BOS Components Design Patterns - Module 1

Leandro Manzanal

manzanal.near.social/

@leomanzanal

calendly.com/manzanal/30min

Why a component design pattern mindset could be important?

Modularity and Reusability

Widgets are self-contained, independent components that can be reused among the same app or even in other projects.
Enable **composability**, that refers to the ability of components to to be easily combined or composed together to build larger, more complex apps.

Maintainability and Readability

Components encapsulate functionality, blockchain interaction. Changes to a specific component won't necessarily affect other components, reducing the risk of introducing bugs or side effects. Clear and comprehensive documentation for each component is a must.

Collaboration

Different teams or devs work on different components, they can operate independently

Discoverability

The ease with which developers can find and understand the available components and features.
Examples:

- Near Social widget search/explorer
- Common Components Library
  https://github.com/leomanza/bos-common-components-library
  https://near.org/near/widget/PostPage?accountId=manzanal.near&blockHeight=85990251

Chain Agnostic Design on Blockchain Operation System

Why a chain agnostic design is important?

Compatible and functional across different blockchains

Chain-agnostic components are designed to be compatible and functional across various blockchains, providing flexibility and interoperability. These components can be designed with a more generic structure that can adapt to different blockchain environments.

Easily onboard developers from various ecosystems

Chain-agnostic components could allow builders from different ecosystems to have an easy onboarding process by using ready-made components, only worrying about using libraries that interact with the blockchain they already know.

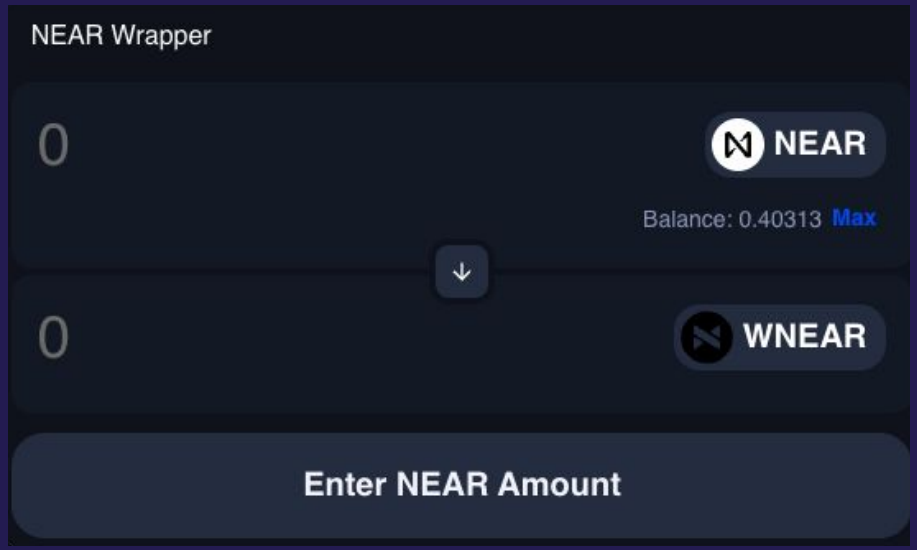
Enable discoverability of common chain-agnostic components across different ecosystems

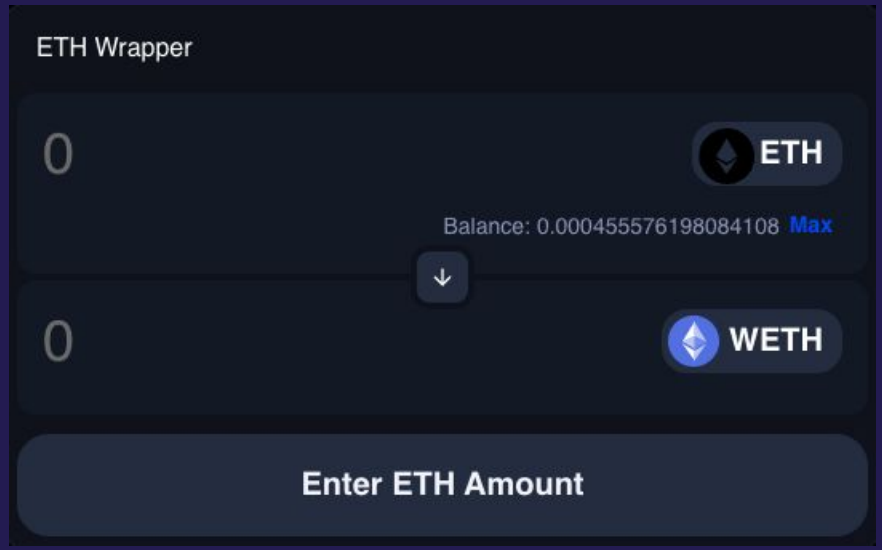
Chain-agnostic components could enable developers to search, discover, and share on-chain code, encouraging adoption of a composability mindset.

