



Alejandro López Jiménez

2º DAW

ÍNDICE

1. Especificaciones del proyecto.....	3
2. Elementos innovadores.....	3
3. Estudio Previo.....	3
4. Documentación del Backend.....	5
5. Documentación del Frontent.....	7
6. Propuestas de mejora.....	15
7. Bibliografía.....	15

ESPECIFICACIONES DEL PROYECTO

Sponix consiste en un reproductor de música, en el cual se puede buscar y reproducir la canción que se escoja en directo.

Tengo que desarrollar el backend en Supabase, con la tabla de usuario que registrará los usuarios que se logueen en el programa; y en el frontend tengo que recoger esa información y también hacer búsquedas en directo por medio de la API de Spotify, a la vez de ser capaz de reproducir las canciones elegidas; todo con su estilo correspondiente.

ELEMENTOS INNOVADORES

React Redux: es uno de los mejores recolectores de estado, es algo complejo pero cuando se entiende rápido, se puede repartir el estado por todo el programa repartido por las variables que se quieran crear para su uso.

Supabase: es un backend bastante simple y fácil de aprender gracias a la API Docs.

Styled-components: es una forma de estilizar con CSS y de una forma más vistosa.

ESTUDIO PREVIO

Mi base para este proyecto fue el propio Spotify.

La página principal muestra un input y un div donde se muestran las canciones relacionadas con lo que se escribió en el input.

Cada canción tiene un botón de play el cual reproduce en directo la misma en el footer, donde se podrá descargar y manejar al gusto del cliente.

Quiero diseñar la app de manera similar, de tal manera que quede el sidebar, el header del usuario, el body compuesto por la búsqueda y respuestas listadas, y el footer donde se muestra la canción actual reproducida.

Lo primero que haré será desarrollar el backend para leer/insertar usuarios.

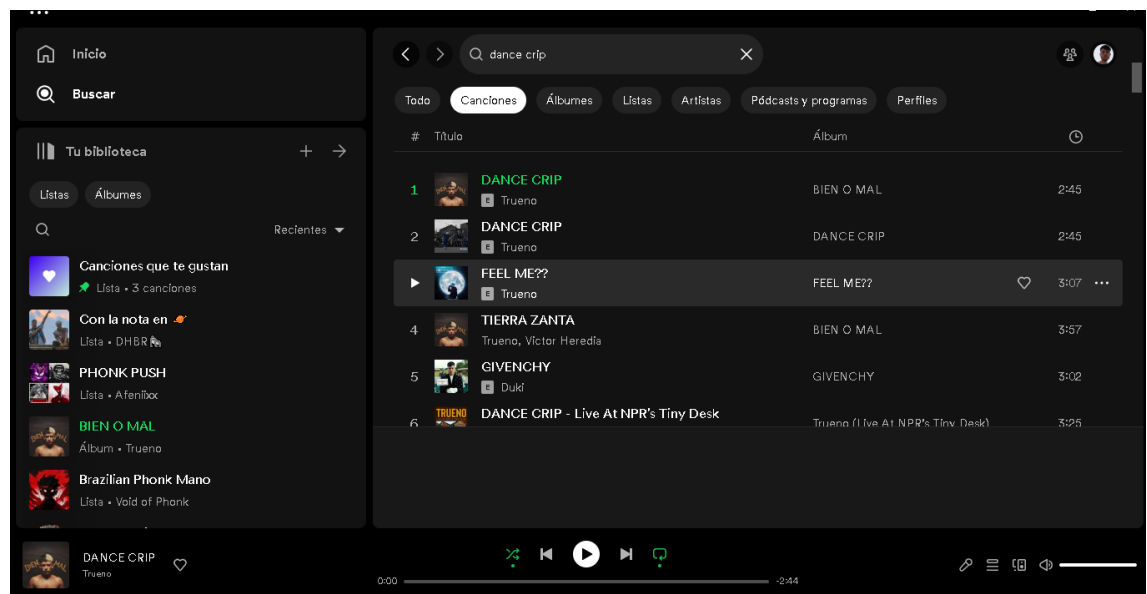
Lo segundo desarrollar el login con supabase, usando el de Spotify.

Lo tercero avanzar con los estilos de la página principal, de manera similar a la de Spotify y estilizado de manera similar al logotipo del programa.

Lo cuarto desarrollar las funciones del backend y de los redux para los estados.

Lo quinto hacer uso de la api de Spotify para poder hacer búsquedas con el token del provider y la búsqueda deseada.

