2018

ICO SMART CONTRACT UPDATE

CRYPTO4ALL

PROJECT TRECENTO
September 20th 2018

₵RYPTO⁴ΛLL

TABLE OF CONTENTS

TRECENTO – SMART CONTRACT UPDATES 2

Context 2

Smart contracts 3

Unit tests 20

Deployment tests 51

TRECENTO – SMART CONTRACT UPDATES

Context

We have been solicitated to implement the recommendations and suggestions provided by our previous audit report delivered on 31/08/2018. This report only highlighted security flaws and vulnerabilities regarding the provided smart contract code. The client didn't provided any functional documentation describing the process of the scheduled crowdfunding scenario to check the compliance with the smart contract code.

High security flaws

NONE

Medium vulnerabilities [Patched]

We have detected mathematical operations within the TOTTokens contract without the use of SafeMath library operations involving a risk of overflow issues. As recommended below you should update the code to prevent it by using mul and div operations. [RESOLVED]

Besides we suggest you to use the latest stable version of solidity 0.4.24 to enforce the compiler and prevent from any previous bugs/flaws detected on previous versions. [RESOLVED]

Low vulnerabilities

NONE

Smart contracts

- ✓ We used the latest stable version of solidity to enforce the compiler and prevent from any previous bugs/flaws detected on previous versions. The latest stable available version is now 0.4.24
- ✓ We created solidity files for each called smart contract to improve and manage future updates.
- ✓ We have updated the mathematical operations by using SafeMath library.

SafeMath.SOL

```
pragma solidity ^0.4.24;
/**
 * @title SafeMath
 * @dev Math operations (only add and sub here) with safety checks that throw on error
 */
library SafeMath {


  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
  uint256 c = a * b;
  assert(c / a == b);
  return c;
 }

 /**
 * @dev Integer division of two numbers, truncating the quotient.
 */
 function div(uint256 a, uint256 b) internal pure returns (uint256) {
  // assert(b > 0); // Solidity automatically throws when dividing by 0
  uint256 c = a / b;
  // assert(a == b * c + a % b); // There is no case in which this doesn't hold
  return c;
 }

 /**
 * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
 */
 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  assert(b <= a);
  return a - b;
 }

 /**
 * @dev Adds two numbers, throws on overflow.
 */
 function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  assert(c >= a);
  return c;
 }
}
```

.**Ownable.SOL**

```
pragma solidity ^0.4.24;
/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;



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



  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  constructor() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

}
```

Pausable.SOL

```
pragma solidity ^0.4.24;
import "./Ownable.sol";
/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable {
 event Pause();
 event Unpause();

 bool public paused = false;



 /**
  * @dev Modifier to make a function callable only when the contract is not paused.
  */
 modifier whenNotPaused() {
   require(!paused);
   _;
 }

 /**
```

```
   * @dev Modifier to make a function callable only when the contract is paused.
   */
  modifier whenPaused() {
    require(paused);
    _;
  }

  /**
   * @dev called by the owner to pause, triggers stopped state
   */
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    emit Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    emit Unpause();
  }
}
```

ERC20.SOL

```
pragma solidity ^0.4.24;
/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

TOTToken.SOL

```
pragma solidity ^0.4.24;
import "./ERC20.sol";
import "./Pausable.sol";
import "./SafeMath.sol";

contract TOTToken is ERC20, Pausable {

  using SafeMath for uint256;

  string public name = "Trecento";      //  token name
  string public symbol = "TOT";          //  token symbol
  uint256 public decimals = 8;           //  token digit



  // Token distribution, must sumup to 1000
  uint256 public constant SHARE_PURCHASERS = 750;
  uint256 public constant SHARE_FOUNDATION = 50;
  uint256 public constant SHARE_TEAM = 150;
  uint256 public constant SHARE_BOUNTY = 50;

  // Wallets addresses
  address public foundationAddress;
  address public teamAddress;
```

```
 address public bountyAddress;


 uint256 private totalSupply_;

 uint256 public cap = 20000000 * 10 ** decimals; // Max cap 20.000.000 token


 mapping(address => uint256) private balances;


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


 bool public mintingFinished = false;


 event Burn(address indexed burner, uint256 value);

 event Mint(address indexed to, uint256 amount);

 event MintFinished();


 modifier canMint() {

   require(!mintingFinished);

   _;

 }


 constructor(address _foundationAddress, address _teamAddress, address _bountyAddress) public {

   require(_foundationAddress != address(0) && _teamAddress != address(0) && _bountyAddress != 
address(0));

   foundationAddress = _foundationAddress;

   teamAddress = _teamAddress;

   bountyAddress = _bountyAddress;

 }


 /**

   * @dev Change token name, Owner only.

   * @param _name The name of the token.

 */

 function setName(string _name) onlyOwner public {
   name = _name;
 }

 /**
   * @dev Change token symbol, Owner only.
   * @param _symbol The symbol of the token.
 */
 function setSymbol(string _symbol) onlyOwner public {
   symbol = _symbol;
 }

 function setWallets(address _foundation, address _team, address _bounty) public onlyOwner canMint {
   require(_foundation != address(0) && _team != address(0) && _bounty != address(0));
   foundationAddress = _foundation;
   teamAddress = _team;
   bountyAddress = _bounty;
 }
 /**
   * @dev total number of tokens in existence
 */
 function totalSupply() public view returns (uint256) {
   return totalSupply_;
 }

 /**
   * @dev Gets the balance of the specified address.
   * @param _owner The address to query the the balance of.
   * @return An uint256 representing the amount owned by the passed address.
 */
 function balanceOf(address _owner) public view returns (uint256 balance) {
   return balances[_owner];
 }
```

```
 /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
 */
 function transfer(address _to, uint256 _value) public whenNotPaused returns (bool success) {
  require(_to != address(0));
  require(_value <= balances[msg.sender]);
  // SafeMath.sub will throw if there is not enough balance.
  balances[msg.sender] = balances[msg.sender].sub(_value);
  balances[_to] = balances[_to].add(_value);
  emit Transfer(msg.sender, _to, _value);
  return true;
 }


 /**
  * @dev Transfer tokens from one address to another
  * @param _from address The address which you want to send tokens from
  * @param _to address The address which you want to transfer to
  * @param _value uint256 the amount of tokens to be transferred
 */
 function transferFrom(address _from, address _to, uint256 _value) public whenNotPaused returns (bool)
{
  require(_to != address(0));
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);


  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  emit Transfer(_from, _to, _value);
  return true;
 }
 /**
  * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
  *
  * 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
  * @param _spender The address which will spend the funds.
  * @param _value The amount of tokens to be spent.
 */
 function approve(address _spender, uint256 _value) public whenNotPaused returns (bool) {
  allowed[msg.sender][_spender] = _value;
  emit Approval(msg.sender, _spender, _value);
  return true;
 }

