

**DIPLOMADO DE ACTUALIZACIÓN EN NUEVAS TECNOLOGÍAS PARA EL DESARROLLO  
DE SOFTWARE**

**TALLER UNIDAD 2 BACKEND “FRUVERMANIA”**

**<https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git>**

**PAOLA ANDREA MONTANCHEZ MONTANCHEZ**

**CÓDIGO: 218034232**

**PRESENTADO A: ING. VICENTE AUX REVELO**

**UNIVERSIDAD DE NARIÑO**

**INGENIERIA DE SISTEMAS**

**2023**

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.

Para la implementación de la base de datos se requerirá las siguientes herramientas:

Para la creación de la base de datos se utilizará **MYSQL**, un sistema de gestión de bases de datos relacional de código abierto. Este es uno de los sistemas de bases de datos más populares y ampliamente utilizados en el mundo.

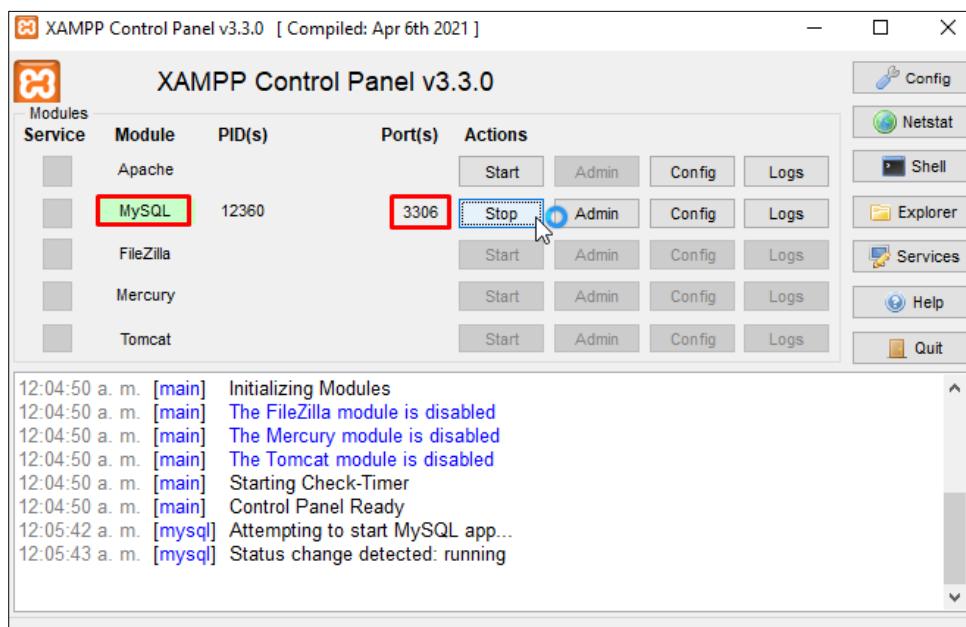
De igual manera se utilizará **XAMPP**, Un paquete de software que proporciona un entorno de desarrollo completo para crear y probar aplicaciones web localmente en la computadora.

Como gestor de base de datos, se hará uso de **DBeaver**, una herramienta gráfica de administración que proporciona una interfaz gráfica que permite editar y visualizar los datos almacenados en las tablas directamente desde la interfaz gráfica. Esto es especialmente útil para realizar cambios rápidos o hacer verificaciones rápidas en los datos.

## 1.1 Creación de la base de datos fruverpm

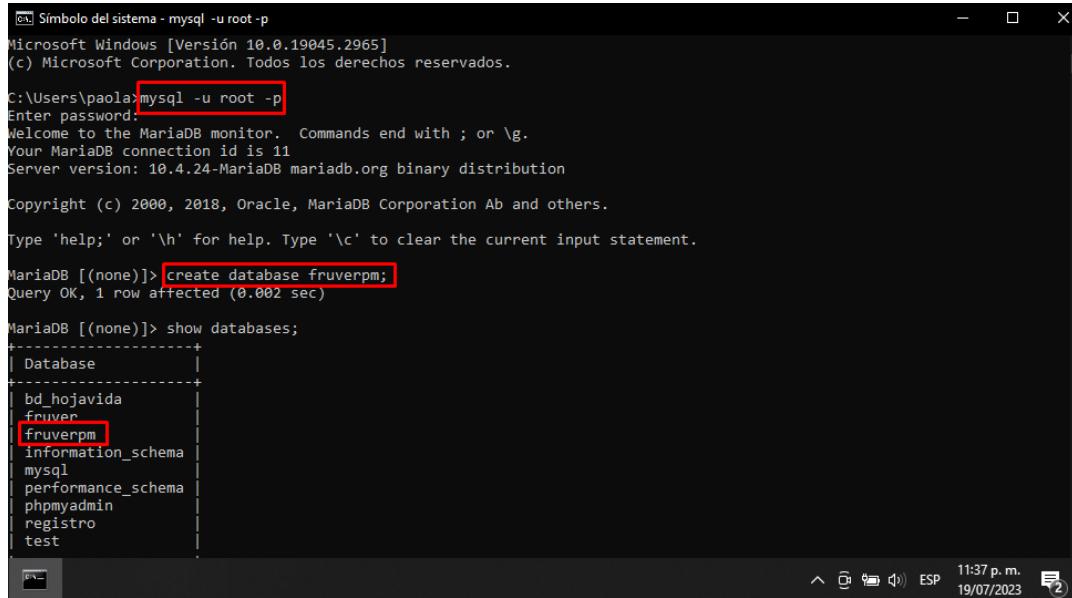
Primero, es necesario asegurarse de tener instalado MySQL en el sistema operativo. Los instaladores adecuados se pueden descargar desde los sitios web oficiales y seguir las instrucciones de instalación correspondientes.

- Inicializamos **Xampp**, Damos clic en el botón Start junto a MySQL para iniciar los servicios correspondientes.



- Creación de la base de datos **fruverpm** desde la consola.

Ingresamos a la consola, y ejecutamos el comando **mysql -u root -p** para acceder al servidor de base de datos MySQL. Con el comando **create database fruverpm** creamos nuestra base de datos.



```

Símbolo del sistema - mysql -u root -p
Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\paola>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 11
Server version: 10.4.24-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

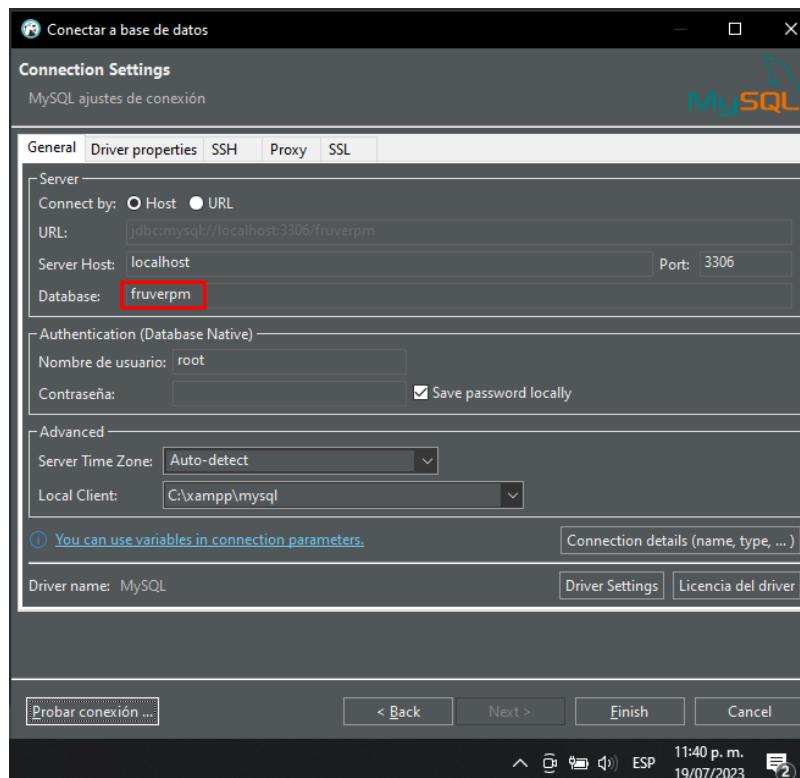
MariaDB [(none)]> Create database fruverpm;
Query OK, 1 row affected (0.002 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| bd_hojavida |
| fruver |
| fruverpm |
| information_schema |
| mysql |
| performance_schema |
| phpmyadmin |
| registro |
| test |
+-----+

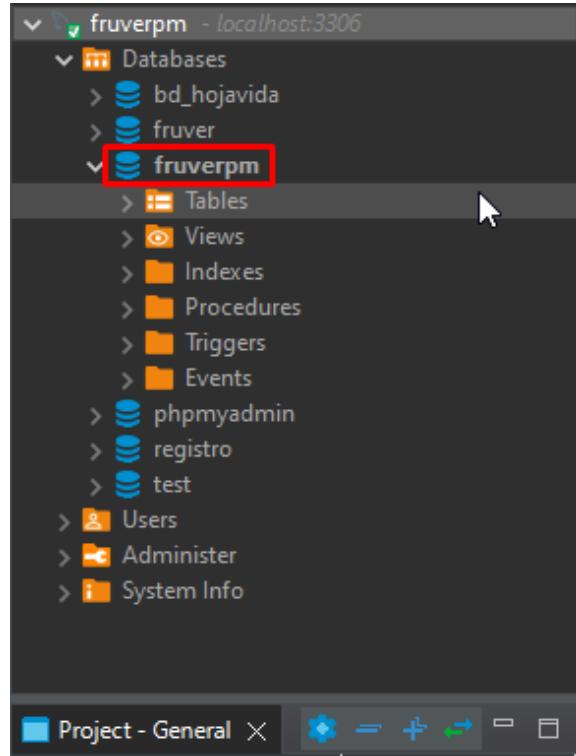
```

## 1.2 Conexión DBeaver

Configuramos una nueva conexión en DBeaver para conectar al servidor MySQL. Utilizamos las credenciales (Server Host: localhost, Database: fruverpm, usuario: root, contraseña: " "). para acceder a la base de datos fruverpm creada anteriormente.



➤ Conexión establecida



### 1.3 Creación de las tablas:

Se procede a crear 3 tablas:

productos: para almacenar los datos de las frutas y verduras.

clientes: para almacenar los datos de los clientes.

compras: para llevar un registro de las solicitudes de compra.

#### Definición de la estructura de las tablas:

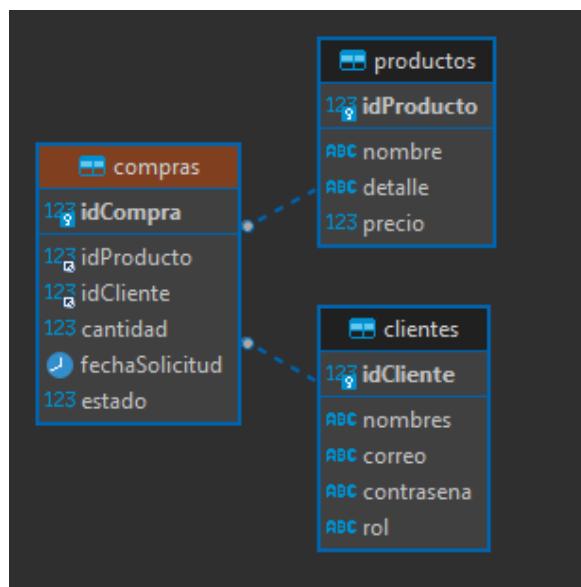
**En la tabla productos:** se definen columnas como idProducto, nombre, detalle y precio.

**En la tabla clientes:** se definen columnas como idClientes, nombres, correo, contraseña y rol, este puede ser cliente o admin.

**En la tabla compras:** se definen columnas idCompra, idProducto, idCliente, cantidad, fechaSolicitud y estado de la compra.

Se establecen las relaciones y las restricciones adecuadas, como llaves primarias.

## ➤ Diagrama entidad relación



- Desarrollar una aplicación Backend implementada en NodeJS y ExpressJS que haga uso de la base de datos del primer punto 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.

Para la realización de la aplicación se hará uso del editor **Visual Studio Code**, el cual es un editor de código fuente desarrollado por Microsoft que se ha vuelto muy popular entre los desarrolladores debido a su flexibilidad, funcionalidades y extensibilidad.