SPOTIFY



LOGIN SPONIX

Inicia sesión con Spotify

Dirección de correo electrónico

Su dirección de correo electrónico

Su contraseña

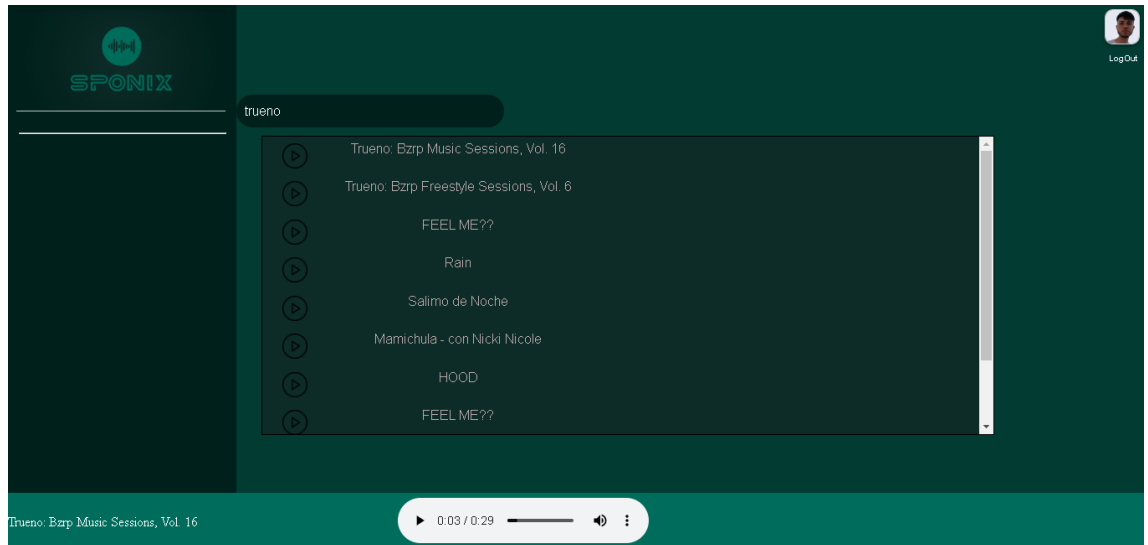
Tu contraseña

Iniciar sesión

[¿Olvidaste tu contraseña?](#)

[¿No tienes una cuenta? Regístrate](#)

PÁGINA PRINCIPAL SPONIX



DOCUMENTACIÓN DEL BACKEND

1. Para hacer una tabla en Supabase tendremos que iniciar sesión e irnos al menú de 'Table Editor' y 'New Table'; donde introduciremos el nombre de la misma y sus campos.

Create a new table under **public**

Name

Description

☒ **Enable Row Level Security (RLS)** Recommended
 Restrict access to your table by enabling RLS and writing Postgres policies.

① **Policies are required to query data**
 You need to write an access policy before you can query data from this table. Without a policy, querying this table will result in an `empty array` of results.
 You can create policies after you create this table.

[RLS Documentation](#)

☐ **Enable Realtime**
 Broadcast changes on this table to authorized subscribers

Cancel Save

- Ahora le daremos a 'API' arriba y observaremos todas las peticiones posibles a la tabla.

API

JavaScript Bash

user

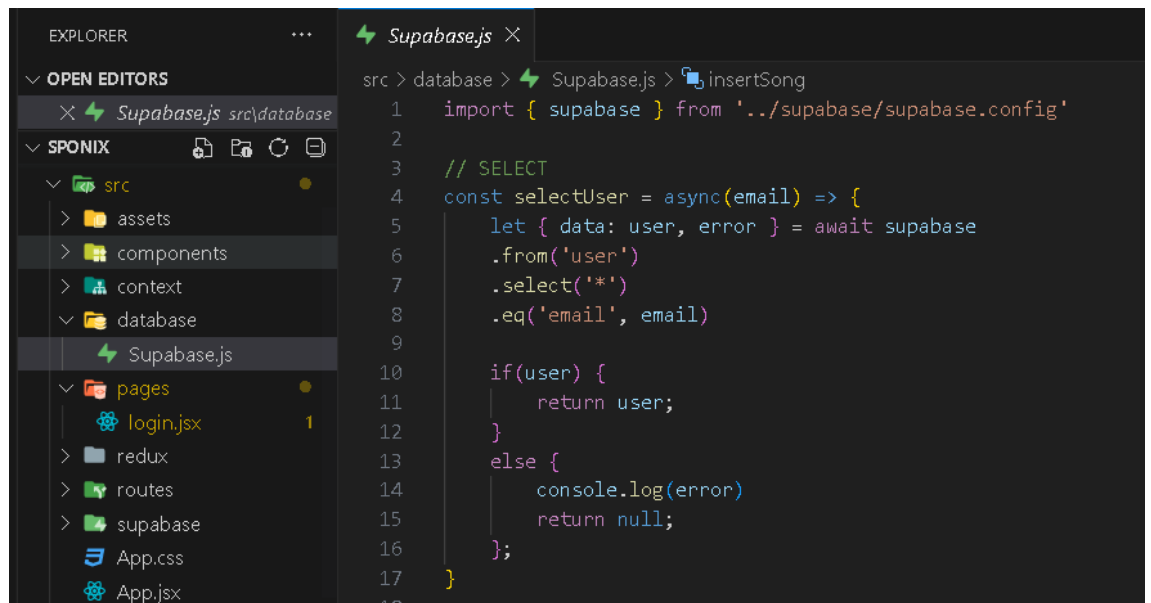
DESCRIPTION

COLUMN	id	Required	SELECT ID
TYPE	<> number		<code>let { data: user, error } = await supabase .from('user') .select('id')</code>
FORMAT	bigint		
DESCRIPTION	<input type="text" value="Click to edit."/>		

