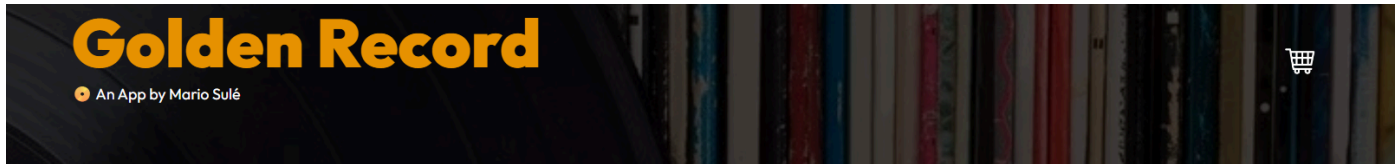


Golden Record

Project Documentation - Mario Sulé Domínguez



<https://github.com/mariosulee/Golden-Record.git>



Vinyl records for sale

RECKLESS

By Bryan Adams



"Reckless" is a melodic rock album that channels high energy and unrestrained emotion. Its songs capture the thrill of youth, boldness, and living without limits, blending powerful guitar riffs and heartfelt lyrics.

19.99 €

ADD TO CART

SWIMMING

By Mac Miller



"Swimming" is Mac Miller's fifth studio album. It explores themes of self-reflection, growth, and struggle. It blends hip-hop, jazz, and soul influences, capturing his journey through life, love, and mental health.

49.99 €

ADD TO CART

DAWN FM

By The Weeknd



This synth-electronic album guides listeners through reflection on life, regrets, and acceptance. Its immersive production conveys a surreal journey toward light and resolution.

13.99 €

ADD TO CART

The developed project consists of the implementation of an application based on a virtual vinyl record store built with React, where the available albums are displayed, each with different attributes such as title, artist, or price, and even a description of the vinyl records for sale along with a button to add each of them to the cart. Users can thus add albums to their shopping cart, edit quantities, see the total price update in real time, and also empty the cart if they wish, providing a complete online shopping experience in a dynamic and interactive way, featuring a modern and responsive design that is easy to use on any device.

This documentation aims to clearly and thoroughly explain how the project is structured and how it works, including both the main functionality and the technical structure (folders, files, components, React Hooks, props, styles, Vite configuration, and dependencies). The goal is for anyone reading the documentation to understand how the application has been built, how the components interact, and how the project runs, without having to analyze or explain all the code, serving as a guide for both maintenance and future improvements or learning.

To create this project, Node.js and NPM must first be installed. Node.js is an environment that allows JavaScript code to be executed outside the browser, usually on the server, while NPM is its package manager, which allows installing dependencies and libraries. The project will be created using Vite, which is a development tool for web projects that works with React.

Regarding the project structure:

- The **dist** folder contains the final optimized code ready for production of the React app, which is generated after running **npm run build**.
- **node_modules** is the folder where all dependencies are stored.
- **public** is a folder where images are stored that will be accessible to visitors of the web application.
- In **src**, all the code and the functional part of the application can be found. It includes the components folder (with the Album and Header components) and the data folder (which contains the simulated JavaScript database represented by an array of objects), the root component App.jsx, and main.jsx (which renders this main App component in the index.html). There is also an index.css stylesheet created with Bootstrap.
- **eslint.config.js** defines rules for how the JS code should be written and is not modified.
- **.gitignore** is used to tell Git which files or folders should not be tracked or uploaded to the repository.
- **package-lock.json** is generated from package.json, which contains the project and development dependencies.
- Finally, **vite.config.js** is the Vite configuration file.

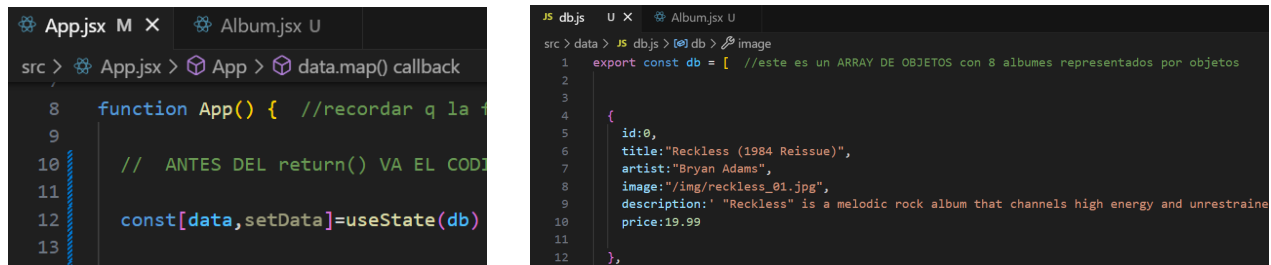
All the albums are placed in an array, iterated over, and then the information is displayed in each component dynamically using React Hooks, which allow using React-specific functions in components.

