

Лабораторная работа 3. Токены ERC-20

Цель

Познакомиться со структурой токенов ERC-20 и библиотекой смарт-контрактов [OpenZeppelin](#).

Шаги:

1. Создать смарт-контракт токена ERC-20
2. Модифицировать смарт-контракт для чеканки монет
3. Развернуть контракт на блокчейне
4. Создать пользовательский интерфейс для взаимодействия со смарт-контрактом

Замечание о связи с предыдущей работой

Данная лабораторная в части технических навыков базируется на лабораторной работе 2. Вы можете ожидать некоторого дублирования шагов.

Замечание об окружении

Ниже подразумевается, что вы работаете в UNIX-подобной системе: Linux, FreeBSD, MacOS, и т.п. Если вы работаете с Microsoft Windows, вам потребуется дополнительно установить [Windows Subsystem for Linux](#) или использовать виртуальную машину с Linux через [Vagrant](#), [VirtualBox](#) и др.

1. Создать смарт-контракт токена ERC-20

Для выполнения лабораторной работы предлагается использовать набор инструментов разработки Truffle. Для его выполнения требуется установить на компьютер последнюю LTS версию Node.js и NPM (поставляется вместе с Node.js) с официального сайта: <https://nodejs.org/ru/>

Когда Node.js установлена, следует установить truffle:

```
npm install -g truffle
```

Truffle создаёт проект в текущей папке. Создадим новую папку для контракта, и перейдём в неё:

```
mkdir token-contract
cd token-contract
```

Воспользуемся уже [существующим шаблоном \(или Truffle Box\)](#) для создания контрактов на сети Polygon:

```
truffle unbox polygon
```

Команда создаст в текущей папке минимальный набор файлов для начала работы:

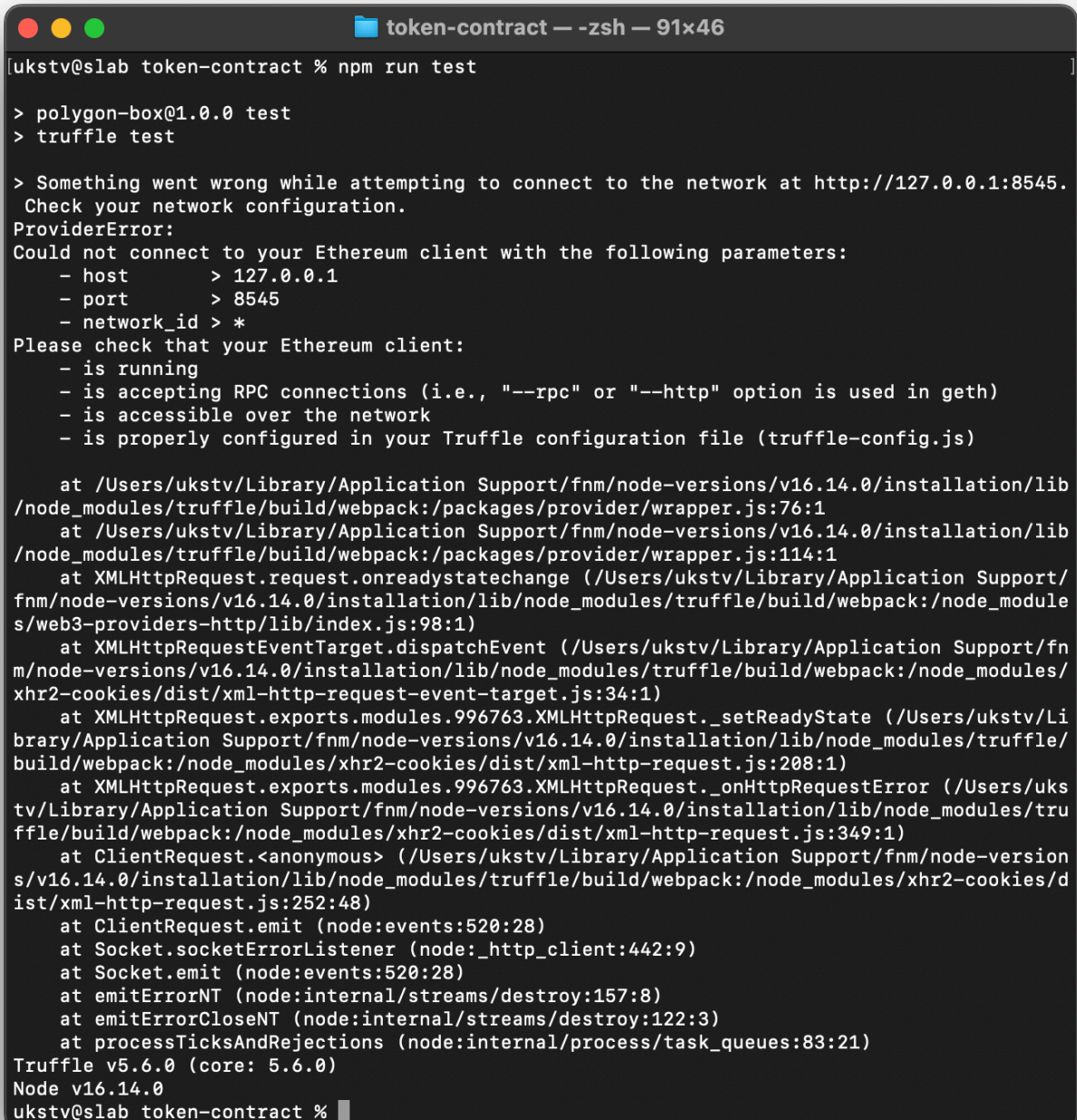
- `contracts` - содержит код смарт-контрактов
 - `ethereum` - код смарт-контрактов для Ethereum (нам не потребуется)
 - `polygon` - код смарт-контрактов для сети Polygon

