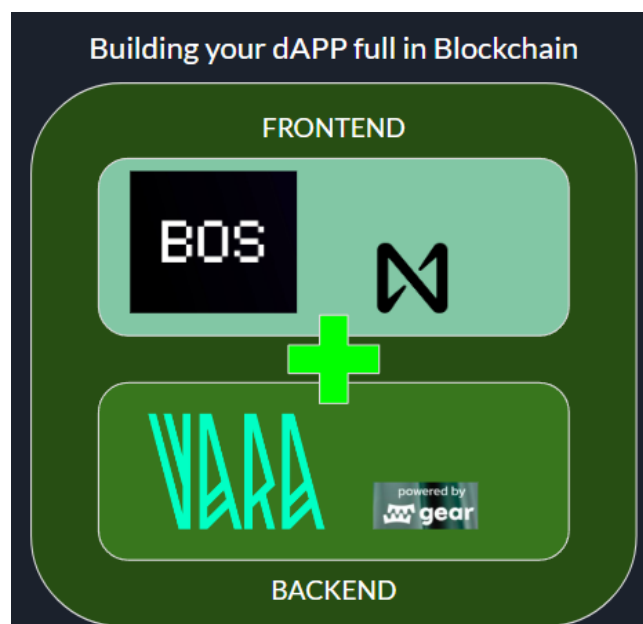


Component development tutorial for interaction with FT contract on VARA + BOS

What is the stack of an application using VARA + BOS?

Hello everyone, welcome once again to an Open Web Academy tutorial. This time, we will present how to develop a component in the Blockchain Operating System (BOS) that utilizes the VARA Network by interacting with a Fungible Tokens (FT) contract, allowing you to read its information and mint tokens.



Component development

The component we will be developing during this tutorial is shown in the following image. As you can see, it is divided into three different sections. The first section contains various buttons that allow us to interact with the VARA Network. In the second section, we can see the information of the wallet selected for interaction, and lastly, the section where the information obtained from the FT contract is displayed.


Vara Interaction Functions

Test Sign Transaction

Reload Data

Reload Account Information

Account Data

 Syi216

0x7425535ece072919c4455f8bb2ded0e8d2cbc873f9c48ad289f626d10ae51a77

Fungible Token Contract Data

Token Name: OWA Token Test

Symbol: OWATest

Decimals: 18

Total Supply: 2525

Balances:

Account	Balance
0x7425535ece072919c4455f8bb2ded0e8d2cbc873f9c48ad289f626d10ae51a77	410
0x2e0f487cf8418bd89edc330586f45c1b56c8ab93305b212c0c5852c8d3ed2402	1004
0x80b92e8c46670db9b72715cf6dbffc5d3c45229b9b8882038d81102e59d6161f	1111

To start the development we declare the variables that we will need for data management and interaction with the contract found in VARA Network, for that we will write the following code.

```
const contract =  
  "Contract address";  
const contractData =  
  "Contract metadata";  
const [ftData, setFtData] = useState(undefined);  
const [account, setAccount] = useState(undefined);
```

As can be seen in the previous code section, two variables are declared. The first one is the **contract** variable, which is where we will store the address of the contract we want to interact with. The other variable is **contractData**, which will contain the metadata corresponding to the contract. This metadata includes all the information about the functions that can be used and the data structures of the contract in an encrypted manner.

It can also be seen in the previous code that two **useState** hooks have been declared. These are the ways we can manage the states of information within React's rendering so that these values do not change when a new render of the React component occurs due to a state change. In this case, we are declaring the state pairs for **ftData** and **account**, which will allow us to store and read the information we obtain during the interaction with the VARA Network.

Now we will begin explaining the first section of our component. In this section, the user will be able to interact with the Fungible Tokens contract through buttons, as seen in the image below. Here we can find a button to sign a transaction (mint tokens), read the contract's state and retrieve all its information, and finally, get the current information of the user's account.

