

Memoria



Índice

Índice	1
Objetivos	2
Introducción	2
Descripción del proyecto	3
Instalaciones y Configuraciones Iniciales	3
1. Crear Proyecto con Vite y Configurar React:	3
2. Instalar y Configurar Tailwind CSS con Vite:	3
3. Añadir Material Tailwind:	4
4. Añadir Modo Oscuro (Dark Mode):	5
5. Añadir Traducción con i18next:	6
6. Añadir react-router-dom:	7
7. Mostrar iconos react-icons	8
Crear un Proyecto en Supabase y Explicación de las Tablas	9
Crear un Proyecto en Supabase	9
Tabla: Usuarios	10
Tablas de los Calzados :	11
Tabla: Compras	12
Tabla: Devoluciones	13
Estructura de Carpetas del Proyecto	14
Código detallado	16
Conclusiones	69
Fuentes	70
Anexos	71
Temporización de las tareas	71

Objetivos

Tienda de calzado donde, para ver los productos, necesitas iniciar sesión. El administrador podrá editar, eliminar productos y gestionar usuarios, además de añadir nuevo calzado.

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} */
import withMT from "@material-tailwind/react/utils/withMT.js";
export default 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:

```
// Estado para el modo oscuro
const [isDarkMode, setIsDarkMode] = useState(() => {
  return localStorage.getItem("theme") === "dark";
});

// Función para cambiar el modo
function changeDarkMode() {
  const newMode = !isDarkMode;
  setIsDarkMode(newMode);
  localStorage.setItem("theme", newMode ? "dark" : "light"); }
```

Y un botón que llame a esta función:

```
<button
  onClick={changeDarkMode}
  className="sm:h-7 sm:w-8 lg:h-7 lg:w-7 xl:h-7 xl:w-7
  bg-white dark:bg-blue-gray-800 rounded-md shadow-lg
  transition duration-150 hover:scale-105"
  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`

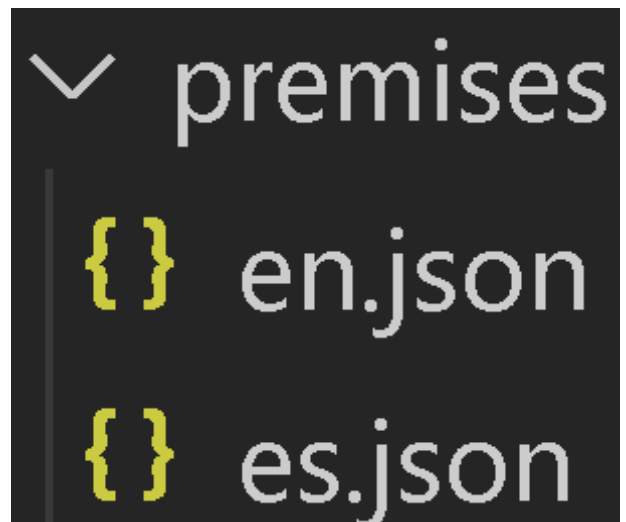
Crea un archivo `i18next.jsx` en la carpeta `src` para configurar `i18next`. Aquí puedes hacer configuraciones necesarias, e inicializar las traducciones que están en la carpeta `premises`, `en.json` para idioma Inglés, y `es.json` para idioma Castellano.

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

// Traducciones
import en from './premises/en.json';
import es from './premises/es.json';

const resources = {
  en: { translation: en },
  es: { translation: es },
};

i18n
  .use(initReactI18next) // Integrar con React
  .init({
    resources, // Usa las traducciones importadas
    lng: 'es', // Idioma predeterminado
    fallbackLng: 'en', // Idioma de respaldo
    interpolation: {
      escapeValue: false, // React ya maneja el escape de valores
    },
  })
  .export default i18n;
```



```
src > premises > {} es.json > ...
You, hace 3 semanas | 1 autor (You)
1  {
2    "Bienvenido": "Bienvenido",
3    "Creado por": "Creado por",
4    "Registro": "Registro",
5    "Ingresa tu correo y contraseña para registrarte": "Ingresa tu correo",
6    "Nombre": "Nombre",
7    "Correo Electrónico": "Correo Electrónico",
8    "Contraseña": "Contraseña",
9    "Confirmar Contraseña": "Confirmar Contraseña",
10   "Registrarse": "Registrarse",
11   "Usuario Existente": "Usuario Existente",
12   "El usuario ya está registrado. Intenta iniciar sesión": "El usuario ya",
13   "¿Ya te registraste?": "¿Ya te registraste?",
14   "Iniciar sesión": "Iniciar sesión",
15   "Las contraseñas no coinciden": "Las contraseñas no coinciden",
16   "Cerrar": "Cerrar",
17   "Registro Exitoso": "Registro Exitoso",
18
19   "Iniciar Sesión": "Iniciar Sesión",
20   "Ingresa su correo electrónico y contraseña para iniciar sesión": "Ingresa",
21   "Tu Correo": "Tu Correo",
22   "Tu Contraseña": "Tu Contraseña",
23   "¿Eres Nuevo?": "¿Eres Nuevo?",
24   "Correo o contraseña equivocada": "Correo o contraseña equivocada",
25
26   "Cerrar Sesión": "Cerrar Sesión",
```