- `migrations` - набор файлов для разворачивания и/или изменения разрабатываемых смарт-контрактов
- `test` - содержит тесты.

Чтобы проверить корректность работы, запустим тесты:

```
npm run test
```

Они упадут с ошибкой:



```
token-contract — -zsh — 91x46
[ukstv@slab token-contract % npm run test

> polygon-box@1.0.0 test
> truffle test

> Something went wrong while attempting to connect to the network at http://127.0.0.1:8545.
Check your network configuration.
ProviderError:
Could not connect to your Ethereum client with the following parameters:
  - host      > 127.0.0.1
  - port      > 8545
  - network_id > *
Please check that your Ethereum client:
  - is running
  - is accepting RPC connections (i.e., "--rpc" or "--http" option is used in geth)
  - is accessible over the network
  - is properly configured in your Truffle configuration file (truffle-config.js)

    at /Users/ukstv/Library/Application Support/fnm/node-versions/v16.14.0/installation/lib
/node_modules/truffle/build/webpack:/packages/provider/wrapper.js:76:1
    at /Users/ukstv/Library/Application Support/fnm/node-versions/v16.14.0/installation/lib
/node_modules/truffle/build/webpack:/packages/provider/wrapper.js:114:1
    at XMLHttpRequest.request.onreadystatechange (/Users/ukstv/Library/Application Support/
fnm/node-versions/v16.14.0/installation/lib/node_modules/truffle/build/webpack:/node_module
s/web3-providers-http/lib/index.js:98:1)
    at XMLHttpRequestEventTarget.dispatchEvent (/Users/ukstv/Library/Application Support/fn
m/node-versions/v16.14.0/installation/lib/node_modules/truffle/build/webpack:/node_modules/
xhr2-cookies/dist/xml-http-request-event-target.js:34:1)
    at XMLHttpRequest.exports.modules.996763.XMLHttpRequest._setReadyState (/Users/ukstv/Li
brary/Application Support/fnm/node-versions/v16.14.0/installation/lib/node_modules/truffle/
build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:208:1)
    at XMLHttpRequest.exports.modules.996763.XMLHttpRequest._onHttpRequestError (/Users/ukst
v/Library/Application Support/fnm/node-versions/v16.14.0/installation/lib/node_modules/tru
ffle/build/webpack:/node_modules/xhr2-cookies/dist/xml-http-request.js:349:1)
    at ClientRequest.<anonymous> (/Users/ukstv/Library/Application Support/fnm/node-version
s/v16.14.0/installation/lib/node_modules/truffle/build/webpack:/node_modules/xhr2-cookies/d
ist/xml-http-request.js:252:48)
    at ClientRequest.emit (node:events:520:28)
    at Socket.socketErrorListener (node:_http_client:442:9)
    at Socket.emit (node:events:520:28)
    at emitErrorNT (node:internal/streams/destroy:157:8)
    at emitErrorCloseNT (node:internal/streams/destroy:122:3)
    at processTicksAndRejections (node:internal/process/task_queues:83:21)
Truffle v5.6.0 (core: 5.6.0)
Node v16.14.0
ukstv@slab token-contract %
```

