MANUAL

How to develop your own dApp on dApp Builder platform?

How to develop your own dApp 2

Step 1: Write your own Smart Contract 2

Step 2: Fill the dApp information 3

Step 3: Construct the dApp creation form 7

Step 4: Customize the dApp Creation JS 10

Step 5: Customize the dApp Interface 12

Step 6: Create a JS preview for your dApp 17

How to test your custom dApp 19

How to develop your own dApp on dApp Builder platform?

To develop **your own dApp** (with your own smart contract and HTML interface) and allow other users to create their own instances of your dApp, go to [this page]() and go through **the 6 steps of Custom dApp creation**.

Step 1: Write your own Smart Contract

In the beginning you need to write your Solidity smart contract. Our platform supports two architectures for smart contracts:

1. 1 dApp = 1 smart contract
2. 1 smart contract for all dApps

1 dApp = 1 smart contract

In this case when a user creates an instance of your dApp, he deploys his own smart contract with your Solidity code. If you prefer this dApp architecture, you need to write all the common dApp logic in your smart contract and to make available to set values of customizable parameters as constructor arguments.

Please look at this example - it's Multisignature Wallet dApp based on this architecture.

1 smart contract for all dApps

In this case your smart contract will contain all the instances of your dApp. The user's dApp instances must be stored in the contract as mappings:

address => dapp[ ] ,

where **address** is the Ethereum address of a user, and **dapp[ ]** is an array of structures of this user's dApp instances.
Also you must create **the getDappId(address creator, bytes32 dappName)** method in your contract. This method must return two values: the dApp instance id in the contract storage and boolean TRUE if the dApp instance exists in the contract by creator's address and dApp instance name.

Please read our smart contract for Betting dApp as example for creating your own contract.

Step 2: Fill the dApp information

After creating your smart contract you need to fill some information about your dApp:

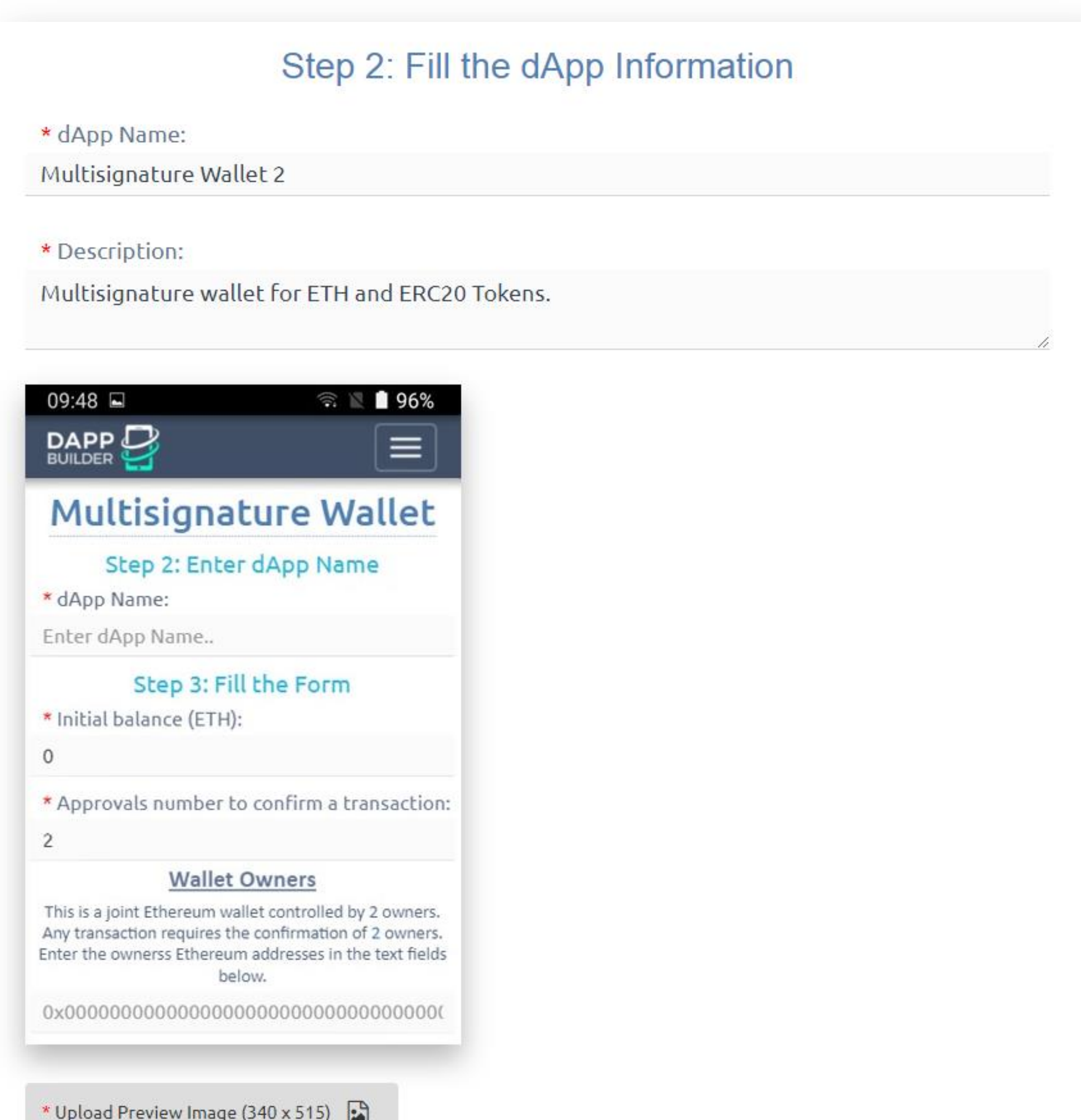
- **dApp Name** - the name of your decentralized application
- **Description** - the short description of your dApp
- **Preview image** - the preview image of your dApp
- **dApp Type** - the type of your dApp's architecture, **1 dApp = 1 smart contract** or **1 smart contract for all dApps**
- **ABI** - application binary interface of your dApp in JSON format, you can get it in Remix while deploying your contract

