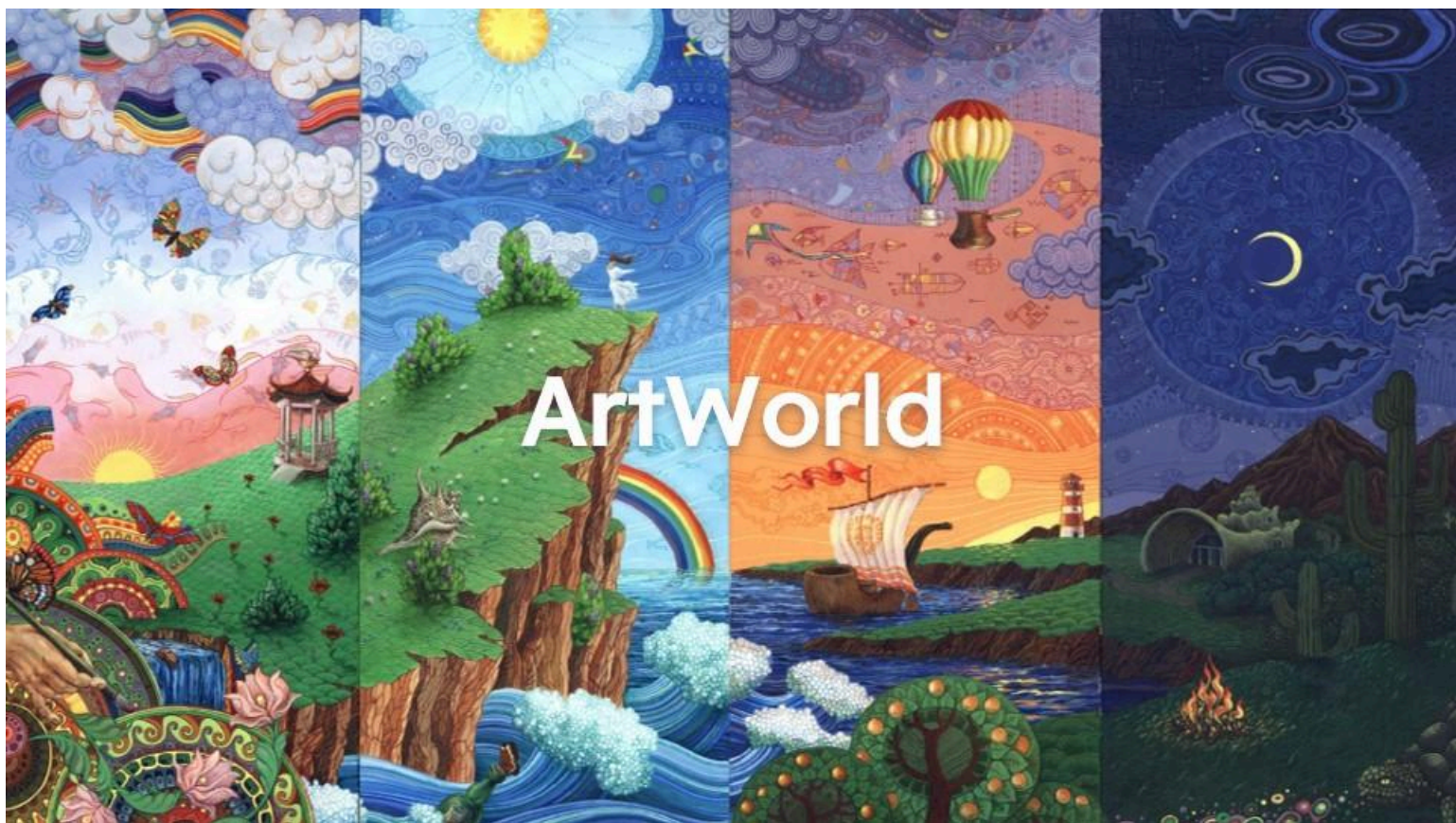


Memoria



Índice

Índice	1
Objetivos	2
Introducción	3
Descripción del proyecto	4
Instalaciones y Configuraciones Iniciales	4
1. Crear Proyecto con Vite y Configurar React:	4
2. Instalar y Configurar Tailwind CSS con Vite:	4
3. Añadir Material Tailwind:	5
4. Añadir Modo Oscuro (Dark Mode):	6
5. Añadir Traducción con i18next:	7
6. Añadir Animaciones con Tailwind CSS Animated:	7
7. Añadir react-router-dom:	8
8. Mostrar iconos react-icons	9
Crear un Proyecto en Supabase y Explicación de las Tablas	10
Crear un Proyecto en Supabase	10
Tabla: Profiles	11
Tabla: Paintings	11
Tabla: Sculptures	12
Estructura de Carpetas del Proyecto	13
Código detallado	14
Conclusiones	49
Fuentes	50
Anexos	52
Temporización de las tareas	52

Objetivos

Página de arte en que para ver los contenidos (Pinturas y Esculturas), necesitas iniciar sesión. Y el administrador podrá editar, eliminar contenido y usuarios, además de añadir más arte.

Introducción

En esta memoria habrá una explicación detallada de mi código en **Descripción del proyecto**, una opinión propia sobre ello en **Conclusiones**, enlaces a toda la información y recursos que me ayudó para hacer este trabajo en **Fuentes**, y cuánto he tardado en **Anexos** con mi github incluido por si quieres preguntarme alguna duda, descargar el proyecto, o algún archivo en concreto.

Descripción del proyecto

Instalaciones y Configuraciones Iniciales

1. Crear Proyecto con Vite y Configurar React:

```
npm create vite .
```

```
# Selecciona React y Javascript cuando te lo solicite
```

```
npm install
```

```
npm run dev
```

Esta serie de comandos inicializa un nuevo proyecto con Vite, elige React como el framework y luego instala las dependencias necesarias. Finalmente, inicia el servidor de desarrollo.

2. Instalar y Configurar Tailwind CSS con Vite:

```
npm install -D tailwindcss postcss autoprefixer
```

```
npx tailwindcss init -p
```

Esto instala Tailwind CSS junto con PostCSS y Autoprefixer, y luego genera los archivos de configuración *tailwind.config.js* y *postcss.config.js*.