Ошибка говорит, что Truffle для тестирования попытался связаться с локальным узлом блокчейна, а тот оказался недоступен. В самом деле, мы не запускали локальный узел блокчейна. Для его запуска нам потребуется ещё один пакет [Ganache](#):

```
npm install -g ganache
```

Теперь в соседнем терминале, в той же папке `token-contract` запустим локальный узел Ganache:

```
ganache
```

В консоль Ganache выведет диагностическую информацию:


```
token-contract — node ~/Library/Caches/fnm_multishells/69570_1669155702009/bi...
Last login: Wed Nov 23 01:06:21 on ttys000
lukstv@slab token-contract % ganache
ganache v7.5.0 (@ganache/cli: 0.6.0, @ganache/core: 0.6.0)
Starting RPC server

Available Accounts
=====
(0) 0x385fbe6Ce8304214cf87C822ac207675F3f13300 (1000 ETH)
(1) 0x8de05068e74a29782641c9301436AB004b42EB58 (1000 ETH)
(2) 0x013B15b35d4852bB91543271e07D07D61F821B21 (1000 ETH)
(3) 0x5297E00e12bcf63A5E0e331C044Ef9708ADB9Eb7 (1000 ETH)
(4) 0x9Aca9a55DCf9a4689CDBcB2Ba20d5CD62C2846C1 (1000 ETH)
(5) 0xCd1e2E2a73b7fc8Ee14EF02b3eA9686654d16690 (1000 ETH)
(6) 0xC2E26f5CD9E74a9fA9F2E6965ccC47c7F5E4A9AC (1000 ETH)
(7) 0xEaABA751199Ebe3f87fA86D399776A702d7C59fd (1000 ETH)
(8) 0x4092eF511d0c2F808Be8DC996eF2F0FAd9950c3d (1000 ETH)
(9) 0xCE9515A024d0C08A4aF2A7B42d41520360F9ddf6 (1000 ETH)

Private Keys
=====
(0) 0x2278875ed5fb754907167497915ed1f00bdef767c5b04fe6c236c46d2499e5cc
(1) 0x14330eb5d4c0eac42f76f2f179acb9ab51a220d103fc967cd358ed6a54dfaf7b
(2) 0xf91145a3f744af9c37d62691ca1154f51cc9d9569e867e4fb9694106ff7dc767
(3) 0x0abe25310477f75fb4fd5c8e7ecbc80ffd7d48bbab3668145bbd0840bbb08099
(4) 0xc0f3a5a9a016c2c31bfa539c8eaae4907dfe6c2cbf99aebb98221ea992d6bb19
(5) 0x6166a083ccce7f172ab6dfe3d2c941a193180f886521d45f4447de031d17480a
(6) 0x738bd59a451c4c6891f70b69ebd94235e25787db71a19c28ef460cae5266e9a9
(7) 0x7df14d853618fb3db41e1c983491df2071f79238bf19b41a8a946f0a1b6a1f02
(8) 0xfce2506b19d073ca321a91e280ab7824ae595f38a1aa3b4487c712f504b14d41
(9) 0x35686fb77006bb177dc918a5aff8d2e37e753e26262f78c2ff7e5781a4cfcc70

HD Wallet
=====
Mnemonic:      winter section walnut spoil multiply midnight carbon tribe knife law climb t
ank
Base HD Path:  m/44'/60'/0'/0/{account_index}

Default Gas Price
=====
2000000000

BlockGas Limit
=====
30000000

Call Gas Limit
=====
50000000

Chain Id
=====
1337

RPC Listening on 127.0.0.1:8545
```