If you choose the type "**1 dApp = 1 smart contract**", you will need to fill also this information:

- **Solidity Code** - the code of your smart contract
- **Compiled Contract Code** - the compiled code of your contract, you can get it in Remix
- **Contract Name** - the name of your smart contract, this information is needed for dApp verification on Etherscan
- **Compiler Version** - the version of the compiler with which you created your compiled code from your Solidity code in Remix
- **Optimization** - set "Yes" if you used optimization in Remix for compiling your contract
- **Types of constructor parameters** - if your smart contract constructor method has some parameters, you need to enter their types separated by commas, for example: **uint, address, string**

If you choose the type "**1 smart contract for all dApps**", you will need to deploy your contract and to verify the contract code on Etherscan and after that fill this data:

- **Main Ethereum Address** - the address of your contract in the Main Ethereum Network
- **Rinkeby Address** - the address of your contract in the Rinkeby Test Net


Step 2: Fill the dApp Information
* dApp Name:
Multisignature Wallet 2
* Description:
Multisignature wallet for ETH and ERC20 Tokens.
09:48
96%
DAPP
BUILDER
Multisignature Wallet
Step 2: Enter dApp Name
* dApp Name:
Enter dApp Name..
Step 3: Fill the Form
* Initial balance (ETH):
0
* Approvals number to confirm a transaction:
2
Wallet Owners
This is a joint Ethereum wallet controlled by 2 owners.
Any transaction requires the confirmation of 2 owners.
Enter the ownerss Ethereum addresses in the text fields below.
* Upload Preview Image (340 x 515)

* dApp Type:

1 dApp = 1 Smart Contract

* Solidity Code:

```
pragma solidity 0.4.24;

interface tokenInterface {
   function transfer(address reciever, uint amount) external;
   function balanceOf(address owner) external returns (uint256);
}

contract dapMultisig {

   /*
   * Types
   */
   struct Transaction {
      uint id;
```

* Compiled Contract Code:

0x608060405260405162001962380380620019628339810160409081528151602083015191
830151909291909101908151831115620003c57600080fd5b600081905560018054600160a
060020a031916331790553460025581516200006c906003906020850190620010e565b508
26007819055507f0d1013664d1afda1c712a76630e584de8dd1d656331b90608aedc997fdb7
839e3383604051808360016 0a060020a0316600160a060020a031681526020018060200182
81038252838181518152602001915080519060200190602002808383600005b838110156200
00f157818101518382015260200162000 0d7565b505050509050019350505050604051809 1
0390a150505062000 1a2565b82805482825590600052602060002090810192821562000166
57916020028201 5b8281111562000166578251825460016 0a060020a031916600160a06002

* Contract Name (for Etherscan verification):

dapMultisig

* Compiler Version (for Etherscan verification):

v0.4.24+commit.e67f0147

* Optimization (for Etherscan verification):