Modifica *tailwind.config.js* para especificar dónde Tailwind debería buscar clases CSS para generar estilos:

```
/** @type {import('tailwindcss').Config} */
```

```
export default {
```

```
content: [  
  "./index.html",  
  "./src/**/*.{js,ts,jsx,tsx}",  
],  
theme: {  
  extend: {},  
},  
plugins: [],  
});
```

3. **Añadir Material Tailwind:**

```
npm i @material-tailwind/react
```

Modifica *tailwind.config.js* para incluir Material Tailwind:

```
/** @type {import('tailwindcss').Config} */  
const withMT = require('@material-tailwind/react/utils/withMT');  
module.exports = withMT({  
  content: [  
    "./index.html",  
    "./src/**/*.{js,ts,jsx,tsx}",  
  ],  
  theme: {  
    extend: {},  
  },  
  plugins: [],  
});
```

4. Añadir Modo Oscuro (Dark Mode):

```
/** @type {import('tailwindcss').Config} */
const withMT = require('@material-tailwind/react/utils/withMT');
module.exports = withMT({
  content: [
    './index.html',
    './src/**/*.{js,ts,jsx,tsx}',
  ],
  darkMode: ['class'],
  theme: {
    extend: {},
  },
  plugins: [],
});
```

Luego, en tu componente *Header.jsx*, añade una función para cambiar al modo oscuro:

```
function changeDarkMode() {
  document.documentElement.classList.toggle('dark');
}
```

Y un botón que llame a esta función:

```
<button onClick={changeDarkMode} className="h-7 w-7
bg-white dark:bg-blue-gray-800 rounded-md shadow-lg"
aria-hidden="true">
```

```
<ImContrast className='w-full dark:text-white
text-blue-gray-800'/>
</button>
```

5. **Añadir Traducción con i18next:**

```
npm install i18next react-i18next
```

Crea un archivo *i18next.jsx* en la carpeta *src* para configurar i18next. Aquí puedes inicializar las traducciones y configuraciones necesarias.

6. **Añadir Animaciones con Tailwind CSS Animated:**

```
npm i tailwindcss-animated
```

Modifica *tailwind.config.js* para incluir el plugin de animaciones:

```
/** @type {import('tailwindcss').Config} */
const withMT = require('@material-tailwind/react/utils/withMT');
module.exports = withMT({
  content: [
    './index.html',
    './src/**/*.{js,ts,jsx,tsx}',
  ],
  darkMode: ['class'],
  theme: {
    extend: {},
  },
  plugins: [
```



```
require('tailwindcss-animated')  
],  
});
```

7. Añadir react-router-dom:

npm i react-router-dom

Se instala para que las páginas estén entrelazadas con la etiqueta Route mediante etiquetas Routes

```
export default function App() {  
  return (  
    <div className="dark:bg-blue-gray-900 min-h-screen">  
      <Header />  
      <Routes>  
        <Route path="/" element={<Home />} />  
        <Route path="/sculptures" element={<Sculptures />} />  
        <Route path="/paintings" element={<Paintings />} />  
        <Route path="/login" element={<Login />} />  
        <Route path="/signUp" element={<SignUp />} />  
        <Route path="/users" element={<AdminRoute><Users /></AdminRoute>} />  
      </Routes>  
      <Footer />  
    </div>  
  );  
}
```

Y para navegar entre ellas, debe de usarse la etiqueta Link

```
<li>  
  <Link  
    to="/"  
    className="..."  
  >  
    {t('Home')}  
  </Link>  
</li>
```

Pero para que el router funcione, se debe de poner una etiqueta BrowserRouter.

```
ReactDOM.createRoot(document.getElementById('root')).render(  
  <React.StrictMode>  
    <GlobalProvider>  
      <BrowserRouter> ←  
        <App />  
      </BrowserRouter>  
    </GlobalProvider>  
  </React.StrictMode>,  
)
```

8. Mostrar iconos react-icons

npm i react-icons --save

Crear un Proyecto en Supabase y Explicación de las Tablas

Crear un Proyecto en Supabase

1. Registro e Ingreso:
 - Ve a [Supabase](https://supabase.com) y regístrate o inicia sesión si ya tienes una cuenta.
2. Crear un Proyecto Nuevo:
 - En el panel de Supabase, haz clic en el botón **"New Project"**.
 - Llena los campos necesarios como el nombre del proyecto y la contraseña de la base de datos.
 - Selecciona la región más cercana a tu ubicación para un mejor rendimiento.
3. Configurar la Base de Datos:
 - Una vez que el proyecto se haya creado, serás redirigido al panel del proyecto.
 - Navega a la sección de **"Database"**.
4. Crear Tablas:
 - En el panel de **"Table Editor"**, puedes comenzar a crear tus tablas.

Explicación de las Tablas:

Las imágenes que he subido contienen capturas de las tablas *Profiles*, *Paintings* y *Sculptures*. Aquí están los detalles de cómo puedes recrear

estas tablas en Supabase, incluyendo las columnas que finalmente no utilizo por falta de tiempo (*status*, *can_access_paintings*, *can_access_sculptures*).

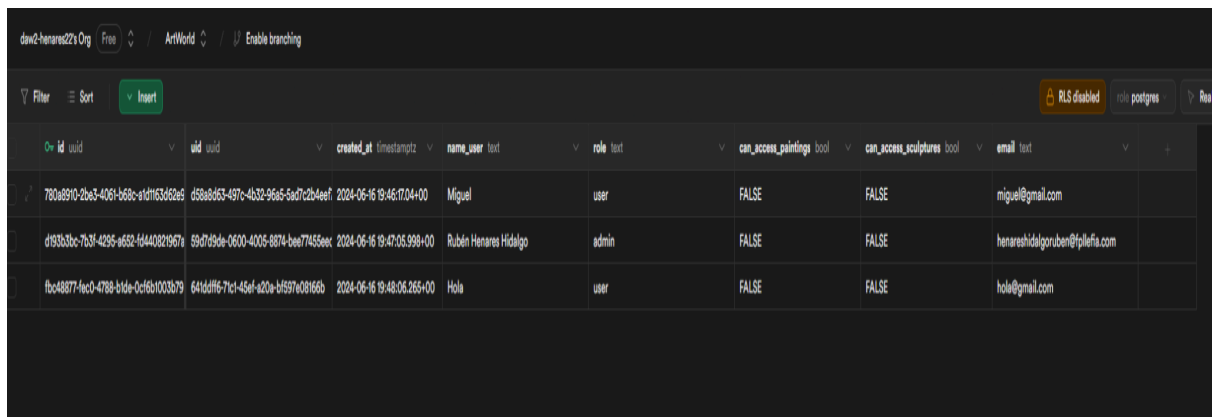
Tabla: Profiles

La tabla *Profiles* contiene información sobre los usuarios.

1. Estructura de la Tabla:

- *id* (UUID): Identificador único de cada perfil.
- *uuid* (UUID): Otro identificador único, posiblemente una referencia cruzada.
- *created_at* (timestamp): Fecha y hora de creación del perfil.
- *name_user* (text): Nombre del usuario.
- *role* (text): Rol del usuario (*user* o *admin*).
- *email* (text): Correo electrónico del usuario.

