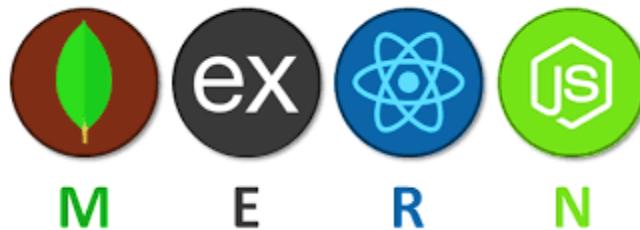


Full Stack MERN



¿Qué es MERN?

MERN es un acrónimo que se utiliza para describir un conjunto de tecnologías de desarrollo web utilizadas juntas para construir aplicaciones web modernas y escalables. Las letras en MERN representan las siguientes tecnologías:

- [MongoDB](#): una base de datos NoSQL que utiliza JSON como formato de almacenamiento de datos.
- [Express](#): un framework de aplicación web de Node.js que proporciona una estructura sólida para construir aplicaciones web.
- [React](#): una biblioteca de JavaScript de código abierto utilizada para construir interfaces de usuario interactivas y escalables.
- [Node.js](#): un entorno de tiempo de ejecución de JavaScript que permite ejecutar código de JavaScript en el servidor.

En conjunto, MERN proporciona una pila de tecnología de desarrollo web completa y escalable para construir aplicaciones web modernas y altamente eficientes.

MERN se utiliza en una amplia variedad de aplicaciones web modernas y escalables. Algunos ejemplos de aplicaciones que utilizan MERN incluyen:

- Redes sociales: aplicaciones web como Facebook, Twitter y LinkedIn utilizan MERN para construir interfaces de usuario interactivas y escalables, y almacenar y administrar grandes cantidades de datos de usuarios.
- Aplicaciones de comercio electrónico: sitios web como Amazon, eBay y Alibaba utilizan MERN para construir plataformas de compras en línea escalables y eficientes.
- Aplicaciones de medios: plataformas de transmisión de video como Netflix y YouTube utilizan MERN para construir interfaces de usuario escalables y manejar grandes cantidades de datos de contenido.
- Aplicaciones de productividad: aplicaciones web como Trello y Slack utilizan MERN para construir interfaces de usuario interactivas y escalables para la gestión de proyectos y la colaboración en equipo.



@hdtoledo

Requisitos para iniciar con el curso

- Nociones básicas de JavaScript
- Nociones básicas de React JS

Descripción

En este curso vas a aprender a crear una aplicación web que será una web personal con panel de Administrador protegido por un login con **JWT**, todo paso a paso usando el **MERN Stack** que está compuesto por **MongoDB**, **Express JS**, **React JS** y **Node JS**.

Crearemos nuestra aplicación desde cero sin usar nada prefabricado, **aprenderemos base de datos no relacional con MongoDB**, en el **Backend crearemos un API REST** con **Node JS** y **Express JS** y en el **Frontend usaremos React JS** con **Hooks** y en la **parte del CSS usaremos SASS**.

Cuando tengamos nuestra aplicación terminada, aprenderemos a **desplegar nuestra aplicación en la nube**.

Este curso tiene como objetivo enseñarte a desarrollar cualquier tipo de aplicación desde cero, **convirtiéndote** en un **desarrollador Full Stack** sobre el **MERN Stack**.

Estructura del curso

- **Instalación y configuración** del entorno de trabajo.
- **Desacoplaremos** nuestro **proyecto** en **tres bloques**, Base de Datos, Backend y Frontend.
- **Crearemos una API REST** desde cero conectada a MongoDB.
- Aprenderemos a usar el **ODM Mongoose**.
- **Añadiremos** al Frontend **SASS**.
- **Crearemos** una **configuración dinámica** de **React Router Dom**.
- Crearemos Sistemas de Layouts.
- Creamos un **sistema de Auth** protegido con **JWT** y tendremos el **AccessToken** para permitir acceso y **RefreshToken** para recuperar sesiones.
- **Creamos un panel de Administrador** para que nuestros usuarios con privilegios puedan **gestionar la web**.
- Construiremos un Menú completamente dinámico gestionado desde el panel de administrador.
- Crearemos una **Home Page** llamativa.
- Programaremos un **Sistema de Newsletter** 100% funcional **desde cero**.
- Crearemos una **sección para subir cursos** conectada a una **API**.

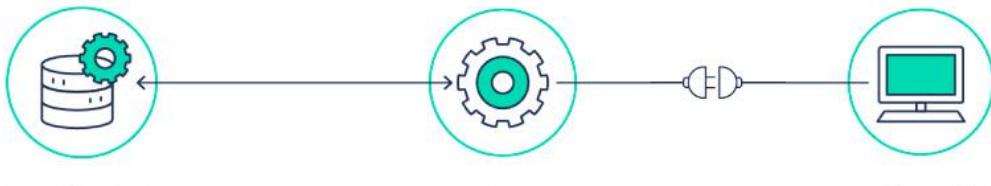


@hdtoledo

- Crearemos un **Sistema de Blog**, con **paginación y creación de URL dinámicas** todo **gestionado desde el panel de Administrador**.
- Gestionaremos el **SEO On Page** para mejorar nuestra visibilidad en **Google**.
- Desplegaremos nuestra aplicación en **varios servidores en la nube**.

Cliente API

¿Cómo funciona una API?



Base de datos

La información o las herramientas desarrolladas por una empresa son utilizadas en servicios de terceros.

API

Una API permite conectar la información o funcionalidades con los requerimientos de una aplicación.

Aplicación

El cliente tiene acceso a toda su información requerida en una sola aplicación.

Un cliente API es una aplicación o componente de software que se utiliza para interactuar con una API (Interfaz de Programación de Aplicaciones).

Una API es un conjunto de reglas y protocolos que permite a diferentes aplicaciones comunicarse y compartir datos entre sí. El cliente API actúa como un intermediario entre el usuario o la aplicación y la API, enviando solicitudes a la API y recibiendo respuestas en un formato específico, como JSON o XML.

El cliente API puede ser desarrollado en diferentes lenguajes de programación y se utiliza para realizar operaciones como enviar solicitudes HTTP, autenticarse con la API, enviar parámetros y recibir datos de respuesta. Por ejemplo, un cliente API puede utilizarse para acceder a servicios web, obtener información de una base de datos remota o interactuar con plataformas de redes sociales.

En resumen, un cliente API es un componente de software que permite a una aplicación o usuario interactuar con una API para acceder y utilizar los servicios y datos proporcionados por la API.

Algunas de ellas:

1. [**Insomnia**](#) es una herramienta de cliente API de código abierto y multiplataforma que se utiliza para realizar solicitudes HTTP y probar APIs. Proporciona una interfaz gráfica intuitiva que permite a los desarrolladores enviar solicitudes a una API, inspeccionar las respuestas y depurar problemas de comunicación.



@hdtoledo

Insomnia ofrece características útiles como autocompletado de URL y parámetros, resultado de sintaxis, guardado de solicitudes y respuestas, gestión de cookies, autenticación y soporte para varios tipos de autenticación (como API key, token de acceso, OAuth, etc.), entre otras.

La herramienta es especialmente útil durante el desarrollo y la integración de APIs, ya que permite realizar pruebas exhaustivas, guardar configuraciones y compartir solicitudes entre miembros del equipo. Además, Insomnia admite la importación y exportación de colecciones de solicitudes, lo que facilita la colaboración y el intercambio de configuraciones de API.

En resumen, Insomnia es una herramienta poderosa y versátil para interactuar con APIs y realizar pruebas de API de manera eficiente.

2. **Postman:** Una herramienta de colaboración y desarrollo de API que permite enviar solicitudes y recibir respuestas de APIs. Proporciona una interfaz gráfica fácil de usar para probar y depurar APIs.
3. **cURL:** Una herramienta de línea de comandos que permite enviar y recibir datos utilizando varios protocolos, incluyendo HTTP, FTP y SMTP. Es muy utilizado para interactuar con APIs mediante solicitudes HTTP.
4. **Axios:** Una biblioteca de JavaScript ampliamente utilizada para realizar solicitudes HTTP desde aplicaciones web o móviles. Proporciona una interfaz fácil de usar y admite promesas para un código más legible y conciso.
5. **Retrofit:** Una biblioteca de Android que simplifica la comunicación con servicios web a través de API RESTful. Proporciona una forma sencilla de definir y realizar solicitudes HTTP en aplicaciones Android.

Node.js

Node.js es un entorno de ejecución de código abierto basado en el motor de JavaScript V8 de Chrome. A diferencia de la ejecución de JavaScript en el navegador, Node.js permite ejecutar JavaScript en el lado del servidor, lo que lo convierte en una opción popular para el desarrollo de aplicaciones web y servidores.

Node.js utiliza un enfoque asíncrono y basado en eventos, lo que significa que puede manejar múltiples solicitudes simultáneamente sin bloquear el flujo de ejecución. Esto se logra utilizando el bucle de eventos y el modelo de E/S no bloqueante, lo que permite una escalabilidad eficiente y un rendimiento de alta concurrencia.

Algunas características clave de Node.js son:

1. **JavaScript en el lado del servidor:** Permite a los desarrolladores utilizar el mismo lenguaje de programación tanto en el lado del cliente como en el lado del servidor, lo que facilita el intercambio de código y la reutilización de habilidades.
2. **Eficiencia y rendimiento:** Node.js utiliza un enfoque sin bloqueo y basado en eventos, lo que le permite manejar grandes cargas de trabajo con eficiencia y alta concurrencia.



3. Amplio ecosistema de módulos: Node.js cuenta con un gestor de paquetes llamado npm (Node Package Manager), que permite a los desarrolladores acceder a un vasto repositorio de módulos y paquetes preexistentes para facilitar el desarrollo de aplicaciones.
4. Desarrollo rápido de prototipos: La facilidad de uso y la naturaleza flexible de JavaScript, junto con la disponibilidad de numerosos módulos de terceros, hacen que Node.js sea una opción popular para el desarrollo rápido de prototipos y la construcción de aplicaciones escalables.

Node.js se utiliza en una amplia gama de aplicaciones, desde el desarrollo de servidores web y aplicaciones de tiempo real hasta herramientas de línea de comandos y aplicaciones de Internet de las cosas (IoT).

Gestor de paquetes

Un gestor de paquetes es una herramienta que facilita la instalación, gestión y actualización de paquetes de software en un entorno de desarrollo o producción. Los paquetes son componentes de software preempaquetados que contienen código, bibliotecas, dependencias y otros recursos necesarios para realizar una funcionalidad específica.

Los gestores de paquetes simplifican el proceso de adquisición y administración de paquetes, proporcionando una forma estandarizada y automatizada de manejar las dependencias y versiones de software. Algunos ejemplos populares de gestores de paquetes son npm (Node Package Manager) para Node.js, pip para Python, gem para Ruby y apt-get para sistemas basados en Debian/Ubuntu.

Las principales funciones de un gestor de paquetes incluyen:

1. Instalación: Permite descargar e instalar paquetes de software en un entorno determinado. El gestor de paquetes se encarga de manejar las dependencias y asegurarse de que todas las dependencias necesarias también se instalen correctamente.
2. Gestión de dependencias: Maneja las dependencias entre los paquetes, asegurándose de que todas las dependencias requeridas estén presentes y en las versiones correctas. Esto simplifica la configuración del entorno de desarrollo y garantiza que las aplicaciones funcionen correctamente.
3. Actualización: Permite actualizar los paquetes instalados a nuevas versiones disponibles. Los gestores de paquetes pueden verificar y descargar automáticamente las actualizaciones, facilitando la tarea de mantener el software actualizado.
4. Control de versiones: Administra las diferentes versiones de los paquetes, permitiendo instalar versiones específicas o manejar múltiples versiones simultáneamente. Esto es especialmente útil cuando se trabaja en proyectos que requieren versiones específicas de bibliotecas o dependencias.

En resumen, un gestor de paquetes es una herramienta esencial para simplificar la gestión y distribución de software, asegurando la consistencia y facilitando la colaboración en proyectos de desarrollo de software.



Yarn

Yarn es un gestor de paquetes de código abierto desarrollado por Facebook que se utiliza principalmente en proyectos de JavaScript. Al igual que npm (Node Package Manager), Yarn permite instalar, gestionar y actualizar paquetes de software y sus dependencias.

Sin embargo, Yarn se diseñó con el objetivo de mejorar la velocidad, la eficiencia y la confiabilidad en comparación con npm. Algunas características destacadas de Yarn son:

1. Resolución de dependencias determinista: Yarn utiliza un algoritmo de resolución de dependencias más robusto que garantiza que las mismas dependencias se instalen en todas las máquinas, evitando problemas de inconsistencias entre los entornos de desarrollo.
2. Instalaciones paralelas: Yarn realiza las instalaciones de paquetes de manera paralela, aprovechando al máximo la capacidad del hardware y acelerando el proceso de instalación.
3. Cache de paquetes: Yarn utiliza un sistema de caché local que almacena las versiones descargadas de los paquetes, lo que permite una instalación más rápida y eficiente en futuros proyectos o actualizaciones.
4. Resolución de conflictos automática: En caso de conflictos en las versiones de las dependencias, Yarn resuelve automáticamente los conflictos y selecciona la mejor versión disponible según las reglas de resolución establecidas.
5. Integridad de los paquetes: Yarn verifica la integridad de los paquetes instalados mediante el uso de sumas de verificación (checksums), asegurando que los paquetes descargados no hayan sido modificados o corrompidos.

En resumen, Yarn es un gestor de paquetes que proporciona mejoras en la velocidad, eficiencia y confiabilidad en comparación con npm. Es especialmente útil en proyectos de JavaScript con muchas dependencias, ya que ayuda a garantizar la consistencia y la estabilidad del entorno de desarrollo.

MongoDB

The screenshot shows the MongoDB homepage on the left and a terminal window on the right. The homepage features the MongoDB logo, navigation links for Products, Solutions, Resources, Company, and Pricing, and a 'Try Free' button. The main content area has a green header with a banner message: '(Event) [LAST CALL] Limited space available for MongoDB Local NYC on June 22. Save 50% with code BANNER50! Learn more >'. Below this is a large 'Build the next big thing' call-to-action with a subtext about being a developer data platform. At the bottom are 'Start Free' and 'Documentation' buttons. The terminal window on the right shows a command-line interface connecting to a MongoDB cluster: 'Connecting to: mongodb+srv://cluster0.ab123.mongodb.net/?retryWrites=true&w=majority' and 'Using MongoDB: 6.0'. The output lists various MongoDB regions and their names.

Region	Name
South America	southamerica-west1
South America	brazilsoutheast
Northern America	northamerica-northeast2
South America	northamerica-northeast1
Asia Pacific	asia-south2
Australia	australia-southeast2
Europe	europe-central2
Middle East	europe-central2
Africa	europe-central2
Asia Pacific	ap-northeast-3
North America	us-gov-west-1
Northern America	us-gov-east-1



MongoDB es un sistema de base de datos NoSQL (no relacional) que se ha vuelto popular debido a su flexibilidad y escalabilidad. Fue diseñado para manejar grandes volúmenes de datos de manera eficiente y brindar un acceso rápido a la información almacenada.

A diferencia de las bases de datos relacionales tradicionales, como MySQL o PostgreSQL, que utilizan tablas y esquemas predefinidos, MongoDB se basa en un modelo de documentos. En lugar de almacenar los datos en filas y columnas, MongoDB almacena los datos en documentos JSON (JavaScript Object Notation), que son estructuras de datos flexibles y autocontenidoas.

Cada documento en MongoDB se compone de pares clave-valor, donde la clave es el nombre del campo y el valor puede ser de varios tipos, como cadenas de texto, números, arreglos, objetos incrustados, etc. Esto permite una representación muy flexible de los datos y la capacidad de almacenar información relacionada dentro de un solo documento.

MongoDB también proporciona una potente consulta y lenguaje de manipulación de datos llamado MongoDB Query Language (MQL). Con MQL, puedes realizar consultas sofisticadas para buscar, filtrar, actualizar y eliminar datos en la base de datos.

Además, MongoDB está diseñado para ser escalable horizontalmente, lo que significa que puedes distribuir la carga de trabajo en varios servidores o clústeres. Esto permite manejar grandes volúmenes de datos y un alto rendimiento en entornos de aplicaciones con alta demanda.

En resumen, MongoDB es una base de datos NoSQL que utiliza un modelo de documentos flexible y escalable. Es especialmente útil para aplicaciones que manejan grandes cantidades de datos y requieren una alta flexibilidad en el esquema de datos.

[Mongoose](#)



Mongoose es una biblioteca de modelado de objetos para Node.js que proporciona una interfaz basada en esquemas para interactuar con bases de datos MongoDB. En pocas palabras, Mongoose es una capa de abstracción que simplifica y facilita el trabajo con MongoDB en aplicaciones Node.js.

Mongoose se utiliza comúnmente junto con MongoDB para definir la estructura de los datos, validarlos y proporcionar métodos para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en la base de datos de manera más intuitiva.

Algunas de las características principales de Mongoose son las siguientes:



1. Modelado de datos: Mongoose permite definir modelos con esquemas claros y consistentes. Los esquemas describen la estructura de los datos, incluidos los campos, los tipos de datos permitidos, las validaciones, los índices y otras configuraciones.
2. Validación de datos: Puedes utilizar los esquemas de Mongoose para aplicar reglas de validación a los datos antes de ser guardados en la base de datos. Esto asegura que los datos cumplan con los requisitos establecidos, como campos requeridos, tipos de datos específicos, longitud de cadenas, entre otros.
3. Middleware: Mongoose también permite agregar funciones de middleware para ejecutar código antes o después de ciertas operaciones de base de datos, como guardar, actualizar o eliminar un documento. Esto es útil para realizar tareas adicionales, como encriptar contraseñas, generar valores predeterminados, entre otros.
4. Consultas y agregaciones: Mongoose proporciona una API rica para realizar consultas y agregaciones en la base de datos. Puedes utilizar métodos encadenados y operadores para buscar datos, ordenarlos, filtrarlos y realizar cálculos de agregación más complejos.
5. Integración con Express y Node.js: Mongoose se integra bien con el marco de desarrollo web Express.js y se utiliza comúnmente en aplicaciones Node.js para interactuar con la base de datos MongoDB de manera sencilla y eficiente.

En resumen, Mongoose es una biblioteca de modelado de objetos para Node.js que simplifica la interacción con MongoDB al proporcionar una interfaz basada en esquemas. Facilita la definición de modelos de datos, la validación, las consultas y otras operaciones comunes en la base de datos.

IDE (Entornos de Desarrollo Integrado)

Hay varios IDE (Entornos de Desarrollo Integrado) populares que puedes considerar según el lenguaje de programación que deseas utilizar. Aquí hay algunas opciones recomendadas para diferentes lenguajes:

1. [Visual Studio Code \(VS Code\)](#): Es un editor de código gratuito y altamente personalizable desarrollado por Microsoft. Es ampliamente utilizado y admite una amplia gama de lenguajes de programación. Tiene una gran cantidad de extensiones disponibles que te permiten personalizar y ampliar sus capacidades según tus necesidades.
2. PyCharm: Es un IDE de JetBrains diseñado específicamente para el desarrollo de Python. Proporciona características avanzadas para la edición, depuración y análisis de código Python. Hay una versión gratuita llamada PyCharm Community Edition, así como una versión de pago llamada PyCharm Professional con características adicionales.
3. IntelliJ IDEA: También desarrollado por JetBrains, IntelliJ IDEA es un IDE potente y versátil que admite varios lenguajes de programación, incluidos Java, Kotlin, Scala, JavaScript y más. Además de sus características estándar, ofrece herramientas inteligentes de refactorización, análisis de código y soporte para frameworks populares.
4. Eclipse: Es un IDE gratuito y de código abierto que es ampliamente utilizado para el desarrollo en Java, pero también admite otros lenguajes a través de complementos. Eclipse tiene una gran



@hdtoledo

comunidad y una amplia gama de complementos disponibles que lo hacen altamente personalizable y adecuado para proyectos de diferentes tamaños.

5. Xcode: Si te interesa el desarrollo de aplicaciones para macOS, iOS, watchOS o tvOS, Xcode es el IDE oficial de Apple. Proporciona un conjunto completo de herramientas para el desarrollo de aplicaciones en Swift y Objective-C, así como para la creación de interfaces de usuario y la depuración en dispositivos de Apple.

Estas son solo algunas opciones populares, pero hay muchos otros IDE disponibles según tus necesidades y preferencias. Considera los lenguajes de programación que deseas utilizar, las características que necesitas y la comunidad y soporte disponibles al elegir un IDE para programar.

Extensiones para tener en cuenta en VS code

The screenshot displays five recommended extensions for Visual Studio Code:

- Bracket Pair Colorization Toggler** v0.0.3 by Dzhavat Ushev (610.204 installs, 4 stars): Quickly toggle 'Bracket Pair Colorization' setting with a simple command. Status: Enabled globally.
- ES7 React/Redux/GraphQL/React-Native snippets** v1.9.3 by rodrigovalladas (711.209 installs, 4 stars): Simple extensions for React, Redux and GraphQL in JS/TS with ES7 syntax (forked from dsznajder). Status: Enabled globally.
- ESLint** v2.4.0 by Microsoft (microsoft.com) (27.640.926 installs, 5 stars): Integrates ESLint JavaScript into VS Code. Status: Enabled globally.
- IntelliSense for CSS class names in HTML** v1.20.0 by Zignld (6.293.804 installs, 5 stars): CSS class name completion for the HTML class attribute based on the definitions found in your workspace. Status: Enabled globally.
- JavaScript (ES6) code snippets** v1.8.0 by charalampos karypidis (12.342.838 installs, 5 stars): Code snippets for JavaScript in ES6 syntax. Status: Enabled globally.



@hdtoledo



Hablemos de [React](#)

[wiki](#)

React es una biblioteca de JavaScript de código abierto para construir interfaces de usuario. Está basada en la componentización de la UI: la interfaz se divide en componentes independientes, que contienen su propio estado. Cuando el estado de un componente cambia, React vuelve a renderizar la interfaz.

Esto hace que React sea una herramienta muy útil para construir interfaces complejas, ya que permite dividir la interfaz en piezas más pequeñas y reutilizables.

Fue creada en 2011 por Jordan Walke, un ingeniero de software que trabajaba en Facebook y que quería simplificar la forma de crear interfaces de usuario complejas. Es una biblioteca muy popular y es usada por muchas empresas como Facebook, Netflix, Airbnb, Twitter, Instagram, etc.

```
state = { products: storeProducts }

render() {
  return (
    <React.Fragment>
      <div className="py-5">
        <div className="container">
          <Title name="our" title="product" />
          <div className="row">
            <ProductConsumer>
              {(value) => {
                console.log(value)
              }}
            </ProductConsumer>
          </div>
        </div>
      </React.Fragment>
    )
}
```

¿Qué es JSX?

JSX (JavaScript XML) es una extensión de sintaxis utilizada en React, una biblioteca de JavaScript para construir interfaces de usuario. JSX permite combinar HTML y JavaScript en un solo archivo, lo que facilita la creación de componentes reutilizables y dinámicos.



@hdtolledo

En JSX, se pueden escribir elementos y componentes de React utilizando una sintaxis similar a HTML, pero también se pueden incluir expresiones de JavaScript dentro de las etiquetas utilizando llaves {}. Esto permite la manipulación dinámica de datos y la generación de contenido basado en lógica.

```
import React from 'react';

// Definición de un componente de React utilizando JSX
const Saludo = ({ nombre }) => {
  return <h1>Hola, {nombre}!</h1>;
};

// Utilización del componente en otra parte de la aplicación
const App = () => {
  const usuario = 'John Doe';
  return (
    <div>
      <Saludo nombre={usuario} />
    </div>
  );
};

export default App;
```

En el ejemplo anterior, se define un componente de React llamado **Saludo** que recibe una prop **nombre**. En el componente principal **App**, se utiliza el componente **Saludo** pasando el valor '**John Doe**' como prop **nombre**. El componente **Saludo** renderiza un elemento **h1** que muestra el saludo personalizado.

Gracias a JSX, es posible combinar de manera elegante y legible HTML y JavaScript en un solo archivo, lo que facilita el desarrollo de interfaces de usuario dinámicas y componentes reutilizables en React.

¿Cuáles son las características principales de React?

Las características principales de React son:

- **Componentes:** React está basado en la componetización de la UI. La interfaz se divide en componentes independientes, que contienen su propio estado. Cuando el estado de un componente cambia, React vuelve a renderizar la interfaz.
- **Virtual DOM:** React usa un DOM virtual para renderizar los componentes. El DOM virtual es una representación en memoria del DOM real. Cuando el estado de un componente cambia, React vuelve a renderizar la interfaz. En lugar de modificar el DOM real, React modifica el DOM virtual y, a continuación, compara el DOM virtual con el DOM real. De esta forma, React sabe qué cambios se deben aplicar al DOM real.



@hdtoledo

- **Declarativo:** React es declarativo, lo que significa que no se especifica cómo se debe realizar una tarea, sino qué se debe realizar. Esto hace que el código sea más fácil de entender y de mantener.
- **Unidireccional:** React es unidireccional, lo que significa que los datos fluyen en una sola dirección. Los datos fluyen de los componentes padres a los componentes hijos.
- **Universal:** React se puede ejecutar tanto en el cliente como en el servidor. Además, puedes usar React Native para crear aplicaciones nativas para Android e iOS.

¿Qué es un componente?

Un componente es una pieza de código que renderiza una parte de la interfaz. Los componentes pueden ser parametrizados, reutilizados y pueden contener su propio estado.

En React los componentes se crean usando funciones o clases.

¿Cuál es la diferencia entre componente y elemento en React?

Un componente es una función o clase que recibe props y devuelve un elemento. Un elemento es un objeto que representa un nodo del DOM o una instancia de un componente de React.

```
// Elemento que representa un nodo del DOM
{
  type: 'button',
  props: {
    className: 'button button-blue',
    children: [
      type: 'b',
      props: {
        children: 'OK!'
      }
    ]
  }
}

// Elemento que representa una instancia de un componente
{
  type: Button,
  props: {
    color: 'blue',
    children: 'OK!'
  }
}
```

¿Qué son las props en React?

Las props son las propiedades de un componente. Son datos que se pasan de un componente a otro. Por ejemplo, si tienes un componente Button que muestra un botón, puedes pasarle una prop text para que el botón muestre ese texto:

```
function Button(props) {
  return <button>{props.text}</button>
}
```

Podríamos entender que el componente Button es un botón genérico, y que la prop text es el texto que se muestra en el botón. Así estamos creando un componente reutilizable.



@hdtoledo

Debe considerarse además que al usar cualquier expresión JavaScript dentro de JSX debe envolverlos con {}, en este caso el objeto props, de otra forma JSX lo considerará como texto plano.

Para usarlo, indicamos el nombre del componente y le pasamos las props que queremos:

```
<Button text="Haz clic aquí" />
<Button text="Seguir a @hdtoledo" />
```

¿Qué son los hooks?

Los Hooks son una API de React que nos permite tener estado, y otras características de React, en los componentes creados con una function.

Esto, antes, no era posible y nos obligaba a crear un componente con class para poder acceder a todas las posibilidades de la librería.

Hooks es gancho y, precisamente, lo que hacen, es que te permiten enganchar tus componentes funcionales a todas las características que ofrece React.

¿Qué hace el hook `useState`?

El hook **useState** es un componente fundamental de React que permite agregar estado a componentes funcionales. Antes de la introducción de los hooks en React, los componentes funcionales no tenían capacidad para mantener estado interno. Sin embargo, con el uso del hook **useState**, ahora es posible declarar y actualizar el estado en componentes funcionales.

La función **useState** es un hook incorporado en React que devuelve un array con dos elementos: la variable de estado actual y una función para actualizar esa variable. La sintaxis básica para utilizar **useState** es la siguiente:

```
const [state, setState] = useState(initialState);
```

Donde:

- **state**: es la variable que representa el estado actual.
- **setState**: es la función utilizada para actualizar el estado.
- **initialState**: es el valor inicial del estado.

Cuando se invoca **useState**, se devuelve un array donde el primer elemento (**state**) es el valor actual del estado y el segundo elemento (**setState**) es una función que permite actualizar ese estado. Al llamar a **setState** con un nuevo valor, React re-renderiza el componente y actualiza el valor del estado.

Aquí tienes un ejemplo simple de cómo utilizar **useState**:



```

import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}

```

En el ejemplo anterior, el componente **Counter** tiene un estado llamado **count**, inicializado en 0 mediante **useState(0)**. La función **increment** utiliza **setCount** para actualizar el valor de **count** cada vez que se hace clic en el botón. El nuevo valor se refleja automáticamente en la interfaz de usuario a través de la expresión **{count}** en el componente JSX.

En resumen, el hook **useState** en React permite agregar y actualizar el estado en componentes funcionales, lo que facilita la gestión de datos dinámicos y el re-renderizado de componentes en respuesta a cambios en el estado.

Empecemos creando una app de REACT

Vamos a abrir nuestra terminal y allí vamos a utilizar el siguiente comando: **npm create vite**

```

Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y|

```

Colocamos **Y** y presionamos enter, a continuación, nos preguntara cual es el nombre del proyecto:

```

Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
? Project name: » vite-project|

```

Vamos a colocar **react_basics** y presionamos enter:



```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
? Select a framework: » - Use arrow-keys. Return to submit.
>  Vanilla
  Vue
  React
  Preact
  Lit
  Svelte
  Others
```

Acá nos permite seleccionar el framework que vamos a utilizar en este caso seleccionamos React y presionamos enter:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
✓ Select a framework: » React
? Select a variant: » - Use arrow-keys. Return to submit.
>  TypeScript
  TypeScript + SWC
  JavaScript
  JavaScript + SWC
```

Ahora seleccionamos javascript y presionamos enter:

```
Microsoft Windows [Versión 10.0.22621.1702]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\DATA\Documents>npm create vite
Need to install the following packages:
  create-vite@4.3.1
Ok to proceed? (y) y
✓ Project name: ... react_basics
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in D:\DATA\Documents\react_basics...

Done. Now run:

  cd react_basics
  npm install
  npm run dev

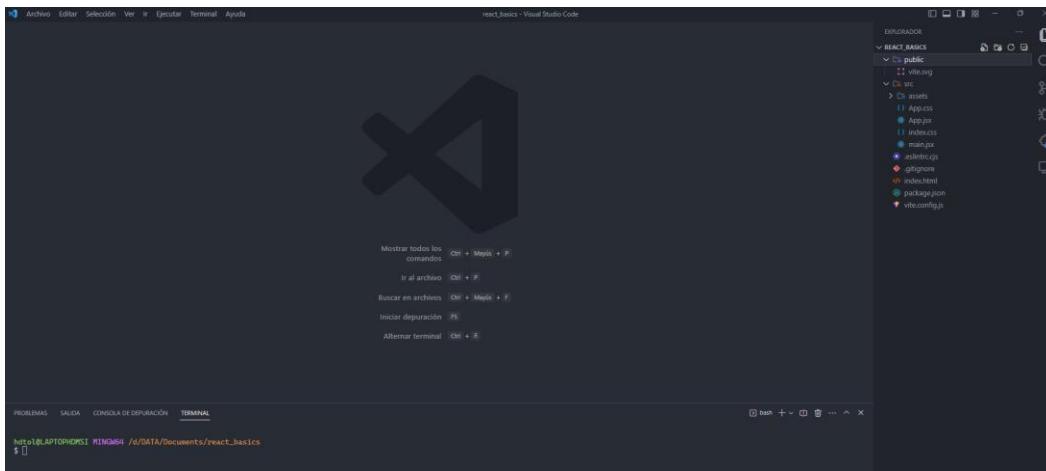
D:\DATA\Documents>
```

Ahora ingresamos a nuestra carpeta recién creada **react_basics** y allí vamos a abrir nuestro VS code con el comando **code .**

```
D:\DATA\Documents\react_basics>code .
```



@hdtoledo



Ahora en nuestro terminal vamos a hacer la instalación de las dependencias a través del comando **npm install**

```
htd@LAPTOPHDMI MINGW64 /d/DATA/Documents/react_basics
$ npm install

added 239 packages, and audited 240 packages in 27s

80 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
```

Ahora vamos a ejecutar el comando **npm run dev** para abrir nuestro servidor y poder ver el proyecto en la web:

```
htd@LAPTOPHDMI MINGW64 /d/DATA/Documents/react_basics
$ npm run dev

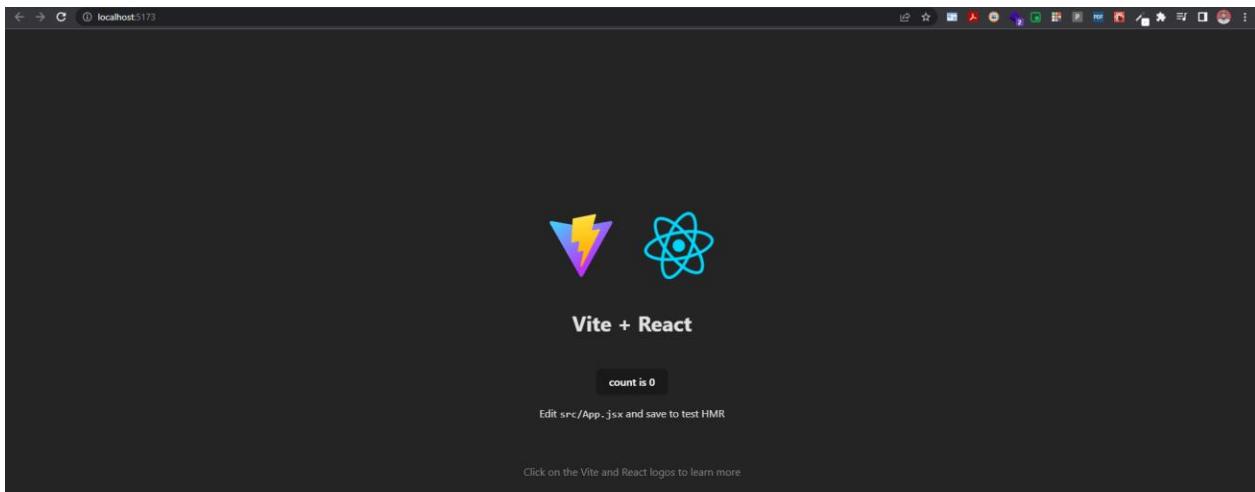
> react_basics@0.0.0 dev
> vite

VITE v4.3.8 ready in 482 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

De esta manera ya tendremos nuestro proyecto ejecutándose en el localhost:





Hablemos de [Vite](#) es una herramienta de desarrollo web rápida y liviana creada por Evan You, el creador de Vue.js. Está diseñada para mejorar la experiencia de desarrollo al permitir un tiempo de compilación instantáneo y un servidor de desarrollo extremadamente rápido.

A diferencia de otras herramientas de construcción como webpack o Parcel, que se basan en la creación de un único paquete final antes de ejecutar la aplicación, Vite adopta un enfoque de desarrollo basado en la modularidad y la carga bajo demanda.

En lugar de agrupar todos los archivos de la aplicación en un solo archivo en tiempo de compilación, Vite utiliza la capacidad de los navegadores modernos para importar módulos directamente desde el sistema de archivos local durante el desarrollo. Esto significa que cada módulo individual se compila y se sirve por separado, lo que resulta en un tiempo de compilación y recarga instantáneos cuando se realizan cambios en el código fuente.

Vite también tiene una integración nativa con los marcos de JavaScript más populares, como Vue.js, React y Preact, lo que permite una configuración sencilla y un flujo de desarrollo optimizado para estas tecnologías.

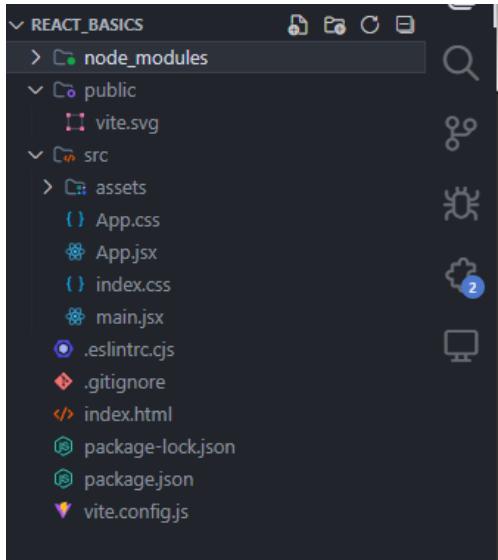
Además de su velocidad de desarrollo, Vite ofrece otras características útiles, como la recarga en caliente (hot module replacement) que actualiza solo los componentes modificados en lugar de recargar toda la página, un entorno de desarrollo con soporte para TypeScript y CSS preprocesados, y la capacidad de generar un paquete optimizado y minificado para la producción.

En resumen, Vite es una herramienta de desarrollo web rápida y eficiente que mejora la experiencia de desarrollo al ofrecer un tiempo de compilación instantáneo, un servidor de desarrollo veloz y una carga bajo demanda de módulos. Es especialmente popular en combinación con frameworks como Vue.js, React y Preact.

Hablemos de las carpetas de nuestro proyecto

La estructura de carpetas de un proyecto de React utilizando Vite sigue una convención comúnmente utilizada en el ecosistema de React. A continuación, se describe cada una de las carpetas principales y para qué se utilizan:





1. **node_modules**: Esta carpeta se crea automáticamente al instalar las dependencias del proyecto utilizando herramientas como npm (Node Package Manager) o Yarn. Aquí se almacenan todas las bibliotecas y paquetes de terceros que son necesarios para el funcionamiento del proyecto.

2. **public**: En esta carpeta se almacenan los archivos estáticos que se servirán directamente al navegador, como imágenes, archivos CSS, fuentes, etc. Estos archivos se copian directamente en la raíz del directorio de salida al construir el proyecto.

3. **src**: Esta carpeta es donde se ubica el código fuente de la aplicación. A continuación, se describen las subcarpetas y archivos más comunes dentro de la carpeta **src**:

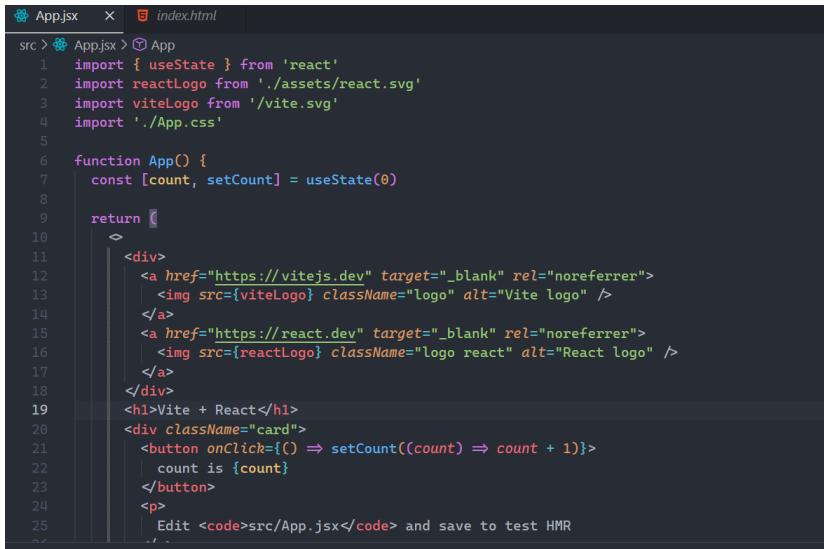
- **index.html**: Es el archivo HTML principal de la aplicación. Aquí se incluirán los puntos de montaje de la aplicación React y se vincularán los archivos CSS o scripts necesarios.
 - **main.js o index.js**: Es el punto de entrada principal de la aplicación. Aquí se configura y se inicia la aplicación React.
 - **components**: Esta carpeta se utiliza para almacenar los componentes de React reutilizables. Cada componente generalmente se coloca en su propio archivo.
 - **pages**: Aquí se pueden almacenar los componentes específicos de cada página de la aplicación. Por lo general, cada archivo en esta carpeta representa una página o una ruta de la aplicación.
 - **assets**: En esta carpeta se pueden almacenar archivos estáticos adicionales, como imágenes, iconos o archivos de datos.
 - **styles**: Aquí se pueden colocar los archivos CSS o preprocesados (como Sass o Less) utilizados en la aplicación.
 - **utils**: En esta carpeta se pueden almacenar funciones de utilidad, helpers o módulos que se utilizan en varias partes del proyecto.
 - **services**: Esta carpeta se utiliza para almacenar módulos de servicios o API que interactúan con servicios externos, como solicitudes HTTP o almacenamiento local.
4. **dist**: Esta carpeta se crea cuando se realiza una compilación o construcción del proyecto. Contiene los archivos estáticos generados y optimizados listos para ser desplegados en producción.
5. **public/index.html**: Este archivo es una plantilla HTML que se utiliza durante el desarrollo y se utiliza para servir la aplicación React. Durante la construcción, este archivo se reemplaza por el archivo **index.html** ubicado en la carpeta **public**.



@hdtoledo

Estas son las carpetas y archivos principales que se encuentran comúnmente en un proyecto de React utilizando Vite. Sin embargo, ten en cuenta que la estructura puede variar dependiendo de las preferencias y necesidades del proyecto, así como de las convenciones establecidas por el equipo de desarrollo.

Conozcamos nuestro código, abriremos App.jsx y lucirá así :



```
App.jsx  X  index.html
src > App.jsx > App
1 import { useState } from 'react'
2 import reactLogo from './assets/react.svg'
3 import viteLogo from '/vite.svg'
4 import './App.css'
5
6 function App() {
7   const [count, setCount] = useState(0)
8
9   return (
10    >
11    <div>
12      <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
13        <img src={viteLogo} className="logo" alt="Vite logo" />
14      </a>
15      <a href="https://react.dev" target="_blank" rel="noreferrer">
16        <img src={reactLogo} className="logo react" alt="React logo" />
17      </a>
18    </div>
19    <h1>Vite + React</h1>
20    <div className="card">
21      <button onClick={() => setCount((count) => count + 1)}>
22        count is {count}
23      </button>
24    </div>
25    <p>Edit <code>src/App.jsx</code> and save to test HMR
...`
```

La primera parte de nuestro código son las importaciones:

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import viteLogo from '/vite.svg'
import './App.css'
```

Aquí, se importa la función **useState** de React, que es un hook para agregar el estado a componentes funcionales. Luego, se importan dos imágenes (**reactLogo** y **viteLogo**) y el archivo **App.css**, que contiene estilos para la aplicación.

Luego tendremos nuestro componente App:



```

function App() {
  const [count, setCount] = useState(0)

  return (
    <>
      <div>
        <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
          <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank" rel="noreferrer">
          <img src={reactLogo} className="logo react" alt="React logo" />
        </a>
      </div>
      <h1>Vite + React</h1>
      <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
          count is {count}
        </button>
        <p>
          Edit <code>src/App.jsx</code> and save to test HMR
        </p>
      </div>
      <p className="read-the-docs">
        Click on the Vite and React logos to learn more
      </p>
    </>
  )
}

export default App

```

- Cuando se carga la aplicación, se muestra una página con los logotipos de Vite y React como enlaces. Al hacer clic en estos logotipos, los usuarios serán dirigidos a sus respectivos sitios web (Vite y React) en una nueva pestaña del navegador.
- Debajo de los logotipos, hay un título que dice "Vite + React".
- Luego, hay una tarjeta que contiene un botón y un párrafo. Al hacer clic en el botón, se incrementa el valor del contador **count**.
- Además, hay un párrafo que indica al usuario que edite el archivo **src/App.jsx** y lo guarde para probar la recarga en caliente (HMR - Hot Module Replacement).
- Por último, hay otro párrafo que sugiere a los usuarios hacer clic en los logotipos de Vite y React para obtener más información.

Hagamos una prueba, modificando el H1 y colando otro texto:

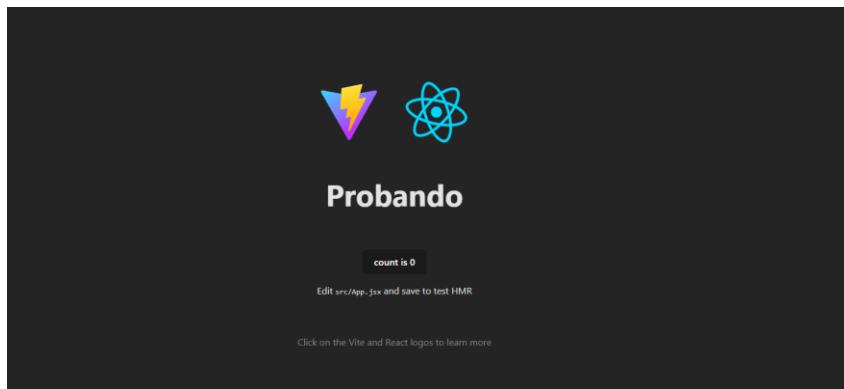
```

15   <a href="https://react.dev" target="_blank" rel="noreferrer">
16     <img src={reactLogo} className="logo react" alt="React logo" />
17   </a>
18 </div>
19 <h1>Probando</h1>
20 <div className="card">
21   <button onClick={() => setCount((count) => count + 1)}>
22     count is {count}
23   </button>

```

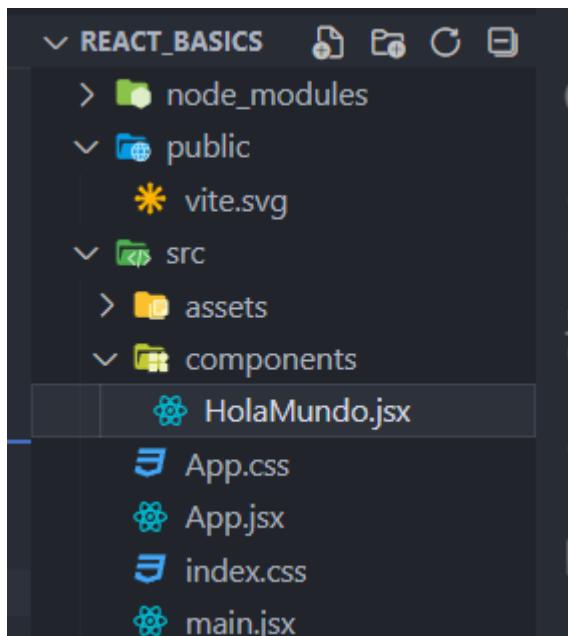


@hdtoledo



De esta manera estamos verificando que podemos cambiar todo lo que encontramos acá y que tenemos prácticamente una plantilla lista para empezar con nuestro proyecto.

Todo componente debe ir en la carpeta SRC, para ello vamos a crear una carpeta que se llame Componentes, allí crearemos nuestro componente HolaMundo.jsx, recordemos que todo componente inicia con mayúsculas por regla general.



Ahora para iniciar nuestro componente, lo primero es realizar la importación de react

```
src > components > HolaMundo.jsx > default
1 import React from "react";
```

Luego creamos nuestra función, y acá muy importante los nombres de las funciones deben iniciar con mayúsculas de igual manera que como lo hacemos con los componentes, dentro de nuestra función siempre debe ir un return que es el que nos devolverá lo que le indiquemos dentro.

```

2
3 | function HolaMundo() {
4 |   return(
5 |     <div>
6 |       <h1>Hola a Todos !</h1>
7 |       <h2>Bienvenidos a REACT Basics</h2>
8 |     </div>
9 |
10 }
11

```

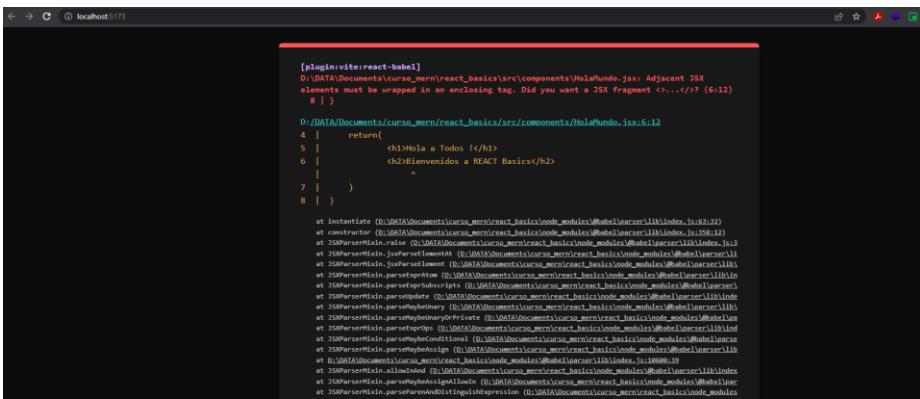
Este return solo podrá devolver un solo objeto a la vez, es decir si colocamos por fuera el h1 y h2 sucedera lo siguiente:

```

src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo() {
4   return[
5     <h1>Hola a Todos !</h1>    Las expresiones JSX deben tener un elemento primario.
6     <h2>Bienvenidos a REACT Basics</h2>
7   ]
8 }

```

Y en nuestro navegador tendremos lo siguiente:



Si no queremos tener un div también podemos utilizar un fragment que simplemente tendrá la siguiente sintaxis `<></>`

```

return(
  <>
    <h1>Hola a Todos !</h1>
    <h2>Bienvenidos a REACT Basics</h2>
  </>
)

```

Y por último siempre debemos exportar nuestro componente para ello hay dos maneras, una al finalizar colocamos `export default NombreComponente`



```
11
12  export default HolaMundo|
```

Y la otra forma es simplemente en la parte de arriba donde mencionamos la función le colocamos el export default.

```
❖ HolaMundo.jsx ×
src > components > ❖ HolaMundo.jsx > ...
1  import React from "react";
2
3  export default function HolaMundo() {
4      return(
5          <div>
6              <h1>Hola a Todos !</h1>
7              <h2>Bienvenidos a REACT Basics</h2>
8          </div>
9      )
10 }
```

Siempre debe haber un export default único, no puede haber dos defaults en el mismo componente.

Ahora vamos a nuestro App.jsx y hacemos la importación del componente

```
❖ HolaMundo.jsx    ❖ App.jsx ×
src > ❖ App.jsx > ...
1  import { useState } from 'react'
2  import reactLogo from './assets/react.svg'
3  import viteLogo from '/vite.svg'
4  import './App.css'
5  import HolaMundo from './components/HolaMundo'
6
7  function App() {
```

Lo siguiente es añadir nuestro componente en nuestro código, como si fuera un HTML

```
19      </div>
20      < HolaMundo/>|
21      </>
22
```

De esta manera quedaría nuestro componente

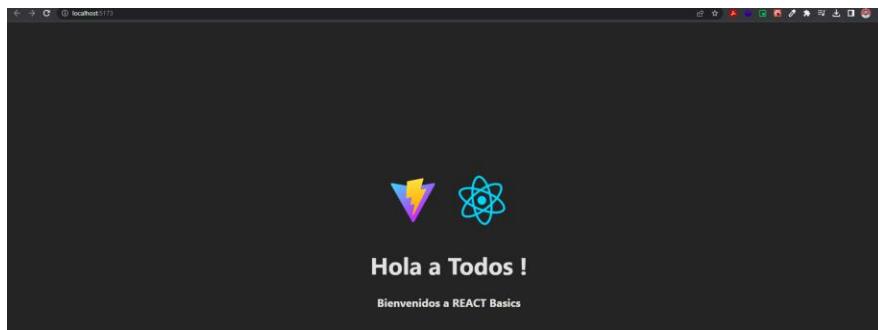


```

❶ HolaMundo.jsx ❷ App.jsx ❸
src > App.jsx > ⚡ App
1 import { useState } from 'react'
2 import reactLogo from './assets/react.svg'
3 import viteLogo from '/vite.svg'
4 import './App.css'
5 import HolaMundo from './components/HolaMundo'
6
7 function App() {
8   const [count, setCount] = useState(0)
9
10  return (
11    <>
12      <div>
13        <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
14          <img src={viteLogo} className="logo" alt="Vite logo" />
15        </a>
16        <a href="https://react.dev" target="_blank" rel="noreferrer">
17          <img src={reactLogo} className="logo react" alt="React logo" />
18        </a>
19      </div>
20      <HolaMundo/>
21    </>
22  )
23}
24
25 export default App
26

```

Y así se vería en nuestra vista:



Modifiquemos un poco nuestro componente HolaMundo, agregamos uno que se llame AdiosMundo y hacemos la exportación sin default:

```

❶ HolaMundo.jsx ❷ App.jsx ❸
src > components > HolaMundo.jsx > ⚡ AdiosMundo
1 import React from "react";
2
3 export default function HolaMundo() {
4   return(
5     <div>
6       <h1>Hola a Todos !</h1>
7       <h2>Bienvenidos a REACT Basics</h2>
8     </div>
9   )
10 }
11
12 export function AdiosMundo(){
13   return(
14     <div>
15       <h3>Adios a Todos!</h3>
16     </div>
17   )
18 }

```

Para hacer la importación de este componente debemos agregarlo no en una línea aparte sino en la misma importación que tenemos de HolaMundo, agregando una coma y utilizando las llaves con el nombre del componente dentro:

```
4  import './App.css'
5  import HolaMundo, { AdiosMundo } from './components/HolaMundo'
6
```

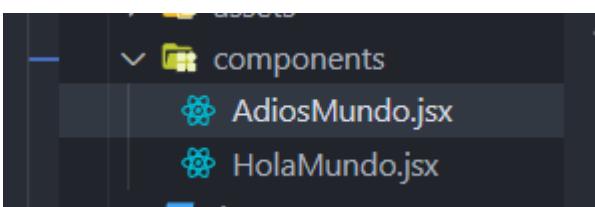
Y de esta manera procedemos a añadir nuestro nuevo componente:

```
18    ||    </a>
19    ||    </div>
20    ||    < HolaMundo/>
21    ||    < AdiosMundo/>
22    ||    </>
23  }
24 }
```

Y nos quedaría de la siguiente manera:



Lo ideal es que generemos componentes separados para no causar equivocaciones, ahora crearemos un componente aparte que se llame AdiosMundo:



Y dentro de nuestro `.jsx` colocaremos de una manera mas breve el código, utilizando `rfce` (que es una abreviación de react functional component export) quedando así:



```
src > components > AdiosMundo.jsx
```

1 rfce	
└─ rfce	reactFunctionalExportComponent
└─ rfce	reactFunctionalExportComponent
└─ rfcredux	reactFunctionalComponentRedux

Para que nos salga debemos tener la extensión reactjs code snippets.

```
src > components > AdiosMundo.jsx > AdiosMundo
```

```
1 import React from 'react'
2
3 function AdiosMundo() {
4   return (
5     <div>AdiosMundo</div>
6   )
7 }
8
9 export default AdiosMundo
```

De esta manera simplificaremos el código y nos creara nuestro componente con su export, hacemos la importación en App.jsx y procedemos a revisarlo en nuestro navegador

```
src > App.jsx > ...
1 import { useState } from 'react'
2 import reactLogo from './assets/react.svg'
3 import viteLogo from '/vite.svg'
4 import './App.css'
5 import HolaMundo from './components/HolaMundo'
6 import AdiosMundo from './components/AdiosMundo'
7
8
9 function App() {
10   const [count, setCount] = useState(0)
11
12   return (
13     <>
14       <div>
15         <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
16           <img src={viteLogo} className="logo" alt="Vite logo" />
17         </a>
18         <a href="https://react.dev" target="_blank" rel="noreferrer">
19           <img src={reactLogo} className="logo react" alt="React logo" />
20         </a>
21       </div>
22       < HolaMundo/>
23       < AdiosMundo/>
24     </>
25   )
26
27
28 export default App
29
```



@hdtoledo



Hola a Todos !

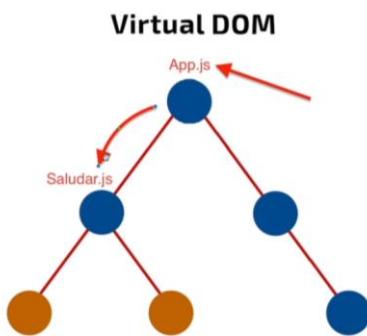
Bienvenidos a REACT Basics

Adios Mundo

No olvidemos eliminar el **AdiosMundo** anterior.

¿Qué son los props en react?

Los Props en react es una zona en donde pasamos información de un componente a otro:



Supongamos que nuestra **app** empieza por **app.js** y es el componente padre, este tiene componentes hijos como lo es **saludar.js**, en el ejercicio imaginamos que en **app.js** nos muestra un saludo: Hola, (**nombreUsuario**) y esto nos mostraría un saludo con el nombre del usuario, este mismo se pasara a **saludar.js** y tiene acceso a la información del usuario que estamos pasando por allí, es decir que la información de este componente padre no termina en **saludar.js** sino que continua vigente y se puede utilizar en los demás componentes.

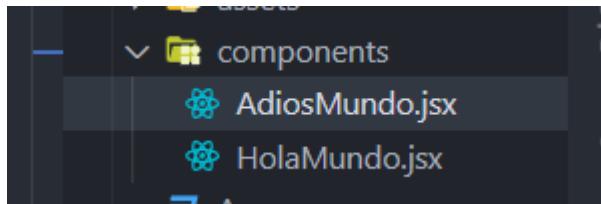
Los **props** permiten pasar información entre componentes y esta información puede venir de diferentes componentes.



@hdtoledo

Pasando Props básicos entre componentes

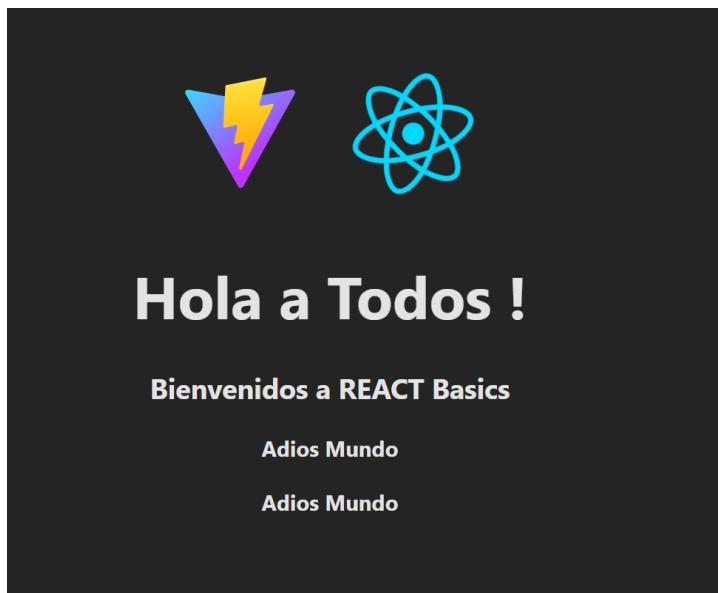
En nuestro proyecto de ejemplo actual tenemos dos componentes:



Que se están visualizando en app.jsx

```
src > App.jsx > App
8
9  function App() {
10    const [count, setCount] = useState(0)
11
12    return (
13      <>
14        <div>
15          <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
16            <img src={viteLogo} className="logo" alt="Vite logo" />
17          </a>
18          <a href="https://react.dev" target="_blank" rel="noreferrer">
19            <img src={reactLogo} className="logo react" alt="React logo" />
20          </a>
21        </div>
22        <HolaMundo/>
23        <AdiosMundo/>
24      </>
25    )
26  }
27
28 export default App
```

Si revisamos estos componentes son estáticos y no traen **props** lo mismo sucede con que no son reutilizables ya que si duplicamos uno de ellos nos repetirá el contenido



Para el ejercicio vamos a pasar una propiedad tipo texto a nuestro componente HolaMundo, y de esta manera podremos hacer que el componente sea reutilizable.



Vamos a modificar nuestro **HolaMundo** que actualmente se encuentra así:

```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo() {
4
5   return(
6     <div>
7       <h1>Hola a Todos !</h1>
8       <h2>Bienvenidos a REACT Basics</h2>
9     </div>
10  );
11}
```

Y lo vamos a dejar de la siguiente manera:

```
src > components > HolaMundo.jsx > ...
1 import React from "react";
2
3 export default function HolaMundo() {
4   return (
5     <div>
6       <h1>Bienvenido Usuario</h1>
7     </div>
8   );
9 }
10
```



Ahora para entender cómo vamos a pasar las **props** en nuestra función vamos a agregar la palabra **props** dentro de los parámetros de **HolaMundo**



```

src > components > HolaMundo.jsx > ...
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props)
5   return (
6     <div>
7       <h1>Bienvenido Usuario</h1>
8     </div>
9   );
10 }
11

```

Vamos a revisar en nuestro navegador como está pasando los **props** a través de la consola

The screenshot shows a browser window at localhost:5173. On the left, there are two logos: Vite and React. Below them, the text "Bienvenido Usuario" is displayed in large white font. At the bottom, there are two buttons: "Adios Mundo" and "Hacer otra vez". On the right, the browser's developer tools are open, specifically the "Console" tab. The console output shows the following:

```

Object { }
  constructor: f_ Object()
  hasOwnProperty: f_ hasOwnProperty()
  isPrototypeOf: f_ isPrototypeOf()
  propertyIsEnumerable: f_ propertyIsEnumerable()
  toLocaleString: f_ toLocaleString()
  toString: f_ toString()
  valueOf: f_ valueOf()
  _defineGetter: f_ _defineGetter_()
  _defineSetter: f_ _defineSetter_()
  _lookupGetter: f_ _lookupGetter_()
  _lookupSetter: f_ _lookupSetter_()
  __proto__: (...)

  get __proto__: f_ __proto__()
  set __proto__: f_ __proto__()

  installHook.js:251

```

The object is empty, which corresponds to the props object passed to the HolaMundo component.

Acá podemos observar que nos llega un objeto completamente vacío, lo que vamos a hacer es pasarle una propiedad a través de nuestro componente, es decir vamos a editar nuestro **App.jsx** y lo dejamos de la siguiente manera:

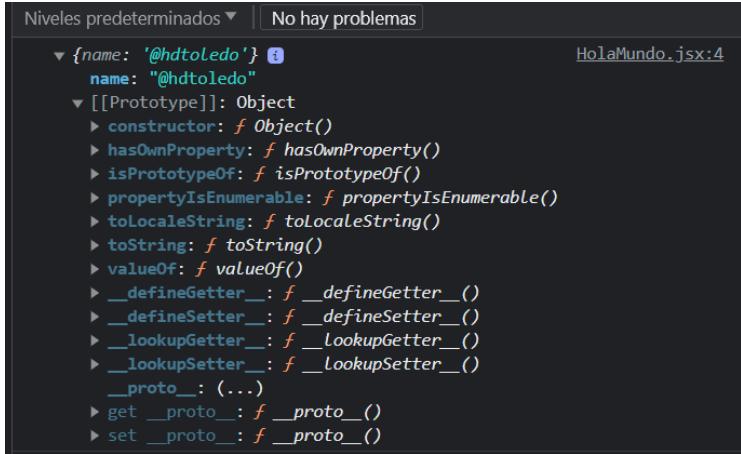
```

src > App.jsx > ...
8
9 function App() {
10   const [count, setCount] = useState(0)
11
12   return (
13     <>
14       <div>
15         <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
16           <img src={viteLogo} className="logo" alt="Vite logo" />
17         </a>
18         <a href="https://react.dev" target="_blank" rel="noreferrer">
19           <img src={reactLogo} className="logo react" alt="React logo" />
20         </a>
21       </div>
22       < HolaMundo name="@hdtoledo" />
23       < AdiosMundo />
24     </>
25   )
26 }
27
28 export default App
29

```

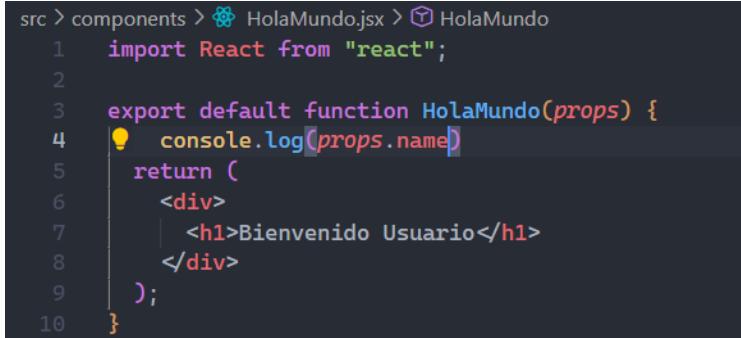


Pasamos la propiedad **name** con un **string** y al verlo reflejado en el navegador observaremos lo siguiente:



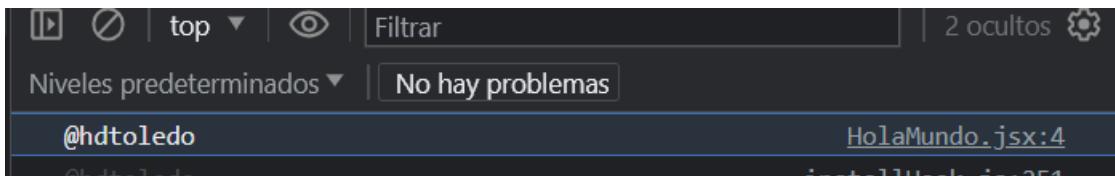
The screenshot shows the Chrome DevTools Elements tab with the title "Niveles predeterminados" and "No hay problemas". A dropdown menu is open, showing the properties of the "name" prop. The properties listed are: name: '@hdtoledo', name: '@hdtoledo', [[Prototype]], constructor, hasOwnProperty, isPrototypeOf, propertyIsEnumerable, toLocaleString, toString, valueOf, __defineGetter__, __defineSetter__, __lookupGetter__, __lookupSetter__, __proto__, get __proto__, and set __proto__. The file "HolaMundo.jsx:4" is indicated next to the first property.

Ahora para acceder solamente a la propiedad del objeto hacemos los siguiente:



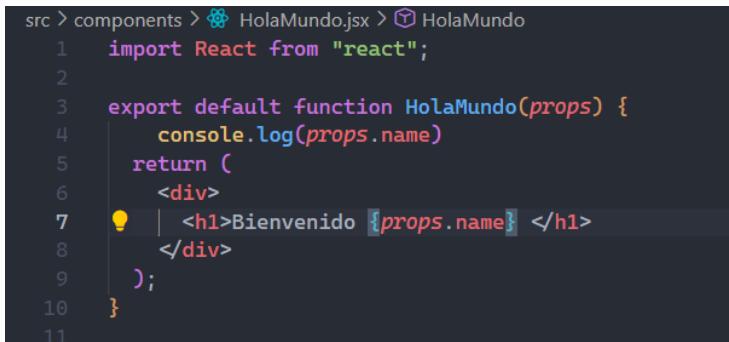
```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props.name)
5   return (
6     <div>
7       <h1>Bienvenido Usuario</h1>
8     </div>
9   );
10 }
```

Simplemente añadimos `props.name` para poder acceder y lo que vemos reflejado en la consola



The screenshot shows the Chrome DevTools Elements tab with the title "Niveles predeterminados" and "No hay problemas". A dropdown menu is open, showing the output of the console.log statement: "@hdtoledo". The file "HolaMundo.jsx:4" is indicated next to the output.

Nuestra propiedad **name** dentro de **props**, ahora simplemente utilizamos las llaves para poder llamar la propiedad `name` dentro y nos quedaría así:



```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props.name)
5   return (
6     <div>
7       <h1>Bienvenido {props.name} </h1>
8     </div>
9   );
10 }
```



Y en nuestro navegador:



De esta manera ya estamos utilizando correctamente nuestro componente para pasarle props y tambien podemos duplicarlo y reutilizarlo, haciendo lo siguiente:

```
    </div>
  < HolaMundo name="@hdtoledo" />
  < HolaMundo />

  < AdiosMundo />
</>
)
```

Si lo dejamos solo, nuestro componente nos mostrara lo siguiente



Quiere decir que podemos pasar otra propiedad de name:



```
21      |     </div>
22      |     < HolaMundo name="@hdtoledo" />
23      |     < HolaMundo name="Diana" />
24
25      |     < AdiosMundo />
26  
```

Ahora en nuestro navegador observamos



También podemos pasar más propiedades dentro, vamos a modificar un poco nuestro HolaMundo:

```
src > components > HolaMundo.jsx > ...
1  import React from "react";
2
3  export default function HolaMundo(props) {
4    console.log(props.name)
5    return (
6      <div>
7        |   <h1>Bienvenido, {props.name} tienes {props.edad} años de edad.</h1>
8        </div>
9    );
10  }
11  
```

Acá podemos observar que vamos a pasar otra propiedad que se llamará edad y la vamos a colocar dentro de nuestro mensaje.

En app.jsx quedaría así:

```
21      |     </div>
22      |     < HolaMundo name="@hdtoledo" edad="37" />
23      |     < AdiosMundo />
24  
```



Y en nuestro navegador quedaría así:



Así de esta manera comprendemos que en un componente podemos tener diferentes props que podemos pasárselos y que podemos reutilizar.

Pasando Variables Objetos entre componentes por los Props

Imaginemos que tenemos 50 datos de personas para pasar a través de nuestro componente, entonces tendríamos que llenar 50 veces nuestro componente con esos datos, lo cual sería una labor bastante tediosa, para ello lo que podemos realizar es pasar variables y objetos dentro de nuestro componente para implementar una solución.

Veamos como pasamos a través de variables el nombre y la edad, modificando de la siguiente manera nuestro `app.jsx`:

```
src > App.jsx > App
  8
  9  function App() {
10
11    const userName = "HD"
12    const edad = 37
13
14    return [
15      <>
16        <div>
17          <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
18            <img src={viteLogo} className="logo" alt="Vite logo" />
19          </a>
20          <a href="https://react.dev" target="_blank" rel="noreferrer">
21            <img src={reactLogo} className="logo react" alt="React logo" />
22          </a>
23        </div>
24        < HolaMundo name={userName} edad={edad} />
25        < AdiosMundo />
26      </>
27    ]
28  }
29
30  export default App
31
```



Cargamos en una **const** las variables **userName** y **edad**, asignamos los valores y por último modificamos nuestras propiedades con las variables utilizando las llaves para poder traer la información de esta manera nos saldría en nuestro navegador:



Para hacer unos pequeños ajustes, vamos a modificar nuestro `console.log` de `HolaMundo` para que se quede solo con los props:

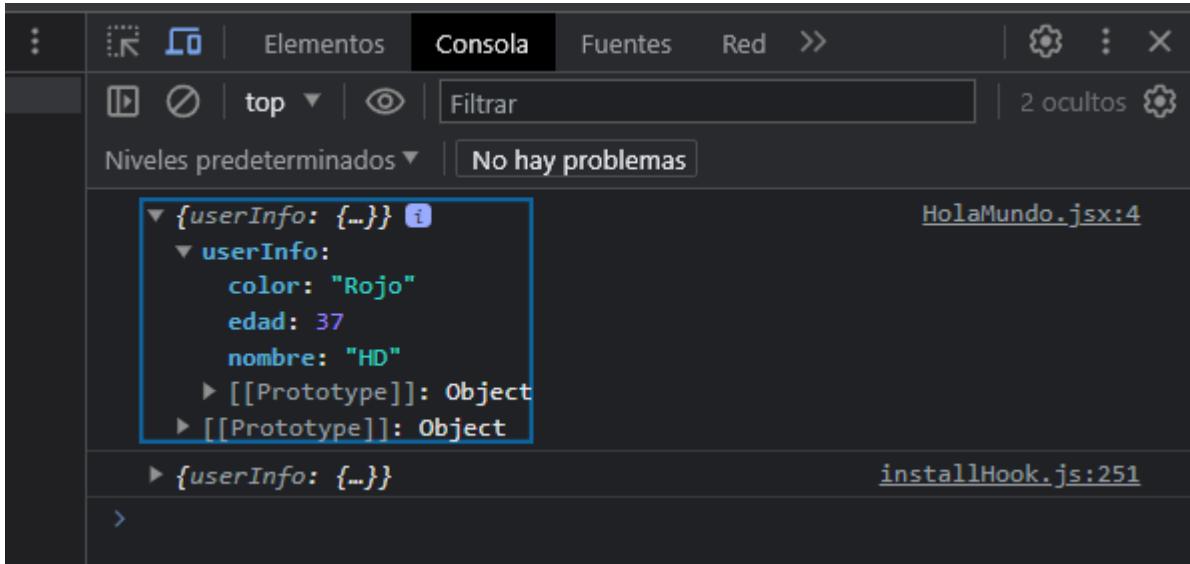
```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props)
5   return (
6     <div>
7       <h1>Bienvenido, {props.name} tienes {props.edad} años de edad.</h1>
8     </div>
9   );
10 }
11
```

Ahora vamos a crear un objeto con unas propiedades del usuario para empezar a utilizarlas mejor dentro de nuestro `app.jsx`:

```
src > App.jsx > ...
8
9 function App() {
10
11   const user = {
12     nombre: "HD",
13     edad: 37,
14     color: "Rojo"
15   }
16
17   return (
18     <>
19       <div>
20         <a href="https://vitejs.dev" target="_blank" rel="noopener">
21           <img src={viteLogo} className="logo" alt="Vite logo" />
22         </a>
23         <a href="https://react.dev" target="_blank" rel="noopener">
24           <img src={reactLogo} className="logo react" alt="React logo" />
25         </a>
26       </div>
27       <HolaMundo userInfo={user} />
28       <AdiosMundo/>
29     </>
30   )
31 }
32
33 export default App
```



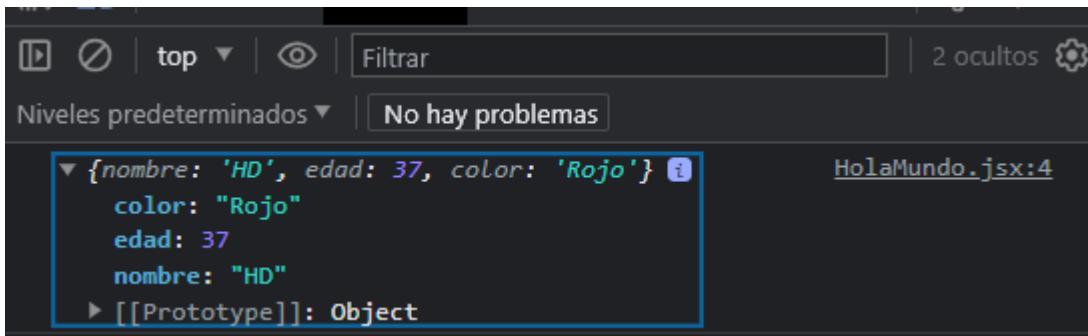
Observemos que hemos creado un **objeto** con las propiedades de **nombre**, **edad**, **color** y que además de ello dentro de nuestro componente **HolaMundo** estamos pasando la información de nuestro usuario, para verificarlo lo podemos ver en nuestro navegador en la consola:



Ahora modificamos un poco el console.log de nuevo para pasar **userInfo** allí:

```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props.userInfo)
5   return (
6     <div>
7       <h1>Bienvenido, {props.name} tienes {props.edad} años de edad.</h1>
8     </div>
9   );
10 }
11
```

De esta manera accedemos a la información también pero ya directamente al objeto



Si queremos acceder solo al nombre del objeto modificamos de la siguiente manera:

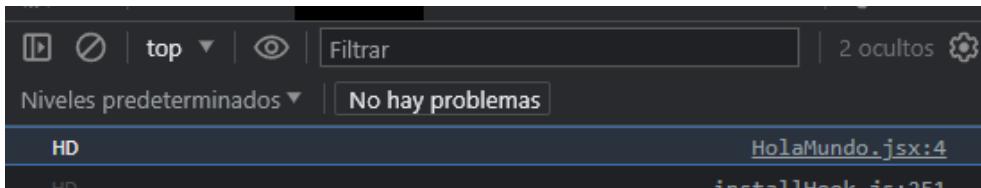


```

src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props.userInfo.nombre)
5   return (
6     <div>
7       <h1>Bienvenido, {props.name} tienes {props.edad} años de edad.</h1>
8     </div>
9   );
10 }

```

Ingresamos directamente al nombre del objeto y saldría así en nuestra consola:



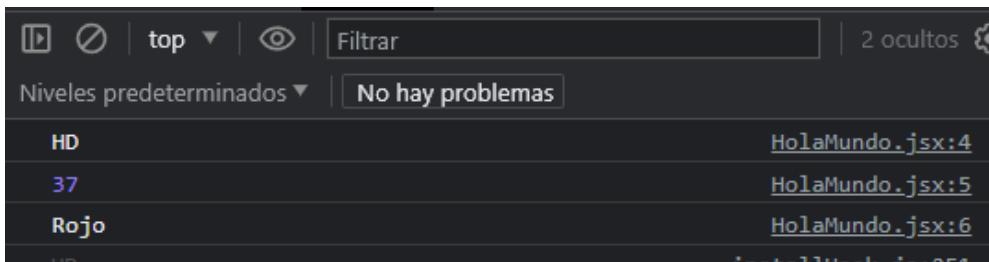
Si colocamos en consola las demás propiedades

```

3 export default function HolaMundo(props) {
4   console.log(props.userInfo.nombre)
5   console.log(props.userInfo.edad)
6   console.log(props.userInfo.color)
7
8   return (
9     <div>
10       <h1>...
11     </div>
12   );
13 }

```

Y de esta manera en consola observaremos los datos



Ahora vamos a realizar unos cambios en nuestro **HolaMundo**:

```

src > components > HolaMundo.jsx > ...
1 import React from "react";
2
3 export default function HolaMundo(props) {
4   console.log(props.userInfo.nombre)
5   console.log(props.userInfo.edad)
6   console.log(props.userInfo.color)
7
8   return (
9     <div>
10       <p>Bienvenido, <b>{props.userInfo.nombre}</b> tienes <b>{props.userInfo.edad}</b> años de edad y tu
11       color favorito es <b>{props.userInfo.color}</b>.</p>
12     </div>
13   );
14 }

```

Cambiamos por un párrafo y ajustamos el mensaje que estamos mostrando mediante las propiedades del objeto **userInfo**, quedándonos de la siguiente manera:





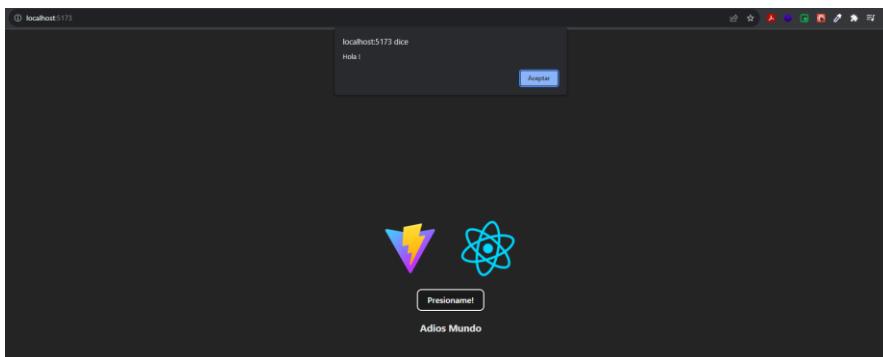
Y de esta manera estamos pasando variables y objetos entre los componentes.

Pasando Funciones entre componentes por los Props

Para pasar las funciones dentro de los componentes e incluir los props vamos a hacer un ejercicio sencillo, realizamos una función tipo flecha para que un botón que simplemente al hacer clic se nos muestre un alert:

```
src > components > HolaMundo.jsx > HolaMundo
1  import React from "react";
2
3  export default function HolaMundo(props) {
4
5    const mensaje = () => {
6      alert('Hola !')
7    }
8
9    return (
10      <div>
11        <button onClick={mensaje}>Presioname!</button>
12      </div>
13    );
14
15 }
```

De esta manera lo visualizamos



Ahora vamos a implementar que al presionar nuestro botón se muestren las props que vamos a enviarle, para ello modificamos nuestro HolaMundo:



```

src > components > HolaMundo.jsx > ...
1  import React from "react";
2
3  export default function HolaMundo(props) {
4    return (
5      <div>
6        <button onClick={() => props.saludarFn(props.userInfo.nombre)}>Presioname!</button>
7      </div>
8    );
9  }
10

```

Vamos a colocar que al hacer clic en nuestro botón se ejecute la función **saludarFn** con el nombre del objeto **userInfo** que tenemos en nuestro **app.jsx**, y dentro de **app.jsx** vamos a crear la función, notemos que para poder ejecutarla dentro de **onClick** realizamos una función tipo flecha para que no se ejecute automáticamente, sino que espere a que demos clic sobre el botón. Y modificamos nuestro **app.jsx** de la siguiente manera:

```

src > App.jsx > App
9  function App() {
10
11    const user = {
12      nombre: "HD",
13      edad: 37,
14      color: "Rojo"
15    }
16
17    const saludarFn = name => {
18      alert("Hola, " + name)
19    }
20
21    return (
22      <>
23        <div>
24          <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
25            <img src={viteLogo} className="logo" alt="Vite logo" />
26          </a>
27          <a href="https://react.dev" target="_blank" rel="noreferrer">
28            <img src={reactLogo} className="logo react" alt="React logo" />
29          </a>
30        </div>
31        <HolaMundo userInfo={user} saludarFn={saludarFn} />
32        <AdiosMundo/>
33      </>
34    )
35  }
36
37  export default App
38

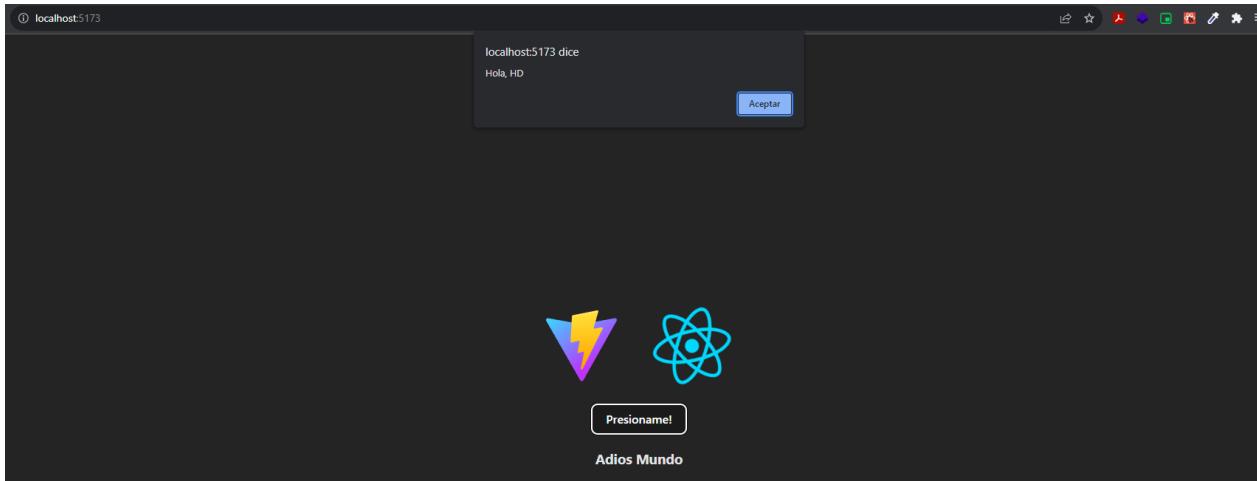
```

Cargamos en una función tipo flecha que se llamará **saludarFn** que va a traer el **name** y lo va a mostrar dentro de un **alert**, además de que pasamos **saludarFn** a través del componente para que se ejecute con los **props** que ya le hemos definido, de esta manera mandamos la función al componente hijo, la



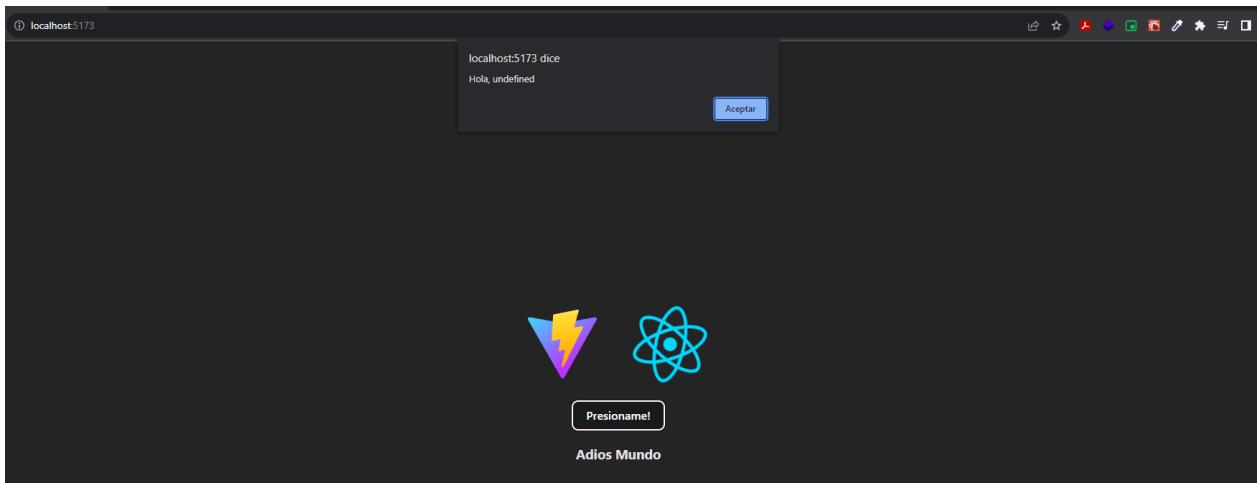
@hdtoledo

ejecutamos y le estamos enviando al componente padre el usuario, de esta forma enviamos props entre ambos tanto al componente padre como al hijo y viceversa.



Ahora que sucede si nuestro objeto no tiene un nombre definido ¿? Pues simplemente nos mostrara undefined:

```
src > ⚛ App.jsx > 🏷 App > 📂 user
  9   function App() {
10
11     const user = {
12       |
13       edad: 37,
14       color: "Rojo"
15     }
16
17     const saludarEn = name => {
```



@hdtoledo

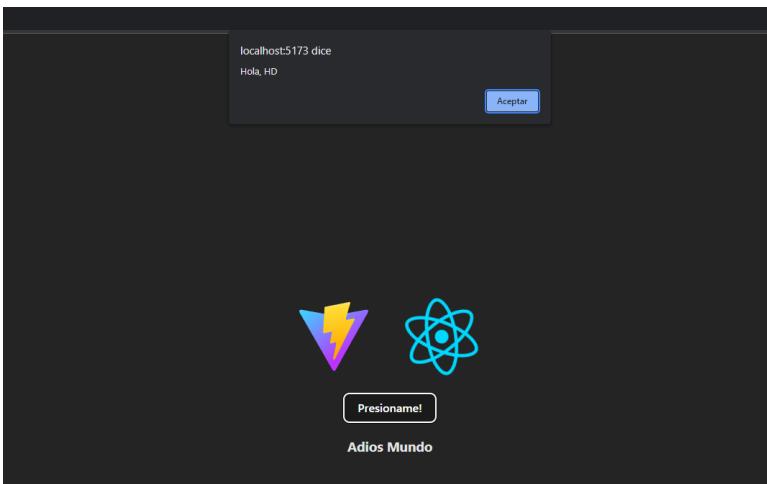
El uso de la asignación por destructuring

Vamos a realizar una **const** dentro de **HolaMundo** con los datos de nuestro **props**, en este vamos a pasarle **userInfo** y **saludarFn**, para ello mostramos a través de consola lo que estamos trayendo:

```
src > components > HolaMundo.jsx > ...
1  import React from "react";
2
3  export default function HolaMundo(props) {
4
5    const {userInfo, saludarFn} = props
6
7    return (
8      <div>
9        <button onClick={() => props.saludarFn(userInfo.nombre)}>Presioname!</button>
10       </div>
11    );
12  }
13
```

Y acá podemos verificar que dentro de nuestro botón no es necesario acceder a **props**, sino que vamos a ingresar a **userInfo** directamente.

Y en nuestro navegador podemos verificar que la información se nos muestra:



También podemos comprobar los datos a través de un `console.log` para que nos los muestre:

```
4
5  const {userInfo, saludarFn} = props
6  console.log(userInfo)
7
8  return (
9
```

localhost:5173 dice

Hola, HD

Aceptar

React Vite

Presioname!

Adios Mundo

No hay problemas

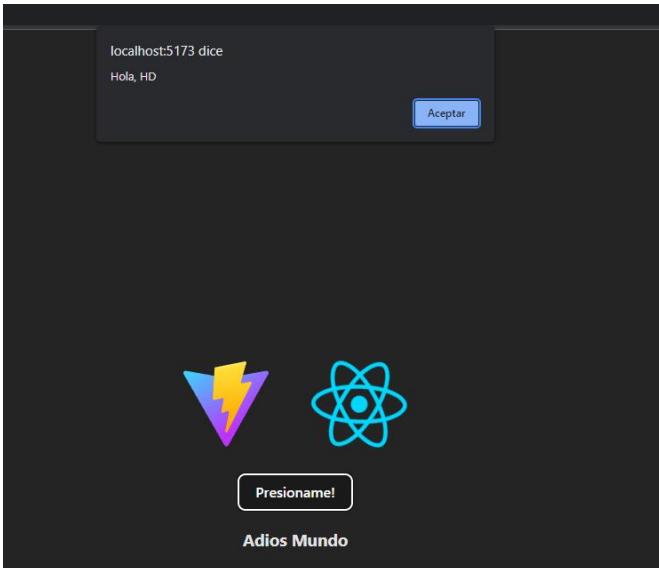
▶ {nombre: 'HD', edad: 37, color: 'Rojo'}

HolaMundo.jsx:6



De la misma manera que estamos accediendo directamente a la información, lo podemos hacer con **saludarFn** sin necesidad de acceder a **props**.

```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4
5   const {userInfo, saludarFn} = props
6   console.log(userInfo)
7
8   return (
9     <div>
10      | <button onClick={() => saludarFn(userInfo.nombre)}>Presioname!</button>
11    </div>
12  );
13}
```



El destructuring lo podemos aplicar directamente sobre nuestros datos asignando directamente sobre una **const** los datos de **userInfo**

```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4
5   const {userInfo, saludarFn} = props
6   const { nombre } = userInfo
7
8   return (
9     <div>
10      | <button onClick={() => saludarFn(nombre)}>Presioname!</button>
11    </div>
12  );
13}
```

Sacamos la información de una variable a otra **const** y así realizamos el destructuring para acceder de una manera mas eficiente y nos ayuda a implementar unas buenas practicas.



Props por default

También podemos asignar **props** por default directamente, si el usuario no trae un nombre debemos asignarle un valor default, para ello vamos a colocarle ese valor de la siguiente manera a **nombre**:

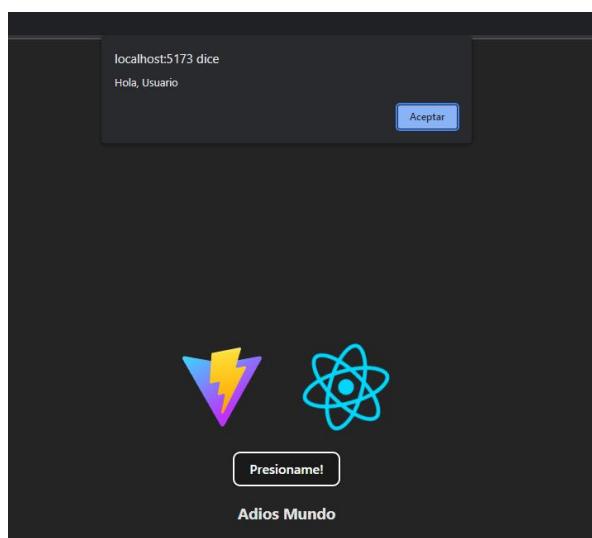
```
src > components > HolaMundo.jsx > HolaMundo > nombre

1 import React from "react";
2
3 export default function HolaMundo(props) {
4
5   const {userInfo, saludarFn} = props
6   const { nombre = "Usuario" } = userInfo
7
8   return (
9     <div>
10       <button onClick={() => saludarFn(nombre)}>
11         Hola, {nombre}
12       </button>
13     </div>
14   );
15 }
```

Para comprobarlo simplemente eliminamos la propiedad de **nombre** en nuestro **app.js**:

```
src > App.jsx > App > user          src > App.jsx > App > user
  9  function App() {                  9
 10    const user = {                  10
 11      nombre: "HD",              11
 12      edad: 37,                 12
 13      color: "Rojo"            13
 14    }                            14
 15  }                            15
 16
```

Y de esta manera al presionar el botón observamos que ya viene el valor por default:



Template Strings o Template Literals

Los "template literals" o "template strings" son una característica introducida en ECMAScript 6 (también conocido como ES6) que permite crear cadenas de texto de una manera más flexible y legible en JavaScript. Antes de la introducción de los template literals, la concatenación de cadenas en JavaScript solía hacerse utilizando el operador `+` o mediante la función `concat()`, lo que a menudo resultaba en código menos legible y más propenso a errores.

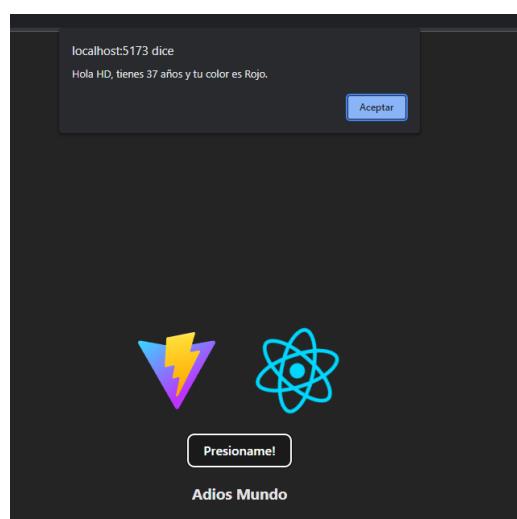
Sintaxis: Los template literals se definen utilizando comillas invertidas (backticks) ``...``, en lugar de comillas simples o dobles, y allí para utilizar las variables abrimos a través del signo de **pesos \$** y **llaves {}** y dentro colocamos la variable, vamos a realizar un pequeño cambio en nuestro código, pasamos primero unas props extras en `app.jsx` agregamos **edad** y **color** a nuestra **const**.

```
src > App.jsx > App
14   color: "Rojo"
15 }
16
17 const saludarFn = (nombre, edad, color) => {
18   alert(`Hola ${nombre}, tienes ${edad} años y tu color es ${color}.`)
19 }
20
21 return (
22   <>
```

Y en nuestro **alert** modificamos con los **backticks** el mensaje e implementamos **\${variable}**, de esta manera pasamos a modificar ahora **HolaMundo**

```
src > components > HolaMundo.jsx > HolaMundo
1 import React from "react";
2
3 export default function HolaMundo(props) {
4
5   const {userInfo, saludarFn} = props
6   const { nombre = "Usuario", edad = 0, color = "Ninguno" } = userInfo
7
8   return (
9     <>
```

Ahora pasamos directamente con el destructuring nuestras variables y les asignamos un valor por default, de esta manera al ir al navegador observaremos como pasamos todos los datos en nuestro **alert**.



Hook de estado – useState

El hook useState es utilizado para crear variables de estado, quiere decir que su valor es dinámico, que este puede cambiar en el tiempo y eso requiere una re-renderización del componente donde se utiliza

Recibe un parámetro:

- El valor inicial de nuestra variable de estado.

Devuelve un array con dos variables:

- En primer lugar, tenemos la variable que contiene el valor
- La siguiente variable es una función set, requiere el nuevo valor del estado, y este modifica el valor de la variable que anteriormente mencionamos
- Cabe destacar que la función proporciona como parámetro el valor actual del propio estado.
Ex: setisOpen(isOpen => !isOpen)

En este ejemplo mostramos como el valor de count se inicializa en 0, y también se renderiza cada vez que el valor es modificado con la función setCount en el evento onClick del button:

```
import { useState } from 'react'

function Counter() {
  const [count, setCount] = useState(0)

  return (
    <>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count => count + 1)}>Aumentar</button>
    </>
  )
}
```

Ahora vamos a colocarlo a prueba, para ello recordemos que tenemos un componente **AdiosMundo**, dentro de este vamos a realizar lo siguiente:

```
src > components > AdiosMundo.jsx > ...
1  import React from 'react'
2
3  function AdiosMundo() {
4
5    const encenderApagar = () => {
6      alert('Encender / Apagar')
7    }
8
9    return (
10      <div>
11        <h3>El carro esta: Encendido</h3>
12        <button onClick={encenderApagar}>Encender / Apagar</button>
13      </div>
14    )
15  }
16  export default AdiosMundo
```

Creamos una **const** función tipo flecha que se llamará **encenderApagar** y nos mostrará un **alert**, y vamos a colocar un **h3** con un mensaje y un **botón** que al ser presionado llamará la función tipo flecha, lo que queremos realizar será que al presionar el botón nos cambie el estado de encendido a apagado, para ello vamos a importar **useState** de la siguiente manera:



```
src > components > AdiosMundo.jsx > ...
1 import React, { useState } from 'react'
2 |
3 function AdiosMundo() {
4 }
```

Y ahora almacenamos en una **const** en forma de array los valores que vamos a pasar y estos los utilizamos a través de **useState** estableciendo un valor booleano

```
src > components > AdiosMundo.jsx > AdiosMundo
1 import React, { useState } from 'react'
2
3 function AdiosMundo() {
4
5   const [stateCar, setStateCar] = useState(false)
6
7   const encenderApagar = () => {
8     alert('Encender / Apagar')
9   }
10 }
```

Y vamos ahora a colocar una condición sobre nuestro mensaje que nos muestre el texto encendido o apagado:

```
10
11   return (
12     <div>
13       <h3>El carro esta: { stateCar ? "Encendido" : "Apagado" }</h3>
14       <button onClick={encenderApagar}>Encender / Apagar</button>
15     </div>
16   )
17 }
18 }
```

Al estar en false nuestro mensaje será Apagado, pero si modificamos a true nos aparecerá encendido:



Para cambiar nuestro estado utilizaremos la función **setStateCar** en donde nos permitirá cambiar su valor, para ello modificamos la función **encenderApagar**:

```
6
7   const encenderApagar = () => {
8     setStateCar(!stateCar)
9   }
10 }
```

El valor que nos recibirá será el contrario al que nosotros establecemos, es decir si esta true, pasara un valor false y viceversa, para comprobarlo presionaremos el botón y veremos como se actualiza.



Hook de Estado – useEffect

El hook useEffect se usa para ejecutar código cuando se renderiza el componente o cuando cambian las dependencias del efecto.

Recibe dos parámetros:

- La función que se ejecutará al cambiar las dependencias o al renderizar el componente.
- Un array de dependencias. Si cambia el valor de alguna dependencia, ejecutará la función.

En este ejemplo mostramos un mensaje en consola cuando carga el componente y cada vez que cambia el valor de count:

```
import { useEffect, useState } from 'react'

function Counter() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    console.log('El contador se ha actualizado')
  }, [count])

  return (
    <>
      <p>Contador: {count}</p>
      <button onClick={() => setCount(count + 1)}>Aumentar</button>
    </>
  )
}
```

Ahora para aplicarlo vamos a crear un contador que lleve la cuenta de cuantos clics hemos dado al botón de encender / apagar, para ello creamos una **const** en donde pasamos **contar** y **setContar**

```
5 |   const [stateCar, setStateCar] = useState(false)
6 |   const [contar, setContar] = useState(0)
7 |
8 |   const encenderApagar = () => {
```

Y ahora lo que vamos a realizar es crear un **useEffect**, esta es su sintaxis:

```
8 |
9 |   useEffect(() => {
10 |
11 |     }, [])
12 |
```

Verificamos que se esté importando:

```
src > components > AdiosMundo.jsx > AdiosMundo
1  import React, { useEffect, useState } from 'react'
2
3  function AdiosMundo()
```

Vamos a agregar antes de usar el **useEffect** un **h4** que nos diga total de **clics** y lo asignamos con **contar**:

```
18 |   <div>
19 |     <h3>El carro esta: { stateCar ? "Encendido" : "Apagado" }</h3>
20 |     <h4>Total de clics: {contar}</h4>
21 |     <button onClick={encenderApagar}>Encender / Apagar</button>
22 |   </div>
```

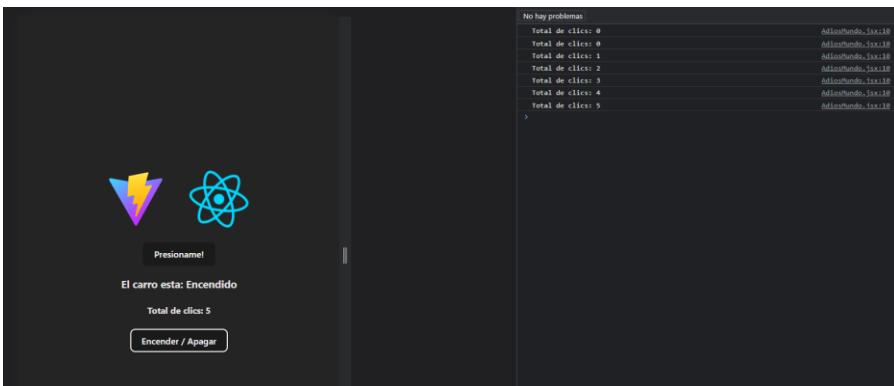
Y ahora vamos a utilizar nuestro **useEffect** de la siguiente manera:

```
8  | useEffect(() => {
9  |   console.log(`Total de clics: ${contar}`)
10 |   , [contar])
11 |
12 | }
```

Mediante un **console.log** llevaremos el conteo también y asignamos contar al **array** para que luego pueda ser utilizado por nuestra función **encenderApagar** que le agregamos la función de **setContar** que sumara la cantidad de clics dados:

```
13 | const encenderApagar = () => {
14 |   setStateCar(!stateCar)
15 |   setContar(contar + 1)
16 | }
17 | }
```

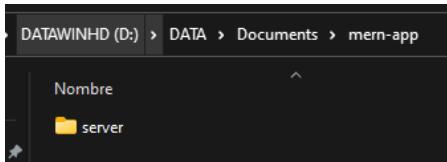
Al revisar en nuestro navegador ya tendremos el conteo tanto en nuestro **h4** como en consola:



Y cada vez que se actualiza la página el contador vuelve a 0.

Creación del proyecto

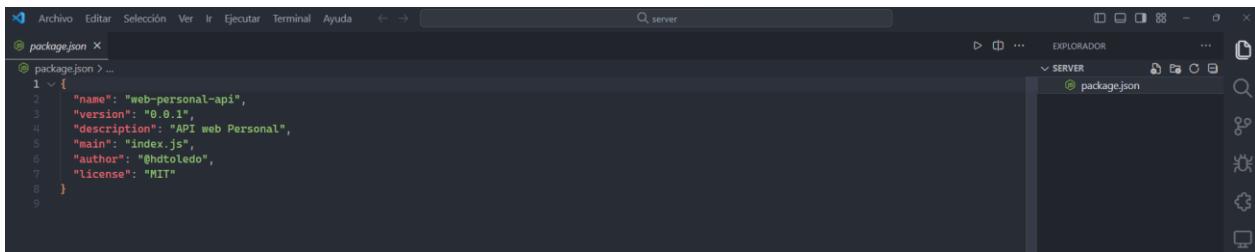
Vamos a iniciar creando nuestro proyecto base, para ello vamos a trabajar en el **frontend** y en el **backend**, también podríamos mencionarlos como el **cliente** y el **server**, lo primero es crear la carpeta principal, en mi caso trabajare en la carpeta **/mern-app** y allí creare una carpeta **server**:



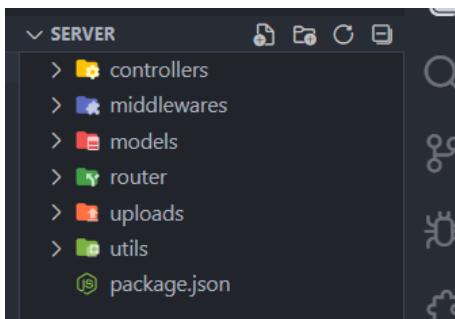
Abrimos nuestra terminal y vamos a entrar a esta carpeta, y colocamos el comando **yarn init**

```
pwsh D:\DATA\Documents\mern-app
> cd .\server\
pwsh D:\DATA\Documents\mern-app\server
> yarn init
yarn init v1.22.19
question name (server): web-personal-api
question version (1.0.0): 0.0.1
question description: API web Personal
question entry point (index.js):
question repository url:
question author: @hdtoledo
question license (MIT):
question private:
success Saved package.json
Done in 58.19s.
```

Aquí nos saldrán una serie de preguntas para iniciar nuestro **package.json**, llenamos cada una de ellas, procedemos a abrirlo en nuestro editor **VS code** y nos mostrara el **package.json**



Ahora procederemos a crear parte de la estructura de nuestro proyecto, es decir creamos las carpetas en la raíz de server:



CREANDO UN CLUSTER MONGO ATLAS

The screenshot shows the MongoDB Atlas landing page. It features a large central text area with the heading "For the next generation of intelligent applications." Below this, there's a brief description: "Build applications on the industry's first developer data platform. From AI-powered and event-driven apps to edge use cases and search, build fast and at the scale users demand." At the bottom, there are two buttons: "Start Free" and "Learn More →". To the right, a separate window titled "Connecting to: mongodb://cluster0.ab123.mongodb.net/myFirstDb" is shown, indicating a successful connection attempt.

Iremos a la página oficial de [mongodb](#) en donde crearemos un **cluster** para nuestra base de datos NOSQL, para ellos nos registramos dentro de mongo y allí veremos lo siguiente:

The screenshot shows the MongoDB Atlas dashboard under the "DEPLOYMENT" tab. It displays a "Database Deployments" section with a "Create a database" button. The button has the text "Choose your cloud provider, region, and specs." and a "Build a Database" button below it. A note below the button says "Once your database is up and running, see regions on existing MongoDB databases into Atlas with our Live Migration service."

Procedemos a darle en **Build Database**, y allí nos mostrara lo siguiente:

The screenshot shows the MongoDB Atlas database configuration page. It lists three plan options: "M10 \$0.08/hour", "SERVERLESS \$0.10/1M reads", and "M0 FREE". The "M0 FREE" plan is highlighted with a blue border. Below each plan, there is a table showing storage, RAM, and vCPU specifications. A "Compare Features" button is located at the bottom of the page.

En donde seleccionaremos el plan gratuito para realizar la exploración de la base de datos.



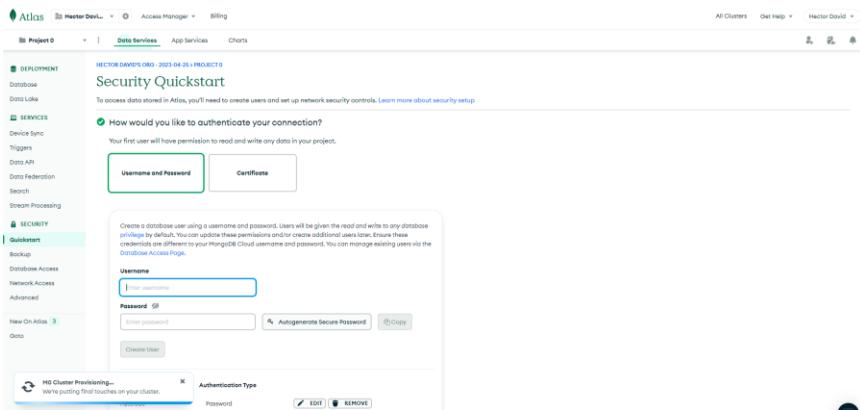
Dejamos tal cual la información que nos sale por default:

The screenshot shows the MongoDB Atlas cluster creation interface. It includes fields for Provider (aws), Region (N. Virginia (us-east-1)), Name (mern-web-personal), Tag (optional), and Auto-scale.

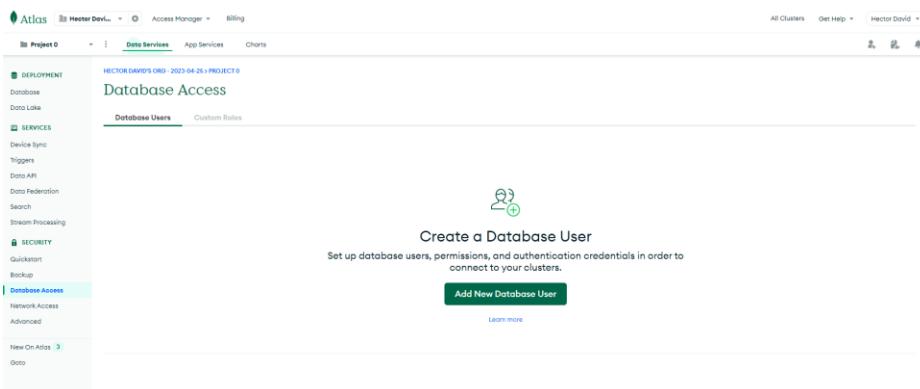
Acá colocamos el nombre de nuestro **cluster**, en mi caso lo dejare **mern-web-personal**



Y procedemos a darle en **create**, allí nos saldrá lo siguiente:



Ahora vamos a configurar el **database Access**



Acá vamos a crear un usuario y contraseña para poder acceder a nuestra **db**, en mi caso utilizare el usuario **mern** y colocare una contraseña, además de agregarle un rol de **admin**.



Add New Database User

Create a database user to grant an application or user access to databases and collections in your clusters in this Atlas project. Granular access control can be configured with default privileges or custom roles. You can grant access to an Atlas project or organization using the corresponding [Access Manager](#).

Authentication Method



MongoDB uses [SCRAM](#) as its default authentication method.

Password Authentication

username: mern
password: [SHOW](#)
[Autogenerate Secure Password](#) [Copy](#)

Database User Privileges

Configure role based access control by assigning database user a mix of one built-in role, multiple custom roles, and multiple specific privileges. A user will gain access to all actions within the roles assigned to them, not just the actions those roles share in common. You must choose at least one role or privilege. [Learn more about roles](#).

Built-in Role SELECTED

Select one [built-in role](#) for this user.

Y procedemos a crearlo, ahora vamos a agregar la **ip** para acceder a la **db**, nos dirigimos al menú **network Access**.

Le damos en **add IP Address**

Seleccionamos **Allow Access From Anywhere**, en este caso recordemos que realizaremos conexiones para aprender, pero en un entorno de producción no lo colocaremos de esta manera.



De esta manera tendremos configurada nuestra **db** para poder iniciar con la carga de la información.

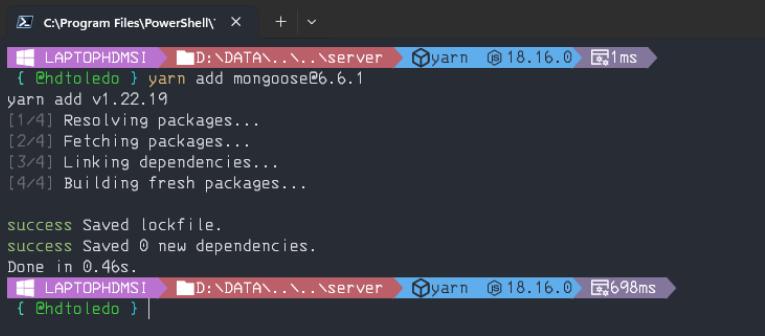
MONGOOSE – CONECTANDO LA APP CON LA DB

Para realizar nuestra conexión a la **DB** vamos a utilizar la dependencia de **mongoose**, nos dirigimos a la página de **yarn** y la buscamos

Ingresamos a **mongoose**, y en la página de **yarn** vamos a seleccionar la versión 6.6.1 allí nos dan las indicaciones de cómo debemos instalar



Para instalarlo vamos a nuestra terminal y vamos a colocar el comando de instalación **yarn add mongoose@6.6.1** y de esta manera ya tenemos instalado **mongoose** en nuestro server, verificamos en nuestro **package.json** que la dependencia este instalada:



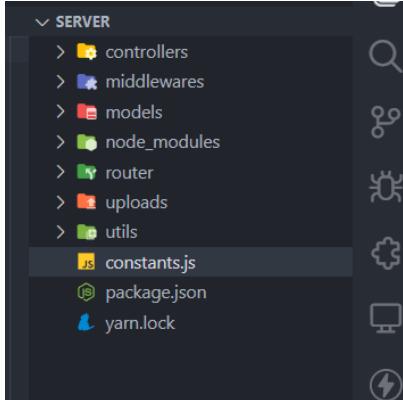
```
package.json > ...
1  {
2    "name": "web-personal-api",
3    "version": "0.0.1",
4    "description": "API web Personal",
5    "main": "index.js",
6    "author": "@hdtoledo",
7    "license": "MIT",
8    "dependencies": {
9      "mongoose": "6.6.1"
10   }
11 }

LAPTOPHDSI D:\DATA\..\..\server C:\Program Files\PowerShell\ > yarn add mongoose@6.6.1
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 0 new dependencies.
Done in 0.46s.

LAPTOPHDSI D:\DATA\..\..\server C:\Program Files\PowerShell\ > yarn 18.16.0
yarn 18.16.0
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
```

Ahora vamos a ir almacenando en un archivo de constantes las variables que vamos a utilizar en nuestro app, creamos nuestro archivo constants.js y colocamos lo siguiente:



Dentro de nuestro archivo asignaremos unas variables:

```
constants.js > ...
1 const DB_USER = "mern"
2 const DB_PASSWORD = "mern123456"
```

Y ahora nos dirigimos a nuestra página inicial de atlas en **mongodb** y le damos clic en **connect**



The screenshot shows the MongoDB Atlas interface for the 'mern-web-personal' database deployment. Key details include:

- Cluster Status:** 0 R/W connections (last 2 hours)
- Network Activity:** 0.0 B/s In, 0.0 B/s Out (last 2 hours)
- Data Size:** 0.0 B / 512.0 MB (8%)
- Sample Datasets:** A button to "Load sample datasets" is present.
- Monitoring:** Links to "View Monitoring" and "Browse Collections".
- Deployment Details:** Version 6.0.0, Region AWS / N. Virginia (us-east-1), Cluster Tier M0 Sandbox (General), Type Replica Set - 3 nodes, Backups Inactive, Linked App Services None Linked, Atlas SQL Connect, Atlas Search Create Index.

Damos clic en **drivers**:

The modal window titled "Connect to mern-web-personal" contains the following steps:

- Set up connection security
- Choose a connection method
- Connect

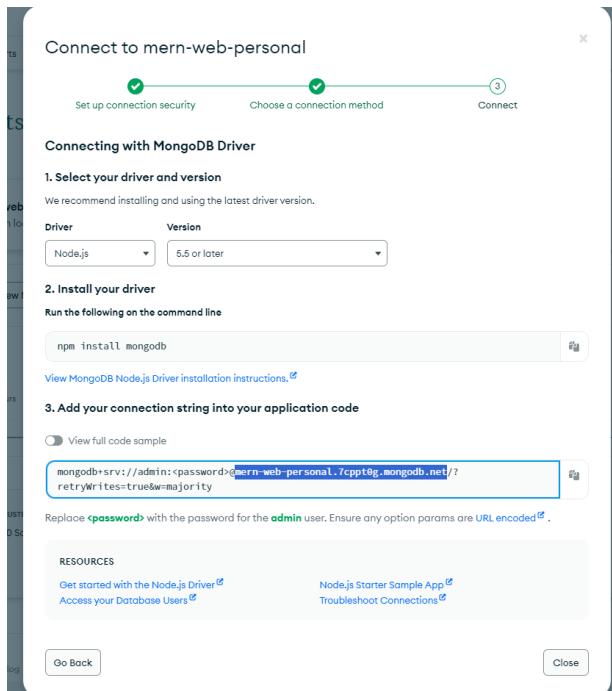
Under "Connect to your application", the "Drivers" section is highlighted, showing the description: "Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)".

Other sections include:

- Access your data through tools:**
 - Compass: Explore, modify, and visualize your data with MongoDB's GUI.
 - Shell: Quickly add & update data using MongoDB's Javascript command-line interface.
 - MongoDB for VS Code: Work with your data in MongoDB directly from your VS Code environment.
 - Atlas SQL: Easily connect SQL tools to Atlas for data analysis and visualization.

Y allí vamos a seleccionar la dirección que tenemos seleccionada:





Esta dirección la vamos a agregar a una **const** de la siguiente manera:

```
JS constants.js > ...
1 const DB_USER = "mern"
2 const DB_PASSWORD = "mern123456"
3 const DB_HOST = "mern-web-personal.7cppt0g.mongodb.net"
4
```

Procedemos a crear dos **const** más en las cuales vamos a vincular la versión de nuestra API y la dirección del servidor:

```
JS constants.js > IP_SERVER
1 const DB_USER = "admin"
2 const DB_PASSWORD = "admin123456"
3 const DB_HOST = "mern-web-personal.7cppt0g.mongodb.net"
4
5 const API_VERSION = "v1"
6 const IP_SERVER = "localhost"
```

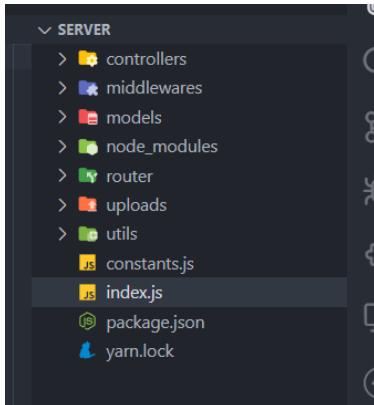
Ahora debemos exportar el módulo de las **const** de la siguiente manera:

```
JS constants.js > ...
1 const DB_USER = "admin"
2 const DB_PASSWORD = "admin123456"
3 const DB_HOST = "mern-web-personal.7cppt0g.mongodb.net"
4
5 const API_VERSION = "v1"
6 const IP_SERVER = "localhost"
7
8 module.exports = {
9   DB_USER,
10  DB_PASSWORD,
11  DB_HOST,
12  API_VERSION,
13  IP_SERVER
14 }
```



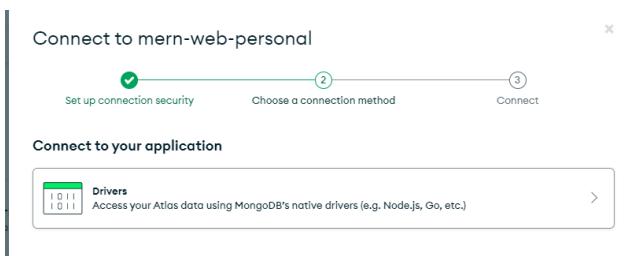
@hdtoledo

Ahora vamos a crear nuestro **index.js** y dentro de este vamos a traer **mongoose** a través del **cliente** y el **serverApi**, además, traemos las variables que vamos a usar de **constants.js**

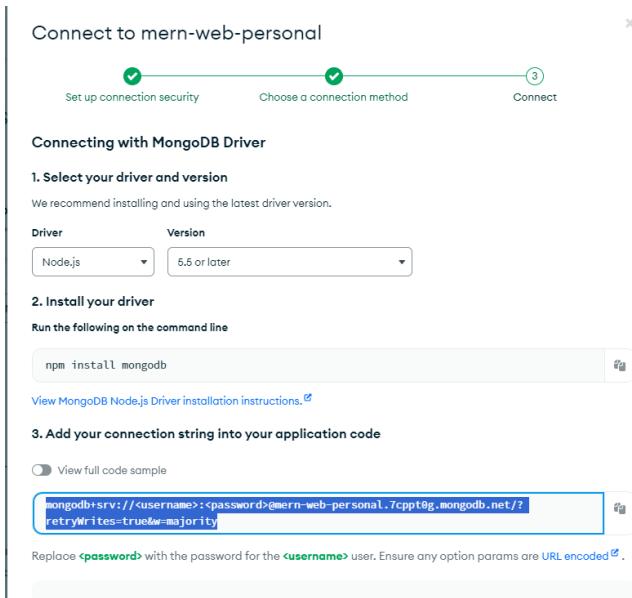


```
JS index.js > ...
1 const mongoose = require("mongoose")
2 const { DB_USER, DB_PASSWORD, DB_HOST, IP_SERVER, API_VERSION } = require("./constants.js")
3 |
```

Ahora procedemos a crear la conexión con **mongoose** y debemos copiar de atlas la dirección completa, al momento de darle **connect** en atlas seleccionamos **drivers**



Y allí nos vamos a la parte de abajo y seleccionamos:



Agregamos lo siguiente:

```
JS index.js > ...
1 const mongoose = require("mongoose")
2 const { DB_USER, DB_PASSWORD, DB_HOST, IP_SERVER, API_VERSION } = require("./constants.js")
3
4
5 mongoose.set('strictQuery', false)
6
7 mongoose.connect(
8   `mongodb+srv://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/`,
9   (error) => {
10     if (error) throw error
11     console.log("La conexión al servidor ha sido exitosa!");
12   }
13 )
14
15 |
```

- **mongoose.set('strictQuery', false)**

Esta línea configura el valor de la opción strictQuery de Mongoose en false. Esto permite que Mongoose acepte consultas que no cumplen con las restricciones de validación predeterminadas. Esto puede ser útil para aplicaciones que necesitan compatibilidad con consultas antiguas o que necesitan realizar consultas que no se pueden validar fácilmente, en nuestro caso lo aplicamos ya que vamos a trabajar con una versión anterior a la actual.

- Dentro de esta conexión utilizamos los backticks, **mongoose.connect(`mongodb+srv://\${DB_USER}:\${DB_PASSWORD}@\${DB_HOST}/`, (error) => { if (error) throw error // ... })`**. Esta línea conecta a la base de datos MongoDB utilizando las credenciales especificadas en las variables de entorno `DB_USER`, `DB_PASSWORD` y `DB_HOST`. Si la conexión falla, se lanza un error. *

Ahora llego la hora de probar si funciona para ello vamos a modificar nuestro **package.json** y vamos a agregar el script **start**:

```
JSON package.json > ...
1 {
2   "name": "web-personal-api",
3   "version": "0.0.1",
4   "description": "API web Personal",
5   "main": "index.js",
6   "author": "@hdtoledo",
7   "license": "MIT",
8   "scripts": {
9     "start": "node index.js"
10   },
11   "dependencies": {
12     "mongoose": "6.6.1"
13   }
14 }
15 |
```

Y ahora procedemos a ejecutar la consola con el comando **yarn start**, si todo va bien nos mostrara el mensaje de que se conectó correctamente:



```
LAPTOPHDMI D:\DATA\...\server yarn 18.16.0 1ms
{ @hdtledo } yarn start
yarn run v1.22.19
$ node index.js
La conexión al servidor ha sido exitosa!
```

CREANDO EL SERVIDOR HTTP CON EXPRESS

Express.js es un marco de trabajo (framework) para construir aplicaciones web en Node.js. Piensa en **Express.js** como un conjunto de herramientas predefinidas que hacen que sea más fácil crear sitios web y aplicaciones web en el lenguaje de programación JavaScript.

Express.js simplifica tareas comunes en el desarrollo web, como manejar rutas, gestionar solicitudes y respuestas, trabajar con bases de datos y mucho más. Te permite construir aplicaciones web de manera más rápida y eficiente al proporcionarte una estructura y un conjunto de funciones que facilitan la creación de servidores web y la gestión de rutas y recursos.

Ahora procedemos a agregarlo a nuestro proyecto con el siguiente comando

```
LAPTOPHDMI D:\DATA\...\server yarn 18.16.0 178ms
{ @hdtledo } yarn add express@4.18.1
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└ express@4.18.1
info All dependencies
└ express@4.18.1
Done in 1.46s.
```

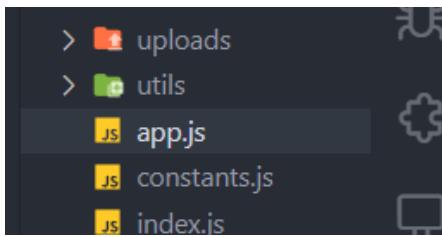
Una vez instalado verificamos en nuestro **package.json** que la dependencia haya sido instalada:

```
8  "scripts": {
9    "start": "node index.js"
10   },
11  "dependencies": {
12    "express": "4.18.1",
13    "mongoose": "6.6.1"
14  }
15 }
16 }
```



@hdtledo

Ahora procedemos a crear nuestro **app.js** en donde colocaremos todo lo relacionado a **express**



Dentro vamos a colocar lo siguiente y vamos a dejar unas tareas pendientes las cuales vamos a ir realizando paso a paso.