6. 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>,
)
```

7. 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 <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 *Usuarios*, *BotasYBotinesHombre*, *BotasYBotinesMujer*, *ZapatillasHombre*, *ZapatillasMujer*, *ZapatosDeVestirHombre*, y *ZapatosDeVestirMujer*. Aquí están los detalles de cómo puedes recrear estas tablas en Supabase.

(*can_access_botasybotineshombre*,
can_access_botasybotinesmujer,
can_access_zapatillashombre,
can_access_zapatillasmujer,
can_access_zapatosdevestirhombre,
can_access_zapatosdevestirmujer).

Tabla: Usuarios

La tabla *Usuarios* 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:

<i>id</i> uuid	<i>uuid</i> uuid	<i>created_at</i> timestamp	<i>name_user</i> text	<i>role</i> text	<i>email</i> text
149b37c5-fae7-4e01-bb6e-e20cea5026b6	6c486002-e194-412c-bb61-b52ae8e6b1b	2025-02-26 20:21:31.258	Pau Ibañez	admin	pau@gmail.com
2efc6041-6121-46b7-a54e-3a07ff0bc9c5	c97b2062-2ced-4396-b4fe-1f60b4b2b126	2025-03-01 19:20:35.198	Aranu	user	aranu@gmail.com
40c56f68-aa0d-4821-b793-a248f756d868	693f8acb-5be8-4957-abad-cb33625702b	2025-03-07 16:16:19.283	Manueh	user	manueh@gmail.com

Tablas de los Calzados :

- BotasYBotinesHombre
- BotasYBotinesMujer
- ZapatillasHombre
- ZapatillasMujer
- ZapatosDeVestirHombre
- ZapatosDeVestirMujer

Estas tablas contienen información sobre sus respectivos calzados.

1. Estructura de la Tabla:

- *id (int8)*: Identificador único de cada calzado.
- *created_at (timestamp)*: Fecha y hora de creación de la entrada.
- *imagen (varchar)*: URL de la imagen.
- *nombre (varchar)*: Nombre del calzado.
- *descripcion (varchar)*: Descripción del calzado.
- *talla (varchar)*: Talla del calzado.
- *precio (varchar)*: Precio del calzado.

2. Creación de la Tabla:


 id	int8	created_at	timestamp	nombre	varchar	imagen	varchar	descripcion	varchar	precio	varchar
1		2025-03-20 11:47:45.137939		Pikolinos Vigo Militar		https://gpbcentersjgy.com/imagenes/2025/03/20/114745137939/pikolinos-vigo-militar-1.jpg		Botines de piel con cordones		115 €	
2		2025-03-20 11:50:15.122489		Levi's Jax Plus		https://gpbcentersjgy.com/imagenes/2025/03/20/115015122489/levis-jax-plus-1.jpg		Botas clásicas de cuero negro		79 €	

Tabla: Compras

La tabla *Compras* contiene información sobre diferentes compras.

1. Estructura de la Tabla:

- *id* (int8): Identificador único de cada *Compra*.
- *created_at* (timestamp): Fecha y hora de creación de la entrada.
- *uid* (uuid): Identificador del Usuario.
- *puid* (int8): Identificador de la Compra.
- *nombre* (varchar): Nombre del Calzado.
- *imagen* (varchar): *URL de la imagen*.
- *tabla_producto* (text): *Coge el tipo de Calzado en Supabase*.
- *seccion* (varchar): *Coge el tipo de Calzado dentro del Código*.
- *talla* (varchar): *Coge la talla del Calzado*.
- *precio* (varchar): *Coge el precio del Calzado*.

2. Creación de la Tabla:

id	int8	created_at	timestamp	uid	uuid	puid	int8	nombre	varchar	imagen	varchar	tabla_producto	text	seccion	varchar	talla	varchar	precio	varchar
97		2025-03-21 09:50:22.024		68fdaed1-bc64-	6			Lacoste Partner		https://gpbcntersjgy		ZapatillasHombre		Zapatillas para Hombre		38		90 €	
98		2025-03-21 09:50:40.43		68fdaed1-bc64-	8			Botines de Ante Tyler		https://gpbcntersjgy		BotasYBotinesMujer		Botas y Botines para Mujer		41		95 €	


Tabla: Devoluciones

La tabla *Devoluciones* contiene información sobre diferentes devoluciones.

1. Estructura de la Tabla:

- *id* (int8): Identificador único de cada *Devolucion*.
- *created_at* (timestamp): Fecha y hora de creación de la entrada.
- *uid* (uuid): Identificador del Usuario.
- *compra_id* (int8): Identificador de la Compra.
- *motivo* (text): Motivo de la Devolución.
- *talla* (varchar): *Coge la talla del Calzado*.
- *estado* (texto): *Indica el estado de la Devolución (Pendiente, Devuelto y Denegado)*.
- *seccion* (varchar): *Coge el tipo de Calzado dentro del Código*.
- *precio* (varchar): *Coge el precio del Calzado*.

2. Creación de la Tabla:

 id	uid	created_at	timestamp	user_id	uid	compra_id	int8	motivo	text	talla	varchar	estado	text	seccion	varchar	precio	varchar
17762c7e-ee7-4		2025-03-21 09:56:12.485583		68fdaed1-bc64-4f		97		Me equivoqué con la talla.		NULL		Pendiente		NULL		NULL	
a86b0bf2-4ed0-		2025-03-21 09:56:35.619742		68fdaed1-bc64-4f		98		Me equivoqué con el color.		NULL		Pendiente		NULL		NULL	

Estructura de Carpetas del Proyecto

```
✓ LAS ZAPAS V1
├── .vscode
├── api
├── backend
├── frontend
├── node_modules
├── public
├── src
│   ├── assets
│   ├── bd
│   ├── components
│   │   ├── hombre
│   │   ├── mujer
│   │   ├── Footer.jsx
│   │   ├── Header copy 2.jsx
│   │   ├── Header copy.jsx
│   │   ├── Header.jsx
│   │   ├── LanguageToggleButton.jsx
│   │   ├── Login.jsx
│   │   ├── Registro.jsx
│   │   └── context
│   │       ├── GlobalContext copy.jsx
│   │       ├── GlobalContext.jsx
│   │       └── premises
│   ├── views
│   │   ├── Comprar copy.jsx
│   │   ├── Comprar.jsx
│   │   ├── Devoluciones copy.jsx
│   │   ├── Devoluciones.jsx
│   │   ├── Hombre.jsx
│   │   ├── Hombreeeeeee.jsx
│   │   ├── HombrePrueba.jsx
│   │   ├── Home.jsx
│   │   ├── Mujer.jsx
│   │   ├── Perfil copy.jsx
│   │   ├── Perfil.jsx
│   │   ├── SobreNosotros.jsx
│   │   └── Usuarios.jsx
│   ├── # App.css
│   ├── App.jsx
│   └── i18n.jsx
```

```
# index.css
🌀 main.jsx
🔒 .gitignore
<> index.html
{} package-lock.json
{} package.json
JS postcss.config.js
📖 README.md
JS tailwind.config.cjs
{} vercel.json
⚡ vite.config.js
```

- **/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:

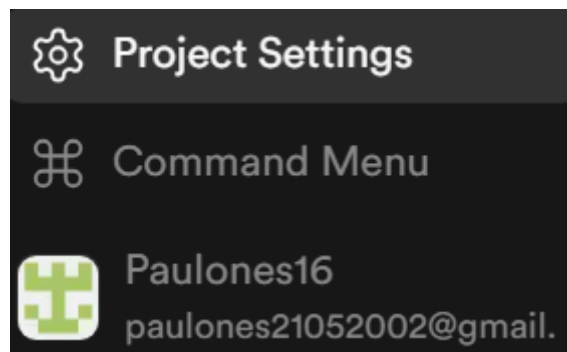
```
const supabaseUrl = import.meta.env.VITE_SUPABASE_URL; const  
supabaseKey =
```

