Comprehensive Guide to Integrating Cardano Smart Contract with React.js

1. Introduction

Please read previous documentation and then read this documentation

2. Setting Up the React.js Project

To begin, let's set up a new React.js project using Create React App:

bash
```
npx create-react-app cardano-staking-app
cd cardano-staking-app
```

3. Installing Dependencies

Next, install necessary dependencies for interacting with Cardano smart contracts:

bash
```
npm install @emurgo/cardano-serialization-lib
```

4. Importing Smart Contract ABI

Copy the ABI (Application Binary Interface) of your Cardano smart contract into your React.js project. You can define it as a constant or import it from a separate file:

javascript
```
// ContractABI.js
export const contractABI = {
  // Paste your ABI here
};
```

5. Setting Up Smart Contract Integration

Create a new JavaScript file to handle smart contract integration:

javascript
```
// CardanoIntegration.js
import { contractABI } from './ContractABI';
import { Wallet } from '@emurgo/cardano-serialization-lib';

const wallet = Wallet.from_mnemonic("your mnemonic here", undefined,
"testnet");
```

```
const contractAddress = "your contract address here";

const contract = new Wallet(wasm).attach(contractAddress,
contractABI);

export const CardanoIntegration = {
  // Define functions for interacting with the smart contract here
};
```

6. Interacting with Smart Contract Functions

Implement functions to interact with the smart contract using its ABI:

javascript

```
// CardanoIntegration.js
export const CardanoIntegration = {
  stake: async (amount, lockPeriod) => {
    try {
      const tx = await contract.stake(amount, lockPeriod);
      console.log('Stake transaction:', tx);
      return tx;
    } catch (error) {
      console.error('Error staking:', error);
      throw error;
    }
  },

  withdraw: async () => {
    try {
      const tx = await contract.withdraw();
      console.log('Withdrawal transaction:', tx);
      return tx;
    } catch (error) {
      console.error('Error withdrawing:', error);
      throw error;
    }
  }
};
```

7. Integrating with React Components

Use the CardanoIntegration functions within your React components:

javascript
```
// App.js
import React, { useState } from 'react';
import { CardanoIntegration } from './CardanoIntegration';

function App() {
  const [stakeAmount, setStakeAmount] = useState('');
  const [lockPeriod, setLockPeriod] = useState('');

  const handleStake = async () => {
    try {
      const tx = await CardanoIntegration.stake(stakeAmount,
lockPeriod);
      console.log('Stake successful! Transaction:', tx);
      // Add logic to update UI or display success message
    } catch (error) {
      console.error('Error staking:', error);
      // Add logic to display error message to user
    }
  };

  const handleWithdraw = async () => {
    try {
      const tx = await CardanoIntegration.withdraw();
      console.log('Withdrawal successful! Transaction:', tx);
      // Add logic to update UI or display success message
    } catch (error) {
      console.error('Error withdrawing:', error);
      // Add logic to display error message to user
    }
  };

  return (
    <div className="App">
      <h1>Cardano Staking App</h1>
      <input type="text" placeholder="Stake Amount"
value={stakeAmount} onChange={(e) => setStakeAmount(e.target.value)}
/>
```

```
      <input type="text" placeholder="Lock Period"
value={lockPeriod} onChange={(e) => setLockPeriod(e.target.value)}
/>
      <button onClick={handleStake}>Stake</button>
      <button onClick={handleWithdraw}>Withdraw</button>
    </div>
  );
}

export default App;
```

8. Testing the Integration

Start your React.js development server and test the integration:

bash
```
npm start
```

.

Example for your staking smart contract and react.js

Implement Contract Interaction

Create a module for interacting with the Cardano smart contract, handling stake transitions.

Step 1: Define Stake Transition Component

Create a React component to handle stake transitions. This component will include UI elements for staking and withdrawing tokens.

Step 2: Call Contract Interaction Functions

Call the contract interaction functions from the React component to perform stake transitions.

Step 3: Handle Responses

Handle responses from contract interaction functions to update UI based on stake transitions.

Step 4: Add Error Handling

Implement error handling to catch and display any errors that occur during stake transitions.

Example Code

Here's an example of how you can integrate the `StakeTransition` data type into a React.js application:

javascript

```
// StakeTransitionComponent.js

import React, { useState } from 'react';

import { CardanoIntegration } from './CardanoIntegration';



function StakeTransitionComponent() {

  const [transition, setTransition] = useState(null);



  const handleStake = async () => {

    try {

      const tx = await CardanoIntegration.stake();

      console.log('Stake transaction:', tx);

      setTransition('Stake');

    } catch (error) {

      console.error('Error staking:', error);

    }

  };
  const handleWithdraw = async () => {

    try {

      const tx = await CardanoIntegration.withdraw();

      console.log('Withdrawal transaction:', tx);

      setTransition('Withdraw');

    } catch (error) {

      console.error('Error withdrawing:', error);

    }

  };


  return (

    <div>

      <h2>Stake Transition Component</h2>

      <button onClick={handleStake}>Stake Tokens</button>

      <button onClick={handleWithdraw}>Withdraw Tokens</button>

      {transition && <p>Transition: {transition}</p>}

    </div>

  );

}


export default StakeTransitionComponent;
```