COLUMN	created_at	Optional	SELECT CREATED_AT

Close

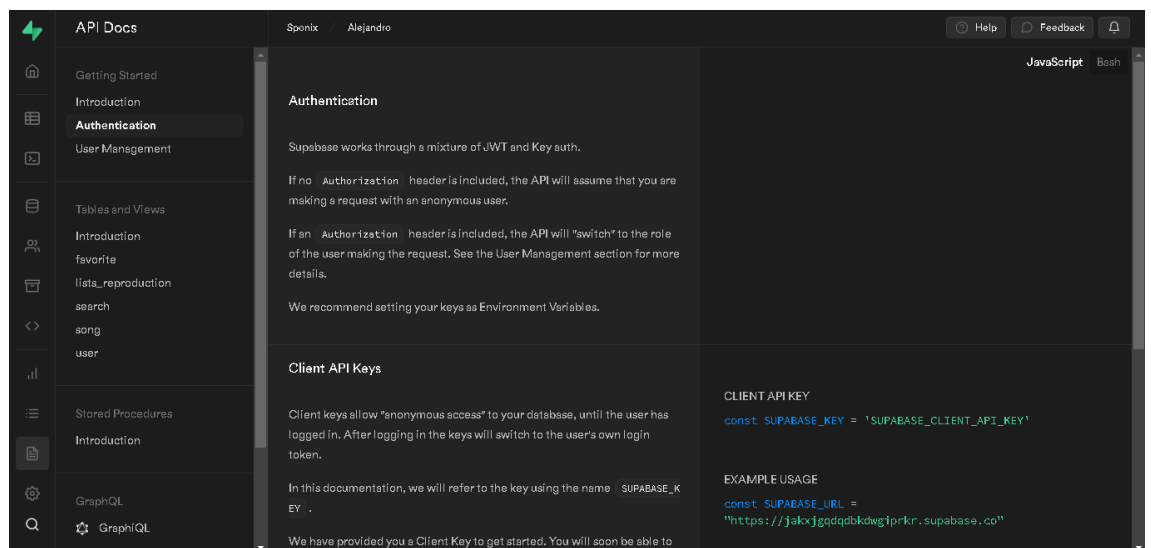
- Creamos un archivo el cual contendrá todas consultas posibles del programa.



```
src > database > Supabase.js > insertSong
1 import { supabase } from '../supabase/supabase.config'
2
3 // SELECT
4 const selectUser = async(email) => {
5   let { data: user, error } = await supabase
6     .from('user')
7     .select('*')
8     .eq('email', email)
9
10   if(user) {
11     return user;
12   }
13   else {
14     console.log(error)
15     return null;
16   }
17 }
```

DOCUMENTACIÓN DEL FRONTEND

1. Ahora que tenemos todas las consultas, crearemos el login. Para ello nos vamos a la documentación de Authentication



Authentication

Supabase works through a mixture of JWT and Key auth.

If no `Authorization` header is included, the API will assume that you are making a request with an anonymous user.

If an `Authorization` header is included, the API will "switch" to the role of the user making the request. See the [User Management](#) section for more details.

We recommend setting your keys as Environment Variables.

Client API Keys

Client keys allow "anonymous access" to your database, until the user has logged in. After logging in the keys will switch to the user's own login token.

In this documentation, we will refer to the key using the name `SUPABASE_CLIENT_API_KEY`.

We have provided you a Client Key to get started. You will soon be able to

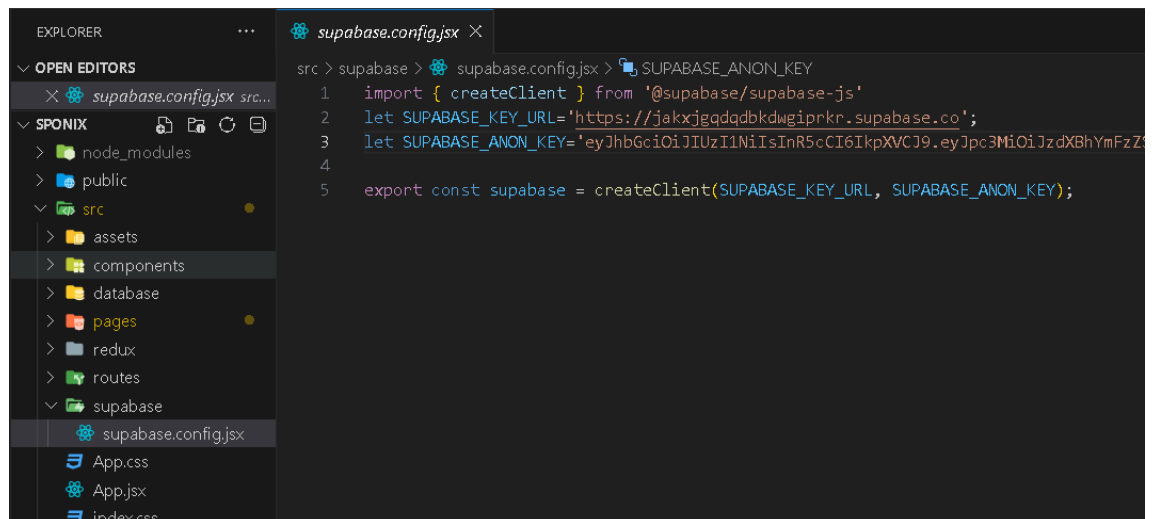
CLIENT API KEY

```
const SUPABASE_KEY = 'SUPABASE_CLIENT_API_KEY'
```