```
js app.js > ...
1 const express = require('express')
2 const { API_VERSION } = require("./constants")
3
4 const app = express()
5
6 // Importar rutas
7
8
9 //Configurar Body Parse
10
11
12 // Configurar Header HTTP - CORS
13
14 // Configurar Rutas
15
16
17 module.exports = app
```

Por el momento dejaremos allí la configuración y vamos a pasar a nuestro **index.js**, para realizar el levantamiento del servidor, hacemos la importación de **express** mediante **app**

```
js index.js > app
1 const mongoose = require("mongoose")
2 const app = require("./app.js")
3 const { DB_USER, DB_PASSWORD, DB_HOST, IP_SERVER, API_VERSION } = require ("./constants.js")
4
```

Ahora vamos a levantar el servicio configurando el puerto a través de una const:

```
js index.js > ...
1 const mongoose = require("mongoose")
2 const app = require("./app.js")
3 const { DB_USER, DB_PASSWORD, DB_HOST, IP_SERVER, API_VERSION } = require ("./constants.js")
4
5
6 const PORT = process.env.PORT || 3000
7
8 mongoose.set('strictQuery', false)
9
```



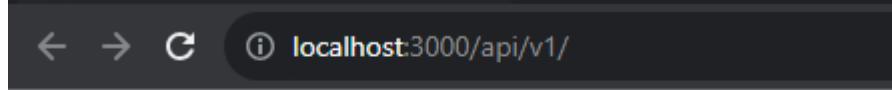
Y por último vamos a configurar la escucha de nuestro servidor en el puerto y que nos muestre un mensaje en consola:

```
42
13 mongoose.set("strictQuery", false);
14
15 mongoose.connect(
16   `mongodb+srv://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/`,
17   (error) => {
18     if (error) throw error
19
20     app.listen(PORT, () => {
21       console.log(`#####
22       #### MERN API REST #####
23       #####`)
24       console.log(`http://:${IP_SERVER}:${PORT}/api/${API_VERSION}`)
25     })
26   }
27 )
28
```

Si todo va bien ejecutamos **yarn start** y nos devolverá lo siguiente:

```
LAPTOPHDMI D:\DATA\..\server yarn 18.16.0 2ms
{ hdtoledo } yarn start
yarn run v1.22.19
$ node index.js
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
```

Ahora que tenemos levantado el servicio nos dirigimos al navegador a la dirección y el puerto y nos debe salir lo siguiente:



De esta manera ya tenemos nuestro servidor escuchando.



RECARGANDO EL SERVIDOR AUTOMATICAMENTE

Vamos a realizar un cambio sobre nuestros scripts de ejecución, y para ello vamos a añadir el script de desarrollo **DEV**, lo primero es hacer la instalación de la dependencia **nodemon** que nos va a permitir cargar un servicio sobre los cambios encontrados en nuestros recursos y que permitirá reiniciar el servidor para que los cambios se vean reflejados en nuestro navegador, ejecutamos en consola el comando **yarn add nodemon@2.0.20**

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
( @hdtledo ) yarn add nodemon@2.0.20
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└ nodemon@2.0.20
info All dependencies
└ nodemon@2.0.20
Done in 1.06s.
```

Verificamos en nuestro **package.json** que este instalada:

```
10  },
11  "dependencies": {
12    "express": "4.18.1",
13    "mongoose": "6.6.1",
14    "nodemon": "2.0.20"
15  }
16 }
17 }
```

Ahora en nuestros scripts agregamos **dev**:

```
7   "license": "MIT",
8     > Depurar
9   "scripts": {
10     "start": "node index.js",
11     "dev": "nodemon index.js"
12   },
13   "dependencies": {
```

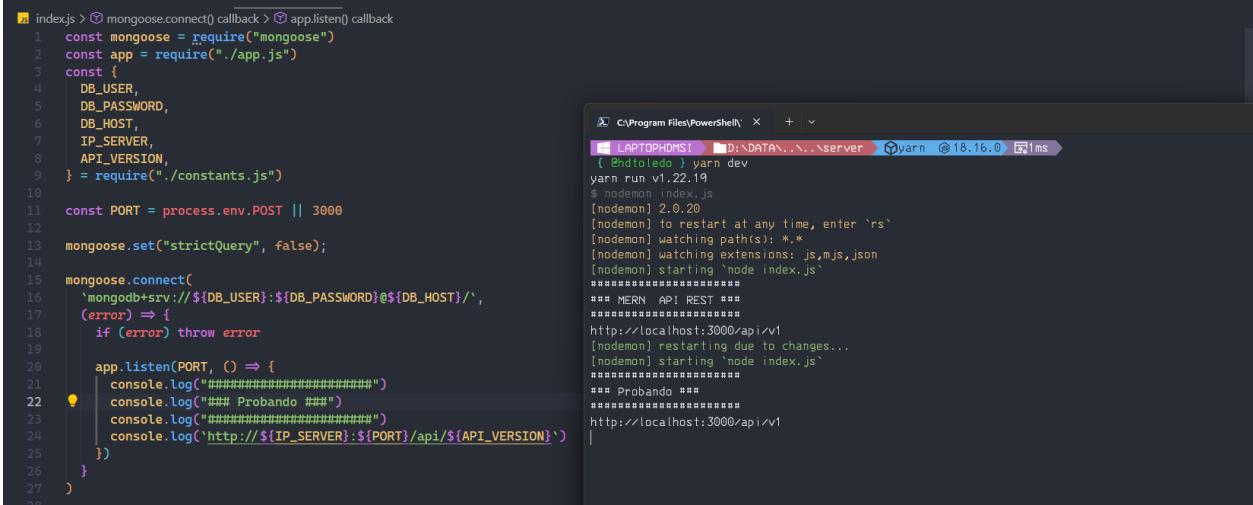
Y ya solo nos queda ejecutar **yarn dev** en nuestra terminal para que se vea el resultado:

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
( @hdtledo ) yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
```



@hdtledo

Vamos a realizar un cambio sobre nuestro **index.js** y una vez guardados los cambios se van a ver reflejados en la terminal y en el navegador:



The image shows a code editor on the left containing the `index.js` file. On the right is a terminal window titled "C:\Program Files\PowerShell\" showing the command `yarn run v1.22.19` and the output of nodemon running. A browser window is also visible, displaying the API endpoint `http://localhost:3000/api/v1`.

```
index.js
1  index.js > mongoose.connect() callback > app.listen() callback
2  const mongoose = require("mongoose")
3  const app = require("./app.js")
4  const {
5    DB_USER,
6    DB_PASSWORD,
7    DB_HOST,
8    IP_SERVER,
9    API_VERSION,
10   } = require("./constants.js")
11
12  const PORT = process.env.PORT || 3000
13
14  mongoose.set("strictQuery", false)
15
16  mongoose.connect(
17    `mongodb+srv://${DB_USER}:${DB_PASSWORD}@${DB_HOST}/`,
18    (error) => {
19      if (error) throw error
20
21      app.listen(PORT, () => {
22        console.log("#####")
23        console.log("## Probando ##")
24        console.log("#####")
25        console.log(`http://${IP_SERVER}:${PORT}/api/${API_VERSION}`)
26      })
27    }
28  )
```

Cada vez que hacemos cambios en nuestro código y los guardamos se verán reflejados y **nodemon** reiniciara el servicio para que nosotros lo podamos ver.

CONFIGURANDO BODY PARSER

El middleware **body-parser** se utiliza para analizar y extraer los datos del cuerpo (**body**) de una solicitud HTTP, en el contexto de una aplicación web, cuando un cliente envía datos al servidor a través de un formulario **HTML** o una solicitud **API**, esos datos se envían en el cuerpo de la solicitud **HTTP**. **body-parser** se encarga de tomar esos datos en formato de texto y convertirlos en un objeto JavaScript utilizable para que puedas trabajar con ellos en tu aplicación.

Por ejemplo, si estás construyendo un formulario de registro en una aplicación web, cuando un usuario complete el formulario y lo envíe, los datos ingresados (como el nombre, el correo electrónico y la contraseña) se enviarán en el cuerpo de la solicitud HTTP. **body-parser** ayudará a tu aplicación Express a tomar esos datos del cuerpo de la solicitud y convertirlos en un objeto JavaScript que puedes usar para registrar al usuario en tu base de datos, por ejemplo.

Así que, en resumen, **body-parser** es una herramienta esencial para procesar y gestionar datos enviados desde el cliente a través de solicitudes HTTP en una aplicación **Express.js**, procedemos a realizar la instalación con el comando **yarn add body-parser@1.20.0**



The image shows a terminal window titled "LAPTOPHDMI" with the command `yarn add body-parser@1.20.0` being run. The output shows the progress of the installation, including resolving packages, fetching packages, linking dependencies, and building fresh packages. It ends with success messages and a total time of 0.47s.

```
LAPTOPHDMI > D:\DATA\..\server > yarn 18.16.0 1ms
{ @hdtolledo } yarn add body-parser@1.20.0
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 0 new dependencies.
Done in 0.47s.
```



Verificamos en nuestro **package.json** que este instalado:

```
12  "dependencies": {  
13    "body-parser": "1.20.0",  
14    "express": "4.18.1",  
15    "mongoose": "6.6.1",  
16    "nodemon": "2.0.20"  
17  }  
18 }
```

Ahora vamos a configurar nuestro **app.js** con el **body parser**:

```
js app.js > ...  
1  const express = require("express")  
2  const bodyParser = require("body-parser")  
3  const { API_VERSION } = require("./constants")  
4  
5  const app = express()  
6  
7  // Importar rutas  
8  
10 //Configurar Body Parse  
11 app.use(bodyParser.urlencoded({ extended: true}))  
12 app.use(bodyParser.json())  
13  
14 // Configurar Header HTTP - CORS  
15  
16 // Configurar Rutas  
17  
18  
19 module.exports = app
```

Llamamos nuestro **body parser** a través de unas **const**, y configuramos para que todo lo que nos llegue a través de la **url** de nuestra **app** se convierta en **json**.



@hdtoledo

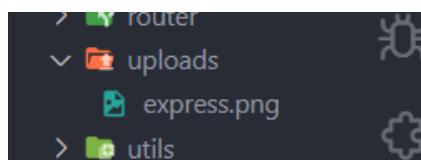
CONFIGURANDO CARPETA ESTATICOS

Todos los archivos **estáticos** los vamos a ubicar en la carpeta **uploads** en la cual tendremos imágenes, archivos, y demás cosas que necesitemos dentro de nuestra **app**, así que vamos a configurar la ubicación de esta carpeta dentro de nuestro archivo **app.js**:

```
js app.js > ...
1 const express = require("express")
2 const bodyParser = require("body-parser")
3 const { API_VERSION } = require("./constants")
4
5
6 const app = express()
7
8 // Importar rutas
9
10
11 //Configurar Body Parse
12 app.use(bodyParser.urlencoded({ extended: true}))
13 app.use(bodyParser.json())
14
15 //Configurar carpeta static
16 app.use(express.static("uploads"))
17
18 // Configurar Header HTTP - CORS
19
20 // Configurar Rutas
21
22
23 module.exports = app
```

Para comprobar que está funcionando es muy sencillo, guardaremos una imagen de prueba dentro de la carpeta **uploads** y simplemente levantamos nuestro servidor, una vez levantado nos dirigimos a nuestro navegador y en la raíz de nuestro **localhost**, añadimos **/imagen.jpg**, si todo va bien debemos obtener una vista de nuestra imagen.

Subimos la imagen:



Levantamos el servidor:

```
LAPTOPHDM! D:\DATA\...\server yarn 18.16.0 1ms
{ hdtoledo } yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
```



Revisamos en el navegador con la url de nuestra imagen:



De esta manera verificamos que este correctamente configurada nuestra carpeta de estáticos.

CONFIGURANDO LAS CORS

Las **CORS** (Cross-Origin Resource Sharing) son una medida de seguridad implementada en los navegadores web para prevenir ataques de seguridad relacionados con solicitudes entre diferentes dominios (sitios web). Básicamente, las **CORS** son un conjunto de reglas que dictan si un sitio web (dominio) tiene permiso para acceder a los recursos de otro sitio web (dominio).

Cuando estás construyendo una aplicación web con Express.js y deseas que tu servidor permita que otros dominios hagan solicitudes **HTTP** a tus rutas y recursos (como **APIs**), es importante configurar las **CORS** adecuadamente. Sin la configuración adecuada de las **CORS**, los navegadores web bloquearán las solicitudes desde otros dominios por razones de seguridad.

En **Express.js**, puedes habilitar las **CORS** utilizando middleware, como **cors**, que es una biblioteca de middleware específicamente diseñada para gestionar las políticas de **CORS**. Cuando configuras **cors** en tu aplicación **Express**, puedes especificar desde qué dominios permites solicitudes y qué tipos de solicitudes (como **GET**, **POST**, etc.) están permitidas.

Por ejemplo, puedes configurar **cors** para permitir que cualquier sitio web acceda a tus recursos o especificar dominios específicos que tienen permiso. También puedes definir las cabeceras que se deben incluir en las respuestas para indicar que las solicitudes **CORS** son permitidas.

Ahora vamos a configurar las **CORS** dentro de nuestro archivo **app.js**, para ello colocamos el comando **yarn add cors@2.8.5**

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
{ @hdtolledo } yarn add cors@2.8.5
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
success Saved 1 new dependency.
info Direct dependencies
└ cors@2.8.5
info All dependencies
└ cors@2.8.5
Done in 1.01s.
```

y ahora configuraremos de la siguiente manera nuestro **app.js** agregando **const cors**:

```
app.js > cors
1 const express = require("express")
2 const bodyParser = require("body-parser")
3 const cors = require("cors")
4 const { API_VERSION } = require("./constants")
5
```

Y haciendo la implementación de este con **app.use**

```
18 // Configurar Header HTTP - CORS
19 app.use(cors())
20 // Configurar Rutas
21
22 // Configurar Rutas
```

De esta manera dejamos las **cors** configuradas y si iniciamos nuestro servidor no nos debe generar ningún error.

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
{ @hdtolledo } yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
```



CREANDO NUESTRO PRIMER ENDPOINT

Un "**endpoint**" se refiere a un punto de acceso o una URL específica a la que se puede hacer una solicitud HTTP para interactuar con una aplicación o un servicio. Los **endpoints** son como puertas de entrada a una aplicación o servicio web que te permiten realizar diversas operaciones.

Cada **endpoint** generalmente está asociado con una función o acción específica en el servidor, y suelen corresponder a operaciones CRUD (Crear, Leer, Actualizar y Borrar) en una base de datos o a la ejecución de una función específica en el servidor.

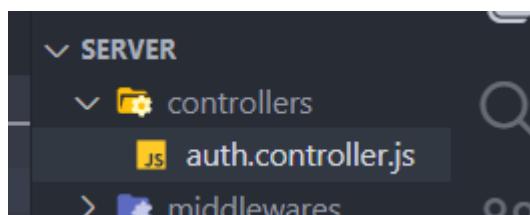
Por ejemplo, en una API de redes sociales, podrías tener los siguientes **endpoints**:

1. **/posts**: Este **endpoint** podría usarse para obtener una lista de publicaciones.
2. **/posts/{id}**: Usado para obtener una publicación específica por su ID.
3. **/users/{username}/posts**: Podría proporcionar las publicaciones de un usuario en particular.
4. **/posts/create**: Para crear una nueva publicación.
5. **/posts/{id}/update**: Para actualizar una publicación existente.
6. **/posts/{id}/delete**: Para eliminar una publicación.

Cada uno de estos **endpoints** puede aceptar diferentes tipos de solicitudes HTTP, como GET para obtener datos, POST para crear datos, PUT para actualizar datos y DELETE para eliminar datos, entre otros métodos HTTP.

Ahora empezaremos a realizar el montaje de los **endpoints** para nuestros usuarios y vamos a validar el sistema de autenticación, registro de usuario y login, tambien vamos a configurar el refresh token del usuario.

Para iniciar vamos a crear nuestro controlador de **auth.controller.js**, para ello nos ubicamos en la carpeta **controllers** y creamos el archivo

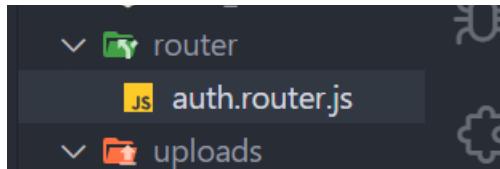


Dentro de nuestro **auth** colocamos lo siguiente:

```
1 function register(req, res){  
2     console.log("Se ha ejecutado el registro")  
3  
4     res.status(200).send({msg:"Funciono Perfecto !"})  
5 }  
6  
7 module.exports = {  
8     register,  
9 }
```

Acá podemos observar como a través de una función pasamos un request y response y mediante consola por el momento le vamos a indicar que se ha ejecutado el registro, también agregamos un mensaje de estado 200 para indicar que todo va bien y por último exportamos el módulo para poderlo utilizar.

Ahora la pregunta es cómo hacemos para que funcione nuestro controlador, y para ello vamos a crear nuestra ruta, en la carpeta **router** vamos a crear nuestra ruta creamos el archivo **auth.router.js**



Ahora vamos a colocar lo siguiente para empezar a utilizar nuestra autenticación en **auth.router.js**:

```
router > js auth.router.js > AuthController
1 const express = require("express")
2 const AuthController = require("../controllers/auth.controller")
3
4 const api = express.Router()
5
6 api.post("/auth/register/", AuthController.register)
7
8 module.exports = api
```

De esta manera le indicamos que vamos a utilizar express y que vamos a utilizar nuestro controlador a través de la ruta **/auth/register/** por último exportamos el módulo.

Ahora para que nos funcione debemos pasarlo a través de **app.js** y hacer la importación para que express lo pueda utilizar:

```
8
9 // Importar rutas
10 const authRoutes = require("./router/auth.router")
11
12 //Configurar Body Parse
```

Y hacemos la configuración en el mismo **app.js** en donde dejamos el comentario de configurar rutas:

```
21
22 // Configurar Rutas
23 app.use(`/api/${API_VERSION}`, authRoutes)
24
```

De esta manera tenemos ya configurada nuestras rutas para poder ingresar, ahora debemos probar esta ruta y para ello vamos a utilizar **insomnia**, es una aplicación o software de código abierto diseñado para ayudar a los desarrolladores a probar y depurar **APIs** (Interfaces de Programación de Aplicaciones) de manera eficiente. Es una herramienta de cliente **REST** y **GraphQL** que permite a los desarrolladores interactuar con **API endpoints**, enviar solicitudes **HTTP**, ver respuestas y analizar los resultados.

Las principales características de Insomnia incluyen:

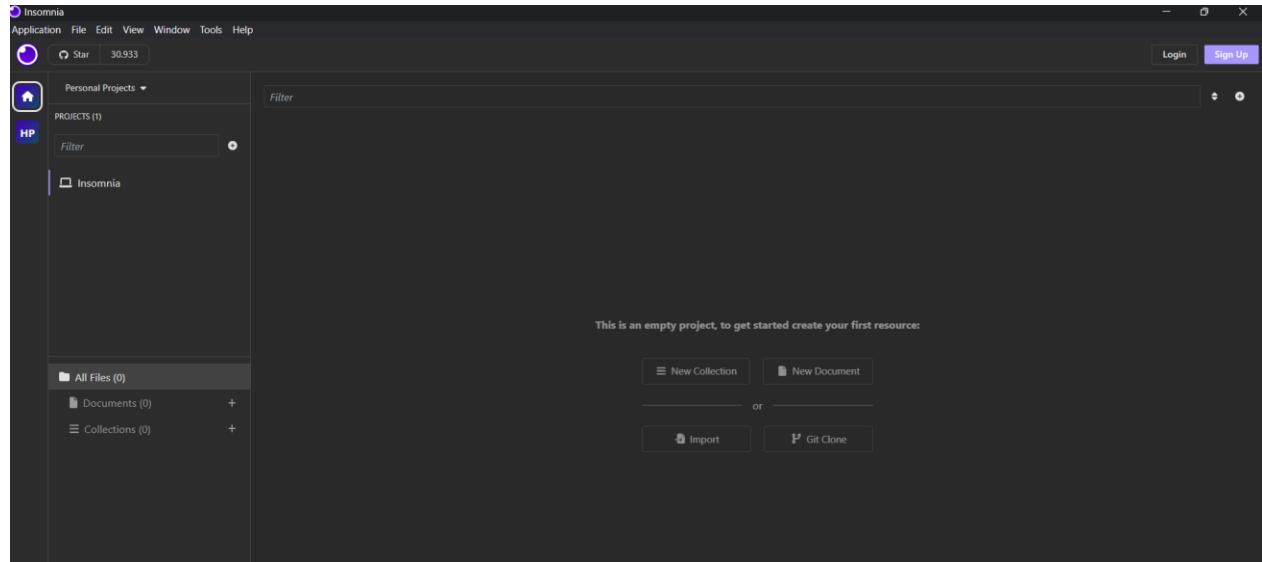
1. Interfaz Gráfica Amigable: Insomnia proporciona una interfaz de usuario intuitiva que facilita la creación, modificación y organización de solicitudes HTTP. Esto es especialmente útil para



aquellos que prefieren una interfaz visual en lugar de trabajar directamente con herramientas de línea de comandos.

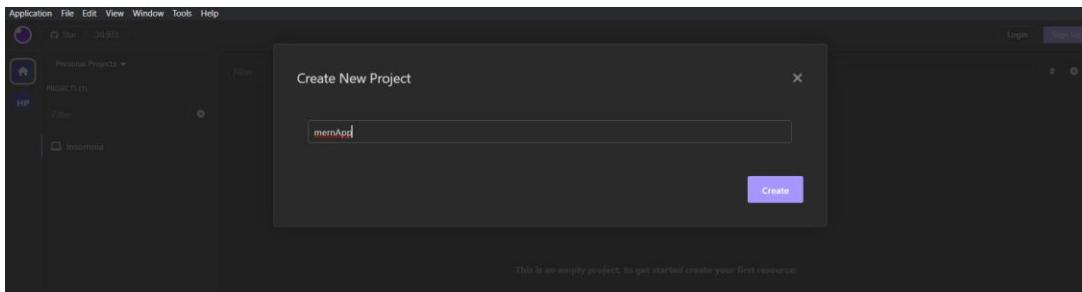
2. Sesiones de Trabajo: Permite organizar solicitudes y respuestas en sesiones de trabajo para proyectos específicos o para diferentes partes de una API. Esto ayuda a mantener todo ordenado y fácil de gestionar.
3. Soporte para Múltiples Protocolos: Además de HTTP, Insomnia admite otros protocolos como GraphQL, WebSockets, gRPC y más. Esto es útil cuando se trabaja con diferentes tipos de API.
4. Variables y Entornos: Insomnia permite definir variables y entornos, lo que facilita la personalización de las solicitudes para diferentes escenarios o ambientes.
5. Autenticación: Proporciona opciones para configurar fácilmente la autenticación en las solicitudes, como agregar encabezados de autorización o tokens.
6. Generación de Código: Puede generar código en varios lenguajes de programación a partir de solicitudes, lo que facilita la integración de las API en aplicaciones.
7. Documentación Interactiva: **Insomnia** permite crear documentación interactiva de API para compartir con otros miembros del equipo o con terceros.

Ahora procedemos a abrir **insomnia** y vamos a realizar una pequeña configuración, la apariencia que veremos será la siguiente:

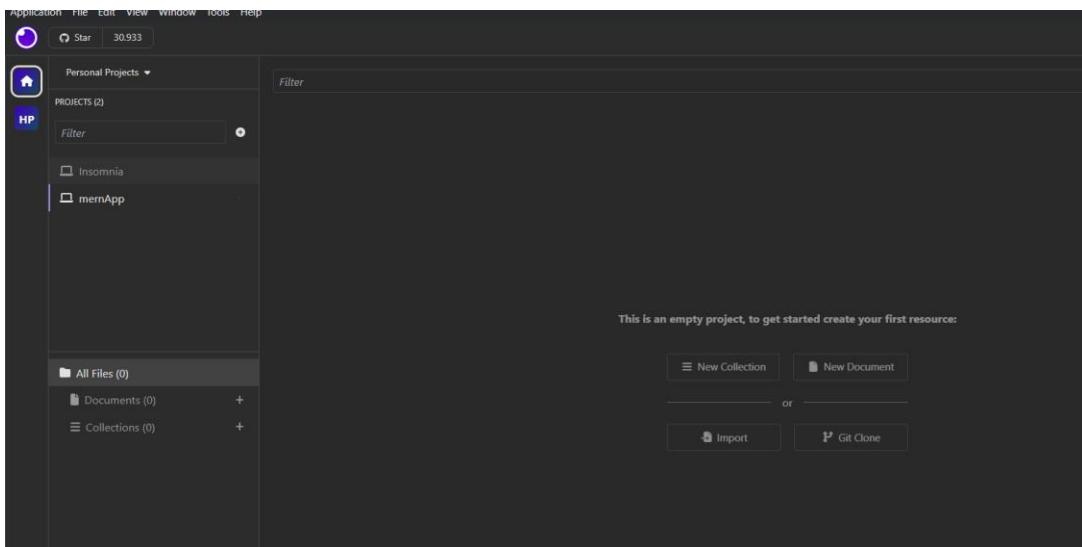


@hdtoledo

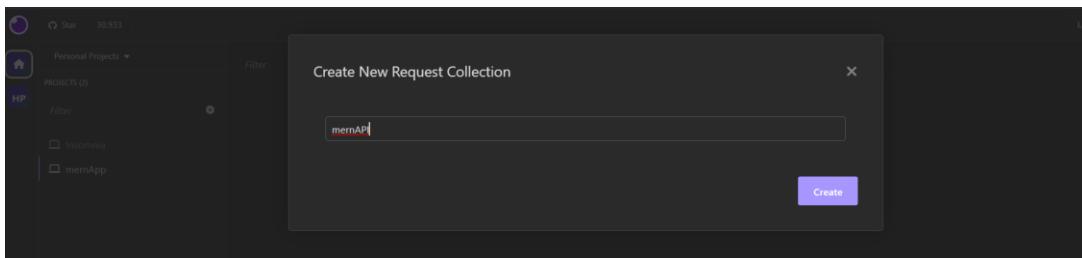
Ahora pasaremos a darle a crear un nuevo proyecto:



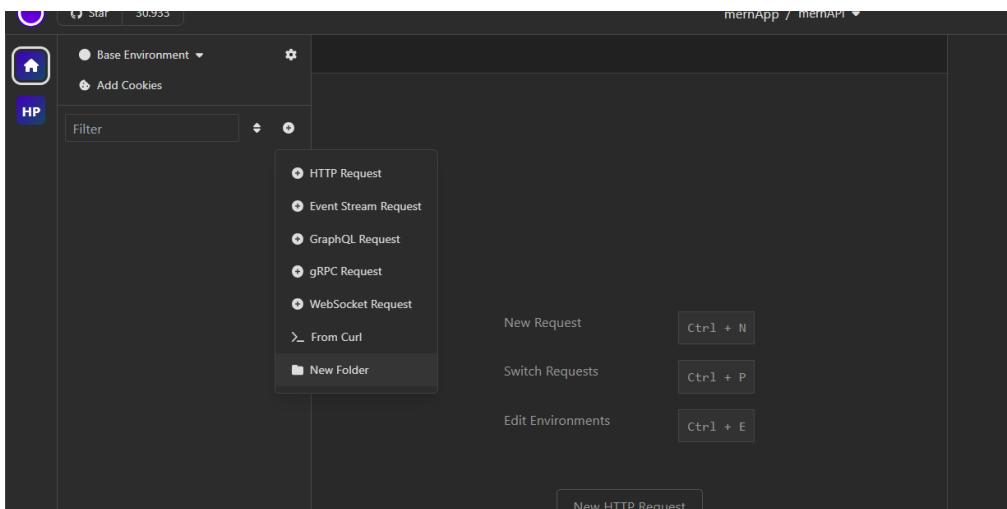
En nuestra interfaz nos mostrara el proyecto nuevo y vamos a crear una nueva colección:



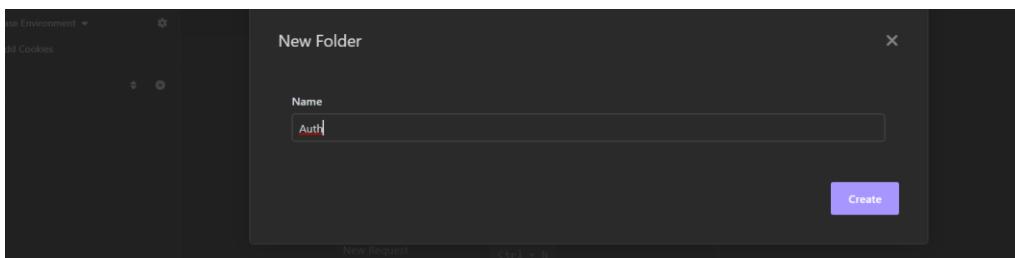
Creamos la nueva colección



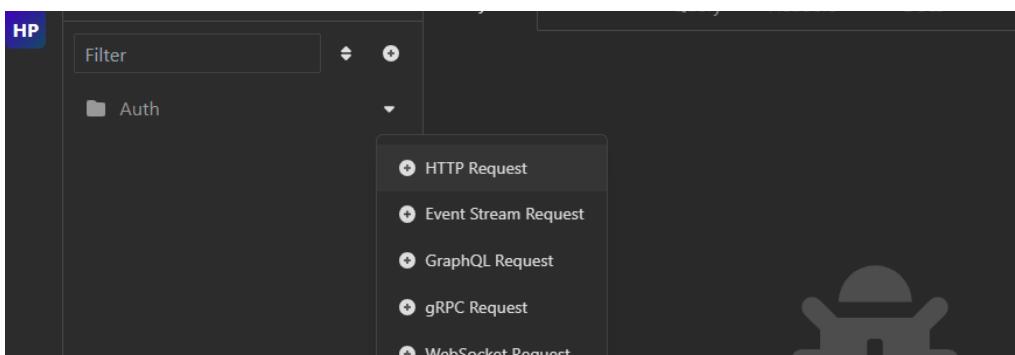
Y dentro de la interfaz nos mostrara para agregar unas variables de entorno, opción para cookies y luego tenemos una opción para agregar una nueva carpeta, presionamos para crear una nueva carpeta:



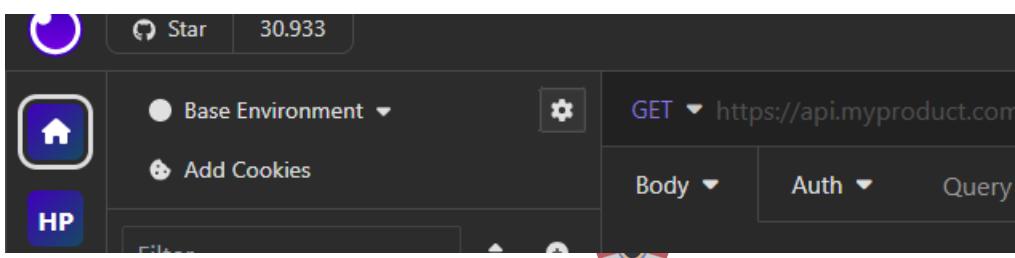
A la cual le colocamos Auth:



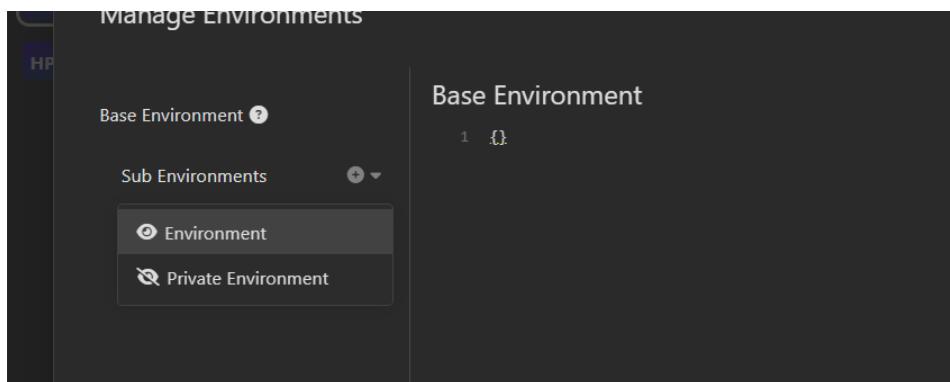
Y ahora le daremos de nuevo para crear una solicitud HTTP:



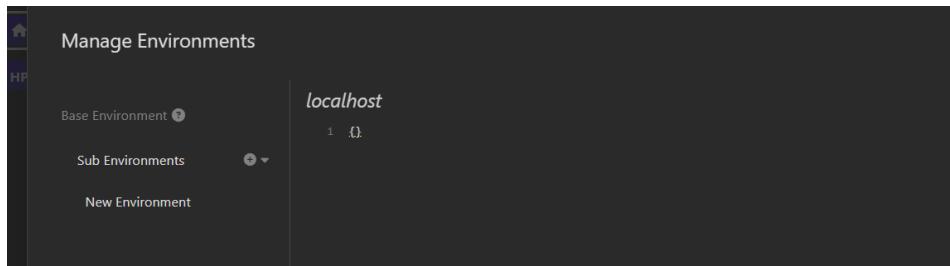
En la parte de arriba podemos observar donde dice **base environment** y le vamos a dar clic sobre la rueda dentada:



Y acá nos muestra para poder agregar unas variables de entorno, en este caso vamos a crear una de nuestra ruta principal a la cual estaremos haciendo las peticiones



Agregamos el nombre principal como lo es localhost:



Y dentro de las llaves vamos a colocar el **base path**



Recordemos que nuestro base path será la dirección principal de nuestro servidor:

```
HTTP/1.1 200 OK
#####
http://localhost:3000/api/v1
```

La seleccionamos, copiamos y pegamos en insomnia, y acá tenemos una opción para poder colocar colores a nuestros **paths** y distinguirlos en mi caso seleccionare un tono azul:



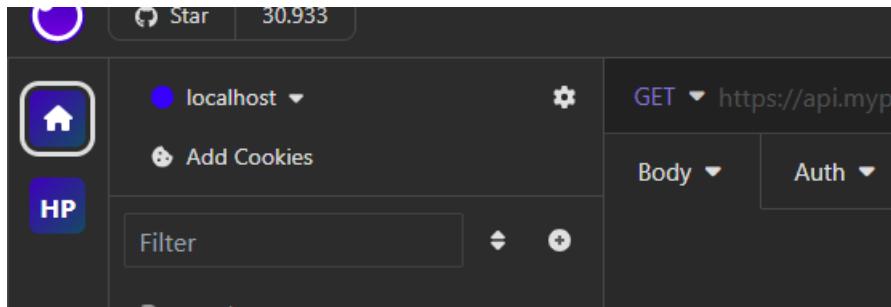
@hdtoledo

Y de esta manera nos debe quedar nuestro **base path**

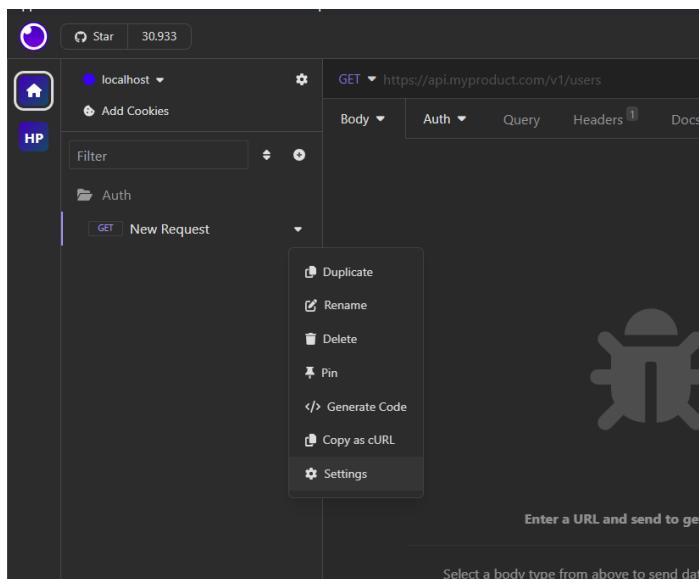


```
1 var {  
2   "BASE_PATH": "http://localhost:3000/api/v1"  
3 };
```

Cerramos y nos debe quedar así en nuestra vista:

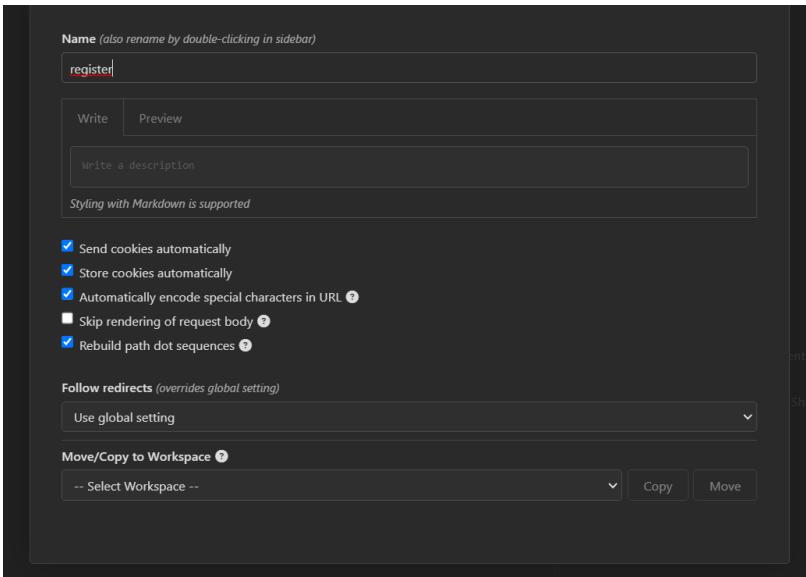


Ahora vamos a colocarle nombre a nuestra petición para ello le damos clic sobre las opciones de new request y seleccionamos settings

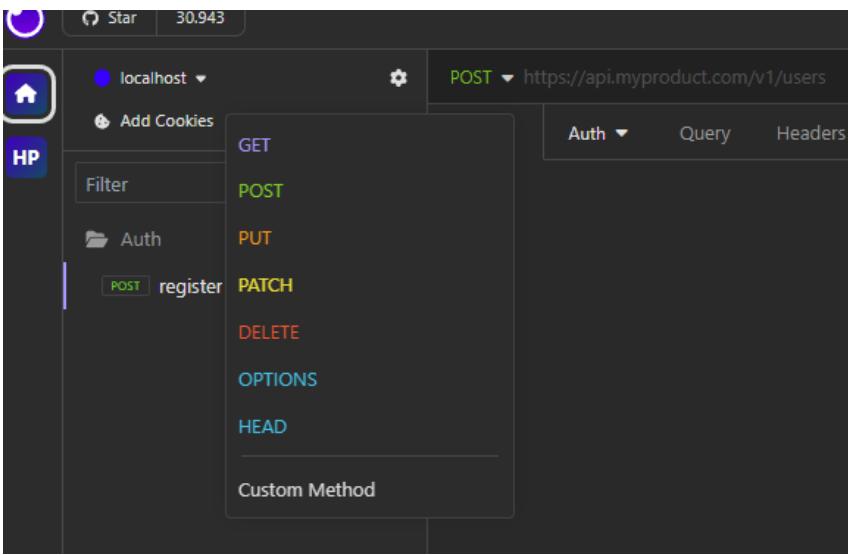


@hdtoledo

Y le vamos a colocar register



Al volver a nuestra vista anterior observamos como podemos hacer las diferentes peticiones HTTP:



Vamos a seleccionar el método **GET** para nuestra solicitud y al darle **ctrl + espacio** nos saldrá nuestro **base path** que configuramos:



Recordemos que nuestro base path es la ruta a la cual vamos a direccionar las solicitudes, en este caso la ruta es **http://localhost:3000/api/v1** de esta manera no tendremos que memorizar o estar escribiendo toda esta ruta, sino que simplemente la tendremos almacenada para poder agilizar el proceso.

Una vez seleccionada nuestra **base path** agregamos el resto de la ruta, que es **/auth/register** al hacer la petición de GET nos debe salir nuestro mensaje previamente configurado:

Y si revisamos nuestra terminal debe salir que se ha ejecutado correctamente el registro:

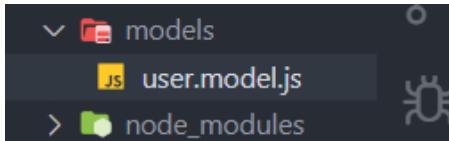
```
#####
#####
#####
#### API REST MERN ####
#####
http://localhost:3000/api/v1/
Se ha ejecutado el registro
```

De esta manera hemos configurado nuestro primer **endpoint**.



MODELO DEL USUARIO

El modelo del usuario es lo que tiene el usuario para poder iniciar su sesión, es decir nombres, apellidos, correo, contraseña, rol, activo, avatar, todas ellas, para ello nos vamos a **models** y vamos a crear el modelo **user.model.js**



Y dentro de nuestro **user.model.js**:

```
models > user.js > <unknown>
1  const mongoose = require("mongoose")
2
3  const UserSchema = mongoose.Schema({
4      firstname: {
5          type: String,
6          required: true,
7          trim: true
8      },
9      lastname: {
10         type: String,
11         required: true,
12         trim: true
13     },
14     email: {
15         type: String,
16         unique: true,
17         required: true
18     },
19     password: {
20         type: String,
21         required: true
22     },
23     role: String,
24     active: Boolean,
25     avatar: String,
26 })
27
28 module.exports = mongoose.model("User", UserSchema)
```

Primero hacemos la importación de **mongoose**, y asignamos en una **const** a través del esquema nuestro modelo de datos que vamos a pasarle, aquí tenemos nombres, apellidos, correo, contraseña, rol, activo, avatar, con tipos de datos **string**, **booleano**, además utilizamos para que sea requerido y que nos limpie los espacios en blanco que el usuario llegue a colocar utilizando **trim**.

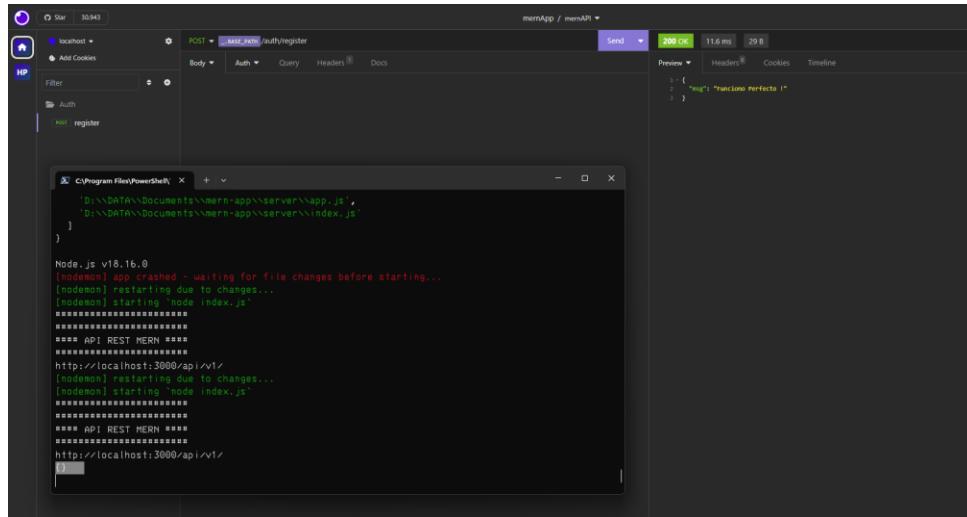


REGISTRANDO EL USUARIO

Para poder realizar el registro del usuario nos vamos a ubicar en nuestro controlador **auth.controller.js** y realizamos lo siguiente:

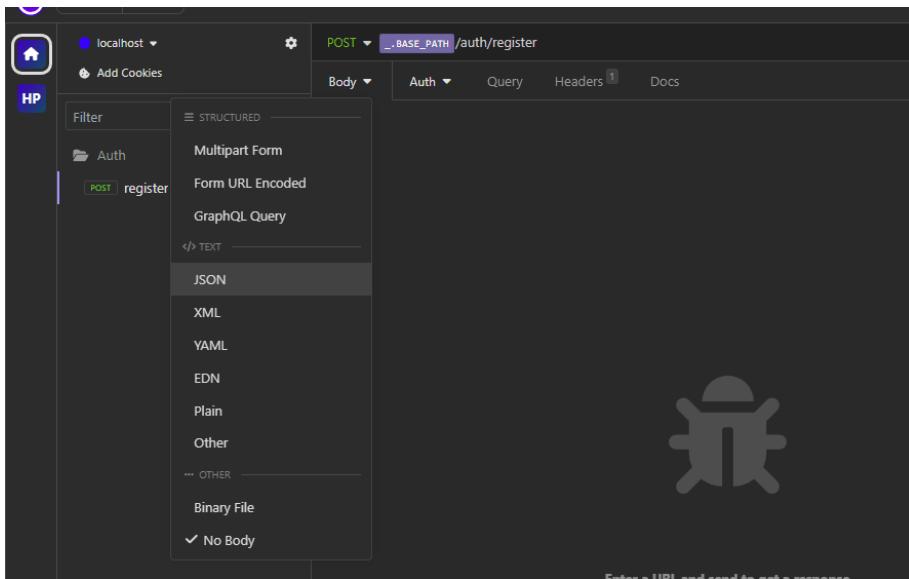
```
1 const User = require("../models/user")
2
3 function register(req, res){
4     console.log(req.body)
5
6     res.status(200).send({msg:"Funciono Perfecto !"})
7 }
8
9 module.exports = {
10     register,
11 }
```

A través de una **const** llamada **User** importamos nuestro **modelo**, y modificamos nuestro **console.log** para que nos capture lo que vamos a enviar en nuestro **body**, vamos a realizar el ejercicio para verificar que estamos enviando, nos vamos a insomnia y hacemos de nuevo la petición y vamos a obtener lo siguiente:

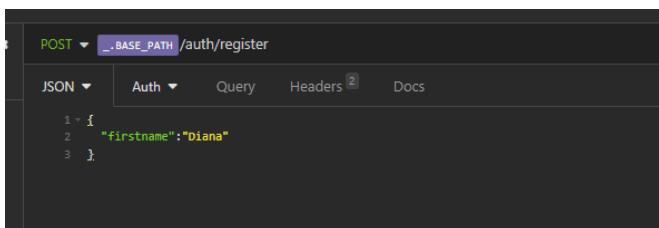


El mensaje se nos muestra de nuevo que todo va bien y en nuestra consola revisamos que hay un objeto vacío, esto quiere decir que todo va bien hasta el momento, lo siguiente es pasarle los datos a través de **JSON** y para ello en nuestro insomnia seleccionamos debajo del **base path** el formato de **JSON**:





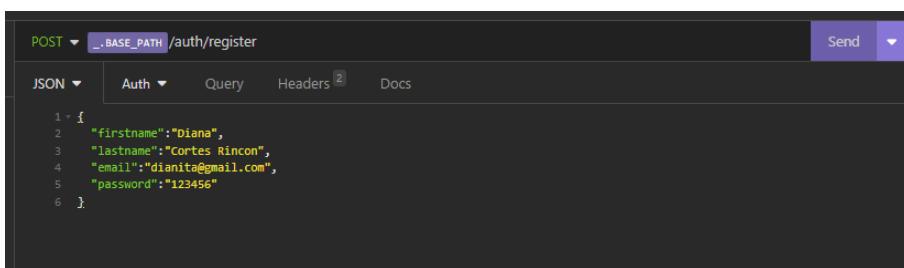
Una vez seleccionado JSON vamos a pasar los siguientes datos para probar:



Si volvemos a ejecutar la solicitud nos saldrá en nuestra consola los datos:

```
http://localhost:3000/api/v1/  
{ firstname: 'Diana' }  
|
```

Y ahora vamos a terminar de estructurar los datos que debemos pasar para verificar:



Estos datos serían los básicos al momento de nosotros realizar un registro de usuario, los demás datos como lo son rol, active y avatar los precargaremos luego, y ahora comprobamos de nuevo que se estén enviando estos datos:



The screenshot shows a browser's developer tools Network tab with a POST request to `/auth/register`. The request body is JSON:

```

1 + {
2   "firstname": "Diana",
3   "lastname": "Cortes Rincon",
4   "email": "dianita@gmail.com",
5   "password": "123456"
6 }

```

The response is a 200 OK with a message: "Funciono Perfecto !".

Below the browser is a terminal window titled "push D:\DATA\Documents\mern-app\server" showing the command `yarn run v1.22.19` and the output of the Node.js application. It logs the received data:

```

yarn run v1.22.19
$ nodemon index.js
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
=====
=====
### API REST MERN ####
=====
http://localhost:3000/api/v1/
{
  firstname: 'Diana'
}
{
  firstname: 'Diana',
  lastname: 'Cortes Rincon',
  email: dianita@gmail.com,
  password: '123456'
}

```

Ahora si vamos a realizar el registro verdadero, una vez comprobado que los datos se están enviando vamos a realizar lo siguiente en nuestro controlador:

```

controllers > ls auth.js > ...
1  const User = require("../models/user")
2
3  function register(req, res){
4    const {firstname, lastname, email, password} = req.body
5
6    if(!email) res.status(400).send({msg: "El Email es Obligatorio"})
7    if(!password) res.status(400).send({msg: "La Contraseña es Obligatoria"})
8
9    res.status(200).send({msg:"Funciono Perfecto !"})
10
11 module.exports = {
12   register,
13 }

```

Agregamos en una **const** los datos que vamos a tomar de nuestro **req.body** y hacemos unas condiciones para comprobar que se deben colocar los datos, revisamos de nuevo con **insomnia** y hacemos un pequeño error con el nombre de email y nos debe de salir así:

The screenshot shows a browser's developer tools Network tab with a POST request to `/auth/register`. The request body is JSON:

```

1 + {
2   "firstname": "Diana",
3   "lastname": "Cortes Rincon",
4   "email2": "dianita@gmail.com",
5   "password": "123456"
6 }

```

The response is a 400 Bad Request with a message: "El Email es Obligatorio".

Ahora coloquemos de nuevo nuestro console.log:



```

1 const User = require("../models/user")
2
3 function register(req, res){
4   const {firstname, lastname, email, password} = req.body
5   console.log(req.body)
6
7   if(!email) res.status(400).send({msg: "El Email es Obligatorio"})
8   if(!password) res.status(400).send({msg: "La Contraseña es Obligatoria"})
9
10  res.status(200).send({msg:"Funciona Perfecto !"})
11 }
12
13 module.exports = {
14   register,
15 }

```

```

C:\Program Files\PowerShell\ ...
#####
http://localhost:3000/api/v1/
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
#####
#### API REST MERN #####
#####
http://localhost:3000/api/v1/
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
#####
#### API REST MERN #####
#####
http://localhost:3000/api/v1/
{
  firstname: 'Diana',
  lastname: 'Cortes Rincon',
  email: 'dianita@gmail.com',
  password: '123456'
}

```

Hablemos un poco del tema de seguridad, nuestra contraseña es visible y no debemos pasar este dato de esta manera ya que la contraseña debe ir encriptada por temas de seguridad, así que vamos a configurar primero la creación del usuario y a pasar los datos como un objeto y allí hacemos la configuración de seguridad, realizamos lo siguiente en nuestro **auth.controller.js**:

```

1 const User = require("../models/user")
2
3 function register(req, res){
4   const {firstname, lastname, email, password} = req.body
5   console.log(req.body)
6
7   if(!email) res.status(400).send({msg: "El Email es Obligatorio"})
8   if(!password) res.status(400).send({msg: "La Contraseña es Obligatoria"})
9
10  const user = new User({
11    firstname,
12    lastname,
13    password,
14    email: email.toLowerCase(),
15    role: "User",
16    active: false,
17  })
18
19  console.log(user)
20
21  res.status(200).send({msg:"Funciona Perfecto !"})
22 }
23
24 module.exports = {
25   register,
26 }

```

Pasamos a través de un objeto los datos de nuestro esquema, en el ámbito del rol vamos a dejar que los administradores se creen solamente a través de nuestro y para los demás serán usuarios normales, volvemos a ejecutar en insomnia la ejecución de los datos y como podemos observar nos saldrá lo siguiente:



```
[nodemon] starting `node index.js`
#####
##### API REST MERN #####
#####
http://localhost:3000/api/v1/
{
  firstname: 'Diana',
  lastname: 'Cortes Rincon',
  email: 'dianita@gmail.com',
  password: '123456',
  role: 'User',
  active: false,
  _id: new ObjectId("65158d6d095108fefafed6463")
}
```

Al ver los datos lo primero que nos fijamos es que la **contraseña** esta en texto plano y esto es un problema de seguridad, para ello vamos a encriptarla y vamos a utilizar **bcryptjs** una dependencia que nos permitirá realizar la encriptación, nos dirijimos a las dependencias de yarn:

The screenshot shows the yarn package manager's search results for 'bcryptjs'. On the left, there's a sidebar with navigation links like 'Get Started', 'Features', 'CLI', 'Configuration', 'Advanced', 'Discord', 'GitHub', and a search bar. The main area displays the 'bcryptjs' package details. It shows the package name 'bcryptjs @ 2.4.3', a '2.4.3' badge, and a '7 years ago' timestamp. A note says 'Types are available via @types/bcryptjs'. Below this is the 'README' section, which includes a 'bcrypt.js' file preview. The preview contains code for an optimized bcrypt function in JavaScript, mentioning compatibility with the C++ bcrypt binding and its use in node.js and the browser. It also discusses security considerations, noting that while bcryptjs is compatible with the C++ bcrypt binding, it is written in pure JavaScript and thus slower (about 30%), effectively reducing the number of iterations that can be processed in an equal time span.

Y vamos a colocar lo siguiente en consola **yarn add bcryptjs@2.4.3**

The terminal window shows the command being run: 'yarn add bcryptjs@2.4.3'. The output indicates the process: [1/4] Resolving packages..., [2/4] Fetching packages..., [3/4] Linking dependencies..., and [4/4] Building fresh packages... Success messages include 'Saved lockfile.', 'Saved 1 new dependency.', and 'All dependencies'. The total duration is 'Done in 1.41s.'

Lo primero que realizaremos será la importación de **bcrypt**, dentro de nuestro **auth.controller.js**:

```
controllers > JS auth.controller.js > register
1  const bcrypt = require("bcryptjs")
2  const User = require("../models/user.model")
```



Y ahora lo que haremos será quitar la contraseña de nuestros objetos de datos y vamos a crear una **const** que se llamará **salt** y otra **hashPassword** en donde utilizaremos **bcrypt**:

```
11 const user = new User({
12   firstname,
13   lastname,
14   email: email.toLowerCase(),
15   role: "User",
16   active: false,
17 }
18
19 const salt = bcrypt.genSaltSync(10)
20 const hashPassword = bcrypt.hashSync(password, salt)
21
22 console.log(password)
23 console.log(hashPassword)
24
25 res.status(200).send({msg:"Funciono Perfecto !"})
26 }
```

- **const salt = bcrypt.genSaltSync(10):** Esta línea usa la función `bcrypt.genSaltSync()` para generar una sal de 10 bytes. La función `bcrypt.genSaltSync()` toma un número como argumento, que indica la longitud de la sal. En este caso, estamos usando un número de 10, que es una longitud de sal segura.
- **const hashPassword = bcrypt.hashSync(password, salt):** Esta línea usa la función `bcrypt.hashSync()` para encriptar la contraseña con la sal. La función `bcrypt.hashSync()` toma dos argumentos: la contraseña y la sal. En este caso, estamos usando la contraseña `password` y la sal que generamos en la línea anterior.

Se le llama sal a la cadena aleatoria que se usa para encriptar la contraseña porque, al igual que la sal en la cocina, agrega un toque de sabor y complejidad al hash de la contraseña. La sal hace que el hash sea más difícil de descifrar, incluso si un atacante conoce la contraseña original.

La sal se usa en el proceso de encriptación para mezclar la contraseña con una cadena aleatoria. Esto hace que sea más difícil para un atacante adivinar la contraseña original, incluso si conoce el hash.

La sal también se usa para evitar ataques de diccionario. Los ataques de diccionario son un tipo de ataque de fuerza bruta que utilizan una lista de contraseñas comunes. Si un atacante conoce la sal, puede usarla para crear una lista de contraseñas encriptadas que coincidan con la contraseña original. Sin embargo, si la sal es aleatoria, es mucho más difícil para un atacante crear una lista de contraseñas encriptadas que coincidan.

Ahora que conocemos como funciona activamos nuestro servidor de nuevo y hacemos el envío de la información a través de insomnia para ver lo que nos muestra nuestro log:

```
[nodemon] starting `node index.js`
#####
#####
### API REST MERN ###
#####
http://localhost:3000/api/v1/
123456
$2a$10$MKQ1J6z0Ag/98YYCwgUy20UkDs3EGYEwLxL0z945NtiLUTs0AYwi6
```



@hdtoledo

Acá observamos como nuestra contraseña pasa a ser segura a través de **bcrypt**. Ahora colocamos nuestra contraseña encriptada dentro de nuestro objeto de datos de la siguiente manera y hacemos el log con **user**:

```
19     const salt = bcrypt.genSaltSync(10)
20     const hashPassword = bcrypt.hashSync(password, salt)
21
22     user.password = hashPassword
23
24     console.log(user)
25
26     res.status(200).send({msg:"Funciona Perfecto !"})
```

Si verificamos de nuevo con insomnia enviando los datos en nuestra consola tendremos lo siguiente:

```
http://localhost:3000/api/v1/
{
  firstname: 'Jeronimo',
  lastname: 'Hernandez Cortes',
  email: 'elinsano2023@gmail.com',
  role: 'User',
  active: false,
  _id: new ObjectId("65163372bafac5255a0ca660"),
  password: '$2a$10$MH.qn.T24z10guuYa3ohyuhW3ph1lkwsbec1jfPzsrg/hD2nbuoq'}
```

Ya que nuestra contraseña esta correctamente encriptada, ahora lo que nos queda es realizar el registro verdadero y para ello lo que vamos a hacer es lo siguiente:

```
19   const salt = bcrypt.genSaltSync(10)
20   const hashPassword = bcrypt.hashSync(password, salt)
21
22   user.password = hashPassword
23
24   user.save((error, userStorage) => {
25     if (error){
26       res.status(400).send({msg: "Error al crear el usuario"})
27     } else {
28       res.status(200).send(userStorage)
29     }
30   })
31
32
33 module.exports = {
```

- **user.save((error, userStorage) => {})**: Esta línea guarda el usuario en la base de datos y devuelve una promesa. La promesa se resuelve con el usuario guardado si la operación es exitosa, o con un error si la operación falla.
- **if (error){}**: Esta línea comprueba si el error es nulo. Si el error es nulo, el código envía el usuario guardado al cliente con un código de estado 200.
- **res.status(400).send({msg: "Error al crear el usuario"})**: Esta línea envía un mensaje de error al cliente con un código de estado 400.
- **else {}**: Esta línea se ejecuta si el error no es nulo.
- **res.status(200).send(userStorage)**: Esta línea envía el usuario guardado al cliente con un código de estado 200.

Ahora lo que nos queda es probar el registro realizando el envío de datos a través de insomnia, ejecutamos y en consola nos debe salir lo siguiente:



@hdtoledo

```

1: {
2:   "firstname": "Jimmy",
3:   "lastname": "Lombana",
4:   "email": "jimmy@gmail.com",
5:   "role": "user",
6:   "active": false,
7:   "_id": "65172aac6ebdfb77523a08d4",
8:   "password": "$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG",
9:   "__v": 0
10: }

```

Todo debe salir correctamente y si verificamos la consola no debemos tener errores:

```

LAPTOPHDMI ~ D:\DATA\...\server yarn 18.16.0 1ms
{ hdtledo } yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1

```

Ahora vamos a mongodb y revisamos en atlas en nuestra DB que tendremos unos datos ya registrados:

HECTOR DAVID'S ORG - 2023-04-25 > PROJECT 0 > DATABASES

mern-web-personal

Overview Real Time Metrics Collections Search Profiler Performance Advisor Online Archive Cmd Line Tools

DATABASES: 1 COLLECTIONS: 1

+ Create Database

Search Namespaces

test

users

test.users

STORAGE SIZE: 20KB LOGICAL DATA SIZE: 200B TOTAL DOCUMENTS: 1 INDEXES TOTAL SIZE: 40KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' }

QUERY RESULTS: 1-1 OF 1

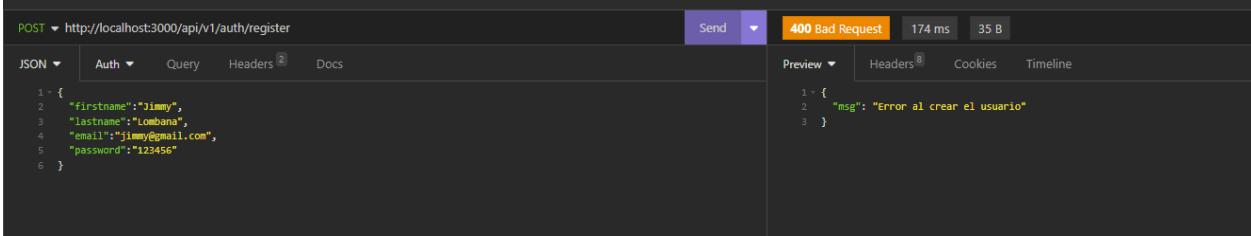
```

_id: ObjectId('65172aac6ebdfb77523a08d4')
firstname: "Jimmy"
lastname: "Lombana"
email: "jimmy@gmail.com"
role: "user"
active: false
password: "$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG"
__v: 0

```



Si volvemos nuevamente a enviar el mismo registro nos debe salir el error de que ya está creado:



POST ▾ http://localhost:3000/api/v1/auth/register

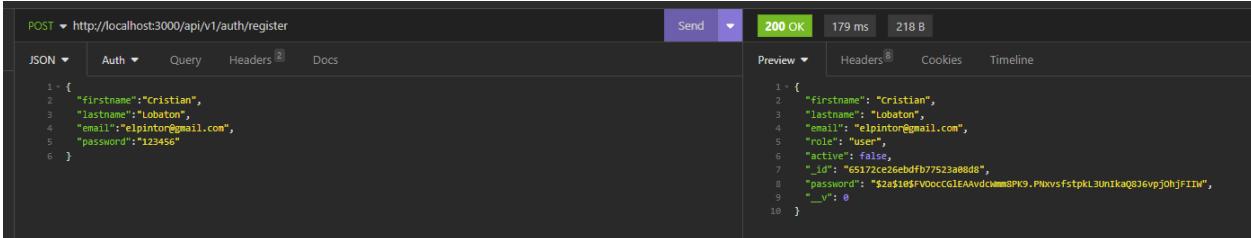
Send ▾ 400 Bad Request | 174 ms | 35 B

Preview ▾ Headers Cookies Timeline

```
1 + {  
2   "firstname": "Jimmy",  
3   "lastname": "Lombana",  
4   "email": "jimmy@gmail.com",  
5   "password": "123456"  
6 }
```

```
1 + {  
2   "msg": "Error al crear el usuario"  
3 }
```

Y si cambiamos de usuario y registramos:



POST ▾ http://localhost:3000/api/v1/auth/register

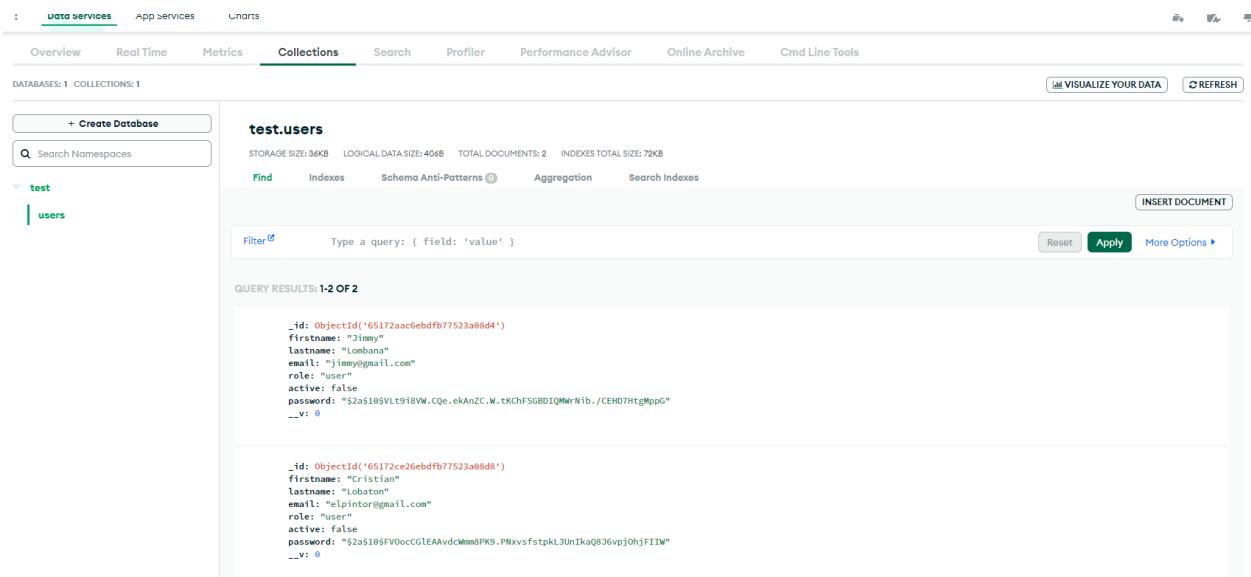
Send ▾ 200 OK | 179 ms | 218 B

Preview ▾ Headers Cookies Timeline

```
1 + {  
2   "firstname": "Cristian",  
3   "lastname": "Lobaton",  
4   "email": "elpintor@gmail.com",  
5   "password": "123456"  
6 }
```

```
1 + {  
2   "firstname": "Cristian",  
3   "lastname": "Lobaton",  
4   "email": "elpintor@gmail.com",  
5   "role": "user",  
6   "active": false,  
7   "_id": "6517cce26ebdfb77523a08d8",  
8   "password": "$2a$10$FVoccCGlEAAvdcMm8PK9.PNxvsfstpkL3UnIkaQ8J6vpjOhjFIIM",  
9   "__v": 0  
10 }
```

Y revisamos en atlas:



DATABASES: 1 COLLECTIONS: 1

+ Create Database

DATA SERVICES App Services Charts

Overview Real Time Metrics Collections Search Profiler Performance Advisor Online Archive Cmd Line Tools

test

users

test.users

STORAGE SIZE: 36KB LOGICAL DATA SIZE: 406B TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 72KB

Find Indexes Schema Anti-Patterns Aggregation Search Indexes

Filter Type a query: { field: 'value' } Reset Apply More Options ▶

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('65172aa6ebdfb77523a08d4')  
firstname: "Jimmy"  
lastname: "Lombana"  
email: "jimmy@gmail.com"  
role: "user"  
active: false  
password: "$2a$10$FVoccCGlEAAvdcMm8PK9.PNxvsfstpkL3UnIkaQ8J6vpjOhjFIIM"  
__v: 0
```

```
_id: ObjectId('6517cce26ebdfb77523a08d8')  
firstname: "Cristian"  
lastname: "Lobaton"  
email: "elpintor@gmail.com"  
role: "user"  
active: false  
password: "$2a$10$FVoccCGlEAAvdcMm8PK9.PNxvsfstpkL3UnIkaQ8J6vpjOhjFIIM"  
__v: 0
```

De esta manera ya tenemos nuestro registro de usuarios configurado.



ACCESS TOKEN Y REFRESH TOKEN

Los "Access Tokens" y "Refresh Tokens" son dos conceptos fundamentales en el ámbito de la autenticación y autorización en aplicaciones web y servicios. Se utilizan comúnmente en sistemas de autenticación y autorización basados en tokens, como OAuth 2.0 y JSON Web Tokens (JWT), para garantizar la seguridad y la gestión de sesiones en aplicaciones web y APIs.

Access Token (Token de Acceso):

1. **Propósito:** Un Access Token es un token de seguridad que se utiliza para autorizar y autenticar a un usuario o una aplicación cuando intenta acceder a recursos protegidos, como datos de usuario o servicios web.
2. **Duración:** Por lo general, los Access Tokens tienen una vida útil relativamente corta y están diseñados para ser válidos solo durante un período breve (pocos minutos u horas).
3. **Uso:** Un cliente (aplicación o usuario) incluye el Access Token en cada solicitud que hace a un servidor para acceder a recursos protegidos. El servidor verifica la validez y los permisos del token antes de permitir el acceso al recurso solicitado.
4. **Renovación:** Debido a su corta vida útil, los Access Tokens a menudo deben renovarse o renovarse antes de que expiren. Esto puede hacerse utilizando el Refresh Token.
5. **Ejemplo:** En una aplicación web que utiliza OAuth 2.0, un usuario inicia sesión y obtiene un Access Token que se incluye en cada solicitud API posterior para acceder a su información de perfil.

Refresh Token (Token de Actualización o Renovación):

6. **Propósito:** Un Refresh Token es un token de seguridad que se utiliza para obtener un nuevo Access Token cuando el Access Token original ha expirado o está a punto de expirar.
7. **Duración:** Los Refresh Tokens suelen tener una vida útil más larga que los Access Tokens, lo que les permite ser válidos durante días o incluso semanas.
8. **Uso:** Cuando un Access Token ha expirado o está cerca de su vencimiento, el cliente (aplicación o usuario) puede utilizar el Refresh Token para solicitar un nuevo Access Token sin necesidad de autenticarse nuevamente. Esto evita que el usuario tenga que volver a ingresar sus credenciales.
9. **Seguridad:** Debido a que los Refresh Tokens tienen una vida útil más larga, deben manejarse con cuidado y almacenarse de manera segura. Los Refresh Tokens son una forma importante de mitigar riesgos de seguridad al evitar que los Access Tokens tengan una vida útil demasiado larga.
10. **Ejemplo:** Un usuario inicia sesión en una aplicación móvil y obtiene tanto un Access Token como un Refresh Token. Cuando el Access Token expira, la aplicación utiliza el Refresh Token para obtener un nuevo Access Token sin requerir que el usuario vuelva a ingresar sus credenciales.



@hdtoledo

Bien para aplicar nuestro control vamos a dirigirnos a **yarn** y ubicamos **JsonWebToken**:

The screenshot shows the yarn package detail page for 'jsonwebtoken'. At the top, there's a navigation bar with links for 'Getting Started', 'Docs', 'Packages', and 'Blog'. On the right, there are social sharing icons for English, GitHub, Twitter, Facebook, and LinkedIn. Below the navigation is a search bar with placeholder text 'Search packages (i.e. babel, webpack, react...)'. The main title is 'Package detail' for 'jsonwebtoken'. The package summary includes: 'jsonwebtoken' (version 9.0.0), 'AUTH0' (status), '56.3M' (size), and 'MIT' (license). It describes the package as 'JSON Web Token implementation (symmetric and asymmetric)'. A 'jwt' link is also present. To the right, there's a sidebar with three cards: 'yarn.pm/jsonwebtoken' with a copy icon, 'auth0/node-jsonwebtoken', and 'npm/jsonwebtoken'.

Ejecutamos el comando **yarn add jsonwebtoken@8.5.1** :

```
LAPTOPHDMI1 D:\DATA\...\server yarn 18.16.0 1ms
{ hdtledo } yarn add jsonwebtoken@8.5.1
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

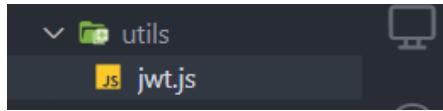
success Saved lockfile.
success Saved 12 new dependencies.
info Direct dependencies
└─ jsonwebtoken@8.5.1
info All dependencies
├─ buffer-equal-constant-time@1.0.1
├─ ecdsa-sig-formatter@1.0.11
├─ jsonwebtoken@8.5.1
├─ jwa@1.4.1
├─ jws@3.2.2
├─ lodash.includes@4.3.0
├─ lodash.isboolean@3.0.3
├─ lodash.isinteger@4.0.4
├─ lodash.isnumber@3.0.3
├─ lodash.isplainobject@4.0.6
└─ lodash.isstring@4.0.1
└─ lodash.once@4.1.1
Done in 1.47s.
```

Revisamos en nuestras dependencias:

```
12 "dependencies": [
13   "bcryptjs": "^2.4.3",
14   "body-parser": "1.20.0",
15   "cors": "2.8.5",
16   "express": "4.18.1",
17   "jsonwebtoken": "8.5.1",
18   "mongoose": "6.6.1",
19   "nodemon": "2.0.20"
20 ]
21 }
```



Ahora vamos a crear el archivo **jwt.js** dentro de **utils** y allí vamos a colocar todo lo que tenga que ver con la creación y demás de nuestro **json web token**:



Y ahora vamos a configurar de la siguiente manera:

```
utils > js jwt.js > ...
1  const jwt = require("jsonwebtoken")
2
3
```

Vamos a nuestro archivo **constants.js** y agregamos una nueva variable llamada **JWT_SECRET_KEY**, el contenido de esta va a ser aleatorio así que pue den perfectamente escribir lo que deseen, y la exportamos junto con las demás:

```
6  const IP_SERVER = "localhost"
7  const JWT_SECRET_KEY = "lkajsdf8967sd09f09453fgh45j3k786as0doif98S7DF908"
8
9  module.exports = {
10    ...
11    DB_USER,
12    DB_PASSWORD,
13    DB_HOST,
14    API_VERSION,
15    IP_SERVER,
16    JWT_SECRET_KEY
}
```

Ahora volvemos a **jwt.js** y hacemos la importación de nuestra nueva variable:

```
utils > js jwt.js > ...
1  const jwt = require("jsonwebtoken")
2  const { JWT_SECRET_KEY } = require("../constants")
3
```

Ahora creamos la función:

```
utils > js jwt.js > ↗ createAccessToken
1  const jwt = require("jsonwebtoken")
2  const { JWT_SECRET_KEY } = require("../constants")
3
4  function createAccessToken(user){
5    const expirationToken = new Date()
6    expirationToken.setHours(expirationToken.getHours() + 3)
7
8    const payload = {
9      token_type: "access",
10     user_id: user._id,
11     iat: Date.now(),
12     exp: expirationToken.getTime()
13   }
14
15   return jwt.sign(payload, JWT_SECRET_KEY)
16 }
```



@hdtoledo

1. Se importa la biblioteca **jsonwebtoken**, que es una biblioteca comúnmente utilizada para crear y verificar tokens JWT en aplicaciones Node.js.
2. Se importa la constante **JWT_SECRET_KEY** desde un archivo llamado **"./constants"**. Esta constante debería contener la clave secreta que se utiliza para firmar y verificar los tokens JWT. La clave secreta es esencial para garantizar la seguridad de los tokens.
3. La función **createAccessToken** acepta un argumento **user**, que se supone que es el objeto de usuario para el cual se está creando el Access Token.
4. Se crea una variable llamada **expirationToken**, que representa la fecha y hora en la que el token expirará. En este caso, se establece la expiración en 3 horas después de la creación del token. Esto significa que el token será válido durante 3 horas a partir de su creación.
5. Luego, se construye un objeto **payload** que contiene la información que se incluirá en el token JWT. El objeto **payload** suele contener información relacionada con el usuario y detalles sobre el token. En este caso, el **payload** incluye:
 - **token_type**: Un campo que indica que este es un token de tipo "access".
 - **user_id**: El identificador único del usuario al que pertenece el token.
 - **iat** (tiempo de emisión): La marca de tiempo que indica cuándo se emitió el token (en milisegundos desde la época UNIX).
 - **exp** (tiempo de expiración): La marca de tiempo en la que el token expirará (en milisegundos desde la época UNIX).
6. Finalmente, se utiliza **jwt.sign()** para firmar el **payload** utilizando la clave secreta **JWT_SECRET_KEY** y se devuelve el token JWT resultante como resultado de la función.

Ahora creamos nuestro **RefreshToken** de la siguiente manera:

```

18 function createRefreshToken(user) {
19   const expirationToken = new Date()
20   expirationToken.getMonth(expirationToken.getMonth() + 1)
21
22   const payload = {
23     token_type: "refresh",
24     user_id: user._id,
25     iat: Date.now(),
26     exp: expirationToken.getTime()
27   }
28
29   return jwt.sign(payload, JWT_SECRET_KEY)
30 }
```

1. La función **createRefreshToken** toma un argumento **user**, que se supone que es el objeto de usuario para el cual se está creando el Refresh Token.
2. Se crea una variable llamada **expirationToken**, que representa la fecha y hora en la que el token expirará. En este caso, se crea una nueva instancia de la fecha actual y luego se llama a



`getMonth` para obtener el mes actual y se le suma 1. Esto se hace para establecer la expiración del token en 1 mes a partir de la creación del token.

3. Luego, se construye un objeto **payload** que contiene la información que se incluirá en el token JWT. Al igual que en el caso del Access Token, el **payload** suele contener información relacionada con el usuario y detalles sobre el token. En este caso, el **payload** incluye:
 - **token_type**: Un campo que indica que este es un token de tipo "refresh".
 - **user_id**: El identificador único del usuario al que pertenece el token.
 - **iat** (tiempo de emisión): La marca de tiempo que indica cuándo se emitió el token (en milisegundos desde la época UNIX).
 - **exp** (tiempo de expiración): La marca de tiempo en la que el token expirará, que se calcula en base al mes actual más uno (en milisegundos desde la época UNIX).
4. Finalmente, se utiliza `jwt.sign()` para firmar el **payload** utilizando la clave secreta `JWT_SECRET_KEY`, y se devuelve el token JWT resultante como resultado de la función.

Ahora creamos nuestra función para decodificar nuestro token:

```
31
32   function decoded(token) {
33     return jwt.decode(token, JWT_SECRET_KEY, true)
34   }
35
```

1. La función **decoded** toma un argumento llamado **token**, que se supone que es el token JWT que se desea decodificar.
2. Dentro de la función, se utiliza la función `jwt.decode()` de la biblioteca `jsonwebtoken` para decodificar el token. Los argumentos de esta función son:
 - **token**: El token JWT que se desea decodificar.
 - **JWT_SECRET_KEY**: La clave secreta que se utiliza para verificar la firma del token. Esta variable debe haber sido definida en algún lugar de tu código y debe contener la clave secreta necesaria para validar la firma del token.
 - **true**: Este tercer argumento indica que se debe verificar la firma del token. Cuando se establece en **true**, la biblioteca verificará la firma del token para asegurarse de que sea válido.
3. La función `jwt.decode()` decodifica el token y devuelve un objeto JavaScript que representa el contenido del token. Este objeto generalmente contiene los datos incluidos en el token, como el **payload** que contiene información sobre el usuario o la entidad a la que pertenece el token.
4. Finalmente, la función **decoded** devuelve el objeto decodificado como resultado.



Y por último creamos la exportación de nuestras funciones:

```
35
36 module.exports = [
37   createAccessToken,
38   createRefreshToken,
39   decoded
40 ]
```

LOGIN DE USUARIO:

Para crear nuestro login lo primero que vamos a realizar es ir a nuestro **auth.controller.js** y vamos a importar **jwt**:

```
controllers > js auth.controller.js > 📁 jwt
1 const bcrypt = require("bcryptjs")
2 const User = require("../models/user.model")
3 const jwt = require("../utils/jwt")
```

Y ahora nos ubicamos abajo para crear nuestra función:

```
31
32 function login(req, res){
33   const { email, password } = req.body
34   if(!email) res.status(400).send({msg: "El email es obligatorio"})
35   if(!password) res.status(400).send({msg: "La contraseña es obligatoria"})
36
37   const emailLowerCase = email.toLowerCase()
38
39   User.findOne({ email: emailLowerCase }, (error, userStore) => {
40     if(error){
41       res.status(500).send({msg: "Error del servidor"})
42     } else {
43       console.log("Password: ", password)
44       console.log(userStore)
45     }
46   })
47 }
```

1. La función **login** toma dos argumentos: **req** y **res**, que representan la solicitud (request) y la respuesta (response) HTTP, respectivamente. Esta función probablemente se utiliza como controlador de una ruta de inicio de sesión en tu servidor web.
2. Se desestructuran los datos de la solicitud **req.body** para obtener el **email** y la **password** proporcionados por el usuario en el formulario de inicio de sesión.
3. Se realiza una validación básica para verificar que se hayan proporcionado tanto el **email** como la **password**. Si falta alguno de estos campos, se responde con un código de estado HTTP 400 (Bad Request) y se envía un objeto JSON con un mensaje de error que indica qué campo falta.
4. Se convierte el **email** a minúsculas utilizando **toLowerCase()**. Esto se hace para asegurarse de que la búsqueda en la base de datos sea insensible a mayúsculas y minúsculas, ya que los correos electrónicos suelen ser insensibles a mayúsculas y minúsculas.

- Se utiliza `User.findOne()` para buscar un documento de usuario en la base de datos que coincida con el `email` proporcionado. Esta función espera una consulta de búsqueda y una función de devolución de llamada (callback) que se ejecutará una vez que se complete la búsqueda.
- En la función de devolución de llamada de `User.findOne()`, se maneja la respuesta de la búsqueda. Si ocurre un error durante la búsqueda, se responde con un código de estado HTTP 500 (Internal Server Error) y se envía un mensaje de error indicando un problema en el servidor.

Por último, si la búsqueda se realiza con éxito y se encuentra un usuario con el correo electrónico proporcionado, se muestra información de depuración en la consola. Esto incluye la contraseña proporcionada en el formulario de inicio de sesión (`password`) y los datos del usuario encontrado (`userStore`). Esto lo hacemos para validar que la información se nos esté mostrando, pero luego lo cambiaremos porque no debemos dejar esto así por seguridad.

Y como paso final exportamos login:

```

49
50   module.exports = {
51     register,
52     login
53   }
54

```

Ahora nos dirigimos a `auth.router.js` y creamos nuestro **endpoint** de login:

```

router > auth.router.js > ...
1  const express = require("express")
2  const AuthController = require("../controllers/auth.controller")
3
4  const api = express.Router()
5
6  api.post("/auth/register", AuthController.register)
7  api.post("/auth/login", AuthController.login)
8
9
10 module.exports = api

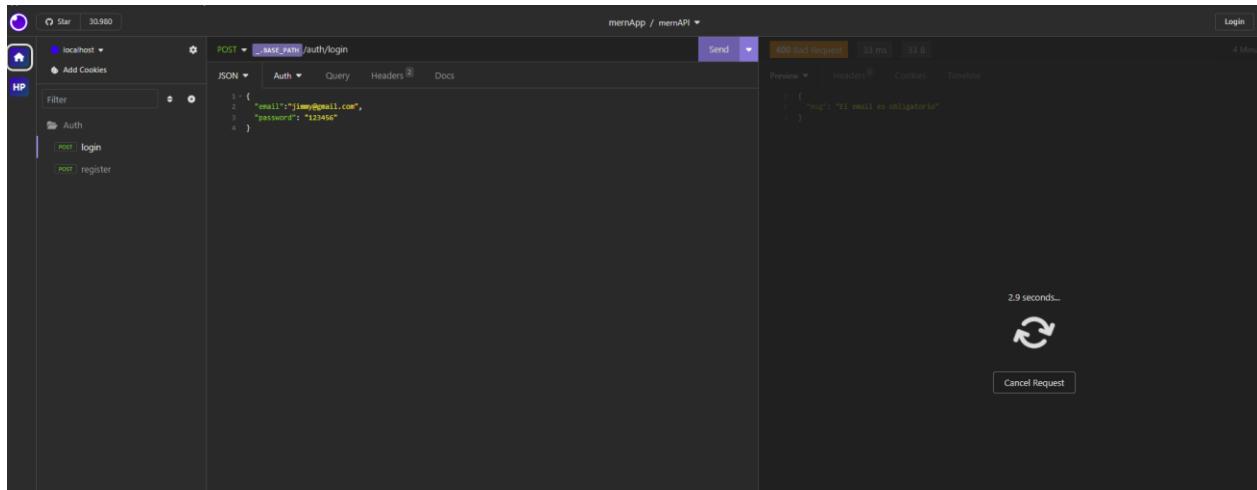
```

Ahora en nuestro **insomnia** vamos a crear un nuevo **endpoint** para el **login**:

The screenshot shows the Insomnia REST Client interface. On the left, there's a sidebar with a tree view showing an 'Auth' folder containing 'POST login' and 'POST register'. The main area shows a POST request to '/auth/login'. The response tab shows a 400 Bad Request status with a JSON body: { "msg": "El email es obligatorio" }. The preview tab shows an empty response.

Si hacemos una prueba sin enviar aun nada nos debe devolver el mail es obligatorio, ahora vamos a colocar la información de logeo:





Al enviar la información notamos que no recibimos nada de validación debido a que nuestra contraseña esta encriptada, si revisamos y comparamos en consola:

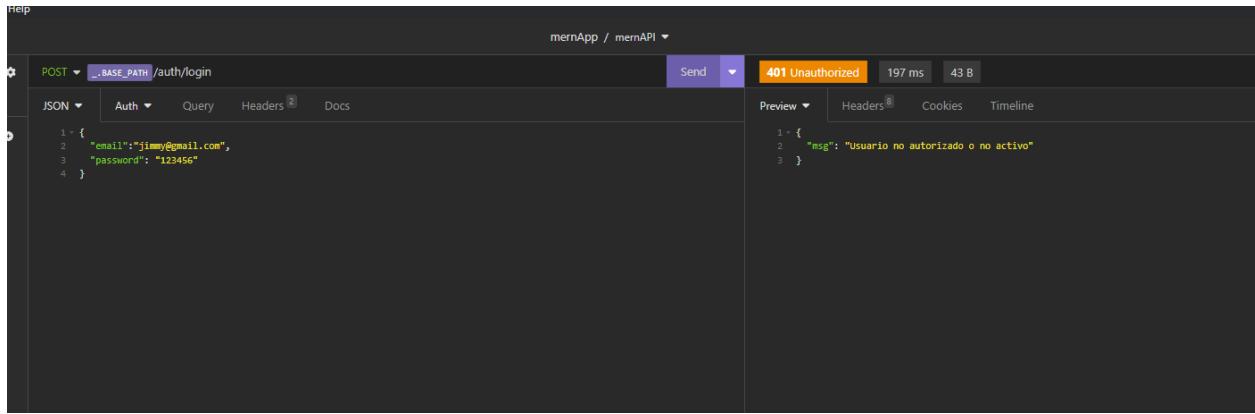
```
at next 13 |     document.setNext();
at Route.dispatch (D:\DATA\Documents\mern-app\server\node_modules\express\lib\router\index.js:102:12)
Password: 123456
{
  _id: new ObjectId("65172aac6ebdfb77523a08d4"),
  firstname: 'Jimmy',
  lastname: 'Lombana',
  email: 'jimmy@gmail.com',
  role: 'user',
  active: false,
  password: '$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG',
  __v: 0
}
```

Tenemos nuestro password sin encriptar y en la db encriptada a lo cual no se validarán para ello debemos hacer el proceso de desencriptar, entonces vamos a realizar lo siguiente en **aut.controller.js**:

```
controllers > auth.controller.js > ...
32   function login(req, res){
33     const { email, password } = req.body
34     if(!email) res.status(400).send({msg: "El email es obligatorio"})
35     if(!password) res.status(400).send({msg: "La contraseña es obligatoria"})
36
37     const emailLowerCase = email.toLowerCase()
38
39     User.findOne({ email: emailLowerCase }, (error, userStore) => {
40       if(error){
41         res.status(500).send({msg: "Error del servidor"})
42       } else {
43         bcrypt.compare(password, userStore.password, (bcryptError, check) => {
44           if (bcryptError) {
45             res.status(500).send({msg: "Error del servidor"})
46           } else if (!check) {
47             res.status(400).send({msg: "Usuario o Contraseña incorrecta"})
48           } else if (!userStore.active) {
49             res.status(401).send({msg: "Usuario no autorizado o no activo"})
50           } else {
51             res.status(200).send({msg: "Login Ok"})
52           }
53         })
54       }
55     })
56 }
```

Acá estamos haciendo las diferentes validaciones, la primera es la comparación de las contraseñas, y dentro de esta hacemos varias opciones de las cuales hacemos errores del servidor, del usuario o contraseña erróneas, si el usuario no está activo y por último si todo va bien enviamos un mensaje de ok.

Ahora vamos a nuestro insomnia a realizar la petición:



The screenshot shows the Insomnia API client interface. A POST request is made to `__BASE_PATH/auth/login`. The request body is a JSON object with `email` set to `jimmy@gmail.com` and `password` set to `123456`. The response status is **401 Unauthorized**, with a response time of 197 ms and a body size of 43 B. The response message is `{ "msg": "Usuario no autorizado o no activo" }`.

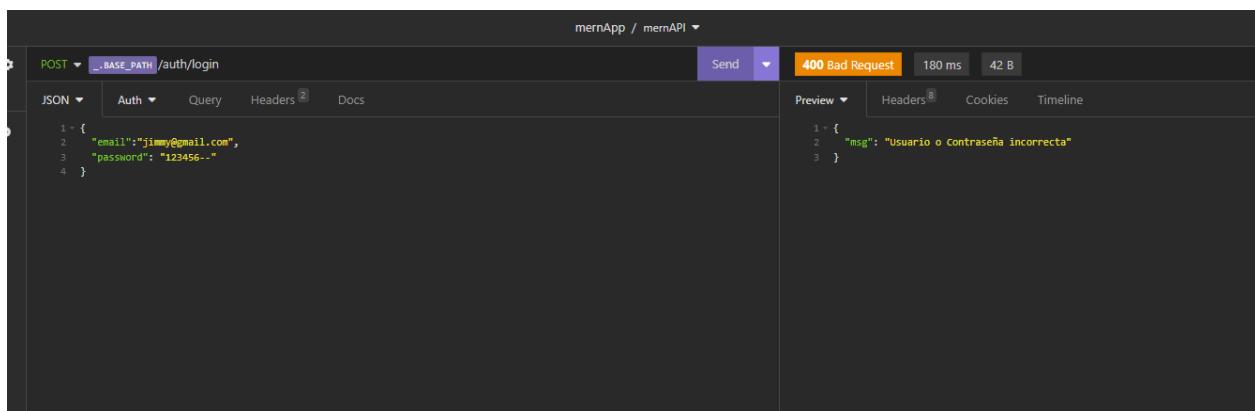
Y como resultado obtenemos usuario no autorizado o no activo, si revisamos nuestra db vamos a observar que el usuario no está activado:



QUERY RESULTS: 1-2 OF 2

```
1 _id: ObjectId('65172aac6ebdfb77523a08d4')
2 firstname: "Jimmy"
3 lastname: "Lombana"
4 email: "jimmy@gmail.com"
5 role: "user"
6 active: false
7 password: "$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG"
8 __v: 0
```

Ahora colocamos una contraseña incorrecta y enviamos:



The screenshot shows the Insomnia API client interface. A POST request is made to `__BASE_PATH/auth/login`. The request body is a JSON object with `email` set to `jimmy@gmail.com` and `password` set to `123456--`. The response status is **400 Bad Request**, with a response time of 180 ms and a body size of 42 B. The response message is `{ "msg": "Usuario o Contraseña incorrecta" }`.

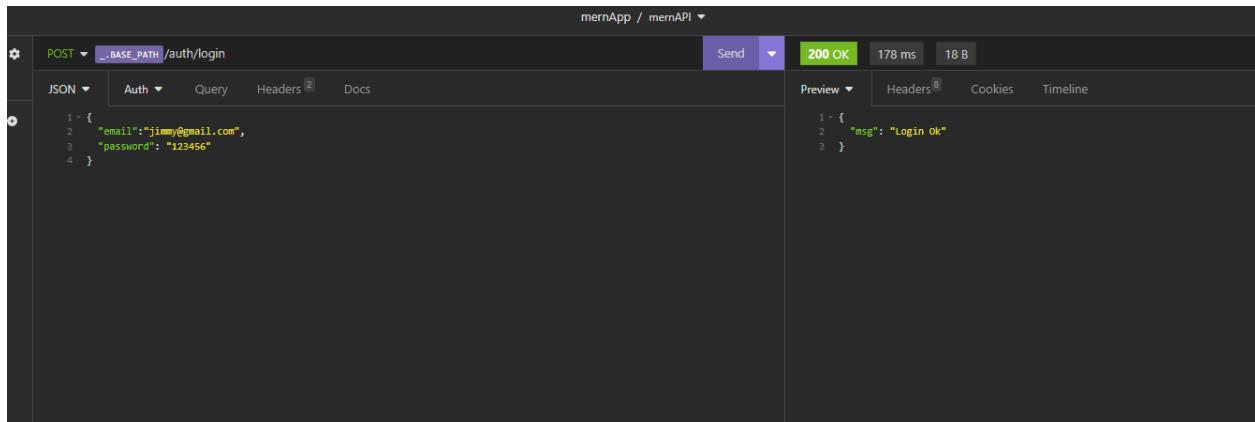
Ahora vamos a modificar nuestro usuario para que este activo, nos dirigimos a nuestra db y actualizamos:



QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('65172aac6ebdfb77523a08d4')
firstname: "Jimmy"
lastname: "Lombana"
email: "jimmy@gmail.com"
role: "user"
active: true
password: "$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG"
---v: 0
```

Nuevamente hacemos el envío de la información:



The screenshot shows a POST request to `/auth/login`. The request body is JSON with the following content:

```
1: {
2:   "email": "jimmy@gmail.com",
3:   "password": "123456"
4: }
```

The response status is **200 OK**, with a response time of 178 ms and 18 B. The response body is:

```
1: {
2:   "msg": "Login Ok"
3: }
```

Y ahí obtenemos nuestro mensaje de que el login está ok.

Ahora vamos a modificar la ultima parte para que se nos generen los JWT tanto de Access como de Refresh:

```
48
49   } else if (!userStore.active) {
50     res.status(401).send({msg: "Usuario no autorizado o no activo"})
51   } else {
52     res.status(200).send({
53       access: jwt.createAccessToken(userStore),
54       refresh: jwt.createRefreshToken(userStore)
55     })
56   }
57 }
58 }
59 }
60 }
```

De esta manera si volvemos a enviar los datos de login, vamos a observar los tokens:



POST `BASE_PATH/auth/login`

JSON **Auth** **Query** **Headers** **Docs**

```

1 + {
2   "email": "jimmy@gmail.com",
3   "password": "123456"
4 }

```

200 OK 199 ms 457 B

Preview Headers Cookies Timeline

```

1 + {
2   "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFnMmViZGZiNzc1MjNhMDhkNCIsIm1hdC16MTY5NjIwNjgxODYw0SwiZXhwIjoxNjk2MjE3NjE4NjA5fQ.EgpW99EW_EA1QmN15Egd7aYa87waEOWb2gwoPhYPBPM"
3   "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFnMmViZGZiNzc1MjNhMDhkNCIsIm1hdC16MTY5NjIwNjgxODYw0SwiZXhwIjoxNjk2MjE3NjE4NjA5fQ.EgpW99EW_EA1QmN15Egd7aYa87waEOWb2gwoPhYPBPM"
4

```

Y cada vez que lo generemos se van a cambiar:

POST `BASE_PATH/auth/login`

JSON **Auth** **Query** **Headers** **Docs**

```

1 + {
2   "email": "jimmy@gmail.com",
3   "password": "123456"
4 }

```

200 OK 163 ms 457 B

Preview Headers Cookies Timeline

```

1 + {
2   "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFnMmViZGZiNzc1MjNhMDhkNCIsIm1hdC16MTY5NjIwNjgxODYw0SwiZXhwIjoxNjk2MjE3NjE4NjA5fQ.EgpW99EW_EA1QmN15Egd7aYa87waEOWb2gwoPhYPBPM"
3   "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpxVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFnMmViZGZiNzc1MjNhMDhkNCIsIm1hdC16MTY5NjIwNjgxODYw0SwiZXhwIjoxNjk2MjE3NjE4NjA5fQ.EgpW99EW_EA1QmN15Egd7aYa87waEOWb2gwoPhYPBPM"
4

```

Para validar el token es muy sencillo lo copiamos y nos dirigimos a la web jwt.io en donde lo validara:

Encoded PASTE A TOKEN HERE

Decoded EDIT THE PAYLOAD AND SECRET

Algorithm HS256

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYOUT: DATA

```
{
  "token_type": "access",
  "user_id": "65172aacebd577523a08d4",
  "iat": 1696206818609,
  "exp": 1696217618609
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) □ secret base64 encoded
```

De esta manera dejamos nuestro login listo y funcional.



@hdtoledo

REFRESCANDO EL ACCESS TOKEN

En esta parte vamos a refrescar el **Access token** una vez haya caducado, para ello lo enviamos mediante el **refresh token**, entonces nos dirigimos a **auth.controller.js** y creamos la función de **refreshAccessToken**:

```
controllers > js auth.controller.js > ...
60
61  function refreshAccessToken(req, res){
62    const { token } = req.body
63    const { user_id } = jwt.decoded(token)
64
65    User.findOne({ _id: user_id }, (error, userStorage) => {
66      if(error){
67        res.status(500).send({msg: "Error del servidor"})
68      } else {
69        res.status(200).send({
70          accessToken: jwt.createAccessToken(userStorage)
71        })
72      }
73    })
74  }
75 }
```

1. La función **refreshAccessToken** toma dos argumentos: **req** y **res**, que representan la solicitud (request) y la respuesta (response) HTTP, respectivamente. Esta función probablemente se utiliza como controlador de una ruta de renovación de tokens en tu servidor web.
2. Se desestructuran los datos de la solicitud **req.body** para obtener el **token**, que es el token JWT que se utilizará para solicitar la renovación del Access Token.
3. Luego, se utiliza **jwt.decoded(token)** para decodificar el token JWT que se proporcionó en la solicitud. Esto debería devolver un objeto que contiene información del usuario y otros datos incluidos en el token.
4. Se busca un usuario en la base de datos utilizando **User.findOne()**, donde se especifica que se desea encontrar un usuario cuyo **_id** coincida con el **user_id** extraído del token decodificado.
5. En la función de devolución de llamada de **User.findOne()**, se maneja la respuesta de la búsqueda. Si ocurre un error durante la búsqueda, se responde con un código de estado HTTP 500 (Internal Server Error) y se envía un mensaje de error indicando un problema en el servidor.
6. Si la búsqueda se realiza con éxito y se encuentra el usuario correspondiente, se crea un nuevo Access Token utilizando **jwt.createAccessToken(userStorage)**, donde **userStorage** es el objeto del usuario recuperado de la base de datos.
7. Finalmente, se responde con un código de estado HTTP 200 (OK) y se envía un objeto JSON que contiene el nuevo Access Token generado en la propiedad **accessToken**. Este nuevo Access Token se puede utilizar para autenticar y autorizar las solicitudes del usuario sin necesidad de volver a iniciar sesión.



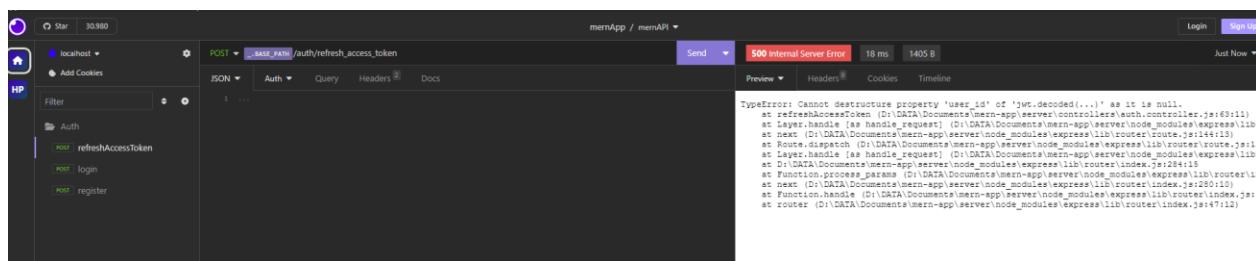
Por último, agregamos la exportación de nuestra función:

```
76
77 module.exports = {
78   register,
79   login,
80   refreshAccessToken
81 }
82
```

Ahora agregamos nuestra ruta en **auth.router.js**:

```
5
6   api.post("/auth/register", AuthController.register)
7   api.post("/auth/login", AuthController.login)
8   api.post("/auth/refresh_access_token", AuthController.refreshAccessToken)
9
10
11 module.exports = api
```

Y ahora vamos a insomnia y creamos nuestra petición:

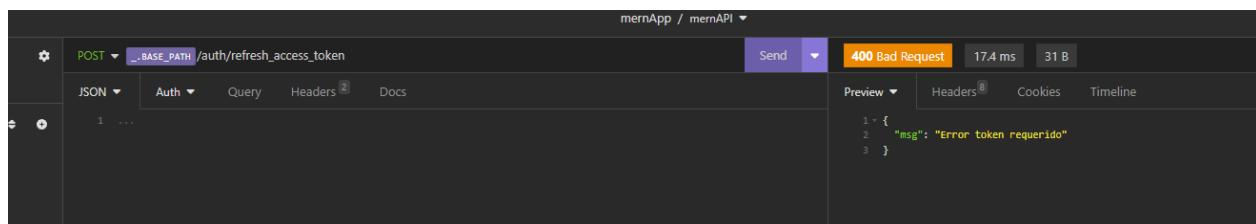


The screenshot shows the Insomnia REST Client interface. A POST request is being made to `/auth/refresh_access_token`. The response is a **500 Internal Server Error** with a timestamp of 18 ms and a body size of 1405 B. The error message in the preview tab is: `TypeError: Cannot destructure property 'user_id' of 'jwt.decoded(..)' as it is null.` This indicates a bug in the `refreshAccessToken` function where it tries to destructure a property from a null object.

Si le damos a enviar solo por probar nos mostrara un error y es normal ya que no hemos definido un mensaje para el error así que vamos a crear uno en **auth.controller.js**:

```
61   function refreshAccessToken(req, res){
62     const { token } = req.body
63
64     if(!token) res.status(400).send({msg: "Error token requerido"})
65
66     const { user_id } = jwt.decoded(token)
67
68     User.findOne({ _id: user_id }, (error, userStorage) => {
```

Si volvemos a probar ya tendremos el mensaje:



The screenshot shows the Insomnia REST Client interface again. A POST request is being made to `/auth/refresh_access_token`. The response is a **400 Bad Request** with a timestamp of 17.4 ms and a body size of 31 B. The message in the preview tab is: `1: { 2: "msg": "Error token requerido" 3: }`.

Y ahora vamos a probar a pasarle nuestro token para ello nos dirigimos en insomnia a nuestro login y copiamos el refresh token:



Y lo vamos a colocar en la solicitud de `refreshAccessToken`:

De esta manera nos fijamos que los tokens son diferentes y que automáticamente hizo el refresh token y está funcionando.

ESTRUCTURA API USER

Acá vamos a obtener, crear, eliminar y todo lo referente a nuestros usuarios, esto lo realizamos a través de los **endpoints** protegidos, ya que solo los usuarios registrados podrán realizarlo, para ello un middleware lo realizará, así que nos vamos a crear un nuevo controlador dentro de la carpeta **controllers** y su nombre será **user.controller.js** y allí colocaremos lo siguiente:

```

controllers > JS user.controller.js ...
1  async function getMe(req, res) {
2      res.status(200).send({msg: "Ok"})
3  }
4
5  module.exports = {
6      ...getMe
7  }

```

Ahora creamos la ruta, vamos a crear dentro de **router** nuestro archivo **user.router.js** y lo dejamos así:



```

router > user.router.js > ...
1 const express = require("express")
2 const UserController = require("../controllers/user.controller")
3
4 const api = express.Router()
5
6 api.get("/user/me", UserController.getMe)
7 |
8 module.exports = api

```

Ahora nos vamos para **app.js** y dentro vamos a importar nuestra ruta:

```

8
9 // Importar rutas
10 const authRoutes = require("./router/auth.router")
11 const userRoutes = require("./router/user.router")
12
// Configuración de rutas

```

Y ahora configuraremos allí mismo la ruta:

```

22
23 // Configurar Rutas
24 app.use(`/api/${API_VERSION}`, authRoutes)
25 app.use(`/api/${API_VERSION}`, userRoutes)
26
27

```

Si todo esta correcto podemos utilizar nuestra nueva ruta, para ello vamos a insomnia y creamos una nueva carpeta llamada **user** y hacemos una nueva solicitud tipo **GET** y nos debe devolver nuestro mensaje de Ok:

The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with a tree view of API endpoints under 'User' (including 'GetMe') and 'Auth' (including 'refreshAccessToken' and 'login'). The main area shows a request configuration: method 'GET', base path '_BASE_PATH', endpoint '/user/me'. The response header shows '200 OK', '9.88 ms', and '12 B'. The preview tab shows the JSON response: { "msg": "OK" }.

Vamos a dejar hasta acá la configuración base de nuestro user.



MIDDLEWARE DE AUTENTICACION

Un middleware de autenticación es un componente de software que se utiliza en el desarrollo de aplicaciones web y servicios para verificar la identidad de un usuario o entidad que realiza una solicitud antes de permitir el acceso a ciertos recursos o funcionalidades. Su función principal es garantizar que solo los usuarios autenticados y autorizados puedan acceder a ciertas partes de una aplicación o servicio.

Aquí hay una descripción más detallada de cómo funciona un middleware de autenticación:

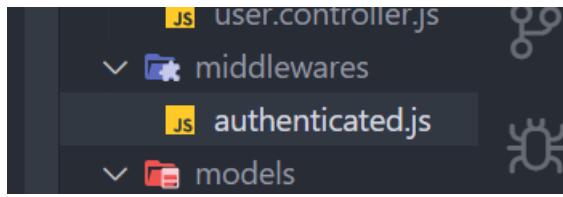
1. **Intercepción de Solicitudes:** El middleware de autenticación se coloca en el flujo de procesamiento de solicitudes HTTP entre el cliente y el servidor. Cuando un cliente realiza una solicitud a una ruta o recurso protegido, el middleware de autenticación interviene antes de que la solicitud alcance la función controladora correspondiente.
2. **Verificación de Identidad:** El middleware de autenticación verifica la identidad del cliente, comúnmente a través de credenciales como un nombre de usuario y una contraseña, tokens de acceso, certificados digitales u otros mecanismos de autenticación. Esta verificación puede implicar consultar una base de datos de usuarios, verificar la validez de un token, o realizar otro proceso de autenticación.
3. **Generación de Contexto de Usuario:** Una vez que se ha autenticado al cliente, el middleware de autenticación puede generar un "contexto de usuario". Este contexto de usuario suele contener información relevante sobre el cliente autenticado, como su identificador, roles y permisos.
4. **Autorización:** Además de la autenticación, algunos middlewares de autenticación también manejan la autorización. Esto implica verificar si el usuario autenticado tiene permisos para acceder al recurso o realizar la acción solicitada. Si el usuario no está autorizado, se le deniega el acceso.
5. **Redirección o Respuesta de Error:** Si la autenticación falla o el usuario no está autorizado, el middleware puede redirigir al usuario a una página de inicio de sesión, mostrar un mensaje de error o tomar otras acciones definidas por el desarrollador.

Los middlewares de autenticación son esenciales para garantizar la seguridad de una aplicación al proteger recursos y funcionalidades sensibles. Se utilizan en una variedad de contextos, como aplicaciones web, API REST, aplicaciones móviles y servicios en la nube. Al implementar un middleware de autenticación sólido, puedes asegurarte de que solo los usuarios legítimos tengan acceso a los datos y las funcionalidades que están autorizados a utilizar.

Para realizar nuestro CRUD debemos tener protegidos los **endpoints**, es decir que solo los usuarios registrados podrán acceder a estas funciones y para ello vamos a utilizar el middleware para que valide que el usuario está registrado y le permita ejecutar las funciones a través de los **endpoints** protegidos.

Nos dirigimos a nuestra carpeta **middlewares** y allí creamos el archivo **authenticated.js** y hacemos lo siguiente:





```
middlewares > JS authenticated.js > ...
1 const jwt = require("jsonwebtoken")
2
3 function assureAuth(req, res, next) {
4     console.log("Hola estoy en assure auth")
5     next()
6 }
7
8 module.exports = {
9     assureAuth
10 }
```

Aunque en este código no vamos a ejecutar validaciones importantes vamos a entender su funcionamiento y para ello vamos a dirigirnos a **user.router.js** y lo vamos a importar y ejecutar:

```
router > JS user.router.js > md_auth
1 const express = require("express")
2 const UserController = require("../controllers/user.controller")
3 const md_auth = require("../middlewares/authenticated")
4
```

Y ahora lo vamos a colocar en medio de la ejecución de nuestra ruta como un array, recuerden que podremos colocar no solo uno sino varios middlewares que se encarguen de validar nuestras rutas:

```
4
5 const api = express.Router()
6
7 api.get("/user/me", [md_auth.assureAuth], UserController.getMe)
8
9 module.exports = api
```

Y ahora nos dirigimos a nuestro insomnia y realizamos una solicitud nuevamente a **/user/me**

The screenshot shows the insomnia REST client. The sidebar on the left has a tree structure with 'User' expanded, showing 'GET GetMe' selected. The main panel on the right shows a GET request to '/user/me'. The top bar indicates a successful response (200 OK), with a response time of 19.7 ms and a size of 12 B. The 'Preview' tab displays the JSON response: { "msg": "Ok" }.

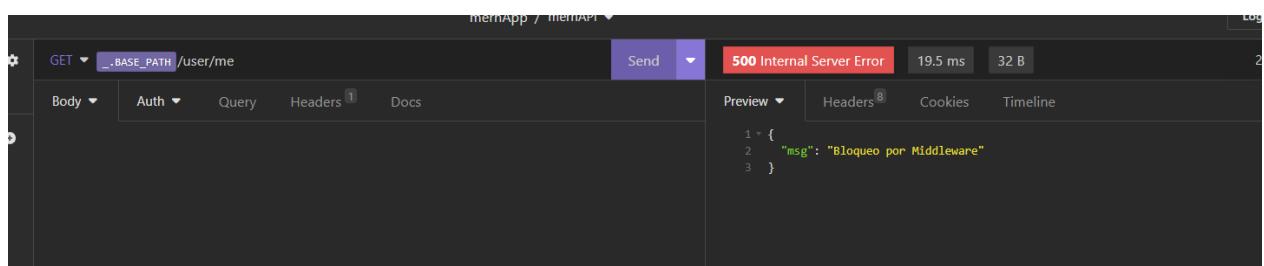
Hasta ahí todo va normal, pero si revisamos en consola obtendremos el mensaje:

```
[nodemon] starting - node index.js
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
Hola estoy en asure auth
```

Vamos a colocarle un status 500 para observar el mensaje en insomnia:

```
middlewares > js authenticated.js > asureAuth > msg
1 const jwt = require("jsonwebtoken")
2
3 function asureAuth(req, res, next) {
4     console.log("Hola estoy en asure auth")
5     res.status(500).send({msg: "Bloqueo por Middleware"})
6     next()
7 }
8
9 module.exports = {
10     asureAuth
11 }
```

Y en nuestro insomnia observamos lo siguiente:



The screenshot shows the insomnia API client interface. A GET request is made to `/_BASE_PATH/user/me`. The response status is **500 Internal Server Error**, with a response time of 19.5 ms and a body size of 32 B. The response body is a JSON object with a single key `msg` containing the value `"Bloqueo por Middleware"`.

De esta manera ya entendemos que va a realizar nuestro middleware, así que vamos a colocarle la lógica a nuestra función de la siguiente manera:

```
middlewares > js authenticated.js > ...
1 const jwt = require("jsonwebtoken")
2
3 function asureAuth(req, res, next) {
4     console.log(req.headers.authorization)
5     next()
6 }
7
8 module.exports = {
9     asureAuth
10 }
```

Vamos a utilizar `req` para poder hacer el envío y así acceder a la autorización, lo colocamos en un log para observar que nos trae, si damos a enviar la petición en insomnia y revisamos la consola nos saldrá lo siguiente:



@hdtoledo

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
undefined
```

Nos dice que no esta definido para lo cual debemos enviarle el token que creamos a través de insomnia:

The screenshot shows the Insomnia interface. On the left, there's a sidebar with a tree view of API endpoints: User (GetMe, refreshAccessToken), Auth (login, register). The main area shows a POST request to `/auth/login`. The request body is JSON with fields `email` and `password`. The response is a 200 OK status with a large JSON object containing `access` and `refresh` tokens.

Lo colocamos en los **headers** y lo dejamos de la siguiente manera:

The screenshot shows the Insomnia interface. A GET request is made to `/user/me`. In the Headers tab, there is an `Authorization` header set to `Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ...`. The response is a 200 OK status with a JSON object containing a `msg` field with the value `"ok"`.

Bearer es un término utilizado en el contexto de autenticación y autorización en aplicaciones web y servicios. Se refiere a un tipo de token que se utiliza para identificar y autenticar a un usuario o una entidad que realiza una solicitud a un servidor.

En el contexto de la autenticación basada en tokens, como en el caso de JSON Web Tokens (JWT) o OAuth, "Bearer" se utiliza como parte de la cabecera de autorización de una solicitud HTTP. La cabecera de autorización tiene el siguiente formato:

Authorization: Bearer <token>

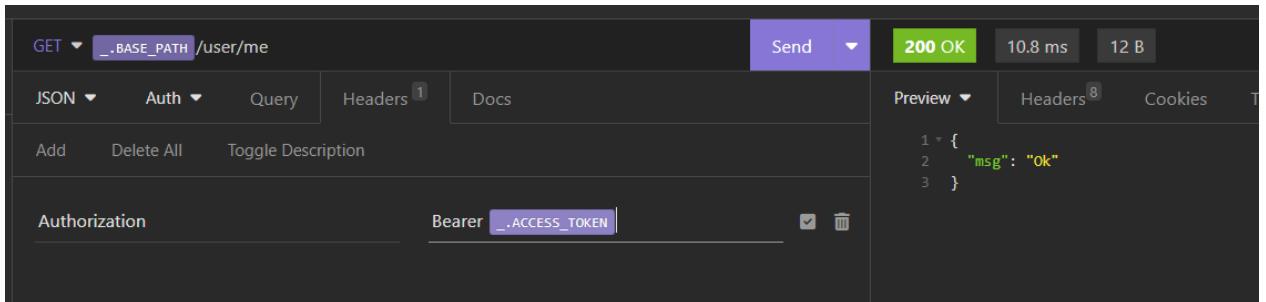
Al darle enviar vamos a obtener el token en consola:

```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9eyJ0b2tlb190eXBIIjo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFjNmViZGZiNzc1MjNhMDhkNCIsImhdCI6MTY5Nj12MTU4MjY0NywiZxhwIjoaNjk2MjcyMzgyNjQ3fQ.33UU8sa18C2wKnJ9CnxOPSladOyIKsxUN0uzF5Z3xfg|
```

De esta manera ya lo vemos reflejado, pero vamos a automatizar un poco el proceso para no tener que estar copiando y pegando, así que vamos a la configuración de las variables de entorno y vamos a agregar una que se llame **Access_Token** y pegamos nuestro token allí:



Cerramos y volvemos nuevamente a configurar de la siguiente manera:



Y si volvemos a ejecutar:

```

#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFjNmViZGZiNzc1MjNhMDhkNCIsImhdCI6MTY5NjI2MTU4MjY0NywiZXhwIjoxNjk2MjcyMzgyNjQ3fQ.33UU8sa18C2wKnJ9Cnx0PSlodDyikSxUN0uzF5Z3xfg
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbl90eXB1Ijo1YWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFjNmViZGZiNzc1MjNhMDhkNCIsImhdCI6MTY5NjI2MTU4MjY0NywiZXhwIjoxNjk2MjcyMzgyNjQ3fQ.33UU8sa18C2wKnJ9Cnx0PSlodDyikSxUN0uzF5Z3xfg
|
```

Ya lo estaremos viendo reflejado allí, ahora vamos a colocar lo siguiente en nuestra función para poder validar nuestra autenticación:

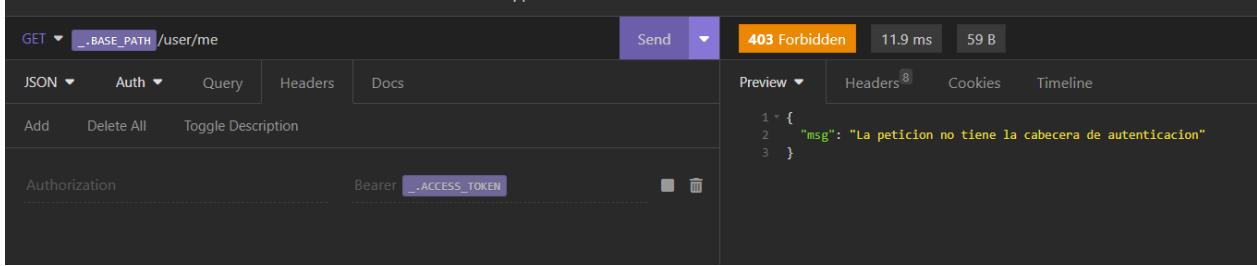
```

middlewares > js authenticated.js > ...
1  const jwt = require("jsonwebtoken")
2
3  function assureAuth(req, res, next) {
4    if(!req.headers.authorization){
5      res.status(403).send({msg: "La petición no tiene la cabecera de autenticación"})
6    }
7    next()
8  }
9
10 module.exports = {
11   assureAuth
12 }

```



Ahora hagamos el ejercicio de nuevo en insomnia para validar que está saliendo el error, quitamos el check y enviamos:



The screenshot shows the Insomnia REST Client interface. A GET request is made to `_BASE_PATH/user/me`. The response is a 403 Forbidden status with a message in Spanish: "La peticion no tiene la cabecera de autenticacion". This indicates that the token was not included in the Authorization header.