### 2.1 Instalación de los paquetes necesarios para realizar la aplicación.

**Package json:** es un archivo de configuración fundamental que se utiliza para definir y gestionar las dependencias de un proyecto.

```
File Edit Selection View Go Run Terminal Help package.json - BackendFruverPM - Visual Studio Code
```

EXPLORER

- BACKEND...
- package.json

package.json

```
1 {
  2   "name": "backendfruverpm",
  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 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> **npm init -y**  
Wrote to C:\Users\paola\OneDrive\Escritorio\BackendFruverPM\package.json:

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

L1, Col 1 Spaces: 2 UTF-8 LF JSON Go Live Prettier

Windows Taskbar: Live Share

Bottom status bar: 12:06 a.m. ESP 20/07/2023

**Express:** es un framework de aplicaciones web para Node.js que facilita la creación de aplicaciones y APIs.

The screenshot shows the Visual Studio Code interface with the following details:

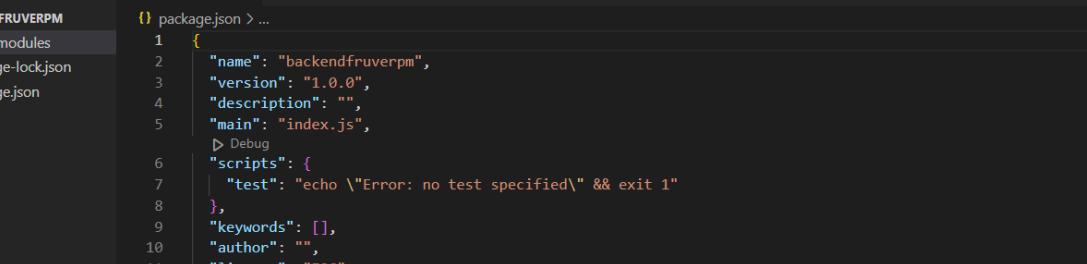
- File Explorer:** Shows a project structure with a red box around the "node\_modules" folder.
- Editor:** Displays the contents of the package.json file.
- Terminal:** Shows the command "npm install express" being run, with a red box around it.
- Status Bar:** Shows the output "added 58 packages, and audited 59 packages in 13s".

```
1 {
  2   "name": "backendfruverpm",
  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     "express": "^4.18.2"
 14   }
 15 }
```

```
PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> npm install express
```

added 58 packages, and audited 59 packages in 13s

**Mysq2:** Es un paquete de Node.js que proporciona una interfaz para interactuar con bases de datos MySQL.



```
EXPLORER    ...  package.json x
BACKENDFRUVERPM
> node_modules
  package-lock.json
  package.json

1  {
2   "name": "backendfruverpm",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   ▷ Debug
7   "scripts": {
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "express": "^4.18.2",
15    "mysql2": "^3.5.2"  }  }

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  GITLENS
```

added 58 packages, and audited 59 packages in 13s

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

found 0 vulnerabilities

```
PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> npm i mysql2
o [=====] \ reify:mysql2: http fetch GET 200 https://registry.npmjs.org/mysql2/-/mysql2-3.5.2.tgz 1081ms
```

Ln 1, Col 1 Spaces: 2 UTF-8 LF { JSON ⚡ Go Live ✅ Prettier 🔍

**Sequelize:** es un ORM para Nodejs que facilita la interacción con bases de datos relacionales desde aplicaciones Node.js.

```
{} package.json x

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
8 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> npm install sequelize
added 22 packages, and audited 92 packages in 20s
```

**Jsonwebtoken (JWT)**: es una librería utilizada para generar tokens de autenticación en la función login.

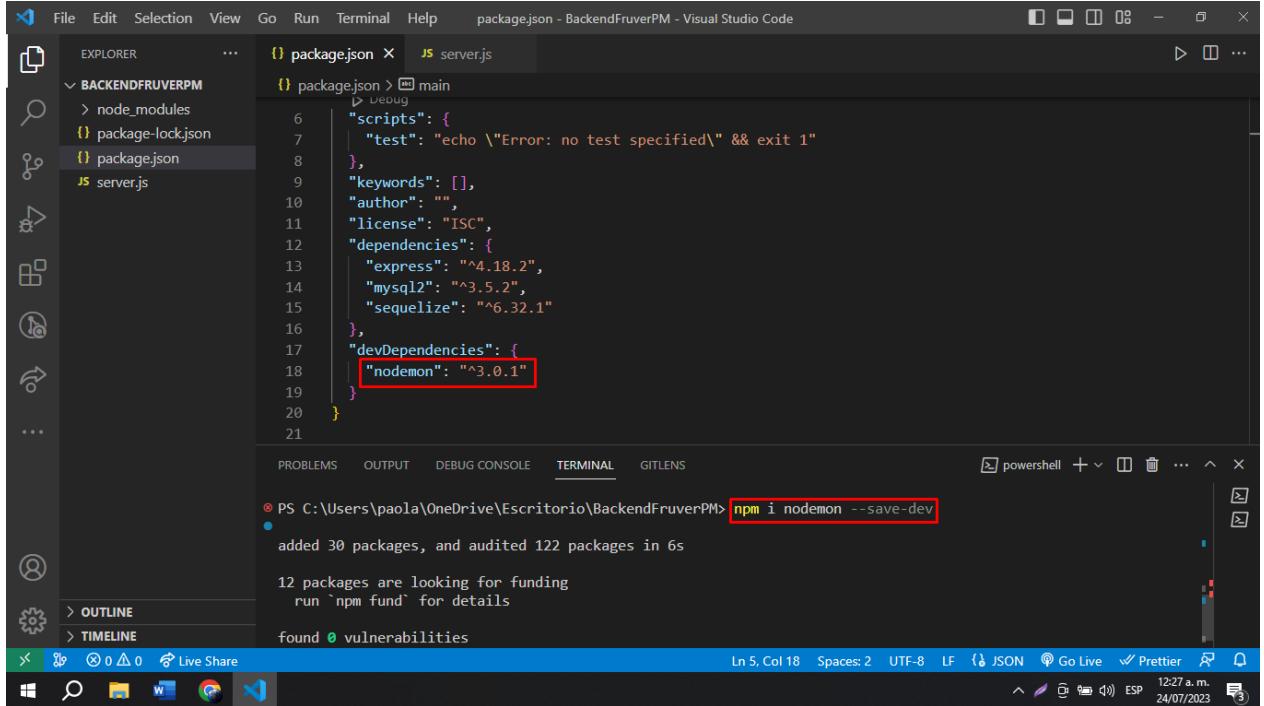
```
{} package.json x
{} package.json > type
1 {
2   "name": "backendfruverpm",
3   "version": "1.0.0",
4   "description": "",
5   "main": "server.js",
6   "type": "module",
7   "scripts": {
8     "start": "nodemon server.js"
9   },
10  "keywords": [],
11  "author": "",
12  "license": "ISC",
13  "dependencies": {
14    "express": "^4.18.2",
15    "jsonwebtoken": "^9.0.1", highlighted
16    "mysql": "^2.18.1",
17    "mysql2": "^3.5.2",
18    "sequelize": "^6.32.1"
19  },
20  "devDependencies": {}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> npm install jsonwebtoken

added 6 packages, and audited 140 packages in 8s

**Nodemon:** es un "monitor de cambios" que permite reiniciar automáticamente la aplicación Node.js cada vez que se detectan modificaciones en los archivos del proyecto. Instalación de dependencia a nivel de desarrollo.

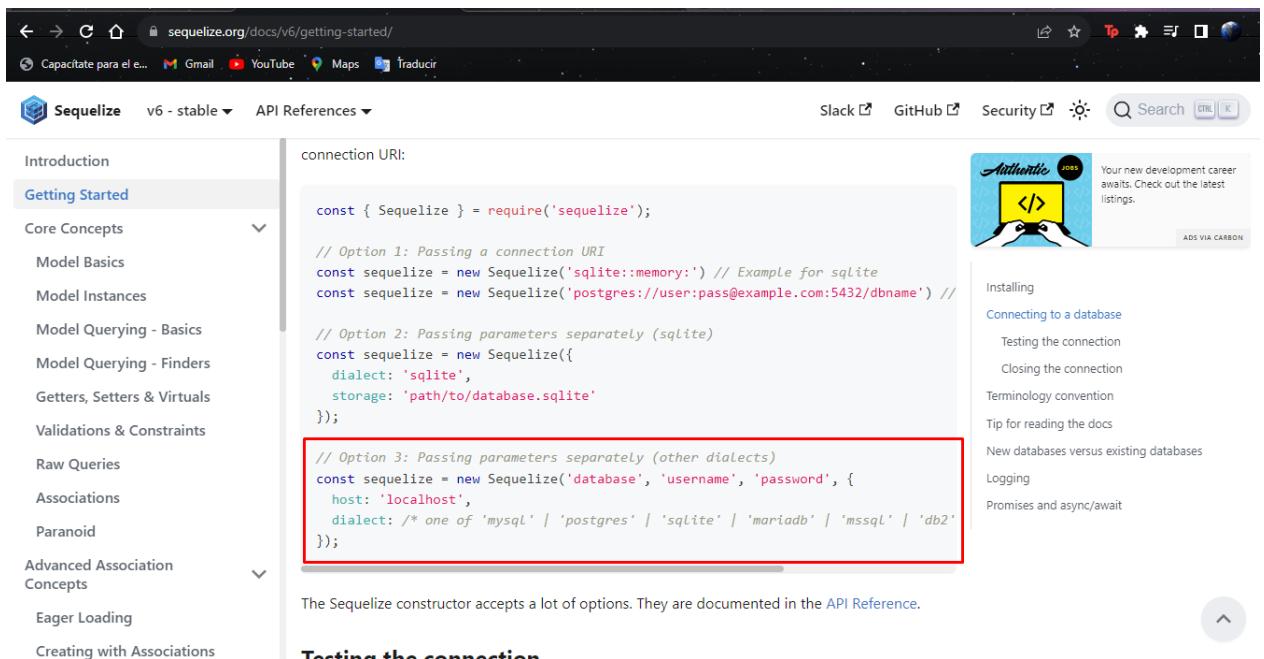


The screenshot shows the Visual Studio Code interface with the following details:

- EXPLORER:** Shows the project structure: BACKENDFRUVERPM, node\_modules, package-lock.json, package.json, and server.js.
- PACKAGE.JSON:** Opened in the editor, showing the contents of the main package.json file. The "devDependencies" section is highlighted with a red box, containing the entry: "nodemon": "^3.0.1".
- TERMINAL:** Shows the command: `PS C:\Users\paola\OneDrive\Escritorio\BackendFruverPM> npm i nodemon --save-dev`. Below it, the output of the command is shown: "added 30 packages, and audited 122 packages in 6s".
- STATUS BAR:** Shows the current file is package.json - BackendFruverPM - Visual Studio Code, and the date and time: 24/07/2023 12:27 a.m.

## 2.2 configuración database.js

Para realizar la conexión a la base de datos, primero se hace uso de la revisión de la documentación de sequelize. (link: <https://sequelize.org/docs/v6/getting-started/>).



The screenshot shows the Sequelize documentation website at [sequelize.org/docs/v6/getting-started/](https://sequelize.org/docs/v6/getting-started/).

- Left Sidebar:** Navigation menu with sections like Introduction, Getting Started (highlighted), Core Concepts, Model Basics, Model Instances, Model Querying - Basics, Model Querying - Finders, Getters, Setters & Virtuals, Validations & Constraints, Raw Queries, Associations, Paranoid, Advanced Association Concepts, Eager Loading, and Creating with Associations.
- Content Area:**
  - connection URI:**
  - Code snippets for different connection options:
    - // Option 1: Passing a connection URI
    - // Option 2: Passing parameters separately (sqlite)
    - // Option 3: Passing parameters separately (other dialects)
- Right Sidebar:**
  - Authentic** advertisement: Your new development career awaits. Check out the latest listings.
  - Links: Installing, Connecting to a database, Testing the connection, Closing the connection, Terminology convention, Tip for reading the docs, New databases versus existing databases, Logging, and Promises and async/await.

Para realizar esta conexión se crea una carpeta con el nombre Database y se crea un archivo llamado **database.js**, en el cual se crea una instancia de Sequelize y se le pasa los parámetros. En este caso, los parámetros de conexión se pasan por separado al constructor de Sequelize.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure with files like routes.js, server.js, and database.js.
- Database.js Content:** The code defines a Sequelize instance for a MySQL database named "fruverpm".

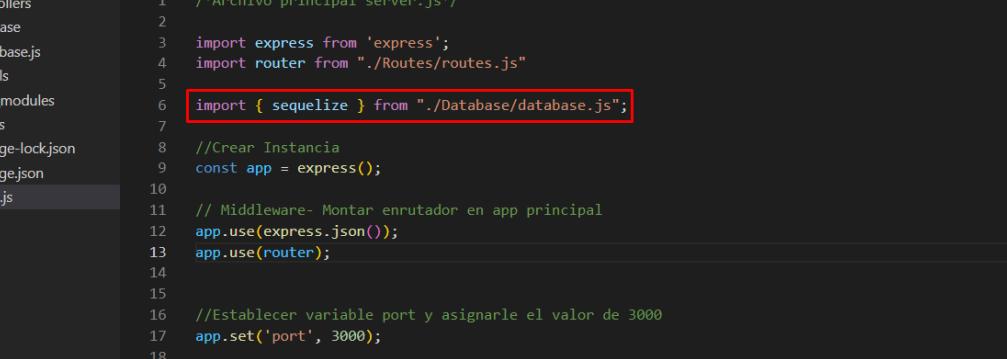
```
//Conexion con la base de datos
import Sequelize from "sequelize";
//Parametros de la base de datos por separado
const sequelize = new Sequelize("fruverpm", "root", "", {
  host: "localhost",
  dialect: "mysql",
});
export {
  sequelize
}
```
- Terminal:** Shows the output of the application's connection status.

```
Conexion realizada con éxito
Servidor escuchando por puerto 3000
```
- Status Bar:** Displays file information (Ln 5, Col 34), code analysis results (346 a.m., 28/07/2023), and developer tools (Live Share).

## 2.3 Configuración archivo server.js

Se crea el archivo `server.js`, a través del cual se iniciará la aplicación node. Se lo configura de la siguiente manera:

Se importa el módulo express, se importa el objeto sequelize creado en el archivo database.js, se define un puerto por el cual se escuchará las solicitudes, en este caso puerto 3000.