Yes

* Types of constructor parameters, separated by commas (for Etherscan verification):

uint, address[], bytes32

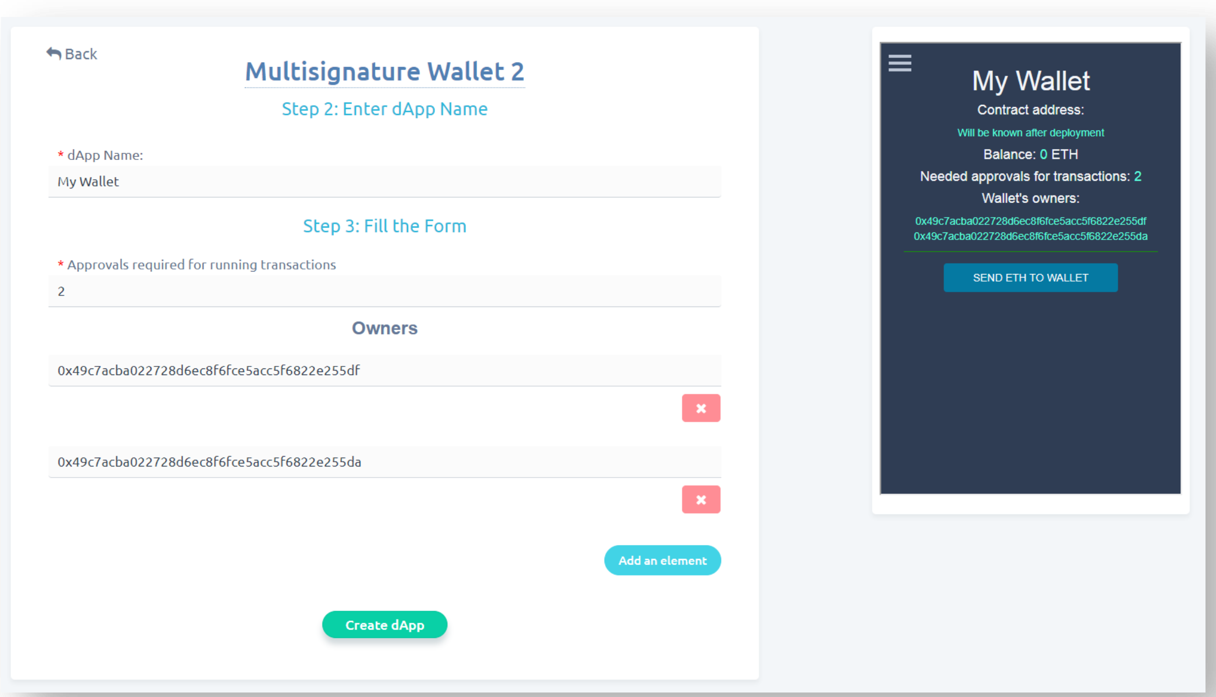
* ABI:

[{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"owners","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"creator","outputs":[{"name":"","type":"address"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"bytes32"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[],"name":"approvalsreq","outputs":[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":true,"inputs":[{"name":"logId","type":"uint256"}],"name":"getLog","outputs":[{"name":"","type":"address"},{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},

Back Next

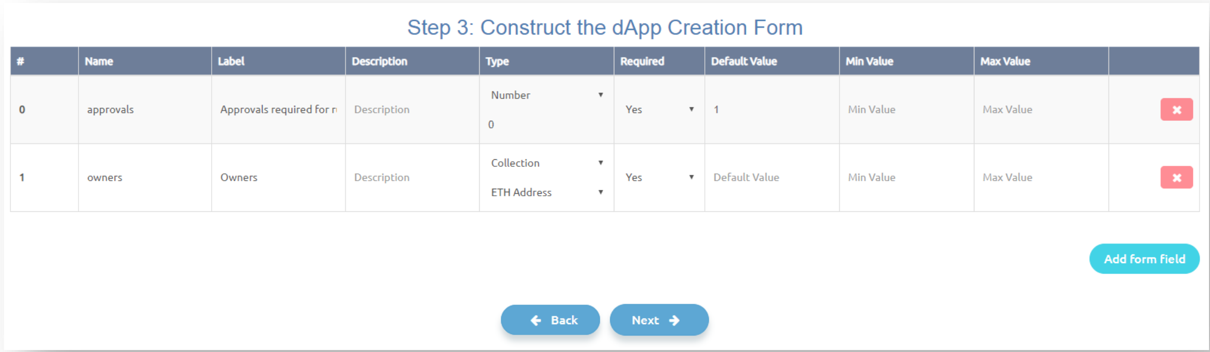
Step 3: Construct the dApp creation form

On this step you need to create the form for the users of dApp Builder. By this form they will create instances of your dApp on this page.