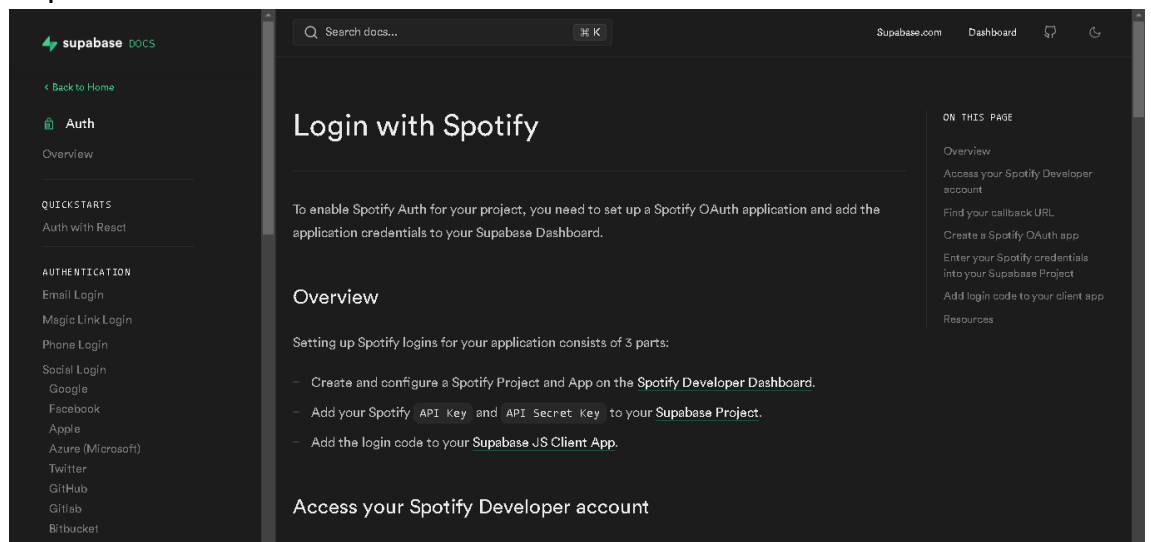
EXAMPLE USAGE

```
const SUPABASE_URL = "https://jakxjgddqdbkdwgiprkr.supabase.co"
const supabase = createClient(SUPABASE_URL,
```

2. Nos creamos un archivo donde crearemos el cliente de supabase para consumirlo; para ello nos hará falta la api cliente privada y el enlace a nuestro supabase.



- Ahora seguimos la página <https://supabase.com/docs/guides/auth/social-login/auth-spotify> para enlazar nuestro proyecto en Developer Spotify con Supabase.



4. Cuando lo hayamos enlazado, vamos a crear nuestra página de login:

- Primero vamos a seguir el enlace <https://react-redux.js.org/introduction/getting-started> para poder instalar Redux en nuestro proyecto
- Luego vamos a crear una carpeta donde guardaremos el slice (donde haremos las funciones necesarias declarar los estados y sus funciones manejadoras) y el store (donde se configurará el redux para poder usarlo). Por último englobamos con Provider la parte más alta del programa y de atributo store para que contenga los estados

The image consists of two screenshots of a code editor, likely VS Code, showing the implementation of Redux slices and the store configuration.

The top screenshot shows the `Slice.js` file. The code defines two slices: `AuthSlice` and `ActualSongSlice`. `AuthSlice` has an initial state with `user` (retrieved from `localStorage`) and reducers for `login` and `logout`. `ActualSongSlice` has an initial state with `actualSong` set to `null`.

```

1 import { createSlice } from '@reduxjs/toolkit'
2
3 const localUser = JSON.parse(localStorage.getItem('user'));
4
5 export const AuthSlice = createSlice({
6   name: 'auth',
7   initialState: {
8     user: localUser ? localUser : null,
9   },
10  reducers: {
11    login: (state, action) => {
12      state.user = action.payload;
13      localStorage.setItem('user', JSON.stringify({user: state.user}));
14    },
15    logout: (state) => {
16      state.user = null;
17      localStorage.removeItem('user');
18    }
19  }
20 });
21
22 export const ActualSongSlice = createSlice({
23   name: 'actualSong',
24   initialState: {
25     actualSong: null,
26   },
27 });

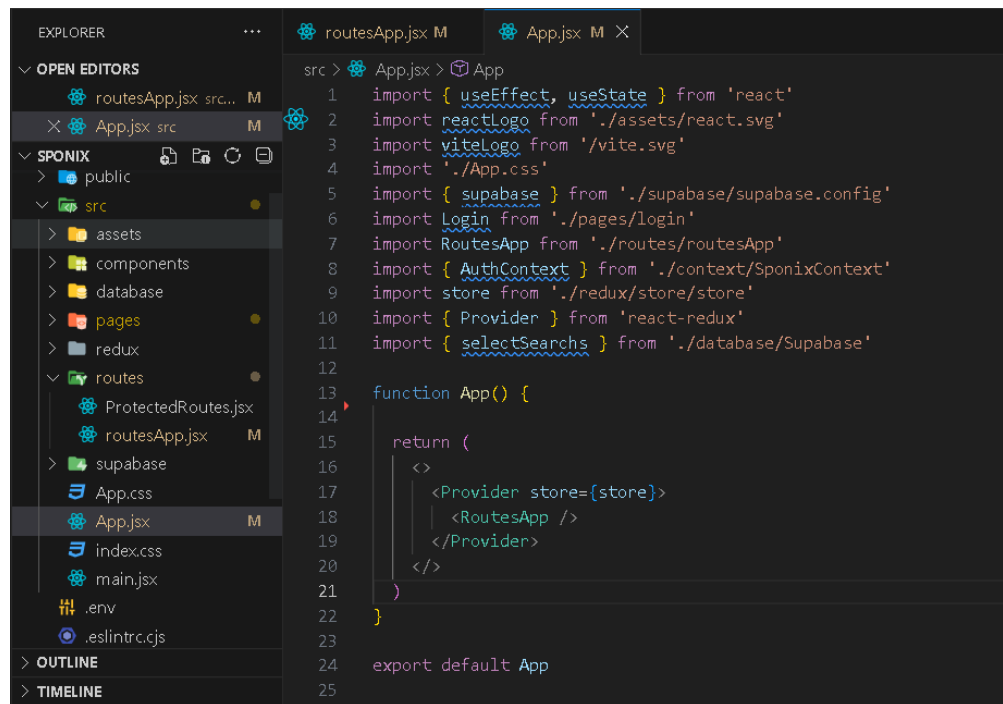
```

