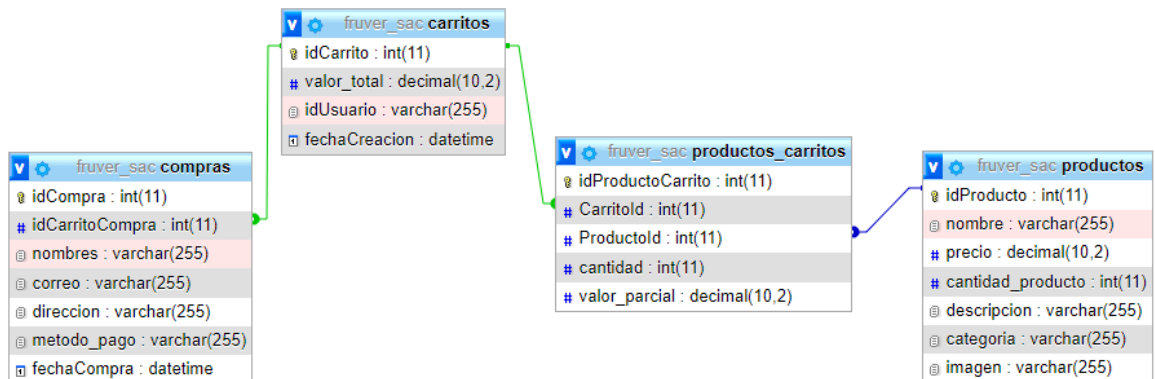


Universidad de Nariño.
Ingeniería de Sistemas.
Diplomado de actualización en nuevas tecnologías para el desarrollo de Software.
Taller Unidad 2 Backend

Santiago Alejandro Coral Ruano
217036022

1. Creación de una base de datos MYSQL/MARIADB que permita llevar el registro de un FRUVER (FRUTAS Y VERDURAS), así como también el proceso de solicitud de compra de estas.



2. Desarrollar una aplicación Backend implementada en NodeJS y ExpressJS que haga uso de la base de datos del primer punto y que permita el desarrollo de todas las tareas asociadas al registro y administración de las frutas y verduras. Se debe hacer uso correcto de los verbos HTTP dependiendo de la tarea a realizar.

Usar el comando `npm init -y` para iniciar un nuevo proyecto en node:

```
$ npm init -y
Wrote to C:\Users\Santiago Coral\Documents\Diplomado\Modulo2 Diplomado Junio 2023\Taller2 Backend\package.json:

{
  "name": "taller2-backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Instalar express usando el comando: *npm install (npm i) express*

```
$ npm install express  
  
added 58 packages, and audited 59 packages in 10s  
  
8 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Ahora, instalar MySQL para conexión con la base de datos, para asegurarnos se usa los comandos: *npm i mysql* y *npm i mysql2*

```
$ npm i mysql npm mysql2  
  
added 24 packages, and audited 331 packages in 29s  
  
35 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities
```

Verificar en el archivo *package.json* las dependencias instaladas

```
{  
  "name": "backend",  
  "version": "1.0.0",  
  "description": "Proyecto de backend",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.18.2",  
    "mysql": "^2.18.1",  
    "mysql2": "^3.5.1",  
    "npm": "^9.8.0"  
  }  
}
```

Para no detener el servicio y estar visualizando los cambios automáticamente instalamos nodemon: *npm i nodemon --save-dev*

```
$ npm install nodemon --save-dev

added 33 packages, and audited 364 packages in 9s

38 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Instalar *sequelize* para el manejo de la base de datos: *npm install sequelize*