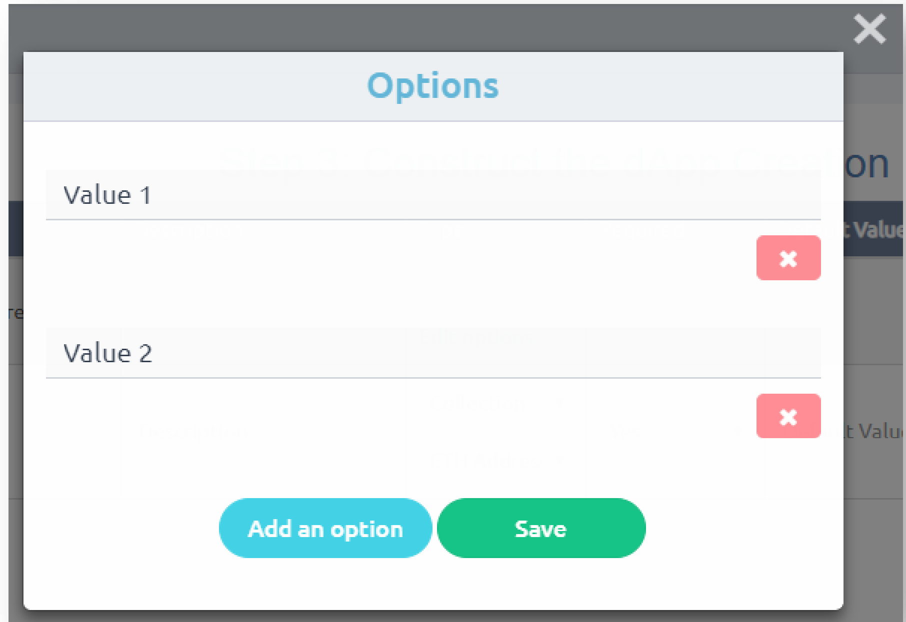


You can create the form by adding and customizing the form fields. For each field you can set:

1. **Index number** - determines the position of the field on the form. The index number of the first field must be 0, 1 for the second field, 2 for the third etc. Pay attention that dApp Name already exists on the form, you need to add the other fields
2. **Name** - the name of property of the form object. When user submits the form, it creates the JS object containing properties matching the form fields
3. **Label**
4. **Description**
5. **Type** - the type of the field: **Text**, **ETH address**, **Number**, **Select** or **Collection**
6. **Required** - is the field required or not
7. **Default Value**
8. **Min Value** - minimum value for the number field or minimal number of characters for the text field
9. **Max Value** - maximum value for the number field or maximal number of characters for the text field



If the form type is **Number**, you can also set the decimals number for the field's values. For the **Select** field you need to set the list of the possible values:



Collection is the field type for the set of values. For example, the field "Owners" in our Multisignature wallet: user can add so many owners so he want. The collection fields also can have **Text**, **ETH Address**, **Number** or **Select** type.

Owners

0x49c7acba022728d6ec8f6fce5acc5f6822e255df

0x49c7acba022728d6ec8f6fce5acc5f6822e255da

Add an element

Step 4: Customize the dApp Creation JS

On this step you need to complete our JavaScript template by your own code for creating an instance of your dApp in the blockchain after submitting the creation form.

- If you choose the "**1 smart contract for all dApps**" dApp type, you need to create an Ethereum transaction for calling the method in your contract for adding a new dApp instance to the contract storage.

- Otherwise, if you choose "**1 dApp = 1 smart contract**", you just need to paste the correct parameters in the contract creating construction. Please read our comments in the JS template, they will help you to write your own code.

