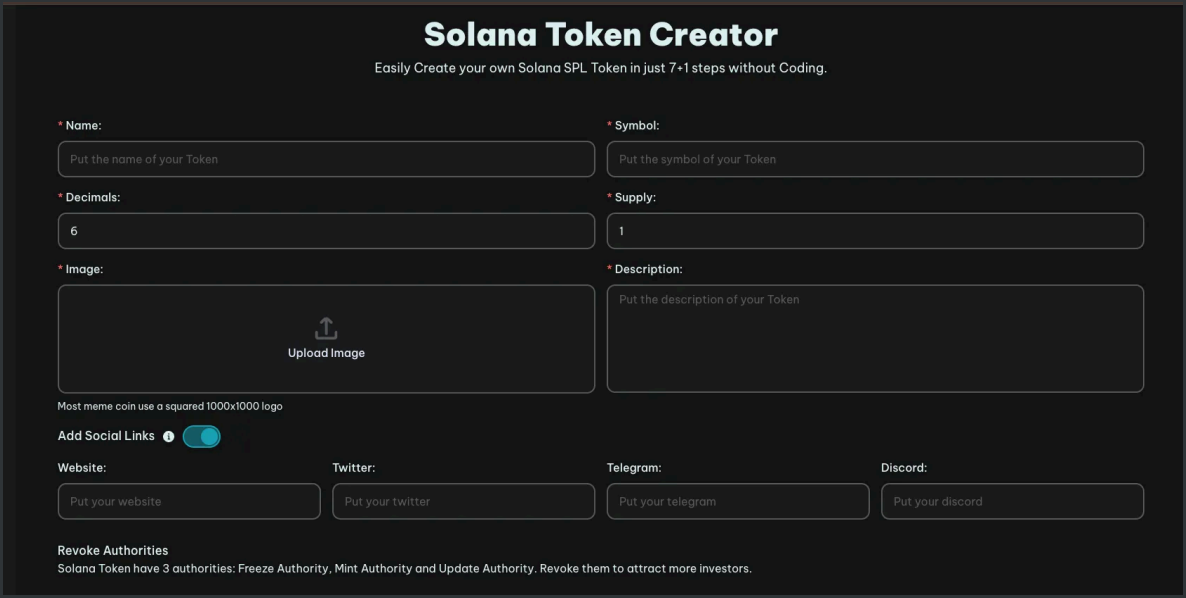
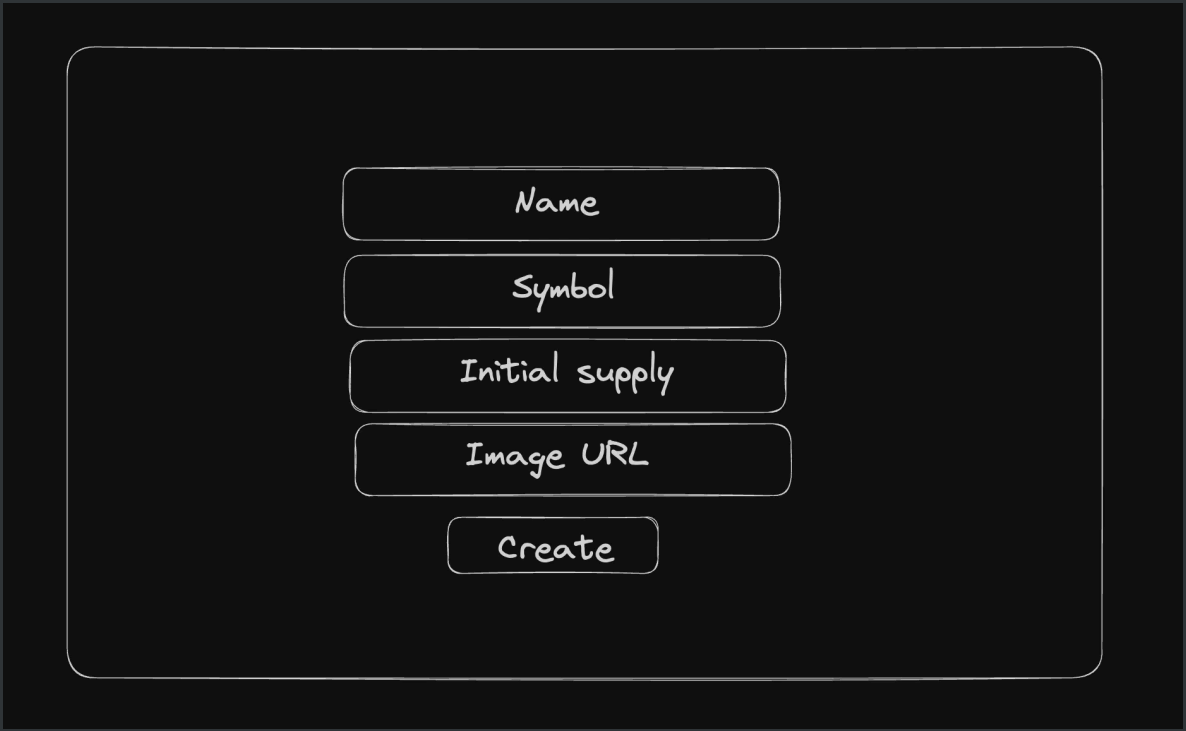
Context

