse(contracts["GLDToken.sol:GLDToken"].abi);

var myContract = new web3.eth.Contract(abi, '0x86191EFF97E95706C630C251E44D17533b4C221E');

// Pass address of the contract owner.
// Will output 1000000.
myContract.methods.balanceOf("0xd6e6d7270d84c361bcd91047da65cbfc01157230").call()
  .then(console.log);
```

ERC20 Token: Call to get Balance Output

Command:
node call-contract-token.js

NodeJS Output:
100000000000000000000000

Geth Output:
<none>

ERC20 Token: Transaction to Transfer

```
...

var myContract = new web3.eth.Contract(abi, '0x86191EFF97E95706C630C251E44D17533b4C221E');

// Send transaction to transfer 1000 Tokens to address.
// Use Web3.utils.toWei to compile 18 decimal token amount to long string.
var amount = 1000;
var tokens = Web3.utils.toWei(amount.toString(), 'ether');

// Send transaction to transfer 1000 * 10 ^ 18 Tokens to address.
myContract.methods.transfer("0xe72055Cc2F4E6555908Ab404aa5EDF6ecCAa66AF", tokens).send({
  from: '0xd6e6d7270d84c361bcd91047da65cbfc01157230', // The address the transaction should be sent from.
  gas: 1500000, // (optional): The maximum gas provided for this transaction (gas limit).
  gasPrice: '30000000000000'  // (optional): The gas price in wei to use for this transaction.
})
.on('transactionHash', function(hash){
  console.log("hash");
  console.log(hash);
})
.on('receipt', function(receipt){
  console.log("receipt");
  console.log(receipt);
})
.on('confirmation', function(confirmationNumber, receipt){
  console.log("confirmation");
  console.log(receipt);