Step 4: Customize the dApp Creation JS

```
var customDapp = (function(){

if (web3.version.network == "1") {
    var network = 'main';
} else if (web3.version.network == "4") {
    var network = 'rinkeby';
}

return {

    create: function(data){

        /*
         * The "data" parameter is an object containing the submitted creation form.
         * data = {"name":"dApp Name","creator":"ETH address of dApp creator","approvals":0,"owners":[]}
         *
         */

        var smartContract = web3.eth.contract([{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"own

        /*
         * Type here your additional code for dApp creation.
         *
         * If you need to do an additional form validation
         * use showError('Error text') function for showing errors
         * and "return false;" construction for stopping the dApp creation.
         *
         */
```

```
        smartContract.new(

            /*
             * Type here the variables (separated by commas)
             * that you need to send to the contract constructor as parameters.
             *
             * For example: data.name, data.approvals
             *
             * /
            data.approvals, data.owners, data.name ,

            {from: data.creator, data: '0x6080604052604051620019623803806200196283398101604090815281516020830151918

            function (e, contract){
                if (!e){
                    //Adding dApp into dApp Builder database
                    window.customContractUndeployed = {txnHash: contract.transactionHash};
                    addApp('custom-dapp', data.name, network);
                }
            }
        );

    }

};

})()
```

Back Next

Step 5: Customize the dApp Interface

On this step you need to create an HTML interface for your dApp by adding your own **HTML**, **JS** and **CSS** code into our template.

dApp interface on dApp Buiilder platform is a single page HTML5 application, that communicates with the blockchain via **Web3.js library**, Your interface must find the user's dApp instance in the blockchain on the loading and to allow users to interact with the smart contract.

Step 5: Customize the dApp Interface

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="assets/css/bootstrap.min.css" rel="stylesheet">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <script type="text/javascript" src="assets/js/jquery.min.js"></script>
</head>
<body>
<div class="main">

    <!-- Put here the HTML code of your dApp interface -->

    <div class="container main-container text-center">
        <div class="dapp-block" id="info-block">
            <h1 id="name"></h1>
            <h4>Contract address: <span class="eth_address" id="address"></span></h4>
            <h4>Balance: <span class="eth_address" id="balance"></span> ETH</h4>
            <h4>Needed approvals for transactions: <span class="eth_address" id="approvals"></span></h4>
            <h4>Wallet's owners:</h4>
            <div id="owners">
            </div>
        </div>
        <div id="unsent-wrapper">
            <div style="display:none;padding-bottom:0;border-bottom:0;" class="dapp-block" id="unsent-block">
                <h4 id="unsent-count"></h4>
                <div class="row"></div>
            </div>
        </div>
        <div style="display:none;padding-bottom:0;border-bottom:0;" class="dapp-block" id="sent-block">
            <h4 id="sent-count"></h4>
            <div class="row"></div>
        </div>

</div>

<!-- Transaction pending modal window -->
<div class="modal-txn container text-center">
    <div class="row">
        <div class="col-md-12">
            <h1>Transaction sent.</h1>
            <p>Please wait until it gets mined.</p>
            <p>Information on the page will update automatically.</p>
            <button class="btn btn-success btn-ok ok-back">Ok, back</button>
```

```
        </div>
    </div>
</div>

<script type="text/javascript">

    var dapp = (function(){

        var contract_address = '{{ SMART_CONTRACT_ADDRESS }}';

        var ABI = [{"constant":true,"inputs":[{"name":"","type":"uint256"}],"name":"owners","outputs":[{"name":"","

        /*
         * You can put some JS code here if it's necessary.
         *
         */

        let erc20ABI = [{"constant":true,"inputs":[],"name":"name","outputs":[{"name":"","type":"string"}],"payable":f
        let wallet = {};
        let isOwner = false;
        wallet.tokens = [];
        wallet.tokenTransactions = [];
        let id = current_dapp_id;
        let contract;


        return {

            /*
             * dApp initializing function.
             *
             * "creator" parameter is the ETH-address of dApp creator,
             * "name" parameter is the dApp name.
             *
             */
            init: function (creator, name) {

                window.contract = web3.eth.contract(ABI).at(contract_address);
```

```
/*
 * Put here your initializing JS code.
 *
 * In the end of dApp interface rendering you should call the managePlaceholders() function,
 * it is a built-in function of dApp builder, that removes the loading placeholder and shows your i
 *
 */

            $('#name').text(name).show();
            $('#address').text(contract_address).show();
            this.getName().then(name => {
                wallet.name = name;
                return this.getOwners();
            })
                .then(owners => {
                    wallet.owners = owners;
                    //check if user is owner of wallet globally
                    for (let i = 0; i < wallet.owners.length; i++) {
                        $("#owners").append('<p class="eth_address">' + wallet.owners[i] + '</p>');
                        if (wallet.owners[i].toLowerCase() == web3.eth.defaultAccount.toLowerCase()) is
                    }
                    return this.getTransactions()
                })
                .then(() => {
                    return this.getCreator();
                })
                .then(creator => {
                    wallet.creator = creator;