To view the components tab when inspecting code in the browser (in my case, Google Chrome), the React Developer Tools extension is installed. This shows the React component tree and is used to see values, such as the State, without constantly using console.log.

A data folder is then created, and inside it, the db.js file is added (this will simulate the database with the information and attributes of each album). This file consists of an array of objects, with each object representing an album available in the virtual store. Each object has attributes such as title, artist, price, description, image, and an identifier.

1- Render each album on the page

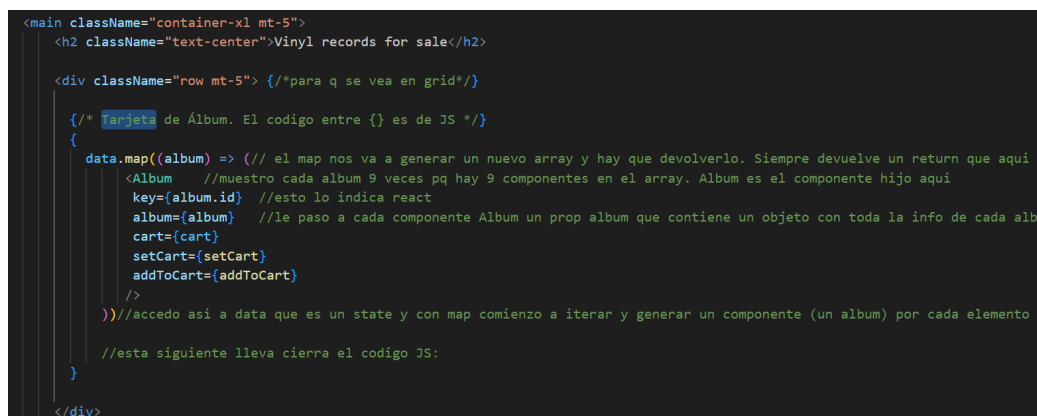
For this, a State is created using the **useState** Hook and initialized with the db array located in db.js. This is an array of objects that contains information about each of the records:



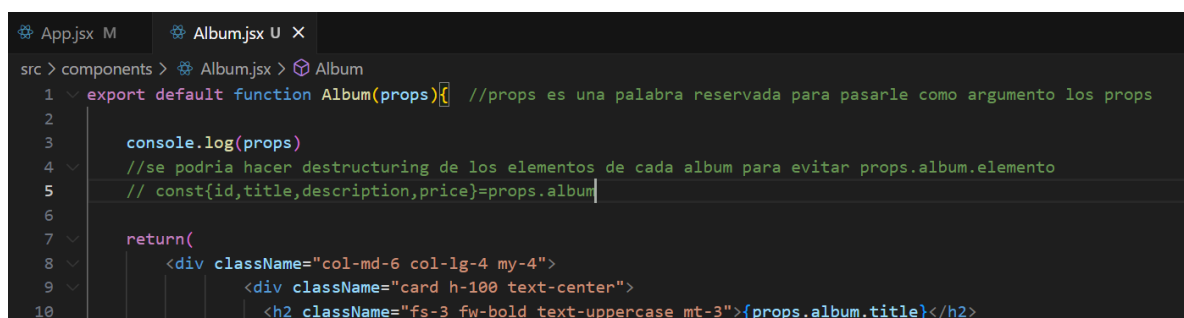
Inside the return of the `App` component, the `Header` component is called (which contains all the HTML code for the page header in `Header.jsx`).

Then, inside the `<main>`, the card for each album is created using JS code wrapped in curly braces. Starting from the `data` state, a new array is created with the `map` method, which iterates over each object in the `data` array and returns it. Each object represents an album, identified by a key, and with a `Prop`, I pass an album object to the child component `<Album/>` that contains all the information for that album. This is how each object in the array is accessed, iterated with `map`, and rendered in `Album.jsx`.

It is important to highlight that props are used by React components to communicate with each other, passing information from a parent component to a child component.



In `Album.jsx`, the component receives the reserved word `props` as a parameter, which allows access to each element of the album in the database using `props.album.title`. The latter must be interpreted as JS code, which is why it is wrapped in curly braces inside the `return`.



2. Implementing the shopping cart

For this, in Album.jsx an onClick attribute is added to the button and the handleClick function is called, which takes an album as a parameter. This function is declared before the return in the component and the album prop is passed to it. Then, in App.jsx, the cart state is created and passed as a prop.

A function AddToCart is implemented to add an album to the cart without duplicating items, and it is passed to Album.jsx via props to be called in the button's onClick. This function checks with findIndex() if the item is already in the cart, and if not, it is added with setCart, updating the cart state. The quantity of items is also checked, so if an item already exists in the cart, a quantity attribute is added or incremented.

Next, the cart section in the header is made dynamic. In Header.jsx, inside the tbody, map iterates over the cart array, passing each album object and modifying its image, title, price, and other attributes. The map returns all the HTML code.