Теперь мы можем вернуться в первый терминал. Запустим тесты ещё раз: `npm run test`. В терминале должен появиться вывод с сообщением об успешных тестах:

```
token-contract — -zsh — 91x22
[ukstv@slab token-contract % npm run test

> polygon-box@1.0.0 test
> truffle test

Compiling your contracts...
=====
> Compiling ./contracts/ethereum/SimpleStorage.sol
> Artifacts written to /var/folders/3x/0gdz5cvn1_l6nls9vn00qrv0000gn/T/test--69672-7ZZTYzn
2eZou
> Compiled successfully using:
  - solc: 0.5.16+commit.9c3226ce.Emscripten.clang

Contract: SimpleStorage
  ✓ ...should store the value 89.

1 passing (31ms)
ukstv@slab token-contract %
```

Теперь время создавать свой смарт-контракт. К несчастью, используемый нами шаблон содержит некоторое количества мусора. Давайте его выкинем. Для этого, перенесите имеющийся контракт `SimpleStorage` в папку `contracts` и удалите ненужные папки внутри `contracts`:

```
mv ./contracts/polygon/SimpleStorage.sol ./contracts
rm -rf ./contracts/ethereum ./contracts/polygon
```

Удалите `truffle-config.js` и переименуйте `truffle-config.polygon.js` на `truffle-config.js`:

```
rm ./truffle-config.js
mv ./truffle-config.polygon.js ./truffle-config.js
```

Внутри конфигурационного файла `./truffle-config.js` вы можете увидеть, что он использует ссылки на Polygon-специфичные папки. Давайте заменим их на значения по умолчанию, удалив поля:

- `contracts_build_directory`: значение по умолчанию `./build/contracts`,
- `contracts_directory`: значение по умолчанию `./contracts`.

Кроме того, имеет смысл переименовать названия сетей:

- `polygon_infura_testnet` → `polygon_mumbai`,
- `polygon_infura_mainnet` → `polygon_matic`.

Дополнительно следует убрать ссылки на Polygon-специфичный конфигурационный файл из `package.json`. Уберите все строки вида `--config=truffle-config.polygon.js` и переименуйте элементы в поле `scripts` так, чтобы они не ссылались на Polygon. Должно получиться такое содержание поля `scripts`:

```
{
  "test": "truffle test --network=$npm_config_network",
  "compile": "truffle compile",
  "migrate": "truffle migrate --network=$npm_config_network"
}
```

Также следует обновить используемую версию компилятора Solidity. Для этого в секции `compilers.solc` следует добавить поле `version`. Должно получиться примерно такое:

```
compilers: {
  solc: {
    version: "0.8.17"
  }
}
```

Ввиду обновленной версии компилятора, измените также требуемую версию в существующем контракте `SimpleStorage.sol`:

```
pragma solidity >=0.4.21 <0.9.0;
```

Мы будем создавать наш контракт токена на основе существующей библиотеки смарт-контрактов [OpenZeppelin](#). Добавим библиотеку как зависимость в наш проект. В консоли, находясь в папке `token-contract` выполните команду:

```
npm add @openzeppelin/contracts
```

Теперь настало время создать контракт токена. Создайте файл `Token.sol` в папке `contracts` следующего вида:

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.17 <0.9.0;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
import "@openzeppelin/contracts/utils/Context.sol";

contract Token is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private balances;
    mapping(address => mapping(address => uint256)) private allowances;

    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint256 public totalSupply;

    constructor(string memory _name, string memory _symbol, uint256 _totalSupply) {
```

```

    name = _name;
    symbol = _symbol;
    totalSupply = _totalSupply;
}

function balanceOf(address account) public view override returns (uint256) {
    return balances[account];
}

function transfer(address to, uint256 amount) public override returns (bool) {
    address from = _msgSender();
    _transfer(from, to, amount);
    return true;
}

function allowance(address _owner, address _spender) public view override returns
(uint256) {
    return allowances[_owner][_spender];
}

function approve(address spender, uint256 amount) public override returns (bool) {
    address from = _msgSender();
    _approve(from, spender, amount);
    return true;
}

function transferFrom(
    address from,
    address to,
    uint256 amount
) public override returns (bool) {
    address spender = _msgSender();
    _spendAllowance(from, spender, amount);
    _transfer(from, to, amount);
    return true;
}

function _transfer(
    address from,
    address to,
    uint256 amount
) internal {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    uint256 fromBalance = balances[from];
    require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        balances[from] = fromBalance - amount;
    }
}

```

```

        balances[to] += amount;
    }

    emit Transfer(from, to, amount);
}

function _approve(
    address _owner,
    address _spender,
    uint256 _amount
) internal {
    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);
}

function _spendAllowance(
    address _owner,
    address _spender,
    uint256 _amount
) internal {
    uint256 currentAllowance = allowance(_owner, _spender);
    if (currentAllowance != type(uint256).max) {
        require(currentAllowance >= _amount, "ERC20: insufficient allowance");
        unchecked {
            _approve(_owner, _spender, currentAllowance - _amount);
        }
    }
}
}

```

Для подтверждения формальной корректности кода следует пропустить файлы через компилятор. В консоли выполните команду:

```
npm run compile
```

Должен появиться вывод, соответствующей безошибочной компиляции.


```
token-contract — -zsh — 80x24
[ukstv@slab token-contract % npm run compile]

> polygon-box@1.0.0 compile
> truffle compile

Compiling your contracts...
=====
> Compiling ./contracts/SimpleStorage.sol
> Compiling ./contracts/Token.sol
> Compiling @openzeppelin/contracts/token/ERC20/IERC20.sol
> Compiling @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
> Compiling @openzeppelin/contracts/utils/Context.sol
> Artifacts written to /Users/ukstv/Desktop/token-contract/build/contracts
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang
ukstv@slab token-contract %
```

Также следует создать файл миграции `migrations/2_token.js`:

```
const Token = artifacts.require("./Token.sol");

module.exports = function (deployer) {
  deployer.deploy(Token, "HelloToken", "HELL0", 10000n * BigInt(1e18));
};
```

Второй и дальше параметры передаются в конструктор контракта при развёртывании на блокчейн. Можете заменить значения переменных на любые по вашему желанию. Обратите внимание на третий параметр. В коде контракта указано, что у токена 18 знаков после запятой, но переменные баланса имеют целочисленный тип. Здесь мы задаём `totalSupply` в 10000 единиц токенов, и добавляем к ним соответствующее значение нулей "после запятой".

Внимательно прочитайте код контракта. Как вы можете видеть, этот `Token` полностью соответствует стандарту [ERC-20](#). Пытливый читатель также может заметить, что при развёртывании контракта на блокчейне, когда выполняется конструктор контракта, выставляется общее число токенов `totalSupply`, но баланс каких-либо аккаунтов не меняется. Фактически это означает, что мы создали `totalSupply` токенов, но никому их не можем раздать, и никто не может ими распоряжаться. Следует исправить этот недостаток.

2. Модифицировать смарт-контракт для чеканки монет

Недостаток контракта заключается в том, что никто не может распоряжаться созданным объёмом `totalSupply` токенов. Следует провести назначение баланса на аккаунт, который создал баланс. Заметим, что любое изменение баланса должно сопровождаться вызовом события `Transfer`. Посмотрим на наиболее похожую для наших целей функцию `_transfer`:

```
function _transfer(
    address from,
    address to,
    uint256 amount
) internal {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    uint256 fromBalance = balances[from];
    require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
    unchecked {
        balances[from] = fromBalance - amount;
        balances[to] += amount;
    }

    emit Transfer(from, to, amount);
}
```

Мы видим два ограничения: проверка адресов на не-нулевость, баланс отправителя должен быть больше, чем запрошенный. Из-за этих ограничений мы не можем использовать эту функцию из конструктора. Вместо этого волонтаристски назначим баланс аккаунту-создателю контракта (можно использовать `_msgSender()` для получения адреса создателя) и вызовем событие `Transfer`, в котором адрес "отправителя" равен `0` (для превращения числа 0 в адрес можно использовать конструкцию `address(0)`)

ЗАДАНИЕ Модифицировать контракт так, чтобы баланс аккаунта-создателя контракта был равен `totalBalance`, и при этом вызывалось событие `Transfer` с полем `from` равным 0.

Для проверки можно использовать следующий тест в файле `test/token.js`:

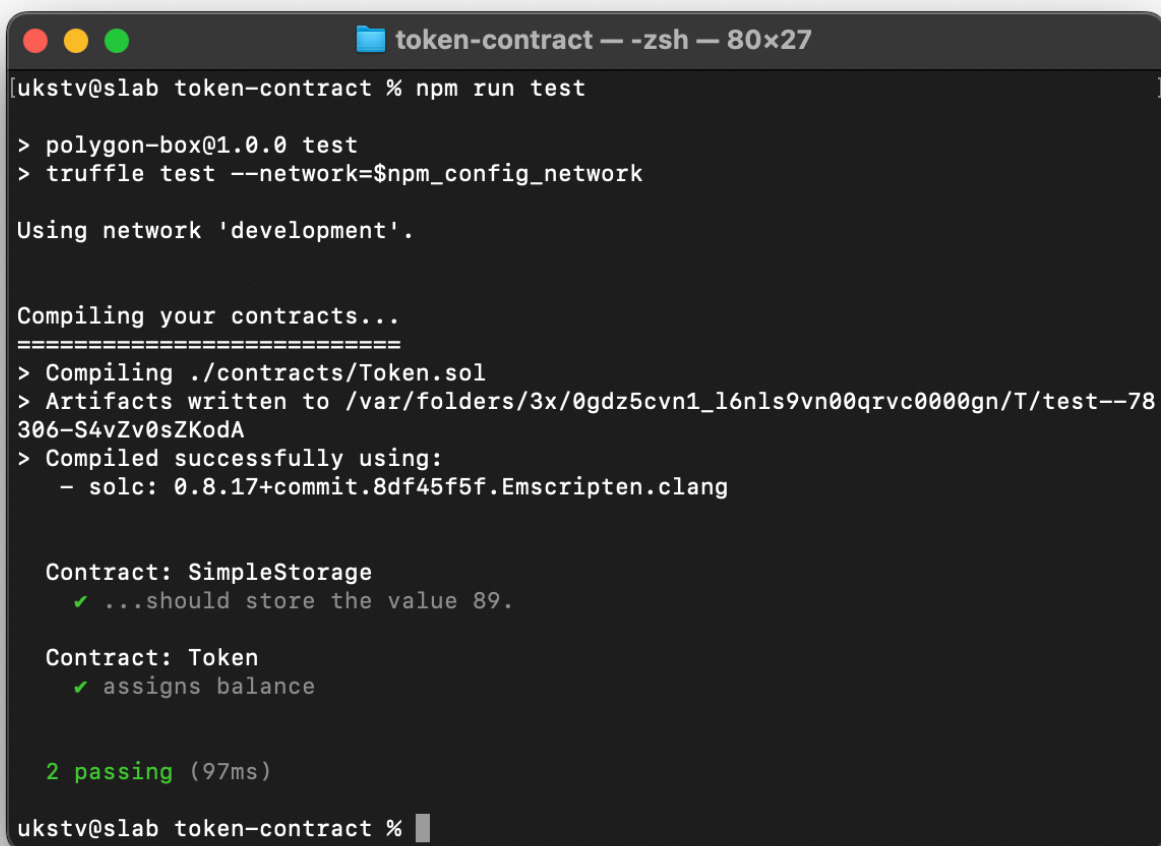
```