},

/*
 * This function shows the transaction pending modal window.
 * Call this function when a user sends a transaction in your dApp.
 *
 */
showModal: function(){
    $('.main').hide();
    $('.modal-txn').show();
    $('.ok-back').unbind();
    $('.ok-back').click(function(){
        $('.modal-txn').hide();
        $('.main').show();
    });
},
```

```
            /*
             * Put here the other functions for your interface.
             */

            closeNav: function(){
                    $('.sidebar-menu, .sidebar-menu > li').css('width', 0);
                    $('.overlay').hide();
                    $('.hamb-btn').show();
                },
            openNav: function(){
                    $('.sidebar-menu, .sidebar-menu > li').css('width', '50%');
                    $('.overlay').show();
                    $('.hamb-btn').hide();
                },
                getName: function(){
                        return new Promise(resolve => {
                            contract.name(function(e,d){
                                if (!e){
                                    resolve(d);
                                }
                            })
                        })
                    },
                    getCreator: function(){

        }
    })();

    window.addEventListener('load', function(){
        if (typeof(window.web3.eth.defaultAccount) != 'undefined' && window.web3.eth.defaultAccount) {
            dapp.init('{{ CREATOR_ADDRESS }}', '{{ DAPP_NAME }}');
        } else {
            var initTimer = setInterval(function(){
                if (typeof(window.web3.eth.defaultAccount) != 'undefined' && window.web3.eth.defaultAccount) {
                    dapp.init('{{ CREATOR_ADDRESS }}', '{{ DAPP_NAME }}');
                    clearInterval(initTimer);
                }
            }, 1000);
        }
    });

</script>
```

```
<style>

    /*
     * TEXT_COLOR, BACKGROUND_COLOR, ETH_ADDRESS_COLOR,
     * LINKS_COLOR, OK_BUTTONS_COLOR, CANCEL_BUTTONS_COLOR
     * and HEADERS_COLOR are the customizable by end-users parameters.
     *
     * For example, if you want the color of ETH-addresses in your dapp to be customizable,
     * put them to the <span> with .eth_address class.
     *
     */
    body{
        color: {{ TEXT_COLOR }};
        background-color: {{ BACKGROUND_COLOR }};
    }
    .eth_address{
        color: {{ ETH_ADDRESS_COLOR }};
    }
    a, a:focus, a:hover, a:active{
        text-decoration: none;
        color: {{ LINKS_COLOR }};
    }
    .btn.btn-danger.btn-cancel{
        background-color: {{ CANCEL_BUTTONS_COLOR }};
        border-color: {{ CANCEL_BUTTONS_COLOR }};
    }
    .btn.btn-success.btn-ok{
        background-color: {{ OK_BUTTONS_COLOR }};
        border-color: {{ OK_BUTTONS_COLOR }};
    }
    h1,h2,h3,h4,h5,h6{
        color: {{ HEADERS_COLOR }};
    }
    .modal-txn{
        display: none;
        z-index: 999;
        width: 100%;
        height: 100%;
        position: absolute;
        left: 0;
        top: 0;
        background-color: {{ BACKGROUND_COLOR }};
    }
```

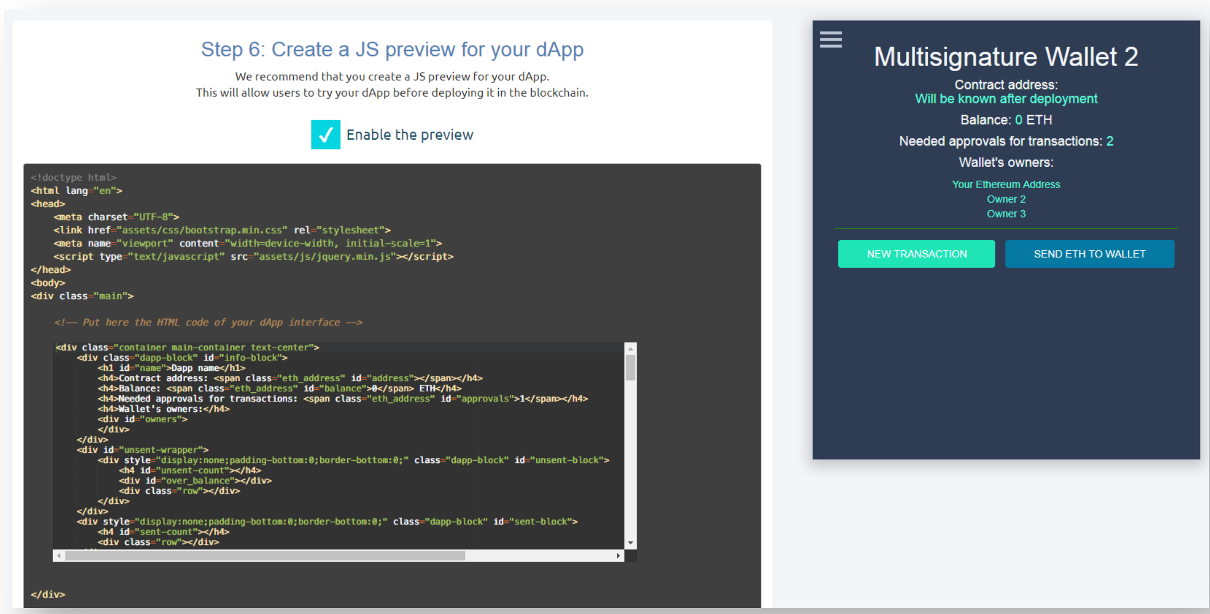
```
    /*
     * Put here your own CSS.
     *
     */

