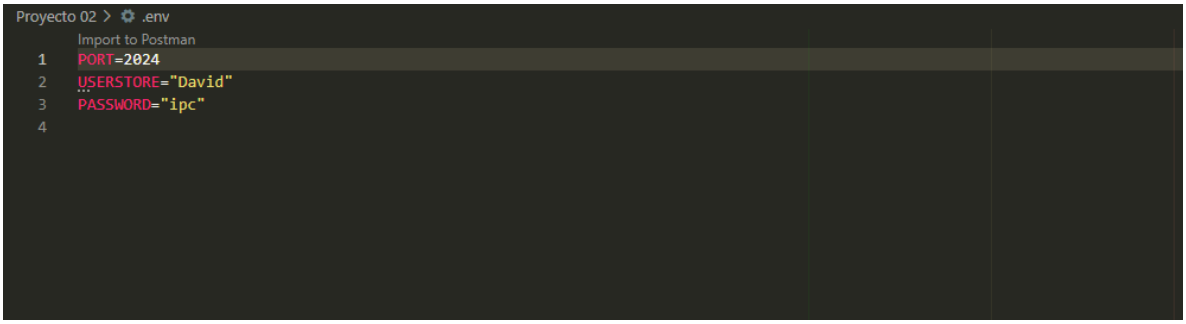
Manual tecnico

Archivo .env

Para la creacion de nuestro BackEnd es necesario la creación de un archivo .env, donde ubicaremos datos como el puerto del backend, el nombre de usuario y contraseña.



Config.js

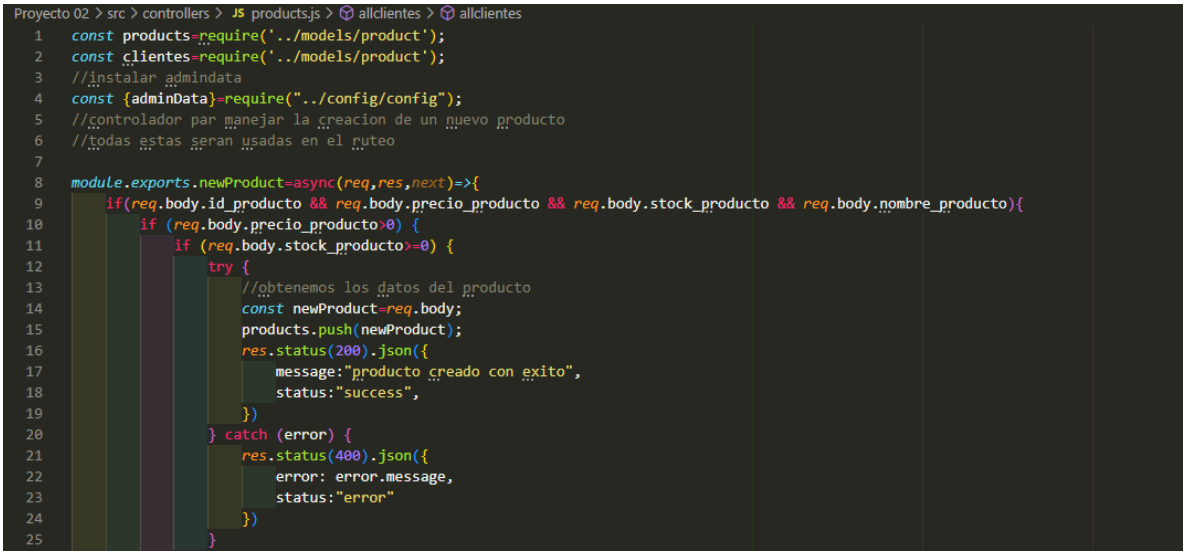
Dentro de este archive haremos el llamado de nuestras variables creadas en el archivo .env, esto para obtener el puerto del mismo, y datos de administrador como usuario y contraseña que nos será útil adelante.

Models

Dentro de este archivo crearemos dos arreglos para almacenar datos que solicitara desde el frontend para la creación del producto, estos serán exportados como módulos.

Controllers

Dentro de controllers podremos usar los datos insertados del frontend para ser guardados en los arreglos antes mencionados, para esto podremos agregar parámetros para que los productos sean ingresados de la manera correcta.