 /**
  * @dev Function to check the amount of tokens that an owner allowed to a spender.
  * @param _owner address The address which owns the funds.
  * @param _spender address The address which will spend the funds.
  * @return A uint256 specifying the amount of tokens still available for the spender.
 */
 function allowance(address _owner, address _spender) public view returns (uint256) {
  return allowed[_owner][_spender];
 }

 /**
  * @dev Increase the amount of tokens that an owner allowed to a spender.
  *
  * approve should be called when allowed[_spender] == 0. To increment
  * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
 function increaseApproval(address _spender, uint _addedValue) public whenNotPaused returns (bool) {
 allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
 emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
 return true;
 }


 /**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
 */
 function decreaseApproval(address _spender, uint _subtractedValue) public whenNotPaused returns
(bool) {
 uint oldValue = allowed[msg.sender][_spender];
 if (_subtractedValue > oldValue) {
  allowed[msg.sender][_spender] = 0;
 } else {
  allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
 }
 emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
 return true;
 }
```

```
/**
 * @dev Function to      mint tokens
 * @param _to The address that will receive the minted tokens.
 * @param _amount The amount of tokens to mint.
 * @return A boolean that indicates if the operation was successful.
 */
function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
  require(_amount != 0);
  require(totalSupply_.add(_amount) <= cap);
  require(_to != address(0));
  totalSupply_ = totalSupply_.add(_amount);
  balances[_to] = balances[_to].add(_amount);
  emit Mint(_to, _amount);
  emit Transfer(address(0), _to, _amount);
  return true;
}


/**
 * @dev Function to stop minting new tokens.
 * @return True if the operation was successful.
 */
function finishMinting() onlyOwner canMint public returns (bool) {


  require(foundationAddress != address(0) && teamAddress != address(0) && bountyAddress !=
address(0));


  // before calling this method totalSupply includes only purchased tokens
  uint256 onePerThousand = totalSupply_.div(SHARE_PURCHASERS); //ignore (totalSupply mod 617) ~=
616e-8,


  uint256 foundationTokens = onePerThousand.mul(SHARE_FOUNDATION);
  uint256 teamTokens = onePerThousand.mul(SHARE_TEAM);
  uint256 bountyTokens = onePerThousand.mul(SHARE_BOUNTY);
  mint(foundationAddress, foundationTokens);
  mint(teamAddress, teamTokens);
  mint(bountyAddress, bountyTokens);

  mintingFinished = true;
  emit MintFinished();
  return true;
}



/**
  * @dev Burns a specific amount of tokens.
  * @param _value The amount of token to be burned.
*/
function burn(uint256 _value) public whenNotPaused {
  require(_value <= balances[msg.sender]);
  // no need to require value <= totalSupply, since that would imply the
  // sender's balance is greater than the totalSupply, which *should* be an assertion failure

  address burner = msg.sender;
  balances[burner] = balances[burner].sub(_value);
  totalSupply_ = totalSupply_.sub(_value);
  emit Burn(burner, _value);
  emit Transfer(burner, address(0), _value);
}



/**
  * @dev This is an especial owner-only function to make massive tokens minting.
  * @param _data is an array of addresses
  * @param _amount is an array of uint256
*/
function batchMint(address[] _data,uint256[] _amount) public onlyOwner canMint {
  for (uint i = 0; i < _data.length; i++) {
            mint(_data[i],_amount[i]);
  }
 }



}
```

Solidity code update:

```
// Wallets addresses
 address public foundationAddress;
 address public teamAddress;
 address public bountyAddress;

 constructor(address _foundationAddress, address _teamAddress, address _bountyAddress) public {
  require(_foundationAddress != address(0) && _teamAddress != address(0) && _bountyAddress !=
address(0));
  foundationAddress = _foundationAddress;
  teamAddress = _teamAddress;
  bountyAddress = _bountyAddress;
 }
```

➔As recommended on our previous audit report we have implemented a constructor function to initialized all wallet addresses (foundationAddress and teamAddress, bountyAddress) during the deployment. These addresses will be filled by your deployment team. more transparency and reduce the risk of forgetting it after the deployment.

```
 /**
  * @dev Change token symbol, Owner only.
  * @param _symbol The symbol of the token.
 */
 function setSymbol(string _symbol) onlyOwner public {
  symbol = _symbol;
```

```
}
```

➔ We added a function that allows you to change the symbol after the deployment in the future if needed. However any update of this value could have the same consequences with exchange platforms and users as suggested on our audit report for the setName function.

```
uint256 private totalSupply_;


mapping(address => uint256) private balances;


mapping (address => mapping (address => uint256)) private allowed;
```

➔ We have to change the visibility of the attributes "totalSupply", "balances" and "allowed" to private as recommended by the OpenZeppelin Standard.

```
function updateWallets(address _foundation, address _team, address _bounty) public onlyOwner canMint
{
  require(_foundation != address(0) && _team != address(0) && _bounty != address(0));
  foundationAddress = _foundation;
  teamAddress = _team;
  bountyAddress = _bounty;
}
```

➔ We have renamed the previous SetWallets function allowing after the deployment of the contract to update if needed the foundation, team and bounty ethereum addresses.

```
function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
  require(_amount > 0);
  require(totalSupply_.add(_amount) <= cap);
  require(_to != address(0));

  totalSupply_ = totalSupply_.add(_amount);

  balances[_to] = balances[_to].add(_amount);

  emit Mint(_to, _amount);

  emit Transfer(address(0), _to, _amount);

  return true;

 }
```

➔ We added a check to not let a call with a zero value amount.

```
 function finishMinting() onlyOwner canMint public returns (bool) {


  require(foundationAddress != address(0) && teamAddress != address(0) && bountyAddress !=
address(0));

  require(SHARE_PURCHASERS + SHARE_FOUNDATION + SHARE_TEAM + SHARE_BOUNTY == 1000);


   // before calling this method totalSupply includes only purchased tokens

  uint256 onePerThousand = totalSupply_.div(SHARE_PURCHASERS); //ignore (totalSupply mod 617) ~=
616e-8,


  uint256 foundationTokens = onePerThousand.mul(SHARE_FOUNDATION);

  uint256 teamTokens = onePerThousand.mul(SHARE_TEAM);

  uint256 bountyTokens = onePerThousand.mul(SHARE_BOUNTY);

   require (balanceOf(foundationAddress) == 0 && balanceOf(teamAddress) == 0 &&
balanceOf(bountyAddress) == 0);

  mint(foundationAddress, foundationTokens);

  mint(teamAddress, teamTokens);

  mint(bountyAddress, bountyTokens);


  mintingFinished = true;

  emit MintFinished();

  return true;

