

Universidad de Nariño.

Ingeniería de Sistemas.

Diplomado de actualización en nuevas tecnologías para el desarrollo de Software.

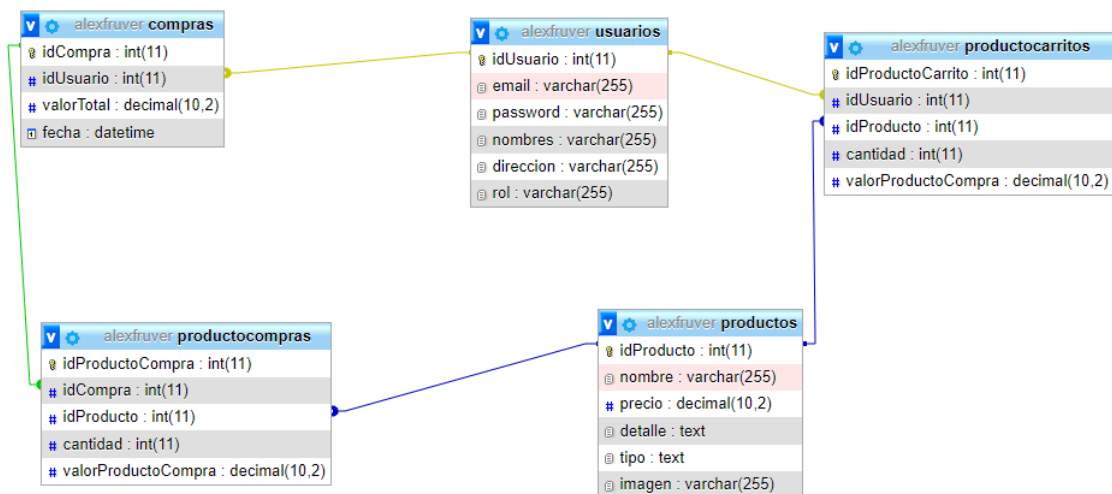
Código: 22036268

Nombres: Alexander Luna

INFORME TALLER UNIDAD 2 BACKEND

Crear la base de datos para el proyecto.

Se usará xampp para la conexión a la base de datos, por lo que en localhost/phpmyadmin se crea una nueva base de datos llamada: “alexfruver” en la que se definen 4 tablas con sus respectivos atributos, como se muestra en la siguiente imagen:



Crear el proyecto Backend con NodeJs y Express.

Luego de crear y guardar la base de datos, se procede a crear el proyecto Backend, por lo que en una terminal se ejecuta el comando: *npm init -y*

```

PS C:\TallerFruverAlex\Backend> npm init -y
Wrote to C:\TallerFruverAlex\Backend\package.json:

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

```

Se instalan las dependencias necesarias para el backend:

npm install express sequelize mysql mysql2 cors nodemon --save-dev

```

PS C:\TallerFruverAlex\Backend> npm install express sequelize mysql mysql2 cors nodemon

added 135 packages, and audited 136 packages in 17s

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

found 0 vulnerabilities

```

En el archivo 'package.json' se verifica que estén listadas las dependencias instaladas:

```

package.json > ...
1  {
2    "name": "backend",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC",
12   "dependencies": {
13     "cors": "^2.8.5",
14     "express": "^4.18.2",
15     "mysql": "^2.18.1",
16     "mysql2": "^3.5.2",
17     "nodemon": "^3.0.1",
18     "sequelize": "^6.32.1"
19   }
20 }

```

Modificar a tipo modulo en el archivo package.json:

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "Backend Fruver",
  "main": "server.js",
  "type": "module",
  "scripts": {
    "start": "nodemon server.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Configurar la base de datos con Sequelize

Creamos un archivo para la base de datos y se definen las propiedades para la conexión:

```
Database > .js database.js > sequelize
1  import Sequelize from "sequelize";
2
3  const sequelize = new Sequelize("alexfruver", "root", "", {
4    |   host: "localhost",
5    |   dialect: "mysql",
6  | });
7
8  export {
9    |   sequelize
10 | };
11
```

Crear los modelos en Backend

Se crea una carpeta llamada 'Models' en la que se guardarán los archivos para cada modelo a definir.

Teniendo en cuenta la base de datos creada inicialmente, se tendrán 4 modelos. Comenzamos con el modelo '**Producto**', el cual lo definimos en un archivo productos.js con los atributos similares a la tabla de la base de datos:

```
Models > producto.js > Producto
1  import { DataTypes } from 'sequelize';
2  import { sequelize } from '../Database/database.js';
3
4  const Producto = sequelize.define('Producto', {
5    idProducto: {
6      type: DataTypes.INTEGER,
7      autoIncrement: true,
8      primaryKey: true,
9      allowNull: false,
10   },
11   nombre: {
12     type: DataTypes.STRING,
13   },
14   precio: {
15     type: DataTypes.DECIMAL(10, 2),
16   },
17   detalle: {
18     type: DataTypes.TEXT,
19   },
20   imagen: {
21     type: DataTypes.STRING,
22   },
23 });
24
25 export { Producto };
26
```

Se crea un nuevo archivo 'usuarios.js' dentro de la carpeta 'Models' para definir el modelo Usuario con sus respectivos atributos.

```
Models > usuarios.js > Usuario > password > allowNull
1  import { DataTypes } from 'sequelize';
2  import { sequelize } from '../Database/database.js';
3
4  const Usuario = sequelize.define('Usuario', {
5    idUsuario: {
6      type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, allowNull: false,
7    },
8    email: {
9      type: DataTypes.STRING, allowNull: false,
10   },
11   password: {
12     type: DataTypes.STRING, allowNull: false,
13   },
14   nombres: {
15     type: DataTypes.STRING,
16   },
17   direccion: {
18     type: DataTypes.STRING,
19   },
20   rol: {
21     type: DataTypes.STRING,
22   },
23 });
24
25 export { Usuario };
26
```

