Documentation for the creation of components in VARA Network + Blockchain Operating System

What is Blockchain operating system?

Blockchain Operating System (BOS) is a blockchain operating system developed by the NEAR protocol that provides a development environment for creating secure, scalable, and user-friendly decentralized applications (dApps). This system stands out as a pioneering category in the industry and serves as a shared layer that facilitates the exploration and discovery of open web experiences, being compatible with any blockchain.

BOS is a real alternative to centralized platform systems and is available at near.org. It is a universally accessible system, available to both those familiar with web3 and those who are not. It facilitates navigation for both users and developers, simplifying the experience in both the web3 and web2 worlds.

BOS eliminates the need to choose between decentralization and visibility. Both developers and users can enjoy the best of both worlds, whether they are exploring web3 experiences for the first time or building an open web.

To accelerate the development of web3 applications, BOS provides a complete set of tools and capabilities that allow developers to get started quickly. These tools enable the rapid creation of open web applications, seamlessly onboard users, receive feedback, and enhance discoverability.

BOS Features

1. **Modularity:** BOS is divided into a series of interconnected modules that can be reused and combined to create different applications.
2. **Security:** BOS uses a layered security approach to protect dApps from external attacks.
3. **Scalability:** BOS is designed to be scalable and capable of handling a large number of simultaneous transactions.
4. **Flexibility:** BOS is compatible with different programming languages and frameworks, allowing developers to use their favorite tools to create dApps.
5. **Interoperability:** BOS enables interoperability between different blockchains, allowing dApps to interact with different protocols easily and seamlessly.
6. **Efficiency:** BOS uses an optimized data structure and an efficient consensus model to ensure high performance and fast response times.

Pillars of BOS

Components / Widgets

Components or Widgets are user interfaces developed to address specific problems. These frontend applications are designed by developers, and their source code is accessible through a gateway. In other words, they are modules designed to solve particular challenges within the BOS ecosystem.

Blockchains

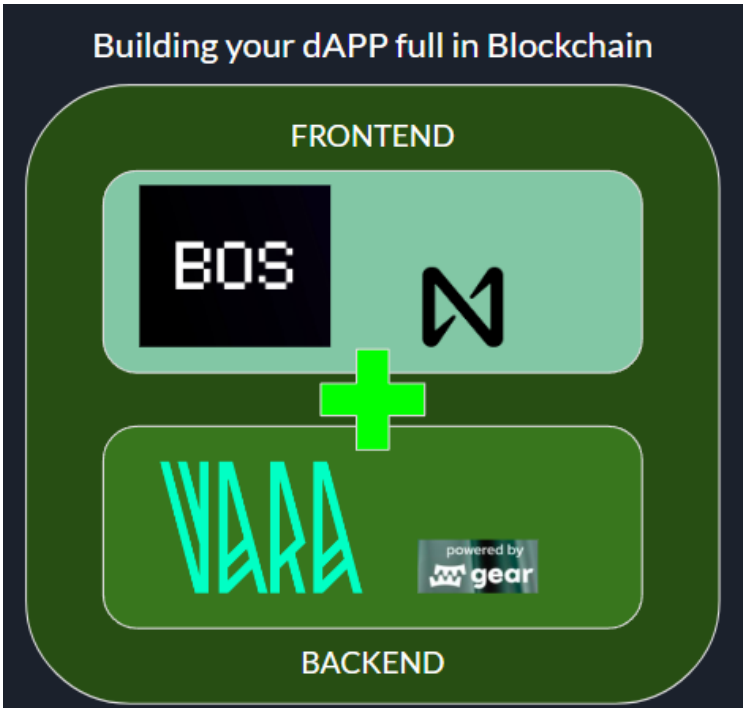
These are the digital platforms where components can invoke functions on any blockchain. Currently, BOS supports various blockchains, including all networks based on EVM (Ethereum Virtual Machine) and NEAR. The source code of the interfaces is stored on NEAR due to its efficient capability to store HTML, CSS, and JavaScript content.

Gateways

Gateways are designed to facilitate the availability of decentralized frontends that run locally for any user. In summary, they act as a simple means to render components in various contexts, ensuring that these applications can be displayed in multiple locations in an accessible manner.