 }
```

➔ The require check the existence of 0x0 addresses. This control is already done within the constructor and also within the updateWallets function preventing this issue so we delete it to improve the behavior and gas consumption of the code.

➔ The “require” is consuming GAS for checking constants that cannot be publicly writable so we deleted it

➔ The “require” is added to check that foundation, team and bounty addresses have not an existing balance to respect the allocation described on the Whitepaper.

➔> We used “mul” and “div” operations instead of “* “ and “ / “to prevent any overflow issues as recommended in our audit report.

Unit tests

TOTToken Testing (TTOToken.test.js)

```
import assertRevert from './helpers/assertRevert';

import BigNumber from 'bignumber.js'

const TOTToken = artifacts.require('TOTToken');


contract('TOTToken', function ([_, owner, recipient, anotherAccount]) {

 const ZERO_ADDRESS = '0x0000000000000000000000000000000000000000';


 const batchListAccount = [

  '0xf17f52151ebef6c7334fad080c5704d77216b732',

  '0xc5fdf4076b8f3a5357c5e395ab970b5b54098fef',

  '0x821aea9a577a9b44299b9c15c88cf3087f3b5544'

 ];


 const TOKEN_DECIMAL = 8;

 const MAX_TOKEN_SUPPLY = new BigNumber(20000000 * 10 ** TOKEN_DECIMAL);




 beforeEach(async function () {

  this.token = await
TOTToken.new('0xBa893462c1b714bFD801e918a4541e056f9bd924','0x2418C46F2FA422fE8Cd0BF56
Df5e27CbDeBB2590','0x84bE27E1d3AeD5e6CF40445891d3e2AB7d3d98e8',{ from: owner });

 });


 describe('constructor', function () {

  it('should revert when creating token with wallets addresses equals to zero', async function () {

   await assertRevert(TOTToken.new('0x0','0x0','0x0',{ from: owner }));

  });

 });

 describe('mint', function () {
  it('should start with the correct cap', async function () {
    let _cap = await this.token.cap();
    assert(MAX_TOKEN_SUPPLY.eq(_cap));
  });

  it('should fail to mint if not owner', async function () {
    await assertRevert(this.token.mint(owner, 100, { from: anotherAccount }));
  });

  describe('when the requested amount of token is less than cap', function () {
    it('should mint', async function () {
      const result = await this.token.mint(owner, 100, { from: owner });
      assert.equal(result.logs[0].event, 'Mint');
    });
  });

  describe('when the requested amount of token exceeds the cap', function () {
    it('should fail to mint and revert', async function () {
      await this.token.mint(owner, MAX_TOKEN_SUPPLY.minus(1), { from: owner });
      await assertRevert(this.token.mint(owner, 100, { from: owner }));
    });
  });

  describe('After the cap is reached', function () {
    it('should fail to mint and revert', async function () {
      await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
      await assertRevert(this.token.mint(owner, 1));
    });
  });
});

describe('finishMinting', function () {
  const PURCHASER_AMOUNT = new BigNumber(10000000 * 10 ** TOKEN_DECIMAL);
  const ONE_PER_THOUSAND = PURCHASER_AMOUNT.dividedToIntegerBy(750);
/*
  it('should revert when finishMinting when wallets are not set', async function () {
    await this.token.mint(owner, PURCHASER_AMOUNT, { from: owner });
    await assertRevert(this.token.finishMinting({ from: owner }));
  });
*/
  it('should allocate Foundation, Team and Bounty ', async function () {
    await this.token.mint(owner, PURCHASER_AMOUNT, { from: owner });
    //await this.token.setWallets('0xBa893462c1b714bFD801e918a4541e056f9bd924',
'0x2418C46F2FA422fE8Cd0BF56Df5e27CbDeBB2590',
'0x84bE27E1d3AeD5e6CF40445891d3e2AB7d3d98e8',{ from: owner });
    await this.token.finishMinting({ from: owner });
    let bountyBalance = await
this.token.balanceOf('0x84bE27E1d3AeD5e6CF40445891d3e2AB7d3d98e8');
    assert(bountyBalance.eq(ONE_PER_THOUSAND.mul(50)));


    let foundationBalance = await
this.token.balanceOf('0xBa893462c1b714bFD801e918a4541e056f9bd924');
    assert(foundationBalance.eq(ONE_PER_THOUSAND.mul(50)));


    let teamBalance = await this.token.balanceOf('0x2418C46F2FA422fE8Cd0BF56Df5e27CbDeBB2590');
    assert(teamBalance.eq(ONE_PER_THOUSAND.mul(150)));



assert(teamBalance.plus(foundationBalance).plus(bountyBalance).plus(ONE_PER_THOUSAND.mul(750)).
eq(ONE_PER_THOUSAND.mul(1000)));
  });
  it('should revert when minting after finishMinting ', async function () {
    await this.token.mint(owner, PURCHASER_AMOUNT, { from: owner });
    //await this.token.setWallets('0xBa893462c1b714bFD801e918a4541e056f9bd924',
'0x2418C46F2FA422fE8Cd0BF56Df5e27CbDeBB2590',
'0x84bE27E1d3AeD5e6CF40445891d3e2AB7d3d98e8',{ from: owner });
    await this.token.finishMinting({ from: owner });
    await assertRevert(this.token.mint(owner, 1));
  });
```

```
});

describe('total supply', function () {
  it('returns the total amount of tokens', async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
    const totalSupply = await this.token.totalSupply();
    assert(MAX_TOKEN_SUPPLY.eq(totalSupply));
  });
});

describe('balanceOf', function () {
  beforeEach(async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
  });

  describe('when the requested account has no tokens', function () {
    it('returns zero', async function () {
      const balance = await this.token.balanceOf(anotherAccount);

      assert.equal(balance, 0);
    });
  });

  describe('when the requested account has some tokens', function () {
    it('returns the total amount of tokens', async function () {
      const balance = await this.token.balanceOf(owner);

      assert(MAX_TOKEN_SUPPLY.eq(balance));
    });
  });
});

describe('transfer', function () {
```

```
describe('when the recipient is not the zero address', function () {
  beforeEach(async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
  });

  const to = recipient;

  describe('when the sender does not have enough balance', function () {
    const amount = MAX_TOKEN_SUPPLY.plus(1);

    it('reverts', async function () {
      await assertRevert(this.token.transfer(to, amount, { from: owner }));
    });
  });

  describe('when the sender has enough balance', function () {
    const amount = MAX_TOKEN_SUPPLY;

    it('transfers the requested amount', async function () {
      await this.token.transfer(to, amount, { from: owner });

      const senderBalance = await this.token.balanceOf(owner);
      assert.equal(senderBalance, 0);

      const recipientBalance = await this.token.balanceOf(to);
      assert(amount.eq(recipientBalance));
    });

    it('emits a transfer event', async function () {
      const { logs } = await this.token.transfer(to, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Transfer');
```