```
import.meta.env.VITE_SUPABASE_ANON_KEY;  
const supabaseServiceKey =  
import.meta.env.SUPABASE_SERVICE_KEY;  
export const supabase = createClient(supabaseUrl, supabaseKey,  
supabaseServiceKey);
```

Todos los códigos de supabase están en la carpeta frontend y backend en su respectivo .env

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

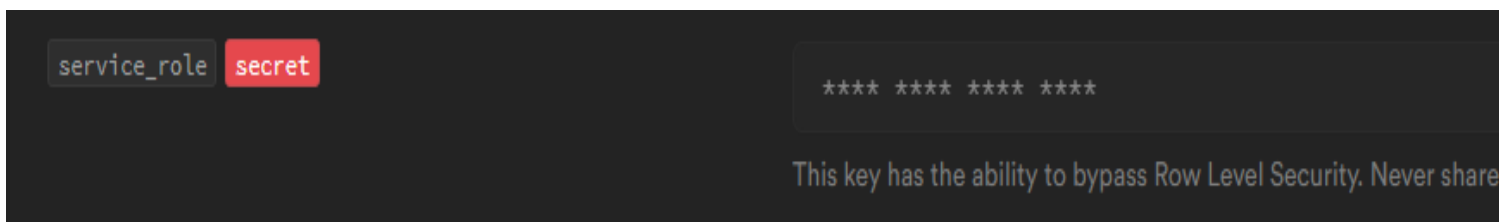
supabaseKey: Es la clave pública de mi proyecto, utilizada para interactuar con la API de Supabase, que está situada en Project Settings.



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 = '
```



```
,
```

```
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 fetchSession = async () => {  
      const { data } = await supabase.auth.getSession();  
      setSession(data.session);  
  
      if (data.session?.user) {  
        await fetchUserData(data.session.user.id);  
        await fetchCompras(data.session.user.id); // Llamar  
función aqui  
      } else {  
        setIsAdmin(false); // Por defecto, no es admin si no hay  
sesion  
        setCompras([]); // Limpiar compras si no hay sesion  
  
      }  
    };  
  
    fetchSession();  
  
    const { data: subscription } =  
supabase.auth.onAuthStateChange((_event, session) => {  
  setSession(session);  
});  
}
```

```

        if (session?.user) {
            fetchUserData(session.user.id);
            fetchCompras(session.user.id); // Actualizar compras al
cambiar usuario
        } else {
            setIsAdmin(false); // Por defecto, no es admin si no hay
sesion
            setCompras([]);
        }
    });
    return () => subscription?.unsubscribe?();
}, []);
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.

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';
```

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

2. Definición del Componente

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

```
export function LanguageToggleButton() {  
  const { i18n } = useTranslation();
```

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

```
  return (
```

```

<button
  onClick={toggleLanguage}
  className="sm:py-1 sm:px-1 lg:py-1 lg:px-2 xl:py-1 xl:px-2
text-white hover:text-blue-gray-400 hover:scale-105 active:scale-95
transition-transform duration-150 border-double border-4
border-white rounded"
>
  {i18n.language === 'es' ? 'EN' : 'ES'}
</button>
);
}

```

3. Estado y Funcionalidad

- Estado del Idioma

i18n.language: Se obtiene directamente el idioma actual sin necesidad de almacenarlo en un estado local.

4. Función toggleLanguage

```

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

```

Alternar entre 'es' y 'en'.

Usa *i18n.changeLanguage(newLanguage)* para cambiar el idioma sin necesidad de actualizar un estado local.

5. Renderizado del Botón

- Botón

```
<button
  onClick={toggleLanguage}
  className="sm:py-1 sm:px-1 lg:py-1 lg:px-2 xl:py-1 xl:px-2
  text-white hover:text-blue-gray-400 hover:scale-105
  active:scale-95 transition-transform duration-150 border-double
  border-4 border-white rounded"
>
  {i18n.language === 'es' ? 'EN' : 'ES'}
</button>
```

- **onClick:** Asigna la función *toggleLanguage* al evento *onClick* del botón.
- **Clases:** Utiliza clases de Tailwind CSS para el estilo. Las clases incluyen:
 - Espaciado dinámico: `sm:py-1 sm:px-1 lg:py-1 lg:px-2 xl:py-1 xl:px-2`.
 - Color de texto: `text-white hover:text-blue-gray-400`.
 - Efecto de escala: `hover:scale-105 active:scale-95`.
 - Transición animada: `transition-transform duration-150`.
 - Borde: `border-double border-4 border-white rounded`.

Raíz del proyecto **i18n.jsx**

1. Importaciones

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

```
// Traducciones
```

```
import en from './premises/en.json';  
import es from './premises/es.json';
```

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.

Importación de traducciones: Se cargan los archivos JSON que contienen las traducciones de inglés (en.json) y español (es.json).

2. Traducciones

```
const resources = {  
  en: { translation: en },  
  es: { translation: es },  
};
```

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

translation: Dentro de cada idioma, se carga el contenido del archivo JSON correspondiente.

3. Inicialización de *i18next*

i18n

```
.use(initReactI18next) // Integrar con React

.init({
  resources, // Usa las traducciones importadas
  lng: 'es', // Idioma predeterminado
  fallbackLng: 'en', // Idioma de respaldo
  interpolation: {
    escapeValue: false, // React ya maneja el escape de valores },
  });
```

- **.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).
- **fallbackLng**: 'en': Si no se encuentra una traducción en español, usa el inglés como respaldo.
- **interpolation.escapeValue**: Configurado en *false* porque React ya se encarga del escape de valores por motivos de seguridad.

4. Uso en la actualización de perfil (handleUpdateProfile)

```
const handleUpdateProfile = async () => {  
  if (!email.trim() || !nombre.trim()) {  
    return showAlert(t("Todos los campos son obligatorios"), "red"); }  
  
  if (!session?.user?.id) {  
    return showAlert(t("Error: Usuario no identificado"), "red"); }  
  
  try {  
    const response = await  
    fetch("https://las-zapass.vercel.app/api/update user", {  
      method: "POST",  
      headers: { "Content-Type": "application/json" },  
      body: JSON.stringify({  
        userId: session.user.id,  
        nombre,
```

```
email,  
  
password: password || undefined, // Solo lo manda si hay contraseña  
}),  
});  
  
const result = await response.json();  
  
if (!response.ok) {  
  throw new Error(result.error || "Error al actualizar perfil");  
}  
  
// Actualizar sesión con el nuevo correo y nombre  
setSession((prevSession) => ({  
  ...prevSession,  
  user: {  
    ...prevSession.user,  
    email,  
    user_metadata: { ...prevSession.user.user_metadata, name: nombre },  
  },  
}));  
  
showAlert(t("Perfil actualizado"), "green");  
  
} catch (error) {  
  console.error("Error al actualizar el perfil:", error);  
}
```

```
showAlert(t("Error al actualizar el perfil"), "red");  
  
}  
  
};
```

A. Verificación de datos:

- i. Si email o nombre están vacíos, se muestra un mensaje de error con t("Todos los campos son obligatorios").
- ii. Si session.user.id no existe, se muestra t("Error: Usuario no identificado").

