

Proyecto #2 . Manual Técnico

Alfredo Geovanni Ramirez Tzunún

Rudy Alexander Amado Soto Rosil



Desarrollo Web

Ingeniero: José Miguel Villatoro Hidalgo

Universidad Mariana Gálvez De Guatemala

Facultad De Ingeniería.

Guatemala, Mixco.

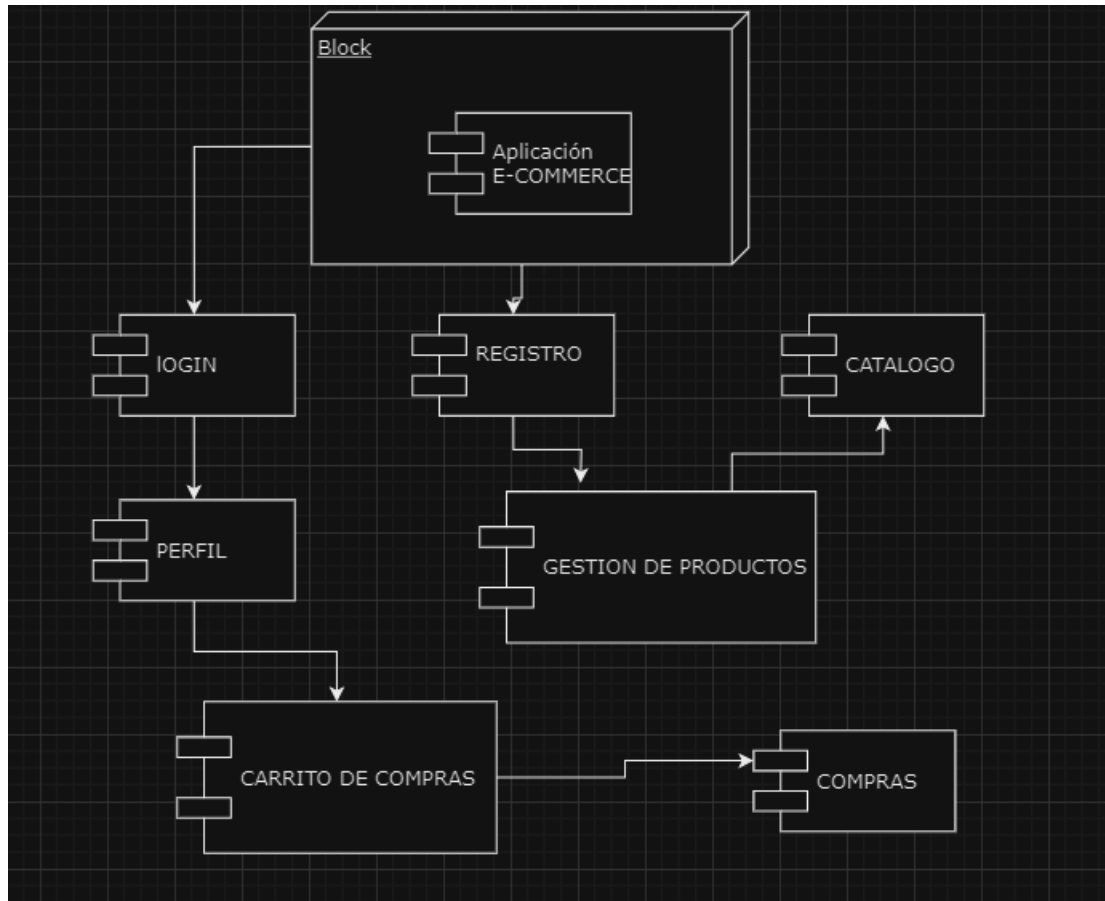
Noviembre De 2023

Contenido

Manual técnico.....	3
Server.....	6
Package.json.....	7
Routes	8
<i>Carritorouter</i>	<i>8</i>
<i>Productoroutes</i>	<i>9</i>
<i>Usuarioroutes.....</i>	<i>9</i>
Models	9
<i>carritoModel.....</i>	<i>10</i>
<i>productoModel</i>	<i>11</i>
<i>usuarioModel</i>	<i>11</i>
Controller	11
Node_modules.....	13
Auth	14
Anexo	15

Manual técnico

Esquema Conceptual de Componentes:



La arquitectura de nuestra aplicación de tienda en línea se organiza en torno a varios componentes clave:

Login y Registro: Este módulo es responsable de la autenticación y registro de usuarios en la plataforma. Permite a los usuarios registrarse, iniciar sesión y gestionar sus cuentas. También es el punto de inicio para acceder a las funcionalidades de la tienda en línea.