```
        assert.equal(logs[0].args.from, owner);
        assert.equal(logs[0].args.to, to);
        assert(logs[0].args.value.eq(amount));
      });
    });
  });
});

describe('batchMint', async function () {

  describe('when the recipient is not the zero address', function () {

    describe('when max cap is exceeded', function () {
      const batchListAmount = [
        5000000 * 10 ** TOKEN_DECIMAL,
        10000000 * 10 ** TOKEN_DECIMAL,
        10000000 * 10 ** TOKEN_DECIMAL
      ];

      it('reverts', async function () {
        await assertRevert(this.token.batchMint(batchListAccount, batchListAmount, { from: owner }));
        const totalSupply = await this.token.totalSupply();
        assert.equal(totalSupply, 0);
      });
    });

    describe('when the max cap is not exceeded', function () {
      const batchListAmount = [
        5000000 * 10 ** TOKEN_DECIMAL,
        5000000 * 10 ** TOKEN_DECIMAL,
        10000000 * 10 ** TOKEN_DECIMAL
      ];
    it('mints the requested amount', async function () {

      await this.token.batchMint(batchListAccount, batchListAmount, { from: owner });

      const totalSupply = await this.token.totalSupply();
      assert(MAX_TOKEN_SUPPLY.eq(totalSupply));

      const recipientBalance1 = await this.token.balanceOf(batchListAccount[0]);
      assert.equal(recipientBalance1, batchListAmount[0]);

      const recipientBalance2 = await this.token.balanceOf(batchListAccount[1]);
      assert.equal(recipientBalance2, batchListAmount[1]);

      const recipientBalance3 = await this.token.balanceOf(batchListAccount[2]);
      assert.equal(recipientBalance3, batchListAmount[2]);
    });

    it('emits the Mint and Transfer events', async function () {
      const { logs } = await this.token.batchMint(batchListAccount, batchListAmount, { from: owner });

      assert.equal(logs.length, 6);

      assert.equal(logs[0].event, 'Mint');
      assert.equal(logs[0].args.to, batchListAccount[0]);
      assert(logs[0].args.amount.eq(batchListAmount[0]));

      assert.equal(logs[1].event, 'Transfer');
      assert.equal(logs[1].args.to, batchListAccount[0]);
      assert(logs[1].args.value.eq(batchListAmount[0]));
    });
  });
});
```

```
  describe('when one recipient is the zero address', function () {

    const batchListAmount = [25000000 * 10 ** TOKEN_DECIMAL, 25000000 * 10 ** TOKEN_DECIMAL,
50000000 * 10 ** TOKEN_DECIMAL];


    it('reverts', async function () {

      await assertRevert(this.token.batchMint([batchListAccount[0], ZERO_ADDRESS,
batchListAccount[2]], batchListAmount, { from: owner }));

           const totalSupply = await this.token.totalSupply();

      assert.equal(totalSupply, 0);

    });

  });


  describe('when 50 recipient ', function () {


    let batch50Account = [50];

    for (var i = 0; i < 50; i++) {

      let address = '0x2bdd21761a483f71054e14f5b827213567971c' + (i + 16).toString(16);

      batch50Account[i] = address;

    }

    let batchListAmount = [50];

    for (var i = 0; i < 50; i++) {

      batchListAmount[i] = 100 * 10 ** TOKEN_DECIMAL;

    }


    it('should perform batch for 50 accounts that should have 100 TOT each', async function () {


      await this.token.batchMint(batch50Account, batchListAmount, { from: owner });

      for (var i = 0; i < batch50Account.length; i++) {

        const tokenBalance = await this.token.balanceOf(batch50Account[i]);

        assert.equal(tokenBalance, 100 * 10 ** TOKEN_DECIMAL);

      }

    });

  });
  describe('when 100 recipient ', function () {


    let batch100Account = [100];
    for (var i = 0; i < 100; i++) {
      let address = '0x2bdd21761a483f71054e14f5b827213567971c' + (i + 16).toString(16);
      batch100Account[i] = address;
    }
    let batchListAmount = [100];
    for (var i = 0; i < 100; i++) {
      batchListAmount[i] = 100 * 10 ** TOKEN_DECIMAL;
    }

    it('should perform batch for 100 accounts that should have 100 TOT each', async function () {


      await this.token.batchMint(batch100Account, batchListAmount, { from: owner });
      for (var i = 0; i < batch100Account.length; i++) {
        const tokenBalance = await this.token.balanceOf(batch100Account[i]);
        assert.equal(tokenBalance, 100 * 10 ** TOKEN_DECIMAL);
      }
    });
  });
});

describe('approve', function () {
  beforeEach(async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
  });

  describe('when the spender is not the zero address', function () {
    const spender = recipient;

    describe('when the sender has enough balance', function () {
      const amount = MAX_TOKEN_SUPPLY;
    it('emits an approval event', async function () {
      const { logs } = await this.token.approve(spender, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Approval');
      assert.equal(logs[0].args.owner, owner);
      assert.equal(logs[0].args.spender, spender);
      assert(logs[0].args.value.eq(amount));
    });

    describe('when there was no approved amount before', function () {
      it('approves the requested amount', async function () {
        await this.token.approve(spender, amount, { from: owner });

        const allowance = await this.token.allowance(owner, spender);
        assert(amount.eq(allowance));
      });
    });

    describe('when the spender had an approved amount', function () {
      beforeEach(async function () {
        await this.token.approve(spender, 1, { from: owner });
      });

      it('approves the requested amount and replaces the previous one', async function () {
        await this.token.approve(spender, amount, { from: owner });

        const allowance = await this.token.allowance(owner, spender);
        assert(amount.eq(allowance));
      });
    });
  });
```

```
describe('when the sender does not have enough balance', function () {
 const amount = MAX_TOKEN_SUPPLY.plus(1);

 it('emits an approval event', async function () {
  const { logs } = await this.token.approve(spender, amount, { from: owner });

  assert.equal(logs.length, 1);
  assert.equal(logs[0].event, 'Approval');
  assert.equal(logs[0].args.owner, owner);
  assert.equal(logs[0].args.spender, spender);
  assert(logs[0].args.value.eq(amount));
 });

 describe('when there was no approved amount before', function () {
  it('approves the requested amount', async function () {
   await this.token.approve(spender, amount, { from: owner });

   const allowance = await this.token.allowance(owner, spender);
   assert(amount.eq(allowance));
  });
 });

 describe('when the spender had an approved amount', function () {
  beforeEach(async function () {
   await this.token.approve(spender, 1, { from: owner });
  });

  it('approves the requested amount and replaces the previous one', async function () {
   await this.token.approve(spender, amount, { from: owner });

   const allowance = await this.token.allowance(owner, spender);
   assert(amount.eq(allowance));
```