Se crea un nuevo archivo 'productoCarrito.js' dentro de la carpeta 'Models' para definir el modelo ProductoCarrito con sus respectivos atributos. Además de definir las relaciones que tiene con los modelos 'Producto' y 'Usuario':

```
Models > JS productoCarrito.js > ...
6
7 const ProductoCarrito = sequelize.define('ProductoCarrito', {
8   idProductoCarrito: {
9     type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, allowNull: false,
10  },
11  idUsuario: {
12    type: DataTypes.INTEGER, allowNull: false, references: { model: Usuario, key: 'idusuario' }
13  },
14  idProducto: {
15    type: DataTypes.INTEGER, allowNull: false, references: { model: Producto, key: 'idProducto' }
16  },
17  cantidad: {
18    type: DataTypes.INTEGER, allowNull: false,
19  },
20  valorProductoCompra: {
21    type: DataTypes.DECIMAL(10, 2), allowNull: false,
22  },
23 },
24 {
25   timestamps: false,
26 });
27
28 ProductoCarrito.belongsTo(Producto, { foreignKey: 'idProducto' });
29 ProductoCarrito.belongsTo(Usuario, { foreignKey: 'idUsuario' });
30 Usuario.hasMany(ProductoCarrito, { foreignKey: 'idUsuario' });
```

Se crea un nuevo archivo 'compras.js' dentro de la carpeta 'Models' para definir el modelo Compra con sus respectivos atributos. Además de definir las relaciones que tiene con el modelo 'Usuario':

```
const Compra = sequelize.define('Compra', {
  idCompra: {
    type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, allowNull: false,
  },
  idUsuario: {
    type: DataTypes.INTEGER, allowNull: false, references: {
      model: Usuario,
      key: 'idUsuario'
    }
  },
  valorTotal: {
    type: DataTypes.DECIMAL(10, 2), allowNull: false,
  },
  fecha: {
    type: DataTypes.DATE, allowNull: false, defaultValue: DataTypes.NOW,
  },
});

Compra.belongsTo(Usuario, { foreignKey: 'idUsuario' });
Usuario.hasMany(Compra, { foreignKey: 'idUsuario' });

export { Compra};
```

Se crea un nuevo archivo 'productoCompra.js' dentro de la carpeta 'Models' para definir el modelo ProductoCompra con sus respectivos atributos. Además de definir las relaciones que tiene con los modelos 'Producto' y 'Compra':

```
const ProductoCompra = sequelize.define('ProductoCompra', {
  idProductoCompra: {
    type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true, allowNull: false,
  },
  idCompra: {
    type: DataTypes.INTEGER, allowNull: false, references: { model: Compra, key: 'idCompra' }
  },
  idProducto: {
    type: DataTypes.INTEGER, allowNull: false, references: { model: Producto, key: 'idProducto' }
  },
  cantidad: {
    type: DataTypes.INTEGER, allowNull: false,
  },
  valorProductoCompra: {
    type: DataTypes.DECIMAL(10, 2), allowNull: false,
  },
});

ProductoCompra.belongsTo(Producto, { foreignKey: 'idProducto' });
ProductoCompra.belongsTo(Compra, { foreignKey: 'idCompra' });
Compra.hasMany(ProductoCompra, { foreignKey: 'idCompra' });

export {ProductoCompra};
```

Configurar los controladores

Crear una carpeta 'Controllers' para los archivos que contendrán los métodos HTTP de cada modelo.

Crear el archivo productosController.js para los métodos del modelo Producto: Se crea el método para obtener todos los productos registrados:

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

Se crea el método para obtener un producto en específico por su id:

```
const getProductoById = async (req, res) => {
  const { idProducto } = req.params;
  try {
    const producto = await Producto.findByPk(idProducto);
    res.status(200).json(producto);
  } catch (error) {
    res.status(400).json({ mensaje: `Error al obtener el producto. ${error}` });
  }
};
```

Se crea el método para registrar un nuevo producto:

```
const postProducto = async (req, res) => {
  const { nombre, precio, detalle, tipo, imagen } = req.body;
  try {
    const newProducto = await Producto.create({ nombre, precio, detalle, tipo, imagen });
    res.status(200).json(newProducto);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
}
```

Se crea el método para actualizar un producto con su id:

```
const putProducto = async (req, res) => {
  const { idProducto } = req.params;
  const { nombre, precio, detalle, tipo, imagen } = req.body;
  try {
    const oldProducto = await Producto.findByPk(idProducto);
    oldProducto.nombre = nombre;
    oldProducto.precio = precio;
    oldProducto.detalle = detalle;
    oldProducto.tipo = tipo;
    oldProducto.imagen = imagen;

    const modProducto = await oldProducto.save();
    res.status(200).json(modProducto);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Se crea un método para eliminar un producto con su id:

```
const deleteProducto = async (req, res) => {
  const { idProducto } = req.params;
  try {
    const producto = await Producto.findByPk(idProducto);
    if (producto) {
      const respuesta = await producto.destroy();
      res.status(200).json({ message: `Producto con id ${idProducto} eliminado exitosamente. ${respuesta}` });
    } else {
      res.status(404).json({ message: 'Producto no encontrado.' });
    }
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `No se pudo eliminar. ${error}` });
  }
};
```

Y, por último, se exportan los métodos definidos:

```
export {getProductos, getProductoById, getProductosTipo, postProducto, putProducto, deleteProducto}
```

Ahora, crear el archivo usuariosController.js para definir los métodos necesarios para el modelo 'Usuario':

Crear el método para obtener los datos de un usuario con su id:

```
const getUsuarioById = async (req, res) => {
  const { idUsuario } = req.params;
  try {
    const usuario = await Usuario.findById(idUsuario);
    res.status(200).json(usuario);
  } catch (error) {
    res.status(400).json({ mensaje: `Error al obtener el Usuario. ${error}` });
  }
};
```

Obtener los datos de un usuario mediante un email, puede servir para verificar si el email está registrado en la base de datos y comparar su contraseña:

```
const getUsuarioEmail = async (req, res) => {
  const { email } = req.params;
  try {
    const usuario = await Usuario.findOne({ where: { email } });
    res.status(200).json(usuario);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `Error al obtener el Usuario. ${error}` });
  }
};
```

Método para registrar un usuario

```
const postUsuario = async (req, res) => {
  const { email, password, nombres, direccion, rol } = req.body;
  try {
    const newUser = await Usuario.create({ email, password, nombres, direccion, rol });
    res.status(200).json(newUser);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Por último, se exportan los métodos:

```
export {getUsuarioById, getUsuarioEmail, postUsuario}
```

Crear el archivo productosCarritoController para definir los métodos para gestionar el carrito de compras.