The bottom screenshot shows the `store.js` file. The code imports `configureStore` from `@reduxjs/toolkit` and the slices defined in `Slice.js`. It then configures the store with these slices as reducers.

```

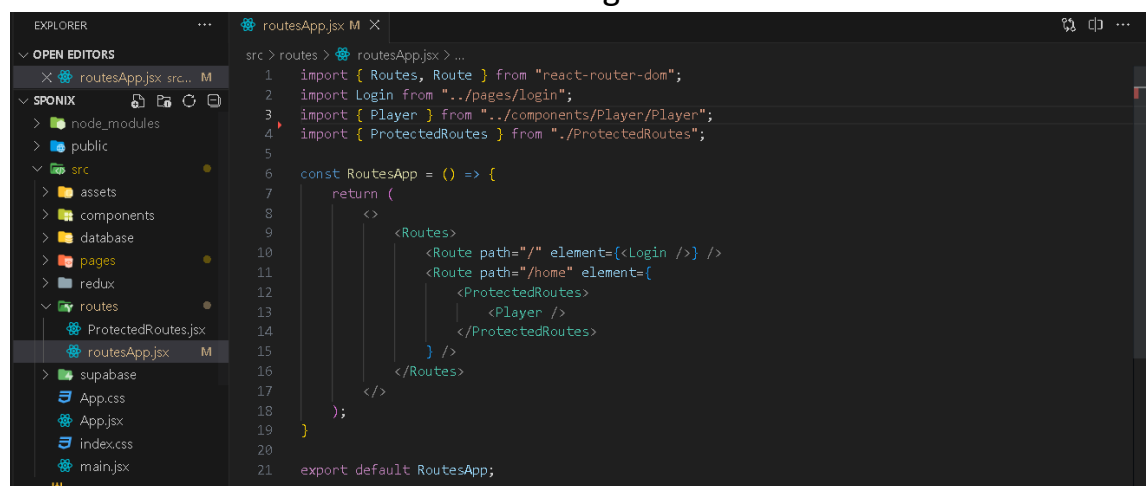
1 import { configureStore } from '@reduxjs/toolkit'
2 import { ActualSongSlice, AuthSlice, FavoriteSlice, ListReproductionSlice, SearchSlice } from '../slice/Slice'
3
4 export default configureStore({
5   reducer: {
6     auth: AuthSlice.reducer,
7     actualSong: ActualSongSlice.reducer,
8     favorite: FavoriteSlice.reducer,
9     listReproduction: ListReproductionSlice.reducer,
10    search: SearchSlice.reducer,
11  },
12 });

```

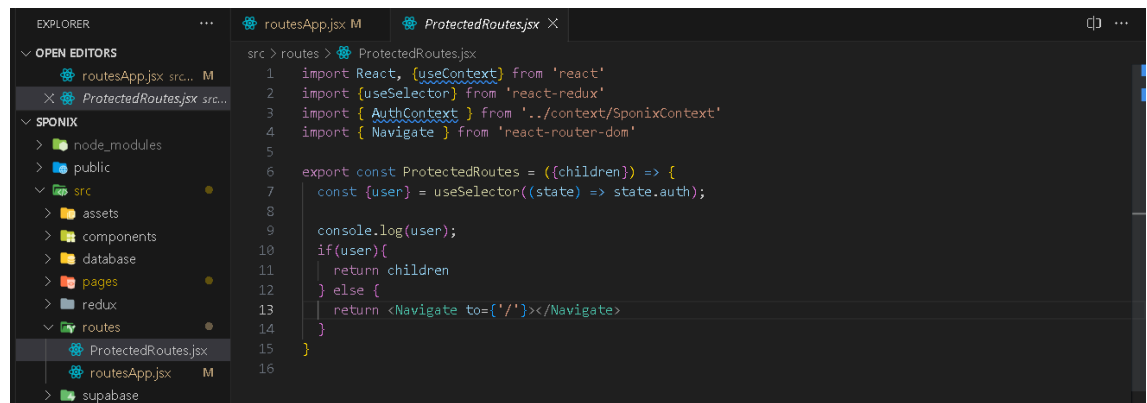


```
src > App.jsx > App
1  import { useEffect, useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5  import { supabase } from './supabase/supabase.config'
6  import Login from './pages/login'
7  import RoutesApp from './routes/routesApp'
8  import { AuthContext } from './context/SponixContext'
9  import store from './redux/store/store'
10 import { Provider } from 'react-redux'
11 import { selectSearchs } from './database/Supabase'
12
13 function App() {
14
15   return (
16     <>
17     <Provider store={store}>
18       <RoutesApp />
19     </Provider>
20   </>
21 )
22
23 }
24
25 export default App
```

