

Taller Unidad 2 Backend

Gustavo Teran Muñoz

Universidad de Nariño

Ingeniería de Sistemas.

**Diplomado de actualización en nuevas tecnologías para el
Desarrollo de Software.**

1. Crear 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.

En primer lugar se utiliza la herramienta de MYSql para crear nuestra base de datos en este caso la base de datos se llama **fruver** , y contendrá tablas para almacenar la respectiva información. A continuación se muestran las instrucciones para crear la base de Datos y las respectivas tablas para su uso.

Como primer paso se utiliza la institución para crear la base de datos
CREATE DATABASE fruver;

Una vez creada la base de datos se crea la tabla para registrar Pedidos con la siguiente estructura

```
CREATE TABLE `pedidos` (  
  `idPedido` int(11) NOT NULL,  
  `cliente` varchar(255) NOT NULL,  
  `correo` varchar(255) NOT NULL,  
  `direccion` varchar(255) NOT NULL,  
  `producto` varchar(255) NOT NULL,  
  `cantidad` int(11) NOT NULL,  
  `precio` decimal(10,0) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Una vez creada la base de datos se crea la tabla para registrar Productos con la siguiente estructura

```
CREATE TABLE `productos` (  
  `idProducto` int(11) NOT NULL,  
  `nombre` varchar(255) NOT NULL,  
  `detalle` varchar(255) NOT NULL,  
  `cantidad` int(11) NOT NULL,  
  `precio` decimal(10,0) NOT NULL,  
  `imagen` varchar(255) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Una vez creada la base de datos se crea la tabla para registrar usuarios con la siguiente estructura

```
CREATE TABLE `usuarios` (  
  `idUserio` int(11) NOT NULL,  
  `nombre` varchar(255) NOT NULL,  
  `direccion` varchar(255) NOT NULL,
```

```

`correo` varchar(255) NOT NULL,
`contrasena` varchar(255) NOT NULL,
`rol` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

```

Ahora se asignan las respectivas sentencias e índices para las tablas creadas anteriormente con las instrucciones tanto para llaves primarias y para que se marque id autoincremental. :

```

-- Indices de la tabla `pedidos`
ALTER TABLE `pedidos`
  ADD PRIMARY KEY (`idPedido`);

```

```

-- Indices de la tabla `productos`
ALTER TABLE `productos`
  ADD PRIMARY KEY (`idProducto`);

```

```

-- Indices de la tabla `usuarios`
ALTER TABLE `usuarios`
  ADD PRIMARY KEY (`idUsuario`);

```

```

-- AUTO_INCREMENT de las tablas volcadas
-- AUTO_INCREMENT de la tabla `pedidos`

```

```

ALTER TABLE `pedidos`
  MODIFY `idPedido` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;

```

```

-- AUTO_INCREMENT de la tabla `productos`
ALTER TABLE `productos`
  MODIFY `idProducto` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=14;

```

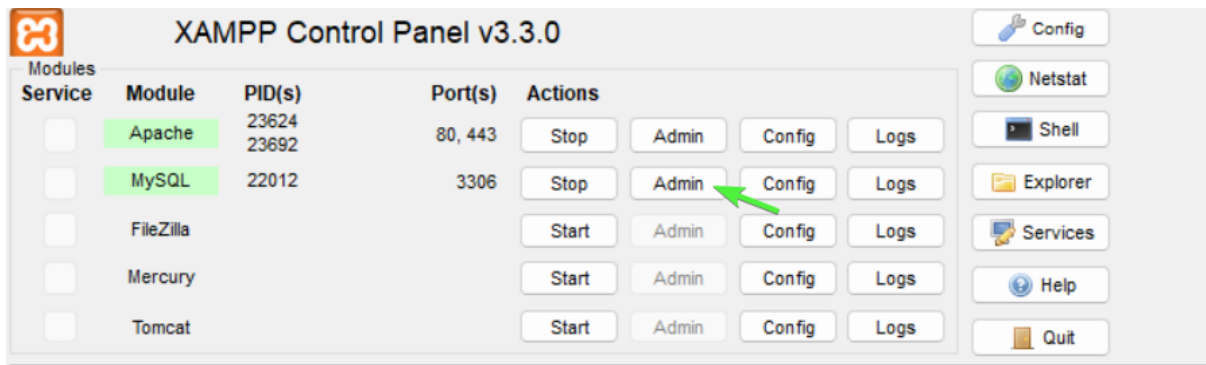
```

-- AUTO_INCREMENT de la tabla `usuarios`
ALTER TABLE `usuarios`
  MODIFY `idUsuario` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=4;
COMMIT;

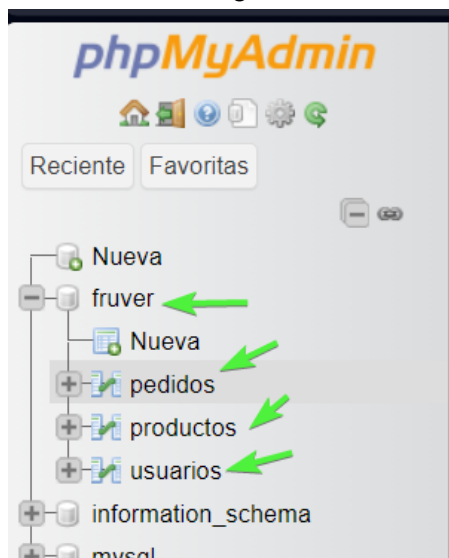
```

Para este respectivo trabajo se utilizó la herramienta de Xampp la cual tiene acceso a administrar bases de datos Mysql

Se levanta el servicio de Mysql



Una vez se ejecuta el script de la tablas se observa en pantalla la creación de la base de datos con la configuración de las tabla creadas.



2. Desarrollar una aplicación Backend implementada en NodeJS 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 (La empresa debe contar con un nombre). Se debe hacer uso correcto de los verbos HTTP dependiendo de la tarea a realizar.

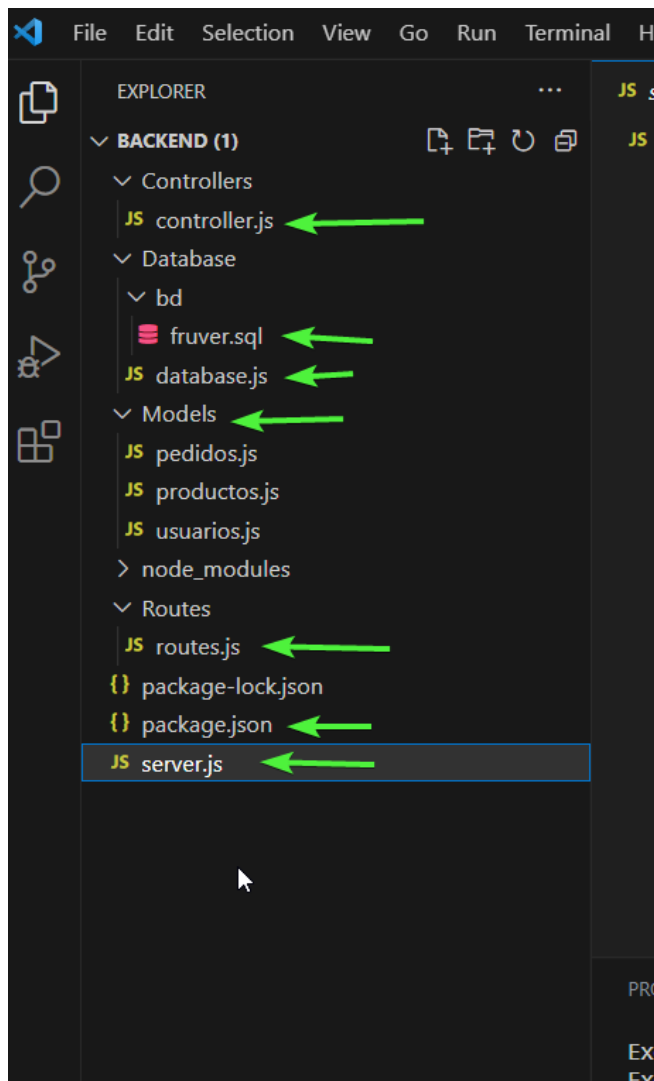
Para desarrollar un aplicación backend se utilizará el editor de código visual studio code y además se llevará a cabo con las tecnologías de NodeJs Y express JS