2. Creación de la Tabla:



id	uuid	created_at	name_user	role	can_access_paintings	can_access_sculptures	email
780a8910-2be3-4061-b68c-e1d163662e9	d5ba8d63-497c-4b32-96a5-5ed7c2b4ee7	2024-06-16 19:46:01.04+00	Miguel	user	FALSE	FALSE	miguel@gmail.com
d193b3bc-763f-4295-a652-4b440821967e	59d7b9de-0600-4005-8874-bae77455ee7	2024-06-16 19:47:05.998+00	Rubén Henares Hidalgo	admin	FALSE	FALSE	henaresidalgoruben@pillefia.com
fbca8877-4ec0-4788-b1de-0c16b1003b79	641ddff6-71c1-45e4-a20a-bf597a08166b	2024-06-16 19:48:06.265+00	Hola	user	FALSE	FALSE	hola@gmail.com

Tabla: Paintings

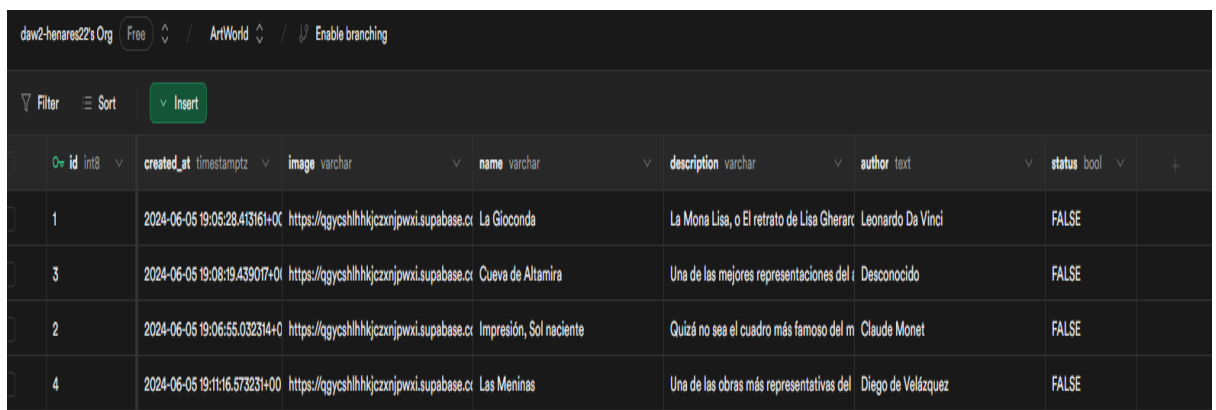
La tabla *Paintings* contiene información sobre diferentes pinturas.

1. Estructura de la Tabla:

- *id* (int8): Identificador único de cada pintura.

- *created_at* (timestamp): Fecha y hora de creación de la entrada.
- *image* (varchar): URL de la imagen de la pintura.
- *name* (varchar): Nombre de la pintura.
- *description* (varchar): Descripción de la pintura.
- *author* (text): Autor de la pintura.
- *status* (bool): No utilizado.

2. Creación de la Tabla:



id	created_at	image	name	description	author	status
1	2024-06-05 19:05:28.413161+00	https://ggycsihlhkjcxnjpwxi.supabase.c...	La Gioconda	La Mona Lisa, o El retrato de Lisa Gherar...	Leonardo Da Vinci	FALSE
3	2024-06-05 19:08:19.43907+00	https://ggycsihlhkjcxnjpwxi.supabase.c...	Cueva de Altamira	Una de las mejores representaciones del...	Desconocido	FALSE
2	2024-06-05 19:06:55.032314+00	https://ggycsihlhkjcxnjpwxi.supabase.c...	Impresión, Sol naciente	Quizá no sea el cuadro más famoso del m...	Claude Monet	FALSE
4	2024-06-05 19:11:16.573231+00	https://ggycsihlhkjcxnjpwxi.supabase.c...	Las Meninas	Una de las obras más representativas del...	Diego de Velázquez	FALSE

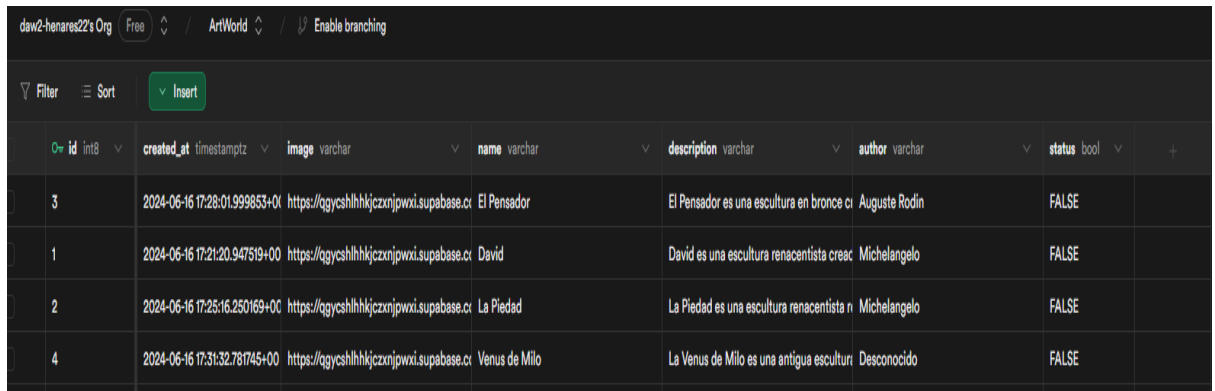
Tabla: Sculptures

La tabla *Sculptures* contiene información sobre diferentes esculturas.

1. Estructura de la Tabla:

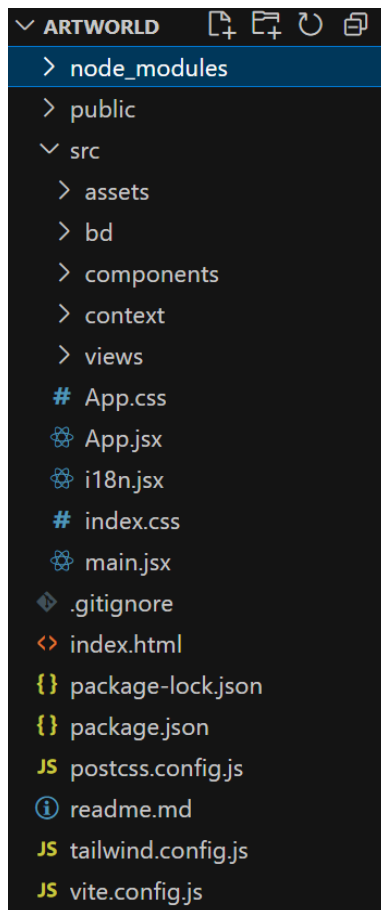
- *id* (int8): Identificador único de cada escultura.
- *created_at* (timestamp): Fecha y hora de creación de la entrada.
- *image* (varchar): URL de la imagen de la escultura.
- *name* (varchar): Nombre de la escultura.
- *description* (varchar): Descripción de la escultura.
- *author* (text): Autor de la escultura.
- *status* (bool): No utilizado.