B. Petición fetch a la API:

- i. Se envía una solicitud POST a <https://las-zapass.vercel.app/api/update-user>.
- ii. Si session.user.id no existe, se muestra t("Error: Usuario no identificado").

C. Manejo de la respuesta:

- i. Si la API devuelve un error, se muestra t("Error al actualizar perfil").
- ii. Si la actualización es exitosa, se actualiza la sesión del usuario con el nuevo nombre y correo.

- iii. Se muestra el mensaje `t("Perfil actualizado")` en verde.

5. 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:

```
import { useTranslation } from "react-i18next";  
  
const { t } = useTranslation();
```

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

t("Clave de traducción"): Busca el texto traducido según el idioma actual.

/views Home.jsx

1. Importaciones

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

Button y Dialog: Componentes de *@material-tailwind/react* utilizados para los botones y ls modals de imagenes e inputs.

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 './App.css';
import { Route, Routes } from 'react-router-dom';
import { Header } from './components/Header';
import { Home } from './views/Home';
import { Footer } from './components/Footer';
import { Hombre } from './views/Hombre';
import { Mujer } from './views/Mujer';
import { Login } from './components/Login';
```

```
import { Registro } from './components/Registro';
import { Usuarios } from './views/Usuarios';
import { ZapatosDeVestirHombre } from
'./components/hombre/ZapatosDeVestirHombre';
import { BotasHombre } from './components/hombre/BotasHombre';
import { ZapatosDeVestirMujer } from
'./components/mujer/ZapatosDeVestirMujer';
import { BotasMujer } from './components/mujer/BotasMujer';
import { ZapatillasMujer } from './components/mujer/ZapatillasMujer';
import { ZapatillasHombre } from
'./components/hombre/ZapatillasHombre';
import { Comprar } from './views/Comprar';
import { Perfil } from './views/Perfil';
import { Devoluciones } from './views/Devoluciones';
import { SobreNosotros } from './views/SobreNosotros';
```

- **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.
- **Vistas Importadas:** Se importan varias vistas desde diferentes carpetas del proyecto, como Comprar, Home, Usuarios, SobreNosotros, Hombre, Mujer, Perfil, y Devoluciones.
- **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. Componente principal App

```
export default function App() {  
  return (  
    <>  
    <div className="bg-gray-100 dark:bg-blue-gray-900 min-h screen">  
      <Header/>  
      { /* Enrutamiento */ }  
      <Routes>  
        <Route path="/" element={ <Home /> } />  
        <Route path="/registro" element={ <Registro /> } /> <Route  
path="/login" element={ <Login /> } />  
        <Route path="/hombre" element={ <Hombre /> } /> <Route  
path="/botasHombre" element={ <BotasHombre /> } /> <Route  
path="/zapatillasHombre"  
element={ <ZapatillasHombre /> } />  
        <Route path="/zapatosHombre"  
element={ <ZapatosDeVestirHombre /> } />  
        <Route path="/botasHombre" element={ <BotasHombre /> } />  
        <Route path="/mujer" element={ <Mujer /> } />  
        <Route path="/botasMujer" element={ <BotasMujer /> } /> <Route  
path="/zapatillasMujer" element={ <ZapatillasMujer /> } />  
        <Route path="/zapatosMujer"  
element={ <ZapatosDeVestirMujer /> } />  
        <Route path="/usuarios" element={ <Usuarios /> } /> <Route  
path="/perfil" element={ <Perfil /> } />  
        <Route path="/perfil/:id" element={ <Perfil /> } /> <Route  
path="/devoluciones" element={ <Devoluciones /> } /> <Route
```

```

path='/comprar/:tableName/:nombre'
element={<Comprar />} />
<Route path='/sobreNosotros' element={<SobreNosotros />} />
<Route path='/footer' element={<Footer />} /> </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="/perfil/:id"**
 - **element={<Perfil />} />**: Vista dinámica de perfil con un id. En la vista Usuarios, puedes acceder a los perfiles mediante a su id.
- **<Footer />**: Renderiza el componente *Footer* en la parte inferior de la página.

/componentes Registro.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*, Input para la UI.

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.

Contexto global: *useGlobalContext* para el control de popups.

Traducción: *useTranslation* para internacionalización.

2. Estados

```
const { t } = useTranslation();
const { activePopup, openPopup } = useGlobalContext(); const
navigate = useNavigate();
const [showPopup] = useState(false);
```

```
const [showUserExistsPopup, setShowUserExistsPopup] =  
useState(false);
```

```
const [showErrorPopup, setShowErrorPopup] = useState(false); const  
[showSuccessPopup, setShowSuccessPopup] = useState(false); const {  
setErrorMessage } = useState(false);
```

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

activePopup, openPopup: Estado global para el control de popups.

showUserExistsPopup: Controla la visibilidad del popup de usuario ya registrado.

showErrorPopup: Muestra un mensaje de error si las contraseñas no coinciden.

showSuccessPopup: Indica que el registro fue exitoso. **dialogData:** Contiene los datos del formulario (nombre, email, contraseña y confirmación).

3. Eventos

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

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

4. Registro

```
async function handleSubmit(e) {
  e.preventDefault();
  if (dialogData.password !== dialogData.confirmPassword) {
    setErrorMessage(t('Las contraseñas no coinciden'));
    setShowErrorPopup(true);
    return;
  }
  try {
    const { data, error } = await supabase.auth.signUp({ email:
dialogData.email,
password: dialogData.password,
options: {
  data: { name: dialogData.name },
},
});
    if (error) {
      if (error.message.includes("User already registered")) {
        setShowUserExistsPopup(true);
        return;
      }
      throw error;
    }
    const uid = data.user.id;
```

```
const role = data.user.email ===  
"rubenhenareshidalgo97@gmail.com" ? "admin" : "user"; const {  
error: profileError } = await  
supabase.from("Usuarios").insert([  
  {  
    uid,  
    email: data.user.email,  
    name_user: dialogData.name,  
    role,  
    created_at: new Date(),  
  },  
]);  
if (profileError) throw profileError;  
openPopup(false);  
setShowSuccessPopup(true);  
setTimeout(() => {  
  setShowSuccessPopup(false);  
  navigate('/');  
}, 2500);  
} catch (error) {  
  setErrorMessage(error.message);  
  setShowErrorPopup(true);  
}  
}
```

- **handleSubmit:** Maneja el envío del formulario de registro.
 - **Valida que las contraseñas coincidan** antes de enviar los datos.