VARA Network + BOS interaction

As can be seen in the following image, the application architecture will be as follows: the frontend will be deployed on the Blockchain Operating System on the NEAR Protocol network, while the backend will run entirely on the VARA Network. This ensures that your applications are executed in a 100% decentralized manner on the blockchain.



How do I deploy my component?

To deploy a component within the Blockchain Operating System, you need NEAR tokens and an account on the NEAR Protocol, either on the Mainnet or Testnet variant. When we want to save the code, it will be necessary to cover the storage costs within the network, and this must be paid with NEAR.

This gateway has two versions: a Mainnet version and a Testnet version where you can test your components, and once they are ready, you can deploy them on the Mainnet version. The links are as follows:

- **Mainnet:** https://vara.ow.academy/
- **Testnet:** To be defined

Creation of Components

For creating components that can interact with the VARA Network within the Blockchain Operating System, we base the style on React programming, using JavaScript, HTML5, and CSS with the possibility of using Bootstrap 5 for easier style management within these components.

To interact with the VARA Network, it is necessary to use the generated elements and meet the characteristics to ensure their correct functioning.

Available Elements

The elements that allow interaction with VARA have a syntax that must be followed to be called, and they have different characteristics for the internal functions they perform. These elements must be called as a React component as shown below:

```
<VaraNetwork.”Variant to call”/>
```

Variants

Different types of variants can be found that perform various actions within the BOS component code. These variants are as follows:

- VaraNetwork.Wrapper
- VaraNetwork.Identicon
- VaraNetwork.Interaction
- New elements under development…

VaraNetwork.Wrapper

This element allows encapsulating the entire component being developed, verifying that the user has connected their wallet with which they want to interact with VARA Network. This

ensures that there is a wallet available to interact with the network and sign transactions correctly. If a selected wallet is found, this element will allow displaying the content enclosed within it, following the normal component process.

Syntax

To use this element, it must be called as follows:

```
<VaraNetwork.Wrapper>
      "Your code"
</VaraNetwork.Wrapper>
```

VaraNetwork.Identicon

This element allows displaying the icon corresponding to the selected wallet within the gateway. This icon is unique and generated based on the user's wallet address. You can specify the size of the icon in pixels. This Identicon is the same as those shown in Polkadot applications and wallets.

Syntax

To implement this element, the code should be written as follows:

```
<VaraNetwork.Identicon size={30} />
```

VaraNetwork.Interaction

With this element, the user can interact with the VARA Network through various functions. Depending on the call made by this element, it can read or write information from a contract deployed on VARA, and use this information within the component deployed on BOS. This element contains several functions that can be called using the **trigger** property. This property allows the user to add HTML code that uses these functions, activating their operation. The available functions are as follows:

- readState
- signTransaction
- getAccountInfo

readState

The ***readState*** function allows fetching the state of any contract deployed on VARA Network. It returns a promise which, if successful, provides the requested information. Any errors encountered will be shown in the console. **readState** takes the following parameters:

- programId (Contract address)
- metadata (Contract metadata)
- params (Parameters sent to perform the Read State operation)

signTransaction

The **signTransaction** function enables the user to generate and send a transaction to a contract deployed on VARA Network. Once the transaction signature is accepted, various alerts are shown to the user. If everything goes correctly, the user will see two alerts: one with the transaction ID currently executing, and another when the transaction is completed. If any issues arise, an alert will inform the user about the error encountered during execution. **signTransaction** takes the following parameters:

- programId (Contract address)
- metadata (Contract metadata)
- params (Parameters sent to execute the Sign Transaction)
- gas (Gas sent to execute the transaction)
- value (Amount of VARA tokens to send in the transaction)

getAccountInfo

This function allows retrieving information about the user's active wallet. It can send the address and display this information in the component interface. If there is a change in the wallet, it's essential to re-run this method to retrieve the updated information. This function does not take any parameters; it simply needs to be called.

Syntax

To implement this element, the code should be written in the editor as follows:

```
<VaraNetwork.Interaction
            trigger={({ readState, signTransaction, getAccountInfo })
=> (
              <>
                <button
                  onClick={() => {
                    let varaAccount = getAccountInfo();
                    let info = readState(contract, contractData,
params);
                    signTransaction(contract, contractData, params,
gas, value)
                    info.then((res) => {
                       //Pulling the data from the contract
                       console.log(res)
                    });
                  }}
                >
                 “Button text”
                </button>
              </>
            )}
          />
```