- Ahora instalamos Router DOM para poder ir a distintos enlaces, como son el login y la página principal con 'yarn add react-router-dom' y creamos el fichero donde contendrá las rutas y otro para proteger la ruta de la página principal ante intento de entrada al no encontrarse logueado.



```
src > routes > routesApp.jsx > ...
1  import { Routes, Route } from "react-router-dom";
2  import Login from "../pages/login";
3  import { Player } from "../components/Player/Player";
4  import { ProtectedRoutes } from "../ProtectedRoutes";
5
6  const RoutesApp = () => {
7    return (
8      <>
9      <Routes>
10        <Route path="/" element={<Login />} />
11        <Route path="/home" element={
12          <ProtectedRoutes>
13            <Player />
14          </ProtectedRoutes>
15        } />
16      </Routes>
17    </>
18  );
19 }
20
21 export default RoutesApp;
```



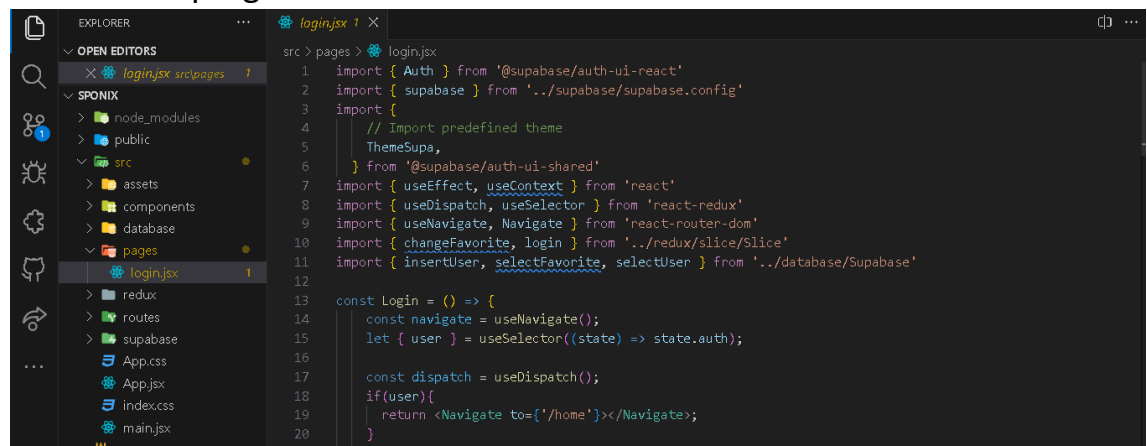
The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer shows a project structure with folders like node_modules, public, src, assets, components, database, pages, redux, routes, and supabase. The 'ProtectedRoutes.jsx' file is selected in the Explorer and is also open in the editor. The code in the editor is as follows:

```

1 import React, { useContext } from 'react'
2 import { useSelector } from 'react-redux'
3 import { AuthContext } from '../context/SponixContext'
4 import { Navigate } from 'react-router-dom'
5
6 export const ProtectedRoutes = ({ children }) => {
7   const { user } = useSelector((state) => state.auth);
8
9   console.log(user);
10  if(user){
11    return children
12  } else {
13    return <Navigate to={'/'}></Navigate>
14  }
15 }
16

```

useSelector es una función la cual nos servirá para leer los estados del programa



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer shows a project structure with folders like node_modules, public, src, assets, components, database, pages, redux, routes, supabase, App.css, App.jsx, index.css, and main.jsx. The 'login.jsx' file is selected in the Explorer and is also open in the editor. The code in the editor is as follows:

```

1 import { Auth } from '@supabase/auth-ui-react'
2 import { supabase } from '../supabase/supabase.config'
3 import {
4   // Import predefined theme
5   ThemeSupa,
6 } from '@supabase/auth-ui-shared'
7 import { useEffect, useContext } from 'react'
8 import { useDispatch, useSelector } from 'react-redux'
9 import { useNavigate, Navigate } from 'react-router-dom'
10 import { changeFavorite, login } from '../redux/slice/Slice'
11 import { insertUser, selectFavorite, selectUser } from '../database/Supabase'
12
13 const Login = () => {
14   const navigate = useNavigate();
15   let { user } = useSelector((state) => state.auth);
16
17   const dispatch = useDispatch();
18   if(user){
19     return <Navigate to={'/home'}></Navigate>;
20   }
21 }
22

```

Nos creamos una página de login, en la cual comprobaremos si está el estado del user logueado.