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows a tree structure of files and folders. The 'BACKENDFRUVERPM' folder is expanded, showing 'Controllers', 'Database' (which contains 'database.js'), 'Models', 'node\_modules', 'Routes', 'package-lock.json', 'package.json', and 'server.js'. 'server.js' is the active file.
- Editor (Center):** Displays the content of 'server.js'. A red box highlights the line: `import { Sequelize } from './Database/database.js';`
- Bottom Status Bar:** Shows the status bar with the message "Executing (default): SELECT 1+1 AS result", "Conexion realizada con éxito", and "Servidor escuchando por puerto 3000". It also displays file navigation information: Ln 13, Col 17, Spaces: 4, UTF-8, CRLF, and icons for JavaScript, Go Live, and Prettier.

También, se realiza una prueba a la base de datos, para lo cual se define la función testDb (primero realiza la prueba de conexión y si está conectado lanza el servicio).

```

File Edit Selection View Go Run Terminal Help serverjs - BackendFruverPM - Visual Studio Code
EXPLORER JS routes.js JS server.js X JS database.js
BACKENDFRUVERPM
    > Controllers
    > Database
    JS database.js
    > Models
    > node_modules
    > Routes
    package-lock.json
    package.json
    JS server.js

17 app.set('port', 3000);
18
19 //Test a Base de datos
20 const testDb = async () => {
21     try {
22         await sequelize.authenticate();
23         console.log(`Conexion realizada con éxito`);
24         //Correr el servicio por puerto 3000
25         app.listen(app.get('port'), ()=>{
26             console.log(`Servidor escuchando por puerto ${app.get('port')}`);
27         });
28     } catch (error) {
29         console.log(`Error al realizar conexión ${error}`);
30     }
31 };
32
33 testDb();

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

Executing (default): SELECT 1+1 AS result  
Conexion realizada con éxito  
Servidor escuchando por puerto 3000

Ln 13, Col 17 Spaces: 4 UTF-8 CRLF ⓘ JavaScript ⓘ Go Live ⓘ Prettier ⓘ

## 2.4 Modelación de la base de datos para que sea reconocida por Sequelize

Se definen los modelos que permiten tener la estructura de la base de datos en el código. Para cada tabla de la base de datos se define un modelo. En cada uno de los modelos se implementa sequelize y datatypes (tipos de datos).

### Modelo de productos

En la documentación de sequelize se pueden visualizar, los tipos de modelos, en nuestro caso utilizamos el modelo básico.

sequelize.org/docs/v6/core-concepts/model-basics/

Using `sequelize.define`:

```

const { Sequelize, DataTypes } = require('sequelize');
const sequelize = new Sequelize('sqlite::memory');

const User = sequelize.define('User', {
  // Model attributes are defined here
  firstName: {
    type: DataTypes.STRING,
    allowNull: false
  },
  lastName: {
    type: DataTypes.STRING
    // allowNull defaults to true
  },
  {
    // Other model options go here
  }

  // `sequelize.define` also returns the model
  console.log(User === sequelize.models.User); // true
}

```

ADS VIA CARBON

Concept  
Model Definition  
Using `sequelize.define`  
Extending Model  
Table name inference  
Enforcing the table name to be equal to the model name  
Providing the table name directly  
Model synchronization  
Synchronizing all models at once  
Dropping tables  
Database safety check  
Synchronization in production  
Timestamps

Se crea un archivo con el nombre **productos.js** donde se implementa Sequelize y datatypes (tipos de datos) con el modelo básico y definimos los atributos correspondientes.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of files and folders. The "Models" folder is expanded, and "JS productos.js" is selected. Other files like "clientes.js", "compras.js", and "server.js" are also listed.
- Code Editor:** Displays the content of the "productos.js" file. The code defines a Sequelize model for "productos".

```
import { DataTypes } from "sequelize";
import { sequelize } from "../Database/database.js";

//Relacion del modelo con la tabla productos
const Producto = sequelize.define(
  "productos",
  {
    // Definición de Atributos
    idProducto: {
      type: DataTypes.INTEGER,
      allowNull: false,
      primaryKey: true,
      autoIncrement: true,
    },
    nombre: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    detalle: {
      type: DataTypes.STRING,
      allowNull: false,
    },
    precio: {
      type: DataTypes.INTEGER,
      allowNull: false,
    },
  },
);
```
- Terminal:** Shows the message "Servidor escuchando por puerto 3000".
- Status Bar:** Shows the current file as "productos.js - BackendFruverPM - Visual Studio Code", the line count as "Ln 30, Col 23 (19 selected)", and the date/time as "28/07/2023 8:10 p.m."

## **Modelo de clientes**

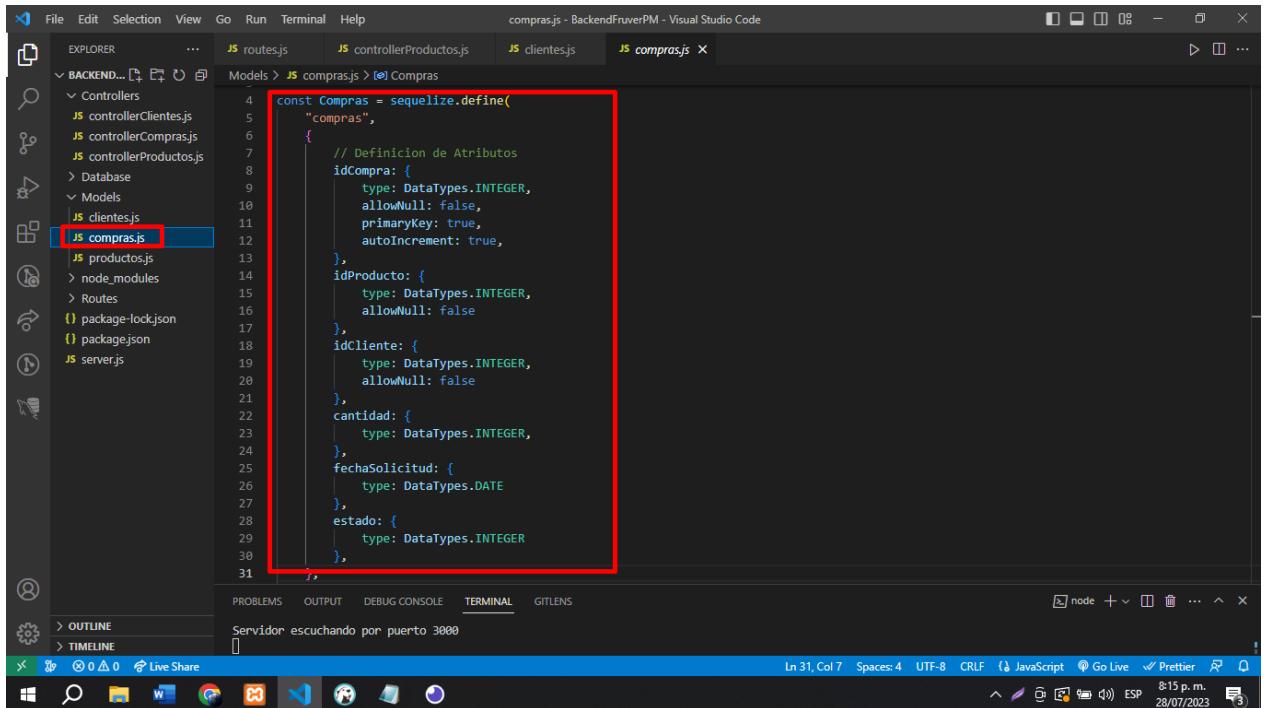
Creamos el archivo **clientes.js** y realizamos el mismo proceso, pero con los atributos de la tabla clientes.

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like routes.js, controllerProductos.js, and clientes.js.
- Editor Area (Center):** Displays the code for the clientes.js file, which defines a Sequelize model for 'clientes'. The code is highlighted with syntax coloring and includes annotations for attributes like idCliente, nombres, correo, contrasena, and rol.
- Bottom Status Bar:** Shows the command bar with node, terminal, and git icons.
- Bottom Taskbar:** Shows the status "Servidor escuchando por puerto 3000".
- Bottom Right:** Includes file navigation icons and a status bar with file path (clients.js - BackendFruverPM - Visual Studio Code), line count (Ln 4, Col 30), character count (Spaces: 4), encoding (UTF-8), and other developer tools.

## Modelo compras

Creamos el archivo **compras.js** y realizamos el mismo proceso, pero con los atributos de la tabla compras.



```
const Compras = sequelize.define('compras', {
  // Definicion de Atributos
  idCompra: {
    type: DataTypes.INTEGER,
    allowNull: false,
    primaryKey: true,
    autoIncrement: true,
  },
  idProducto: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  idCliente: {
    type: DataTypes.INTEGER,
    allowNull: false
  },
  cantidad: {
    type: DataTypes.INTEGER,
  },
  fechaSolicitud: {
    type: DataTypes.DATE
  },
  estado: {
    type: DataTypes.INTEGER
  },
})
```

## 2.5 Controllers

En Node.js, el archivo controllers(controladores) es una parte importante de la arquitectura del patrón Modelo-Vista-Controlador (MVC). El propósito principal de los controladores es manejar la lógica de negocio y actuar como intermediarios entre las rutas (routers) y los modelos de datos.

En este contexto, los controladores son módulos que se encargan de recibir las solicitudes del cliente (por ejemplo, una solicitud HTTP) desde las rutas definidas en la aplicación. Luego, el controlador procesa esa solicitud utilizando la lógica del negocio adecuada y realiza interacciones con los modelos para obtener o modificar los datos necesarios.

una solicitud HTTP consta de elementos como:

**Métodos:** Los cuales especifican el tipo de acción que se desea realizar sobre el recurso. Para esta aplicación haremos uso de los siguientes:

**GET:** Solicitar datos del servidor (lectura).

**POST:** Enviar datos al servidor para crear un nuevo recurso (creación).

**PUT:** Enviar datos al servidor para actualizar un recurso existente (actualización).

**DELETE:** Solicitar al servidor eliminar un recurso específico (eliminación).

Es importante mencionar los códigos de estado HTTP, los cuales son códigos numéricos que se utilizan para indicar el resultado de una solicitud HTTP realizada por un cliente al servidor.

Para el desarrollo de esta aplicación se hará uso de los códigos HTTP más comunes como son:

**Código de estado 200 - OK:** indica que la solicitud ha sido exitosa y el servidor ha devuelto la respuesta solicitada.

**El código de estado HTTP 400 Bad Request:** indica que el servidor no pudo comprender la solicitud del cliente, ya que esta es incorrecta o mal formada. En otras palabras, el servidor no puede procesar la solicitud porque no entiende la sintaxis de la solicitud.

**Código de estado 401- No autorizado:** se utiliza para indicar que la solicitud requiere autenticación, es decir, el cliente no tiene permiso para acceder al recurso solicitado sin proporcionar credenciales válidas (por ejemplo, inicio de sesión).

**Código de estado 500 - Error del servidor interno:** indica que se ha producido un error en el servidor mientras intentaba cumplir con una solicitud. Este código es un código de error genérico que se utiliza cuando el servidor encuentra una situación inesperada y no puede completar la solicitud del cliente correctamente.

Teniendo en cuenta la información anterior, Procedemos a realizar los controladores para cada uno de los modelos:

## Controlador de producto

Definimos las funciones `getProductos`, `getProducto`, `postProducto`, `putProducto` y `deleteProducto`, las cuales se utilizan para realizar diferentes operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre la tabla `productos`.

**getProductos**: Esta función permite obtener una lista de todos los productos disponibles en la base de datos.

**getProducto:** Esta función se utiliza para obtener los detalles de un producto específico. En este caso utilizara un identificador único (idProducto).