```
$ npm install sequelize

added 19 packages, and audited 383 packages in 34s

39 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Creamos un archivo a nivel raíz del proyecto `server.js` en el que hacemos las siguientes configuraciones:

```
import express from "express";
import router from "../Routes/routes.js";
import { sequelize } from "../Database/database.js";
import { Producto } from "../Models/productos.js";
import cors from 'cors';
//Crear instancia de express
const app = express();
app.use(cors());
app.use(express.json());
app.use(router);
app.set('port', 3000);
//Test a la base de datos
const testDb = async () => {
  try {
    // await sequelize.authenticate();
    await sequelize.sync();
    console.log('Conexion realizada con exito');
    app.listen(app.get('port'), () => {
      console.log(`Servidor corriendo en http://localhost:${app.get('port')}`);
    });
  } catch (error) {
    console.error(`Error al realizar conexion:: ${error}`);
  }
};
testDb();
```

Creamos una carpeta 'Database' con un archivo database.js para configurar la conexión a la base de datos en MySQL:

```
import Sequelize from "sequelize";

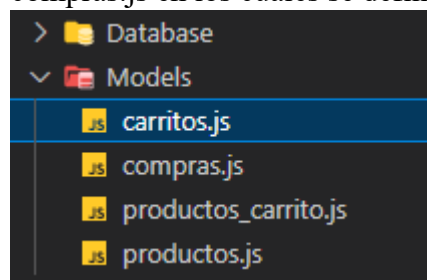
const sequelize = new Sequelize("fruver_sac", "root", "", {
  host: "localhost",
  dialect: "mysql",
});

export {
  sequelize
};
```

Modificar el archivo package.json

```
"type": "module",
  ▶ Depurar
  "scripts": {
    "start": "nodemon server.js"
  },
```

Crearemos 4 modelos teniendo en cuenta la base de datos creada. Por lo que dentro de la carpeta Models se crean 4 archivos: productos.js, productos_carritos.js, carritos.js y compras.js en los cuales se definen los atributos de igual manera que en la base de datos.



Por ejemplo, para el modelo productos.js se definen los atributos de la siguiente manera:

```

const Producto = sequelize.define('productos', {
  // Model attributes are defined here. Los atributos de la tabla
  idProducto: {
    type: DataTypes.INTEGER,
    allowNull: false,
    primaryKey: true,
    autoIncrement: true
  },
  nombre: {
    type: DataTypes.STRING
  },
  precio: {
    type: DataTypes.DECIMAL(10,2)
  },
  cantidad_producto: {
    type: DataTypes.INTEGER
  },
  descripcion: {
    type: DataTypes.STRING
  },
  categoria: {
    type: DataTypes.STRING
  },
  imagen: {
    type: DataTypes.STRING
  }
},

```

Para los demás modelos se tiene en cuenta los atributos de cada tabla de la base de datos.

Se crea un archivo controller.js dentro de una carpeta 'Controllers' para definir las funciones que retornaran los datos a las peticiones get, post, put y delete. Aquí se definen al menos 4 métodos para cada modelo de acuerdo a las peticiones HTTP. Pueden ser más, dependiendo de la necesidad en Frontend.

Por ejemplo, para el modelo producto uno de los métodos definidos es el método getProductos con el que se obtendrán todos los productos disponibles en la base de datos:

```
const getProductos = async (req, res) => { //Todos los
  try {
    const productos = await Producto.findAll();
    res.status(200).json(productos);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Para el modelo Carrito uno de los métodos que se definen es postCarrito con el que se creará un carrito de compras en la base de datos:

```
const postCarrito = async (req, res) => { //Agregar un carrito
  const { idUsuario, valor_total } = req.body;
  try {
    const newCarrito = await Carrito.create({ idUsuario: idUsuario, valor_total: valor_total });
    res.status(200).json(newCarrito);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Para el modelo Productocarrito uno de los métodos que se definen es deleteProductoCarrito con el que se podrá eliminar o sacar un producto de un carrito de compras en la base de datos:

```
const deleteProductoCarrito = async (req, res) => { //Eliminar un producto del carrito
  const { idProductoCarrito } = req.params;
  try {
    const respuesta = await Productocarrito.destroy({ where: { idProductoCarrito: idProductoCarrito } });
    res.status(200).json({
      body: {
        mensaje: `Registro con id ${idProductoCarrito} Eliminado Satisfactoriamente`
      }
    });
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `No se pudo eliminar. ${error}` });
  }
};
```