The screenshot shows a close-up of the VS Code editor with a useEffect hook. The code is as follows:

```

useEffect(() => {
  const {data} = supabase.auth.onAuthStateChange(
    async (event, session) => {
      console.log(event);
      if(event == 'SIGNED_IN'){
        navigate('/home', {replace:true});
        localStorage.setItem('user', session.user.user_metadata);
        dispatch(login(session.user.user_metadata));
        selectUser(session.user.email).then((u) => {
          if(!u[0]){
            insertUser(session.user.user_metadata);
          }
        })
      }
    }
  )

  return () => data.subscription.unsubscribe(data.subscription.id);
}, []);

```

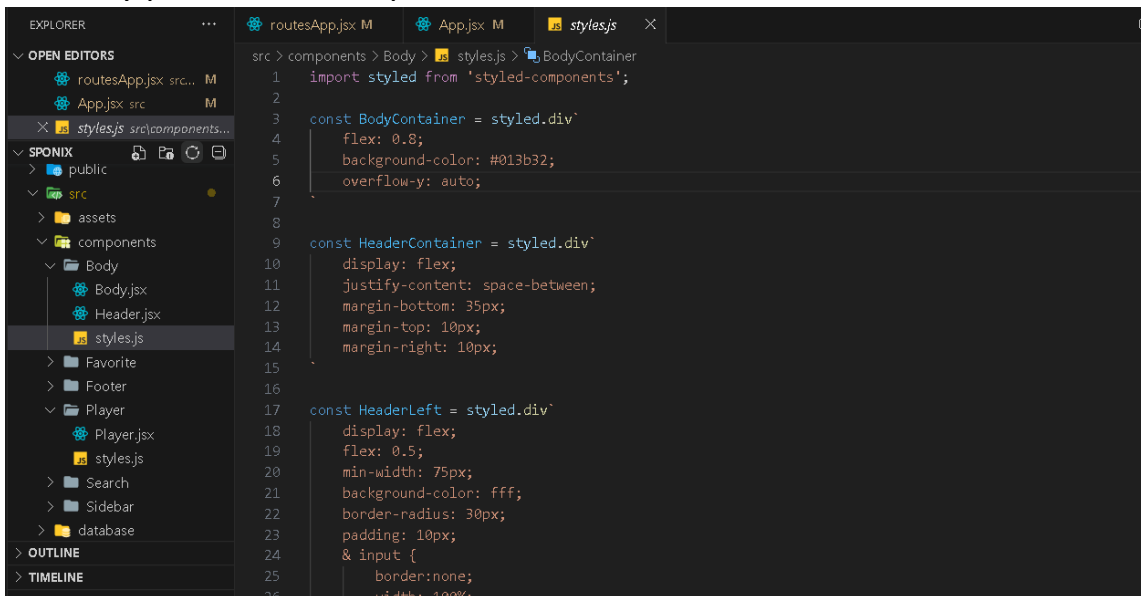
Con el objeto evento comprobamos la situación en la que está el usuario (SIGNED_ID, INITIAL_SESSION), e insertamos el usuario si no existe en la base de datos.

```
return (  
  <div className="auth">  
    <Auth  
      supabaseClient={supabase}  
      providers={['spotify']}  
      appearance={{theme: ThemeSupa}}  
      theme="dark"  
      localization={{  
        variables: {  
          sign_up : {  
            "email_label" : " Dirección de correo electrónico " ,  
            "password_label" : " Crear una contraseña " ,  
            "email_input_placeholder" : " Su dirección de correo electrónico "  
            "password_input_placeholder" : " Tu contraseña " ,  

```

Y devolvemos el objeto Auth de Supabase para poder autenticarnos con Spotify solamente con la propiedad providers (también podemos añadir estilos).

- Ahora para los componentes ejecutaremos 'yarn add styled-components' para separar los estilos de los componentes en variables. Funciona de tal manera que se selecciona el atributo que va a representar ese componente y meter estilos, y por último se exportan.

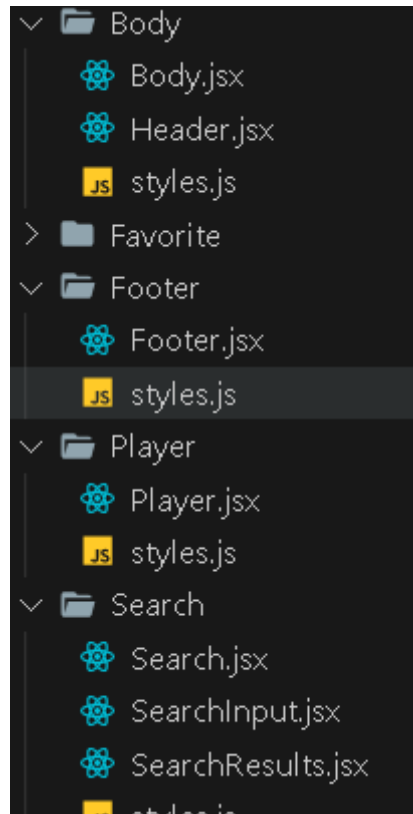


```
src > components > Body > styles.js > BodyContainer  
1  import styled from 'styled-components';  
2  
3  const BodyContainer = styled.div`  
4    flex: 0.8;  
5    background-color: #013b32;  
6    overflow-y: auto;  
7  
8  
9  const HeaderContainer = styled.div`  
10   display: flex;  
11   justify-content: space-between;  
12   margin-bottom: 35px;  
13   margin-top: 10px;  
14   margin-right: 10px;  
15  
16  
17  const HeaderLeft = styled.div`  
18   display: flex;  
19   flex: 0.5;  
20   min-width: 75px;  
21   background-color: fff;  
22   border-radius: 30px;  
23   padding: 10px;  
24   & input {  
25     border:none;  
26     width: 100%;
```

- En el componente Player lo que haremos es meter en el estado user nuestro usuario logueado pero recogido de la

base de datos, para tener los datos justos en los componentes necesarios.

- Después de esto, podemos hacer los componentes para que queden los más adecuado posibles.