2. Creación de la Tabla:



id	created_at	image	name	description	author	status	
3	2024-06-16 17:28:01.999853+00	https://qgytshlhhkjcxnjpwxi.supabase.co	El Pensador	El Pensador es una escultura en bronce de	Auguste Rodin	FALSE	
1	2024-06-16 17:21:20.947519+00	https://qgytshlhhkjcxnjpwxi.supabase.co	David	David es una escultura renacentista creada	Michelangelo	FALSE	
2	2024-06-16 17:25:16.250169+00	https://qgytshlhhkjcxnjpwxi.supabase.co	La Piedad	La Piedad es una escultura renacentista de	Michelangelo	FALSE	
4	2024-06-16 17:31:32.781745+00	https://qgytshlhhkjcxnjpwxi.supabase.co	Venus de Milo	La Venus de Milo es una antigua escultura	Desconocido	FALSE	

Estructura de Carpetas del Proyecto



- **/src:** Carpeta principal del código fuente.
- **/assets:** Recursos estáticos.
- **/bd:** Configuraciones o funciones de la base de datos.

- **/context**: Manejo del estado global de la aplicación.
- **/views**: Componentes de vistas principales.
- **App.css**: Estilos del componente raíz.
- **App.jsx**: Componente raíz de la aplicación.
- **i18n.jsx**: Configuración de internacionalización.
- **main.jsx**: Punto de entrada principal de React.
- **index.css**: Estilos globales.
- **index.html**: Estructura HTML base de la aplicación.

Código detallado

/bd supabase.js

1. Importación del Cliente de Supabase:

```
import { createClient } from '@supabase/supabase-js'
```

Aquí estás importando la función *createClient* desde el paquete *@supabase/supabase-js*, que es la biblioteca oficial de JavaScript para interactuar con Supabase.


2. Configuración del Cliente Público de Supabase:

```
const supabaseUrl = 'https://qgycshlhhkjcxnjpwxi.supabase.co'  
const supabaseKey = '
```

Project API keys

Your API is secured behind an API gateway which requires an API Key for every request.

You can use the keys below in the Supabase client libraries.

 Client Docs

anon public

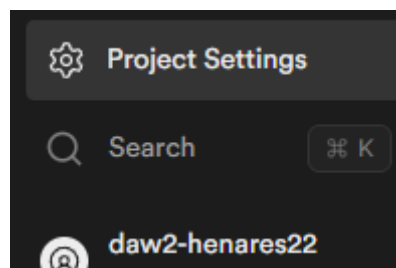
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzIiwiZnVzIjoiIiwiaWF0IjoiYXNjaXZlLnFneWNzaGxoaGtqY3p4bmI

This key is safe to use in a browser if you have enabled Row Level Security for your tables and configured policies.

```
export const supabase = createClient(supabaseUrl, supabaseKey)
```

supabaseUrl: Es la URL de mi instancia de Supabase. Cada proyecto en Supabase tiene una URL única.

supabaseKey: Es la clave pública de mi proyecto, utilizada para interactuar con la API de Supabase, que está situada en *Project Settings*.



createClient(supabaseUrl, supabaseKey): Esta función crea un cliente de Supabase que puedes usar en tu aplicación para realizar operaciones CRUD y otras interacciones con tu base de datos.

3. Configuración del Cliente de Servicio de Supabase:

```
const supabaseServiceKey = '
```

service_role secret

**** **** **** ****

This key has the ability to bypass Row Level Security. Never share

```
export const supabaseService = createClient(supabaseUrl,
supabaseServiceKey);
```


supabaseServiceKey: Es una clave de servicio que generalmente tiene más permisos que la clave pública, como la capacidad de realizar acciones administrativas o acceder a datos restringidos que al igual que *supabaseKey*, también está situada en *Project Settings*.
createClient(supabaseUrl, supabaseServiceKey): Esta instancia del cliente se usa para tareas que requieren más permisos, y no debería exponerse en el frontend para evitar problemas de seguridad.

/context GlobalContext.jsx

1. Importaciones

```
import React, { createContext, useContext, useEffect, useState }  
from 'react';  
import { supabase } from '../bd/supaBase';
```

React y Hooks: *createContext*, *useContext*, *useEffect*, *useState* son hooks de React para crear y manejar el estado y los efectos secundarios.

Supabase: *supabase* es el cliente configurado en *supaBase.js* para interactuar con la base de datos de Supabase.

2. Creación del Contexto

```
const GlobalContext = createContext();
```

GlobalContext: Crea un nuevo contexto de React que puede ser usado para compartir el estado globalmente en la aplicación.

3. Proveedor del Contexto

```
export const GlobalProvider = ({ children }) => {  
  const [session, setSession] = useState(null);  
  const [isAdmin, setIsAdmin] = useState(false);  
  
  useEffect(() => {  
    const checkSession = async () => {  
      const { data: { session } } = await  
supabase.auth.getSession();  
      if (session) {  
        setSession(session);  
        fetchUserRole(session.user.email);  
      } else {  
        setSession(null);  
        setIsAdmin(false);  
      }  
    };  
  
    checkSession();  
  
    const { data: authListener } =  
supabase.auth.onAuthStateChange((_event, session) => {  
      if (session?.user) {  
        setSession(session);  
        fetchUserRole(session.user.email);  
      } else {
```

```
        setSession(null);
        setIsAdmin(false);
    }
});

return () => {
    authListener.subscription.unsubscribe();
};
}, []);

const fetchUserRole = async (userEmail) => {
    if (userEmail === 'henareshidalgoruben@fpillefia.com') {
        setIsAdmin(true);
        return;
    }
    const { data, error } = await supabase
        .from('Profiles')
        .select('role')
        .eq('email', userEmail)
        .single();
    if (data) {
        setIsAdmin(data.role === 'admin');
    } else if (error) {
        console.error('Error fetching user role:', error);
    }
};

return (
```

```
    <GlobalContext.Provider value={{ session, isAdmin,  
    setSession, setIsAdmin }}>  
      {children}  
    </GlobalContext.Provider>  
  );  
};
```

Estados: *session* para manejar la sesión actual del usuario y *isAdmin* para determinar si el usuario es administrador.

useEffect: Se usa para ejecutar efectos secundarios. En este caso, para verificar la sesión del usuario al cargar el componente y para suscribirse a los cambios en el estado de autenticación.

- **checkSession:** Verifica la sesión actual del usuario usando *supabase.auth.getSession()*. Si hay una sesión, se guarda en el estado y se obtiene el rol del usuario. Si no, se resetean los estados.
- **authListener:** Se suscribe a los cambios en el estado de autenticación (inicio/cierre de sesión). Actualiza el estado en consecuencia y obtiene el rol del usuario si hay una sesión.
- **fetchUserRole:** Obtiene el rol del usuario desde la tabla *Profiles* en Supabase. Si el correo del usuario coincide con uno específico, se establece como administrador automáticamente.

