Coreum

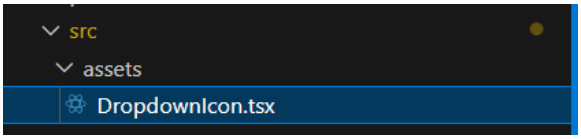
Coreum Bootstrap web App development

What You Will Learn

- Setting up a full-stack web development environment tailored for Coreum.
- Effective interaction and signing of Coreum custom messages using web-based wallets.
- Step-by-step guidance on creating a decentralized application from scratch.
- Hands-on experience with Smart Tokens in the browser, including creating NFT collections, minting NFTs, and sending NFTs.

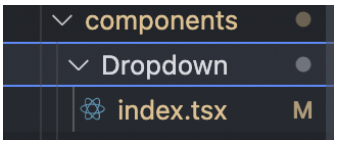
Application Process

Assets folder



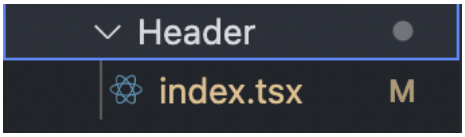
File DropdownIcon.tsx:
The DropdownIcon component dynamically renders different SVG icons based on the type prop passed to it. It supports rendering icons for actions like 'copy', 'disconnect', 'switch', and 'explorer', each represented by its own SVG markup. The className prop allows for custom styling to be applied to the SVG element. If the type does not match any of the cases, the function returns null, meaning no icon is rendered. This component is a flexible way to include various icons in a dropdown menu or any other part of a UI, where the icon needed can change based on the user's actions or choices.

Components



Dropdown folder(index.tsx):

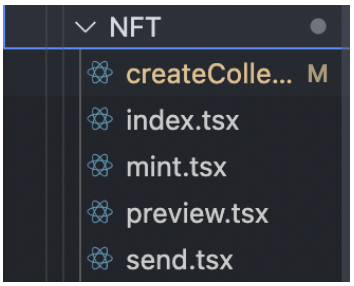
The DropdownIcon component dynamically renders different SVG icons based on the type prop passed to it. It supports rendering icons for actions like 'copy', 'disconnect', 'switch', and 'explorer', each represented by its own SVG markup. The className prop allows for custom styling to be applied to the SVG element. If the type does not match any of the cases, the function returns null, meaning no icon is rendered. This component is a flexible way to include various icons in a dropdown menu or any other part of a UI, where the icon needed can change based on the user's actions or choices.



Header Component(index.tsx):

Blockchain interaction within a web application. It utilizes the **useChain** hook to manage blockchain-related context, such as checking if a wallet is connected and handling wallet addresses. Key functionalities include:

- Connecting to a Wallet: It provides a UI element for users to connect their blockchain wallet to the application. If the wallet is not connected, clicking this element triggers the wallet connection process.
- Copying Wallet Address: The component includes functionality to copy the user's wallet address to the clipboard, enhancing user experience by simplifying how users share or save their address.
- Disconnecting the Wallet: Users can disconnect their wallet directly from the header, giving them control over their connection status.
- Dropdown Menu: For connected wallets, the header displays the wallet address and a dropdown menu with options to copy the address, disconnect the wallet, or view the address on a blockchain explorer. This menu is dynamically populated based on the wallet's connection status.
- Visual Design and Navigation: The header also includes a logo, serving as a navigational link back to the homepage, and employs a responsive design to adjust its layout across different screen sizes.



Coreum

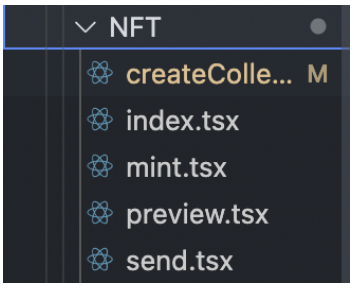
NFT Folder:

Preview.tsx Component:

Your code seems to be a React component that fetches NFTs (Non-Fungible Tokens) owned by a specific wallet address from a given endpoint. Here's a brief explanation of what each part does:

- import { chainName } from "@/config/defaults";: This imports chainName from a file named defaults located in the config directory.
- import { useChain } from "@cosmos-kit/react";: This imports the useChain hook from a package named @cosmos-kit/react.
- import { useEffect, useState } from "react";: This imports the useEffect and useState hooks from the React library.
- export default function Preview() { ... }: This declares a functional component named Preview.
- const chainContext = useChain(chainName);: This uses the useChain hook to get the chain context based on the chainName.
- const walletAddress = chainContext.address;: This extracts the wallet address from the chain context.
- const [nfts, setNfts] = useState<any[]>([]);: This initializes a state variable named nfts as an empty array, using the useState hook. setNfts is the function used to update this state.
- async function getNFTs() { ... }: This is an asynchronous function named getNFTs responsible for fetching NFTs owned by the wallet address.
- Inside getNFTs(), it fetches NFTs data from a specific endpoint using fetch and updates the state variable nfts with the fetched data.
- useEffect(() => { ... }, []);: This useEffect hook runs once when the component mounts. It calls the getNFTs function to fetch NFTs data.