- En el componente Search haremos las peticiones a la API de Spotify para recoger las canciones. Lo primero será recoger el provider_token que lo encontraremos en un objeto de localStorage creado al iniciar sesión, y lo utilizamos de la siguiente forma:

```
const [searchInput, setSearchInput] = useState('');
const [tracks, setTracks] = useState();
const dispatch = useDispatch();

useEffect(() => {
  if(searchInput.length >= 5){
    const provider_token = JSON.parse(localStorage.getItem('sb-jakxjgqddbkdwgipkr-auth-token')).provider_token;

    const options = {
      headers: {
        Authorization: `Bearer ${provider_token}`
      }
    };

    fetch(`https://api.spotify.com/v1/search?q=remaster%20track%3A${searchInput}&type=track&market=ES&limit=10`)
      .then(response => response.json())
      .then(response => setTracks(response.tracks.items))
      .catch(err => console.error(err));
  } else {
    setTracks([]);
  }
}, [searchInput])
```

Las especificaciones de los tipos de objetos y las diferentes peticiones a la api que se pueden hacer se encuentran en <https://developer.spotify.com/documentation/web-api/reference>

- Esta función nos ayudará para poder reproducir la canción escogida en el footer.

```
const changeActSong = (event) => {
  const provider_token = JSON.parse(localStorage.getItem('sb-jakxjgqddbkwgipkr-auth-token')).provider_token;

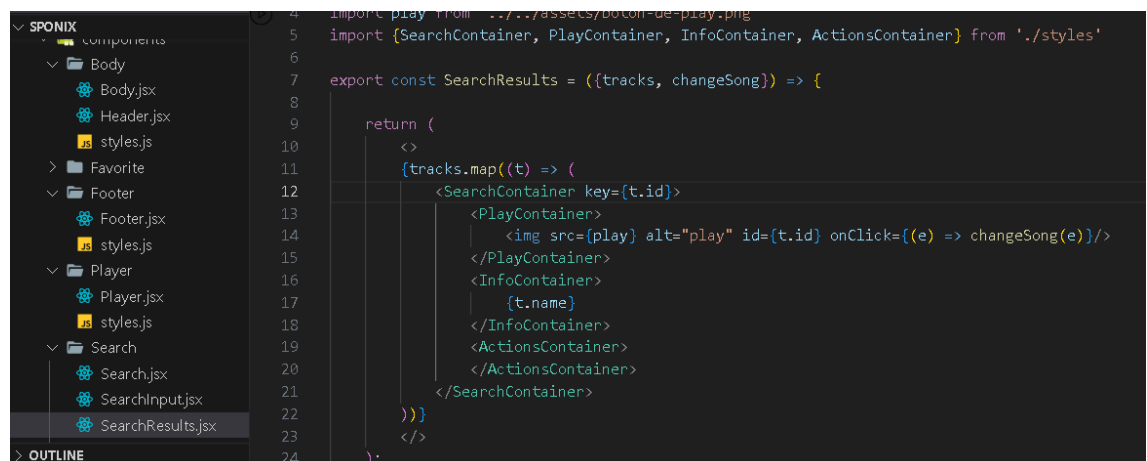
  const options = {
    headers: {
      Authorization: `Bearer ${provider_token}`
    }
  };

  console.log(event.target.id);

  fetch(`https://api.spotify.com/v1/tracks/${event.target.id}`, options)
    .then(response => response.json())
    .then(response => dispatch(changeActualSong(response)))
    .catch(err => console.error(err));
}
```

Con dispatch podemos ejecutar las funciones de los estados para cambiarlos.

- Y ya representamos los resultados en otro componente con la funcionalidad de cambiar la canción actual:



```
import play from '../assets/button-de-play.png'
import {SearchContainer, PlayContainer, InfoContainer, ActionsContainer} from './styles'

export const SearchResults = ({tracks, changeSong}) => {
  return (
    <>
      {tracks.map((t) => (
        <SearchContainer key={t.id}>
          <PlayContainer>
            <img src={play} alt="play" id={t.id} onClick={(e) => changeSong(e)} />
          </PlayContainer>
          <InfoContainer>
            {t.name}
          </InfoContainer>
          <ActionsContainer>
            </ActionsContainer>
        </SearchContainer>
      ))}
    </>
  );
}
```

- Y en el componente footer recogemos la canción actual escogida antes de la siguiente manera:

```

let {actualSong} = useSelector((state) => state.actualSong);

if(document.querySelector('#audioActual')) {
  document.querySelector('#audioActual').play();
}

return (
  <FooterContainer>
    {actualSong &&
      <>
        <FooterLeft>
          {actualSong.name}
        </FooterLeft>
        <FooterCenter>
          <AudioContainer>
            <audio id="audioActual" controls>
              <source id="sourceActual" src={actualSong?.preview_url} type="audio/mpeg" />
              Tu navegador no soporta la reproducción de audio.
            </audio>
          </AudioContainer>
        </FooterCenter>
      </>
    }
  </FooterContainer>
)

```

PROPUESTAS DE MEJORA

Se podría añadir la funcionalidad de añadir a favoritos y tener la propia lista donde escuchar tus canciones favoritas.

También el hecho de crear listas de reproducciones para agrupar las canciones al estilo del cliente.

BIBLIOGRAFÍA

<https://app.supabase.com/>

<https://developer.spotify.com/>

<https://react-redux.js.org/introduction/getting-started>