Y para el modulo Compras uno de los métodos definidos es postCompra, con el que agregamos una compra a la base de datos:

```
const postCompra = async (req, res) => { //Agregar una compra
  const { idCarritoCompra, nombres, correo, direccion, metodo_pago } = req.body;
  try {
    const newCompra = await Compra.create({ idCarritoCompra: idCarritoCompra, nombres: nombres,
      correo: correo, direccion: direccion, metodo_pago: metodo_pago });
    res.status(200).json(newCompra);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Así definimos los diferentes métodos con los que podremos manipular la base de datos de acuerdo a las peticiones HTTP. Y luego los exportamos:

```
export {
  getProductos,
  getProductosByCategoria,
  getProducto,
  postProductos,
  putProductos,
  deleteProductos,
  getAllCarritos,
  getCarrito,
  getCarritoById,
  postCarrito,
  putCarrito,
  deleteCarrito,
  getAllProductosAllCarritos,
  getAllProductosCarrito,
```

Crear una carpeta Routes y dentro un archivo routes.js para administrar las rutas de app. En este archivo definimos las rutas para los diferentes módulos. Importamos todos los métodos que se crearon en el archivo controller.js

```
import { Router } from 'express';
// const {getProductos, postProductos, putProductos, deleteProductos, getAllCarritos, getCarrito, getCarritoById, postCarrito, putCarrito, deleteCarrito, getAllProductosAllCarritos, getAllProductosCarrito, } from '../Controllers/controller.js';

const router = Router();
```

Creamos el objeto router para definir todas las rutas de acuerdo a cada uno de los métodos

```
router.get("/productos", getProductos);
router.get("/productos_cat/:categoria", getProductosByCategoria);
router.get("/productos/:idProducto", getProducto);
router.post("/productos", postProductos);
router.put("/productos/:idProducto", putProductos);
router.delete("/productos/:idProducto", deleteProductos); //Recibe como parametro un idProducto

router.get("/carritos", getAllCarritos);
router.get("/carritoUser/:idUser", getCarrito);
router.get("/carritoId/:idCarrito", getCarritoById);
router.post("/carrito", postCarrito);
router.put("/carrito/:idCarrito", putCarrito);
router.delete("/carrito/:idCarrito", deleteCarrito);
```

Por último, exportamos el objeto router

```
router.get("/compras", getAllCompras);
router.get("/comprasCorreo/:correo", getComprasByCorreo);
router.get("/comprasId/:idCompra", getComprasById);
router.post("/compras", postCompra);
router.put("/compras/:idCompra", putCompra);
router.delete("/compras/:idCompra", deleteCompra);

// module.exports=router;
export default router;
```


Ahora, ejecutamos: *npm run start* para correr nuestra aplicación backend:

```
$ npm run start

> backend@1.0.0 start
> nodemon server.js

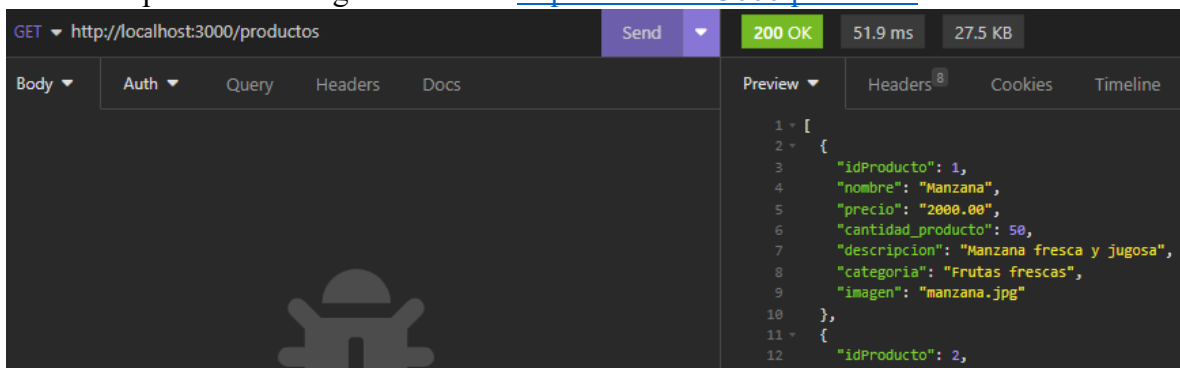
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'carritos' AND TABLE_SCHEMA = 'fruver_sac'
Executing (default): SHOW INDEX FROM `carritos`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'compras' AND TABLE_SCHEMA = 'fruver_sac'
Executing (default): SHOW INDEX FROM `compras`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'productos' AND TABLE_SCHEMA = 'fruver_sac'
Executing (default): SHOW INDEX FROM `productos`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'productos_carritos' AND TABLE_SCHEMA = 'fruver_sac'
Executing (default): SHOW INDEX FROM `productos_carritos`
Conexion realizada con exito
Servidor corriendo en http://localhost:3000
[]
```

Como vemos nos realiza la conexión exitosamente.

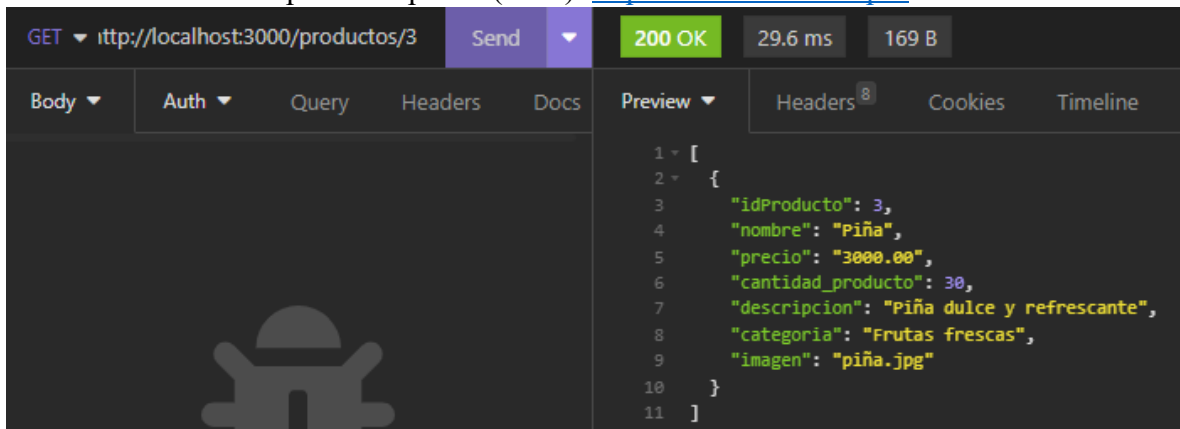
3. Realizar verificación de las diferentes operaciones a través de un cliente grafico (Insomnia), tomar capturas de pantalla que evidencien el resultado de las solicitudes realizadas.

Se crea un nuevo proyecto en Insomnia para hacer las verificaciones de los métodos definidos.

Probamos para la ruta de getProductos: <http://localhost:3000/productos>



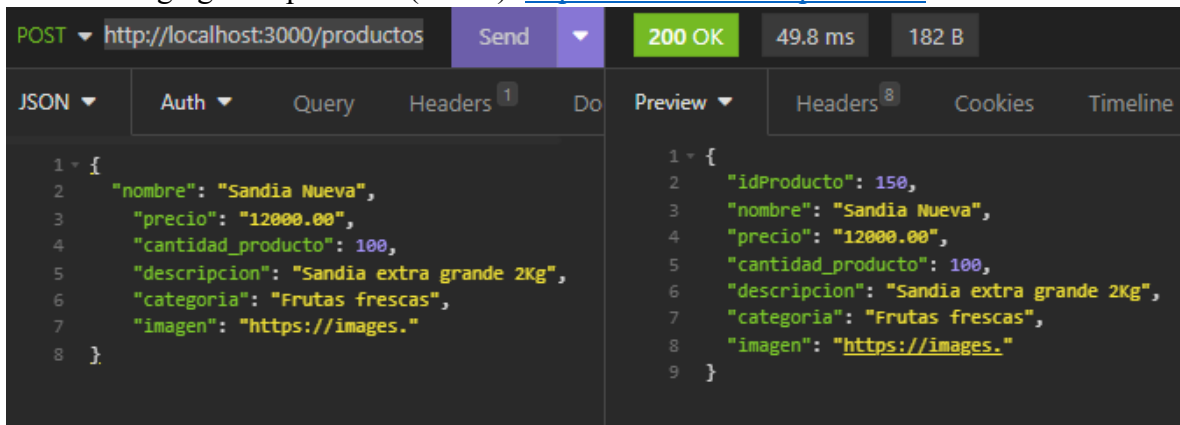
Probamos obtener un producto por id (GET): <http://localhost:3000/productos/idProducto>