Catálogo de Productos: En este componente, los usuarios pueden explorar y buscar productos disponibles en la tienda en línea. Proporciona información detallada sobre cada producto y es esencial para la experiencia de compra.

Perfil: Este módulo permite a los usuarios gestionar su información personal, como nombre, dirección, datos de contacto y configuración de la cuenta. Los usuarios pueden actualizar su perfil según sea necesario.

Gestión de Productos: Diseñado específicamente para administradores de la plataforma, este componente es fundamental para mantener y administrar la información de los productos en la tienda en línea. Los administradores pueden agregar, modificar o eliminar productos, así como ajustar los niveles de inventario.

Carrito de Compra: El carrito de compra es esencial para que los usuarios gestionen los productos que desean comprar. Los usuarios pueden ver los productos que han agregado al carrito, ajustar cantidades y eliminar productos si es necesario.

Compras: Este módulo se encarga de gestionar y finalizar las compras. Procesa los pagos, actualiza el inventario de productos y registra la información de la transacción. Es el último paso en el proceso de compra en línea.

Uso de TypeScript:

TypeScript desempeña un papel fundamental en el desarrollo de nuestra aplicación de tienda en línea. Algunas de las formas en que se utiliza incluyen:

Declaración de tipos: Cada componente de la aplicación hace uso extensivo de la declaración de tipos para garantizar la integridad del código y reducir errores. Esto se logra definiendo los tipos de variables, funciones y estructuras de datos en toda la aplicación.

Interfaces: Utilizamos interfaces de TypeScript para definir estructuras de datos y garantizar que se mantenga la consistencia en la manipulación de datos en toda la aplicación. Esto facilita la comprensión del código y la colaboración entre desarrolladores.

Comprobación de tipos: TypeScript realiza una comprobación de tipos durante la compilación del código. Esto atrapa errores antes de la ejecución, lo que es esencial para garantizar que el código sea robusto y que no haya problemas inesperados en tiempo de ejecución.

Uso de SASS/SCSS:

SASS/SCSS se utiliza en todo el proyecto para mejorar la organización y el estilo de la interfaz de usuario. Algunas de las formas en que se aplica SASS/SCSS incluyen:

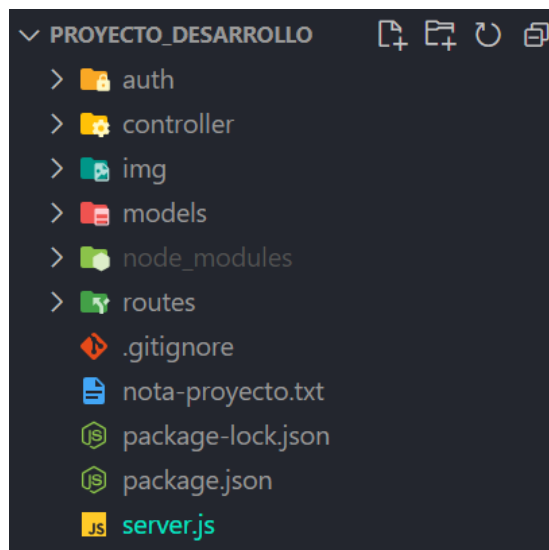
Variables: Definimos variables para colores, tamaños de fuente, márgenes y otros estilos reutilizables en toda la aplicación. Esto asegura que los estilos se mantengan coherentes y puedan modificarse de manera eficiente en un solo lugar.

Anidación: Los estilos CSS se anidan para mejorar la organización del código. Los estilos de un componente pueden anidarse dentro de los estilos de contenedor, lo que facilita la comprensión y el mantenimiento del código.

Mixins: Utilizamos mixins de SASS/SCSS para aplicar reglas de estilo de manera eficiente en múltiples componentes. Esto evita la repetición de código y mejora la consistencia visual en toda la aplicación.

Compilación: El código SASS/SCSS se compila en CSS estándar para su implementación en la aplicación web. El proceso de compilación es parte de nuestra cadena de desarrollo y garantiza que los estilos se apliquen correctamente en la interfaz de usuario.