Nos muestra el error que acabamos de configurar y va todo perfecto hasta acá, ahora lo que realizaremos será extraer nuestro token para poderlo procesar y para ello vamos a realizar lo siguiente:

```
middlewares > js authenticated.js > asureAuth
1  const jwt = require("jsonwebtoken")
2
3  function asureAuth(req, res, next) {
4      if(!req.headers.authorization){
5          res.status(403).send({msg: "La peticion no tiene la cabecera de autenticacion"})
6      }
7
8      const token = req.headers.authorization.replace("Bearer", "")
9      console.log(token)
10
11     next()
12 }
13
14 module.exports = {
15     asureAuth
16 }
```

Almacenamos en nuestra **const** el valor del token y remplazamos el **Bearer** que viene dentro por ningún **string**, ejecutamos el log para revisar que va limpio el token:

```
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlbgI90eXBIIjojYWNjZXNzIiwidXNlc19pZCI6IjY1MTcyYWFiNmViZGZiNzc1MjNhMDhkNCIsImhdCI6MTY5NjI2MTU4MjY0NywiZXhwIjoxNjk2MjcyMzgyNjQ3fQ.33UU8sa18C2wKnJ9Cnx0PSlod0yjKsxUN0uzF5Z3xfg
```

Ahora que validamos que esta correcto nuestro token, modificamos de la siguiente manera colocando un try catch:



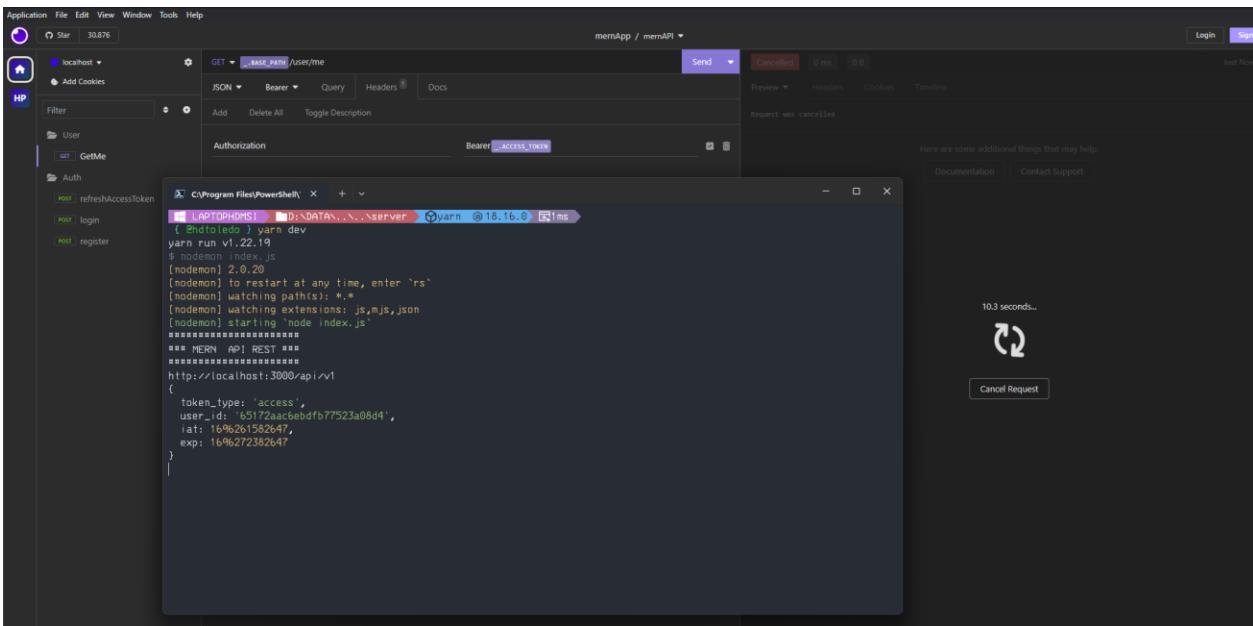
```

middlewares > js authenticated.js > asureAuth
1 const jwt = require("../utils/jwt")
2
3 function asureAuth(req, res, next) {
4   if(!req.headers.authorization){
5     return res.status(403).send({msg: "La peticion no tiene la cabecera de autenticacion"})
6   }
7
8   const token = req.headers.authorization.replace("Bearer", "")
9
10  try {
11    const payload = jwt.decoded(token)
12    console.log(payload);
13  } catch (error) {
14    return res.status(400).send({msg: "Token Invalido"})
15  }
16}
17
18 module.exports = {
19   asureAuth
20 }

```

De esta manera cambiamos en jwt nuestro archivo jwt, modificamos asureAuth y agregamos el return, y en el try catch cargamos nuestro payload con la información del token si ocurre un error nos mostrara el error del token invalido.

Vamos a probar de nuevo enviando la petición a través de insomnia:



Y observamos que ya tenemos los datos que necesitamos, ahora realizaremos la extracción de iat y la fecha de la siguiente manera:



```

7
8     const token = req.headers.authorization.replace("Bearer", "")
9
10    try {
11        const payload = jwt.decoded(token)
12
13        const { exp } = payload
14        const currentDate = new Date().getTime()
15
16        console.log(exp)
17        console.log(currentDate)
18
19    } catch (error) {
20        return res.status(400).send({msg: "Token Invalido"})
21    }
22}
23

```

Lo que hicimos fue hacer una extracción de la data que necesitamos para validar, mediante exp y currentDate y por ultimo los revisamos en consola, ejecutamos de nuevo insomnia y obtenemos lo siguiente:

```

[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
1696272382647
1696872653294
|

```

Ahora comprobaremos que la fecha de expiración no sea menor o igual a la fecha actual, si la fecha de exp es anterior a la fecha actual significa que ha caducado el token y si es superior que no ha caducado, para ello vamos a realizar lo siguiente:

```

10    try {
11        const payload = jwt.decoded(token)
12
13        const { exp } = payload
14        const currentDate = new Date().getTime()
15
16        console.log(exp)
17        console.log(currentDate)
18
19        if(exp <= currentDate){
20            return res.status(400).send({msg: "El token ha expirado"})
21        }
22
23        req.user = payload
24        next()
25
26    } catch (error) {
27        return res.status(400).send({msg: "Token Invalido"})
28    }
29}
30

```



@hdtoledo

Ahora comprobamos mediante insomnia que todo vaya bien, si ejecutamos nos sale lo siguiente:

The screenshot shows the Insomnia REST client interface. A GET request is made to `$_BASE_PATH/_user/me`. The Authorization header is set to `Bearer $_ACCESS_TOKEN`. The response status is **200 OK**, with a response time of 13 ms and 12 B. The response body is a JSON object:

```
1 - {  
2 -   "msg": "ok"  
3 - }
```

Ya nos deja hacer la petición correctamente y si desmarcamos de nuevo y enviamos nos saldrá lo siguiente:

The screenshot shows the Insomnia REST client interface. A GET request is made to `$_BASE_PATH/_user/me`. The Authorization header is set to `Bearer $_ACCESS_TOKEN`. The response status is **400 Bad Request**, with a response time of 1.58 ms and 24 B. The response body is a JSON object:

```
1 - {  
2 -   "msg": "Token Invalido"  
3 - }
```

De esta manera lo podemos emplear en cualquier endpoint.

Obtener los datos del Usuario Logueado

Ahora vamos a terminar la función del `getMe` en donde devolveremos los datos de usuario y el token, así que realizaremos los siguiente en **nuestro user.controller.js**

```
controllers > js user.controller.js >  <unknown>
1  async function getMe(req, res) {
2    console.log(req.user)
3    res.status(200).send({msg: "Ok"})
4  }
5
6  module.exports = {
7     getMe,
8  }
```

De esta manera comprobamos que este devolviendo el objeto con los datos del usuario:

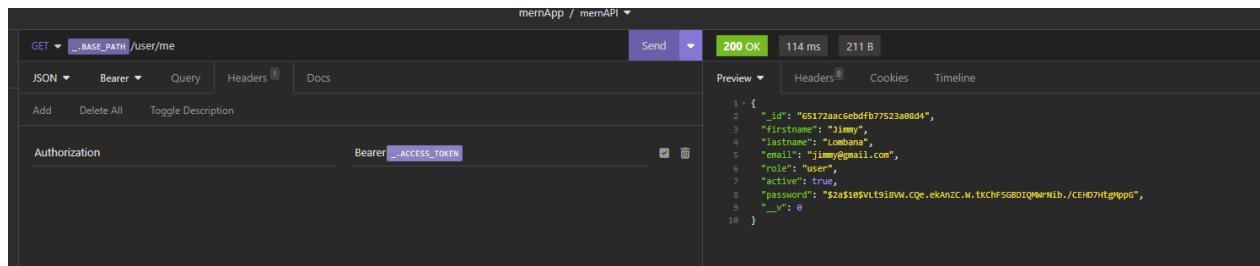
```
Lnodemini$ starting node index.js
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
{
  token_type: 'access',
  user_id: '65172aac6ebdfb77523a08d4',
  iat: 1696873680840,
  exp: 1696884480839
}
```



Ahora que comprobamos que nos esta devolviendo los datos procedemos a dejarlo de la siguiente manera:

```
controllers > js user.controller.js > <unknown>
1  const User = require("../models/user.model")
2
3  async function getMe(req, res) {
4
5      const { user_id } = req.user
6      const response = await User.findById(user_id)
7
8      if(!response){
9          res.status(400).send({msg: "No se ha encontrado el usuario"})
10     } else {
11         res.status(200).send(response)
12     }
13 }
14
15 module.exports = {
16     getMe,
17 }
```

Ahora verificamos de nuevo en nuestro insomnia al enviar la petición:



The screenshot shows the insomnia API client interface. A GET request is made to `/user/me`. The response is **200 OK** with a response time of 114 ms and a size of 211 B. The response body is a JSON object:

```
1 + {
2   "_id": "65172aac6ebdfb77523a08d4",
3   "firstname": "Jimmy",
4   "lastname": "Lombana",
5   "email": "jimmy@gmail.com",
6   "role": "user",
7   "active": true,
8   "password": "$2a$10$VLT9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMrNib./CEHD7HtgMppG",
9   "__v": 0
10 }
```

Y podemos observar que nos esta devolviendo los datos del usuario, para verificar de nuevo podemos probar con otro usuario de los registrados recordemos que tenemos en nuestra DB más:



The screenshot shows the insomnia API client interface displaying the results of a query. The results are labeled "QUERY RESULTS: 1-2 OF 2". There are two documents shown:

```
_id: ObjectId('65172ace26ebdfb77523a08d8')
firstname: "Jimmy"
lastname: "Lombana"
email: "jimmy@gmail.com"
role: "user"
active: true
password: "$2a$10$VLT9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMrNib./CEHD7HtgMppG"
__v: 0

_id: ObjectId('65172ce26ebdfb77523a08d8')
firstname: "Cristian"
lastname: "Lobaton"
email: "elipinor@gmail.com"
role: "user"
active: false
password: "$2a$10$FVOocCGleAAvdcm8m8PK9.PNxvsfstpkL3nIkaQ8J6vpjOhjFIIW"
__v: 0
```

Y para ello los enviamos a través de insomnia:



```

POST _/BASE_PATH/auth/login
Send 401 Unauthorized 187 ms 43 B
JSON Auth Query Headers Docs
1 + {
2   "email": "elpintor@gmail.com",
3   "password": "123456"
4 }
Preview Headers Cookies Timeline
1 + {
2   "msg": "Usuario no autorizado o no activo"
3 }

```

Y recordemos que el usuario que tengo no esta activo, así que lo voy a activar en mongo y vuelvo a enviar la petición:

```

_id: ObjectId('65172ce26ebdfb77523a08d8')
firstname: "Cristian"
lastname: "Lobaton"
email: "elpintor@gmail.com"
role: "user"
active: true
password: "$2a$10$FV0ocCGlEAvgdcWmm8PK9.PNvxfstpkL3nIkaQ8J6vpj0hjFIIW"
__v: 0

```

```

POST _/BASE_PATH/auth/login
Send 200 OK 204 ms 457 B Just Now
JSON Auth Query Headers Docs
1 + {
2   "email": "elpintor@gmail.com",
3   "password": "123456"
4 }
Preview Headers Cookies Timeline
1 + {
2   "access": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvXCV9eyJ0b2tlbl90eXAiLCJpZCI6JyMTCzyUHMiIzGzinCmJNmNmK0isInIiLCJtZXgtYSGjcywngWtLwDwpxjzokmndgj0lyQ.57wigtzounK0CzJgshqJuAueigPevon4LcsgQa39",
3   "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvXCV9eyJ0b2tlbl90eXAiLCJpZCI6JyMTCzyUHMiIzGzinCmJNmNmK0isInIiLCJtZXgtYSGjcywngWtLwDwpxjzokmndgj0lyQ.57wigtzounK0CzJgshqJuAueigPevon4LcsgQa39",
4   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpvXCV9eyJ0b2tlbl90eXAiLCJpZCI6JyMTCzyUHMiIzGzinCmJNmNmK0isInIiLCJtZXgtYSGjcywngWtLwDwpxjzokmndgj0lyQ.57wigtzounK0CzJgshqJuAueigPevon4LcsgQa39"
}

```

Aquí ya me devuelve los tokens así que vamos a tomar de nuevo el token y lo ingresamos en insomnia con una nueva autorización:

```

GET _/BASE_PATH/user/me
Send 200 OK 107 ms 217 B
JSON Bearer Query Headers Docs
Add Delete All Toggle Description
Authorization Bearer <access_token>
Authorization BearereyJhbGciOiJIUzI1NiIsInR5cCI6IkpvXCV9eyJ0b2tlbl90eXAi
Preview Headers Cookies Timeline
1 + {
2   "_id": "65172ce26ebdfb77523a08d8",
3   "firstname": "Cristian",
4   "lastname": "Lobaton",
5   "email": "elpintor@gmail.com",
6   "role": "user",
7   "active": true,
8   "password": "$2a$10$FV0ocCGlEAvgdcWmm8PK9.PNvxfstpkL3nIkaQ8J6vpj0hjFIIW",
9   "__v": 0
10 }

```

Ahora tengo los datos de este usuario, y si comprobamos de nuevo con el otro usuario:

```

GET _/BASE_PATH/user/me
Send 200 OK 105 ms 211 B
JSON Bearer Query Headers Docs
Add Delete All Toggle Description
Authorization Bearer <access_token>
Authorization BearereyJhbGciOiJIUzI1NiIsInR5cCI6IkpvXCV9eyJ0b2tlbl90eXAi
Preview Headers Cookies Timeline
1 + {
2   "_id": "65172ac6ebdfb77523a08d4",
3   "firstname": "Jimmy",
4   "lastname": "Lobatana",
5   "email": "jimmy@gmail.com",
6   "role": "user",
7   "active": true,
8   "password": "$2a$10$VLT9iBwCQEekAnzC.W.tkChF5GBDIQMrh1b./CEH07HtgHpp",
9   "__v": 0
10 }

```

Me devuelve los datos de este usuario, de esta manera ya hemos validado que los tokens nos funcionen y validen la información de nuestro usuario.



OBTENER TODOS LOS USUARIOS

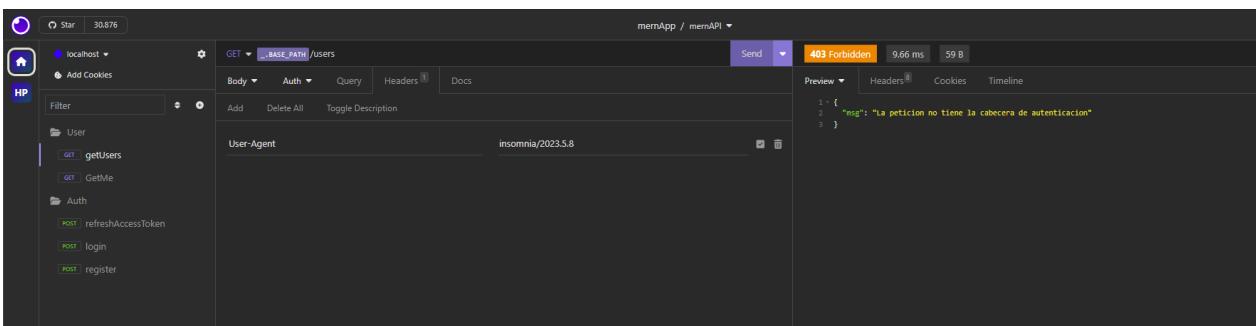
Vamos a crear un endpoint que nos devuelva todos los usuarios, con las condiciones de que nos muestre los activos y los no activos, para ello vamos a nuestro **user.controller.js**

```
controllers > ↵ user.controller.js > ↵ <unknown>
1  const User = require("../models/user.model")
2
3  async function getMe(req, res) {
4
5      const { user_id } = req.user
6      const response = await User.findById(user_id)
7
8      if(!response){
9          res.status(400).send({msg: "No se ha encontrado el usuario"})
10     } else {
11         res.status(200).send(response)
12     }
13 }
14
15 async function getUsers(req, res){
16     res.status(200).send({msg: "Ok"})
17 }
18
19 module.exports = [
20     getMe,
21     getUsers,
22 ]
```

Acá vamos primero a realizar una validación de nuestra ruta, para ello creamos **getUsers** y lo exportamos, y nos dirigimos a nuestro **user.router.js** y vamos a crear la ruta:

```
router > ↵ user.router.js > ...
1  const express = require("express")
2  const UserController = require("../controllers/user.controller")
3  const md_auth = require("../middlewares/authenticated")
4
5  const api = express.Router()
6
7  api.get("/user/me", [md_auth.asureAuth], UserController.getMe)
8  api.get(["/users", [md_auth.asureAuth], UserController.getUsers])
9
10 module.exports = api
```

Ahora nos vamos a nuestro insomnia y vamos a crear una nueva petición que se llame **getUsers**:



Al momento de darle a la solicitud me dice que no tiene la cabecera de autenticación, quiere decir que está funcionando la validación de nuestro token, para ello recordemos que vamos a darle en headers y cargamos la información para la autorización:



De esta manera ya me devuelve el mensaje ok para indicar que la ruta esta validada, ahora vamos a aplicar la lógica de la función, lo primero es obtener el query de la función:

```
controllers > js user.controller.js >  <unknown>
14
15  async function getUsers(req, res){
16
17    const { active } = req.query
18    console.log("Active → ", active)
19
20    res.status(200).send({msg: "Ok"})
21
22  }
23
```

De esta manera vamos a obtener nuestro query en consola de los activos, ahora vamos a insomnia ejecutamos la petición y para poder verificarlo revisamos la consola:

```
[nodemon] 1.3.7 starting 'node index.js'
[nodemon] to quit, press c
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
Active -> undefined
|
```

De esta manera nos muestra que no esta definido si son activos o no activos, para que nos muestre debemos indicarle a través de insomnia de la siguiente manera:

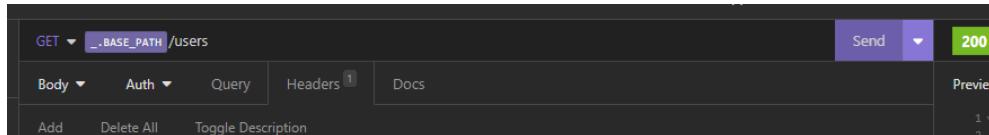
Y en consola nos muestra:

```
#####
http://localhost:3000/api/v1
Active -> undefined
Active -> true
|
```

Y si colocamos false:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
Active -> undefined
Active -> true
Active -> false
|
```

Por el momento lo dejaremos así de esta manera:



Para poder manejarlos todos, así que ahora vamos a nuestro **user.controller.js**:

```
14
15  async function getUsers(req, res){
16
17      const { active } = req.query
18
19      let response = null
20
21      if (active === undefined){
22          response = await User.find()
23      } else {
24          response = await User.find({ active })
25      }
26
27      console.log(response);
28      res.status(200).send({msg: "Ok"})
29  }
30
31  module.exports = {
32      getMe,
33      getUsers,
34  }
```

Le pasamos en un **let response**, y mediante una condición le indicamos si es igual a **undefined**, y le decimos que nos devuelva los datos de estos, por medio de consola mostramos **response**:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
[
  {
    _id: new ObjectId("65172aac6ebdfb77523a08d4"),
    firstname: 'Jimmy',
    lastname: 'Lombana',
    email: 'jimmy@gmail.com',
    role: 'user',
    active: true,
    password: '$2a$10$VLt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMUrNjb./CEHD7HtgMppG',
    __v: 0
  },
  {
    _id: new ObjectId("65172ce26ebdfb77523a08d8"),
    firstname: 'Cristian',
    lastname: 'Lobaton',
    email: 'elpintor@gmail.com',
    role: 'user',
    active: true,
    password: '$2a$10$FVOocCGIEAAvdclmm8PK9.PNxvsfstpkL3UnIkaQ8J6vpj0h.jFIIW',
    __v: 0
  }
]
```

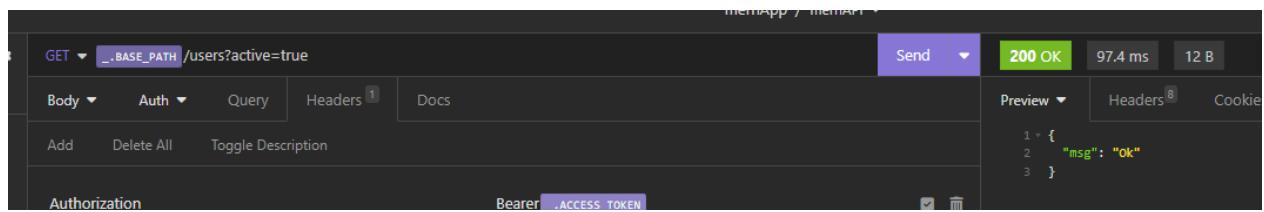
Y acá podemos observar nuestros datos de usuarios, todos sin excepción, vamos a modificar a uno de los usuarios y lo dejaremos en estado active false:



```
QUERY RESULTS: 1-2 OF 2

_id: ObjectId('65172aac6ebdfb77523a08d4')
firstname: "Jimmy"
lastname: "Lombana"
email: "jimmy@gmail.com"
role: "user"
active: true
password: "$2a$10$Vlt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG"
__v: 0
```

Y volvemos a ejecutar cambiando la petición de insomnia por true:



memApp / memapi

GET `BASE_PATH` /users?active=true

Send **200 OK** 97.4 ms 12 B

Body Auth Query Headers 1 Docs

Add Delete All Toggle Description

Authorization Bearer `ACCESS_TOKEN`

Preview Headers Cookies

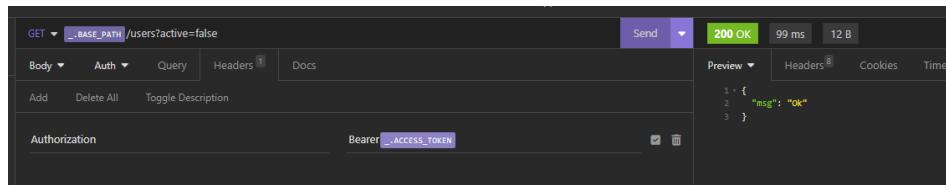
```
1: {
2:   "msg": "ok"
3: }
```

Y en consola nos mostrara:



```
http://localhost:3000/api/v1
[
  {
    _id: new ObjectId("65172ce26ebdfb77523a08d8"),
    firstname: 'Cristian',
    lastname: 'Lobaton',
    email: 'elpinfor@gmail.com',
    role: 'user',
    active: true,
    password: '$2a$10$FVDocCGLEAvdcUmm8PK9.PNxvsfstpkL3UnIkaQ8J6vp.j0hJFIU',
    __v: 0
  }
]
```

Y si la dejamos en false:



GET `BASE_PATH` /users?active=false

Send **200 OK** 99 ms 12 B

Body Auth Query Headers 1 Docs

Add Delete All Toggle Description

Authorization Bearer `ACCESS_TOKEN`

Preview Headers Cookies Timeline

```
1: {
2:   "msg": "ok"
3: }
```

Nos mostrara lo siguiente:



```
[
  {
    _id: new ObjectId("65172aac6ebdfb77523a08d4"),
    firstname: 'Jimmy',
    lastname: 'Lombana',
    email: 'jimmy@gmail.com',
    role: 'user',
    active: false,
    password: '$2a$10$Vlt9i8VW.CQe.ekAnZC.W.tKChFSGBDIQMWrNib./CEHD7HtgMppG',
    __v: 0
  }
]
```

De esta manera ya tenemos el control sobre nuestra data, por último, eliminamos el log con el que mostramos la información y modificamos nuestro status 200 con response:



```

controllers > js user.controller.js > ...
14
15     async function getUsers(req, res){
16
17         const { active } = req.query
18
19         let response = null
20
21         if (active === undefined){
22             response = await User.find()
23         } else {
24             response = await User.find({ active })
25         }
26
27         res.status(200).send(response)
28
29     }
30
31     module.exports = {
32         getMe,
33         getUsers,
34     }

```

De esta manera nos devolverá en nuestro insomnia la data:

```

GET <BASE_PATH> /users
Send ▾ 200 OK | 114 ms | 432 B
Body ▾ Auth ▾ Query Headers Docs
Add Delete All Toggle Description
Authorization: Bearer 
Preview ▾ Headers Cookies Timeline
1: [
2:   {
3:     "_id": "65172aaac6ebdfb77523a08d4",
4:     "firstname": "Jimmy",
5:     "lastname": "Tombara",
6:     "email": "jimmy@gmail.com",
7:     "role": "user",
8:     "active": false,
9:     "password": "$2a$10$V0ocCGlEAwdcmMBPK9.PNxvsfstpkL3UnIkaQ8J6vpjOhjF1IM",
10:    "__v": 0
11  },
12  {
13    "_id": "65172ce56ebdfb77523a08d8",
14    "firstname": "Cristian",
15    "lastname": "Lobaton",
16    "email": "elpintor@gmail.com",
17    "role": "user",
18    "active": true,
19    "password": "$2a$10$V0ocCGlEAwdcmMBPK9.PNxvsfstpkL3UnIkaQ8J6vpjOhjF1IM",
20    "__v": 0
21  }
22 ]

```

CREANDO USUARIOS

Ahora vamos a crear nuestros usuarios desde el panel de administrador para ello vamos a crear la función **createUser** dentro de **user.controller.js**:

```

controllers > js user.controller.js >  <unknown>
30
31     async function createUser(req, res){
32         res.status(200).send({msg: "Funciona!"})
33     }
34
35     module.exports = {
36         getMe,
37         getUsers,
38          createUser,
39     }

```

Acá dejamos un mensaje de estatus 200 para verificar que funciona, ahora nos vamos a nuestras rutas **user.router.js** y allí vamos a crear nuestro endpoint:



```

router > user.router.js > ...
1  const express = require("express")
2  const UserController = require("../controllers/user.controller")
3  const md_auth = require("../middlewares/authenticated")
4
5  const api = express.Router()
6
7  api.get("/user/me", [md_auth.asureAuth], UserController.getMe)
8  api.get("/users", [md_auth.asureAuth], UserController.getUsers)
9  api.post("/user", [md_auth.asureAuth], UserController.createUser)
10 |
11 module.exports = api

```

Ahora que lo creamos vamos a insomnia y vamos a crear una nueva petición tipo **POST** que será **createUser** a través de **/user**

The screenshot shows the Insomnia interface. On the left, there's a sidebar with a tree view of API endpoints under 'User' and 'Auth'. In the main area, a POST request is being made to `./BASE_PATH/user`. The status bar at the top right shows **403 Forbidden**, **17.3 ms**, and **59 B**. The preview tab shows the response body:

```
1: { 2: "msg": "La petición no tiene la cabecera de autenticación" 3: }
```

.

Cuando la ejecutamos nos dice que no tenemos la cabecera de **auth**, para ello debemos agregar la validación de la petición y colocamos nuestro token:

The screenshot shows the same Insomnia interface after adding an Authorization header with the value `Bearer ACCESS_TOKEN`. The status bar now shows **200 OK**, **12.3 ms**, and **19 B**. The preview tab shows the response body:

```
1: { 2: "msg": "Funcional" 3: }
```

.

De esta manera ya nos funciona, ahora realizaremos un log de nuestro `req.body`:

```

controllers > user.controller.js > ...
30
31  async function createUser(req, res){
32    console.log(req.body)
33    res.status(200).send({msg: "Funciona!"})
34  }
35
36  module.exports = {
37    getMe,
38    getUsers,
39    createUser,
40  }

```

Vamos a insomnia en donde realizaremos de nuevo la petición y en consola obtendremos lo siguiente:

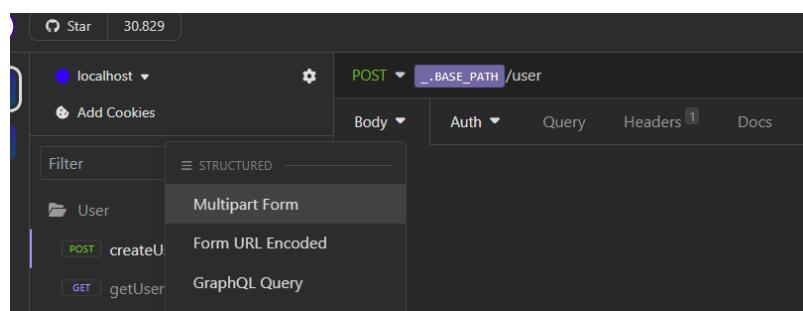


```
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
{}
```

Nos llega un objeto vacío, pero acá no debería llegar vacío, nos deben pasar los datos de nuestro usuario, recordemos que los datos de nuestro modelo son los siguientes:

```
3 const UserSchema = mongoose.Schema({
4   firstname: {
5     type: String,
6     required: true,
7     trim: true
8   },
9   lastname: {
10    type: String,
11    required: true,
12    trim: true
13 },
14   email: {
15     type: String,
16     unique: true,
17     required: true
18 },
19   password: {
20     type: String,
21     required: true
22 },
23   role: String,
24   active: Boolean,
25   avatar: String,
26 })
```

Y estos datos son los que nos van a llegar a través de esta petición, y en especial recordemos que no todos son strings, tambien tendremos uno que será un fichero de imagen como lo es avatar, en insomnia este tipo de datos se enviaran a través de multipart, para ello vamos a configurarlo:



Y dejamos los datos que vamos a enviar a través de multipart de la siguiente manera:



Un multipart es un tipo de formulario que se utiliza para enviar datos binarios, como archivos, a un servidor web. Los datos binarios se dividen en partes más pequeñas, cada una con un encabezado que especifica el tipo de contenido y el nombre del campo, al momento de pasar el valor de nuestro avatar podemos seleccionar como tipo archivo:

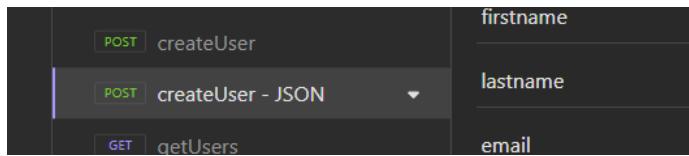
Por el momento lo dejaremos sin cargar un archivo, procedemos a enviar la solicitud y obtendremos que nos devuelve nuevamente un objeto vacío:

```
# This file is part of mern-project.
# MERN API REST
#
# http://localhost:3000/api/v1
{ }
{ }
```

Si lo enviáramos a través de un JSON esto sería diferente, vamos a hacer la prueba de la siguiente manera, vamos a duplicar nuestra solicitud primero:

Y le colocaremos createUser – JSON





Y ahora en lugar de enviarlo como multipart lo enviaremos como JSON:

Y en consola obtendríamos lo siguiente:

```
#####
http://localhost:3000/api/v1
{}
{}
{ User: 'TestUser' }
```

Nos llega perfectamente los datos, pero cuando utilizamos **multipart** cambia el modo en el cual yo envío los datos y en especial si vamos a manejar imágenes, acá debemos utilizar un middleware que nos permita realizarlo y para ello vamos a utilizar uno que se llama **connect-multiparty**:

Y para ello procedemos a instalarlo en nuestro terminal ejecutando **yarn add connect-multiparty**

```
LAPTOPHDMI ~ D:\DATA\...\server yarn 18.16.0 2ms
{ hdtoledo } yarn add connect-multiparty
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
```

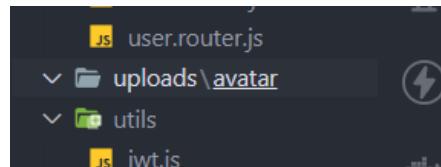
Vamos a levantar de nuevo nuestro servidor y procedemos a implementarlo en **user.router.js**

```
router > js user.router.js > multiparty
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const UserController = require("../controllers/user.controller")
4 const md_auth = require("../middlewares/authenticated")
```

Y ahora vamos a realizar lo siguiente allí mismo:

```
router > js user.router.js > md_upload
  5
  6  const md_upload = multiparty({ uploadDir: "./uploads/avatar" })
  7  const api = express.Router()
  8
```

Acá estamos definiendo nuestra const del middleware y utilizamos multiparty para indicarle donde queremos subir el avatar de nuestro usuario, recordemos que la ubicación debe estar creada así que vamos a crear la carpeta avatar dentro de **uploads**:



Y por último en nuestra ruta le pasamos el middleware **md_upload** que acabamos de implementar:

```
router > js user.router.js > ...
  8
  9  api.get("/user/me", [md_auth.asureAuth], UserController.getMe)
 10 api.get("/users", [md_auth.asureAuth], UserController.getUsers)
 11 api.post("/user", [md_auth.asureAuth, md_upload], UserController.createUser)
 12
 13 module.exports = api
```

Acá nosotros podemos pasarle cualquier cantidad de middlewares a nuestra ruta y no habrá ningún inconveniente, ahora probamos de nuevo ejecutando la petición a través del multipart en insomnia y en consola observaremos:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
{
  firstname: 'name01',
  lastname: 'lastname02',
  email: 'test01@gmail.com',
  password: '123456',
  role: 'admin',
  avatar: ''
}
```

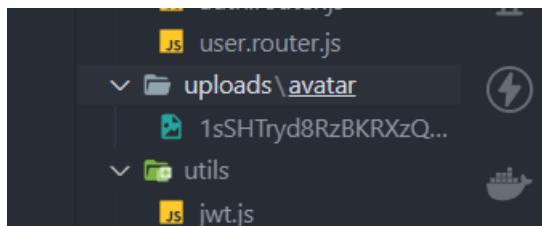
De esta manera hemos logrado pasar los datos a través del middleware y procesar el multipart, ahora probaremos con una imagen para revisar que funciona el cargue a través de insomnia:



Una vez cargada la imagen y enviada la solicitud nos fijamos en consola que nos saldrá lo siguiente:

```
[nodemon] starting `node index.js`
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
{
  firstname: 'name01',
  lastname: 'lastname02',
  email: 'test01@gmail.com',
  password: '123456',
  role: 'admin'
}
```

Se pasan los datos perfectamente y si revisamos nuestra carpeta de **uploads/avatar** observaremos que ya tenemos nuestra imagen cargada:



Nuestra imagen nos indica que fue procesada correctamente y se le cambia el nombre por seguridad y la almacenamos correctamente en nuestra carpeta, ahora lo que sigue es configurar nuestra función **createUser**, para ello vamos a realizar lo siguiente dentro de **user.controller.js**, lo primero será observar que el formato de nuestra contraseña nos llega en string y para ello debemos dejarla encriptada así que vamos a importar **bcrypt**

```
controllers > js user.controller.js > bcrypt
1 const bcrypt = require("bcryptjs")
2 const User = require("../models/user.model")
3
```

Y ahora procedemos a dejar nuestra función de la siguiente manera:



```

controllers > user.controller.js > createUser
31
32  async function createUser(req, res){
33
34      const { password } = req.body
35      const salt = bcrypt.genSaltSync(10)
36      const hashPassword = bcrypt.hashSync(password, salt)
37
38      console.log(hashPassword)
39
40      res.status(200).send({msg: "Funciona!"})
41
42

```

Dentro de la función pasamos nuestra contraseña, aplicamos bcrypt para encriptarla y utilizamos el log para revisar que este pasando encriptada, en insomnia ejecutamos nuevamente la solicitud y deshabilitamos la imagen para que no nos esté guardando más esta:

Y en nuestro log nos muestra lo siguiente:

```

[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
$2a$10$4AZ7gfrH/xYVs6sddVvuJeB7L.W2myZqcW.y13Bj0K7gyGZPRBSCm
|

```

Ya tenemos encriptada nuestra contraseña, ahora seguiremos con la funcionalidad:

```

controllers > user.controller.js > createUser
32  async function createUser(req, res){
33
34      const { password } = req.body
35      const salt = bcrypt.genSaltSync(10)
36      const hashPassword = bcrypt.hashSync(password, salt)
37
38      const user = new User({ ...req.body, active: false, password: hashPassword })
39      console.log(user)
40
41      res.status(200).send({msg: "Funciona!"})
42
43

```

En la siguiente linea **const user = new User({ ...req.body, active: false, password: hashPassword })** crea una nueva instancia del modelo User, estableciendo la propiedad active en false y la propiedad password en la contraseña encriptada.



@hdtoledo

El operador de propagación `...req.body` copia todas las propiedades del objeto `req.body` en el nuevo objeto `user`. Esta es una forma conveniente de poblar el objeto `User` con los datos que se enviaron en el cuerpo de la solicitud.

La función `hashPassword` es responsable de encriptar la contraseña antes de almacenarla en la base de datos. Esta es una medida de seguridad importante para evitar que los atacantes accedan a las contraseñas de los usuarios en caso de una violación de datos.

Y ahora volvemos a enviar la solicitud de nuevo a través de insomnia y al verla reflejada en consola obtendremos los datos con nuestra contraseña ya encriptada:

```
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
{
  firstname: 'name01',
  lastname: 'lastname02',
  email: 'test01@gmail.com',
  password: '$2a$10$14.m8DlexXoTdFX/e4VpJ.txTnbfh/vFarSKqdv3H1rbQ0aW/oMtu',
  role: 'admin',
  active: false,
  _id: new ObjectId("6526e107c843bbe21cb8490e")
}
```

Vamos a realizar una pequeña modificación en la estructura del código y agregar nuestro avatar:

```
controllers > user.controller.js > create
32  async function createUser(req, res){
33
34    const { password } = req.body
35    const user = new User({ ...req.body, active: false })
36
37    const salt = bcrypt.genSaltSync(10)
38    const hashPassword = bcrypt.hashSync(password, salt)
39
40    user.password = hashPassword
41
42    console.log(user)
43
44    res.status(200).send({msg: "Funciona!"})
45 }
```

La modificación dentro de este código fue simplemente subir nuestra const `user`, eliminamos `password` de allí y la dejamos mas abajo, esto no altera nuestra funcionalidad si revisamos en consola saldrá igual:

```
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
{
  firstname: 'name01',
  lastname: 'lastname02',
  email: 'test01@gmail.com',
  password: '$2a$10$tR0AvFNaqPz.YoqosFRXfe8cT.qH2y241QTEuXEq2jeLqDC5SKGDC',
  role: 'admin',
  active: false,
  _id: new ObjectId("6526e2e376d9c330cf56f50e")
```



@hdtoledo

Ahora procedemos a dejar lista nuestra imagen, recordemos que el usuario no esta obligado a subir un avatar, vamos a revisar como manipular la data del avatar, mediante un logo realizamos lo siguiente:

```
39     user.password = hashPassword
40
41     console.log(req.files.avatar)
42
43
44     res.status(200).send({msg: "Funciona!"})
45 }
```

En consola obtendremos esto:

```
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
{
  fieldName: 'avatar',
  originalFilename: 'logo512.png',
  path: 'uploads\avatar\ge56QvkLYaBgksUVDgKuTiCO.png',
  headers: {
    'content-disposition': 'form-data; name="avatar"; filename="logo512.png"',
    'content-type': 'image/png'
  },
  size: 27548,
  name: 'logo512.png',
  type: 'image/png'
}
```

Todo esto es la data que debemos procesar de nuestra imagen, por ahora dejaremos un pendiente allí ya que lo realizaremos de otra manera asi que vamos a dejarlo de la siguiente manera:

```
39     user.password = hashPassword
40
41
42     if(req.files.avatar){
43       //TODO:
44       console.log("Procesar avatar")
45     }
46
47     res.status(200).send({msg: "Funciona!"})
48 }
```

Y vamos a dejar el guardado de nuestros datos de la siguiente manera:

```
controllers > user.controller.js > ...
41
42   if(req.files.avatar){
43     //TODO:
44     console.log("Procesar avatar")
45   }
46
47   user.save((error, userStored) => {
48     if (error){
49       res.status(400).send({msg: "Error al crear el usuario !"})
50     } else {
51       res.status(201).send(userStored)
52     }
53   })
54
55   res.status(200).send({msg: "Funciona!"})
56 }
57 }
```

Vamos a ejecutar de nuevo en insomnia la petición y al realizarla observaremos:



```

POST _/BASE_PATH/_User
Send 201 Created 231 ms 218 B Just Now
Preview Headers Cookies Timeline
1 * {
2   "firstname": "name01",
3   "lastname": "lastname02",
4   "email": "test01@gmail.com",
5   "password": "$2a$05$057r3/fUpvjlIcI4Xjk.P1l0u.SUr7lvQss9Nr9EBI3EdkZzciW4W",
6   "role": "admin",
7   "active": false,
8   "_id": "6526e706964b3efffd9c6f9c",
9   "__v": 0
10 }

```

De esta manera observamos que nos devuelve los datos en mensaje satisfactorio, y al revisar en **mongodb** vamos a ver como se nos creó un tercer usuario:

```

test.users
STORAGE SIZE: 36KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 3 INDEXES TOTAL SIZE: 72KB
Find Indexes Schema Anti-Patterns Aggregation Search Indexes
INSERT DOCUMENT
Filter Type a query: { field: 'value' }
Reset Apply More Options ▾
QUERY RESULTS: 1-3 OF 3
_id: ObjectId('65172aac6ebdfb77523a88d4')
firstname: "Immy"
lastname: "Lebanan"
email: "immy@gmail.com"
role: "user"
active: false
password: "$2a$10$VLTs918W.W.CQo.ekAnZC.W.tKChFSGBDIQMrN1b./CEHDYItgMppG"
__v: 0

_id: ObjectId('65172ce26ebdfb77523a88d8')
firstname: "Cristian"
lastname: "Lobaton"
email: "elspinotor@gmail.com"
role: "user"
active: true
password: "$2a$10$FVOocCG1EAAvdcLmmPK9.PNxvsfstpkL3UlnIkaQ36vpj0hJFIW6"
__v: 0

_id: ObjectId('6536e706964b3efffd9c6f9c')
firstname: "name01"
lastname: "lastname02"
email: "test01@gmail.com"
password: "$2a$10$57r3/fUpvjlIcI4Xjk.P1l0u.SUr7lvQss9Nr9EBI3EdkZzciW4W"
role: "admin"
active: false
__v: 0

```

Y al volver a intentar enviar los mismos datos en insomnia vamos a obtener el error al crear el usuario:

```

POST _/BASE_PATH/_User
Send 400 Bad Request 183 ms 37 B
Preview Headers Cookies Timeline
1 * {
2   "msg": "Error al crear el usuario !"
3 }

```

Y si cambiamos el correo electrónico y volvemos a enviarlo, notaremos que nos crea el usuario, ya que como hemos dejado requerido el correo pues no tendremos errores al momento de crearlos:



```

1 + {
2   "firstname": "Probando",
3   "lastname": "lastname02",
4   "email": "test02@gmail.com",
5   "password": "$2a$10$803POg2epbQf7gxYJPa/..ZNgnx0Q6Ir9G47RusTeLM7DNYZkq6u",
6   "role": "admin",
7   "active": false,
8   "_id": "6526e8bf964b3efffd9c6fa0",
9   "__v": 0
10  }

```

Y en mongodb:

```

{
  "_id": ObjectId("6526e706964b3efffd9c6f9c"),
  "firstname": "name01",
  "lastname": "lastbato",
  "email": "elpintor@gmail.com",
  "password": "$2a$10$FV0ocCG1EAAvdcMm8PK9.PNxvsfstpkL3UnIkaQsJ6vpj0hjFIW",
  "__v": 0
}

{
  "_id": ObjectId("6526e706964b3efffd9c6fa0"),
  "firstname": "name01",
  "lastname": "lastname02",
  "email": "test02@gmail.com",
  "password": "$2a$10$803POg2epbQf7gxYJPa/..ZNgnx0Q6Ir9G47RusTeLM7DNYZkq6u",
  "role": "admin",
  "active": false,
  "__v": 0
}

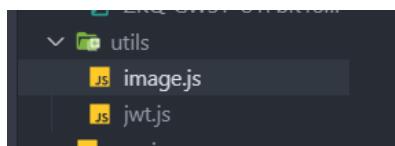
{
  "_id": ObjectId("6526e8bf964b3efffd9c6fa0"),
  "firstname": "Probando",
  "lastname": "lastname02",
  "email": "test02@gmail.com",
  "password": "$2a$10$803POg2epbQf7gxYJPa/..ZNgnx0Q6Ir9G47RusTeLM7DNYZkq6u",
  "role": "admin",
  "active": false,
  "__v": 0
}

```

Así de esta manera ya tenemos el modulo de creación de usuario desde nuestro panel de administrador y este lo conectaremos cuando estemos con el front.

PROCESANDO LA IMAGEN

Ahora vamos a procesar la imagen y debemos definirla y setearla, para ello vamos a crear una función para poderla procesar, para ello nos vamos a **utils** y vamos a crear **image.js**:



Y dentro vamos a colocar lo siguiente:



```
utils > js image.js > ...
1  function getFilePath(file) {
2    const filePath = file.path
3
4    return filePath
5  }
6
7  module.exports = {
8    getFilePath,
9  }
```

En esta función que vamos a empezar a estructurar primero almacenamos la ruta de nuestra imagen y nos mostrara el path, ahora vamos a **user.controller.js** en donde vamos a importarlo:

```
controllers > js user.controller.js > image
1  const bcrypt = require("bcryptjs")
2  const User = require("../models/user.model")
3  const image = require("../utils/image")
4
```

Y ahora en nuestra función va a quedar de la siguiente manera:

```
controllers > js user.controller.js > createUser
41
42    user.password = hashPassword
43
44    if (req.files.avatar){
45      const imagePath = image.getFilePath(req.files.avatar)
46      console.log(imagePath)
47    }
48
49    user.save((error, userStored) => {
```

Y ahora vamos a nuestro insomnia y ejecutamos nuevamente la petición:

```
#####
## MERN API REST ##
#####
http://localhost:3000/api/v1
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
#####
## MERN API REST ##
#####
http://localhost:3000/api/v1
uploads\avatar\xBcdhyttRJqeGTu9Y-w_fXSy.png
```

Justo debajo obtenemos la ruta de la imagen, ahora vamos a limpiar nuestra ruta para almacenarla correctamente en un string, lo primero que vamos a realizar es lo siguiente:

```
utils > js image.js > ...
1  function getFilePath(file) {
2    const filePath = file.path
3    const fileSplit = filePath.split("\\\\")
4
5    return fileSplit
6  }
7
8  module.exports = {
9    getFilePath,
10 }
```

En donde vamos a eliminar nuestro “/” si es en mac o “\\\\” si es en Windows, ahora volvemos a darle en insomnia para verificar y en consola nos saldrá lo siguiente:



@hDTOledo

```
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
[ 'uploads', 'avatar', 'Mm-7kjODyMe1CtRago8XHJw3.png' ]
```

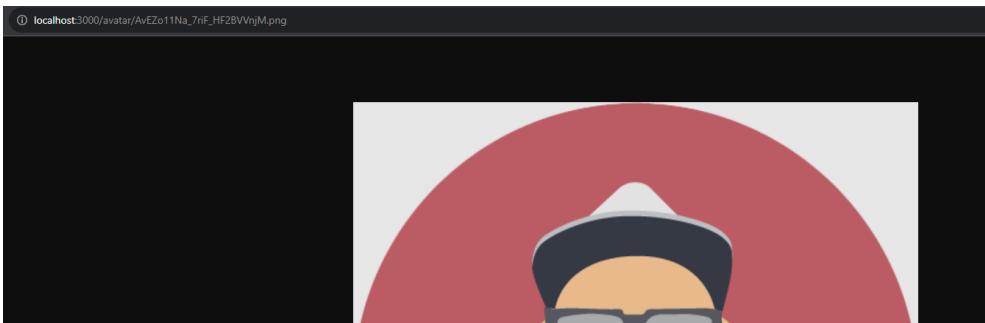
Un array con los datos del path, ahora vamos a armar la ruta correcta:

```
utils > js imagejs > ...
1 function getFilePath(file) {
2   const filePath = file.path
3   const fileSplit = filePath.split("\\\\")
4
5   return `${fileSplit[1]}/${fileSplit[2]}`
6 }
7
8 module.exports = {
9   getFilePath,
10 }
```

En nuestro return utilizamos backticks y colocamos la posición de nuestro array que nos interesa que seria la 1 y 2, volvemos a ejecutar insomnia y en consola obtenemos lo siguiente:

```
[nodemon] starting `node index.js`
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
avatar/AvEZo11Na_7riF_HF2BVn.jM.png
```

Ahora si copiamos esta dirección de enlace y la colocamos en nuestro navegador nos debe devolver como resultado la imagen del usuario:



Ahora lo que nos queda es almacenarlo directamente sobre la data que vamos a enviar de User, para ello hacemos que **user.avatar** sea igual a **imagePath**:

```
controllers > js user.controller.js > ⚡ createUser
...
41     user.password = hashPassword
42
43     if (req.files.avatar){
44       const imagePath = image.getPath(req.files.avatar)
45       user.avatar = imagePath
46
47
48     user.save((error, userStored) => {
```

Vamos a comprobar con un nuevo usuario y hacemos el envío a través de insomnia:

```
1 + {
2   "firstname": "Probando 2",
3   "lastname": "lastname03",
4   "email": "test04@gmail.com",
5   "password": "t$2a$10$Komi9YwJWU9XjntsZ4m20.zOZprr8GvbjPJ2M9Vkg8gSM1xJTTvaa",
6   "role": "admin",
7   "active": false,
8   "_id": "65270059605b8ad666ca60ff",
9   "avatar": "avatar/p8Tj0pyG8782-ULfyJndZRM.png",
10  "_v": 0
11 }
```

ahora si nos fijamos en la data enviada ya tenemos nuestra ruta del avatar.

ACTUALIZANDO EL USUARIO

Ahora vamos a proceder a crear la función para poder actualizar los datos del usuario y para ello vamos a crear dentro de nuestro **user.controller.js** la función **updateUser**

```
57  async function updateUser(req, res) {
58    const { id } = req.params
59    const userData = req.body
60
61    //Password
62    //Avatar
63
64    User.findByIdAndUpdate({ _id: id }, userData, (error) => {
65      if(error) {
66        res.status(400).send({msg: "Error al actualizar el usuario"})
67      } else {
68        res.status(200).send({msg: "Datos de usuario actualizados"})
69      }
70    })
71  }
72
73
74  module.exports = [
75    getMe,
76    getUsers,
77    createUser,
78    updateUser,
79  ]
```

La función comienza por obtener el identificador del usuario a actualizar de los parámetros de la solicitud. Luego, obtiene los datos del usuario del cuerpo de la solicitud. Si es así, la función actualiza los campos correspondientes en el documento del usuario.

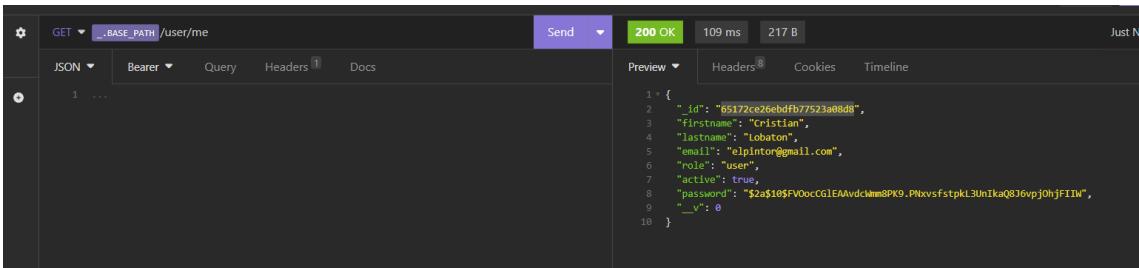
Finalmente, la función utiliza el método **findByIdAndUpdate** de Mongoose para actualizar los datos del usuario en la base de datos. Si la actualización se realiza correctamente, la función envía una respuesta al cliente con el código de estado 200 y el mensaje "Datos de usuario actualizados". Si se produce un error durante la actualización, la función envía una respuesta al cliente con el código de estado 400 y el mensaje "Error al actualizar el usuario".



Ahora vamos a crear el endpoint para nuestra función de actualización, nos vamos a **user.router.js** y vamos a colocar lo siguiente:

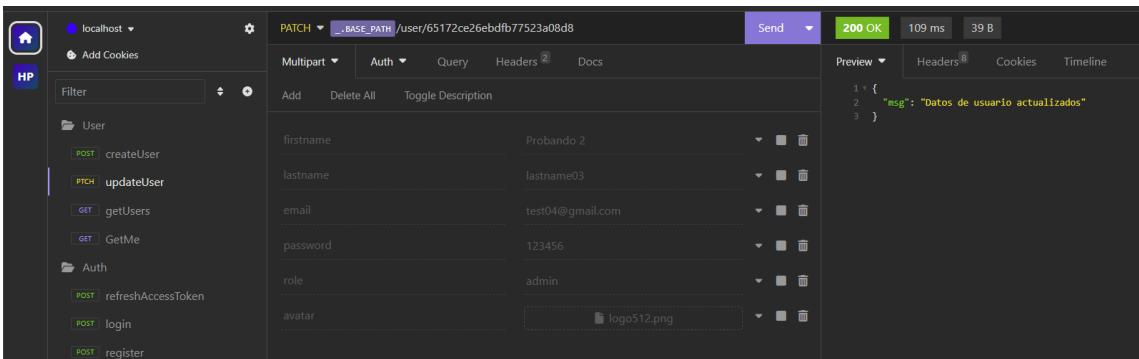
```
router > js user.router.js > ...
5
6  const md_upload = multiparty({ uploadDir: "./uploads/avatar" })
7  const api = express.Router()
8
9  api.get("/user/me", [md_auth.asureAuth], UserController.getMe)
10 api.get("/users", [md_auth.asureAuth], UserController.getUsers)
11 api.post("/user", [md_auth.asureAuth, md_upload], UserController.createUser)
12 api.patch("/user/:id", [md_auth.asureAuth, md_upload], UserController.updateUser)
13
14 module.exports = api
```

En esta ruta colocamos patch para poder realizar la actualización de los datos a través del id, implementamos los middlewares y cargamos la función de updateUser, para probarlo nos vamos a insomnia y vamos getMe para obtener nuestro id de usuario:



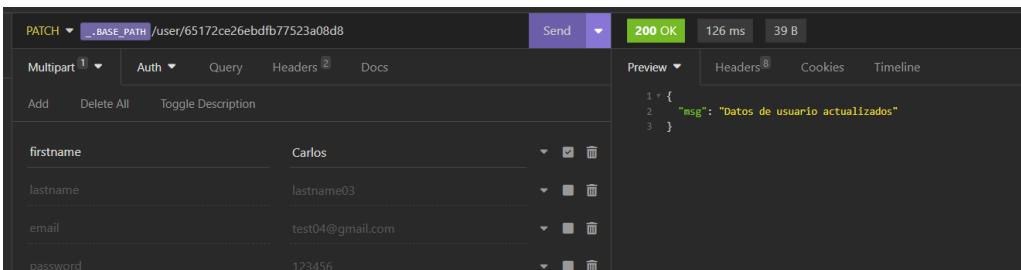
```
1: {
2:   "id": "65172ce26ebdbf77523a08d8",
3:   "firstname": "Cristian",
4:   "lastname": "Lobaton",
5:   "email": "elpinor@gmail.com",
6:   "role": "user",
7:   "active": true,
8:   "password": "$2a$10$FVOocCGlEAAvdclmmSPK9.PNxvsfstpkL3UnIkaQ8J6vpjOjhFIIW",
9:   "_v": 8
10 }
```

Y ahora vamos a crear una nueva petición y probamos el envío, desmarcamos todo solo para probar:

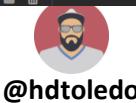


```
1: {
2:   "msg": "Datos de usuario actualizados"
3: }
```

Ahora que ya verificamos que si esta funcionando, vamos a probar a actualizar el nombre de usuario y para ello lo dejamos así:



```
1: {
2:   "msg": "Datos de usuario actualizados"
3: }
```



Para verificarlo podemos hacerlo en insomnia por medio de getMe enviamos la petición y obtenemos que si ha cambiado el nombre:

```

1 < {
2   "_id": "65172ce26ebdfb77523a08d8",
3   "firstname": "Carlos",
4   "lastname": "Lobaton",
5   "email": "elpintor@gmail.com",
6   "role": "user",
7   "active": true,
8   "password": "$2a$10$FVOocCG1EAvdchm8PK9.PNxvsfstpl3UnIkaQ8J6vpjOhjFIIM",
9   "__v": 0
10 }

```

Y si verificamos en mongodb obtenemos que el usuario tambien ha sido actualizado:

```

QUERY RESULTS: 1-6 OF 6

1 < _id: ObjectId('65172aae6ebdfb77523a08d4')
  firstname: "Jimmy"
  lastname: "Lombana"
  email: "jimmy@gmail.com"
  role: "user"
  active: false
  password: "$2a$10$FVOocCG1EAvdchm8PK9.PNxvsfstpl3UnIkaQ8J6vpjOhjFIIM"
  __v: 0

2 < _id: ObjectId('65172ce26ebdfb77523a08d8')
  firstname: "Carlos"
  lastname: "Lobaton"
  email: "elpintor@gmail.com"
  role: "user"
  active: true
  password: "$2a$10$FVOocCG1EAvdchm8PK9.PNxvsfstpl3UnIkaQ8J6vpjOhjFIIM"
  __v: 0

3 < _id: ObjectId('6526a706964b3effd9c6f9c')

```

Ahora vamos a probar a cambiar los demás datos:

```

PATCH < _BASE_PATH /user/65172ce26ebdfb77523a08d8
Send 200 OK 109 ms 39 B
Preview Headers Cookies Timeline
1 < {
2   "msg": "Datos de usuario actualizados"
3 }

```

firstname	Cristian Fernando
lastname	Lobaton
email	test04@gmail.com
password	123456
role	user
avatar	logo512.png

Validamos en getMe:

```

GET < _BASE_PATH /user/me
Send 200 OK 105 ms 226 B
Preview Headers Cookies Timeline
1 <
2   "_id": "65172ce26ebdfb77523a08d8",
3   "firstname": "Cristian Fernando",
4   "lastname": "Lobaton",
5   "email": "elpintor@gmail.com",
6   "role": "user",
7   "active": true,
8   "password": "$2a$10$FVOocCG1EAvdchm8PK9.PNxvsfstpl3UnIkaQ8J6vpjOhjFIIM",
9   "__v": 0
10 }

```

Probamos de nuevo cambiando solo el rol:

PATCH `./BASE_PATH` /user/65172ce26ebdfb77523a08d8

Multipart 1 Auth Query Headers 2 Docs

Add Delete All Toggle Description

firstname: Cristian Fernando
lastname: Lobaton
email: test04@gmail.com
password: 123456
role: admin
avatar: logo512.png

Send 200 OK 100 ms 39 B

Preview Headers 8 Cookies Timeline

```
1: {  
2:   "msg": "Datos de usuario actualizados"  
3: }
```

Y en getMe nos muestra:

GET `./BASE_PATH` /user/me

JSON Bearer Query Headers 1 Docs

1: ...

Send 200 OK 105 ms 227 B

Preview Headers 8 Cookies Timeline

```
1: {  
2:   "_id": "65172ce26ebdfb77523a08d8",  
3:   "firstname": "Cristian Fernando",  
4:   "lastname": "Lobaton",  
5:   "email": "elpintor@gmail.com",  
6:   "role": "admin",  
7:   "active": true,  
8:   "password": "$2a$10$FVOocCG1EAvdchmm8PK9.PNxvsfstpkL3Un1kaQ8JevpjOhjFIIM",  
9:   "__v": 0  
10: }
```

Ahora vamos con la contraseña y el avatar, para ello vamos a hacer el envío de la contraseña por el momento así como esta para verificar que pasa:

PATCH `./BASE_PATH` /user/65172ce26ebdfb77523a08d8

Multipart 1 Auth Query Headers 2 Docs

Add Delete All Toggle Description

firstname: Cristian Fernando
lastname: Lobaton
email: test04@gmail.com
password: 123456
role: admin
avatar: logo512.png

Send 200 OK 107 ms 39 B

Preview Headers 8 Cookies Timeline

```
1: {  
2:   "msg": "Datos de usuario actualizados"  
3: }
```

Vemos que la contraseña pasa sin problema, pero al revisar en **getMe** observamos que la contraseña ya no está encriptada:

Preview Headers 8 Cookies Timeline

```
1: {  
2:   "_id": "65172ce26ebdfb77523a08d8",  
3:   "firstname": "Cristian Fernando",  
4:   "lastname": "Lobaton",  
5:   "email": "elpintor@gmail.com",  
6:   "role": "admin",  
7:   "active": true,  
8:   "password": "123456",  
9:   "__v": 0  
10: }
```



Para ello vamos a realizar lo siguiente dentro de nuestra función **updateUser**:

```
controllers > user.controller.js > updateUser
  ...
57  async function updateUser(req, res) {
58    const { id } = req.params
59    const userData = req.body
60
61    if (userData.password) {
62      const salt = bcrypt.genSaltSync(10)
63      const hashPassword = bcrypt.hashSync(userData.password, salt)
64      userData.password = hashPassword
65    } else {
66      delete userData.password
67    }
68
69
70
71    //Avatar
72
```

Comprobamos si el usuario ha proporcionado una nueva contraseña. Si es así, el código genera una sal y hash, la contraseña con la sal. El hash de la contraseña se almacena en el campo password del objeto userData, y si el usuario no ha proporcionado una nueva contraseña, el código elimina el campo password del objeto userData. Esto se hace porque es importante no almacenar contraseñas sin hash en una base de datos.

Vamos a comprobar de nuevo que pasa con la contraseña:

The screenshot shows a POSTMAN interface. The method is set to PATCH, the URL is `_.BASE_PATH /user/65172ce26ebdfb77523a08d8`, and the status is 200 OK. The response body is a JSON object with a single key "msg": "Datos de usuario actualizados". The request body contains fields for fname, lname, email, role, and avatar, with the password field explicitly set to "123456".

Y al comprobarlo con **getMe**:

The screenshot shows a POSTMAN interface. The method is set to GET, the URL is `_.BASE_PATH /user/me`, and the status is 200 OK. The response body is a JSON object containing the user's details, including the password field which is now encrypted.

Nos damos cuenta que ahora si esta encriptada nuestra contraseña.



@hdtoledo

Ahora vamos con la imagen o avatar de nuestro usuario, si comprobamos en insomnia:

PATCH `_BASE_PATH /user/65172ce26ebdfb77523a08d8`

Send ▾ **200 OK** 109 ms 39 B

Multipart 1 ▾ Auth ▾ Query Headers 2 Docs

Add Delete All Toggle Description

firstname: Cristian Fernando
lastname: Lobaton
email: test04@gmail.com
password: 123456
role: admin
avatar: logo512.png

Preview ▾
1 ▾ {
2 "msg": "Datos de usuario actualizados"
3 }

Me dice que se ha enviado la información, pero al realizar la petición en getMe observamos:

Preview ▾ Headers 8 Cookies Timeline

1 ▾ {
2 "id": "65172ce26ebdfb77523a08d8",
3 "firstname": "Cristian Fernando",
4 "lastname": "Lobaton",
5 "email": "elpintor@gmail.com",
6 "role": "admin",
7 "active": true,
8 "password": "\$2a\$10\$5tPCAXjPW8RrzuuXHrXAeFtA/cWxwI/VK2Pz8MHHK2XxKPuEEURq",
9 "__v": 0
10 }

Que no estamos gestionando la ubicación del avatar para ello vamos a realizar lo siguiente dentro de nuestra función **updateUser** y verificamos que se estén enviando los datos:

```
controllers > user.controller.js > updateUser
65     } else {
66         delete userData.password
67     }
68
69     if (req.files.avatar) {
70         console.log(req.files.avatar)
71     }
72
73     User.findByIdAndUpdate({ _id: id }, userData, (error) => {
74         if(error) {
```

Vamos primero mediante la condición si hay un archivo de avatar pues que nos muestre el log de este, realizamos en insomnia la petición y en consola nos saldrá:



```
#####
### MERN API REST #####
#####
http://localhost:3000/api/v1
{
  fieldName: 'avatar',
  originalFilename: 'logo512.png',
  path: 'uploads\avatar\6AGM8ijG9ju6oDiJddfHIG1U.png',
  headers: {
    'content-disposition': 'form-data; name="avatar"; filename="logo512.png"',
    'content-type': 'image/png'
  },
  size: 27548,
  name: 'logo512.png',
  type: 'image/png'
}
```

Verificamos que nos devuelve el objeto con sus datos, para ello vamos a realizar lo siguiente:

```
controllers > user.controller.js > updateUser
65     } else {
66       delete userData.password
67     }
68
69     if (req.files.avatar) {
70       const imagePath = image.getFilePath(req.files.avatar)
71       userData.avatar = imagePath
72     }
73
74   User.findByIdAndUpdate({ _id: id }, userData, (error) => {
75     if (error) {
```

Agregamos al igual que lo hicimos con **imagePath** la ruta y se la pasamos a los datos de usuario, al comprobarlo de nuevo en insomnia por medio de **getMe** vamos a observar lo siguiente:

```
1 + {
2   "_id": "65172ce26ebfb77523a08d8",
3   "firstname": "Cristian Fernando",
4   "lastname": "Lobato",
5   "email": "elpinor@gmail.com",
6   "role": "admin",
7   "active": true,
8   "password": "$2a$10$5tPCAXjPw8Rrzui0irXFaeFTA/cbxwI/VK2Pz8MHK2XxKPuEEURq",
9   "__v": 0,
10  "avatar": "avatar/kAqX8gAGST1dx-Fji49xMkbQ.png"
11 }
```

Ya obtenemos la ubicación de nuestro avatar y se pasa a los datos de nuestro usuario para ser almacenado en la base de datos.



@hdtoledo

ELIMINANDO USUARIO

Ahora realizaremos el ultimo endpoint que será la eliminación del usuario para ello vamos a crear la función **deleteUser** dentro de **user.controller.js**:

```
controllers > js user.controller.js > <unknown>
  ...
84  async function deleteUser(req, res){
85      const { id } = req.params
86
87      User.findByIdAndDelete(id, (error) => {
88          if(error){
89              res.status(400).send({msg: "Error al eliminar el usuario"})
90          } else {
91              res.status(200).send({msg: "Usuario Eliminado"})
92          }
93      })
94  }
95
96
97  module.exports = [
98      getMe,
99      getUsers,
100     createUser,
101    updateUser,
102  deleteUser,
103]
```

Realizamos nuestra función asíncrona para eliminar el registro a través del **id** del usuario, lo cargamos en una **const** como hicimos anteriormente y utilizamos la función para eliminar, colocamos los mensajes en caso de error y si se ejecuta normalmente, por último, exportamos la función, y ahora nos vamos para nuestro archivo de rutas **user.router.js** y allí vamos a crear la ruta:

```
router > js user.router.js > ...
  ...
5
6  const md_upload = multiparty({ uploadDir: "./uploads/avatar" })
7  const api = express.Router()
8
9  api.get("/user/me", [md_auth.asureAuth], UserController.getMe)
10 api.get("/users", [md_auth.asureAuth], UserController.getUsers)
11 api.post("/user", [md_auth.asureAuth, md_upload], UserController.createUser)
12 api.patch("/user/:id", [md_auth.asureAuth, md_upload], UserController.updateUser)
13 api.delete("/user/:id", [md_auth.asureAuth], UserController.deleteUser)
14
15
16  module.exports = api
```

Agregamos la ruta a través de delete y ejecutamos nuestro middleware de autenticación junto con la función de eliminar.

Procedemos a comprobarlo a través de insomnia, para ello vamos a consultar nuestros usuarios que se encuentran inactivos:





GET `_.BASE_PATH /users?active=false`

Send **200 OK** 97.1 ms 1145 B Just Now

Body Auth Query Headers Docs

Preview Headers Cookies Timeline

```

1 * [
2 *   {
3 *     "_id": "65172aac6ebdfb77523a08d4",
4 *     "firstname": "Jimmy",
5 *     "lastname": "Lombana",
6 *     "email": "jimmy@gmail.com",
7 *     "role": "user",
8 *     "active": false,
9 *     "password": "$2a$10$VLt918W.CQe.ekAnZC.W.tKChFSGBDIQ%wRib./CEHD7HtgPppG",
10 *     "__v": 0
11 *   },
12 *   {
13 *     "_id": "6526e706964b3efffd9c6f9c",
14 *     "firstname": "name01",
15 *     "lastname": "lastname02",
16 *     "email": "test0@gmail.com",
17 *     "password": "$2a$10$7rJUpvjl1cI4XJK.P1lOu.5Um71vQ5s9IN9EBI3Edk22ctb4W",
18 *     "role": "admin",
19 *     "active": false,
20 *     "__v": 0
21 *   }
22 *

```

Ya que obtenemos los usuarios, seleccionamos uno de los id y lo copiamos, nos dirigimos a crear un nuevo endpoint en insomnia que se llame deleteUser:



Star 30.839

mernApp / mernAPI

localhost Add Cookies

Filter User

- DELETE** `_.BASE_PATH /user/6526e706964b3efffd9c6f9c`
- Body Auth Query Headers Docs
- Add Delete All Toggle Description
- Authorization Bearer `__ACCESS_TOKEN`

DEL deleteUser

POST createUser

PATCH updateUser

GET getUsers

Aplicamos el id en la ruta y nos disponemos a enviar la petición:



DELETE `_.BASE_PATH /user/6526e706964b3efffd9c6f9c`

Send **200 OK** 124 ms 27 B Just Now

Body Auth Query Headers Docs

Add Delete All Toggle Description

Authorization Bearer `__ACCESS_TOKEN`

Preview Headers Cookies Timeline

```

1 * {
2 *   "msg": "Usuario Eliminado"
3 *

```

Si todo va bien nos mostrara que se elimino el usuario y podemos disponernos a revisar mediante getUsers:



GET `_.BASE_PATH /users?active=false`

Send **200 OK** 106 ms 926 B Just Now

Body Auth Query Headers Docs

Add Delete All Toggle Description

Authorization Bearer `__ACCESS_TOKEN`

Preview Headers Cookies Timeline

```

1 * [
2 *   {
3 *     "_id": "65172aac6ebdfb77523a08d4",
4 *     "firstname": "Jimmy",
5 *     "lastname": "Lombana",
6 *     "email": "jimmy@gmail.com",
7 *     "role": "user",
8 *     "active": false,
9 *     "password": "$2a$10$VLt918W.CQe.ekAnZC.W.tKChFSGBDIQ%wRib./CEHD7HtgPppG",
10 *     "__v": 0
11 *   },
12 *   {
13 *     "_id": "6526e8bf964b3efffd9c6fa0",
14 *     "firstname": "Probando",
15 *     "lastname": "lastname02",
16 *     "email": "test0@gmail.com",
17 *     "password": "$2a$10$80JPog2ephQf7gxYJPa...ZNgnx0Q6ir9G47RusTelM7DihYZkqgu",
18 *     "role": "admin",
19 *     "active": false,
20 *     "__v": 0
21 *   },
22 *   {
23 *     "_id": "6526ef9393b71e8ea2b213c21"
24 *

```

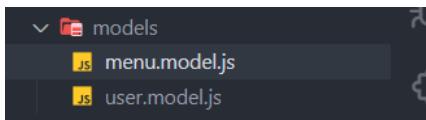
De esta manera terminamos de configurar nuestro CRUD de usuario.



@hdtoledo

MODELO MENU

El sistema de menú no se va quedará de manera estática, sino que se podrá activar y modificar directamente desde el panel del administrador, para ello vamos a crear un esquema que nos permita ser modificado directamente por el administrador, vamos a definirlo de la siguiente manera, creamos un archivo llamado **menu.model.js** dentro de **models**:



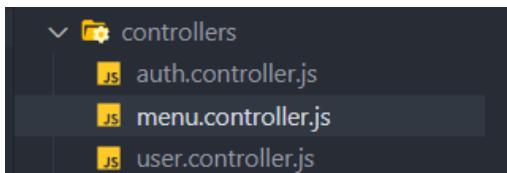
Y dentro vamos a definir nuestro menú de la siguiente manera:

```
models > JS menu.model.js > <unknown>
1  const mongoose = require("mongoose")
2
3  const MenuSchema = mongoose.Schema({
4      title: String,
5      path: String,
6      order: Number,
7      active: Boolean,
8  })
9
10 module.exports = mongoose.model("Menu", MenuSchema)
```

Nuestro modelo va a ser simple, ya que solo requerimos un título, una ruta, un orden y que se active o desactive.

ESTRUCTURA API MENU

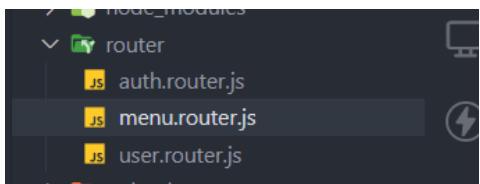
Ahora vamos a definir la estructura para el API menú, desde el controlador hasta exponerse al navegador, vamos a crear nuestro controlador **menu.controller.js** dentro de **controllers**:



Y dentro vamos a colocar toda la parte lógica para empezar a estructurar el menú:

```
controllers > JS menu.controller.js > ...
1  const Menu = require("../models/menu.model")
2
3  |
4
5  module.exports = {
```

Por el momento lo dejaremos así, ya que a continuación vamos a crear la ruta dentro de router nuestro **menu.router.js**:



Y dentro lo vamos a dejar de la siguiente manera:

```
router > js menu.router.js > ...
1 const express = require("express")
2 const MenuController = require("../controllers/menu.controller")
3 const md_auth = require("../middlewares/authenticated")
4
5 const api = express.Router()
6
7 //EndPoints
8
9
10 module.exports = api
```

Importamos express, nuestro controlador del menú, y nuestro middleware de autenticación y por último exportamos el módulo.

Ahora vamos a **app.js** y hacemos la importación de nuestro modulo:

```
js app.js > menuRoutes
9 // Importar rutas
10 const authRoutes = require("./router/auth.router")
11 const userRoutes = require("./router/user.router")
12 const menuRoutes = require("./router/menu.router")
13
14 //Configurar Body Parse
```

Y mas abajo colocamos la configuración de la ruta de la siguiente manera:

```
js app.js > ...
25
26 // Configurar Rutas
27 app.use(`/api/${API_VERSION}`, authRoutes)
28 app.use(`/api/${API_VERSION}`, userRoutes)
29 app.use(`/api/${API_VERSION}`, menuRoutes)
30
31
32 module.exports = app
```



@hdtoledo

Ahora por último levantamos nuestro servidor y no debe generarnos ningún error:

```
LAPTOPHDMI | D:\DATA\...\server | yarn 18.16.0 1ms
{ hdtledo } yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
|
```

CREANDO EL MENU

Ahora vamos a empezar con el primer menú así que nos ubicamos dentro de **menu.controller.js** y creamos nuestra función **createMenu**:

```
controllers > menu.controller.js > <unknown>
1 const Menu = require("../models/menu.model")
2
3 async function createMenu(req, res) {
4     const menu = new Menu(req.body)
5
6     menu.save((error, menuStored) => {
7         if(error) {
8             res.status(400).send({msg: "Error al crear el menu"})
9         } else {
10             res.status(200).send(menuStored)
11         }
12     })
13 }
14
15 module.exports = {
16     createMenu,
17
18 }
```

Creamos la función asíncrona llamada **createMenu()**, que se utiliza para crear un nuevo menú en una base de datos. La función recibe dos parámetros: **req** y **res**, que representan la solicitud y la respuesta **HTTP**, respectivamente.

El primer paso que realiza la función es crear un nuevo objeto **Menu** a partir del cuerpo de la solicitud **HTTP**. El cuerpo de la solicitud contiene los datos del nuevo menú, como su nombre, descripción, etc.

Una vez creado el objeto **Menu**, la función llama al método **save()** para guardarlo en la base de datos. El método **save()** es asíncrono, lo que significa que la función **createMenu()** no esperará a que se complete antes de devolver una respuesta.



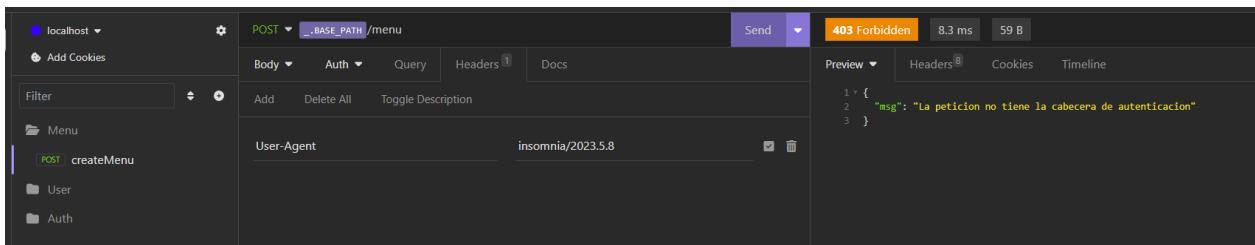
@hdtledo

Si el menú se guarda correctamente, la función `createMenu()` devolverá una respuesta HTTP con el estado 200 OK y el menú guardado. Si se produce un error al guardar el menú, la función devolverá una respuesta HTTP con el estado 400 Bad Request y un mensaje de error.

Ahora vamos a `menu.router.js` y creamos nuestro primer endpoint:

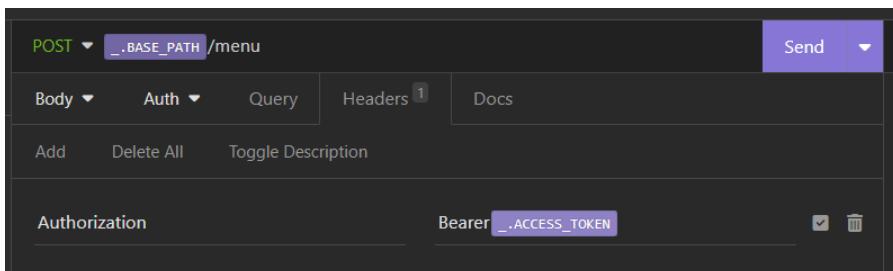
```
router > js menu.router.js > ...
1 const express = require("express")
2 const MenuController = require("../controllers/menu.controller")
3 const md_auth = require("../middlewares/authenticated")
4
5 const api = express.Router()
6
7 api.post("/menu", [md_auth.ensureAuth], MenuController.createMenu)
8
9
10 module.exports = api
```

Ahora vamos a probarla utilizando nuestro insomnia de la siguiente manera, configuramos una nueva carpeta con un nuevo endpoint y hacemos la prueba para validar sin cabecera:



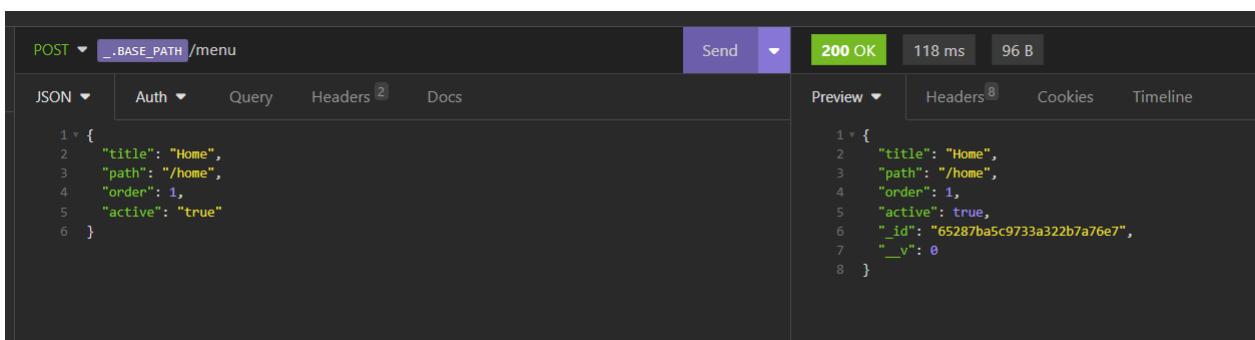
The screenshot shows the insomnia REST client interface. On the left, there's a sidebar with a tree view containing 'localhost', 'Add Cookies', 'Filter', 'Menu' (which has a 'POST createMenu' entry), 'User', and 'Auth'. The main area shows a request configuration for a 'POST' to '_BASE_PATH/menu'. The 'Headers' tab is selected, showing 'User-Agent: insomnia/2023.5.8'. The 'Send' button is highlighted in purple. To the right, the status bar shows '403 Forbidden', '8.3 ms', and '59 B'. Below the status bar, the 'Preview' tab is open, displaying the JSON response: { "msg": "La petición no tiene la cabecera de autenticación" }.

Y ahora configuramos con la cabecera correcta:



This screenshot shows the same insomnia interface after setting the correct header. In the 'Headers' tab, there is a single entry 'Authorization: Bearer _ACCESS_TOKEN'. The rest of the interface is identical to the previous screenshot, showing the 'POST /menu' endpoint and the 403 Forbidden response.

Y por último pasamos los datos a través de JSON y le damos a enviar:



This screenshot shows the final step where JSON data is passed through the 'Body' tab. The JSON payload is: { "title": "Home", "path": "/home", "order": 1, "active": "true" }. The 'Send' button is highlighted in purple. The status bar shows '200 OK', '118 ms', and '96 B'. The 'Preview' tab shows the successful response: { "title": "Home", "path": "/home", "order": 1, "active": true, "_id": "65287ba5c9733a322b7a76e7", "__v": 0 }.

Y ahora vemos que nos está creando nuestro menú, para ello vamos a revisar en mongodb:



Y aca podemos observar que ya tenemos nuestra collection menus, vamos a probar a enviar uno nuevo:

Y revisamos en mongo:

De esta manera ya tenemos nuestra creación de menus funcional.



OBTENIENDO LOS MENUS

Ahora vamos a crear el endpoint para obtener el menú completo de nuestra aplicación, para ello nos vamos a continuar en **menu.controller.js** y creamos nuestra función **getMenus**:

```
controllers > js menu.controller.js > ↑ getMenus
14
15  async function getMenus(req, res) {
16    const { active } = req.query
17
18    let response = null
19
20    if (active === undefined){
21      response = await Menu.find()
22    } else {
23      response = await Menu.find({ active })
24    }
25
26    if(!response) {
27      res.status(400).send({msg: "No se ha encontrado ningun menu"})
28    } else {
29      res.status(200).send(response)
30    }
31
32  }
33
34  module.exports = {
35    createMenu,
36    getMenus,
37  }
38 }
```

Así como realizamos en nuestro controlador de usuarios para obtener el menú lo aplicamos de la misma manera y controlamos en caso de que no se traiga nada con mensajes de status.

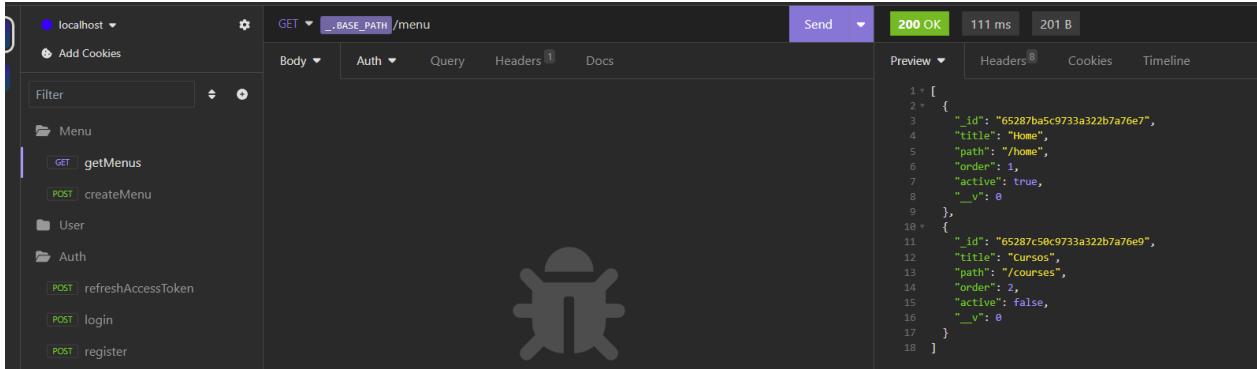
Ahora nos vamos a **menu.router.js** y dejamos nuestra ruta:

```
router > js menu.router.js > ...
1  const express = require("express")
2  const MenuController = require("../controllers/menu.controller")
3  const md_auth = require("../middlewares/authenticated")
4
5  const api = express.Router()
6
7  api.post("/menu", [md_auth.asureAuth], MenuController.createMenu)
8  api.get("/menu", MenuController.getMenus)
9
10 module.exports = api
```

En nuestra ruta no es necesario que utilizar el middleware de autenticación ya que este menú al obtenerlo lo vamos a mostrar tanto en la web normal y logueados.

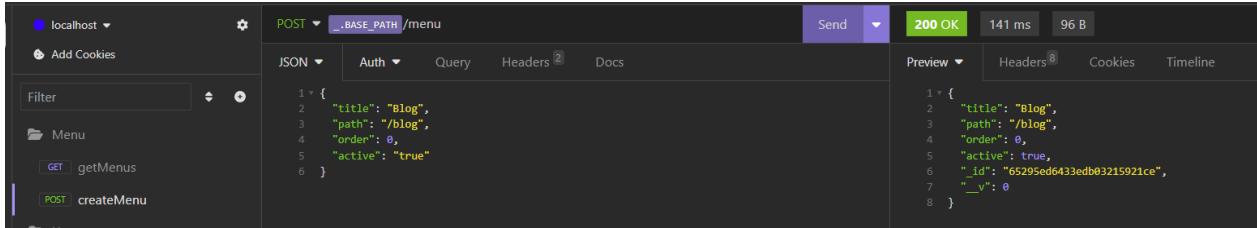


Ahora nos dirigimos a insomnia y allí vamos a probar nuestra nueva ruta:



```
1: [
2:   {
3:     "_id": "65287ba5c9733a322b7a76e7",
4:     "title": "Home",
5:     "path": "/home",
6:     "order": 1,
7:     "active": true,
8:     "_v": 0
9:   },
10:  {
11:    "_id": "65287c50c9733a322b7a76e9",
12:    "title": "Cursos",
13:    "path": "/courses",
14:    "order": 2,
15:    "active": false,
16:    "_v": 0
17:  }
18: ]
```

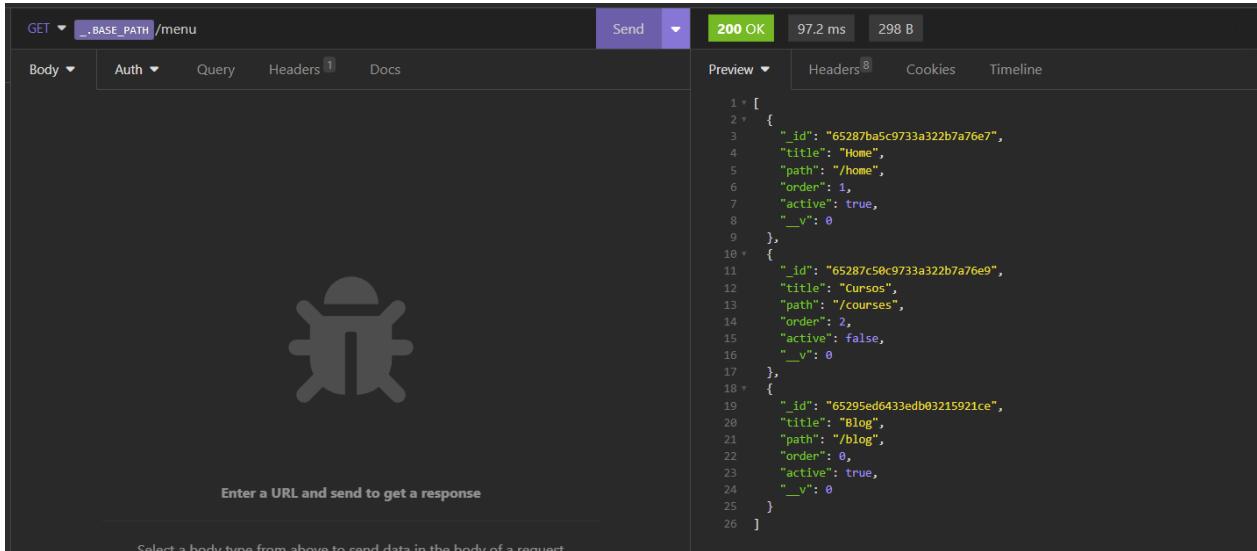
Acá vemos la respuesta de nuestra solicitud y todo va correcto, ahora vamos a probar agregando un nuevo menú:



```
1: {
2:   "title": "Blog",
3:   "path": "/blog",
4:   "order": 0,
5:   "active": true
6: }
```

```
1: {
2:   "title": "Blog",
3:   "path": "/blog",
4:   "order": 0,
5:   "active": true,
6:   "_id": "65295ed6433edb03215921ce",
7:   "_v": 0
8: }
```

Notemos que el menú lo hemos dejado en el orden 0, ahora vamos nuevamente a **getMenus**:



```
1: [
2:   {
3:     "_id": "65287ba5c9733a322b7a76e7",
4:     "title": "Home",
5:     "path": "/home",
6:     "order": 1,
7:     "active": true,
8:     "_v": 0
9:   },
10:  {
11:    "_id": "65287c50c9733a322b7a76e9",
12:    "title": "Cursos",
13:    "path": "/courses",
14:    "order": 2,
15:    "active": false,
16:    "_v": 0
17:  },
18:  {
19:    "_id": "65295ed6433edb03215921ce",
20:    "title": "Blog",
21:    "path": "/blog",
22:    "order": 0,
23:    "active": true,
24:    "_v": 0
25:  }
26: ]
```

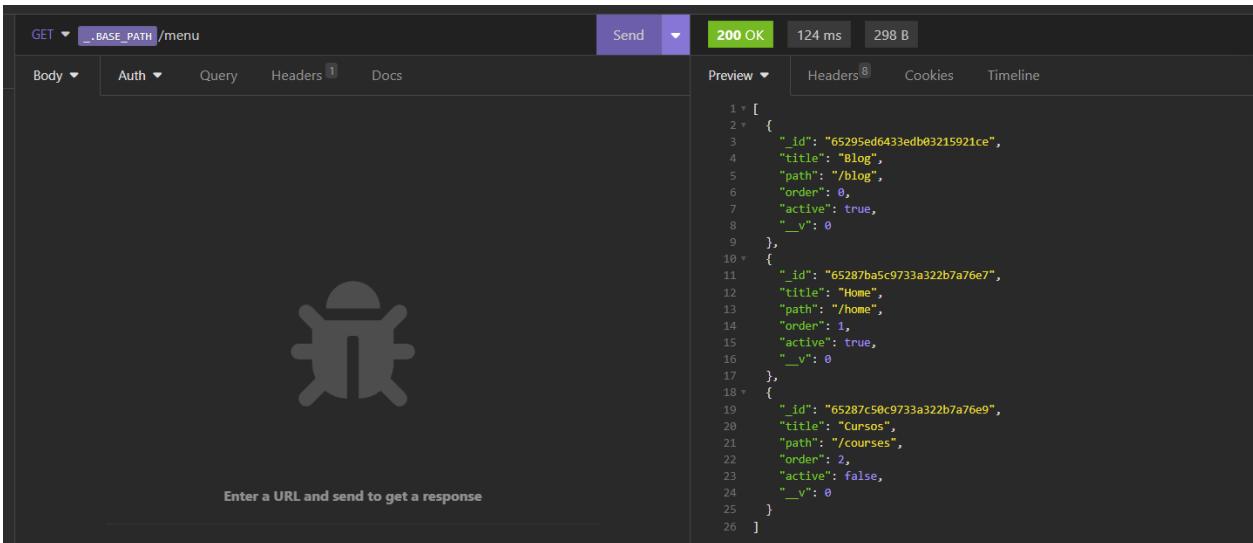
Y acá observamos que al obtenerlo el orden de posición 0 es el tercero, vamos a realizar un ajuste al momento de obtenerlo para que se nos muestre en el orden correcto así que vamos a menu.controller.js y modificamos la respuesta de la siguiente manera:



```
controllers > JS menu.controller.js > getMenus
19
20     if (active === undefined){
21         response = await Menu.find().sort({ order: "asc"})
22     } else {
23         response = await Menu.find({ active }).sort({ order: "asc"})
24     }
25
```

El método `.sort()` se utiliza en una consulta de base de datos para ordenar los resultados según un campo específico. En el contexto de la línea de código que proporcionaste (`Menu.find().sort({ order: "asc" })`), esto significa que los documentos encontrados en la colección **Menu** se ordenarán en función del valor del campo "order" en orden ascendente ("asc").

Ahora volvemos a insomnia a realizar la solicitud y de esta manera obtendremos el orden que necesitábamos:



The screenshot shows the insomnia API client interface. A GET request is made to `.../menu`. The response status is **200 OK**, with a response time of 124 ms and a body size of 298 B. The response body is a JSON array of three menu items:

```
[{"_id": "65295ed6433edb03215921ce", "title": "Blog", "path": "/blog", "order": 0, "active": true, "__v": 0}, {"_id": "65287ba5c9733a322b7a76e7", "title": "Home", "path": "/home", "order": 1, "active": true, "__v": 0}, {"_id": "65287c50c9733a322b7a76e9", "title": "Cursos", "path": "/courses", "order": 2, "active": false, "__v": 0}]
```



ACTUALIZANDO EL MENU

Vamos a proceder a actualizar el menú mediante su id, para ello vamos a crear dentro de **menu.controller.js** la función **updateMenu**:

```
controllers > js menu.controller.js > <unknown>
33
34     async function updateMenu(req, res) {
35         const { id } = req.params
36         const menuData = req.body
37
38         Menu.findByIdAndUpdate({ _id: id }, menuData, (error) => {
39             if (error) {
40                 res.status(400).send({msg: "Error al actualizar el menu"})
41             } else {
42                 res.status(200).send({msg: "Actualizacion correcta"})
43             }
44         })
45     }
46
47     module.exports = {
48         createMenu,
49         getMenus,
50         updateMenu,
51     }

```

En esta función observamos que es muy similar a nuestra función de actualizar usuario, recordemos que ubicamos por el id el usuario que queremos actualizar, una vez validados los datos ejecutamos la acción y se procede a cargar, si hay error nos dará el mensaje de status de igual manera si se ejecuta correctamente. Y por último lo exportamos, ahora nos dirigimos a **menu.router.js** y creamos la ruta para nuestro **update**:

```
router > js menu.router.js > ...
1  const express = require("express")
2  const MenuController = require("../controllers/menu.controller")
3  const md_auth = require("../middlewares/authenticated")
4
5  const api = express.Router()
6
7  api.post("/menu", [md_auth.asureAuth], MenuController.createMenu)
8  api.get("/menu", MenuController.getMenus)
9  api.patch("/menu/:id", [md_auth.asureAuth], MenuController.updateMenu)
10
11
12  module.exports = api

```

De esta manera cargamos la ruta utilizando patch, cargamos el middleware de autenticación y ejecutamos la función.

Ahora nos dirigimos a insomnia y creamos para probarlo completamente vamos a generar un nuevo menú primero:



```

POST .BASE_PATH /menu
Send 200 OK 134 ms 104 B
JSON Auth Query Headers 2 Docs
1 v {
2   "title": "Contacto",
3   "path": "/contact",
4   "order": 10,
5   "active": "true"
6 }
Preview Headers Cookies Timeline
1 v {
2   "title": "Contacto",
3   "path": "/contact",
4   "order": 10,
5   "active": true,
6   "_id": "652968e8525ccdbfe9aa4166",
7   "__v": 0
8 }

```

Vamos a tomar el id que nos genero y nos vamos a crear una nueva ruta que se llamará **updateMenus**:

```

PATCH .BASE_PATH /menu/652968e8525ccdbfe9aa4166
Send 403 Forbidden 1.88 ms 59 B
JSON Auth Query Headers Cookies Timeline
1 v ...
Preview Headers Cookies Timeline
1 v {
2   "msg": "La petición no tiene la cabecera de autenticación"
3 }

```

Recordemos que para procesar la actualización de los menus debemos estar autenticados para ello configuramos nuestro header:

```

PATCH .BASE_PATH /menu/652968e8525ccdbfe9aa4166
Send
JSON Auth Query Headers 2 Docs
Add Delete All Toggle Description
Content-Type application/json
Authorization Bearer .ACCESS_TOKEN

```

Y ahora procedemos a enviar los datos mediante JSON, recordemos que podemos enviar uno o varios al tiempo:

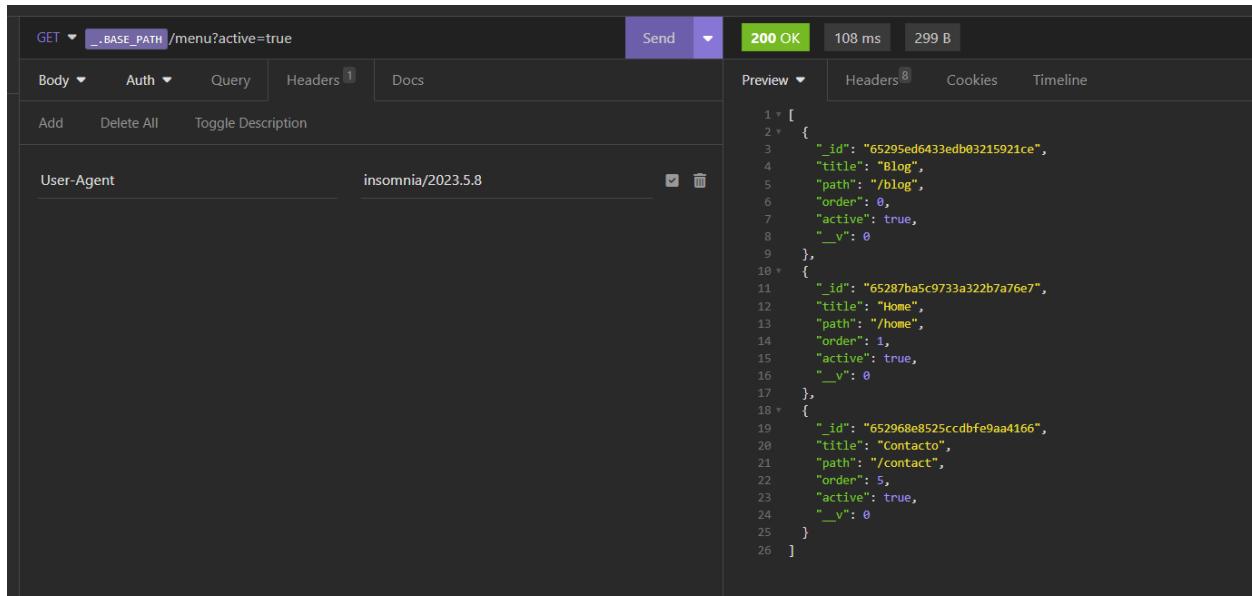
```

PATCH .BASE_PATH /menu/652968e8525ccdbfe9aa4166
Send 200 OK 98.4 ms 32 B
JSON Auth Query Headers 1 Docs
1 v {
2   "title": "Contacto",
3   "path": "/contact",
4   "order": 5,
5   "active": "true"
6 }
Preview Headers Cookies Timeline
1 v {
2   "msg": "Actualización correcta"
3 }

```



Y vemos como se ha actualizado el order, revisamos en getMenus para constatar que si se ha cambiado:



The screenshot shows the Insomnia REST client interface. A GET request is made to `__BASE_PATH /menu?active=true`. The response is **200 OK** with a duration of **108 ms** and a size of **299 B**. The **Preview** tab displays the JSON response:

```
1 [  
2 {  
3   "_id": "65295ed6433edb03215921ce",  
4   "title": "Blog",  
5   "path": "/blog",  
6   "order": 0,  
7   "active": true,  
8   "__v": 0  
9 },  
10 {  
11   "_id": "65287ba5c9733a322b7a76e7",  
12   "title": "Home",  
13   "path": "/home",  
14   "order": 1,  
15   "active": true,  
16   "__v": 0  
17 },  
18 {  
19   "_id": "652968e8525ccdbfe9aa4166",  
20   "title": "Contacto",  
21   "path": "/contact",  
22   "order": 5,  
23   "active": true,  
24   "__v": 0  
25 }  
26 ]
```

De esta manera ya tenemos nuestra función de updateMenu lista para poder realizar cualquier edición.

ELIMINACION DE MENU

Vamos a crear nuestra función **deleteMenu** dentro de **menu.controller.js**:

```
controllers > js menu.controller.js > ✖ deleteMenu  
46  
47   async function deleteMenu(req, res) {  
48  
49     const { id } = req.params  
50  
51     Menu.findByIdAndDelete(id, (error) => {  
52       if(error){  
53         res.status(400).send({msg: "Error al eliminar el menu"})  
54       } else {  
55         res.status(200).send({msg: "Menu Eliminado Correctamente"})  
56       }  
57     })  
58   }  
59  
60   module.exports = {  
61     createMenu,  
62     getMenus,  
63     updateMenu,  
64     deleteMenu,  
65   }
```

La estructura que observamos es muy similar a la usada al eliminar nuestro usuario, de esta manera lo que sigue es dirigirnos a crear la ruta en **menu.router.js**:



```

router > js menu.router.js > ...
1 const express = require("express")
2 const MenuController = require("../controllers/menu.controller")
3 const md_auth = require("../middlewares/authenticated")
4
5 const api = express.Router()
6
7 api.post("/menu", [md_auth.asureAuth], MenuController.createMenu)
8 api.get("/menu", MenuController.getMenus)
9 api.patch("/menu/:id", [md_auth.asureAuth], MenuController.updateMenu)
10 api.delete("/menu/:id", [md_auth.asureAuth], MenuController.deleteMenu)
11
12
13 module.exports = api

```

De esta manera configuramos nuestra ruta y ahora procedemos a configurar en insomnia la ruta, primero revisamos que menus tenemos y para mi caso voy a eliminar el menú contacto:

The screenshot shows the Insomnia REST client interface. The request URL is `GET /menu?active=true`. The response is **200 OK** with a duration of 107 ms and a size of 299 B. The response body is a JSON array containing three menu items:

```

1 [
2   {
3     "_id": "65295ed6433edb83215921ce",
4     "title": "Blog",
5     "path": "/blog",
6     "order": 0,
7     "active": true,
8     "__v": 0
9   },
10  {
11    "_id": "65287ba5c9733a322b7a7e7",
12    "title": "Home",
13    "path": "/home",
14    "order": 1,
15    "active": true,
16    "__v": 0
17  },
18  {
19    "_id": "652960e8525ccdbfe9aa4166",
20    "title": "Contacto",
21    "path": "/contact",
22    "order": 5,
23    "active": true,
24    "__v": 0
25 }
26 ]

```

Vamos a crear la ruta de la siguiente manera:

The screenshot shows the Insomnia REST client interface. The request URL is `DELETE /menu/652960e8525ccdbfe9aa4166`. The response is **403 Forbidden** with a duration of 2.05 ms and a size of 59 B. The response body is a JSON object:

```

1 {
2   "msg": "La peticion no tiene la cabecera de autenticacion"
3 }

```

Recordemos que al intentar enviarlo debemos configurar nuestra cabecera:



Todo va bien, ahora revisamos si nuestro menú contacto fue eliminado:

Y solo nos aparece los menus de blog y home, así que ya tenemos funcionando nuestra función de eliminación.

MODELO CURSOS

Vamos a realizar nuestro modelo para la sección de cursos en los cuales vamos a crear de la siguiente manera, primero en **models** vamos a crear **course.model.js**:

```
models > js course.model.js > ...
1  const mongoose = require("mongoose")
2
3  const CourseSchema = mongoose.Schema({
4    title: String,
5    miniature: String,
6    description: String,
7    url: String,
8    price: Number,
9    score: Number
10 })
11
12 module.exports = mongoose.model("Course", CourseSchema)
```

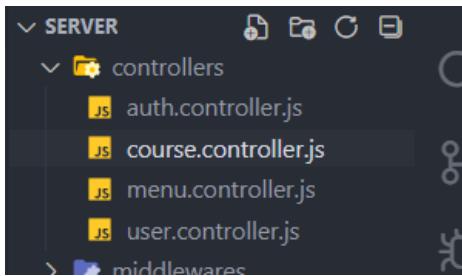
Establecemos nuestro esquema para la db de curso



@hdtoledo

ESTRUCTURA API CURSO

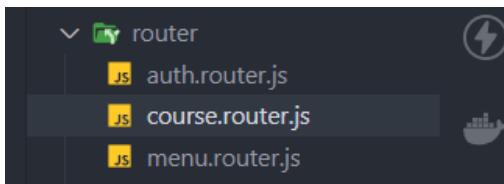
Vamos a crear la estructura de la API del curso para ello vamos a crear el controlador **course.controller.js**



Dentro vamos a dejar lo siguiente:

```
controllers > course.controller.js > ...
1 const Course = require("../models/course.model")
2
3 //Funciones
4
5 |
6 module.exports = {
7
8 }
```

Por el momento vamos a armar la estructura, y ahora creamos la ruta **course.router.js**:

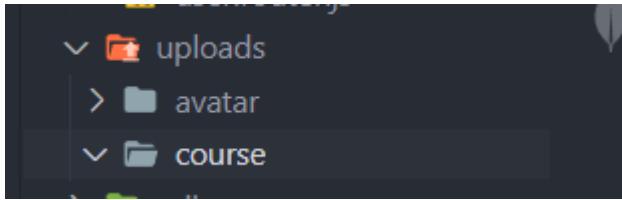


Y dentro dejamos la siguiente estructura:

```
router > course.router.js > <unknown>
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const CourseController = require("../controllers/course.controller")
4 const md_auth = require("../middlewares/authenticated")
5 const md_upload = multiparty({ uploadDir: "/uploads/course" })
6
7
8 const api = express.Router()
9
10 //APIs ...
11
12
13 module.exports = api
```



Las **const** que vemos son prácticamente las mismas que hemos implementado en **user.router**, ahora vamos a crear la carpeta **course** dentro de **uploads**:



Ahora nos dirigimos a **app.js** y hacemos la integración:

```
js app.js > courseRoutes
9  // Importar rutas
10 const authRoutes = require("./router/auth.router")
11 const userRoutes = require("./router/user.router")
12 const menuRoutes = require("./router/menu.router")
13 const courseRoutes = require("./router/course.router")
14
```

Importamos nuestra nueva ruta y hacemos la configuración:

```
js app.js > ...
25 // Configurar Rutas
26 app.use('/api/${API_VERSION}', authRoutes)
27 app.use('/api/${API_VERSION}', userRoutes)
28 app.use('/api/${API_VERSION}', menuRoutes)
29 app.use('/api/${API_VERSION}', courseRoutes)
30
31
32 module.exports = app
```

Para validar que todo este correcto levantamos el servidor y no debe arrojarnos ningún error:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
|
```

Así que con esto ya tenemos la estructura base de cursos.

CREACION DEL CURSO

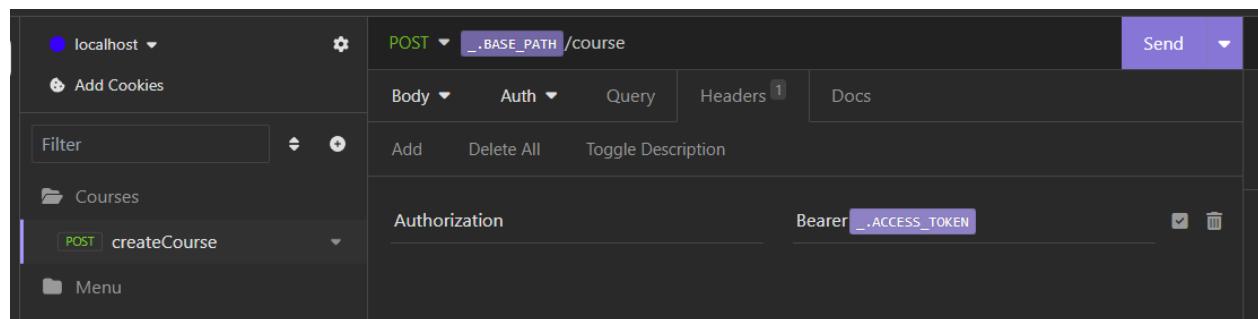
Vamos a empezar por nuestro **course.controller.js** así que lo vamos a estructurar de la siguiente manera:

```
controllers > js course.controller.js > ...
1  const Course = require("../models/course.model")
2  const image = require("../utils/image")
3
4
5  async function createCourse(req, res) {
6      const course = new Course(req.body)
7
8      const imagePath = image.getFilePath(req.files.miniature)
9      course.miniature = imagePath
10
11     course.save((error, courseStored) => {
12         if(error) {
13             res.status(400).send({msg: "Error al crear el curso"})
14         } else {
15             res.status(201).send(courseStored)
16         }
17     })
18 }
19
20 }
21 |
22 module.exports = {
23     createCourse,
24 }
```

Ahora vamos a **course.router.js** y vamos a dejar la siguiente estructura:

```
router > js course.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const CourseController = require("../controllers/course.controller")
4  const md_auth = require("../middlewares/authenticated")
5  const md_upload = multiparty({ uploadDir: "./uploads/course" })
6
7
8  const api = express.Router()
9
10 api.post("/course", [md_auth.ensureAuth, md_upload], CourseController.createCourse)
11
12
13 module.exports = api
```

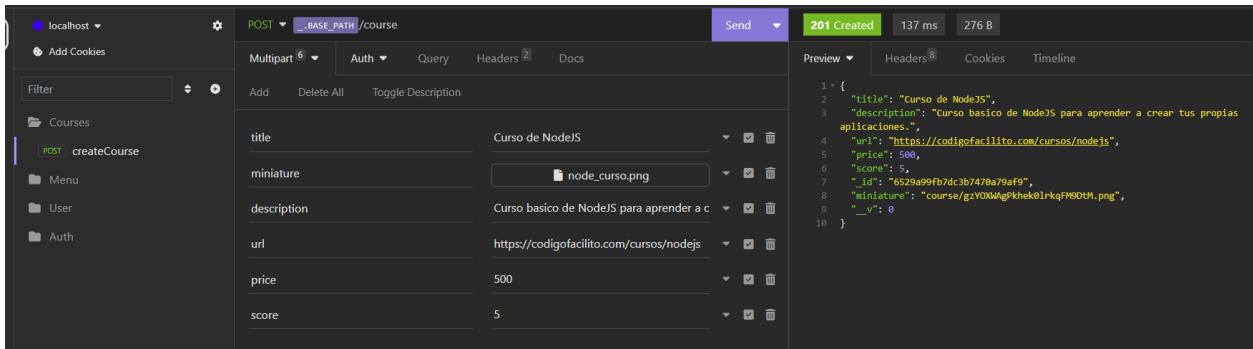
De esta manera con esta creación de la ruta ya deberíamos poder crear un curso, para ello utilizamos insomnia y vamos a crear una nueva carpeta con ruta:



The screenshot shows the insomnia API client interface. On the left, there's a sidebar with a 'localhost' dropdown, a 'Courses' folder containing a 'createCourse' item (marked with a green 'POST' icon), and a 'Menu' folder. The main area shows a POST request to '/course'. The 'Authorization' tab is selected, displaying 'Bearer _ACCESS_TOKEN'. Other tabs include 'Body', 'Auth', 'Query', and 'Headers' (with one entry). A 'Send' button is at the top right.

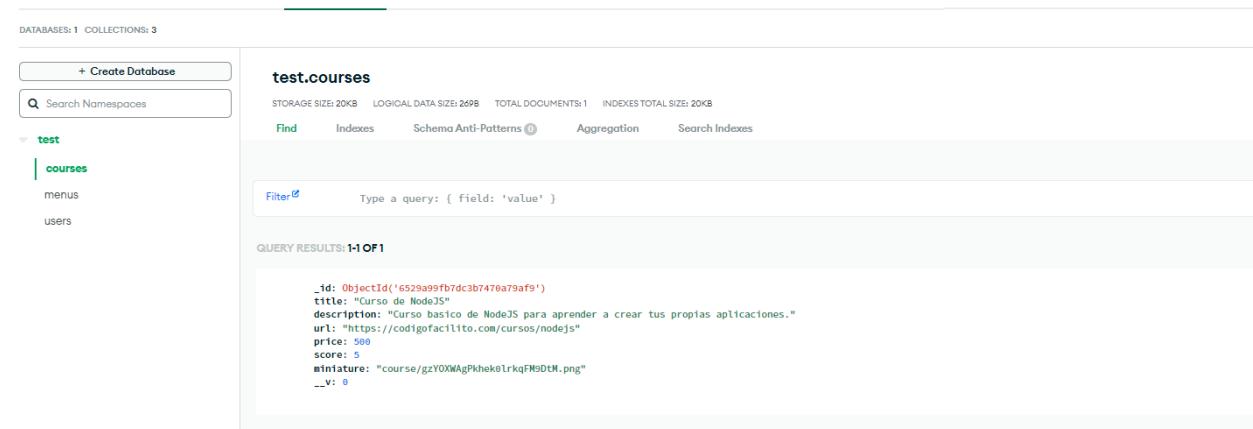


Una vez configurado vamos a pasar a través de **multipart** nuestros datos del curso:



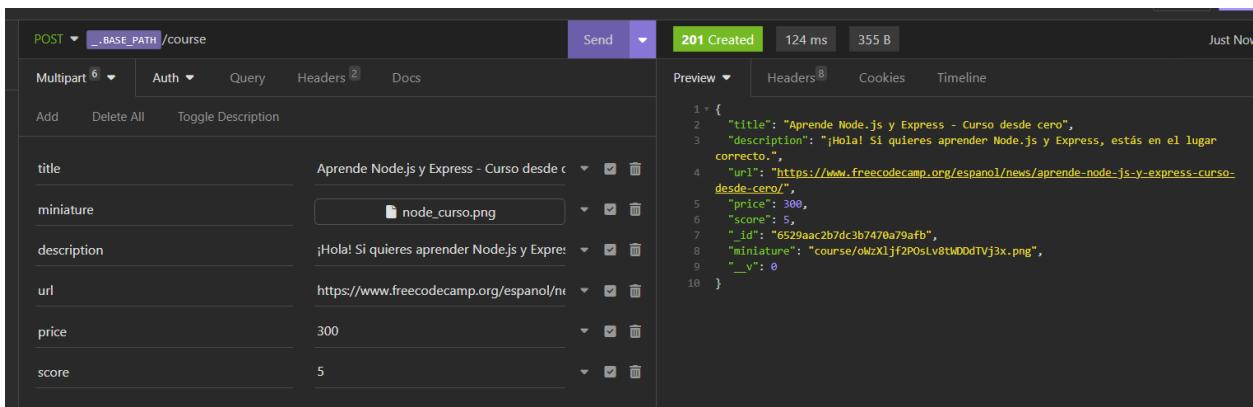
```
1: {
2:   "title": "Curso de NodeJS",
3:   "descripcion": "Curso basico de NodeJS para aprender a crear tus propias aplicaciones.",
4:   "url": "https://codigofacilito.com/cursos/nodejs",
5:   "price": 500,
6:   "score": 5,
7:   "_id": "6529a99fb7dc3b7470a79af9",
8:   "miniature": "course/gzYOXWAgPkhekolrkqFm0tM.png",
9:   "__v": 0
10: }
```

Ahora para verificar vamos a mongodb y nos encontramos con nuestro nuevo modelo:



```
_id: ObjectId('6529a99fb7dc3b7470a79af9')
title: "Curso de NodeJS"
description: "Curso basico de NodeJS para aprender a crear tus propias aplicaciones."
url: "https://codigofacilito.com/cursos/nodejs"
price: 500
score: 5
miniature: "course/gzYOXWAgPkhekolrkqFm0tM.png"
__v: 0
```

Vamos a comprobar nuevamente con otros datos de curso y validamos:



```
1: {
2:   "title": "Aprende Node.js y Express - Curso desde cero",
3:   "description": "¡Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto.",
4:   "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
5:   "price": 300,
6:   "score": 5,
7:   "_id": "6529aac2b7dc3b7470a79afb",
8:   "miniature": "course/0M2Xljf2P0sLv8t0DDtVj3x.png",
9:   "__v": 0
10: }
```



@hdtoledo

Validamos en mongodb:

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with a '+ Create Database' button, a search bar for namespaces, and a tree view showing the 'test' database with 'courses', 'menus', and 'users' collections. The main area is titled 'test.courses' and shows storage details: 36KB, logical data size: 617B, total documents: 2, and index total size: 36KB. Below this are tabs for 'Find', 'Indexes', 'Schema Anti-Patterns', 'Aggregation', and 'Search Indexes'. A 'Filter' input field is present, and a 'Results' button is on the right. The results section is titled 'QUERY RESULTS: 1-2 OF 2' and lists two documents:

```
_id: ObjectId('6529a89fb7dc3b7476a79af9')
title: "Curso de Node.js"
description: "Curso basico de NodeJS para aprender a crear tus propias aplicaciones."
url: "https://codigofacilito.com/cursos/nodejs"
price: 500
score: 5
miniature: "course/gzYXWAgPkhokelrlqfMSdth.png"
__v: 0

_id: ObjectId('6529aac2b7dc3b7476a79af9')
title: "Aprende Node.js y Express - Curso desde cero"
description: "¡Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto."
url: "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-cu-"
price: 300
score: 5
miniature: "course/oWzXljf2P0s1v8tW0DdTvj3x.png"
__v: 0
```

Y de esta manera ya tenemos nuestra función de crear cursos.

OBTENER CURSOS

Para obtener nuestros cursos nos ubicamos en `course.controller.js` y hacemos la función `getCourse`, aca debemos tener en cuenta que debemos paginar la cantidad de datos que vamos a mostrar y para ello vamos a implementar algo luego:

```
controllers > course.controller.js >  <unknown>
22  async function getCourse(req, res) {
23    Course.find((error, courses) => {
24      if (error) {
25        res.status(400).send({msg: "Error al obtener los cursos"})
26      } else {
27        res.status(200).send(courses)
28      }
29    })
30  }
31
32
33  module.exports = {
34    createCourse,
35     getCourse,
36  }
```

Y ahora nos dirigimos a `course.router.js` y creamos la ruta:



```

router > js course.router.js > ...
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const CourseController = require("../controllers/course.controller")
4 const md_auth = require("../middlewares/authenticated")
5 const md_upload = multiparty({ uploadDir: "./uploads/course" })

6
7
8 const api = express.Router()
9
10 api.post("/course", [md_auth.asureAuth, md_upload], CourseController.createCourse)
11 api.get("/course", CourseController.getCourse)
12
13
14 module.exports = api

```

Recordemos que esta ruta es pública y que tanto los visitantes del sitio como los logueados van a poder verlo, por ello no aplicamos el middleware.

Vamos ahora a insomnia y creamos una nueva petición:

The screenshot shows the Insomnia REST Client interface. On the left, there's a sidebar with a navigation tree: Courses (GET getCourses, POST createCourse), Menu, User, and Auth. The main area has a large placeholder icon. At the top, it says "localhost" and "GET _BASE_PATH /course". Below that are tabs for Body, Auth, Query, Headers, and Docs. To the right, the response details are shown: "200 OK", "116 ms", "634 B", and "Just Now". Under "Preview", there is JSON output representing two course documents:

```

1 [
2   {
3     "_id": "6529a9fb7dc3b7470a79af9",
4     "title": "Curso de NodeJS",
5     "description": "Curso basico de NodeJS para aprender a crear tus propias aplicaciones.",
6     "url": "https://codigofacilito.com/cursos/nodejs",
7     "price": 500,
8     "score": 5,
9     "miniature": "course/gzY0WAgPhek0lrkqFM9Dtt4.png",
10    "__v": 0
11  },
12  {
13    "_id": "6529aac2b7dc3b7470a79af0",
14    "title": "Aprende Node.js y Express - Curso desde cero",
15    "description": "Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto!",
16    "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
17    "price": 300,
18    "score": 5,
19    "miniature": "course/dKzX1jf2PosLv8tMDDdTVj3x.png",
20    "__v": 0
21  }
22 ]

```

De esta manera ya obtenemos todos los cursos que acabamos de subir.