Tenemos la carpeta de proyecto se inicializa el proyecto Node.js **con `npm init -y`**

Se instalan las dependencias

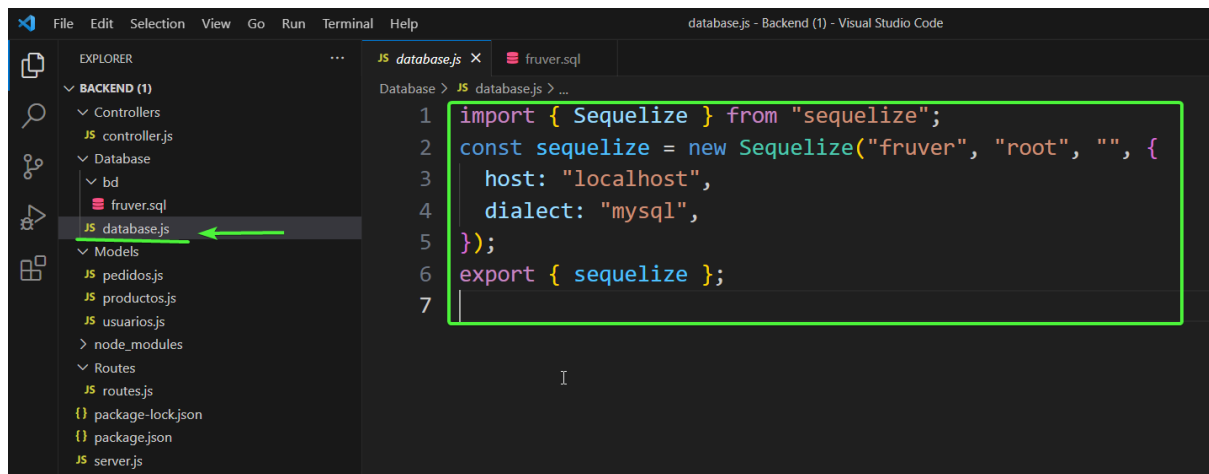
`npm install express mysql2 body-parser`

A continuación se muestra la estructura del proyecto funcional y se explicara cada unos de los componentes y módulos para un mejor entendimiento



Se explica la estructura del proyecto para esto una vez importado los módulos se necesita crear una conexión para la base de datos en este caso está en la carpeta database y el archivo es database.js

El siguiente código crea una instancia de Sequelize para conectarse a una base de datos MySQL llamada "fruver" alojada en "localhost" con el usuario root, sin una contraseña específica.



En el apartado de modelos se encuentran los modelos para las respectivas tablas que se creó en la base de datos como pedidos, productos y usuarios

A continuación se describe la explicación del código el cual se maneja la misma estructura para los más módulos

Este código define un modelo Sequelize para un "Pedido" en una aplicación Node.js. Sequelize es un ORM (Object-Relational Mapping) para Node.js que permite interactuar con bases de datos relacionales utilizando objetos JavaScript, facilitando el trabajo con bases de datos.

1. Importación de módulos necesarios:

- `DataTypes`: Esto es parte de Sequelize y proporciona tipos de datos que se utilizan para definir la estructura de los atributos (campos) del modelo.
- `sequelize`: Esta es una instancia de Sequelize que representa una conexión con la base de datos.

2. Definición del modelo "Pedido":

El método `sequelize.define()` se utiliza para definir el modelo "Pedido" y sus atributos. El método toma dos argumentos:

- El primer argumento es el nombre del modelo, que en este caso es "pedido".
- El segundo argumento es un objeto que define los atributos del modelo.

3. Atributos (campos) del modelo "Pedido":

El modelo tiene los siguientes atributos:

- `idPedido`: Un campo de tipo entero que se autoincrementa y sirve como clave primaria del modelo.
- `cliente`: Un campo de tipo cadena que representa el nombre del cliente. No puede ser nulo.
- `correo`: Un campo de tipo cadena que representa la dirección de correo electrónico del cliente. No puede ser nulo.
- `direccion`: Un campo de tipo cadena que representa la dirección de entrega. No puede ser nulo.

- `producto`: Un campo de tipo cadena que representa el nombre del producto que se está ordenando. No puede ser nulo.
- `cantidad`: Un campo de tipo entero que representa la cantidad del producto que se está ordenando. No puede ser nulo.
- `precio`: Un campo de tipo decimal que representa el precio del producto. No puede ser nulo.

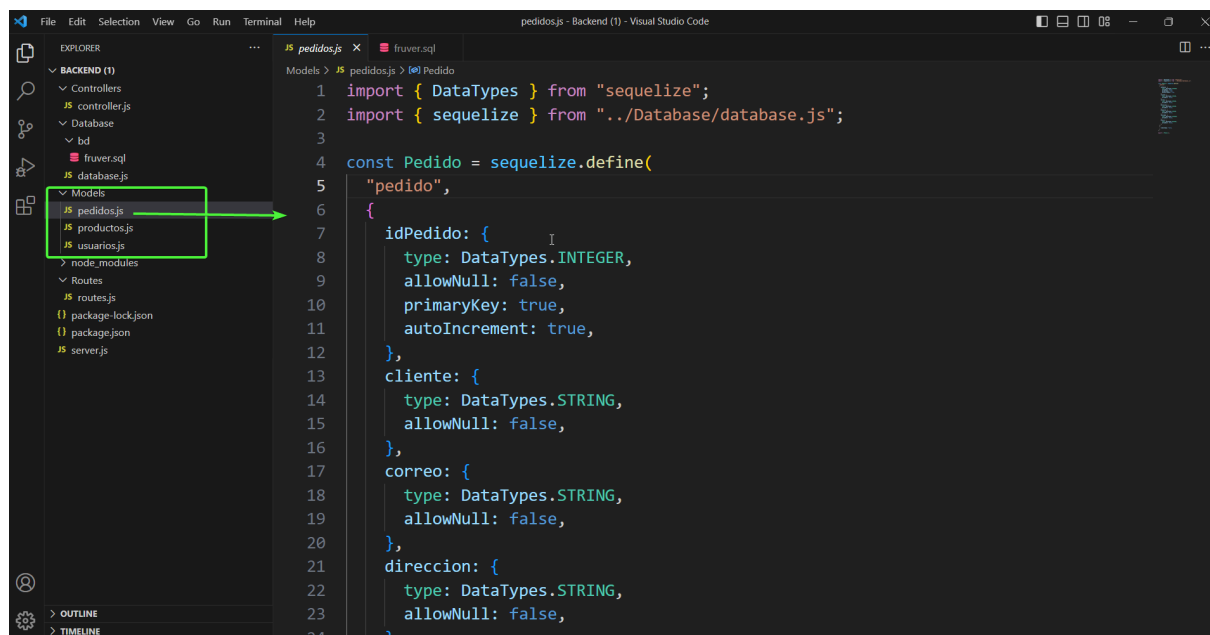
4. Objeto de opciones:

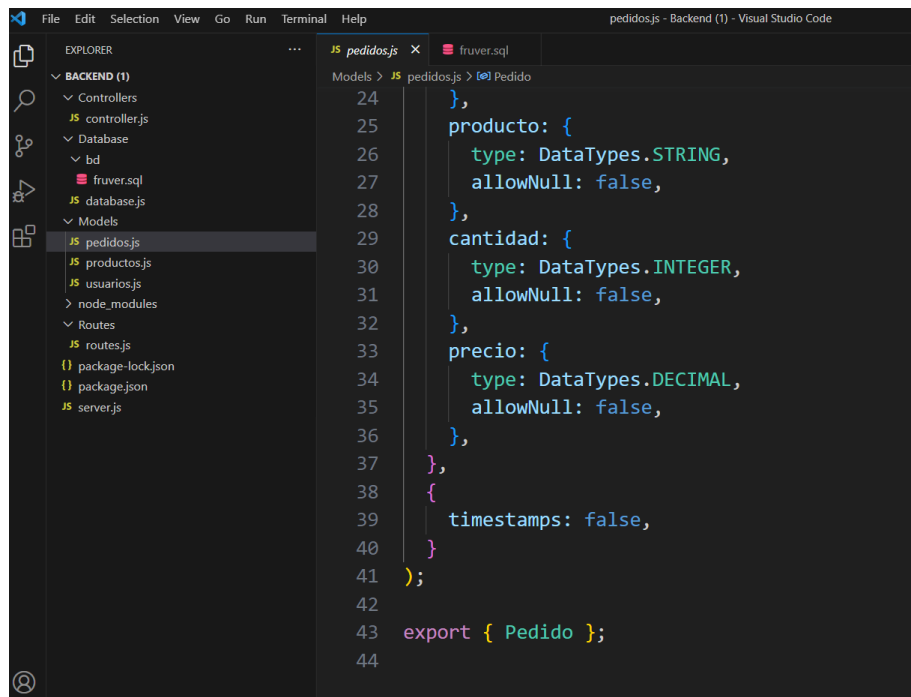
El tercer argumento de `sequelize.define()` es un objeto de opciones, donde se establece `timestamps: false`. Esto significa que Sequelize no agregará automáticamente campos de marca de tiempo (`createdAt` y `updatedAt`) al modelo, que se utilizan comúnmente para rastrear las fechas de creación y modificación de registros.

5. Exportación del modelo "Pedido":

Finalmente, el modelo "Pedido" se exporta utilizando `export { Pedido };`, lo que lo hace disponible para su uso en otras partes de la aplicación.

Con esta definición del modelo, puedes realizar varias operaciones de base de datos en el modelo "Pedido", como crear, actualizar, eliminar y consultar pedidos en la base de datos.



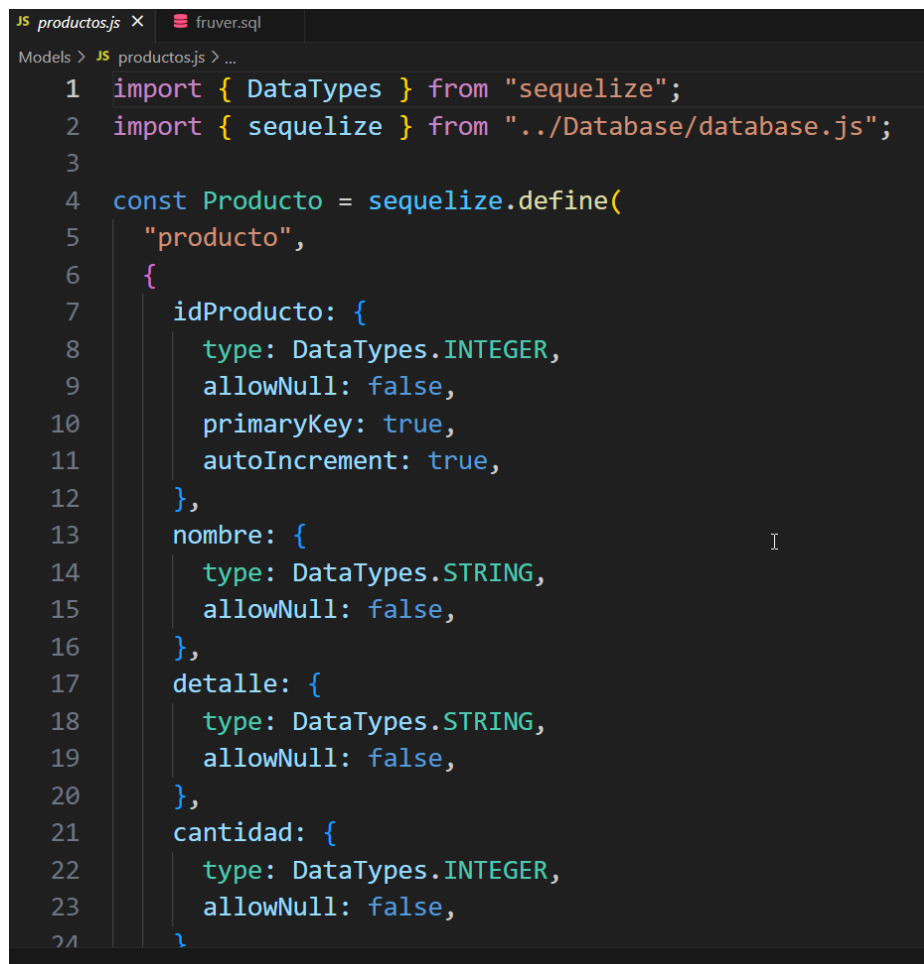


```
File Edit Selection View Go Run Terminal Help
pedidos.js - Backend (1) - Visual Studio Code

EXPLORER
BACKEND (1)
  Controllers
  Database
  bd
  fruver.sql
  database.js
  Models
    pedidos.js
    productos.js
    usuarios.js
  node_modules
  Routes
  routes.js
  package-lock.json
  package.json
  server.js

JS pedidos.js x fruver.sql
Models > JS pedidos.js > Pedido
24 },
25 producto: {
26   type: DataTypes.STRING,
27   allowNull: false,
28 },
29 cantidad: {
30   type: DataTypes.INTEGER,
31   allowNull: false,
32 },
33 precio: {
34   type: DataTypes.DECIMAL,
35   allowNull: false,
36 },
37 },
38 {
39   timestamps: false,
40 }
41 );
42
43 export { Pedido };
44
```

Módulo de productos



```
JS productos.js x fruver.sql
Models > JS productos.js > ...
1 import { DataTypes } from "sequelize";
2 import { sequelize } from "../Database/database.js";
3
4 const Producto = sequelize.define(
5   "producto",
6   {
7     idProducto: {
8       type: DataTypes.INTEGER,
9       allowNull: false,
10      primaryKey: true,
11      autoIncrement: true,
12    },
13     nombre: {
14       type: DataTypes.STRING,
15       allowNull: false,
16     },
17     detalle: {
18       type: DataTypes.STRING,
19       allowNull: false,
20     },
21     cantidad: {
22       type: DataTypes.INTEGER,
23       allowNull: false,
24     }
25   }
26 );
27 export { Producto };
28
```


Modulo de usuarios

```
JS usuarios.js x fruver.sql
Models > JS usuarios.js > [?] Usuario > [?] idUsuario
1 import { DataTypes } from "sequelize";
2 import { sequelize } from "../Database/database.js";
3
4 const Usuario = sequelize.define(
5   "usuario",
6   {
7     idUsuario: {
8       type: DataTypes.INTEGER,
9       allowNull: false,
10      primaryKey: true,
11      autoIncrement: true,
12    },
13    nombre: {
14      type: DataTypes.STRING,
15      allowNull: false,
16    },
17    direccion: {
18      type: DataTypes.STRING,
19      allowNull: false,
20    },
21    correo: {
22      type: DataTypes.STRING,
23      allowNull: false,
24    }
25  }
26 );
```