Aquí veremos información muy útil para mantenimiento del código y actualizaciones respectivas para el aplicativo. El código consiste en el desarrollo de API's donde se puede interactuar, incluyendo productos para luego ser vendidos en un carrito de compras, todo por medio de servicios generados con node.js y mongo db para el resguardo de información.



Server

Aquí se incluye parte fundamental para la conexión a mongo, con credenciales de base de datos, como también llamadas a los archivos usados de rutas que harán las llamadas a los controladores de cada sección.

A continuación, vemos la parte principal del archivo donde incluimos las rutas que se utilizarán para los servicios creados.

```
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const cors = require('cors');
const usuarioRoutes = require('./routes/usuarioRoutes');
const productoRoutes = require('./routes/productoRoutes');
const cartRoutes = require('./routes/carritoRouter');

const app = express();
```

Hacemos la conexión a la base de datos, pasando el string de conexión que nos proporciona mongo db, donde el string de la base de datos se puede cambiar según sea la necesidad, las colecciones se crearan automáticamente cuando se levante el proyecto node.js. Para levantar un proyecto se debe de ejecutar el comando el **npm run dev**, dentro del path del proyecto.

```
const app = express();

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(cors());
mongoose.connect('mongodb+srv://geovanni:alfredo12345@cluster0.icxrihs.mongodb.net/PROYECTO_DEV', { useNew
  .then(() => {
    console.log('Conectado a la base de datos MongoDB');
  })
  .catch(error => {
    console.error('Error al conectarse a la base de datos: ', error);
  });


app.get('/', (req, res) => {
  res.json({ message: 'Servicios levantados de carrito de compras, Ok' });
});

app.use('/api', usuarioRoutes);
app.use('/api', productoRoutes);
app.use('/api', cartRoutes);

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Servidor corriendo en el puerto ${PORT}`);
});
```

Package.json

Aquí incluimos las librerías y dependencias necesarias para el uso correcto del proyecto, obviamente aquí se agregaran si fueran necesarias mas librerías, según sean las necesidades futuras. Conjunto al archivo package-lock.json estos archivos son fundamentales para el servicio y funcionamiento de la herramienta.



package-lock.json

package.json

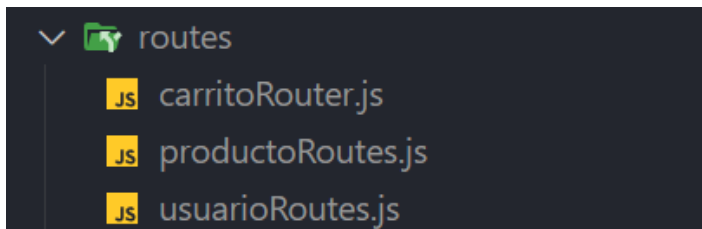
```

package.json > {} dependencies
1  {
2    "name": "proyecto_desarrollo",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "dev": "nodemon server.js"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "bcrypt": "^5.1.1",
14     "body-parser": "^1.20.2",
15     "cors": "^2.8.5",
16     "email-validator": "^2.0.4",
17     "express": "^4.18.2",
18     "jsonwebtoken": "^9.0.2",
19     "mongoose": "^7.5.0",
20     "nodemon": "^3.0.1"
21   }
22 }
23

```

Routes

Carpeta con archivos de rutas usadas en microservicios



Carritorouter

Contiene las rutas del archivo del controlador del carrito, aquí se puede agregar todas las rutas relacionadas a la colección del carrito

```

routes > JS carritoRouter.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  const verifyToken = require('../auth/authMiddleware');
5  const { getCart, updateCartItem, deleteCartItem } = require('../controller/carritoController');
6
7  router.get('/carrito', verifyToken, getCart);
8
9  router.post('/carrito', verifyToken, updateCartItem);
10
11 router.delete('/carrito', verifyToken, deleteCartItem);
12
13 module.exports = router;

```


Productoroutes

Contiene las rutas del archivo del controlador de los productos, así como anteriormente mencionado, aquí se puede agregar más rutas que tengan relación a la colección de producto

```
s > js productoRoutes.js > ...
const express = require('express');
const { getCatalog, getProduct, createOrUpdateProduct, deleteProduct } = require('../controller');
const verifyToken = require('../auth/authMiddleware');
const verifyAdmin = require('../auth/adminMiddleware');

const router = express.Router();

router.get('/productos', verifyToken, getCatalog);

router.get('/Producto/:ID', verifyToken, getProduct);
router.post('/Producto/:ID', verifyToken, verifyAdmin, createOrUpdateProduct);
router.post('/Producto/', verifyToken, createOrUpdateProduct);
router.delete('/Producto/:ID', verifyToken, verifyAdmin, deleteProduct);

module.exports = router;
```