Smooth and streamlining process for multi-chain apps implementation

Chain-agnostic widgets examples

Token Wrapper App





All the examples could be found here: https://github.com/leomanza/chain-agnostic-bos-workshop

v0 - single widget approach

Single widget containing app functionality and blockchain interaction

In this example we can see a single widget for the entire app. We can see that challenging for:

- lack of modularity: organize and manage the codebase. As the app grows, it becomes difficult to find and understand specific functionalities
- difficult collaboration: multiple devs working on the same app simultaneously, having a single widget can create conflicts.It becomes harder to merge changes and increases the risk of overwriting each other's code.
- scalability: as the app grows in complexity and features, a single file becomes unmanageable
- readability and maintainability: harder to read and understand
- reusability limitations: inhibit reuse of smaller components or functionality in other projects

Demo v0

v1 - abstract widget approach

Widget containing app functionality and abstract blockchain interaction functions

In this example we can see two separate widgets for the specific token wrapper implementations.

The app is composed by:

1. an abstract wrapper implementation - a chain-agnostic widget with all the common logic and functionality.
2. a ETH/WETH wrapper implementation widget
3. a NEAR/WNEAR wrapper implementation widget

With this approach, we have made significant improvements in modularity and reusability, as both implementations use the same codebase for the app functionality, leaving the blockchain interaction to the specific implementations.

Demo v1

v2 - multi-chain widget approach

Widget containing app functionality and managing blockchain interaction based on the connected chain

As a result of the previous approach, two separate apps will exist, one for each chain.

While this outcome may be expected, let's explore the possibility of having a single entry point for a multi-chain app. Let's do it!

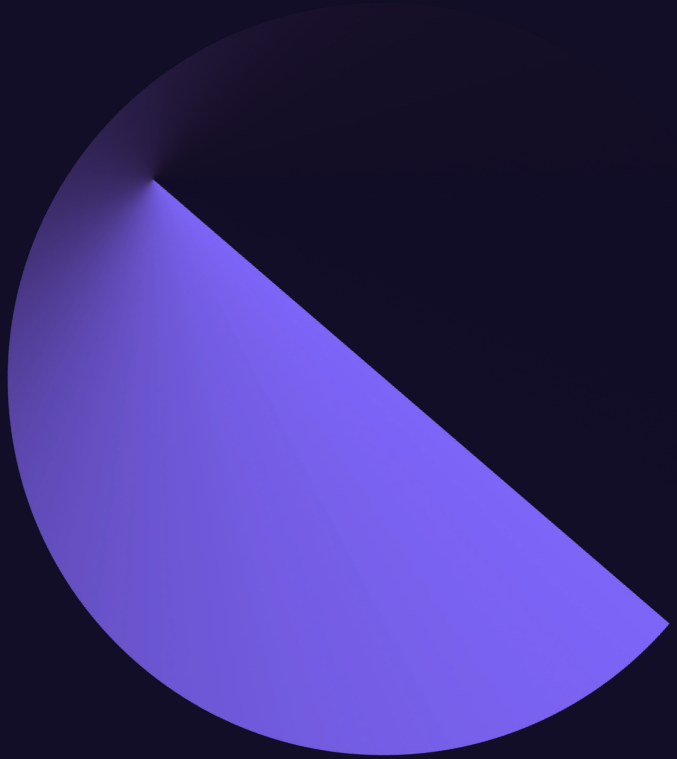
The app will be composed by:

1. an abstract wrapper implementation. Chain-agnostic widget with all the common logic and functionality.
2. a ETH/WETH wrapper implementation widget
3. a NEAR/WNEAR wrapper implementation widget
4. **a multi-chain token wrapper implementation widget**

Demo v2

Extra v3 - challenge: add a new wrapper implementation

Challenge: Extend the version 2 (v2) of the Token Wrapper to enable a new chain.



Thanks.