PAGINACION DE CURSOS

Ahora imaginemos como paginamos si tuviéramos 100 cursos o más, sería bastante tedioso, así que para ello vamos a utilizar la dependencia de **mongoose paginate**:

The screenshot shows the npm package page for "mongoose-paginate" version 5.0.3. The page includes the package name, version, and a note that types are available via "@types/mongoose-paginate". The "README" section contains the following text:

mongoose-paginate

Pagination plugin for Mongoose

npm v5.0.3 build no longer available

Note: This plugin will only work with Node.js >= 4.0 and Mongoose >= 4.0.

Installation

```

npm install mongoose-paginate

```



En nuestro terminal ejecutamos **yarn add mongoose-paginate**

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
{ @hdtolledo } yarn add mongoose-paginate
yarn add v1.22.19
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...
```

Verificamos en nuestro **package.json**:

```
11  },
12  "dependencies": {
13    "bcryptjs": "^2.4.3",
14    "body-parser": "1.20.0",
15    "connect-multiparty": "^2.2.0",
16    "cors": "2.8.5",
17    "express": "4.18.1",
18    "jsonwebtoken": "8.5.1",
19    "mongoose": "6.6.1",
20    "mongoose-paginate": "^5.0.3",
21    "nodemon": "2.0.20"
22  }
23 }
24 }
```

Ahora levantamos nuestro servidor de nuevo:

```
LAPTOPHDMI1 D:\DATA\..\..\server yarn 18.16.0 1ms
{ @hdtolledo } yarn dev
yarn run v1.22.19
$ nodemon index.js
[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
|
```

Vamos a ubicarnos en **course.model.js** y vamos a importar nuestra nueva dependencia:

```
models > course.model.js > ...
1 const mongoose = require("mongoose")
2 const mongoosePaginate = require("mongoose-paginate")
3
4 const CourseSchema = mongoose.Schema({
5   title: String,
6   miniature: String,
7   description: String,
8   url: String,
9   price: Number,
10  score: Number
11 })
12
13 CourseSchema.plugin(mongoosePaginate)
14
15 module.exports = mongoose.model("Course", CourseSchema)
```



@hdtolledo

y lo iniciamos a través del uso del plugin, de esta manera ya podemos empezar a utilizarlo, para verificarlo nos vamos a **course.controller.js** y dentro de nuestra función **getCourse** hacemos lo siguiente:

```
controllers > js course.controller.js > <unknown>
22  async function getCourse(req, res) {
23
24    const options = {
25      page: 1,
26      limit: 10,
27    }
28
29    Course.paginate({}, options, (error, courses) => {
30      if (error) {
31        res.status(400).send({msg: "Error al obtener los cursos"})
32      } else {
33        res.status(200).send(courses)
34      }
35    })
36
37
38  module.exports = [
39    createCourse,
40    getCourse,
41  ]
42
```

De esta manera vamos a ver la implementación de nuestra dependencia, así que ejecutamos de nuevo insomnia y vamos a notar como cambia el contenido que nos llega:

The screenshot shows the Insomnia REST Client interface. A GET request is made to `./BASE_PATH/course`. The response is a 200 OK status with a response time of 125 ms and a body size of 683 B. The response content is a JSON object representing paginated course data:

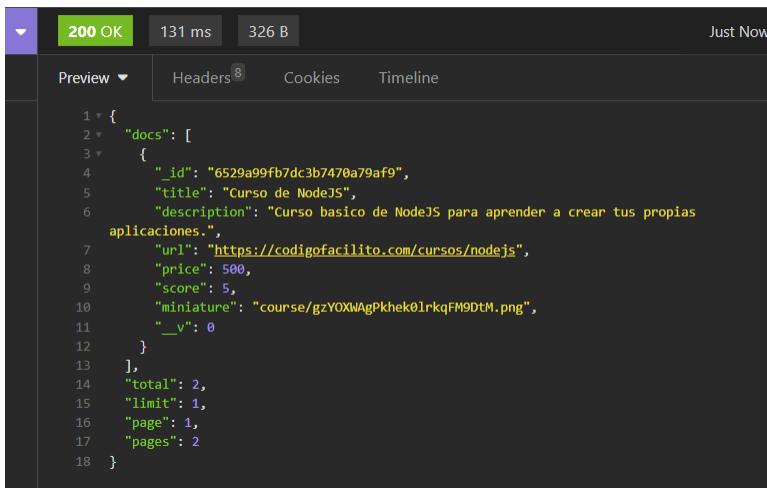
```
{
  "docs": [
    {
      "_id": "6529a09fb7dc3b7470a79af9",
      "title": "Curso de NodeJS",
      "description": "Curso basico de NodeJS para aprender a crear tus propias aplicaciones.",
      "url": "https://codigofacilito.com/cursos/nodejs",
      "price": 500,
      "score": 5,
      "miniature": "course/gzYOXWAgPkhk0lrkqFM9DtM.png",
      "_v": 0
    },
    {
      "_id": "6529aac2b7dc3b7470a79afb",
      "title": "Aprende Node.js y Express - Curso desde cero",
      "description": "¡Hola! Si quieres aprender Node.js y Express, estás en el lugar correcto.",
      "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
      "price": 300,
      "score": 5,
      "miniature": "course/oNzXljf2P0sLw8tW0DdTVj3x.png",
      "_v": 0
    }
  ],
  "total": 2,
  "limit": 10,
  "page": 1,
  "pages": 1
}
```

Ahora podemos observar cómo nos llega mucho mas organizado los datos y que al finalizar tenemos el total de cursos, el limite de la paginación, la pagina y la cantidad de páginas.

Vamos a modificar un poco el funcionamiento de la siguiente manera:

```
22  async function getCourse(req, res) {
23
24    const options = [
25      page: 1,
26      limit: 1,
27    ]
28
29    Course.paginate({}, options, (error, courses) => {
```

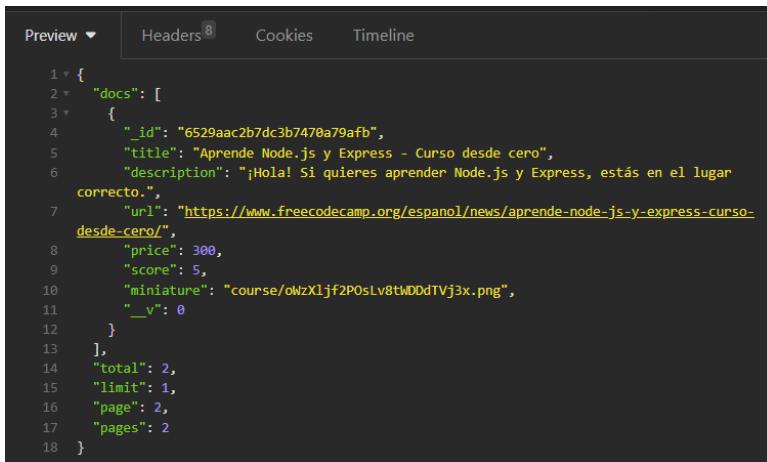
En insomnia nos saldría de la siguiente manera:



The screenshot shows the insomnia API client interface. At the top, it displays a green "200 OK" status, "131 ms" response time, and "326 B" size. To the right, it says "Just Now". Below this, there are tabs for "Preview", "Headers", "Cookies", and "Timeline". The "Preview" tab is selected and shows the following JSON response:

```
1 var {
2   "docs": [
3     {
4       "_id": "6529a99fb7dc3b7470a79af9",
5       "title": "Curso de NodeJS",
6       "description": "Curso básico de NodeJS para aprender a crear tus propias aplicaciones.",
7       "url": "https://codigofacilito.com/cursos/nodejs",
8       "price": 500,
9       "score": 5,
10      "miniature": "course/gzYOXWAgPkhek0lrkqFM9DtM.png",
11      "_v": 0
12    },
13  ],
14  "total": 2,
15  "limit": 1,
16  "page": 1,
17  "pages": 2
18 }
```

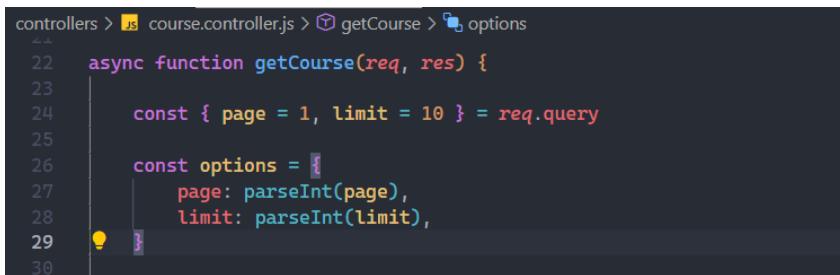
En una sola pagina la información, a hora colocamos la siguiente página:



The screenshot shows the insomnia API client interface, similar to the previous one. It displays a green "200 OK" status, "131 ms" response time, and "326 B" size. The "Preview" tab is selected, showing the same JSON response as the first screenshot:

```
1 var {
2   "docs": [
3     {
4       "_id": "6529aac2b7dc3b7470a79afb",
5       "title": "Aprende Node.js y Express - Curso desde cero",
6       "description": "¡Hola! Si quieres aprender Node.js y Express, estás en el lugar correcto.",
7       "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
8       "price": 300,
9       "score": 5,
10      "miniature": "course/oWzXljf2P0sLv8tWDDdTVj3x.png",
11      "_v": 0
12    },
13  ],
14  "total": 2,
15  "limit": 1,
16  "page": 2,
17  "pages": 2
18 }
```

De esta manera podemos modificar nuestra paginación a través de la dependencia, ahora vamos a modificar este proceso ya que debemos tener en cuenta que estos datos de paginación los debe controlar el usuario y los vamos a dejar como variables, vamos a hacer la siguiente modificación:



The screenshot shows a code editor with the file "course.controller.js" open. The code defines an asynchronous function "getCourse" that takes "req" and "res" as parameters. Inside the function, it retrieves query parameters "page" and "limit" and stores them in a new object "options". The code looks like this:

```
controllers > course.controller.js > getCourse > options
1 var {
2   "docs": [
3     {
4       "_id": "6529aac2b7dc3b7470a79afb",
5       "title": "Aprende Node.js y Express - Curso desde cero",
6       "description": "¡Hola! Si quieres aprender Node.js y Express, estás en el lugar correcto.",
7       "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
8       "price": 300,
9       "score": 5,
10      "miniature": "course/oWzXljf2P0sLv8tWDDdTVj3x.png",
11      "_v": 0
12    },
13  ],
14  "total": 2,
15  "limit": 1,
16  "page": 2,
17  "pages": 2
18 }
```

Establecemos en un objeto nuestras variables y modificamos la const options a través de nuestras nuevas variables. Ahora en nuestro insomnia configuramos de la siguiente manera la ruta:



GET `_.BASE_PATH`/course?page=1&limit=2

Send **200 OK** 119 ms 68 B

Body	Auth	Query	Headers 1	Docs	Preview	Headers 8	Cookies	Timeline
------	------	-------	-----------	------	---------	-----------	---------	----------

```

1 * {
2 >   "docs": [ ↵ 2 ↵ ],
24   "total": 2,
25   "limit": 2,
26   "page": 1,
27   "pages": 1
28 }

```

Nos muestra que tenemos 2 docs y que tenemos un limite de 2 y estoy en la página 1, si nosotros modificamos de nuevo en insomnia:

GET `_.BASE_PATH`/course?page=1&limit=1

Send **200 OK** 98.1 ms 326 B

Body	Auth	Query	Headers 1	Docs	Preview	Headers 8	Cookies	Timeline
------	------	-------	-----------	------	---------	-----------	---------	----------

```

1 * {
2 >   "docs": [ ↵ 1 ↵ ],
14   "total": 2,
15   "limit": 1,
16   "page": 1,
17   "pages": 2
18 }

```

Y si le digo que me voy a la pagina 2:

GET `_.BASE_PATH`/course?page=2&limit=1

Send **200 OK** 98.8 ms 405 B Just Now

Body	Auth	Query	Headers 1	Docs	Preview	Headers 8	Cookies	Timeline
------	------	-------	-----------	------	---------	-----------	---------	----------



```

1 * {
2 >   "docs": [
3 >     {
4       "_id": "6529aac2b7dc3b7470a79af",
5       "title": "Aprende Node.js y Express - Curso desde cero",
6       "description": "¡Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto.",
7       "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
8       "price": 300,
9       "score": 5,
10      "miniature": "course/owzX1jf2P0sLv8tMDdTVj3x.png",
11      "__v": 0
12    },
13  ],
14  "total": 2,
15  "limit": 1,
16  "page": 2,
17  "pages": 2
18 }

```

De esta manera ya tenemos implementado nuestro sistema de paginación ya creado con la obtención de cursos.



@hdtoledo

ACTUALIZANDO CURSOS

Vamos a realizar la actualización de nuestros cursos para ello creamos dentro de **course.controller.js** nuestra función **updateCourse**:

```
controllers > course.controller.js > updateCourse
41  async function updateCourse(req, res) {
42      const { id } = req.params
43      const courseData = req.body
44
45      if (req.files.miniature) {
46          const imagePath = image.getFilePath(req.files.miniature)
47          courseData.miniature = imagePath
48      }
49
50      Course.findByIdAndUpdate({ _id: id }, courseData, (error) => {
51          if (error) {
52              res.status(400).send({msg: "Error al actualizar el curso"})
53          } else {
54              res.status(200).send({msg: "Actualizacion correcta"})
55          }
56      })
57  }
58
59  module.exports = {
60      createCourse,
61      getCourse,
62      updateCourse
63 }
```

Como observamos es muy similar a la función utilizada con nuestro usuario, cambiando una pequeña condición en donde preguntamos si la imagen no se actualiza que se mantenga con la misma url, de resto la función actúa de manera muy similar a lo que venimos trabajando, nos dirigimos a nuestro **course.router.js** y cargamos la ruta:

```
router > course.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const CourseController = require("../controllers/course.controller")
4  const md_auth = require("../middlewares/authenticated")
5  const md_upload = multiparty({ uploadDir: "./uploads/course"})
6
7
8  const api = express.Router()
9
10 api.post("/course", [md_auth.asureAuth, md_upload], CourseController.createCourse)
11 api.get("/course", CourseController.getCourse)
12 api.patch("/course/:id", [md_auth.asureAuth, md_upload], CourseController.updateCourse)
13
14
15 module.exports = api
```



@hdtoledo

Para realizar la prueba vamos a crear otro curso en donde este mal su título:

The screenshot shows a POST request to `./BASE_PATH/course`. The response status is **201 Created** with a response time of 148 ms and a body size of 364 B. The Headers tab shows the following headers:
Content-Type: application/json
Content-Length: 260
Date: Mon, 10 Dec 2018 14:44:00 GMT
Server: Apache/2.4.29 (Ubuntu)
X-Powered-By: PHP/7.2.10-0ubuntu0.18.04.1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
The body of the response is a JSON object representing the newly created course:

```
1 * {
2     "title": "Python para prin",
3     "description": "Introducción a Python Aprenda a crear programas y proyectos en Python. Trabaje con cadenas, listas, bucles, diccionarios y funciones.",
4     "url": "https://learn.microsoft.com/es-es/training/paths/beginner-python/",
5     "price": 0,
6     "score": 5,
7     "_id": "652abc994001b82a70f44eb9",
8     "miniature": "course/IouWRYENLCI-j30gGuRQFl6k.png",
9     "__v": 0
10 }
```

Ahora vamos a probarlo en nuestro insomnia de la siguiente manera creando la ruta para updateCourse y colocando el id de nuestro nuevo curso:

The screenshot shows a PATCH request to `./BASE_PATH/course/652abc994001b82a70f44eb9`. The response status is **200 OK** with a response time of 137 ms and a body size of 32 B. The Headers tab shows the following headers:
Content-Type: application/json
Content-Length: 260
Date: Mon, 10 Dec 2018 14:44:00 GMT
Server: Apache/2.4.29 (Ubuntu)
X-Powered-By: PHP/7.2.10-0ubuntu0.18.04.1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
The body of the response is a JSON object with a message:

```
1 * {
2     "msg": "Actualizacion correcta"
3 }
```

Ahora vamos a verificar si es correcto:

The screenshot shows a GET request to `./BASE_PATH/course?page=1&limit=10`. The response status is **200 OK** with a response time of 131 ms and a body size of 1067 B. The Headers tab shows the following headers:
Content-Type: application/json
Content-Length: 1067
Date: Mon, 10 Dec 2018 14:44:00 GMT
Server: Apache/2.4.29 (Ubuntu)
X-Powered-By: PHP/7.2.10-0ubuntu0.18.04.1
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
The body of the response is a JSON array of courses, including the updated one:

```
1 * [
2     {
3         "title": "Python para principiantes - Nivel 0",
4         "description": "¡Hola! Si quieres aprender Node.js y Express, est\u00e1s en el lugar correcto.",
5         "url": "https://www.freecodecamp.org/espanol/cursos/python-principiantes-nivel-0",
6         "price": 0,
7         "score": 5,
8         "miniature": "course/gzYOWAgPkhek0lrkqFM9DtM.png",
9         "__v": 0
10     },
11     {
12         "title": "Aprende Node.js y Express - Curso desde cero",
13         "description": "¡Hola! Si quieres aprender Node.js y Express, est\u00e1s en el lugar correcto.",
14         "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
15         "price": 300,
16         "score": 5,
17         "miniature": "course/oWzXljf2POsLv8tWDDdTj3x.png",
18         "__v": 0
19     },
20     {
21         "title": "Python para principiantes - Nivel 0",
22         "description": "Introducci\u00f3n a Python Aprenda a crear programas y proyectos en Python. Trabaje con cadenas, listas, bucles, diccionarios y funciones.",
23         "url": "https://learn.microsoft.com/es-es/training/paths/beginner-python/",
24         "price": 0,
25         "score": 5,
26         "miniature": "course/IouWRYENLCI-j30gGuRQFl6k.png",
27         "__v": 0
28     }
29 ],
30 {
31     "total": 3
32 }
```

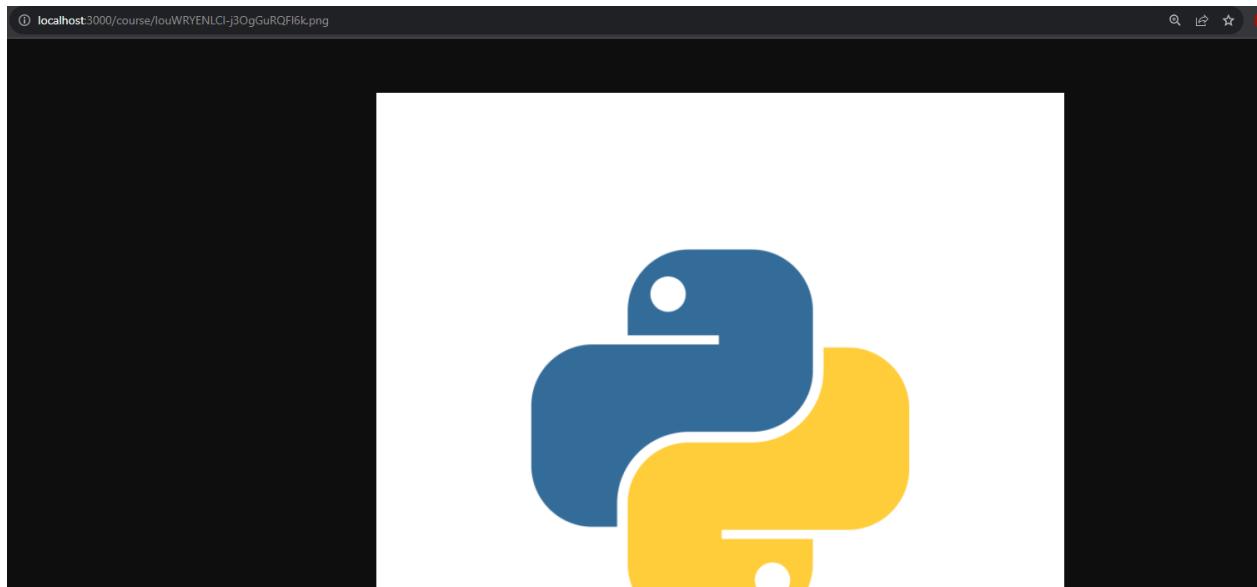


@hDTOLEDO

Y en mongodb:

```
_id: ObjectId('652abc994001b82a70f44eb9')
title: "Python para principiantes - Nivel 0"
description: "Introducción a Python Aprenda a crear programas y proyectos en Python..."
url: "https://learn.microsoft.com/es-es/training/patterns/beginner-python/"
price: 0
score: 5
miniature: "course/IouWRYENLCI-j30gGuRQFl6k.png"
__v: 0
```

Ahora vamos a comprobar si la url de nuestros cursos funcionan, copiamos la url y la colocamos en nuestro navegador y verificamos:



De esta manera estamos verificando que nuestra función de actualizar ha sido correctamente creada.

ELIMINAR CURSOS

Ahora vamos a seguir con la función de **deleteCourse** dentro de nuestro **course.controller.js**:

```
controllers > js course.controller.js > 📁 <unknown>
~~
59  √  async function deleteCourse(req, res) {
60      const { id } = req.params
61
62  √      Course.findByIdAndDelete(id, (error) => {
63      if (error) {
64          res.status(400).send({msg: "Error al eliminar el curso"})
65  √      } else {
66          res.status(200).send({msg: "Curso Eliminado"})
67      }
68  √  })
69  }
70
71  √  module.exports = {
72      createCourse,
73      getCourse,
74      updateCourse,
75      deleteCourse,
76  }

```

Como observamos la función de eliminación es similar a la de usuario solo que acá lo hacemos con el id de cursos, ahora vamos a crear nuestra ruta en **course.router.js**:

```
router > js course.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const CourseController = require("../controllers/course.controller")
4  const md_auth = require("../middlewares/authenticated")
5  const md_upload = multiparty({ uploadDir: "./uploads/course"})
6
7
8  const api = express.Router()
9
10 api.post("/course", [md_auth.asureAuth, md_upload], CourseController.createCourse)
11 api.get("/course", CourseController.getCourse)
12 api.patch("/course/:id", [md_auth.asureAuth, md_upload], CourseController.updateCourse)
13 api.delete("/course/:id", [md_auth.asureAuth], CourseController.deleteCourse)
14
15
16  module.exports = api

```

Ahora vamos a comprobar que funciona para ello vamos a crear un curso nuevo en insomnia:



```

1 + {
2   "title": "Academia de profesores de Minecraft: Education",
3   "description": "Aprendizaje de Minecraft Education",
4   "url": "https://learn.microsoft.com/es-es/training/paths/minecraft-teacher-academy/",
5   "price": 0,
6   "score": 5,
7   "_id": "652ac04d79651500110b1010",
8   "miniature": "course/Si8EYfeuQw54MjYMo570FAZk.png",
9   "__v": 0
10 }

```

Ahora creamos una nueva ruta para nuestro deleteCourse en insomnia:

```

1 + {
2   "msg": "Curso Eliminado"
3 }

```

Verificamos en mongodb e insomnia:

```

1 + [
2   {
3     "_id": "652aaac2b7dc3b7478a79af9",
4     "title": "Curso de NodeJS",
5     "description": "Curso basico de NodeJS para aprender a crear tus propias aplicaciones.",
6     "url": "https://codigofacilito.com/cursos/nodejs",
7     "price": 500,
8     "score": 5,
9     "miniature": "course/gzYOWAgPikeh@LrkqFM90tN.png",
10    "__v": 0
11  },
12  {
13    "_id": "652aaac2b7dc3b7478a79af9",
14    "title": "Aprende Node.js y Express - Curso desde cero",
15    "description": "¡Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto.",
16    "url": "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/",
17    "price": 300,
18    "score": 5,
19    "miniature": "course/oMzX1jf2POsLv8tM00tVj3x.png",
20    "__v": 0
21  },
22  {
23    "_id": "652abc9949001b82a7ef44eb9",
24    "title": "Python para principiantes - Nivel 9",
25    "description": "Introducción a Python Aprenda a crear programas y proyectos en Python. Trabaje con cadenas, listas, bucles, diccionarios y funciones.",
26    "url": "https://learn.microsoft.com/es-es/training/paths/beginner-python/",
27    "price": 0,
28    "score": 5,
29    "miniature": "course/InslHRYENLCl-j30gGuQF16k.png",
30    "__v": 5
31  }
32 ],
33 {
34   "total": 3,
35   "limit": 10,
36   "page": 1,
37   "pages": 1
38 }

```

```

1 + ObjectId("652aaac2b7dc3b7478a79af9")
title: "Curso de NodeJS"
description: "Curso basico de NodeJS para aprender a crear tus propias aplicaciones."
url: "https://codigofacilito.com/cursos/nodejs"
price: 500
score: 5
miniature: "course/gzYOWAgPikeh@LrkqFM90tN.png"
__v: 0

1 + ObjectId("652aaac2b7dc3b7478a79af9")
title: "Aprende Node.js y Express - Curso desde cero"
description: "¡Hola! Si quieras aprender Node.js y Express, estás en el lugar correcto."
url: "https://www.freecodecamp.org/espanol/news/aprende-node-js-y-express-curso-desde-cero/"
price: 300
score: 5
miniature: "course/oMzX1jf2POsLv8tM00tVj3x.png"
__v: 0

1 + ObjectId("652abc9949001b82a7ef44eb9")
title: "Python para principiantes - Nivel 9"
description: "Introducción a Python Aprenda a crear programas y proyectos en Python. Trabaje con cadenas, listas, bucles, diccionarios y funciones."
url: "https://learn.microsoft.com/es-es/training/paths/beginner-python/"
price: 0
score: 5
miniature: "course/InslHRYENLCl-j30gGuQF16k.png"
__v: 5

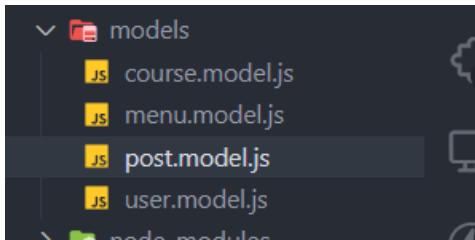
```

De esta manera ya terminamos nuestra función de eliminación.



MODELO POST PARA EL BLOG

Ahora vamos a desarrollar el modelo de datos para nuestro blog de la aplicación, lo primero es crear nuestro modelo **post.model.js** dentro de **models**



Ahora dejamos de la siguiente manera nuestro modelo:

```
models > post.model.js > <unknown>
1  const mongoose = require("mongoose")
2  const mongoosePaginate = require("mongoose-paginate")
3
4  const PostSchema = mongoose.Schema({
5      title: String,
6      miniature: String,
7      content: String,
8      path: {
9          type: String,
10         unique: true,
11     },
12     created_at: Date,
13 })
14
15 PostSchema.plugin(mongoosePaginate)
16
17 module.exports = mongoose.model("Post", PostSchema)
```

Dentro de nuestro modelo dejamos implementado el sistema de paginación ya que tendremos una gran cantidad de post en esta sección. Revisamos en nuestro servidor que todo vaya bien sin errores:

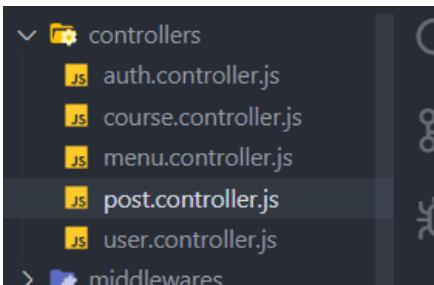
```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
```



@hdtoledo

ESTRUCTURA API BLOG

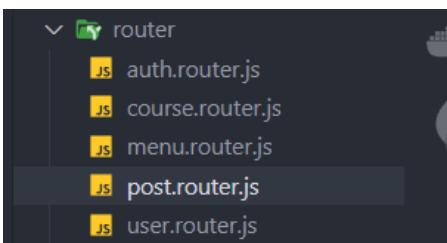
Vamos a crear nuestro controlador **post.controller.js** dentro de **controllers**:



Y dejamos de la siguiente manera nuestra estructura inicial:

```
controllers > post.controller.js > <unknown>
1 const Post = require("../models/post.model")
2
3 //Funciones ...
4
5
6 module.exports = [
7   |
8 ]
```

Ahora creamos nuestra ruta **post.router.js**:



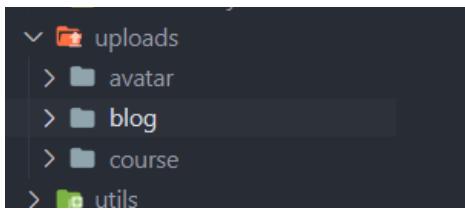
Y lo dejamos con la siguiente estructura:

```
router > post.router.js > ...
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const PostController = require("../controllers/post.controller")
4 const md_auth = require("../middlewares/authenticated")
5
6 const md_upload = multiparty({ uploadDir: "./uploads/blog" })
7 const api = express.Router()
8
9 //Rutas ...
10
11
12 module.exports = api
```



@hdtoledo

Ahora creamos nuestra carpeta para **blog** dentro de **uploads**:



Y por último configuramos nuestro **app.js** importamos nuestra ruta:

```
js app.js > postRoutes
 9 // Importar rutas
10 const authRoutes = require("./router/auth.router")
11 const userRoutes = require("./router/user.router")
12 const menuRoutes = require("./router/menu.router")
13 const courseRoutes = require("./router/course.router")
14 const postRoutes = require("./router/post.router")
15
```

Y configuramos la ruta:

```
js app.js > ...
25
26 // Configurar Rutas
27 app.use(`/api/${API_VERSION}`, authRoutes)
28 app.use(`/api/${API_VERSION}`, userRoutes)
29 app.use(`/api/${API_VERSION}`, menuRoutes)
30 app.use(`/api/${API_VERSION}`, courseRoutes)
31 app.use(`/api/${API_VERSION}`, postRoutes)
32
33
34 module.exports = app
```

Revisamos que nuestro servidor este correctamente funcionando:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
|
```

Y aquí ya dejamos nuestra configuración inicial.



@hdtoledo

CREACION DE POST

Ahora vamos a continuar con la función de creación del post, nos dirigimos a **post.controller.js** y vamos a crear nuestra función **createPost**:

```
controllers > js post.controller.js > ...
1  const Post = require("../models/post.model")
2  const image = require("../utils/image")
3
4  function createPost(req, res) {
5      const post = new Post(req.body)
6      post.created_at = new Date()
7
8      const imagePath = image.getFilePath(req.files.miniature)
9      post.miniature = imagePath
10
11     post.save((error, postStored) => {
12         if (error) {
13             res.status(400).send({msg: "Error al crear el post"})
14         } else {
15             res.status(201).send(postStored)
16         }
17     })
18 }
19
20 module.exports = {
21     createPost,
22 }
```

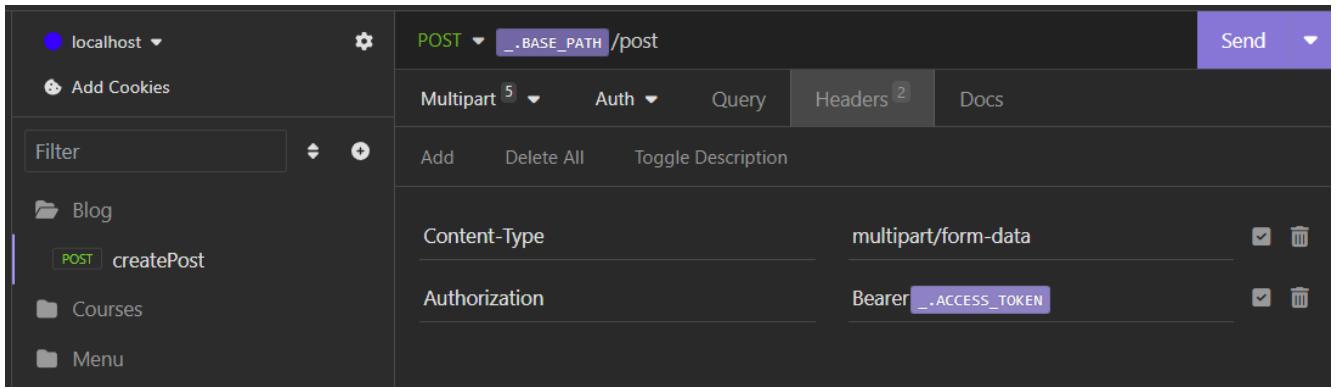
Creamos nuestra función del post y definimos path de imagen y fecha de publicación, ahora vamos a crear nuestra ruta dentro de **post.router.js**:

```
router > js post.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const PostController = require("../controllers/post.controller")
4  const md_auth = require("../middlewares/authenticated")
5
6  const md_upload = multiparty({ uploadDir: "./uploads/blog" })
7  const api = express.Router()
8
9
10 api.post("/post", [md_auth.ensureAuth, md_upload], PostController.createPost)
11
12
13 module.exports = api
```



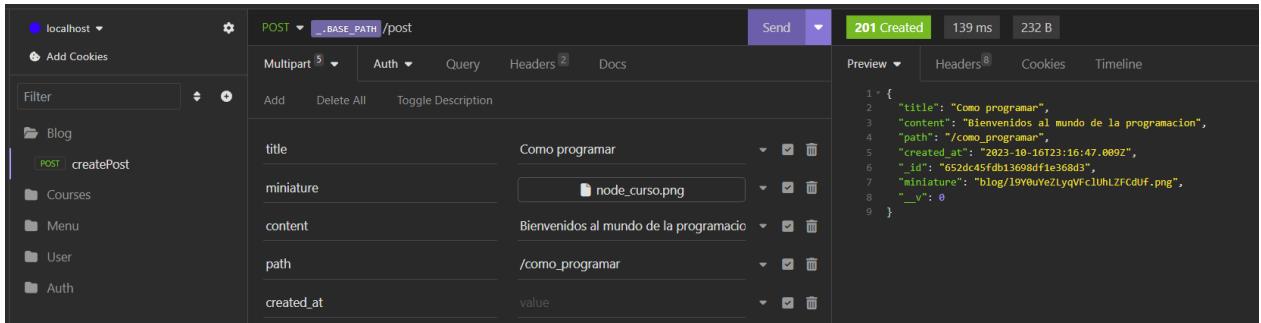
@hdtoledo

Ahora en insomnia vamos a crear una carpeta nueva con su ruta:



The screenshot shows the insomnia REST client interface. On the left, there's a sidebar with a tree view: 'Blog' (selected), 'Courses', 'Menu', and a 'POST' entry for 'createPost'. The main area shows a POST request to '_BASE_PATH/post'. The 'Headers' tab is selected, containing 'Content-Type: multipart/form-data' and 'Authorization: Bearer _ACCESS_TOKEN'. The 'Send' button is at the top right.

Recordemos pasar la configuración de nuestro base path y en los headers aplicamos auth y Access token, ahora aplicamos la configuración de multipart:

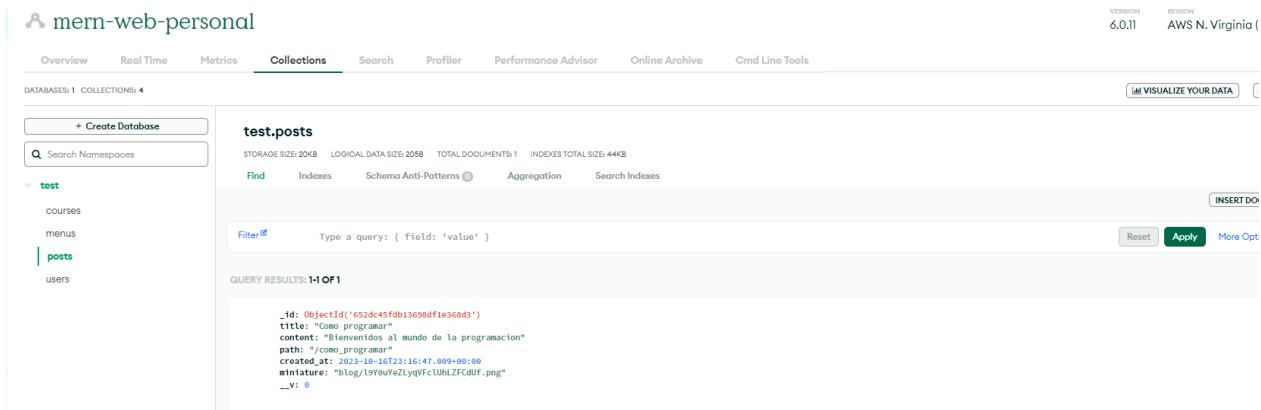


The screenshot shows the insomnia REST client interface after sending the POST request. The status bar indicates '201 Created', '139 ms', and '232 B'. The 'Preview' tab shows the JSON response body:

```
1 {  
2   "title": "Como programar",  
3   "content": "Bienvenidos al mundo de la programacion",  
4   "path": "/como_programar",  
5   "created_at": "2023-10-16T23:16:47.089Z",  
6   "_id": "652dc45fd13698df1e368d3",  
7   "miniature": "blog/19y0uYeZLyqVFclUhlZFCduF.png",  
8   "__v": 0  
9 }
```

Nota: al colocar la ruta del **path** no debemos colocar el **slash "/"**.

Al enviarlo nos debe devolver nuestro objeto, vamos a verificarlo en mongodb y tambien vamos a verificar copiando y pegando la dirección de la miniatura en nuestro navegador:

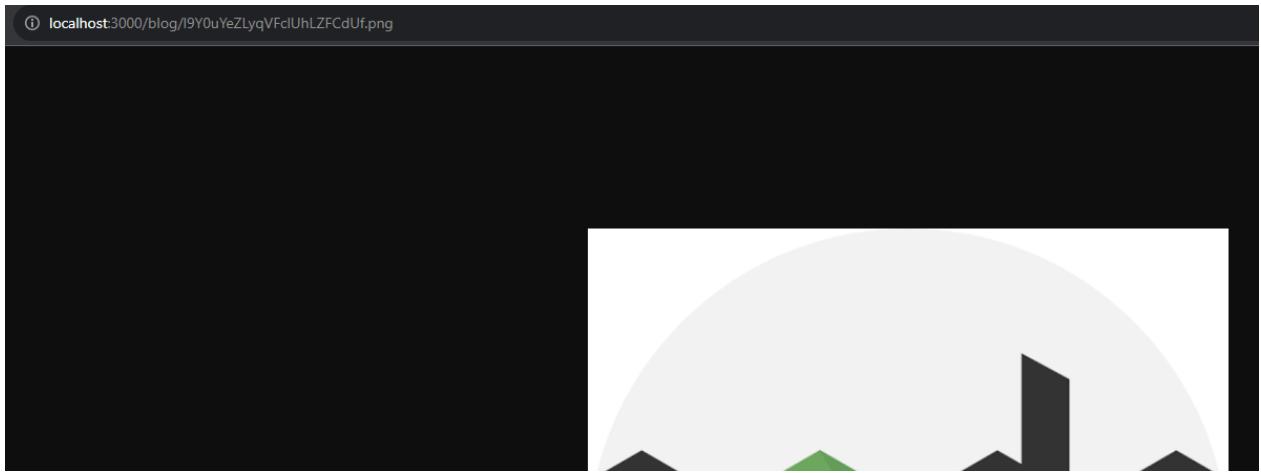


The screenshot shows the MongoDB Compass interface. On the left, the database 'mern-web-personal' is selected, with the 'test' database expanded to show 'posts' and 'users' collections. The 'Collections' tab is active, showing the 'posts' collection with 1 document. The document details are:

```
_id: ObjectId('652dc45fd13698df1e368d3')  
title: "Como programar"  
content: "Bienvenidos al mundo de la programacion"  
path: "/como_programar"  
created_at: 2023-10-16T23:16:47.089Z  
miniature: "blog/19y0uYeZLyqVFclUhlZFCduF.png"  
__v: 0
```



@hdtoledo



Vamos a dejar cargada una publicación más como ejercicio:

POST `"/post"`

Multipart	Auth	Query	Headers	Docs	Send	201 Created	129 ms	232 B		
Add	Delete All	Toggle Description				Preview	Headers	Cookies	Timeline	
title	Como programar nivel 2					<pre>1 + { 2 "title": "Como programar nivel 2", 3 "content": "El mundo de la programacion 2", 4 "path": "/como_programar_2", 5 "created_at": "2023-10-16T23:22:32.225Z", 6 "_id": "652dc5b8db13698df1e368d5", 7 "miniature": "blog/n10bMyGbjLeHr8xSM1GiwEP.png", 8 "__v": 0 9 }</pre>				
miniature	phyton.png									
content	El mundo de la programacion 2									
path	/como_programar_2									
created_at	value									

Filter Type a query: { field: 'value' } Reset Apply More

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('652dc45fd8db13698df1e368d3')
title: "Como programar"
content: "Bienvenidos al mundo de la programacion"
path: "/como_programar"
created_at: 2023-10-16T23:16:47.009+00:00
miniature: "blog/l9Y0uYeZLyqVFclUhLZFCdUf.png"
__v: 0

_id: ObjectId('652dc5b8db13698df1e368d5')
title: "Como programar nivel 2"
content: "El mundo de la programacion 2"
path: "/como_programar_2"
created_at: 2023-10-16T23:22:32.225+00:00
miniature: "blog/n10bMyGbjLeHr8xSM1GiwEP.png"
__v: 0
```

De esta manera comprobamos y dejamos lista nuestra función de crear post.



OBTENCION Y PAGINACION DE LOS POST

Vamos a crear nuestra función `getPosts` dentro de `post.controller.js`:

```
controllers > js post.controller.js > ...
20
21     function getPosts(req, res) {
22         const { page = 1, limit = 10 } = req.query
23
24         const options = {
25             page: parseInt(page),
26             limit: parseInt(limit),
27             sort: { created_at: "desc" }
28         }
29
30         Post.paginate({}, options, (error, postsStored) => {
31             if (error) {
32                 res.status(400).send({msg: "Error al obtener los post"})
33             } else {
34                 res.status(200).send(postsStored)
35             }
36         })
37     }
38
39
40     module.exports = {
41         createPost,
42         getPosts,
43     }

```

De esta manera aplicamos de manera similar nuestra paginación dentro de la obtención del post, ahora vamos a crear la ruta dentro de `post.router.js`:

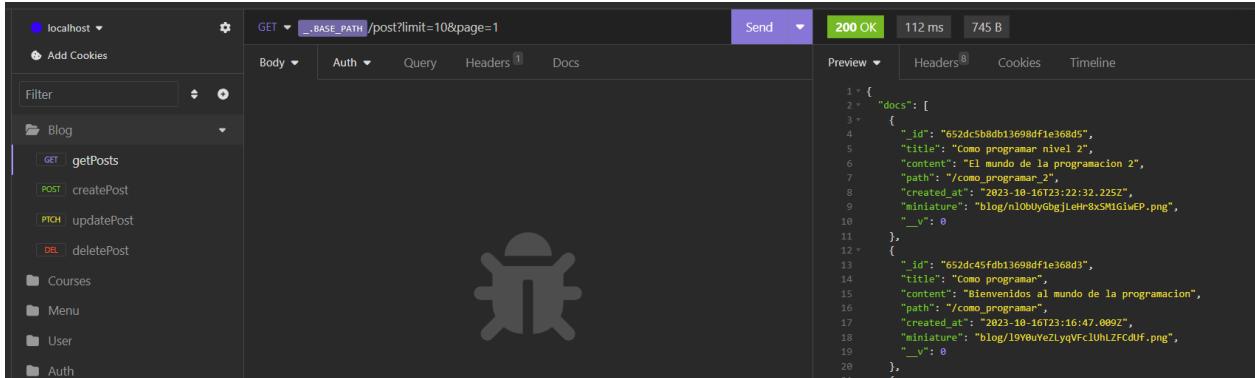
```
router > js post.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const PostController = require("../controllers/post.controller")
4  const md_auth = require("../middlewares/authenticated")
5
6  const md_upload = multiparty({ uploadDir: "./uploads/blog" })
7  const api = express.Router()
8
9
10 api.post("/post", [md_auth.asureAuth, md_upload], PostController.createPost)
11 api.get("/post", PostController.getPosts)
12
13
14 module.exports = api

```



@hdtoledo

Recordemos que es un **get** y debe ser publico así que ahora nos vamos a insomnia y creamos la ruta y probamos:



```
1: {
2:   "docs": [
3:     {
4:       "_id": "652dc5b8db13698df1e368d5",
5:       "title": "Como programar nivel 2",
6:       "content": "El mundo de la programacion 2",
7:       "path": "/como_programar_2",
8:       "created_at": "2023-10-16T23:22:32.225Z",
9:       "miniature": "blog/n10bUygbgjLehr8xSM1GiwEP.png",
10:      "__v": 6
11:    },
12:    {
13:       "_id": "652dc45fd8b13698df1e368d3",
14:       "title": "Como programan",
15:       "content": "Bienvenidos al mundo de la programación",
16:       "path": "/como_programan",
17:       "created_at": "2023-10-16T23:16:47.009Z",
18:       "miniature": "blog/19YeuYeZLyqVFclUhLZFcduf.png",
19:      "__v": 0
20:    }
21:  ]
22: }
```

De esta manera ya tenemos nuestra función completa.

ACTUALIZACION DE LOS POST

Vamos a crear nuestra función **updatePost** dentro de **post.controller.js**:

```
controllers > post.controller.js >  <unknown>
39  function updatePost (req, res) {
40    const { id } = req.params
41    const postData = req.body
42
43    if (req.files.miniature) {
44      const imagePath = image.getFilePath(req.files.miniature)
45      postData.miniature = imagePath
46    }
47
48    Post.findByIdAndUpdate({ _id: id }, postData, (error) => {
49      if (error) {
50        res.status(400).send({msg: "Error al actualizar el post"})
51      } else {
52        res.status(200).send({msg: "Actualizacion correcta"})
53      }
54    })
55  }
56
57  module.exports = {
58    createPost,
59    getPosts,
60    updatePost,
61  }
```

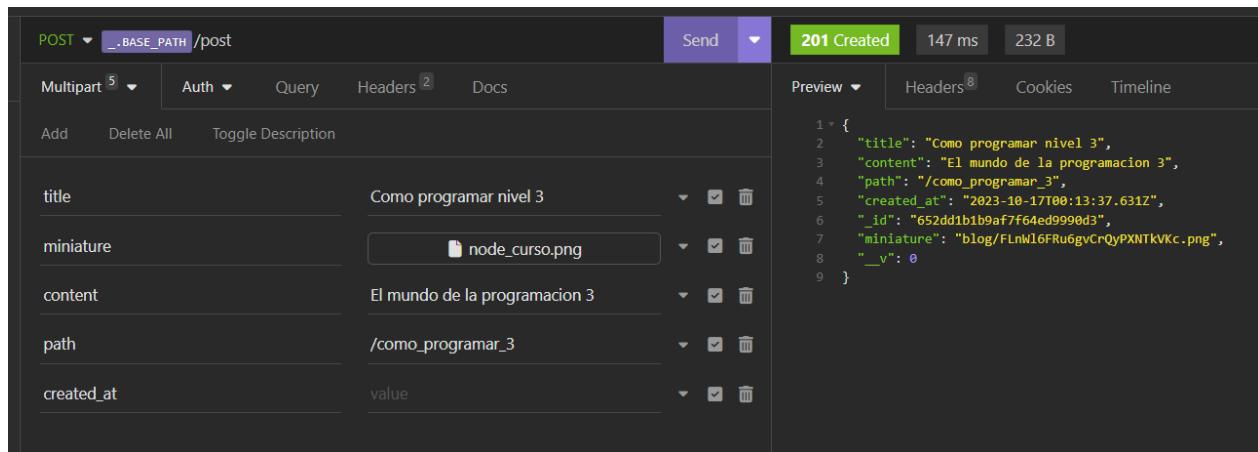


@hdtoledo

Nuestra función es muy similar a la que utilizamos anteriormente, ahora procedemos a crear la ruta dentro de **post.router.js**

```
router > js post.router.js > ...
1  const express = require("express")
2  const multiparty = require("connect-multiparty")
3  const PostController = require("../controllers/post.controller")
4  const md_auth = require("../middlewares/authenticated")
5
6  const md_upload = multiparty({ uploadDir: "./uploads/blog"})
7  const api = express.Router()
8
9
10 api.post("/post", [md_auth.asureAuth, md_upload], PostController.createPost)
11 api.get("/post", PostController.getPosts)
12 api.patch("/post/:id", [md_auth.asureAuth, md_upload], PostController.updatePost)
13
14
15 module.exports = api
```

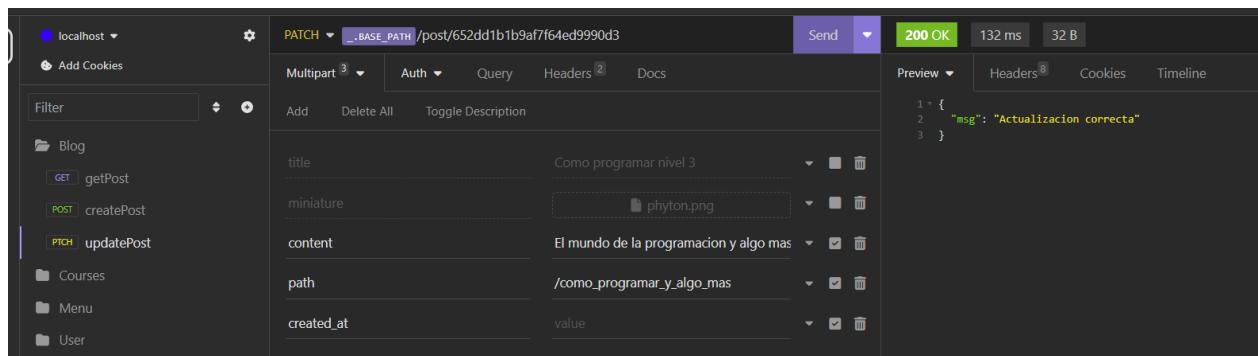
Ahora vamos a nuestro insomnia y creamos un nuevo post:



The screenshot shows the insomnia API client interface. A POST request is being made to `_.BASE_PATH /post`. The request body contains the following JSON payload:

```
1 < {
2   "title": "Como programar nivel 3",
3   "content": "El mundo de la programacion 3",
4   "path": "/como_programar_3",
5   "created_at": "2023-10-17T00:13:37.631Z",
6   "_id": "652dd1b1b0af7f64ed9990d3",
7   "miniature": "blog/FLnWl6FRu6gvCrQyPXNTkVKc.png",
8   "__v": 0
9 }
```

Vamos a utilizar el id de nuestro post en la ruta nueva:



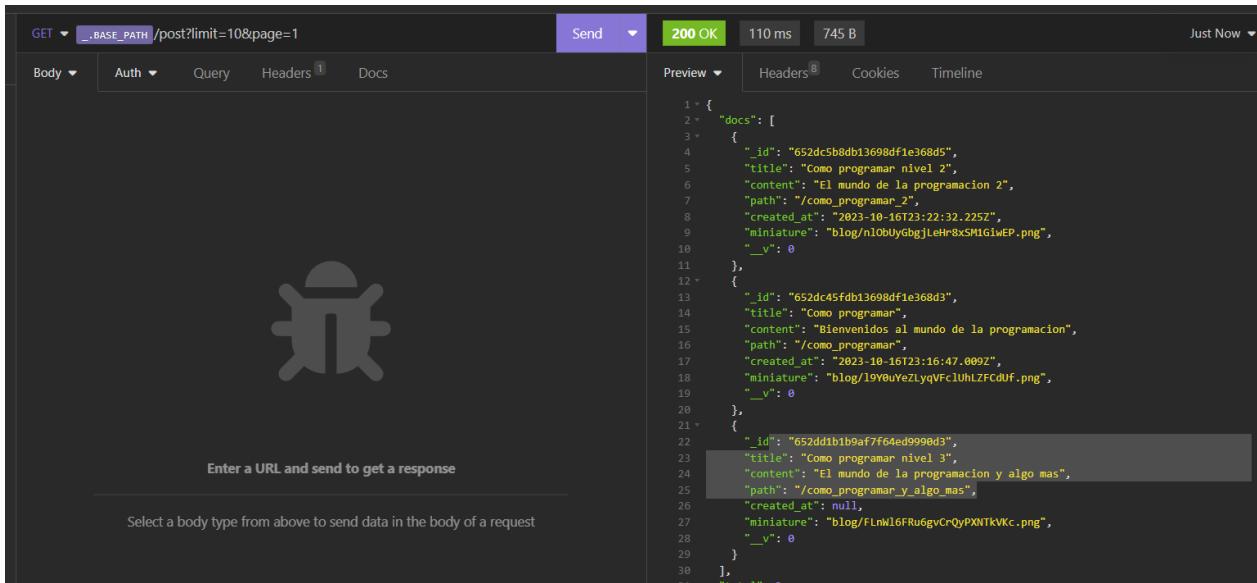
The screenshot shows the insomnia API client interface. A PATCH request is being made to `_.BASE_PATH /post/652dd1b1b0af7f64ed9990d3`. The request body contains the following JSON payload:

```
1 < {
2   "msg": "Actualizacion correcta"
3 }
```



@hdtoledo

Revisamos en nuestro `getPost`:



```
1+ {
2+   "docs": [
3+     {
4+       "_id": "652dc5b8db13698df1e368d5",
5+       "title": "Como programar nivel 2",
6+       "content": "El mundo de la programacion 2",
7+       "path": "/como_programar_2",
8+       "created_at": "2023-10-16T23:22:32.225Z",
9+       "miniature": "blog/nlObUyGbgjLeHr8xSM1GiwEP.png",
10+      "__v": 0
11+    },
12+    {
13+      "_id": "652dc45fdb13698df1e368d3",
14+      "title": "Como programar",
15+      "content": "Bienvenidos al mundo de la programacion",
16+      "path": "/como_programar",
17+      "created_at": "2023-10-16T23:16:47.009Z",
18+      "miniature": "blog/l9Y0uYeZlyqVFc1UhLZFcduF.png",
19+      "__v": 0
20+    },
21+    {
22+      "_id": "652dd1b1b9af7f64ed9990d3",
23+      "title": "Como programar nivel 3",
24+      "content": "El mundo de la programacion y algo mas",
25+      "path": "/como_programar_y_algo_mas",
26+      "created_at": null,
27+      "miniature": "blog/FlnWloFRu6gvCrQyPXNTkVkc.png",
28+      "__v": 0
29+    }
30+  ],
31+  "total": 3
32+ }
```

De esta manera ya dejamos nuestra función lista.

ELIMINAR POST

Vamos a crear nuestra función de eliminación, para ello nos ubicamos en `post.controller.js` y creamos la función `deletePost`:

```
controllers > post.controller.js > deletePost
56
57  function deletePost (req, res) {
58    const { id } = req.params
59
60    Post.findByIdAndDelete(id, (error) => {
61      if (error) {
62        res.status(400).send({msg: "Error al eliminar el post"})
63      } else {
64        res.status(200).send({msg: "Post Eliminado"})
65      }
66    })
67  }
68
69
70  module.exports = {
71    createPost,
72    getPosts,
73    updatePost,
74    deletePost
75  }
```

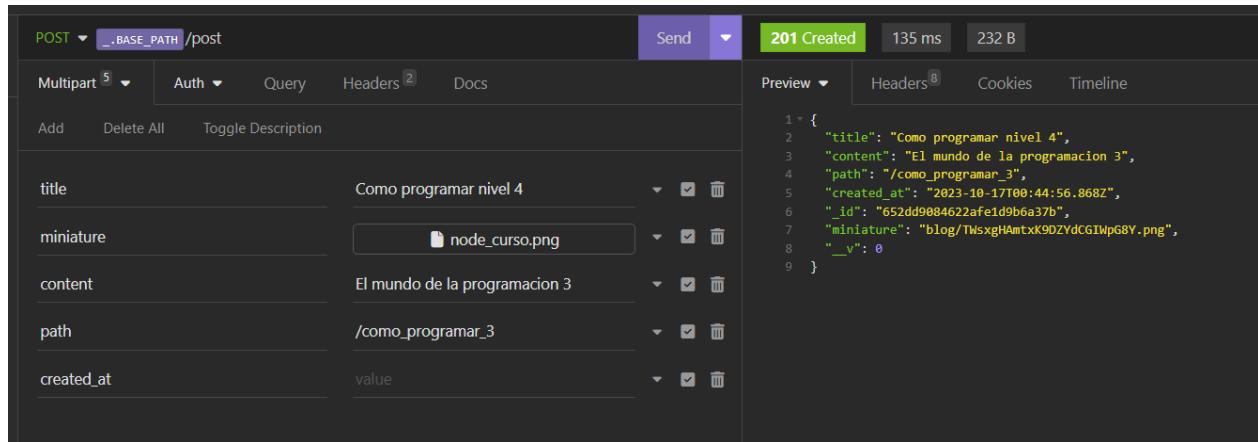


@hdtoledo

Nuestra función es muy similar a la que utilizamos anteriormente, ahora vamos a nuestro post.router.js y creamos la ruta:

```
router > js post.router.js > ...
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const PostController = require("../controllers/post.controller")
4 const md_auth = require("../middlewares/authenticated")
5
6 const md_upload = multiparty({ uploadDir: "./uploads/blog" })
7 const api = express.Router()
8
9
10 api.post("/post", [md_auth.asureAuth, md_upload], PostController.createPost)
11 api.get("/post", PostController.getPosts)
12 api.patch("/post/:id", [md_auth.asureAuth, md_upload], PostController.updatePost)
13 api.delete("/post/:id", [md_auth.asureAuth], PostController.deletePost)
14
15
16 module.exports = api
```

Ahora vamos a crear un post nuevo en insomnia:



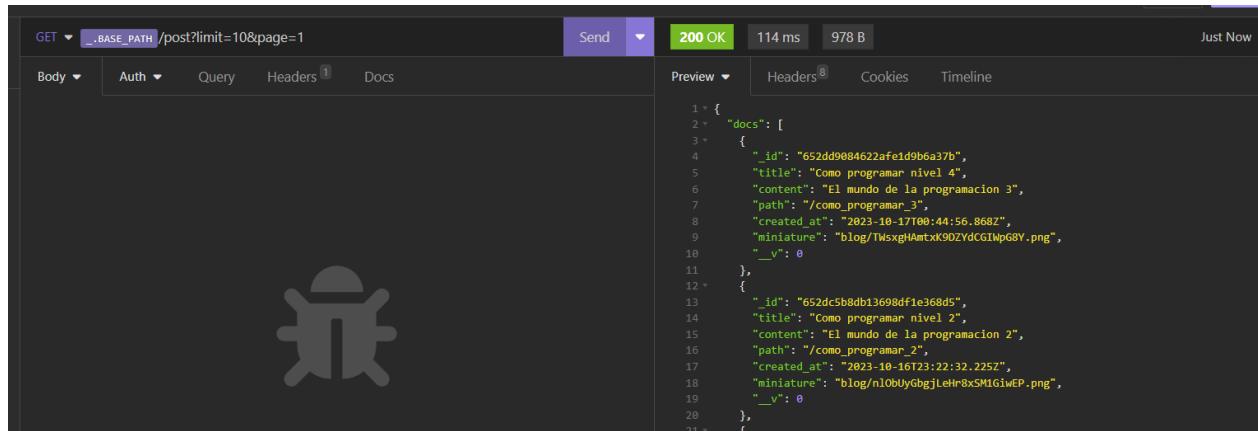
The screenshot shows the insomnia API client interface. A POST request is being made to `.BASE_PATH /post`. The request body contains the following fields:

- title**: Como programar nivel 4
- miniature**: node_curso.png
- content**: El mundo de la programacion 3
- path**: /como_programar_3
- created_at**: value

The response status is **201 Created**, with a response time of 135 ms and a response size of 232 B. The response body is displayed in the preview tab:

```
1 < {
2   "title": "Como programar nivel 4",
3   "content": "El mundo de la programacion 3",
4   "path": "/como_programar_3",
5   "created_at": "2023-10-17T00:44:56.868Z",
6   "_id": "652dd9084622afe1d9b6a37b",
7   "miniature": "blog/TNsxgHAmtxK9DZYdCGIwG8Y.png",
8   "__v": 0
9 }
```

Verificamos en `getPost`:

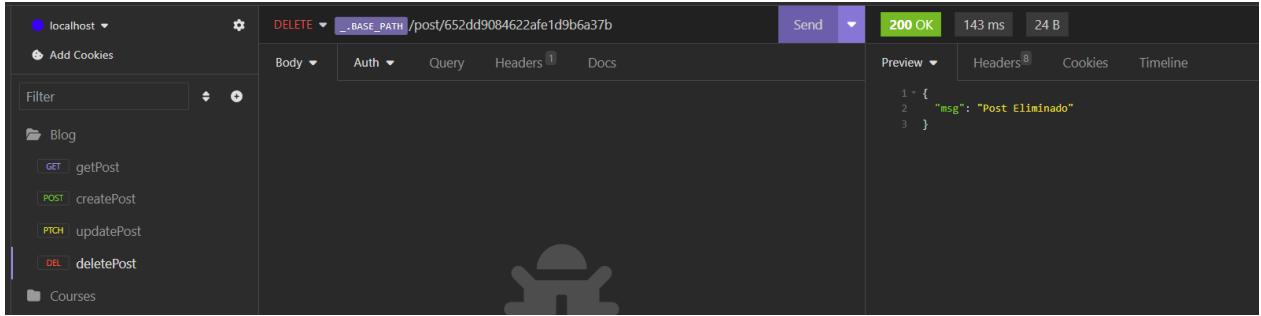


The screenshot shows the insomnia API client interface. A GET request is being made to `.BASE_PATH /post?limit=10&page=1`. The response status is **200 OK**, with a response time of 114 ms and a response size of 978 B. The response body is displayed in the preview tab:

```
1 < {
2   "docs": [
3     {
4       "_id": "652dd9084622afe1d9b6a37b",
5       "title": "Como programar nivel 4",
6       "content": "El mundo de la programacion 3",
7       "path": "/como_programar_3",
8       "created_at": "2023-10-17T00:44:56.868Z",
9       "miniature": "blog/TNsxgHAmtxK9DZYdCGIwG8Y.png",
10      "__v": 0
11    },
12    {
13      "_id": "652dc5b8db13698df1e368d5",
14      "title": "Como programar nivel 2",
15      "content": "El mundo de la programacion 2",
16      "path": "/como_programar_2",
17      "created_at": "2023-10-16T23:22:32.225Z",
18      "miniature": "blog/nlObUyGbgjLeHr8xSMIGiWEP.png",
19      "__v": 0
20    }
21 }
```

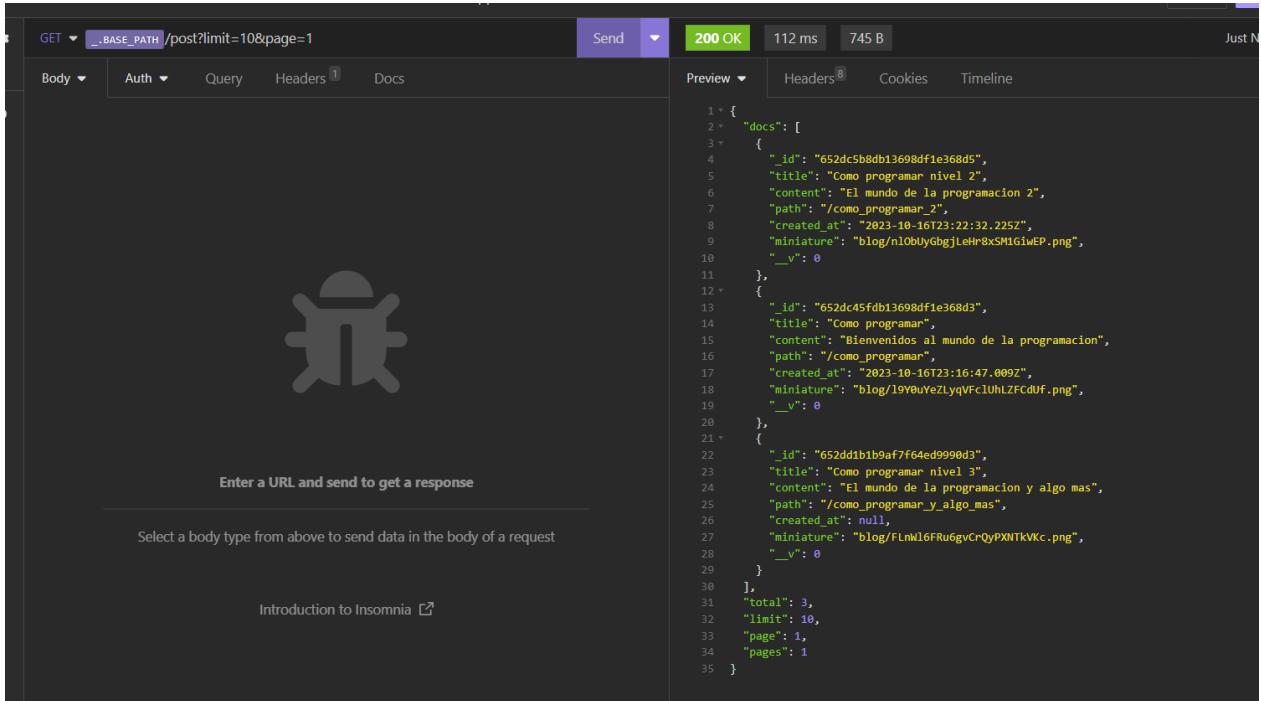


Tomamos el id de nuestro nuevo post y ahora creamos la ruta de delete en nuestro insomnia:



The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with a 'Blog' section containing 'getPost', 'createPost', 'updatePost', and 'deletePost' endpoints. The main area shows a 'DELETE' request to `__BASE_PATH/post/652dd9084622afe1d9b6a37b`. The response is a 200 OK status with a 'msg' key containing the value 'Post Eliminado'.

De esta manera comprobamos que se ha eliminado nuestro post, vamos a verificar en `getPost`:



The screenshot shows a 'GET' request to `__BASE_PATH/post?limit=10&page=1`. The response is a 200 OK status with a JSON body containing a list of three posts. Each post includes fields like _id, title, content, path, created_at, and miniature.

```
1 {  
2   "docs": [  
3     {  
4       "_id": "652dc5b0db13698df1e368d5",  
5       "title": "Como programar nivel 2",  
6       "content": "El mundo de la programacion 2",  
7       "path": "/como_programar_2",  
8       "created_at": "2023-10-16T23:22:32.225Z",  
9       "miniature": "blog/n10blyGhgjLeHr8x5MIGiweP.png",  
10      "__v": 0  
11    },  
12    {  
13      "_id": "652dc45fdb13698df1e368d3",  
14      "title": "Como programar",  
15      "content": "Bienvenidos al mundo de la programacion",  
16      "path": "/como_programar",  
17      "created_at": "2023-10-16T23:16:47.009Z",  
18      "miniature": "blog/l9Y0uYe2LyqVFc1UhLzFCdUf.png",  
19      "__v": 0  
20    },  
21    {  
22      "_id": "652dd1b1b9af7f64ed9990d3",  
23      "title": "Como programar nivel 3",  
24      "content": "El mundo de la programacion y algo mas",  
25      "path": "/como_programar_y_algo_mas",  
26      "created_at": null,  
27      "miniature": "blog/FlnWl6FRu6gvCrQyPXNTkVKc.png",  
28      "__v": 0  
29    }  
30  ],  
31  "total": 3,  
32  "limit": 10,  
33  "page": 1,  
34  "pages": 1  
35 }
```

De esta manera damos por terminado nuestra función delete.



OBTENER UN POST POR SU PATH

Ahora vamos a crear nuestra función para poder obtener el path de un post, para que podamos acceder directamente a ese post en nuestras publicaciones, así que vamos a crear la función **getPost** en singular para poder obtener esa ruta de la publicación:

```
controllers > post.controller.js > ...
...
70  function getPost(req, res) {
71      const { path } = req.params
72
73      Post.findOne({ path }, (error, postStored) => {
74          if (error) {
75              res.status(500).send({msg: "Error del servidor"})
76          } else if (!postStored) {
77              res.status(400).send({msg: "No se ha encontrado ningun post"})
78          } else {
79              res.status(400).send(postStored)
80          }
81      })
82  }
83
84  module.exports = {
85      createPost,
86      getPosts,
87      updatePost,
88      deletePost,
89      getPost
90  }
```

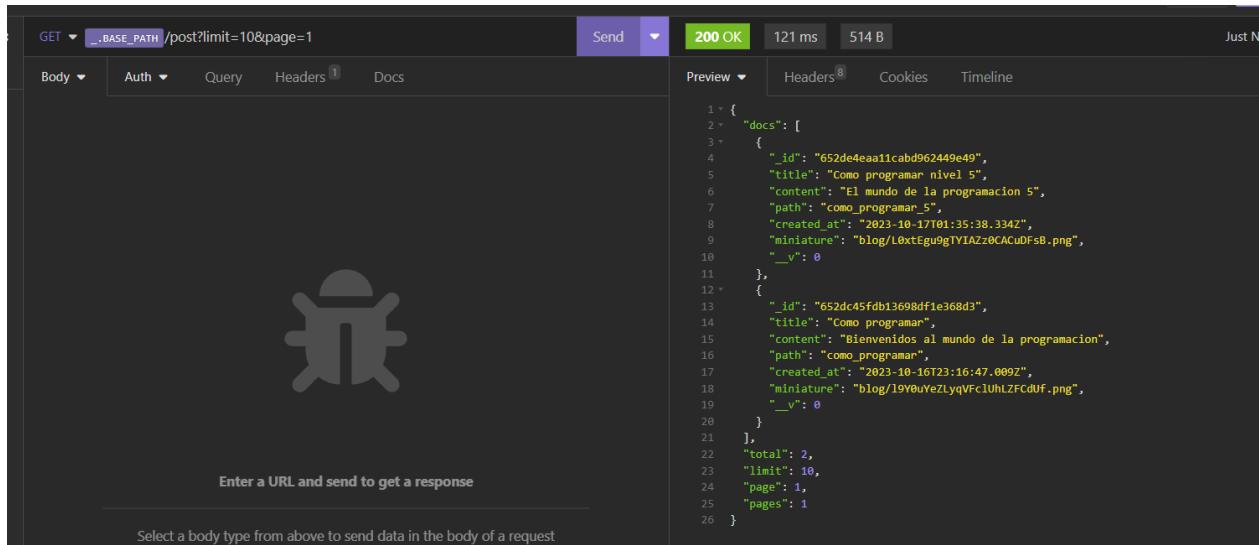
Como podemos observar a través de la función `findOne` obtenemos el path y validamos a través de los diferentes errores que nos puede generar con mensajes, vamos a agregar nuestra ruta en `post.router.js`:

```
router > [js] post.router.js > ...
1 const express = require("express")
2 const multiparty = require("connect-multiparty")
3 const PostController = require("../controllers/post.controller")
4 const md_auth = require("../middlewares/authenticated")
5
6 const md_upload = multiparty({ uploadDir: "./uploads/blog"})
7 const api = express.Router()
8
9
10 api.post("/post", [md_auth.asureAuth, md_upload], PostController.createPost)
11 api.get("/post", PostController.getPosts)
12 api.patch("/post/:id", [md_auth.asureAuth, md_upload], PostController.updatePost)
13 api.delete("/post/:id", [md_auth.asureAuth], PostController.deletePost)
14 api.get("/post/:path", PostController.getPost)
15
16
17 module.exports = api
```



@hdtoledo

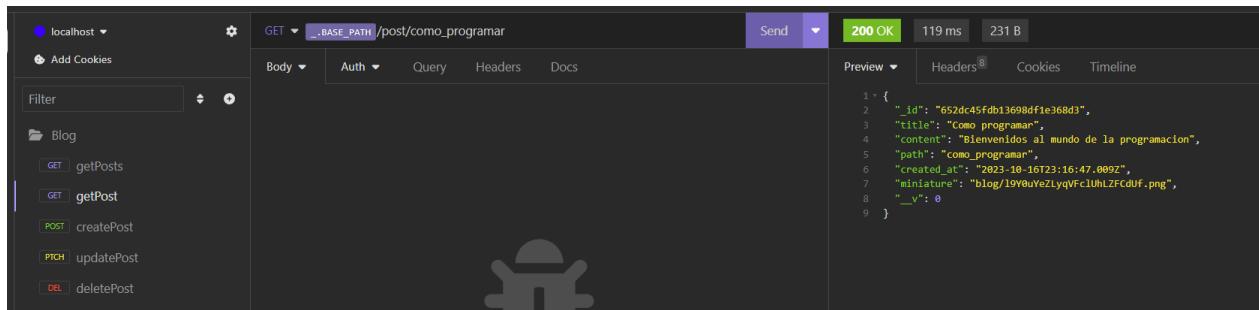
ahora vamos a obtener la ruta de nuestro post a través de getPosts:



The screenshot shows the Postman interface with a successful response from a GET request to `/_BASE_PATH/post?limit=10&page=1`. The response is a JSON object containing an array of documents (posts). Each post includes fields like _id, title, content, path, created_at, and miniature.

```
1 + {
2   "docs": [
3     {
4       "_id": "652de4ea11cabd962449e49",
5       "title": "Como programar nivel 5",
6       "content": "El mundo de la programacion 5",
7       "path": "como_programar_5",
8       "created_at": "2023-10-17T01:35:38.334Z",
9       "miniature": "blog/L0xtEgu9gTYIAZz0CACuDFsB.png",
10      "__v": 0
11    },
12    {
13      "_id": "652dc45fdb13698df1e368d3",
14      "title": "Como programar",
15      "content": "Bienvenidos al mundo de la programacion",
16      "path": "como_programar",
17      "created_at": "2023-10-16T23:16:47.009Z",
18      "miniature": "blog/l9v0uYeZlyqVFc1UhLZFCdUf.png",
19      "__v": 0
20    }
21  ],
22  "total": 2,
23  "limit": 10,
24  "page": 1,
25  "pages": 1
26 }
```

Copiamos el path y vamos a crear la ruta de nuestro `getPost` y enviamos la petición:



The screenshot shows the Postman interface with a successful response from a GET request to `/_BASE_PATH/post/como_programar`. The response is a JSON object containing a single post document.

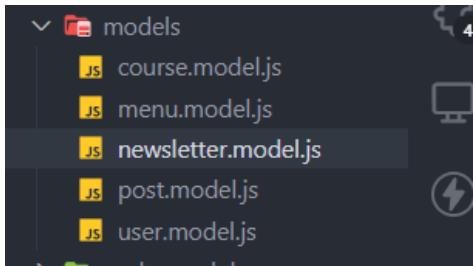
```
1 + {
2   "_id": "652dc45fdb13698df1e368d3",
3   "title": "Como programar",
4   "content": "Bienvenidos al mundo de la programacion",
5   "path": "como_programar",
6   "created_at": "2023-10-16T23:16:47.009Z",
7   "miniature": "blog/l9v0uYeZlyqVFc1UhLZFCdUf.png",
8   "__v": 0
9 }
```

De esta manera vamos a obtener los datos de nuestro post a través del path.



MODELO NEWSLETTER

En nuestra aplicación vamos a almacenar los correos de las personas que quieren que les llegue la información relacionada de nuestra web, con todo el contenido, así que vamos a crear nuestro modelo de una manera sencilla, ya que lo único que requerimos es el correo electrónico, vamos a crear en la carpeta **models** nuestro archivo **newsletter.model.js**



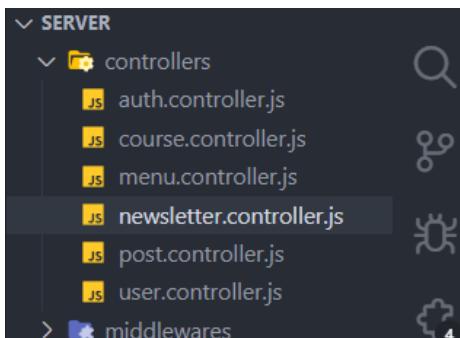
Ahora dentro vamos a dejar la estructura siguiente:

```
models > newsletter.model.js > <unknown>
1  const mongoose = require("mongoose")
2
3  const NewsletterSchema = mongoose.Schema({
4      email: {
5          type: String,
6          unique: true,
7      }
8  })
9
10 module.exports = mongoose.model("Newsletter", NewsletterSchema)
```

De esta manera dejamos nuestro modelo.

ESTRUCTURA API NEWSLETTER

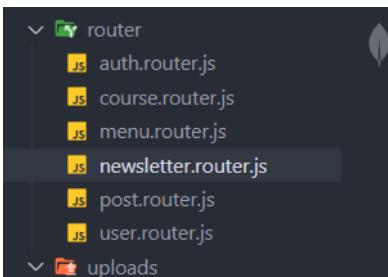
Vamos a crear la estructura de nuestro API newsletter, para ello vamos a crear en **controller** nuestro archivo **newsletter.controller.js**:



Ahora dejamos la siguiente estructura:

```
controllers > newsletter.controller.js > <unknown>
1 const Newsletter = require("../models/newsletter.model")
2
3 //Funciones ...
4
5 module.exports = {
6   |
7 }
```

Creamos nuestro archivo **newsletter.router.js** dentro de **router**:



Ahora dejamos la siguiente estructura:

```
router > newsletter.router.js > ...
1 const express = require("express")
2 const NewsletterController = require("../controllers/newsletter.controller")
3
4 const md_auth = require("../middlewares/authenticated")
5
6 const api = express.Router()
7
8 //Routes
9
10
11 module.exports = api
```

Ahora por último lo añadimos a nuestro **app.js** importamos la ruta:

```
js app.js > newsletterRoutes
9 // Importar rutas
10 const authRoutes = require("./router/auth.router")
11 const userRoutes = require("./router/user.router")
12 const menuRoutes = require("./router/menu.router")
13 const courseRoutes = require("./router/course.router")
14 const postRoutes = require("./router/post.router")
15 const newsletterRoutes = require("./router/newsletter.router")
16
```



Configuramos nuestra ruta:

```
js app.js > ...
27 // Configurar Rutas
28 app.use('/api/${API_VERSION}', authRoutes)
29 app.use('/api/${API_VERSION}', userRoutes)
30 app.use('/api/${API_VERSION}', menuRoutes)
31 app.use('/api/${API_VERSION}', courseRoutes)
32 app.use('/api/${API_VERSION}', postRoutes)
33 app.use('/api/${API_VERSION}', newsletterRoutes)
34
```

Recordemos que todo debe funcionar correctamente en nuestro servidor:

```
[nodemon] starting `node index.js`
#####
### MERN API REST ###
#####
http://localhost:3000/api/v1
|
```

REGISTRAR EMAIL EN NEWSLETTER

Vamos a realizar la función para poder registrar un correo en nuestro newsletter, para ello vamos a irnos a nuestro **newsletter.controller.js** y creamos la función **suscribeEmail**:

```
controllers > js newsletter.controller.js > ✎ suscribeEmail
1 const Newsletter = require("../models/newsletter.model")
2
3 function suscribeEmail(req, res) {
4     const { email } = req.body
5
6     if (!email) res.status(400).send({ msg: "Email obligatorio" })
7
8     const newsletter = new Newsletter({
9         email: email.toLowerCase(),
10    })
11
12     newsletter.save((error) => {
13         if (error) {
14             res.status(400).send({ msg: "El email ya esta registrado" })
15         } else {
16             res.status(200).send({ msg: "Email registrado con exito" })
17         }
18     })
19 }
20
21 module.exports = {
22     suscribeEmail,
23 }
```

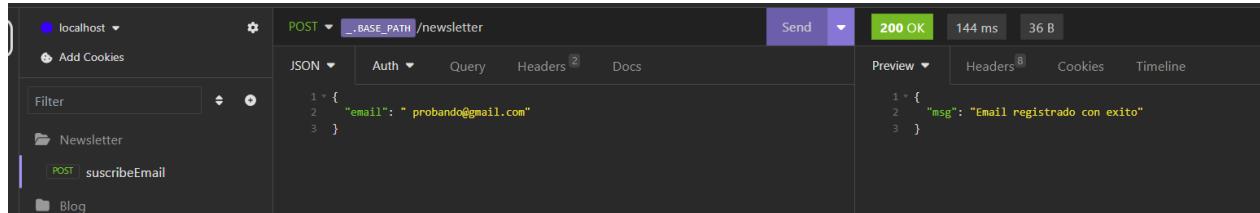


@hdtoledo

Ahora nos vamos a nuestra ruta **newsletter.router.js** y agregamos la ruta:

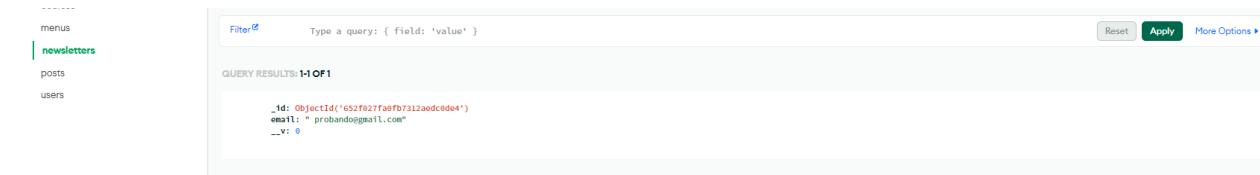
```
router > js newsletter.router.js > ...
1  const express = require("express")
2  const NewsletterController = require("../controllers/newsletter.controller")
3
4  const md_auth = require("../middlewares/authenticated")
5
6  const api = express.Router()
7
8  api.post("/newsletter", NewsletterController.subscribeEmail)
9
10
11 module.exports = api
```

Ahora en nuestro insomnia vamos a crear una nueva carpeta que se llamará **newsletter** y creamos la nueva petición **subscribeEmail**:



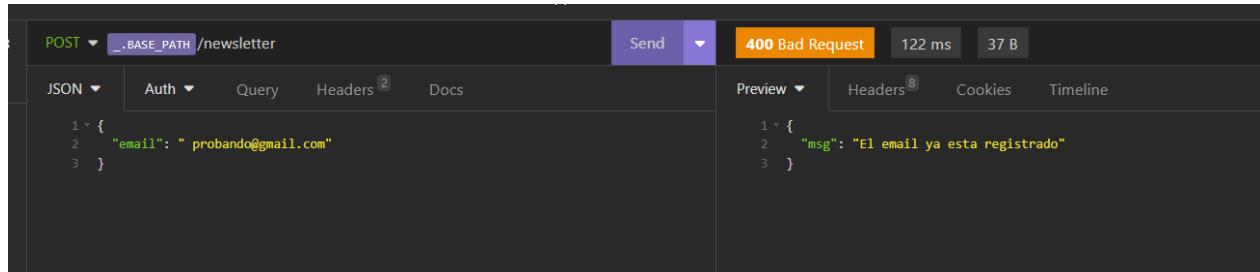
The screenshot shows the insomnia API client interface. A POST request is made to `_.BASE_PATH /newsletter`. The JSON body is `{ "email": "probando@gmail.com" }`. The response is a 200 OK status with the message `{ "msg": "Email registrado con exito" }`.

Revisamos en nuestro mongodb:



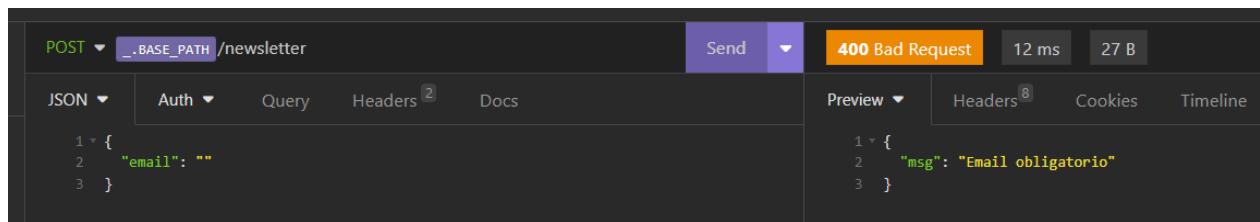
The screenshot shows the MongoDB Compass interface. It displays a single document in the newsletters collection with the following fields: `_id: ObjectId("652f027fa0fb7312a0dc0de4")`, `email: "probando@gmail.com"`, and `__v: 0`.

Y si volvemos a intentar hacer la petición con el mismo correo en insomnia:



The screenshot shows the insomnia API client interface. A POST request is made to `_.BASE_PATH /newsletter`. The JSON body is `{ "email": "probando@gmail.com" }`. The response is a 400 Bad Request status with the message `{ "msg": "El email ya esta registrado" }`.

Y si lo enviamos sin email en nuestro insomnia:



The screenshot shows the insomnia API client interface. A POST request is made to `_.BASE_PATH /newsletter`. The JSON body is `{ "email": "" }`. The response is a 400 Bad Request status with the message `{ "msg": "Email obligatorio" }`.

OBTENER TODOS LOS EMAIL CON PAGINACION

Vamos a realizar una modificación dentro de nuestro modelo para poder hacer la paginación vamos a **newsletter.model.js**:

```
models > JS newsletter.model.js > ...
1  const mongoose = require("mongoose")
2  const mongoosePaginate = require("mongoose-paginate")
3
4  const NewsletterSchema = mongoose.Schema({
5      email: {
6          type: String,
7          unique: true,
8      }
9  })
10
11 NewsletterSchema.plugin(mongoosePaginate)
12
13 module.exports = mongoose.model("Newsletter", NewsletterSchema)
```

Agregamos **mongoose paginate** y lo ponemos en funcionamiento, ahora vamos a crear nuestra función **getEmails** dentro de **newsletter.controller.js**:

```
controllers > JS newsletter.controller.js > 📄 <unknown>
1
2
3
4
5
6
7
8
9
10
11 function getEmails(req, res) {
12     const { page = 1, limit = 10 } = req.query;
13
14     const options = {
15         page: parseInt(page),
16         limit: parseInt(limit),
17     }
18
19     Newsletter.paginate({}, options, (error, emailsStored) => {
20         if (error) {
21             res.status(400).send({ msg: "Error al obtener los emails" })
22         } else {
23             res.status(200).send(emailsStored)
24         }
25     })
26 }
27
28
29
30
31
32
33
34
35
36
37
38
39 module.exports = {
40     suscribeEmail,
41     getEmails,
42 }
```

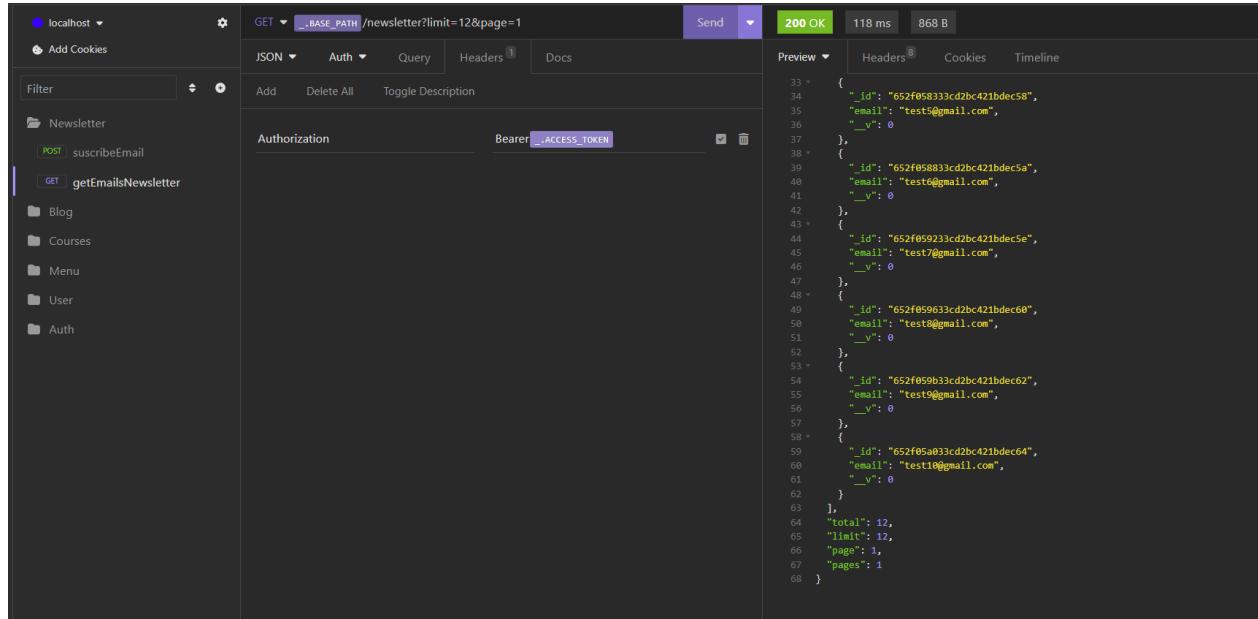


@hdtoledo

Esta función es muy similar a la utilizada anteriormente, ahora vamos a crear la ruta de nuestros mails en **newsletter.router.js**:

```
router > js newsletter.router.js > ...
1 const express = require("express")
2 const NewsletterController = require("../controllers/newsletter.controller")
3 const md_auth = require("../middlewares/authenticated")
4
5 const api = express.Router()
6
7 api.post("/newsletter", NewsletterController.subscribeEmail)
8 api.get("/newsletter", [ md_auth.ensureAuth ], NewsletterController.getEmails)
9
10
11 module.exports = api
```

Recordemos que esta ruta es get y que solo las personas autenticadas podrán acceder a ella, ahora vamos a crear en nuestro insomnia la ruta para poder obtener los correos:



The screenshot shows the Insomnia REST client interface. On the left, there's a sidebar with project navigation (localhost, Add Cookies) and a list of API endpoints under 'Newsletter': subscribeEmail (POST) and getEmailsNewsletter (GET). The main area shows a request configuration for a GET request to `/newsletter?limit=12&page=1`. The 'Headers' tab contains an 'Authorization' header set to 'Bearer `_ACCESS_TOKEN`'. The 'Send' button is highlighted in purple. To the right, the response details are shown: status **200 OK**, time **118 ms**, and size **868 B**. Below this, the 'Preview' tab displays the JSON response. The response body is a JSON array of 12 objects, each representing an email with fields: `_id`, `email`, and `_v`. The last object in the array also contains `total`, `limit`, `page`, and `pages`.

```
33 +
34 {
35   "id": "652f05833cd2bc421bdec58",
36   "email": "test1@gmail.com",
37   "_v": 0
38 },
39 {
40   "id": "652f058833cd2bc421bdec5a",
41   "email": "test6@gmail.com",
42   "_v": 0
43 },
44 {
45   "id": "652f059233cd2bc421bdec5e",
46   "email": "test7@gmail.com",
47   "_v": 0
48 },
49 {
50   "id": "652f059633cd2bc421bdec60",
51   "email": "test8@gmail.com",
52   "_v": 0
53 },
54 {
55   "id": "652f059b33cd2bc421bdec62",
56   "email": "test9@gmail.com",
57   "_v": 0
58 },
59 {
60   "id": "652f05a033cd2bc421bdec64",
61   "email": "test10@gmail.com",
62   "_v": 0
63 },
64 {
65   "total": 12,
66   "limit": 12,
67   "page": 1,
68   "pages": 1
69 }
```

De esta manera ya validamos la obtención de los correos registrados y los tenemos paginados.



@hdtoledo

ELIMINAR UN REGISTRO POR EL EMAIL

vamos a crear nuestra función **deleteEmail** dentro de **newsletter.controller.js**

```
controllers > JS newsletter.controller.js > <unknown>
38  function deleteEmail(req, res) {
39    const { id } = req.params
40
41    Newsletter.findByIdAndDelete(id, (error) => {
42      if (error) {
43        res.status(400).send({ msg: "Error al eliminar el registro" })
44      } else {
45        res.status(200).send({ msg: "Eliminacion correcta" })
46      }
47    })
48  }
49
50
51  module.exports = [
52    subscribeEmail,
53    getEmails,
54    deleteEmail,
55  ]
```

Nuestra función de eliminación es muy similar a la anterior, ahora vamos a nuestro **newsletter.router.js** y agregamos la ruta **delete**:

```
router > JS newsletter.router.js > ...
1  const express = require("express")
2  const NewsletterController = require("../controllers/newsletter.controller")
3  const md_auth = require("../middlewares/authenticated")
4
5  const api = express.Router()
6
7  api.post("/newsletter", NewsletterController.subscribeEmail)
8  api.get("/newsletter", [ md_auth.asureAuth ], NewsletterController.getEmails)
9  api.delete(["/newsletter/:id"], [ md_auth.asureAuth ], NewsletterController.deleteEmail)
10
11
```

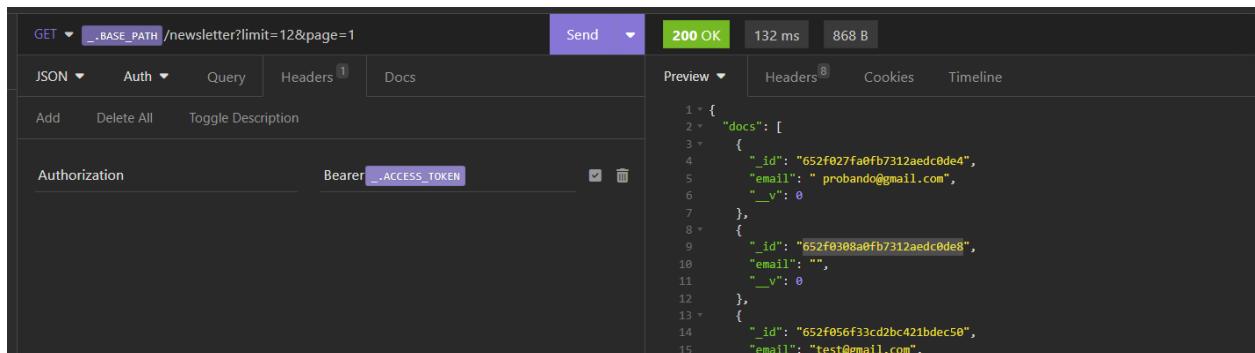
Ahora vamos a crear nuestra ruta **delete** en insomnia con las configuraciones necesarias:

The screenshot shows the insomnia API client interface. On the left sidebar, there is a tree view with a 'Newsletter' folder containing two items: 'deleteEmail' (marked with a red 'DEL' icon) and 'subscribeEmail' (marked with a green 'POST' icon). The main panel shows a 'DELETE' request to the path '_BASE_PATH /newsletter/'. The 'Authorization' header is set to 'Bearer _ACCESS_TOKEN'. Other tabs like 'Body', 'Auth', 'Query', and 'Headers' are visible at the top.



@hdtoledo

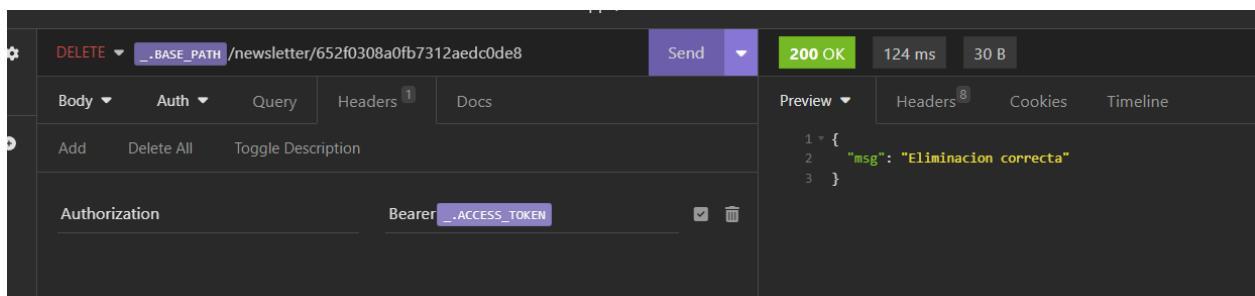
Y vamos a pasarle el id de uno de los correos:



```
1 v {
2 v   "docs": [
3 v     {
4 v       "_id": "652f027fa0fb7312aedc0de4",
5 v       "email": "probando@gmail.com",
6 v       "__v": 0
7 v     },
8 v     {
9 v       "_id": "652f0308a0fb7312aedc0de8",
10 v      "email": "",
11 v      "__v": 0
12 v    },
13 v    {
14 v      "_id": "652f056f33cd2bc421bdec50",
15 v      "email": "test@gmail.com",

```

Seleccionare en este caso el que está sin correo electrónico y lo voy a ejecutar:



```
1 v {
2 v   "msg": "Eliminacion correcta"
3 }
```

De esta manera nos ha quedado configurado nuestra eliminación de correo registrado.

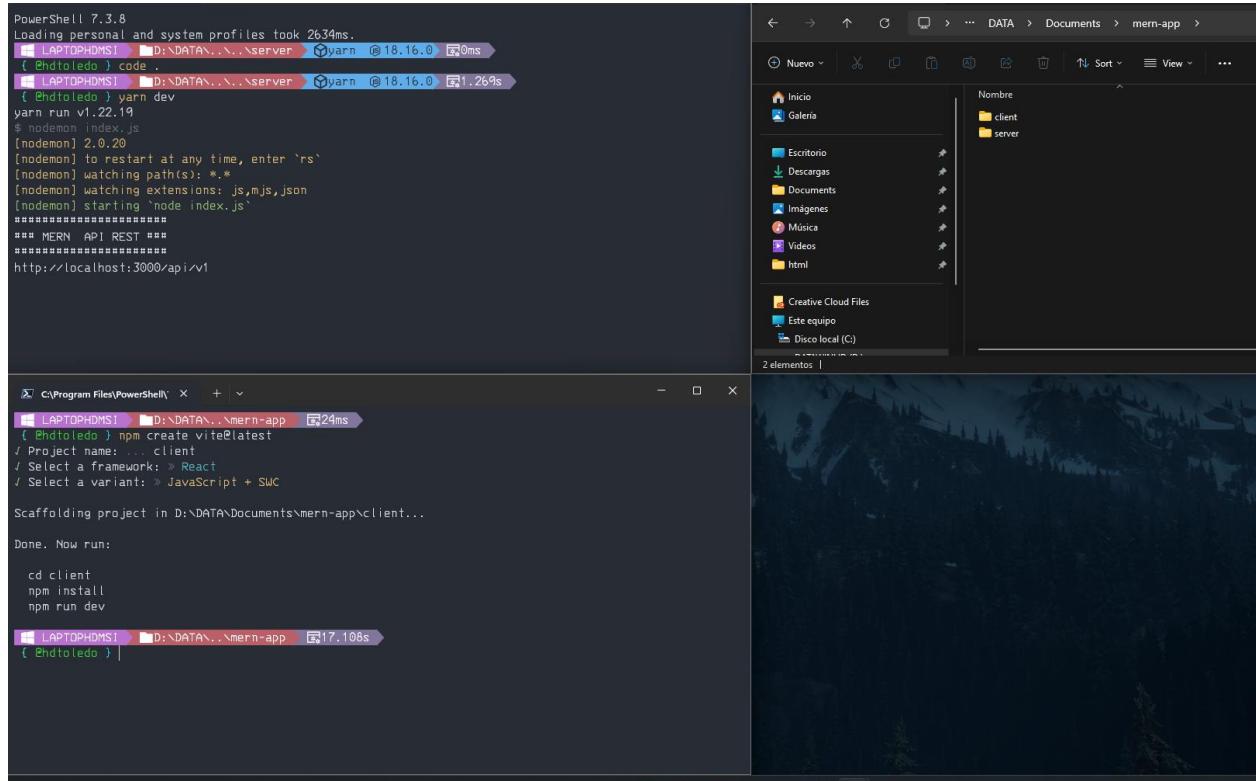


@hdtoledo

FRONTEND - CREANDO EL PROYECTO DE REACT

Ya tenemos nuestro **backend** funcionando y ahora llego el turno de crear todo el **frontend** que se encargara de consumir nuestra API.

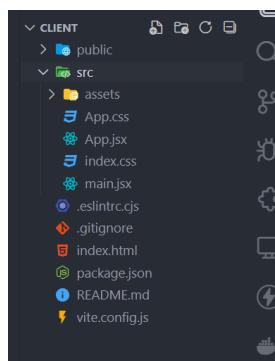
Vamos a iniciar a crear nuestro proyecto de **react** para ello vamos a dejar en una terminal ejecutando nuestro servidor **backend** y en la otra la vamos a dejar para ejecutar la instalación de **react** recordemos que ambas carpetas de **backend** y **frontend** deben estar ubicadas en la misma posición:



The screenshot shows a Windows desktop environment. On the left, there are two terminal windows in PowerShell. The top window shows the command 'yarn dev' being run, with output indicating the start of a nodemon process watching for changes in index.js and json files, and a message about restarting at any time by entering 'rs'. It also shows the URL 'http://localhost:3000/api/v1'. The bottom window shows the command 'npm create vite@latest' being run, with a series of prompts for project name ('client'), framework ('React'), and variant ('JavaScript + SWC'). The response 'Scaffolding project in D:\DATA\Documents\mern-app\client...' is shown. On the right, a file explorer window is open, showing a folder structure with 'client' and 'server' subfolders under 'mern-app', which is located under 'DATA\Documents'. The desktop background features a dark landscape scene.

Ejecutamos **npm create vite@latest**, y seleccionamos el nombre del proyecto en mi caso lo dejare **client** y seleccionamos react, javascript + swc.

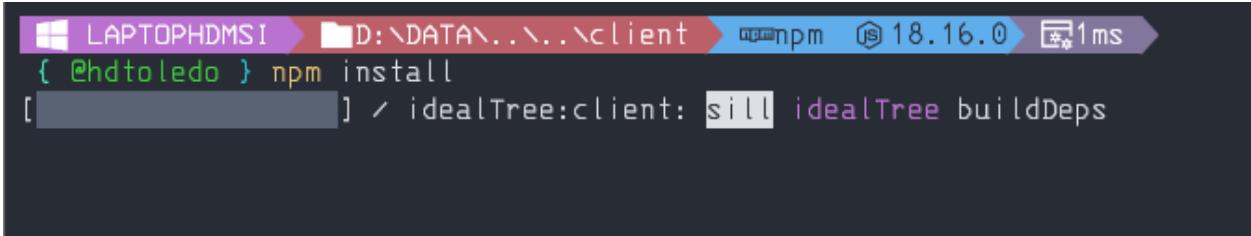
Abrimos nuestro proyecto en VS code a través de la terminal y recordemos que primero debemos entrar a la carpeta en mi caso **cd client** y luego con el comando **code .** se abrirá nuestro VS.



Vamos a obtener un proyecto base a través del cual lo iremos modificando para dejarlo de acuerdo a lo que necesitamos.

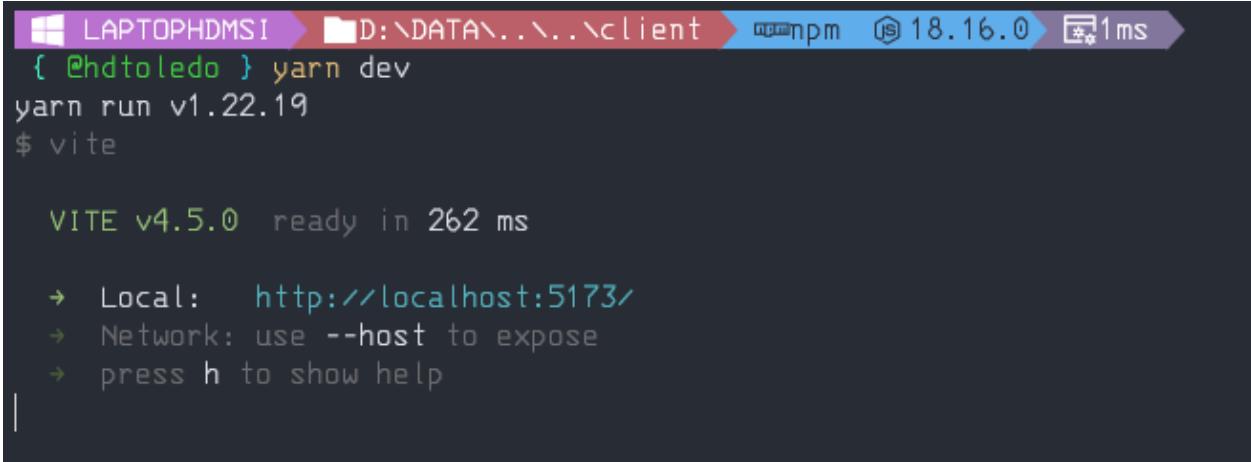


Nuestro siguiente paso será instalar las dependencias a través de la terminal ejecutando **npm install**



```
LAPTOPHDMI ~ D:\DATA\...\client ➜ npm 18.16.0 1ms
{ @hdtolledo } npm install
[...] ✘ idealTree:client: sill idealTree buildDeps
```

Una vez instaladas procedemos a ejecutar el servidor con el comando **yarn dev**

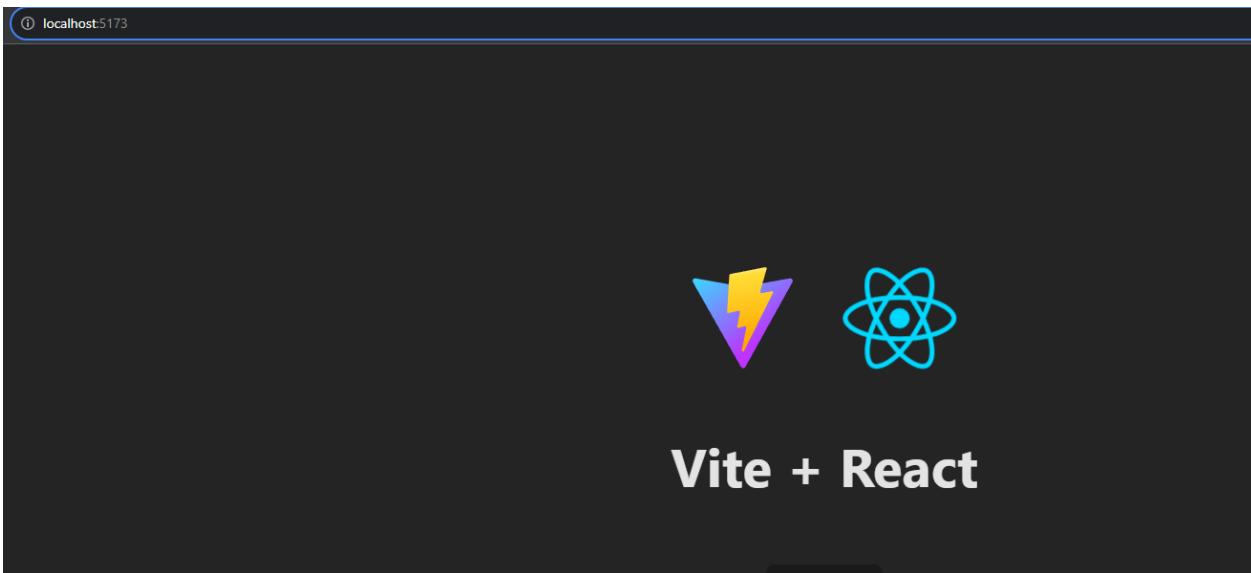


```
LAPTOPHDMI ~ D:\DATA\...\client ➜ npm 18.16.0 1ms
{ @hdtolledo } yarn dev
yarn run v1.22.19
$ vite

VITE v4.5.0 ready in 262 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```

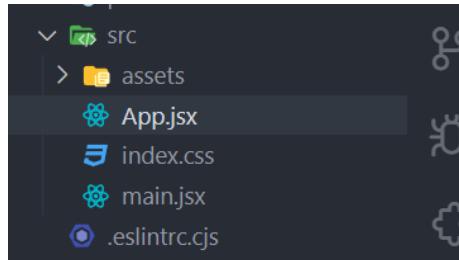
Abrimos nuestro navegador en la dirección de nuestro localhost:



Y allí ya tenemos nuestro proyecto funcionando, ahora vamos a realizar unos pequeños cambios sobre el proyecto.



1. Eliminamos App.css



2. Eliminamos todo el contenido de App.jsx

A screenshot of a code editor showing the 'App.jsx' file. The file is currently empty, with only a single digit '1' visible on the first line.

3. Creamos un rfc

A screenshot of a code editor showing the 'App.jsx' file. A code completion dropdown is open at the top of the file, displaying suggestions: 'rfc', 'rfcE', 'rfcP', and 'rfcRedux'. Each suggestion has a corresponding tooltip or description below it.

4. Y nos quedaría así:

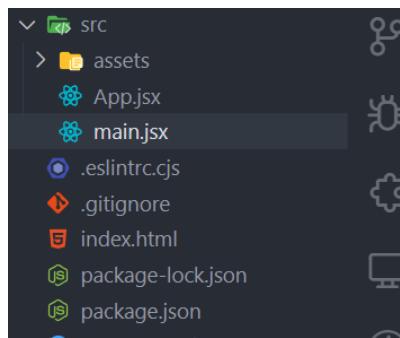
A screenshot of a code editor showing the 'App.jsx' file. The code has been auto-generated by the 'rfc' completion. It starts with an import statement: 'import React from 'react''. Below that is the definition of a function component named 'App': 'export default function App() {'. The component returns a single 'div' element with the text 'App'.

```
1 import React from 'react'
2
3 export default function App() {
4   return (
5     <div>App</div>
6   )
7 }
8 |
```



@hdtoledo

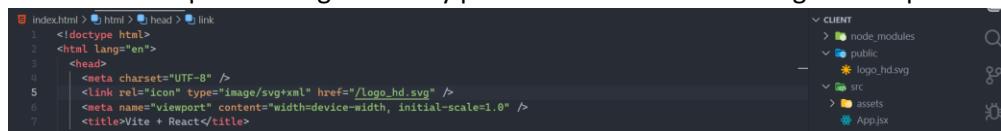
5. Eliminamos index.css



6. Eliminamos la importación de index.css de Main.jsx

```
src > main.jsx
1  import React from 'react'
2  import ReactDOM from 'react-dom/client'
3  import App from './App.jsx'
4
5  ReactDOM.createRoot(document.getElementById('root')).render(
6    <React.StrictMode>
7      <App />
8    </React.StrictMode>,
9  )
10
```

7. Eliminamos de public el logo de vite y podemos colocar nuestro logo de la aplicación:



Y modificamos nuestro index.html para agregar el logo.

8. Modificamos un poco nuestro componente de App.jsx

```
src > App.jsx > App
1  import React from 'react'
2
3  export default function App() {
4    return (
5      <div>
6        <h1> Bienvenidos !! </h1>
7      </div>
8    )
9  }
10
```

Y nos quedaría de la siguiente manera en nuestro navegador:



Bienvenidos !!



INSTALANDO SASS

The screenshot shows the official Sass website. At the top, there's a navigation bar with links for 'Playground', 'Install', 'Learn Sass', 'Blog', 'Documentation', and 'Get Involved'. A search bar is also present. The main heading 'CSS with superpowers' is displayed above a large illustration of a pair of teal-rimmed glasses. Below the heading, a subtext states: 'Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.' At the bottom of the page, a pink footer bar contains links for 'Current Releases: Dart Sass 1.69.4 LibSass 3.6.5 Ruby Sass' and 'Implementation Guide'.

Sass (Syntactically Awesome Stylesheets) es un preprocesador de CSS (Cascading Style Sheets) que extiende las capacidades del lenguaje CSS convencional. Sass proporciona una sintaxis más poderosa y eficiente para escribir estilos en comparación con CSS puro. Algunas de las características clave de Sass incluyen:

1. **Variables:** Puedes definir variables para almacenar valores que se utilizan en múltiples lugares, lo que facilita la modificación de estilos en un solo lugar.
2. **Nesting:** Permite anidar selectores CSS dentro de otros selectores, lo que resulta en un código más estructurado y legible.
3. **Mixins:** Los mixins son bloques reutilizables de estilos que se pueden incluir en múltiples partes de tu hoja de estilo.
4. **Herencia:** Puedes heredar estilos de un selector a otro, lo que ahorra tiempo y reduce la duplicación de código.
5. **Operaciones matemáticas:** Sass permite realizar operaciones matemáticas en los valores, lo que es útil para calcular tamaños, márgenes y otros atributos.
6. **Partials:** Puedes dividir tu hoja de estilo en varios archivos más pequeños (llamados "partials") y luego importarlos en un archivo principal.
7. **Extensibility:** Sass es altamente personalizable y extensible, lo que significa que puedes crear tus propias funciones y extensiones para adaptarse a tus necesidades específicas.

Sass es particularmente útil en proyectos web grandes y complejos, ya que ayuda a mantener el código CSS organizado, facilita las actualizaciones y reduce la redundancia. Para utilizar Sass, debes compilar tus archivos Sass en archivos CSS tradicionales que los navegadores web puedan entender. Esto se puede hacer con herramientas de compilación, como Node.js y Gulp, que convierten tus archivos Sass en CSS.



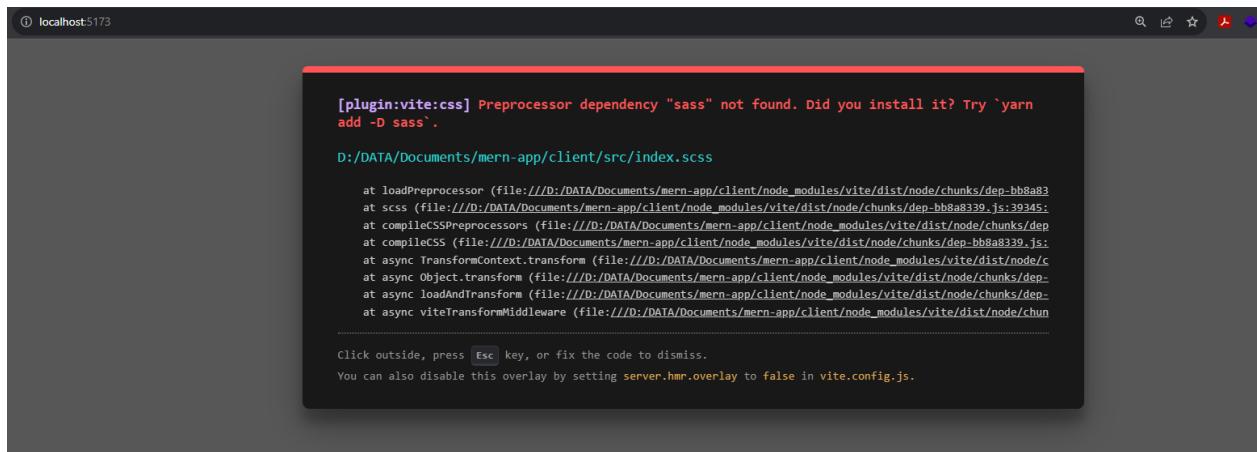
Vamos a empezar por crear nuestro **index.scss** dentro de **src**:



Y lo llamaremos dentro de nuestro **main.jsx**:

```
src > ⚡ main.jsx
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import App from './App.jsx'
4 import './index.scss'
5
6 ReactDOM.createRoot(document.getElementById('root')).
```

Esta pequeña configuración nos va a mostrar en nuestro navegador lo siguiente:

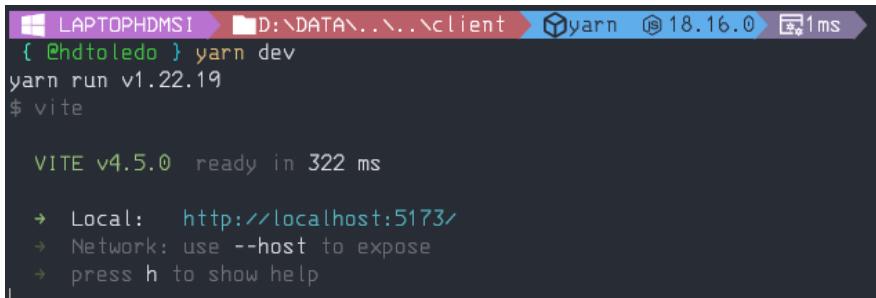


Un error ya que no lo tenemos instalado, así que vamos a ir a nuestra terminal y vamos a colocar el comando **yarn add sass**:

```
LAPTOPHDMI1 D:\DATA\...\client ➜ npm 18.16.0 44ms
{ Ehdtoledo } yarn add sass
yarn add v1.22.19
info No lockfile found.
warning package-lock.json found. Your project contains lock files generated by other packages to ensure deterministic behavior. It is advised not to mix package managers in order to avoid resolution inconsistencies between them. To clear this warning, remove package-lock.json.
[1/4] Resolving packages...
[2/4] Fetching packages...
[***] Fetching packages...
```



Una vez instalado procedemos a levantar el servidor del cliente:

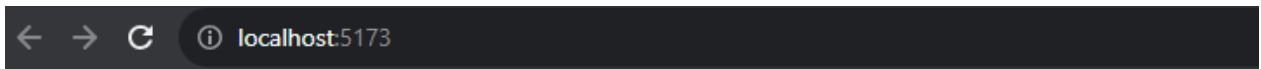


```
LAPTOPHDMI ➔ D:\DATA\...\client ➔ yarn 18.16.0 1ms
{ @hdtolledo } yarn dev
yarn run v1.22.19
$ vite

VITE v4.5.0 ready in 322 ms

+ Local: http://localhost:5173/
+ Network: use --host to expose
+ press h to show help
```

Y si vamos a revisar en nuestro navegador nos debe levantar los servicios:



Bienvenidos !!

INSTALANDO SEMANTIC UI REACT

[Semantic UI React](#) es una biblioteca de componentes de interfaz de usuario (UI) para React, que se basa en Semantic UI, un framework de diseño CSS. Semantic UI React proporciona una serie de componentes preestilizados y altamente personalizables que se pueden utilizar para construir interfaces de usuario de alta calidad en aplicaciones web basadas en React.

Las características principales de Semantic UI React incluyen:

1. **Componentes personalizables:** La biblioteca ofrece una amplia gama de componentes de interfaz de usuario, como botones, menús desplegables, formularios, tarjetas y más. Estos componentes son altamente personalizables y se pueden adaptar para satisfacer las necesidades de diseño específicas.
2. **Integración con React:** Semantic UI React se integra perfectamente con React, lo que significa que los componentes se crean y se actualizan de acuerdo con el flujo de datos y el estado de React. Puedes utilizar el estado y las propiedades de React para controlar el comportamiento de los componentes de Semantic UI React.
3. **Uso de Semantic UI:** Semantic UI React se basa en Semantic UI, lo que significa que hereda la filosofía de diseño y las clases de estilo de Semantic UI. Esto facilita la creación de interfaces de usuario coherentes y visualmente atractivas.



4. **Documentación completa:** La biblioteca cuenta con una documentación completa y ejemplos detallados que facilitan el aprendizaje y la implementación de sus componentes en proyectos React.
5. **Comunidad activa:** Semantic UI React cuenta con una comunidad activa de desarrolladores y una amplia base de usuarios, lo que significa que puedes encontrar soporte, ejemplos y recursos en línea.
6. **Estilos responsivos:** Los componentes de Semantic UI React están diseñados para ser responsivos, lo que significa que se adaptan automáticamente a diferentes tamaños de pantalla y dispositivos.

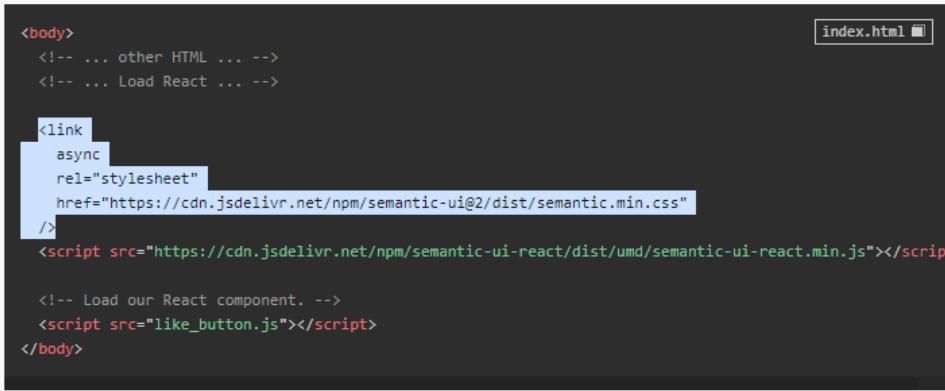
En resumen, Semantic UI React es una biblioteca que simplifica la creación de interfaces de usuario atractivas y personalizables en aplicaciones web React. Es especialmente útil para desarrolladores que desean aprovechar las ventajas de Semantic UI junto con las ventajas de React para construir interfaces de usuario sofisticadas.