Today we're building a token launchpad.

Very similar to - ▸



Our version



References

1. `@solana/spl-token-metadata` - https://github.com/solana-labs/solana-program-library/blob/master/token-metadata/js/package.json
2. `@solana/spl-token` - https://github.com/solana-labs/solana-program-library/tree/master/token/js
3. `@solana/web3.js` - https://github.com/solana-labs/solana-web3.js

Things to learn -

Reading through solana functions, creating transaction instructions and attaching them to the wallet adapter.

Transactions vs Instructions

Transactions vs instructions

When you send a `transaction` to the solana blockchain, you are actually sending a bunch of instructions (with a limit to the max number of instructions you can send)

Initializing the project

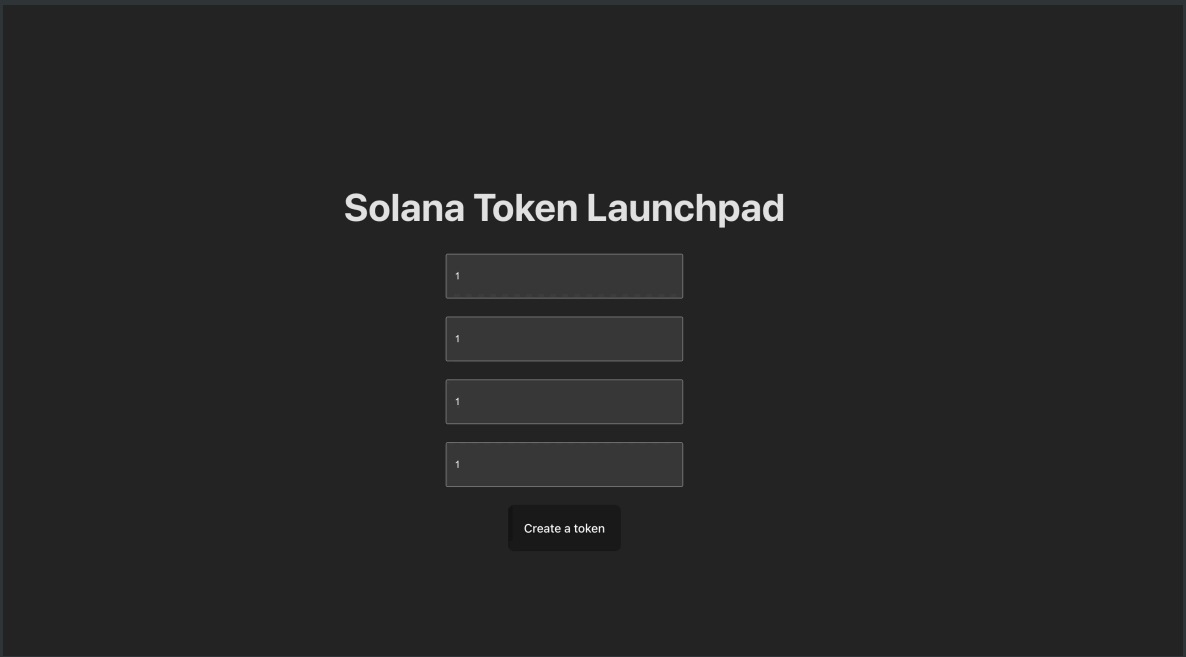
Starting the project locally

- Clone the starter repo - https://github.com/100xdevs-cohort-3/week-6-web3-token-launchpad
- Go to the `1-token-launchpad-starter` folder
- Install the dependencies

```
npm install
```