Ln 10, C

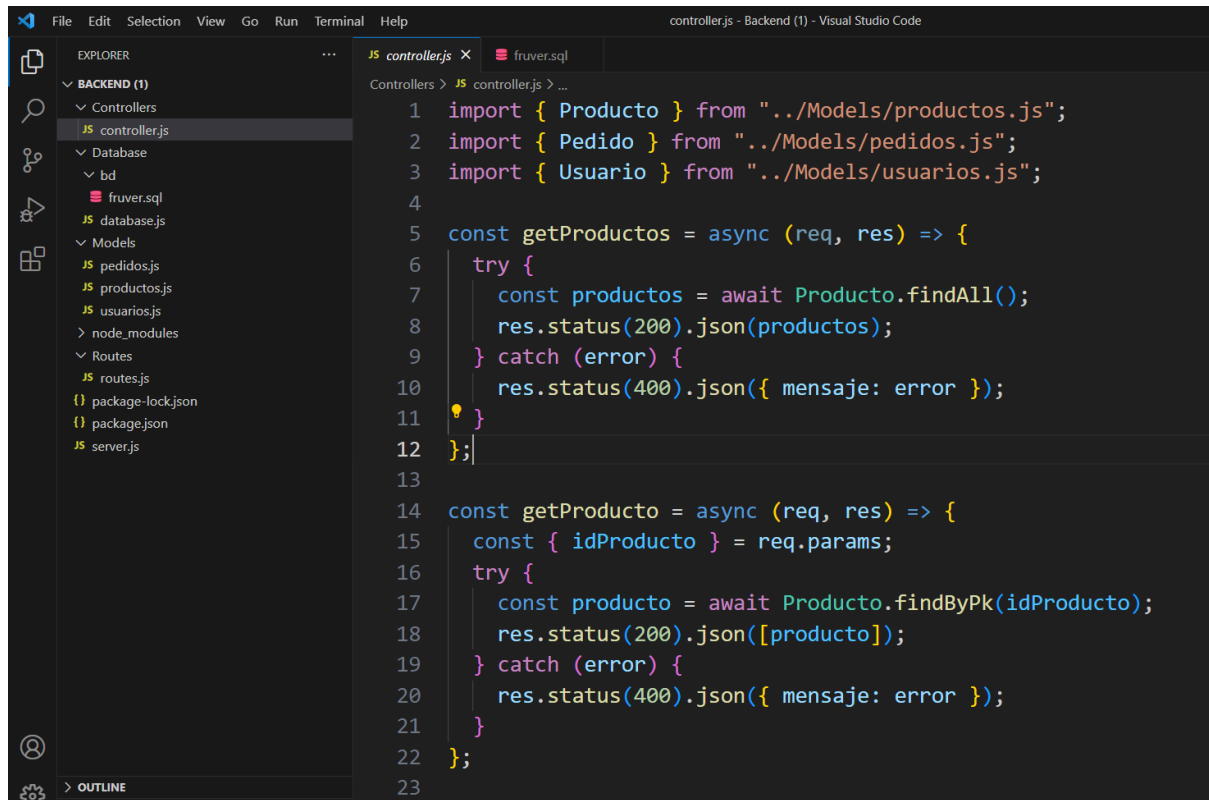
Ahora se crea el controlador para manejar solicitudes y respuestas HTTP en un servidor backend. Se implementan operaciones de CRUD para productos, pedidos y usuarios en un servidor web. Forma parte de un servidor backend diseñado para interactuar con una base de datos y proporcionar datos a un cliente frontend.

Se desglosa el código y ver qué hace cada función:

1. ``getProductos``: Esta función maneja la solicitud HTTP GET para obtener todos los productos de la base de datos del servidor (asumiendo que ``Producto`` es un modelo que representa productos). Consulta la base de datos usando ``Producto.findAll()`` y envía los productos como una respuesta JSON al cliente.

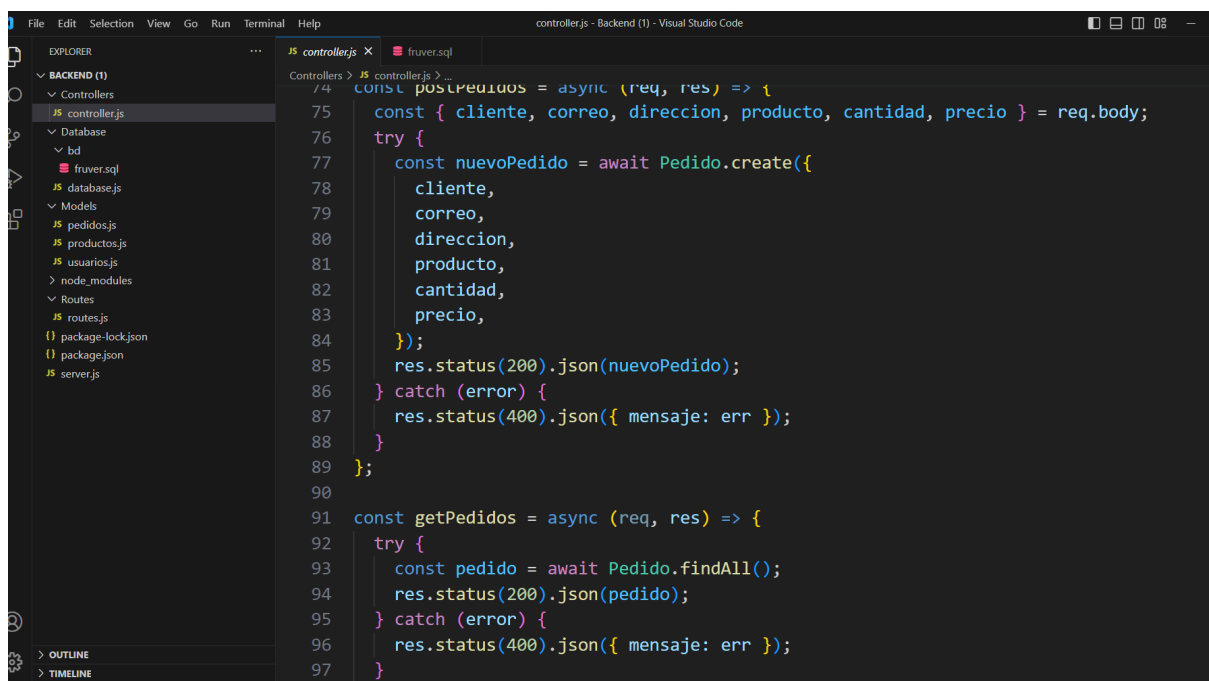
2. ``getProducto``: Esta función maneja la solicitud HTTP GET para obtener un producto específico por su ID. Extrae el ``idProducto`` de los parámetros de la solicitud, consulta la base de datos usando ``Producto.findByPk(idProducto)`` y envía el producto como una respuesta JSON al cliente.
3. ``postProductos``: Esta función maneja la solicitud HTTP POST para crear un nuevo producto. Extrae varias propiedades del producto del cuerpo de la solicitud, crea un nuevo objeto ``Producto`` con los datos proporcionados, lo guarda en la base de datos usando ``Producto.create()`` y envía el producto recién creado como una respuesta JSON al cliente.
4. ``putProductos``: Esta función maneja la solicitud HTTP PUT para actualizar un producto existente. Extrae el ``idProducto`` de los parámetros de la solicitud y varias propiedades del producto del cuerpo de la solicitud. Encuentra el producto en la base de datos usando ``Producto.findByPk(idProducto)``, actualiza sus propiedades, guarda los cambios usando ``save()`` y envía el producto modificado como una respuesta JSON al cliente.
5. ``deleteProductos``: Esta función maneja la solicitud HTTP DELETE para eliminar un producto por su ID. Extrae el ``idProducto`` de los parámetros de la solicitud, encuentra y elimina el producto correspondiente de la base de datos usando ``Producto.destroy()`` y envía un mensaje de éxito como una respuesta JSON al cliente.
6. ``postPedidos``: Esta función maneja la solicitud HTTP POST para crear un nuevo pedido (asumiendo que ``Pedido`` es un modelo que representa pedidos). Extrae varias propiedades del pedido del cuerpo de la solicitud, crea un nuevo objeto ``Pedido`` con los datos proporcionados, lo guarda en la base de datos usando ``Pedido.create()`` y envía el pedido recién creado como una respuesta JSON al cliente.
7. ``getPedidos``: Esta función maneja la solicitud HTTP GET para obtener todos los pedidos de la base de datos del servidor. Consulta la base de datos usando ``Pedido.findAll()`` y envía los pedidos como una respuesta JSON al cliente.
8. ``deletePedido``: Esta función maneja la solicitud HTTP DELETE para eliminar un pedido por su ID. Extrae el ``idPedido`` de los parámetros de la solicitud, encuentra y elimina el pedido correspondiente de la base de datos usando ``Pedido.destroy()`` y envía un mensaje de éxito como una respuesta JSON al cliente.
9. ``getUsuario``: Esta función maneja la solicitud HTTP GET para obtener un usuario específico por su ID (asumiendo que ``Usuario`` es un modelo que representa usuarios). Extrae el ``idUsuario`` de los parámetros de la solicitud, consulta la base de datos usando ``Usuario.findByPk(idUsuario)`` y envía el usuario como una respuesta JSON al cliente.
10. ``postUsuarios``: Esta función maneja la solicitud HTTP POST para crear un nuevo usuario. Extrae varias propiedades del usuario del cuerpo de la solicitud, crea un nuevo objeto ``Usuario`` con los datos proporcionados, lo guarda en la base de datos usando ``Usuario.create()`` y envía el usuario recién creado como una respuesta JSON al cliente.