4. Uso del Contexto

```
export const useGlobalContext = () => useContext(GlobalContext);
```

useGlobalContext: Es un hook personalizado que facilita el acceso al contexto global en los componentes de React.

/components LanguageToggleButton.jsx

1. Importaciones

```
import { useTranslation } from 'react-i18next';  
import { useState } from 'react';
```

useTranslation: Es un hook de *react-i18next* que proporciona acceso a las funciones de internacionalización, incluyendo la capacidad de cambiar el idioma actual.

useState: Es un hook de React que permite agregar estado a un componente funcional.

2. Definición del Componente

```
export function LanguageToggleButton() {  
  const { i18n } = useTranslation();  
  const [language, setLanguage] = useState(i18n.language);  
  
  const toggleLanguage = () => {  
    const newLanguage = language === 'es' ? 'en' : 'es';  
    i18n.changeLanguage(newLanguage);  
    setLanguage(newLanguage);  
  };  
  
  return (  
    <button onClick={toggleLanguage} className="px-2 py-1 mx-1  
text-white dark:text-blue-gray-800 dark:hover:text-blue-gray-400
```

```

hover:text-blue-gray-400 hover:scale-x-105 hover:scale-y-105
transition duration-150 border-double border-4 border-spacing-4
border-white dark:border-blue-gray-800 rounded">
  {language === 'es' ? 'EN' : 'ES'}
</button>
);
}

```

3. Estado y Funcionalidad

- Estado del Idioma

```
const [language, setLanguage] = useState(i18n.language);
```

language: Almacena el idioma actual. Inicialmente, se establece en el idioma actual de *i18n*.

setLanguage: Función para actualizar el estado del idioma.

4. Función toggleLanguage

```

const toggleLanguage = () => {
  const newLanguage = language === 'es' ? 'en' : 'es';
  i18n.changeLanguage(newLanguage);
  setLanguage(newLanguage);
};

```

Esta función alterna entre 'es' (español) y 'en' (inglés).

Cambia el idioma en *i18n* usando

i18n.changeLanguage(newLanguage).

Actualiza el estado local *language* con el nuevo idioma.

5. Renderizado del Botón

- Botón

```
<button onClick={toggleLanguage} className="px-2 py-1
mx-1 text-white dark:text-blue-gray-800
dark:hover:text-blue-gray-400 hover:text-blue-gray-400
hover:scale-x-105 hover:scale-y-105 transition duration-150
border-double border-4 border-spacing-4 border-white
dark:border-blue-gray-800 rounded">
  {language === 'es' ? 'EN' : 'ES'}
</button>
```

- **onClick:** Asigna la función *toggleLanguage* al evento *onClick* del botón.
- **Classes:** Utiliza clases de Tailwind CSS para el estilo. Las clases incluyen:
 - *px-2 py-1 mx-1*: Espaciado interno y externo.
 - *text-white dark:text-blue-gray-800*
dark:hover:text-blue-gray-400
hover:text-blue-gray-400: Colores de texto para temas claros y oscuros.
 - *hover:scale-x-105 hover:scale-y-105 transition*
duration-150: Efecto de escalado y transición al pasar el ratón sobre el botón.

- *border-double border-4 border-spacing-4 border-white dark:border-blue-gray-800 rounded:*
Estilo de borde y redondeo.

6. Texto del Botón

```
{language === 'es' ? 'EN' : 'ES'}
```

Muestra 'EN' si el idioma actual es español y 'ES' si el idioma actual es inglés.

Raíz del proyecto **i18n.jsx**

1. Importaciones

```
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';
```

i18n: Es el núcleo de la biblioteca *i18next* que maneja la lógica de internacionalización.

initReactI18next: Es el módulo que integra *i18next* con React, permitiendo utilizar hooks como *useTranslation* dentro de componentes React.

2. Traducciones

```
const resources = {
  en: { translation: { "Home":"Home", "Name":"Name", /*...más traducciones...*/ } },
```



```

    es: { translation: { "Home":"Inicio", "Name":"Nombre", /*...más
    traducciones...*/ } }
  };

```

resources: Es un objeto que contiene las traducciones para los idiomas soportados. En este caso, hay dos idiomas: inglés (*en*) y español (*es*).

translation: Es el objeto dentro de cada idioma que contiene las claves y los valores traducidos.

3. Inicialización de *i18next*

i18n

```

.use(initReactI18next) // Vincula con react-i18next
.init({
  resources,
  lng: 'es', // Idioma predeterminado
  interpolation: {
    escapeValue: false // React ya hace escaping
  }
});

```

- **.use(initReactI18next):** Conecta *i18next* con *react-i18next*, permitiendo el uso de traducciones dentro de componentes React.
- **.init:** Configura *i18next* con las opciones proporcionadas:
 - **resources:** Proporciona el objeto de recursos con las traducciones.

- **lng:** Establece el idioma predeterminado de la aplicación (en este caso, 'es' para español).
- **interpolation.escapeValue:** Configurado en *false* porque React ya se encarga del escape de valores por motivos de seguridad.

4. Exportación

```
export default i18n;
```

export default i18n: Exporta la instancia configurada de *i18next* para que pueda ser utilizada en otras partes de la aplicación.

Todas las vistas o componentes que tengan traducción, tendrán este hook:

useTranslation: Hook de react-i18next para la traducción de textos

/views Home.jsx

1. Importaciones

```
import { Carousel, Typography } from "@material-tailwind/react";  
import { useTranslation } from "react-i18next";
```

Carousel y Typography: Componentes de *@material-tailwind/react* utilizados para el carrusel y la tipografía.

useTranslation: Hook de *react-i18next* para la traducción de textos

La vista Home es la página principal del proyecto

/components Footer.jsx

1. Importaciones

```
import React from 'react';  
import { useTranslation } from 'react-i18next';  
import { FaGithub } from 'react-icons/fa';
```

FaGithub: Ícono de GitHub con tamaño y color especificados para modo normal y oscuro.

El componente Footer siempre estará en toda la página gracias a App.jsx y main.jsx por el Router.