Vara Interaction Functions

[Test Sign Transaction](#)[Test Read State](#)[Get Account Information](#)

In order to generate this section, the following code is used:

```
<div className="border border-black p-3 rounded">
  <h4 className="mb-2">Vara Interaction Functions</h4>
  <div className="d-flex flex-row gap-3">
    //Button to execute a transaction on VARA Network
    <VaraNetwork.Interaction
      trigger=(({ signTransaction }) => (
        <>
          <button
            onClick={() => {
              signTransaction(
                contract,
                contractData,
                { mint: 10 },
                899819245,
                0
              );
            }}
          >
            Test Sign Transaction
          </button>
        </>
      )}
    />
    //Button to read the contract status in VARA Network
    <VaraNetwork.Interaction
      trigger=(({ readState }) => (
        <>
          <button
            onClick={() => {
              const info = readState(contract, contractData, "");
              info.then((res) => {
                console.log("ReadState", res);
                setFtData(res);
              });
            }}
          >
            {ftData ? "Reload Data" : "Test Read State"}
          </button>
        </>
      )}
    />
    //Button to obtain user account information
    <VaraNetwork.Interaction
```

```

trigger=(({ getAccountInfo }) => (
  <>
    <button
      onClick={ () => {
        setAccount(getAccountInfo());
      }}
    >
      {account
        ? "Reload Account Information"
        : "Get Account Information"}
    </button>
  </>
)}
/>
</div>
</div>

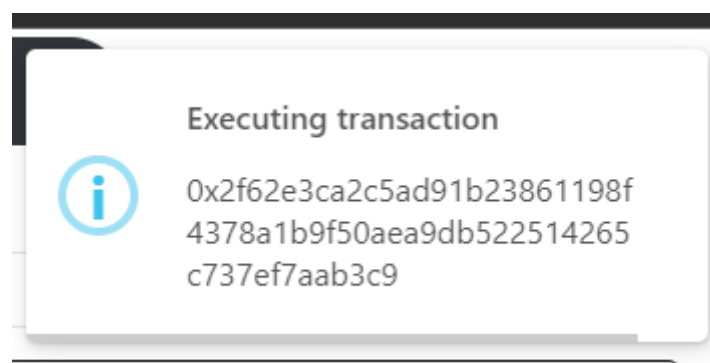
```

As you can see in the previous code, to interact with the VARA Network, we need to use the **<VaraNetwork.Interaction/>** element. This element contains all the necessary functions to use the network, and we can choose between the different functions through the **trigger** parameter. This parameter can call the functions **signTransaction**, **readState**, and **getAccountInfo**.

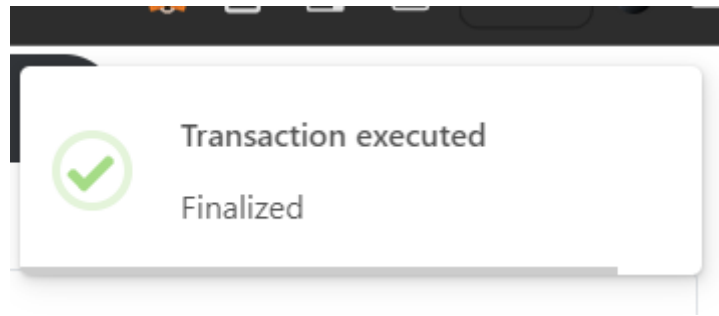
Each of the functions interacts differently. For example, **signTransaction** receives the following parameters:

- **programId** (Contract address)
- **metadata** (Metadata belonging to the contract)
- **params** (Parameters sent to execute the Sign Transaction)
- **gas** (Gas that will be sent to execute the transaction)
- **value** (Amount of VARA tokens to send in the transaction)

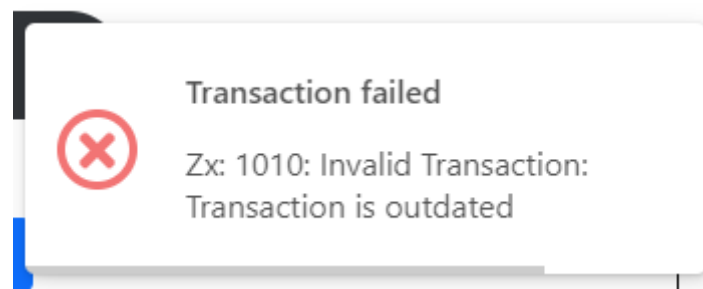
With these parameters, it allows the execution of the function we want to call, in this case, the **mint** function found in the FT contract. During the execution of the transaction, different alerts will be shown to see the status of the transaction. These alerts are as follows:



The previous image shows an alert that warns that the transaction is being executed, where it also shows us the ID of the transaction that we have just carried out.



The image above shows an alert notifying us that the transaction has been executed correctly.



Finally, in this alert we can see that there was an error when executing the transaction, and we can see what this error was.