- Start the code locally

```
npm run dev
```



Add the solana wallet adapter

Ref - https://github.com/anza-xyz/wallet-adapter/blob/master/APP.md

- Install the wallet adapter dependencies

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

- Add the new set of imports to `App.jsx`

```
// wallet adapter imports
import { ConnectionProvider, WalletProvider } from '@solana/wallet-adapter-r
import {
    WalletModalProvider,
    WalletDisconnectButton,
    WalletMultiButton
} from '@solana/wallet-adapter-react-ui';
import '@solana/wallet-adapter-react-ui/styles.css';
```

- Create a topbar, wrap the `TokenLaunchpad` component inside the wallet adapter providers

```
function App() {
  return (
    <div>
      <ConnectionProvider endpoint={"https://api.devnet.solana.com"}>
        <WalletProvider wallets={[]} autoConnect>
          <WalletModalProvider>
            <div style={{
              display: 'flex',
              justifyContent: 'space-between',
              padding: 20
            }}>
              <WalletMultiButton />
              <WalletDisconnectButton />
            </div>
```

```
            <TokenLaunchpad></TokenLaunchpad>
          </WalletModalProvider>
        </WalletProvider>
      </ConnectionProvider>
    </div>
  )
}
```

Final code - https://github.com/100xdevs-cohort-3/week-6-web3-token-launchpad/blob/main/2-token-launchpad-with-adapter/src/App.jsx

Installing dependencies

Adding functionality to create tokens

- Install dependencies

```
npm install @solana/spl-token @solana/web3.js
```

- Add polyfills to ensure a few `node apis` are available in the browser

```
npm install --save-dev vite-plugin-node-polyfills
```

- Add the plugin to your `vite.config.ts` file.

```
import { defineConfig } from 'vite'
import { nodePolyfills } from 'vite-plugin-node-polyfills'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [
    nodePolyfills(),
  ],
})
```

- Add `onclick` handler in `TokenLaunchpad.jsx`

```
export function TokenLaunchpad() {

    function createToken() {
        const name = document.getElementById('name').value;
        const symbol = document.getElementById('symbol').value;
        const image = document.getElementById('image').value;
        const initialSupply = document.getElementById('initialSupply').value;
    }

    return  <div style={{
        height: '100vh',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        flexDirection: 'column'
    }}>
```

```
        <h1>Solana Token Launchpad</h1>
        <input className='inputText' type='text' placeholder='Name'></input> <br />
        <input className='inputText' type='text' placeholder='Symbol'></input> <br />
        <input className='inputText' type='text' placeholder='Image URL'></input> <br />
        <input className='inputText' type='text' placeholder='Initial Supply'></input> <br
        <button onClick={createToken} className='btn'>Create a token</button>
    </div>
}
```

Wallet adapter vs local wallet

💡 Do you think we should use the `createMint` function to create the token mint?
Like we did here - https://www.notion.so/Equivalent-code-in-JS-afed3cf599d64ee5bae4cc05a7b9f346 ?