```
GET http://localhost:3000/productos/3 200 OK 29.6 ms 169 B
```

```
{
  "idProducto": 3,
  "nombre": "Piña",
  "precio": "3000.00",
  "cantidad_producto": 30,
  "descripcion": "Piña dulce y refrescante",
  "categoria": "Frutas frescas",
  "imagen": "piña.jpg"
}
```

Probamos agregar un producto (POST): <http://localhost:3000/productos>

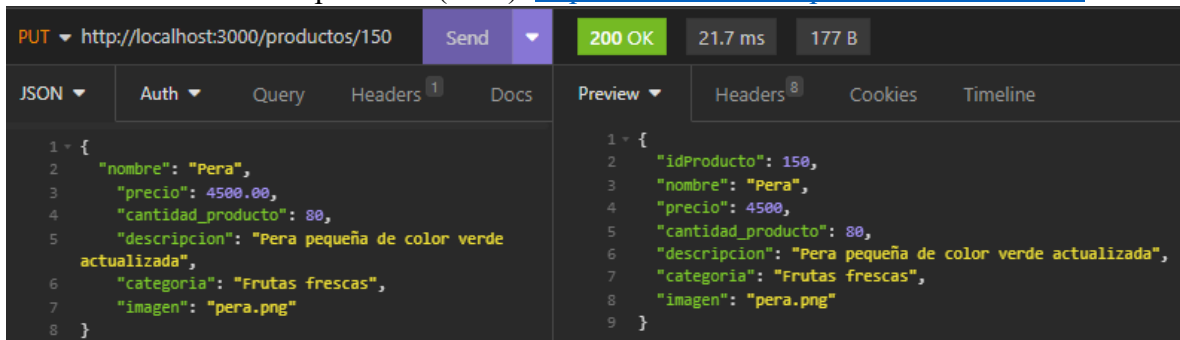


```
POST http://localhost:3000/productos 200 OK 49.8 ms 182 B
```

```
{
  "nombre": "Sandia Nueva",
  "precio": "12000.00",
  "cantidad_producto": 100,
  "descripcion": "Sandia extra grande 2Kg",
  "categoria": "Frutas frescas",
  "imagen": "https://images."
}
```