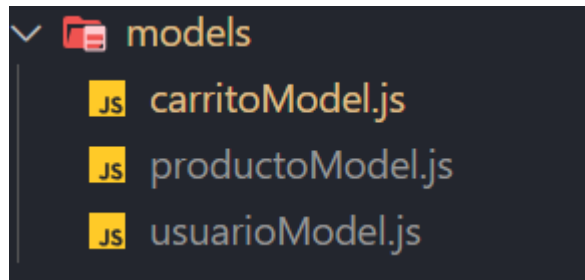
Usuarioroutes

Contiene las rutas del archivo del controlador de los usuarios y perfiles, se agregan rutas relacionadas a la colección de usuarios.

```
routes > js usuarioRoutes.js > ...
1  const express = require('express');
2  const verifyToken = require('../auth/authMiddleware');
3  const { registerUser, loginUser, getUsuarios, getProfile, updateProfile, deleteProfile } = require('../controller');
4
5
6  const router = express.Router();
7
8  router.post('/registro/:DPI', registerUser);
9  router.post('/login', loginUser);
10
11 router.get('/usuarios/', verifyToken, getUsuarios);
12 router.get('/perfil/:DPI', verifyToken, getProfile);
13 router.post('/perfil/:DPI', verifyToken, updateProfile);
14 router.delete('/perfil/:DPI', verifyToken, deleteProfile);
15
16 module.exports = router;
```

Models

Carpeta que contiene los archivos para uso de las colecciones en el proyecto, sin necesidad de tocar directamente la colección con la base de datos. En los archivos ubicados en esta carpeta tienen como función la conexión con la colección de carrito, producto, usuario o incluso otras colecciones. Lo ideal de estos archivos solo es crear las entradas de valores ya sea si tiene que crear, mostrar o actualizar alguna información según sea el caso.



carritoModel

```
const mongoose = require('mongoose');

const carritoItemSchema = new mongoose.Schema({
  ProductoID: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'productos',
    required: true
  },
  Cantidad: {
    type: Number,
    required: true,
    min: 1
  }
});

const carritoSchema = new mongoose.Schema({
  UsuarioID: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'usuarios',
    required: true,
    unique: true
  },
  Productos: [carritoItemSchema],
  Total: {
    type: Number,
    default: 0
  }
});

module.exports = mongoose.model('Carrito', carritoSchema);
```

productoModel

```
s > js productoModel.js > ...  
const mongoose = require('mongoose');  
  
const ProductSchema = new mongoose.Schema({  
  Identificador: String,  
  Nombre: String,  
  Marca: String,  
  Disponibilidad: Number,  
  Descuento: Number,  
  PrecioDescuento: Number,  
  Imagen: String,  
  Descripcion: String,  
  Categorias: [String]  
});  
  
module.exports = mongoose.model('productos', ProductSchema);
```

usuarioModel

```
models > js usuarioModel.js > ...  
1 const mongoose = require("mongoose");  
2 const bcrypt = require("bcrypt");  
3  
4 const userSchema = new mongoose.Schema({  
5   DPI: { type: String, required: true, unique: true },  
6   Nombres: { type: String, required: true },  
7   Apellidos: { type: String, required: true },  
8   FechaNacimiento: { type: Date, required: true },  
9   Clave: { type: String, required: true },  
10  DireccionEntrega: { type: String, required: true },  
11  NIT: { type: String, required: true, unique: true },  
12  NúmeroTelefonico: { type: String, required: true },  
13  CorreoElectronico: { type: String, required: true, unique: true },  
14  Rol: { type: String, enum: ["user", "admin", "cliente"], default: "user" },  
15 });  
16  
17 userSchema.pre("save", async function (next) {  
18   const salt = await bcrypt.genSalt(10);  
19   this.Clave = await bcrypt.hash(this.Clave, salt);  
20   next();  
21 });  
22  
23 const User = mongoose.model("usuarios", userSchema);  
24  
25 module.exports = User;  
26
```

Controller

Carpeta que contiene el backend de nuestra herramienta, es decir la logia del aplicativo. Aquí encontraremos archivos que interactúen con los servicios de proporcionados por las rutas ya sea por método Post, Get o algún otro método. Además será el intermediario entre los modelos y las rutas.