- **Registra el usuario en *Supabase*** y lo guarda en la base de datos.
- **Maneja errores** y muestra popups según la situación.
- **Redirige al usuario a la página principal** tras el registro exitoso.

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

(Simetría ya vista anteriormente, no hace falta que lo explique el código de nuevo)

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

2. Estado

```
const { t } = useTranslation();
const { activePopup, openPopup, setSession } = useGlobalContext();
const handleOpen = () => openPopup("login");
let navigate = useNavigate();
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')) {
      openPopup('loginError');
    } 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 un popup 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 { useEffect, useState } from "react";
import { Button } from "@material-tailwind/react";
import { useGlobalContext } from "../context/GlobalContext"; import {
Login } from "./Login";
import { Link, useLocation, useNavigate } from "react-router-dom";
import { ImContrast } from "react-icons/im";
import { LanguageToggleButton } from "./LanguageToggleButton";
import { useTranslation } from "react-i18next";
```

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

2. Estado y Funciones

```
const { t } = useTranslation();
const { isButtonDisabled, handleButtonClick, session, setSession,
isAdmin, setIsAdmin, openPopup, logout, loadingUser } =
useGlobalContext();
const [isMenuOpen, setIsMenuOpen] = useState(false); let navigate =
useNavigate();
const location = useLocation();
const [isDarkMode, setIsDarkMode] = useState(() =>
localStorage.getItem("theme") === "dark");
useEffect(() => {
  if (isDarkMode) {
```

```
document.documentElement.classList.add("dark"); } else {
document.documentElement.classList.remove("dark"); }
}, [isDarkMode]));
function changeDarkMode() {
const newMode = !isDarkMode;
setIsDarkMode(newMode);
localStorage.setItem("theme", newMode ? "dark" : "light"); }
async function handleLogout() {
openPopup(null);
await logout();
setSession(null);
setIsAdmin(false);
navigate("/");
}
function handleLoginClick() {
setIsMenuOpen(false);
handleButtonClick(() => {
openPopup("login");
});
}
const menuRoutes = [
{ path: "/hombre", label: t('Hombre') },
{ path: "/mujer", label: t('Mujer') }
];
```

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 Hombre.jsx:

1. Importaciones

```
import { Button, Dialog } from "@material-tailwind/react";  
import { useGlobalContext } from "../context/GlobalContext";  
import { Link } from "react-router-dom";  
import { useTranslation } from "react-i18next";
```

Botones y diálogos (*@material-tailwind/react*).

useGlobalContext para manejar el estado global.

Navegación (*react-router-dom*) y **traducción** (*i18next*).

2. Estado y Funciones

```
const { t } = useTranslation();  
const { handleOpen, activePopup, openPopup, session } =  
useGlobalContext();  
const openLoginForCategory = (category) => {  
  handleOpen(null, category);  
  openPopup("login");  
};
```

openLoginForCategory: Abre el popup de login antes de mostrar la categoría.

3. Renderización

```

return (
  <div className="min-h-screen flex flex-col items-center py-16
bg-gradient-to-bl from-gray-200 dark:from-gray-800">
    <h2 className="text-4xl md:text-5xl font-extrabold mb-8 mt-8
text-center dark:text-white">
      {t('Bienvenido a Las Zapas')}
    </h2>
    <p className="text-lg text-center max-w-2xl w-[600px] mb-12
dark:text-gray-200 text-gray-700">
      {t('Descubre nuestra colección de zapatillas para todas las
ocasiones.')}
    </p>

    <div className="grid md:grid-cols-2 gap-20 px-4">
      {[ "Hombres", "Mujeres"].map((categoria) => {
        const imgSrc = categoria === "Hombres" ? "/3perfecto1.png" :
"/zvm11.png";
        const link = categoria === "Hombres" ? "/hombre" : "/mujer";
        const categoryKey = categoria.toLowerCase();
        return (
          <div key={categoria} className="bg-white dark:bg-gray-800
shadow-lg rounded-lg overflow-hidden group w-[500px]
md:w-[300px] lg:w-[500px] xl:w-[610px]">
            <button className="w-full transition hover:scale-105"
onClick={() => handleOpen(null, `zapato${categoria}`)}>

```



```

        <img src={imgSrc} alt={categoria} className="w-full h-56
object-cover" />
    </button>
    <Dialog size="xs" open={activePopup ===
`zapato${categoria}`} handler={() => openPopup(null)}
className="bg-transparent shadow-none">
        <img src={imgSrc} alt={`zapato${categoria}`}
className="w-full mb-4 rounded-md" />
    </Dialog>
    <div className="p-4 text-center">
        <h3 className="text-2xl font-bold mb-2
dark:text-white">{t(categoria)}</h3>
        <p className="text-gray-600 dark:text-gray-100
font-semibold mb-4">
            {t(categoria === "Hombres" ? "Estilo y comodidad en cada
paso." : "Diseños elegantes para cualquier ocasión.")}
        </p>
        {session ? (
            <Link to={link}>
                <Button size="sm" color="blue">{t("Ver Más")}</Button>
            </Link>
        ) : (
            <Button size="sm" color="blue" onClick={() =>
openLoginForCategory(`zapato${categoria}`)}>{t("Ver
Más")}</Button>
        )}
    </div>
</div>

```

```

    );
  }}}
</div>
</div>
);

```

/views Mujer.jsx

Es lo mismo que en la vista Hombre pero para Mujer.