Overall, this component fetches NFTs data associated with a specific wallet address when it mounts and stores it in the component's state.



Mint.tsx Component:

This React component is responsible for minting NFTs (Non-Fungible Tokens). Here's a breakdown of the code:

Imports:

- chainName from the defaults file in the config directory.
- useChain hook from @cosmos-kit/react.
- StdFee from @cosmjs/amino.
- NFT and convertStringToAny from coreum-js.
- useContext and useState from React.
- CoreumSigner from the CoreumSigner context.

Component State:

- classId, nftId, uri, and data are state variables for capturing input values.
- response and error are state variables for handling responses and errors from the minting process.

Chain Context:

- chainContext fetches the chain context using useChain hook, which contains the current wallet address.

Fee:

- fee specifies the fee structure for the transaction.

Coreum Signer:

- coreumSigner is obtained from the CoreumSigner context using useContext.

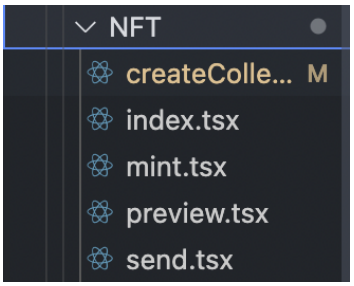
mintNFT Function:

- This function is responsible for minting the NFT.
- It constructs a mint message using NFT.Mint.
- It then signs and broadcasts the transaction using the coreumSigner.
- response and error states are updated based on the result.

Return:

- Renders a form for inputting NFT details.
- Displays a "Mint" button to trigger the minting process.
- Shows response and error messages.

Overall, this component provides a user interface for minting NFTs by capturing necessary details from the user, interacting with the Coreum blockchain, and displaying relevant feedback.



Create Collection.tsx Component:

This React component is responsible for creating a collection of NFTs (Non-Fungible Tokens). Here's a breakdown of the code:

Imports:

- Switch from @headlessui/react for toggle switches.
- useChain from @cosmos-kit/react for accessing blockchain data.
- chainName from the defaults file in the config directory.
- NFT and StdFee from respective packages.
- useContext and useState from React.
- Disclosure from @headlessui/react for displaying additional information.
- ChevronUpIcon from @heroicons/react for toggle icons.
- CoreumSigner from the CoreumSigner context.

Component State:

- State variables for toggles (isBurningEnabled, isFreezingEnabled, etc.).
- State variables for input fields (name, symbol, etc.).
- State variables for response and error handling (response, error).

Helper Functions:

- createStateArray: A function to create an array representing the enabled features based on toggle states.

Chain Context:

- chainContext retrieves the chain context using useChain, including the current wallet address.

Fee:

- fee specifies the fee structure for the transaction.

NFT Collection:

- newNFTCollection defines a new NFT collection object with various properties, including features enabled by toggles.

Coreum Signer:

- coreumSigner is obtained from the CoreumSigner context using useContext.

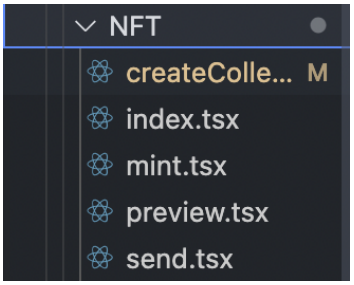
createCollection Function:

- This function signs and broadcasts the creation of the NFT collection.
- It updates response and error states based on the result.

UI Rendering:

- Renders a form for inputting collection details.
- Displays toggle switches for enabling/disabling features.
- Provides a "Deploy" button to trigger the collection creation process.
- Shows response and error messages.

Overall, this component offers a user interface for creating NFT collections by capturing necessary details, defining features, interacting with the Coreum blockchain, and providing feedback to the user.



Send.tsx component:

This React component is designed for sending a Smart Token NFT (Non-Fungible Token) to another user. Here's an overview of the code:

Imports:

- chainName from the defaults file in the config directory.
- useChain from @cosmos-kit/react for accessing blockchain data.
- StdFee from @cosmjs/amino for defining transaction fees.
- NFT from coreum-js for handling NFT-related operations.
- useContext and useState from React.
- CoreumSigner from the CoreumSigner context for signing transactions.

Component State:

- response and error states for handling responses and errors from transaction attempts.
- State variables for capturing input values: receiver, classId, and nftId.

Chain Context:

- chainContext retrieves the chain context using useChain, including the current wallet address.

Fee:

- fee specifies the fee structure for the transaction.

Coreum Signer:

- coreumSigner is obtained from the CoreumSigner context using useContext.

sendNFT Function:

- This function constructs a message to send the NFT to the specified receiver.
- It then signs and broadcasts the transaction using coreumSigner.
- response and error states are updated based on the result.