Tomar en cuenta que estos archivos dependerán de modelos para extraer información de colecciones o integrar o gestionar información de las colecciones.

```
const mongoose = require('mongoose');
const jwt = require('jsonwebtoken');

const User = require('../models/usuarioModel');
const Product = require('../models/productoModel');
const Cart = require('../models/carritoModel');

exports.getCart = async (req, res) => {
  try {
    const userId = req.user._id;

    const cart = await Cart.findOne({ UsuarioID: userId })

    if (!cart) {
      return res.status(404).json({ Mensaje: "Carrito de usuario no encontrado" });
    }

    res.json(cart);
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al obtener el carrito" });
  }
};

exports.updateCartItem = async (req, res) => {
  try {
    const userId = req.user._id;
    const { ProductoID, Cantidad } = req.body;

    const cart = await Cart.findOne({ UsuarioID: userId });

    if (!cart) {
      return res.status(404).json({ Mensaje: "Carrito de usuario no encontrado" });
    }

    const item = cart.items.find(item => item.ProductoID === ProductoID);

    if (!item) {
      return res.status(404).json({ Mensaje: "Producto no encontrado en el carrito" });
    }

    item.Cantidad = Cantidad;

    await cart.save();

    res.json(cart);
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al actualizar el carrito" });
  }
};

exports.getCatalog = async (req, res) => {
  try {
    const productos = await Product.find({});
    res.json(productos);
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al obtener los productos" });
  }
};

exports.getProduct = async (req, res) => {
  try {
    const product = await Product.findById(req.params.ID);
    if (!product) {
      return res.status(404).json({ Mensaje: "Producto no encontrado" });
    }
    res.json(product);
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al obtener el producto" });
  }
};

exports.createOrUpdateProduct = async (req, res) => {
  try {
    const {
      Identificador,
      Nombre,
      Marca,
      Disponibilidad,
      Precio,
      Descripcion
    } = req.body;

    const product = await Product.findOne({ Identificador });

    if (product) {
      product.Nombre = Nombre;
      product.Marca = Marca;
      product.Disponibilidad = Disponibilidad;
      product.Precio = Precio;
      product.Descripcion = Descripcion;
      await product.save();
    } else {
      const newProduct = new Product({
        Identificador,
        Nombre,
        Marca,
        Disponibilidad,
        Precio,
        Descripcion
      });
