Recap until now

1. Week 1 - Bitcoin Whitepaper, Decentralized currencies
2. Encryption, Decryption, hashing, ed25519, Public and Private keys
3. Creating a Web based wallet
4. Solana Jargon, Accounts, Tokens and The Token program on Solana

Building a web based wallet

We've not built an app yet. We've built a `wallet` but no one would use a `new` wallet.

People have wallets already, they use new `dapps`

Building a `dapp`

We haven't built a `dapp` yet.
Today's video, we'll try to build a `decentralized app` and understand about the `solana wallet adapter`

Pre-requisite

React (although we'll try to keep things easier/not use a lot of react constructs (like useState))

Code - https://github.com/100xdevs-cohort-3/week-5-web3

Creating a react app

- Initialize a react app

```
npm create vite@latest
```

- Clean up `App.jsx`

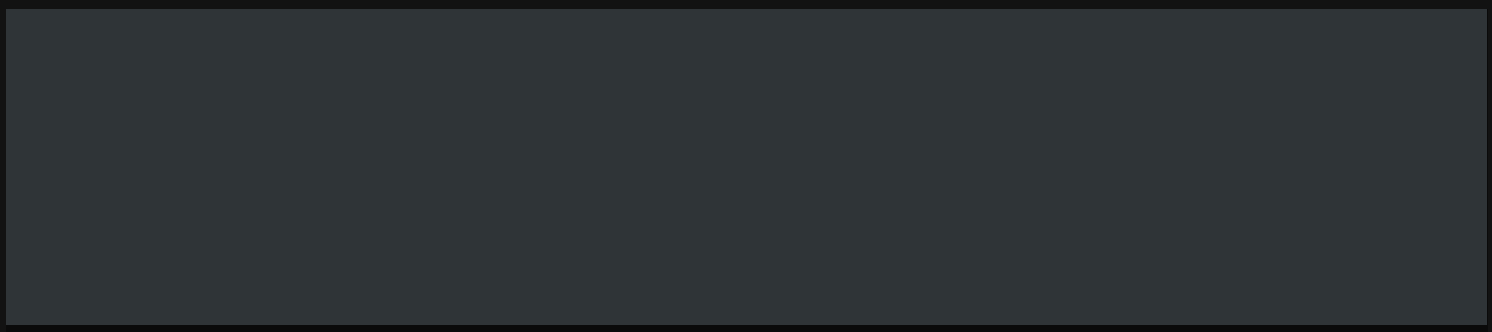
```
import './App.css'

function App() {

  return (
    <>
      <div>
        hi there
      </div>
    </>
  )
}

export default App
```

- Add dependencies

```
npm install @solana/wallet-adapter-base \
    @solana/wallet-adapter-react \
    @solana/wallet-adapter-react-ui \
    @solana/wallet-adapter-wallets \
    @solana/web3.js
```

- Clean up css files

```
index.css , App.css
```



Things to do

Once a user connects to your `dapp`, you usually ask the user to do a few things with their wallet balance -

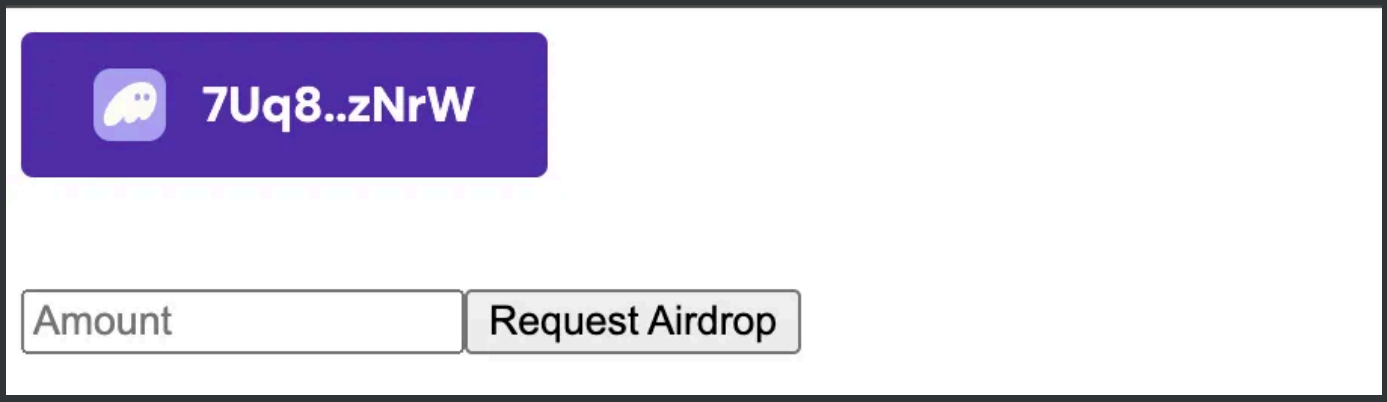
1. Request Airdrop
2. Show SOL balances (GET data from the blockchain)
3. Send a transaction (Send a transaction to the blockchain)
4. Sign a message (Verify the ownership of the wallet)