Raíz del proyecto main.jsx

1. Importaciones

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
import App from './App';  
import './index.css';  
import { BrowserRouter } from 'react-router-dom';  
import './i18n';
```

```
import { GlobalProvider } from './context/globalContext';
```

- *React*: Biblioteca principal de React.
- *ReactDOM*: Módulo para interactuar con el DOM en una aplicación React.
- *App*: Componente principal de la aplicación.
- *index.css*: Archivo de estilos globales.
- *BrowserRouter*: Componente para manejar la navegación en la aplicación.
- *i18n*: Archivo de configuración para la internacionalización.
- *GlobalProvider*: Proveedor de contexto global para la gestión del estado global.

2. Renderización de la Aplicación

```
ReactDOM.createRoot(document.getElementById('root')).render(  
  <React.StrictMode>  
    <GlobalProvider>  
      <BrowserRouter>  
        <App />  
      </BrowserRouter>  
    </GlobalProvider>  
  </React.StrictMode>,  
);
```

- *ReactDOM.createRoot*: Crea una raíz de React en el elemento con el ID *root* del DOM.

- `<React.StrictMode>`: Activa comprobaciones adicionales y advertencias en modo desarrollo.
- `<GlobalProvider>`: Envuelve la aplicación para proporcionar el contexto global.
- `<BrowserRouter>`: Proporciona funcionalidad de enrutamiento basado en navegador.
- `<App>`: Componente principal de la aplicación que contiene el resto de los componentes.

Raíz del proyecto `App.jsx`

1. Importaciones

```
import { Route, Routes } from 'react-router-dom';
import './App.css';
import { Header } from './components/Header';
import { Home } from './views/Home';
import { Sculptures } from './views/Sculptures';
import { Paintings } from './views/Paintings';
import { Users } from './views/Users';
import { Login } from './components/Login';
import { SignUp } from './components/SignUp';
import { Footer } from './components/Footer';
import { useGlobalContext } from './context/globalContext';
import { Navigate } from 'react-router-dom';
```

- ***Route y Routes***: Son componentes de *react-router-dom* usados para definir y manejar las rutas de la aplicación.
- ***./App.css***: Archivo de estilos CSS para la aplicación.

- **Componentes Importados:** Se importan varios componentes desde diferentes carpetas del proyecto, como *Header*, *Home*, *Sculptures*, *Paintings*, *Users*, *Login*, *SignUp* y *Footer*.
- **useGlobalContext:** Hook personalizado que obtiene el contexto global de la aplicación.
- **Navigate:** Componente de *react-router-dom* utilizado para redirigir a una ruta diferente.

2. Función AdminRoute

```
const AdminRoute = ({ children }) => {  
  const { isAdmin } = useGlobalContext();  
  return isAdmin ? children : <Navigate to="/" />;  
};
```

- **AdminRoute:** Componente que actúa como una ruta protegida. Solo permite el acceso a sus hijos (*children*) si el usuario es administrador.
 - **useGlobalContext:** Obtiene el valor de *isAdmin* del contexto global.
 - **isAdmin ? children : <Navigate to="/" />:** Si *isAdmin* es *true*, renderiza los componentes hijos; de lo contrario, redirige a la página de inicio (/).

3. Componente principal App

```
export default function App() {
  return (
    <div className="dark:bg-blue-gray-900 min-h-screen">
      <Header />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/sculptures" element={<Sculptures />} />
        <Route path="/paintings" element={<Paintings />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signUp" element={<SignUp />} />
        <Route path="/users" element={<AdminRoute><Users
      /></AdminRoute>} />
    </Routes>
    <Footer />
  </div>
);
}
```

- **<div className="dark:bg-blue-gray-900 min-h-screen">**: Envuelve toda la aplicación en un *div* con estilos CSS que aseguran que el fondo sea de color azul grisáceo oscuro en modo oscuro y que ocupe al menos la altura de la pantalla.
- **<Header />**: Renderiza el componente *Header* en la parte superior de la página.
- **<Routes>**: Contiene todas las rutas definidas para la aplicación.

- **<Route path="/" element={< />} />**: Define la ruta para la página principal (/) o cualquier ruta que quieras y renderiza el componente que pongas.
- **<Route path="/users" element={<AdminRoute><Users /></AdminRoute>} />**: Define la ruta para la gestión de usuarios (/users). Esta ruta está protegida por el componente *AdminRoute*, que solo permite el acceso si el usuario es un administrador. Renderiza el componente *Users*.
- **<Footer />**: Renderiza el componente *Footer* en la parte inferior de la página.

/components SignUp.jsx:

1. Importaciones

```
import { Button, Card, CardBody, CardFooter, Dialog, Input,
  Typography } from "@material-tailwind/react";
```

```
import { useState } from "react";
import { Link } from "react-router-dom";
import { supabase } from "../bd/supaBase";
import { useTranslation } from "react-i18next";
```

Componentes de Material Tailwind: *Button, Card, CardBody, CardFooter, Dialog, Input, Typography.*

React Hooks: *useState* para manejar el estado del componente.

React Router: *Link* para la navegación.

Supabase: *supabase* para la autenticación y base de datos.

i18next: *useTranslation* para la traducción.

2. Estados

```
const [open, setOpen] = useState(false);  
const [dialogData, setDialogData] = useState({ name: "", email: "",  
password: "" });
```

open: Estado para controlar la visibilidad del diálogo de registro.

dialogData: Estado para manejar los datos del formulario de registro.

3. Eventos

```
const handleOpen = () => setOpen((cur) => !cur);
```

```
function handleChange(event) {  
  setDialogData((prevDialogData) => ({  
    ...prevDialogData,  
    [event.target.name]: event.target.value  
  }));  
}
```

handleOpen: Alterna la visibilidad del diálogo.

handleChange: Actualiza los datos del formulario a medida que el usuario escribe.

4. Registro

```
async function handleSubmit(e) {
  e.preventDefault();
  try {
    let { data, error } = await supabase.auth.signUp({
      email: dialogData.email,
      password: dialogData.password,
      options: { data: { name: dialogData.name } }
    });
    if (error) throw error;

    const uid = data.user.id;
    const role = data.user.email ===
'henareshidalgoruben@fpillefia.com' ? 'admin' : 'user';
    let { error: profileError } = await
supabase.from('Profiles').insert([
      {
        uid: uid,
        email: data.user.email,
        name_user: dialogData.name,
        role: role,
        created_at: new Date()
      }
    ]);
    if (profileError) throw profileError;

    alert('Register successfully');
```

```
    } catch (error) {  
      console.error('Error:', error);  
      alert(error.message);  
    }  
  }  
}
```

- **handleSubmit:** Maneja el envío del formulario de registro.
 - Previene el comportamiento predeterminado del formulario.
 - Intenta registrar un nuevo usuario con Supabase.
 - Si el registro es exitoso, guarda los datos adicionales del usuario en la base de datos.
 - Muestra una alerta de éxito o error según el resultado.

5. Renderizado

- **Botón de Registro:** Abre el diálogo de registro al hacer clic.
- **Diálogo de Registro:** Contiene el formulario de registro.
 - **Formulario:** Incluye campos para el nombre, correo electrónico y contraseña.
 - **Botón de Envío:** Envía el formulario de registro.
 - **Enlace a Iniciar Sesión:** Redirige a la página de inicio de sesión si ya se tiene una cuenta.

/components Login.jsx:

1. Importaciones

(Código ya visto anteriormente no hace falta que lo explique de nuevo)