    .btn{
            padding: 8px 38px; text-transform: uppercase; margin:5px;
        }
        .dapp-block, #nav-block{
            padding-top: 10px;
            padding-bottom: 10px;
        }
        .dapp-block, .delimiter{
            border-bottom:1px solid #18960a;
        }
    .transaction{
            padding-top: 10px;
        }
        .delimiter{
            padding-top:10px;
        }
        .close{
            position: absolute;
            top: 15px;
            right: 15px;
        }

</style>
</body>
</html>
```

Back Next

Step 6: Create a JS preview for your dApp

On the last step of your dApp creating you need to create a **JS preview** for your dApp.

JS preview does not communicate with the blockchain, it just simulates the working of the real dApp. The creation of preview is not necessary but it is strongly recommended, because it allows users to try your dApp working before deploying it into the blockchain.



Please try our JS preview for [multisignature wallet]() for example to understand the purpose of the JS preview and it's workflow.

You can test your JS preview in real time while developing it. Use the "**Test parameters**" field to send the test data for your dApp into the simulator. This data must be in **JSON** format. Also the data that you enter in this field will be shown in the marketplace when you add your dapp there.

```
 * Put here your own CSS.
 *
 */

.btn{
        padding: 8px 38px; text-transform: uppercase; margin:5px;
    }
    .dapp-block, #nav-block{
        padding-top: 10px;
        padding-bottom: 10px;
    }
    .dapp-block, .delimiter{
        border-bottom:1px solid #18960a;
    }
.transaction{
        padding-top: 10px;
    }
    .delimiter{
        padding-top:10px;
    }
    .close{
        position: absolute;
        top: 15px;
        right: 15px;
    }