```
{
  "idProducto": 150,
  "nombre": "Sandia Nueva",
  "precio": "12000.00",
  "cantidad_producto": 100,
  "descripcion": "Sandia extra grande 2Kg",
  "categoria": "Frutas frescas",
  "imagen": "https://images."
}
```

Probamos actualizar un producto (PUT): <http://localhost:3000/productos/idProducto>

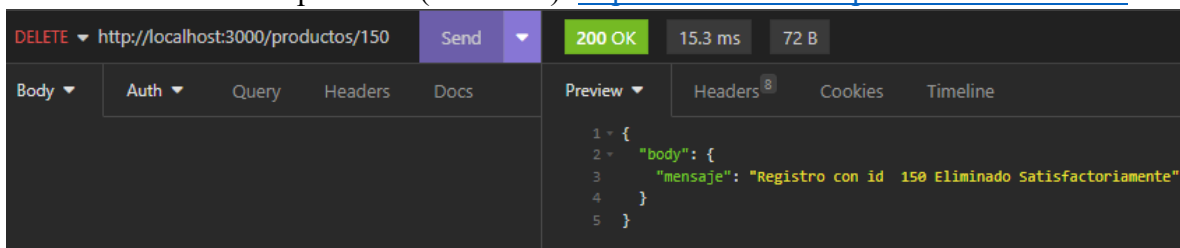


```
PUT http://localhost:3000/productos/150 200 OK 21.7 ms 177 B
```

```
{
  "nombre": "Pera",
  "precio": 4500.00,
  "cantidad_producto": 80,
  "descripcion": "Pera pequeña de color verde actualizada",
  "categoria": "Frutas frescas",
  "imagen": "pera.png"
}
```

```
{
  "idProducto": 150,
  "nombre": "Pera",
  "precio": 4500,
  "cantidad_producto": 80,
  "descripcion": "Pera pequeña de color verde actualizada",
  "categoria": "Frutas frescas",
  "imagen": "pera.png"
}
```

Probamos eliminar un producto (DELETE): <http://localhost:3000/productos/idProducto>



```
DELETE http://localhost:3000/productos/150 200 OK 15.3 ms 72 B
```

```
{
  "body": {
    "mensaje": "Registro con id 150 Eliminado Satisfactoriamente"
  }
}
```

Probamos agregar un carrito de compras (POST): <http://localhost:3000/carrito>

POST <http://localhost:3000/carrito> Send 200 OK 34.6 ms 104 B

JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies Timeline

```
1 {
2   "idUsuario": "usuario_prueba",
3   "valor_total": 0
4 }
```

```
1 {
2   "fechaCreacion": {
3     "val": "CURRENT_TIMESTAMP"
4   },
5   "idCarrito": 5,
6   "idUsuario": "usuario_prueba",
7   "valor_total": 0
8 }
```

Probamos obtener un carrito de compras por su id (GET):

<http://localhost:3000/carritoId/idCarrito>

```
[
  {
    "idCarrito": 5,
    "idUsuario": "usuario_prueba",
    "valor_total": "0.00",
    "fechaCreacion": "2023-07-17T00:21:50.000Z"
  }
]
```

Probamos agregar un producto al carrito (POST):

http://localhost:3000/productos_carrito/idCarrito

POST http://localhost:3000/productos_carrito/5 Send 200 OK 35.7 ms 91 B

JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies

```
1 {
2   "ProductoId": 1,
3   "cantidad": "5",
4   "valor_parcial": 10000
5 }
```

```
1 {
2   "idProductoCarrito": 8,
3   "CarritoId": "5",
4   "ProductoId": 1,
5   "cantidad": "5",
6   "valor_parcial": 10000
7 }
```

Probamos a actualizar el valor del carrito de compras agregado (PUT):

<http://localhost:3000/carrito/idCarrito>

PUT <http://localhost:3000/carrito/5> Send 200 OK 6.43 ms 37 B

JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies

```
1 {
2   "valor_total": 10000
3 }
```

```
1 {
2   "idCarrito": "5",
3   "valor_total": 10000
4 }
```

Probamos eliminar un carrito de compras:

DELETE ▼ http://localhost:3000/carrito/5 Send ▼ 200 OK 9.81 ms 70 B

Body ▼ Auth ▼ Query Headers Docs Preview ▼ Headers 8 Cookies Timeline

```
1 {  
2   "body": {  
3     "mensaje": "Registro con id 5 Eliminado Satisfactoriamente"  
4   }  
5 }
```

Probamos actualizar el producto del carrito (PUT):

PUT ▼ http://localhost:3000/producto_carrito/8 Send ▼ 200 OK 11.8 ms 86 B

JSON ▼ Auth ▼ Query Headers 1 Docs Preview ▼ Headers 8 Cookies