Crear un método para obtener los productos agregados a un carrito de compras por un usuario:


```
const getProductosCarritoUsuario = async (req, res) => {
  const { idUsuario } = req.params;
  try {
    const productosCarrito = await ProductoCarrito.findAll({
      where: { idUsuario },
      include: [{
        model: Producto, // attributes: ['idUsuario', 'em
      ]},
    });
    res.status(200).json(productosCarrito);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Crear un método para agregar un producto al carrito de compras:

```
const postProductoCarrito = async (req, res) => {
  const { idUsuario, idProducto, cantidad, valorProductoCarrito } = req.body;
  try {
    const productoCarrito = await ProductoCarrito.create({ idUsuario, idProducto, cantidad,
      valorProductoCarrito });
    res.status(201).json(productoCarrito);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Crear un método para actualizar un producto del carrito de compras:

```
const putProductoCarrito = async (req, res) => {
  const { idProductoCarrito } = req.params;
  const { cantidad, valorProductoCarrito } = req.body;
  try {
    const oldProductoCarrito = await ProductoCarrito.findByPk(idProductoCarrito);
    oldProductoCarrito.cantidad = cantidad;
    oldProductoCarrito.valorProductoCarrito = valorProductoCarrito;

    const modProductoCarrito = await oldProductoCarrito.save();
    res.status(200).json(modProductoCarrito);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Crear un método para eliminar un producto del carrito de compras:

```
const deleteProductoCarrito = async (req, res) => {
  const { idProductoCarrito } = req.params;
  try {
    const productoCarrito = await ProductoCarrito.findByPk(idProductoCarrito);
    if (productoCarrito) {
      const respuesta = await productoCarrito.destroy();
      res.status(200).json(respuesta);
    } else {
      res.status(404).json({ message: 'Prod no encontrado.' });
    }
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `No se pudo eliminar. ${error}` });
  }
};
```

Crear un método para vaciar los productos del carrito de compras de un usuario:

```
const deleteCarrito = async (req, res) => {
  const { idUsuario } = req.params;
  try {
    await ProductoCarrito.destroy({
      where: { idUsuario },
    });
    res.status(200).json({ mensaje: 'Productos del carrito eliminados con éxito.' });
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `No se pudo eliminar. ${error}` });
  }
};
```

Por último, se exportan los métodos:

```
export {
  getProductosCarritoUsuario, postProductoCarrito, putProductoCarrito, deleteProductoCarrito,
  deleteCarrito };
```

Crear el archivo comprasController.js para definir los métodos correspondientes al modelo

Compra: Crear el método para obtener los datos de una compra con su id:

```
const getCompraById = async (req, res) => {
  const { idCompra } = req.params;
  try {
    const compra = await Compra.findAll({
      where: { idCompra },
    });
    res.status(200).json(compra);
  } catch (error) {
    res.status(400).json({ mensaje: `Error al obtener la compra. ${error}` });
  }
};
```

En caso que sea necesario se define un método para obtener datos de compras de un usuario:

```
const getComprasUsuario = async (req, res) => {
  const { idUsuario } = req.params;
  try {
    const compras = await Compra.findAll({
      where: { idUsuario },
    });
    res.status(200).json(compras);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: `Error al obtener la Compra. ${error}` });
  }
};
```

Agregar una compra a la base de datos:

```
const postCompra = async (req, res) => {
  const { idUsuario, valorTotal, fecha } = req.body;
  try {
    const compra = await Compra.create({ idUsuario, valorTotal, fecha });
    res.status(201).json(compra);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
}
```

Por último, se exportan los métodos definidos:

```
export { getCompraById, getComprasUsuario, postCompra }
```

Crear un archivo productosComprasController.js para definir los métodos necesarios para el modelo 'ProductoCompra':

Obtener los productos de una compra:

```
const getProductosCompradosByCompraId = async (req, res) => {
  const { idCompra } = req.params;
  try {
    const productosComprados = await ProductoCompra.findAll({
      where: { idCompra },
      order: [['createdAt', 'DESC']],
    });
    res.status(200).json(compras);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
};
```

Agregar productos comprados con el id (factura) de la compra:

```
const postProductoCompra = async (req, res) => {
  const { idCompra, idProducto, cantidad, valorProductoCompra } = req.body;
  try {
    const productoCompra = await ProductoCompra.create({ idCompra, idProducto, cantidad, valorProductoCompra });
    res.status(201).json(productoCompra);
  } catch (error) {
    console.error(error);
    res.status(400).json({ mensaje: error });
  }
}
```

Se exportan los métodos:

```
export { getProductosCompradosByCompraId, postProductoCompra }
```

Estos serían los principales métodos que serán utilizados para gestionar los diferentes modelos en la base de datos, en caso que se necesiten otros diferentes ya se crearán en su momento.

Crear las rutas del proyecto Backend

Ahora, se procede a crear una carpeta 'Routes', donde se guardará en el archivo routes.js las diferentes rutas que servirán para invocar a los métodos definidos antes.

Se importan los diferentes métodos creados en los controladores para luego definir las rutas. Además, se debe crear una instancia del enrutador de Express: router=Router()

```
Routes > routes.js > ...
1  import { Router } from 'express';
2  import { getProductos, getProductoById, postProducto, putProducto, deleteProducto } from '../
  Controllers/productosController.js';
3  import { getUsuarioById, getUsuarioEmail, postUsuario } from '../Controllers/usuariosController.js';
4  import { getCompraById, getComprasUsuario, postCompra } from '../Controllers/comprasController.js';
5  import { getProductosCompradosByCompraId, postProductoCompra } from '../Controllers/
  productosCompradosController.js';
6
7  const router = Router();
```

Se definen las rutas para el modelo Producto:

```
router.get("/productos", getProductos);
router.get("/productos/:idProducto", getProductoById);
router.post("/productos", postProducto);
router.put("/productos/:idProducto", putProducto);
router.delete("/productos/:idProducto", deleteProducto);
```

Se definen las rutas para el modelo Usuario:

```
router.get("/usuario/:idUsuario", getUsuarioById);
router.get("/usuario_email/:email", getUsuarioEmail);
router.post("/usuario", postUsuario);
```

Se definen las rutas para el modelo ProductoCarrito:

```
router.get("/productos_carrito/:idUsuario", getProductosCarritoUsuario);
router.post("/producto_carrito", postProductoCarrito);
router.put("/producto_carrito/:idProductoCarrito", putProductoCarrito);
router.delete("/producto_carrito/:idProductoCarrito", deleteProductoCarrito);
router.delete("/carrito/:idUsuario", deleteCarrito);
```

Se definen las rutas para el modelo Compra:

```
router.get("/compra/:idCompra", getCompraById);
router.get("/compras_usuario/:idUsuario", getComprasUsuario);
router.post("/compra", postCompra);
```

Se definen las rutas para el modelo ProductoCompra:

```
router.get("/productos_comprados/:idCompra", getProductosCompradosByCompraId);
router.post("/producto_compra", postProductoCompra);
```

En caso de necesitar nuevas rutas se agregarán en su momento.

Por último, en este archivo routes.js se exporta el enrutador:

```
export default router;
```

Crear el servidor del Backend

Crear el archivo server.js en la raíz del proyecto en el que se debe importar express, el enrutador y la configuración de la base de datos con sequelize, y se crea una instancia de la app con express:

```
import express from "express";
import router from "../Routes/routes.js";
import { sequelize } from "../Database/database.js";
import cors from 'cors';

const app = express();
app.use(cors());
app.use(express.json());
app.use(router);
app.set('port', 3000);
```

Es muy importante tener en cuenta `app.use(cors())`, ya que así se habilita el acceso a recursos desde diferentes orígenes, lo que es útil cuando se tiene un frontend como cliente y un backend (servidor) que se ejecutan en diferentes dominios.

`express.json()` permite a la aplicación procesar datos en formato JSON enviados en el cuerpo de las solicitudes HTTP.

Se usa el router creado anteriormente y se establece el puerto 3000.

Al final, se crea una función `testDb` que se encarga de establecer una conexión con la base de datos utilizando Sequelize, para sincronizar los modelos con la base de datos. Luego, la función inicia el servidor Express para que la aplicación esté lista para recibir solicitudes HTTP.

```
const testDb = async () => {
  try {
    // await sequelize.authenticate();
    await sequelize.sync();
    console.log('La conexión fue exitosa');
    app.listen(app.get('port'), () => {
      console.log(`Backend en http://localhost:${app.get('port')}`);
    });
  } catch (error) {
    console.error(`La conexión presentó un error. ${error}`);
  }
};
testDb();
```

Iniciar el proyecto Backend

Al terminar de definir todo el proyecto, se probará su funcionamiento mediante el comando: `npm run start` el cual ejecutamos en una terminal dentro de la raíz del proyecto.

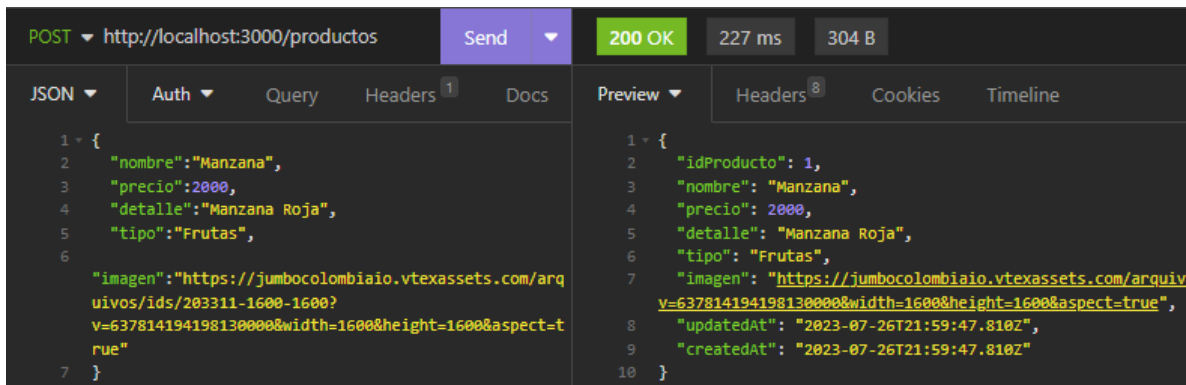
```
PS C:\TallerFruverAlex\Backend> npm run start
> backend@1.0.0 start
> nodemon server.js

[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 server.js`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Productos' AND TABLE_SCHEMA = 'alexfruver'
Executing (default): SHOW INDEX FROM `Productos`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Usuarios' AND TABLE_SCHEMA = 'alexfruver'
Executing (default): SHOW INDEX FROM `Usuarios`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'Compras' AND TABLE_SCHEMA = 'alexfruver'
Executing (default): SHOW INDEX FROM `Compras`
Executing (default): SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE = 'BASE TABLE' AND TABLE_NAME = 'ProductoCompras' AND TABLE_SCHEMA = 'alexfruver'
Executing (default): SHOW INDEX FROM `ProductoCompras`
La conexión fue exitosa
Backend en http://localhost:3000
```

Como podemos ver ya está corriendo con éxito el Backend, por lo que se procederá a hacer verificaciones a través de un cliente gráfico (Insomnia).

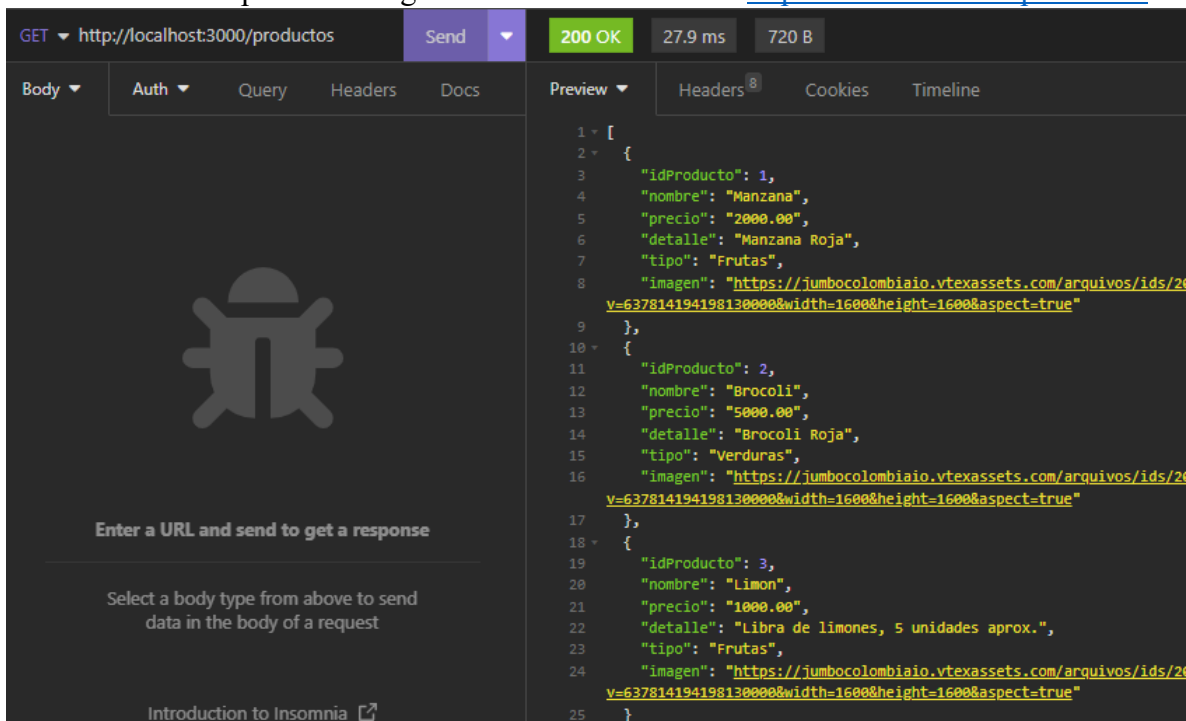
Verificación de las operaciones a través de Insomnia

Verificamos las rutas para el modelo 'Producto'. Iniciamos registrando un nuevo producto, mediante la ruta: <http://localhost:3000/productos> por POST



Y como se puede ver responde exitosamente.

Vamos a listar los productos registrados mediante la ruta: <http://localhost:3000/productos>



Vamos a actualizar el producto con id:2