</style>
</body>
</html>
```

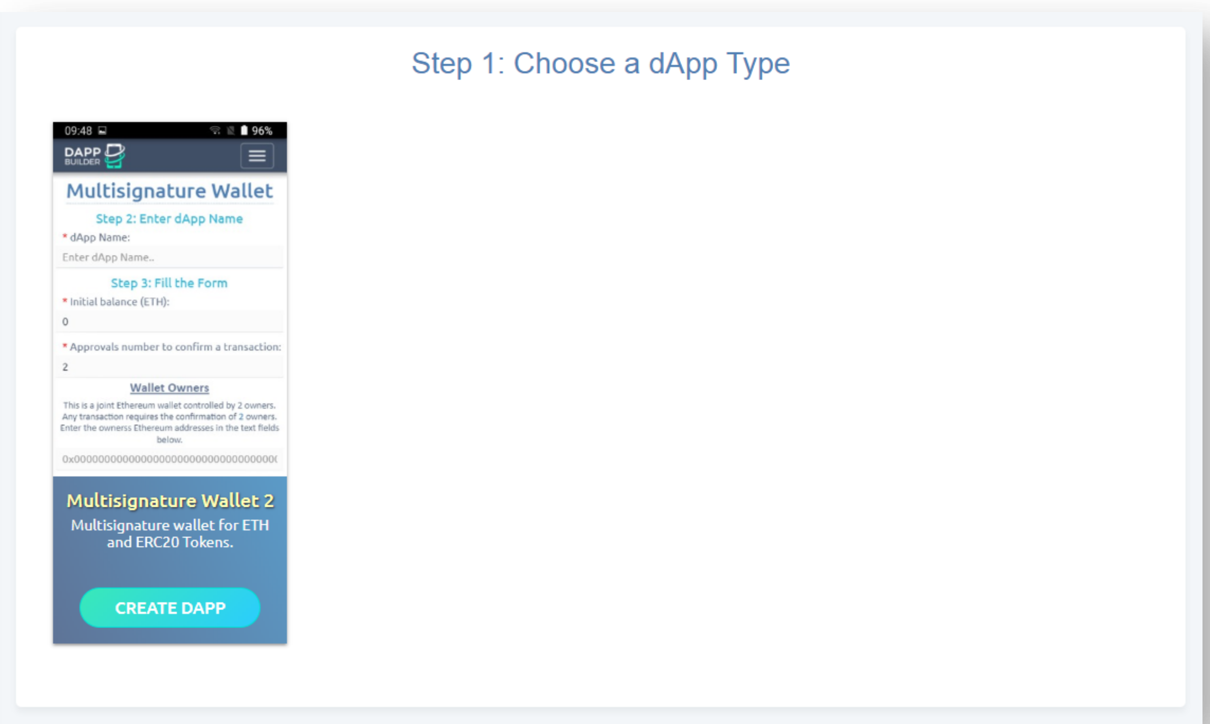
Test parameters:

{"name":"Multisignature Wallet 2","creator":"Your ETH address","approvals":2,"owners":["Your Ethereum Address", "Owne

Back Next

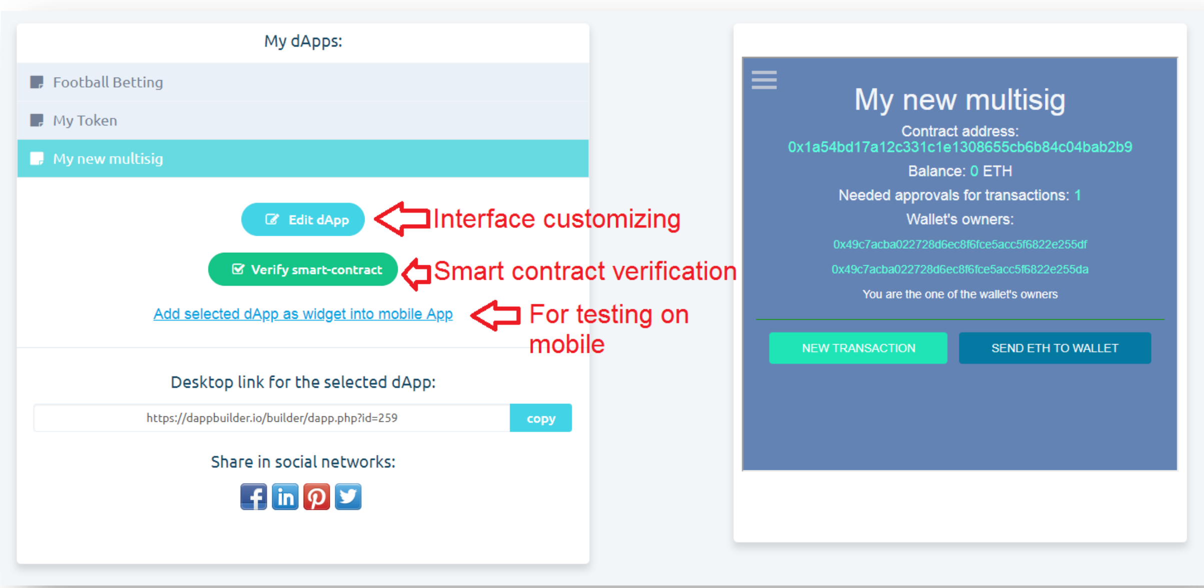
How to test your custom dApp

After creating of your dApp you can test it on [Custom dApps testing page](). You can create an instance of your new dApp and then use it in the desktop browser or on Android mobile device.



What is necessary to test:

1. dApp creation form
2. JS preview
3. dApp creation script
4. Working with dApp via your HTML interface in desktop browser and on Android device
5. Smart contract validation (only for "1 smart contract = 1 dApp" dApps)
6. Customizing the dApp interface



If you need to **modify** your custom dApp, you can do it in [the list of your custom dApps.]