```
        });
      });
    });
  });

  describe('when the spender is the zero address', function () {
    const amount = MAX_TOKEN_SUPPLY;
    const spender = ZERO_ADDRESS;

    it('approves the requested amount', async function () {
      await this.token.approve(spender, amount, { from: owner });

      const allowance = await this.token.allowance(owner, spender);
      assert(amount.eq(allowance));
    });

    it('emits an approval event', async function () {
      const { logs } = await this.token.approve(spender, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Approval');
      assert.equal(logs[0].args.owner, owner);
      assert.equal(logs[0].args.spender, spender);
      assert(logs[0].args.value.eq(amount));
    });
  });
});

describe('transfer from', function () {
  beforeEach(async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
  });
```

```
const spender = recipient;

describe('when the recipient is not the zero address', function () {
  const to = anotherAccount;

  describe('when the spender has enough approved balance', function () {
    beforeEach(async function () {
      await this.token.approve(spender, MAX_TOKEN_SUPPLY, { from: owner });
    });

    describe('when the owner has enough balance', function () {
      const amount = MAX_TOKEN_SUPPLY;

      it('transfers the requested amount', async function () {
        await this.token.transferFrom(owner, to, amount, { from: spender });

        const senderBalance = await this.token.balanceOf(owner);
        assert.equal(senderBalance, 0);

        const recipientBalance = await this.token.balanceOf(to);
        assert(amount.eq(recipientBalance));
      });

      it('decreases the spender allowance', async function () {
        await this.token.transferFrom(owner, to, amount, { from: spender });

        const allowance = await this.token.allowance(owner, spender);
        assert(allowance.eq(0));
      });

      it('emits a transfer event', async function () {
        const { logs } = await this.token.transferFrom(owner, to, amount, { from: spender });
        assert.equal(logs.length, 1);
        assert.equal(logs[0].event, 'Transfer');
        assert.equal(logs[0].args.from, owner);
        assert.equal(logs[0].args.to, to);
        assert(logs[0].args.value.eq(amount));
      });
    });

    describe('when the owner does not have enough balance', function () {
      const amount = MAX_TOKEN_SUPPLY.plus(1);

      it('reverts', async function () {
        await assertRevert(this.token.transferFrom(owner, to, amount, { from: spender }));
      });
    });
  });

  describe('when the spender does not have enough approved balance', function () {
    beforeEach(async function () {
      await this.token.approve(spender, MAX_TOKEN_SUPPLY.minus(1), { from: owner });
    });

    describe('when the owner has enough balance', function () {
      const amount = MAX_TOKEN_SUPPLY;

      it('reverts', async function () {
        await assertRevert(this.token.transferFrom(owner, to, amount, { from: spender }));
      });
    });

    describe('when the owner does not have enough balance', function () {
      const amount = MAX_TOKEN_SUPPLY.plus(1);
      it('reverts', async function () {
        await assertRevert(this.token.transferFrom(owner, to, amount, { from: spender }));
      });
    });
  });
 });

 describe('when the recipient is the zero address', function () {
   const amount = MAX_TOKEN_SUPPLY;
   const to = ZERO_ADDRESS;

   beforeEach(async function () {
     await this.token.approve(spender, amount, { from: owner });
   });

   it('reverts', async function () {
     await assertRevert(this.token.transferFrom(owner, to, amount, { from: spender }));
   });
 });
});

describe('decrease approval', function () {
 beforeEach(async function () {
   await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
 });

 describe('when the spender is not the zero address', function () {
   const spender = recipient;

   describe('when the sender has enough balance', function () {
     const amount = MAX_TOKEN_SUPPLY;

     it('emits an approval event', async function () {
      const { logs } = await this.token.decreaseApproval(spender, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Approval');
      assert.equal(logs[0].args.owner, owner);
      assert.equal(logs[0].args.spender, spender);
      assert(logs[0].args.value.eq(0));
    });

    describe('when there was no approved amount before', function () {
      it('keeps the allowance to zero', async function () {
        await this.token.decreaseApproval(spender, amount, { from: owner });

        const allowance = await this.token.allowance(owner, spender);
        assert.equal(allowance, 0);
      });
    });

    describe('when the spender had an approved amount', function () {
      beforeEach(async function () {
        await this.token.approve(spender, amount.plus(1), { from: owner });
      });

      it('decreases the spender allowance subtracting the requested amount', async function () {
        await this.token.decreaseApproval(spender, amount, { from: owner });

        const allowance = await this.token.allowance(owner, spender);
        assert.equal(allowance, 1);
      });
    });
  });

  describe('when the sender does not have enough balance', function () {
```

```
const amount = MAX_TOKEN_SUPPLY.plus(1);

it('emits an approval event', async function () {
  const { logs } = await this.token.decreaseApproval(spender, amount, { from: owner });

  assert.equal(logs.length, 1);
  assert.equal(logs[0].event, 'Approval');
  assert.equal(logs[0].args.owner, owner);
  assert.equal(logs[0].args.spender, spender);
  assert(logs[0].args.value.eq(0));
});

describe('when there was no approved amount before', function () {
  it('keeps the allowance to zero', async function () {
    await this.token.decreaseApproval(spender, amount, { from: owner });

    const allowance = await this.token.allowance(owner, spender);
    assert.equal(allowance, 0);
  });
});

describe('when the spender had an approved amount', function () {
  beforeEach(async function () {
    await this.token.approve(spender, amount.plus(1), { from: owner });
  });

  it('decreases the spender allowance subtracting the requested amount', async function () {
    await this.token.decreaseApproval(spender, amount, { from: owner });

    const allowance = await this.token.allowance(owner, spender);
    assert.equal(allowance, 1);
  });
});
```

```
    });
  });

  describe('when the spender is the zero address', function () {
    const amount = MAX_TOKEN_SUPPLY;
    const spender = ZERO_ADDRESS;

    it('decreases the requested amount', async function () {
      await this.token.decreaseApproval(spender, amount, { from: owner });

      const allowance = await this.token.allowance(owner, spender);
      assert.equal(allowance, 0);
    });

    it('emits an approval event', async function () {
      const { logs } = await this.token.decreaseApproval(spender, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Approval');
      assert.equal(logs[0].args.owner, owner);
      assert.equal(logs[0].args.spender, spender);
      assert(logs[0].args.value.eq(0));
    });
  });
});

describe('increase approval', function () {
  beforeEach(async function () {
    await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
  });
  const amount = MAX_TOKEN_SUPPLY;

  describe('when the spender is not the zero address', function () {
```

```
const spender = recipient;


describe('when the sender has enough balance', function () {
 it('emits an approval event', async function () {
   const { logs } = await this.token.increaseApproval(spender, amount, { from: owner });


   assert.equal(logs.length, 1);
   assert.equal(logs[0].event, 'Approval');
   assert.equal(logs[0].args.owner, owner);
   assert.equal(logs[0].args.spender, spender);
   assert(logs[0].args.value.eq(amount));
 });


 describe('when there was no approved amount before', function () {
   it('approves the requested amount', async function () {
     await this.token.increaseApproval(spender, amount, { from: owner });


     const allowance = await this.token.allowance(owner, spender);
     assert(amount.eq(allowance));
   });
 });


 describe('when the spender had an approved amount', function () {
   beforeEach(async function () {
     await this.token.approve(spender, 1, { from: owner });
   });


   it('increases the spender allowance adding the requested amount', async function () {
     await this.token.increaseApproval(spender, amount, { from: owner });


     const allowance = await this.token.allowance(owner, spender);
     assert(amount.plus(1).eq(allowance));
   });
```

