```
import React, { useEffect, useState } from "react";
import { Connection, PublicKey } from "@solana/web3.js";
import { Program, Provider, web3 } from "@project-serum/anchor";
import { MintLayout, TOKEN_PROGRAM_ID, Token } from "@solana/spl-
token";
import { sendTransactions } from "./connection";
import "./CandyMachine.css";
import {
  candyMachineProgram,
  TOKEN_METADATA_PROGRAM_ID,
  SPL_ASSOCIATED_TOKEN_ACCOUNT_PROGRAM_ID,
  getAtaForMint,
  getNetworkExpire,
  getNetworkToken,
  CIVIC,
} from "./helpers";

const { SystemProgram } = web3;
const opts = {
  preflightCommitment: "processed",
};

const CandyMachine = ({ walletAddress }) => {
  useEffect(() => {
    getCandyMachineState();
  }, []);

  const [candyMachine, setCandyMachine] = useState(null);

  const getProvider = () => {
    const rpcHost = process.env.REACT_APP_SOLANA_RPC_HOST;
    const connection = new Connection(rpcHost);
    const provider = new Provider(
      connection,
      window.solana,
      opts.preflightCommitment
    );
    return provider;
  };

  const getCandyMachineState = async () => {
    const provider = getProvider();
    const idl = await Program.fetchIdl(candyMachineProgram,
provider);
    const program = new Program(idl, candyMachineProgram, provider);
    const candyMachine = await program.account.candyMachine.fetch(
      process.env.REACT_APP_CANDY_MACHINE_ID
    );
    const itemsAvailable =
candyMachine.data.itemsAvailable.toNumber();
    const itemsRedeemed = candyMachine.itemsRedeemed.toNumber();
    const itemsRemaining = itemsAvailable - itemsRedeemed;
    const goLiveDate = candyMachine.data.goLiveDate.toNumber();
    const presale =
```

```
      candyMachine.data.whitelistMintSettings &&
      candyMachine.data.whitelistMintSettings.presale &&
      (!candyMachine.data.goLiveDate ||
        candyMachine.data.goLiveDate.toNumber() > new
Date().getTime / 1000);

    const goLiveDateTimeString = `${new Date(goLiveDate *
1000).toGMTString()}`;

    setCandyMachine({
      id: process.env.REACT_APP_CANDY_MACHINE_ID,
      program,
      state: {
        itemsAvailable,
        itemsRedeemed,
        itemsRemaining,
        goLiveDate,
        goLiveDateTimeString,
        isSoldOut: itemsRemaining === 0,
        isActive:
          (presale ||
            candyMachine.data.goLiveDate.toNumber() <
              new Date().getTime() / 1000) &&
          (candyMachine.endSettings
            ? candyMachine.endSettings.endSettingType.date
              ? candyMachine.endSettings.number.toNumber() >
                new Date().getTime() / 1000
              : itemsRedeemed <
candyMachine.endSettings.number.toNumber()
            : true),
        isPresale: presale,
        goLiveDate: candyMachine.data.goLiveDate,
        treasury: candyMachine.wallet,
        tokenMint: candyMachine.tokenMint,
        gatekeeper: candyMachine.data.gatekeeper,
        endSettings: candyMachine.data.endSettings,
        whitelistMintSettings:
candyMachine.data.whitelistMintSettings,
        hiddenSettings: candyMachine.data.hiddenSettings,
        price: candyMachine.data.price,
      },
    });

    console.log({
      itemsAvailable,
      itemsRedeemed,
      itemsRemaining,
      goLiveDate,
      goLiveDateTimeString,
    });
  };

  const getCandyMachineCreator = async (candyMachine) => {
    const candyMachineID = new PublicKey(candyMachine);
```

```
    return await web3.PublicKey.findProgramAddress(
      [Buffer.from("candy_machine"), candyMachineID.toBuffer()],
      candyMachineProgram
    );
  };

  const getMetadata = async (mint) => {
    return (
      await PublicKey.findProgramAddress(
        [
          Buffer.from("metadata"),
          TOKEN_METADATA_PROGRAM_ID.toBuffer(),
          mint.toBuffer(),
        ],
        TOKEN_METADATA_PROGRAM_ID
      )
    )[0];
  };

  const getMasterEdition = async (mint) => {