A derived state is used to show whether the cart is empty or not. To calculate the total price, another derived state uses the reduce method on the cart array, taking two parameters: the accumulated total and the current item.

Additional functions are created in App.jsx to be used in Header.jsx via props: removeFromCart to remove items using the item id, increaseQuantity and decreaseQuantity to modify quantities, and clearCart to empty the cart, which does not take any parameters.

To use Google Chrome's local storage and keep the cart from being emptied on page reload, a useEffect hook is used to update localStorage whenever the cart state changes. When the page reloads, the code first checks if there is anything in local storage, and if not, an empty array (initialCart) is used. This value is assigned as the initial cart state so that the content from local storage is preserved if present. The following images show the implementation of these functions.

```
useEffect( () => {
  localStorage.setItem('cart', JSON.stringify(cart)) //localStorage no permite objetos o arrays asíq lo paso a string
}, [cart]) //cada vez que cart cambie ejecutar el código ese

function addToCart(item){

  const itemExists=cart.findIndex( (album) => album.id ===item.id) //este método no muta el array, y compruebo que el id del item

  if(itemExists>=0){
    console.log("Este elemento ya existe en el carrito")

    const copiaCart=[...cart] //tomo una copia del carrito
    copiaCart[itemExists].quantity++ //aumento la cantidad en el album cuyo índice sea usando una copia para no modificar el estado
    setCart( copiaCart ) //siempre q escriba en mi carrito tengo que setearlo, OJO que no hay corchetes
  } else{
    console.log("agregando item al carrito...")
    item.quantity=1
    setCart( [...cart, item] ) // llamo a la función setCart q modifica el estado cart, la cual añade a lo q hay en carrito el album
    //con los corchetes devuelvo un array, sino estuviesen estaría devolviendo un objeto ya que item (album) es un objeto. asíq lo
  }

  saveLocalStorage()
}
```

```
function decreaseQuantity(id){
  console.log("Se ha decrementado la cantidad del articulo", id)
  const updatedCart=cart.map(item=> {
    if(item.id===id && item.quantity>1){
      return{
        ...item,
        quantity:item.quantity-1
      }
    }
    return item
  })
  setCart(updatedCart)
}

function clearCart(){
  setCart([])
  console.log("Se ha vaciado el carrito")
}


function saveLocalStorage(){
  localStorage.setItem('cart', JSON.stringify(cart)) //localStorage no puede almacenar arrays asi q lo convierto a string
}
```



Finally, the command `npm run build` is used in the project terminal to generate the optimized, production-ready version of the application, which is saved, as explained earlier, in the **dist** folder.

FIRST TWO PAGES

By The National




"First Two Pages of Frankenstein" blends The National's melancholic indie rock with introspective lyrics, exploring memory, solitude, and emotional fragility through rich, atmospheric instrumentation.

21.99 €

ADD TO CART

ZACH BRYAN

By Zach Bryan




"Zach Bryan" is a melodic country album that tells stories of love, loss, and life's struggles. Its music blends gentle melodies with emotional lyrics, creating a reflective and moving listening experience.

19.99 €

ADD TO CART

EVERMORE

By Taylor Swift



"Evermore" is a folk and alternative album that reflects on the impermanence of life and nostalgia. Its sound captures moments of sadness, but with the hope that neither the good nor the bad lasts forever.

39.99 €

ADD TO CART

Update: Application using custom Hooks and migrating the code to TypeScript

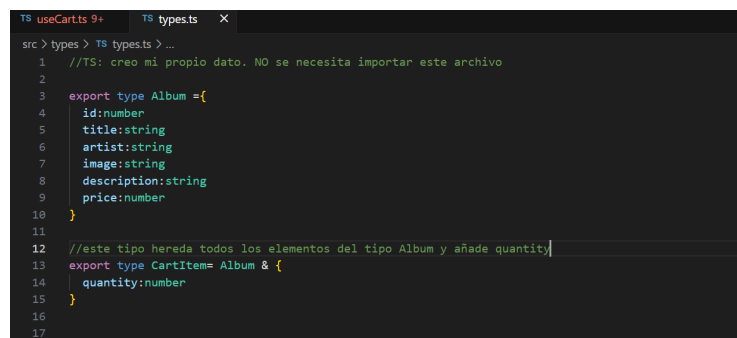
A hooks folder is created where I add my custom hooks, including useCart.js. All the JavaScript code that was previously in the App component (before the return) is migrated to the useCart custom hook, which returns the states and functions that are used. In App.jsx, when the App component is rendered, the custom hook is called, and within it, the states and useEffects are created, and the functions are defined. Then, when App calls its child components Header and Album, these components use the functions encapsulated in the useCart custom hook. Finally, the functions previously declared in the Header component are also migrated.