Ahora que ya conocemos un poco acerca de semantic UI vamos a nuestra terminal y vamos a instalar ejecutando el comando **yarn add semantic-ui-react semantic-ui-css**



```
LAPTOPHDMISI D:\DATA\...\client yarn 18.16.0 2ms
{ @hdtoledo } yarn add semantic-ui-react semantic-ui-css
yarn add v1.22.19
```

Para realizar la importación de nuestro css de semantic lo vamos a hacer a través del CDN:



```
<body>
  <!-- ... other HTML ... -->
  <!-- ... Load React ... -->

  <link
    async
    rel="stylesheet"
    href="https://cdn.jsdelivr.net/npm/semantic-ui@2/dist/semantic.min.css"
  />
  <script src="https://cdn.jsdelivr.net/npm/semantic-ui-react/dist/umd/semantic-ui-react.min.js"></script>

  <!-- Load our React component. -->
  <script src="like_button.js"></script>
</body>
```

Este enlace lo vamos a colocar en nuestro **index.html** de la siguiente manera:



```
index.html > html > head > link
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="UTF-8" />
5      <link rel="icon" type="image/svg+xml" href="/Logo_hd.svg" />
6      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7      <link async rel="stylesheet" href="https://cdn.jsdelivr.net/npm/semantic-ui@2/dist/semantic.min.css"/>
8      <title>MERN App</title>
9    </head>
10   <body>
11     <div id="root"></div>
12     <script type="module" src="/src/main.jsx"></script>
13   </body>
14 </html>
```

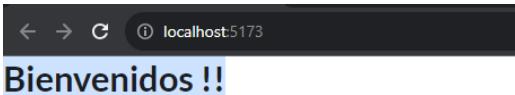


Ahora vamos a levantar nuestro servidor del cliente de nuevo y debe funcionar sin novedades:

```
LAPTOPHDSI D:\DATA\...\client yarn dev
yarn run v1.22.19
$ vite

VITE v4.5.0 ready in 288 ms

→ Local: http://localhost:5173/
→ Network: use --host to expose
→ press h to show help
```



El primer cambio que notamos en el navegador es el tipo de fuente que ya se está realizando el cambio con semantic UI, ahora vamos a probar desde la web de semantic un botón:

A screenshot of the Semantic UI React documentation website. On the left, there's a sidebar with navigation links like 'Introduction', 'Get Started', 'Composition', etc. The main area shows a code editor with two tabs: 'Primary' and 'Secondary'. The 'Primary' tab is selected. It displays the following code:

```
1 import React from 'react'
2 import { Button } from 'semantic-ui-react'
3
4 const ButtonExampleEmphasis = () => (
5   <div>
6     <Button primary>Primary</Button>
7     <Button secondary>Secondary</Button>
8   </div>
9 )
10
11 export default ButtonExampleEmphasis
12
```

Below the code, a note says 'You can do the same using shorthands.' There are also buttons for 'Prettier', 'Reset', 'Copy', and 'Edit' at the top right of the code editor.

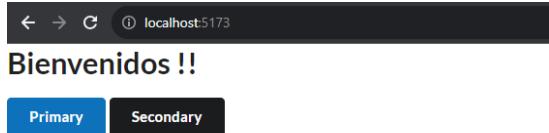
Vamos a copiar la línea de la importación y la vamos a colocar en nuestro App.jsx:

```
src > App.jsx > ...
1 import React from 'react'
2 import { Button } from 'semantic-ui-react'
3
4 export default function App() {
5   return (
6     <div>
```

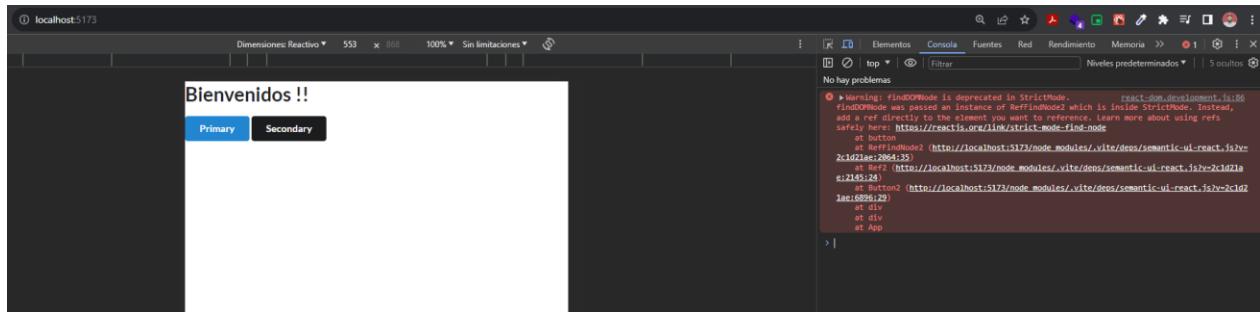
Y ahora copiamos los dos botones debajo de nuestro h1:

```
src > App.jsx > App
1 import React from "react";
2 import { Button } from "semantic-ui-react";
3
4 export default function App() {
5   return (
6     <div>
7       <h1> Bienvenidos !!</h1>
8       <div>
9         <Button primary>Primary</Button>
10        <Button secondary>Secondary</Button>
11      </div>
12    </div>
13  );
14}
```

Y el resultado en nuestro navegador debe ser el siguiente:



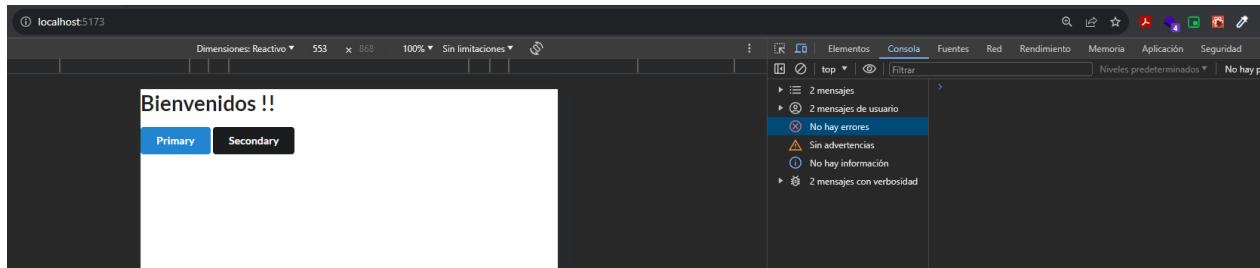
Vamos a revisar en el modo de inspección para ver si tenemos errores en nuestra app:



Este error es normal y para darle solución es muy sencillo, nos indica que el `findDOMNode` que se utiliza en el `StrictMode` está obsoleto, la solución es muy sencilla, nos dirigimos a `main.jsx` y vamos a quitar el `StrictMode`:

```
src > main.jsx
1 <import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import App from './App.jsx'
4 import './index.scss'
5
6 <ReactDOM.createRoot(document.getElementById('root')).render(
7   <App />
8 )
9
```

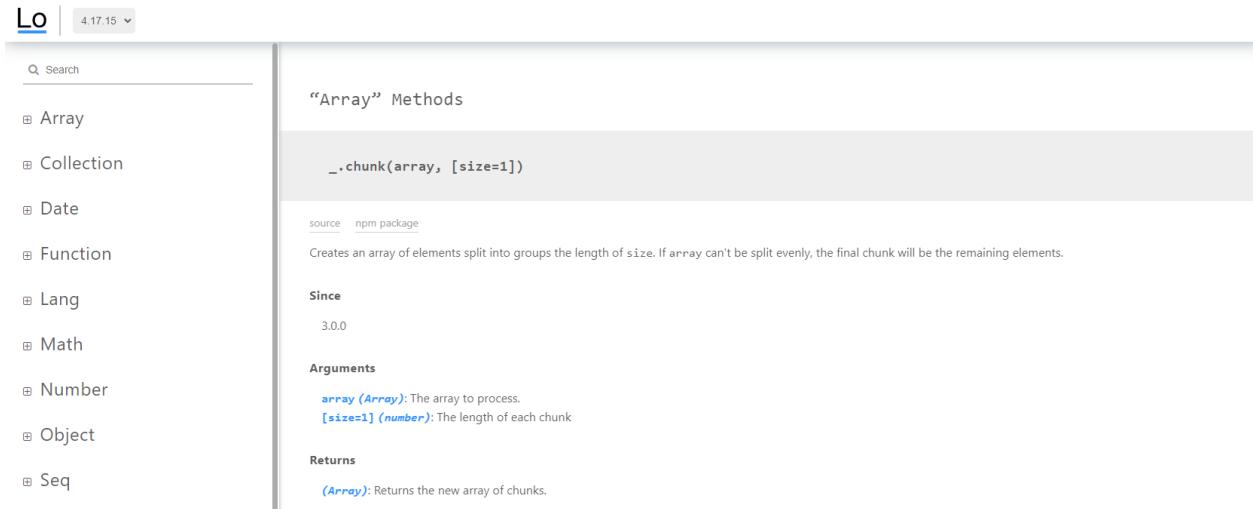
Si ahora revisamos en nuestro navegador no tendremos errores:



`findDOMNode` se solía utilizar para obtener una referencia al DOM real de un componente de React. Sin embargo, debido a los cambios y optimizaciones en React, esta función se ha vuelto obsoleta y puede causar problemas de rendimiento y otros efectos secundarios no deseados. Por lo tanto, se aconseja no utilizar `findDOMNode` en nuevos desarrollos y, en su lugar, optar por enfoques más modernos y seguros.



INSTALANDO LODASH



The screenshot shows the Lodash documentation page for the `_chunk` method. On the left, there's a sidebar with a search bar and a list of categories: Array, Collection, Date, Function, Lang, Math, Number, Object, and Seq. The main content area is titled "Array Methods" and shows the `_chunk` method. It includes a "source" link, an "npm package" link, a description ("Creates an array of elements split into groups the length of size. If array can't be split evenly, the final chunk will be the remaining elements."), a "Since" section (3.0.0), an "Arguments" section (with parameters `array (Array)` and `[size=1] (number)`), and a "Returns" section (with the return type `(Array)`). Below the sidebar, there's a navigation bar with links like Home, API, and Examples.

Lodash es una biblioteca de utilidades de JavaScript que proporciona muchas funciones y métodos de utilidad para realizar operaciones comunes en objetos, arreglos, cadenas, funciones y otros tipos de datos en JavaScript. Lodash simplifica la escritura de código y mejora la legibilidad al proporcionar una amplia gama de funciones que no están disponibles en el conjunto estándar de JavaScript o que son difíciles de implementar por sí mismas.

Algunas de las características y ventajas de Lodash incluyen:

1. **Funciones de utilidad:** Lodash ofrece una variedad de funciones de utilidad que van desde operaciones matemáticas básicas hasta manipulación de objetos y colecciones.
2. **Sintaxis consistente:** Las funciones de Lodash siguen una convención de nomenclatura consistente y fácil de recordar, lo que facilita su uso.
3. **Rendimiento:** Lodash está optimizado para el rendimiento y es eficiente en términos de uso de recursos.
4. **Manejo de arreglos y colecciones:** Lodash proporciona una amplia gama de funciones para ordenar, filtrar, mapear, agrupar y transformar arreglos y colecciones de datos.
5. **Manipulación de cadenas:** Ofrece funciones para manipular y formatear cadenas de texto de manera efectiva.
6. **Operaciones de objetos:** Facilita la clonación, mezcla y comparación de objetos.
7. **Funciones de flujo de datos:** Lodash proporciona funciones para trabajar con flujos de datos, lo que facilita la manipulación de datos de manera fluida.
8. **Compatibilidad con navegadores y entornos Node.js:** Lodash es compatible tanto con navegadores web como con entornos de servidor basados en Node.js.

Lodash se ha convertido en una biblioteca esencial en el desarrollo de JavaScript y es ampliamente utilizada en proyectos web y aplicaciones. Aunque muchas de las características de Lodash ahora están



@hdtoledo

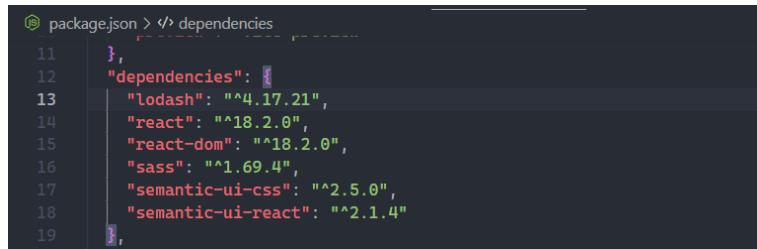
disponibles en el estándar ECMAScript (ES6 y posteriores), Lodash sigue siendo una opción popular debido a su facilidad de uso y amplio conjunto de utilidades.

Vamos a nuestra terminal de nuestro cliente y procedemos a ejecutar el comando **yarn add lodash**:



```
LAPTOPHDMI D:\DATA\..\..\client yarn 18.16.0 2ms
{ @hdtledo } yarn add lodash
yarn add v1.22.19
warning package-lock.json found. Your project contains lock files generated by
multiple tools. It's recommended to either update or remove them.
```

Revisamos nuestro package.json:



```
package.json > //> dependencies
  11   },
  12   "dependencies": {
  13     "Lodash": "^4.17.21",
  14     "react": "^18.2.0",
  15     "react-dom": "^18.2.0",
  16     "sass": "^1.69.4",
  17     "semantic-ui-css": "^2.5.0",
  18     "semantic-ui-react": "^2.1.4"
  19   },
```

INSTALANDO FORMIK Y YUP

[Formik](#) es una biblioteca de gestión de formularios para aplicaciones React. Está diseñada para simplificar la creación y gestión de formularios en aplicaciones web, lo que puede ser una tarea complicada y propensa a errores. Formik proporciona una serie de características y ventajas que hacen que trabajar con formularios en React sea más fácil y efectivo.

Algunas de las características y ventajas clave de Formik incluyen:

- Gestión de estado de formularios:** Formik se encarga de rastrear el estado de los formularios, lo que significa que no necesitas administrar manualmente los valores de los campos y los estados de validación.
- Validación de formularios:** Formik permite definir reglas de validación para los campos del formulario, lo que facilita la validación de datos y la presentación de mensajes de error.
- Manejo de envío de formularios:** Formik maneja automáticamente la lógica de envío de formularios, lo que incluye la prevención de envíos repetidos, la administración de estados de carga y éxito, y la interacción con servicios web o API.
- Integración con componentes de React:** Puedes utilizar componentes de entrada, campos de texto y otros componentes de React con Formik de manera sencilla, lo que facilita la creación de formularios personalizados.
- Campos reutilizables:** Formik proporciona componentes de campo reutilizables que cubren una amplia variedad de tipos de entrada y validaciones.
- Manejo de campos dinámicos:** Puedes agregar o quitar campos dinámicamente en función de las necesidades de tu formulario.



7. **Soporte para tocar campos:** Formik maneja el seguimiento de "toque" (touch) de campos, lo que es útil para mostrar errores solo cuando el usuario ha interactuado con un campo.
8. **Amplia documentación y comunidad activa:** Formik cuenta con una documentación completa y una comunidad activa de desarrolladores, lo que facilita aprender a usarlo y encontrar soluciones a problemas comunes.

Formik es ampliamente utilizado en proyectos de React para gestionar formularios de manera eficiente y efectiva. Simplifica el desarrollo de formularios complejos y ayuda a mejorar la experiencia del usuario al proporcionar un manejo robusto de validación y envío de datos.

[Yup](#) es una biblioteca de validación de esquemas en JavaScript que se utiliza comúnmente para validar datos de entrada, como formularios, configuraciones y otros objetos estructurados. Yup permite definir esquemas de validación de manera sencilla y clara, lo que facilita la implementación de reglas de validación y la detección de errores en los datos.

Las características clave de Yup incluyen:

1. **Definición de esquemas:** Puedes definir un esquema de validación que describe la estructura y las reglas de validación para tus datos. Esto incluye la validación de tipos de datos, valores mínimos o máximos, patrones de cadena y más.
2. **Validación en cascada:** Yup realiza validaciones en cascada, lo que significa que detendrá la validación en cuanto se detecte el primer error. Esto es útil para proporcionar comentarios rápidos al usuario.
3. **Mensajes de error personalizables:** Puedes personalizar los mensajes de error que se muestran cuando la validación falla, lo que permite una experiencia de usuario más amigable.
4. **Validación asincrónica:** Yup admite la validación asincrónica, lo que es útil cuando necesitas realizar validaciones que dependen de llamadas a API u otras operaciones asincrónicas.
5. **Integración con formularios y aplicaciones de React:** Yup es comúnmente utilizado con bibliotecas de gestión de formularios en React, como Formik, para proporcionar validación de datos en aplicaciones web.
6. **Extensible y personalizable:** Puedes crear esquemas de validación complejos y personalizados según tus necesidades, lo que lo hace altamente extensible.
7. **Documentación y comunidad activa:** La biblioteca tiene una documentación sólida y una comunidad activa de usuarios y desarrolladores que comparten ejemplos y soluciones.

Yup se ha convertido en una elección popular para la validación de datos en aplicaciones web y es ampliamente utilizado en combinación con bibliotecas de gestión de formularios y otras tecnologías de frontend y backend. Su facilidad de uso y potencia lo hacen útil en una variedad de casos de uso donde se necesita validación de datos.



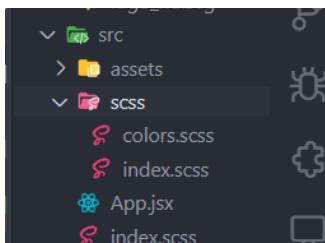
Ahora vamos a combinar estas dos dependencias que nos permitirán validar muchas cosas dentro de nuestro aplicativo de una manera rápida y sencilla, vamos a instalar, ejecutamos el siguiente comando en nuestra terminal del cliente **yarn add yup formik**



```
LAPTOPHDMI D:\DATA\..\..\client yarn 18.16.0 7ms
{ @hdtolledo } yarn add yup formik
yarn add v1.22.19
```

VARIABLES DE SASS

Vamos a empezar definir unas variables de SASS globales en donde vamos a definir nuestros colores y demás, para ello vamos a crear una carpeta dentro de **src** que se llamará **scss** y dentro colocaremos dos archivos, **colors.scss** e **index.scss**



Y dentro de nuestro archivo **colors.scss** vamos a dejar la estructura base de la siguiente manera:

```
src > scss > colors.scss > ...
1 $primary: #1890ff;
2 $primary-hover: #0280b3;
3
4 // Text
5 $text-light: #ffff;
6
7 // Background
8 $background-dark: #001b35;
9 $background-grey: #f0f3f4;
10 $background-dark-web: #16212b;
11
12 // Border
13 $border-grey: #808080;
14
15 // Status
16 $success: #84b84c;
17 $error: #9f3a38;
18
19 // Social Color
20 $youtube: #cd201f;
21 $twitter: #1da1f2;
22 $facebook: #3b5998;
23 $linkedin: #0077b5;
```

Notemos que estamos manejando variables a través del signo **\$** de esta manera lo declaramos. Ahora vamos a nuestro **index.scss** y vamos a realizar la importación de colors:

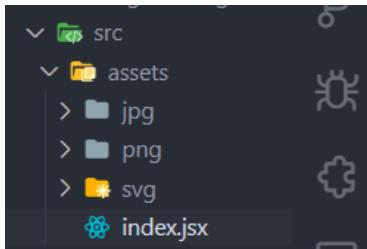
```
src > index.scss
1 @import "./colors.scss";
2
3
```

De esta manera dejaremos nuestro archivo.



ASSETS

En esta ubicación vamos a dejar nuestros archivos de imágenes predefinidos como logo, fondos, avatares y demás, dentro de **src** crearemos la carpeta **assets** y dentro dejaremos la siguiente estructura:



También pueden descargar el archivo de **assets** desde [acá](#). Ahora vamos a dejar nuestro **index.jsx** de **assets** de la siguiente manera:

```
src > assets > index.jsx > ...
1 import iconLogo from "./svg/logo_icon.svg"
2 import authBg from "./jpg/auth-bg.jpg"
3 import homeBanner from "./jpg/home-banner.jpg"
4 import noAvatar from "./jpg/no-avatar.jpg"
5 import academyLogo from "./png/academy-logo.png"
6
7
8 const image = {
9   authBg,
10  homeBanner,
11  noAvatar,
12  academyLogo,
13}
14
15 export { iconLogo, image }
```

Realizamos las importaciones de nuestros archivos de imagen, en específico las imágenes de formato jpg, png las vinculamos a una **const** que se llama **image** y dentro de este lo pasamos como un objeto, para nuestro **logo svg** vamos a exportarlo directamente y lo vamos a utilizar a través de una etiqueta img mas adelante haciendo desestructuración y por último hacemos la exportación del **componente** y de **image**.

INSTALANDO REACT ROUTER DOM

[React Router DOM](#) es una biblioteca de enrutamiento para aplicaciones web desarrolladas con React. Su principal propósito es permitir la navegación y la gestión de las rutas (URL) en una aplicación React de una manera declarativa y amigable. Aquí hay un resumen de para qué sirve React Router DOM y por qué es importante:

1. **Gestión de rutas:** React Router DOM facilita la creación de rutas para tu aplicación. Puedes definir rutas para diferentes componentes de React y asignar URL a cada ruta. Por ejemplo, puedes tener una ruta para la página de inicio (/), una para un perfil de usuario (/profile), y así sucesivamente.
2. **Enrutamiento declarativo:** React Router DOM utiliza una sintaxis declarativa para definir rutas. Esto significa que puedes declarar las rutas como componentes de React y vincularlos directamente a las vistas que deseas mostrar cuando el usuario navega a una URL específica. Esto hace que el enrutamiento sea más claro y fácil de mantener.
3. **Manejo de historial de navegación:** React Router DOM gestiona automáticamente el historial de navegación del navegador. Esto significa que los usuarios pueden usar los botones "atrás" y "adelante" del navegador para navegar por tu aplicación web como lo harían en cualquier otro sitio.
4. **Navegación programática:** Puedes realizar la navegación programáticamente desde tu código JavaScript. Esto es útil para redirigir a los usuarios después de realizar una acción, como enviar un formulario o autenticarse.
5. **Gestión de parámetros en las rutas:** React Router DOM permite la captura de parámetros en las URL, lo que es útil para crear rutas dinámicas. Por ejemplo, puedes tener una ruta como /user/:id y capturar el valor de id en tu componente.
6. **Rutas anidadas:** Puedes crear rutas anidadas para representar jerarquías en tu aplicación. Por ejemplo, podrías tener una ruta principal para el tablero de control y luego rutas secundarias para las diferentes secciones del tablero.
7. **Enrutamiento protegido:** Puedes usar React Router DOM para implementar enrutamiento protegido, lo que significa que ciertas rutas solo son accesibles para usuarios autenticados.

React Router DOM es esencial para crear aplicaciones web de una sola página (SPA) basadas en React y asegura que la navegación y la gestión de rutas sean fluidas y eficientes. Facilita la creación de aplicaciones web complejas con una navegación suave y una experiencia del usuario más agradable.

Vamos a realizar la instalación para ello en nuestro terminal del cliente vamos a colocar lo siguiente **yarn add react-router-dom**

```
LAPTOPHDMI D:\DATA\...\client yarn 18.16.0 5ms
{ @hdtledo } yarn add react-router-dom
U 1:22:19
```

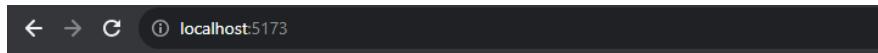
Subimos nuestro servidor del cliente y debemos verlo funcionando correctamente en el navegador.



```
LAPTOPHDMI ➜ D:\DATA\...\client ➜ yarn 18.16.0 1ms
{ hdtledo } yarn dev
yarn run v1.22.19
$ vite

  VITE v4.5.0 ready in 263 ms

  + Local: http://localhost:5173/
  + Network: use --host to expose
  + press h to show help
```



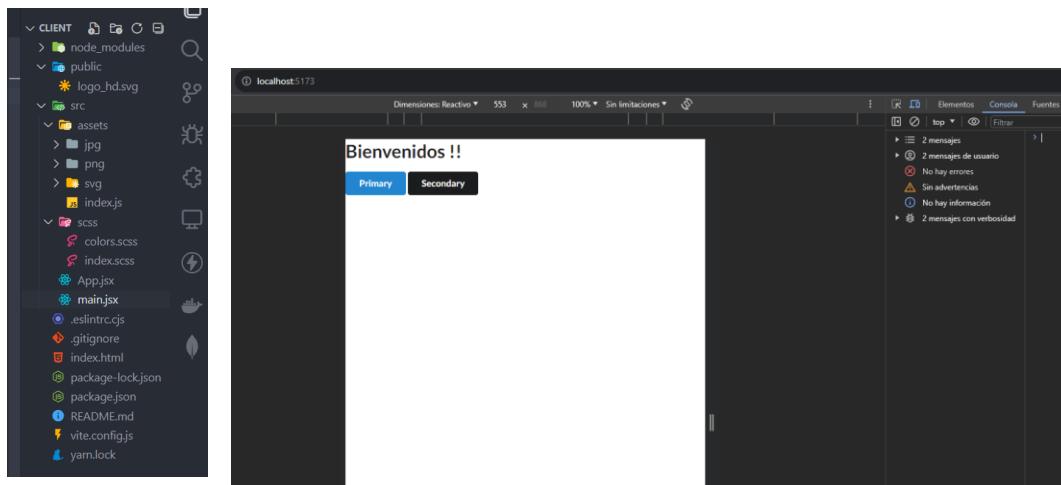
Bienvenidos !!

Primary Secondary

Por ultimo realizamos un pequeño cambio de nuestro **main.jsx** en el cual cambiaremos la ruta del **index.scss**:

```
src > main.jsx
1 import React from 'react'
2 import ReactDOM from 'react-dom/client'
3 import App from './App.jsx'
4 import './scss/index.scss'
5
6 ReactDOM.createRoot(document.getElementById('root')).render(
7   <App />
8 )
9
```

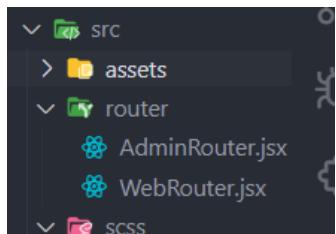
Y podemos eliminar el **index.scss** que está por fuera de nuestra carpeta **scss**



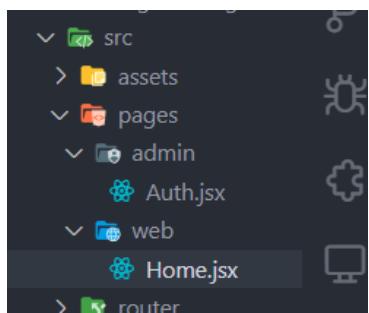
@hdtledo

SEPARANDO ZONA ADMIN DE LA WEB

Ahora dentro de **src** vamos a crear una carpeta que se llamará **router** y dentro crearemos **AdminRouter.jsx** y **WebRouter.jsx**



Y también vamos a crear dentro de **src** una carpeta llamada **pages** y dentro colocaremos dos más, **admin** que tendrá el archivo **Auth.jsx** y la carpeta **web** que tendrá el archivo **Home.jsx**



Dentro de **Auth.jsx** vamos a estructurar un componente funcional:

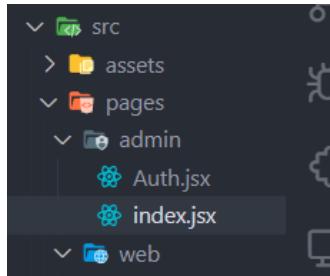
```
src > pages > admin > Auth.jsx
1   rfc
      █ rfc                         reactFunctionalComponent
      █ rfce                         reactFunctionalExportComponent
```

Eliminamos el default del componente y creamos un h1 con un texto:

```
src > pages > admin > Auth.jsx > ...
1   import React from 'react'
2
3   export function Auth() {
4       return (
5           <div>
6               <h1>Estamos Ubicados en Auth</h1>
7           </div>
8       )
9   }
10
```



La idea es poder pasar nuestros componentes a través de un index principal que se encargue de facilitarnos la entrega de estos, así que vamos a crear dentro de **admin index.jsx**



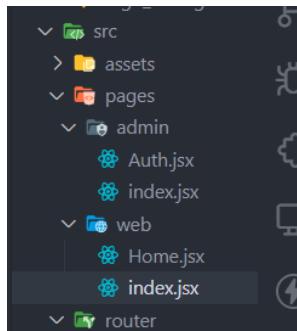
Y dentro estructuramos con una exportación:

```
src > pages > admin > index.jsx
1   export * from "./Auth"
```

Ahora nos ubicamos en **Home.jsx** y creamos de la misma manera:

```
src > pages > web > Home.jsx > ...
1   import React from 'react'
2
3   export function Home() {
4     return (
5       <div>
6         <h1>Estamos en Home</h1>
7       </div>
8     )
9   }
10
```

Y creamos nuestro **index.jsx** y hacemos la exportación:



```
src > pages > web > index.jsx
1  export * from "./Home"
```

De esta forma vamos solo a importar desde una sola ubicación nuestro contenido y aplicarlo de una manera más ordenada.

Ahora vamos a ubicarnos en **WebRouter.jsx** y lo vamos a estructurar de esta manera:

```
src > router > WebRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { Home } from "../pages/web"
4
5  export function WebRouter() {
6    return (
7      <Routes>
8        <Route path="/" element={<Home />} />
9      </Routes>
10     )
11   }
12 }
```

Ahora estructuramos un componente funcional en **AdminRouter.jsx**:

```
src > router > AdminRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { Auth } from "../pages/admin"
4
5  export function AdminRouter() {
6    return (
7      <Routes>
8        <Route path="/admin/*" element={<Auth />} />
9      </Routes>
10     )
11   }
12 }
```

Ahora vamos a aplicar la misma lógica con un **index.jsx** dentro de **router**:



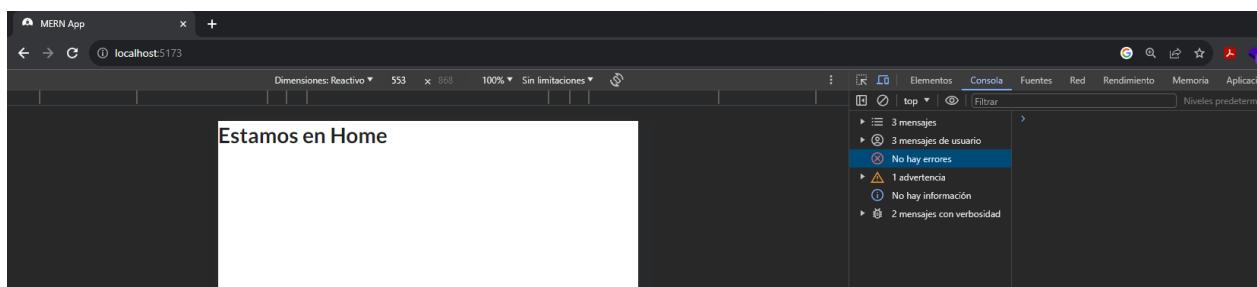
Y dentro lo dejamos así:

```
src > router > index.jsx
1  export * from "./AdminRouter"
2  export * from "./WebRouter"
3
```

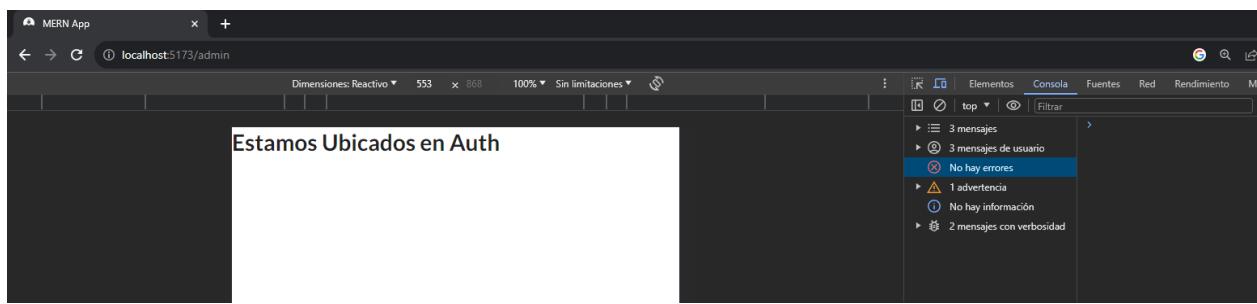
Y ahora vamos a hacer que funcione, para ello vamos a aplicarle el sistema de rutas dentro de **App.jsx** y lo dejamos así:

```
src > App.jsx > ...
1 import React from "react"
2 import {BrowserRouter} from "react-router-dom"
3 import {WebRouter, AdminRouter} from "./router"
4
5 export default function App() {
6   return (
7     <BrowserRouter>
8       <WebRouter />
9       <AdminRouter />
10    </BrowserRouter>
11  )
12}
13
```

Ahora revisamos en nuestro navegador en la ubicación principal y nos sale de esta manera:



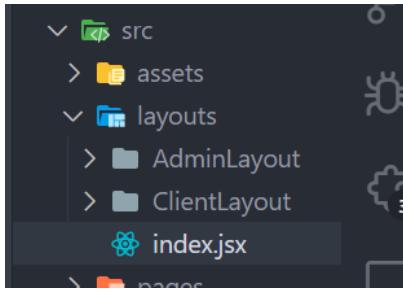
Y si colocamos la ubicación para /admin:



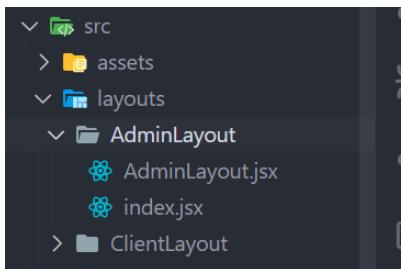
@hdtoledo

AÑADIENDO LAYOUTS A LAS RUTAS

Ahora vamos a implementar un sistema de **layouts** dentro de nuestras rutas para que podamos tener el mismo diseño ya sea para **admin** o para **cliente**, así que vamos a crear nuestros **layouts**, lo primero será crear una carpeta **layouts** dentro de **src** y allí vamos a dejar la siguiente jerarquía:



Recordemos que estamos realizando nuestros **layouts** de una manera ordenada en donde creamos dos carpetas independientes para poder realizar nuestros componentes tanto de **admin** como de **client**; nuestro **index.jsx** será el encargado de exportar todo el contenido de nuestros **layouts**, ahora dentro de **./AdminLayout/** vamos a dejar la siguiente estructura:



Los **index** principales serán los que nos permitirán exportar los **layouts**, ahora dentro de cada **index** debemos exportar lo que viene del **layout**, para nuestro **index** de **AdminLayout** lo dejaremos así:

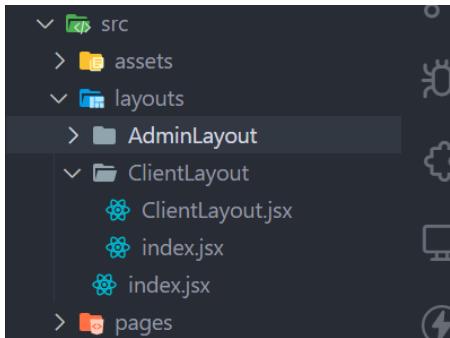
```
src > layouts > AdminLayout > index.jsx
1   export * from "./AdminLayout"
```

Y en nuestro **AdminLayout.jsx** vamos a dejar la siguiente estructura por el momento:

```
src > layouts > AdminLayout > AdminLayout.jsx > ...
1   import React from 'react'
2
3   export function AdminLayout(props) {
4     const { children } = props
5     return (
6       <div>
7         <h2>Se esta usando el AdminLayout</h2>
8         {children}
9       </div>
10    )
11  }
12 }
```

El componente **AdminLayout** sirve como una plantilla o diseño común que se puede utilizar para páginas o secciones de una aplicación destinadas a administradores. Ayuda a mantener la estructura y la apariencia consistentes en todas estas páginas al proporcionar un encabezado estático y permitir la inserción de contenido específico a través de las propiedades **children**.

Esta misma estructura la mantendremos en nuestro **ClientLayout**:



En nuestro **index.jsx** de **ClientLayout** lo dejaremos así:

```
src > layouts > ClientLayout > index.jsx
1   export * from './ClientLayout'
```

Y para nuestro **ClientLayout.jsx** lo dejaremos así:

```
src > layouts > ClientLayout > ClientLayout.jsx > ...
1   import React from 'react'
2
3   export function ClientLayout(props) {
4     const { children } = props
5     return (
6       <div>
7         <h2>Estamos en ClientLayout</h2>
8         {children}
9       </div>
10    )
11  }
```

Ahora para nuestro **index** de principal de carpetas lo dejaremos con las exportaciones de los dos **layouts**:

```
src > layouts > index.jsx
1   export * from './AdminLayout'
2   export * from './ClientLayout'
```



Nuestro siguiente paso será cargarlo en nuestro **AdminRouter.jsx** y vamos a hacer las siguientes modificaciones:

```
src > router > AdminRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { AdminLayout } from "../layouts"
4  import { Auth } from "../pages/admin"
5
6  export function AdminRouter() {
7
8      const loadLayout = (Layout, Page) => {
9          return (
10             <Layout>
11                 <Page />
12             </Layout>
13         )
14     }
15
16     return (
17         <Routes>
18             <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
19         </Routes>
20     )
21 }
22 |
```

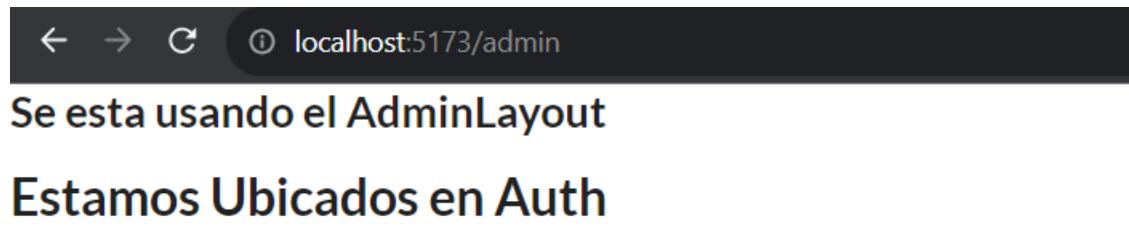
1. **Importación de módulos y componentes:** Al comienzo del código, importamos las bibliotecas y componentes necesarios. Esto incluye React y las funcionalidades de React Router. También importamos dos componentes: **AdminLayout** y **Auth**.
2. **Definición del componente AdminRouter:** Creamos un componente de función llamado **AdminRouter**. Este componente se encargará de gestionar las rutas de las páginas de administrador en tu aplicación.
3. **Función loadLayout:** Definimos una función llamada **loadLayout** que acepta dos argumentos, **Layout** y **Page**. Esta función se utilizará para cargar una página dentro de un diseño específico. En este contexto, **Layout** es **AdminLayout** (el diseño de administrador) y **Page** es **Auth** (la página de autenticación).
4. **Configuración de rutas con <Routes>:** Utilizas el componente **<Routes>** proporcionado por React Router para definir las rutas de la aplicación. Este componente encapsula todas las rutas que deseamos manejar.
5. **Configuración de una ruta específica con <Route>:** Dentro de **<Routes>**, definimos una sola ruta (o regla) usando el componente **<Route>**. Esta ruta corresponde a cualquier URL que comience con **/admin/**. La propiedad **path** se establece en **"/admin/*"**, lo que significa que coincide con cualquier ruta que tenga **/admin/** seguido de cualquier cosa.
6. **Elemento de la ruta y función loadLayout:** En la configuración de la ruta, la propiedad **element** se establece en una llamada a la función **loadLayout**. Esto significa que cuando un usuario



navega a una ruta que coincide con "**/admin/***", se utilizará la función **loadLayout** para determinar qué se debe renderizar.

7. **Renderización del diseño y la página:** Dentro de la función **loadLayout**, primero se renderiza el componente **Layout** (en este caso, **AdminLayout**). Este es el diseño común para todas las páginas de administrador. Luego, se coloca dentro de este diseño el componente **Page** (en este caso, **Auth**). Esto crea la estructura de un diseño (**AdminLayout**) que rodea la página de administración (**Auth**).

De esta manera aplicamos la lógica a nuestro **Adminlayout** ahora si revisamos en nuestro navegador a través de la ruta de **Admin** observaremos lo siguiente:



A screenshot of a browser window. The address bar shows "localhost:5173/admin". The main content area displays the text "Se esta usando el AdminLayout" followed by "Estamos Ubicados en Auth".

Y si colocamos en la dirección cualquier cosa después de **admin**, observaremos que ya tenemos control sobre dicha ruta:



A screenshot of a browser window. The address bar shows "localhost:5173/admin/ajhsgdalksd". The main content area displays the text "Se esta usando el AdminLayout" followed by "Estamos Ubicados en Auth".

Y sino dirigimos a Home nos saldrá lo siguiente:



A screenshot of a browser window. The address bar shows "localhost:5173". The main content area displays the text "Estamos en Home".

Pero aun no nos muestra nuestro **layout**, porque aún no lo configuramos así que ahora vamos a realizar la configuración de nuestro cliente, vamos a nuestro **WebRouter.jsx**, y vamos a dejar prácticamente la misma funcionalidad, obviamente cambiando ciertas cosas para nuestro cliente:



```
src > router > WebRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { ClientLayout } from "../layouts/ClientLayout"
4  import { Home } from "../pages/web"
5
6  export function WebRouter() {
7
8      const loadLayout = (Layout, Page) => {
9          return (
10             <Layout>
11                 <Page />
12             </Layout>
13         )
14     }
15
16     return (
17         <Routes>
18             <Route path="/" element={ loadLayout(ClientLayout, Home)} />
19         </Routes>
20     )
21 }
22 }
```

Si nos vamos para nuestro navegador vamos a observar como se actualiza nuestro layout:

← → ⌂ ⓘ localhost:5173

Estamos en ClientLayout

Estamos en Home

De esta manera dejamos la configuración básica para nuestra aplicación.



@hdtoledo

SEPARANDO AUTH DEL ADMINPANEL

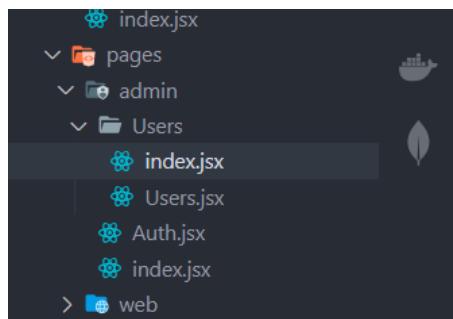
Ahora vamos a separar nuestro **login/register** y poder hacer la autenticación, recordemos que para acceder a estas rutas debemos estar logueados para poder entrar a las rutas protegidas, para ello vamos a realizar la protección de estas rutas.

Nos vamos a ubicar en **AdminRouter.jsx**, y vamos a cargar una constante con **user** para simular el ingreso mientras hacemos todo, realizamos lo siguiente:

```
src > router > AdminRouter.jsx > AdminRouter
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { AdminLayout } from "../layouts"
4  import { Auth } from "../pages/admin"
5
6  const user = null
7
8  export function AdminRouter() {
9
10    const loadLayout = (Layout, Page) => {
11      return (

```

Ahora vamos a crear otra página para los usuarios y que nos permita validar el panel **admin** dentro de nuestra ruta, vamos a **pages/admin** y creamos la siguiente estructura:



Dejamos nuestro **index.jsx** y el **Users.jsx** dentro de la carpeta **Users**, y ahora dentro de la estructura inicial creamos un componente para nuestro **Users.jsx**:

```
src > pages > admin > Users > Users.jsx > ...
1  import React from 'react'
2
3  export function Users() {
4    return (
5      <div>
6        <h1>Estamos en Users - Admin</h1>
7      </div>
8    )
9  }
10
```



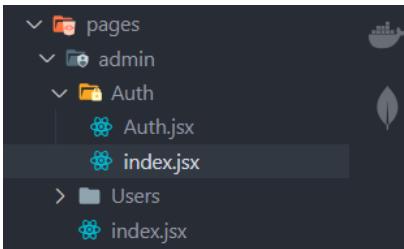
Y en nuestro **index.jsx** hacemos la exportación de contenidos:

```
src > pages > admin > Users > index.jsx
1   export * from "./Users"
```

Y hacemos también la exportación de nuestro **index – Users** en el index de **admin**

```
src > pages > admin > index.jsx
1   export * from "./Auth"
2   export * from "./Users"|
```

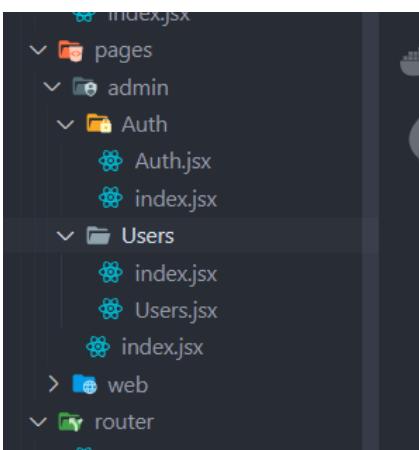
Y vamos a dejar dentro de **admin** la carpeta para **auth** y movemos el archivo **Auth.jsx** para esa ubicación:



Ajustamos nuestro index de auth:

```
src > pages > admin > Auth > index.jsx
1   export * from "./Auth"
```

De esta manera recordemos como queda la estructura de nuestro admin:



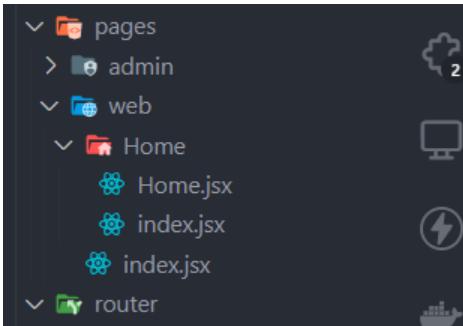
Si vamos a nuestro navegador debemos verificar que este todo funcional:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

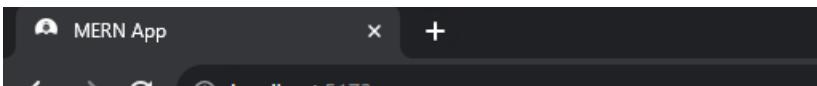
Ahora vamos a realizar el mismo procedimiento para nuestra carpeta **web** en la cual vamos a dejar la configuración de la siguiente manera:



Dentro de la carpeta **Home**, movemos nuestro **Home.jsx** y creamos el archivo **index.jsx**, ahora vamos a dejar dentro de nuestro **index** la exportación de **Home**:

```
src > pages > web > Home > index.jsx
1  export * from "./Home"
```

Al revisar en nuestro navegador en la ruta **Home** debemos ver nuevamente todo correcto:



Estamos en ClientLayout

Estamos en Home

Ahora que ya tenemos nuestras carpetas configuradas vamos a empezar a modificar **AdminRouter.jsx**, de esta manera, primero realizamos la importación de **Users** en conjunto con **Auth**, recordemos que estamos desestructurando:

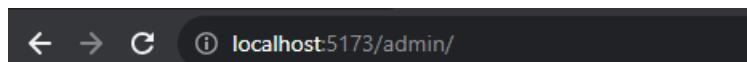
```
src > router > AdminRouter.jsx > AdminRouter
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { AdminLayout } from "../layouts"
4  import { Auth, Users } from "../pages/admin"
5
6  const user = null
```

De esta manera es mucho mas sencillo de aplicar las importaciones de nuestros **layouts** o **componentes**. Ahora analicemos un poco como esta actuando nuestro **Router**, desde el navegador puedo ingresar a ambas rutas sin importar los permisos y de esta manera no debemos hacer, cuando **user** sea **null** solo deberia ingresar a nuestro **Home** y en el caso contrario si **user** trae informacion es porque estamos logueados.

Para ello vamos a colocar una condicion en **Routes** de la siguiente manera:

```
17
18     return (
19       <Routes>
20         {!user ? (
21           <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
22         ) : (
23           <>
24             <Route path="/admin/users" element={ loadLayout(AdminLayout, Users) } />
25           </>
26         )}
27       </Routes>
28     )
29   }
30 }
```

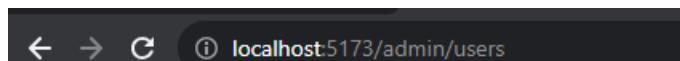
En donde especificamos si no trae nada **user** que nos muestre solo el **admin**, pero si trae algo que nos permita ingresar a nuestra ruta de **/admin/users/**, vamos a comprobarlo en nuestro navegador:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

Hasta aca todo va bien, pero si intentamos ingresar en **/admin/users/**:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

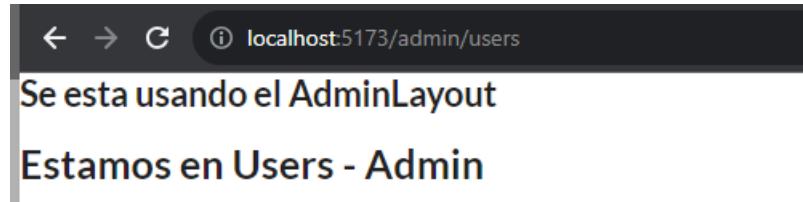
No nos deja avanzar de allí, ahora comprobemos sobre esta misma ruta pero agregando contenido a nuestra const **user**:

```
4   import { Auth, Users } from "../pages/admin"
5
6   const user = { email: "test@test.com" }
7
8   export function AdminRouter() {
9 }
```



@hdtoledo

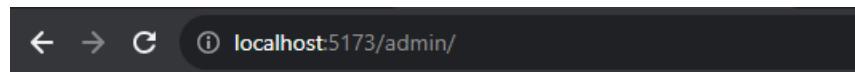
Al volver a refrescar la página observamos que ya nos permite ingresar a nuestro layout de `/admin/users/`:



A screenshot of a browser window. The address bar shows `localhost:5173/admin/users`. The main content area displays the text "Se esta usando el AdminLayout" and "Estamos en Users - Admin".

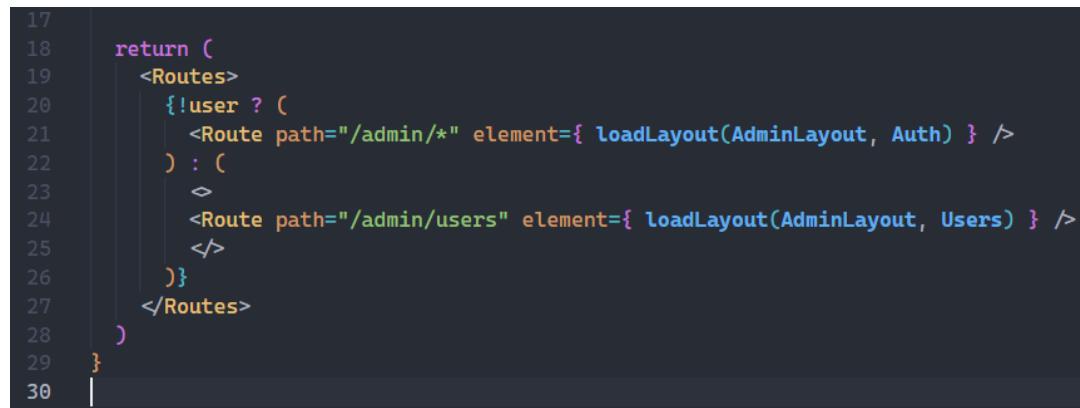
UNA PAGINA DOS PATH

Observemos que si nos ubicamos solo en `/admin/` vamos a notar que no nos sale nada:



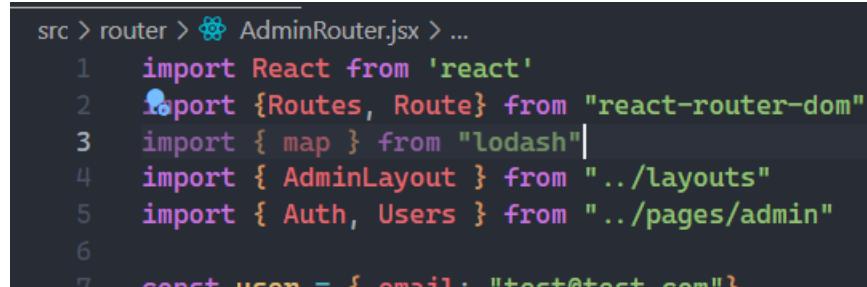
A screenshot of a browser window. The address bar shows `localhost:5173/admin/`. The main content area is completely blank.

Y esto se debe a que en este momento no hemos indicado la ruta de `/admin/`:



```
17
18     return (
19       <Routes>
20         {!user ? (
21           <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
22         ) : (
23           <>
24             <Route path="/admin/users" element={ loadLayout(AdminLayout, Users) } />
25           </>
26         )}
27       </Routes>
28     )
29   }
30 }
```

Lo que vamos a realizar para poder aplicarle una mejor lógica es crear un bucle de array de path, de esta manera sin importar que rutas tengamos el array se encargara de ello y no tendremos que agregarlo de modo manual, ahora realizamos lo siguiente:



```
src > router > AdminRouter.js > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { map } from "lodash"
4  import { AdminLayout } from "../layouts"
5  import { Auth, Users } from "../pages/admin"
6
7  const user = { email: "test@test.com" }
```



@hdtoledo

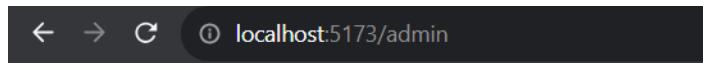
Importamos **map** de **lodash**, para poder realizar lo planteado utilizaremos esta dependencia, ahora dejaremos de la siguiente manera nuestro bucle:

```
19  return (
20    <Routes>
21      {!user ? (
22        <Route path="/admin/*" element={ LoadLayout(AdminLayout, Auth) } />
23      ) : (
24        <>
25          {[ "/admin", "/admin/blog" ].map((path) => (
26            <Route key={path} path={path} element={ LoadLayout(AdminLayout, Users) } />
27          )));
28        <Route path="/admin/users" element={ LoadLayout(AdminLayout, Users) } />
29      </>
30    )}
31  </Routes>
32}
33}
34}
```

Hagamos un resumen de lo aplicado:

1. **Definición de la función AdminRouter:** Se inicia declarando una función llamada **AdminRouter** que se encarga de gestionar las rutas de la sección de administración de la aplicación.
2. **Función loadLayout:** Se define una función llamada **loadLayout** que toma dos argumentos, **Layout** y **Page**. Esta función se utiliza para cargar una página (**Page**) dentro de un diseño común (**Layout**).
3. **Configuración de rutas con <Routes>:** Se utiliza el componente **<Routes>** para definir las rutas de la aplicación. Esto encapsula todas las rutas que la aplicación manejará.
4. **Condición de usuario (variable user):** Se verifica si existe un usuario (**!user**). Si no hay un usuario autenticado, se configura una ruta que comienza con **"/admin/*"** para cargar el diseño **AdminLayout** alrededor de la página de autenticación (**Auth**).
5. **Rutas para usuarios autenticados:** Si existe un usuario autenticado, se configuran varias rutas. Esto incluye rutas para **"/admin"** y **"/admin/blog"**, que cargan el diseño **AdminLayout** alrededor de la página de usuarios (**Users**). También hay una ruta para **"/admin/users"** que hace lo mismo.

Ahora si verificamos en nuestro navegador debe funcionar de la misma manera sin problemas y ya tendremos la ruta para solo **/admin/**:



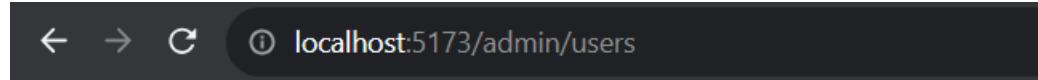
Se está usando el AdminLayout

Estamos en Users - Admin



@hdtoledo

Y si vamos a `/admin/users/`



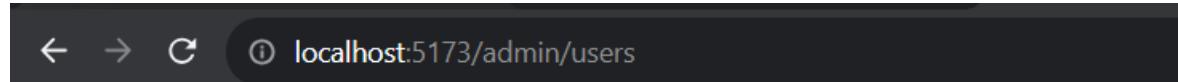
Se esta usando el AdminLayout

Estamos en Users - Admin

Si verificamos quitando la propiedad de email de user:

```
● 5 import { Auth, Users } from "../pages/admin"
  6
  7 const user = null
  8
  9 export function AdminRouter() {
10 }
```

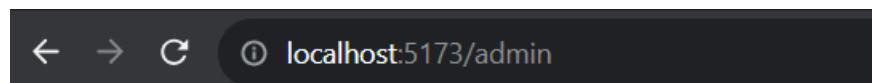
Y vamos al navegador:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

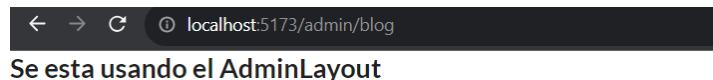
Y lo mismo para admin:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

Ahora tenemos agregada la ruta de nuestro blog también, la cual configuramos más adelante, pero si vamos a esa ruta:



Se esta usando el AdminLayout

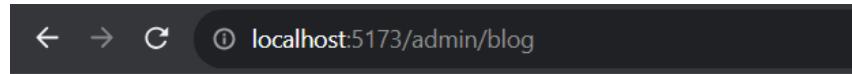
Estamos Ubicados en Auth



Y si volvemos a habilitar la propiedad de user:

```
4 import { AdminLayout } from "../layouts"
5 import { Auth, Users } from "../pages/admin"
6
7 const user = { email: "test@test.com" }
8
9 export function AdminRouter() {
10 }
```

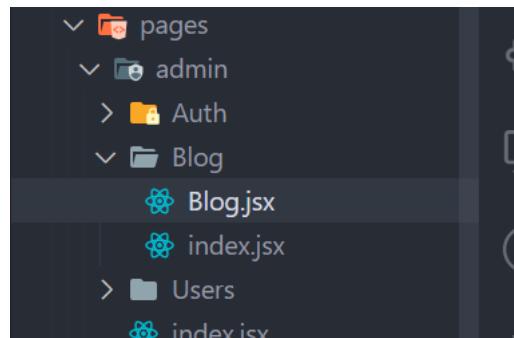
Y revisamos en el navegador:



Se esta usando el AdminLayout

Estamos en Users - Admin

Ahora vamos a crear la pagina de blog, para ello nos ubicamos en **/pages/admin/** y creamos blog con sus respectivos archivos **index.jsx** y **Blog.jsx**:



Dejamos en el index la exportación:

```
src > pages > admin > Blog > index.jsx
1   export * from "./Blog"
```

Y en nuestro **Blog.jsx** creamos un componente base:

```
src > pages > admin > Blog > Blog.jsx > ...
1   import React from 'react'
2
3   export function Blog() {
4     return (
5       <div>
6         <h1>Estamos en Blog</h1>
7       </div>
8     )
9   }
10
```



@hdtoledo

De igual manera en `/admin/index.jsx` hacemos la exportación del contenido:

```
src > pages > admin > index.jsx
1  export * from './Auth/Auth"
2  export * from './Users"
3  export * from './Blog"
```

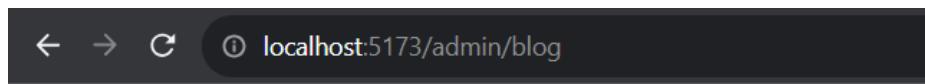
Ahora dentro de `/router/AdminRouter.jsx` importamos `Blog`:

```
src > router > AdminRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { map } from "lodash"
4  import { AdminLayout } from "../layouts"
5  import { Auth, Users, Blog } from "../pages/admin"
6
7  const user = { email: "test@test.com"}
```

Y lo colocamos aca sobre `Blog`:

```
19  return (
20    <Routes>
21      {user ? (
22        <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
23      ) : (
24        <>
25          {[ "/admin", "/admin/blog" ].map((path) => (
26            <Route key={path} path={path} element={ loadLayout(AdminLayout, Blog) } />
27          ))}
28          <Route path="/admin/users" element={ loadLayout(AdminLayout, Users) } />
29        </>
30      )}
31    </Routes>
32  )
33}
34}
```

Ahora si vamos a revisar en nuestro navegador la ruta de blog:



```
localhost:5173/admin/blog
```

Se esta usando el `AdminLayout`

Estamos en Blog

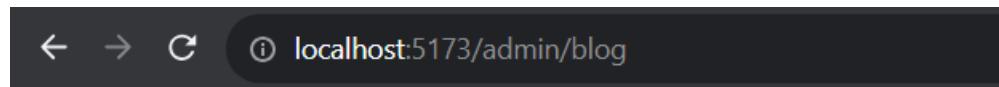
Recordemos que si quitamos las propiedades de user:

```
5  import { Auth, Users } from "../pages/admin"
6
7  const user = null
8
9  export function AdminRouter() {
10  }
```



@hdtoledo

Y vamos al navegador observaremos que nos lleva a Auth:



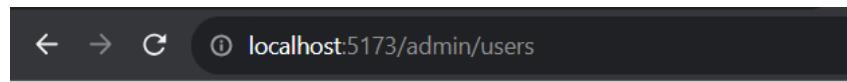
Se esta usando el AdminLayout

Estamos Ubicados en Auth

Volvemos a activar nuestras propiedades:

```
5 import { Auth, Users, Blog } from "../pages/admin"
6
7 const user = { email: "test@test.com" }
8
9 export function AdminRouter() {
10 }
```

Y si vamos a /admin/users/



Se esta usando el AdminLayout

Estamos en Users - Admin

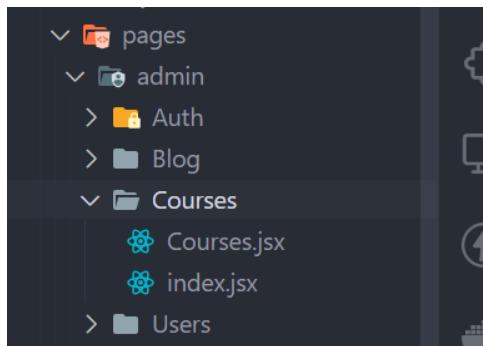
De esta manera estamos cargando una misma página en dos rutas distintas.



@hdtoledo

CREANDO LAS PAGINAS DEL ADMIN

Para terminar de dejar la estructura base nos falta un par de paginas para el **admin**, asi que vamos a crear dentro de **/admin** la ruta de **Courses** con la misma estructura que venimos manejando:



Dentro de **Courses.jsx**:

```
src > pages > admin > Courses > Courses.jsx > ...
1  import React from 'react'
2
3  export function Courses() {
4      return (
5          <div>
6              <h1>Estamos en Cursos</h1>
7          </div>
8      )
9  }
10
```

Y dentro de **index**:

```
src > pages > admin > Courses > index.jsx
1  export * from "./Courses"
```

Y en nuestro **/admin/index.jsx**

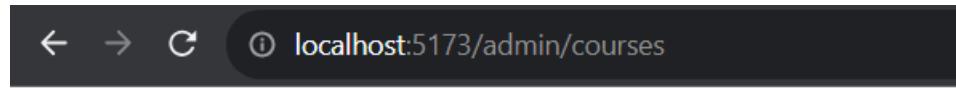
```
src > pages > admin > index.jsx
1  export * from "./Auth/Auth"
2  export * from "./Users"
3  export * from "./Blog"
4  export * from "./Courses"
```



Ahora en nuestro `/router/AdminRouter.jsx`:

```
19  return (
20    <Routes>
21      {!user ? (
22        <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
23      ) : (
24        <>
25          {[ "/admin", "/admin/blog" ].map((path) => (
26            <Route key={path} path={path} element={ loadLayout(AdminLayout, Blog) } />
27          ))}
28          <Route path="/admin/users" element={ loadLayout(AdminLayout, Users) } />
29          <Route path="/admin/courses" element={ loadLayout(AdminLayout, Courses) } />
30        </>
31      )}
32    </Routes>
33  )
34}
35
```

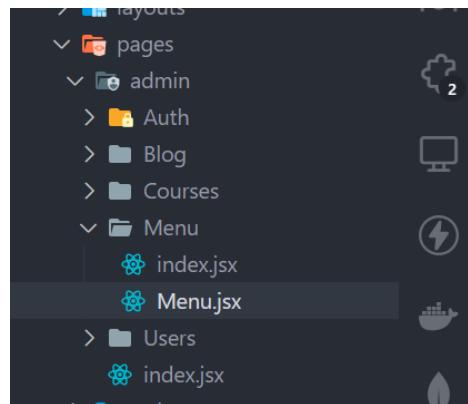
Y si nos vamos a nuestro navegador en la ruta **Courses**:



Se esta usando el AdminLayout

Estamos en Cursos

Ahora vamos a crear la ruta de nuestros menus, para ello aplicamos la misma base anterior creamos las carpetas con los archivos:



Dentro de index dejamos la exportación:

```
src > pages > admin > Menu > index.jsx
1   export * from "./Menu"
```



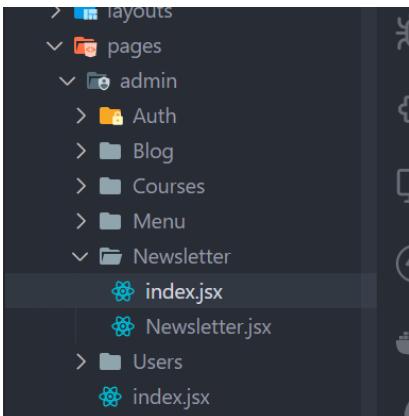
Y dentro de **Menu.jsx** dejamos el componente inicial:

```
src > pages > admin > Menu >  Menu.jsx > ...
1  import React from 'react'
2
3  export function Menu() {
4    return (
5      <div>
6        <h1>Menu</h1>
7      </div>
8    )
9  }
10
```

Hacemos la exportación en **/admin/index.jsx**

```
src > pages > admin >  index.jsx
1  export * from "./Auth/Auth"
2  export * from "./Users"
3  export * from "./Blog"
4  export * from "./Courses"
5  export * from "./Menu"
```

Ahora vamos a crear la ruta de nuestro NewsLetter, y las iniciamos con la creación de la carpeta y sus archivos iniciales:



En nuestro index hacemos la exportacion:

```
src > pages > admin > Newsletter >  index.jsx
1  export * from "./Newsletter"
```



Y en nuestro `Newsletter.jsx`:

```
src > pages > admin > Newsletter > Newsletter.jsx > ...
1 import React from 'react'
2
3 export function Newsletter() {
4   return (
5     <div>
6       |   <h1>Estamos en newsletter</h1>
7     </div>
8   )
9 }
10
```

Y exportamos en `/admin/index.jsx`

```
src > pages > admin > index.jsx
1 export * from './Auth/Auth"
2 export * from './Users"
3 export * from './Blog"
4 export * from './Courses"
5 export * from './Menu"
6 export * from './Newsletter"
```

Ahora en nuestro `/router/AdminRouter.jsx` importamos a Menu y Newsletter:

```
src > router > AdminRouter.jsx > ...
1 import React from 'react'
2 import {Routes, Route} from "react-router-dom"
3 import { map } from "Lodash"
4 import { AdminLayout } from "../layouts"
5 import { Auth, Users, Blog, Courses, Menu, Newsletter } from "../pages/admin"
6
7 const user = { email: "test@test.com" }
8
```

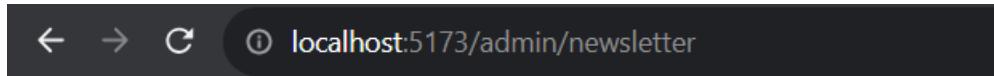
Y los llamamos de la siguiente manera:

```
19 return (
20   <Routes>
21     {!user ? (
22       <Route path="/admin/*" element={ loadLayout(AdminLayout, Auth) } />
23     ) : (
24       &
25       {[ "/admin", "/admin/blog" ] .map((path) => (
26         <Route key={path} path={path} element={ loadLayout(AdminLayout, Blog) } />
27       )))
28       <Route path="/admin/users" element={ loadLayout(AdminLayout, Users) } />
29       <Route path="/admin/courses" element={ loadLayout(AdminLayout, Courses) } />
30       <Route path="/admin/menu" element={ loadLayout(AdminLayout, Menu) } />
31       <Route path="/admin/newsletter" element={ loadLayout(AdminLayout, Newsletter) } />
32     )
33   )
34 </Routes>
35 )
36
37
```



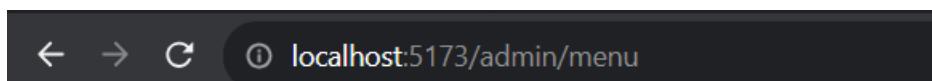
@hdtoledo

Ahora si todo va bien podemos observar cómo se nos muestran en nuestro navegador las diferentes rutas:



Se esta usando el AdminLayout

Estamos en newsletter



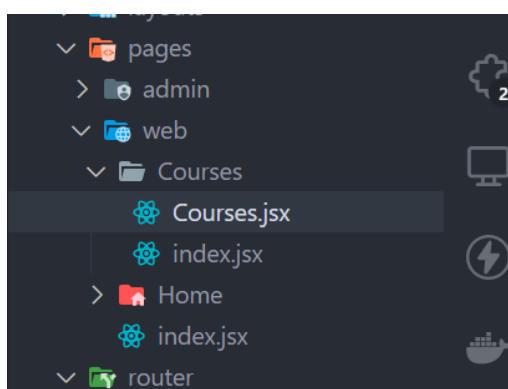
Se esta usando el AdminLayout

Menu

Estas serían nuestras páginas básicas de la aplicación.

CREANDO LAS PAGINAS DE LA WEB

Ahora el siguiente paso es dejar establecidas las páginas de nuestro cliente al momento de visualizar en la web, como lo son la Home, Cursos , Contacto, páginas de errores, así que nos ubicamos en Web y procedemos a crear la carpeta de Courses y mantenemos la base de los archivos:



@hdtoledo

Dentro de Courses.jsx:

```
src > pages > web > Courses > Courses.jsx > ...
1 import React from 'react'
2
3 export function Courses() {
4   return (
5     <div>
6       <h1>Estamos en Cursos Web</h1>
7     </div>
8   )
9 }
10
```

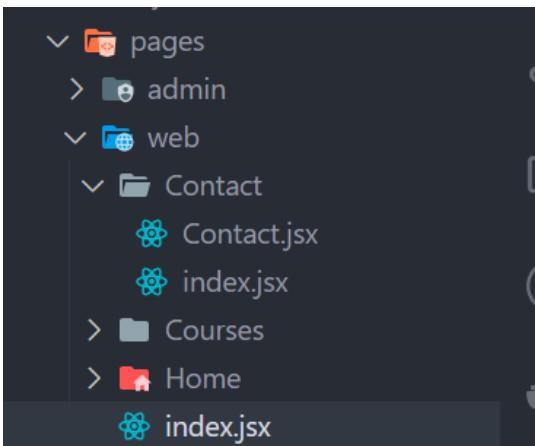
Y dentro de index:

```
src > pages > web > Courses > index.jsx
1 export * from "./Courses"
```

Recordemos que en /web/index.jsx exportamos:

```
src > pages > web > index.jsx
1 export * from "./Home"
2 export * from "./Courses"
```

Ahora realizamos Contact y mantenemos la misma estructura:



@hdtoledo

Dentro de Contact.jsx:

```
src > pages > web > Contact > Contact.jsx > ...
1 import React from 'react'
2
3 export function Contact() {
4     return (
5         <div>
6             <h1>Estamos en Contacto</h1>
7         </div>
8     )
9 }
10
```

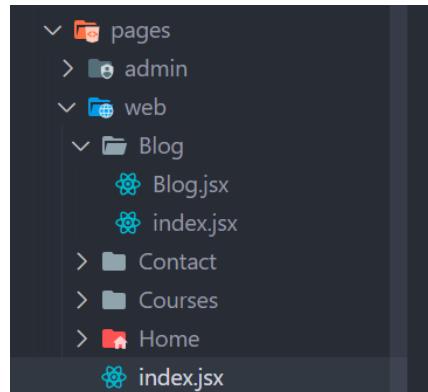
Y dentro de index:

```
src > pages > web > Contact > index.jsx
1 export * from "./Contact"
```

Y no olvidemos en nuestro /web/index.jsx:

```
src > pages > web > index.jsx
1 export * from "./Home"
2 export * from "./Courses"
3 export * from "./Contact"
```

Ahora realizamos el montaje de la misma manera para Blog:



@hdtoledo

Para nuestro Blog.jsx:

```
src > pages > web > Blog > Blog.jsx > ...
1  import React from 'react'
2
3  export function Blog() {
4    return (
5      <div>
6        <h1>Estamos en Blog Web</h1>
7      </div>
8    )
9  }
10
```

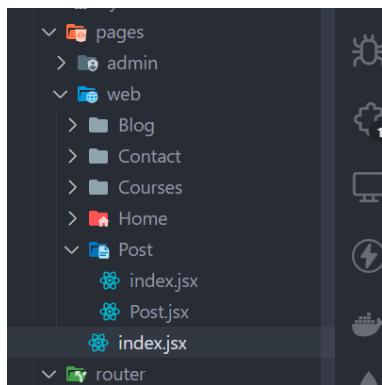
Para nuestro index.jsx:

```
src > pages > web > Blog > index.jsx
1  export * from "./Blog"
```

Y en nuestro /web/index.jsx:

```
src > pages > web > index.jsx
1  export * from "./Home"
2  export * from "./Courses"
3  export * from "./Contact"
4  export * from "./Blog"
```

Ahora realizamos el montaje de la misma manera para Post:



Seguimos con nuestro Post.jsx:

```
src > pages > web > Post > Post.jsx > ...
1  import React from 'react'
2
3  export function Post() {
4      return (
5          <div>
6              <h1>Estamos en Post</h1>
7          </div>
8      )
9  }
10
```

Y con nuestro index.jsx:

```
src > pages > web > Post > index.jsx
1  export * from "./Post"
```

Y en el /web/index.jsx:

```
src > pages > web > index.jsx
1  export * from "./Home"
2  export * from "./Courses"
3  export * from "./Contact"
4  export * from "./Blog"
5  export * from "./Post"
```

Ahora vamos a realizar la importación de nuestras nuevas páginas en **WebRouter.jsx**:

```
src > router > WebRouter.jsx > ...
1  import React from 'react'
2  import {Routes, Route} from "react-router-dom"
3  import { ClientLayout } from "../layouts/ClientLayout"
4  import { Home, Blog, Contact, Courses, Post } from "../pages/web"
5
```



@hdtoledo

Y ahora realizamos las rutas y su cargue:

```
15
16  return (
17    <Routes>
18      <Route path="/" element={ loadLayout(ClientLayout, Home) } />
19      <Route path="/blog" element={ loadLayout(ClientLayout, Blog) } />
20      <Route path="/contact" element={ loadLayout(ClientLayout, Contact) } />
21      <Route path="/courses" element={ loadLayout(ClientLayout, Courses) } />
22      <Route path="/blog/:path" element={ loadLayout(ClientLayout, Post) } />
23    </Routes>
24  )
25}
26
```

Para el caso de nuestro Post, vamos a enviarlo a través de la ruta **/blog/:path** en donde **path** es una variable de las publicaciones a las cuales vamos a acceder, de esta manera ya tenemos nuestras rutas establecidas, ahora vamos a revisar en el navegador:

localhost:5173/blog

Estamos en ClientLayout

Estamos en Blog Web

localhost:5173/contact

Estamos en ClientLayout

Estamos en Contacto

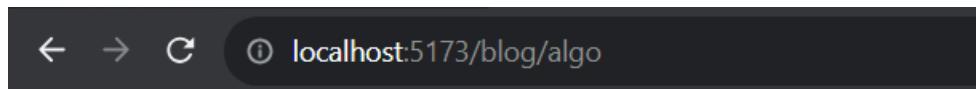
localhost:5173/courses

Estamos en ClientLayout

Estamos en Cursos Web



@hdtoledo



Estamos en ClientLayout

Estamos en Post

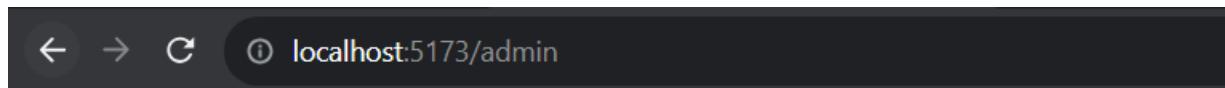
En esta última ruta ya tenemos verificado que funciona nuestro post.

ZONA DE LOGIN Y REGISTRO

Ahora vamos a crear nuestras páginas para el login y registro, lo primero que vamos a realizar es ubicarnos en **AdminRouter.jsx** y vamos a dejar nuestra variable de **user** en **null**:

```
4 import { AdminLayout } from '../layouts'
5 import { Auth, Users, Blog, Courses, Menu, Newsletter
6
7 const user = null
8 |
9 export function AdminRouter() {
10 |
```

Ahora si nos vamos a revisar en nuestro navegador nos encontramos con lo siguiente:



Se esta usando el AdminLayout

Estamos Ubicados en Auth

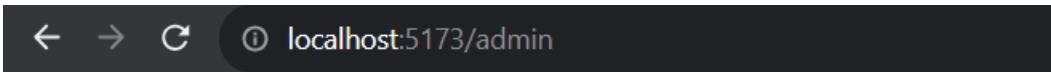
Para lo cual estamos viendo el **AdminLayout** pero este no es necesario, lo que si es necesario es nuestro **Auth** y allí es donde vamos a realizar las validaciones, vamos a modificar nuestra ruta dentro de **AdminRouter.jsx** de la siguiente manera:

```
19 return (
20   <Routes>
21     {!user ? (
22       | <Route path="/admin/*" element={<Auth>} />
23     ) : (
24       |
25         {[ "/admin", "/admin/blog" ]}.map((path) => (
```



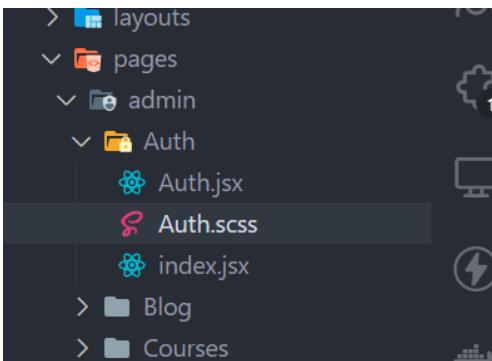
@hdtoledo

De esta manera solo nos mostrara **Auth** y nos quedara de la siguiente manera:



Estamos Ubicados en Auth

Ahora nos ubicamos en **/pages/admin/auth/** y vamos a crear nuestro archivo **Auth.scss**:



Y dentro de **Auth.scss** dejaremos lo siguiente:

```
src > pages > admin > Auth > Auth.scss > ...
1  @import "/src/scss/index.scss";
2
3  .auth {
4      background-color: red;
5  }
```

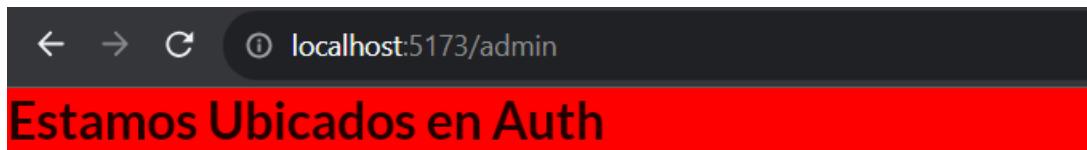
Ahora nos dirigimos a **Auth.jsx** y hacemos la importación de **Auth.scss** y la llamamos:

```
src > pages > admin > Auth > Auth.jsx > ...
1  import React from 'react'
2  import './Auth.scss'
3
4  export function Auth() {
5      return (
6          <div className='auth'>
7              <h1>Estamos Ubicados en Auth</h1>
8          </div>
9      )
10 }
```



@hdtoledo

Y si vamos a nuestro navegador obtendremos lo siguiente:



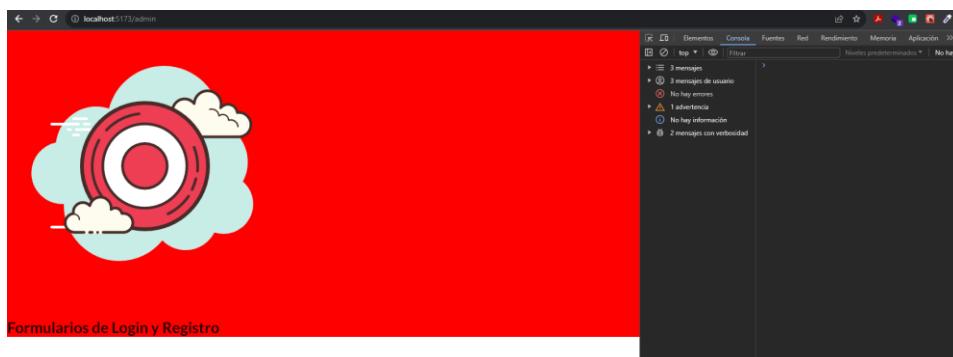
Ahora vamos a cargar una imagen, así que realizamos la importación primero:

```
src > pages > admin > Auth > Auth.jsx > ...
1  import React from "react"
2  import { iconLogo } from "../../../../../assets"
3  import "./Auth.scss"
4
```

Y más abajo lo dejaremos de la siguiente manera:

```
5  export function Auth() {
6    return (
7      <div className="auth">
8        <img src={iconLogo} className="logo"/>
9        <h1>Formularios de Login y Registro</h1>
10       </div>
11     )
12   }
13 }
```

Notemos que estamos haciendo una desestructuración de iconLogo y que lo estamos pasando a través de una etiqueta img con su ruta, además que le colocamos una clase llamada logo, si nos dirigimos al navegador:

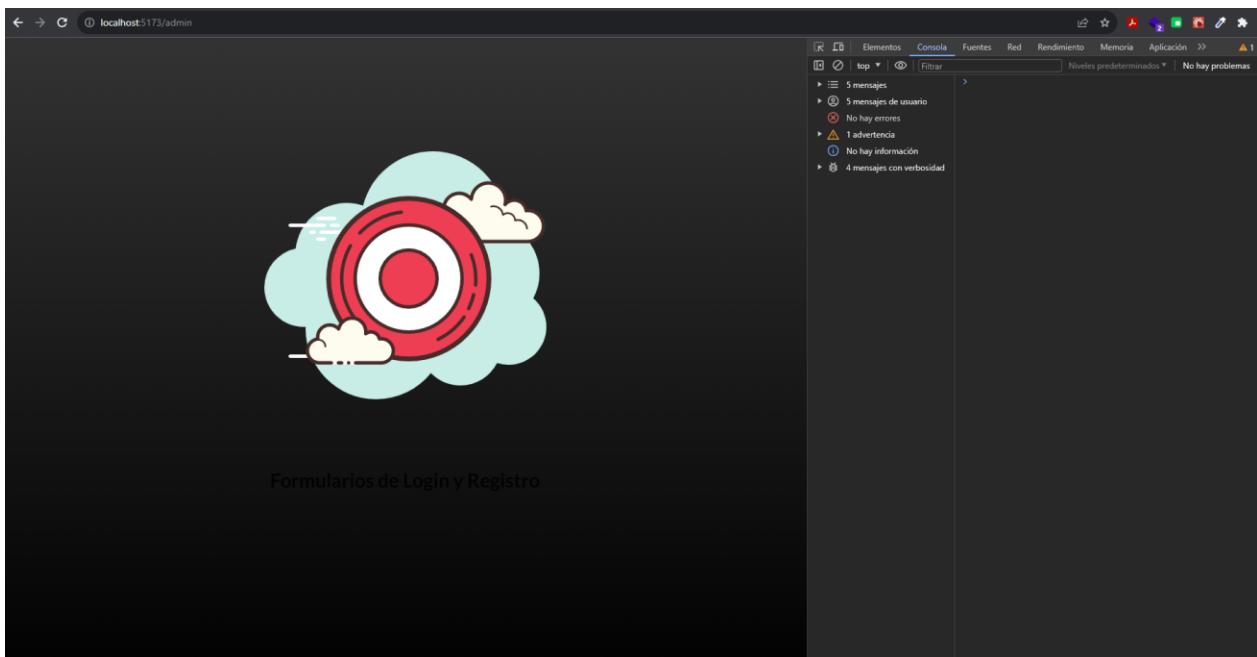


@hdtoledo

Ahora vamos a empezar a estilizar dentro de nuestro **Auth.scss** de la siguiente manera:

```
src > pages > admin > Auth >  Auth.scss >  .auth
1  @import "/src/scss/index.scss";
2
3  .auth {
4      background: linear-gradient(to bottom, #333, #000);
5      background-size: cover;
6      height: 100vh;
7      background-position: center;
8      display: flex;
9      align-items: center;
10     flex-direction: column;
11     padding-top: 100px;
12
13 }
14
15 }
```

Y el resultado en el navegador es el siguiente:



Ahora vamos a ajustar nuestro logo de la siguiente manera:



```
src > pages > admin > Auth >  Auth.scss > ...
1  @import "/src/scss/index.scss";
2
3  .auth {
4      background: linear-gradient(to bottom, #333, #000);
5      background-size: cover;
6      height: 100vh;
7      background-position: center;
8      display: flex;
9      align-items: center;
10     flex-direction: column;
11     padding-top: 100px;
12
13     .logo{
14         width: 100px;
15         margin-bottom: 30px;
16     }
17
18
19 }
20
21 |
```

En Sass colocamos dentro de una clase contenedora la clase logo, a la cual nos va a quedar de esta manera, ahora vamos a importar semantic UI dentro de Auth.jsx:

```
src > pages > admin > Auth >  Auth.jsx >  Auth
1  import React from "react"
2  import {Tab} from "semantic-ui-react"
3  import { iconLogo } from "../../assets"
4  import "./Auth.scss"
5
```

Ahora vamos a realizar lo siguiente en nuestro Auth.jsx:



@hdtoledo

```

5
6  export function Auth() {
7    const panes = [
8      {
9        menuItem: "Entrar",
10       render: () => (
11         <Tab.Pane>
12           <h2>Login Form</h2>
13         </Tab.Pane>
14       )
15     },
16     {
17       menuItem: "Nuevo Usuario",
18       render: () => (
19         <Tab.Pane>
20           <h2>Register Form</h2>
21         </Tab.Pane>
22       )
23     }
24   ]
25
26   return (

```

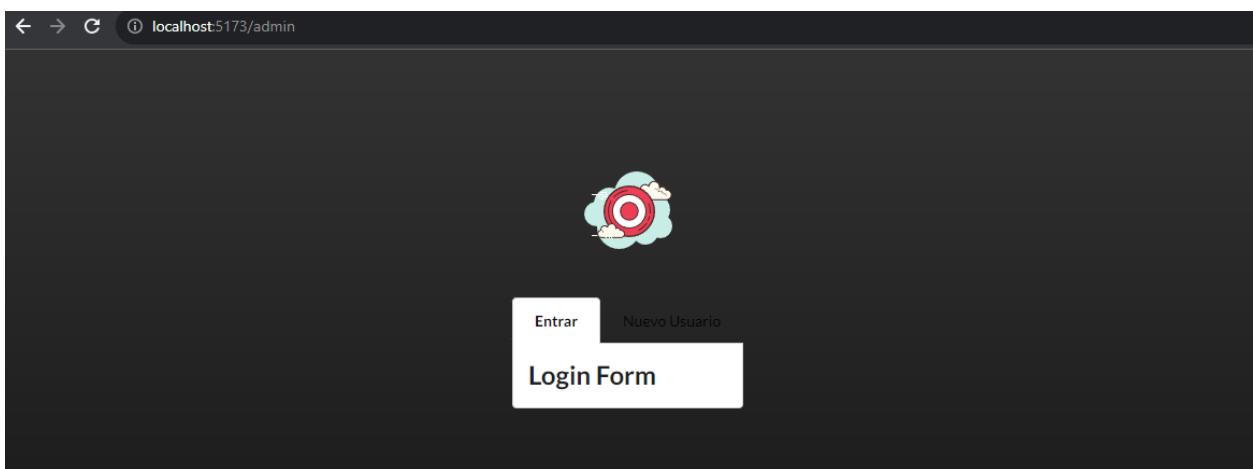
Acá estamos definiendo nuestros paneles a través de panes y dentro colocamos las pestañas de la navegación a la cual vamos a acceder, ahora colocamos más debajo de la siguiente manera:

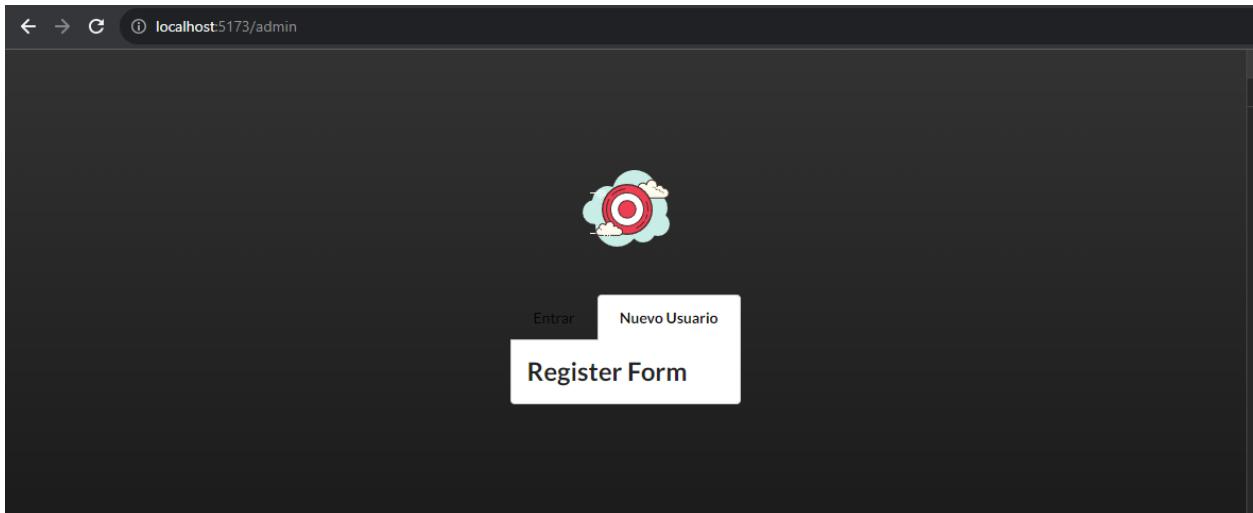
```

25   ]
26
27   return (
28     <div className="auth">
29       <img src={iconLogo} className="logo" />
30       <Tab panes={panes} className="auth__forms" />
31     </div>
32   )
33

```

Y en nuestro navegador lo vamos a observar de la siguiente manera:





Ahora vamos a importar `useState` para poder aplicarle estados a nuestro panel y así saber cual panel esta activo o no:

```
src > pages > admin > Auth > Auth.js > ...
1  import React, { useState } from "react"
2  import { Tab } from "semantic-ui-react"
3  import { iconLogo } from "../../assets"
4  import "./Auth.scss"
5
```

Y dentro de nuestra función vamos a dejar nuestro `useState` de la siguiente manera:

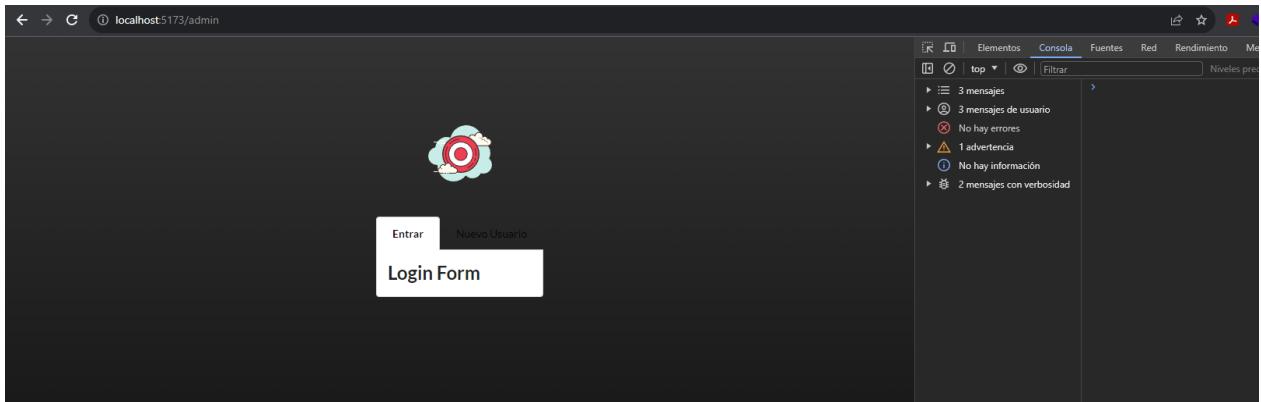
```
3  import { iconLogo } from "../../assets"
4  import "./Auth.scss"
5
6  export function Auth() {
7    const [activeIndex, setActiveIndex] = useState(0)
8
9    const panes = [
10      {
11        menuItem: "Entrar",
```

A través del uso de estado vamos a indicarle si esta activo o no uno de los paneles, para ello vamos a realizar lo siguiente más abajo:

```
28  return (
29    <div className="auth">
30      <img src={iconLogo} className="logo"/>
31      <Tab panes={panes} className="auth__forms" activeIndex={activeIndex}>
32        </div>
33    )
34  }
35
```

Si nos vamos a nuestro navegador vamos a observar lo siguiente:





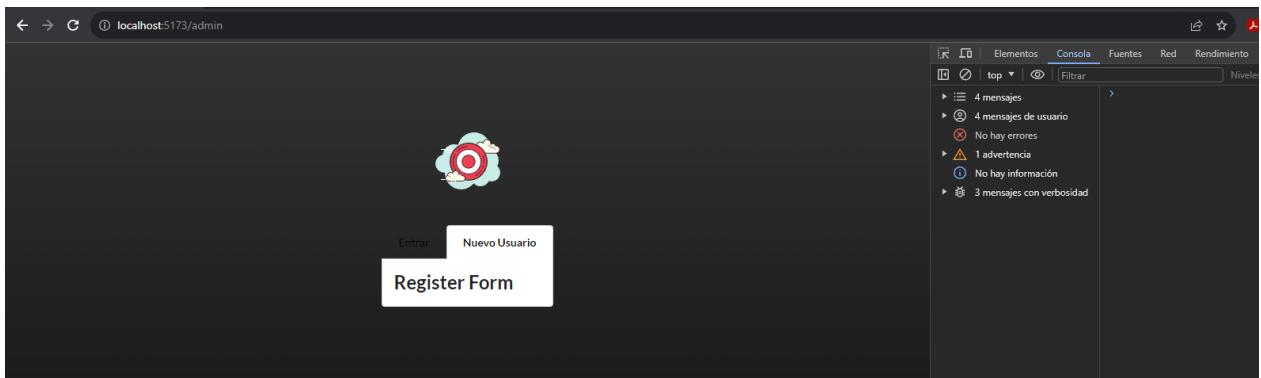
Nuestro login se encuentra activo y el de nuevo usuario no, ahora si lo cambio a 1 el estado:

```

5
6  export function Auth() {
7    const [activeIndex, setActiveIndex] = useState(1)
8
9    const panes = [

```

Vamos a observar que solo nos deja el de registro:



Vamos a dejar una nueva constante agregada en donde ejecutaremos una función tipo flecha que me va a permitir abrir el login, así que la vamos a dejar de la siguiente manera:

```

5
6  export function Auth() {
7    const [activeIndex, setActiveIndex] = useState(1)
8    const openLogin = () => setActiveIndex(0)
9
10   const panes = [

```

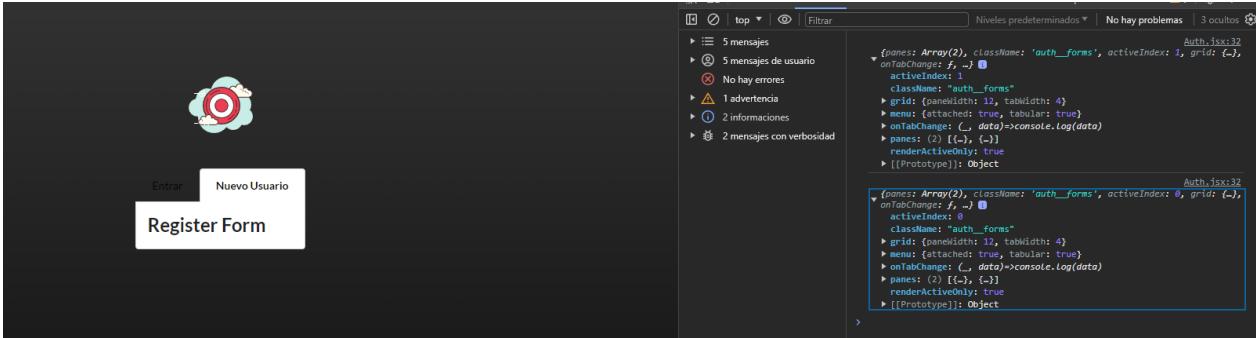
Y para lograr que vuelva a trabajar manualmente al darle click nuestro tab, vamos a realizar lo siguiente:

```

30  <div className="auth">
31    <img src={iconLogo} className="logo"/>
32    <Tab panes={panes} className="auth__forms" activeIndex={activeIndex} onTabChange={(_, data) => console.log(data)} />
33  </div>
34}
35}

```

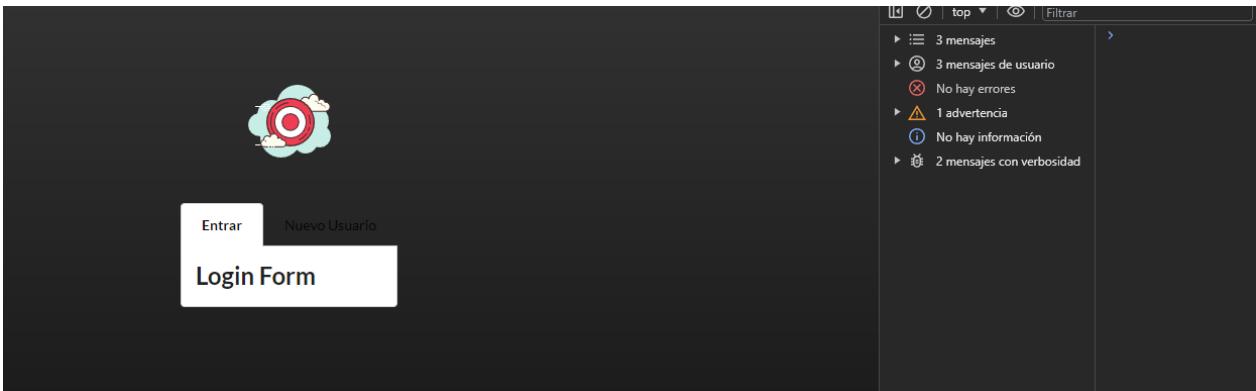
Vamos a agregar nuestro **onTabChange** que nos permitirá saber si se cambia o no de una pestaña a otra, ahora revisemos en nuestro navegador para revisar que nos trae:



Acá podemos observar como en la propiedad **activeIndex** traemos 1 o 0 para indicarnos en cual **tab** se encuentra, ahora lo que vamos a hacer es simplemente indicarle que nos envíe la información de la data del **activeindex** de la siguiente manera:

```
29   return [
30     <div className="auth">
31       <img src={iconLogo} className="logo"/>
32       <Tab panes={panes} className="auth__forms" activeIndex={activeIndex} onTabChange={(_, data) => setActiveIndex(data.activeIndex)} />
33     </div>
34   ]
35 }
36 }
```

De esta manera si volvemos al navegador observaremos que ya podremos cambiar de tab:



Ahora vamos a darle un poco mas de estilos nos dirigimos a nuestro **Auth.scss** y vamos a realizar lo siguiente:



@hdtoledo

```

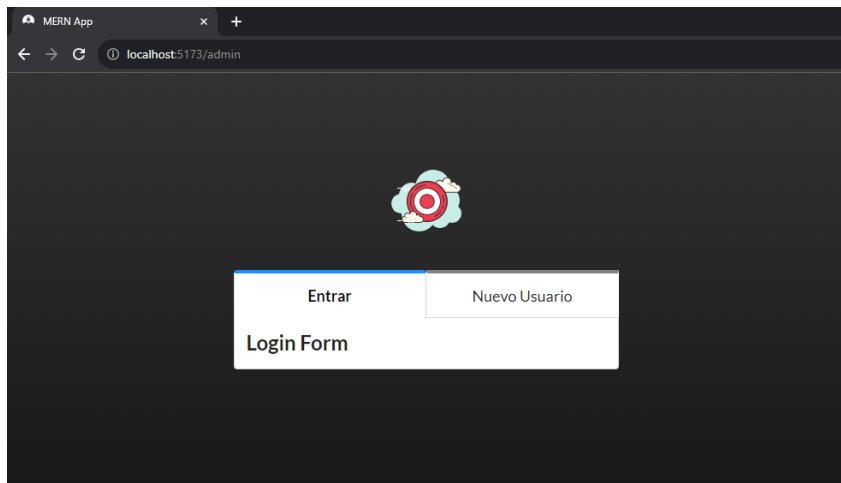
12
13  .logo{
14      width: 100px;
15      margin-bottom: 30px;
16  }
17
18 &__forms {
19     width: 450px;
20 }
21
22 .ui.menu{
23     .item {
24         width: 50%;
25         justify-content: center;
26         background-color: #fff !important;
27         border-radius: 0 !important;
28         border-top: 4px solid $border-grey !important;
29         font-size: 18px;
30     }
31
32     .active {
33         font-weight: normal;
34         border-radius: 0 !important;
35         border-top-color: $primary !important;
36     }
37 }
38
39 }
40
41

```

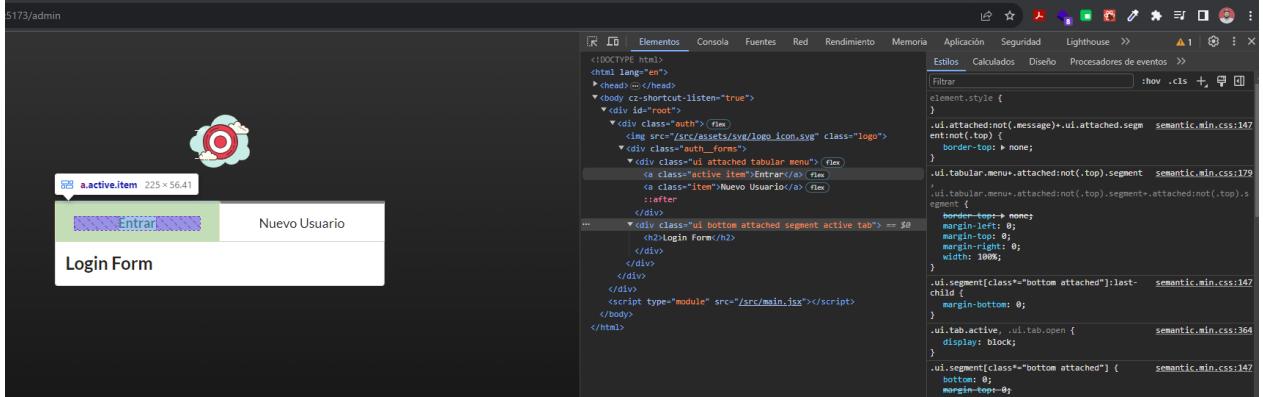
Acá vamos a agregar varias sintaxis nuevas como es el caso de **&__forms** que nos va a permitir añadir un ancho de 450px a nuestra clase, al momento de indicarle con **&** vamos a decirle que nos tome toda la clase padre y acceda a forms para modificarle el ancho.

Luego, tenemos algunas reglas para elementos llamados "item" dentro de un "menú" especial. Les hemos dado un ancho del 50%, un fondo blanco y un borde en la parte superior. Además, si un elemento tiene la clase "active," cambiamos el texto a la fuente normal, quitamos los bordes redondeados y cambiamos el color del borde superior a un color especial.

Ahora si revisamos en nuestro navegador vamos a obtener lo siguiente:

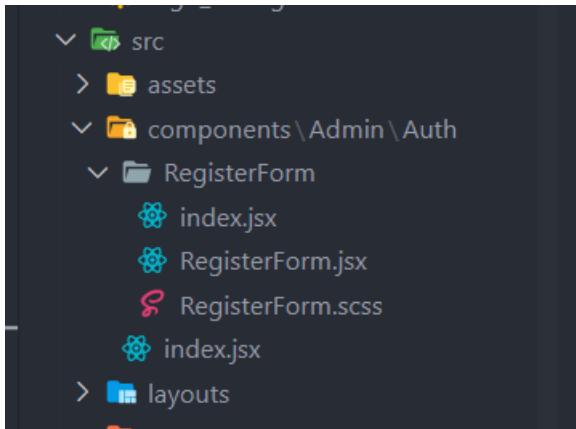


Si revisamos a fondo lo que estamos viendo notaremos las diferentes clases que estamos aplicando sobre el formulario:



FORMULARIO DE REGISTRO

Ahora vamos a crear nuestro componente para el formulario de **login/register**, lo primero es crear una carpeta de nombre **components** dentro de **src** y estructuramos de la siguiente manera
/src/components/Admin/Auth/RegisterForm creamos allí los archivos **index.jsx – RegisterForm.jsx – RegisterForm.scss** :



Ahora vamos a configurar nuestros archivos, el primero de ellos es nuestro **/src/components/Admin/Auth/RegisterForm/index.jsx**:

```
src > pages > admin > Auth > index.jsx
1   export * from "./Auth"
```

En donde exportamos todo el contenido de nuestra ubicación.



Seguimos con nuestro [/src/components/Admin/Auth/RegisterForm/RegisterForm.jsx](#):

```
src > components > Admin > Auth > RegisterForm >  RegisterForm.jsx > ...
1 import React from 'react'
2 import './RegisterForm.scss'
3 |
4 export function RegisterForm() {
5   return (
6     <div>
7       <h2>Formulario de Registro</h2>
8     </div>
9   )
10 }
11
```

En donde creamos un componente funcional para verificar en un momento como va nuestro componente, ahora seguimos con [/src/components/Admin/Auth/RegisterForm/RegisterForm.scss](#):

```
src > components > Admin > Auth > RegisterForm >  RegisterForm.scss
1 @import "/src/scss/index.scss";
2
3
```

En donde hacemos una importación de nuestro **scss**, ahora vamos a crear [/src/components/Admin/Auth /index.jsx](#) :

```
src > components > Admin > Auth >  index.jsx
1 export * from "./RegisterForm"
```

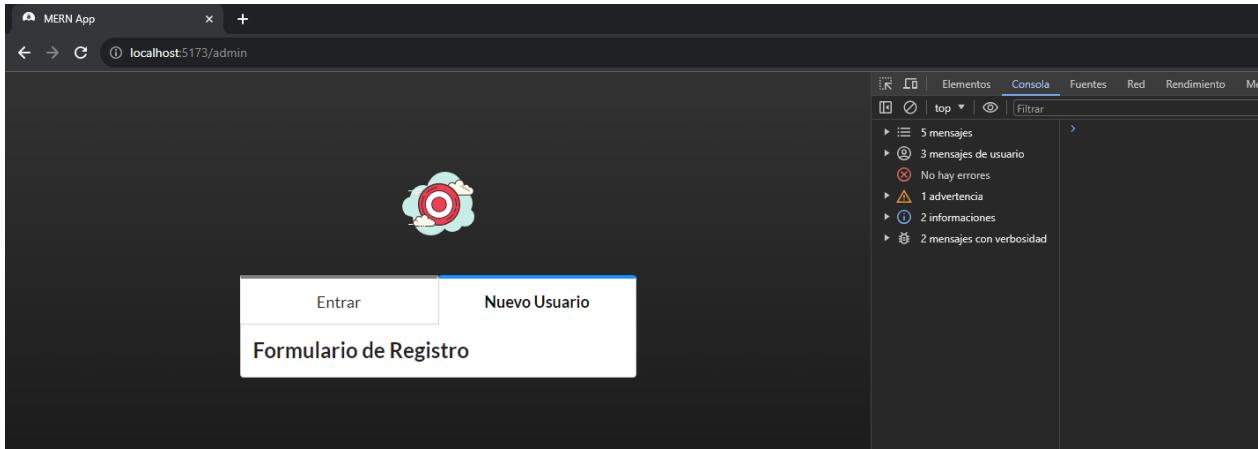
En donde hacemos la exportación de todo lo que se encuentre en RegisterForm, ahora en nuestro Auth.jsx vamos a realizar la importación de nuestro componente:

```
src > pages > admin > Auth >  Auth.jsx > ...
1 import React, { useState } from "react"
2 import { Tab } from "semantic-ui-react"
3 import { iconLogo } from "../../assets"
4 import "./Auth.scss"
5 import { RegisterForm } from "../../components/Admin/Auth/RegisterForm"
6
```

Y vamos a llamarlo de la siguiente manera:

```
21   menuItem: "Nuevo Usuario",
22   render: () => [
23     <Tab.Pane>
24       | <RegisterForm openLogin={openLogin}/>
25     </Tab.Pane>
26   ]
27 }
```

De esta manera vamos a decirle que tan proto se registre inicie la sesión nuestro usuario mediante la función openLogin, si nos vamos a nuestro navegador debemos ver nuestro componente:



Ahora vamos a empezar a estilizar nuestro **RegisterForm.jsx**, lo primero es importar **semantic UI**:

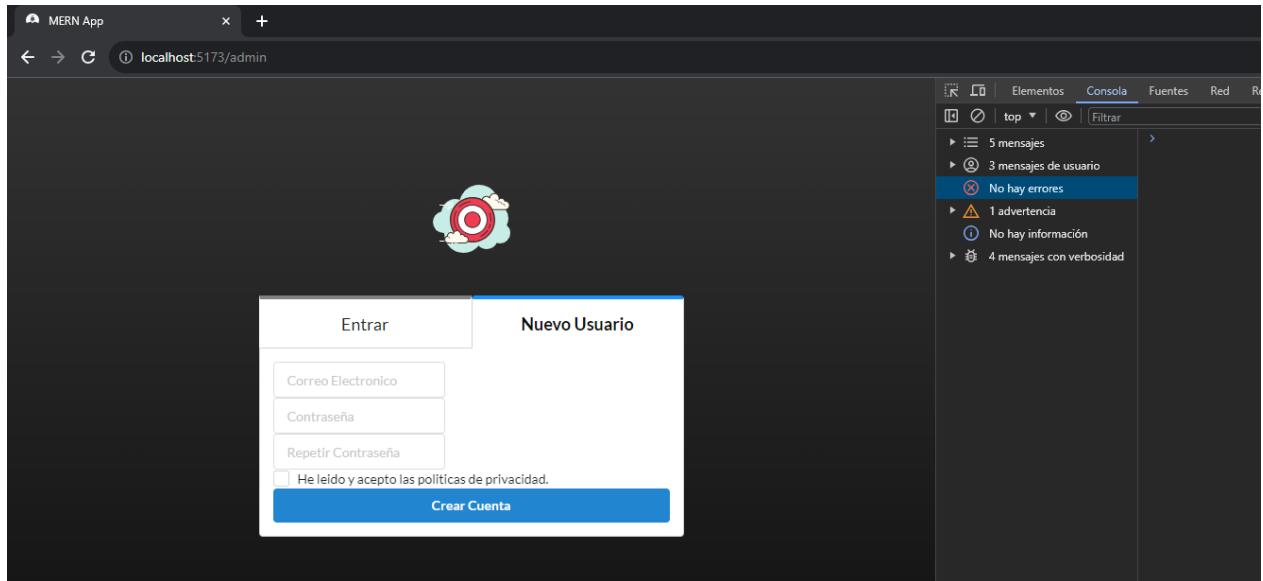
```
src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > ...
1  import React from 'react'
2  import { Form } from "semantic-ui-react"
3  import "./RegisterForm.scss"
4
5
```

Y ahora vamos a dejar nuestro componente de la siguiente manera:

```
20
21  return (
22    <Form className="register-form">
23      <Form.Input name="email" placeholder="Correo Electronico" />
24      <Form.Input name="password" type="password" placeholder="Contraseña" />
25      <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña"/>
26      <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad."/>
27      <Form.Button type="submit" primary fluid>Crear Cuenta</Form.Button>
28
29    </Form>
30
31  }
32
```

De esta manera podremos observar en nuestro navegador lo siguiente:





Ahora vamos a modificar nuestro **RegisterForm.scss**:

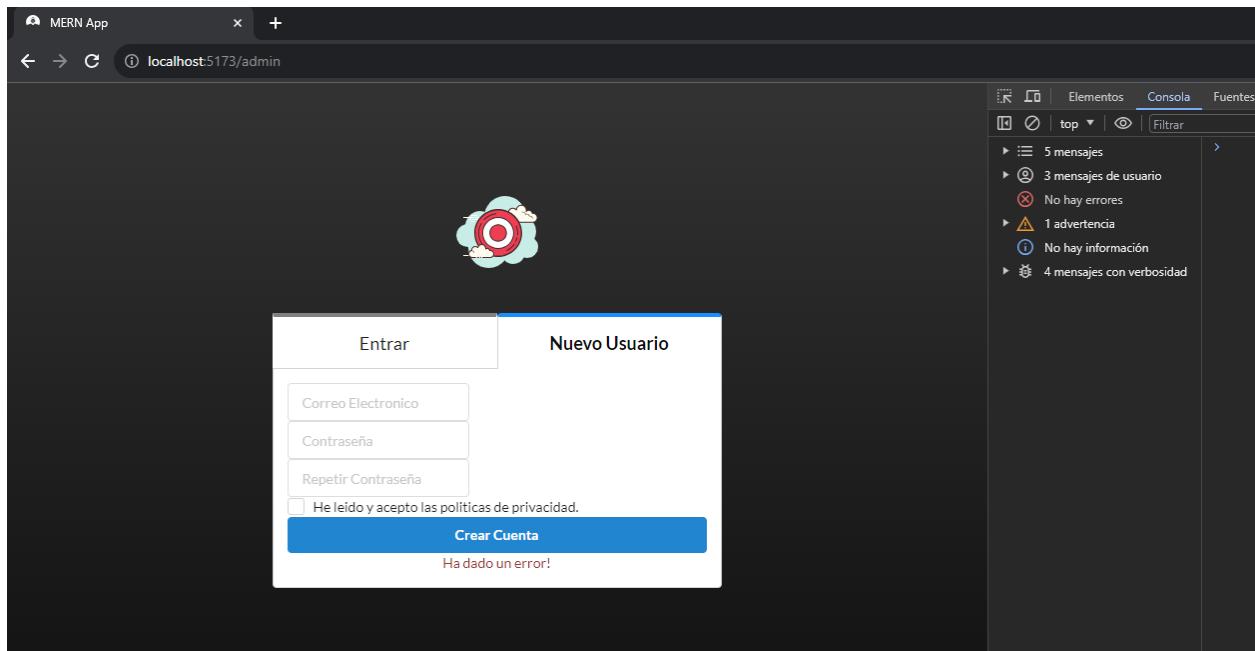
```
src > components > Admin > Auth > RegisterForm > ⚡ RegisterForm.scss > ...
1  @import "/src/scss/index.scss";
2
3  .register-form{
4      &__error {
5          color: $error;
6          text-align: center;
7      }
8  }
9
```

Vamos a colocar la validación de errores para que se nos muestre a través de un mensaje en rojo, para ello vamos a modificar **RegisterForm.jsx** de la siguiente manera:

```
20
21  return (
22      <Form className="register-form">
23          <Form.Input name="email" placeholder="Correo Electronico" />
24          <Form.Input name="password" type="password" placeholder="Contraseña" />
25          <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña"/>
26          <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." />
27          <Form.Button type="submit" primary fluid>Crear Cuenta</Form.Button>
28
29          <p className="register-form__error">Ha dado un error !</p>
30      </Form>
31  )
32
33
```

Agregamos un párrafo con el texto y a este le vamos a colocar un estado para validar, si vamos a nuestro navegador debe salir así:





Ahora vamos a importar el useState en nuestra cabecera:

```
src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > ...
1  import React, { useState } from 'react'
2  import { Form } from "semantic-ui-react"
3  import "./RegisterForm.scss"
```

Para poderlo usar aplicamos la sintaxis de useState:

```
4
5  export function RegisterForm() {
6    const [error, setError] = useState("")
7
8    return (

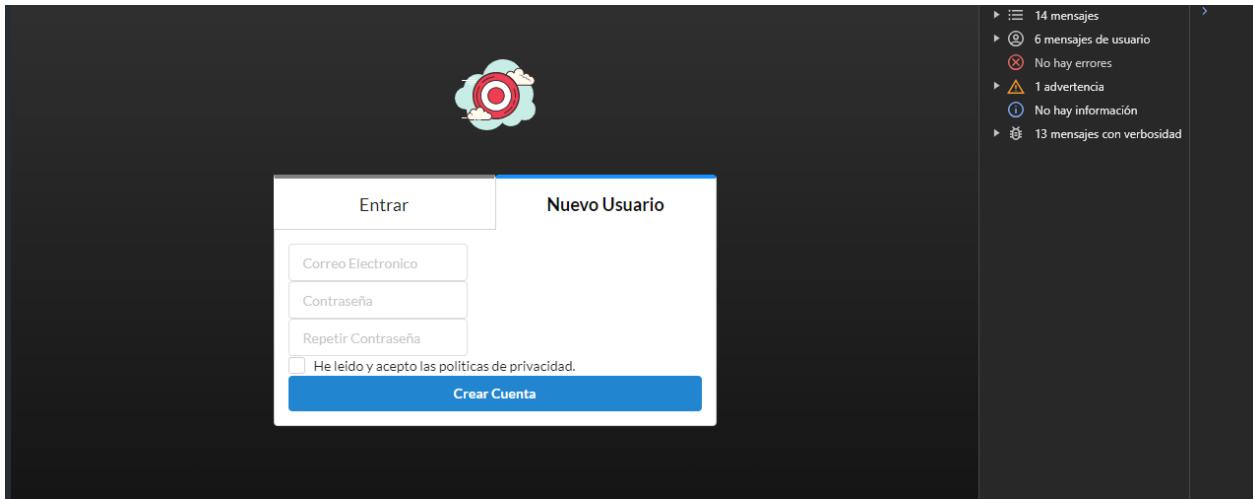
```

Establecemos su uso y en la parte de nuestro párrafo vamos a pasar los mensajes de la siguiente manera:

```
26
27    <Form.Checkbox name="conditionsAccepted" label="He leido y acepto"/>
28    <Form.Button type="submit" primary fluid>Crear Cuenta</Form.Button>
29
30    <p className="register-form__error">{error}</p>
31
32 }
```

Si vamos a verificar no se vera reflejado nada por el momento porque lo tenemos vacío:



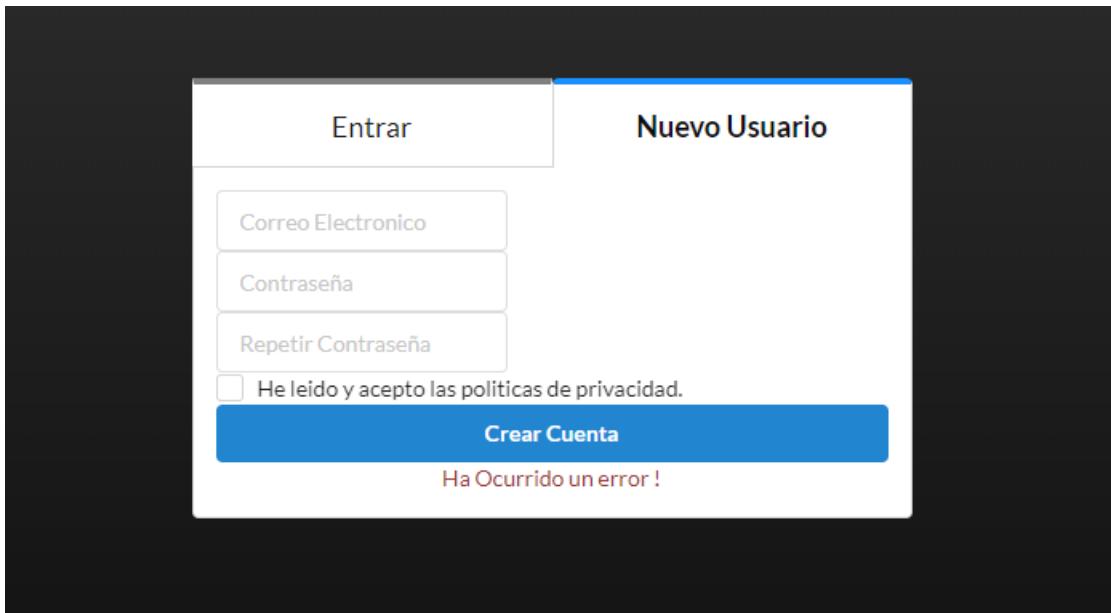


Pero si cambiamos el `useState`:

```
4
5  export function RegisterForm() {
6    const [error, setError] = useState("Ha ocurrido un error !")
7
8    return (

```

Y en nuestro navegador:

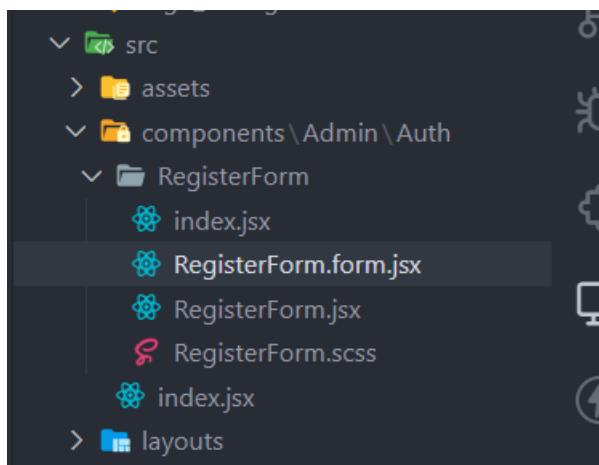


De esta manera vamos a empezar a controlar los errores y que nos muestre en mensaje.