11. `postUsuarioLogin`: Esta función maneja la solicitud HTTP POST para el inicio de sesión de un usuario. Espera el correo electrónico y la contraseña del usuario en el cuerpo de la solicitud. Consulta la base de datos usando `Usuario.findOne()` para encontrar un usuario con el correo electrónico y la contraseña proporcionados, y si lo encuentra, envía el usuario como una respuesta JSON al cliente, confirmando el inicio de sesión exitoso. Si no se encuentra al usuario o la contraseña es incorrecta, envía un mensaje de error.



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the code for the `controller.js` file. The code defines two asynchronous functions: `getProductos` and `getProducto`.

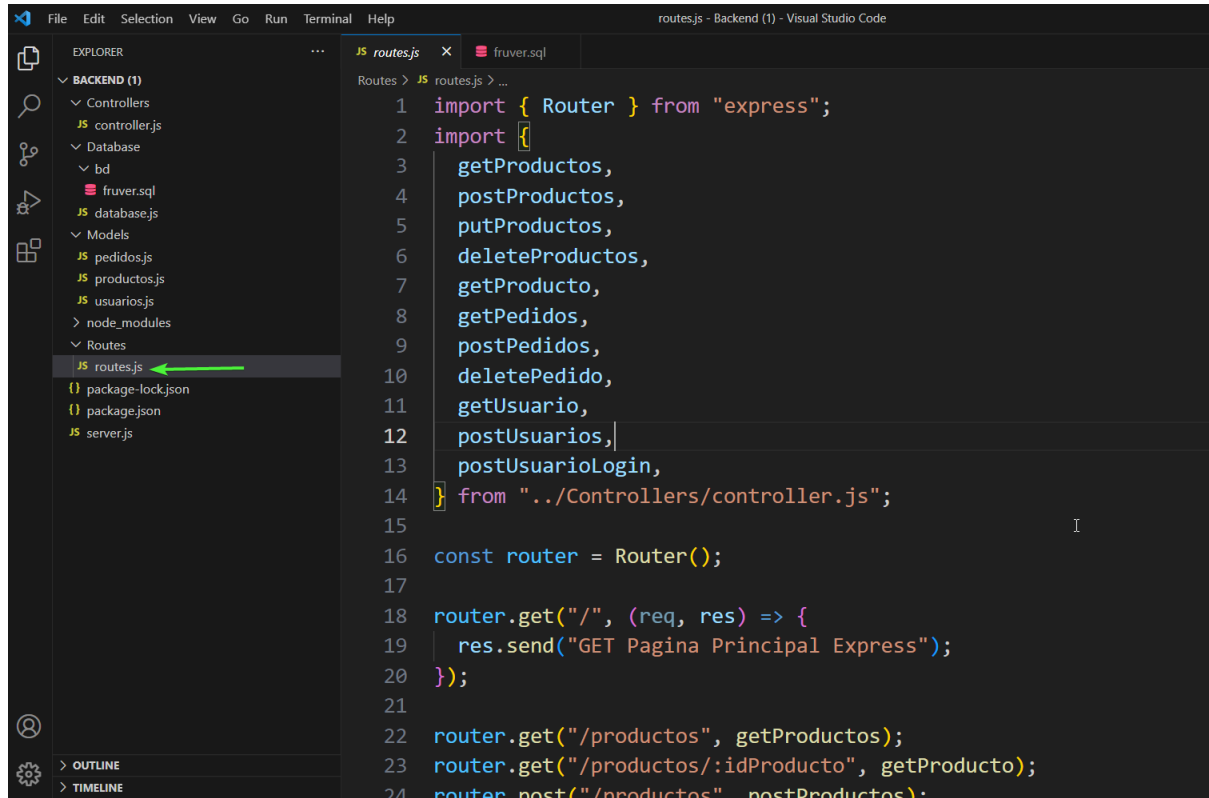
```
1 import { Producto } from "../Models/productos.js";
2 import { Pedido } from "../Models/pedidos.js";
3 import { Usuario } from "../Models/usuarios.js";
4
5 const getProductos = async (req, res) => {
6   try {
7     const productos = await Producto.findAll();
8     res.status(200).json(productos);
9   } catch (error) {
10    res.status(400).json({ mensaje: error });
11  }
12 };
13
14 const getProducto = async (req, res) => {
15   const { idProducto } = req.params;
16   try {
17     const producto = await Producto.findByPk(idProducto);
18     res.status(200).json([producto]);
19   } catch (error) {
20     res.status(400).json({ mensaje: error });
21   }
22 };
23
```



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying the project structure. The main editor window shows the code for the `controller.js` file. The code defines two asynchronous functions: `postPedidos` and `getPedidos`.

```
75 const postPedidos = async (req, res) => {
76   const { cliente, correo, direccion, producto, cantidad, precio } = req.body;
77   try {
78     const nuevoPedido = await Pedido.create({
79       cliente,
80       correo,
81       direccion,
82       producto,
83       cantidad,
84       precio,
85     });
86     res.status(200).json(nuevoPedido);
87   } catch (error) {
88     res.status(400).json({ mensaje: err });
89   }
90 };
91
92 const getPedidos = async (req, res) => {
93   try {
94     const pedido = await Pedido.findAll();
95     res.status(200).json(pedido);
96   } catch (error) {
97     res.status(400).json({ mensaje: err });
98   }
99 }
```

Se tiene el archivo de rutas para crear un enrutador que define varias rutas para interactuar con productos, usuarios y pedidos en una aplicación web. Cada ruta está vinculada a una función controladora específica que realiza las operaciones correspondientes en la base de datos o en el servidor.



```
1 import { Router } from "express";
2 import {
3   getProductos,
4   postProductos,
5   putProductos,
6   deleteProductos,
7   getProducto,
8   getPedidos,
9   postPedidos,
10  deletePedido,
11  getUserio,
12  postUsuarios,
13  postUsuarioLogin,
14 } from "../Controllers/controller.js";
15
16 const router = Router();
17
18 router.get("/", (req, res) => {
19   res.send("GET Pagina Principal Express");
20 });
21
22 router.get("/productos", getProductos);
23 router.get("/productos/:idProducto", getProducto);
24 router.post("/productos", postProductos);
```

Ahora se crea un archivo para el servidor este código configura un servidor web utilizando Express y Sequelize en Node.js. El servidor escucha en el puerto 3000 y tiene rutas definidas en el enrutador para manejar las solicitudes HTTP relacionadas con productos, usuarios y pedidos. Además, se realiza una conexión a la base de datos utilizando Sequelize antes de iniciar el servidor.