Next, the project is migrated from JavaScript to TypeScript. A new project is created from scratch using the terminal command `npm create vite@latest`. By selecting React with TypeScript, two new files are added to the structure: `tsconfig.json` and `tsconfig.node.json`, which configure TypeScript and allow compilation rules to be set. Another change is that the React files are now `.tsx` instead of `.jsx`.

In the database file (`db.ts`), a custom type `Album` is created using `type`, specifying the type of each property of the albums. The `db` array is therefore an array of objects of type `Album`.

In `Album.tsx`, a type `AlbumProps` is created for the props that the Album component receives. These props come from the useCart custom hook. Additionally, a file for declaring types is created inside the `types` folder: `types.ts`. In this file, the `Album` type is defined.

Next, the `useCart.ts` custom hook is modified by adding a type for the items in the cart. To do this, the `Album` type is extended with `CartItem` and a `quantity` attribute is added.



```
1 //TS: creo mi propio dato, NO se necesita importar este archivo
2
3 export type Album = {
4   id:number
5   title:string
6   artist:string
7   image:string
8   description:string
9   price:number
10 }
11
12 //este tipo hereda todos los elementos del tipo Album y añade quantity
13 export type CartItem= Album & {
14   quantity:number
15 }
16
17
```

Next, the type of `item` is changed. `item` is a parameter used by three functions and was previously declared as `any`. In TypeScript, no value should be left as type `any`.

To fix this, a lookup type `AlbumId` is created in the `types.ts` file, which refers to the same type as the `id` attribute of the `Album` type. Then, the parameter passed to these functions is updated to use this new type. The first image shows the `types` file, and the second shows the usage of this new type in the `useCart.ts` custom hook.

```
TS useCartts 2 TS types.ts X
src > types > TS types.ts > ...
17 export type AlbumId= Album['id'] //Esto es lo que se conoce como un lookup. Es un tipo q sea el atributo id del tipo Album
18
```

```
function removeFromCart(id: Album['id']){
```

Finally, the Header.tsx component needs to be updated. A type for the props called HeaderProps is created, so the props of the Header component are properly typed and TypeScript knows exactly which data and functions it can receive.

```
TS types.ts Header.tsx 1 X App.tsx 2
src > components > Header.tsx > HeaderProps
1 import {useMemo} from 'react' //es un hook q sirve para no hacer render completo de la aplicacion hasta que cambie algo q yo
2
3 import type { CartItem, Album } from '../types/types'
4
5 type HeaderProps={
6   cart:CartItem[],
7   removeFromCart: (id:Album['id']) => void //es una funcion q no retorna nada y que toma un id
8   increaseQuantity: (id:Album['id']) => void
9   decreaseQuantity: (id:Album['id']) => void
10  clearCart: ()=> void
11  estaVacio:boolean
12  cartTotal: () =>number
13
14 }
15
16
17 function Header( {cart, removeFromCart, increaseQuantity, decreaseQuantity, clearCart, estaVacio, cartTotal} :HeaderProps ){
```

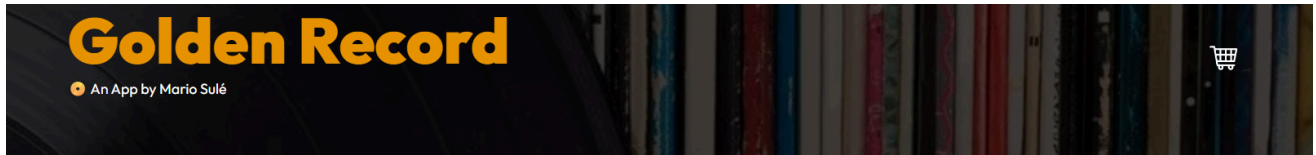
To finish, npm run build is run to build the project, errors related to unused states and data are fixed, and the dist folder is generated again.



Golden Record

Documentación del proyecto - Mario Sulé Domínguez

<https://github.com/mariosulee/Golden-Record.git>



Vinyl records for sale

RECKLESS

By Bryan Adams



"Reckless" is a melodic rock album that channels high energy and unrestrained emotion. Its songs capture the thrill of youth, boldness, and living without limits, blending powerful guitar riffs and heartfelt lyrics.

19.99 €

ADD TO CART

SWIMMING

By Mac Miller



"Swimming" is Mac Miller's fifth studio album. It explores themes of self-reflection, growth, and struggle. It blends hip-hop, jazz, and soul influences, capturing his journey through life, love, and mental health.

49.99 €

ADD TO CART

DAWN FM

By The Weeknd



This synth-electronic album guides listeners through reflection on life, regrets, and acceptance. Its immersive production conveys a surreal journey toward light and resolution.