VALIDANDO EL FORMULARIO CON FORMIK

Para realizar la validación y control de nuestro registro lo vamos a realizar a través de formik, así que vamos a empezar con la creación de [/RegisterForm/RegisterForm.form.jsx](#)



en donde vamos a cargar las validaciones que debemos hacer para ello vamos a hacer lo siguiente en nuestro archivo:

```
src > components > Admin > Auth > RegisterForm > RegisterForm.form.jsx > initialValues
1  export function initialValues(){
2    return{
3      email: "",
4      password: "",
5      repeatPassword: "",
6      conditionsAccepted: false,
7    }
8 }
```

Aca vamos a indicarle unos valores iniciales a nuestro registro y ahora lo que vamos a hacer es importar nuestro archivo y tambien el uso de formik a lo que sería **RegisterForm.jsx**:

```
src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > ...
1  import React, { useState } from 'react'
2  import { Form } from "semantic-ui-react"
3  import { useFormik } from "formik"
4  import { initialValues } from "./RegisterForm.form"
5  import "./RegisterForm.scss"
6
```

Importamos el uso de formik y nuestros valores iniciales, para luego modificar más debajo de la siguiente manera:



```

7  export function RegisterForm() {
8    const [error, setError] = useState("")
9
10   const formik = useFormik({
11     initialValues: initialValues(),
12     onSubmit: async (formValue) => {
13       try {
14         console.log(formValue)
15       } catch (error) {
16         console.log(error)
17       }
18     }
19   })
20
21   return (

```

Cargamos a través de una constante el uso de **formik** y dentro le pasamos los valores iniciales para poder realizar nuestra función tipo flecha la cual va a realizar una función asincrónica con try catch y va a ejecutar el envío de la información, por el momento lo que realizaremos será un log en nuestra consola para poder indicarnos que estamos pasando a través de ella:

Ahora vamos a modificar nuestro formulario agregando al **onSubmit** el uso de **formik** de la siguiente manera:

```

21   return [
22     <Form className="register-form" onSubmit={formik.handleSubmit}>
23       <Form.Input name="email" placeholder="Correo Electronico" />
24       <Form.Input name="password" type="password" placeholder="Contraseña" />
25       <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" />
26       <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." />
27       <Form.Button type="submit" primary fluid >Crear Cuenta</Form.Button>
28
29       <p className="register-form__error">{error}</p>
30     </Form>
31   ]
32 }
33

```

Ahora para controlar el envío de nuestra data a través de los input se lo vamos a pasar a través del evento **onChange** y que utilice **formik.handleChange** y de igual manera le vamos a pasar el value con **formik.values** de esta manera estaremos procesando nuestros datos:

```

21   return [
22     <Form className="register-form" onSubmit={formik.handleSubmit}>
23       <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email}/>
24       <Form.Input name="password" type="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password}/>
25       <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" onChange={formik.handleChange} value={formik.values.repeatPassword}/>
26       <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." />
27       <Form.Button type="submit" primary fluid >Crear Cuenta</Form.Button>
28
29       <p className="register-form__error">{error}</p>
30     </Form>
31   ]
32 }
33

```

Ahora para nuestro check lo que vamos a realizar es muy sencillo, ya que no funciona de la misma manera que con los inputs, acá lo que vamos a realizar es la verificación de si esta check o no, para ello aplicamos de la siguiente manera:



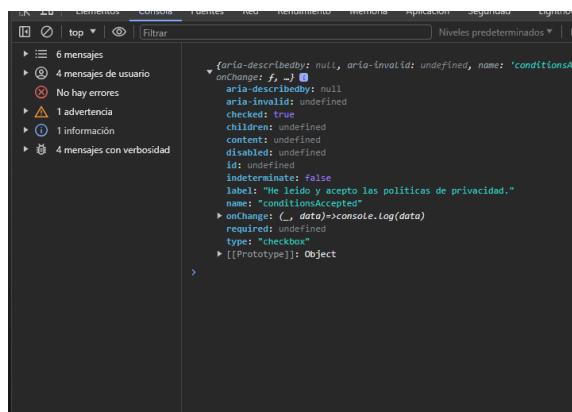
@hdtoledo

```

20
21     return [
22       <Form className="register-form" onSubmit={formik.handleSubmit}>
23         <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email}/>
24         <Form.Input name="password" type="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password}/>
25         <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" onChange={formik.handleChange} value={formik.values.repeatPassword}/>
26         <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." onChange={(_, data) => console.log(data)} />
27         <Form.Button type="submit" primary fluid >Crear Cuenta</Form.Button>
28
29       <p className="register-form__error">{error}</p>
30     </Form>
31   ]

```

Realizamos un log de la data que esta pasando del check, y como observamos al momento de pincharlo se nos va a mostrar la siguiente información en el navegador:



La propiedad que nos interesa es el **checked: true** de esta manera vamos a pasarle este dato, a formik de la siguiente forma:

```

23
24
25
26     <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email}/>
27     <Form.Input name="password" type="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password}/>
28     <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" onChange={formik.handleChange} value={formik.values.repeatPassword}/>
29     <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." onChange={(_, data) => formik.setFieldValue
("conditionsAccepted", data.checked)} checked={ formik.values.conditionsAccepted } />
<Form.Button type="submit" primary fluid>Crear Cuenta</Form.Button>
<p className="register-form__error">{error}</p>

```

Le indicamos que queremos pasarle **conditionsAccepted** a través del **check** y lo almacenamos en **checked** el valor a través de **formik**. Ahora para validar el botón al enviar la información lo vamos a dejar de la siguiente manera:

```

26
27     <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." onChange={(_, data) =>
("conditionsAccepted", data.checked)} checked={ formik.values.conditionsAccepted } />
<Form.Button type="submit" primary fluid loading={ formik.isSubmitting }>Crear Cuenta</Form.Button>
28
29     <p className="register-form__error">{error}</p>
30   </Form>

```

A través del **loading** hacemos el **envio** utilizando **formik** para que nos valide todo. Ahora vamos a verificar si nos esta funcionando, para ello nos ubicamos en **register** y vamos a darle al botón crear cuenta:



Dentro de información vamos a observar cómo envía la data que estamos pasando, hacemos el ejercicio indicando de nuevo con los inputs rellenados:

Ahora realizamos un pequeño ajuste de nuestro login en la parte de estilos, nos dirigimos a **/pages/admin/Auth/Auth.jsx** y vamos a importar **Grid** y **GridColumn** para dejar un poco mas estilizado nuestro responsive dentro de lo que estamos viendo:

```
src > pages > admin > Auth > Auth.jsx > ...
1  import React, { useState } from "react"
2  import { Tab, Grid, GridColumn } from "semantic-ui-react"
3  import { iconLogo } from "../../../../../assets"
4  import "./Auth.scss"
5  import { RegisterForm } from "../../../../../components/Admin/Auth/RegisterForm"
6
```

Y vamos a realizar un pequeño cambio en la estructura de Auth:



```

    29
    30     return (
    31       <Grid columns={1}>
    32         <GridColumn mobile={16} tablet={16} computer={16}>
    33           <div className="auth">
    34             <div>
    35               <img src={iconLogo} className="logo"/>
    36             </div>
    37             <Tab panes={panes} className="auth__forms" activeIndex={activeIndex} onTabChange={(_, data) => setActiveIndex(data.activeIndex)} />
    38           </div>
    39           </GridColumn>
    40       </Grid>
    41     )
    42   }
    43 }
    44

```

Llamamos nuestro **Grid** y ajustamos a 1 columna y dentro utilizamos **GridColumn** y le indicamos la cantidad de columnas para **mobile**, **Tablet** y **Computer**, esta dependencia nos permitirá hacer los ajustes de responsive de una manera simple, si lo que queremos full responsive recordemos que podemos adaptarlo a través de **media queries** de la siguiente manera en nuestro **Auth.scss**:

```

src > pages > admin > Auth >  Auth.scss > ...
...
39   }
40
41   @media (max-width: 280px) {
42     .auth {
43       width: 100%;
44     }
45
46     .auth__forms {
47       width: 250px;
48     }
49   }
50
51   @media (max-width: 320px) {
52     .auth {
53       width: 100%;
54     }
55
56     .auth__forms {
57       width: 300px;
58     }
59   }
60
61   @media (max-width: 390px) {
62     .auth {
63       width: 100%;
64     }
65
66     .auth__forms {
67       width: 250px;
68     }
69   }
70
71   @media (max-width: 500px) {
72     .auth {
73       width: 100%;
74     }
75
76     .auth__forms {
77       width: 250px;
78     }
79   }

```



@hdtoledo

Esta sería una manera básica de controlarlo para los diferentes dispositivos y tamaños de pantalla, recordemos que podemos ir a nuestro navegador y ajustarlo para verlo como se está comportando, notemos que agregamos nombres y apellidos también a nuestro formulario y debemos cargarlos en **initialValues** de igual manera:

```

<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body cz-shortcut-listen="true">
    <div id="root">
      <div class="ui one column grid"><flex>
        <div class="sixteen wide computer sixteen wide mobile sixteen wide tablet column">...</div>
        <div class="auth"><flex>--$0
          <div>...</div>
          <div class="auth__forms">--</div>
        </div>
      </div>
    </div>
  </body>
</html>

```

Elementos Consola Fuentes Red Rendimiento > ▲ 2 🔍

Estilos Calculados Diseño Procesadores de eventos Puntos de interrupción DOM Propiedades

Filtrar element.style { } @media (max-width: 500px) .auth { width: 100%; } .auth { background: linear-gradient(to bottom, #333, #000); background-size: cover; height: 100vh; background-position: center; display: flex; align-items: center; flex-direction: column; padding-top: 50px; } *, :after, :before { box-sizing: inherit; } div { display: block; } hoja de estilo de user

```

<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body cz-shortcut-listen="true">
    <div id="root">
      <div class="ui one column grid"><flex>
        <div class="sixteen wide computer sixteen wide mobile sixteen wide tablet column">...</div>
        <div class="auth"><flex>--$0
          <div>...</div>
          <div class="auth__forms">--</div>
        </div>
      </div>
    </div>
  </body>
</html>

```

Elementos Consola Fuentes Red Rendimiento > ▲ 1 🔍

Estilos Calculados Diseño Procesadores de eventos Puntos de interrupción DOM Propiedades

Filtrar element.style { } @media (max-width: 500px) .auth { width: 100%; } @media (max-width: 390px) .auth { width: 100%; } .auth { background: linear-gradient(to bottom, #333, #000); background-size: cover; height: 100vh; background-position: center; display: flex; align-items: center; flex-direction: column; padding-top: 50px; } *, :after, :before { box-sizing: inherit; } div { display: block; } hoja de estilo de user-adjunto Heredado de body



@hdtoledo

VALIDANDO EL FORMULARIO CON YUP

Ahora llego el momento de hacer las validaciones para nuestro registro, nos vamos a ubicar en **RegisterForm.form.jsx** y vamos a realizar lo siguiente:

```
src > components > Admin > Auth > RegisterForm >  RegisterForm.form.jsx > ...
1 import * as Yup from "yup"
2
3
```

Hacemos la importación de Yup, ahora vamos a crear un esquema de validacion para empezar a utilizar y validar los campos que tenemos del formulario y pasar una serie de mensajes en nuestro **register**, nos ubicamos más debajo de **initialValues**:

```
15 export function validationSchema(){
16   return Yup.object({
17     firstname: Yup.string().required("Campo Obligatorio"),
18     lastname: Yup.string().required("Campo Obligatorio"),
19     email: Yup.string().email("El mail no es valido").required("Campo Obligatorio"),
20     password: Yup.string().required("Campo Obligatorio"),
21     repeatPassword: Yup.string().required("Campo Obligatorio").oneOf([Yup.ref("password")], "Las contraseñas deben ser iguales"),
22     conditionsAccepted: Yup.bool().isTrue(true),
23   })
24 }
```

Acá estamos validando lo siguiente:

1. **export function validationSchema() {** - Esto define una función llamada **validationSchema** que se exporta para que pueda ser utilizada desde otros archivos.
2. **return Yup.object({** - Esta línea utiliza la biblioteca Yup para definir un esquema de validación. En este caso, se está creando un objeto Yup para validar múltiples campos.
3. **email: Yup.string().email("El mail no es válido").required("Campo Obligatorio"),** - Aquí se está configurando la validación para el campo "email". Se especifica que debe ser una cadena de texto (**Yup.string()**) y que debe ser una dirección de correo electrónico válida. Si no es una dirección de correo válida, se muestra un mensaje de error ("El mail no es válido"). Además, se marca como obligatorio, por lo que si está vacío, se muestra un mensaje de error ("Campo Obligatorio").
4. **password: Yup.string().required("Campo Obligatorio"),** - Esto configura la validación para el campo "password". Se requiere que sea una cadena de texto y también se marca como obligatorio. Si está vacío, mostrará un mensaje de error ("Campo Obligatorio").
5. **repeatPassword: Yup.string().required("Campo Obligatorio").oneOf([Yup.ref("password")], "Las contraseñas deben ser iguales"),** - Esta línea establece la validación para el campo "repeatPassword". Al igual que los campos anteriores, debe ser una cadena de texto y se requiere que no esté vacío. Además, se utiliza **.oneOf([Yup.ref("password")])**, "Las contraseñas deben ser iguales" para verificar que el valor sea igual al campo "password". Si no coinciden, se mostrará un mensaje de error ("Las contraseñas deben ser iguales").



@hdtoledo

- conditionsAccepted: Yup.bool().isTrue(true), - Aquí se configura la validación para el campo "conditionsAccepted", que se asume que es un valor booleano (verdadero o falso). Se utiliza .isTrue(true) para asegurarse de que sea verdadero. Si no es verdadero, se mostrará un mensaje de error.

Ahora nos dirigimos a nuestro **RegisterForm.jsx** en donde vamos a realizar la implementación, lo primero es importarlo:

```
src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > ...
1  import React, { useState } from 'react'
2  import { Form } from "semantic-ui-react"
3  import { useFormik } from "formik"
4  import { initialValues, validationSchema } from "./RegisterForm.form"
5  import "./RegisterForm.scss"
6
```

Ahora implementamos dentro de Formik de la siguiente manera:

```
6
7  export function RegisterForm() {
8      const [error, setError] = useState("")
9
10     const formik = useFormik({
11
12         initialValues: initialValues(),
13         validationSchema: validationSchema(),
14         validateOnChange: false,
15
16         onSubmit: async (formValue) => {
17             try {
18                 console.log(formValue)
19             } catch (error) {
20
21             setError(error.message)
22         }
23     })
24
25     return (
26         <Form className="register-form" onSubmit={formik.handleSubmit}>
27             <Form.Input name="firstname" type="text" placeholder="Nombres" onChange={formik.handleChange} value={formik.values.firstname} error={formik.errors.firstname}/>
28             <Form.Input name="lastname" type="text" placeholder="Apellidos" onChange={formik.handleChange} value={formik.values.lastname} error={formik.errors.lastname}/>
29             <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email} error={formik.errors.email}/>
30             <Form.Input name="password" type="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password} error={formik.errors.password}/>
31             <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" onChange={formik.handleChange} value={formik.values.repeatPassword} error={formik.errors.repeatPassword}/>
32             <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." onChange={(_, data) => formik.setFieldValue("conditionsAccepted", data.checked)} checked={formik.values.conditionsAccepted} error={formik.errors.conditionsAccepted}/>
33             <Form.Button type="submit" primary fluid loading={formik.isSubmitting}>Crear Cuenta</Form.Button>
34         </Form>
35     )
36 }
```

Ahora empezamos a implementar en cada uno de nuestros inputs:

```
32     return (
33         <Form className="register-form" onSubmit={formik.handleSubmit}>
34             <Form.Input name="firstname" type="text" placeholder="Nombres" onChange={formik.handleChange} value={formik.values.firstname} error={formik.errors.firstname}/>
35             <Form.Input name="lastname" type="text" placeholder="Apellidos" onChange={formik.handleChange} value={formik.values.lastname} error={formik.errors.lastname}/>
36             <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email} error={formik.errors.email}/>
37             <Form.Input name="password" type="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password} error={formik.errors.password}/>
38             <Form.Input name="repeatPassword" type="password" placeholder="Repetir Contraseña" onChange={formik.handleChange} value={formik.values.repeatPassword} error={formik.errors.repeatPassword}/>
39             <Form.Checkbox name="conditionsAccepted" label="He leido y acepto las politicas de privacidad." onChange={(_, data) => formik.setFieldValue("conditionsAccepted", data.checked)} checked={formik.values.conditionsAccepted} error={formik.errors.conditionsAccepted}/>
40             <Form.Button type="submit" primary fluid loading={formik.isSubmitting}>Crear Cuenta</Form.Button>
41         </Form>
42     )
43 
```

De esta manera hacemos las validaciones con error mediante Yup y agilizamos el proceso para poder avanzar de una manera más práctica, ahora revisemos en nuestro navegador cada uno de los campos y opciones:



localhost:5173/admin



Nuevo Usuario

Entrar	Nuevo Usuario
Nombres	<input type="text"/>
Apellidos	<input type="text"/>
Correo Electronico	<input type="text"/>
Contraseña	<input type="password"/>
Repetir Contraseña	<input type="password"/>
<input type="checkbox"/> He leido y acepto las políticas de privacidad.	
<input type="button" value="Crear Cuenta"/>	

localhost:5173/admin

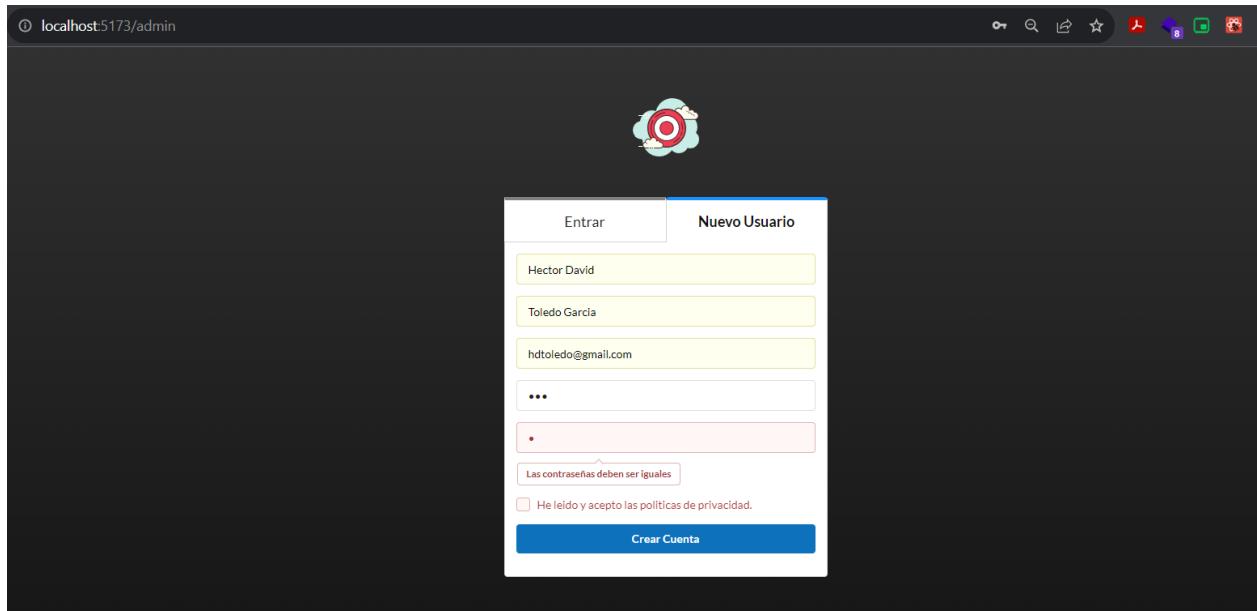


Nuevo Usuario

Entrar	Nuevo Usuario
Hector David	<input type="text"/>
Toledo Garcia	<input type="text"/>
hdtoledo@gmail.com	<input type="text"/>
Contraseña	<input type="password"/>
Repetir Contraseña	<input type="password"/>
<input type="checkbox"/> He leido y acepto las políticas de privacidad.	
<input type="button" value="Crear Cuenta"/>	



@hdtoledo



localhost:5173/admin

Nuevo Usuario

Hector David

Toledo Garcia

hdtoledo@gmail.com

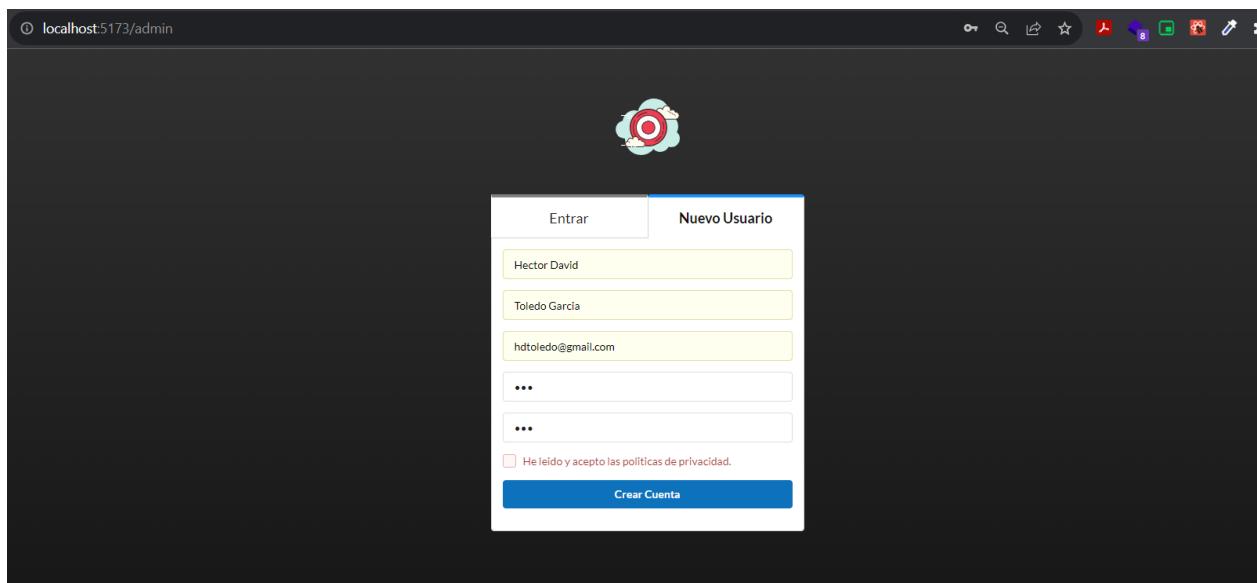
•••

•••

Las contraseñas deben ser iguales

He leido y acepto las politicas de privacidad.

Crear Cuenta



localhost:5173/admin

Nuevo Usuario

Hector David

Toledo Garcia

hdtoledo@gmail.com

•••

•••

He leido y acepto las politicas de privacidad.

Crear Cuenta

De esta manera comprobaremos como en cada situación de nuestros inputs nos permite hacer las validaciones incluyendo el check.

Ahora realizamos un pequeño cambio en nuestro try catch de **RegisterForm.jsx**:



```

src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > RegisterForm > formik > onSubmit

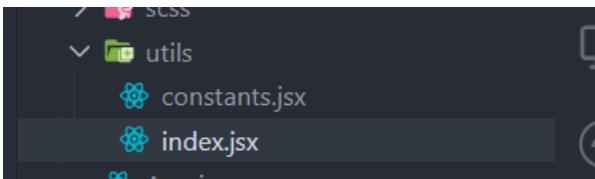
11     initialValues: initialValues(),
12     validationSchema: validationSchema(),
13     validateOnChange:false,
14
15     onSubmit: async (formValue) => {
16       try {
17         setError("")
18         console.log(formValue)
19       } catch (error) {
20         setError("Error en el servidor")
21       }
22     }
23   }
24 }
25
26 return (

```

Establecemos nuestro **setError** para que se ejecute al momento de generarnos errores.

REGISTRANDO EL USUARIO

Vamos a realizar nuestra conexión a nuestro backend y empezar a hacer el registro de nuestros usuarios, para ello nos vamos a crear la carpeta **/src/utils/** y vamos a crear dos archivos, **index.jsx** y **constants.jsx**:



Y dentro de nuestro **constants.jsx**:

```

src > utils > constants.jsx > ENV
1 const SERVER_IP = "localhost:3000"
2
3 export const ENV = {
4   BASE_PATH: `http://${SERVER_IP}`,
5   BASE_API: `http://${SERVER_IP}/api/v1`,
6   API_ROUTES: {
7     REGISTER: "auth/register",
8   }
9 }
10

```

Acá estamos dejando nuestras constantes para hacer la conexión a nuestro API REST, y ahora en nuestro **index.jsx**:

```

src > utils > index.jsx
1 export * from "./constants"

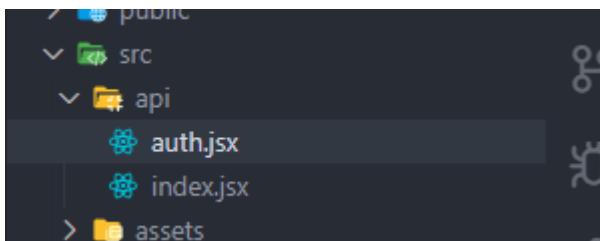
```

Hacemos la exportación de nuestras constantes. Ahora vamos a crear un lugar en donde se van a llevar a cabo todas las conexiones **http** y vamos a dejar esta ubicación para que sea mas escalable y en caso tal



@hdtoledo

de que queramos realizar algún cambio sea mucho fácil acceder, vamos a crear la carpeta dentro de `/src/api` y creamos `index.jsx` y `auth.jsx`:



Y dentro de `index.jsx` vamos a dejar nuestra exportación de Auth:

```
src > api > index.jsx
1   export * from "./auth"
```

Y en nuestro `Auth.jsx` vamos a dejarlo de la siguiente manera:

```
src > api > auth.jsx > Auth
1   import { ENV } from "../utils"
2
3   export class Auth {
4     baseApi = ENV.BASE_API;
5
6     async register(data) {
7       try {
8         const url = `${this.baseApi}/${ENV.API_ROUTES.REGISTER}`
9
10        const params = {
11          method: "POST",
12          headers: {
13            "Content-Type": "application/json"
14          },
15          body: JSON.stringify({
16            firstname: data.firstname,
17            lastname: data.lastname,
18            email: data.email,
19            password: data.password,
20          })
21        }
22
23        const response = await fetch(url, params)
24        const result = await response.json()
25        if (response.status !== 200) throw result
26        return result
27
28      } catch (error) {
29        throw error
30      }
31    }
32  }
```

1. **import { ENV } from "../utils"**: Importa el objeto **ENV** desde un módulo llamado "utils". Este objeto probablemente contiene configuraciones de entorno y rutas para la aplicación.
2. **export class Auth {** - Define una clase llamada **Auth** que se exporta para que pueda ser utilizada en otros lugares.



@hdtoledo

3. **baseApi = ENV.BASE_API;** - Esto declara una propiedad llamada **baseApi** en la clase **Auth** y le asigna el valor de **ENV.BASE_API**. Es probable que **ENV.BASE_API** sea una URL base para las solicitudes de la API de la aplicación.
4. **async register(data) {** - Define un método asincrónico llamado **register** que toma un parámetro **data**. Este método se utiliza para registrar un usuario en la aplicación.
5. **try {** - Inicia un bloque **try**, que se utiliza para manejar cualquier error que pueda ocurrir dentro de él.
6. **const url = \${this.baseApi}/\${ENV.API_ROUTES.REGISTER}``** - Crea una URL de registro combinando la **baseApi** con una ruta específica de registro obtenida de **ENV.API_ROUTES**.
7. **const params = { ... }** - Crea un objeto **params** que contiene configuraciones para una solicitud HTTP, incluyendo el método "POST", encabezados y los datos de registro (correo electrónico y contraseña) en formato JSON.
8. **const response = await fetch(url, params)** - Realiza una solicitud a la URL de registro utilizando la función **fetch()** y las configuraciones definidas en **params**. Esta línea espera a que la solicitud se complete y almacena la respuesta en la variable **response**.
9. **const result = await response.json()** - Extrae el contenido de la respuesta como JSON y lo almacena en la variable **result**. Esto se hace para procesar la respuesta de la API.
10. **if (response !== 200) throw result** - Comprueba si el código de estado de la respuesta no es igual a 200 (que generalmente indica una creación exitosa). Si no es igual, lanza un error con el contenido de la respuesta.
11. **return result** - Si todo va bien y la respuesta es 200, se devuelve el resultado exitoso.
12. **} catch (error) {** - Inicia un bloque **catch** que se ejecuta si se produce un error dentro del bloque **try**.
13. **throw error** - Lanza el error capturado para que pueda ser manejado en otro lugar de la aplicación.

En resumen, enviamos una solicitud de registro a una API utilizando la URL y los datos proporcionados, y luego maneja la respuesta, lanzando un error si la respuesta no es satisfactoria o devolviendo el resultado exitoso si la creación se realiza con éxito.

Ahora nos vamos a **RegisterForm.jsx** y hacemos la importación:

```
src > components > Admin > Auth > RegisterForm > RegisterForm.jsx > ...
1  import React, { useState } from 'react'
2  import { Form } from "semantic-ui-react"
3  import { useFormik } from "formik"
4  import { Auth } from "../../../../../api"
5  import { initialValues, validationSchema } from "./RegisterForm.form"
6  import "./RegisterForm.scss"
7
```

Y ahora la implementamos de la siguiente manera:



@hdtoledo

```

5 import { initialValues, validationSchema } from "./RegisterForm.form"
6 import "./RegisterForm.scss"
7
8 const authController = new Auth()
9
10 export function RegisterForm() {
11     const [error, setError] = useState("")

```

Y ahora lo utilizamos dentro de nuestro try catch de la siguiente manera:

```

17         validateOnchange: false,
18
19         onSubmit: async (formValue) => {
20             try {
21                 setError("")
22                 console.log(formValue)
23                 await authController.register(formValue)
24             }
25             catch (error) {
26                 setError("Error en el servidor")
27             }
28         }

```

Y si todo va bien le vamos a indicar que nos lleve al login, recordemos que tenemos nuestra función de **openLogin**, para ello vamos a recuperar los props en nuestro **RegisterForm** y hacemos una desestructuración de openlogin:

```

9
10 export function RegisterForm(props) {
11
12     const { openLogin } = props
13     const [error, setError] = useState("")
14

```

Y ahora implementamos nuestro **openlogin** debajo de nuestro **authcontroller**:

```

21         onSubmit: async (formValue) => {
22             try {
23                 setError("")
24                 await authController.register(formValue)
25                 openLogin()
26             } catch (error) {
27                 setError("Error en el servidor")
28             }
29         }
30     }
31 }

```

Ahora llego la hora de comprobar si esta funcionando nuestro registro, para ello vamos a nuestro navegador y vamos a hacer un registro:



Al momento de enviar nos debe validar todo y enviarnos al login:

Para verificar que el usuario ha sido creado podemos ir a nuestro insomnia y verificar en getUsers:

```

23 "id": "6527005960b8ad666ca60ff",
24 "firstname": "Probando 2",
25 "lastname": "Iastname03",
26 "email": "test04@mail.com",
27 "password": "$2a$10$Komi9WtWU9Xjnts2m20.20Zprr86rvjPJ2M0Vkg8gSWIxJTTvaa",
28 ",
29 "role": "admin",
30 "active": false,
31 "avatar": "avatar/p8Tj0pyG8702-ULfyJndZRM.png",
32 "__v": 0
33 },
34 {
35 "id": "6541adc912da19349f802748",
36 "firstname": "Jimmy",
37 "lastname": "Lombana",
38 "email": "jimmy.lombana@gmail.com",
39 "role": "user",
40 "active": false,
41 "password": "$2a$10$J1tz8gFONv6WKz1ZGwTv.URJlq0wogog1SUHeP8IwdG.LMNUNiro2",
42 "__v": 0
43 ]

```



De igual manera podemos hacerlo en mongodb Atlas:

The screenshot shows the MongoDB Atlas interface. In the left sidebar, under the 'test' database, there are three collections: 'courses', 'menus', and 'users'. The 'users' collection is currently selected. On the right, the 'Find' tab is active, and a query is being typed into the search bar: '_id: ObjectId('65270059605b8ad666ca60ff')'. Below the search bar, two user documents are displayed in a table format. The first document has the following fields and values:

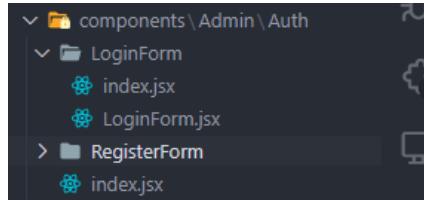
_id	ObjectId('65270059605b8ad666ca60ff')
firstname	'Probando 2'
lastname	'lastname03'
email	"test04@gmail.com"
password	"\$2a\$10\$Koni9YwTWU9Xjnts24m20.z0Zprr86rvjPJ2M9Vkg8gSW1xJTTvaa"
role	"admin"
active	false
avatar	"avatar/p8Tj0pyG8702-ULfyJcndZRM.png"
_v	0

The second document has similar fields but different values, such as 'firstname' being 'Jimmy' and 'lastname' being 'Lombana'. Both documents have an '_v' field set to 0.

Y de esta manera ya tenemos funcionando nuestro registro de usuario en el **frontend** y consumiendo el API REST.

FORMULARIO PARA LOGIN

Ahora vamos a crear nuestro formulario para login, vamos a crear la misma estructura que realizamos con registro dentro de **Auth** creamos **/LoginForm/** y dentro hacemos nuestros dos archivos **index.jsx** y **LoginForm.jsx**:



Dentro de nuestro **index** hacemos la exportación de siempre:

```
src > components > Admin > Auth > LoginForm > index.jsx
  1  export * from "./LoginForm"
```

Y en nuestro **LoginForm** vamos a crear un componente base:

```
src > components > Admin > Auth > LoginForm > LoginForm.jsx > ...
  1  import React from 'react'
  2
  3  export function LoginForm() {
  4    return (
  5      <div>
  6        <h1>Login Form</h1>
  7      </div>
  8    )
  9  }
 10
```

Recordemos hacer la exportación dentro de nuestro **index** de carpetas:

```
src > components > Admin > Auth > index.jsx
1  export * from "./RegisterForm"
2  export * from "./LoginForm"
```

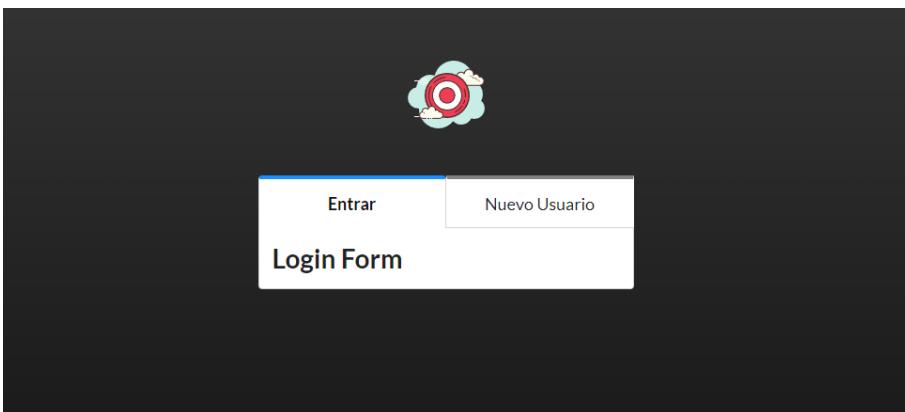
Y ahora nos vamos a nuestro **/Pages/Admin/Auth/Auth.jsx** y vamos a hacer la importación de **LoginForm** y **RegisterForm** en la misma línea:

```
src > pages > admin > Auth > Auth.jsx > ...
1  import React, { useState } from "react"
2  import { Tab, Grid,GridColumn } from "semantic-ui-react"
3  import { iconLogo } from "../../assets"
4  import "./Auth.scss"
5  import { LoginForm, RegisterForm } from "../../components/Admin/Auth"
6
7  export function Auth() {
```

Y ahora hacemos la implementación de nuestro **LoginForm**:

```
11  const panes = [
12    {
13      menuItem: "Entrar",
14      render: () => (
15        <Tab.Pane>
16          | <LoginForm />
17        </Tab.Pane>
18      )
19    },
20    {
21      menuItem: "Nuevo Usuario"
```

En nuestro navegador ya podríamos verlo:

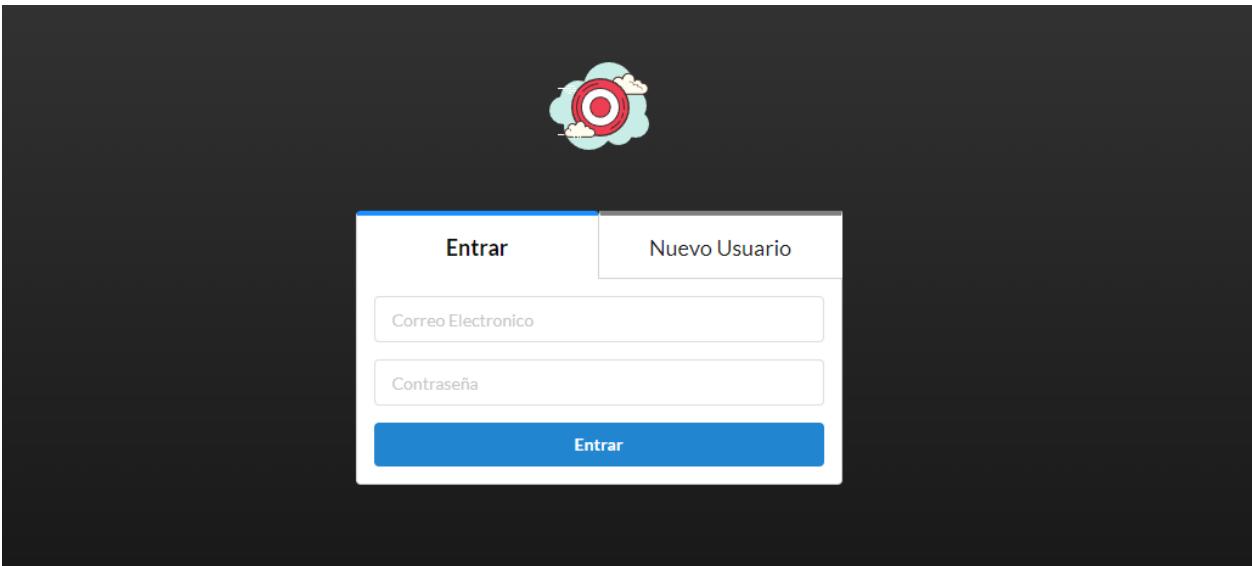


@hdtoledo

Ahora vamos a hacer las siguientes importaciones y dejar la estructura inicial de nuestro formulario:

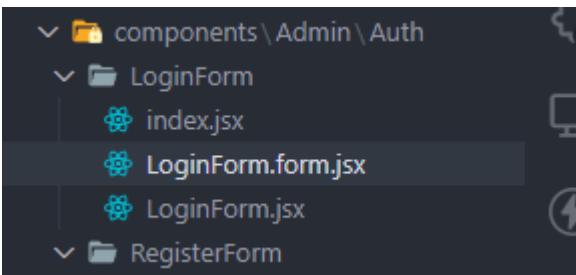
```
src > components > Admin > Auth > LoginForm >  LoginForm.jsx > ...
1  import React from 'react'
2  import { Form } from "semantic-ui-react"
3
4  export function LoginForm() {
5    return (
6      <Form>
7        <Form.Input name="email" placeholder="Correo Electronico" />
8        <Form.Input name="password" placeholder="Contraseña" />
9        <Form.Button type="submit" primary fluid>Entrar</Form.Button>
10     </Form>
11   )
12 }
13 |
```

Y si vamos a nuestro navegador observamos:



VALIDANDO EL FORMULARIO

Ahora vamos a realizar las validaciones, así como las hicimos con nuestro **RegisterForm**, lo primero es crear el archivo **LoginForm.form.jsx** en donde vamos a pasar nuestras funciones:



Y dentro vamos a dejar lo siguiente:

```
src > components > Admin > Auth > LoginForm > LoginForm.form.jsx > validationSchema
1 import * as Yup from "yup"
2
3 export function initialValues(){
4     return {
5         email: "",
6         password: "",
7     }
8 }
9
10 export function validationSchema() {
11     return Yup.object({
12         email: Yup.string().email("El email no es valido").required("Campo obligatorio"),
13         password: Yup.string().required("Campo obligatorio"),
14     })
15 }
```

Ahora vamos a realizar las importaciones en nuestro **LoginForm.jsx**:

```
src > components > Admin > Auth > LoginForm > LoginForm.jsx > ...
1 import React from 'react'
2 import { Form } from "semantic-ui-react"
3 import { useFormik } from "formik"
4 import { initialValues, validationSchema } from "./LoginForm.form"
5
6 export function LoginForm() {
```

Y vamos a realizar nuestras implementaciones de la siguiente manera:

```
5
6 export function LoginForm() {
7
8     const formik = useFormik({
9         initialValues: initialValues(),
10        validationSchema: validationSchema(),
11        validateOnChange: false,
12        onSubmit: async (formValue) => {
13            try {
14                console.log(formValue)
15            } catch (error) {
16                console.log(error)
17            }
18        }
19    )
20
21    return (
22        <Form>
23            <Form.Input name="email" placeholder="Correo Electronico" />
```

Hacemos la carga inicial de nuestros valores, aplicamos nuestro esquema y por el momento vamos a pasar en consola los valores. Ahora lo implementamos en nuestro formulario de la siguiente manera:



@hdtoledo

```

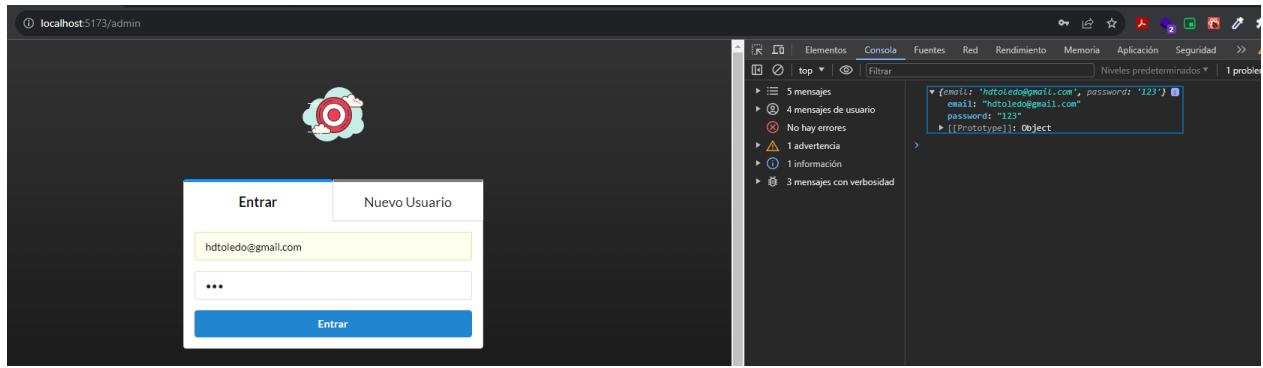
21   return (
22     <Form onSubmit={formik.handleSubmit}>
23       <Form.Input name="email" placeholder="Correo Electronico" onChange={formik.handleChange} value={formik.values.email} error={formik.errors.email} />
24       <Form.Input type="password" name="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password} error={formik.errors.password} />
25       <Form.Button type="submit" primary fluid loading={formik.isSubmitting}>Entrar</Form.Button>
26     </Form>
27   )
28 }
29

```

Si notamos es muy similar a lo que hicimos con nuestro register, de esta manera vamos a comprobar si nos funciona en el navegador:



@hdtoledo



De esta manera ya tenemos nuestros datos del formulario validados y correctamente funcionando.

REALIZANDO EL LOGIN

Ahora lo que vamos a hacer es crear la parte del login dentro de `/api/auth.js`:

```
src > api > auth.js > Auth
30     }
31   }
32
33   async login() {
34     try {
35
36     } catch (error) {
37
38     }
39   }
40 }
```

Creamos dentro de nuestra función de Auth en la parte de abajo nuestro login y allí vamos a dejar esta estructura que la vamos a empezar a modificar, lo primero es cargar la constante de nuestra url y el control de errores:

```
32
33   async login() {
34     try {
35       const url = `${this.baseApi}/`
36     } catch (error) {
37       throw error
38     }
39   }
40 }
```

Ahora nos vamos a nuestro archivo de `constants.jsx` y vamos a crear la ruta de login:



```

src > utils > constants.jsx > ENV > API_ROUTES
1 const SERVER_IP = "localhost:3000"
2
3 export const ENV = {
4     BASE_PATH: `http://${SERVER_IP}`,
5     BASE_API: `http://${SERVER_IP}/api/v1`,
6     API_ROUTES: [
7         REGISTER: "auth/register",
8         LOGIN: "auth/login",
9     ],
10    }
11}

```

Nuevamente volvemos a nuestro **/api/Auth.jsx** y vamos a terminar nuestra ruta y login:

```

src > api > auth.jsx > Auth
1
2     async login(data) {
3         try {
4             const url = `${this.baseApi}/${ENV.API_ROUTES.LOGIN}`;
5             const params = {
6                 method: "POST",
7                 headers: {
8                     "Content-Type": "application/json"
9                 },
10                body: JSON.stringify(data)
11            }
12
13
14            const response = await fetch(url, params)
15            const result = await response.json()
16
17            if (response.status !== 200) throw result
18            return result
19
20        } catch (error) {
21            throw error
22        }
23    }
24}

```

Si analizamos el código es muy similar al que estamos utilizando con **register**, lo que nos queda es hacer la importación de nuestro Auth e implementarlo en **LoginForm.jsx**:

```

src > components > Admin > Auth > LoginForm > LoginForm.jsx > authController
1 import React from 'react'
2 import { Form } from "semantic-ui-react"
3 import { useFormik } from "formik"
4 import { Auth } from "../../../../../api"
5 import { initialValues, validationSchema } from "./LoginForm.form"
6
7 const authController = new Auth()
8
9 export function LoginForm() {
10}

```

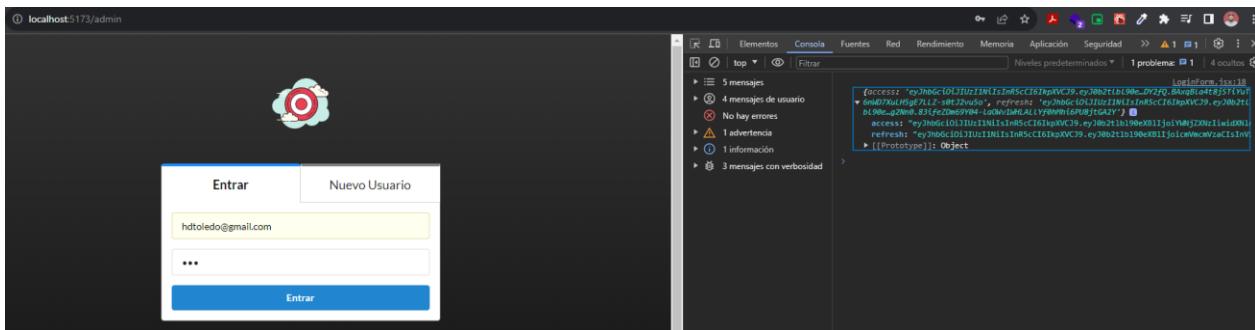
Ahora en nuestro **try Catch** vamos realizar la implementación y dejamos un logo para response:



@hdtoledo

```
13 validationSchema: validationSchema(),
14 validateOnChange: false,
15 onSubmit: async (formValue) => {
16     try {
17         const response = await authController.login(formValue)
18         console.log(response)
19
20     } catch (error) {
21         console.log(error)
22     }
23 }
```

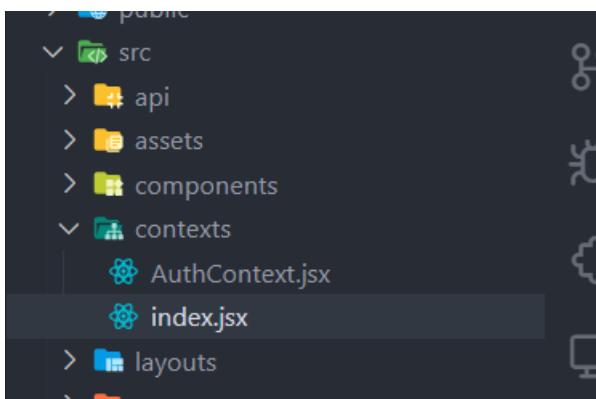
De esta manera ya nos queda ir a nuestro navegador y encontrarnos con lo siguiente:



Nuestros tokens de Access y refresh en nuestro response, quiere decir que todo va correcto y estamos haciendo login, pero aun no nos movemos al dashboard principal, ahora lo que tenemos que controlar son estos tokens y validar nuestra sesión de usuario.

CONTEXTO Y HOOK PARA LA SESIÓN

Vamos a crear en **/srccontexts/** dos archivos **index.jsx** y **AuthContext.jsx**:



En nuestro archivo index.jsx hacemos nuestra exportación:

```
src > contexts > ✨ index.jsx
```



Ahora vamos a crear nuestro AuthContext y aquí lo que vamos a realizar será que nos permitirá mover un estado a nivel global de la aplicación y que cualquier componente hijo lo pueda ver utilizando un hook.

```
src > contexts > AuthContext.jsx > ...
1 import {useState, useEffect, createContext} from 'react'
2
3 export const AuthContext = createContext()
4
5 export function AuthProvider(props) {
6
7   const { children } = props
8   const [ user, setUser ] = useState(null)
9   const [ token, setToken ] = useState(null)
10
11  useEffect(() => {
12    //Comprobar si el usuario esta logueado o no
13  })
14
15  const login = async (accessToken) => {
16    console.log("Login Context")
17    console.log(accessToken)
18  }
19
20  const data = {
21    accessToken: token,
22    user,
23    login,
24  }
25
26  return <AuthContext.Provider value={data}>{children}</AuthContext.Provider>
27}
28|
```

1. Importamos los hooks **useState** y **useEffect** y la función **createContext** de React para crear un contexto.
2. Creamos el contexto de autenticación con **createContext** y lo exportamos como **AuthContext**. Este contexto se utilizará para proporcionar datos de autenticación a otros componentes de la aplicación.
3. Definimos un componente funcional **AuthProvider** que acepta **props** como argumento.
4. Destructuramos **children** de las **props**. Los **children** son los componentes anidados dentro de **<AuthProvider>** en la jerarquía de componentes.
5. Usamos **useState** para inicializar dos estados: **user** y **token**. Estos estados almacenarán información sobre el usuario y el token de acceso.
6. Se utiliza **useEffect** para realizar alguna acción cuando el componente **AuthProvider** esto significa que, cuando el componente se monta o se actualiza, se realizará la comprobación de autenticación para determinar si el usuario está autenticado.
7. Se define una función **login** que se utilizará para manejar el proceso de inicio de sesión. En este caso, se imprime un mensaje en la consola con el token de acceso proporcionado. Esto debe reemplazarse con la lógica real de inicio de sesión.



@hdtoledo

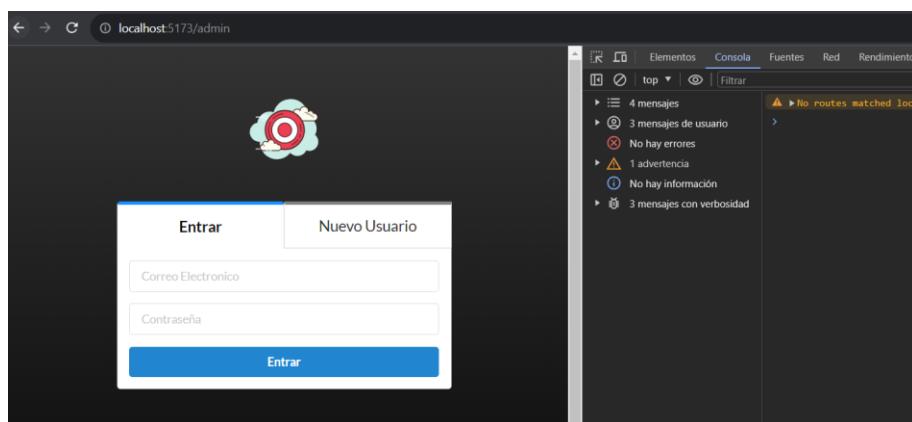
8. Se crea un objeto **data** que contiene **accessToken**, **user**, y **login**. Este objeto representa los datos de autenticación que se proporcionarán a través del contexto.
9. Se utiliza el componente **<AuthContext.Provider>** para proporcionar el contexto y los datos alrededor de los componentes hijos (**children**). Los datos de autenticación, como **accessToken**, **user**, y **login**, estarán disponibles para los componentes que estén dentro del árbol de componentes envuelto por **<AuthProvider>**.

En resumen, establecemos un contexto de autenticación y un proveedor que se utilizarán para gestionar la información de autenticación y proporcionarla a los componentes que la necesiten en la aplicación. La lógica de autenticación real, como la verificación del usuario, debería implementarse en el espacio reservado que se indica en el **useEffect**.

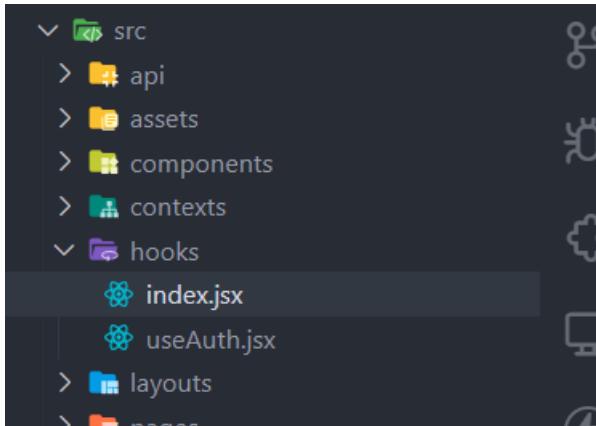
Ahora vamos a implementarlo sobre nuestro **App.jsx** lo primero es la importación y lo segundo es dejarlo para que envuelva toda nuestra aplicación:

```
src > App.jsx > App
1  import React from "react";
2  import { BrowserRouter } from "react-router-dom";
3  import { WebRouter, AdminRouter } from "./router";
4  import { AuthProvider } from "./contexts";
5
6  export default function App() {
7    return (
8      <AuthProvider>
9        <BrowserRouter>
10          <WebRouter />
11          <AdminRouter />
12        </BrowserRouter>
13      </AuthProvider>
14    );
15  }
16
```

Validamos en nuestro navegador que no se haya crasheado:



Ahora para poder acceder a este contexto vamos a crear nuestra carpeta **/src/hooks/** y dentro vamos a crear nuestros archivos **index.jsx** y **useAuth.jsx**



Dentro de nuestro **index.jsx** vamos a dejar nuestra exportación:

```
src > hooks > index.jsx
1   export * from "./useAuth"
```

Y dentro de nuestro **useAuth.jsx** vamos a dejar lo siguiente:

```
src > hooks > useAuth.jsx > useAuth
1   import { useContext } from "react"
2   import { AuthContext } from "../contexts"
3
4   export const useAuth = () => useContext(AuthContext)
```

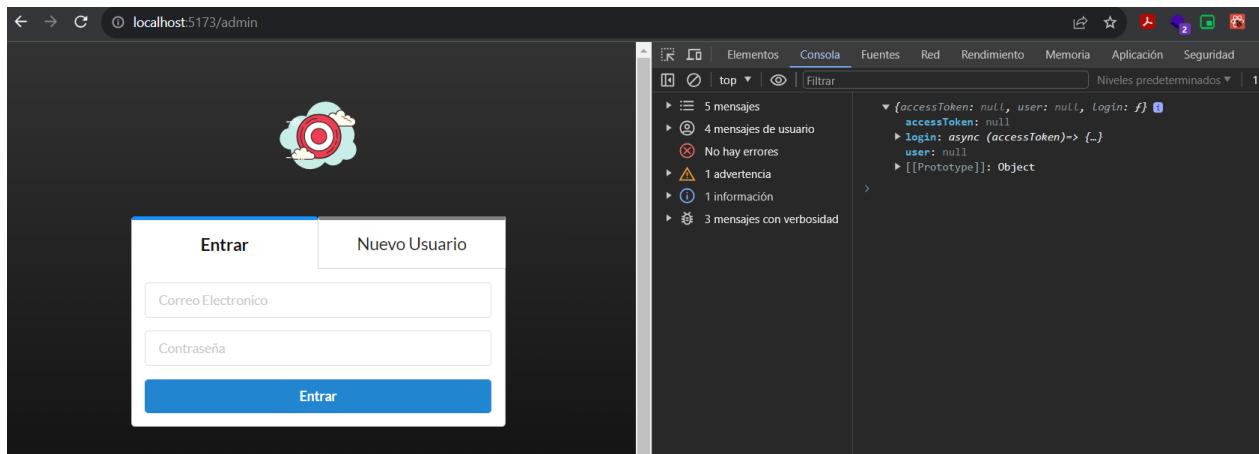
Ahora para empezar a utilizarlo nos dirigimos a **/src/router/AdminRouter.jsx** y vamos a importarlo:

```
src > router > AdminRouter.jsx > ...
1   import React from 'react'
2   import { Routes, Route } from "react-router-dom"
3   import { AdminLayout } from "../layouts"
4   import { Auth, Users, Blog, Courses, Menu, Newsletter } from "../pages/admin"
5   import { useAuth } from "../hooks"
6
7   const user = null
```

Para verificar como funciona lo que vamos a realizar primero es un log de **useAuth**:

```
8
9   const user = null
10
11  export function AdminRouter() {
12
13    const loadLayout = (Layout, Page) => {
14      return (
15        <Layout>
16          <Page />
17        </Layout>
18      )
19    }
20
21    return (
22      <Routes>
23        <Route path="/" element={loadLayout(Home)} />
24        <Route path="/admin" element={loadLayout(Admin)} />
25        <Route path="/users" element={loadLayout(Users)} />
26        <Route path="/blog" element={loadLayout(Blog)} />
27        <Route path="/courses" element={loadLayout(Courses)} />
28        <Route path="/menu" element={loadLayout(Menu)} />
29        <Route path="/newsletter" element={loadLayout(Newsletter)} />
30      </Routes>
31    )
32  
```

Y si vamos a nuestro navegador vamos a observar lo siguiente:



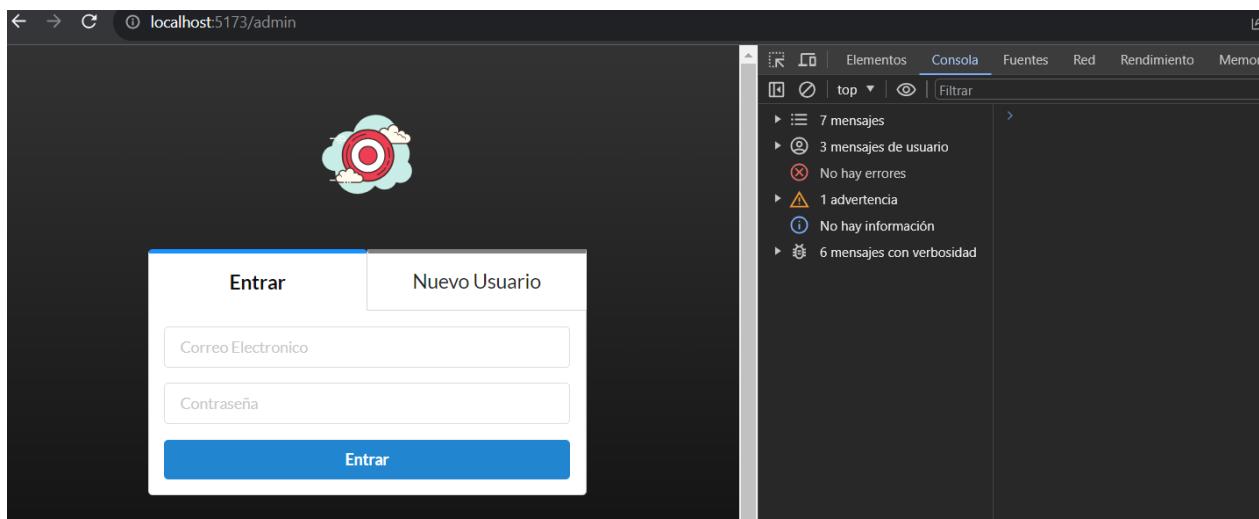
The screenshot shows a browser window with the URL `localhost:5173/admin`. On the left, there is a login form with fields for 'Correo Electronico' and 'Contraseña', and a blue 'Entrar' button. On the right, the developer tools' 'Consola' tab is open, displaying the following message:

```
▼ {accessToken: null, user: null, login: f} ⓘ
  accessToken: null
  login: async (accessToken) => {..}
  user: null
  [[Prototype]]: Object
```

Tenemos un objeto con el **Access token**, **login** y **user**, esto quiere decir que está funcionando correctamente, ahora vamos a modificar un poco:

```
4 import { Auth, Users, Blog, Courses, Menu, Newsletter } from "next-auth"
5 import { useAuth } from "../hooks"
6
7
8 export function AdminRouter() {
9   const { user } = useAuth
10
11   const loadLayout = (Layout, Page) => {
12     return (
13       <Layout>
```

Hacemos desestructuración de **useAuth** y traemos solamente **user**, de esta manera ya podemos eliminar la constante que tenía **user** en uso anteriormente, y ahora si revisamos la aplicación debe seguir funcionando correctamente:



The screenshot shows the same browser setup as before, but the developer tools' 'Consola' tab now displays a different message:

```
▼ {accessToken: null, user: null, login: f} ⓘ
  accessToken: null
  login: async (accessToken) => {..}
  user: null
  [[Prototype]]: Object
```

This message is identical to the one in the previous screenshot, indicating that the application is still functioning correctly despite the code modification.

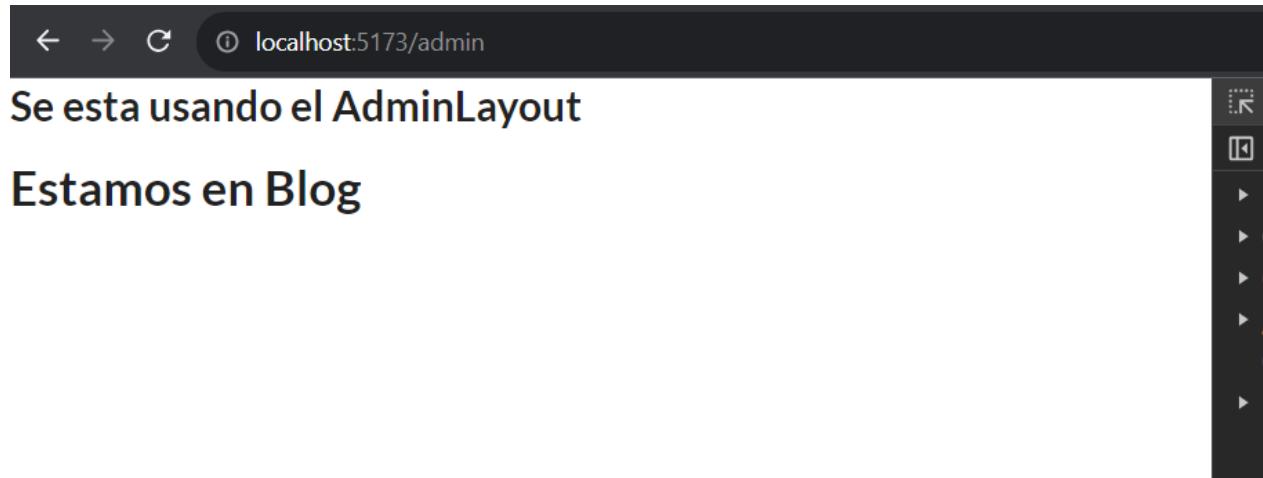


@hdtoledo

Vamos a probar ahora un poco con el estado de nuestro usuario y vamos a pasar unos datos, para ello nos dirigimos a `/context/AuthContext.jsx` y vamos a realizar un cambio en nuestro `useState` de `user`:

```
src > contexts > AuthContext.jsx > AuthProvider
  4
  5  export function AuthProvider(props) {
  6
  7    const { children } = props
  8    const [ user, setUser ] = useState({username:"hdtoledo"})
  9    const [ token, setToken ] = useState(null)
10
11    useEffect(() => {
```

Y vamos a nuestro navegador:



Nos muestra directamente nuestro `AdminLayout` ya que estamos validando correctamente. Ahora volvemos a dejarlo como estaba en null ya que esto se debe autocompletar solo para poder realizar las validaciones.

GENERANDO LA SESION

Ahora vamos a generar la sesión para poder realizar el manejo del token y poder ingresar, para ello nos vamos a nuestro `AuthContext.jsx` y vamos a realizar lo siguiente en nuestra const login:

```
src > contexts > AuthContext.jsx > AuthProvider
  14
  15  const login = async (accessToken) => {
  16    try {
  17      console.log("Token: ", accessToken)
  18    } catch (error) {
  19      console.log(error)
  20    }
  21  }
  22
  23  const data = {
```

Dejamos un try catch en donde vamos a empezar a organizar nuestra lógica, por el momento verificamos con el log lo que nos va a llegar, ahora vamos a nuestro **LoginForm.jsx** y vamos a hacer la importación de **useAuth**:

```
src > components > Admin > Auth > LoginForm >  LoginForm.jsx > ...
1  import React from 'react'
2  import { Form } from "semantic-ui-react"
3  import { useFormik } from "formik"
4  import { Auth } from "../../../../../api"
5  import { useAuth } from "../../../../../hooks"
6  import { initialValues, validationSchema } from "./LoginForm.form"
7
8  const authController = new Auth()
```

Y en nuestro componente lo implementamos así:

```
10 export function LoginForm() {
11
12   const { login } = useAuth()
13
14   const formik = useFormik({
15     initialValues: initialValues(),
```

Y mas abajito vamos a realizar la implementación de **login()**, pero primero vamos a dejarlo comentado para verificar que nos envía **response**:

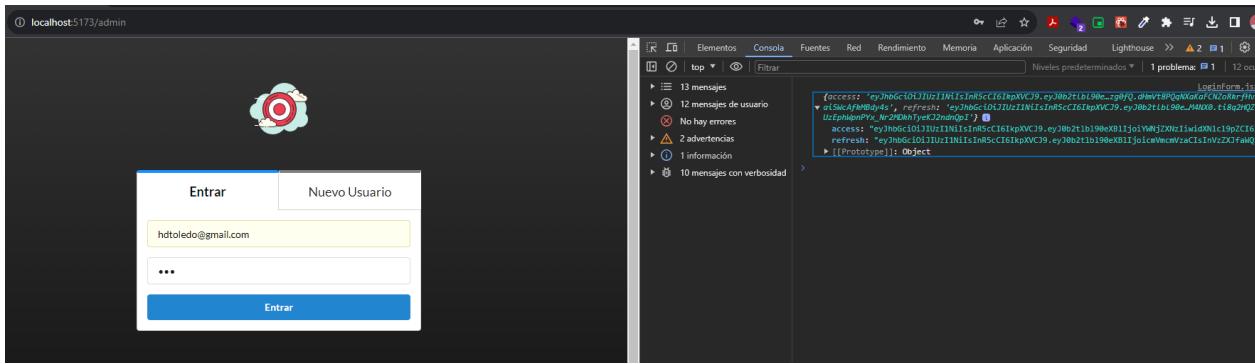
```
18   onSubmit: async (formValue) => {
19     try {
20       const response = await authController.login(formValue)
21       // login()
22       console.log(response)
23
24     } catch (error) {
25       console.log(error)
```

Recordemos dejar nuestro **user** en **null** dentro de **AuthContext.jsx**:

```
6
7   const { children } = props
8   const [ user, setUser ] = useState(null)
9   const [ token, setToken ] = useState(null)
10
11   useEffect(() => {
12     //Comprobar si el usuario esta logueado o no
```

Si vamos al navegador vamos a observar lo siguiente al iniciar la sesión:



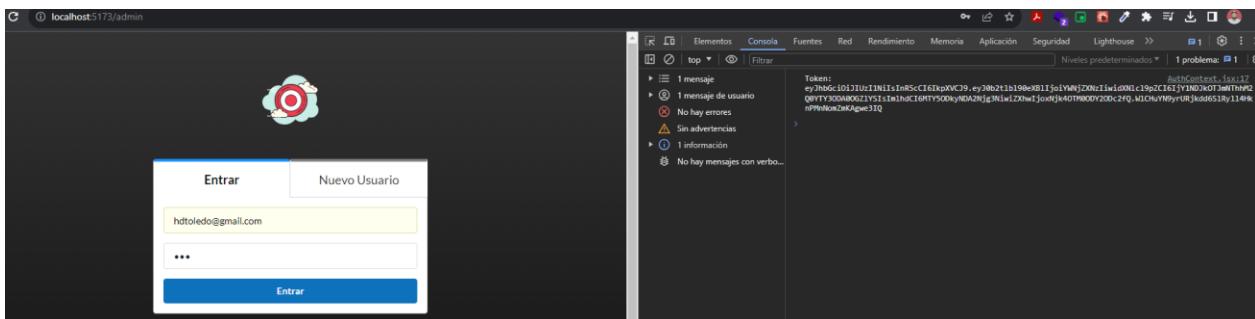


Lo que nos interesa es pasar nuestro **accessToken** de response dentro de **login** para poder validar la sesión:

```
18     onSubmit: async (formValue) => {
19         try {
20             const response = await authController.login(formValue)
21             login(response.access)
22         } catch (error) {
23             console.log(error)
24         }
25     }

```

Y al volver a darle en entrar nos debe salir el token que dejamos listo en log de **AuthContext.jsx**:

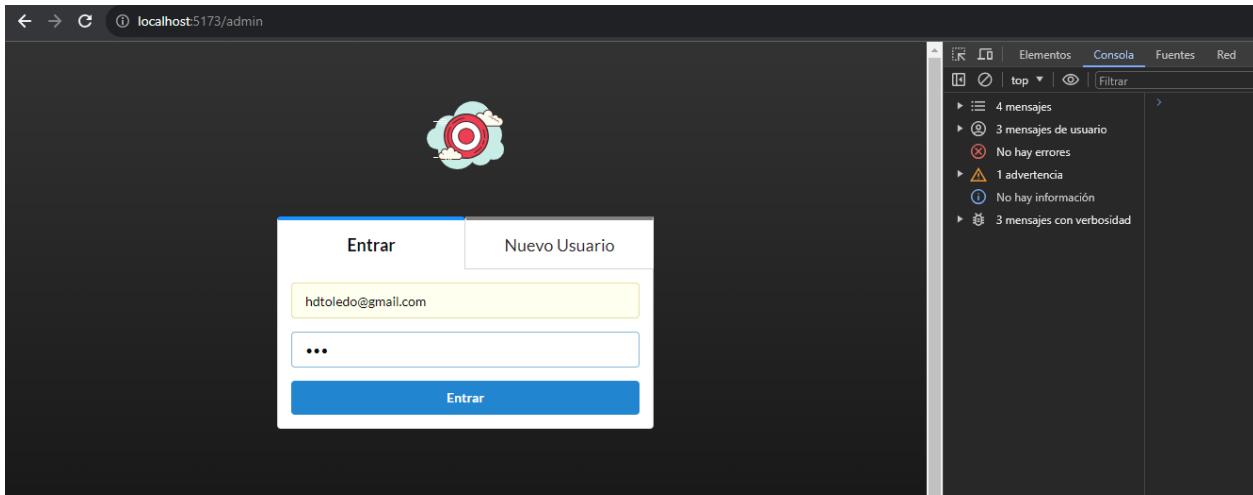


Ahora ya sabemos que esta funcionando correctamente, lo que sigue es setear nuestro token dentro de **AuthContext.jsx** y tambien nuestro **setUser** pero por el momento con informacion de ejemplo, para ello lo vamos a realizar de la siguiente manera:

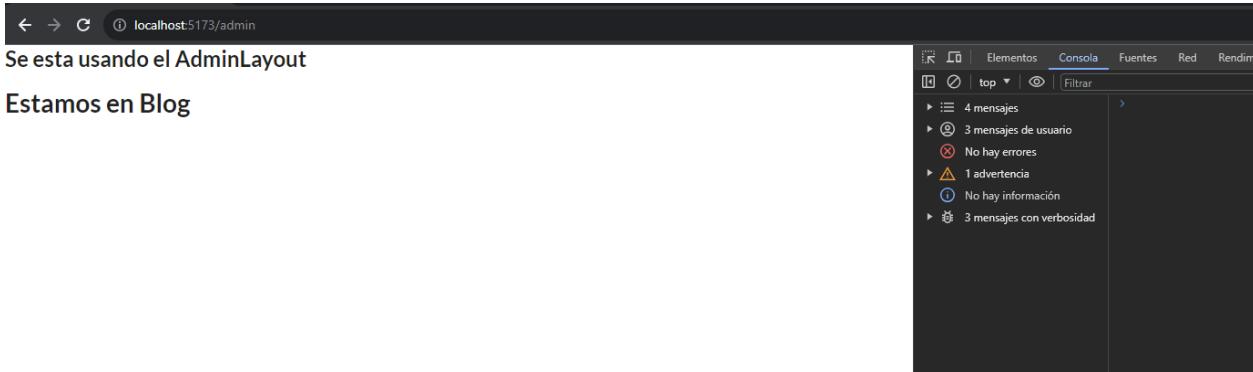
```
14
15     const login = async (accessToken) => {
16         try {
17             setUser({username: "HD"})
18             setToken(accessToken)
19         } catch (error) {
20             console.log(error)
21         }
22     }
23
24     const data = {
```



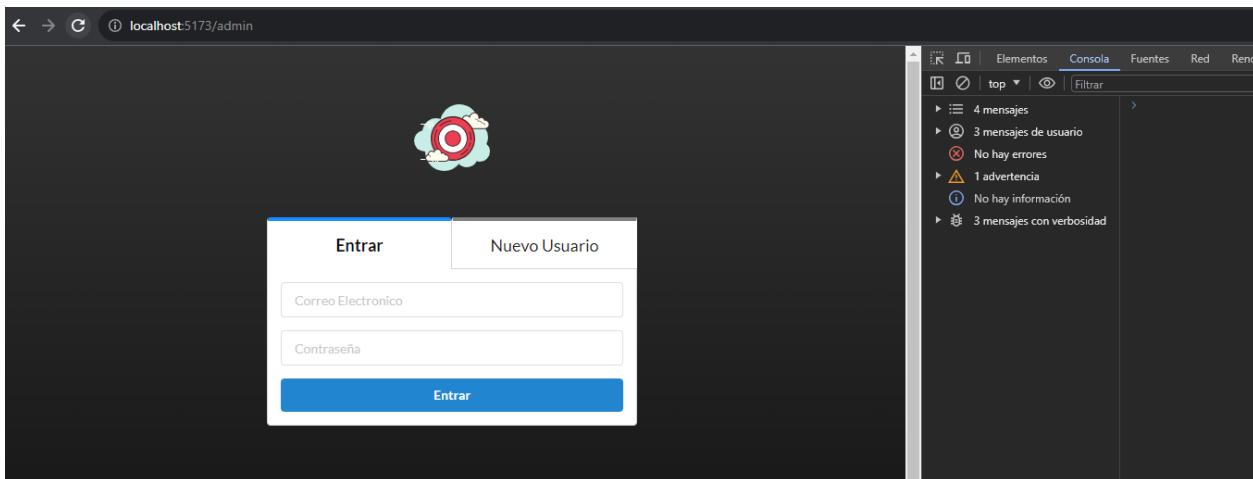
Vamos a nuestro navegador y volvemos a iniciar la sesión:



Y al darle en entrar nos ingresa:



Pero si refrescamos la página sucederá lo siguiente:



Se nos pierde el token y se cierra la sesión, porque por el momento solo estamos guardando la sesión en un estado, y por ende desaparece, para solucionarlo vamos a utilizar getMe y para almacenarlos lo haremos a través de localStorage.

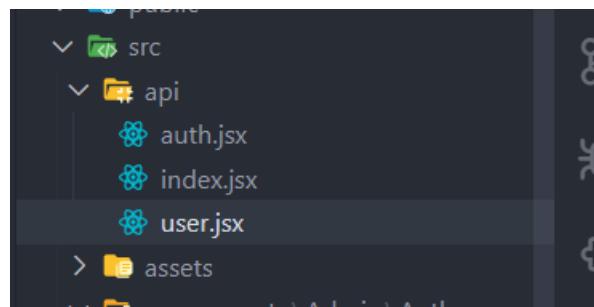


OBTENIENDO LOS DATOS DEL USUARIO

Vamos a realizar la obtención de los datos del usuario, para ello vamos a nuestro `/utils/constants.jsx` y vamos a crear nuestro `user/me`:

```
src > utils > constants.jsx > ENV > API_ROUTES
1  const SERVER_IP = "localhost:3000"
2
3  export const ENV = {
4      BASE_PATH: `http://${SERVER_IP}`,
5      BASE_API: `http://${SERVER_IP}/api/v1`,
6      API_ROUTES: {
7          REGISTER: "auth/register",
8          LOGIN: "auth/login",
9          USER_ME: "user/me",
10
11      }
12 }
```

Ahora vamos a crear dentro de `/src/api/` nuestro archivo `user.jsx`:



Recordemos ir a nuestro `index` y hacer la exportación de `user`:

```
src > api > index.jsx
1  export * from "./auth"
2  export * from "./user"
```

Ahora dentro de `user` vamos a dejarlo así:



```

src > api > user.jsx > User
1 import { ENV } from "../utils"
2
3 export class User {
4
5   baseApi = ENV.BASE_API
6
7
8   async getMe(accessToken) {
9     try {
10       const url = `${this.baseApi}/${ENV.API_ROUTES.USER_ME}`
11       const params = {
12         headers: {
13           Authorization: `Bearer ${accessToken}`,
14         },
15       }
16
17       const response = await fetch(url, params)
18       const result = await response.json()
19
20       if (response.status !== 200) throw result
21
22       return result
23
24     } catch (error) {
25       throw error
26     }
27   }
28 }

```

1. Importamos **ENV** desde un archivo llamado "**../utils**". Para utilizar variables de entorno.
2. Creamos la clase **user** y definimos una propiedad **baseApi**, que almacena la URL base de la API. Esto facilita la construcción de las URLs para realizar solicitudes.
3. Dentro de la clase **user**, se define un método **getMe(accessToken)**. Este método se encarga de obtener información del usuario autenticado.
4. En el cuerpo del método **getMe**, se construye la URL de la solicitud utilizando la URL base y la ruta específica para obtener información del usuario.
5. Se crea un objeto **params** que contiene la configuración de la solicitud. En este caso, se establece el encabezado **Authorization** con el token de acceso (**accessToken**) proporcionado. Esto es típico en las solicitudes de autenticación, ya que se utiliza el token para verificar la identidad del usuario.
6. Se utiliza la función **fetch** para realizar la solicitud a la API. La respuesta se almacena en la variable **response**.
7. Luego, se espera a que se resuelva la promesa **response.json()** para obtener los datos de la respuesta en formato JSON.



@hdtoledo

- A continuación, se verifica el estado de la respuesta. Si el estado no es igual a 200 se lanza un error.
- Finalmente, si la solicitud se completó con éxito y el estado es 200, se devuelve el resultado, que debería ser la información del usuario.

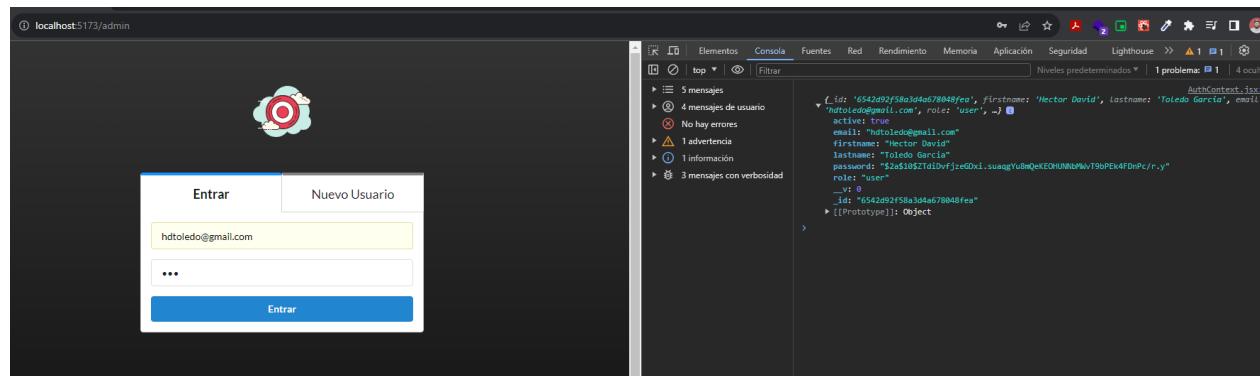
Ahora vamos a importar nuestra función dentro de **AuthContext.jsx** y lo dejamos dentro de una const:

```
src > contexts > AuthContext.jsx > UserController
1  import { useState, useEffect, createContext } from 'react'
2  import { User } from "../api"
3
4  const UserController = new User
5
6
7  export const AuthContext = createContext()
8
9  export function AuthProvider({ children }) {
10}
```

Mas abajo lo implementamos de la siguiente manera en nuestro login:

```
18
19  const login = async (accessToken) => {
20    try {
21      const response = await UserController.getMe(accessToken)
22      console.log(response)
23      //setUser({username: "HD"})
24      //setToken(accessToken)
25    } catch (error) {
26      console.log(error)
27    }
28  }
29
30  const data = {
```

Dejamos comentado por el momento nuestro setUser y setToken, mientras verificamos nuestro log en el navegador:



De esta manera estamos verificando que nuestros datos están siendo traídos correctamente con el login y podemos ver la información del usuario, lo que vamos a realizar ahora será eliminar de esa información que nos trae el password por seguridad, y dejamos de la siguiente manera:

@hdtledo

```

18
19     const login = async (accessToken) => {
20         try {
21             const response = await userController.getMe(accessToken)
22             delete response.password
23             console.log(response)
24             //setUser(response)
25             //setToken(accessToken)
26         } catch (error) {
27             console.log(error)
28         }
29     }

```

Al revisarlo en nuestro navegador vamos a ver lo siguiente:

The screenshot shows a browser window at `localhost:5173/admin`. On the left, there's a login form with fields for email and password, and a blue "Entrar" button. On the right, the developer tools' "Console" tab is open, displaying the following user object:

```

{
  id: '6542d02f58e3d6a079048fea',
  firstname: 'Hector David',
  lastname: 'Toledo Garcia',
  email: 'hdtoledo@gmail.com',
  role: 'user',
  active: true,
  password: 'hdtoledo'
}

```

Ya no nos trae el password de esta manera prevenimos por seguridad la muestra de nuestra contraseña, ahora si activamos nuestro setUser y setToken:

```

18
19     const login = async (accessToken) => {
20         try {
21             const response = await userController.getMe(accessToken)
22             delete response.password
23             setUser(response)
24             setToken(accessToken)
25
26         } catch (error) {
27             console.log(error)
28         }
29     }

```

Y si validamos de nuevo:

The screenshot shows a browser window at `localhost:5173/admin`. The page content includes "Se esta usando el AdminLayout" and "Estamos en Blog". On the right, the developer tools' "Console" tab is open, showing the user object:

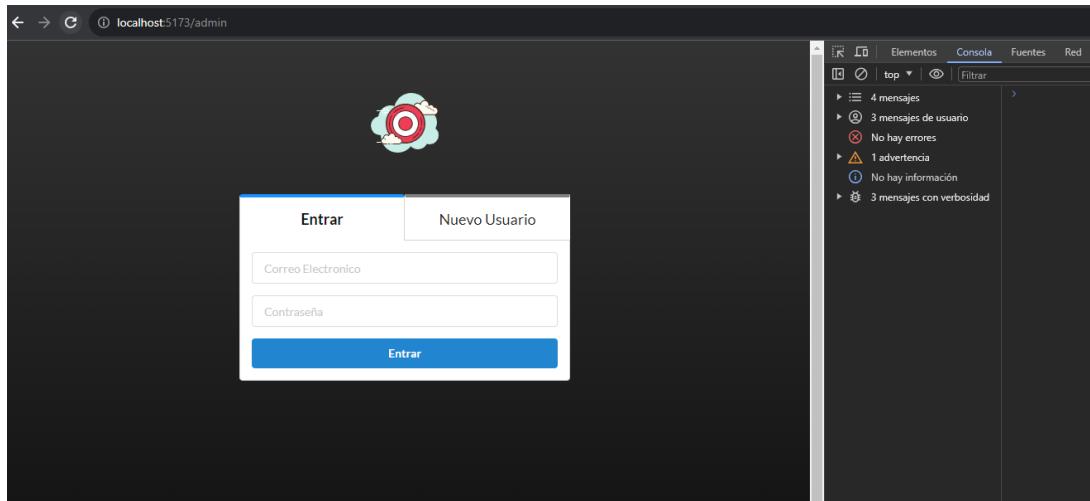
```

{
  id: '6542d02f58e3d6a079048fea',
  firstname: 'Hector David',
  lastname: 'Toledo Garcia',
  email: 'hdtoledo@gmail.com',
  role: 'user',
  active: true,
  password: 'hdtoledo'
}

```

Ya estamos iniciando la sesión e ingresando a nuestro panel administrativo. Aun no hemos realizado lo de nuestro localStorage y si refrescamos la pagina nos cerrara la sesión:





GUARDANDO Y OBTENIENDO EL TOKEN DEL LOCALSTORAGE

Nuestro token lo vamos a guardar en el **localStorage** para poder acceder a el y poder permitirle realizar otras opciones, así que nos dirigimos ahora a nuestro [/src/api/auth.jsx](#):

```
src > api > auth.jsx > Auth > setAccessToken
  51   throw error
  52 }
  53 }
  54
  55 setAccessToken(token) {
  56   localStorage.setItem("token", token)
  57 }
  58 }
```

Por el momento dejaremos de esta manera nuestro seteo del token, vamos a configurar en nuestro archivo de [/utils/constants.jsx](#) una nueva con nuestro JWT:

```
src > utils > constants.jsx > ENV
  1 const SERVER_IP = "localhost:3000"
  2
  3 export const ENV = [
  4   BASE_PATH: `http://${SERVER_IP}`,
  5   BASE_API: `http://${SERVER_IP}/api/v1`,
  6   API_ROUTES: {
  7     REGISTER: "auth/register",
  8     LOGIN: "auth/login",
  9     USER_ME: "user/me",
 10   },
 11   JWT: {
 12     ACCESS: "access",
 13     REFRESH: "refresh",
 14   },
 15 ]
```



@hdtoledo

De esta manera le pasamos nuestro token **Access** y **Refresh**, ahora en nuestro **Auth.jsx** lo dejamos de la siguiente manera:

```
src > api > auth.jsx > Auth
51           throw error
52       }
53   }
54
55   setAccessToken(token) {
56     localStorage.setItem(ENV.JWT.ACCESS, token)
57   }
58
59 }
```

De esta manera ya le estamos indicando que almacene en **localStorage** nuestro **token**, ahora debemos crear la función para obtenerlo, la cual realizamos de la siguiente manera:

```
src > api > auth.jsx > Auth
51           throw error
52       }
53   }
54
55   setAccessToken(token) {
56     localStorage.setItem(ENV.JWT.ACCESS, token)
57   }
58
59   getAccessToken() {
60     return localStorage.getItem(ENV.JWT.ACCESS)
61   }
62 }
```

Así de esta manera podemos acceder a nuestro token almacenado, ahora debemos realizar de la misma manera para nuestro **refreshAccessToken**:

```
src > api > auth.jsx > Auth
61   }
62
63   setRefreshAccessToken(token) {
64     localStorage.setItem(ENV.JWT.REFRESH, token)
65   }
66
67   getRefreshAccessToken() {
68     return localStorage.getItem(ENV.JWT.REFRESH)
69   }
70 }
```



@hdtoledo

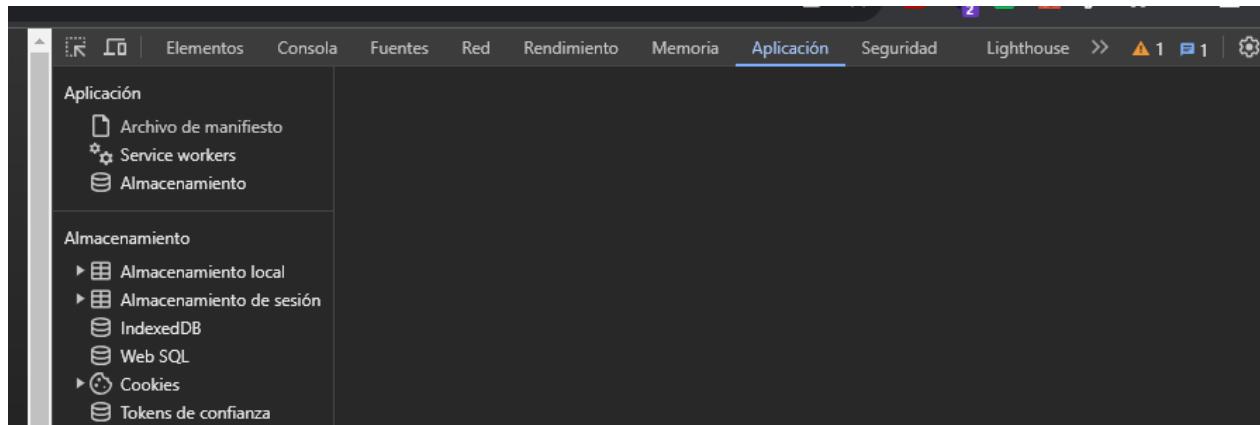
Ahora lo que nos queda por hacer sería una función también para remover los **Access** y **refresh** token, para ello los dejamos de la siguiente manera:

```
67     getRefreshAccessToken() {
68         return localStorage.getItem(ENV.JWT.REFRESH)
69     }
70
71     removeTokens() {
72         localStorage.removeItem(ENV.JWT.ACCESS)
73         localStorage.removeItem(ENV.JWT.REFRESH)
74     }
75 }
```

Ahora vamos a nuestro **loginForm.jsx** y vamos a utilizarlos de la siguiente manera:

```
17     validateonChange: false,
18     onSubmit: async (formValue) => {
19         try {
20             const response = await authController.login(formValue)
21             authController.setAccessToken(response.access)
22             authController.setRefreshAccessToken(response.refresh)
23             login(response.access)
24
25         } catch (error) {
26             console.log(error)
27         }
28     }
29 }
```

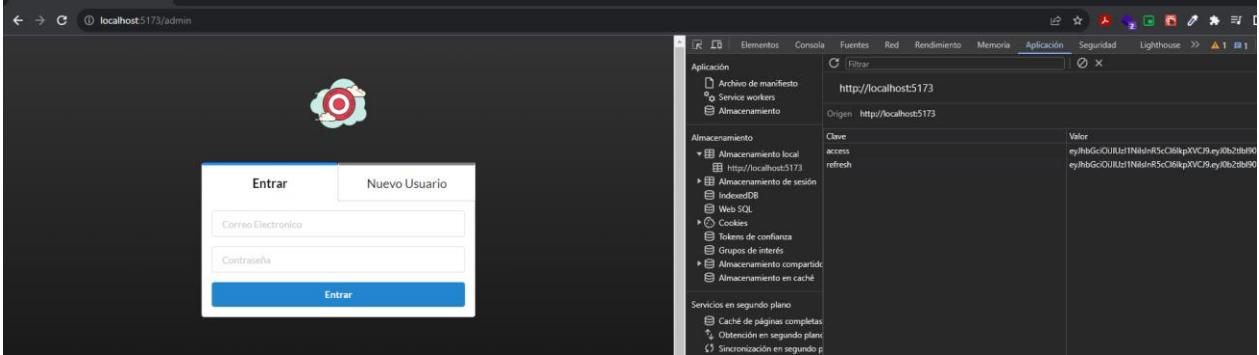
Ahora vamos a comprobar si realmente esta funcionando, para ello en nuestro navegador nos vamos a dirigir a la opción de **aplicación**:



Allí nos dirigimos a almacenamiento y vamos a nuestro servidor, iniciamos la sesión y revisamos que estén apareciendo allí:

A screenshot showing a browser window at localhost:5173/admin with the message "Se está usando el AdminLayout" and "Estamos en Blog". To its right is the Chrome DevTools Application tab, which shows the same storage information as the previous screenshot. A specific entry in the storage table is highlighted: a key named 'acces' with a value of 'eyJhbGciOiJzI1NlslhR5cO6kpVNC9eyIib2ib6QeX8ljovYW...'. Below the DevTools is a watermark with a shield logo and the handle '@hdtoledo'.

Con esto validamos que se están almacenando nuestros tokens, pero si refrescamos la pagina:



The screenshot shows a browser window with the URL `localhost:5173/admin`. On the left, there is a login form with fields for 'Correo Electronico' and 'Contraseña', and a 'Entrar' button. On the right, the browser's developer tools Network tab is open, showing a list of requests. One request for `http://localhost:5173` is selected, and its 'Headers' section shows the 'access_token' header with a long token value.

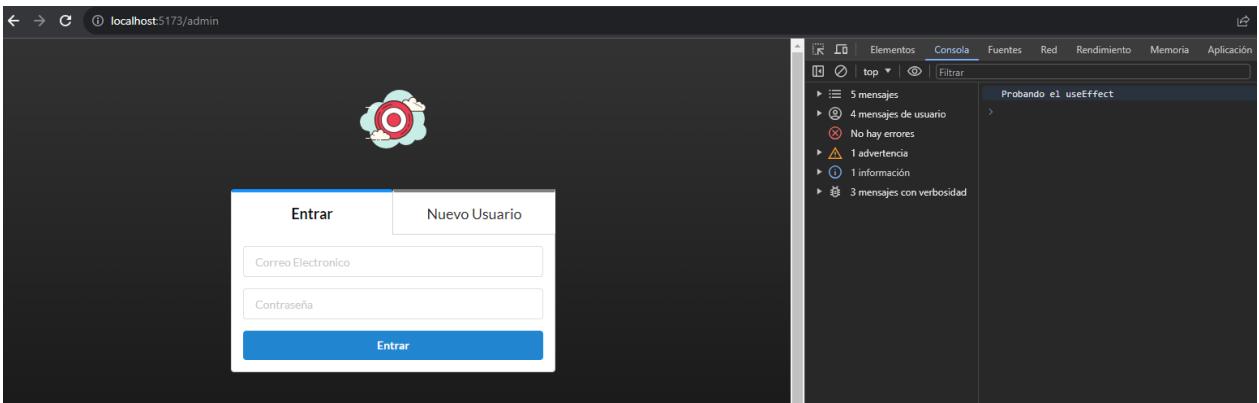
Nos aparecerá obviamente nuestro login ya que aun no hemos configurado la validación de los tokens para mantener la sesión abierta.

RECUPERANDO LA SESION DEL USUARIO:

Para recuperar la sesión lo haremos de dos maneras, para ello vamos a iniciar con la manera fácil, nos dirigimos `AuthContext.jsx` y en `useEffect` vamos a realizar un log:

```
src > contexts > AuthContext.jsx >AuthProvider > useEffect() callback
14
15     useEffect(() => {
16       |   console.log("Probando el useEffect")
17       }, [])
18
19     const login = async (accessToken) => {
20       try {
```

Si vamos a nuestro navegador y refrescamos vamos a ver lo siguiente:



The screenshot shows a browser window with the URL `localhost:5173/admin`. On the left, there is a login form. On the right, the browser's developer tools Network tab is open, showing a list of requests. The 'Console' tab is active, displaying the log message "Probando el useEffect". Below the log, the developer tools status bar shows "5 mensajes", "4 mensajes de usuario", "No hay errores", "1 advertencia", "1 información", and "3 mensajes con verbosidad".

Ya estamos observando que esta funcionando nuestro `useEffect`, ahora vamos a colocar un `useState` para manejar el estado de nuestro loading:



```
src > contexts > AuthContext.jsx > AuthProvider
11     const { children } = props
12     const [ user, setUser ] = useState(null)
13     const [token, setToken] = useState(null)
14     const [loading, setLoading] = useState(true)
15
16     useEffect(() => {
17         console.log("Probando el useEffect")
18     }, [])

```

Ahora mas abajo vamos a colocar la condición de que si loading es true no le vamos a pasar ninguna información:

```
35         login,
36     }
37
38     if (loading) return null
39
40     return <AuthContext.Provider value={data}>{children}</AuthContext.Provider>
41
42 }
```

Y cambiamos nuestro useEffect de la siguiente manera para validar, primero importamos Auth y lo colocamos dentro de una const:

```
src > contexts > AuthContext.jsx > authController
1  import { useState, useEffect, createContext } from 'react'
2  import { User, Auth } from "../api"
3
4  const userController = new User
5  const authController = new Auth
6
7  export const AuthContext = createContext()
```

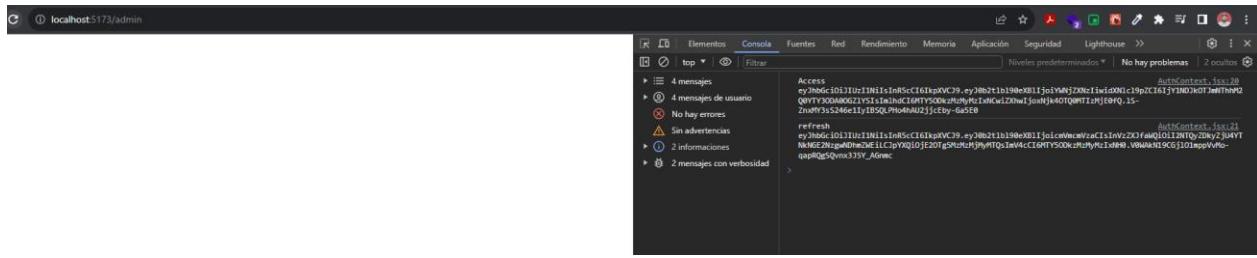
Y en nuestro useEffect:

```
15
16     useEffect(() => {
17         const accessToken = authController.getAccessToken()
18         const refreshToken = authController.getRefreshAccessToken()
19
20         console.log("Access ", accessToken)
21         console.log("refresh ", refreshToken)
22     }, [])
23
```

Dejamos de la siguiente manera para obtenerlos, pasamos por un log y en nuestro navegador vamos a observar lo siguiente:



@hdtoledo



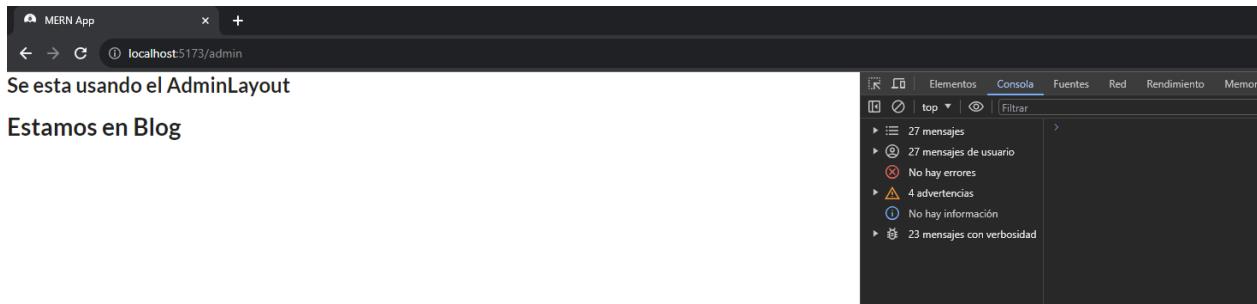
Estamos accediendo a nuestros tokens, ahora vamos a llamar a login y vamos a pasarlo el accesstoken, ahora vamos a modificar un poco la función, para poder pasar estos cambios debo indicarle que setLoading debe ser false, para ello recordemos que estamos trabajando con funciones asíncronas y la manera correcta para poder aplicarlo sería de la siguiente manera:

```

15  useEffect(() => {
16
17
18    (async () => {
19      const accessToken = authController.getAccessToken()
20      const refreshToken = authController.getRefreshAccessToken()
21      await login(accessToken)
22      setLoading(false)
23    })()
24
25  }, [])
26
27  const login = async (accessToken) => {
28    try {

```

Ahora si vamos a revisar nuestro navegador notamos que ya estamos dentro y que al momento de refrescar no se vuelve a salir:



Ahora si nos vamos a la aplicación en el navegador y elimino los tokens:

The screenshot shows the Chrome DevTools Application tab open. On the left, there's a sidebar with sections for Archivo de manifiesto, Service workers, and Almacenamiento. The Almacenamiento section is expanded, showing sub-sections for Almacenamiento local, Almacenamiento de sesión, IndexedDB, Web SQL, Cookies, Tokens de confianza, Grupos de interés, and Almacenamiento compartido. Under Almacenamiento local, there's an entry for http://localhost:5173. The main area shows a table with two columns: Clave and Valor. The first row has 'access' as the clave and a long token as the valor. The second row has 'refresh' as the clave and another long token as the valor. A context menu is open over the 'refresh' row, with options: Opciones de encabezado, Actualizar, Editar "Clave", and Eliminar. The 'Eliminar' option is highlighted.

Clave	Valor
access	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tl
refresh	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tl

Ya tendremos que iniciar la sesión de nuevo:

Y si volvemos a iniciar sesión nos saldrán nuevos tokens:

The screenshot shows the Network tab of a browser's developer tools. The left sidebar lists resources: 'Archivo de manifesto' (Manifest file), 'Service workers', and 'Almacenamiento' (Storage). The main table lists network requests for 'localhost:5173'. The first request is 'Almacenamiento local' (LocalStorage) with URL 'https://localhost:5173'. The second request is 'Almacenamiento de sesión' (Session Storage) with URL 'https://localhost:5173'. Both requests have status 'OK' and are timestamped '2023-08-29T11:51:28.90z'. The table has columns for 'Clave' (Key), 'Valor' (Value), and 'access' (Access).

Clave	Valor
refresh	eyJhbGciOiJlIzI1NjklR5C6l8kpXVCJ9.eyJ0b2tlbi9xIjIiLCJleHAiOjE2MjUyOTQwODQsImV4cCI6MTYyNTI4NDQwOH0.eyJhbGciOiJlIzI1NjklR5C6l8kpXVCJ9.eyJ0b2tlbi9xIjIiLCJleHAiOjE2MjUyOTQwODQsImV4cCI6MTYyNTI4NDQwOH0.

Ahora lo que no tenemos establecido es que no estamos verificando los tokens, si se han aun activos y demás, ahora vamos a empezar a realizar este proceso.



RECUPERANDO LA SESION COMPLETA

Ahora vamos a empezar a validar todo en cuanto a nuestros tokens, lo primero que realizaremos será si existe el token o viene null, para ello nos ubicamos en **AuthContext.jsx** y vamos a dejar de la siguiente manera:

```
16  useEffect(() => {
17
18    (async () => {
19      const accessToken = authController.getAccessToken()
20      const refreshToken = authController.getRefreshAccessToken()
21
22      if (!accessToken || !refreshToken) {
23        setLoading(false)
24        return
25      }
26
27      setLoading(false)
28    })()
29
```

Lo siguiente es crear una función para cerrar la sesión para ello nos ubicamos debajo de nuestro login:

```
40    }
41  }
42
43
44  const logout = () => {
45    setUser(null)
46    setToken(null)
47    authController.removeTokens()
48
49
50  const data = {
51    accessToken: token,
52    user,
```

Y ahora lo implementamos en nuestra condición:

```
16
17
18    (async () => {
19      const accessToken = authController.getAccessToken()
20      const refreshToken = authController.getRefreshAccessToken()
21
22      if (!accessToken || !refreshToken) {
23        logout()
24        setLoading(false)
25        return
26      }
27
28      setLoading(false)
29    })()
30
```

De esta manera hacemos la implementación de nuestra función de logout.



@hdtoledo

Ahora en caso de que existan los tokens debemos verificar si son válidos, para ello vamos a crear en la ruta de `/utils/` nuestro archivo `token.jsx` en donde vamos a dejar nuestras utilidades, en este caso para nuestros tokens, nuestro archivo lo dejaremos así:



Hacemos nuestra exportación en `index.jsx`:

```
src > utils > index.jsx
1  export * from "./constants"
2  export * from "./token"
```

A continuación, vamos a utilizar la dependencia [jwt-decode](#)

The screenshot shows the Yarn package page for `jwt-decode@4.0.0`. The page includes a sidebar with navigation links like Back to search, Information, Website, Repository, Runkit, Tags, Versions, and Files. The main content area displays the package name `jwt-decode`, version `4.0.0`, and a note that the package doesn't have a commonjs entry point. It also shows the `README` file content, which describes it as a browser library for decoding JWTs. A large blue banner with the Auth0 by Okta logo is prominently displayed.

Hacemos nuestra instalación con el comando `yarn add jwt-decode@3.1.2`

```
LAPTOPHDMI1 D:\DATA\..\..\client yarn 18.16.0 1m 45.972s
{ @hdtledo } yarn add jwt-decode@3.1.2
```



Y ahora procedemos a dejar nuestro **token.jsx** de la siguiente manera:

```
src > utils > token.jsx > hasExpiredToken
1 import jwtDecode from "jwt-decode"
2
3 export const hasExpiredToken = (token) => {
4   const { exp } = jwtDecode(token)
5   const currentDate = new Date().getTime()
6
7   if (exp ≤ currentDate) {
8     return true
9   }
10
11  return false
12}
```

1. Importamos la biblioteca **jwt-decode**. Esta biblioteca se utiliza para decodificar tokens JWT y obtener información sobre su contenido.
2. Definimos la función **hasExpiredToken**, que toma un token JWT como argumento.
3. Utilizamos **jwtDecode** para decodificar el token, lo que nos permite acceder a su contenido. Algunos de los datos importantes que obtenemos del token decodificado son **exp**, que representa la fecha de vencimiento del token, y **currentData**, que representa la fecha actual en milisegundos.
4. Comparamos el valor de **exp** (fecha de vencimiento del token) con **currentData** (fecha actual). Si la fecha de vencimiento (**exp**) es menor o igual a la fecha actual, significa que el token ha caducado.
5. Si el token ha caducado (es decir, **exp** es menor o igual que **currentData**), la función devuelve **true**. De lo contrario, devuelve **false**.

Esta función se utiliza para verificar si un token JWT ha expirado o no. Si el token ha caducado, se devuelve **true**, lo que indica que el token ya no es válido. Si el token aún es válido, se devuelve **false**. Esta es una funcionalidad importante en la autenticación para garantizar que los usuarios no utilicen tokens que hayan expirado.

Ahora nos vamos para **AuthContext.jsx** y hacemos la importación:

```
src > contexts > AuthContext.jsx > ...
1 import { useState, useEffect, createContext } from 'react'
2 import { User, Auth } from "../api"
3 import { hasExpiredToken } from "../utils"
4
5 const userController = new User
```



@hdtoledo

Y hacemos la implementación dentro de nuestro **useEffect** de la siguiente manera:

```
28      if (hasExpiredToken(accessToken)) {
29        if (hasExpiredToken(refreshToken)) {
30          logout()
31        } else {
32          |
33        }
34      } else {
35        await login(accessToken)
36      }
37    }
38
39    setLoading(false)
40
41  })()
42
43 }, [])
```

Le indicamos que si el token ha expirado nos haga el **logout** lo mismo que si el **refresh**, en caso contrario tenemos un **await** para mantener el login, y ahora vamos a implementar un **reLogeo** en caso contrario, para ello creamos lo siguiente:

```
30      if (hasExpiredToken(refreshToken)) {
31        logout()
32      } else {
33        await reLogin(refreshToken)
34      }
35    } else {
36      await login(accessToken)
37    }
38
39    setLoading(false)
40
41  })()
42 }, [])
43
44 const reLogin = async (refreshToken) => {
45   console.log(refreshToken)
46 }
47
48 const login = async (accessToken) => {
49   try {
50     const response = await userController.login(accessToken)
```

Y hacemos la implementación de **reLogin** en nuestro caso contrario, ahora realizamos la función par a nuestro **refreshAccessToken**, para ello vamos a crear dentro de **Auth.jsx** la función:



@hdtoledo

```

src > api > auth.jsx > Auth > refreshAccessToken
54
55     async refreshToken (refreshToken) {
56         try {
57             }
58         catch (error) {
59             throw error
60         }
61     }
62
63     setAccessToken(token) {
64         localStorage.setItem(ENV.JWT.ACCESS, token)
65     }
66

```

Ahora vamos a nuestro `/utils/constants.jsx` y vamos a crear la ruta para nuestro refresh:

```

src > utils > constants.jsx > ENV > API_ROUTES
1 const SERVER_IP = "localhost:3000"
2
3 export const ENV = {
4     BASE_PATH: `http://${SERVER_IP}`,
5     BASE_API: `http://${SERVER_IP}/api/v1`,
6     API_ROUTES: [
7         REGISTER: "auth/register",
8         LOGIN: "auth/login",
9         REFRESH_ACCESS_TOKEN: "/auth/refresh_access_token",
10        USER_ME: "user/me",
11    ],
12    JWT: {
13        ACCESS: "access",
14        REFRESH: "refresh",
15    },
16 }

```

Ahora en nuestro `Auth.jsx` vamos a terminar de construir la ruta de la siguiente manera:

```

src > api > auth.jsx > Auth
54
55     async refreshToken (refreshToken) {
56         try {
57             const url = `${this.baseApi}/${ENV.API_ROUTES.REFRESH_ACCESS_TOKEN}`
58             const params = {
59                 method: "POST",
60                 headers: {
61                     "Content-Type": "application/json",
62                 },
63                 body: JSON.stringify({
64                     token: refreshToken,
65                 }),
66             }
67
68             const response = await fetch(url, params)
69             const result = await response.json()
70
71             if (response.status !== 200) throw result
72
73             return result
74
75         } catch (error) {
76             throw error
77         }
78     }
79
80     setAccessToken(token) {
81         localStorage.setItem(ENV.JWT.ACCESS, token)

```

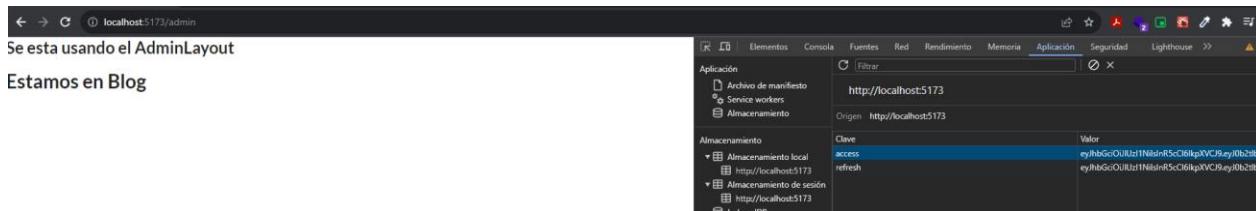


@hdtoledo

Notemos que la construcción de estas funciones es muy similar y que simplemente estamos cambiando unos parámetros para validarlos, ahora vamos a implementarlo en nuestro **reLogin** de **AuthContext.jsx**:

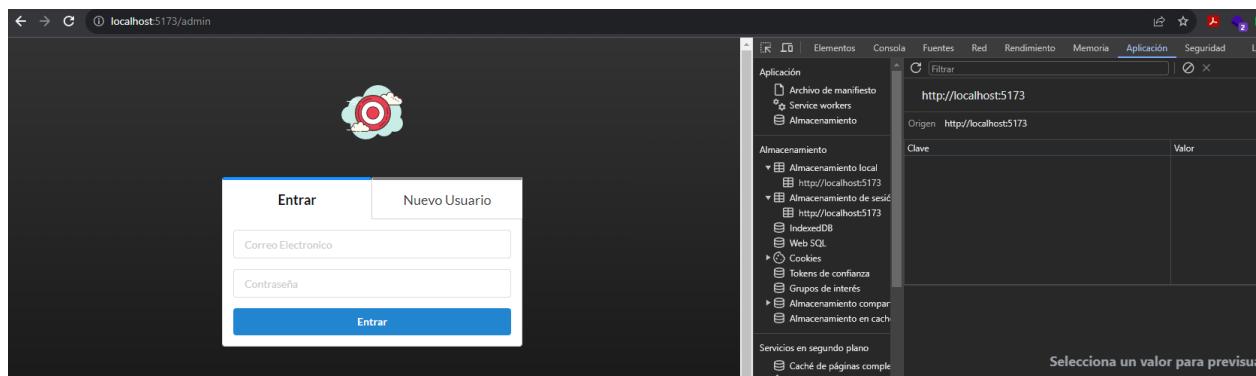
```
41     }})
42   }, [])
43 
44   const reLogin = async (refreshToken) => {
45     try {
46       const { accessToken } = await authController.refreshAccessToken(refreshToken)
47       authController.setAccessToken(accessToken)
48       await login(accessToken)
49     } catch (error) {
50       console.error(error)
51     }
52   }
53 
54   const login = async (accessToken) => {
55     try {
56       const user = await authController.login(accessToken)
```

Ahora si nos vamos a nuestra aplicación en nuestro navegador, si tenemos la sesión iniciada vamos a eliminar el token y refrescamos:



The screenshot shows a browser window with the URL `localhost:5173/admin`. The page content says "Se esta usando el AdminLayout" and "Estamos en Blog". In the background, the browser's developer tools Network tab is open, showing a request to `http://localhost:5173`. The response status is 200 OK. The response body contains JSON data related to session storage.

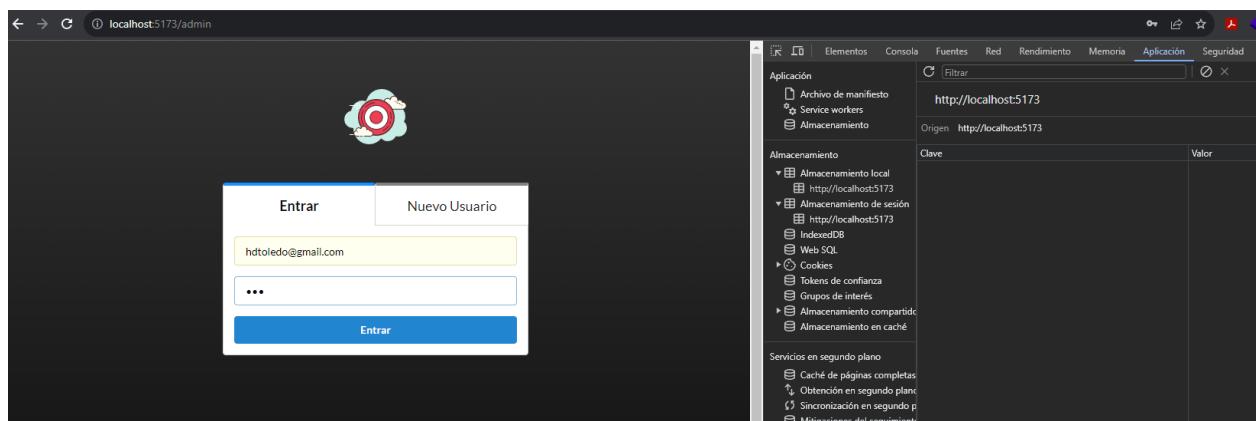
Clave	Valor
access	eyJhbGciOiJIUzI1NiJ9.RSc6IkpxVCJ9eyJ0b2tlbiI6ImF1dGhvcml0eSIsInR5cGUiOiJsb2dpbiJ9
refresh	eyJhbGciOiJIUzI1NiJ9.RSc6IkpxVCJ9eyJ0b2tlbiI6ImF1dGhvcml0eSIsInR5cGUiOiJsb2dpbiJ9



The screenshot shows a browser window with the URL `localhost:5173/admin`. The page content shows a login form with fields for "Correo Electronico" and "Contraseña", and buttons for "Entrar" and "Nuevo Usuario". In the background, the browser's developer tools Network tab is open, showing a request to `http://localhost:5173`. The response status is 200 OK. The response body contains JSON data related to session storage.

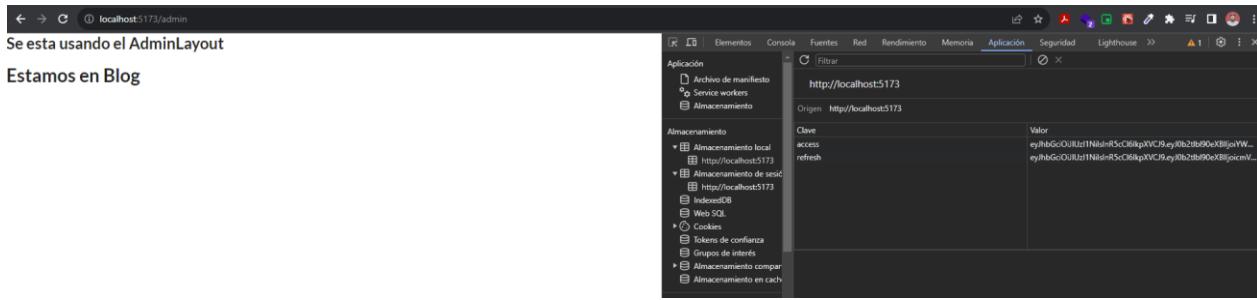
Clave	Valor
access	
refresh	

Automáticamente al refrescar se nos cerrara la sesión y nos pedirá logueo, si hacemos nuevamente el login:



The screenshot shows a browser window with the URL `localhost:5173/admin`. The page content shows a login form with fields for "Correo Electronico" containing "hdtoledo@gmail.com" and "Contraseña" containing "***". The "Entrar" button is highlighted. In the background, the browser's developer tools Network tab is open, showing a request to `http://localhost:5173`. The response status is 200 OK. The response body contains JSON data related to session storage.

Clave	Valor
access	
refresh	



Y si refrescamos nuestro navegador notaremos que ya no se nos cierra nuestro logueo y estamos dentro de nuestra aplicación.

CERRAR SESIÓN

Vamos a exportar la función de logout para que la podamos utilizar desde cualquier botón y cerrar la sesión, para ello dentro de **AuthContext.jsx** vamos a exportarla:

```
src > contexts >  AuthContext.jsx >  AuthProvider >  data
67  const logout = () => {
68    setUser(null)
69    setToken(null)
70    authController.removeTokens()
71  }
72
73  const data = [
74    accessToken: token,
75    user,
76    login,
77    logout,
78  ]
79
80  if (loading) return null
81
82  return <AuthContext.Provider value={data}>{children}</AuthContext.Provider>
83
```

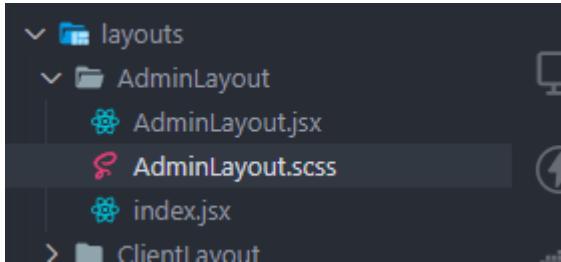
De esta manera ya tendremos nuestro Registro, Login completamente funcional y validado.