En este código se encuentra

Este código configura un servidor web utilizando el framework Express en Node.js. A continuación, te explico lo que hace cada parte del código:

1. Importaciones:

- Se importa el módulo `express` para crear la aplicación del servidor.
- Se importa el enrutador (`router`) desde el archivo `routes.js`. Este enrutador contiene todas las rutas y sus respectivas funciones controladoras para manejar las solicitudes HTTP.
- Se importa `sequelize`, que parece ser una instancia de la biblioteca Sequelize para interactuar con la base de datos.

2. Configuración del servidor:

- Se crea una instancia de la aplicación del servidor utilizando `express()`, que se almacena en la constante `app`.

3. Middlewares:

- Se utilizan tres middlewares para el procesamiento de las solicitudes:
 - `bodyParser.json()`: Permite al servidor analizar el cuerpo de las solicitudes HTTP con formato JSON.
 - `cors()`: Habilita el Cross-Origin Resource Sharing (CORS), lo que permite que el servidor reciba solicitudes desde diferentes dominios o puertos.
 - `express.json()`: Analiza el cuerpo de las solicitudes con formato JSON (esta línea es innecesaria, ya que `bodyParser.json()` también realiza esta función).

4. Configuración del enrutador:

- Se agrega el enrutador (`router`) a la aplicación del servidor utilizando `app.use(router)`. Esto permite que todas las rutas definidas en el enrutador estén disponibles para la aplicación.

5. Configuración del puerto:

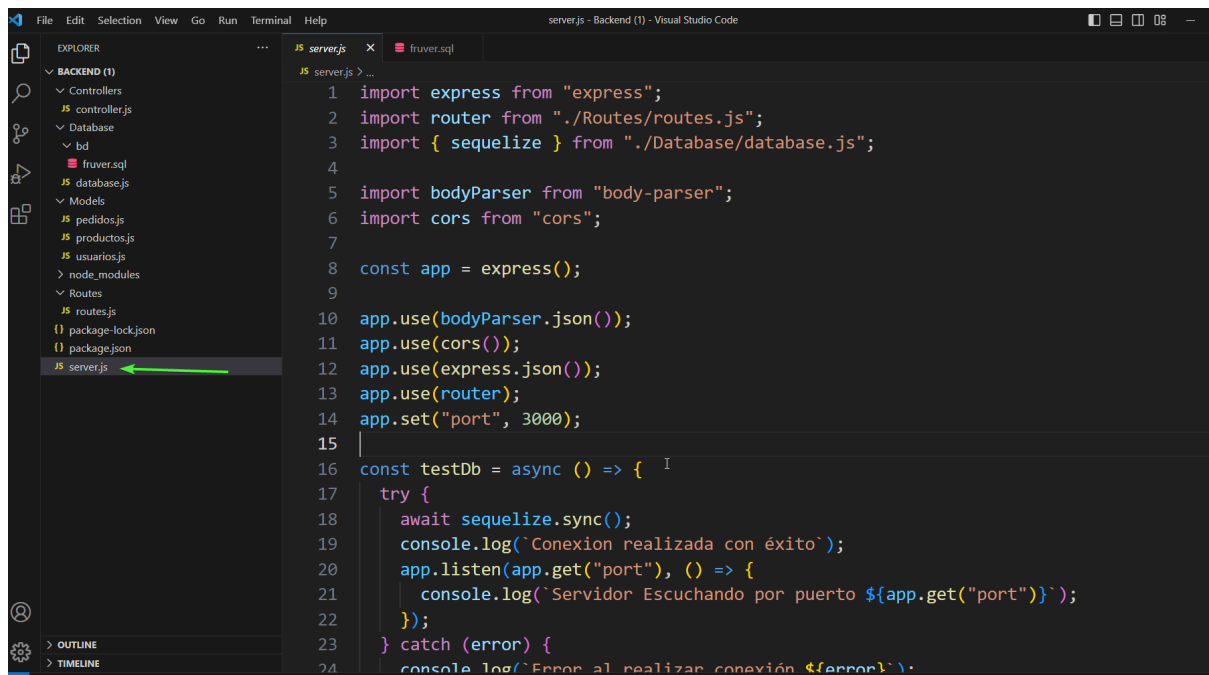
- Se establece el puerto en el que el servidor escuchará las solicitudes utilizando `app.set("port", 3000)`. En este caso, el servidor está configurado para escuchar en el puerto 3000.

6. Conexión a la base de datos y inicio del servidor:

- Se define una función llamada `testDb` que se encarga de realizar la conexión a la base de datos utilizando `sequelize.sync()`. Esta función utiliza el método `sync()` para sincronizar los modelos definidos en la base de datos y asegurarse de que estén disponibles para su uso.
- Si la conexión a la base de datos es exitosa, se imprime "Conexion realizada con éxito" en la consola.
- Después de eso, el servidor se inicia llamando al método `app.listen()`, que hace que el servidor comience a escuchar las solicitudes en el puerto especificado (`app.get("port")`).
- Si hay algún error al conectar a la base de datos, se imprime "Error al realizar conexión" seguido del mensaje de error en la consola.

7. Llamado a la función `testDb()`:

- Finalmente, se llama a la función `testDb()` para establecer la conexión con la base de datos y poner en marcha el servidor.