```
PUT http://localhost:3000/productos/2 200 OK 21.8 ms 234 B
JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies Timeline
1 {
2   "nombre": "Brocoli",
3   "precio": 7000.00,
4   "detalle": "Brocoli en bandeja",
5   "tipo": "Verduras",
6   "imagen": "https://jumbocolombiaoio.vtexassets.com/arquivos/ids/209211-1600-1600?v=637814209992700000&width=1600&height=1600&aspect=true"
7 }
```

Verificamos cambios listando el producto con id:2

```
GET http://localhost:3000/productos/2 200 OK 32.5 ms 239 B
Body Auth Query Headers Docs Preview Headers 8 Cookies Timeline
1 {
2   "idProducto": 2,
3   "nombre": "Brocoli",
4   "precio": "7000.00",
5   "detalle": "Brocoli en bandeja",
6   "tipo": "Verduras",
7   "imagen": "https://jumbocolombiaoio.vtexassets.com/arquivos/ids/209211-1600-1600?v=637814209992700000&width=1600&height=1600&aspect=true"
8 }
```

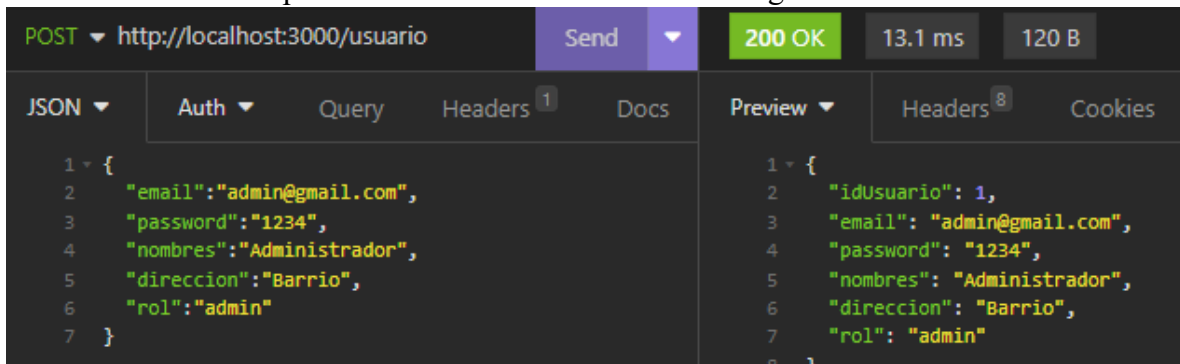
Verificamos eliminar el producto con id:3

```
DELETE http://localhost:3000/productos/3 200 OK 30.1 ms 91 B
Body Auth Query Headers Docs Preview Headers 8 Cookies Timeline
1 {
2   "message": "Producto con id 3 eliminado exitosamente."
3 }
```

Y verificamos si existe el producto con id:3 en la base de datos:

```
GET http://localhost:3000/productos/3 200 OK 7.1 ms 4 B
Body Auth Query Headers Docs Preview Headers 8 Cookies Timeline
1 null
```


Verificamos las rutas para el modelo 'Usuario. Iniciamos registrando un nuevo usuario:



The screenshot shows a REST client interface with a POST request to `http://localhost:3000/usuario`. The request body is a JSON object with the following fields: `email`, `password`, `nombres`, `direccion`, and `rol`. The response is a 200 OK status with a response time of 13.1 ms and a body size of 120 B. The response body is a JSON object with the following fields: `idUsuario`, `email`, `password`, `nombres`, `direccion`, and `rol`.

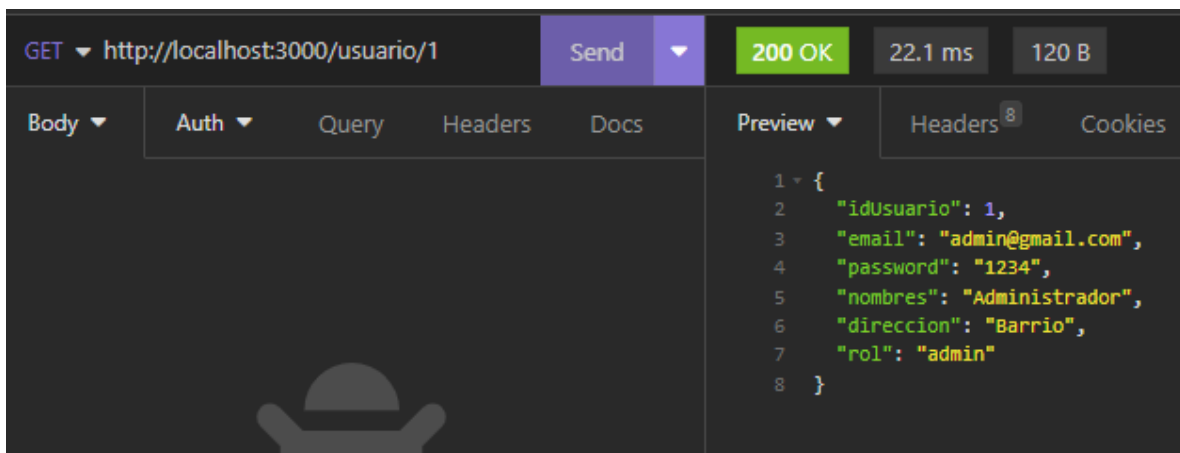
```
POST http://localhost:3000/usuario Send 200 OK 13.1 ms 120 B
```

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