  // Check the result.
  myContract.methods.balanceOf("0xd6e6d7270d84c361bcd91047da65cbfc01157230").call()
    .then(console.log);
  myContract.methods.balanceOf("0x97a4f1fd5ff01b5cf056cc5c1fb44b2abbdb99cb").call()
    .then(console.log);
})
.on('error', console.error);
```

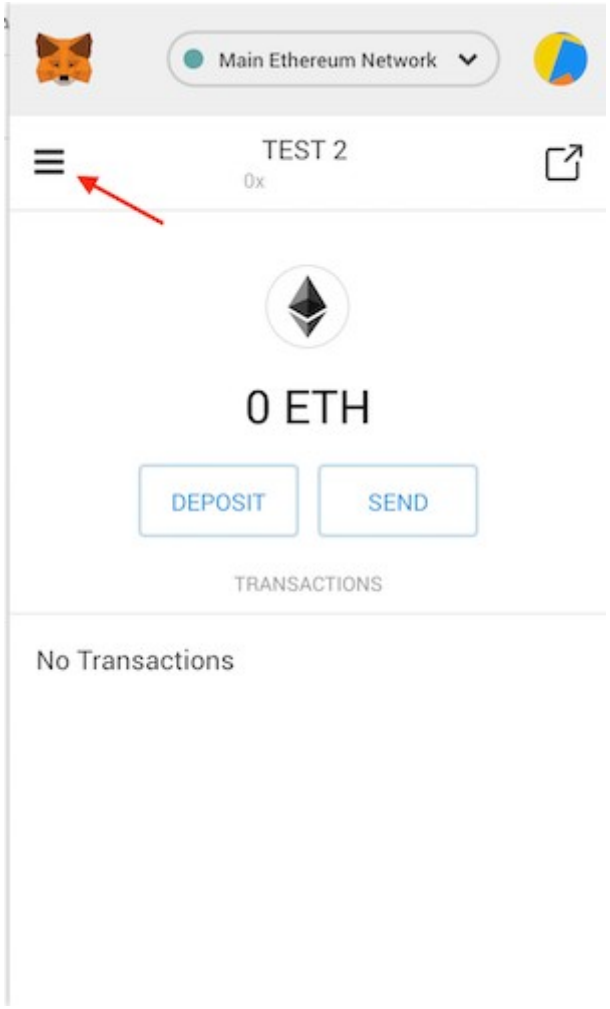
ERC20 Token: Transaction to Transfer Output

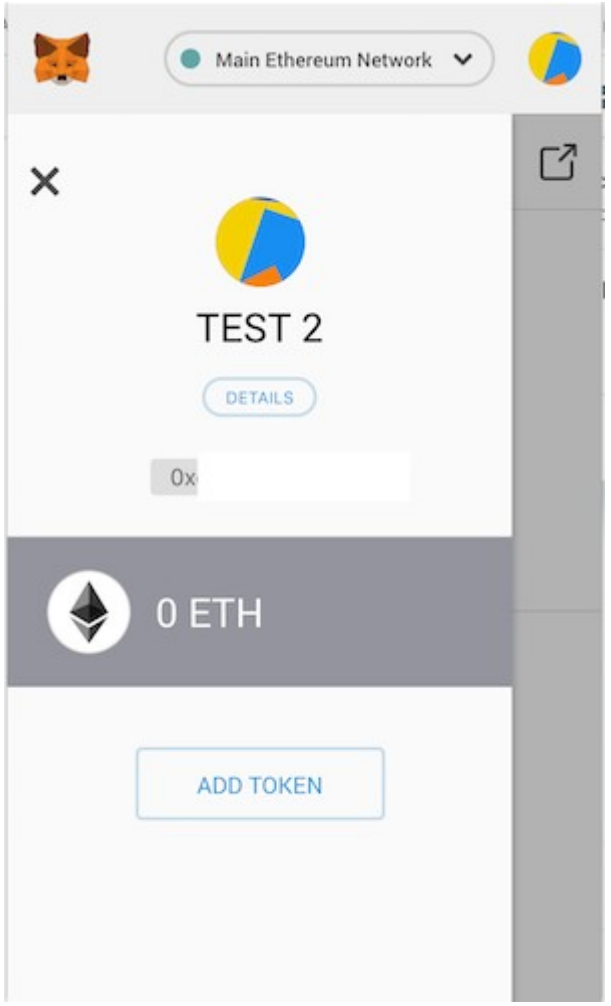
Command:
node call-contract-transaction-token.js

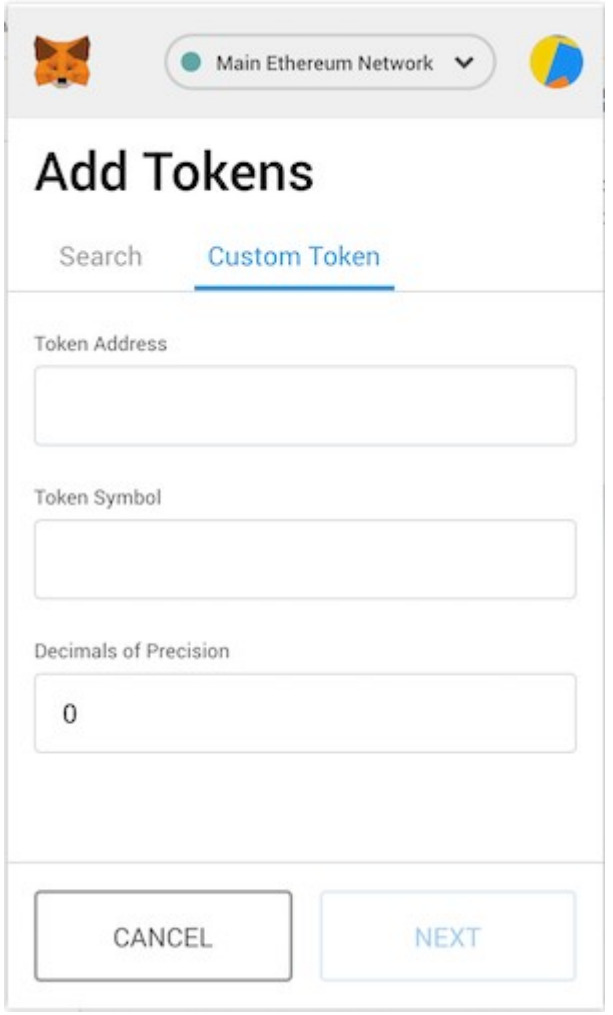
NodeJS Output:
hash
0x068f103701d84a163ec94efa5c40c8b73866a46875b407047f6b9a1fa67d45c0
receipt
…
Confirmation
…
99900000000000000000000000
10000000000000000000000

Geth Output:
INFO [05-20|14:18:30.322] Submitted transaction ...

Metamask add custom Token

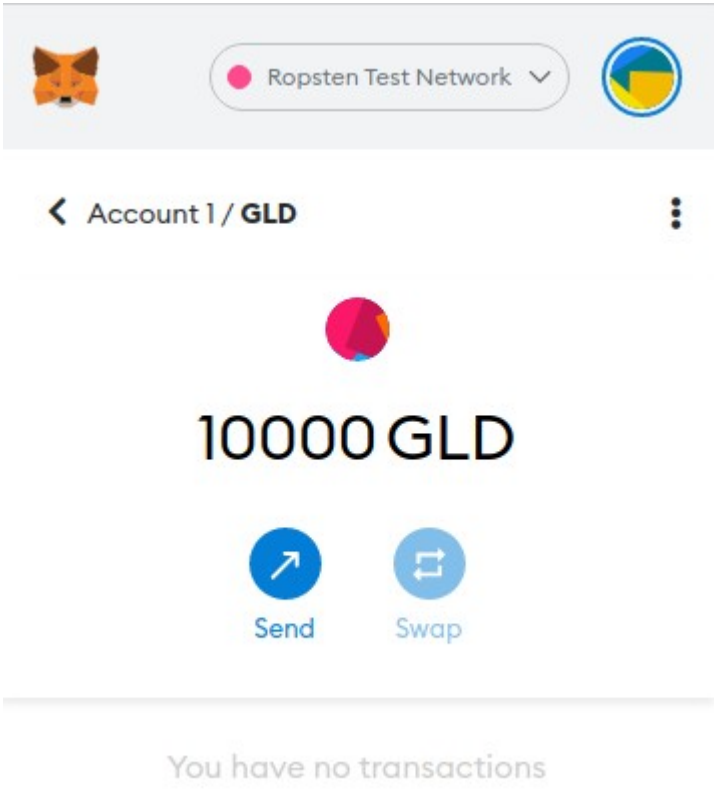