13.99 €

ADD TO CART

El proyecto desarrollado consiste en la implementación de una aplicación que se base en una tienda virtual de vinilos desarrollada con React, donde se muestran los álbumes disponibles, cada uno con distintos atributos como título, artista o precio, e incluso una descripción de los vinilos que están en venta y un botón para añadir cada uno de ellos al carrito. Los usuarios pueden así incluir álbumes en su carrito de compra, editar las cantidades, ver cómo se actualiza el precio total en tiempo real, y también vaciar el carrito si lo desean, ofreciendo una experiencia completa de compra online de manera dinámica e interactiva, contando con un diseño de aplicación moderno y responsivo, siendo fácil de usar en cualquier dispositivo.

Con esta documentación se pretende explicar de manera clara y detallada cómo está estructurado y funciona el proyecto, incluyendo tanto la funcionalidad principal como la estructura técnica (carpetas, archivos, componentes, Hooks de React, props, estilos, configuración de Vite y dependencias). El objetivo es que cualquier persona que lea la documentación pueda entender cómo se ha construido la aplicación, cómo interactúan los componentes y cómo se ejecuta el proyecto, sin entrar a analizar o explicar todo el código, sirviendo como guía tanto para mantenimiento como para futuras mejoras o aprendizaje.

Para crear este proyecto, primeramente se debe tener instalado **Node.js** y **NPM**. Node.js es un entorno que permite ejecutar código de JavaScript fuera del navegador, normalmente en el servidor, mientras que NPM es el gestor de paquetes de este, que permite instalar dependencias y librerías. El proyecto se creará usando Vite, que es una herramienta de desarrollo para proyectos web que sirve para React.

Respecto a la estructura del proyecto:

- La carpeta **dist** es la que contiene el código final optimizado y listo para producción de la app de React que se crea después de ejecutar `npm run build`.
- **node_modules** es la carpeta en la que están las dependencias.
- **public** es una carpeta en la que se guardarán las imágenes que serán accesibles para los visitantes de la aplicación web.
- En `src` se encontrará todo el código y la parte funcional de la aplicación. Incluye la carpeta `components` (con los componentes `Album` y `Header`) y la de `data` (en la que se encuentra la base de datos simulada en JavaScript, representada por un array de objetos), el componente raíz `App.jsx` y `main.jsx` (que renderiza este componente principal de App en el `index.html`). También existe una hoja de estilos `index.css` hecha con Bootstrap.
- **eslint.config.js** define reglas de cómo debe escribirse el código de JS, y no se modifica.
- **.gitignore** sirve para indicarle a Git qué archivos o carpetas no debe guardar ni subir al repositorio.
- **package-lock.json** se genera a partir del `package.json` que contiene las dependencias del proyecto y desarrollo.
- Finalmente, **vite.config.js** es el archivo de configuración de Vite.

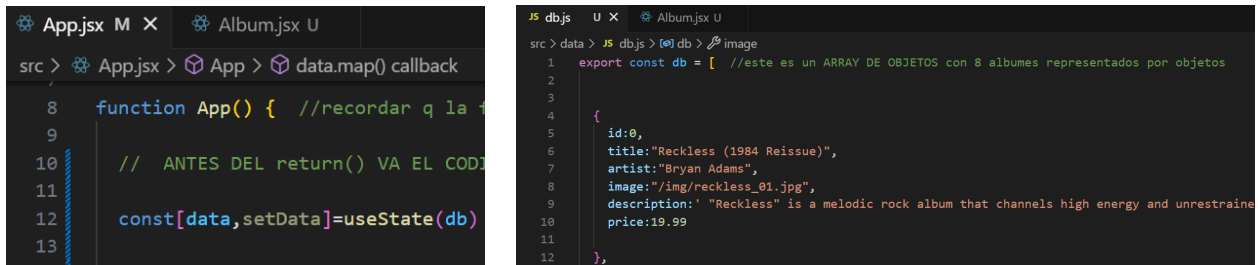
Lo que se hace es colocar todos los álbumes en un array, iterar sobre él, y luego mostrar la información en cada componente de forma dinámica usando Hooks de React, los cuales permiten usar funciones propias de React en componentes.

Para ver la pestaña `components` al inspeccionar código del navegador (en mi caso Google Chrome), se instala la extensión React Developer Tools. Esto nos muestra el árbol de componentes de React, y se usará para poder ver valores de, por ejemplo, el `State`, sin tener que hacer todo el `console.log`.

Se crea entonces una carpeta de `data` y dentro se le añade el archivo [db.js](#) (que va a ser lo que simule la base de datos con la información y atributos de cada álbum). Este archivo se compone de un array de objetos, siendo cada objeto un álbum que estará disponible en la tienda virtual. Cada objeto tendrá atributos como el título, el artista, su precio, una descripción, una imagen, y un identificador.