const Token = artifacts.require("./Token.sol");

contract("Token", accounts => {
  it("assigns balance when created", async () => {
    const totalBalance = 10000n * BigInt(1e18)
    const creator = accounts[0]
    const instance = await Token.new("HelloToken", "HELLO", totalBalance, {from:
creator})
    const creatorBalance = await instance.balanceOf.call(creator).then(BigInt)
    assert.equal(creatorBalance, totalBalance)
    const events = await instance.getPastEvents('Transfer')
    assert.equal(events.length, 1)
  });
});

```

Если после модификации контракта запустить тесты через `npm run test`, должен получиться следующий вывод в консоль:



```

token-contract — -zsh — 80x27
[ukstvt@slab token-contract % npm run test]

> polygon-box@1.0.0 test
> truffle test --network=$npm_config_network

Using network 'development'.

Compiling your contracts...
=====
> Compiling ./contracts/Token.sol
> Artifacts written to /var/folders/3x/0gdz5cvn1_l6nls9vn00qrv0000gn/T/test--78
306-S4vZv0sZKodA
> Compiled successfully using:
   - solc: 0.8.17+commit.8df45f5f.Emscripten.clang

Contract: SimpleStorage
  ✓ ...should store the value 89.

Contract: Token
  ✓ assigns balance

2 passing (97ms)
ukstvt@slab token-contract %

```

У нас вышел годный контракт, который не позволяет чеканить токены из воздуха. Давайте всё-таки добавим такую возможность.

Нам необходимо сделать так, чтобы только выделенный адрес, так называемый владелец контракта, мог чеканить токены. Назначим владельцем адрес, который развернул контракт, и позволим ему передавать права владения. Для этого добавим переменную `owner` к контракту, модифицируем конструктор и добавим несколько новых функций:

```
contract Token is Context, IERC20, IERC20Metadata {
    // ... предыдущее содержание
    address public owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor(string memory _name, string memory _symbol, uint256 _totalSupply) {
        // ... предыдущее содержание
        owner = _msgSender();
        emit OwnershipTransferred(address(0x0), owner);
    }

    modifier onlyOwner() {
        require(owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    function transferOwnership(address _newOwner) public virtual onlyOwner {
        address oldOwner = owner;
        owner = _newOwner;
        emit OwnershipTransferred(oldOwner, _newOwner);
    }

    // ... предыдущее содержание
}
```

Теперь можно приступать к чеканке.

ЗАДАНИЕ Добавить функцию `mint(address _account, uint256 amount)`. Функция должна добавлять `amount` к наличному `totalBalance`, должна вызывать событие `Transfer` с адреса `0` на адрес `_account`, должна добавлять `amount` к существующему балансу у `_account`. Функцию может вызывать только `owner` контракта.

Для проверки можно использовать следующий тест-кейс (добавить к тест-кейсам в `test/token.js`):

```
it("mint", async () => {
    // Mint amount
    const totalBalance = 10000n * BigInt(1e18)
    const creator = accounts[0]
    const instance = await Token.new("HelloToken", "HELLO", totalBalance, {from:
creator})
    const destination = accounts[1]
    const mintedAmount = 10n * BigInt(1e18)
    const tx = await instance.mint(destination, mintedAmount, {from: creator})
    assert.equal(tx.logs.length, 1)
```