/components /hombre BotasHombre.jsx

1. Importaciones

```

import { Dialog, Card, CardBody, CardFooter, Input, Typography,
Button } from '@material-tailwind/react';
import { useEffect } from "react";
import { useGlobalContext } from '../..context/GlobalContext';
import { useNavigate } from "react-router-dom";
import { useTranslation } from 'react-i18next';
import { MdAddToPhotos } from 'react-icons/md';

```

2. Estados y Funciones

```

export function BotasHombre() {
  const { t } = useTranslation();
  const {
    fetchTableData, zapass, setZapass, activePopup, openPopup,
    editData,

```

```

    handleOpenEdit, deleteTableData, newZapatoBota,
    setNewZapatoBota,
    handleOpenPut, handleSubmit, errorSubmit, handleChange,
    isAdmin
  } = useGlobalContext();
  const navigate = useNavigate();

  useEffect(() => {
    fetchTableData("BotasYBotinesHombre").then(setZapass);
  }, [fetchTableData]);

```

Carga de datos simplificada.

Funciones agrupadas para mejor *legibilidad*.

3. Renderización de la Lista

```

return (
  <div className="min-h-screen bg-gradient-to-bl from-gray-200
dark:from-gray-800">
    <div className="container mx-auto py-20 pb-16">
      <h1 className="text-3xl font-bold mt-14 mb-4
dark:text-white">{t('Botas y Botines para Hombre')}</h1>
      {zapass.length === 0 ? (
        <p className="text-blue-gray-600
dark:text-blue-gray-100">{t('No hay botas disponibles')}</p>
      ) : (
        <div className="grid grid-cols-1 sm:grid-cols-2 md:grid-cols-3
lg:grid-cols-4 gap-6">

```

```

    {zapass.map((zapatoBota) => (
      <div key={zapatoBota.id} className="bg-gray-200
dark:bg-gray-800 p-4 rounded-lg shadow-md flex flex-col
cursor-pointer"
      onClick={() =>
navigate(`/comprar/BotasYBotinesHombre/${zapatoBota.nombre}`
)}}>

        <img src={zapatoBota.imagen ||
"https://via.placeholder.com/150"} alt={zapatoBota.nombre}
className="w-full h-48 object-cover mb-4 rounded-md
hover:scale-105 transition"/>

        <h2 className="text-xl font-semibold mb-2
dark:text-white">{zapatoBota.nombre}</h2>

        <p className="text-blue-gray-600 dark:text-blue-gray-100
mb-2">{zapatoBota.descripcion}</p>

        <p className="text-blue-gray-600
dark:text-blue-gray-100">{zapatoBota.precio}</p>

        {isAdmin && (
          <div className="mt-4 flex justify-between">
            <Button size="sm" color="blue" onClick={(e) => {
e.stopPropagation(); handleOpenEdit("BotasYBotinesHombre",
zapatoBota); }}>{t('Editar')}</Button>

            <Button size="sm" color="red" onClick={(e) => {
e.stopPropagation(); deleteTableData("BotasYBotinesHombre",
zapatoBota.id); }}>{t('Borrar')}</Button>

          </div>
        )}
      </div>
    ))
  </div>

```

```
    )})
  </div>
})
```

Código más compacto y legible.

Menos repetición en los estilos y estructura.

4. Botón de añadir (*Admin*)

```
{isAdmin && (
  <div className="fixed bottom-20 right-4">
    <Button onClick={handleOpenPut} variant="gradient">
      <MdAddToPhotos className="text-lg xl:hidden"/>
      <span className="hidden xl:inline">{t('Añadir Zapatos de
Vestir')}}</span>
    </Button>
  </div>
)}
```

Optimizado para pantallas pequeñas.

5. Dialog de Añadir y Editar

```
<Dialog open={activePopup === "newZapatoBota" || activePopup
=== "editZapatoBota"} handler={openPopup} size="xs"
className="bg-transparent shadow-none">
  <Card className="dark:bg-blue-gray-900 dark:text-white
mx-auto w-full max-w-[24rem]">
```

```

    <form onSubmit={e => { e.preventDefault();
handleSubmit("BotasYBotinesHombre", newZapatoBota); }}>
      <CardBody className="flex flex-col gap-4">
        <Typography variant="h4">{editData ? 'Editar Bota' : 'Añadir
Nueva Bota'}</Typography>
        {[ "nombre", "imagen", "descripcion", "precio"].map((campo)
=> (
          <Input key={campo} label={t(campo)} size="lg"
color="blue-gray" name={campo} required
value={newZapatoBota[campo]} onChange={e =>
campo === "precio"
? setNewZapatoBota((prev) => ({ ...prev, precio:
e.target.value.replace(/\D/g, "") + " €" }))
: handleChange(e)}
className="dark:text-gray-300"/>
        )))
      </CardBody>
      <CardFooter className="pt-0">
        {errorSubmit && <Typography variant="small" color="red"
className="text-center">{errorSubmit}</Typography>}
        <Button variant="gradient" fullWidth
type="submit">{editData ? t('Actualizar Botas') : t('Añadir
Botas')}</Button>
      </CardFooter>
    </form>
  </Card>
</Dialog>
</div>

```

```
    </div>  
  );  
}
```

Uso de .map() para reducir código repetitivo.

Manejo de precio directamente en *onChange*

/components /hombre y /mujer

Dentro de las carpetas hombre y mujer hay ***ZapatillasHombre.jsx***, ***ZapatillasMujer.jsx***, ***ZapatosDeVestirHombre.jsx***, ***ZapatosDeVestirMujer.jsx***, ***BotasHombre.jsx*** y ***BotasMujer.jsx***. Ya expliqué anteriormente el código de ***BotasHombre.jsx***, y es exactamente el **mismo** a los otros jsx mencionados pero cambiando el nombre de las tablas.

/views Perfil.jsx

1. Importaciones

```
import { useState, useEffect } from "react";  
import { useGlobalContext } from "../context/GlobalContext";  
import { Dialog, Button, Typography, Input, Card, CardBody, Alert }  
from "@material-tailwind/react";  
import { supabase } from "../bd/supabase";  
import { useTranslation } from "react-i18next";  
import { useParams } from "react-router-dom";
```

React Hooks (*useState*, *useEffect*): Manejo de estado y efectos secundarios.

useGlobalContext: Acceso al contexto global de la aplicación.

Componentes de *@material-tailwind/react*: Interfaz de usuario (botones, tarjetas, diálogos, inputs, alertas).

supabase: Base de datos para gestionar datos de usuario y compras.

useTranslation: Traducción de textos.

useParams: Obtención de parámetros de la URL.