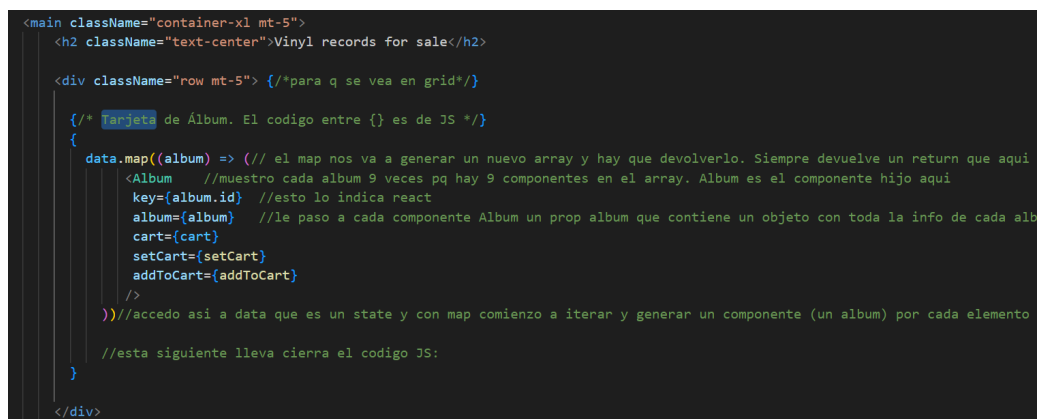
1. Renderizar cada álbum en la página

Para ello, se crea un State con el Hook **useState** y lo inicio al array db localizado en [db.js](#). Este es un array de objetos que contiene información sobre cada uno de los discos:

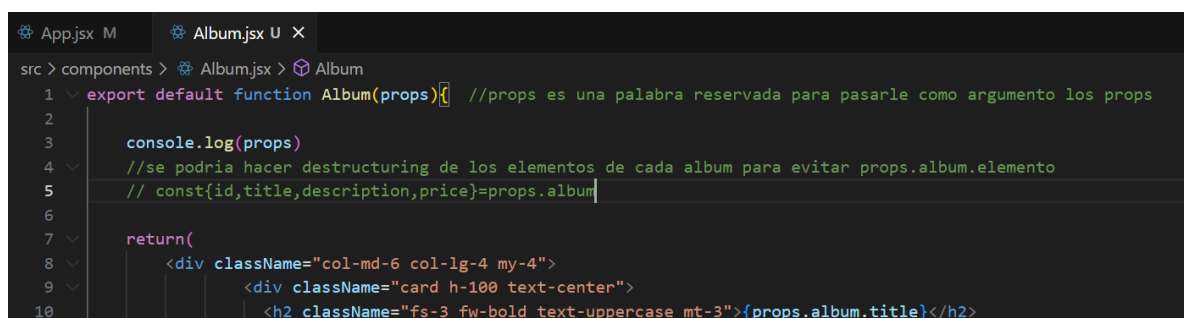


Dentro del return del componente App se llama al componente Header (que contiene todo el código HTML del cabecero de la página en Header.jsx).

Después, en el <main> se crea la tarjeta de cada álbum con código JS encerrado entre llaves. Partiendo del estado data creo un nuevo array con el método map, el cual va a iterar sobre cada objeto del array data y devolverlo. Cada objeto es un álbum, identificado por key y que con un Prop hago que el componente hijo <Album/> tenga un objeto álbum que contenga toda la información de cada uno de ellos. Accedo de esta forma a cada objeto del array, iterando con map y renderizando en Album.jsx. Es importante destacar que los props son usados por los componentes de React para comunicarse entre ellos pasando información de un componente padre a uno hijo.



En Album.jsx al componente se le pasa como parámetro la palabra reservada props, el cual accede a cada elemento del album en la base de datos usando props.album.title. Esto último se debe interpretar como código en JS y por eso va entre llaves dentro del return.



2. Implementar el carrito de compras

Para ello, en Album.jsx se agrega un atributo onClick en el <button> y se llama a la función handleClick la cual toma por parámetro un álbum. Esta función se declara antes del return en el componente y se le pasa por parámetro el prop álbum. A continuación, en App.jsx se crea el estado carrito y se pasa por prop el estado de cart.

Se implementa entonces una función AddToCart para añadir un álbum al carrito y que no se repitan elementos y se le pasa a Album.jsx con props para llamar a esa función en el onClick del botón 'add to cart'. Esta función comprueba con findIndex() si ya estaba en el carrito ese item, y si no estaba se añade con setCart actualizando así el estado del carrito. Se debe comprobar también la cantidad de los elementos por si un elemento ya existe en el carrito añadir el atributo quantity e incrementarlo.

Ahora se debe hacer dinámica la parte del carrito del header. Para ello vuelvo al Header.jsx y en el <tbody> se itera con el método map sobre el carrito de compras pasándole el objeto álbum y modificando imagen título precio y demás atributos. Sobre este map se retornatodo el código HTML.