      await newProduct.save();
    }

    res.json(product);
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al crear o actualizar el producto" });
  }
};
```

```
const User = require("../models/usuarioModel");
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const emailValidator = require("email-validator");

exports.registerUser = async (req, res) => {
  try {
    const {
      DPI,
      Nombres,
      Apellidos,
      FechaNacimiento,
      Clave,
      ValidacionClave,
      DireccionEntrega,
      NIT,
      NumeroTelefonico,
      CorreoElectronico,
      Rol
    } = req.body;

    if (
      !DPI ||
      !Nombres ||
      !Apellidos ||
      !FechaNacimiento ||
      !Clave ||
      !ValidacionClave ||
      !DireccionEntrega ||
      !NIT ||
      !NumeroTelefonico ||
      !CorreoElectronico ||
      !Rol
    ) {
      return res
        .status(400)
        .json({ Mensaje: "Por favor, complete todos los campos." });
    }

    if (Clave !== ValidacionClave) {
      return res.status(400).json({ Mensaje: "Las contraseñas no coinciden." });
    }

    if (Clave.length < 8) {
      return res
        .status(400)
        .json({ Mensaje: "La contraseña debe tener al menos 8 caracteres." });
    }

    const email = CorreoElectronico;
    if (!emailValidator.validate(email)) {
      return res
        .status(400)
        .json({ Mensaje: "El correo electrónico no es válido." });
    }

    const user = await User.findOne({ CorreoElectronico: email });

    if (user) {
      return res
        .status(400)
        .json({ Mensaje: "El correo electrónico ya está registrado." });
    }

    const hashedClave = bcrypt.hashSync(Clave, 10);

    const newUser = new User({
      DPI,
      Nombres,
      Apellidos,
      FechaNacimiento,
      Clave: hashedClave,
      ValidacionClave: hashedClave,
      DireccionEntrega,
      NIT,
      NumeroTelefonico,
      CorreoElectronico,
      Rol
    });

    await newUser.save();

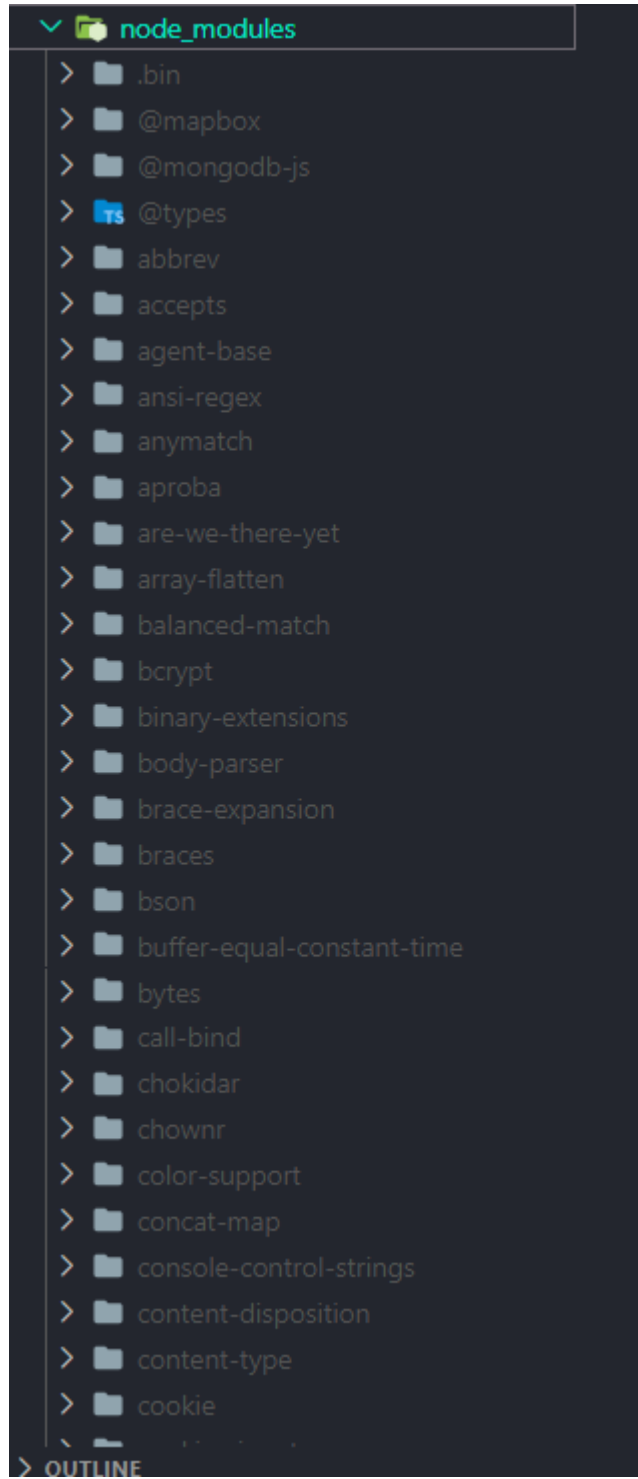
    const token = jwt.sign({ id: newUser._id }, process.env.JWT_SECRET, {
      expiresIn: process.env.JWT_EXPIRATION
    });

    res.json({
      token,
      user: {
        id: newUser._id,
        nombre: newUser.Nombres,
        apellido: newUser.Apellidos,
        fechaNacimiento: newUser.FechaNacimiento,
        nit: newUser.NIT,
        numeroTelefonico: newUser.NumeroTelefonico,
        correoElectronico: newUser.CorrreoElectronico,
        rol: newUser.Rol
      }
    });
  } catch (error) {
    res.status(500).json({ Mensaje: "Error al registrar el usuario" });
  }
};
```