@hdtoledo

ESTRUCTURA DEL PANEL

Vamos a empezar a crear la estructura de nuestro panel administrativo para ello vamos a nuestro `/layout/AdminLayout` y vamos a crear nuestro archivo `AdminLayout.scss`



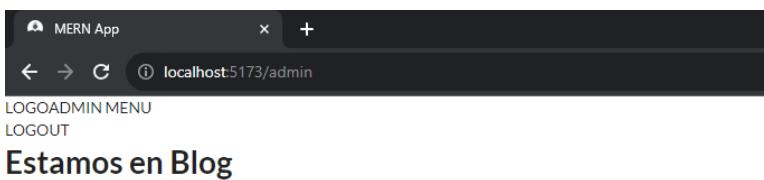
Y vamos a hacer la importación de nuestro archivo dentro de `AdminLayout.jsx`:

```
src > layouts > AdminLayout > AdminLayout.jsx > ...
1  import React from 'react'
2  import './AdminLayout.scss'
3
```

Y vamos a dejar la siguiente estructura:

```
5
6  export function AdminLayout(props) {
7    const { children } = props
8    return (
9      <div className="admin-layout" >
10        <div className="admin-layout__left" >
11          <div>LOGO</div>
12          <div>ADMIN MENU</div>
13        </div>
14        <div className="admin-layout__right" >
15          <div className="admin-layout__right-header" >
16            <span>LOGOUT</span>
17          </div>
18          <div className="admin-layout__right-content">{children}</div>
19        </div>
20      </div>
21    )
22  }
23
```

Si lo vamos a ver en nuestro navegador:

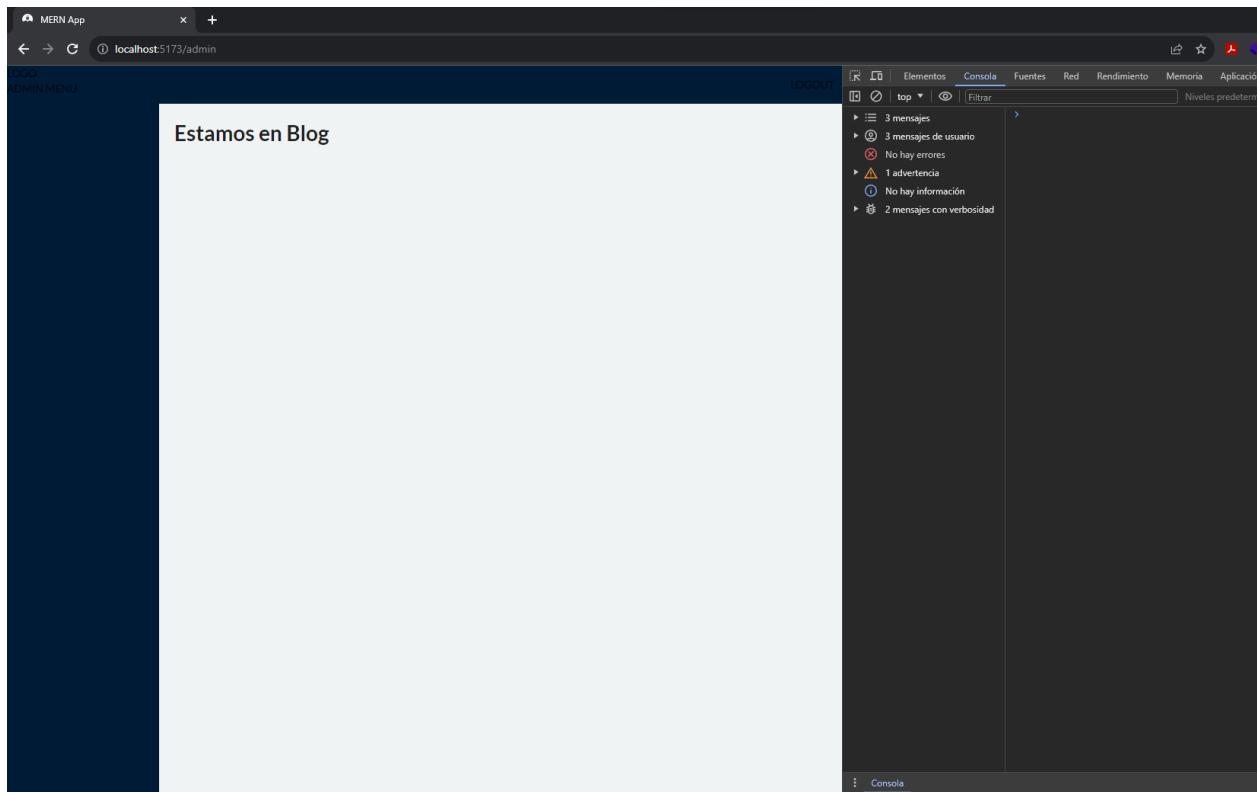


Ahora vamos a estilizar nuestro admin layout:

```
src > layouts > AdminLayout > AdminLayout.scss > ...
1  @import "/src/scss/index.scss";
2
3  $widthLeftMenu: 200px;
4
5  .admin-layout {
6    display: flex;
7    min-height: 100vh;
8    max-height: 100vh;
9    background-color: $background-grey;
10
11  &__left {
12    background-color: $background-dark;
13    width: $widthLeftMenu;
14
15    .logo {
16      height: 30px;
17      width: 100%;
18      margin: 10px 0;
19    }
20  }
21
22  &__right {
23    width: calc(100% - $widthLeftMenu);
24
25  &-header {
26    background-color: $background-dark;
27    height: 50px;
28    display: flex;
29    align-items: center;
30    justify-content: flex-end;
31    padding: 0 10px;
32  }
33
34  &-content {
35    position: relative;
36    margin: 20px;
37  }
38}
39}
40}
```

Esto se nos vera reflejado de la siguiente manera en el navegador:





ADMIN MENU

Vamos a empezar a configurar nuestro menu de admin para ello vamos a importar nuestro icono en **AdminLayout.jsx**:

```
src > layouts > AdminLayout > AdminLayout.jsx > ...
1  import React from "react"
2  import { iconLogo } from "../../assets"
3  import "./AdminLayout.scss";
4
5  export function AdminLayout(props) {
```

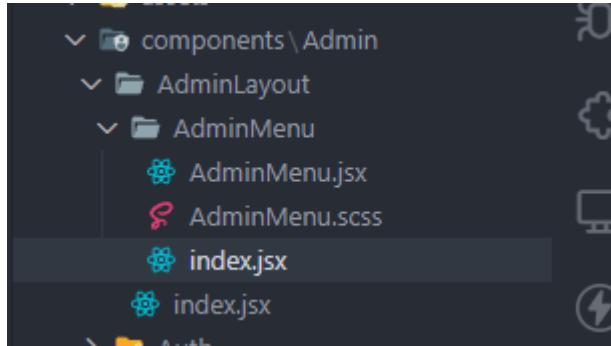
Y hacemos la implementación:

```
5  export function AdminLayout(props) {
6    const { children } = props;
7    return [
8      <div className="admin-layout">
9        <div className="admin-layout__left">
10          |   <img src={iconLogo} className="logo" />
11          |   <div>ADMIN MENU</div>
12        </div>
13        <div className="admin-layout__right">
```



@hdtoledo

Ahora para hacer mucho más escalable nuestro menú vamos a crear dentro de `/components/Admin/` nuestra carpeta **AdminLayout** y dentro vamos a tener nuestra carpeta **AdminMenu** y mantendremos los **index.jsx** dentro de cada una de estas carpetas para realizar las exportaciones además de tener nuestro **AdminMenu.jsx** y **AdminMenu.scss**:



Ahora dentro de nuestro `/AdminLayout/index.jsx`

```
src > components > Admin > AdminLayout > index.jsx
1   export * from "./AdminMenu"
```

Ahora dentro de nuestro `/AdminMenu/index.jsx`

```
src > components > Admin > AdminLayout > AdminMenu > index.jsx
1   export * from "./AdminMenu"
```

En nuestro archivo **AdminMenu.jsx**:

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1   import React from 'react'
2
3   export function AdminMenu() {
4     return (
5       <div className="admin-menu">
6         <h2>Admin Menu</h2>
7       </div>
8     )
9   }
10
```

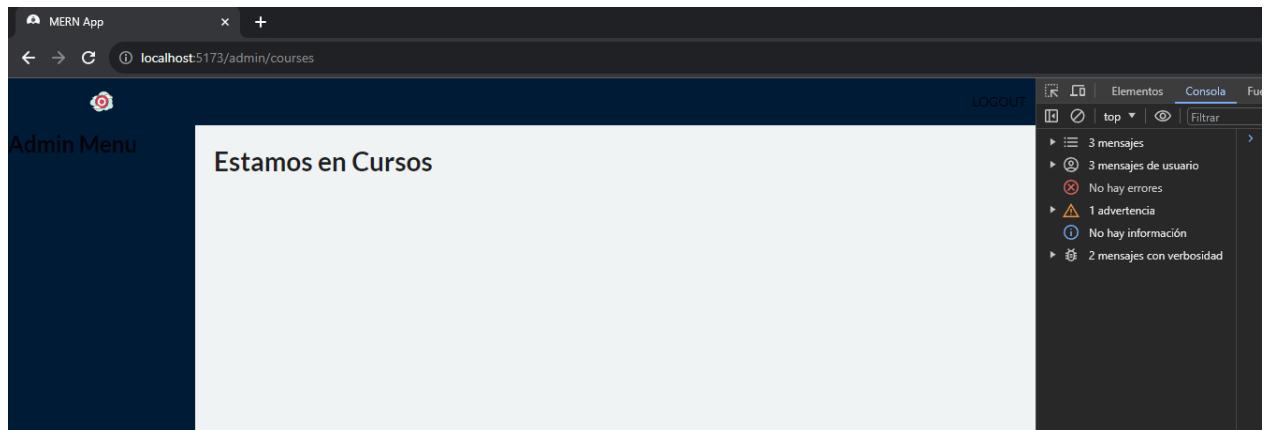
Ahora en nuestro **AdminLayout.jsx** vamos a hacer la importación:

```
src > layouts > AdminLayout > AdminLayout.jsx > ...
1  import React from "react"
2  import { iconLogo } from "../../assets"
3  import { AdminMenu } from "../../components/Admin/AdminLayout"
4  import "./AdminLayout.scss";
5
6  const AdminLayout = () =>
```

Ahora hacemos la implementación de **AdminMenu**:

```
8   return (
9     <div className="admin-layout">
10       <div className="admin-layout__left">
11         |   <img src={iconLogo} className="logo" />
12         |   <AdminMenu />
13       </div>
14       <div className="admin-layout__right">
15         <div className="admin-layout__right-header">
```

Si vamos a nuestro navegador debemos observar nuestro menú:



Ahora nos ubicamos en **AdminMenu.jsx** y vamos a importar Menu e Icon de semantic UI:

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1  import React from 'react'
2  import { Menu, Icon } from "semantic-ui-react"
3
4  export function AdminMenu() {
```

Ahora empezamos a dejar la siguiente estructura:



```
5
4  export function AdminMenu() {
5    return (
6      <Menu fluid vertical icon text className="admin-menu">
7        <Menu.Item>
8          <Icon name="user outline" />
9          Usuario
10         </Menu.Item>
11       </Menu>
12     )
13   }
14 }
```

En nuestro archivo **AdminMenu.scss**

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.scss > ...
1  @import "/src/scss/index.scss";
2
3  .ui.menu.admin-menu {
4    > .item {
5      display: flex;
6      color: $text-light;
7      padding: 15px 20px !important;
8      margin: 0 !important;
9      border-radius: 0 !important;
10
11      &:hover,
12      &.active,
13      &.active:hover {
14        color: $text-light;
15        background-color: $primary-hover;
16      }
17
18      > i {
19        margin: 0 !important;
20        margin-right: 10px !important;
21      }
22    }
23  }
24 }
```

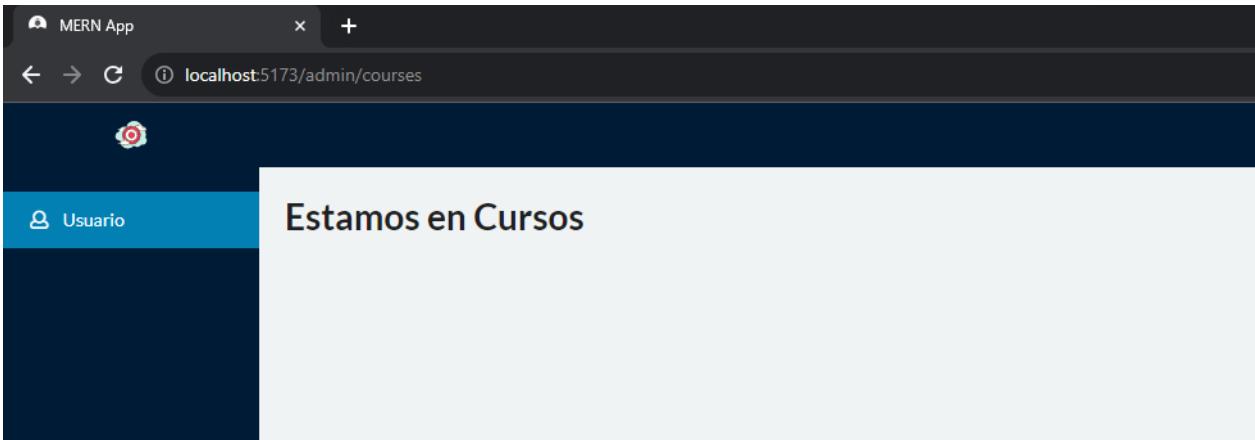
Y ahora vamos a hacer la importación en **AdminMenu.jsx**

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1  import React from 'react'
2  import { Menu, Icon } from "semantic-ui-react"
3  import "./AdminMenu.scss"
4
5  export function AdminMenu() {
```



@hdtoledo

De esta manera si nos vamos a nuestro navegador vamos a obtener lo siguiente:



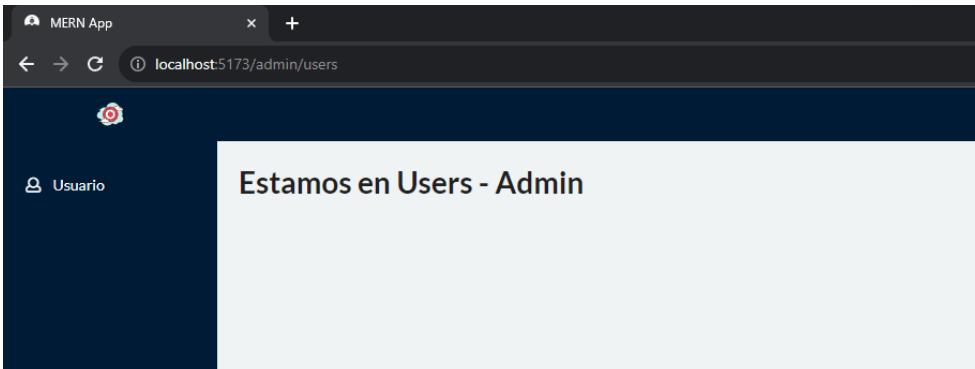
Ahora vamos a hacer que funcione nuestro menú de usuario al momento de darle clic, para ello vamos a realizar la importación de **Link** de React Router Dom:

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1  import React from 'react'
2  import { Menu, Icon } from "semantic-ui-react"
3  import { Link } from "react-router-dom"
4  import "./AdminMenu.scss"
5
```

Y ahora para implementarlo hacemos lo siguiente:

```
7
8      return [
9        <Menu fluid vertical icon text className="admin-menu">
10       <Menu.Item as={Link} to="/admin/users">
11         <Icon name="user outline" />
12         Usuario
13       </Menu.Item>
14     </Menu>
15   ]
16 }
```

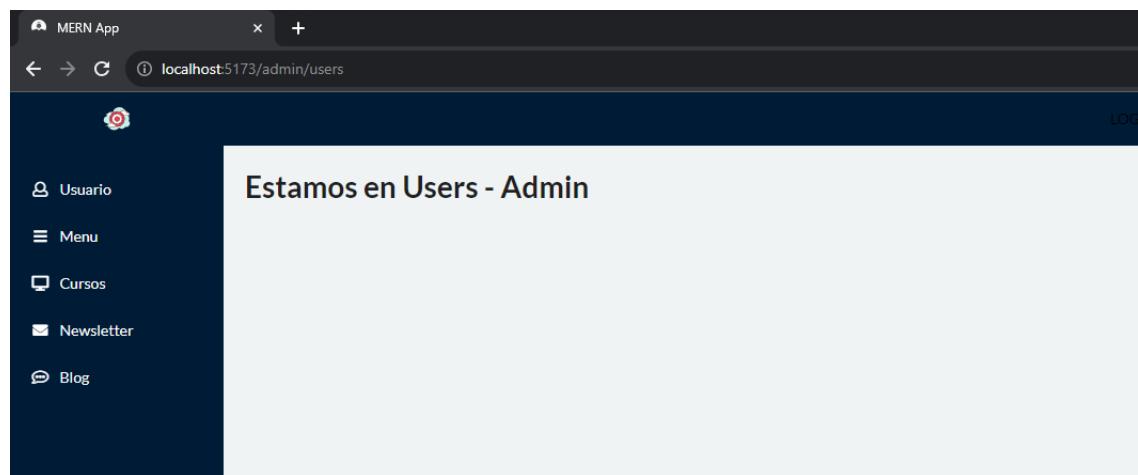
Si nos vamos a nuestro navegador y damos clic sobre el menú:



Ahora procedemos a aplicar los demás menus y lo dejamos de la siguiente manera:

```
6  export function AdminMenu() {
7
8    return [
9      <Menu fluid vertical icon text className="admin-menu">
10
11        <Menu.Item as={Link} to="/admin/users">
12          <Icon name="user outline" />
13          Usuario
14        </Menu.Item>
15
16        <Menu.Item as={Link} to="/admin/menu">
17          <Icon name="bars" />
18          Menu
19        </Menu.Item>
20
21        <Menu.Item as={Link} to="/admin/courses">
22          <Icon name="computer" />
23          Cursos
24        </Menu.Item>
25
26        <Menu.Item as={Link} to="/admin/newsletter">
27          <Icon name="mail" />
28          Newsletter
29        </Menu.Item>
30
31        <Menu.Item as={Link} to="/admin/blog">
32          <Icon name="comment alternate outline" />
33          Blog
34        </Menu.Item>
35
36      </Menu>
37    ]
38  }
39
```

De esta manera obtendríamos en nuestro navegador lo siguiente:



Y si verificamos cada uno de los menus nos llevan a los diferentes enlaces:



A screenshot of a web browser window titled "MERN App". The address bar shows the URL "localhost:5173/admin/menu". The page content is titled "Menu" and lists several items: "Usuario", "Menu", "Cursos", "Newsletter", and "Blog". Each item has a small icon next to it. On the far right of the page, there is some very faint text that appears to start with "LO".

A screenshot of a web browser window titled "MERN App". The address bar shows the URL "localhost:5173/admin/courses". The page content is titled "Estamos en Cursos". The left sidebar contains the same navigation items as the previous screen: "Usuario", "Menu", "Cursos", "Newsletter", and "Blog".



@hdtoledo

A screenshot of a web browser window titled "MERN App". The address bar shows "localhost:5173/admin/newsletter". The page content area displays the text "Estamos en newsletter". On the left side, there is a sidebar with a dark blue background containing the following menu items:

- User icon: Usuario
- Menu icon: Menu
- Courses icon: Cursos
- Newsletter icon: Newsletter
- Blog icon: Blog

The "Newsletter" item is highlighted with a light gray background.

A screenshot of a web browser window titled "MERN App". The address bar shows "localhost:5173/admin/blog". The page content area displays the text "Estamos en Blog". On the left side, there is a sidebar with a dark blue background containing the following menu items:

- User icon: Usuario
- Menu icon: Menu
- Courses icon: Cursos
- Newsletter icon: Newsletter
- Blog icon: Blog

The "Blog" item is highlighted with a light blue background.



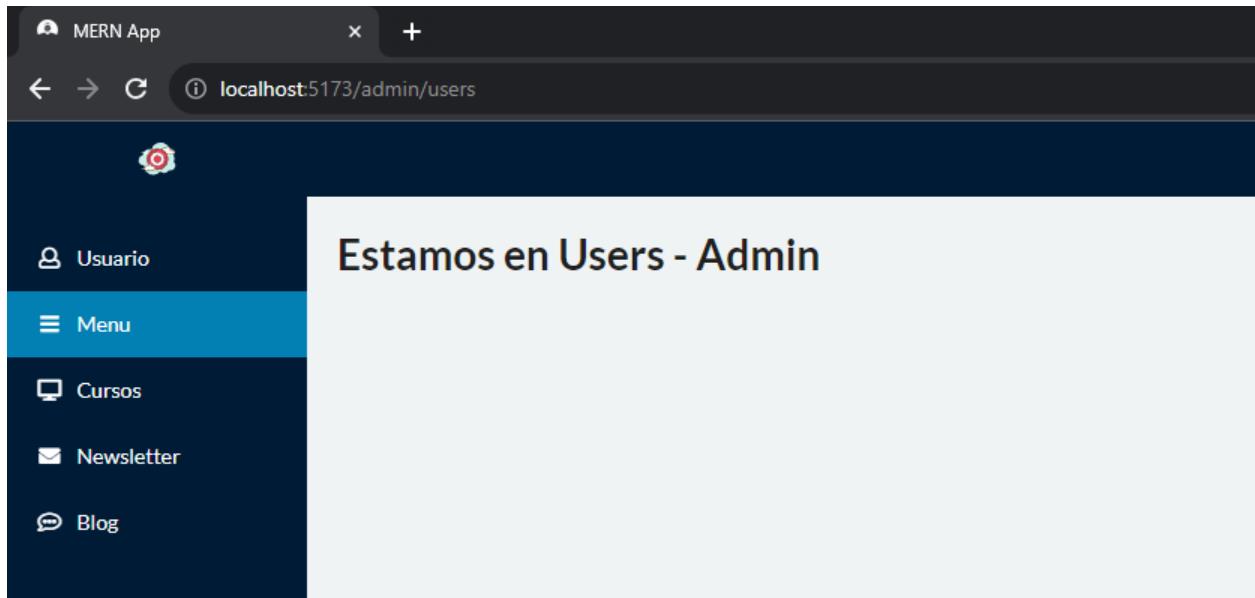
@hdtoledo

ACTIVANDO EL MENU ACTIVO

Ahora vamos a realizar la activación de nuestro menú que al momento de presionarlo y estemos allí se nos quede activo para saber en donde estamos ubicados, para ello es muy sencillo vamos a hacer lo siguiente aplicando la propiedad active de Menu.Item:

```
14      </Menu.Item>
15
16      <Menu.Item as={Link} to="/admin/menu" active>
17          <Icon name="bars" />
18          Menu
19      </Menu.Item>
20
```

Si lo verificamos en el navegador se nos pondrá de la siguiente manera:



Este efecto lo vamos a pasar a través del path y vamos a hacer que se verifique de manera automática para ello utilizaremos un hook que se llama useLocation vamos a realizar lo importación:

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1  import React from 'react'
2  import { Menu, Icon } from "semantic-ui-react"
3  import { Link, useLocation } from "react-router-dom"
4  import "./AdminMenu.scss"
5
```

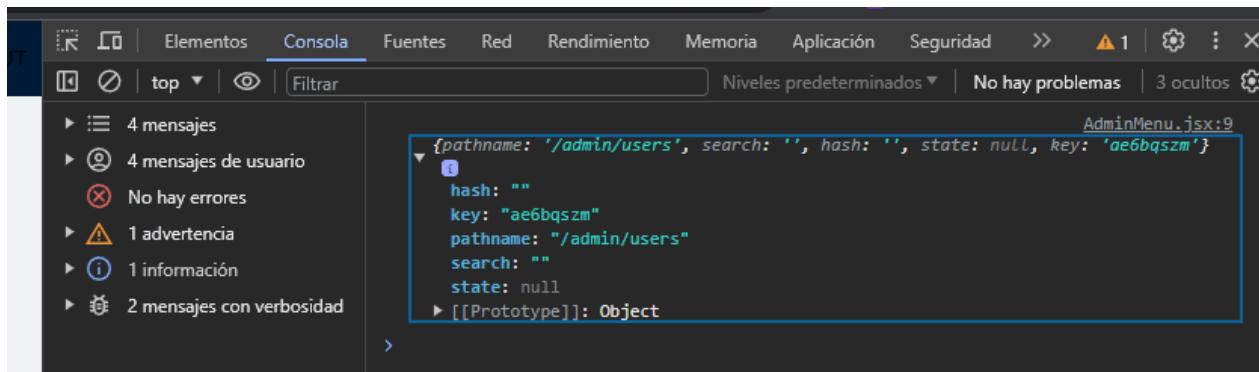
Y ahora vamos a hacer la implementación y verificamos a través del log que nos trae:

```

6  export function AdminMenu() {
7
8      const data = useLocation()
9      console.log(data)
10
11     return (

```

Vamos a observar la data que nos trae:



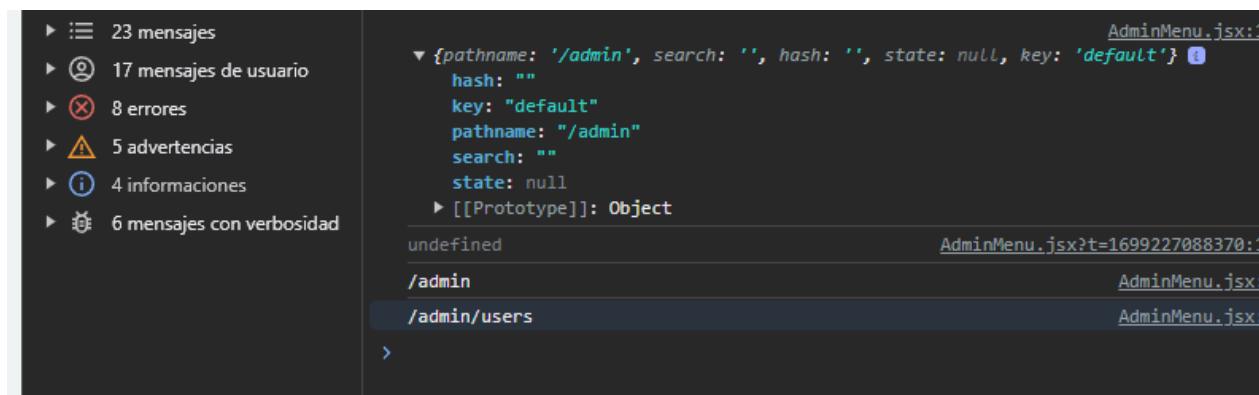
Ahora hacemos desestructuración de pathname:

```

5
6  export function AdminMenu() {
7
8      const { pathname } = useLocation()
9      console.log(pathname)
10
11     return (

```

Y verificamos en nuestro navegador al acceder a cualquier ruta nos trae el pathname:



Ahora vamos a indicarle en cada uno de los ítems a través de una función lo siguiente:



```

6  export function AdminMenu() {
7
8      const { pathname } = useLocation()
9
10     const isCurrentPath = (path) => {
11         if (path === pathname) return true
12         return false
13     }
14
15     return (
16         <Menu fluid vertical icon text className="admin-menu">

```

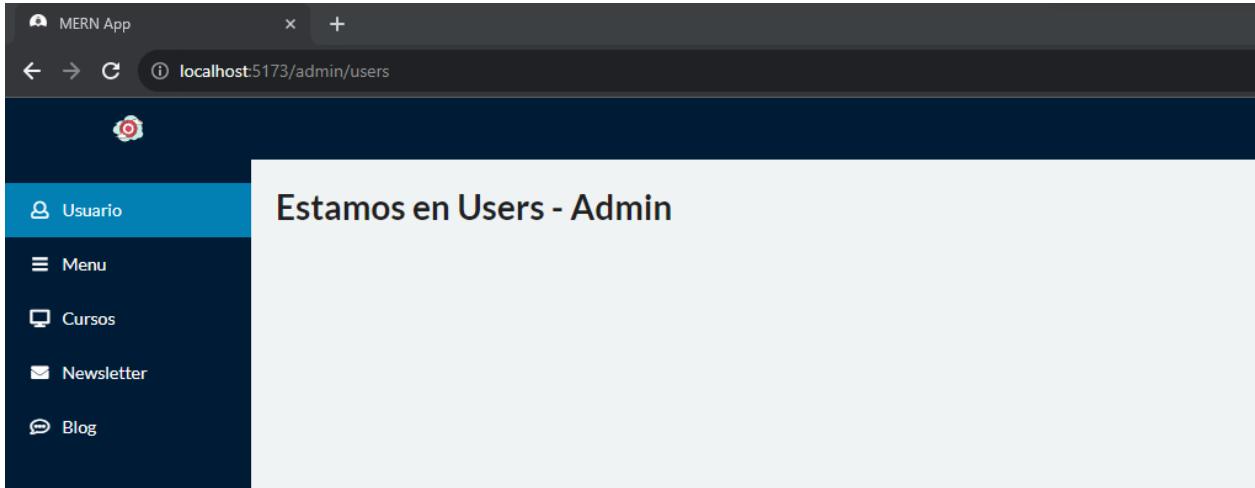
Le indicamos que si la ubicación en la que estamos es igual al pathname que nos lo devuelva true sino false, y para nuestros ítems lo vamos a dejar de la siguiente manera:

```

15    return (
16        <Menu fluid vertical icon text className="admin-menu">
17
18        <Menu.Item as={Link} to="/admin/users" active={isCurrentPath("/admin/users")}>
19            <Icon name="user outline" />
20            Usuario
21        </Menu.Item>
22
23        <Menu.Item as={Link} to="/admin/menu">

```

Si vamos al navegador vamos a observar lo siguiente:



Que se esta habilitando el path en el cual estamos ubicados, ahora hacemos lo mismo para los demás path con la propiedad active:

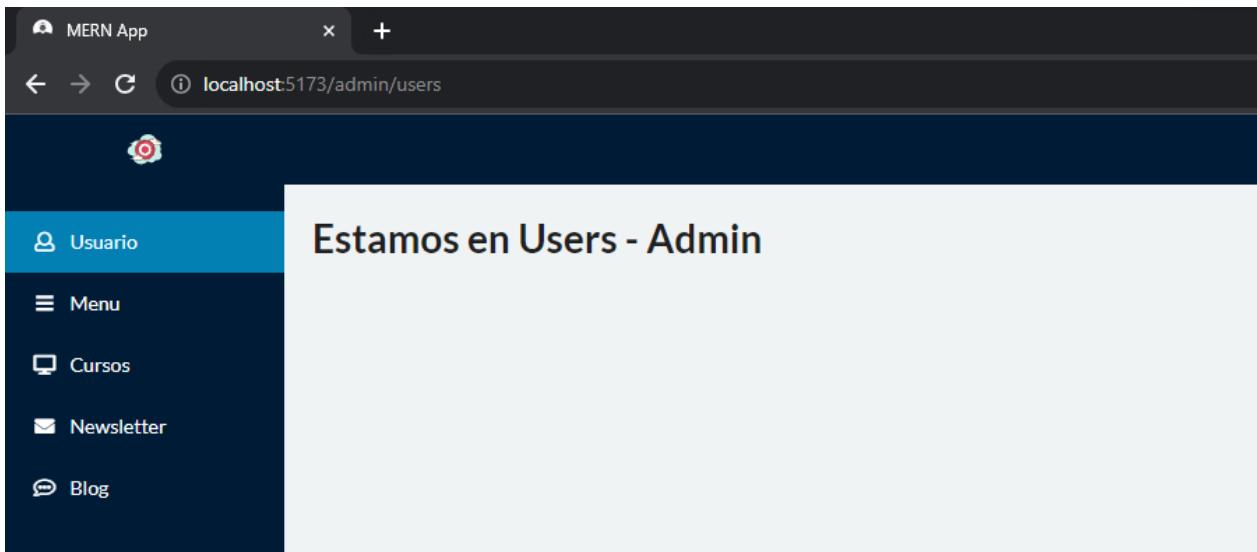


```

15  return (
16    <Menu fluid vertical icon text className="admin-menu">
17
18      <Menu.Item as={Link} to="/admin/users" active={isCurrentPath("/admin/users")}>
19        <Icon name="user outline" />
20        Usuario
21      </Menu.Item>
22
23      <Menu.Item as={Link} to="/admin/menu" active={isCurrentPath("/admin/menu")}>
24        <Icon name="bars" />
25        Menu
26      </Menu.Item>
27
28      <Menu.Item as={Link} to="/admin/courses" active={isCurrentPath("/admin/courses")}>
29        <Icon name="computer" />
30        Cursos
31      </Menu.Item>
32
33      <Menu.Item as={Link} to="/admin/newsletter" active={isCurrentPath("/admin/newsletter")}>
34        <Icon name="mail" />
35        Newsletter
36      </Menu.Item>
37
38      <Menu.Item as={Link} to="/admin/blog" active={isCurrentPath("/admin/blog")}>
39        <Icon name="comment alternate outline" />
40        Blog
41      </Menu.Item>
42
43    </Menu>
44
45  }

```

Verificamos en nuestro navegador que esté funcionando:



@hdtoledo

MERN App

localhost:5173/admin/menu

Usuario

Menu

Cursos

Newsletter

Blog

Menu

MERN App

localhost:5173/admin/courses

Usuario

Menu

Cursos

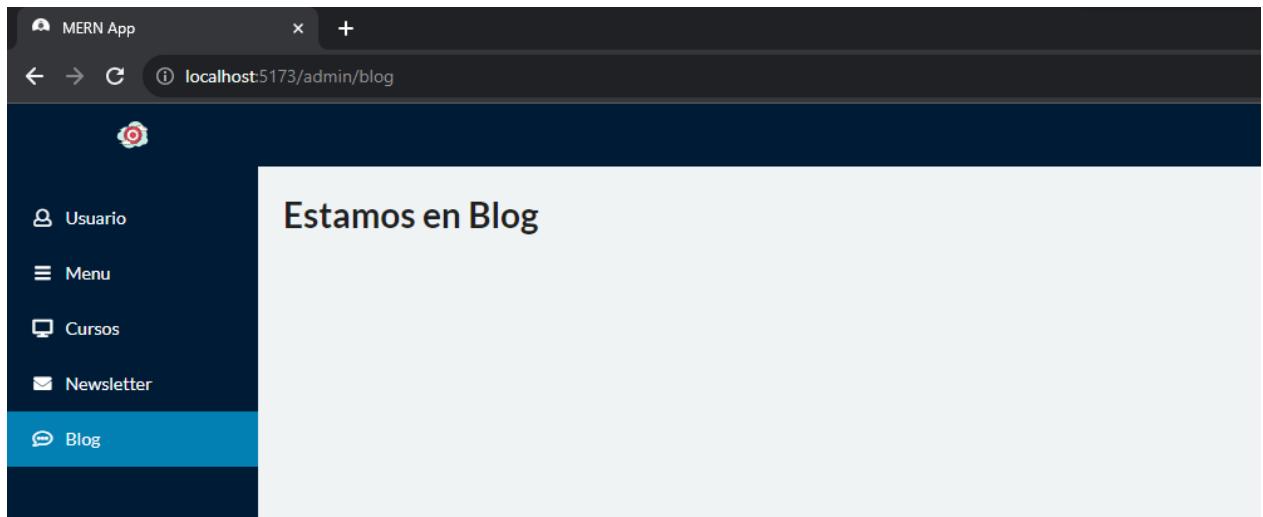
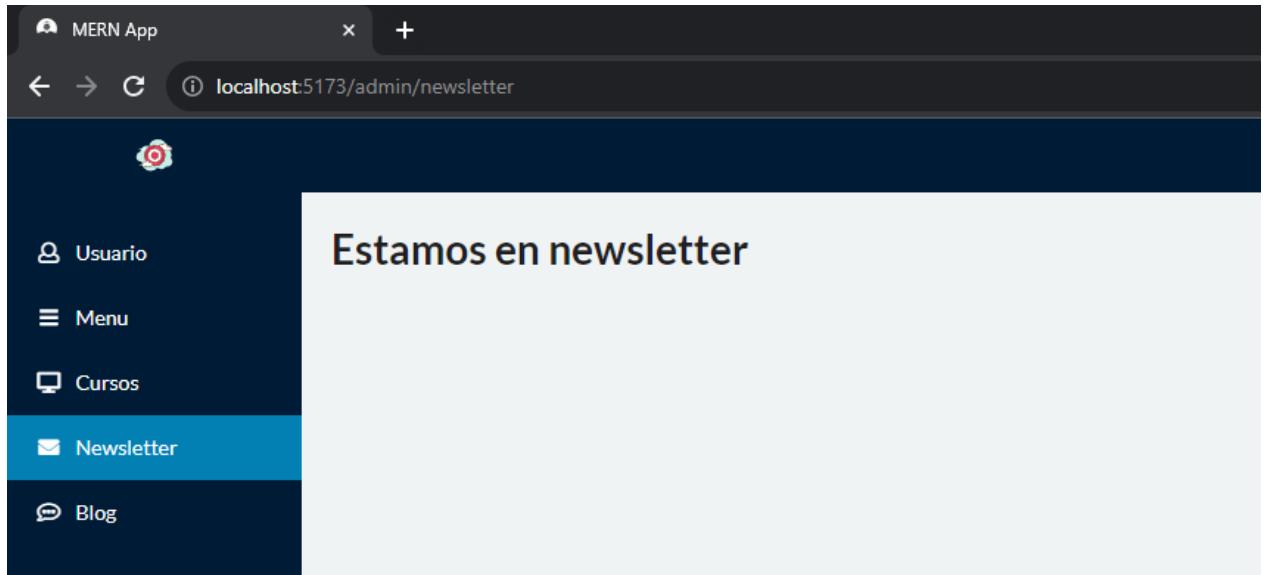
Newsletter

Blog

Estamos en Cursos



@hdtoledo



De esta manera vamos dejando más pulido nuestros menús.

PERMISOS PARA LOS MENUS

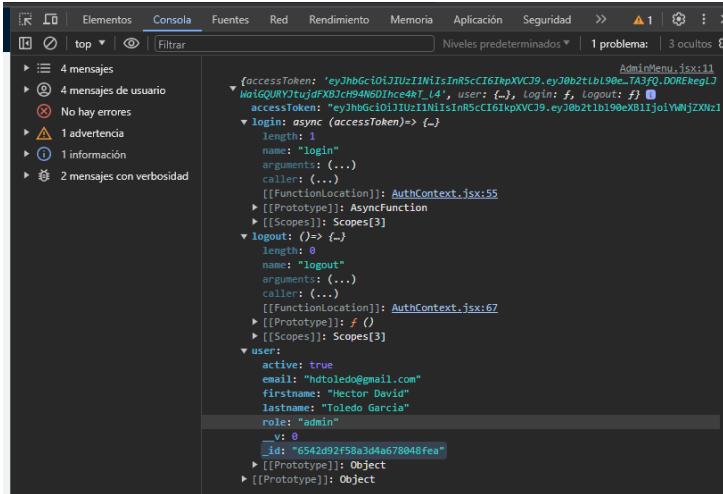
Ahora vamos a establecer los permisos de nuestro menú, ya que actualmente en nuestra aplicación tenemos dos roles, **admin** y **user**, cada uno puede acceder a cosas diferentes y no vamos a permitir que el usuario normal vea lo que el administrador, para ello vamos a nuestro **AdminMenu.jsx** y vamos a hacer la importación de **useAuth**:

```
src > components > Admin > AdminLayout > AdminMenu > AdminMenu.jsx > ...
1  import React from 'react'
2  import { Menu, Icon } from "semantic-ui-react"
3  import { Link, useLocation } from "react-router-dom"
4  import { useAuth } from "../../../../../hooks"
5  import "./AdminMenu.scss"
6
7  export function AdminMenu() {
```

Ahora revisemos que nos trae **useAuth()** a través de un log:

```
7  export function AdminMenu() {
8
9    const { pathname } = useLocation()
10
11  console.log(useAuth())
12
13  const isCurrentPath = (path) => {
14    if (path === pathname) return true
15  }
```

Vamos a nuestro navegador y observamos lo siguiente:



Dentro de **users** tenemos el role en el cual esta nuestro usuario, en este caso es admin, para ello vamos a realizar lo siguiente:

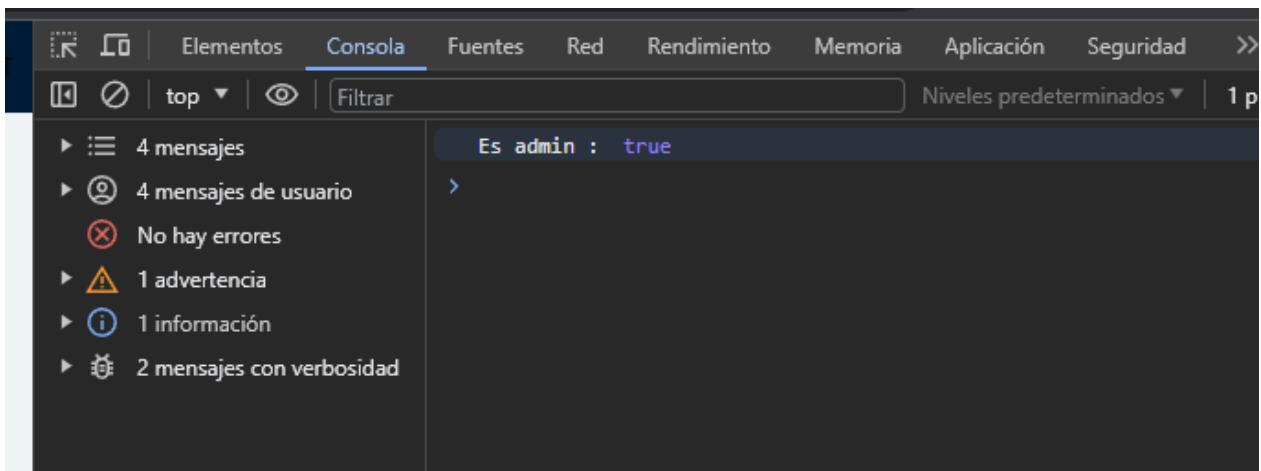


```

7  export function AdminMenu() {
8
9      const { pathname } = useLocation()
10
11     const { user: { role }, } = useAuth()
12     const isAdmin = role === "admin"
13     console.log("Es admin : ", isAdmin)
14
15
16     const isCurrentPath = (path) => {

```

Aca realizamos una pequeña verificación de si es admin y a través del log nos damos cuenta de lo siguiente:



Nos indica que, si es administrador, lo que vamos a realizar es a través de una condición si el usuario que inicia sesión es nuestro administrador que nos muestre los menús y sino pues cambiamos los menús que se ven, para ello realizamos lo siguiente:

```

18
19     return (
20         <Menu fluid vertical icon text className="admin-menu">
21             {isAdmin && (
22                 <>
23                     En este espacio va nuestros menus
24                 </>
25             )}
26
27             <Menu.Item as={Link} to="/admin/users" active={isCurrentPath("/admin/users")}>
28                 <Icon name="user outline" />
29             </Menu.Item>
30         </Menu>
31     )
32 
```

Con esta simple condicion que estamos aplicando vamos a dejar dentro los menus que queremos mostrar de la siguiente manera:

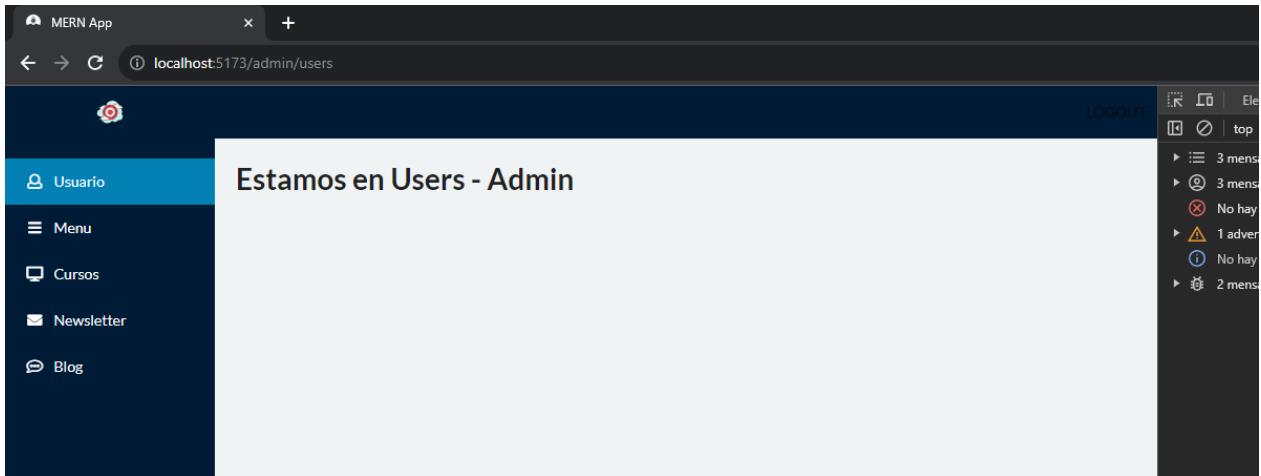


```

16
17     return (
18       <Menu fluid vertical icon text className="admin-menu">
19         {isAdmin && (
20           </>
21
22           <Menu.Item as={Link} to="/admin/users" active={isCurrentPath("/admin/users")}>
23             <Icon name="user outline" />
24             Usuario
25           </Menu.Item>
26
27           <Menu.Item as={Link} to="/admin/menu" active={isCurrentPath("/admin/menu")}>
28             <Icon name="bars" />
29             Menu
30           </Menu.Item>
31
32           <Menu.Item as={Link} to="/admin/courses" active={isCurrentPath("/admin/courses")}>
33             <Icon name="computer" />
34             Cursos
35           </Menu.Item>
36
37           <Menu.Item as={Link} to="/admin/newsletter" active={isCurrentPath("/admin/newsletter")}>
38             <Icon name="mail" />
39             Newsletter
40           </Menu.Item>
41
42           </>
43         )</>
44
45       <Menu.Item as={Link} to="/admin/blog" active={isCurrentPath("/admin/blog")}>
46         <Icon name="comment alternate outline" />
47         Blog
48       </Menu.Item>
49
50     </Menu>
51   )
52 }
53
54 }
55

```

Si lo verificamos en nuestro navegador vamos a ver que funciona:



Ahora para verificar lo que realizaremos será eliminar nuestros tokens para cerrar la sesión:



The screenshot shows the Network tab in the Chrome DevTools Application panel. The URL is `http://localhost:5173`. In the Headers section, there is a row for the key `refresh` with the value `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2tlb190eXBlIjoiYWNjZ`. A context menu is open over this row, with the option `Eliminar` (Delete) highlighted.

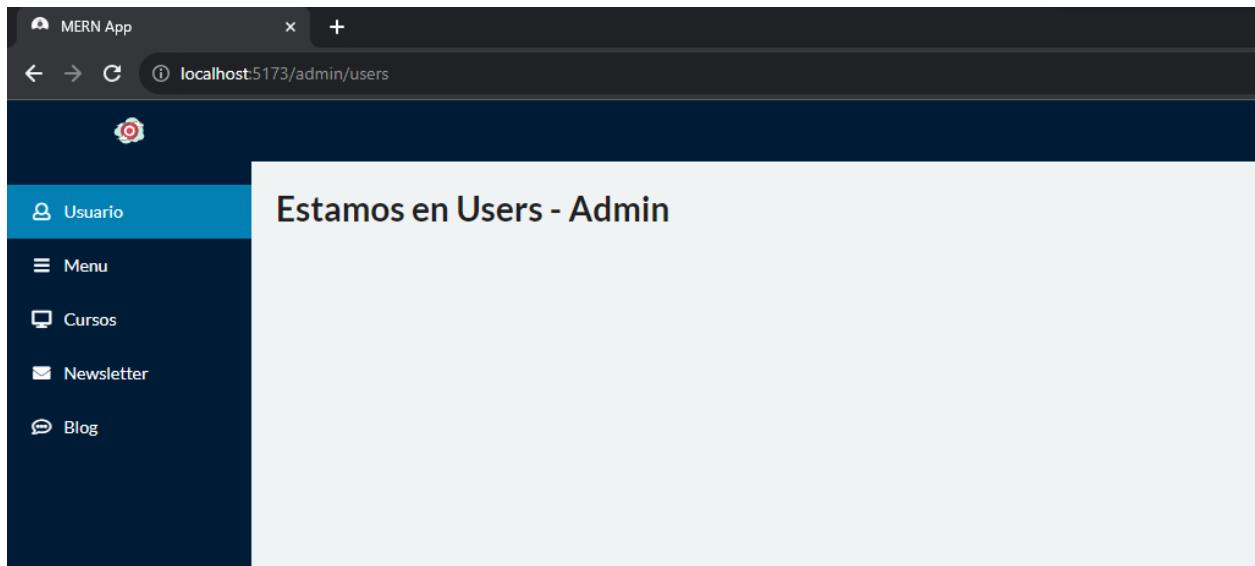
Y vamos a iniciar la sesión con usuario que solo tenga el rol de user:

The screenshot shows a browser window with the title "MERN App". The URL in the address bar is `localhost:5173/admin/users`. A modal dialog box is displayed, containing two tabs: "Entrar" (Login) and "Nuevo Usuario" (New User). The "Entrar" tab is active. Inside the form, the email field contains `ronald@gmail.com`. Below the email field is a password field with three dots visible. At the bottom of the form is a blue "Entrar" button.

Y al ingresar nos damos cuenta que solo tenemos habilitado este menú:

The screenshot shows a browser window with the title "MERN App". The URL in the address bar is `localhost:5173/admin/users`. The main content area displays the message "Estamos en Users - Admin". On the left side, there is a sidebar with a "Blog" icon. The overall theme is dark.

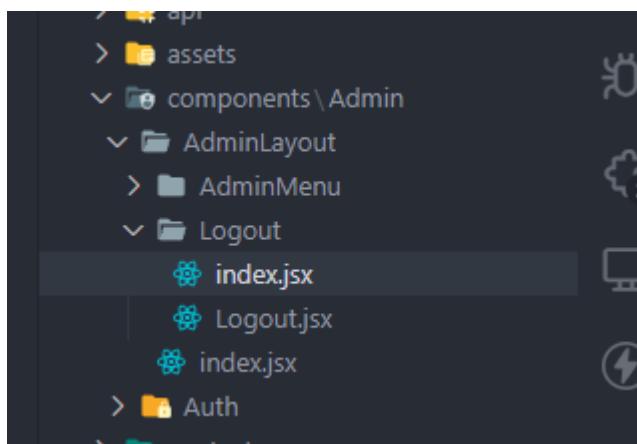
Ahora si vuelvo a cerrar la sesión e ingreso de nuevo con la cuenta de admin:



Tendré habilitado los menús del administrador.

LOGOUT

Ahora vamos a realizar nuestro componente para cerrar la sesión del usuario, para ello nos vamos a crear la carpeta **Logout** con los archivos **index.jsx** y **Logout.jsx** dentro de la ruta **/components/admin/adminlayout/**



Dentro de nuestro **/Logout/index.jsx** dejamos la exportación:

```
src > components > Admin > AdminLayout > Logout > index.jsx
1   export * from "./Logout"
```

Dentro de nuestro `/Logout/Logout.jsx`:

```
src > components > Admin > AdminLayout > Logout > Logout.jsx > Logout
1 import React from 'react'
2
3 export function Logout() {
4   return (
5     <div>Logout</div>
6   )
7 }
8
```

Y en nuestro `/AdminLayout/index.jsx`:

```
src > components > Admin > AdminLayout > index.jsx
1 export * from "./AdminMenu"
2 export * from "./Logout"
```

Ahora vamos a modificar nuestro `Logout.jsx`:

```
src > components > Admin > AdminLayout > Logout > Logout.jsx > Logout
1 import React from 'react'
2 import { Button, Icon } from "semantic-ui-react"
3
4 export function Logout() {
5   return [
6     <Button icon>
7       | <Icon name="power off"/> Cerrar Sesión
8     </Button>
9   ]
10 }
11
```

Ahora vamos a importarlo en nuestro `/layouts/AdminLayout/AdminLayout.jsx`

```
src > layouts > AdminLayout > AdminLayout.jsx > ...
1 import React from "react"
2 import { iconLogo } from "../../assets"
3 import { AdminMenu, Logout } from "../../components/Admin/AdminLayout"
4 import "./AdminLayout.scss";
5
6 export function AdminLayout(props) {
```

Y vamos a hacer la implementación de la siguiente manera:

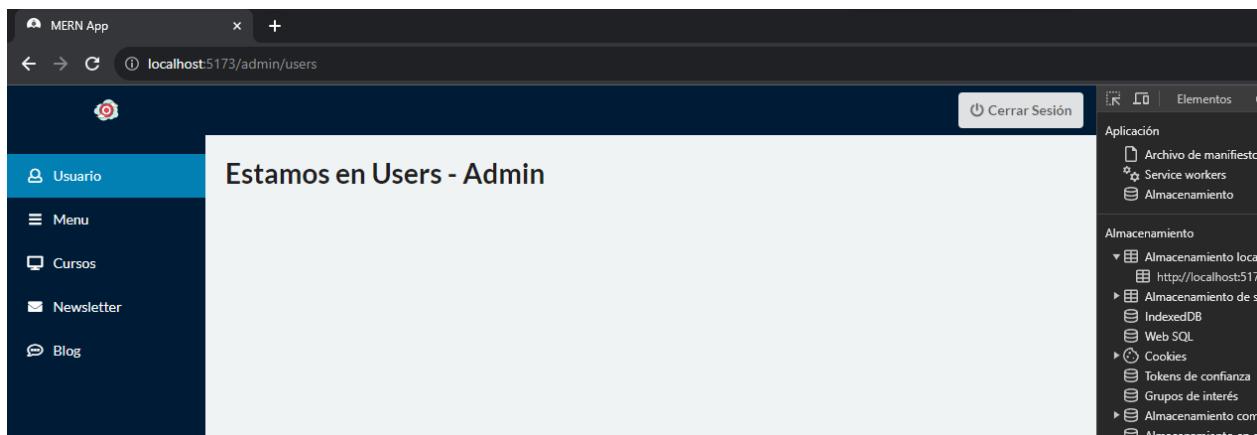


```

9   return (
10    <div className="admin-layout">
11      <div className="admin-layout__left">
12          <img src={iconLogo} className="logo" />
13          <AdminMenu />
14      </div>
15      <div className="admin-layout__right">
16          <div className="admin-layout__right-header">
17              | <Logout />
18          </div>
19          <div className="admin-layout__right-content">{children}</div>
20      </div>
21  </div>
22 );
23 }
24

```

Y si vamos a nuestro navegador vamos a observar lo siguiente:



Ahora vamos a estilizar y a volverlo funcional nuestro componente de la siguiente manera **Logout.jsx**:

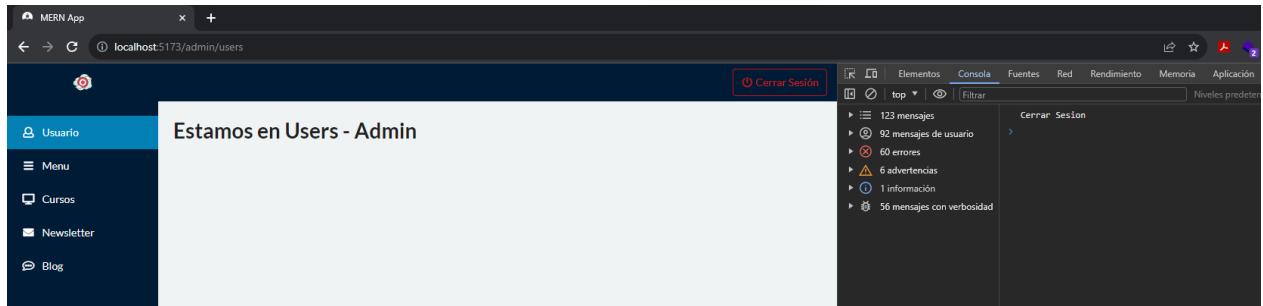
```

src > components > Admin > AdminLayout > Logout > Logout.jsx > Logout
1  import React from 'react'
2  import { Button, Icon } from "semantic-ui-react"
3
4  export function Logout() {
5
6      const onLogout = () => {
7          console.log("Cerrar Sesión")
8      }
9
10     return (
11         <Button icon basic color="red" onClick={onLogout}>
12             | <Icon name="power off"/> Cerrar Sesión
13         </Button>
14     )
15 }
16

```

Ahora si vamos a nuestro navegador vamos a obtener lo siguiente:





Si le damos clic a nuestro botón este nos enviará el mensaje en consola, ahora para cerrar la sesión de verdad vamos a importar lo siguiente y hacemos la ejecución:

```
src > components > Admin > AdminLayout > Logout > ✨ Logout.jsx > ...
1  import React from 'react'
2  import { Button, Icon } from "semantic-ui-react"
3  import { useNavigate } from "react-router-dom"
4  import { useAuth } from "../../../../../hooks"
5  |
6  export function Logout() {
7
8      export function Logout() {
9          const { logout } = useAuth()
10         const { navigate } = useNavigate()
11
12         const onLogout = () => {
13             logout()
14             navigate("/admin")
15         }
16
17         return (
18             <Button icon basic color="red" onClick={onLogout}>
19                 <Icon name="power off"/> Cerrar Sesión
20             </Button>
21         )
22     }
```

Vamos a nuestro navegador a comprobar que realmente está funcionando:



MERN App

localhost:5173/admin/users

Cerrar Sesión

Consola

Elementos Fuentes

top Filtrar

6 mensajes
3 mensajes de usuario
2 errores
1 advertencia
No hay información
3 mensajes con verbosidad

Al pincharlo se nos debe cerrar la sesión:

MERN App

localhost:5173/admin/users

Entrar Nuevo Usuario

Correo Electrónico

Contraseña

Entrar

Nuevo Usuario

4
3
N
1
N
3

Si se recarga la pagina queda en el login, así que ya con esto tenemos nuestro cierre de sesión.

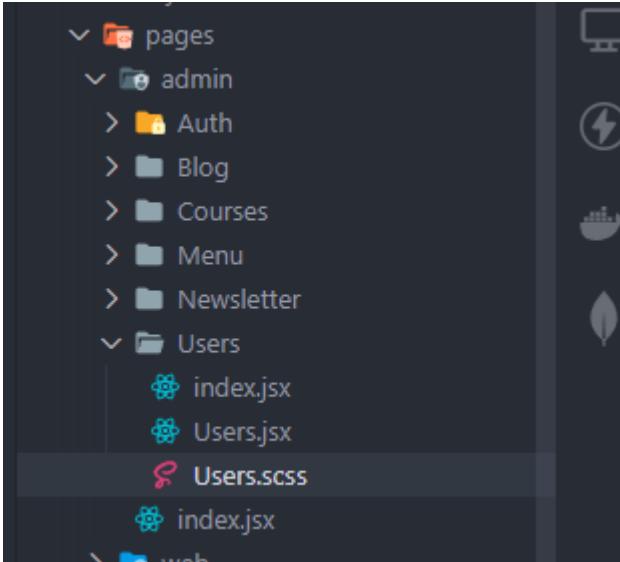


ESTRUCTURA DE LA PAGINA - USERS

Vamos a empezar a montar la estructura de los usuarios para ello vamos a nuestro [/pages/admin/users/Users.jsx](#) y vamos a realizar las siguientes importaciones:

```
src > pages > admin > Users >  Users.jsx > ...
1  import React, { useState } from "react"
2  import { Tab, Button } from "semantic-ui-react"
3
4
```

Ahora vamos a crear dentro de [/pages/admin/users](#) nuestro archivo **Users.scss**



Y hacemos la importación de nuestro archivo:

```
src > pages > admin > Users >  Users.jsx > ...
1  import React, { useState } from "react"
2  import { Tab, Button } from "semantic-ui-react"
3  import "./Users.scss"
4
```

Y empezamos con la estructuración de nuestro **Users.jsx**:



```

0
7  export function Users() {
8
9    const panes = [
10      {
11        menuItem: "Usuarios Activos",
12        render: () => (
13          <Tab.Pane attached={false}>
14            <h2>Usuarios Activos</h2>
15          </Tab.Pane>
16        )
17      },
18      {
19        menuItem: "Usuarios Inactivos",
20        render: () => (
21          <Tab.Pane attached={false}>
22            <h2>Usuarios Inactivos</h2>
23          </Tab.Pane>
24        )
25      },
26    ],
27  ]
28
29  return (
30    <div className="users-page">
31      <Button className="users-page__add" primary onClick={() => console.log("Abrir formulario")}>
32        Nuevo usuario
33      </Button>
34      <Tab menu={{ secondary: true }} panes={panes} />
35    </div>
36  )
37

```

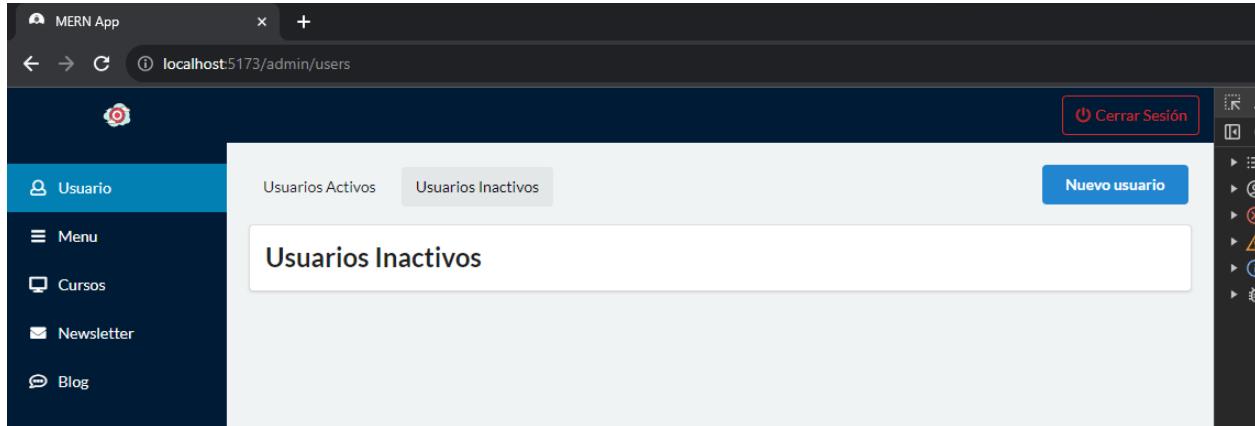
De esta manera estamos organizando nuestro menú de usuarios en donde vamos a observar los usuarios activos e inactivos, mantenemos la estructura similar a la de nuestro login y register utilizando los Tabs de semantic UI, si nos dirigimos al navegador observaremos lo siguiente:

Al dar clic al botón obtendremos en consola su funcionalidad, de igual manera al cambiar entre activos o inactivos estaremos viendo cómo cambia:

Ahora vamos a estilizar un poco nuestro **Users.scss** para ubicar nuestro botón al lado derecho:

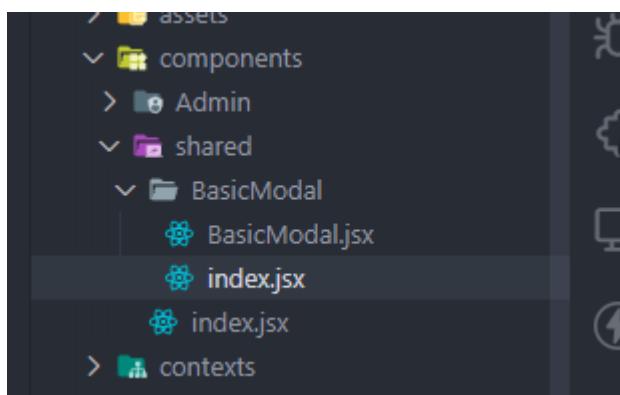
```
src > pages > admin > Users > Users.scss > .users-page
1  .users-page {
2    &__add {
3      position: absolute;
4      right: 0;
5      top: 0;
6
7      > .ui.button {
8        margin: 0;
9      }
10 }
11 }
```

Y si vamos al navegador observamos lo siguiente:



MODAL PARA MOSTRAR FORMULARIOS

Vamos a crear un modal personalizado a través de un componente, este modal servirá para cuando el usuario le de clic a nuestro botón nuevo usuario, para ello vamos a crear dentro de **/components/shared** con su respectivo **index** y dentro creamos la carpeta **BasicModal** con los archivos **BasicModal.jsx** **index.jsx**



Hacemos las exportaciones que normalmente realizamos en nuestros index:

```
src > components > shared > index.jsx
1   export * from "./BasicModal"
```

```
src > components > shared > BasicModal > index.jsx
1   export * from "./BasicModal"
```

Y para nuestro **BasicModal.jsx** hacemos nuestro componente:

```
src > components > shared > BasicModal > BasicModal.jsx > BasicModal
1   import React from 'react'
2   import { Modal } from "semantic-ui-react"
3
4   export function BasicModal(props) {
5
6     const { show, close, title, size, children } = props
7
8     return (
9       <Modal closeIcon open={show} onClose={close} size={size}>
10      {title && <Modal.Header>{title}</Modal.Header>}
11      <Modal.Content>{children}</Modal.Content>
12    </Modal>
13  )
14}
15
16 BasicModal.defaultProps = {
17   size: "tiny"
18 }
19
```

En donde llamamos Modal de semantic UI, aplicamos unas props a nuestro modal y le damos unas características. Ahora nos vamos a **Users.jsx** y realizamos la importación:

```
src > pages > admin > Users > Users.jsx > ...
1   import React, { useState } from "react"
2   import { Tab, Button } from "semantic-ui-react"
3   import { BasicModal } from "../../components/shared"
4   import "./Users.scss"
5
6
```

Ahora vamos a crear un estado con useState:

```
7
8   export function Users() {
9
10   const [showModal, setShowModal] = useState(false)
11
12   const panes = [
13   {
14     menuItem: "Usuarios Activos",
```

Vamos a crear la función de que nos verifique si esta abierto o no nuestro modal de la siguiente manera:

```
7
8  export function Users() {
9
10    const [showModal, setShowModal] = useState(false)
11
12    const [onOpenCloseModal] = () => setShowModal((prevState) => !prevState)
13
14
15    const panes = [
16      {
```

Y ahora hacemos la implementación de nuestro BasicModal:

```
34  return (
35    <>
36
37    <div className="users-page">
38      <Button className="users-page__add" primary onClick={() => console.log("Abrir formulario")}>
39        Nuevo usuario
40      </Button>
41      <Tab menu={{ secondary: true }} panes={panes} />
42    </div>
43
44    <BasicModal show={showModal} close={onOpenCloseModal} title="Crear nuevo usuario">
45      <h2>Formulario para crear usuarios</h2>
46    </BasicModal>
47
48  </>
49
50  }
51 }
```

Ahora por último al presionar nuestro botón debemos decirle que la función que se ejecutará será la de **onOpenCloseModal**:

```
37    <div className="users-page">
38      <Button className="users-page__add" primary onClick={onOpenCloseModal}>
39        Nuevo usuario
40      </Button>
41      <Tab menu={{ secondary: true }} panes={panes} />
42    </div>
43
```

Ahora vamos a nuestro navegador y observaremos:



@hdtoledo

The screenshot shows a web application interface for managing users. At the top, the browser title bar reads "MERN App" and the address bar shows "localhost:5173/admin/users". The main content area has a dark header with "Usuarios Activos" and "Usuarios Inactivos" tabs, and a "Nuevo usuario" button. A sidebar on the left includes links for "Usuario", "Menu", "Cursos", "Newsletter", and "Blog". A modal window titled "Crear nuevo usuario" is displayed in the center, containing the text "Formulario para crear usuarios". The background shows a list of active users.

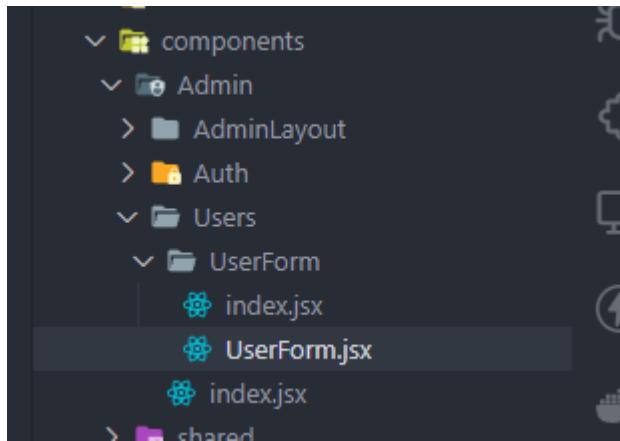
Que al presionar nuestro botón se ejecutara el modal y si damos clic en cerrar este se cerrara incluso al dar clic por fuera del modal.



@hdtoledo

FORMULARIO PARA CREAR USUARIOS

Ahora vamos a crear nuestro formulario para crear usuarios para ello nos ubicamos en **/components/admin/** y creamos la carpeta **Users** hacemos nuestro **index.jsx** dentro hacemos **UserForm** con su **index** y **UserForm.jsx**



Hacemos las exportaciones dentro de nuestros index.jsx

```
src > components > Admin > Users > index.jsx
1   export * from "./UserForm"
```

```
src > components > Admin > Users > UserForm > index.jsx
1   export * from "./UserForm"
```

Y para nuestro **UserForm.jsx** dejamos lo siguiente:

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
1   import React from 'react'
2
3   export function UserForm(props) {
4
5     const { close, onReload, User } = props
6
7     return (
8       <div>
9         <h2>UserForm</h2>
10        </div>
11      )
12    }
13  |
```

Vamos a hacer la importación de este componente en **Users.jsx**



```

src > pages > admin > Users > Users.jsx > ...
1 import React, { useState } from "react"
2 import { Tab, Button } from "semantic-ui-react"
3 import { BasicModal } from "../../components/shared"
4 import { UserForm } from "../../components/Admin/Users"
5 import "./Users.scss"
6

```

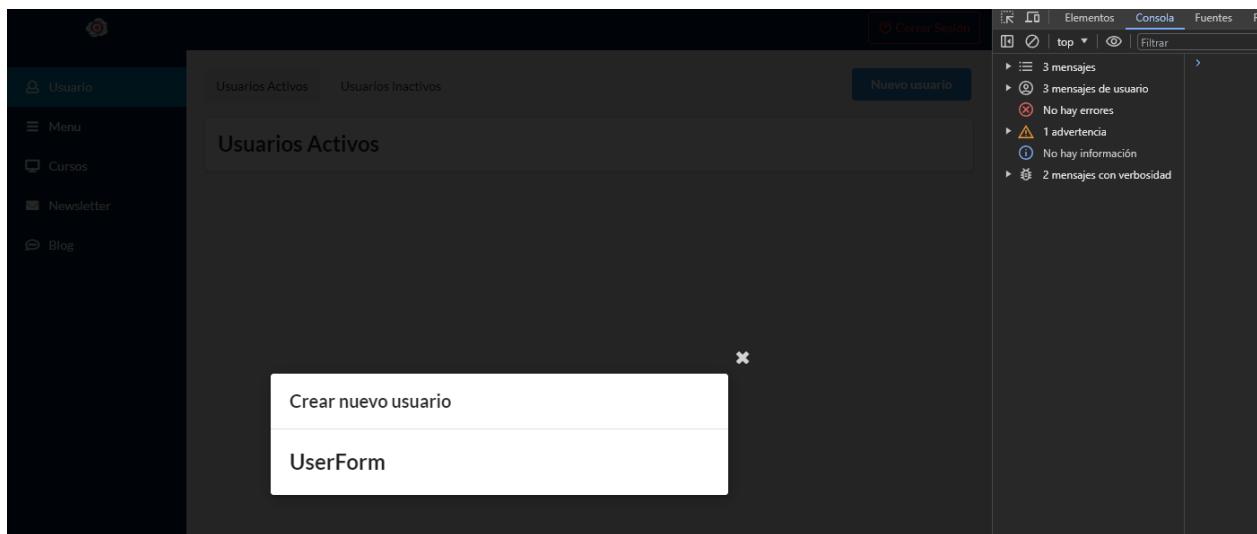
Y lo implementamos de la siguiente manera:

```

44
45     <BasicModal show={showModal} close={onOpenCloseModal} title="Crear nuevo usuario">
46     | <UserForm />
47     </BasicModal>
48
49   </>
50
51 }

```

Ahora si vamos a revisar en nuestro navegador en el botón de nuevo usuario observaremos nuestro componente base:



Ahora le pasamos una de las props que sería close:

```

44
45     <BasicModal show={showModal} close={onOpenCloseModal} title="Crear nuevo usuario">
46     | <UserForm close={onOpenCloseModal}/>
47     </BasicModal>
48

```

Ahora vamos a empezar a construir nuestro formulario en **UserForm.jsx** hacemos la importación de semantic UI con Form e Image:

```

src > components > Admin > Users > UserForm > UserForm.jsx > ...
1 import React from 'react'
2 import { Form, Image } from "semantic-ui-react"
3
4 export function UserForm(props) {

```

Y ahora empezamos a estructurar nuestro formulario:

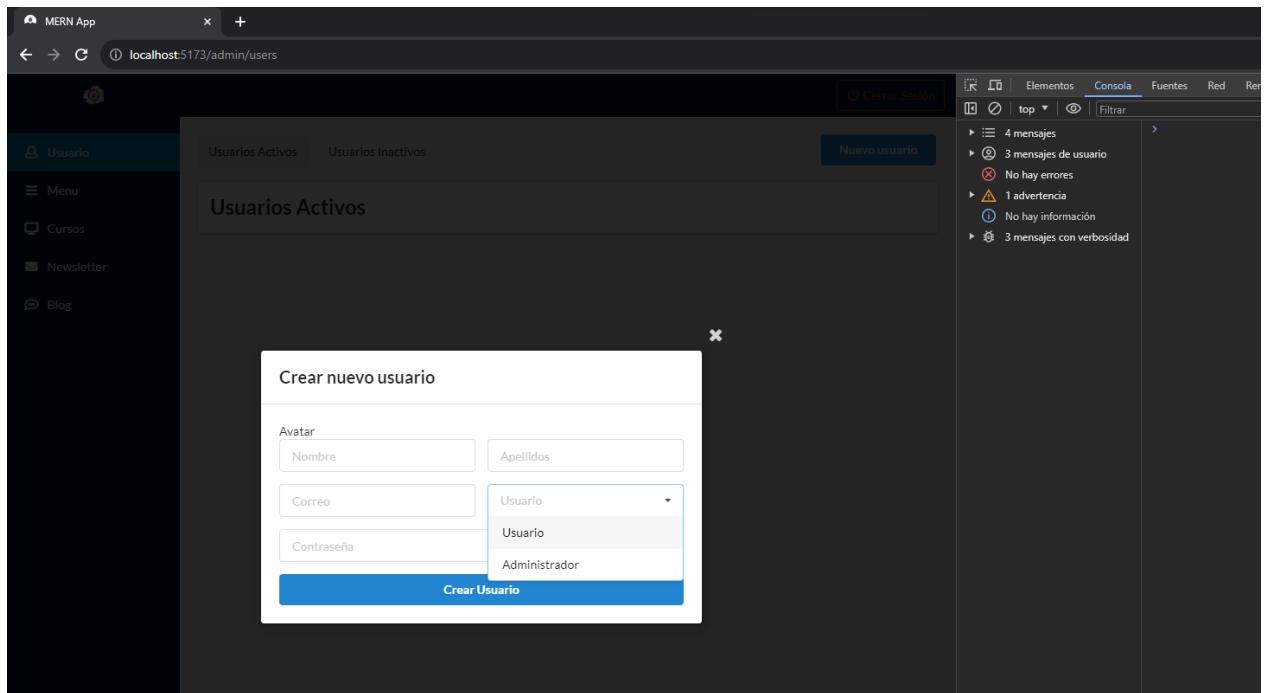
```
src > components > Admin > Users > UserForm > UserForm.jsx ...
1 import React from 'react'
2 import { Form, Image } from "semantic-ui-react"
3
4 export function UserForm(props) {
5
6   const { close, onReload, user } = props
7
8   return (
9     <Form className="user-form">
10      <div className="user-form__avatar">
11        <span>Avatar</span>
12      </div>
13
14      <Form.Group widths="equal">
15        <Form.Input name="firstsname" placeholder="Nombre"/>
16        <Form.Input name="lastname" placeholder="Apellidos"/>
17      </Form.Group>
18      <Form.Group widths="equal">
19        <Form.Input name="email" placeholder="Correo"/>
20        <Form.Dropdown placeholder="Selecciona un rol" options={roleOptions} selection/>
21      </Form.Group>
22      <Form.Input type="password" name="password" placeholder="Contraseña"/>
23      <Form.Button type="submit" primary fluid>
24        { user ? "Actualizar Usuario" : "Crear Usuario" }
25      </Form.Button>
26    </Form>
27  )
28}
29
30 const roleOptions = [
31   {
32     key: "user",
33     text: "Usuario",
34     value: "user"
35   },
36   {
37     key: "admin",
38     text: "Administrador",
39     value: "admin"
40   },
41 ]
```

Dejamos un formulario base con los datos de nuestro usuario recordemos que este mismo lo vamos a utilizar para editar y crear usuarios, realizamos una condicional ternaria para saber si trae o no el usuario y así activar el nombre del botón y le pasamos a través de roleOptions las opciones que queremos para guardar en el select si es admin o user.

Al revisar nuestro navegador observamos:



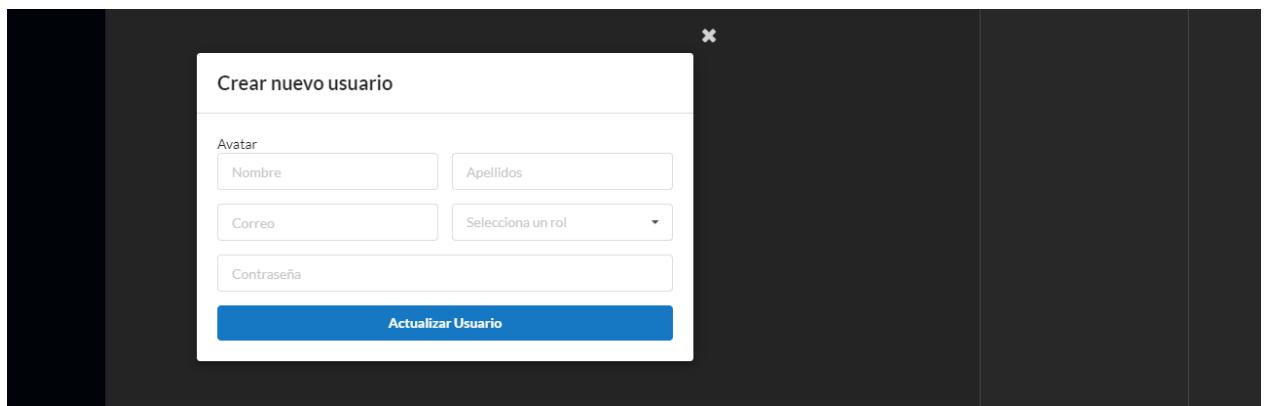
@hdtoledo



Ahora si queremos pasarle un dato de un usuario vamos a colocarlo en **Users.jsx**:

```
src > pages > admin > Users > 🌐 Users.jsx > 📁 Users
43   </div>
44
45   <BasicModal show={showModal} close={onOpenCloseModal} title="Crear nuevo usuario">
46     <UserForm close={onOpenCloseModal} user={{name: "HD"}} />
47   </BasicModal>
48
49   </>
50
51 }
52 }
```

Y si vamos al navegador observamos que dice actualizar:

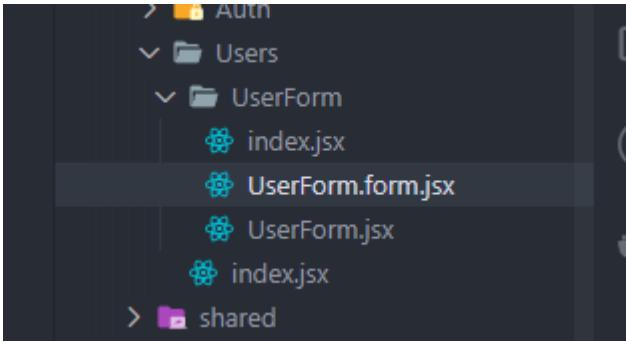


Eliminamos lo que acabamos de colocar y dejamos allí ya que actualmente está funcionando.



CONTROLANDO Y VALIDANDO EL USUARIO

Ahora vamos a crear nuestro archivo de control para validar el usuario, creamos **UserForm.form.jsx**



Y dentro vamos a colocar lo siguiente:

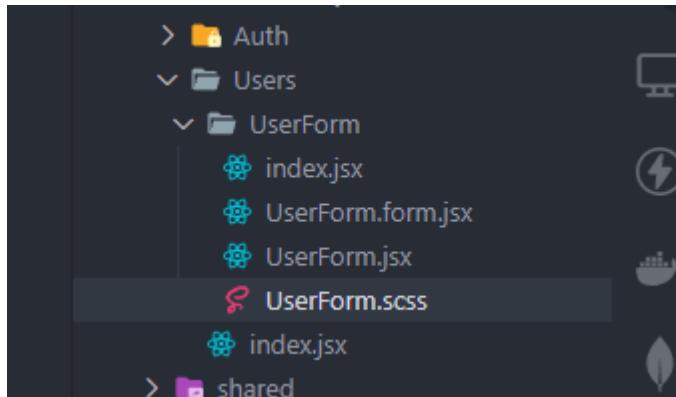
```
src > components > Admin > Users > UserForm > UserForm.form.jsx > validationSchema
1 import * as Yup from "yup"
2
3 export function initialValues() {
4     return {
5         avatar: "",
6         fileAvatar: null,
7         firtsname: "",
8         lastname: "",
9         email: "",
10        role: "",
11        password: ""
12    }
13}
14
15 export function validationSchema() {
16     return Yup.object({
17         firtsname: Yup.string().required(true),
18         lastname: Yup.string().required(true),
19         email: Yup.string().email(true).required(true),
20         role: Yup.string().required(true),
21         password: Yup.string().required(true),
22     })
23}
```

Este archivo es muy similar al de nuestro control y validación para register utilizando Yup. Ahora vamos a realizar la importación dentro de **UserForm.jsx**:

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
1 import React from 'react'
2 import { Form, Image } from "semantic-ui-react"
3 import { useFormik } from "formik"
4 import { initialValues, validationSchema } from "./UserForm.form"
```

Vamos a crear tambien nuestro archivo de **UserForm.scss**:





El cual también importaremos:

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
1 import React from 'react'
2 import { Form, Image } from "semantic-ui-react"
3 import { useFormik } from "formik"
4 import { initialValues, validationSchema } from "./UserForm.form"
5 import "./UserForm.scss"
6
7
```

Ahora dejaremos la siguiente estructura dentro:

```
6 export function UserForm(props) {
7
8     const { close, onReload, user } = props
9
10    const formik = useFormik({
11        initialValues: initialValues(),
12        validationSchema: validationSchema(),
13        validateOnChange: false,
14        onSubmit: async (formValue) => {
15            try {
16                console.log(formValue)
17            } catch (error) {
18                console.error(error)
19            }
20        }
21    })
22
23    return (
24        <Form className="user-form">
25            <!-- Content -->
```

Prácticamente hacemos las mismas validaciones que con register, cargamos nuestros valores iniciales, hacemos validación y vamos a dejar unos logs para revisar que nos trae, ahora vamos a implementarlo en nuestro formulario:



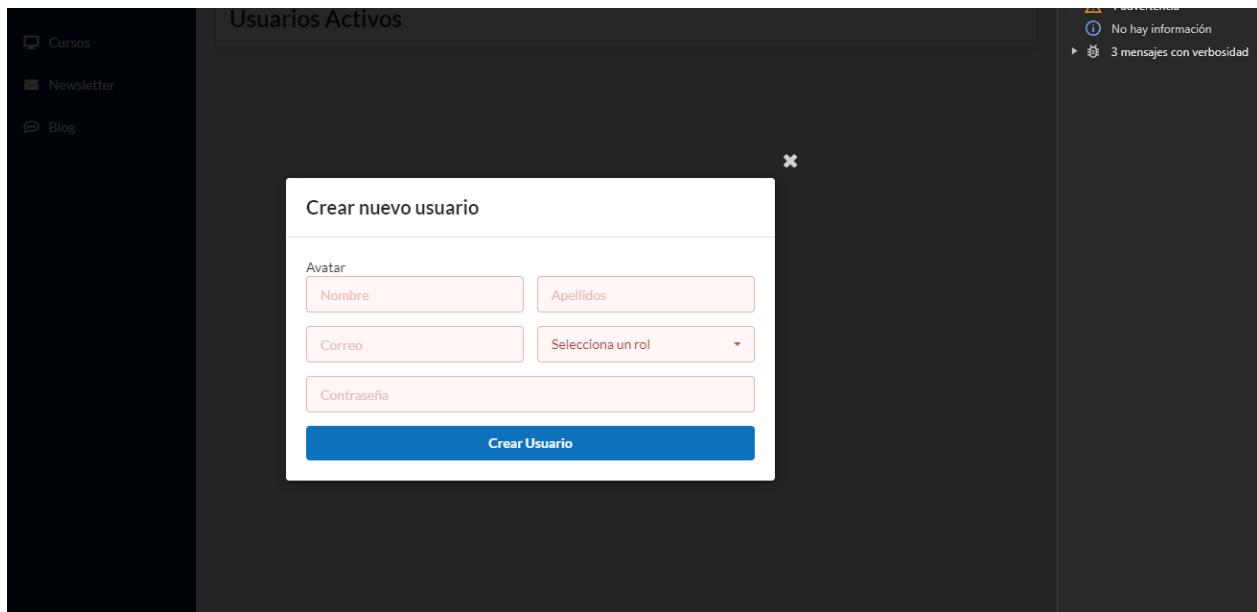
@hdtoledo

```

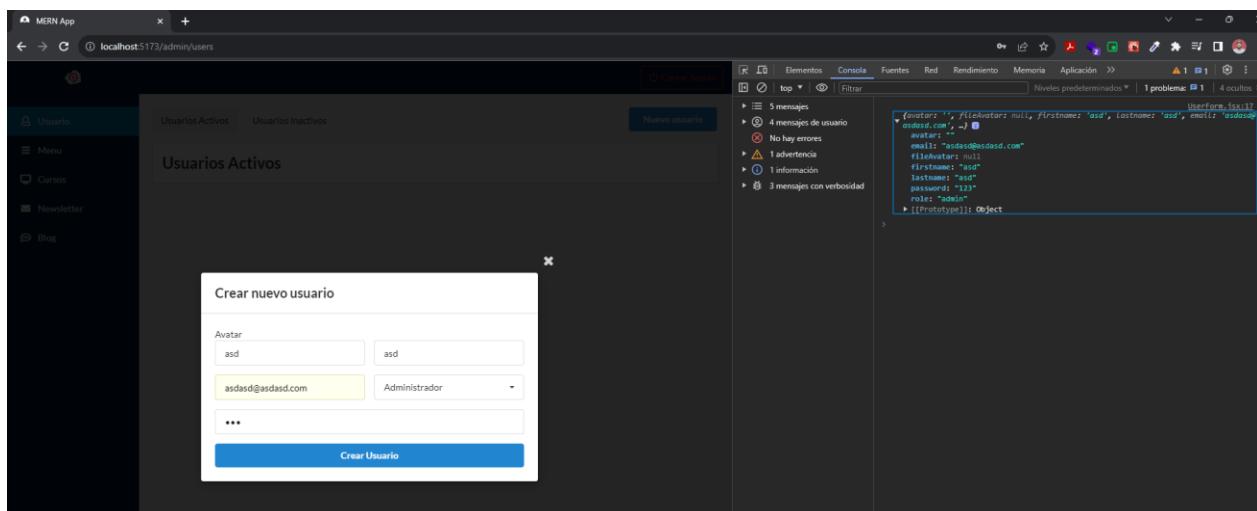
24
25    return (
26      <Form className="user-form" onSubmit={formik.handleSubmit}>
27        <div className="user-form__avatar">
28          <span>Avatar</span>
29        </div>
30
31        <Form.Group widths="equal">
32          <Form.Input name="firstname" placeholder="Nombre" onChange={formik.handleChange} value={formik.values.firstname} error={formik.errors.firstname}/>
33          <Form.Input name="lastname" placeholder="Apellidos" onChange={formik.handleChange} value={formik.values.lastname} error={formik.errors.lastname}/>
34        </Form.Group>
35        <Form.Group widths="equal">
36          <Form.Input name="email" placeholder="Correo" onChange={formik.handleChange} value={formik.values.email} error={formik.errors.email}/>
37          <Form.Dropdown placeholder="Selecciona un rol" options={roleOptions} selection onChange={(_, data) => formik.setFieldValue("role", data.value)} value={formik.values.role} error={formik.errors.role}/>
38        </Form.Group>
39        <Form.Input type="password" name="password" placeholder="Contraseña" onChange={formik.handleChange} value={formik.values.password} error={formik.errors.password}/>
40        <Form.Button type="submit" primary fluid loading={formik.isSubmitting}>
41          { user ? "Actualizar Usuario" : "Crear Usuario" }
42        </Form.Button>
43      </Form>
44    )

```

Establecemos cada uno de nuestro input con las validaciones de formik. Ahora vamos a nuestro navegador para revisar si se realizan las validaciones:



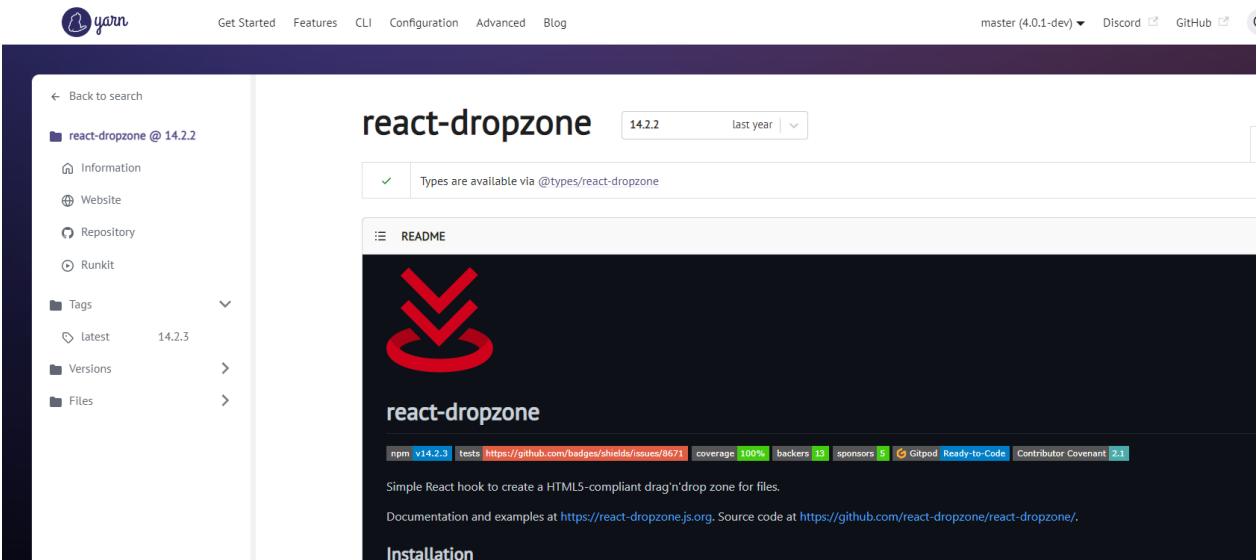
Ahora si colocamos todos los campos correctamente y presionamos en crear usuario debemos obtener en el log nuestros datos:



@hdtoledo

CONTROLANDO LA IMAGEN DEL USUARIO

Ahora vamos a controlar y gestionar el avatar del usuario para ello vamos a utilizar una dependencia que se llama [react-dropzone](#):



Así que vamos a instalarla utilizando el comando [yarn add react-dropzone@14.2.2](#)



Levantamos nuevamente el servidor y vamos a realizar la importación de nuestra dependencia Dropzone y useCallback:

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
  1 import React, { useCallback } from 'react'
  2 import { Form, Image } from "semantic-ui-react"
  3 import { useFormik } from "formik"
  4 import { useDropzone } from "react-dropzone"
  5 import { initialValues, validationSchema } from "./UserForm.form"
  6 import "./UserForm.scss"
  7
```

Ahora vamos a empezar a implementar nuestro dropzone:



```

src > components > Admin > Users > UserForm > UserForm.jsx > UserForm
22     }
23   )
24
25   const onDrop = useCallback((acceptedFiles) => {
26     console.log(acceptedFiles)
27   )
28
29   const { getRootProps, getInputProps } = useDropzone({
30     accept: "image/jpeg, image/jpg, image/png",
31     onDrop,
32   })
33
34   return (
35     <Form className="user-form" onSubmit={formik.handleSubmit}>

```

Utilizando useCallback vamos a revisar que es lo que nos trae en un log, y utilizando getRootProps – getInputProps vamos a indicarle el tipo de imágenes que vamos a recibir, y ahora vamos a modificar nuestro div para recibir la imagen:

```

src > components > Admin > Users > UserForm > UserForm.jsx > UserForm
22   ...
23
24   return (
25     <Form className="user-form" onSubmit={formik.handleSubmit}>
26
27       <div className="user-form__avatar" {...getRootProps()}>
28         <input {...getInputProps()} />
29         <Image avatar size="small" src={null} />
30       </div>
31
32     <Form.Group widths="equal">

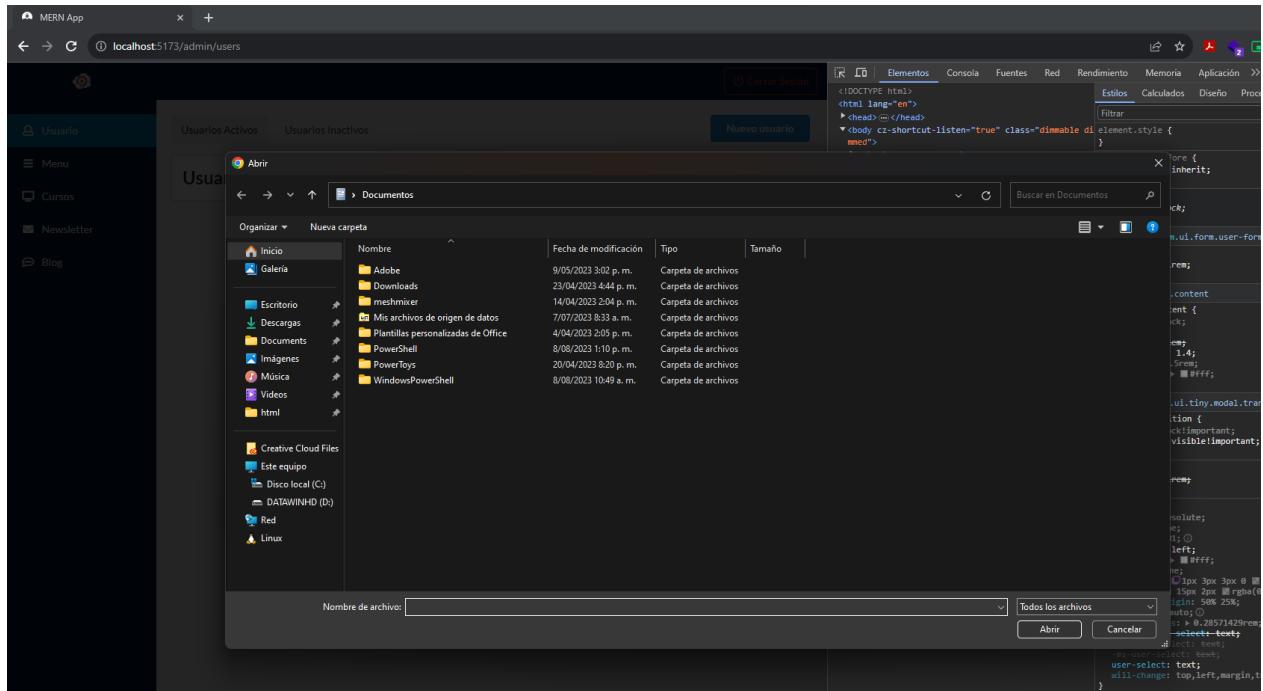
```

Ahora que estamos ya indicándole como lo va a recibir, vamos a ver en nuestro navegador, pero vamos a observar que no se ve nada en donde se supone que debe estar la imagen:

The screenshot shows a modal dialog titled "Crear nuevo usuario". Inside the dialog, there is a placeholder element with the class "user-form__avatar" and dimensions of 468x19.59. Below this, there are input fields for Nombre, Apellidos, Correo, and Contraseña, and a "Crear Usuario" button. To the right of the dialog, a detailed CSS tree is displayed, showing the inheritance of styles from various classes like ".ui.page.modals.dimmer.transition", ".ui.tiny.modal", ".ui.form", ".ui.modal", and ".ui.modal.content". The styles include flexbox properties, font sizes, line heights, and background colors.

Pero si lo ubicamos y le damos clic se nos debe abrir la ventana de carga de imágenes:





Ahora vamos a empezar a estilizarlo para que se nos vea mucho mejor, para ello nos hacemos la importación de las imágenes de assets:

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
1  import React, { useCallback } from 'react'
2  import { Form, Image } from "semantic-ui-react"
3  import { useFormik } from "formik"
4  import { useDropzone } from "react-dropzone"
5  import { image } from "../../../../../assets"
6  import { initialValues, validationSchema } from "./UserForm.form"
7  import "./UserForm.scss"
8
9  export function UserForm(props) {
10
11
```

Creamos una const que se llame getAvatar:

```
src > components > Admin > Users > UserForm > UserForm.jsx > getAvatar
30  const { getRootProps, getInputProps } = useDropzone({
31    accept: "image/jpeg, image/jpg, image/png",
32    onDrop,
33  })
34
35  const getAvatar = () => {
36    return image.noAvatar
37  }
38
39  return (
40    <Form className="user-form" onSubmit={formik.handleSubmit}>
41
```

A través de esta función vamos a colocar la imagen por default y la vamos a colocar en nuestro src:



```

38
39     return (
40       <Form className="user-form" onSubmit={formik.handleSubmit}>
41
42         <div className="user-form__avatar" { ...getRootProps() }>
43           <input { ...getInputProps() } />
44           <Image avatar size="small" src={getAvatar()} />
45         </div>
46
47         <Form.Group widths="equal">
48           <Form.Input name="firstname" placeholder="Nombre" onChange={formik.ha

```

Ahora con esta función ya nos devolverá una imagen por defecto, vamos a nuestro navegador y observamos lo siguiente:

The screenshot shows a web application interface. On the left, there's a sidebar with navigation links: 'Usuario', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos', with 'Nuevo usuario' button. A modal window titled 'Crear nuevo usuario' is open, featuring a placeholder image of a person's head in the 'Avatar' input field. The browser's developer tools are open, showing the DOM structure and styles applied to the page.

Ya tenemos nuestra imagen por defecto, ahora vamos a darle estilo para que se vea mucho mejor, vamos a colocar lo siguiente en nuestro **UserForm.scss**:

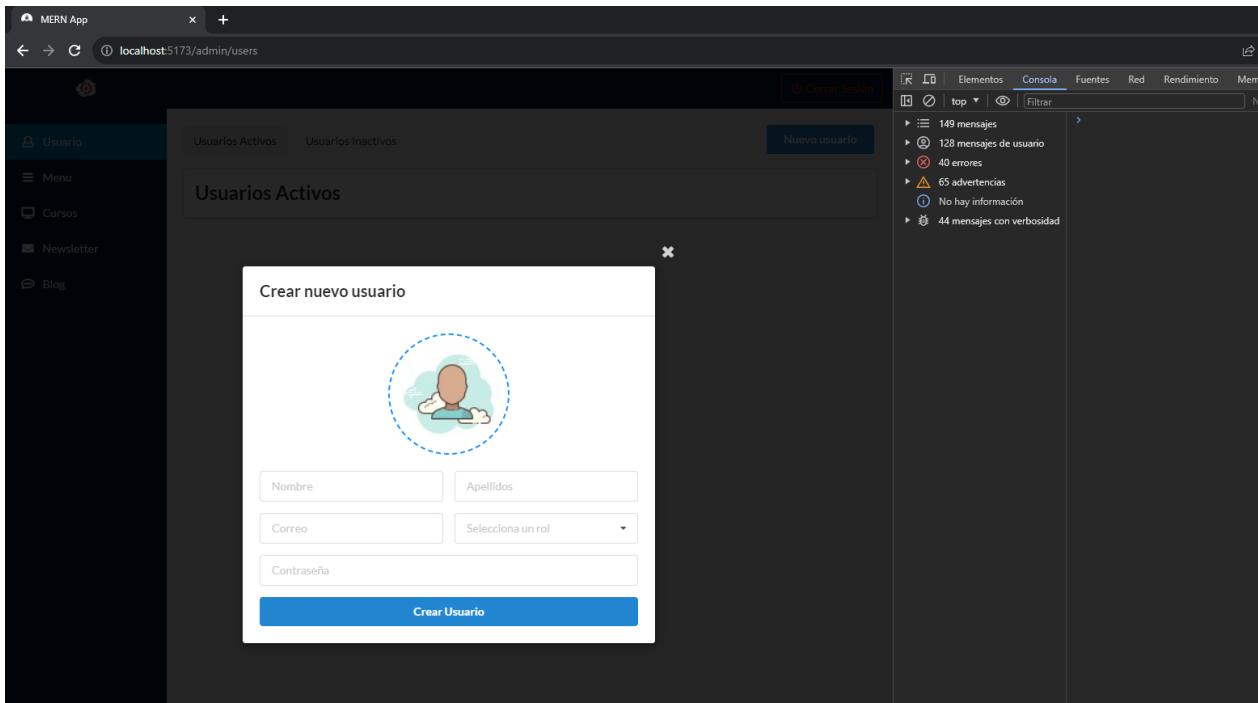


```

src > components > Admin > Users > UserForm > UserForm.scss > .user-form
1  @import "/src/scss/index.scss";
2
3  .user-form {
4    &__avatar {
5      display: flex;
6      align-items: center;
7      justify-content: center;
8      margin-bottom: 20px;
9
10   > .ui.image {
11     margin: 0;
12     border: 2px dashed $primary;
13     padding: 5px;
14     cursor: pointer;
15   }
16 }
17 }

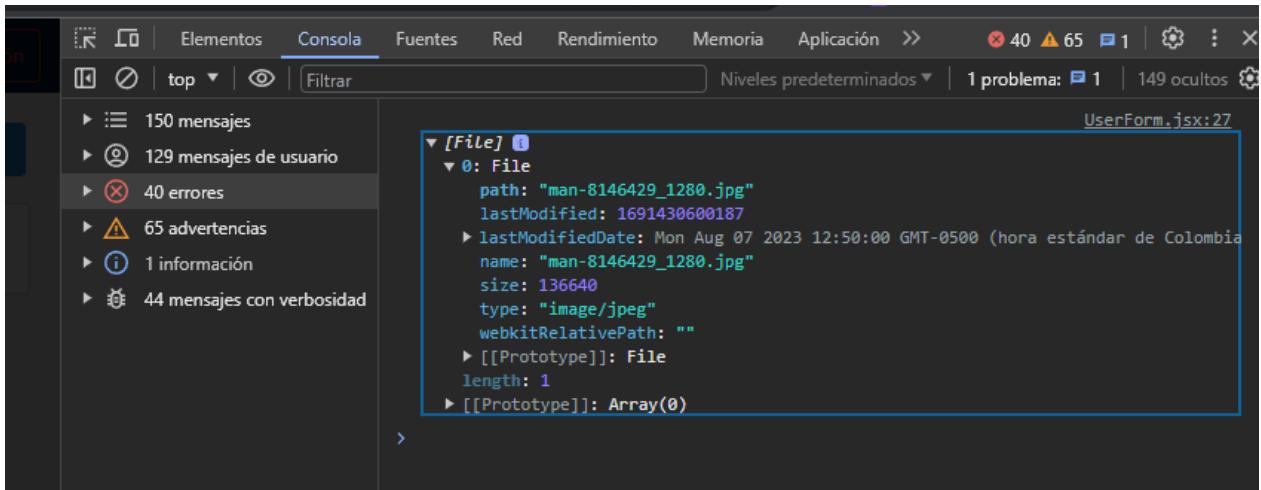
```

De esta manera le damos un toque mas bonito y se vería así:

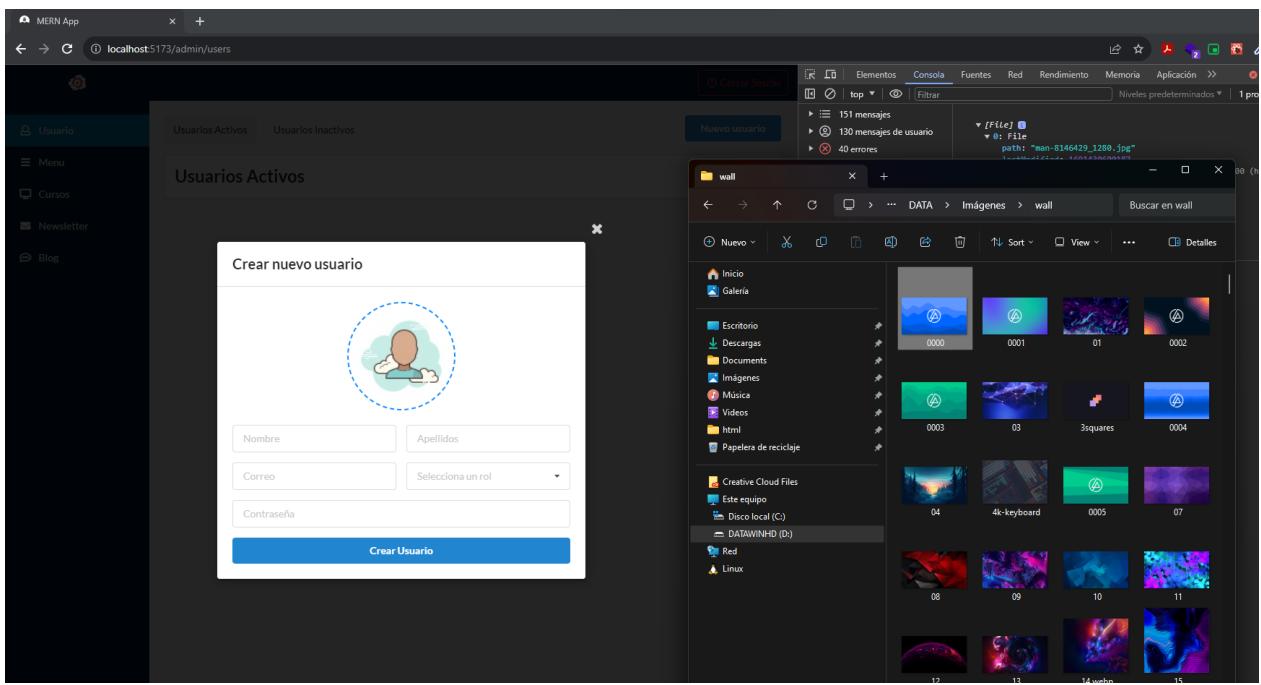


Si le damos clic y cargamos una imagen vamos a observar en nuestra consola lo siguiente:



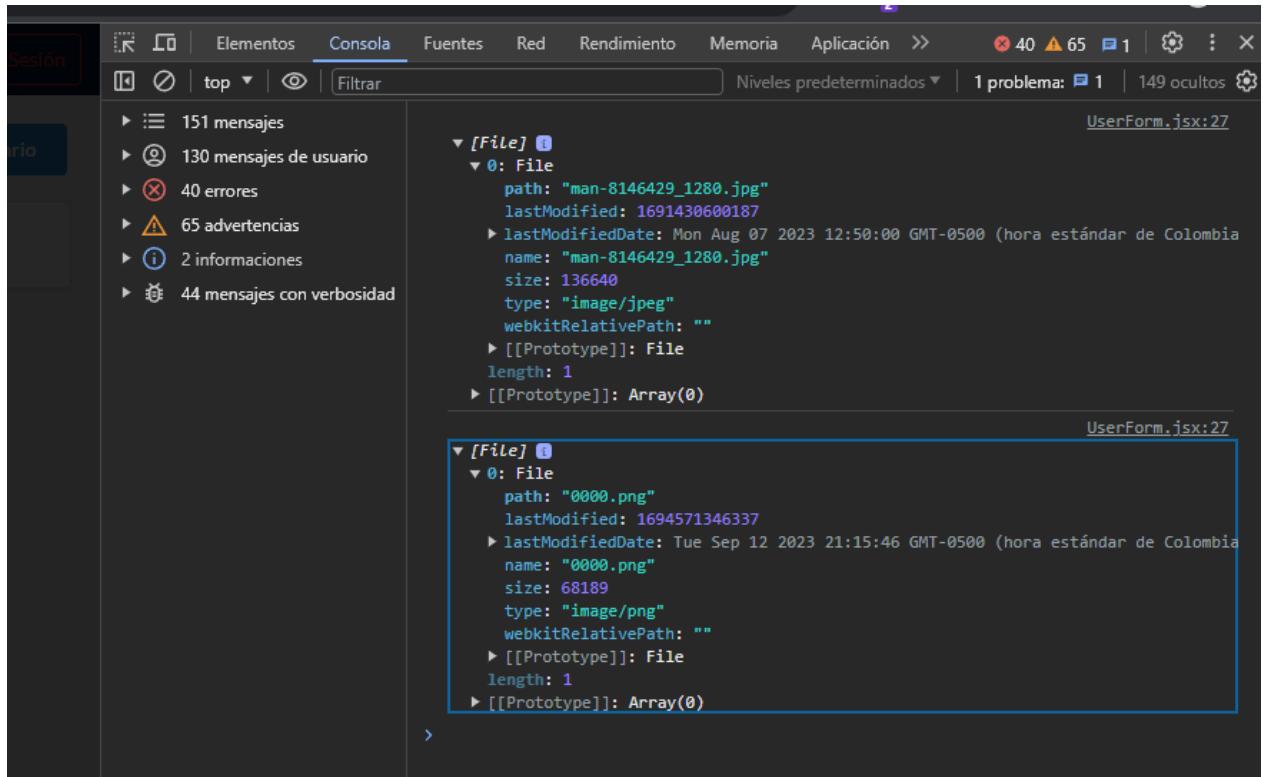


Esta manera funciona al igual que la de arrastrar y soltar sobre el modal la imagen:



Vamos a observar en nuestra consola que sale la información de nuestra imagen:





Ahora vamos a cargar la imagen dentro del avatar para que se visualice, para ello vamos a realizar lo siguiente:

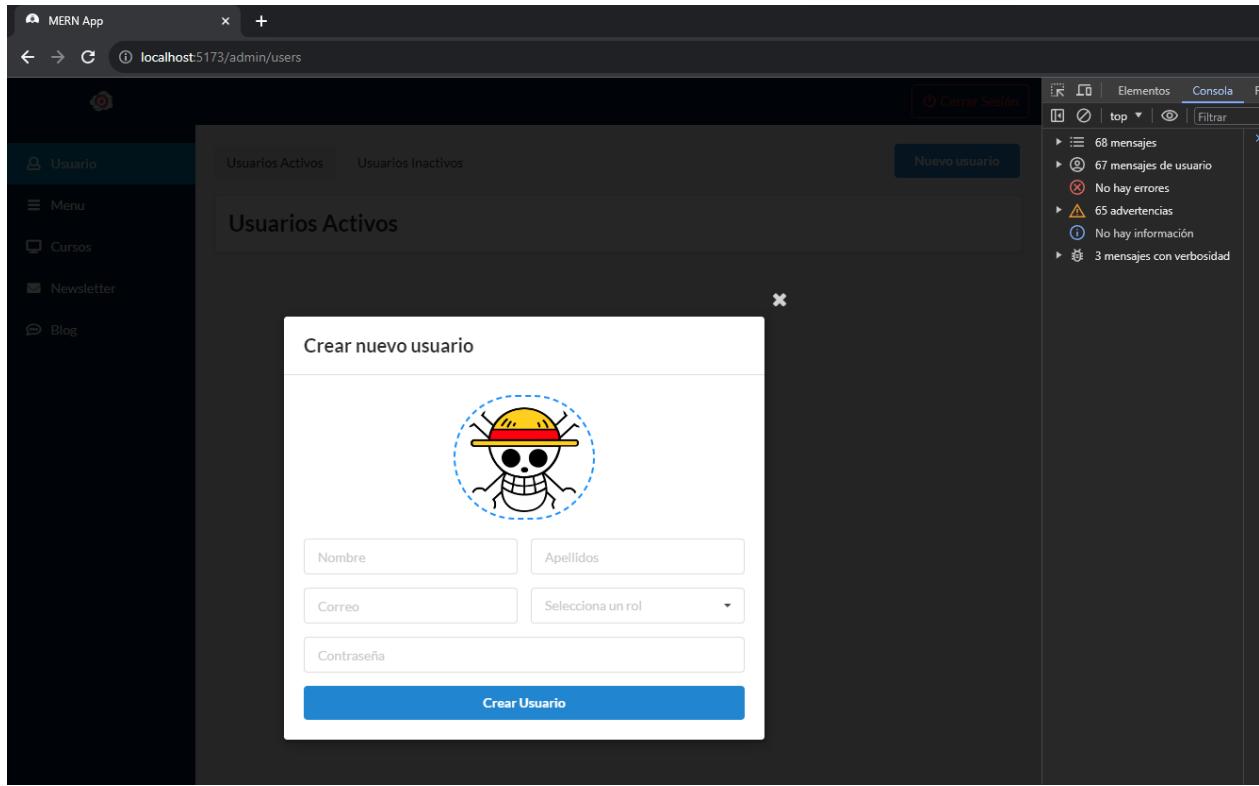
```
src > components > Admin > Users > UserForm > UserForm.jsx > UserForm > onDrop > useCallback() callback
25
26   const onDrop = useCallback((acceptedFiles) => {
27     const file = acceptedFiles[0]
28     formik.setFieldValue("avatar", URL.createObjectURL(file))
29     formik.setFieldValue("fileAvatar", file)
30   })
31
32   const { getRootProps, getInputProps } = useDropzone({
33     accept: "image/jpeg, image/jpg, image/png",
34   })
```

Tomamos el elemento 0 del array que estamos pasando de nuestro archivo y lo seteamos a través de formik para poderlo utilizar, ahora en nuestro getAvatar realizamos lo siguiente:

```
src > components > Admin > Users > UserForm > UserForm.jsx > UserForm > getAvatar
35
36
37   const getAvatar = () => [
38     if (formik.values.fileAvatar) {
39       return formik.values.avatar
40     }
41     return image.noAvatar
42   ]
43 }
```

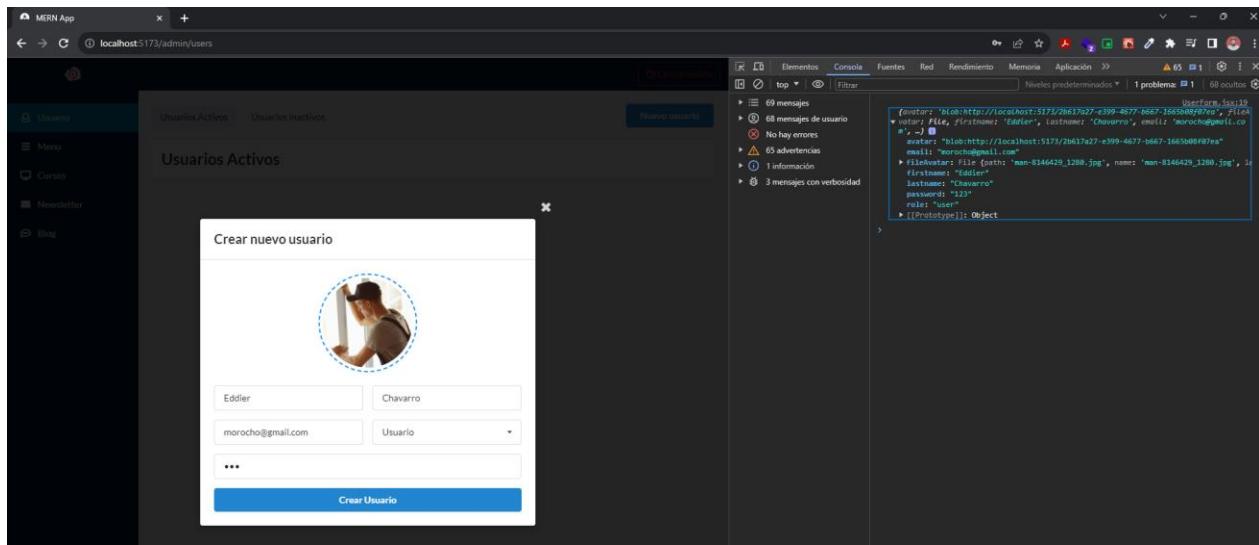
Ahora vamos a comprobar si funciona para ello vamos a realizar la carga de una imagen:





Y ya de esta manera nos está cargando la previsualización de nuestra imagen, notemos que la podemos arrastrar también directamente a nuestro modal sobre la imagen y lo va a cargar.

Ahora llenemos los datos del usuario y marquemos en crear usuario:



Ahora nos fijamos que todos los datos están siendo procesados para poder ser enviados a nuestra API.



CREANDO EL USUARIO

Ahora vamos a implementar la creación del usuario, para ello nos vamos a nuestra ruta `/api/user.jsx` y vamos a realizar una función nueva:

```
src > api > user.jsx > User > createUser
  25   |           throw error
  26   |
  27   |
  28
  29   async createUser(accessToken, data) {
  30     try {
  31       |       console.log(data)
  32     } catch (error) {
  33       throw error
  34     }
  35   }
  36 }
```

Y ahora nos vamos a importarla dentro de `UserForm.jsx` tambien importamos `useAuth` de `hooks`

```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
  1  import React, { useCallback } from 'react'
  2  import { Form, Image } from "semantic-ui-react"
  3  import { useFormik } from "formik"
  4  import { useDropzone } from "react-dropzone"
  5  import { User } from "../../../../../api"
  6  import { useAuth } from "../../../../../hooks"
  7  import { image } from "../../../../../assets"
  8  import { initialValues, validationSchema } from "./UserForm.form"
  9  import "./UserForm.scss"
 10
```

Y hacemos la implementación de la siguiente manera a través de `userController`:

```
 7  import { initialValues, validationSchema } from "./UserForm.form"
 8  import "./UserForm.scss"
 9
10 const userController = new User()
11
12 export function UserForm(props) {
13
```

Y ahora nos ubicamos dentro de `onSubmit`:



```

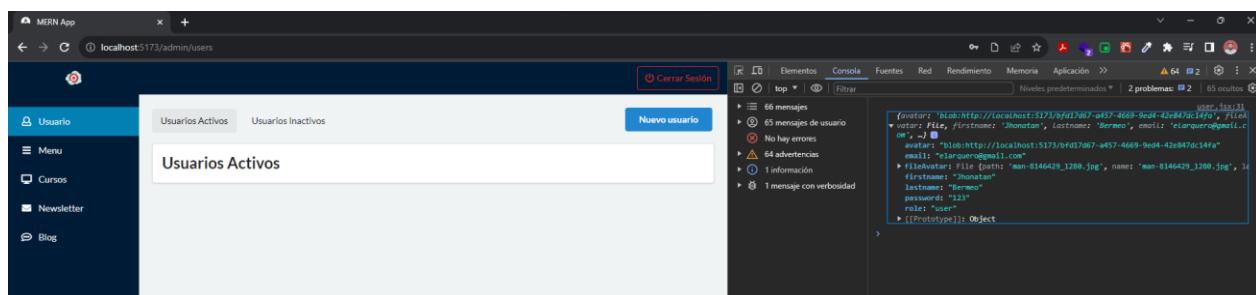
src > components > Admin > Users > UserForm > UserForm.jsx > formik > onSubmit

14
15     const { close, onReload, user } = props
16
17     const { accessToken } = useAuth()
18
19     const formik = useFormik({
20         initialValues: initialValues(),
21         validationSchema: validationSchema(),
22         validateOnChange: false,
23         onSubmit: async (formValue) => {
24             try {
25                 await userController.createUser(accessToken, formValue)
26                 close()
27             } catch (error) {
28                 console.error(error)
29             }
30         }
31     })
32
33     const onDrop = useCallback((acceptedFiles) => {
34         const file = acceptedFiles[0]

```

Pasamos nuestro accessToken y creamos un await para userController con los valores de formValue y hacemos un cierre del formulario con close.

Ahora comprobemos, llenamos los datos del formulario y los enviamos:



Observamos que se ha enviado a nuestro **user.jsx** y que esta saliendo en consola. Por el momento vamos a dejar comentado nuestro **close()** para evitar tener que estar llenando los datos a cada momento:

```

21     validationSchema: validationSchema(),
22     validateOnChange: false,
23     onSubmit: async (formValue) => {
24         try {
25             await userController.createUser(accessToken, formValue)
26             //close()
27         } catch (error) {
28             console.error(error)
29         }
30     }
31 }
32

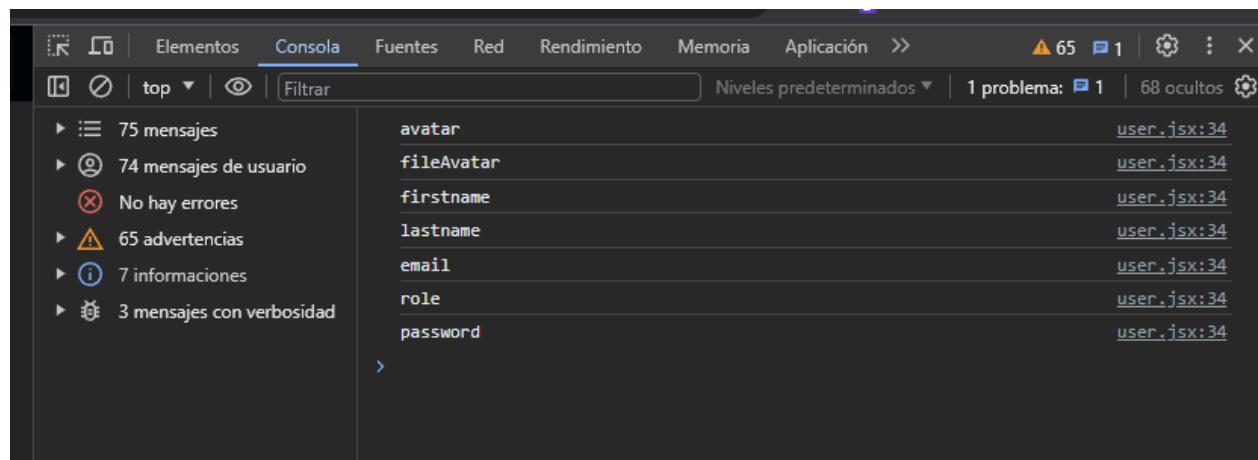
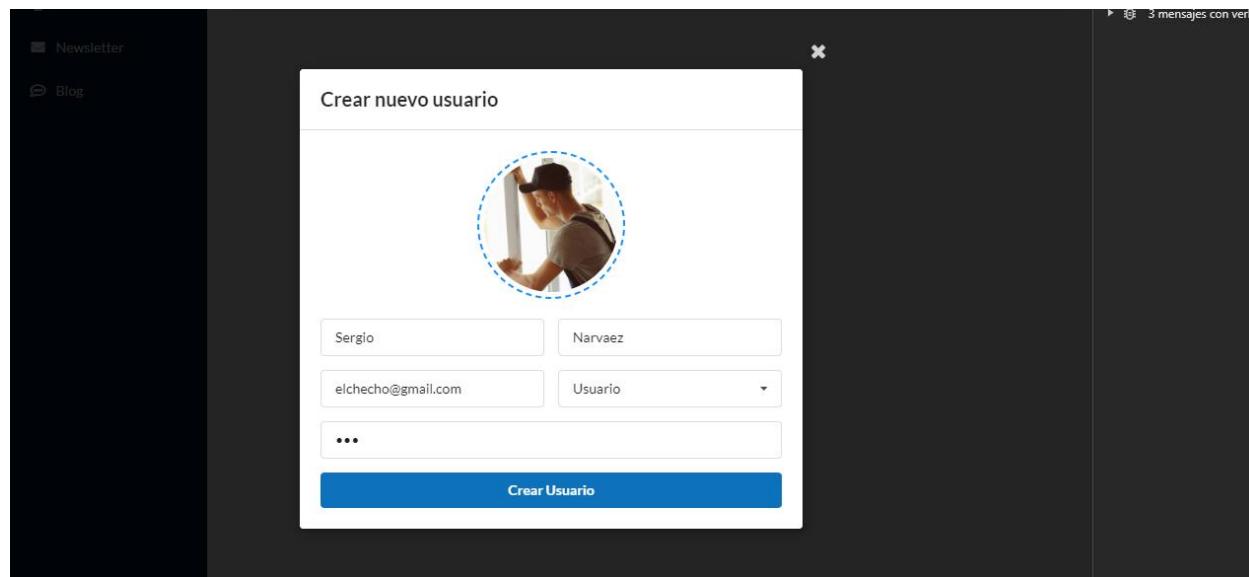
```



Ahora en nuestro user.jsx vamos a realizar un **Object.keys** para convertir nuestro objeto de datos en un array y tomar las keys de cada uno, esto para agilizar el proceso de guardado que vamos a realizar:

```
src > api > user.jsx > User > createUser > forEach() callback
29     async createUser(accesToken, data) {
30         try {
31             const formData = new FormData()
32
33             Object.keys(data).forEach((key) => {
34                 console.log(key)
35             })
36
37         } catch (error) {
38             throw error
39         }
40     }
41 }
```

Volvemos nuevamente a cargar datos en nuestro formulario y le damos a enviar:



Ahora que ya estoy sacando las keys vamos a realizar una pequeña modificación al código para sacar clave y valor:

```
32
33     Object.keys(data).forEach((key) => {
34         |     console.log(` ${key} = ${data[key]} `)
35     })
36
37     } catch (error) {
38         throw error
```

Y vuelvo nuevamente a enviar los datos:

The screenshot shows a user creation form titled "Crear nuevo usuario". The form fields are: Nombre (Paula), Apellido (Medina), Email (paulis@gmail.com), and Role (Usuario). Below the form is a "Crear Usuario" button. To the right, a network request log shows the data being sent in JSON format:

```
> 15 errores
> 140 advertencias
> 14 informaciones
> 93 mensajes con verbosidad
firstname          user.jsx:34
lastname           user.jsx:34
email              user.jsx:34
role               user.jsx:34
password          user.jsx:34
avatar             blob:https://localhost:5273/c0e85719-a700-4ad2-aec4-809e47720e user.jsx:34
fileAvatar         [Object File] user.jsx:34
firstname = Paula user.jsx:34
lastname = Medina user.jsx:34
email = paulis@gmail.com user.jsx:34
role = user user.jsx:34
password = 123 user.jsx:34
>
```

Ahora que ya entendimos como simplificamos el proceso vamos a colocar nuestro append:

```
31     const formData = new FormData()
32
33     Object.keys(data).forEach((key) => {
34         |     formData.append(key, data[key])
35     })
36
37     } catch (error) {
```

Ahora vamos a realizar una condición porque el avatar no siempre va a llegar, así que vamos a realizar lo siguiente:

```
32
33     Object.keys(data).forEach((key) => {
34         formData.append(key, data[key])
35     })
36
37     if (data.fileAvatar) {
38         |     formData.append("avatar", data.fileAvatar)
39     }
40
41     } catch (error) {
42         throw error
43     }
```



@hdtoledo

Y ahora si realizamos nuestra función de la ruta, para ello nos vamos a `/utils/constants.jsx` y creamos `USER`:

```
src > utils > constants.jsx > ENV > API_ROUTES
1 const SERVER_IP = "localhost:3000"
2
3 export const ENV = {
4     BASE_PATH: `http://${SERVER_IP}`,
5     BASE_API: `http://${SERVER_IP}/api/v1`,
6     API_ROUTES: [
7         REGISTER: "auth/register",
8         LOGIN: "auth/login",
9         REFRESH_ACCESS_TOKEN: "/auth/refresh_access_token",
10        USER_ME: "user/me",
11        USER: "user",
12    ],
13    JWT: {
14        ACCESS: "access",
15        REFRESH: "refresh",
16    },
17}
```

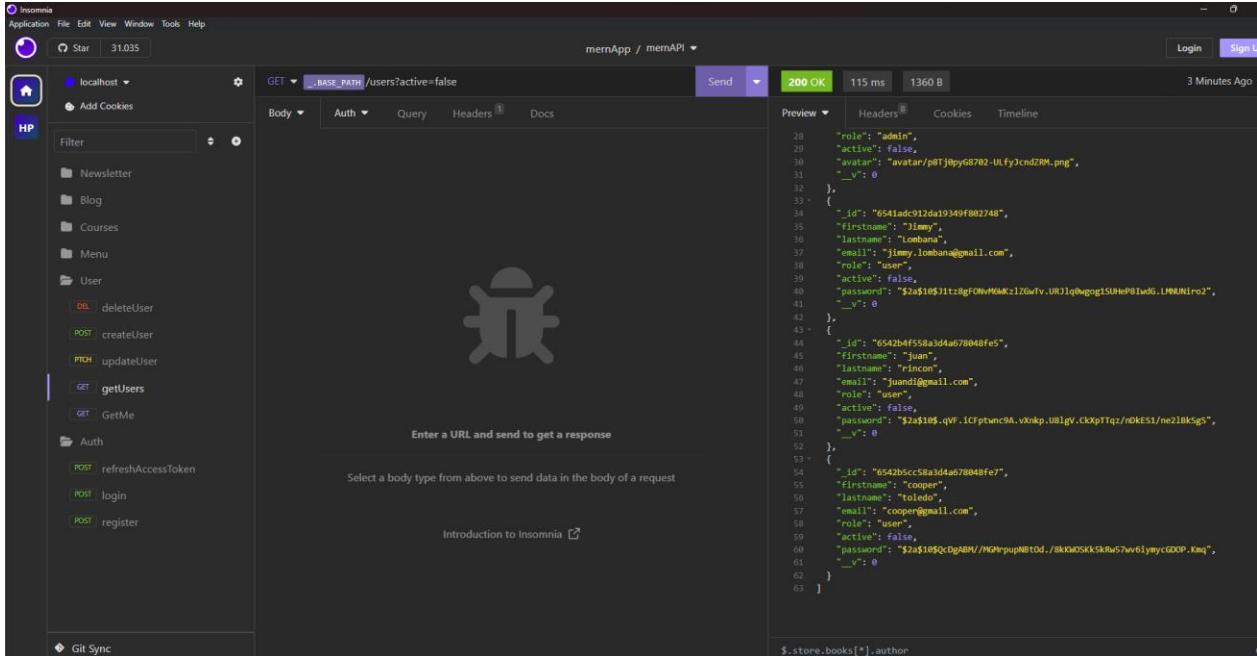
Y ahora colocamos nuestra const de URL en `user.jsx`:

```
src > api > user.jsx > User > createUser
29     async createUser(accessToken, data) {
30         try {
31             const formData = new FormData()
32
33             Object.keys(data).forEach((key) => {
34                 formData.append(key, data[key])
35             })
36
37             if (data.fileAvatar) {
38                 formData.append("avatar", data.fileAvatar)
39             }
40
41             const url = `${this.baseApi}/${ENV.API_ROUTES.USER}`
42             const params = {
43                 method: "POST",
44                 headers: {
45                     Authorization: `Bearer ${accessToken}`,
46                 },
47                 body: formData,
48             }
49
50             const response = await fetch(url, params)
51             const result = await response.json()
52
53             if (response.status !== 201) throw result
54             return result
55
56         } catch (error) {
57             throw error
58         }
59     }
60 }
```



@hdtoledo

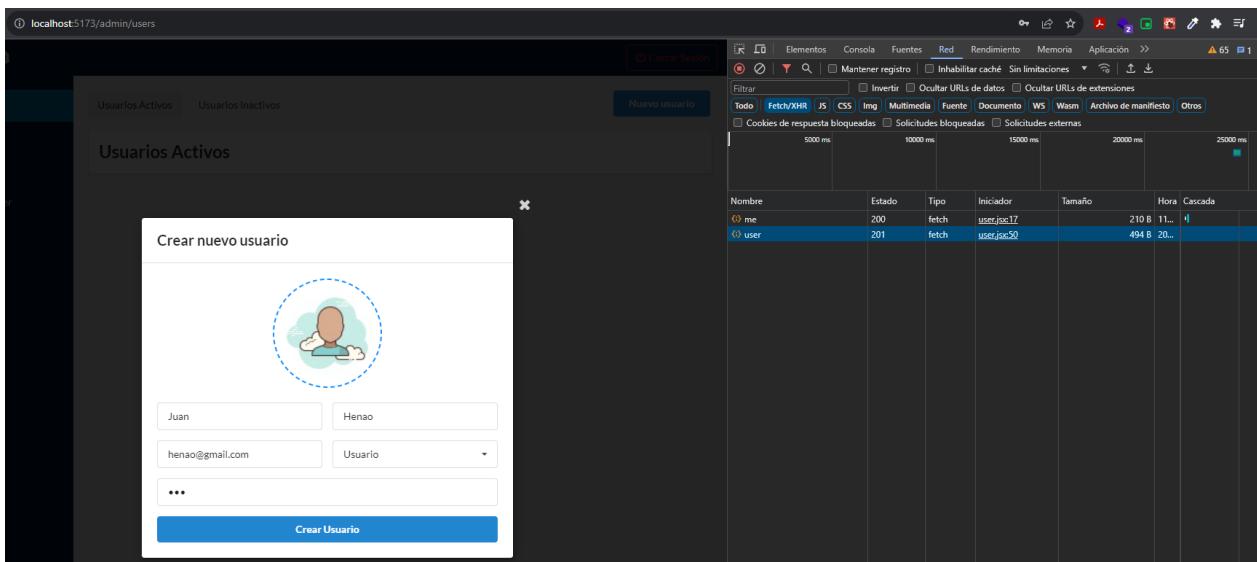
Ahora para comprobar que si estamos creando usuarios vamos a tener listo nuestro insomnia y vamos a dejarlo listo en getUsers:



The screenshot shows the Insomnia API client interface. At the top, it says "memApp / memAPI". Below that, the URL is "GET /users?active=false". The status is "200 OK" with "115 ms" and "1360 B". The response body is a JSON array of user documents:

```
[{"role": "admin", "active": true, "avatar": "avatar/p8Tj0eyg8702-ILfy3nd7RM.png", "_v": 0}, {"_id": "6541adc912da19349f802748", "firstname": "Jimmy", "lastname": "Lombana", "email": "jimmy.lombana@gmail.com", "role": "user", "active": true, "password": "$2a$10$1tz8gFOnW6Kc1ZGwIv.U0JqBwgogt5UHeP8Jw65.U00Niroz", "_v": 0}, {"_id": "6542b4ff558a3d4a078048fe5", "firstname": "juan", "lastname": "ricon", "email": "juand@email.com", "role": "user", "active": false, "password": "$2a$10$Q1tz8gFOnW6Kc1ZGwIv.U0JqBwgogt5UHeP8Jw65.U00Niroz", "_v": 0}, {"_id": "6542b5cc58a3d4a078048fe7", "firstname": "cooper", "lastname": "toledo", "email": "cooper@gmail.com", "role": "user", "active": false, "password": "$2a$10$Q1tz8gFOnW6Kc1ZGwIv.U0JqBwgogt5UHeP8Jw65.U00Niroz", "_v": 0}], ".store.books[*].author"
```

Vamos a llenar nuestros datos de nuevo de usuario y vamos a enviarlo:



The screenshot shows a browser window at "localhost:5173/admin/users". On the left, there's a sidebar with "Usuarios Activos" and "Usuarios Inactivos". A modal window titled "Crear nuevo usuario" is open, showing fields for "Nombre" (Juan), "Apellido" (Henao), "Email" (henao@gmail.com), and "Rol" (User). Below these fields is a "Crear Usuario" button. To the right, the developer tools Network tab is open, showing a successful POST request to "/users" with a status of 201.

Me dice que se ha creado correctamente:



```

Nombre: user
1 {
  "firstname": "Juan",
  "lastname": "Henao",
  "email": "henao@gmail.com",
  "password": "$2a$10$zcDjcCwcwS.eNdinFdIgu06P0.y9rCdrsbyvg8/vvZmvjN.0S9eGy",
  "role": "user",
  "active": false,
  "avatar": "",
  "_id": "654960ea0eba3ed601772867",
  "__v": 0
}

```

Los datos están pasando, voy a crear uno nuevo con avatar:

```

Nombre: user
1 {
  "firstname": "Juan Carlos",
  "lastname": "Henao Rubio",
  "email": "elrubio@gmail.com",
  "password": "92a3105249cavue.h0CeQXtu00PYb2B/QiCp3V0-4r7WjT3dIkM./yQK",
  "role": "user",
  "active": false,
  "avatar": "avatar/J19MD040uvqyIr0T5ylatReB.jpg",
  "_id": "654960ea0eba3ed601772867",
  "__v": 0
}

```

De igual manera me ha creado la solicitud ahora vamos a revisar en insomnia:



@hdtoledo

```

GET .BASE_PATH /users?active=false
Send 200 OK 105 ms 1854 B Just Now
Body Auth Query Headers Docs
Preview Headers Cookies Timeline
50 "password": "$2a$10$.qVF.ICFptwnc9A.vXnkp.UBlgV.CkXpTTqz/nDkE51/ne2lBksg5",
51 "_v": 0
52 },
53 {
54 "id": "6542b5cc58a3d4a678048fe7",
55 "firstname": "cooper",
56 "lastname": "toledo",
57 "email": "cooper@gmail.com",
58 "role": "user",
59 "active": false,
60 "password": "$2a$10$QcDgABM//KGhrpupNBt0d./8kkWOSkk5krw57vv6iymyccGDOP.Knq",
61 "_v": 0
62 },
63 {
64 "id": "654960ea0eba3ed601772867",
65 "firstname": "Juan",
66 "lastname": "Hernao",
67 "email": "hernao@gmail.com",
68 "password": "$2a$10$zcdjCwCs.eNd1nrdigu06P0.y9rCdrsbvgy8/vvZmjN.0S9e6y",
69 "role": "user",
70 "active": false,
71 "avatar": "",
72 "_v": 0
73 },
74 {
75 "id": "6549614f0eba3ed601772869",
76 "firstname": "Juan Carlos",
77 "lastname": "Hernao Rubio",
78 "email": "elrubio@gmail.com",
79 "password": "$2a$10$z4MpCauvmp.h2Ce00KtuBOPYbB2B/QLCp3Vo4r7vjtJdIbkM.yQK",
80 "role": "user",
81 "active": false,
82 "avatar": "avatar/JT9WDmZukqyLrOT5YlatBe8.jpg",
83 "_v": 0
84 }
85 ]

```

Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

Introduction to Insomnia

De esta manera estamos verificando que se están creando nuestros usuarios.

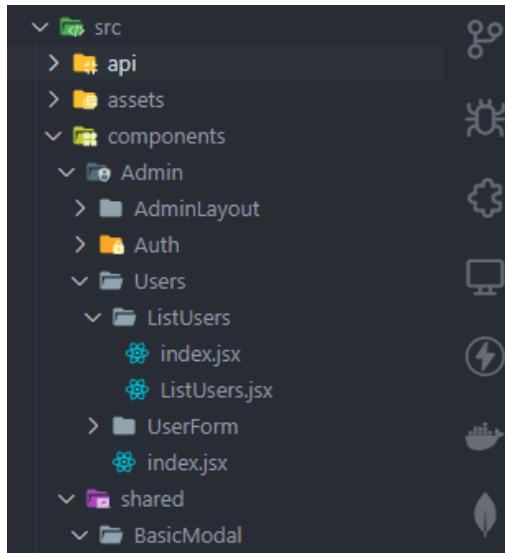
Quitamos el comentario que tenemos de Close()

```

22 validateOnchange: false,
23 onSubmit: async (formValue) => {
24     try {
25         await userController.createUser(accesToken, formValue)
26         close() // This line is being removed
27     } catch (error) {
28         console.error(error)
29     }
30 }
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
614
615
615
616
617
617
618
619
619
620
621
622
623
623
624
625
625
626
627
627
628
629
629
630
631
631
632
633
633
634
635
635
636
637
637
638
639
639
640
641
641
642
643
643
644
645
645
646
647
647
648
649
649
650
651
651
652
653
653
654
655
655
656
657
657
658
659
659
660
661
661
662
663
663
664
665
665
666
667
667
668
669
669
670
671
671
672
673
673
674
675
675
676
677
677
678
679
679
680
681
681
682
683
683
684
685
685
686
687
687
688
689
689
690
691
691
692
693
693
694
695
695
696
697
697
698
699
699
700
701
701
702
703
703
704
705
705
706
707
707
708
709
709
710
711
711
712
713
713
714
715
715
716
717
717
718
719
719
720
721
721
722
723
723
724
725
725
726
727
727
728
729
729
730
731
731
732
733
733
734
735
735
736
737
737
738
739
739
740
741
741
742
743
743
744
745
745
746
747
747
748
749
749
750
751
751
752
753
753
754
755
755
756
757
757
758
759
759
760
761
761
762
763
763
764
765
765
766
767
767
768
769
769
770
771
771
772
773
773
774
775
775
776
777
777
778
779
779
780
781
781
782
783
783
784
785
785
786
787
787
788
789
789
790
791
791
792
793
793
794
795
795
796
797
797
798
799
799
800
801
801
802
803
803
804
805
805
806
807
807
808
809
809
810
811
811
812
813
813
814
815
815
816
817
817
818
819
819
820
821
821
822
823
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1001
1001
1002
1003
1003
1004
1005
1005
1006
1007
1007
1008
1009
1009
1010
1011
1011
1012
1013
1013
1014
1015
1015
1016
1017
1017
1018
1019
1019
1020
1021
1021
1022
1023
1023
1024
1025
1025
1026
1027
1027
1028
1029
1029
1030
1031
1031
1032
1033
1033
1034
1035
1035
1036
1037
1037
1038
1039
1039
1040
1041
1041
1042
1043
1043
1044
1045
1045
1046
1047
1047
1048
1049
1049
1050
1051
1051
1052
1053
1053
1054
1055
1055
1056
1057
1057
1058
1059
1059
1060
1061
1061
1062
1063
1063
1064
1065
1065
1066
1067
1067
1068
1069
1069
1070
1071
1071
1072
1073
1073
1074
1075
1075
1076
1077
1077
1078
1079
1079
1080
1081
1081
1082
1083
1083
1084
1085
1085
1086
1087
1087
1088
1089
1089
1090
1091
1091
1092
1093
1093
1094
1095
1095
1096
1097
1097
1098
1099
1099
1100
1101
1101
1102
1103
1103
1104
1105
1105
1106
1107
1107
1108
1109
1109
1110
1111
1111
1112
1113
1113
1114
1115
1115
1116
1117
1117
1118
1119
1119
1120
1121
1121
1122
1123
1123
1124
1125
1125
1126
1127
1127
1128
1129
1129
1130
1131
1131
1132
1133
1133
1134
1135
1135
1136
1137
1137
1138
1139
1139
1140
1141
1141
1142
1143
1143
1144
1145
1145
1146
1147
1147
1148
1149
1149
1150
1151
1151
1152
1153
1153
1154
1155
1155
1156
1157
1157
1158
1159
1159
1160
1161
1161
1162
1163
1163
1164
1165
1165
1166
1167
1167
1168
1169
1169
1170
1171
1171
1172
1173
1173
1174
1175
1175
1176
1177
1177
1178
1179
1179
1180
1181
1181
1182
1183
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1191
1192
1193
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1201
1202
1203
1203
1204
1205
1205
1206
1207
1207
1208
1209
1209
1210
1211
1211
1212
1213
1213
1214
1215
1215
1216
1217
1217
1218
1219
1219
1220
1221
1221
1222
1223
1223
1224
1225
1225
1226
1227
1227
1228
1229
1229
1230
1231
1231
1232
1233
1233
1234
1235
1235
1236
1237
1237
1238
1239
1239
1240
1241
1241
1242
1243
1243
1244
1245
1245
1246
1247
1247
1248
1249
1249
1250
1251
1251
1252
1253
1253
1254
1255
1255
1256
1257
1257
1258
1259
1259
1260
1261
1261
1262
1263
1263
1264
1265
1265
1266
1267
1267
1268
1269
1269
1270
1271
1271
1272
1273
1273
1274
1275
1275
1276
1277
1277
1278
1279
1279
1280
1281
1281
1282
1283
1283
1284
1285
1285
1286
1287
1287
1288
1289
1289
1290
1291
1291
1292
1293
1293
1294
1295
1295
1296
1297
1297
1298
1299
1299
1300
1301
1301
1302
1303
1303
1304
1305
1305
1306
1307
1307
1308
1309
1309
1310
1311
1311
1312
1313
1313
1314
1315
1315
1316
1317
1317
1318
1319
1319
1320
1321
1321
1322
1323
1323
1324
1325
1325
1326
1327
1327
1328
1329
1329
1330
1331
1331
1332
1333
1333
1334
1335
1335
1336
1337
1337
1338
1339
1339
1340
1341
1341
1342
1343
1343
1344
1345
1345
1346
1347
1347
1348
1349
1349
1350
1351
1351
1352
1353
1353
1354
1355
1355
1356
1357
1357
1358
1359
1359
1360
1361
1361
1362
1363
1363
1364
1365
1365
1366
1367
1367
1368
1369
1369
1370
1371
1371
1372
1373
1373
1374
1375
1375
1376
1377
1377
1378
1379
1379
1380
1381
1381
1382
1383
1383
1384
1385
1385
1386
1387
1387
1388
1389
1389
1390
1391
1391
1392
1393
1393
1394
1395
1395
1396
1397
1397
1398
1399
1399
1400
1401
1401
1402
1403
1403
1404
1405
1405
1406
1407
1407
1408
1409
1409
1410
1411
1411
1412
1413
1413
1414
1415
1415
1416
1417
1417
1418
1419
1419
1420
1421
1421
1422
1423
1423
1424
1425
1425
1426
1427
1427
1428
1429
1429
1430
1431
1431
1432
1433
1433
1434
1435
1435
1436
1437
1437
1438
1439
1439
1440
1441
1441
1442
1443
1443
1444
1445
1445
1446
1447
1447
1448
1449
1449
1450
1451
1451
1452
1453
1453
1454
1455
1455
1456
1457
1457
1458
1459
1459
1460
1461
1461
1462
1463
1463
1464
1465
1465
1466
1467
1467
1468
1469
1469
1470
1471
1471
1472
1473
1473
1474
1475
1475
1476
1477
1477
1478
1479
1479
1480
1481
1481
1482
1483
1483
1484
1485
1485
1486
1487
1487
1488
1489
1489
1490
1491
1491
1492
1493
1493
1494
1495
1495
1496
1497
1497
1498
1499
1499
1500
1501
1501
1502
1503
1503
1504
1505
1505
1506
1507
1507
1508
1509
1509
1510
1511
1511
1512
1513
1513
1514
1515
1515
1516
1517
1517
1518
1519
1519
1520
1521
1521
1522
1523
1523
1524
1525
1525
1526
1527
1527
1528
1529
1529
1530
1531
1531
1532
1533
1533
1534
1535
1535
1536
1537
1537
1538
1539
1539
1540
1541
1541
1542
1543
1543
1544
1545
1545
1546
1547
1547
1548
1549
1549
1550
1551
1551
1552
1553
1553
1554
1555
1555
1556
1557
1557
1558
1559
1559
1560
1561
1561
1562
1563
1563
1564
1565
1565
1566
1567
1567
1568
1569
1569
1570
1571
1571
1572
1573
1573
1574
1575
1575
1576
1577
1577
1578
1579
1579
1580
1581
1581
1582
1583
1583
1584
1585
1585
1586
1587
1587
1588
1589
1589
1590
1591
1591
1592
1593
1593
1594
1595
1595
1596
1597
1597
1598
1599
1599
1600
1601
1601
1602
1603
1603
1604
1605
1605
1606
1607
1607
1608
1609
1609
1610
1611
1611
1612
1613
1613
1614
1615
1615
1616
1617
1617
1618
1619
1619
1620
1621
1621
1622
1623
1623
1624
1625
1625
1626
1627
1627
1628
1629
1629
1630
1631
1631
1632
1633
1633
1634
1635
1635
1636
1637
1637
1638
1639
1639
1640
1641
1641
1642
1643
1643
1644
1645
1645
1646
1647
1647
1648
1649
1649
1650
1651
1651
1652
1653
1653
1654
1655
1655
1656
1657
1657
1658
1659
1659
1660
1661
1661
1662
1663
1663
1664
1665
1665
1666
1667
1667
1668
1669
1669
1670
1671
1671
1672
1673
1673
1674
1675
1675
1676
1677
1677
1678
1679
1679
1680
1681
1681
1682
1683
1683
1684
1685
1685
1686
1687
1687
1688
1689
1689
1690
1691
1691
1692
1693
1693
1694
1695
1695
1696
1697
1697
1698
1699
1699
1700
1701
1701
1702
1703
1703
1704
1705
1705
1706
1707
1707
1708
1709
1709
1710
1711
1711
1712
1713
1713
1714
1715
1715
1716
1717
1717
1718
1719
1719
1720
1721
1721
1722
1723
1723
1724
1725
1725
1726
1727
1727
1728
1729
1729
1730
1731
1731
1732
1733
1733
1734
1735
1735
1736
1737
1737
1738
1739
1739
1740
1741
1741
1742
1743
1743
1744
1745
1745
1746
1747
1747
1748
1749
1749
1750
1751
1751
1752
1753
1753
1754
1755
1755
1756
1757
1757
1758
1759
1759
1760
1761
1761
1762
1763
1763
1764
1765
1765
1766
1767
1767
1768
1769
1769
1770
1771
1771
1772
1773
1773
1774
1775
1775
1776
1777
1777
1778
1779
1779
1780
1781
1781
1782
1783
178
```

OBteniendo los usuarios activos e inactivos

Ahora nos vamos a ubicar en `/src/components/admin/users/` y vamos a crear la carpeta `ListUser` y dentro creamos nuestro `ListUsers.jsx index.jsx`



En nuestros index exportamos:

```
src > components > Admin > Users > index.jsx
1  export * from "./UserForm"
2  export * from "./ListUsers"
```

```
src > components > Admin > Users > ListUsers > index.jsx
1  export * from "./ListUsers"
```

Y en nuestro `ListUsers.jsx`:

```
src > components > Admin > Users > ListUsers > ListUsers.jsx > ...
1  import React from 'react'
2
3  export function ListUsers(props) {
4
5      const { usersActive } = props
6
7      return (
8          <div>
9              <h2>Estamos viendo los usuarios</h2>
10             <p>{ usersActive ? "Activos" : "Inactivos"}</p>
11         </div>
12     )
13 }
14 }
```



@hdtoledo

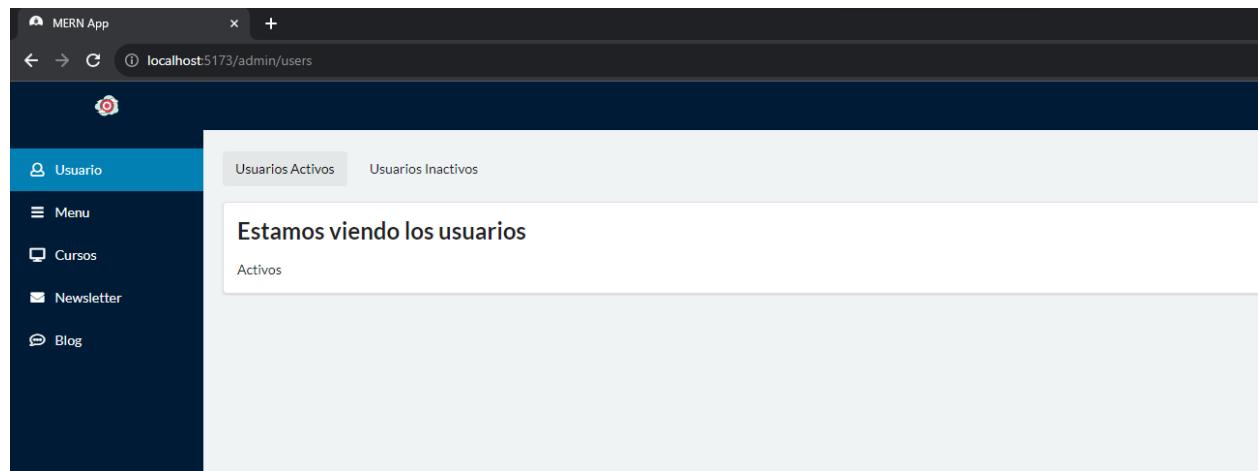
Ahora hacemos la importación de nuestro componente en **Users.jsx**

```
src > pages > admin > Users >  Users.jsx > ...
1 import React, { useState } from "react"
2 import { Tab, Button } from "semantic-ui-react"
3 import { BasicModal } from "../../components/shared"
4 import { UserForm, ListUsers } from "../../components/Admin/Users"
5 import "./Users.scss"
6
7
```

Y ahora hacemos la implementación:

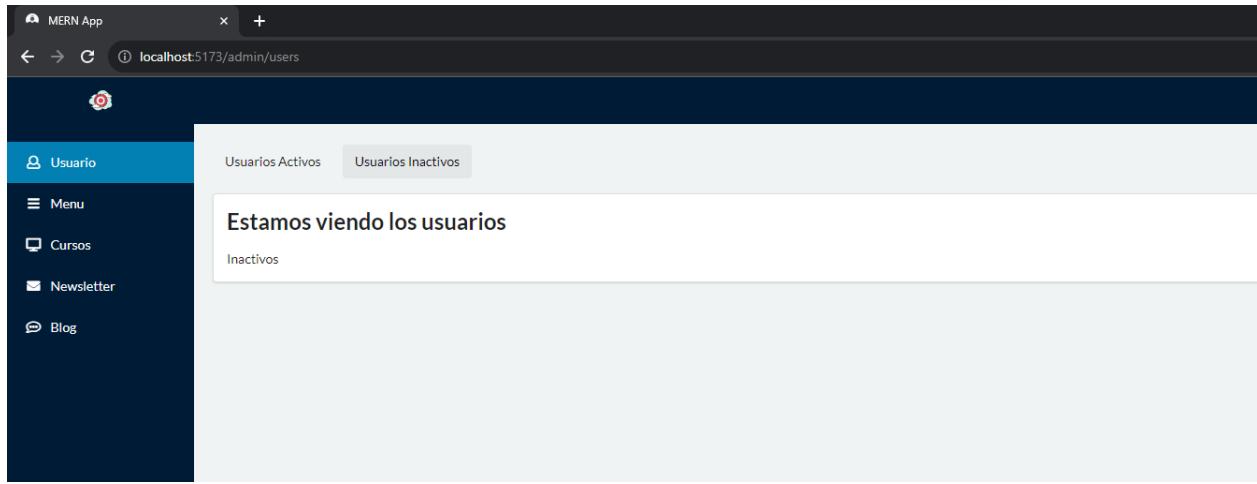
```
16 const panes = [
17   {
18     menuItem: "Usuarios Activos",
19     render: () => (
20       <Tab.Pane attached={false}>
21         <ListUsers usersActive={true}/>
22       </Tab.Pane>
23     )
24   },
25   {
26     menuItem: "Usuarios Inactivos",
27     render: () => (
28       <Tab.Pane attached={false}>
29         <ListUsers usersActive={false}/>
30       </Tab.Pane>
31     )
32   },
33 ]
34
35 return (
36
```

Dentro de nuestro componente colocamos **usersActive** y lo dejamos en true or false, vamos a observar lo siguiente en nuestro navegador:



Y si pasamos a nuestros usuarios inactivos:





Ya tenemos nuestro componente funcionando entre comillas, ahora vamos a construir la función para que se nos muestren nuestros usuarios activos y/o inactivos, para ello lo primero que vamos a hacer es ubicarnos en [/api/user.jsx](#):

```
src > api > user.jsx > User
58     }
59   }
60
61   async getUsers(accessToken, active = undefined ) {
62     try {
63       const url = ''
64     } catch (error) {
65       throw error
66     }
67   }
68 }
69 }
```

Vamos a construir poco a poco nuestra función ahora para continuar debemos generar la ruta de nuestra url, nos dirigimos a [/utils/constants.jsx](#):

```
src > utils > constants.jsx > ENV > API_ROUTES
1  const SERVER_IP = "localhost:3000"
2
3  export const ENV = {
4    BASE_PATH: `http://${SERVER_IP}`,
5    BASE_API: `http://${SERVER_IP}/api/v1`,
6    API_ROUTES: [
7      REGISTER: "auth/register",
8      LOGIN: "auth/login",
9      REFRESH_ACCESS_TOKEN: "auth/refresh_access_token",
10     USER_ME: "user/me",
11     USER: "user",
12     USERS: "users",
13   ],
14   JWT: {
15     ACCESS: "access",
16     REFRESH: "refresh",
17   },
18 }
```



@hdtoledo

Y ahora continuamos con nuestra función:

```
src > api > User.js > User > getUsers
...
58     }
59   }
60
61   async getUsers(accessToken, active = undefined ) {
62     try {
63       const url = `${this.baseApi}/${ENV.API_ROUTES.USERS}?active=${active}`
64       const params = {
65         headers: {
66           Authorization: `Bearer ${accessToken}`
67         }
68       }
69
70       const response = await fetch(url, params)
71       const result = await response.json()
72
73       if (response.status !== 200) throw result
74       return result
75     }
76     catch (error) {
77       throw error
78     }
79   }
80 }
81 }
```

Ahora realizamos la importación de nuestra función en **ListUsers.js**, además de importar useState y useEffect

```
src > components > Admin > Users > ListUsers > ListUsers.jsx > UserController
1  import React, { useState, useEffect } from 'react'
2  import { User } from "../../../../../api"
3  import { useAuth } from "../../../../../hooks"
4
5  const UserController = new User()
```

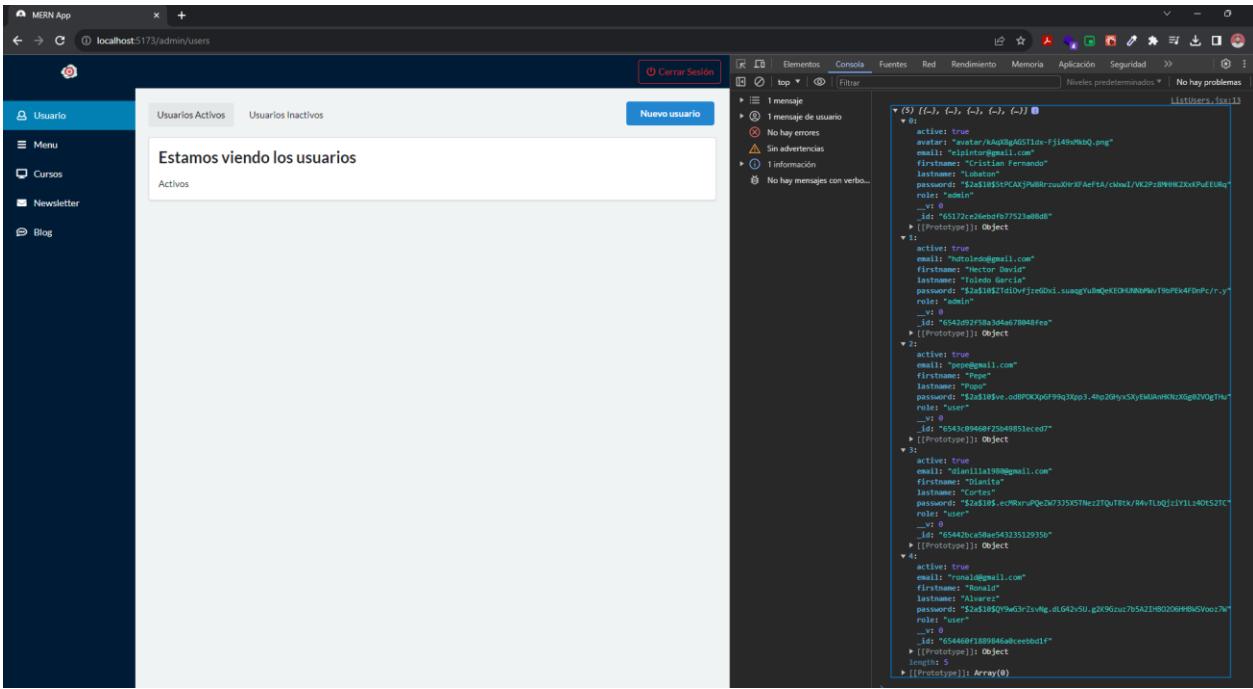
Hacemos nuestra implementación a través de UserController, y ahora realizamos lo siguiente:

```
1  const userController = new UserController()
2
3  export function ListUsers(props) {
4
5    const { usersActive } = props
6    const [users, setUsers] = useState(null)
7    const { accessToken } = useAuth()
8
9    console.log(users)
10
11    useEffect(() => {
12      (async () => {
13        try {
14          const response = await userController.getUsers(accessToken, usersActive)
15          setUsers(response)
16        } catch (error) {
17          console.log(error)
18        }
19      })()
20    }, [users])
21
22
23    return (
24      <div>
25        <h2>Estamos viendo los usuarios</h2>
26        <p>{ usersActive ? "Activos" : "Inactivos" }</p>
27      </div>
28    )
29  }
```



@hDTOledo

Ahora vamos a nuestro navegador y nos dirigimos a nuestra consola y recargamos:



The screenshot shows a browser window for a 'MERN App' at 'localhost:5173/admin/users'. On the left, there's a sidebar with 'Usuario' selected, followed by 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos', with 'Activos' currently selected. A blue button labeled 'Nuevo usuario' is at the top right. The central area displays the message 'Estamos viendo los usuarios' and 'Activos'. To the right of the main content is a developer console window titled 'Consola'. It shows an array of five user objects under the variable 'users'. Each user object contains properties like 'active', 'avatar', 'email', 'firstname', 'lastname', 'password', 'role', and '_id'. The developer console also shows other status indicators: 1 message, 1 message de usuario, No hay errores, Sin advertencias, 1 información, and No hay mensajes con verbo...

```
[{"_id": "653172c2e1ebefb7e523a0d0d", "active": true, "avatar": "/img/0gAGStIox-Fj149x0tBQ.png", "email": "elipitero@gmail.com", "firstname": "Cesar", "lastname": "Fernando", "password": "82a10515pCAxPMbRrzuuXhrXfAeftA/cIwZPzBmHh2XxPuEEUu", "role": "admin", "vi": 0}, {"_id": "653172c2e1ebefb7e523a0d0e", "active": true, "email": "hdtoledo@gmail.com", "firstname": "Hector", "lastname": "Toledo", "password": "82a10515pCAxPMbRrzuuXhrXfAeftA/cIwZPzBmHh2XxPuEEUu", "role": "admin", "vi": 0}, {"_id": "653172c2e1ebefb7e523a0d0f", "active": true, "email": "hdtoledo@gmail.com", "firstname": "Hector", "lastname": "David", "password": "82a10515pCAxPMbRrzuuXhrXfAeftA/cIwZPzBmHh2XxPuEEUu", "role": "user", "vi": 0}, {"_id": "653172c2e1ebefb7e523a0d0g", "active": true, "email": "hdtoledo@gmail.com", "firstname": "Hector", "lastname": "Hector", "password": "82a10515pCAxPMbRrzuuXhrXfAeftA/cIwZPzBmHh2XxPuEEUu", "role": "user", "vi": 0}, {"_id": "653172c2e1ebefb7e523a0d0h", "active": true, "email": "hdtoledo@gmail.com", "firstname": "Hector", "lastname": "Toledo", "password": "82a10515pCAxPMbRrzuuXhrXfAeftA/cIwZPzBmHh2XxPuEEUu", "role": "user", "vi": 0}], [{"text": "No hay problemas"}]
```

Ya estamos observando nuestros usuarios los cuales están activos pero si le damos en usuarios inactivos no se nos muestran, ya que solo estamos haciendo el de usuarios inactivos para ello vamos a realizar lo siguiente:

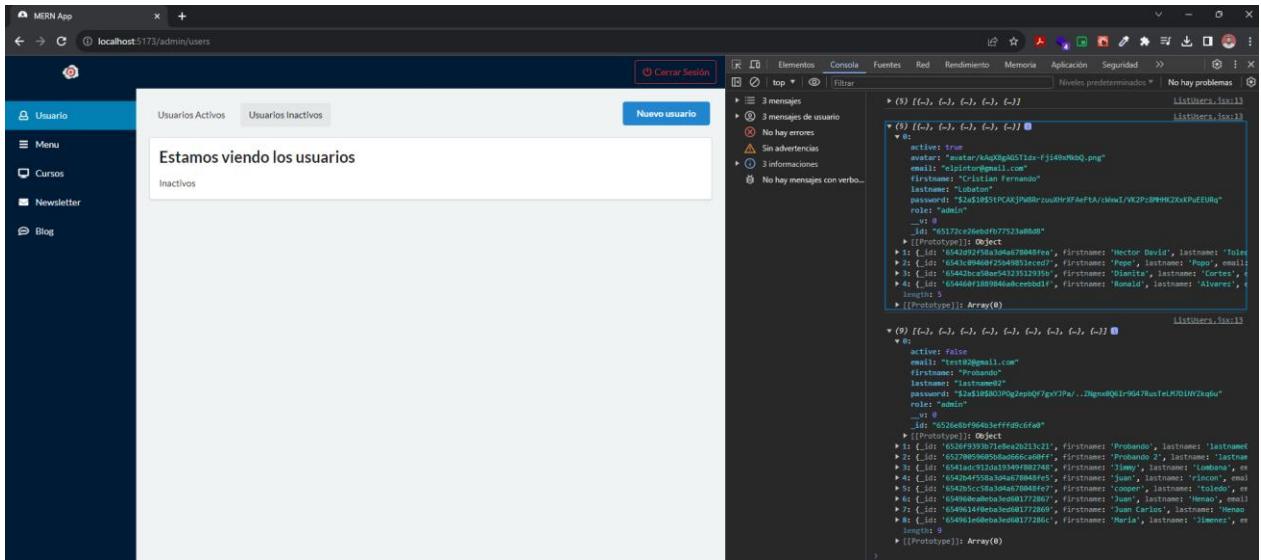


```
13 console.log(users)
14
15 useEffect(() => {
16   (async () => {
17     try {
18       const response = await userController.getUsers(accesToken, usersActive)
19       setUsers(response)
20     } catch (error) {
21       console.log(error)
22     }
23   })()
24 }, [usersActive])
25
26
27 return (

```

Pasamos nuestro `usersActive` en el array del `useEffect` y si vamos al navegador al realizar el cambio de activos o inactivos me muestra en consola los usuarios que se encuentran con dicho estado:





Ahora que ya los tenemos de esta manera vamos a realizar que cuando no tengamos información que mostrar nos valide de igual manera, para ello vamos a importar Loader de semantic UI y size de lodash

```
src > components > Admin > Users > ListUsers >  ListUsers.jsx > ...
1  import React, { useState, useEffect } from 'react'
2  import { Loader } from "semantic-ui-react"
3  import { size } from "lodash"
4  import { User } from "../../../../../api"
5  import { useAuth } from "../../../../../hooks"
6
```

Y para ello realizamos las siguientes condiciones:

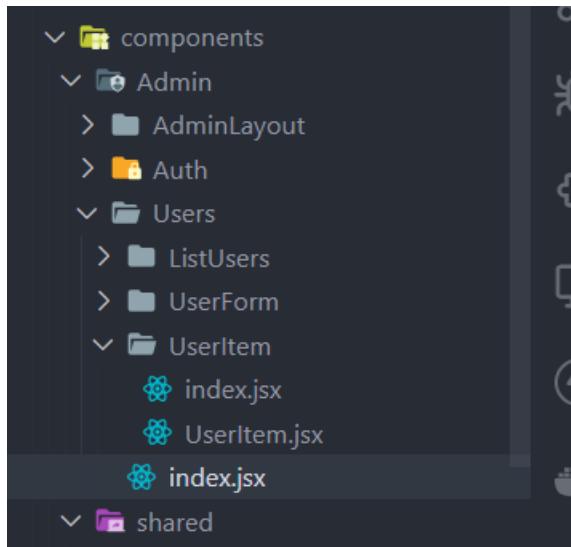
```
23    }
24  },
25
26  if(!users) return <Loader active inline="centered"/>
27  if(size(users) === 0) return "No hay ningun usuario"
28
29  return (
30    <div>
```

Si vamos a nuestro navegador el componente realizara una pequeña animación del loader al momento de traernos la información.



RENDERIZANDO LOS USUARIOS

Ahora para realizar la renderización de los usuarios vamos a crearnos un componente nuevo, que nos permitirá listar nuestros ítems, creamos dentro de **/components/admin/users/** nuestra carpeta **UserItem** y dentro colocamos **index.jsx – UserItem.jsx**:



Hacemos nuestros exports:

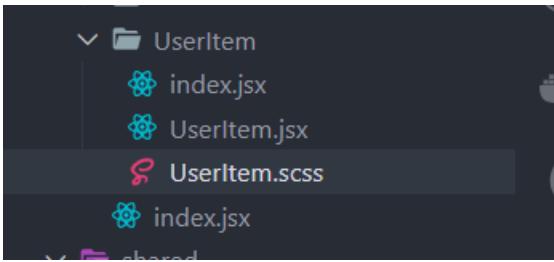
```
src > components > Admin > Users > index.jsx
1  export * from "./UserForm"
2  export * from "./ListUsers"
3  export * from "./UserItem"
```

```
src > components > Admin > Users > UserItem > index.jsx
1  export * from "./UserItem"
```

Y creamos el componente funcional base en **UserItem.jsx**

```
src > components > Admin > Users > UserItem > UserItem.jsx > ...
1 import React from 'react'
2 import './UserItem.scss'
3
4 export function UserItem() {
5   return (
6     <div>
7       <h2>Esto es un usuario</h2>
8     </div>
9   )
10 }
11
```

Además, creamos nuestro UserItem.scss



Y lo dejamos por el momento con la importación base:

```
src > components > Admin > Users > UserItem > UserItem.scss
1 @import "/src/scss/index.scss";
2
```

Ahora nos vamos para nuestro **ListUser.jsx** y realizamos la importación de UserItem y tambien de lodash:

```
src > components > Admin > Users > ListUsers > ListUsers.jsx > ...
1 import React, { useState, useEffect } from 'react'
2 import { Loader } from "semantic-ui-react"
3 import { size, map } from "lodash"
4 import { User } from "../../../../../api"
5 import { useAuth } from "../../../../../hooks"
6 import { UserItem } from "../UserItem"
7
```

Ahora lo implementamos de la siguiente manera:



@hdtoledo

```

src > components > Admin > Users > ListUsers > ListUsers.jsx > ListUsers
24   |   }()
25   |   }, [usersActive])
26
27   if(!users) return <Loader active inline="centered"/>
28   if(size(users) === 0) return "No hay ningun usuario"
29
30   return map(users, (user) => <UserItem key={user._id} user={user} />)
31
32
33

```

De esta manera empezamos a mapear nuestros usuarios, si lo revisamos en el navegador quedaría de la siguiente manera:

The screenshot shows a web application interface. On the left is a sidebar with icons for User, Menu, Courses, Newsletter, and Blog. The main content area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos'. The 'Nuevo usuario' button is visible. The main content area displays the text 'Esto es un usuario' repeated multiple times.

El resultado es que me devuelve una línea por cada usuario encontrado, si revisamos los inactivos:

The screenshot shows the same web application interface, but the 'Usuarios Inactivos' tab is now selected. The main content area displays the text 'Esto es un usuario' repeated multiple times.



Ahora nos ubicamos en **UserItem.jsx** y vamos a realizar unas importaciones:

```
src > components > Admin > Users > UserItem > UserItem.jsx > ...
1  import React from 'react'
2  import { Image, Button, Icon, Confirm } from "semantic-ui-react"
3  import { image } from "../../../../../assets"
4  import { ENV } from "../../../../../utils"
5  import "./UserItem.scss"
6
```

Ahora vamos a dejar nuestro componente base de la siguiente manera:

```
6
7  export function UserItem(props) {
8    const { user } = props;
9
10   return (
11     <>
12       <div className="user-item">
13         <div className="user-item__info">
14           <Image
15             avatar
16             src={
17               user.avatar ? `${ENV.BASE_PATH}/${user.avatar}` : image.noAvatar
18             }
19           />
20           <div>
21             <p>
22               {user.firstname} {user.lastname}
23             </p>
24             <p>{user.email}</p>
25           </div>
26         </div>
27       </div>
28     </>
29   );
30 }
31 |
```

De esta manera vamos a estar trayendo el avatar de la persona si lo tiene sino nos colocará el por defecto, y colocamos los datos básicos de nuestro usuario, si lo vemos en nuestro navegador saldrá así:



@hdtoledo

MERN App

localhost:5173/admin/users

Cerrar Sesión

Usuario

Menu

Cursos

Newsletter

Blog

Usuarios Activos

Nuevo usuario

Cristian Fernando Lobaton
elpintor@gmail.com

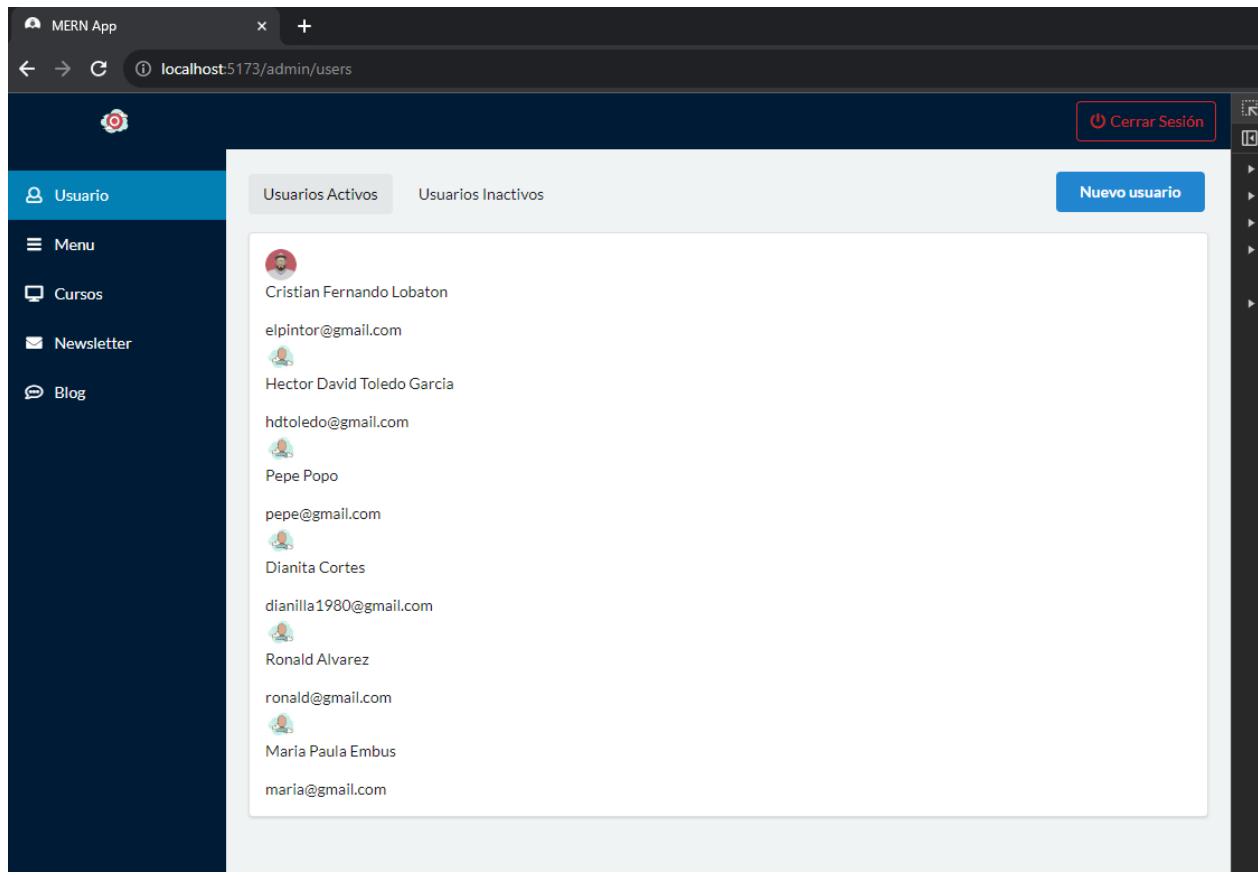
Hector David Toledo Garcia
hdtoledo@gmail.com

Pepe Popo
pepe@gmail.com

Dianita Cortes
dianilla1980@gmail.com

Ronald Alvarez
ronald@gmail.com

Maria Paula Embus
maria@gmail.com

A screenshot of a web application titled "MERN App" running on "localhost:5173/admin/users". The interface has a dark blue header with a logo and navigation links for "Usuario", "Menu", "Cursos", "Newsletter", and "Blog". On the right, there are buttons for "Nuevo usuario" and "Cerrar Sesión". The main content area is divided into two tabs: "Usuarios Activos" (Active Users) and "Usuarios Inactivos" (Inactive Users). The "Usuarios Activos" tab is selected, displaying a list of seven users with their names and email addresses. Each user entry includes a small circular profile picture. The users listed are: Cristian Fernando Lobaton (email: elpintor@gmail.com), Hector David Toledo Garcia (email: hdtoledo@gmail.com), Pepe Popo (email: pepe@gmail.com), Dianita Cortes (email: dianilla1980@gmail.com), Ronald Alvarez (email: ronald@gmail.com), and Maria Paula Embus (email: maria@gmail.com).

@hdtoledo

The screenshot shows a web application interface for managing users. On the left, there is a sidebar with navigation links: 'Usuario' (selected), 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area has two tabs: 'Usuarios Activos' (selected) and 'Usuarios Inactivos'. A blue button labeled 'Nuevo usuario' is located in the top right corner. The 'Usuarios Activos' tab displays a list of users with their names and email addresses:

User	Email
Probando lastname02	test02@gmail.com
Probando lastname02	test03@gmail.com
Probando 2 lastname03	test04@gmail.com
Jimmy Lombana	jimmy.lombana@gmail.com
juan rincon	juandi@gmail.com
cooper toledo	cooper@gmail.com
Juan Henao	henao@gmail.com
Juan Carlos Henao Rubio	elrubio@gmail.com
Maria Jimenez	lamarria@gmail.com

Ahora vamos a estilizar un poco nuestro UserItem.scss:

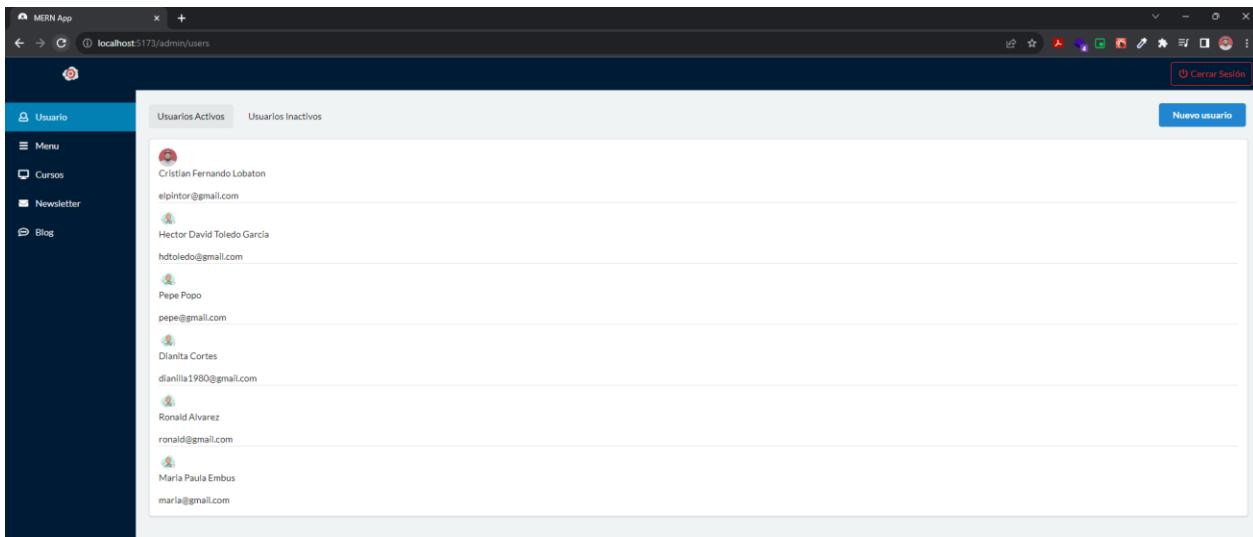


```

src > components > Admin > Users > UserItem >  UserItem.scss > ...
1  @import "/src/scss/index.scss";
2
3  .user-item {
4      display: flex;
5      justify-content: space-between;
6      align-content: center;
7      border-bottom: 1px solid #e4e4e4;
8      margin-bottom: 10px;
9
10     &:last-of-type {
11         margin-bottom: 0;
12         padding-bottom: 0;
13         border: 0;
14     }
15 }
16

```

Si lo vemos en nuestro navegador observaremos:



Usuarios Activos	Usuarios Inactivos
Cristian Fernando Lobaton eljintor@gmail.com	
Hector David Toledo Garcia hdtolledo@gmail.com	
Pepe Popo pepe@gmail.com	
Dianita Cortes dianilita1980@gmail.com	
Ronald Alvarez ronald@gmail.com	
Maria Paula Embus maria@gmail.com	

Ya lo tenemos un poco mas estilizado, luego terminaremos con los acabados, ahora continuamos con las acciones como lo son eliminar editar activar que van a ir dentro de nuestro componente:



```

23           </p>
24           <p>{user.email}</p>
25       </div>
26   </div>
27
28   <div>
29       <Button icon primary>
30           <Icon name="pencil"/>
31       </Button>
32       <Button icon color={user.active ? "orange" : "teal"}>
33           <Icon name={user.active ? "ban" : "check"/}>
34       </Button>
35       <Button icon color="red">
36           <Icon name="trash"/>
37       </Button>
38   </div>
39
40   </div>
41   </>
42 }
43
44

```

De esta manera vamos a tener nuestro botones de acción, si vamos al navegador vamos a observar lo siguiente:

User	Name	Email	Action Icons
1	Cristian Fernando Lobatón	elprintor@gmail.com	
2	Hector David Toledo García	hdtoledo@gmail.com	
3	Pepe Popo	pepe@gmail.com	

Y con los inactivos:

User	Name	Email	Action Icons
1	Probando lastname02	test02@gmail.com	
2	Probando lastname02	test03@gmail.com	
3	Probando 2 lastname03	test04@gmail.com	
4	Jimmy Lombana	jimmy.lombana@gmail.com	

Recordemos que se van a mostrar de colores diferentes en activos o inactivos, y que el botón de check me permitirá activar el usuario.

Ahora continuamos con nuestro archivo de UserItem.scss:



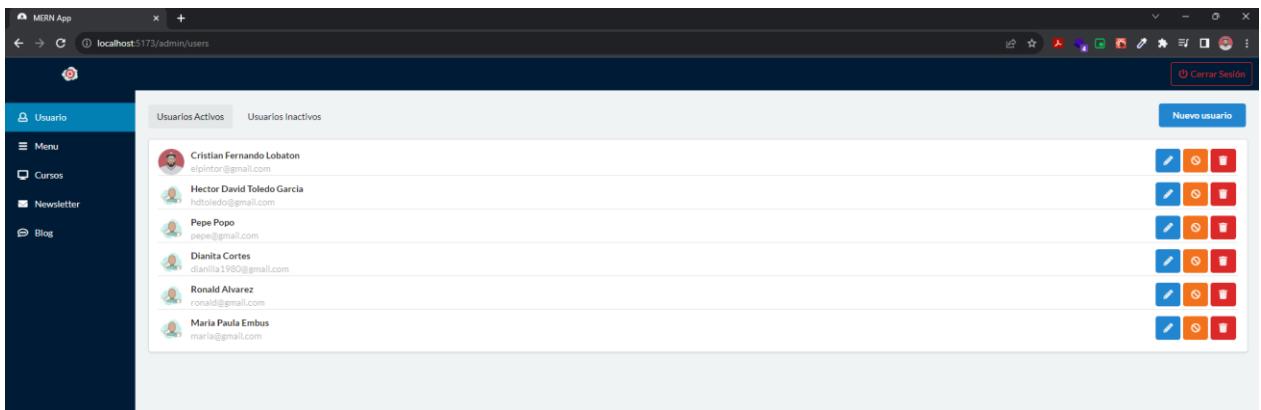
```

src > components > Admin > Users > Useritem > UserItem.scss > user-item > &_info > p > &last-of-type
1  @import "/src/scss/index.scss";
2
3  .user-item {
4      display: flex;
5      justify-content: space-between;
6      align-content: center;
7      border-bottom: 1px solid #e4e4e4;
8      margin-bottom: 10px;
9
10     &:last-of-type {
11         margin-bottom: 0;
12         padding-bottom: 0;
13         border: 0;
14     }
15
16     &_info {
17         display: flex;
18         align-items: center;
19
20         &gt; .ui.image {
21             width: 40px;
22             height: 40px;
23             margin-right: 10px;
24         }
25
26         p {
27             margin: 0;
28             &:first-of-type {
29                 font-weight: bold;
30             }
31             &:last-of-type {
32                 opacity: 0.4;
33             }
34         }
35     }
36 }
37

```

Lín. 32, col. 29 Espacios: 4 UTF-8 C

De esta manera ya terminamos nuestro estilizado quedando de la siguiente manera:



@hdtoledo

RECARGANDO USUARIOS AL CREAR UNO

Ahora vamos a hacer que se nos recargue nuestros datos de usuario ya que cuando lo creamos este no se esta recargando y es necesario que lo hagamos manualmente para que nos funcione, lo que vamos a realizar es muy sencillo, nos ubicamos en `/pages/admin/users/users.jsx` y acá vamos a utilizar un `useState` reload de la siguiente manera con un `onReload`:

```
src > pages > admin > Users > 🌐 Users.jsx > 🗂️ Users > 🖨️ onReload
  ...
  8
  9  export function Users() {
10
11    const [showModal, setShowModal] = useState(false)
12    const [reload, setReload] = useState(false)
13
14    const onOpenCloseModal = () => setShowModal((prevState) => !prevState)
15    const onReload = () => setReload((prevState) => !prevState)
16
17
18    const panes = [
19      {
```

Para implementarlo lo realizamos de la siguiente manera:



```

17
18 const panes = [
19   {
20     menuItem: "Usuarios Activos",
21     render: () => (
22       <Tab.Pane attached={false}>
23         <ListUsers usersActive={true} reload={reload}/>
24       </Tab.Pane>
25     )
26   },
27   {
28     menuItem: "Usuarios Inactivos",
29     render: () => (
30       <Tab.Pane attached={false}>
31         <ListUsers usersActive={false} reload={reload}/>
32       </Tab.Pane>
33     )
34   },
35 ]
36

```

De esta manera recargaremos automáticamente nuestros datos de usuario, ahora nos dirigimos a **ListUsers.jsx** y vamos a pasarlo por las props el reload:

```

10 export function ListUsers(props) {
11
12   const { usersActive, reload } = props
13   const [users, setUsers] = useState(null)
14   const { accessToken } = useAuth()
15
16   useEffect(() => {
17     (async () => {
18       try {
19         const response = await userController.getUsers(accessToken, usersActive)
20         setUsers(response)
21       } catch (error) {
22         console.log(error)
23       }
24     })()
25   }, [usersActive, reload])
26
27   if(!users) return <Loader active inline="centered"/>
28   if(size(users) === 0) return "No hay ningún usuario"

```

Además de ubicarlo también en nuestro array de **usersActive**. Ahora hay que ejecutar nuestra función cuando se cree el nuevo usuario entonces nos ubicamos de nuevo en **Users.jsx** y vamos a enviar el **onReload** a nuestro formulario:

```

45
46   <BasicModal show={showModal} close={onOpenCloseModal} title="Crear nuevo usuario">
47     <UserForm close={onOpenCloseModal} onReload={onReload}/>
48   </BasicModal>
49   </>
50 }
51

```

Ahora nos ubicamos en nuestro **/component/admin/users/userform/UserForm.jsx** y vamos a realizar lo siguiente:



@hdtoledo

```

src > components > Admin > Users > UserForm > UserForm.jsx > UserForm > formik > onSubmit
14
15  const { close, onReload, user } = props
16
17  const { accessToken } = useAuth()
18
19  const formik = useFormik({
20      initialValues: initialValues(),
21      validationSchema: validationSchema(),
22      validateOnChange: false,
23      onSubmit: async (formValue) => {
24          try {
25              await UserController.createUser(accessToken, formValue)
26              onReload()
27              close()
28          } catch (error) {
29              console.error(error)
30          }
31      }
32  })

```

Al momento de recargar debemos primero pasarlo antes del cierre porque sino nos genera un error, ahora si probamos el funcionamiento al crear un usuario vamos a observar cómo nos recarga automáticamente en nuestro navegador y nos mostrara el usuario creado inactivo.

FORMULARIO PARA ACTUALIZAR EL USUARIO

Vamos a empezar con nuestros botones de acción y para ello vamos a iniciar con el formulario de actualización de usuario, vamos a realizar unos cambios sobre **UserItem.jsx** y vamos a importar BasicModal, al igual que useState:

```

src > components > Admin > Users > UserItem > UserItem.jsx > ...
1  import React, { useState } from 'react'
2  import { Image, Button, Icon, Confirm } from "semantic-ui-react"
3  import { image } from "../../../../../assets"
4  import { ENV } from "../../../../../utils"
5  import { BasicModal } from "../../../../../shared"
6  import "./UserItem.scss"
7

```

Y hacemos la implementación, creamos unos estados:

```

7
8  export function UserItem(props) {
9      const { user } = props
10     const [showModal, setShowModal] = useState(false)
11     const [titleModal, setTitleModal] = useState("")
12
13     const onOpenCloseModal = () => setShowModal((prevState) => !prevState)
14
15     return (
16         <>
17         | <div className="user-item">

```

Y hacemos la implementación:



@hdtoledo

```

42         </Button>
43     </div>
44   </div>
45
46   <BasicModal show={showModal} close={onOpenCloseModal} title={titleModal}>
47     <p>UserForm</p>
48   </BasicModal>
49 </>
50 ];
51
52 }

```

Este modal se debe abrir al momento de pinchar el botón de editar el del lápiz, para ello vamos a hacer una función openUpdateUser:

```

13   const onOpenCloseModal = () => setShowModal((prevState) => !prevState)
14
15   const openUpdateuser = () => {
16     setTitleModal(`Actualizar ${user.email}`)
17     onOpenCloseModal()
18   }
19
20   return (

```

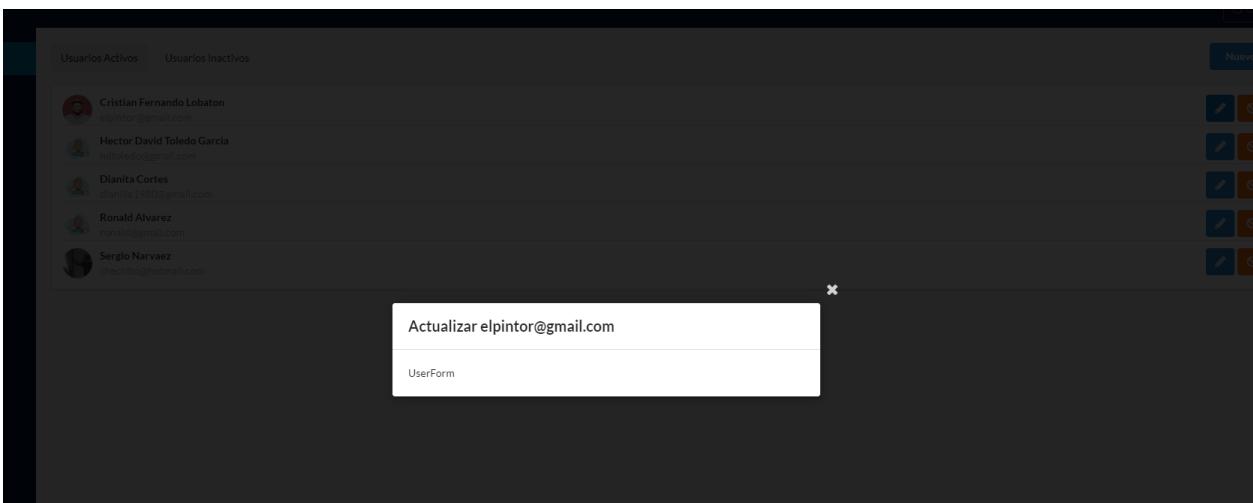
Y ahora nuestra función se la aplicamos a nuestro botón:

```

37
38   <div>
39     <Button icon primary onClick={openUpdateuser}>
40       <Icon name="pencil"/>
41     </Button>
42     <Button icon color={user.active ? "orange" : "teal"}>
43       <Icon name={user.active ? "ban" : "check"/>
44     </Button>
45     <Button icon color="red">
46       <Icon name="trash"/>

```

Al momento de pinchar el botón de uno de nuestros usuarios nos saldrá el modal con el título del usuario:



Ahora vamos a añadir el formulario para que nos empiece a salir la información del usuario, hacemos la importación del formulario `userForm`:

```
src > components > Admin > Users > UserItem > UserItem.jsx > ...
1  import React, { useState } from 'react'
2  import { Image, Button, Icon, Confirm } from "semantic-ui-react"
3  import { image } from "../../../../../assets"
4  import { ENV } from "../../../../../utils"
5  import { BasicModal } from "../../../../../shared"
6  import { UserForm } from "../UserForm"
7  import "./UserItem.scss"
8
9  export function UserItem(props) {
10
11    return (
12      </div>
13      </div>
14
15      <BasicModal show={showModal} close={onOpenCloseModal} title={titleModal}>
16        <UserForm close={onOpenCloseModal} onReload={() => console.log("RELOAD")} user={user}/>
17      </BasicModal>
18    </>
19  );
20}
21
```

Y hacemos la implementación de nuestro formulario pasando los props:

```
49
50    </div>
51
52    <BasicModal show={showModal} close={onOpenCloseModal} title={titleModal}>
53      <UserForm close={onOpenCloseModal} onReload={() => console.log("RELOAD")} user={user}/>
54    </BasicModal>
55  </>
56
57  ];
58}
```

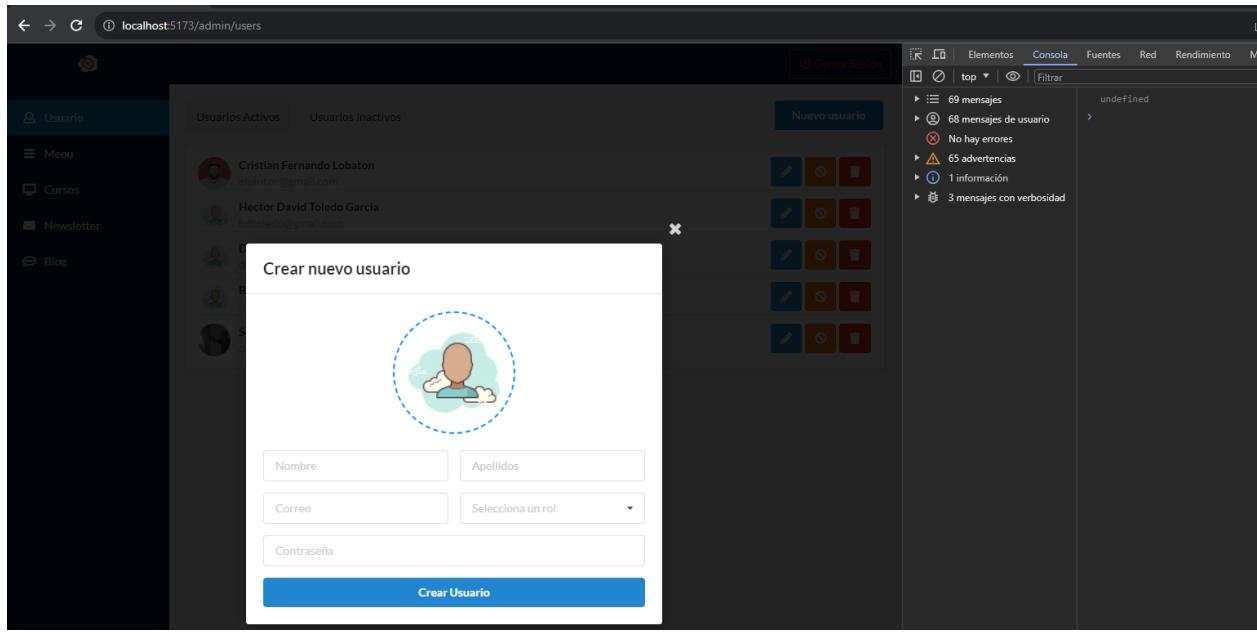
Ahora vamos a verificar con un log de `user` dentro de nuestro `UserForm.jsx` para ver que nos trae:

```
src > components > Admin > Users > UserForm > UserForm.jsx > UserForm
11  const userController = new User()
12
13  export function UserForm(props) {
14
15    const { close, onReload, user } = props
16
17    const { accessToken } = useAuth()
18
19    console.log(user)
20
21    const formik = useFormik({
22      initialValues: initialValues(),
23      validationSchema: validationSchema(),
```

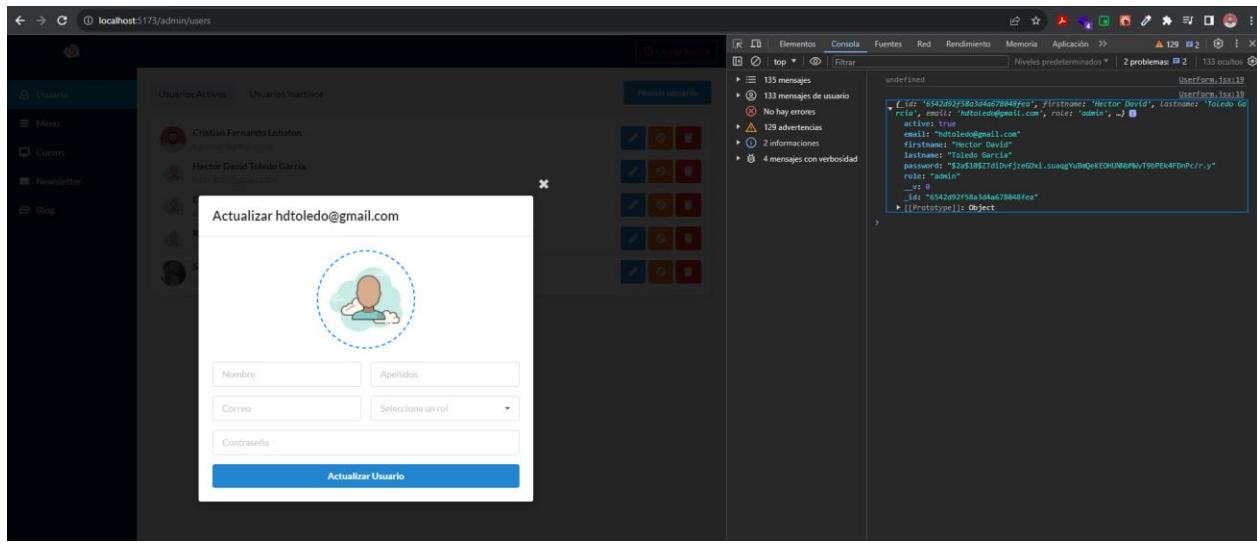
Si vamos a revisar en el navegador y presionamos nuevo usuario obtenemos lo siguiente:



@hdtoledo



Undefined, porque es un nuevo usuario, pero si pinchamos sobre el botón de editar vamos a observar:



Nos trae los datos de nuestro usuario directamente ahora tenemos que pasar los datos de nuestro usuario a nuestro formulario, vamos a adaptar este formulario para que nos muestre los initialValues, para ello vamos a realizar lo siguiente:

```

18
19
20      const formik = useFormik({
21        initialValues: initialValues(user),
22        validationSchema: validationSchema(user),
23        validateOnChange: false,
24        onSubmit: async (formValue) => {
25          try {
    
```



Pasamos nuestro user por los valores iniciales y el validation schema, y nos dirigimos a nuestro **UserForm.form.jsx** y pasamos nuestro user y hacemos unas validaciones:

```
src > components > Admin > Users > UserForm > UserForm.form.jsx > validationSchema
1 import * as Yup from "yup"
2
3 export function initialValues(user) {
4   return {
5     avatar: user?.avatar || "",
6     fileAvatar: null,
7     firstname: user?.firstname || "",
8     lastname: user?.lastname || "",
9     email: user?.email || "",
10    role: user?.role || "",
11    password: ""
12  }
13}
14
15 export function validationSchema(user) {
16   return Yup.object({
17     firstname: Yup.string().required(true),
18     lastname: Yup.string().required(true),
19     email: Yup.string().email(true).required(true),
20     role: Yup.string().required(true),
21     password: user ? Yup.string() : Yup.string().required(true),
22   })
23}
```

De esta manera le estamos pasando la información de nuestro usuario y le indicamos que si contiene algo user pues nos muestre y sino que se setean en blanco, para la validación le indicamos que si la contraseña trae algo que nos la envie.

Ahora continuamos con nuestro **UserForm.jsx**, y vamos a agregar una condicional:

```
20   validateOnChange: false,
21   onSubmit: async (formValue) => {
22     try {
23       if (!user) {
24         await UserController.createUser(accessToken, formValue)
25       } else {
26         console.log("UPDATE")
27       }
28       onReload()
29       close()
30     } catch (error) {
31       console.error(error)
32     }
33   }
34 }
35 }
```

Si user no trae nada es porque lo va a crear y en caso contrario vamos a pasar un log con update por el momento, ahora vamos a importar ENV para continuar con nuestra próxima modificación:



```
src > components > Admin > Users > UserForm > UserForm.jsx > ...
1 import React, { useCallback } from 'react'
2 import { Form, Image } from "semantic-ui-react"
3 import { useFormik } from "formik"
4 import { useDropzone } from "react-dropzone"
5 import { User } from "../../../../../api"
6 import { useAuth } from "../../../../../hooks"
7 import { image } from "../../../../../assets"
8 import { ENV } from "../../../../../utils"
9 import { initialValues, validationSchema } from "./UserForm.form"
10 import "./UserForm.scss"
11
```

A través de nuestro nuestro getAvatar:

```
src > components > Admin > Users > UserForm > UserForm.jsx > getAvatar
47
48     const getAvatar = () => {
49         if (formik.values.fileAvatar) {
50             return formik.values.avatar
51         } else if(formik.values.avatar) {
52             return `${ENV.BASE_PATH}/${formik.values.avatar}`
53         }
54         return image.noAvatar
55     }
56
```

Con esta condición agregada vamos a traer la ruta de nuestro avatar, vamos a verificar en nuestro navegador:



@hdtoledo

MERN App

localhost:5173/admin/users

Cerrar Sesión

Usuario

Menu

Cursos

Newsletter

Blog

Usuarios Activos

Usuarios Inactivos

Nuevo usuario

Cristian Fernando Lobaton
elpintor@gmail.com

Hector David Toledo Garcia
hdtoledo@gmail.com

Actualizar elpintor@gmail.com

Cristian Fernando Lobaton

elpintor@gmail.com Administrador

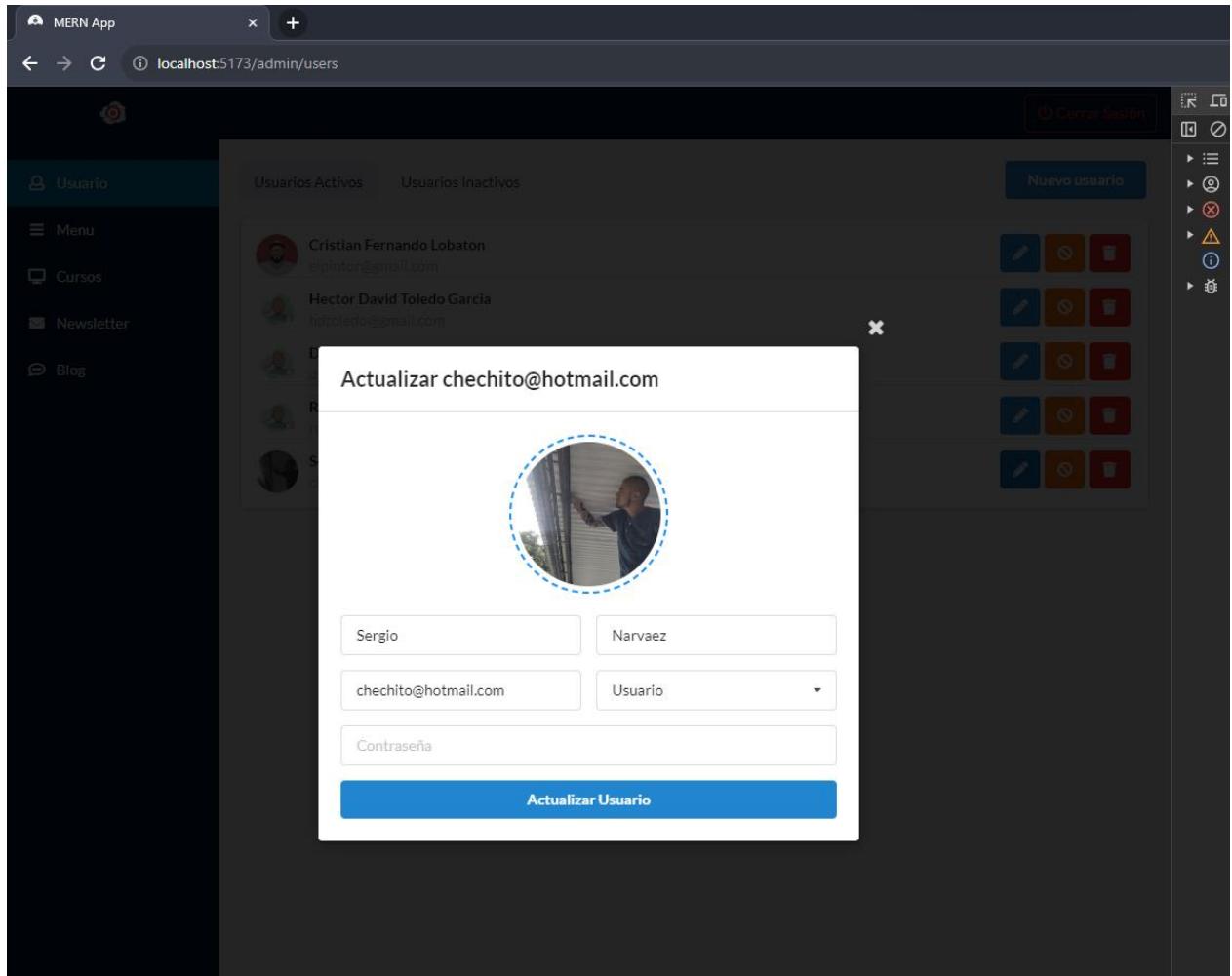
Contraseña

Actualizar Usuario

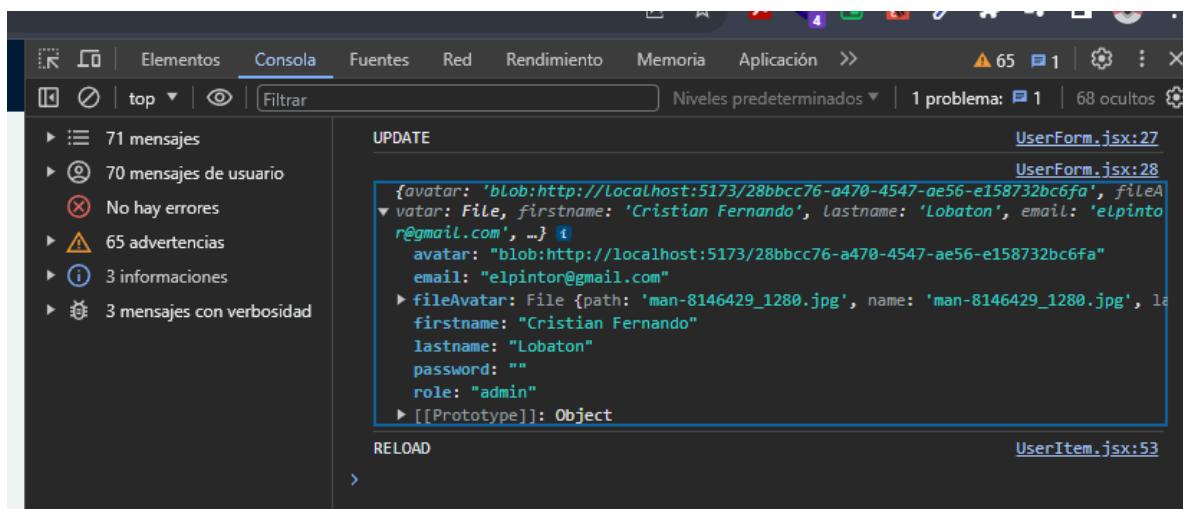
The screenshot displays a web-based administration interface for a MERN application. On the left, a sidebar menu includes 'Usuario', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area shows two tabs: 'Usuarios Activos' and 'Usuarios Inactivos', with 'Activos' selected. It lists two users: 'Cristian Fernando Lobaton' (email: elpintor@gmail.com) and 'Hector David Toledo Garcia' (email: hdtoledo@gmail.com). A modal window is centered over the page, titled 'Actualizar elpintor@gmail.com', containing a placeholder profile picture of a person with a beard and sunglasses, and input fields for the user's name ('Cristian Fernando', 'Lobaton'), email ('elpintor@gmail.com'), and role ('Administrador'). A password field and a blue 'Actualizar Usuario' button are also present. The background shows a dark theme with a grid of icons.



@hdtoledo



De esta manera ya nos esta funcionando correctamente, ahora vamos a agregar un log de `formValue` tambien y vamos a intentar modificar un usuario:



Al presionar actualizar usuario nos envia el log de los datos completos, ahora si vamos a realizar la actualización.



@hdtoledo

ACTUALIZANDO EL USUARIO

Nos ubicamos en `/api/user.jsx` y vamos a crear nuestro `updateUser`:

```
src > api > user.jsx > User > updateUser
1
2  async updateUser(token, idUser, userData) {
3    try {
4      const data = userData
5      if(!data.password) {
6        delete data.password
7      }
8
9      const formData = new FormData()
10     Object.keys(data).forEach((key) => {
11       formData.append(key, data[key])
12     })
13
14     if (data.fileAvatar) {
15       formData.append("avatar", data.fileAvatar)
16     }
17
18     const url = `${ENV.BASE_API}/${ENV.API_ROUTES.USER}/${idUser}`
19     const params = {
20       method: "PATCH",
21       headers: {
22         Authorization: `Bearer ${token}`
23       },
24       body: formData,
25     }
26
27     const response = await fetch(url, params)
28     const result = await response.json()
29
30     if(response.status !== 200) throw result
31     return result
32   }
33   catch (error) {
34     throw error
35   }
36 }
37 }
```

Es una función muy similar a la de nuestro `createUser` solo que esta vez enviamos por patch y verificamos los cambios de los campos a través de condiciones.

Vamos ahora a modificar nuestros logs por la función creada:

```
21   validateOnChange: false,
22   onSubmit: async (formValue) => {
23     try {
24       if (!user) {
25         await userController.createUser(token, formValue)
26       } else {
27         await userController.updateUser(token, user._id, formValue)
28       }
29       onReload()
30       close()
31     } catch (error) {
32       console.error(error)
33     }
34 }
```

Y ya nos queda probar que realmente este funcionando, llenamos una actualización y la enviamos:



@hdtoledo

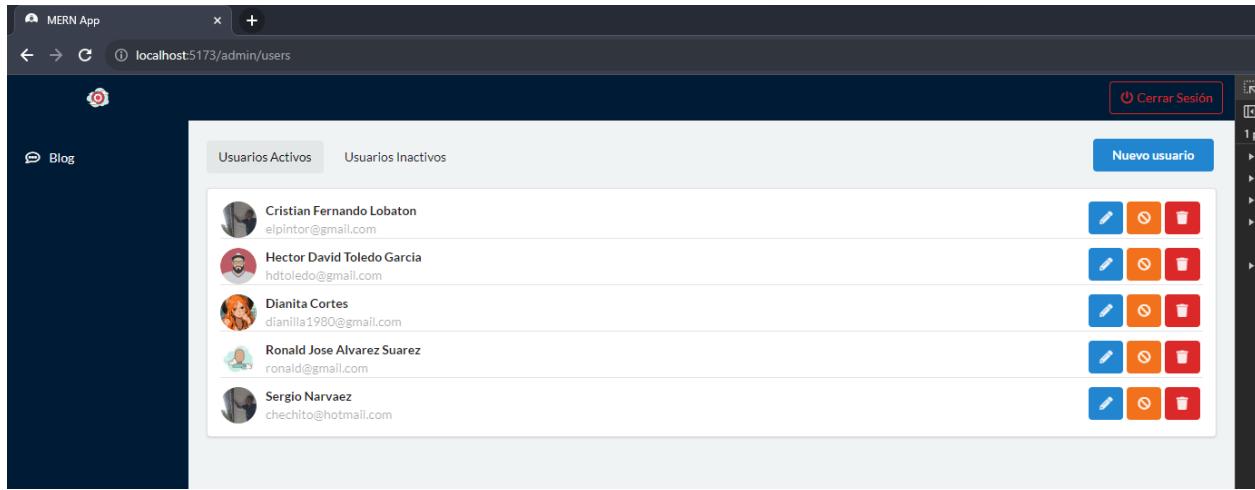
The screenshot shows a user profile update interface. The user's current profile picture is a placeholder image of a man painting a wall. The form fields are pre-filled with the user's information: first name 'Cristian Fernando', last name 'Lobaton', email 'elpintor@gmail.com', role 'Administrador', and a password field. A blue 'Actualizar Usuario' button is at the bottom.

Notamos que nos falta la recarga de la data pero ahora la configuramos, refrescamos y nos debe salir la actualización del avatar en mi caso:

The screenshot shows the user list page after refreshing. The user profile picture has been successfully updated to the one from the previous screenshot, showing a man painting a wall.

Probamos nuevamente con otros datos de usuario y verificamos el login con uno de nuestros usuarios para validar que no se este cambiando la contraseña:





Ahora para realizar la recarga de nuestra data vamos a realizar lo siguiente en **Users.jsx** agregando nuestro **onReload**:

```
src > pages > admin > Users > 🌐 Users.jsx > 📁 Users > 📄 panes > 🏷 render
14
15  const panes = [
16    {
17      menuItem: "Usuarios Activos",
18      render: () => (
19        <Tab.Pane attached={false}>
20          <ListUsers usersActive={true} reload={reload} onReload={onReload} />
21        </Tab.Pane>
22      )
23    },
24    {
25      menuItem: "Usuarios Inactivos",
26      render: () => (
27        <Tab.Pane attached={false}>
28          <ListUsers usersActive={false} reload={reload} onReload={onReload} />
29        </Tab.Pane>
30      )
31    },
32  ],
33]
```

Y en nuestro **ListUsers.jsx** pasamos en props **onReload**:

```
src > components > Admin > Users > ListUsers > 🌐 ListUsers.jsx > 📁 ListUsers > 📄 onReload
9
10  export function ListUsers(props) {
11
12    const { usersActive, reload, onReload } = props
13    const [users, setUsers] = useState(null)
14    const { accessToken } = useAuth()
15
16    useEffect(() => {
17      async () => {
18        try {
```

Y se lo pasamos a nuestro userItem:



```

27     if(!users) return <Loader active inline="centered" />
28     if(size(users) === 0) return "No hay ningun usuario"
29
30     return map(users, (user) => <UserItem key={user._id} user={user} onReload={onReload} />)
31
32   }
33

```

Y ahora en nuestro **UserItem.jsx** tambien se lo pasamos:

```

src > components > Admin > Users > Useritem > Useritem.jsx > Useritem > onReload
6   import { UserForm } from "../UserForm"
7   import "./UserItem.scss"
8
9   export function UserItem(props) {
10     const { user, onReload } = props
11     const [showModal, setShowModal] = useState(false)
12     const [titleModal, setTitleModal] = useState("")
13
14     const onOpenCloseModal = () => setShowModal((prevState) => !prevState)

```

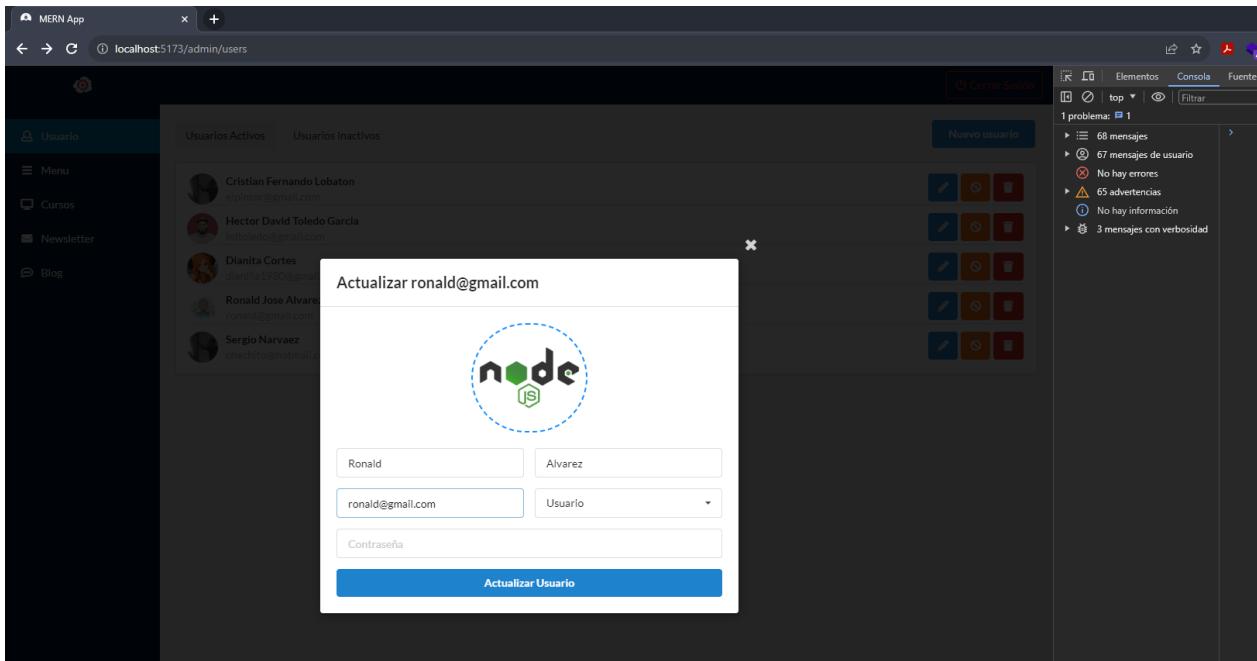
Y lo implementamos así:

```

51
52     <BasicModal show={showModal} close={onOpenCloseModal} title={titleModal}>
53       <UserForm close={onOpenCloseModal} onReload={onReload} user={user}/>
54     </BasicModal>
55   </>
56 }
57
58

```

Ahora revisemos si realmente funciona nuestro onReload:



The screenshot shows a web application interface for managing users. On the left is a sidebar with navigation links: 'Usuario' (selected), 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos'. Under 'Usuarios Activos', there is a list of five users with their names, emails, and small profile icons. To the right of each user entry are three small blue buttons with white icons: a pencil, a circular arrow, and a trash can. A red rectangular box highlights the user 'Hector David Toledo Garcia' (email: hdtoledo@gmail.com). At the top right of the main content area is a red button labeled 'Nuevo usuario'. On the far right, there is a vertical sidebar with a '1 problema' notification and several small icons.

Y observamos que ya se está refrescando correctamente los datos de nuestro usuario.

DESACTIVANDO Y ACTIVANDO USUARIOS

Ahora vamos a realizar la función de nuestro botón de activar/desactivar un usuario, para ello nos vamos a ubicarnos en **UserItem.jsx** y vamos a crear un par de estados:

```
src > components > Admin > Users > UserItem > UserItem.jsx > UserItem
  9  export function UserItem(props) {
10    const { user, onReload } = props
11
12    const [showModal, setShowModal] = useState(false)
13    const [titleModal, setTitleModal] = useState("")
14
15    const [showConfirm, setShowConfirm] = useState(false)
16    const [confirmMessage, setConfirmMessage] = useState("")
17    const [isDelete, setIsDelete] = useState(false)
18
19
20    const onOpenCloseModal = () => setShowModal((prevState) => !prevState)
21
```

En donde vamos a configurar nuestro mensaje, la confirmación y si se elimina, ahora cargamos en **onOpenCloseConfirm** el modal:

```
19
20    const onOpenCloseModal = () => setShowModal((prevState) => !prevState)
21    const onOpenCloseConfirm = () => setShowConfirm((prevState) => !prevState)
22
23    const openUpdateuser = () => {
24      setTitleModal(`Actualizar ${user.email}`)
```



@hdtledo

Y ahora vamos a colocar nuestro componente Confirm de la siguiente manera:

```
72      </BasicModal>
73
74  <Confirm open={showConfirm} onCancel={onOpenCloseConfirm} onConfirm={isDelete ? () => console.log("Eliminado") : onActivateDesactivate} content={confirmMessage} size="mini"/>
75
76  );
77
78
```

Vamos a crear la base de nuestra función **onActivateDesactivate**:

```
src > components > Admin > Users > UserItem > UserItem.jsx > UserItem > onActivateDesactivate
22
23  const openUpdateUser = () => {
24    setTitleModal('Actualizar ${user.email}')
25    onOpenCloseModal()
26  }
27
28  const onActivateDesactivate = async () => {
29    console.log("Activar o desactivar Usuario")
30  }
31
32
33  return (
34    <>
```

Y la aplicamos en nuestro Confirm:

```
73      </BasicModal>
74  <Confirm open={showConfirm} onCancel={onOpenCloseConfirm} onConfirm={isDelete ? () => console.log("Eliminado") : onActivateDesactivate} content={confirmMessage} size="mini"/>
75
76  );
77
78
```

Ahora creamos la función para que al momento de darle clic nos permita confirmar si se va a activar/desactivar:

```
25
26  onOpenCloseModal()
27
28  const openDesactivateActivateConfirm = () => {
29    setIsDelete(false)
30    setConfirmMessage(user.active ? 'Desactivar Usuario ${user.email}' : 'Activar Usuario ${user.email}')
31  }
32
33  const onActivateDesactivate = async () => {
34    console.log("Activar o desactivar Usuario")
35  }
36
```

Para que nuestra función se ejecute debemos darle clic a nuestro botón de activar/desactivar y añadimos nuestra función **openDesactivateActivateConfirm**:

```
58
59
60  <Button icon primary onClick={openUpdateUser}>
61    <Icon name="pencil"/>
62  </Button>
63  <Button icon color={user.active ? "orange" : "teal"} onClick={openDesactivateActivateConfirm}>
64    <Icon name={user.active ? "ban" : "check"} />
65  </Button>
66  <Button icon color="red">
67    <Icon name="trash"/>
68  </Button>
69
70  </div>
71
72  </div>
```

Y ahora nos quedaría ejecutar nuestro **onOpenCloseConfirm**:



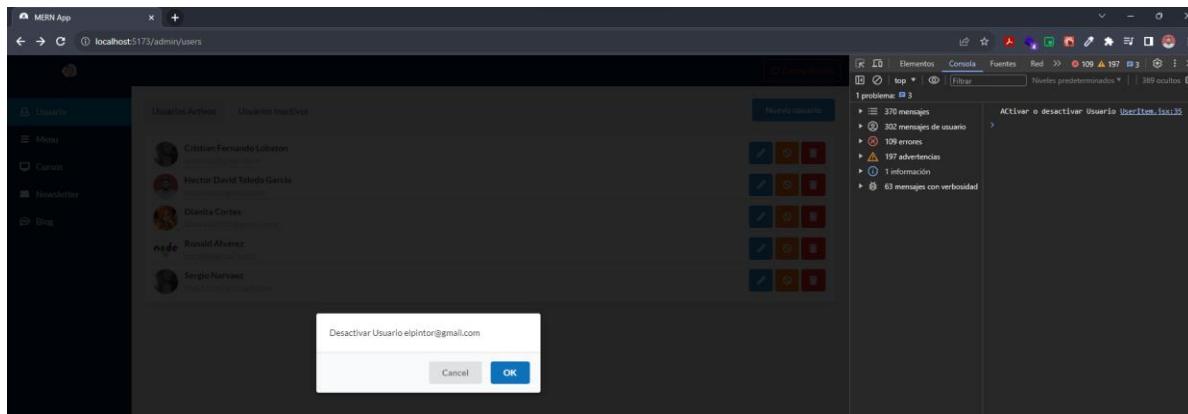
@hdtoledo

```

27
28     const openDesactivateActivateConfirm = () => {
29         setIsDelete(false)
30         setConfirmMessage(user.active ? 'Desactivar Usuario ${user.email}' : 'Activar Usuario ${user.email}')
31         onOpenCloseConfirm()
32     }
33
34     const onActivateDesactivate = async () => {

```

Y si nos vamos a nuestro navegador y damos clic en el botón nos va a salir la opción de confirmar, al darle nos envía un log:



Si pulsamos sobre cancelar nos debe cancelar la opción de nuestro mensaje de confirmación. Ahora vamos a implementar la lógica para poder enviar el activáte, lo primero es hacer la importación de User de api y nuestro useAuth:

```

src > components > Admin > Users > UserItem > UserItem.jsx > ...
1  import React, { useState } from 'react'
2  import { Image, Button, Icon, Confirm } from "semantic-ui-react"
3  import { image } from "../../../../../assets"
4  import { User } from "../../../../../api"
5  import { useAuth } from "../../../../../hooks"
6  import { BasicModal } from "../../../../../shared"
7  import { ENV } from "../../../../../utils"
8  import { UserForm } from "../UserForm"
9  import "./UserItem.scss"
10

```

Y ahora los implementamos de la siguiente manera User y useAuth:

```

src > components > Admin > Users > UserItem > UserItem.jsx > UserItem
9   import "./UserItem.scss"
10
11  const userController = new User()
12
13  export function UserItem(props) {
14      const { user, onReload } = props
15      const { accessToken } = useAuth()
16
17      const [showModal, setShowModal] = useState(false)

```

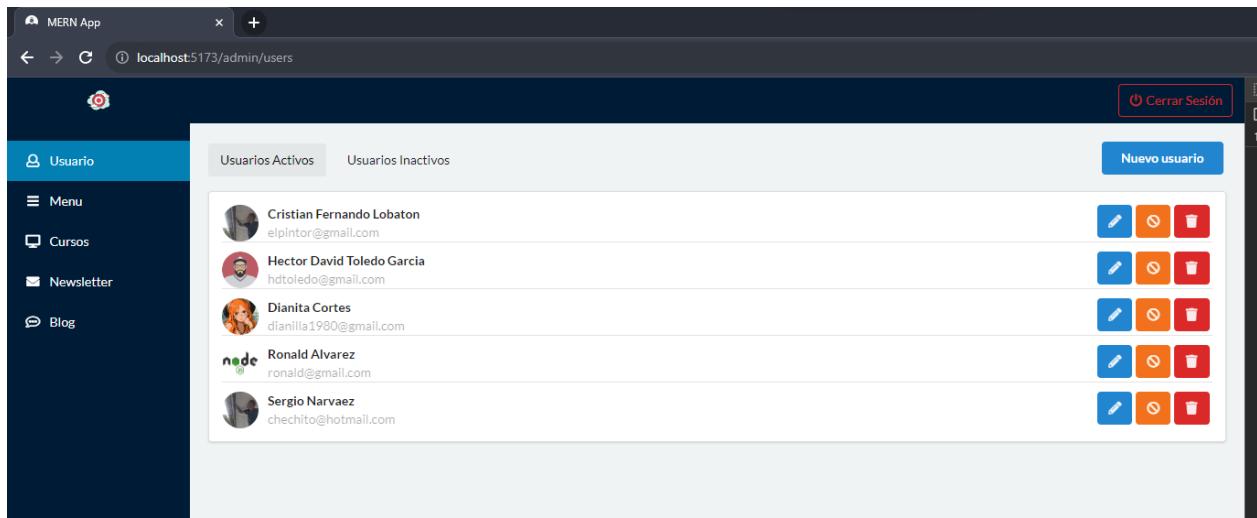


@hdtoledo

Vamos a realizar un **tryCatch** en la función **onActivateDesactivate**:

```
src > components > Admin > Users > UserItem > UserItem.jsx > UserItem > onActivateDesactivate
38
39  const onActivateDesactivate = async () => {
40    try {
41      await userController.updateUser(accessToken, user._id, {
42        active: !user.active,
43      })
44      onReload()
45    } catch (error) {
46      onOpenCloseConfirm()
47      console.error(error)
48    }
49  }
50
```

Ahora llego el momento de verificar si esta funcionando, para ello nos vamos al navegador y vamos a realizar el cambio de estado:



The screenshot shows a web application interface for managing users. On the left, there is a sidebar with navigation links: 'Usuario' (selected), 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area has a header with 'localhost:5173/admin/users' and a 'Cerrar Sesión' button. Below the header, there are two tabs: 'Usuarios Activos' (selected) and 'Usuarios Inactivos'. A blue button labeled 'Nuevo usuario' is located in the top right of the main content area. The main content displays a list of active users with their names, emails, and three small icons (edit, delete, and another unknown icon). The user list includes:

User	Email	Action Icons
Cristian Fernando Lobaton	elplintor@gmail.com	[Edit, ?]
Hector David Toledo Garcia	hdtoledo@gmail.com	[Edit, ?]
Dianita Cortes	dianita1980@gmail.com	[Edit, ?]
Ronald Alvarez	ronald@gmail.com	[Edit, ?]
Sergio Narvaez	chechito@hotmail.com	[Edit, ?]



The screenshot shows the MERN App's administration interface for users. On the left, a sidebar menu includes 'Usuario', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main content area displays two tabs: 'Usuarios Activos' and 'Usuarios Inactivos'. Under 'Usuarios Activos', five users are listed: Cristian Fernando Lobaton (elpintor@gmail.com), Hector David Toledo Garcia (hdtoledo@gmail.com), Dianita Cortes (dianita1980@gmail.com), Ronald Alvarez (ronald@gmail.com), and Sergio Narvaez (chechito@hotmail.com). To the right of each user are three small icons for edit, deactivate, and delete. A modal dialog box is centered over the user list, prompting 'Desactivar Usuario elpintor@gmail.com' with 'Cancel' and 'OK' buttons.

This screenshot shows the same MERN App administration interface after the user 'elpintor@gmail.com' has been deactivated. The 'Usuarios Activos' tab is now selected, and the user 'Cristian Fernando Lobaton' is no longer visible in the list. The other four users remain active.

El usuario paso a ser inactivo, ahora le doy clic a los inactivos:



The screenshot shows the 'User' management section of the MERN App. On the left is a sidebar with navigation links: 'Usuario', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main area has tabs for 'Usuarios Activos' (Active Users) and 'Usuarios Inactivos' (Inactive Users). Under 'Usuarios Activos', there is a list of six users with their names, email addresses, and profile icons:

User	Email
Cristian Fernando Lobaton	el pintor@gmail.com
Jimmy Lombana	jimmy.lombana@gmail.com
juan rincon	juandi@gmail.com
Juan Henao	henao@gmail.com
Juan Carlos Henao Rubio	elrubio@gmail.com
Jose Ramirez	joselito@gmail.com

To the right of each user entry are three small icons: a blue pencil for edit, a green checkmark for status, and a red trash can for deletion.

Y vuelvo a activarlo:

The screenshot shows the same 'User' management section, but a modal dialog box is overlaid on the screen. The dialog is titled 'Activar Usuario' and contains the text '@el pintor@gmail.com'. It has two buttons at the bottom: 'Cancel' and 'OK'. The background of the page is dimmed to indicate the modal is active.



@hdtoledo

The screenshot shows the 'User' management section of the MERN App. On the left, a sidebar menu includes 'Usuario', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main area has tabs for 'Usuarios Activos' (selected) and 'Usuarios Inactivos'. A blue button 'Nuevo usuario' is at the top right. Below are five user entries:

User	Email	Action Buttons
Jimmy Lombana	jimmy.lombana@gmail.com	[Edit, Checkmark, Delete]
juan rincon	juandi@gmail.com	[Edit, Checkmark, Delete]
Juan Henao	henao@gmail.com	[Edit, Checkmark, Delete]
Juan Carlos Henao Rubio	elrubio@gmail.com	[Edit, Checkmark, Delete]
Jose Ramirez	joselito@gmail.com	[Edit, Checkmark, Delete]

This screenshot shows the same 'User' management section after a reload. The user list has been updated to show different individuals:

User	Email	Action Buttons
Cristian Fernando Lobaton	elpintor@gmail.com	[Edit, Checkmark, Delete]
Hector David Toledo Garcia	hdtoledo@gmail.com	[Edit, Checkmark, Delete]
Dianita Cortes	dianilla1980@gmail.com	[Edit, Checkmark, Delete]
Ronald Alvarez	ronald@gmail.com	[Edit, Checkmark, Delete]
Sergio Narvaez	chechito@hotmail.com	[Edit, Checkmark, Delete]

Nuestro **onReload** funciona correctamente y vemos como se cambia el estado del usuario automáticamente.



@hdtoledo

ELIMINANDO EL USUARIO

Ahora nos queda solo la parte de eliminar el usuario y lo que vamos a realizar será muy similar a lo que hicimos anteriormente con activar/desactivar, para ello nos ubicamos en **UserItem.jsx** y vamos a crear **openDeleteConfirm**:

```
src > components > Admin > Users > UserItem > UserItem.jsx > UserItem > openDeleteConfirm
46     } catch (error) {
47       console.error(error)
48     }
49   }
50
51   const openDeleteConfirm = () => {
52     setIsDelete(true)
53     setConfirmMessage(`Eliminar usuario ${user.email}`)
54     onOpenCloseConfirm()
55   }
56
57
58   return (
59     <>
```

Y hacemos la implementación de la función en nuestro botón:

```
src > components > Admin > Users > UserItem > UserItem.jsx > UserItem
75
76   <div>
77     <Button icon primary onClick={openUpdateUser}>
78       <Icon name="pencil"/>
79     </Button>
80     <Button icon color={user.active ? "orange" : "teal"} onClick={openDesactivateActivateConfirm}>
81       <Icon name={user.active ? "ban" : "check"} />
82     </Button>
83     <Button icon color="red" onClick={openDeleteConfirm}>
84       <Icon name="trash"/>
85     </Button>
86   </div>
87 </div>
88
89 <BasicModal show={showModal} close={onOpenCloseModal} title={titleModal}>
90   <UserForm close={onOpenCloseModal} onReload={onReload} user={user}/>
91 </BasicModal>
```

Si vamos al navegador y presionamos nuestro botón se activara el mensaje:



@hdtoledo

The screenshot shows a user management application. On the left is a sidebar with links: 'Usuario' (selected), 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos'. A list of users is displayed with columns for name, email, and edit/delete icons. A modal dialog box is centered over the list, asking 'Eliminar usuario chechito@hotmail.com' with 'Cancel' and 'OK' buttons.

Y si presionamos nos saldrá en consola:

The screenshot shows the browser's developer tools open, specifically the 'Consola' tab. It displays a warning message: '1 problema:' followed by a list of log entries. One entry is highlighted in red: 'Eliminado' at line 93 of 'UserItem.jsx'. The other entries are: '4 mensajes', '4 mensajes de usuario', 'No hay errores', '1 advertencia', '1 información', and '2 mensajes con verbosidad'.

Ahora vamos a realizar la función de eliminación nos ubicamos en [/api/user.jsx](#):



```

src > api >  user.js >  User
116     }
117
118     async deleteUser(accessToken, idUser) {
119         try {
120             const url = `${this.baseApi}/${ENV.API_ROUTES.USER}/${idUser}`
121             const params = {
122                 method: "DELETE",
123                 headers: {
124                     Authorization: `Bearer ${accessToken}`,
125                 },
126             }
127
128             const response = await fetch(url, params)
129             const result = await response.json()
130
131             if (response.status !== 200) throw result
132             return result
133
134         } catch (error) {
135             throw error
136         }
137     }
138 }

```

Ahora en nuestro **onDelete** realizamos lo siguiente:

```

src > components > Admin > Users > UserItem >  UserItem.jsx >  UserItem >  onDelete
51     const openDeleteConfirm = () => {
52         setIsDelete(true)
53         setConfirmMessage(`Eliminar usuario ${user.email}`)
54         onOpenCloseConfirm()
55     }
56
57     const onDelete = async () => {
58         try {
59             await userController.deleteUser(accessToken, user._id)
60             onReload()
61             onOpenCloseConfirm()
62         } catch (error) {
63             console.log(error)
64         }
65     }
66

```

Y hacemos la implementación de onDelete en nuestro confirm:

```

100     <UserForm close={onOpenCloseConfirm} onReload={onReload} user={user}/>
101     </BasicModal>
102     <Confirm open={showConfirm} onCancel={onOpenCloseConfirm} onConfirm={isDelete ? onDelete : onActivateDesactivate} content={confirmMessage} size="mini"/>
103   </>
104 ;
105 }
106

```

Ahora realizamos la prueba eliminando los registros:



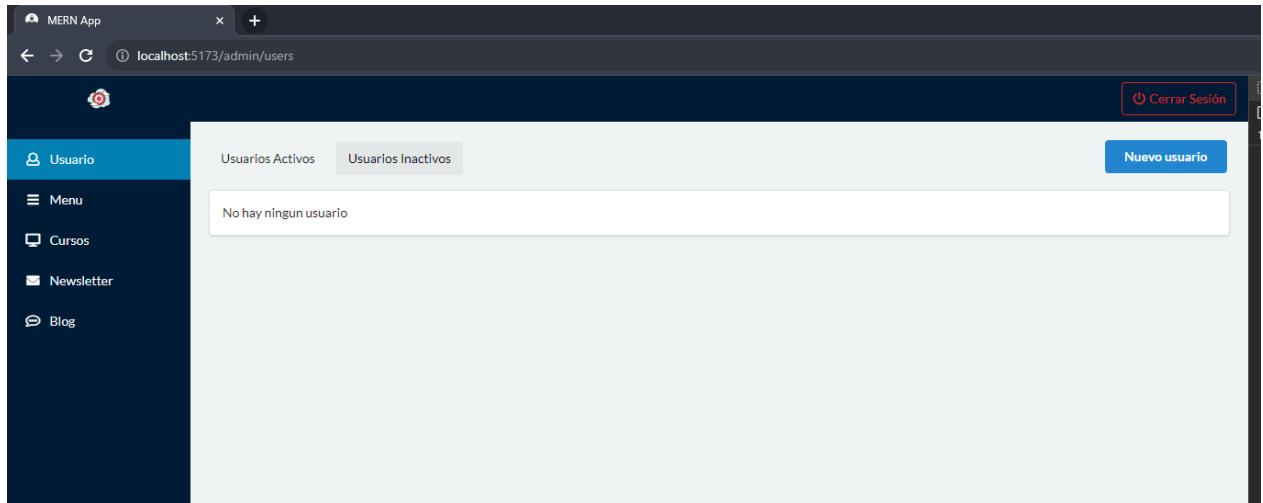
@hdtoledo

The screenshot shows the 'User' management section of the MERN App. On the left is a sidebar with 'User', 'Menu', 'Cursos', 'Newsletter', and 'Blog'. The main area has tabs for 'Usuarios Activos' and 'Usuarios Inactivos'. Under 'Activos', there are five user entries: Jimmy Lombana (jimmy.lombana@gmail.com), juan rincon (juandi@gmail.com), Juan Henao (henao@gmail.com), Juan Carlos Henao Rubio (elrubio@gmail.com), and Jose Ramirez (joselito@gmail.com). Each entry has edit, checkmark, and trash icons. A red 'Nuevo usuario' button is at the top right. A sidebar on the right shows '1 problema:' with three messages: '3 mensajes', '3 mensajes de usuario', and 'No hay errores'.

This screenshot shows the same user management interface. A modal dialog box is centered, asking 'Eliminar usuario jimmy.lombana@gmail.com' with 'Cancel' and 'OK' buttons. The background is dimmed.

The screenshot shows the user list after the deletion of Jimmy Lombana. The 'Activos' tab now lists juan rincon, Juan Henao, Juan Carlos Henao Rubio, and Jose Ramirez. The sidebar and other UI elements remain the same.





De esta manera hemos revisado que funciona correctamente nuestra función eliminar y la parte de usuario ha sido realizada, la lógica de nuestro CRUD se puede implementar en los diferentes menús.



@hdtoledo