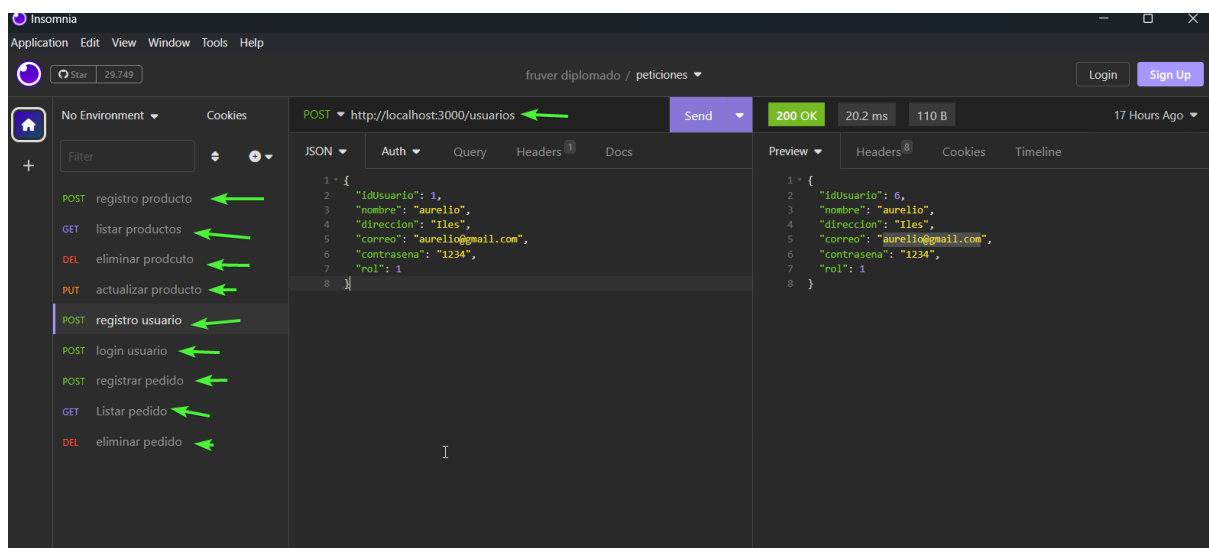
```
1 import express from "express";
2 import router from "../Routes/routes.js";
3 import { sequelize } from "../Database/database.js";
4
5 import bodyParser from "body-parser";
6 import cors from "cors";
7
8 const app = express();
9
10 app.use(bodyParser.json());
11 app.use(cors());
12 app.use(express.json());
13 app.use(router);
14 app.set("port", 3000);
15
16 const testDb = async () => {
17   try {
18     await sequelize.sync();
19     console.log(`Conexion realizada con éxito`);
20     app.listen(app.get("port"), () => {
21       console.log(`Servidor Escuchando por puerto ${app.get("port")}`);
22     });
23   } catch (error) {
24     console.log(`Error al realizar conexión ${error}`);
25   }
26 }
```

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

Para este caso se utiliza la herramienta de Insomnia para llevar a cabo las operaciones

Se abra la herramienta y se crean las respectivas operaciones para las distintas tablas creadas en la base de datos anteriormente se añade la ruta cada operación por la cual va escuchar nuestro servidor en este caso

<http://localhost:3000/ruta>



Ahora se muestra cada una de las operaciones:

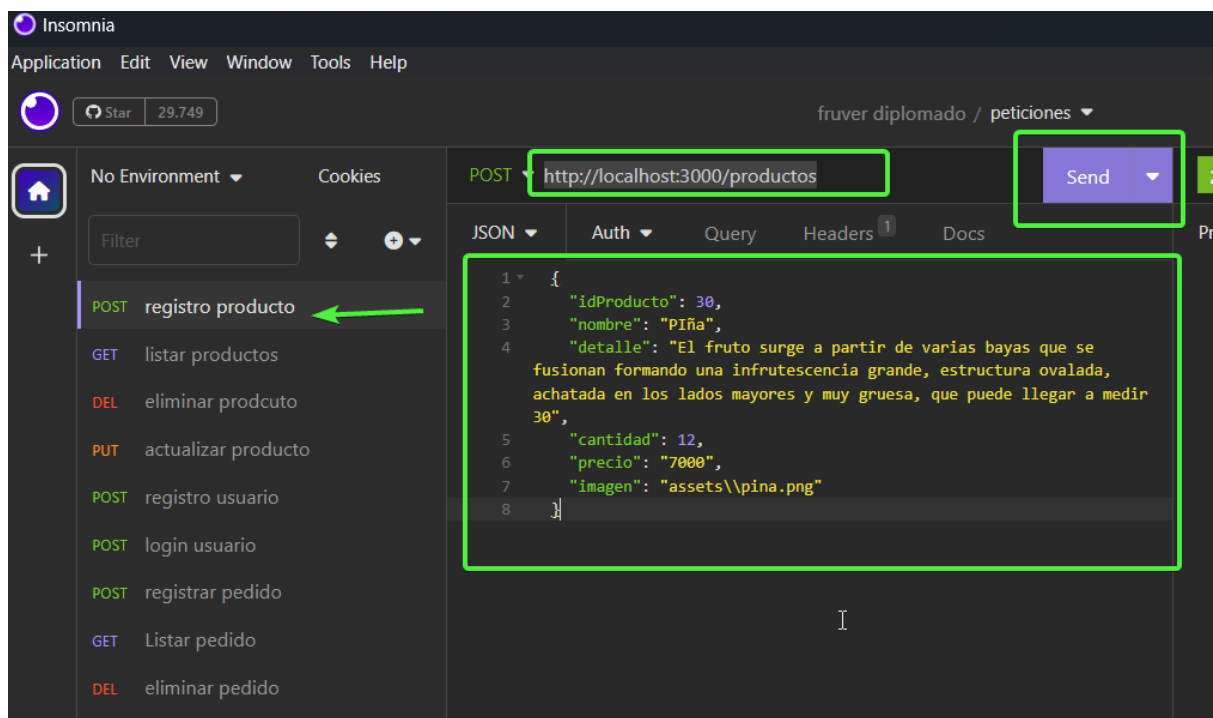
Registrar productos

<http://localhost:3000/productos>

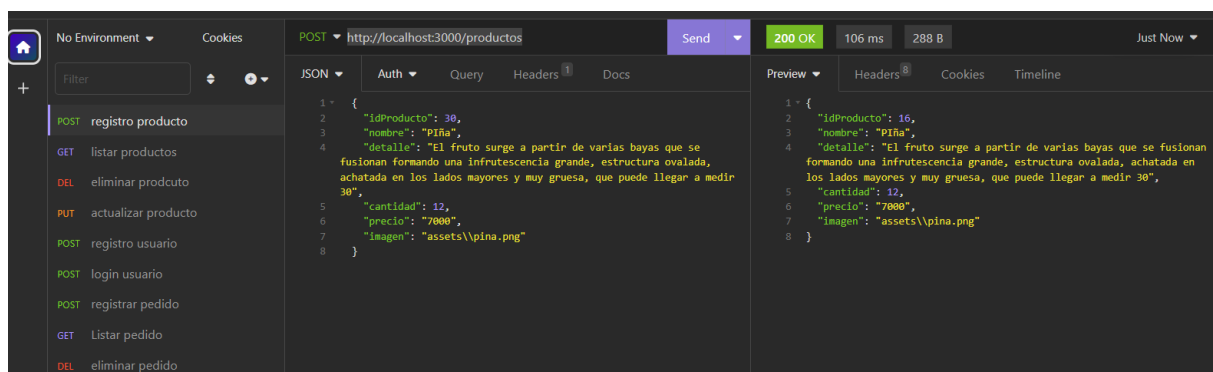
Se muestra la base de datos actual para realizar la evidencia, en este caso vamos a registrar un nuevo producto como ejemplo **Piña**

| | | idProducto | nombre | detalle | cantidad | precio | imagen |
|--------------------------|--------|------------|----------|-------------------------------------------------------|----------|--------|---------------------|
| <input type="checkbox"/> | Editar | 9 | Papa1 | Solanum tuberosum, de nombre común papa (América, ... | 10 | 45000 | assets/papas.png |
| <input type="checkbox"/> | Editar | 10 | Aji | Nombre común de diversas plantas herbáceas america... | 50 | 2000 | assets/aji.png |
| <input type="checkbox"/> | Editar | 11 | Cilantro | Es una de las hierbas protagonistas de la cocina e... | 12 | 1500 | assets/cilantro.png |
| <input type="checkbox"/> | Editar | 12 | Manzana | Fruto del manzano, comestible, de forma redondeada... | 12 | 5000 | assets/manzanas.png |
| <input type="checkbox"/> | Editar | 13 | Pera | Fruto del peral, comestible, de color verde, amari... | 25 | 5000 | assets/peras.png |

Entramos a insomnia y reguistramos un producto con los atributos creadois ne la base de datos



se envía la instrucción y se evidencia el registro en insomnia y en la base de datos Mysql