Endpoint en backend

Ubicando el endpoint en el archivo “store.js”, guardara los datos del login en un archivo diferente llamado “producto.js” que se encuentra en otra carpeta, usamos la ruta POST, que envia información al frontend y es recibida en el backend.

Store.js

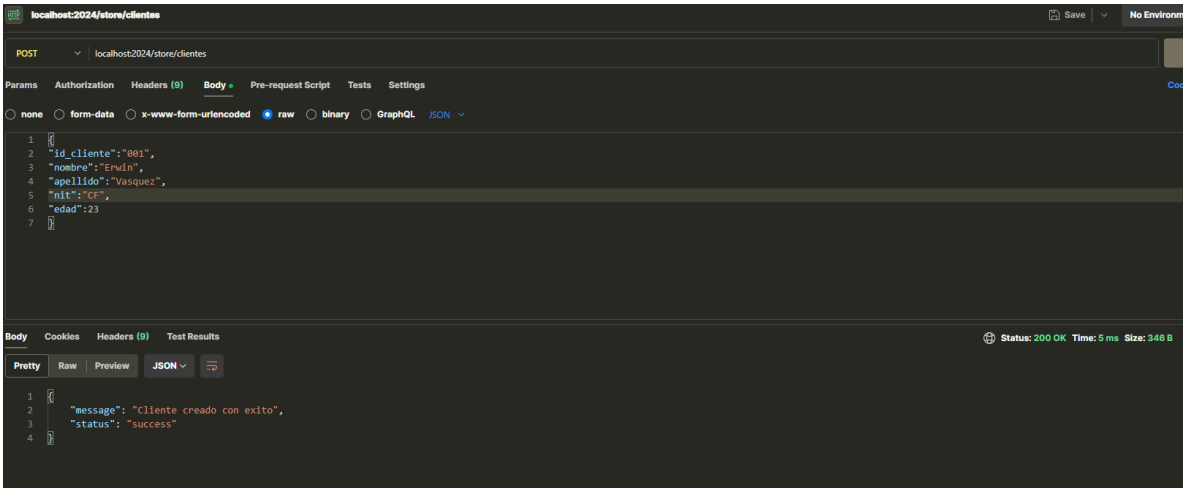
Este archivo nos permitirá demostrar los datos enviados y recibidos, desde aquí podríamos decir que se hace el llamado alas productos y clientes para ser mostrados tambien en el frontend.

Index.js

Una vez tengamos Nuestro archivo env. Podremos iniciar con nuestro backend con la creación de un archivo index.js, donde llamaremos a nuveos archivos donde llamaremos las rutas necesarios para el seguimiento de nuestro programa. A su vez podemos mandar un mensaje sobre el funcionamiento de nuestro puerto.

POSTMAN

Usamos postman para verificar que el backend funcione de la manera correcta.



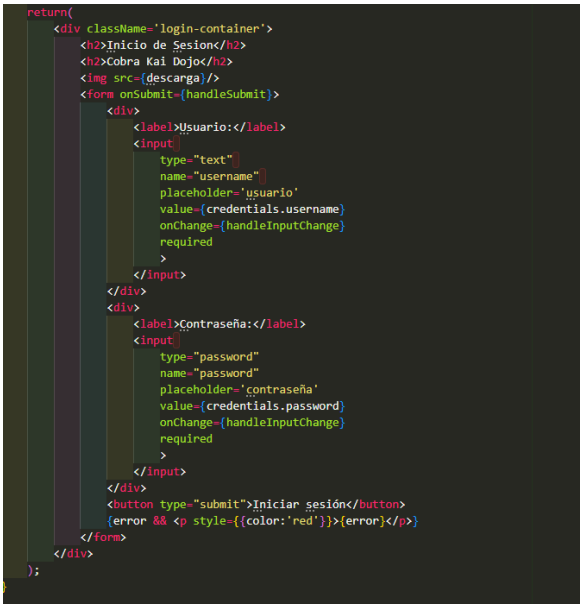
Petición de frontend hacia backend

Creando una variable constante llamada “handlesubmit” para llamar a un evento, en donde codificamos un try para esperar una respuesta con “await” donde esperaremos un llamado hacia el backend, en nuestro endpoint donde tendremos nuestro login, y guardando los datos en credentials, que es donde guardaremos nuestros datos de inicio de sesión(usuario y contraseña”, para posteriormente ser comparados con un “if” los datos ya preestablecidos en el backend.

```
const handleSubmit=async(e)=>{
    e.preventDefault();
    try{
        const response = await axios.post('http://localhost:2024/store/login',credentials);
        if(response.data.status==="success"){
            await Swal.fire({
                title:'<span style="color:#A0A4B4;">Exito</span>',
                html:'<span style="color:#A0A4B4;">Inicio de sesion exitoso</span>',
                icon:"success",
                confirmButtonText:"Cerrar",
                background:"#222",
                confirmButtonColor:"blue",
            })
            navigate('/dashboard');
        }
```