Node_modules

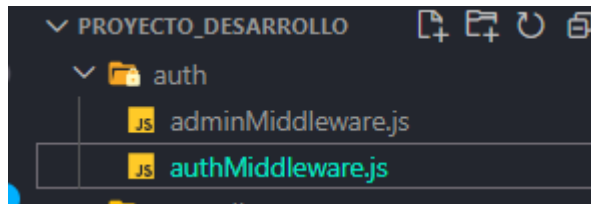
Carpeta con archivos descargados y mencionados en las llamadas de referencia de package.json

Esta por demás mencionar pero esta carpeta no se toca, esta se genera y actualiza automáticamente cuando se incluyen librerías o recursos externos para la funcionalidad de nuestro aplicativo.

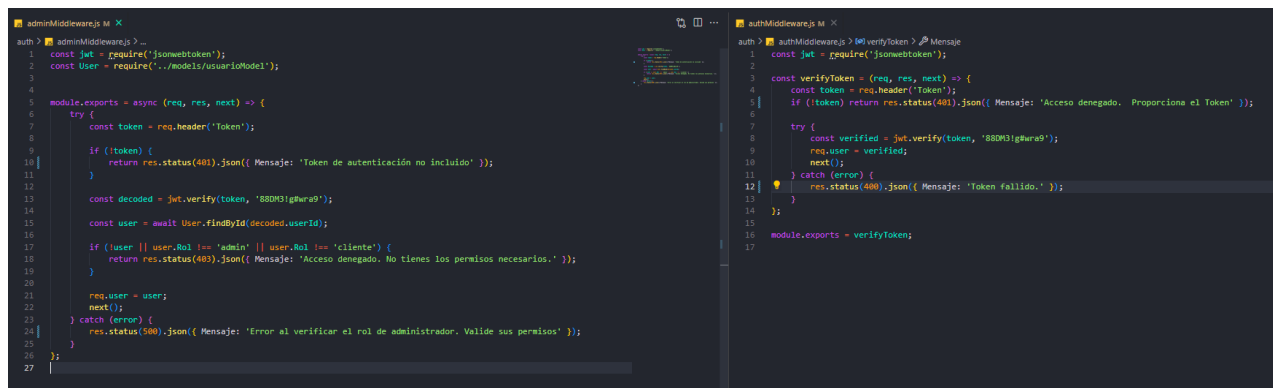


Auth

Carpeta que contiene la lógica de verificación de autenticación y logueo del sistema como también la validación del rol del usuario.

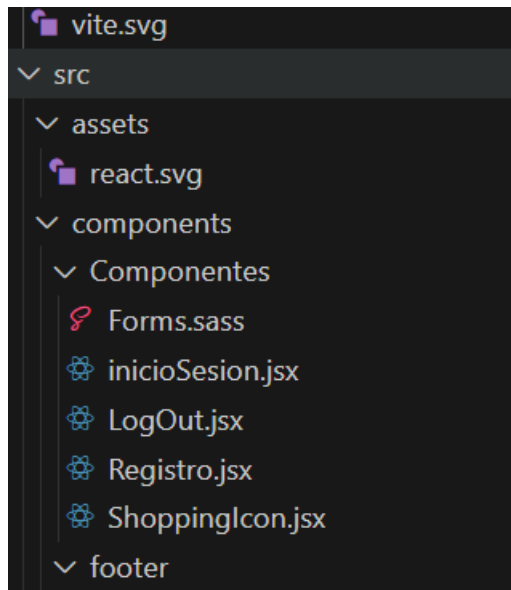
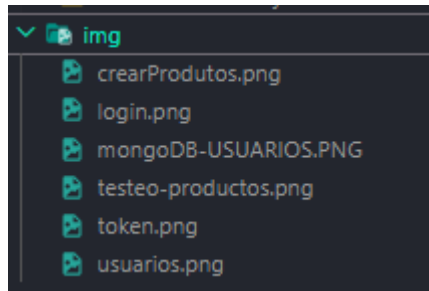


El código de estos es entendible, uno verifica el rol del usuario logueado para poder permisos y accesos a ciertas áreas y el otro permite la interacción con el aplicativo no importando su rol, ya que este da acceso a que pueda usar los recursos según su rol que tenga.



Anexo

Existe una carpeta con imágenes, estas son evidencias de la funcionalidad y testeo de la herramienta, donde se probaron algunas rutas y dando evidencia en Postman y sus datos reflejados en mongo db



```
61 background-color: #ffffff;
62 }
63 a:hover {
64   color: #747bff;
65 }
66 button {
67   background-color: #f9f9f9;
68 }
69 }
70
71 /* asfssdfsdfsdf
72
```