```
1 {
2   "email": "admin@gmail.com",
3   "password": "1234",
4   "nombres": "Administrador",
5   "direccion": "Barrio",
6   "rol": "admin"
7 }
```

```
1 {
2   "idUsuario": 1,
3   "email": "admin@gmail.com",
4   "password": "1234",
5   "nombres": "Administrador",
6   "direccion": "Barrio",
7   "rol": "admin"
8 }
```

Verificamos buscar los datos de un usuario mediante un id:1



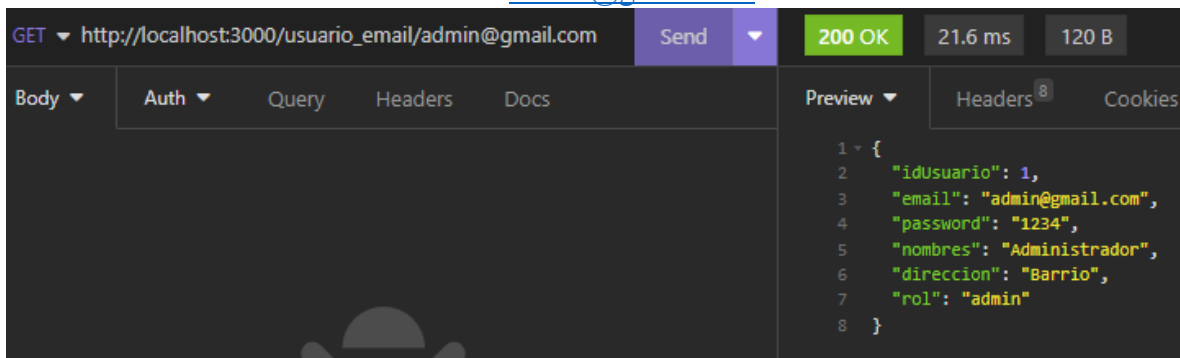
The screenshot shows a REST client interface with a GET request to `http://localhost:3000/usuario/1`. The response is a 200 OK status with a response time of 22.1 ms and a body size of 120 B. The response body is a JSON object with the following fields: `idUsuario`, `email`, `password`, `nombres`, `direccion`, and `rol`.

```
GET http://localhost:3000/usuario/1 Send 200 OK 22.1 ms 120 B
```

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

```
1 {
2   "idUsuario": 1,
3   "email": "admin@gmail.com",
4   "password": "1234",
5   "nombres": "Administrador",
6   "direccion": "Barrio",
7   "rol": "admin"
8 }
```

Y verificamos si funciona con el email admin@gmail.com



The screenshot shows a REST client interface with a GET request to `http://localhost:3000/usuario_email/admin@gmail.com`. The response is a 200 OK status with a response time of 21.6 ms and a body size of 120 B. The response body is a JSON object with the following fields: `idusuario`, `email`, `password`, `nombres`, `direccion`, and `rol`.

```
GET http://localhost:3000/usuario_email/admin@gmail.com Send 200 OK 21.6 ms 120 B
```

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

```
1 {
2   "idusuario": 1,
3   "email": "admin@gmail.com",
4   "password": "1234",
5   "nombres": "Administrador",
6   "direccion": "Barrio",
7   "rol": "admin"
8 }
```

Verificamos agregar un producto al carrito, desde el idUsuario:2

```
POST http://localhost:3000/producto_carrito Send 201 Created 172 ms 94 B
JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies
1 { "idUsuario":2, "idProducto":2, "cantidad":3, "valorProductoCarrito":21000}
```

```
1 {
2   "idProductoCarrito": 1,
3   "idUsuario": 2,
4   "idProducto": 2,
5   "cantidad": 3,
6   "valorProductoCarrito": 21000
7 }
```

Verificamos listar los productos en carrito del usuario con id:2

```
GET http://localhost:3000/productos_carrito/2 Send 200 OK 11.7 ms 352 B
Body Auth Query Headers Docs Preview Headers 8 Cookies Tim
1 [
2   {
3     "idProductoCarrito": 1,
4     "idUsuario": 2,
5     "idProducto": 2,
6     "cantidad": 3,
7     "valorProductoCarrito": "21000.00",
8     "Producto": {
9       "idProducto": 2,
10      "nombre": "Brocoli",
11      "precio": "7000.00",
12      "detalle": "Brocoli en bandeja",
13      "tipo": "Verduras",
14      "imagen": "https://jumbocolombiaid
1600-1600?v=637814209992700000&width=1600"
15    }
16  }
17 ]
```

Verificamos agregando otro producto al carrito