Se registra exitosamente

| | | idProducto | nombre | detalle | cantidad | precio | imagen |
|--------------------------|----------------------|------------|----------|-------------------------------------------------------|----------|--------|---------------------|
| <input type="checkbox"/> | Editar Copiar Borrar | 9 | Papa1 | Solanum tuberosum, de nombre común papa (América, ... | 10 | 45000 | assets\papas.png |
| <input type="checkbox"/> | Editar Copiar Borrar | 10 | Ají | Nombre común de diversas plantas herbáceas america... | 50 | 2000 | assets\ajaj.png |
| <input type="checkbox"/> | Editar Copiar Borrar | 11 | Cilantro | Es una de las hierbas protagonistas de la cocina e... | 12 | 1500 | assets\cilantro.png |
| <input type="checkbox"/> | Editar Copiar Borrar | 12 | Manzana | Fruto del manzano, comestible, de forma redondeada... | 12 | 5000 | assets\manzanas.png |
| <input type="checkbox"/> | Editar Copiar Borrar | 13 | Pera | Fruto del peral, comestible, de color verde, amari... | 25 | 5000 | assets\peras.png |
| <input type="checkbox"/> | Editar Copiar Borrar | 16 | Piña | El fruto surge a partir de varias bayas que se fus... | 12 | 7000 | assets\pina.png |

ahora la operación get para listar productos y se observa que nos lista los productos que miranos en la base de datos.

Insomnia REST client interface showing a successful GET request to `http://localhost:3000/productos`. The response is a 200 OK status with a JSON array of product data. Green arrows point to the URL and the response body.

```
GET http://localhost:3000/productos 200 OK 11.4 ms 1730 B 18 Hours Ago
```

Body:

```
[{"idProducto": 12, "nombre": "Cilantro", "detalle": "Es una de las hierbas protagonistas de la cocina e...", "cantidad": 12, "precio": 1500, "imagen": "assets\\cilantro.png"}, {"idProducto": 12, "nombre": "Manzana", "detalle": "Fruto del manzano, comestible, de forma redondeada y algo hundida por los extremos, piel fina, de color verde, amarillo o rojo, carne blanca y jugosa, de sabor dulce o ácido, y semillas en forma de pepitas encerradas en una cápsula de cinco divisiones.", "cantidad": 12, "precio": 5000, "imagen": "assets\\manzanas.png"}, {"idProducto": 13, "nombre": "Pera", "detalle": "Fruto del peral, comestible, de color verde, amarillo o encarnado, ancho por la parte de abajo y delgado por la de arriba (donde tiene el pedúnculo), de piel fina y pulpa blanca, muy jugosa y sabor dulce.", "cantidad": 25, "precio": 5000, "imagen": "assets\\peras.png"}]
```

Ahora la operación eliminar producto en donde se envía el id de 16 que corresponde a la piña que se había creado anteriormente se obtiene un registro eliminado exitosamente
















Insomnia REST client interface showing a successful DELETE request to `http://localhost:3000/productos/16`. The response is a 200 OK status with a JSON object containing a message. Green arrows point to the URL and the response body.

```
DELETE http://localhost:3000/productos/16 200 OK 28.4 ms 32 B 18 Hours Ago
```
















Body:

```
{ "mensaje": "Registro Eliminado" }
```

Se revisa en base de datos y de igual manera se elimina el registro del producto Piña

| | | | | | idProducto | nombre | detalle | cantidad | precio | imagen |
|--------------------------|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|----|------------|-------------------------------------------------------|---------|----------|---------------------|--------|
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 9 | Papa1 | Solanum tuberosum, de nombre común papa (América, ... | 10 | 45000 | assets\papas.png | |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 10 | Ají | Nombre común de diversas plantas herbáceas america... | 50 | 2000 | assets\aji.png | |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 11 | Cilantro | Es una de las hierbas protagonistas de la cocina e... | 12 | 1500 | assets\cilantro.png | |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 12 | Manzana | Fruto del manzano, comestible, de forma redondeada... | 12 | 5000 | assets\manzanas.png | |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 13 | Pera | Fruto del peral, comestible, de color verde, amari... | 25 | 5000 | assets\peras.png | |

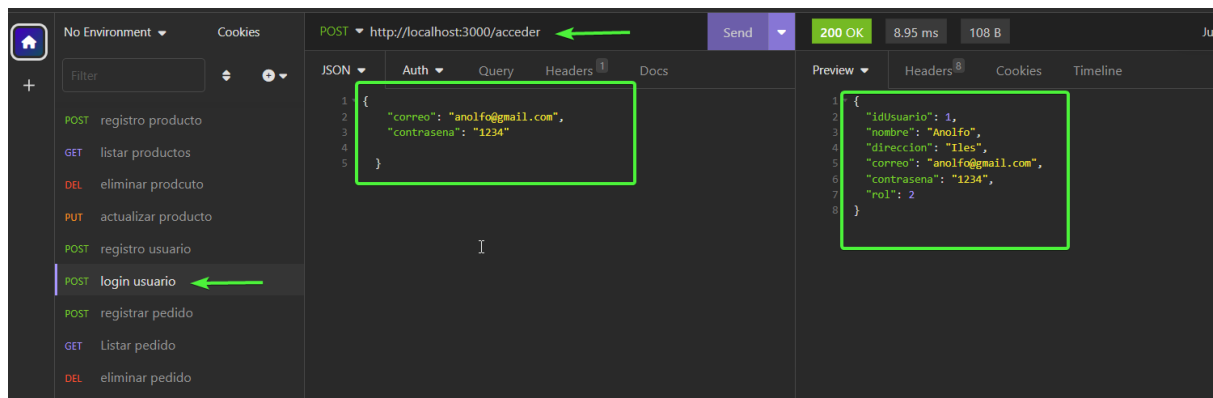
Ahora la operación de Put de actualizar producto, se actualizar el id 12 que corresponde a manzana le actualizamos el nombre a manzanaroja y se observa el cambio

| | | idProducto | nombre | detalle | cantidad | precio | imagen | | |
|--------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|---------|-------------|-------------------------------------------------------|--------|-------|---------------------|
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 9 | Papa1 | Solanum tuberosum, de nombre común papa (América, ... | 10 | 45000 | assets/papas.png |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 10 | Ají | Nombre común de diversas plantas herbáceas america... | 50 | 2000 | assets/ají.png |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 11 | Cilantro | Es una de las hierbas protagonistas de la cocina e... | 12 | 1500 | assets/cilantro.png |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 12 | Manzanaroja | El fruto surge a partir de varias bayas que se fus... | 12 | 7000 | assets/manzana.png |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | 13 | Pera | Fruto del peral, comestible, de color verde, amari... | 25 | 5000 | assets/peras.png |

ahora la operación de registrar usuario , se crea de manera exitosa

| | | | | | idUsuario | nombre | direccion | correo | contrasena | rol |
|--------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|--|-----------|---------------|-----------------|---------------------------|------------|-----|
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 1 | Anolfo | Iles | anolfo@gmail.com | 1234 | 2 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 2 | david | Pasto | josecamilreyes9@gmail.com | 12345 | 2 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 3 | administrador | Bogota | administrador@gmail.com | admin123 | 1 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 4 | carmen | Iles | carmen@gmail.com | 1234 | 2 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 5 | Milton | Pupiales Nariño | milton@gmail.com | 12345 | 2 |
| <input type="checkbox"/> |  Editar |  Copiar |  Borrar | | 7 | Pancrasio | Pasto | pancrasio@gmail.com | 1234 | 1 |

Ahora la operación de login de usuario se la simula de tal manera que se obtiene ingreso



Ahora se simula que la contraseña es invalida al momento de loguearse , colocamos la contraseña incorrecta y saldrá usuario no registrado