```
import { Button, Card, CardBody, CardFooter, Checkbox, Dialog,
Input, Typography } from "@material-tailwind/react";
import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import { supabase } from "../bd/supaBase";
import { useGlobalContext } from '../context/globalContext';
import { useTranslation } from "react-i18next";
```

2. Estado

```
const [open, setOpen] = useState(false);
const [dialogData, setDialogData] = useState({ email: "", password:
" });
```

3. Inicio de sesión

```
async function handleSubmit(e) {
  e.preventDefault();
  try {
    let { data, error } = await supabase.auth.signInWithPassword({
      email: dialogData.email,
      password: dialogData.password
    });
    if (error) throw error;
    setSession(data.session);
```

```
    navigate('/');  
  } catch (error) {  
    console.error('Error logging in:', error.message);  
    if (error.message.includes('Invalid login credentials')) {  
      alert('Wrong email or password');  
    } else {  
      alert('Error logging in');  
    }  
  }  
}
```

Previene el comportamiento predeterminado del formulario.

Intenta iniciar sesión con Supabase.

Si el inicio de sesión es exitoso, guarda la sesión en el contexto global y redirecciona a la página principal.

Muestra una alerta de error si las credenciales son incorrectas o si ocurre algún otro error.

4. Renderizado

- **Botón de Inicio de Sesión:** Abre el diálogo de inicio de sesión al hacer clic.
- **Diálogo de Inicio de Sesión:** Contiene el formulario de inicio de sesión.
 - **Formulario:** Incluye campos para el correo electrónico y la contraseña.
 - **Botón de Envío:** Envía el formulario de inicio de sesión.

- **Enlace a Registro:** Redirige a la página de registro si no se tiene una cuenta.

/components Header.jsx:

1. Importaciones

```
import { useTranslation } from "react-i18next";
import { ImContrast } from "react-icons/im";
import { Link, useNavigate } from "react-router-dom";
import { LanguageToggleButton } from
"./LanguageToggleButton";
import { Login } from "./Login";
import { SignUp } from "./SignUp";
import { Button } from "@material-tailwind/react";
import { useGlobalContext } from '../context/globalContext';
import { supabase } from '../bd/supaBase';
```

Iconos: *ImContrast* de *react-icons* para el icono de cambio de modo oscuro.

2. Estado y Funciones

```
const { t } = useTranslation();
const { session, setSession, isAdmin, setIsAdmin } =
useGlobalContext();
let navigate = useNavigate();

function changeDarkMode() {
```

```
document.documentElement.classList.toggle('dark');  
}  
  
async function handleLogout() {  
  const { error } = await supabase.auth.signOut();  
  if (error) {  
    console.error('Error during logout:', error);  
  } else {  
    setSession(null);  
    setIsAdmin(false);  
    navigate('/');  
  }  
}
```

handleLogout: Maneja la lógica de cierre de sesión, cerrando sesión en Supabase, actualizando el contexto global y redireccionando a la página principal.

3. Renderizado

- **Encabezado y Título:** Muestra el nombre de la aplicación "ArtWorld" con un enlace a la página principal.
- **Mensaje de Bienvenida:** Muestra un mensaje de bienvenida al usuario si está autenticado.
- **Barra de Navegación:** Contiene enlaces a diferentes secciones de la aplicación y opciones de autenticación:
 - **Botón de Modo Oscuro:** Permite cambiar entre modo claro y oscuro.

- **Enlaces de Navegación:** *Home, Sculptures, Paintings, Users* (solo visible para administradores).
- **Botón de Cambio de Idioma:** Utiliza *LanguageToggleButton*.
- **Botones de Autenticación:** *SignUp* y *Login* (solo visibles si no hay una sesión activa).
- **Botón de Cierre de Sesión:** Visible si hay una sesión activa, maneja la lógica de cierre de sesión.

/views Paintings.jsx:

1. Importaciones

```
import { Dialog, Card, CardBody, CardFooter, Input, Typography, Button } from '@material-tailwind/react';
```

```
import { useEffect, useState } from 'react';  
import { useTranslation } from 'react-i18next';  
import { supabase } from '../bd/supaBase';  
import { useGlobalContext } from '../context/globalContext';
```

2. Obtención

```
useEffect(() => {  
  const fetchPaintings = async () => {  
    try {  
      const { data: paintings, error } = await  
supabase.from('Paintings').select('*');
```



```
    if (error) {  
      throw error;  
    }  
    setPaintings(paintings);  
  } catch (error) {  
    console.error('Error fetching paintings:', error.message);  
  }  
};  
  
fetchPaintings();  
}, []);
```

Obtiene todas las pinturas de la base de datos.

3. Eliminar

```
const handleDelete = async (paintingId) => {  
  try {  
    const { error } = await supabase.from('Paintings').delete().eq('id',  
paintingId);  
    if (error) {  
      throw error;  
    }  
    setPaintings(paintings.filter(painting => painting.id !==  
paintingId));  
  } catch (error) {  
    console.error('Error deleting painting:', error.message);  
  }  
}
```

```
};
```

Elimina una pintura de la base de datos y actualiza el estado.

4. Editar

```
const handleEditSubmit = async (e) => {  
  e.preventDefault();  
  try {  
    const { data, error } = await  
supabase.from('Paintings').update(newPainting).eq('id',  
editPainting.id).select();  
    if (error) {  
      throw error;  
    }  
    if (!data || data.length === 0) {  
      throw new Error('Update failed, no data returned.');    }  
    setPaintings(paintings.map(painting => (painting.id ===  
editPainting.id ? data[0] : painting)));  
    setNewPainting({ name: "", description: "", author: "", image: "",  
status: false });  
    setEditPainting(null);  
    setOpenN(false);  
  } catch (error) {  
    console.error('Error updating painting:', error.message);  
  }  
};
```

Actualiza una pintura en la base de datos y en el estado.

5. Añadir

```
const handleSubmit = async (e) => {
  e.preventDefault();
  try {
    const { data, error } = await
supabase.from('Paintings').insert([newPainting]).select();
    if (error) {
      throw error;
    }
    if (!data || data.length === 0) {
      throw new Error('Insert failed, no data returned.');
```

Agrega una nueva pintura a la base de datos y actualiza el estado.

6. Renderizar

Renderiza una lista de pinturas.

Muestra un diálogo con la imagen de la pintura cuando se hace clic en ella.

Si el usuario es administrador, muestra botones para editar y eliminar cada pintura, así como un botón para agregar nuevas pinturas.

/views Sculptures.jsx

Es lo mismo que Paintings.jsx pero en vez de tener pinturas como datos, tiene esculturas.

/views Users.jsx

1. Importaciones