Con un state derivado muestro si el carrito está vacío o no. Para saber el total a pagar uso otro state derivado con el array method reduce, el cual iterara sobre el estado cart y tomará dos parámetros: uno es el total acumulado, y otro es el item (que es el elemento actual).

Para demás funcionalidades se van creando funciones en App.jsx para usarse en Header.jsx por medio de props: Para eliminar artículos del carrito de compras, se crea otra función llamada *removeFromCart* tomando como parámetro el id del artículo. Para incrementar o decrementar las cantidades de cada artículo en el carrito, *increaseQuantity* y *decreaseQuantity*. Para implementar el vaciar todo el carrito hago otra función llamada *clearCart*, pero esta no toma ningún parámetro.

Para usar el Local Storage de Google Chrome y que al recargar la página el carrito no se vacíe y guarde en un String todos los artículos del carrito, uso un hook useEffect que ejecute localStorage cada vez que el estado cart cambie. Para que al recargarse no se pierda el carrito, primero se revisa si hay algo en local storage, y si no, se coloca un array vacío (initialCart). Este valor se asigna al state inicial del estado cart, para que tenga (en caso de que haya), el contenido de local storage. Las siguientes imágenes muestran la implementación de las funciones mencionadas.

```
useEffect( () => {
  localStorage.setItem('cart', JSON.stringify(cart)) //localStorage no permite objetos o arrays asíq lo paso a string
}, [cart]) //cada vez que cart cambie ejecutar el código ese

function addToCart(item){

  const itemExists=cart.findIndex( (album) => album.id ===item.id) //este metodo no muta el array, y compruebo que el id del item

  if(itemExists>=0){
    console.log("Este elemento ya existe en el carrito")

    const copiaCart=[...cart] //tomo una copia del carrito
    copiaCart[itemExists].quantity++ //aumento la cantidad en el album cuyo indice sea usando una copia para no modificar el estado
    setCart( copiaCart ) //siempre q escriba en mi carrito tengo que setearlo, OJO que no hay corchetes
  } else{
    console.log("agregando item al carrito...")
    item.quantity=1
    setCart( [...cart, item] ) // llamo a la función setCart q modifica el estado cart, la cual añade a lo q hay en carrito el album
    //con los corchetes devuelvo un array, sino estuviesen estaria devolviendo un objeto ya que item (album) es un objeto. asíq lo
  }

  saveLocalStorage()
}
```

```
function decreaseQuantity(id){
  console.log("Se ha decrementado la cantidad del articulo", id)
  const updatedCart=cart.map(item=> {
    if(item.id===id && item.quantity>1){
      return{
        ...item,
        quantity:item.quantity-1
      }
    }
    return item
  })
  setCart(updatedCart)
}

function clearCart(){
  setCart([])
  console.log("Se ha vaciado el carrito")
}

function saveLocalStorage(){
  localStorage.setItem('cart', JSON.stringify(cart)) //localStorage no puede almacenar arrays asi q lo convierto a string
}
```

Golden Record

An App by Mario Sulé

Vinyl records for sale

RECKLESS

By Bryan Adams

SWIMMING

By Mac Miller

Image	Title	Price	Quantity
	Zach Bryan	19.99 €	<input type="text" value="1"/>
	Mira Dentro	16.99 €	<input type="text" value="1"/>
	Fearless (Taylor's Version)	29.99 €	<input type="text" value="1"/>

Total to pay: 66.97€

EMPTY CART

Finalmente, se usa el comando `npm run build` en la terminal dentro del proyecto para generar la versión optimizada y lista para la producción de la aplicación, la cual se guarda como se explica anteriormente, en la carpeta `dist`.

FIRST TWO PAGES

By The National

"First Two Pages of Frankenstein" blends The National's melancholic indie rock with introspective lyrics, exploring memory, solitude, and emotional fragility through rich, atmospheric instrumentation.

21.99 €

ADD TO CART

ZACH BRYAN

By Zach Bryan

"Zach Bryan" is a melodic country album that tells stories of love, loss, and life's struggles. Its music blends gentle melodies with emotional lyrics, creating a reflective and moving listening experience.

19.99 €

ADD TO CART

EVERMORE

By Taylor Swift

"Evermore" is a folk and alternative album that reflects on the impermanence of life and nostalgia. Its sound captures moments of sadness, but with the hope that neither the good nor the bad lasts forever.