```

const transferEvent = tx.logs[0]
assert.equal(transferEvent.event, 'Transfer')
assert.equal(transferEvent.args.from, '0x0000000000000000000000000000000000000000')
assert.equal(transferEvent.args.to, destination)
assert.equal(BigInt(transferEvent.args.value), mintedAmount)

// Only owner
try {
  await instance.mint(destination, mintedAmount, {from: destination})
  assert.fail("unreachable")
} catch (e) {
  assert.match(e.message, /Ownable: caller is not the owner/)
}
});

```

3. Развернуть контракт на блокчейне

Воспользуйтесь шагами из лабораторной работы 2 для разворачивания контракта на блокчейне и его верификации на блокчейн-эксплорере.

В рамках развёртывания контракта вам может показаться удобным сменить владельца контракта на ваш адрес в Метамаске, используя функцию `transferOwnership`. Для этого воспользуйтесь следующей миграцией `migrations/2_token.js`:

```

const Token = artifacts.require("./Token.sol");

module.exports = async function (deployer) {
  const deployment = deployer.deploy(Token, "HelloToken", "HELLO", 10000n *
  BigInt(1e18));
  const instance = await deployment.await
  const newOwner = '<ваш-адрес-в-метамаске>'
  await instance.transferOwnership(newOwner)
};

```

Вы также можете вызвать любую другую функцию контракта.

4. Создать пользовательский интерфейс для взаимодействия со смарт-контрактом

Теперь задача - создать пользовательский интерфейс для работы с контрактом. В пользовательском интерфейсе для адреса пользователя следует показать его баланс. Следует предоставить функцию отправки токена, вызывающую функцию `transfer` с заданной пользователем величиной перевода на заданный пользователем адрес получателя. Важно: учитывайте количество точек после запятой, предоставляемые контрактом в значении `decimals`.

Если пользователь является владельцем контракта, следует дополнительно предоставить функцию чеканки токена, вызывающую функцию контракта `mint`. При чеканке пользователь должен предоставить адрес получателя и объём создаваемых токенов.

Если кошелёк пользователя поддерживает [EIP-747](#), выведите кнопку (или другой элемент взаимодействия), позволяющий человеку добавить токен для отображения к себе в кошелёк. Можно воспользоваться документацией, которую предоставляет MetaMask о вызове `wallet_watchAsset`

Вопросы для проверки

1. Что означает быть "владельцем" контракта?
2. Могут ли несколько человек быть "владельцами" контракта? Каким образом?
3. Насколько обязательны события Transfer и ChangeOwnership при вызове функций контракта?
4. Может ли контракт токена ERC-20 (не обязательно тот, который вы создали в рамках лабораторной работы) при вызове функции `transfer` вызывать функции другого контракта?
5. Какие бывают модификаторы функций смарт-контрактов в Solidity?
6. Известно, что за транзакции нужно платить. Известно, что транзакция может вызывать функцию смарт-контракта, и оплата происходит в зависимости от сложности функции. Какие функции смарт-контрактов можно вызывать без оплаты транзакционной комиссии?
7. Чем отличается токен от криптовалюты?
8. Кто и как фиксирует общий доступный объём токенов?
9. Каким образом криптокошелёк отображает баланс токенов?
10. Как можно позволить пользователю вашего интерфейса осуществлять перевод токенов без того, чтобы оплачивать комиссию за транзакцию?
11. Как можно заблокировать пользователю возможность переводить его токены?
12. Как можно позволить пользователю передавать право распоряжаться частью его токенов третьему лицу?

Формат предоставления отчёта

В результате работы у вас должно получиться совсем небольшое, но законченное приложение. Код приложения - контракты и пользовательский интерфейс - должен быть доступен в системе контроля версий: GitHub, GitLab, Radicle и т.д.

По завершении кодирования попросите своих одноклассников воспользоваться вашим приложением.

Отчёт должен содержать:

- ссылку на верифицированный контракт на блокчейн-эксплорере,
- ссылку на репозиторий,
- описание вашего процесса работы над приложением,
- описание приложения как краткое руководство пользователя,
- ответы на вопросы.