```
POST http://localhost:3000/producto_carrito Send 201 Created 18.1 ms 93 B
JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies
1 { "idUsuario":2, "idProducto":1, "cantidad":2, "valorProductoCarrito":4000}
```

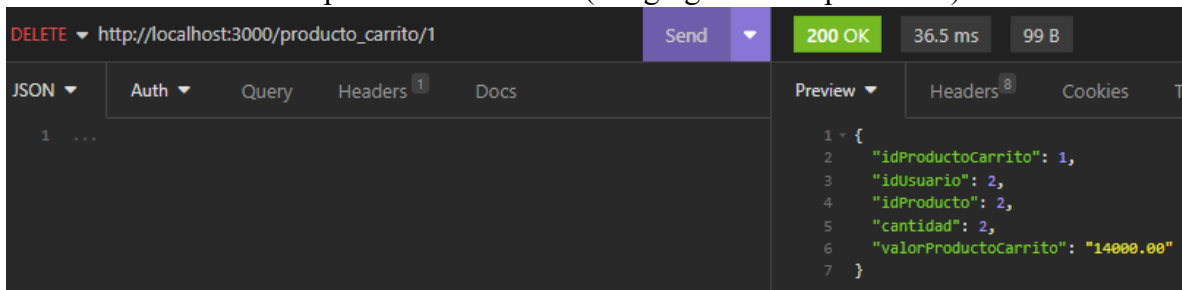
```
1 {
2   "idProductoCarrito": 2,
3   "idUsuario": 2,
4   "idProducto": 1,
5   "cantidad": 2,
6   "valorProductoCarrito": 4000
7 }
```

Verificamos actualizar un producto del carrito con id:1

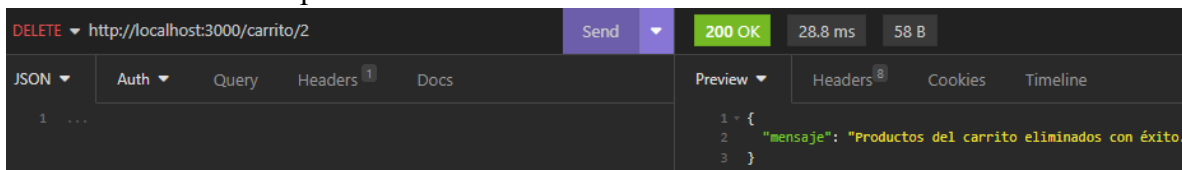
```
PUT http://localhost:3000/producto_carrito/1 Send 200 OK 18 ms 94 B
JSON Auth Query Headers 1 Docs Preview Headers 8 Cookies
1 { "idUsuario":2, "idProducto":1, "cantidad":2, "valorProductoCarrito":14000}
```