```
1 {  
2   "CarritoId": 5,  
3   "ProductoId": 4,  
4   "cantidad": 2,  
5   "valor_parcial": 5000.00  
6 }
```

```
1 {  
2   "idProductoCarrito": 8,  
3   "CarritoId": 5,  
4   "ProductoId": 4,  
5   "cantidad": 2,  
6   "valor_parcial": 5000  
7 }
```

Probamos a agregar otro producto:

POST ▼ http://localhost:3000/productos_carrito/5 Send ▼ 200 OK 26.1 ms 90 B

JSON ▼ Auth ▼ Query Headers 1 Docs Preview ▼ Headers 8 Cookies

```
1 {  
2   "ProductoId": 1,  
3   "cantidad": 5,  
4   "valor_parcial": 10000  
5 }
```

```
1 {  
2   "idProductoCarrito": 11,  
3   "CarritoId": "5",  
4   "ProductoId": 1,  
5   "cantidad": 5,  
6   "valor_parcial": 10000  
7 }
```

Probamos a eliminar un producto del carrito:

DELETE ▼ http://localhost:3000/producto_carrito/6 Send ▼ 200 OK 34.6 ms 70 B

Body ▼ Auth ▼ Query Headers Docs Preview ▼ Headers 8 Cookies Timeline

```
1 {  
2   "body": {  
3     "mensaje": "Registro con id 6 Eliminado Satisfactoriamente"  
4   }  
5 }
```

Probamos agregar una compra:

```
POST http://localhost:3000/compras Send 200 OK 81.8 ms 141 B
```

```
JSON Auth Query Headers 1 Docs Preview 8 Cookies
```

```
1 {
2   "idCarritoCompra": 2,
3   "nombres": "Santiago Coral",
4   "correo": "santi",
5   "direccion": "Potosi",
6   "metodo_pago": "PSE"
7 }
```

```
1 {
2   "fechaCompra": {
3     "val": "CURRENT_TIMESTAMP"
4   },
5   "idCompra": 2,
6   "idCarritoCompra": 2,
7   "nombres": "Santiago Coral",
8   "correo": "santi",
9   "direccion": "Potosi"
10 }
```

Probamos obtener una compra:

```
GET http://localhost:3000/comprasId/2 Send 200 OK 12.8 ms 465 B
```

```
Body Auth Query Headers Docs Preview 8 Cookies Timeline
```

```
1 [
2   {
3     "idCompra": 2,
4     "idCarritoCompra": 2,
5     "nombres": "Santiago",
6     "correo": "santiagocoral8@gmail.com",
7     "direccion": "Potosi",
8     "metodo_pago": "tarjeta de credito",
9     "fechaCompra": "2023-07-16T19:07:26.000Z"
10  }
11 ]
```

Probamos a eliminar una compra:

```
PUT http://localhost:3000/compras/2 Send 200 OK 8.59 ms 3 B
```

```
JSON Auth Query Headers 1 Docs Preview 8 Cookies
```

```
1 {
2   "idCarritoCompra": 2,
3   "nombres": "Santiago A Coral",
4   "correo": "santiagocoral8@gmail.com",
5   "direccion": "Ipiiales",
6   "metodo_pago": "PSE"
7 }
```

```
1 "2"
```

Probamos eliminar una compra:

```
DELETE http://localhost:3000/compras/2 Send 200 OK 16.7 ms 70 B
```

```
Body Auth Query Headers Docs Preview 8 Cookies Timeline
```

```
1 {
2   "body": {
3     "mensaje": "Registro con id 2 Eliminado Satisfactoriamente"
4   }
5 }
```

Y como se puede ver, se ha verificado que los métodos definidos están bien configurados ya que se obtuvo una buena respuesta desde el cliente gráfico Insomnia.