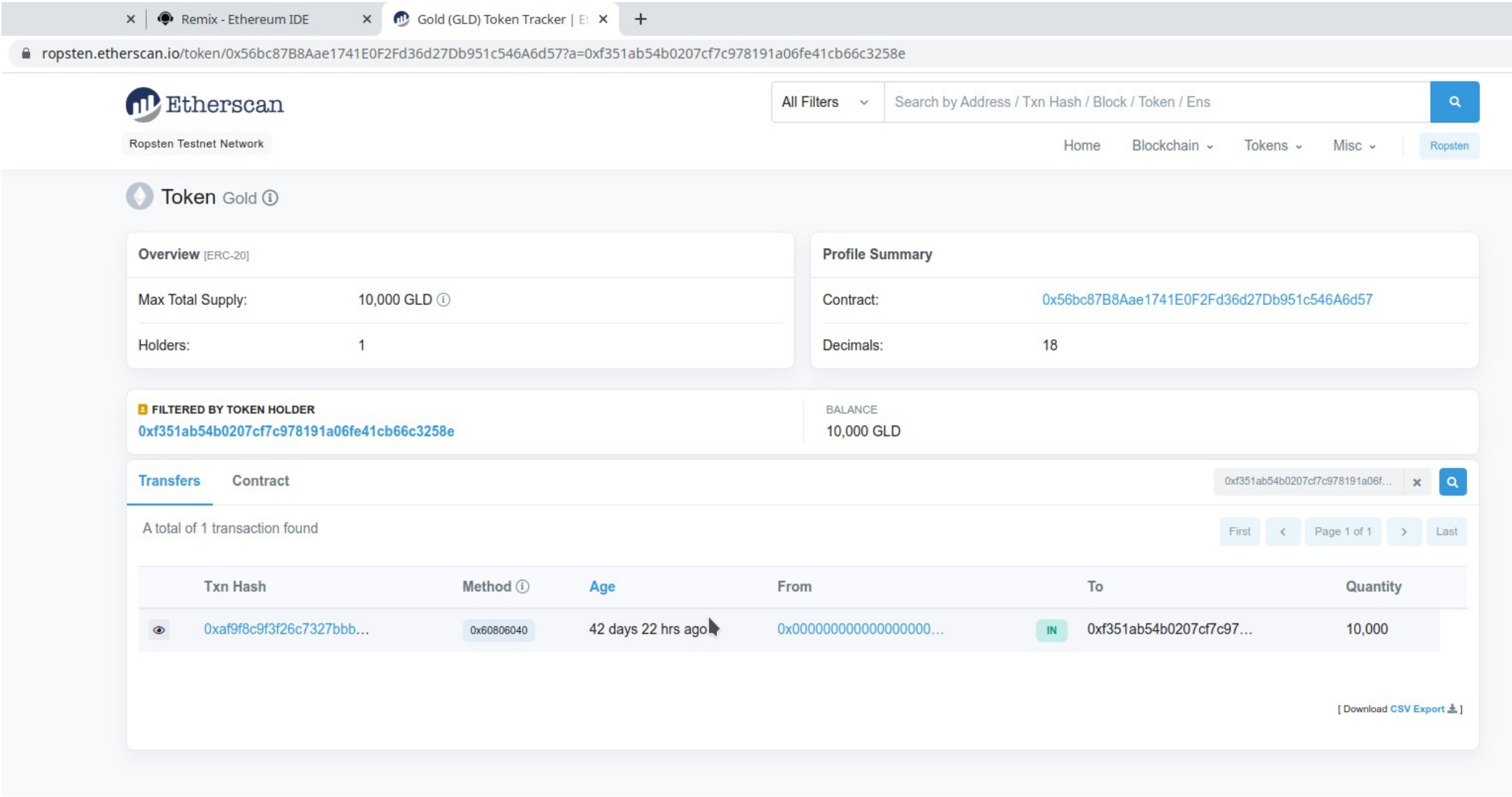
See https://metamask.io/

Metamask add custom Token



See https://metamask.io/

Etherscan Token Details



Information about Cryptocurrency Exchange and Stock Market development and operation funding

Cryptocurrency and trading systems development expert with 12 years experience, Sergey Mereutsa:

«The development cost is starting from $500k USD, 1+ year work for 6-10 person team.»

«The Liquidity investment to fill the Depth of Market starts from $10M USD. And the Exchange will not even be in top 100.»

«Trading Platforms and Stock Market engines are usually written in C/C++/Rust. Java is rarely used as JVM it not always predictable due to GC and Java may decide to perform its memory keeping operations unexpectedly.»

«Stock Market speculations are not directly related to Blockchain because internal trading transactions are inside the engine and are not on the Blockchain and only input and output from the exchange are done via Blockchain transactions programmatically.»

Links

https://eips.ethereum.org/EIPS/eip-20

https://github.com/ConsenSys/Tokens/blob/fdf687c69d998266a95f15216b1955a4965a0a6d/contracts/eip20/EIP20.sol

https://medium.com/ethex-market/erc20-approve-allow-explained-88d6de921ce9

https://medium.com/blockchannel/the-anatomy-of-erc20-c9e5c5ff1d02

https://www.wealdtech.com/articles/understanding-erc20-token-contracts/

https://hashnode.com/post/how-to-build-your-own-ethereum-based-erc20-token-and-launch-an-ico-in-next-20-minutes-cjbcpwzec01c93awtbij90uzn

Book:

'Ethereum for Web Developers' by Santiago Palladino (https://www.apress.com/gp/book/9781484252772)