```
    });
  });

  describe('when the sender does not have enough balance', function () {
    const amount = MAX_TOKEN_SUPPLY.plus(1);

    it('emits an approval event', async function () {
      const { logs } = await this.token.increaseApproval(spender, amount, { from: owner });

      assert.equal(logs.length, 1);
      assert.equal(logs[0].event, 'Approval');
      assert.equal(logs[0].args.owner, owner);
      assert.equal(logs[0].args.spender, spender);
      assert(logs[0].args.value.eq(amount));
    });

    describe('when there was no approved amount before', function () {
      it('approves the requested amount', async function () {
        await this.token.increaseApproval(spender, amount, { from: owner });

        const allowance = await this.token.allowance(owner, spender);
        assert(amount.eq(allowance));
      });
    });

    describe('when the spender had an approved amount', function () {
      beforeEach(async function () {
        await this.token.approve(spender, 1, { from: owner });
      });

      it('increases the spender allowance adding the requested amount', async function () {
        await this.token.increaseApproval(spender, amount, { from: owner });
        const allowance = await this.token.allowance(owner, spender);
        assert(amount.plus(1).eq(allowance));
      });
    });
  });
 });

 describe('when the spender is the zero address', function () {
  const spender = ZERO_ADDRESS;

  it('approves the requested amount', async function () {
   await this.token.increaseApproval(spender, amount, { from: owner });

   const allowance = await this.token.allowance(owner, spender);
   assert(amount.eq(allowance));
  });

  it('emits an approval event', async function () {
   const { logs } = await this.token.increaseApproval(spender, amount, { from: owner });

   assert.equal(logs.length, 1);
   assert.equal(logs[0].event, 'Approval');
   assert.equal(logs[0].args.owner, owner);
   assert.equal(logs[0].args.spender, spender);
   assert(logs[0].args.value.eq(amount));
  });
 });
});

describe('burn', function () {
 beforeEach(async function () {
  await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
 });
```

```
const from = owner;

describe('when the given amount is not greater than balance of the sender', function () {
 const amount = 100;

 it('burns the requested amount', async function () {
   await this.token.burn(amount, { from });

   const balance = await this.token.balanceOf(from);
   assert(MAX_TOKEN_SUPPLY.minus(100).eq(balance));
 });

 it('emits a burn event', async function () {
   const { logs } = await this.token.burn(amount, { from });
   const ZERO_ADDRESS = '0x0000000000000000000000000000000000000000';
   assert.equal(logs.length, 2);
   assert.equal(logs[0].event, 'Burn');
   assert.equal(logs[0].args.burner, owner);
   assert.equal(logs[0].args.value, amount);

   assert.equal(logs[1].event, 'Transfer');
   assert.equal(logs[1].args.from, owner);
   assert.equal(logs[1].args.to, ZERO_ADDRESS);
   assert.equal(logs[1].args.value, amount);
 });
});

describe('when the given amount is greater than the balance of the sender', function () {
 const amount = MAX_TOKEN_SUPPLY.plus(1);

 it('reverts', async function () {
   await assertRevert(this.token.burn(amount, { from }));
```

```
    });
  });
});
describe('pause', function () {
  describe('when the sender is the token owner', function () {
    const from = owner;

    describe('when the token is unpaused', function () {
      it('pauses the token', async function () {
        await this.token.pause({ from });

        const paused = await this.token.paused();
        assert.equal(paused, true);
      });

      it('emits a paused event', async function () {
        const { logs } = await this.token.pause({ from });

        assert.equal(logs.length, 1);
        assert.equal(logs[0].event, 'Pause');
      });
    });

    describe('when the token is paused', function () {
      beforeEach(async function () {
        await this.token.pause({ from });
      });

      it('reverts', async function () {
        await assertRevert(this.token.pause({ from }));
      });
    });
  });
  describe('when the sender is not the token owner', function () {
    const from = anotherAccount;

    it('reverts', async function () {
      await assertRevert(this.token.pause({ from }));
    });
  });
});

describe('unpause', function () {
  describe('when the sender is the token owner', function () {
    const from = owner;

    describe('when the token is paused', function () {
      beforeEach(async function () {
        await this.token.pause({ from });
      });

      it('unpauses the token', async function () {
        await this.token.unpause({ from });

        const paused = await this.token.paused();
        assert.equal(paused, false);
      });

      it('emits an unpaused event', async function () {
        const { logs } = await this.token.unpause({ from });

        assert.equal(logs.length, 1);
        assert.equal(logs[0].event, 'Unpause');
      });
    });
```

```
      describe('when the token is unpaused', function () {
        it('reverts', async function () {
          await assertRevert(this.token.unpause({ from }));
        });
      });
    });

    describe('when the sender is not the token owner', function () {
      const from = anotherAccount;

      it('reverts', async function () {
        await assertRevert(this.token.unpause({ from }));
      });
    });
  });

  describe('pausable token', function () {
    beforeEach(async function () {
      await this.token.mint(owner, MAX_TOKEN_SUPPLY, { from: owner });
    });

    const from = owner;

    describe('paused', function () {
      it('is not paused by default', async function () {
        const paused = await this.token.paused({ from });

        assert.equal(paused, false);
      });

      it('is paused after being paused', async function () {
        await this.token.pause({ from });
        const paused = await this.token.paused({ from });

        assert.equal(paused, true);
      });

      it('is not paused after being paused and then unpaused', async function () {
        await this.token.pause({ from });
        await this.token.unpause({ from });
        const paused = await this.token.paused();

        assert.equal(paused, false);
      });
    });

    describe('transfer', function () {
      it('allows to transfer when unpaused', async function () {
        await this.token.transfer(recipient, MAX_TOKEN_SUPPLY, { from: owner });

        const senderBalance = await this.token.balanceOf(owner);
        assert.equal(senderBalance, 0);

        const recipientBalance = await this.token.balanceOf(recipient);
        assert(MAX_TOKEN_SUPPLY.eq(recipientBalance));
      });