UI Rendering:

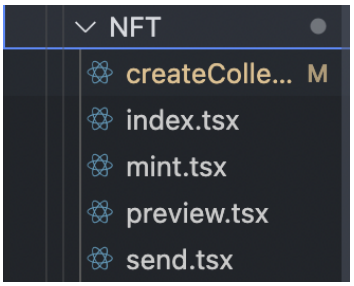
- Renders a form for inputting the class ID, NFT ID, and receiver address.
- Provides a "Send" button to trigger the NFT sending process.
- Displays response and error messages.

Overall, this component offers a user interface for sending Smart Token NFTs to other users on the Coreum network, handling input validation, transaction signing, and feedback display.

Registry: A registry in this context is a data structure that maps message types to their respective handler classes or types. It is used to ensure that when a message of a certain type is received, the correct code is executed to process that message. This is crucial for the modular and extensible design of blockchain applications, where different types of transactions need to be handled efficiently.
coreumRegistryTypes Array: This array contains tuples, where each tuple represents a message type and its associated handler. In this case, the array has one entry:

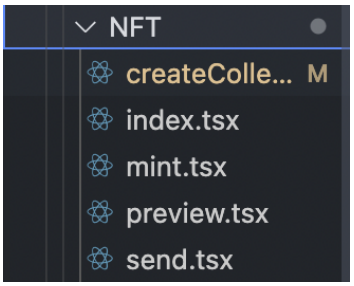
- "/coreum.nft.v1beta1.MsgSend": This is a string that uniquely identifies the message type. It follows a naming convention that typically includes the blockchain name (coreum), the module or functionality it pertains to (nft for non-fungible tokens), the version of the API (v1beta1 indicating version 1, beta 1), and the specific action or message (MsgSend indicating a message to send an NFT).
- MsgSend: This is the class or type that handles the logic for the MsgSend message. Whenever a message of type "/coreum.nft.v1beta1.MsgSend" is received, an instance of MsgSend will be used to process that message.

new Registry(coreumRegistryTypes): This line creates a new instance of a Registry, passing the coreumRegistryTypes array as an argument. This effectively registers the MsgSend message type and its handler in the registry. Once registered, the blockchain application knows how to process MsgSend messages, delegating the task to the MsgSend handler.



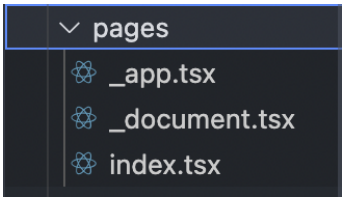
UTILS Folder(Bar.tsx):

The Bar component is a responsive navigation bar designed for switching between tabs in a web application. It supports two views based on screen size: a dropdown menu for smaller screens (sm:hidden) and a horizontal tab list for larger screens (hidden sm:block). The component dynamically highlights the current active tab and allows users to change the active tab by clicking on a tab name or selecting from the dropdown. The classNames function is used to conditionally apply CSS classes for styling based on whether a tab is currently active, providing visual feedback to the user. This setup enhances the user interface by making it adaptable to various device sizes while maintaining functionality and aesthetics.



Pages Folder(_app.tsx):
The App component is the root component for a Next.js application that integrates blockchain functionality through the @cosmos-kit/react library. It sets up a ChainProvider to manage blockchain connections, incorporating multiple wallets from different providers (keplrWallets, leapWallets, frontier, cosmostationWallets) to offer a variety of options for users. The provider is configured with chains and assets information from the chain-registry, ensuring compatibility with a wide range of blockchain networks. Additionally, it specifies walletConnectOptions for further customization and integration with WalletConnect, including project-specific metadata. The component also includes standard HTML head elements for web app metadata, such as viewport settings, character encoding, and the site's favicon, and dynamically renders pages and their props through the Component and pageProps

provided by Next.js. This setup forms a comprehensive foundation for building blockchain-connected web applications in the Cosmos ecosystem.



Pages Folder(index.tsx):

The Home function defines the main page of a web application focused on NFT (Non-Fungible Token) operations such as creating collections, minting, previewing, and sending NFTs. It utilizes React's useState hook for state management, dynamically rendering content based on the user's interaction with tab selections. The structure includes:

- Google Fonts Integration: Utilizes the Noto_Sans font with specified weights for typography, enhancing the visual presentation.
- Dynamic Tab Navigation: Implemented through the Bar component, which controls which tab is active based on user interaction. The tabs change the displayed content without reloading the page, creating a seamless user experience.
- Conditional Rendering: Based on the currently selected tab (curentTab), different components (CreateCollection, Mint, Preview, Send) are rendered. This showcases the application's functionality in creating and managing NFTs.
- Layout and Styling: The application features a responsive design, with a header and footer for navigation and informational purposes. The main content area uses a background image and CSS classes for styling, indicating a modern, visually appealing design approach.
- Interactivity: The application includes interactive elements, such as clickable tabs in the navigation bar and links within the text, providing users with a dynamic way to explore the app's features.