**React Utils AI**

**Calculating Total Airdrop Amount Needed**

In our AirdropForm component, the handleSubmit function orchestrates the process of initiating an airdrop. After confirming our development setup (local Anvil node, deployed contracts, running frontend), we dive into the handleSubmit logic.

We've already fetched the amount of tokens our tsender contract is currently approved to spend on the user's behalf:

// Inside handleSubmit function of AirdropForm.tsx

async function handleSubmit() {

// ... other setup

const tsenterAddress = chainsToTsenter[chainId]["tsender"];

const approvedAmount = await getApprovedAmount(tsenterAddress);

console.log(approvedAmount); // Log the fetched approved amount

// ... rest of the function

}

The next crucial step is to compare this approvedAmount with the *total amount* the user actually intends to send in this specific airdrop batch. This comparison determines whether we need to prompt the user for a new approval transaction before proceeding with the airdrop itself.

// Inside handleSubmit function

​

// ... fetching approvedAmount ...

​

// Check if we have approved enough tokens

// We need to calculate 'totalAmountNeeded' first

if (approvedAmount < totalAmountNeeded) {

// Logic to request approval will go here

// ...

} else {

// Logic to proceed with the airdrop directly

// ...

}

This highlights the immediate need: how do we calculate totalAmountNeeded based on the user's input?

**Calculating the Total Amount Needed**

The total amount required for the airdrop depends directly on the values entered into the "Amount" input field. This field accepts a list of numbers, potentially separated by newlines or commas, where each number corresponds to an amount to be sent to a recipient.

For example:

* If the user enters 100, the total needed is 100.
* If the user enters:

100

100

or

100,100

The total needed is 200.

Therefore, we must parse the amounts input string, extract all valid numbers, and sum them up to determine the totalAmountNeeded.

**Optimizing Calculation with**useMemo

Calculating this total involves string manipulation and arithmetic. We want this calculation to happen whenever the user modifies the amounts input, but crucially, we *don't* want to recalculate it on every single component re-render, especially those triggered by changes in other inputs (like recipient addresses). Performing this calculation unnecessarily can impact performance.

React provides the useMemo hook precisely for this scenario. useMemo allows us to *memoize* (cache) the result of a function call. It will only re-execute the function and update the cached value if one of the dependencies specified in its dependency array has changed.

To implement this, we'll store the calculated total in a variable and wrap the calculation logic within useMemo, specifying the amounts state variable as its sole dependency.

// Import useMemo from React

import { useState, useMemo } from "react";

// ... other imports and component setup

​

export default function AirdropForm() {

// State for the amounts input field

const [amounts, setAmounts] = useState("");

// ... other state and hooks

​

// Calculate the total only when the 'amounts' string changes

const totalAmountNeeded: number = useMemo(() => {

// We'll define the calculation logic in a separate function

return calculateTotal(amounts);

}, [amounts]); // Dependency array: recalculate only if 'amounts' changes

​

// ... rest of the component including handleSubmit that uses totalAmountNeeded

}

With this setup, totalAmountNeeded will hold the correctly summed value, and the calculation will only run when the amounts string is modified by the user, ensuring efficiency.

**Creating the**calculateTotal**Utility Function**

The actual logic for parsing the amounts string and summing the numbers belongs in a dedicated function, which we'll call calculateTotal. Instead of defining this function directly within the AirdropForm component, we'll place it in a separate utility file. This approach offers several advantages:

1. **Reusability:** The function might be useful elsewhere in the application.
2. **Testability:** It's much easier to write isolated unit tests for a standalone utility function.
3. **Organization:** It keeps the AirdropForm component focused on its primary rendering and state management responsibilities, improving readability.

We'll create this function in src/utils/calculateTotal/calculateTotal.ts. Note the .ts extension; since this file contains only TypeScript logic and no React JSX, it doesn't need the .tsx extension.

**Implementing the**calculateTotal**Logic**

The calculateTotal function needs to take the raw amounts string as input and return a single number representing the sum of all valid amounts listed in the string. It should handle numbers separated by newlines or commas, ignore extra whitespace, and gracefully handle non-numeric or empty entries.

Here's the implementation of the calculateTotal function:

// src/utils/calculateTotal/calculateTotal.ts

​

export function calculateTotal(amounts: string): number {

// If the input string is empty or null, the total is 0

if (!amounts) {

return 0;

}

​

// 1. Split the string by one or more commas or newlines

const amountArray = amounts

.split(/[\n,]+/) // Regex: matches one or more newline or comma characters

// 2. Trim whitespace from each resulting string segment

.map(amt => amt.trim())

// 3. Filter out any empty strings that might result from splitting

.filter(amt => amt !== '')

// 4. Convert each valid string segment to a number using parseFloat

.map(amt => parseFloat(amt));

​

// 5. Filter out any results that are not valid numbers (NaN)

// 6. Sum the remaining valid numbers using reduce

return amountArray

.filter(num => !isNaN(num)) // Keep only valid numbers

.reduce((sum, num) => sum + num, 0); // Sum them, starting from 0

}

**Explanation:**

1. **Handle Empty Input:** An initial check returns 0 if the input string is empty or null.
2. **Split:** .split(/[\n,]+/) uses a regular expression to split the input string wherever one or more (+) newline (\n) or comma (,) characters occur. This handles various user input formats robustly.
3. **Trim:** .map(amt => amt.trim()) removes any leading or trailing whitespace from each piece obtained after splitting.
4. **Filter Empty:** .filter(amt => amt !== '') removes any empty strings that might arise (e.g., from consecutive delimiters or trailing newlines).
5. **Convert to Number:** .map(amt => parseFloat(amt)) attempts to convert each cleaned string segment into a floating-point number. parseFloat is suitable here as token amounts can have decimals. If a string cannot be converted (e.g., "abc"), it results in NaN (Not a Number).
6. **Filter NaN & Reduce:** Finally, .filter(num => !isNaN(num)) removes any NaN values produced during the conversion. .reduce((sum, num) => sum + num, 0) iterates through the remaining valid numbers, accumulating their sum, starting from an initial value of 0.

This utility function provides a reliable way to calculate the totalAmountNeeded from the user's input, which is then efficiently memoized using useMemo within our AirdropForm component, ready for comparison against the approvedAmount in the handleSubmit function.