      it('allows to transfer when paused and then unpaused', async function () {
        await this.token.pause({ from: owner });
        await this.token.unpause({ from: owner });

        await this.token.transfer(recipient, MAX_TOKEN_SUPPLY, { from: owner });

        const senderBalance = await this.token.balanceOf(owner);
        assert.equal(senderBalance, 0);
        const recipientBalance = await this.token.balanceOf(recipient);
        assert(MAX_TOKEN_SUPPLY.eq(recipientBalance));
      });

      it('reverts when trying to transfer when paused', async function () {
        await this.token.pause({ from: owner });

        await assertRevert(this.token.transfer(recipient, MAX_TOKEN_SUPPLY, { from: owner }));
      });
    });

    describe('approve', function () {
      it('allows to approve when unpaused', async function () {
        await this.token.approve(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 40);
      });

      it('allows to transfer when paused and then unpaused', async function () {
        await this.token.pause({ from: owner });
        await this.token.unpause({ from: owner });

        await this.token.approve(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 40);
      });

      it('reverts when trying to transfer when paused', async function () {
        await this.token.pause({ from: owner });
        await assertRevert(this.token.approve(anotherAccount, 40, { from: owner }));
      });
    });

    describe('transfer from', function () {
      beforeEach(async function () {
        await this.token.approve(anotherAccount, 50, { from: owner });
      });

      it('allows to transfer from when unpaused', async function () {
        await this.token.transferFrom(owner, recipient, 40, { from: anotherAccount });

        const senderBalance = await this.token.balanceOf(owner);
        assert(MAX_TOKEN_SUPPLY.minus(40).eq(senderBalance));

        const recipientBalance = await this.token.balanceOf(recipient);
        assert.equal(recipientBalance, 40);
      });

      it('allows to transfer when paused and then unpaused', async function () {
        await this.token.pause({ from: owner });
        await this.token.unpause({ from: owner });

        await this.token.transferFrom(owner, recipient, 40, { from: anotherAccount });

        const senderBalance = await this.token.balanceOf(owner);
        assert(MAX_TOKEN_SUPPLY.minus(40).eq(senderBalance));

        const recipientBalance = await this.token.balanceOf(recipient);
        assert.equal(recipientBalance, 40);
      });

      it('reverts when trying to transfer from when paused', async function () {
        await this.token.pause({ from: owner });

        await assertRevert(this.token.transferFrom(owner, recipient, 40, { from: anotherAccount }));
      });
    });

    describe('decrease approval', function () {
      beforeEach(async function () {
        await this.token.approve(anotherAccount, 100, { from: owner });
      });

      it('allows to decrease approval when unpaused', async function () {
        await this.token.decreaseApproval(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 60);
      });

      it('allows to decrease approval when paused and then unpaused', async function () {
        await this.token.pause({ from: owner });
        await this.token.unpause({ from: owner });

        await this.token.decreaseApproval(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 60);
      });

      it('reverts when trying to transfer when paused', async function () {
        await this.token.pause({ from: owner });

        await assertRevert(this.token.decreaseApproval(anotherAccount, 40, { from: owner }));
      });
      });

      describe('increase approval', function () {
       beforeEach(async function () {
        await this.token.approve(anotherAccount, 100, { from: owner });
       });

       it('allows to increase approval when unpaused', async function () {
        await this.token.increaseApproval(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 140);
       });

       it('allows to increase approval when paused and then unpaused', async function () {
        await this.token.pause({ from: owner });
        await this.token.unpause({ from: owner });

        await this.token.increaseApproval(anotherAccount, 40, { from: owner });

        const allowance = await this.token.allowance(owner, anotherAccount);
        assert.equal(allowance, 140);
       });

       it('reverts when trying to increase approval when paused', async function () {
        await this.token.pause({ from: owner });

        await assertRevert(this.token.increaseApproval(anotherAccount, 40, { from: owner }));
       });
      });
     });
    });
```

TTOToken.test.js code updated

We have proceeded to the update of unit test javascript file to integrate new tests related to the improvements of the code described in the previous section.

```
beforeEach(async function () {

  this.token = await
TOTToken.new('0xBa893462c1b714bFD801e918a4541e056f9bd924','0x2418C46F2FA422fE8Cd0BF56
Df5e27CbDeBB2590','0x84bE27E1d3AeD5e6CF40445891d3e2AB7d3d98e8',{ from: owner });

});
```

➔ We have initialized the declaration of wallet addresses with foundation, team and bounty addresses during deployment.

```
describe('constructor', function () {

  it('should revert when creating token with wallets addresses equals to zero', async function () {

    await assertRevert(TOTToken.new('0x0','0x0','0x0',{ from: owner }));

  });

});
```

➔ We have added a function to make sure you can not deploy the contract with null addresses.

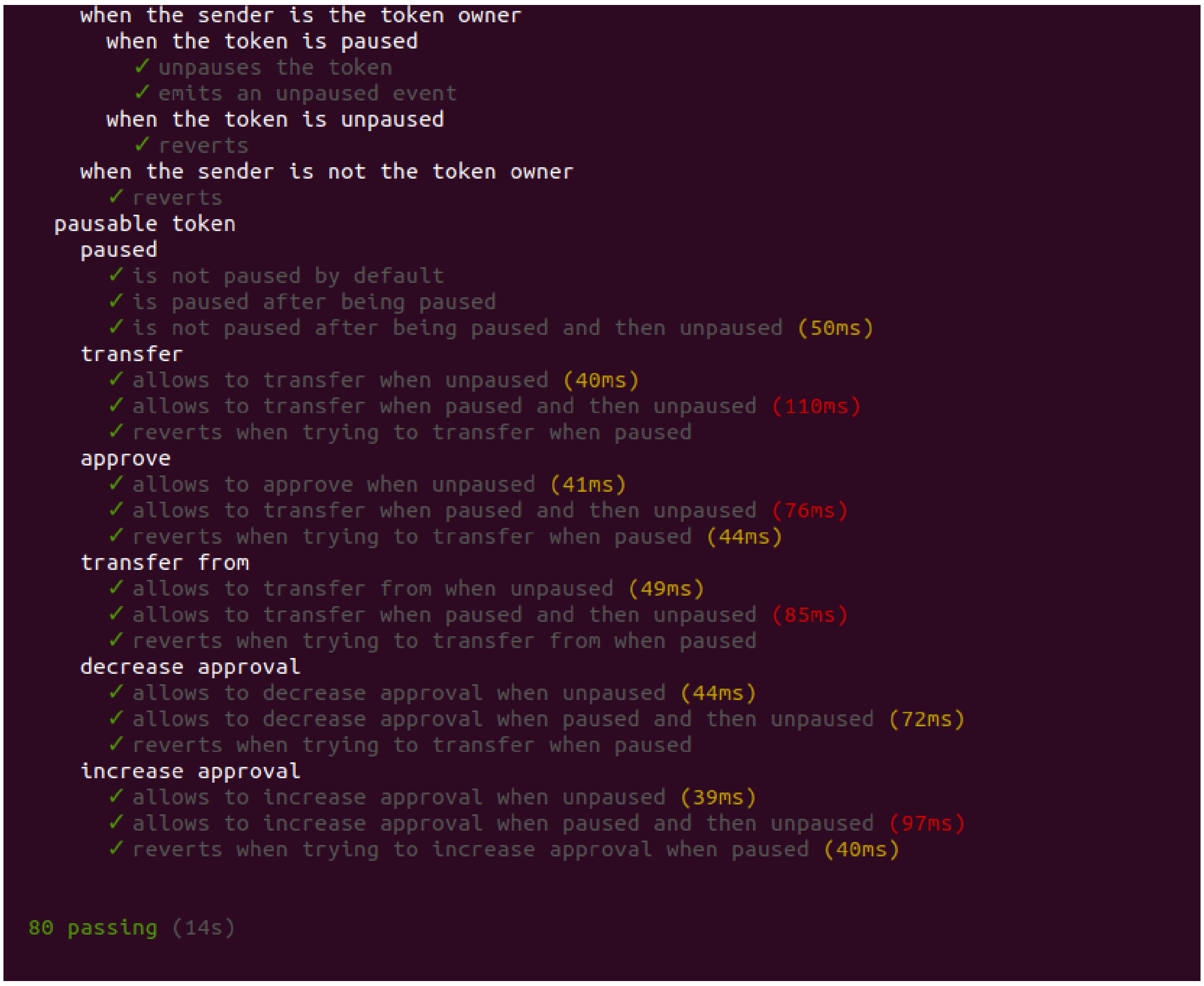
```
it('should revert when finishMinting when wallets are not set', async function () {