Renderizado componente Login

Una vez hecho esto, nuestro programa retornara con un return, que desplegara el código html que podremos visualizar en pantalla, con el titulo de la pagina y los datos que solicita el login, guardando los datos para ser comparados en el backend cuando el boton de “iniciar sesión” sea oprimido.



```
return(
    <div className='login-container'>
        <h2>Inicio de Sesion</h2>
        <h2>Cobra Kai Dojo</h2>
        <img src={descarga}/>
        <form onSubmit={handleSubmit}>
            <div>
                <label>Usuario:</label>
                <input
                    type="text"
                    name="username"
                    placeholder='usuario'
                    value={credentials.username}
                    onChange={handleInputChange}
                    required
                    >
                </input>
            </div>
            <div>
                <label>Contraseña:</label>
                <input
                    type="password"
                    name="password"
                    placeholder='contraseña'
                    value={credentials.password}
                    onChange={handleInputChange}
                    required
                    >
                </input>
            </div>
            <button type="submit">Iniciar sesión</button>
            {error && <p style={{color:'red'}}>{error}</p>}
        </form>
    </div>
);
}
```

Login.css

Creamos un archivo css para personalizar nuestro login, dentro de este podremos añadirle colores al texto y los bloques de texto.

```
src > pages > # Login.css > .login-container h2
body{
    font-family: Arial,sans-serif;
    background-color: cyan;
    display: flex;
    justify-content: center;
    align-items: center;
}

.login-container{
    background-color: #CD5C5C;
    padding: 2rem 2.5rem;
    border-radius: 8px;
    width: 100%;
    max-width: 800px;
}
.login-container h2{
    text-align: center;
    color: white;
    font-size: 50px;
}
.login-container label{
    display: block;
    margin-bottom: 0.5px;
    color: aqua;
    font-weight: bold;
    font-size: 18px;
}
```

Importaciones en nuestro Dashboard

Importamos las librerías necesarias para el perfecto funcionamiento de nuestro programa, entre ellas react para hacer uso de algunas funciones útiles como efectos visuales, swal para mandar alarmas al usuario, el dashboard.css donde tendremos las personalizaciones de la pagina web, table para agregar la tabla donde se almacenaran los productos y clientes.

```
import React, {useEffect, useState} from 'react';
import Swal from 'sweetalert2';
import 'bootstrap/dist/css/bootstrap.min.css';
import './Dashboard.css';

import Table from 'react-bootstrap/Table';
import {Bar,Pie} from 'react-chartjs-2';
import {
    Chart as ChartJS,
    CategoryScale,
    LinearScale,
    BarElement,
    ArcElement,
    Title,
    Tooltip,
    Legend
} from 'chart.js'
```

Dashboard

Al inico de nuestro bloque de opciones, asignaremos variables constanes que servirán para el envio y recibmiento de datos como productos y clientes, así como la creación de datos para la comparación de datos para las graficas.

```
const Dashboard=()=>{
    const [productBody, setProductBody]=useState('');//contenido del nuevo producto
    const [ClienteBody, setClienteBody]=useState('');//contenido del nuevo cliente
    //UseState para administrar los productos que obtengamos de nuestro backend
    const[products,setProducts]=useState([]);
    const[clientes,setclients]=useState([]);
    // Datos de mi gráfica de barras de productos por precio
    const[barChartData,setBarChartData]=useState(null);
    // Datos de mi gráfica de pie (Productos con precio mayor a 100 vs productos con precio menor/igual a 100)
    const[pieChartData,setPieChartData]=useState(null);
    //funcion que maneja el cambio de texto en el textarea
    const handleChange=(event)=>{
        setProductBody(event.target.value);
    };
    const handleChange2=(event)=>{
        setClienteBody(event.target.value);
    };
    //funcion que maneja el envio de mi producto hacia el backend
    const handleSubmit=async()=>{
        const data={
            content:productBody//contenido del textarea
        }
        //realizamos la solicitud del backend
        const response = await fetch('http://localhost:2024/store/new-product',{
            method: 'POST',
            headers:{
                'Content-Type':'application/json',
            },
            body: JSON.stringify(JSON.parse(data.content)),
```