```
1 {
2   "idProductoCarrito": 1,
3   "idUsuario": 2,
4   "idProducto": 2,
5   "cantidad": 2,
6   "valorProductoCarrito": 14000
7 }
```

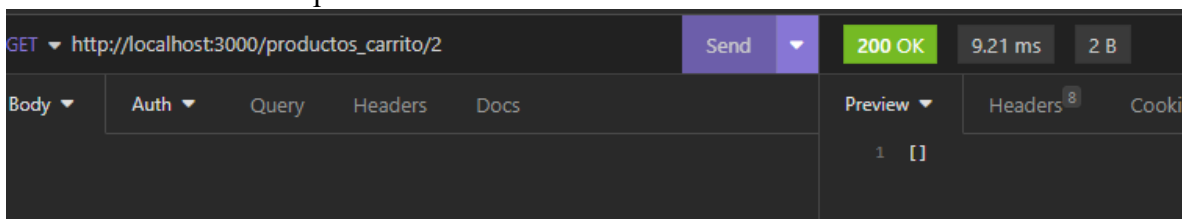
Verificamos eliminar un producto del carrito (se agregaron más productos):



Verificamos vaciar los productos del carrito del usuario mediante el id del Usuario:2

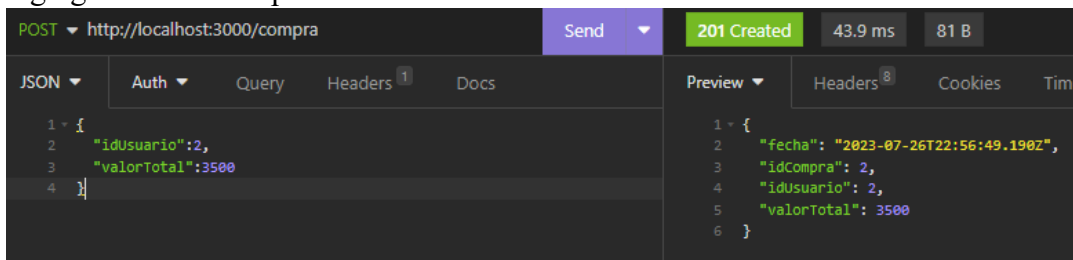


Volvemos a obtener los productos del carrito del usuario con id:2

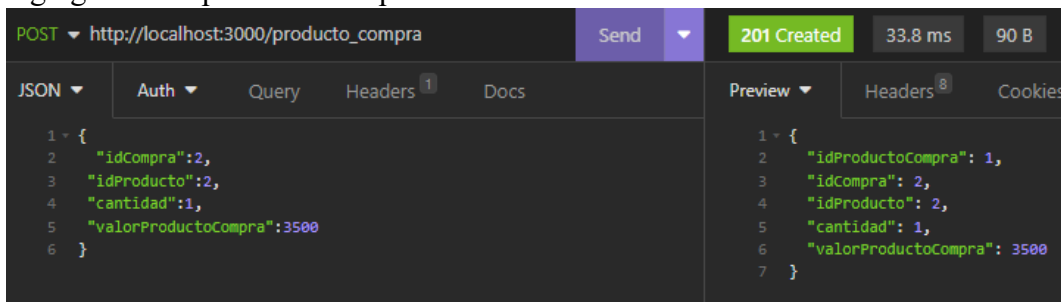


Como podemos corroborar, las operaciones dan buenos resultados. Ahora verificaremos las rutas para los modelos 'Compra' y 'ProductoCompra'.

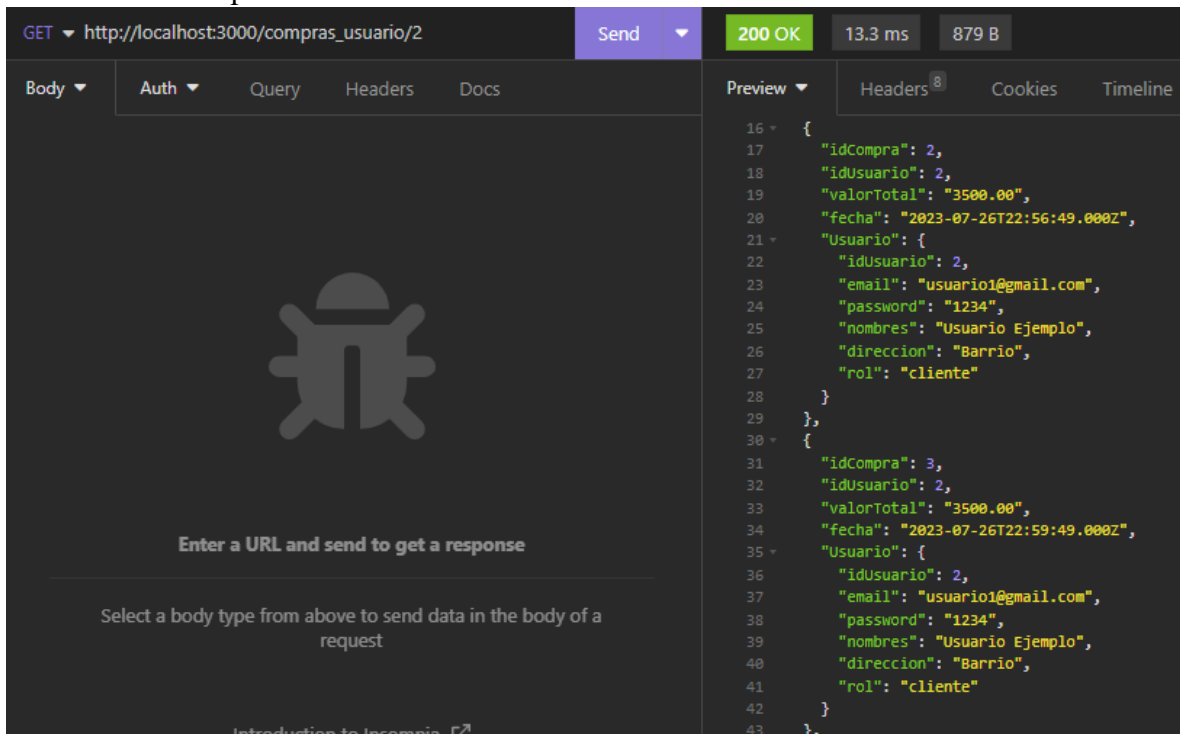
Agregamos una compra



Agregamos un producto comprado



Listamos las compras del usuario con id:2



GET `http://localhost:3000/compras_usuario/2` Send 200 OK 13.3 ms 879 B

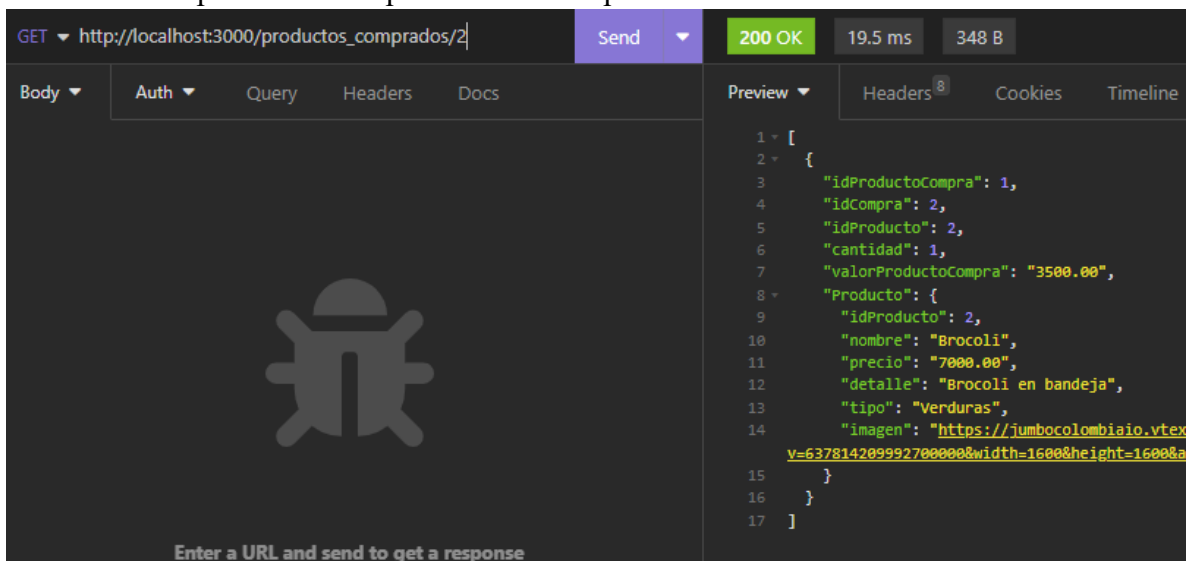
Body Auth Query Headers Docs Preview Headers 8 Cookies Timeline

Enter a URL and send to get a response

Select a body type from above to send data in the body of a request

```
16 {
17   "idCompra": 2,
18   "idUsuario": 2,
19   "valorTotal": "3500.00",
20   "fecha": "2023-07-26T22:56:49.000Z",
21   "Usuario": {
22     "idUsuario": 2,
23     "email": "usuario1@gmail.com",
24     "password": "1234",
25     "nombres": "Usuario Ejemplo",
26     "direccion": "Barrio",
27     "rol": "cliente"
28   }
29 },
30 {
31   "idCompra": 3,
32   "idUsuario": 2,
33   "valorTotal": "3500.00",
34   "fecha": "2023-07-26T22:59:49.000Z",
35   "Usuario": {
36     "idUsuario": 2,
37     "email": "usuario1@gmail.com",
38     "password": "1234",
39     "nombres": "Usuario Ejemplo",
40     "direccion": "Barrio",
41     "rol": "cliente"
42   }
43 },
```

Y listamos los productos comprados en la compra con id:2



GET `http://localhost:3000/productos_comprados/2` Send 200 OK 19.5 ms 348 B

Body Auth Query Headers Docs Preview Headers 8 Cookies Timeline

Enter a URL and send to get a response

```
1 [
2   {
3     "idProductoCompra": 1,
4     "idCompra": 2,
5     "idProducto": 2,
6     "cantidad": 1,
7     "valorProductoCompra": "3500.00",
8     "Producto": {
9       "idProducto": 2,
10      "nombre": "Brocoli",
11      "precio": "7000.00",
12      "detalle": "Brocoli en bandeja",
13      "tipo": "Verduras",
14      "imagen": "https://iumbocolombiaio.vtex
15              v=637814209992700000&width=1600&height=1600&a
16      }
17    ]
```

Así concluimos el proyecto Backend con las rutas y métodos funcionales para ser aplicados al proyecto en Frontend.