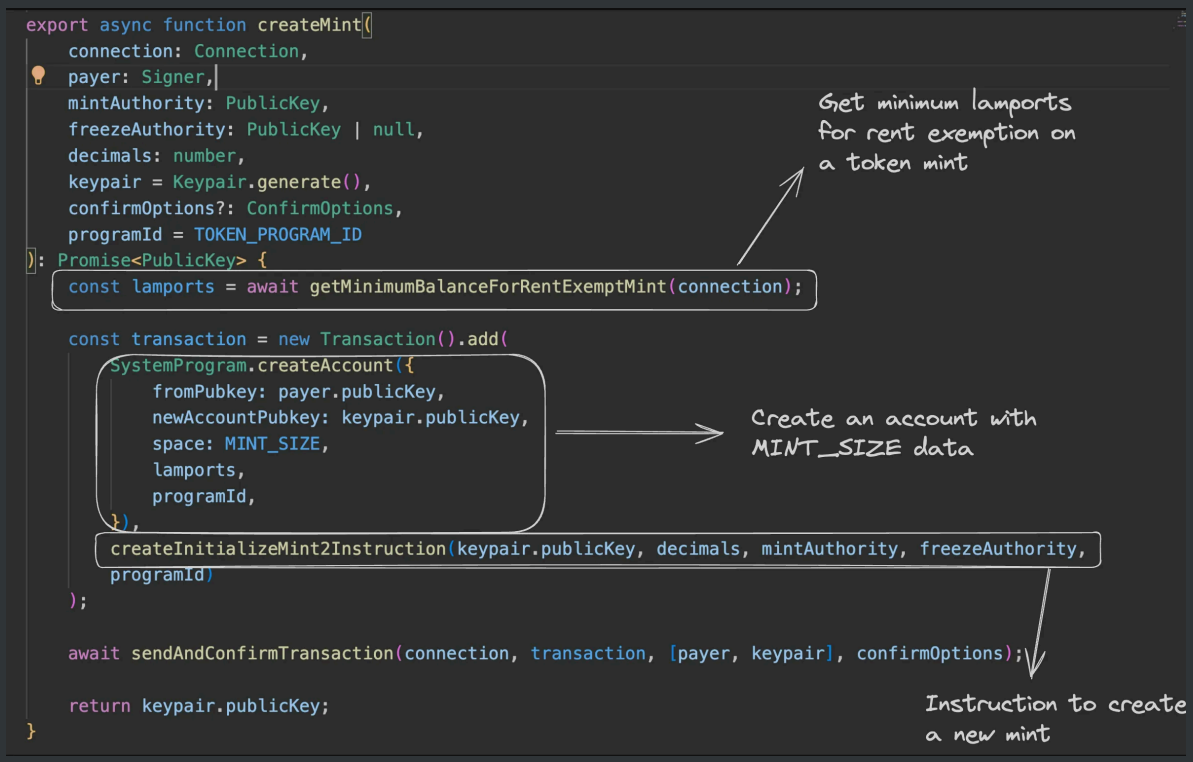
```
const { createMint } = require('@solana/spl-token');
const mint = await createMint(
    connection,
    payer,
    mintAuthority,
    null,
    6,
    TOKEN_PROGRAM_ID
);
```

Since we want an `end user` to create their own token, pay for gas for creating that token, we need to ask `THEIR WALLET` for approval to create a token. We `CANT` create our own `KeyPair` and create a token using it.

Inspecting the createMint call

The createMint function sends a transaction with 2 instructions

1. Create a fresh mint account
2. Initialise data inside the mint account such that it stores mint data (decimals, mintAuthority etc)



We need to bring in the same code in our codebase and make the user sign it

Adding Node polyfill

https://www.npmjs.com/package/vite-plugin-node-polyfills

- Install `vite-plugin-node-polyfills`

```
npm install --save-dev vite-plugin-node-polyfills
```

- Update `vite.config.js`

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import { nodePolyfills } from 'vite-plugin-node-polyfills'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react(), nodePolyfills(),],
})
```

Creating the mint function

```
import { Keypair, SystemProgram, Transaction } from "@solana/web3.js";
import { useConnection, useWallet } from '@solana/wallet-adapter-react';
import { MINT_SIZE, TOKEN_PROGRAM_ID, createInitializeMint2Instruction, creat

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

    async function createToken() {
        const mintKeypair = Keypair.generate();
        const lamports = await getMinimumBalanceForRentExemptMint(connectio

        const transaction = new Transaction().add(
            SystemProgram.createAccount({
                fromPubkey: wallet.publicKey,
                newAccountPubkey: mintKeypair.publicKey,
                space: MINT_SIZE,
                lamports,
                programId: TOKEN_PROGRAM_ID,
            }),
            createInitializeMint2Instruction(mintKeypair.publicKey, 9, wallet.publicKe
        );

        transaction.feePayer = wallet.publicKey;
        transaction.recentBlockhash = (await connection.getLatestBlockhash()).b
        transaction.partialSign(mintKeypair);

        await wallet.sendTransaction(transaction, connection);
        console.log(`Token mint created at ${mintKeypair.publicKey.toBase58()}`)
    }

    return  <div style={{
        height: '100vh',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        flexDirection: 'column'
    }}>
        <h1>Solana Token Launchpad</h1>
        <input className='inputText' type='text' placeholder='Name'></input> <br
        <input className='inputText' type='text' placeholder='Symbol'></input> <b
```