HandleSubmit

EL handlesubmit es una herramienta útil para llamar al backend y así enviar datos en formato JSON, esto para la creación de productos, llamando nuestro backend store en la creación de productos, crearemos los datos respetando los parámetros anteriormente asignados, una vez hecho esto podremos visualizar un mensaje donde nos mostrara que lo hicimos de la manera correcta, o en todo caso un mensaje de error, gracias ala librería SweetAlert2

```
const handleSubmit=async()=>{
    const data={
        content:productBody//contenido del textarea
    }
    //realizamos la solicitud del backend
    const response = await fetch('http://localhost:2024/store/new-product',{
        method: 'POST',
        headers:{
            'Content-Type':'application/json',
        },
        body: JSON.stringify(JSON.parse(data.content)),
    });
    const result = await response.json();
    if(result.status==="success"){
        await Swal.fire({
            title:'<span style="color:#A0A4B4;">Nuevo producto</span>',
            html:'<span style="color:#A0A4B4;">Creacion de producto exitoso</span>',
            icon:"success",
            confirmButtonText:"Cerrar",
            background:"#222",
            confirmButtonColor:"red",
        })
        await updateProducts();
```

getproducts

Una vez ingresemos los datos, haremos un llamado al arreglo de los datos creados, por medio de la ruta store, esto para poder ingresar los datos ala tabla que crearemos mas adelante.

```
const getProducts = async()=>{
    const response=await fetch("http://localhost:2024/store/get-products",{
        method:"GET",
        headers:{
            'Content-Type':'application.json',
        },
    });
    const products=await response.json();
```

Updateproducts

El updateproductos nos será útil para actualizar los datos de manera instantánea una vez ingresemos los datos.

```
const updateProducts=async()=>{
    try {
        var products=await getProducts();
        setProducts(products);
    } catch (error) {
        console.log(error)
    }
}
```

HandleSubmit2

El handlesubmit2 nos permitirá replicar los pasos anteriores, pero enfocándonos en la creación de clientes.

```
    const data={
        content:ClienteBody//contenido del textarea
    }
    //realizamos la solicitud del backend
    const response2 = await fetch('http://localhost:2024/store/clientes',{
        method: 'POST',
        headers:{
            'Content-Type':'application/json',
        },
        body: JSON.stringify(JSON.parse(data.content)),
    });
    const result = await response2.json();
    if(result.status==="success"){
        await Swal.fire({
            title:'<span style="color:#A0A4B4;">Nuevo cliente</span>',
            html:'<span style="color:#A0A4B4;">Creacion de cliente exitoso</span>',
            icon:"success",
            confirmButtonText:"Cerrar",
            background:"#222",
            confirmButtonColor:"red",
        })
        await updateClients();
    }else{
        await Swal.fire({
            title:'<span style="color:#A0A4B4;">Nuevo cliente</span>',
            html:'<span style="color:#A0A4B4;">Error al crear cliente</span>',
            icon:"error",
            confirmButtonText:"Cerrar",
            background:"#222",
            confirmButtonColor:"red",
        })
    }
}
```

```
const getclientes=async()=>{
    const response2=await fetch("http://localhost:2024/store/get-clientes",{
        method:"GET",
        headers:{
            'Content-Type':'application.json',
        },
    });
    const clientes=await response2.json();
    return clientes;
```

useEffect

haciendo uso del useeffect, podremos crear las dos graficas para la comparación de datos, tanto de productos como clientes.