2. Estado y Funciones

```
const { t } = useTranslation();
const { compras, setCompras, session, setSession, fetchCompras,
fetchUserData, setError } = useGlobalContext();
const [selectedCompra, setSelectedCompra] = useState(null);
const [showModal, setShowModal] = useState(false);
const [motivo, setMotivo] = useState("");
const [view, setView] = useState(() =>
localStorage.getItem("perfilView") || "compras");
const [showMotivoInput, setShowMotivoInput] = useState(false);
const [loading, setLoading] = useState(false);
const [errorMotivo, setErrorMotivo] = useState("");
const [alertMessage, setAlertMessage] = useState(null);
const [devoluciones, setDevoluciones] = useState({});
const [cancelCompraId, setCancelCompraId] = useState(null);
const [email, setEmail] = useState(session?.user?.email || "");
const [nombre, setNombre] = useState("");
const [password, setPassword] = useState("");
```



```
const [usuario, setUsuario] = useState(null);  
const { id } = useParams();  
const userId = id || session?.user?.id;
```

view: Guarda en *localStorage* la vista seleccionada (*compras* o *editar*).

usuario: Almacena datos del usuario, ya sea de la sesión o de la URL.

devoluciones: Almacena el estado de devoluciones de compras.

selectedCompra: Almacena la compra seleccionada para gestionar devoluciones.

showModal: Controla la visibilidad del modal.

motivo y showMotivoInput: Manejan la lógica para solicitar devoluciones.

3. Efectos secundarios

Cargar compras del usuario:

```
useEffect(() => {  
  if (!userId) return;  
  if (userId !== session?.user?.id) {  
    setCompras([]); // Limpia si no es el perfil propio  
  }  
  
  fetchCompras(userId);  
}, [userId]);
```

Si el *userId* cambia, se actualiza la lista de compras.

Obtener datos del usuario:

```
useEffect(() => {
  setUsuario(null);

  const fetchUsuario = async () => {
    try {
      if (!id) {
        setUsuario({
          uid: session?.user?.id,
          name_user: session?.user?.user_metadata?.name || "Usuario",
        });
        return;
      }

      let userId = id;
      if (!id.match(/^[0-9a-fA-F-]{36}$/)) {
        const { data, error } = await supabase
          .from("Usuarios")
          .select("uid, name_user")
          .eq("name_user", id)
          .single();
        if (error) throw error;
        userId = data.uid;
      }

      const { data, error } = await supabase
        .from("Usuarios")
        .select("uid, name_user")
```

```
.eq("uid", userId)
.single();

if (error) throw error;
setUsuario(data);
window.history.replaceState(null, "",
`/perfil/${data.name_user}`);
} catch (error) {
  console.error("Error fetching user:", error.message);
  setError(error.message);
}
};

fetchUsuario();
}, [id, session]);
```

Obtiene el usuario por *id* o *name_user*.
Si el *id* no es un UUID, lo busca por nombre de usuario.
Actualiza la URL sin recargar la página.

Obtener devoluciones del usuario:

```
useEffect(() => {
  const fetchDevoluciones = async () => {
    const { data } = await
supabase.from("Devoluciones").select("*").eq("user_id",
session?.user?.id);
    const devolMap = {};
```

```
data?.forEach((d) => (devolMap[d.compra_id] = d));  
setDevoluciones(devolMap);  
};  
if (session?.user?.id) fetchDevoluciones();  
}, [session]);
```

Recupera las devoluciones asociadas al usuario.

4. Funciones

Actualizar perfil:

```
const handleUpdateProfile = async () => {  
  if (!email.trim() || !nombre.trim()) {  
    return showAlert(t("Todos los campos son obligatorios"), "red");  
  }  
  
  try {  
    const response = await  
fetch("https://laszapas.vercel.app/api/update-user", {  
  method: "POST",  
  headers: { "Content-Type": "application/json" },  
  body: JSON.stringify({ userId: session.user.id, nombre, email,  
password: password || undefined }),  
});  
  
    const result = await response.json();
```

```
if (!response.ok) throw new Error(result.error || "Error al actualizar perfil");
```

```
setSession((prevSession) => ({
  ...prevSession,
  user: {
    ...prevSession.user,
    email,
    user_metadata: { ...prevSession.user.user_metadata, name:
nombre },
  },
}));
```

```
await fetchUserData(session.user.id);
showAlert(t("Perfil actualizado"), "green");
} catch (error) {
  showAlert(t("Error al actualizar el perfil"), "red");
}
};
```

Valida los datos antes de enviarlos.

Envía una petición para actualizar los datos del usuario.

Actualiza el *session* y recarga los datos del usuario.

Solicitar devolución:

```
const handleEnviarDevolucion = async () => {
```

```

    if (!motivo.trim()) return setErrorMotivo(t("Por favor, ingresa un
motivo"));

    try {
      const { error } = await supabase.from("Devoluciones").insert([
        { user_id: session.user.id, compra_id: selectedCompra.id, motivo,
estado: "Pendiente" }
      ]);

      if (error) throw error;
      showAlert(t("Devolución enviada"), "green");
      setShowModal(false);
      setDevoluciones({ ...devoluciones, [selectedCompra.id]: { motivo,
estado: "Pendiente" } });
    } catch (error) {
      showAlert(t("Error al solicitar devolución"), "red");
    }
  };

```

Verifica que haya un motivo antes de solicitar la devolución.

Inserta un registro en la tabla *Devoluciones*.

5. Renderizado

```

return (
  <div className="min-h-screen bg-gradient-to-bl from-gray-200
dark:from-gray-800 p-6 pt-24 pb-20 flex justify-center items-center">
    <div className="max-w-3xl w-full bg-white dark:bg-gray-800
shadow-2xl rounded-lg p-8 border border-gray-200
dark:border-gray-600">

```

```

<h1 className="text-3xl font-semibold">{t('Perfil')}</h1>

<button onClick={() => setView("compras")}>{t("Mis
Compras")}</button>
<button onClick={() =>
setView("editar")}>{t("Editar")}</button>

{view === "editar" ? (
  <div>
    <Input label={t("Nombre")} value={nombre} onChange={(e)
=> setNombre(e.target.value)} />
    <Button onClick={handleUpdateProfile}>{t("Guardar
cambios")}</Button>
  </div>
) : (
  <p>{t("Historial de compras")}</p>
)}
</div>
</div>
);

```

/views Usuarios.jsx

1. Importaciones

```
import { useEffect, useState } from "react";
import { useGlobalContext } from "../context/GlobalContext"; import {
  Button } from "@material-tailwind/react";
import { useTranslation } from "react-i18next";
import { supabase } from "../bd/supabase";
import { useNavigate } from "react-router-dom";
```

2. Estado y Funciones

```
const { t } = useTranslation();
const { setError } = useGlobalContext();
const navigate = useNavigate();
const [usuarios, setUsuarios] = useState([]);
const [editingUsers, setEditingUsers] = useState({});
```

usuarios: Estado que almacena la lista de usuarios obtenidos de Supabase.

editingUsers: Estado que almacena los valores de edición de cada usuario.