Requesting airdrop (Creating a faucet)

Create something like - https://solfaucet.com/

Hints

- `@solana/web3.js` provides you with a `requestAirdrop` function.
- You can get the `current users` public key using the `useWallet` hook




```
export function RequestAirdrop() {
    const wallet = useWallet();
    const { connection } = useConnection();

    async function requestAirdrop() {
        let amount = document.getElementById("amount").value;
        await connection.requestAirdrop(wallet.publicKey, amount * LAMPORTS_PI
        alert("Airdropped " + amount + " SOL to " + wallet.publicKey.toBase58());
    }

    return <div>
        <br/><br/>
        <input id="amount" type="text" placeholder="Amount" />
```

```
            <button onClick={requestAirdrop}>Request Airdrop</button>
        </div>
    }
```

Showing user balance

```
import { useConnection, useWallet } from "@solana/wallet-adapter-react";
import { LAMPORTS_PER_SOL } from "@solana/web3.js";

export function ShowSolBalance() {
    const { connection } = useConnection();
    const wallet = useWallet();

    async function getBalance() {
        if (wallet.publicKey) {

            const balance = await connection.getBalance(wallet.publicKey);
            document.getElementById("balance").innerHTML = balance / LAMPORTS_
        }
    }

    getBalance();
    return <div>
        <p>SOL Balance:</p> <div id="balance"></div>
    </div>
}
```

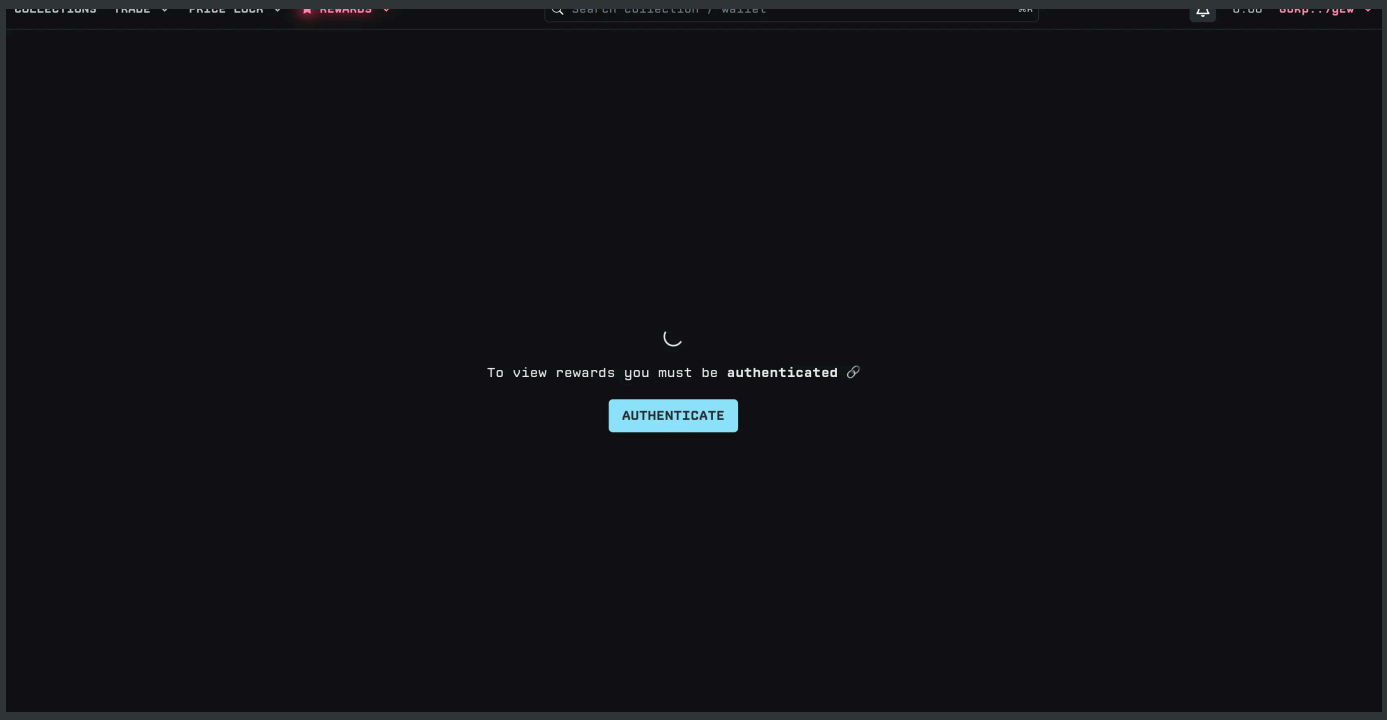
SOL Balance:

1

Sign a message

Usage

Prove ownership of a wallet to a centralised backend. For eg - https://www.tensor.trade/rewards



Ref - https://github.com/anza-xyz/wallet-adapter/blob/3761cd8cc867da39da7c0b070bbf8779402cff36/packages/starter/example/src/components/SignMessage.tsx#L9

How to sign messages

- Install `@noble/curves`

```
npm install @noble/curves
```

- Coding the `SignMessage` component

```
import { ed25519 } from '@noble/curves/ed25519';
import { useWallet } from '@solana/wallet-adapter-react';
import bs58 from 'bs58';
import React from 'react';

export function SignMessage() {
    const { publicKey, signMessage } = useWallet();

    async function onClick() {
        if (!publicKey) throw new Error('Wallet not connected!');
        if (!signMessage) throw new Error('Wallet does not support message signi

        const message = document.getElementById("message").value;
        const encodedMessage = new TextEncoder().encode(message);
        const signature = await signMessage(encodedMessage);

        if (!ed25519.verify(signature, encodedMessage, publicKey.toBytes())) throv
        alert('success', `Message signature: ${bs58.encode(signature)}`);
    };

    return (
        <div>
            <input id="message" type="text" placeholder="Message" />
            <button onClick={onClick}>
                Sign Message
            </button>
        </div>
    );
};
```

Sending Solana

In this section, we'll learn about transactions and sending them on the solana blockchain

We need to write code that lets users `send` native SOL over to a different solana address.

This should `require approval`

```
import { useConnection, useWallet } from "@solana/wallet-adapter-react"
import { LAMPORTS_PER_SOL, PublicKey, SystemProgram, Transaction} from "@


export function SendTokens() {
    const wallet = useWallet();
    const {connection} = useConnection();

    async function sendTokens() {
        let to = document.getElementById("to").value;
        let amount = document.getElementById("amount").value;
        const transaction = new Transaction();
        transaction.add(SystemProgram.transfer({
            fromPubkey: wallet.publicKey,
            toPubkey: new PublicKey(to),
            lamports: amount * LAMPORTS_PER_SOL,
        }));

        await wallet.sendTransaction(transaction, connection);
        alert("Sent " + amount + " SOL to " + to);
    }

    return <div>
        <input id="to" type="text" placeholder="To" />
        <input id="amount" type="text" placeholder="Amount" />
        <button onClick={sendTokens}>Send</button>
```

```
        </div>
    }
```

Assignment

1. Show user token balances
2. Let user transfer tokens

Challenges/Things to learn

1. Extract metadata and show the user the `ticker` and the `logo` of the token.
2. Use the `@solana/spl-token` library, make it work purely from the frontend.

Very hard assignment

Add payments to https://github.com/code100x/muzer