    await this.token.mint(owner, PURCHASER_AMOUNT, { from: owner });

    await assertRevert(this.token.finishMinting({ from: owner }));

  });
```

➔ We have deleted this function because we have already declared all the addresses and they can not be null.



```
      when the sender is the token owner
        when the token is paused
          ✓ unpauses the token
          ✓ emits an unpaused event
        when the token is unpaused
          ✓ reverts
      when the sender is not the token owner
        ✓ reverts
    pausable token
      paused
        ✓ is not paused by default
        ✓ is paused after being paused
        ✓ is not paused after being paused and then unpaused (50ms)
      transfer
        ✓ allows to transfer when unpaused (40ms)
        ✓ allows to transfer when paused and then unpaused (110ms)
        ✓ reverts when trying to transfer when paused
      approve
        ✓ allows to approve when unpaused (41ms)
        ✓ allows to transfer when paused and then unpaused (76ms)
        ✓ reverts when trying to transfer when paused (44ms)
      transfer from
        ✓ allows to transfer from when unpaused (49ms)
        ✓ allows to transfer when paused and then unpaused (85ms)
        ✓ reverts when trying to transfer from when paused
      decrease approval
        ✓ allows to decrease approval when unpaused (44ms)
        ✓ allows to decrease approval when paused and then unpaused (72ms)
        ✓ reverts when trying to transfer when paused
      increase approval
        ✓ allows to increase approval when unpaused (39ms)
        ✓ allows to increase approval when paused and then unpaused (97ms)
        ✓ reverts when trying to increase approval when paused (40ms)


  80 passing (14s)
```

Deployment tests

All contracts are deployed with success on the top of the ropsten network with the updated code. You can check on etherscan :

TOTToken.sol :
https://ropsten.etherscan.io/tx/0x9469efe8f5d2891149e548710198159f34d053a94d1a68a2af23a359c17278d8

Interaction tests :

Here we will describe the testing process :

1. Deploy TOTToken : OK (with wallet addresses equals to zero)

@bountyAddress : 0x0000000000000000000000000000000000000000

@teamAddress : 0x0000000000000000000000000000000000000000

@foundationAddress : 0x0000000000000000000000000000000000000000

https://ropsten.etherscan.io/tx/0x5243def784e0c31c2f2cbef0c4caddd1e4877ae68181adf3ddac648f57b699e7

2. Deploy TOTToken : OK

@bountyAddress : 0x96022C4667aF840528BB95eaBe8Cd3da705452c4

@teamAddress : 0x59c6cF7b15B8bf692c20Ae13Ebe2Aa3d3A996887

@foundationAddress : 0xDf722C698C43b9556c32D95efB9D60de2FdE4a0a

https://ropsten.etherscan.io/tx/0xe381572fc8b886c9a7e516240ab82b35eb16efc0630eca7dab68e8d6fa108196

3. Deploy updateWallet() : OK

@bountyAddress : 0x08D1896Ed3029C93Db708eA026670695e29236f8

@teamAddress : 0xb13a10cD2F7eCC3085a311dBEAC2a3F28d2CedCd

@foundationAddress :

https://ropsten.etherscan.io/tx/0x9b932e237b1d81769788079a8c92cbfc996ba80727132620f5504d27cab69837

4. Call setName() (When the sender is not the Owner) : OK

https://ropsten.etherscan.io/tx/0xc76cadc2052f389b3d187656b193c8254b667002068582479d75cb695e9f5317

5. Call setSymbol() : OK

https://ropsten.etherscan.io/tx/0x47d3ef20f3aeb434c32f2a04c5593b7556284ee380eee931dfeb3d8121c3fc4a

6. batchMint() : OK

@contributor1 : 0x96022C4667aF840528BB95eaBe8Cd3da705452c4 (amount = 37500000000)

@contributor2 : 0x23669cD2061FdfF6f1E04B43709324A4E8120C21 (amount = 37500000000)

https://ropsten.etherscan.io/tx/0x12aa90c0413ce36a49ec0944f32bf74bcf7082d51f8e8ebf79d16237f3aa8609

7. Check TotalSupply = 75000000000 OK

8. Call transferFrom() (from contributor1 to contributor2 when paused = true) : OK

https://ropsten.etherscan.io/tx/0xba21c367cca43d80e1826d6adeed8717b2b7f9b106e04fe77ffc2a3f865cf02f

9. Call FinishMinting() : OK

https://ropsten.etherscan.io/tx/0x2f34111b7577f6deb56302c3d25653f763b833556a2d348ef874ef04ff8b47d0

10. Check balance of bountyAddress, teamAddress and foundationAddress : OK

@balanceOf("0x08D1896Ed3029C93Db708eA026670695e29236f8") = 5000000000

@balanceOf("0xb13a10cD2F7eCC3085a311dBEAC2a3F28d2CedCd") = 15000000000

@balanceOf("0x24202079C04feC583944d01166F1CdD67c09f930") = 5000000000

11. Check TotalSupply = 100000000000 OK