```
File Edit Selection View Go Run Terminal Help controllerProductos.js - BackendFruterPM - Visual Studio Code

EXPLORER JS routes.js JS controllerProductos.js X JS server.js JS clientes.js JS productos.js
Controllers > JS controllerProductos.js ...
4 import { Producto } from './Models/productos.js';
5
6 //Función para obtener los productos
7 export const getProductos = async (req, res) => {
8   try {
9     //recibe los datos del modelo producto
10    const productos = await Producto.findAll();
11    res.status(200).json(productos);
12  } catch (error) {
13    res.status(400).json({ mensaje: error });
14  }
15};

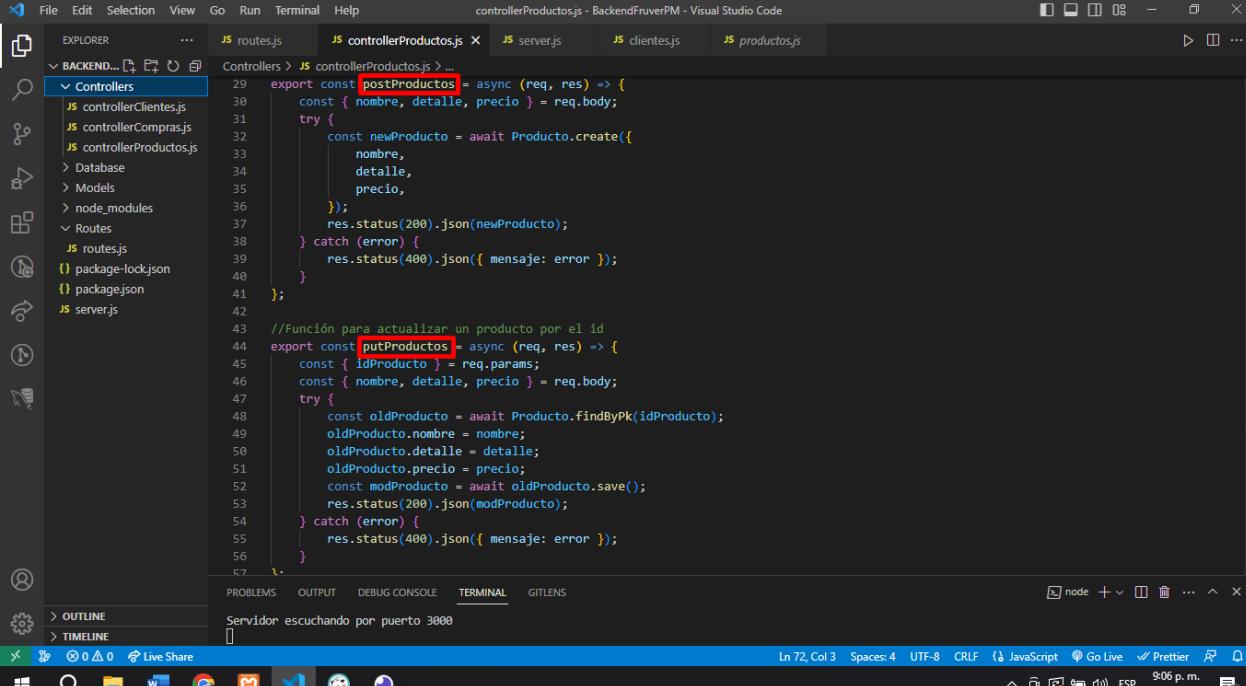
17 //Función para obtener un producto por el id
18 export const getProducto = async (req, res) => {
19   const { idProducto } = req.params;
20   try {
21     const producto = await Producto.findByPk(idProducto);
22     res.status(200).json(producto);
23   } catch (error) {
24     res.status(400).json({ mensaje: error });
25   }
26};

28 //Función para agregar un producto
29 export const postProductos = async (req, res) => {
30   const { nombre, detalle, precio } = req.body;
31   try {
32     const nuevoProducto = await Producto.create();
33   } catch (error) {
34     res.status(400).json({ mensaje: error });
35   }
36};

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
Servidor escuchando por puerto 3000
Ln 72, Col 3 Spaces: 4 UFT-8 CRLF ↴ JavaScript ⚡ Go Live ⚡ Prettier ⚡
9:01 p.m. 28/07/2023
```

**postProducto:** Esta función se utiliza para agregar un nuevo producto a la base de datos. Recibirá los datos del nuevo producto como parámetros de entrada (como nombre, detalle y precio) y luego los guardará en la base de datos.

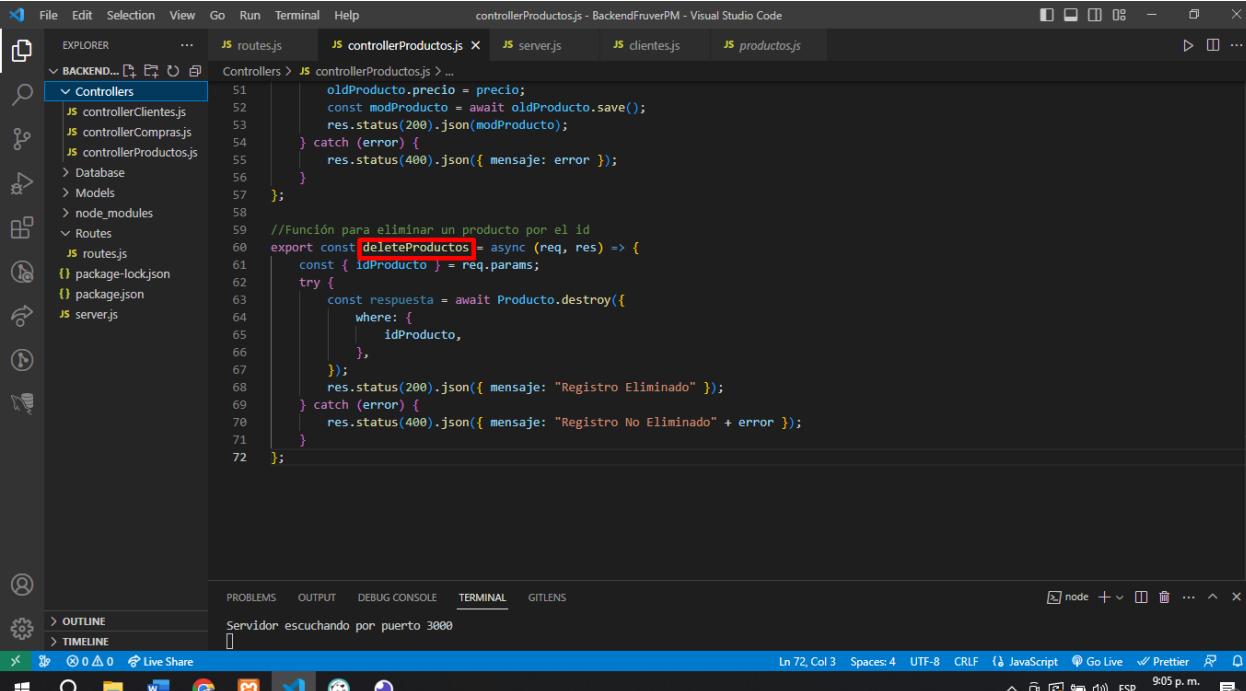
**putProducto:** Esta función se utiliza para actualizar la información de un producto existente en la base de datos. Utilizará un identificador único del producto (idProducto) que se desea actualizar y los nuevos datos que se desean modificar.



```
File Edit Selection View Go Run Terminal Help controllerProductos.js - BackendFruverPM - Visual Studio Code
EXPLORER JS routesjs JS controllerProductos.js X JS server.js JS clientes.js JS productos.js
Controllers > JS controllerProductos.js > ...
29 export const postProductos = async (req, res) => {
30   const { nombre, detalle, precio } = req.body;
31   try {
32     const newProducto = await Producto.create({
33       nombre,
34       detalle,
35       precio,
36     });
37     res.status(200).json(newProducto);
38   } catch (error) {
39     res.status(400).json({ mensaje: error });
40   }
41 };
42
43 //Función para actualizar un producto por el id
44 export const putProductos = async (req, res) => {
45   const { idProducto } = req.params;
46   const { nombre, detalle, precio } = req.body;
47   try {
48     const oldProducto = await Producto.findById(idProducto);
49     oldProducto.nombre = nombre;
50     oldProducto.detalle = detalle;
51     oldProducto.precio = precio;
52     const modProducto = await oldProducto.save();
53     res.status(200).json(modProducto);
54   } catch (error) {
55     res.status(400).json({ mensaje: error });
56   }
57 };

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
Servidor escuchando por puerto 3000
Ln 72, Col 3 Spaces: 4 UTF-8 ⓘ JavaScript ⌂ Go Live ⌂ Prettier ⌂ node + ⌂ ... ⌂
28/07/2023
```

**deleteProducto:** Esta función permite eliminar un producto de la base de datos. Utilizará un identificador único del producto (idProducto) que se desea eliminar y luego procedería a eliminarlo de la base de datos.



```
File Edit Selection View Go Run Terminal Help controllerProductos.js - BackendFruverPM - Visual Studio Code
EXPLORER JS routesjs JS controllerProductos.js X JS server.js JS clientes.js JS productos.js
Controllers > JS controllerProductos.js > ...
51   oldProducto.precio = precio;
52   const modProducto = await oldProducto.save();
53   res.status(200).json(modProducto);
54 } catch (error) {
55   res.status(400).json({ mensaje: error });
56 }

57
58 //Función para eliminar un producto por el id
59 export const deleteProductos = async (req, res) => {
60   const { idProducto } = req.params;
61   try {
62     const respuesta = await Producto.destroy({
63       where: {
64         idProducto,
65       },
66     });
67     res.status(200).json({ mensaje: "Registro Eliminado" });
68   } catch (error) {
69     res.status(400).json({ mensaje: "Registro No Eliminado" + error });
70   }
71 }

72 };

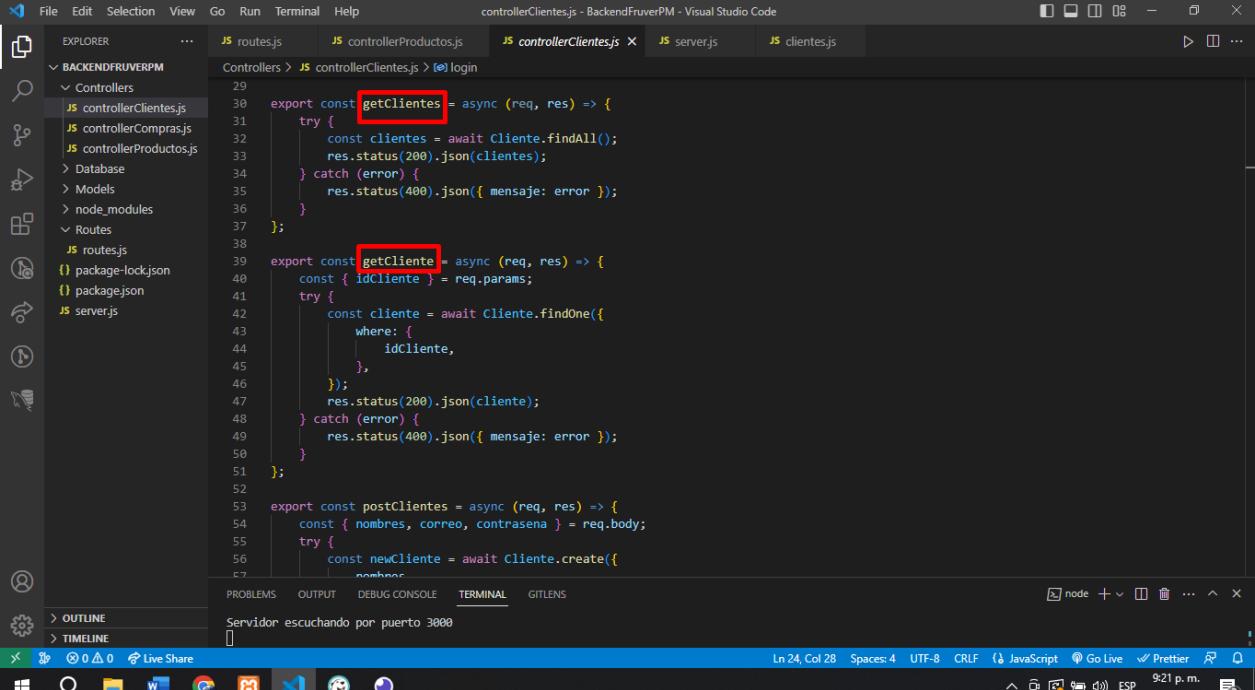
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS
Servidor escuchando por puerto 3000
Ln 72, Col 3 Spaces: 4 UTF-8 ⓘ JavaScript ⌂ Go Live ⌂ Prettier ⌂ node + ⌂ ... ⌂
28/07/2023
```

## Controlador de clientes

Definimos las funciones `getCliente`, `getCliente`, `postCliente`, `putCliente` y `deleteCliente`, las cuales se utilizan para realizar diferentes operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre la tabla `clientes`.

**getClientes:** Esta función permite obtener una lista de todos los clientes disponibles en la base de datos.

**getCliente:** Esta función se utiliza para obtener los datos de un cliente utilizando su `IdCliente`.



```
File Edit Selection View Go Run Terminal Help controllerClientes.js - BackendFruverPM - Visual Studio Code
EXPLORER JS routes.js JS controllerProductos.js JS controllerClientes.js X JS server.js JS clientes.js
Controllers > JS controllerClientes.js > login
29
30 export const getClientes = async (req, res) => {
31   try {
32     const clientes = await Cliente.findAll();
33     res.status(200).json(clientes);
34   } catch (error) {
35     res.status(400).json({ mensaje: error });
36   }
37 };
38
39 export const getCliente = async (req, res) => {
40   const { idCliente } = req.params;
41   try {
42     const cliente = await Cliente.findOne({
43       where: {
44         idCliente,
45       },
46     });
47     res.status(200).json(cliente);
48   } catch (error) {
49     res.status(400).json({ mensaje: error });
50   }
51 };
52
53 export const postClientes = async (req, res) => {
54   const { nombres, correo, contrasena } = req.body;
55   try {
56     const newCliente = await Cliente.create({
57       nombres,
```

**postCliente:** Esta función se utilizará para agregar un nuevo cliente a la base de datos. Recibirá los datos del nuevo cliente (nombres, correo, contraseña y rol) y luego los guardará en la base de datos.

**putCliente:** Esta función se utilizará para actualizar la información de un cliente existente en la tabla `clientes` utilizando su ID.