39.99 €

ADD TO CART

Update: Application using custom Hooks and migrating the code to TypeScript

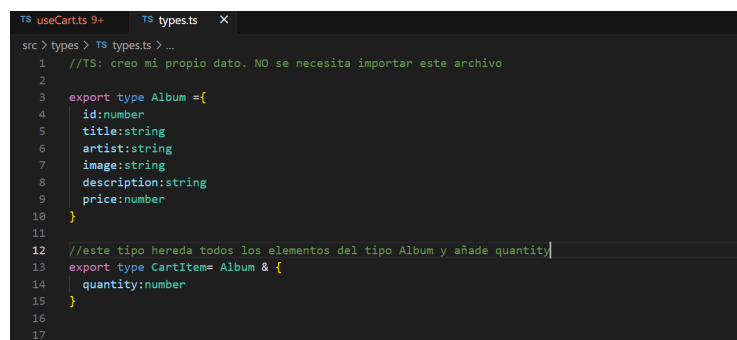
Se crea una carpeta hooks en la que voy colocando mis custom hooks y dentro [useCart.js](#). Se migra todo el código JavaScript que se encontraba en el componente App (antes del return) al custom hook useCart, el cual devuelve las funciones y estados que se usan. En App.jsx, al renderizarse el componente App, se llama al custom hook y dentro de este, se crean los estados y useEffects, y se definen las funciones. Luego, al llamar App a sus componentes hijos Header y Album en ellos se llaman a las funciones encapsuladas en el custom hook de useCart. Finalmente también se migran las funciones declaradas en el componente Header.

A continuación se propone migrar el proyecto de JavaScript a TypeScript. Se crea un proyecto nuevo desde cero por terminal usando el comando `npm create vite@latest`. Al seleccionar hacer el proyecto de React con TypeScript se añaden dos archivos nuevos a la estructura: `tsconfig.json` y `tsconfig.node.json` los cuales configuran TS en los cuales se pueden configurar reglas para la compilación. Lo que también cambia es que los archivos de REACT ya no son `jsx` sino `tsx`.

En el archivo de base de datos ([db.ts](#)) se crea un tipo propio (Album) con `type` en el cual se declara el tipo de cada parámetro de los álbumes. El array `db` entonces será un array de objetos del tipo Album.

En `Album.tsx` se crea un tipado `AlbumProps` para los props que recibe el componente de Album. Recordar que los recibe del custom hook [UseCart.ts](#). También se crea un archivo para declarar tipos dentro de la carpeta `types`, el archivo [types.ts](#). En el se declara el tipado de Album.

A continuación, se modifica el custom hook [useCart.ts](#) se le añade un tipo para los elementos del carrito. Para ello se extiende el tipo Album con `CartItem` y se añade el atributo de cantidad:



```
TS useCart.ts  TS types  X
src > types > TS types > ...
1 //TS: creo mi propio dato. NO se necesita importar este archivo
2
3 export type Album = {
4   id:number
5   title:string
6   artist:string
7   image:string
8   description:string
9   price:number
10 }
11
12 //este tipo hereda todos los elementos del tipo Album y añade quantity
13 export type CartItem= Album & {
14   quantity:number
15 }
16
17
```

Seguidamente, se cambia el tipo de item, que es un parámetro que usan tres funciones y que antes estaba declarado como `any`. Si se programa en TypeScript no se puede dejar ningún dato en un tipo `any`.

Para ello creo un LookUp Type `AlbumId` en el archivo de [types.ts](#) el cual será un tipo que se refiera al mismo tipo que es el atributo `id` del tipo Album. Seguidamente cambio el parámetro

que se les pasa a estas funciones para que sea de este nuevo tipo. La primera imagen es del archivo de types y la segunda del uso de este nuevo tipo en el custom hook useCart.ts

```
TS useCartts 2 TS types.ts X
src > types > TS types.ts > ...
17 export type AlbumId= Album['id'] //Esto es lo que se conoce como un lookup. Es un tipo q sea el atributo id del tipo Album
18
```

```
function removeFromCart(id: Album['id']){
```

Finalmente, falta modificar el componente Header.tsx. Se crea un tipado para los props llamado HeaderProps y así se tipan los props del componente Header para que TypeScript sepa exactamente qué datos y funciones puede recibir:

```
TS types.ts Header.tsx 1 X App.tsx 2
src > components > Header.tsx > [e] HeaderProps
1 import {useMemo} from 'react' //es un hook q sirve para no hacer render completo de la aplicacion hasta que cambie algo q yo
2
3 import type { CartItem, Album } from '../types/types'
4
5 type HeaderProps={
6   cart:CartItem[],
7   removeFromCart: (id:Album['id']) => void //es una funcion q no retorna nada y que toma un id
8   increaseQuantity: (id:Album['id']) => void
9   decreaseQuantity: (id:Album['id']) => void
10  clearCart: ()=> void
11  estaVacio:boolean
12  cartTotal: () =>number
13
14 }
15
16
17 function Header( {cart, removeFromCart, increaseQuantity, decreaseQuantity, clearCart, estaVacio, cartTotal} :HeaderProps ){
```

Para terminar, se hace npm run build para construir el proyecto, se corrigen los errores sobre información estados y datos no usados, y se genera la carpeta **dist** otra vez.