3. Obtener Usuarios

```
const fetchUsuarios = async () => {
  try {
    const { data, error } = await
    supabase.from("Usuarios").select("*");
```



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

useEffect(() => {
  fetchUsuarios();
}, []);
```

fetchUsuarios(): Obtiene todos los usuarios de la tabla ***Usuarios*** y actualiza el estado *usuarios*.

4. Actualizar

```
const updateUser = async (id, updates) => {
  try {
    const { error } = await
supabase.from("Usuarios").update(updates).eq("id", id); if (error)
throw error;
    setUsuarios((prev) =>
prev.map((user) => (user.id === id ? { ...user, ...updates } : user))
);
    setEditingUsers((prev) => ({ ...prev, [id]: undefined })); } catch
(error) {
    console.error("Error updating user:", error.message);
    setError(error.message);
  }
};
```

updateUser (id, updates): Actualiza los datos de un usuario en Supabase.

Si la actualización es exitosa, se actualiza el estado usuarios y se finaliza la edición.

5. Eliminar

```
const deleteUser = async (id) => {
  try {
    const { error } = await
    supabase.from("Usuarios").delete().eq("id", id);
    if (error) throw error;
    setUsuarios((prev) => prev.filter((user) => user.id !== id)); } catch
    (error) {
      console.error("Error deleting user:", error.message);
      setError(error.message);
    }
  };
};
```

deleteUser (id): Elimina un usuario de la base de datos y actualiza el estado usuarios.

6. Editar

```
return (
  <div className="min-h-screen bg-gradient-to-bl from-gray-200
  dark:from-gray-800">
    <div className="container mx-auto py-20 pb-16"> <h1
    className="dark:text-white text-blue-gray-800 text-3xl font-bold
    mb-4 mt-14">
```

```

    {t("Gestión de Usuarios")}
  </h1>
  <div className="overflow-x-auto bg-white shadow-md rounded-lg">
    <table className="w-full table-auto text-left text-sm text gray-500
dark:text-gray-300">
      <thead>
        <tr>
          <th>{t("Nombre")}</th>
          <th>{t("Correo")}</th>
          <th>{t("Rol")}</th>
          <th>{t("Acciones")}</th>
        </tr>
      </thead>
      <tbody>
        {usuarios?.map((user) => (
          <tr key={user.id}>
            <td>
              {editingUsers[user.id] !== undefined ? ( <input
type="text"
value={editingUsers[user.id]} onChange={(e) =>
setEditingUsers((prev) => ({ ...prev, [user.id]: e.target.value })))
              }
              onKeyDown={(e) =>
                e.key === "Enter" &&
                updateUser(user.id, { name_user: editingUsers[user.id] })
              }
            </td>
          </tr>
        ))}
      </tbody>
    </table>
  ) : (

```

```

<button onClick={() => navigate(`/perfil/$ {user.uid}`)}>
  {user.name_user}
</button>
)}
</td>
<td>{user.email}</td>
<td>
<span>{t(`roles.${user.role}`)}</span> </td>
<td>
<Button onClick={() => deleteUser(user.id)}
>{t("Eliminar")}</Button>
{editingUsers[user.id] !== undefined ? ( <>
  <Button onClick={() => updateUser(user.id, { name_user:
editingUsers[user.id] })}>{t("Guardar")}</Button> <Button
onClick={() =>
setEditingUsers((prev) => ({ ...prev, [user.id]: undefined })))}
>{t("Cancelar")}</Button>
</>
): (
  <Button onClick={() => setEditingUsers((prev) => ({ ...prev,
[user.id]: user.name_user })))}>{t("Editar Nombre")} </Button>
)}
  <Button onClick={() => updateUser(user.id, { role: user.role ===
"admin" ? "user" : "admin" })}> {user.role === "admin" ? t("Convertir
a Usuario") : t("Hacer Admin")}
</Button>
</td>
</tr>

```

```
    }}  
  </tbody>  
</table>  
</div>  
</div>  
</div>  
);
```

El componente retorna una tabla donde se listan los usuarios con opciones para editar, eliminar y cambiar el rol.

Conclusiones

El proyecto lo hice por mi mismo y gracias a las teconologías IA como chatgpt, es increíble como evoluciona todo que hasta una ia se puede transformar en mi profesor particular. El código lo hice durante 5 meses, he intentado que no haya ningún error en cada vista, componente, context, etc del proyecto, y lo logré, estoy muy feliz por ello. Pienso que si me hubiese puesto hace años a hacer el proyecto, hubiese tardado menos porque llevaba mucho tiempo sin programar y necesitaba refrescar mucho código, sobretodo backend. Pero hay que mirar el lado bueno, y ya por fin acabé el proyecto final, además el tema seleccionado que es la moda del calzado siempre me fascinó mucho.

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/>

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 los Calzados:

<https://www.nike.com/es/>

<https://www.elcorteingles.es/>

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/Las-Zapas>

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

<https://las-zapass.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 y configuración

- Proyecto Vite: **1 minuto máximo**

Componentes principales

- Home.jsx: **40 minutos**
- i18next.jsx: **15 minutos**
- LanguageToggleButton.jsx: **13 minutos**
- Footer.jsx: **6 minutos**
- GlobalContext.jsx: **8 horas**

Autenticación y usuarios

- Registro.jsx: **6 horas**
- Login.jsx: **4 horas**
- Usuarios.jsx: **7 horas**

- Perfil.jsx: **5 horas**

Navegación y administración

- Header.jsx: **5 horas**
- SobreNosotros.jsx: **10 minutos**
- Compras.jsx: **4 horas**
- Devoluciones.jsx: **4 horas**

Secciones de productos

- Hombre.jsx: **6 horas**
- Mujer.jsx: **6 horas**
- Botas.jsx (Hombre y Mujer): **3 horas c/u**
- Zapatillas.jsx (Hombre y Mujer): **3 horas c/u**
- ZapatosDeVestir.jsx (Hombre y Mujer): **3 horas c/u**

Integraciones

- premises (En.json, Es.json): **2 horas**
- **backend**
 - delete-user.js: **4 horas**
 - updateuser.js: **4 horas**
 - supabaseBackend.js: **8 horas**
- **frontend**
 - supabase.js: **7 horas**

Pruebas e intentos adicionales

- Intentos de otros métodos: **7 - 9 horas**