    return (
      await PublicKey.findProgramAddress(
        [
          Buffer.from("metadata"),
          TOKEN_METADATA_PROGRAM_ID.toBuffer(),
          mint.toBuffer(),
          Buffer.from("edition"),
        ],
        TOKEN_METADATA_PROGRAM_ID
      )
    )[0];
  };

  const createAssociatedTokenAccountInstruction = (
    associatedTokenAddress,
    payer,
    walletAddress,
    splTokenMintAddress
  ) => {
    const keys = [
      { pubkey: payer, isSigner: true, isWritable: true },
      { pubkey: associatedTokenAddress, isSigner: false, isWritable:
true },
      { pubkey: walletAddress, isSigner: false, isWritable: false },
      { pubkey: splTokenMintAddress, isSigner: false, isWritable:
false },
      {
        pubkey: web3.SystemProgram.programId,
        isSigner: false,
        isWritable: false,
      },
      { pubkey: TOKEN_PROGRAM_ID, isSigner: false, isWritable:
false },
      {
```

```
        pubkey: web3.SYSVAR_RENT_PUBKEY,
        isSigner: false,
        isWritable: false,
      },
    ];
    return new web3.TransactionInstruction({
      keys,
      programId: SPL_ASSOCIATED_TOKEN_ACCOUNT_PROGRAM_ID,
      data: Buffer.from([]),
    });
  };

  const mintToken = async () => {
    const mint = web3.Keypair.generate();

    const userTokenAccountAddress = (
      await getAtaForMint(mint.publicKey, walletAddress.publicKey)
    )[0];

    const userPayingAccountAddress = candyMachine.state.tokenMint
      ? (
          await getAtaForMint(
            candyMachine.state.tokenMint,
            walletAddress.publicKey
          )
        )[0]
      : walletAddress.publicKey;

    const candyMachineAddress = candyMachine.id;
    const remainingAccounts = [];
    const signers = [mint];
    const cleanupInstructions = [];
    const instructions = [
      web3.SystemProgram.createAccount({
        fromPubkey: walletAddress.publicKey,
        newAccountPubkey: mint.publicKey,
        space: MintLayout.span,
        lamports:
          await
candyMachine.program.provider.connection.getMinimumBalanceForRentExe
mption(
            MintLayout.span
          ),
        programId: TOKEN_PROGRAM_ID,
      }),
      Token.createInitMintInstruction(
        TOKEN_PROGRAM_ID,
        mint.publicKey,
        0,
        walletAddress.publicKey,
        walletAddress.publicKey
      ),
      createAssociatedTokenAccountInstruction(
        userTokenAccountAddress,
```

```
    walletAddress.publicKey,
    walletAddress.publicKey,
    mint.publicKey
  ),
  Token.createMintToInstruction(
    TOKEN_PROGRAM_ID,
    mint.publicKey,
    userTokenAccountAddress,
    walletAddress.publicKey,
    [],
    1
  ),
];

if (candyMachine.state.gatekeeper) {
  remainingAccounts.push({
    pubkey: (
      await getNetworkToken(
        walletAddress.publicKey,
        candyMachine.state.gatekeeper.gatekeeperNetwork
      )
    )[0],
    isWritable: true,
    isSigner: false,
  });
  if (candyMachine.state.gatekeeper.expireOnUse) {
    remainingAccounts.push({
      pubkey: CIVIC,
      isWritable: false,
      isSigner: false,
    });
    remainingAccounts.push({
      pubkey: (
        await getNetworkExpire(
          candyMachine.state.gatekeeper.gatekeeperNetwork
        )
      )[0],
      isWritable: false,
      isSigner: false,
    });
  }
}
if (candyMachine.state.whitelistMintSettings) {
  const mint = new web3.PublicKey(
    candyMachine.state.whitelistMintSettings.mint
  );

  const whitelistToken = (
    await getAtaForMint(mint, walletAddress.publicKey)
  )[0];
  remainingAccounts.push({
    pubkey: whitelistToken,
    isWritable: true,
    isSigner: false,
```

```
      });

      if
(candyMachine.state.whitelistMintSettings.mode.burnEveryTime) {
        const whitelistBurnAuthority = web3.Keypair.generate();

        remainingAccounts.push({
          pubkey: mint,