```

    53 export const postClientes = async (req, res) => {
    54   const { nombres, correo, contrasena } = req.body;
    55   try {
    56     const nuevoCliente = await Cliente.create({
    57       nombres,
    58       correo,
    59       contrasena,
    60     });
    61     res.status(200).json(nuevoCliente);
    62   } catch (error) {
    63     res.status(400).json({ mensaje: error });
    64   }
    65 };
    66 };

    69 export const putClientes = async (req, res) => {
    70   const { idCliente } = req.params;
    71   const { nombres, correo, contrasena } = req.body;
    72   try {
    73     const clienteExistente = await Cliente.findById(idCliente);
    74     clienteExistente.nombres = nombres;
    75     clienteExistente.correo = correo;
    76     clienteExistente.contrasena = contrasena;
    77
    78     const clienteModificado = await clienteExistente.save();
    79     res.status(200).json(clienteModificado);
    80   } catch (error) {
    81     res.status(400).json({ mensaje: error });
    82   }
    83 };

    85 export const deleteClientes = async (req, res) => {
    86   const { idCliente } = req.params;
    87   try {
    88     const respuesta = await Cliente.destroy({
    89       where: {
    90         idCliente,
    91       },
    92     });
    93     res.status(200).json({ mensaje: "Registro Eliminado" });
    94   } catch (error) {
    95     res.status(400).json({ mensaje: "Registro No Eliminado" + error });
    96   }
    97 };
    98 };

```

**deleteCliente:** Esta función se utilizará para eliminar un cliente existente de la tabla clientes utilizando su ID.

```

    71 const { nombres, correo, contrasena } = req.body;
    72 try {
    73   const clienteExistente = await Cliente.findById(idCliente);
    74   clienteExistente.nombres = nombres;
    75   clienteExistente.correo = correo;
    76   clienteExistente.contrasena = contrasena;
    77
    78   const clienteModificado = await clienteExistente.save();
    79   res.status(200).json(clienteModificado);
    80 } catch (error) {
    81   res.status(400).json({ mensaje: error });
    82 }

    85 export const deleteClientes = async (req, res) => {
    86   const { idCliente } = req.params;
    87   try {
    88     const respuesta = await Cliente.destroy({
    89       where: {
    90         idCliente,
    91       },
    92     });
    93     res.status(200).json({ mensaje: "Registro Eliminado" });
    94   } catch (error) {
    95     res.status(400).json({ mensaje: "Registro No Eliminado" + error });
    96   }
    97 };
    98 };

```

## Controlador compras

Definimos las funciones getCompras, postCompras, putCompras y deleteCompras, las cuales se utilizan para realizar diferentes operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre la tabla compras.

**getCompras:** Esta función permite obtener información de las compras existentes en la tabla compras.

**postCompras:** Esta función se utiliza para agregar una nueva compra a la tabla compras. Recibirá los datos de la nueva compra (idProducto, idCliente, cantidad) y los guardará en la base de datos.

```
File Edit Selection View Go Run Terminal Help controllerCompras.js - BackendFruruPM - Visual Studio Code

EXPLORER JS routes.js JS server.js JS controllerCompras.js JS controllerClientes.js
BACKENDFRURUVERPM Controllers > JS controllerCompras.js ...
Controllers > JS controllerCompras.js ...
1 // Importar el modelo compras
2 import { Compras } from '../Models/compras.js';
3
4 export const getCompras = async (req, res) => {
5     try {
6         const compras = await Compras.findAll();
7         res.status(200).json(compras);
8     } catch (error) {
9         res.status(400).json({ mensaje: error });
10    }
11}
12
13 export const postCompras = async (req, res) => {
14     const { idCliente, idProducto, cantidad } = req.body;
15     try {
16         const newCompra = await Compras.create({
17             idCliente,
18             idProducto,
19             cantidad
20         });
21         res.status(200).json(newCompra);
22     } catch (error) {
23
24         res.status(400).json({ mensaje: error });
25     }
26 }
27
28 export const putCompras = async (req, res) => {
29     const { idCompra } = req.params;
30 }
```

**putCompras:** Esta función se utiliza para actualizar una compra existente en la tabla compras. Se Debes pasar el idCompra y los datos que se desea actualizar de la compra.

**deleteCompras:** Esta función se utiliza para eliminar una compra existente de la tabla compras. Se debe pasar el idCompra de la compra que se desea eliminar.

```
File Edit Selection View Go Run Terminal Help controllerCompras.js - BackendFruterPM - Visual Studio Code

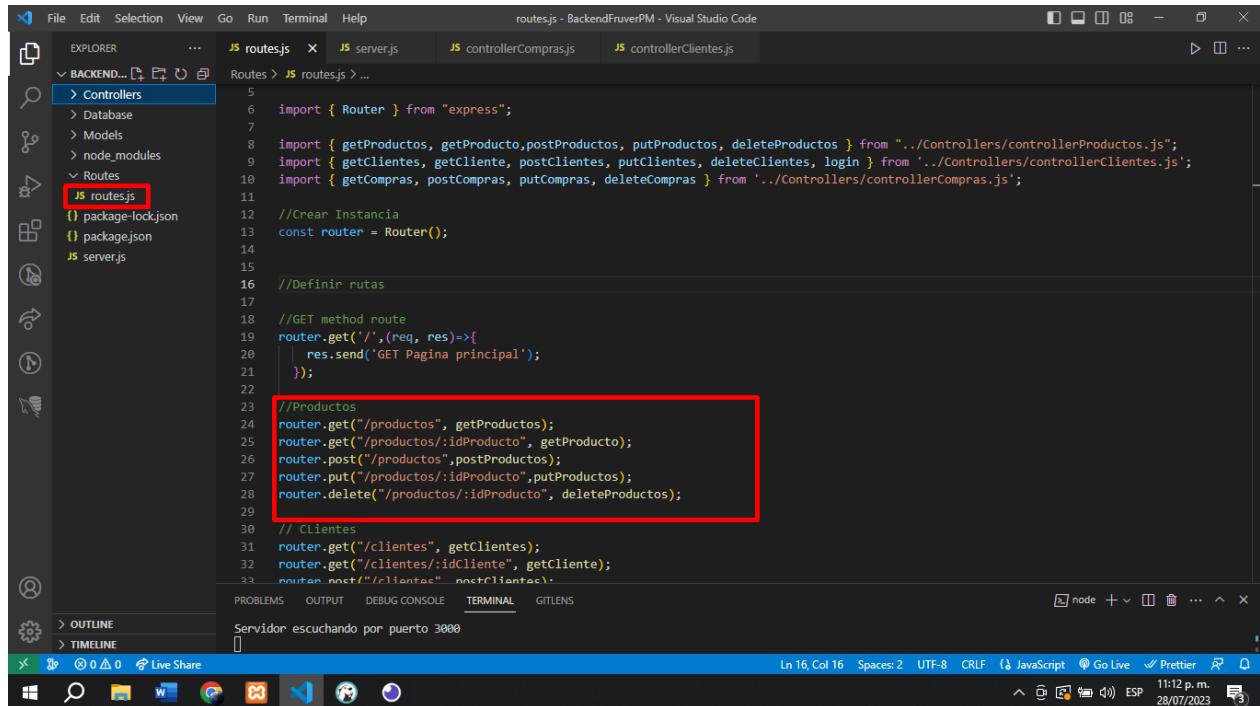
EXPLORER JS routes.js JS server.js JS controllerCompras.js × JS controllerClientes.js

BACKENDFRUTERPM Controllers > JS controllerCompras.js > ...
2/
28 export const putCompras = async (req, res) => {
29   const { idCompra } = req.params;
30   const { idCliente, idProducto, cantidad, estado } = req.body;
31   try {
32     const oldCompra = await Compras.findByPk(idCompra);
33     oldCompra.idCliente = idCliente;
34     oldCompra.idProducto = idProducto;
35     oldCompra.cantidad = cantidad;
36     oldCompra.estado = estado;
37
38     const modCompra = await oldCompra.save();
39     res.status(200).json(modCompra);
40   } catch (error) {
41     console.log(error);
42     res.status(400).json({ mensaje: error });
43   }
44 }

45
46
47 export const deleteCompras = async (req, res) => {
48   const { idCompra } = req.params;
49   try {
50     const respuesta = await Compras.destroy({
51       where: {
52         idCompra,
53       },
54     });
55   }
56 }
```

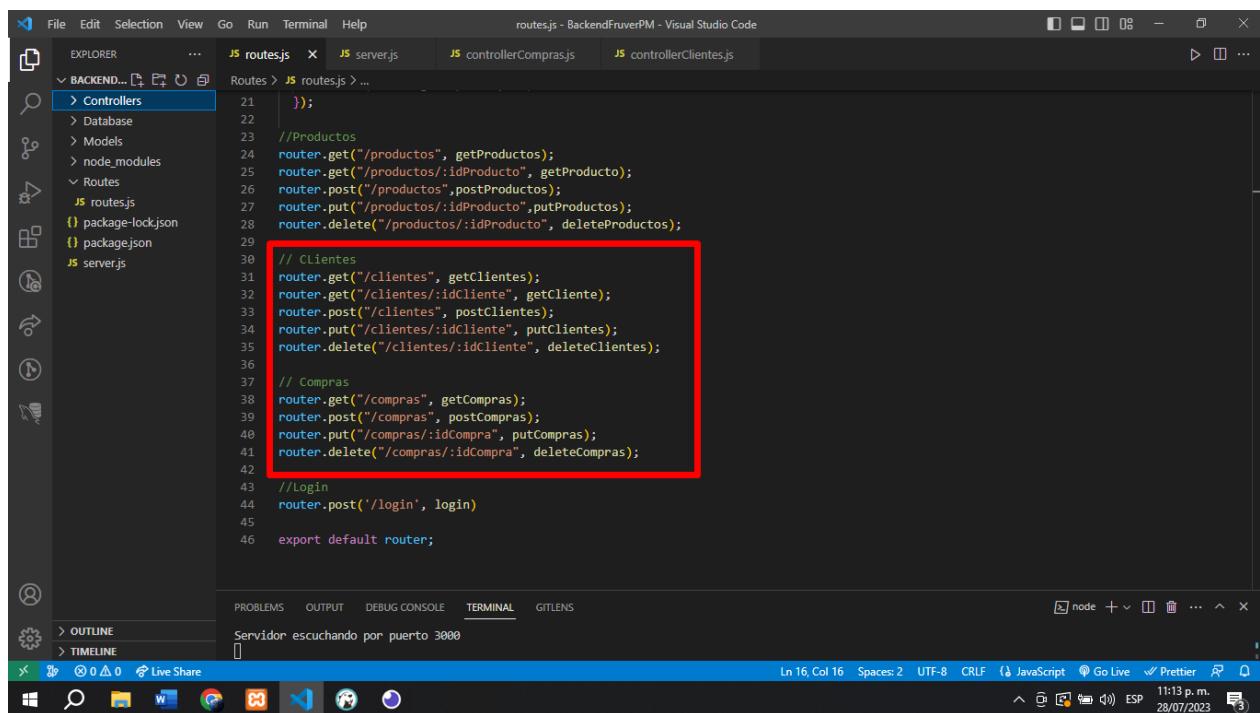
## 2.6 Definición de las rutas

Para organizar y definir las rutas en Express.js, crearemos un archivo llamado routes.js. En este archivo, importaremos Express y definiremos las rutas para cada operación CRUD (Crear, Leer, Actualizar y Eliminar) de la tabla productos, cliente y compras.