La primera se hará una comparación de los precios de los productos para poder ser ordenados de menor a mayor en una grafica de barras.

```
useEffect(()=>{
    if(products.length>0){
        // Gráfica de barras
        const sortedProducts= [...products].sort((a,b)=>a.precio_producto-b.precio_producto);
        const barData={
            labels: sortedProducts.map((product)=>product.nombre_producto),
            datasets:[
                {
                    label:'Precio',
                    data:sortedProducts.map((product)=>product.precio_producto),
                    backgroundColor:['rgba(10, 255, 0)'],
                }
            ]
        }
        // Actualizamos la información de mi gráfica de barras
        setBarChartData(barData);
    }
},
[products]);
```

El segundo nos permitirá realizar una grafica de pie, donde se comparara los clientes mayores y menores de edad.

```
useEffect(()=>{
    if(clientes.length>0){
        // Gráfica de pie
        // Cantidad de productos con precio mayor a 100
        const Mayores = clientes.filter((cliente)=> cliente.edad>18).length;
        // Cantidad de productos con precio menor o igual a 100
        const Menores=clientes.length-Mayores;
        const pieData={
            labels:['Mayores de edad', 'Menores de edad'],
            datasets:[
                {
                    data:[Mayores,Menores],
                    backgroundColor:['rgba(192, 192, 192)','rgba(139, 0, 0)'],
                }
            ]
        };
        setPieChartData(pieData);
    }
},
[clientes]);
```

Una vez realizado esto, podemos seguir con el código HTML para la pagina web

```
return(
    <>
    <div>
        <h2>Cobra Kai Dashboard</h2>
        <textarea
            rows="10"
            cols="50"
            placeholder='crea un nuevo producto aqui..'
            value={productBody}
            onChange={handleChange}
        />
        <textarea
            rows="12"
            cols="50"
            placeholder='crea un usuario aqui'
            value={ClienteBody}
            onChange={handleChange2}
        />
        <br/>
        <button className="productos_container" onClick={handleSubmit}>Enviar</button>
        <button className="clientes_container"onClick={handleSubmit2}>Enviar</button>
    </div>
```

Colocando una variable de retorno, ingresaremos el titulo del dashboard, y seguiremos con un bloque de texto para poder ingresar productos y clientes.

Una vez ingresemos los datos seguimos con el código de los botones para enviar los datos al backend.

```
<div className='table-container'>
    <Table striped bordered hover variant='solid black'>
        <thead>
            <tr>
                <th>Id</th>
                <th>Nombre</th>
                <th>Precio</th>
                <th>Stock</th>
                <th>Estado</th>
            </tr>
        </thead>
        <tbody>{/*recorremos los productos*/}
            {products.length>0?(
                products.map((product)=>(
                    <tr ley={product.id_producto}>
                        <td>{product.id_producto}</td>
                        <td>{product.nombre_producto}</td>
                        <td>{product.precio_producto}</td>
                        <td>{product.stock_producto}</td>
                        //<button onClick={deleteProducts}>Eliminar</button>
                    </tr>
                ))
            ):(
                <tr>
                    <td colSpan="4">No hay productos registrados</td>
                </tr>
            )}
        </tbody>
    </Table>
    <h1>Clientes</h1>
    <Table striped bordered hover variant='dark'>
```

Una vez hecho esto, creamos una tabla para contener los datos que fueron escritos en el bloque de texto, organizando los datos de los productos y clients.

```
<h1>Clientes</h1>
<Table striped bordered hover variant='dark'>
    <thead>
        <tr>
            <th>Id cliente</th>
            <th>Nombre del cliente</th>
            <th>Apellido del cliente</th>
            <th>nit</th>
            <th>edad</th>
        </tr>
    </thead>
    <tbody>{/*recorremos los productos*/}
        {clientes.length>0?(
            clientes.map((cliente)=>(
                <tr ley={cliente.id_cliente}>
                    <td>{cliente.id_cliente}</td>
                    <td>{cliente.nombre}</td>
                    <td>{cliente.apellido}</td>
                    <td>{cliente.nit}</td>
                    <td>{cliente.edad}</td>
                </tr>
            ))
        ):(
            <tr>
                <td colSpan="5">No hay clientes registrados</td>
            </tr>
        )}
    </tbody>
</Table>
```

Una vez hecho esto, haremos un llamado ala grafica que crearemos con los datos a ingresar.

```
        <h1>Gráfica de productos ordenados por precio</h1>
            <div style={{width:'70%', margin:'auto'}}>
                {barChartData? <Bar data={barChartData}/>:<p>Cargando gráfica......</p>}
            </div>
            <h1>Grafica de edades&gt; 18 vs edad&lt;=18</h1>
            <div style={{width:'70%', margin:'auto'}}>
                {pieChartData? <Pie data={pieChartData}/>:<p>Cargando gráfica......</p>}
            </div>
    </div>
    </>
);
```