```
        <input className='inputText' type='text' placeholder='Image URL'></input>
        <input className='inputText' type='text' placeholder='Initial Supply'></inp
        <button onClick={createToken} className='btn'>Create a token</button>
      </div>
    }
```

Attaching metadata

Ref -

1. https://github.com/100xdevs-cohort-3/week-6-web3-token-launchpad/tree/main/4-token-launchpage-with-metadata
2. https://solana.com/developers/guides/token-extensions/metadata-pointer
3. Convert the mint to use `token 2022 program`
4. After creating the token, create the metadata

Ref - https://spl.solana.com/token-2022/extensions#example-create-a-mint-with-metadata

Using the token22 program, let's create the mint with some metadata

```
import { Keypair, SystemProgram, Transaction } from "@solana/web3.js";
import { useConnection, useWallet } from '@solana/wallet-adapter-react';
import { TOKEN_2022_PROGRAM_ID, getMintLen, createInitializeMetadataPointe
import { createInitializeInstruction, pack } from '@solana/spl-token-metadata'


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

    async function createToken() {
        const mintKeypair = Keypair.generate();
        const metadata = {
            mint: mintKeypair.publicKey,
            name: 'KIRA',
            symbol: 'KIR    ',
            uri: 'https://cdn.100xdevs.com/metadata.json',
            additionalMetadata: [],
        };

        const mintLen = getMintLen([ExtensionType.MetadataPointer]);
        const metadataLen = TYPE_SIZE + LENGTH_SIZE + pack(metadata).length;

        const lamports = await connection.getMinimumBalanceForRentExemptior
```

```
        const transaction = new Transaction().add(
          SystemProgram.createAccount({
            fromPubkey: wallet.publicKey,
            newAccountPubkey: mintKeypair.publicKey,
            space: mintLen,
            lamports,
            programId: TOKEN_2022_PROGRAM_ID,
          }),
          createInitializeMetadataPointerInstruction(mintKeypair.publicKey, wallet
          createInitializeMintInstruction(mintKeypair.publicKey, 9, wallet.publicKey
          createInitializeInstruction({
            programId: TOKEN_2022_PROGRAM_ID,
            mint: mintKeypair.publicKey,
            metadata: mintKeypair.publicKey,
            name: metadata.name,
            symbol: metadata.symbol,
            uri: metadata.uri,
            mintAuthority: wallet.publicKey,
            updateAuthority: wallet.publicKey,
          }),
        );

        transaction.feePayer = wallet.publicKey;
        transaction.recentBlockhash = (await connection.getLatestBlockhash()).b
        transaction.partialSign(mintKeypair);

        await wallet.sendTransaction(transaction, connection);
      }

      return <div style={{
        height: '100vh',
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center',
        flexDirection: 'column'
      }}>
        <h1>Solana Token Launchpad</h1>
        <input className='inputText' type='text' placeholder='Name'></input> <br
        <input className='inputText' type='text' placeholder='Symbol'></input> <b
        <input className='inputText' type='text' placeholder='Image URL'></input>
        <input className='inputText' type='text' placeholder='Initial Supply'></inpu
        <button onClick={createToken} className='btn'>Create a token</button>
      </div>
    }
```

Actually minting the tokens

Ref - https://github.com/100xdevs-cohort-3/week-6-web3-token-launchpad/tree/main/5-token-launchpage-with-metadata-and-mint

Finally, lets write the logic to actually mint the tokens

1. Find the associated token account
2. mint the tokens to the ata

```
 const associatedToken = getAssociatedTokenAddressSync(
   mintKeypair.publicKey,
   wallet.publicKey,
   false,
   TOKEN_2022_PROGRAM_ID,
);

console.log(associatedToken.toBase58());

const transaction2 = new Transaction().add(
   createAssociatedTokenAccountInstruction(
      wallet.publicKey,
      associatedToken,
      wallet.publicKey,
      mintKeypair.publicKey,
      TOKEN_2022_PROGRAM_ID,
   ),
);

await wallet.sendTransaction(transaction2, connection);

const transaction3 = new Transaction().add(
   createMintToInstruction(mintKeypair.publicKey, associatedToken, wallet.pub
);

await wallet.sendTransaction(transaction3, connection);
```