          isWritable: true,
          isSigner: false,
        });
        remainingAccounts.push({
          pubkey: whitelistBurnAuthority.publicKey,
          isWritable: false,
          isSigner: true,
        });
        signers.push(whitelistBurnAuthority);
        const exists =
          await
candyMachine.program.provider.connection.getAccountInfo(
            whitelistToken
          );
        if (exists) {
          instructions.push(
            Token.createApproveInstruction(
              TOKEN_PROGRAM_ID,
              whitelistToken,
              whitelistBurnAuthority.publicKey,
              walletAddress.publicKey,
              [],
              1
            )
          );
          cleanupInstructions.push(
            Token.createRevokeInstruction(
              TOKEN_PROGRAM_ID,
              whitelistToken,
              walletAddress.publicKey,
              []
            )
          );
        }
      }
    }

    if (candyMachine.state.tokenMint) {
      const transferAuthority = web3.Keypair.generate();

      signers.push(transferAuthority);
      remainingAccounts.push({
        pubkey: userPayingAccountAddress,
        isWritable: true,
        isSigner: false,
      });
```

```
      remainingAccounts.push({
        pubkey: transferAuthority.publicKey,
        isWritable: false,
        isSigner: true,
      });

      instructions.push(
        Token.createApproveInstruction(
          TOKEN_PROGRAM_ID,
          userPayingAccountAddress,
          transferAuthority.publicKey,
          walletAddress.publicKey,
          [],
          candyMachine.state.price.toNumber()
        )
      );
      cleanupInstructions.push(
        Token.createRevokeInstruction(
          TOKEN_PROGRAM_ID,
          userPayingAccountAddress,
          walletAddress.publicKey,
          []
        )
      );
    }
    const metadataAddress = await getMetadata(mint.publicKey);
    const masterEdition = await getMasterEdition(mint.publicKey);

    const [candyMachineCreator, creatorBump] = await
getCandyMachineCreator(
      candyMachineAddress
    );

    instructions.push(
      await candyMachine.program.instruction.mintNft(creatorBump, {
        accounts: {
          candyMachine: candyMachineAddress,
          candyMachineCreator,
          payer: walletAddress.publicKey,
          wallet: candyMachine.state.treasury,
          mint: mint.publicKey,
          metadata: metadataAddress,
          masterEdition,
          mintAuthority: walletAddress.publicKey,
          updateAuthority: walletAddress.publicKey,
          tokenMetadataProgram: TOKEN_METADATA_PROGRAM_ID,
          tokenProgram: TOKEN_PROGRAM_ID,
          systemProgram: SystemProgram.programId,
          rent: web3.SYSVAR_RENT_PUBKEY,
          clock: web3.SYSVAR_CLOCK_PUBKEY,
          recentBlockhashes: web3.SYSVAR_RECENT_BLOCKHASHES_PUBKEY,
          instructionSysvarAccount: web3.SYSVAR_INSTRUCTIONS_PUBKEY,
        },
        remainingAccounts:
```

```
          remainingAccounts.length > 0 ? remainingAccounts :
undefined,
      })
    );

    try {
      return (
        await sendTransactions(
          candyMachine.program.provider.connection,
          candyMachine.program.provider.wallet,
          [instructions, cleanupInstructions],
          [signers, []]
        )
      ).txs.map((t) => t.txid);
    } catch (e) {
      console.log(e);
    }
    return [];
  };

  return (
    candyMachine && (
      <div className="machine-container">
        <p>Drop Date: {candyMachine.state.goLiveDateTimeString}</p>
        <p>
          Items Minted: {candyMachine.state.itemsRedeemed} /
          {candyMachine.state.itemsAvailable}
        </p>
        <button className="cta-button mint-button"
onClick={mintToken}>
          Mint NFT
        </button>
      </div>
    )
  );
};

export default CandyMachine;
```