```
import { useState, useEffect } from 'react';  
import { useGlobalContext } from '../context/globalContext';  
import { useNavigate } from 'react-router-dom';  
import { supabase, supabaseService } from '../bd/supaBase';  
import { useTranslation } from 'react-i18next';
```

2. Estado y Funciones

```
export const Users = () => {  
  const { t } = useTranslation();  
  const { token, isAdmin, setIsAdmin } = useGlobalContext();
```

```

const [users, setUsers] = useState([]);
const navigate = useNavigate();

useEffect(() => {
  if (!isAdmin) {
    navigate('/');
    return;
  }
  fetchUsers();
}, [isAdmin, navigate]);

```

token, isAdmin, setIsAdmin: Valores del contexto global, donde *isAdmin* determina si el usuario actual es administrador.

users: Array que almacena los usuarios obtenidos de la base de datos.

navigate: Función para redirigir al usuario.

3. Obtener Usuarios

```

const fetchUsers = async () => {
  try {
    let { data, error } = await supabase
      .from('Profiles')
      .select('*')
      .neq('email', 'henareshidalgoruben@fpillefia.com'); //
  } catch (error) {
    console.log(error);
  }
}

```

Excluir el usuario con el correo específico

```
        if (error) throw error;
        setUsers(data);
    } catch (error) {
        console.error("Error fetching users:", error);
    }
};
```

fetchUsers(): Función que obtiene todos los usuarios de la base de datos, excluyendo uno específico, y actualiza el estado *users*.

4. Eliminar

```
const handleDeleteUser = async (userId) => {
    try {
        const { data: user, error: fetchError } = await supabase
            .from('Profiles')
            .select('uid')
            .eq('id', userId)
            .single();

        if (fetchError) throw fetchError;
        const userUID = user.uid;

        const { error: authError } = await
supabaseService.auth.admin.deleteUser(userUID);
        if (authError) throw authError;

        let { error } = await supabase
```

```

        .from('Profiles')
        .delete()
        .eq('id', userId);
    if (error) throw error;

    setUsers(users.filter(user => user.id !== userId));
  } catch (error) {
    console.error("Error deleting user:", error);
  }
};

```

handleDeleteUser(userId): Función que elimina un usuario tanto de la tabla *Profiles* como del autenticador de Supabase, actualizando el estado *users*.

5. Editar

```

const handleEditUser = async (userId, updatedUser) => {
  try {
    let { error } = await supabase
      .from('Profiles')
      .update(updatedUser)
      .eq('id', userId);
    if (error) throw error;
    setUsers(users.map(user => user.id === userId ? { ...user,
      ...updatedUser } : user));
  }
};

```

```
const { data: { session }, error: sessionError } = await
supabase.auth.getSession();
if (sessionError) throw sessionError;
if (session && session.user.id === userId) {
  fetchUserRole(userId);
}
} catch (error) {
  console.error("Error updating user:", error);
}
};
```

handleEditUser(userId, updatedUser): Función que actualiza un usuario en la base de datos y actualiza el estado *users*. Si el usuario actualiza su propio rol, se refresca el contexto global.

6. Obtener Rol

```
const fetchUserRole = async (userId) => {
  const { data, error } = await supabase
    .from('Profiles')
    .select('role')
    .eq('id', userId)
    .single();
  if (data) {
    setIsAdmin(data.role === 'admin');
  } else if (error) {
    console.error('Error fetching user role:', error);
  }
}
```



```
};
```

fetchUserRole(userId): Función que obtiene el rol de un usuario y actualiza el estado global *isAdmin*.

7. Renderizar

Verifica si el usuario es administrador. Si no, retorna *null*.

Muestra una tabla con los usuarios, permitiendo la edición de su nombre y rol, y la eliminación de usuarios mediante botones y entradas de formulario.

Conclusiones

El proyecto aunque sea sencillo, le metí cada día trabajo de tardes enteras, e incluso en ocasiones un poco de mañana y varias noches. Muchas de esas horas eran de error tras error, ya sea por código que al final he podido hacer como otros que no. Esos que quería hacer, pienso que si me hubiese puesto antes a hacer el proyecto seguramente lo hubiese conseguido. Pero gracias a muchos errores he podido aprender, para seguir este o futuros proyectos. La web que he creado, a mi me gusta y me siento orgulloso de ello, por programador y como un hobby que es el arte. Por eso me gusta más frontend que backend, aunque en este tipo de proyectos hay que hacer más a la perfección el backend.

Fuentes

Información sobre como usar Supabase con javascript:

<https://supabase.com/docs/reference/javascript/initializing>

Instalación y como usar tailwindcss: <https://tailwindcss.com/>

Instalación y como usar material-tailwind:

<https://www.material-tailwind.com/>

Instalación y como usar tailwindcss animated:

<https://www.tailwindcss-animated.com/>

Vistazo a tailwind components. Es una web que sirve de repositorio para la comunidad de tailwindcss, incluso hay código gratis, que en la página de tailwindcss es de pago. No usé nada de esto porque ya tenía un css preparado, sino, sin duda lo usaba: <https://tailwindcomponents.com/>

Instalar y como usar react icons:

<https://react-icons.github.io/react-icons/>

Instalación y como usar i18next (aprendí a usarlo sobretodo por el video de youtube en su página web): <https://www.i18next.com/>

Imágenes de Sculptures creadas con IA:

<https://firefly.adobe.com/inspire/images>

Datos de Sculptures: <https://chatgpt.com/>

Imágenes y datos de Paintings:

<https://www.elledecor.com/es/arte/a39595095/cuadros-famosos-historia-obras-arte/>

Anexos

En la **Descripción del proyecto**, ya expliqué mi código entero. Si alguien lo necesita, se encuentra en mi github:

<https://github.com/daw2-henares22/ArtWorld>

Y para ver mi proyecto desplegado es desde este enlace de vercel:

<https://art-world-rubenhenareshidalgo.vercel.app/>

Temporización de las tareas

El tiempo de cada archivo puede variar dependiendo de cambios por culpa de otros archivos durante el transcurso del proyecto y del transcurso de información almacenada en las fuentes:

Proyecto vite: 1 minuto máximo.

Home.jsx: 40 minutos.

i18next.jsx: 15 minutos.

LanguageToggleButton.jsx: 13 minutos.

Footer.jsx: 6 minutos.

GlobalContext.jsx: 8 horas.

SignUp.jsx: 6 horas.

Login.jsx: 4 horas.

Header.jsx: 5 horas por sessionStorage y admin.

Painting.jsx y Sculptures.jsx: 9 horas.

Users.jsx: 7 horas.

App.jsx: 40 minutos por tantos Routes y sobre todo por el AdminRoute.

Intentos de otros métodos: 7 - 9 horas.