In the case of the **readState** function, it receives the following parameters:

- **programId** (Contract address)
- **metadata** (Metadata belonging to the contract)
- **params** (Parameters sent to perform the Read State)

By executing the function with these parameters, we will receive a promise containing the contract's state information. To handle this, we can store the function execution in a variable and later retrieve this information using **.then((res) => {})**, and in this case, save the information using **setFtData()** that we previously declared in the **useState**.

Finally, we have the **getAccountInfo** function, which does not receive any parameters. When called, it returns the object corresponding to the active wallet within the gateway. This object contains information such as addresses and metadata, which we can display to the user if desired. In this case, we store the information in the **account** state that we declared earlier using the **setAccount()** function.

To call these functions, we need to use the **trigger** parameter found in the **Interaction** element. Within the **trigger**, we can specify which functions to call along with their corresponding buttons to execute them. The basic structure of the element along with the trigger is written as follows:

```
<VaraNetwork.Interaction
  trigger=(({ signTransaction, readState, getAccountInfo }) => (
    <>
```

```

        <button
          onClick={ () => {
            //Here we write the function we want to call
          }}
        >
          "Reload Account Information"
        </button>
      </>
    ) }
  />

```

All functions must be called through a button within the trigger to execute them within the component.

And with this, we conclude the section where we can interact with the VARA Network within BOS.

Now we will move on to the second section of the code for our component. In this part, the user's account information will be displayed as shown in the following image:



Here we can visualize the user's account name, the Identicon corresponding to the Polkadot address, and the decoded address. For this, the following code is used:

```

{account && (
  <div className="border border-black p-3 rounded">
    <h4 className="mb-2">Account Data</h4>
    <div className="d-flex flex-row gap-2">
      <VaraNetwork.Identicon size={30} />
      <p className="m-0 fw-bold">{account.meta.name}</p>
    </div>
    <p className="fw-semibold">{account.decodedAddress}</p>
  </div>
) }

```

As you can see in the code above, a validation has been added so that once the user's account information is obtained, it is displayed in the interface along with the implementation of the `<VaraNetwork.Identicon size={30} />` element, which receives the **size** parameter indicating its size in pixels.

Finally, there is the section where we display the information we have obtained from the VARA Network contract. As seen in the image below, here we show the token name, symbol, decimals, supply, and the current balances of the users in a table.

Fungible Token Contract Data

Token Name: **OWA Token Test**

Symbol: **OWATest**

Decimals: **18**

Total Supply: **2545**

Balances:

Account	Balance
0x7425535ece072919c4455f8bb2ded0e8d2cbc873f9c48ad289f626d10ae51a77	430
0x2e0f487cf8418bd89edc330586f45c1b56c8ab93305b212c0c5852c8d3ed2402	1004
0x80b92e8c46670db9b72715cf6dbffc5d3c45229b9b8882038d81102e59d6161f	1111

In order to display this section we write the following code:

```
{ftData && (  
  <div className="border border-black p-3 rounded mt-2">  
    <h4 className="mb-2">Fungible Token Contract Data</h4>  
    <div>  
      <p>  
        Token Name: <b>{ftData.name}</b>  
      </p>  
      <p>  
        Symbol: <b>{ftData.symbol}</b>  
      </p>  
      <p>  
        Decimals: <b>{ftData.decimals}</b>  
      </p>  
      <p>  
        Total Supply: <b>{ftData.totalSupply}</b>  
      </p>  
      <div>  
        <p>Balances:</p>  
        <table class="table">  
          <thead>  
            <tr>  
              <th scope="col">Account</th>  
              <th scope="col">Balance</th>  
            </tr>  
          </thead>  
          <tbody>  
            {ftData.balances.map((data) => {  
              return (  
                <tr>  
                  <td>{data[0]}</td>
```

```

        <td>{data[1]}</td>
    </tr>
    );
    }}}
</tbody>
</table>
</div>
</div>
</div>
) }

```

As can be seen above, this section will not be displayed in the interface until the information corresponding to the state of the VARA Network contract is stored, ensuring that everything necessary is available to be shown to the user.

With this, we can interact with a Fungible Tokens contract within the VARA Network through a component deployed on the Blockchain Operating System. If you wish to see the code in more detail, you can view it at the following link:

[Component Code for Basic Interaction of a FT Contract.](#)

And with this we say goodbye, hoping that this tutorial has helped you understand how to correctly implement a VARA Network contract within your BOS components, until next time.