```
File Edit Selection View Go Run Terminal Help
routes.js - BackendFruverPM - Visual Studio Code
EXPLORER JS routes.js JS server.js JS controllerCompras.js JS controllerClientes.js
BACKEND... Routes > JS routes.js > ...
JS routes.js
5 import { Router } from "express";
6 import { getProductos, getProducto, postProductos, putProductos, deleteProductos } from "../Controllers/controllerProductos.js";
7 import { getClientes, getClient, postClientes, putClientes, deleteClientes, login } from '../Controllers/controllerClientes.js';
8 import { getCompras, postCompras, putCompras, deleteCompras } from '../Controllers/controllerCompras.js';
9
10 //Crear Instancia
11 const router = Router();
12
13 //Definir rutas
14
15 //GET method route
16 router.get('/',(req, res)=>{
17   | res.send('GET Pagina principal');
18 });
19
20 //Productos
21 router.get("/productos", getProductos);
22 router.get("/productos/:idProducto", getProducto);
23 router.post("/productos",postProductos);
24 router.put("/productos/:idProducto",putProductos);
25 router.delete("/productos/:idProducto", deleteProductos);
26
27 // Clientes
28 router.get("/clientes", getClientes);
29 router.get("/clientes/:idCliente", getClient);
30 router.post("/clientes", postClientes);
31 router.put("/clientes/:idCliente", putClientes);
32 router.delete("/clientes/:idCliente", deleteClientes);
33
34 // Compras
35 router.get("/compras", getCompras);
36 router.post("/compras", postCompras);
37 router.put("/compras/:idCompra", putCompras);
38 router.delete("/compras/:idCompra", deleteCompras);
39
40 //Login
41 router.post('/login', login)
42
43 export default router;
```

The screenshot shows the Visual Studio Code interface with the routes.js file open. The code defines a Router instance and sets up routes for products, clients, and purchases. The product routes (lines 21-29) are highlighted with a red box. The client and purchase routes are also defined but not highlighted.



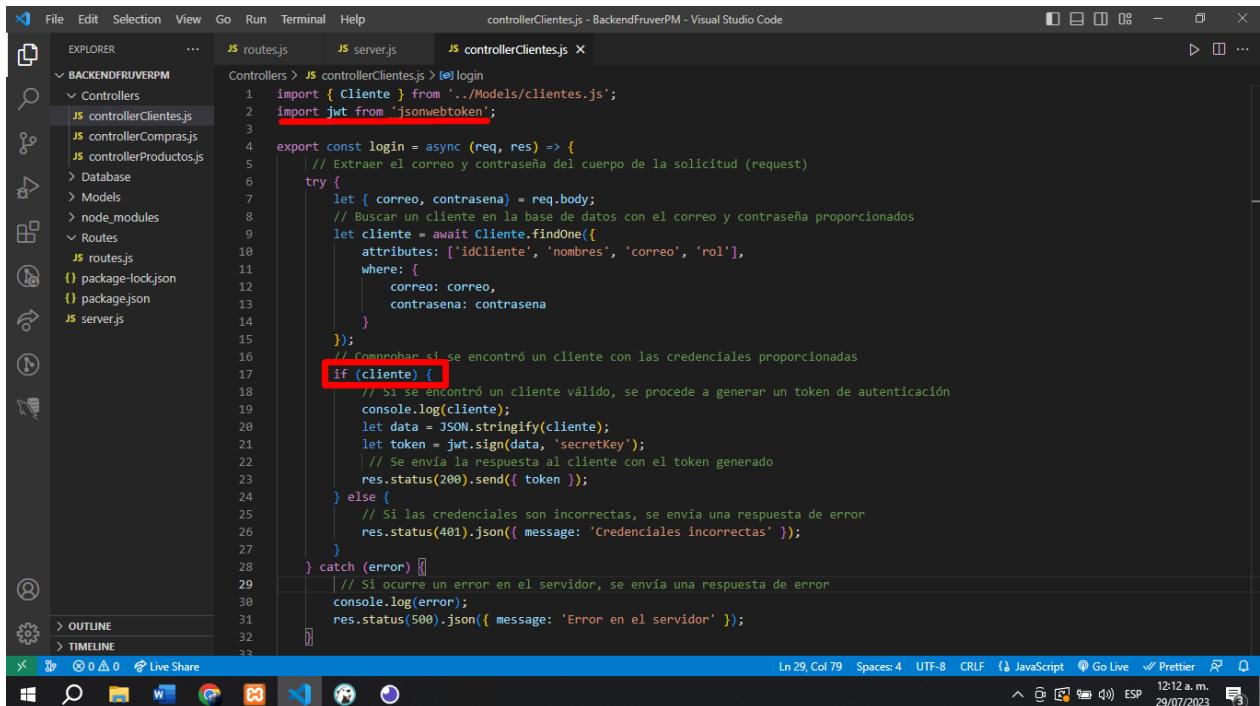
```
File Edit Selection View Go Run Terminal Help
routes.js - BackendFruverPM - Visual Studio Code
EXPLORER JS routes.js JS server.js JS controllerCompras.js JS controllerClientes.js
BACKEND... Routes > JS routes.js > ...
JS routes.js
21 };
22
23 //Productos
24 router.get("/productos", getProductos);
25 router.get("/productos/:idProducto", getProducto);
26 router.post("/productos",postProductos);
27 router.put("/productos/:idProducto",putProductos);
28 router.delete("/productos/:idProducto", deleteProductos);
29
30 // Clientes
31 router.get("/clientes", getClientes);
32 router.get("/clientes/:idCliente", getClient);
33 router.post("/clientes", postClientes);
34 router.put("/clientes/:idCliente", putClientes);
35 router.delete("/clientes/:idCliente", deleteClientes);
36
37 // Compras
38 router.get("/compras", getCompras);
39 router.post("/compras", postCompras);
40 router.put("/compras/:idCompra", putCompras);
41 router.delete("/compras/:idCompra", deleteCompras);
42
43 //Login
44 router.post('/login', login)
45
46 export default router;
```

This screenshot shows the same Visual Studio Code interface as the previous one, but the code has been modified. The client and purchase routes (lines 30-45) are now highlighted with a red box, while the product routes remain unhighlighted.

## 2.7 Login- inicio de sesión

Para la implementación de la función de login se hace uso de la tabla clientes y se utiliza la biblioteca jsonwebtoken (jwt) para generar tokens de autenticación.

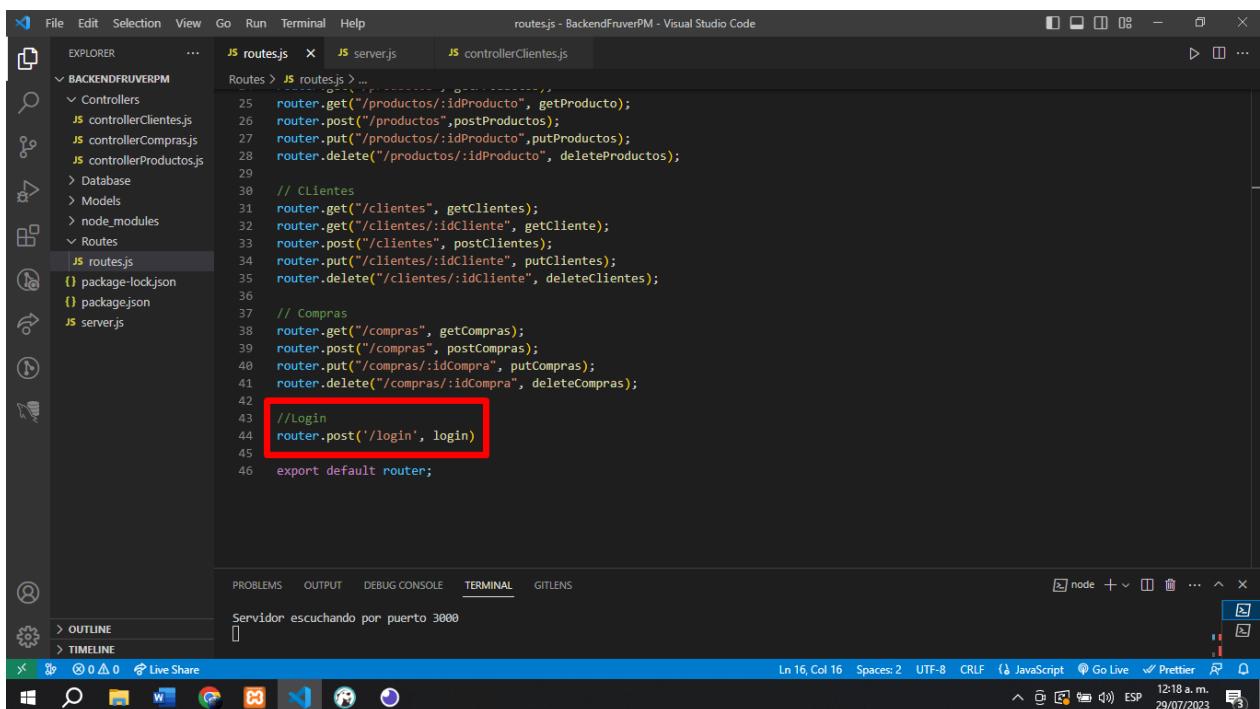
En el archivo controllerClientes.js se implementa la función login. Se busca en la base de datos si existe un cliente con esas credenciales y se responde con un token de autenticación si las credenciales son válidas. Caso contrario o si ocurre algún error en el servidor, se envían respuestas de error.



```
File Edit Selection View Go Run Terminal Help controllerClientes.js - BackendFruberPM - Visual Studio Code

EXPLORER JS routesjs JS server.js JS controllerClientes.js
Controllers > JS controllerClientes.js > login
1 import { Client } from '../Models/clientes.js';
2 import jwt from 'jsonwebtoken';
3
4 export const login = async (req, res) => {
5   // Extraer el correo y contraseña del cuerpo de la solicitud (request)
6   try {
7     let { correo, contrasena } = req.body;
8     // Buscar un cliente en la base de datos con el correo y contraseña proporcionados
9     let cliente = await Client.findOne({
10       attributes: ['idCliente', 'nombres', 'correo', 'rol'],
11       where: {
12         correo: correo,
13         contrasena: contrasena
14       }
15     });
16     // Comprobar si se encontró un cliente con las credenciales proporcionadas
17     if (cliente) {
18       // Si se encontró un cliente válido, se procede a generar un token de autenticación
19       console.log(cliente);
20       let data = JSON.stringify(cliente);
21       let token = jwt.sign(data, 'secretKey');
22       // Se envía la respuesta al cliente con el token generado
23       res.status(200).send({ token });
24     } else {
25       // Si las credenciales son incorrectas, se envía una respuesta de error
26       res.status(401).json({ message: 'Credenciales incorrectas' });
27     }
28   } catch (error) {
29     // Si ocurre un error en el servidor, se envía una respuesta de error
30     console.log(error);
31     res.status(500).json({ message: 'Error en el servidor' });
32   }
33 }
34
35
36
37
38
39
40
41
42
43
44
45
46
```

Por último se define la ruta en el archivo routes.js



```
File Edit Selection View Go Run Terminal Help routesjs - BackendFruberPM - Visual Studio Code

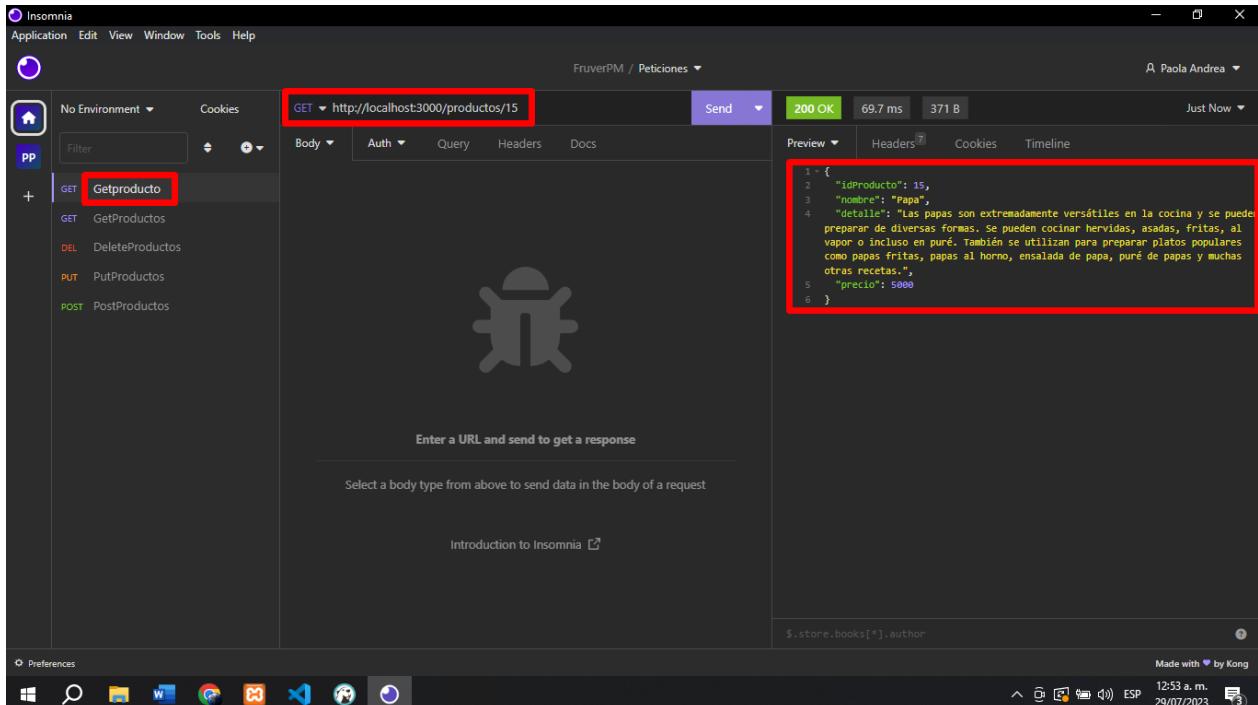
EXPLORER JS routesjs X JS server.js JS controllerClientes.js
Routes > JS routesjs > ...
1 router.get('/productos/:idProducto', getProducto);
2 router.post('/productos', postProductos);
3 router.put('/productos/:idProducto', putProductos);
4 router.delete('/productos/:idProducto', deleteProductos);
5
6 // Clientes
7 router.get('/clientes', getClientes);
8 router.get('/clientes/:idCliente', getCliente);
9 router.post('/clientes', postClientes);
10 router.put('/clientes/:idCliente', putClientes);
11 router.delete('/clientes/:idCliente', deleteClientes);
12
13 // Compras
14 router.get('/compras', getCompras);
15 router.post('/compras', postCompras);
16 router.put('/compras/:idCompra', putCompras);
17 router.delete('/compras/:idCompra', deleteCompras);
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
```

### 3. Realizar verificación de las diferentes operaciones a través de Insomnia.

Se procede a realizar prueba de conexiones con Insomnia. Esta aplicación permite realizar pruebas de API y conexiones HTTP. Es una herramienta útil para realizar pruebas de Backend y comprobar que los servicios funcionen correctamente antes de integrarla con otras partes de la aplicación o con servicios externos.

#### 3.1 Productos

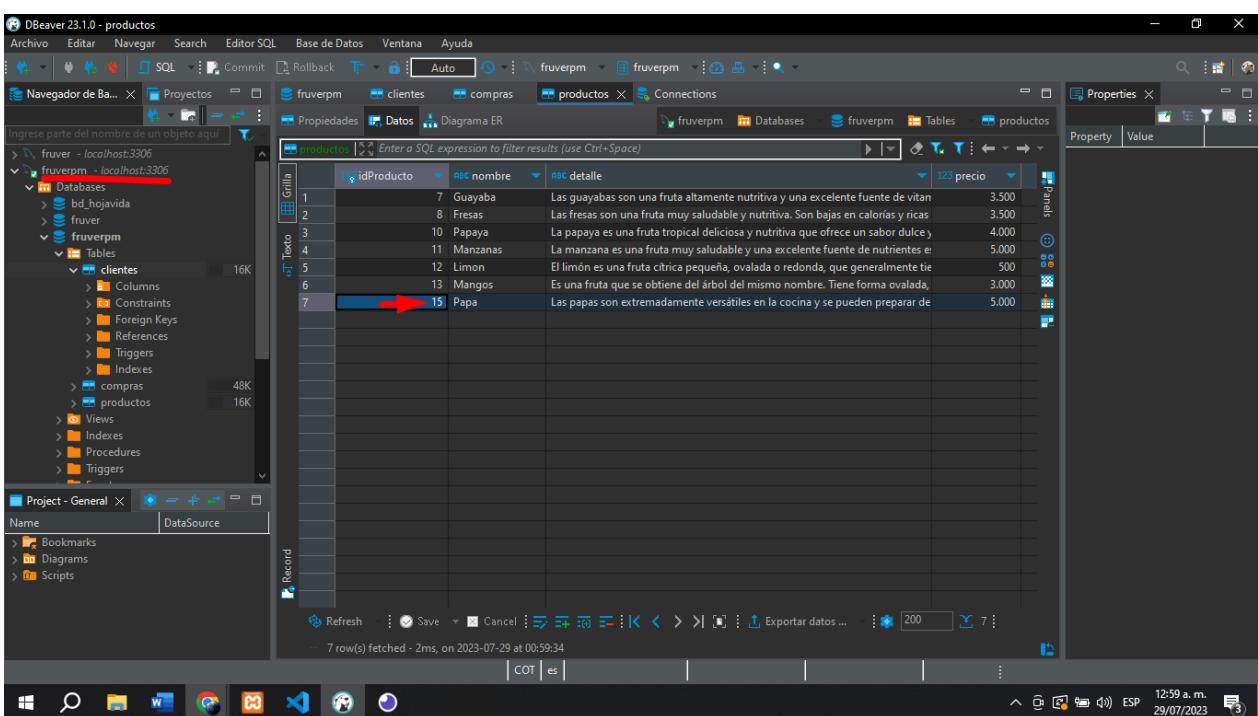
GetProducto id: 15, producto: papa



The screenshot shows the Insomnia REST Client interface. In the top navigation bar, the URL `GET http://localhost:3000/productos/15` is entered. The status bar indicates a `200 OK` response with `69 ms` and `371 B`. The response body is displayed in a code editor-like panel, with the JSON content for product ID 15 highlighted in red:

```
1: {
2:   "idProducto": 15,
3:   "nombre": "Papa",
4:   "detalle": "Las papas son extremadamente vers\u00e1tiles en la cocina y se puede preparar de diversas formas. Se pueden cocinar hervidas, asadas, fritas, al vapor o incluso en pur\u00e9. Tambi\u00e9n se utilizan para preparar platos populares como papas fritas, papas al horno, ensalada de papa, pur\u00e9 de papas y muchas otras recetas.",
5:   "precio": 5000
6: }
```

#### Verificación en DBeaver



The screenshot shows the DBeaver 23.1.0 database client interface. The left sidebar shows the database structure for the `fruverpm` database, including tables like `clientes`, `compras`, and `productos`. The main pane displays the `productos` table with the following data:

	idProducto	nombre	detalle	precio
1	7	Guayaba	Las guayabas son una fruta altamente nutritiva y una excelente fuente de vitamina C.	3.500
2	8	Fresas	Las fresas son una fruta muy saludable y nutritiva. Son bajas en calor\u00edas y ricas en fibra.	3.500
3	10	Papaya	La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y jugoso.	4.000
4	11	Manzanas	La manzana es una fruta muy saludable y una excelente fuente de nutrientes esenciales.	5.000
5	12	Limon	El lim\u00f3n es una fruta c\u00f3trica peque\u00e1a, ovalada o redonda, que generalmente tiene un sabor amargo y ac\u00f3nico.	500
6	13	Mangos	Es una fruta que se obtiene del \u00e1rbol del mismo nombre. Tiene forma ovalada, su carne es suave y dulce.	3.000
7	15	Papa	Las papas son extremadamente vers\u00e1tiles en la cocina y se pueden preparar de muchas formas.	5.000

## GetProductos

The screenshot shows the Insomnia REST Client interface. The top bar displays "GET http://localhost:3000/productos" and "200 OK". The left sidebar has a tree view with "GetProductos" selected. The main area shows a JSON response with two fruit entries. The right panel shows the raw JSON code and a preview of the response.

```
1: [
2:   {
3:     "idProducto": 7,
4:     "nombre": "Guayaba",
5:     "detalle": "Las guayabas son una fruta altamente nutritiva y una excelente fuente de vitaminas, minerales y fibra dietética. Entre sus componentes nutricionales se destacan:\n\nvitamina C: Rica en vitamina C, lo que ayuda a fortalecer el sistema inmunológico y promueve una piel sana.\n\nvitamina A: Contiene vitamina A, beneficiosa para la salud ocular y el mantenimiento de las mucosas.",
6:     "precio": 3500
7:   },
8:   {
9:     "idProducto": 8,
10:    "nombre": "Fresas",
11:    "detalle": "Las fresas son una fruta muy saludable y nutritiva. Son bajas en calorías y ricas en vitaminas, minerales y antioxidantes. Entre sus componentes nutricionales se destacan:\n\nvitamina C: Son una excelente fuente de vitamina C, que ayuda a fortalecer el sistema inmunológico y favorece la salud de la piel.\n\nfibra: Son ricas en fibra, lo que favorece la digestión y ayuda a mantener un sistema digestivo saludable.\n\npotasio: Contienen potasio, importante para el equilibrio de fluidos y la función muscular.",
12:    "precio": 3500
13:  },
14:  {
15:    "idProducto": 10,
16:    "nombre": "Papaya",
17:    "detalle": "La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y refrescante. Es una excelente fuente de vitaminas y minerales beneficiosos para la salud. Su versatilidad en la cocina la convierte en una opción popular para disfrutar fresca, en jugos, batidos y"
18:  }
]
```

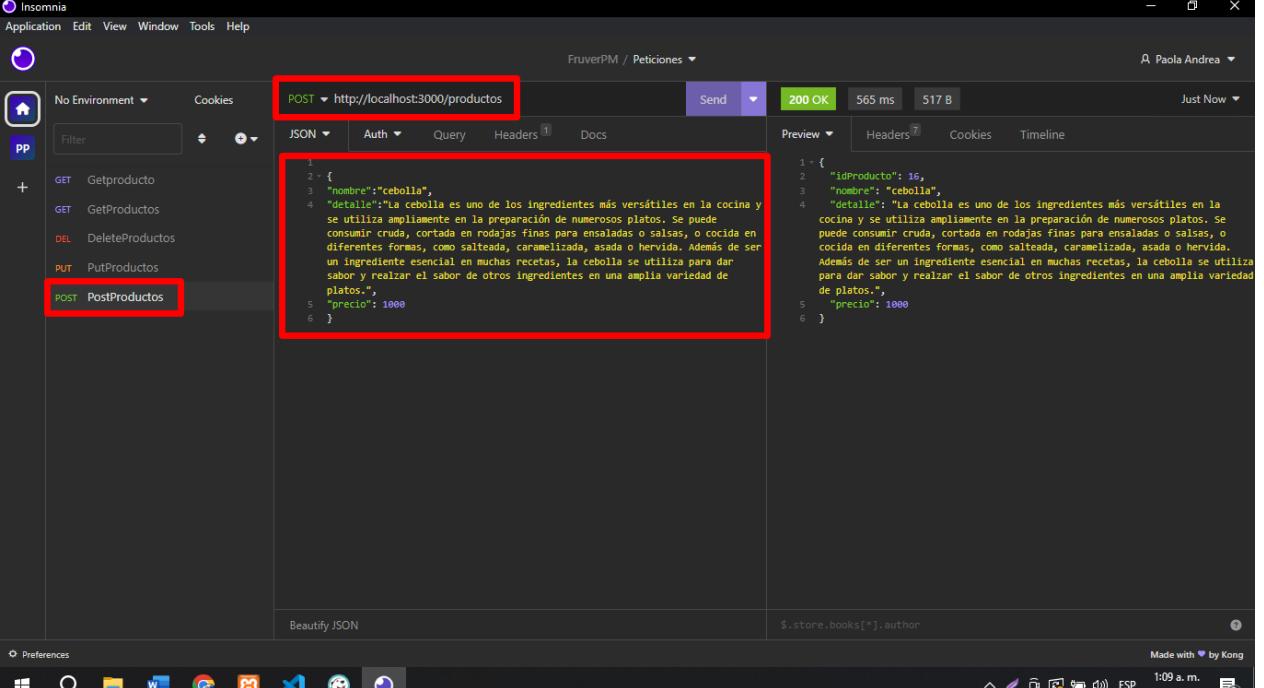
## Verificación en DBeaver

The screenshot shows the DBeaver database interface. The left sidebar shows a tree view of databases and tables. The main area is a grid view of the "productos" table, which contains 15 rows of fruit data. The right panel shows the table's properties.

idProducto	nombre	detalle	precio
1	Guayaba	Las guayabas son una fruta altamente nutritiva y una excelente fuente de vitamina C. Son ricas en fibra dietética y minerales.	3.500
2	Fresas	Las fresas son una fruta muy saludable y nutritiva. Son bajas en calorías y ricas en vitaminas, minerales y antioxidantes.	3.500
3	Papaya	La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y refrescante. Es una excelente fuente de vitaminas y minerales beneficiosos para la salud.	4.000
4	Manzanas	La manzana es una fruta muy saludable y una excelente fuente de nutrientes esenciales.	5.000
5	Limon	El limón es una fruta cítrica pequeña, ovalada o redonda, que generalmente tiene un sabor ácido y amargo.	500
6	Mangos	Es una fruta que se obtiene del árbol del mismo nombre. Tiene forma ovalada, suave y dulce.	3.000
7	Papa	Las papas son extremadamente versátiles en la cocina y se pueden preparar de muchas formas diferentes.	5.000
8			
9			
10			
11			
12			
13			
14			
15			

## PostProducto

se prueba agregando el producto cebolla

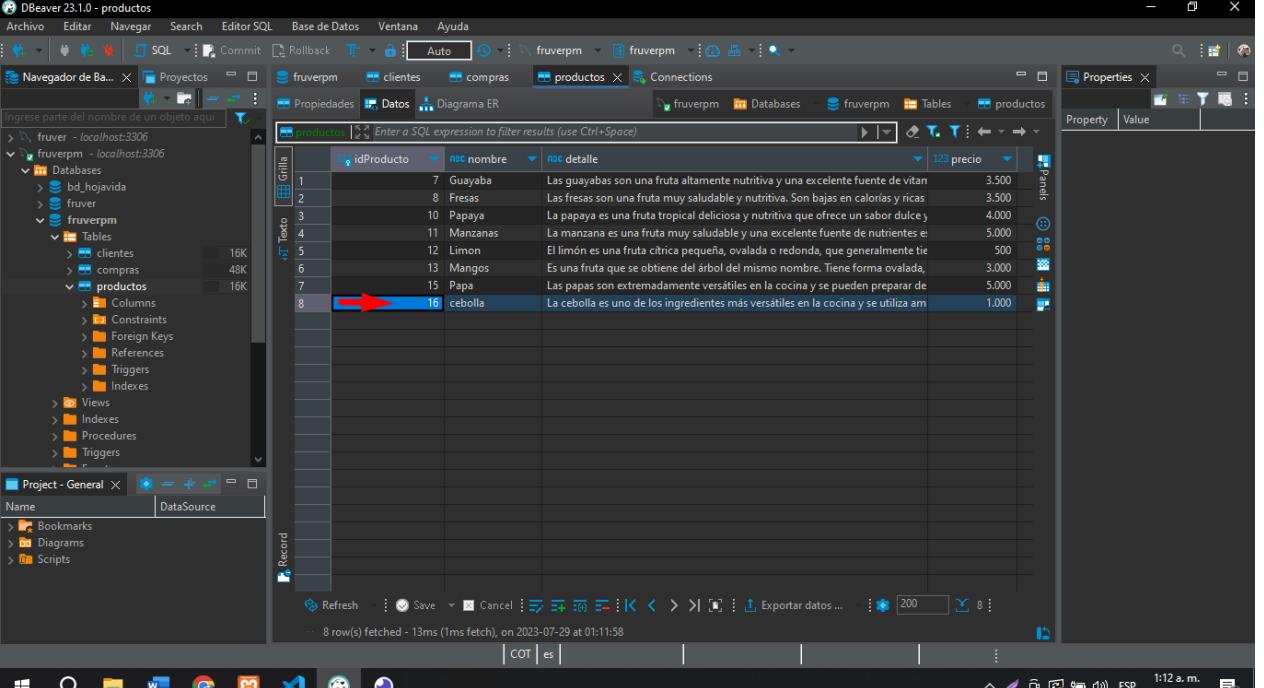


The screenshot shows the Insomnia REST Client interface. The URL in the header is `POST http://localhost:3000/productos`. The response status is `200 OK` with `565 ms` and `517 B`. The request body contains the following JSON:

```
1
2  {
3    "nombre": "cebolla",
4    "detalle": "La cebolla es uno de los ingredientes más versátiles en la cocina y se utiliza ampliamente en la preparación de numerosos platos. Se puede consumir cruda, cortada en rodajas finas para ensaladas o salsas, o cocida en diferentes formas, como salteada, caramelizada, asada o hervida. Además de ser un ingrediente esencial en muchas recetas, la cebolla se utiliza para dar sabor y realzar el sabor de otros ingredientes en una amplia variedad de platos.",
5    "precio": 1000
6 }
```

The response body is identical to the request body.

## Verificación en DBeaver

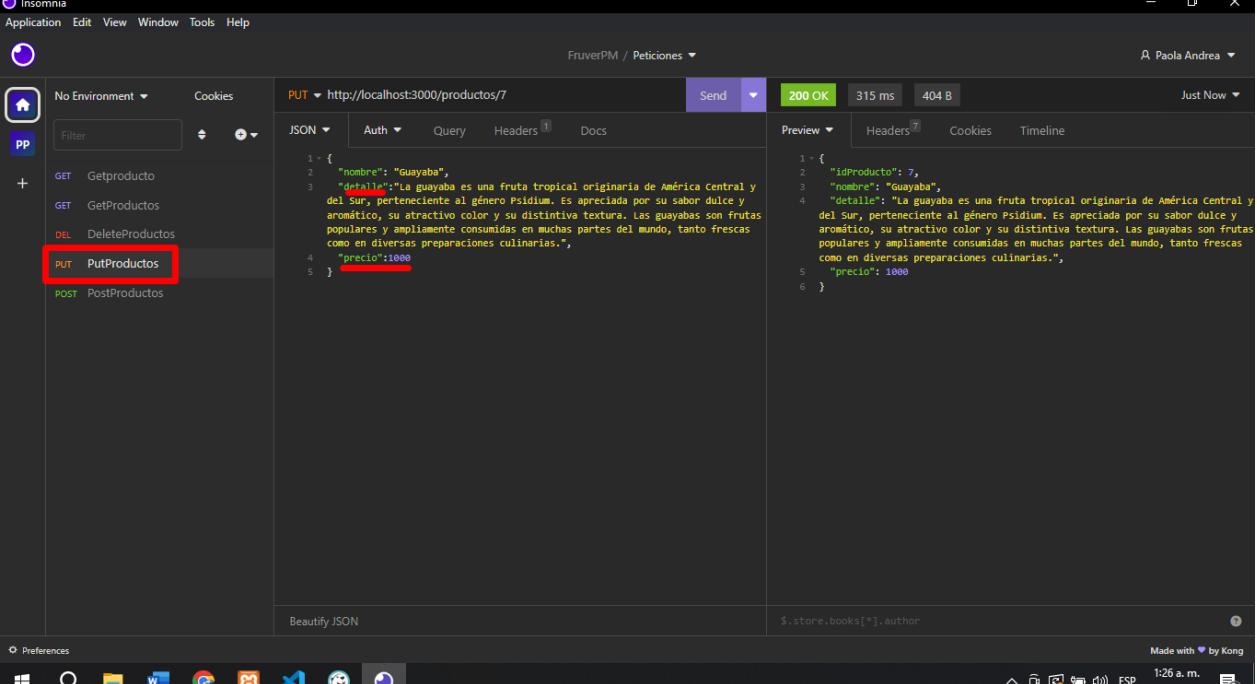


The screenshot shows the DBeaver interface connected to the `fruverpm` database. The `productos` table is selected. A red arrow points to the last inserted row, which contains the product `cebolla`.

	idProducto	nombre	detalle	precio
1	7	Guayaba	Las guayabas son una fruta altamente nutritiva y una excelente fuente de vitamina C.	3.500
2	8	Fresas	Las fresas son una fruta muy saludable y nutritiva. Son bajas en calorías y ricas en fibra.	3.500
3	10	Papaya	La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y jugoso.	4.000
4	11	Manzanas	La manzana es una fruta muy saludable y una excelente fuente de nutrientes.	5.000
5	12	Limon	El limón es una fruta cítrica pequeña, ovalada o redonda, que generalmente tiene un sabor ácido.	500
6	13	Mangos	Es una fruta que se obtiene del árbol del mismo nombre. Tiene forma ovalada, suave y dulce.	3.000
7	15	Papa	Las papas son extremadamente versátiles en la cocina y se pueden preparar de muchas formas.	5.000
8	16	cebolla	La cebolla es uno de los ingredientes más versátiles en la cocina y se utiliza ampliamente en la preparación de numerosos platos.	1.000

## PutProductos

Actualizamos el precio de la guayaba a 1000 y cambiamos su descripción.

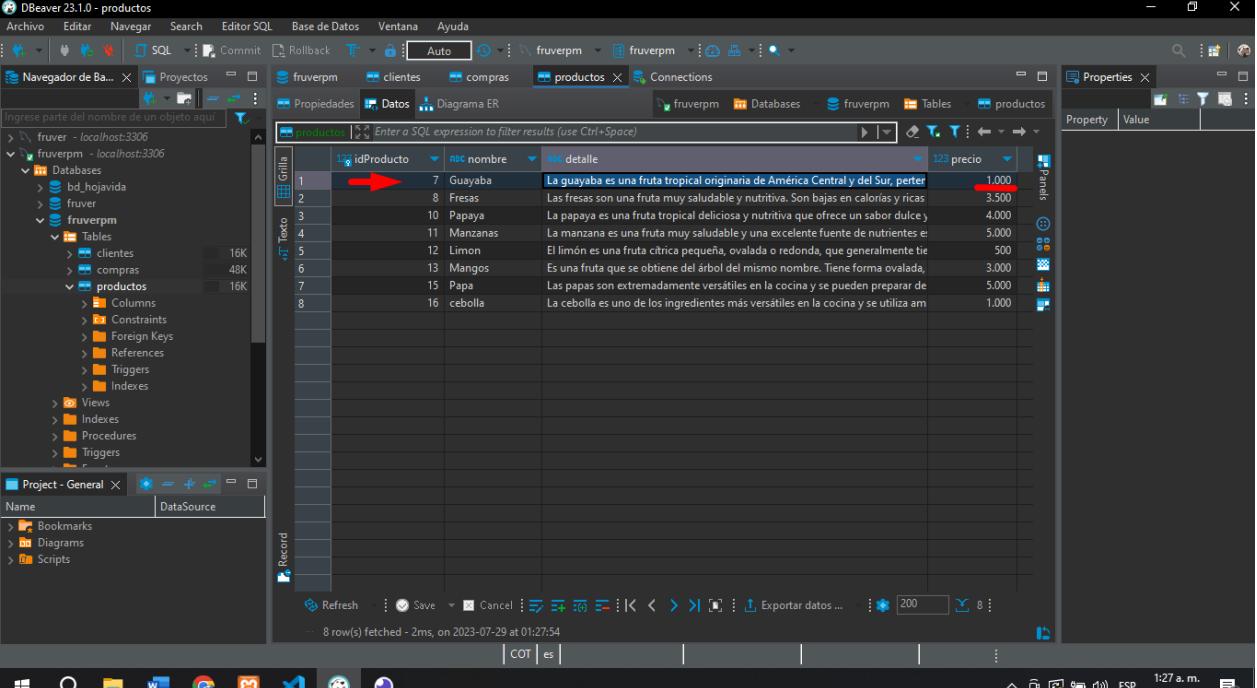


The screenshot shows the Insomnia REST Client interface. On the left, the sidebar lists methods: GET producto, GET Productos, DELETE DeleteProductos, PUT PutProductos (which is selected and highlighted with a red box), and POST PostProductos. The main area shows a PUT request to `http://localhost:3000/productos/7`. The JSON payload is:

```
1: {  
2:   "nombre": "Guayaba",  
3:   "detalle": "La guayaba es una fruta tropical originaria de América Central y del Sur, perteneciente al género Psidium. Es apreciada por su sabor dulce y aromático, su atractivo color y su distintiva textura. Las guayabas son frutas populares y ampliamente consumidas en muchas partes del mundo, tanto frescas como en diversas preparaciones culinarias.",  
4:   "precio": 1000  
5: }
```

The response on the right shows a 200 OK status with a response body identical to the payload, indicating the update was successful.

## Verificación en DBeaver



The screenshot shows the DBeaver 23.1.0 database client interface. The left sidebar shows the database structure for the 'fruverpm' database, including the 'productos' table. The main area displays the contents of the 'productos' table in a grid view. A red arrow points to the row where the product ID is 7, which corresponds to the updated 'Guayaba' entry. The table columns are: idProducto, nombre, detalle, and precio. The updated row shows 'Guayaba' with a new description and a price of 1.000.

	idProducto	nombre	detalle	precio
1	7	Guayaba	La guayaba es una fruta tropical originaria de América Central y del Sur, perteneciente al género Psidium. Es apreciada por su sabor dulce y aromático, su atractivo color y su distintiva textura. Las guayabas son frutas populares y ampliamente consumidas en muchas partes del mundo, tanto frescas como en diversas preparaciones culinarias.	1.000
2	8	Fresas	Las fresas son una fruta muy saludable y nutritiva. Son bajas en calorías y ricas en fibra.	3.500
3	10	Papaya	La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y jugoso.	4.000
4	11	Manzanas	La manzana es una fruta muy saludable y una excelente fuente de nutrientes.	5.000
5	12	Limon	El limón es una fruta cítrica pequeña, ovalada o redonda, que generalmente tiene un sabor ácido.	500
6	13	Mangos	Es una fruta que se obtiene del árbol del mismo nombre. Tiene forma ovalada, suave y dulce.	3.000
7	15	Papa	Las papas son extremadamente versátiles en la cocina y se pueden preparar de muchas formas.	5.000
8	16	cebolla	La cebolla es uno de los ingredientes más versátiles en la cocina y se utiliza ampliamente en muchas recetas.	1.000

## DeleteProductos

Se eliminará el producto con id: 12, producto limón.

The screenshot shows the Insomnia REST client interface. The URL in the header is `DELETE http://localhost:3000/productos/12`. The response body contains the JSON object: 

```
1: {  
2:   "mensaje": "Registro Eliminado"  
3: }
```

. The status bar at the bottom right shows the date and time as 29/07/2023 2:08 a.m.

## Verificación en DBeaver

Podemos verificar que efectivamente se eliminó el producto limón

The screenshot shows the DBeaver database tool interface. The left sidebar shows the database structure with the `productos` table selected. The main pane displays the data in the `productos` table:

idProducto	nombre	detalle	precio
1			
2			
3			
4			
5			
6			
7	Guayaba	La guayaba es una fruta tropical originaria de América Central y del Sur, perteneciente a la familia de las rosáceas. Es conocida por su sabor dulce y jugoso.	1.000
8	Fresas	Las fresas son una fruta muy saludable y nutritiva. Son bajas en calorías y ricas en fibra.	3.500
10	Papaya	La papaya es una fruta tropical deliciosa y nutritiva que ofrece un sabor dulce y jugoso.	4.000
11	Manzanas	La manzana es una fruta muy saludable y una excelente fuente de nutrientes.	5.000
13	Mangos	Es una fruta que se obtiene del árbol del mismo nombre. Tiene forma ovalada, pulpa blanca y sabor dulce.	3.000
15	Papa	Las papas son extremadamente versátiles en la cocina y se pueden preparar de muchas formas.	5.000
16	cébolla	La cebolla es uno de los ingredientes más versátiles en la cocina y se utiliza en numerosas recetas.	1.000

The status bar at the bottom right shows the date and time as 29/07/2023 2:08 a.m.

## 3.2 Clientes

GetCliente: id: 10, cliente: Rosita

The screenshot shows the Insomnia REST client interface. On the left, the sidebar lists API endpoints under 'clients'. A red box highlights the 'GET GetCliente' endpoint. The main panel shows a successful 'GET' request to 'http://localhost:3000/clientes/10'. The response body contains the following JSON:

```
1 - {
2   "idCliente": 10,
3   "nombres": "Rosita",
4   "correo": "Rosy@gmail.com",
5   "contrasena": "123",
6   "rol": "cliente"
7 }
```

The status bar at the bottom right indicates the request was made 'Just Now'.

## Verificación en DBeaver

The screenshot shows the DBeaver 23.1.0 interface. The left sidebar shows the database structure with 'fruverpm' selected. The central area displays the 'clients' table. A red arrow points to the second row, which corresponds to the client 'Rosita'. The table has columns: idCliente, nombres, correo, contrasena, and rol. The data for the second row is:

	idCliente	nombres	correo	contrasena	rol
1	7	paola	paolaan2903@gmail.com	456	cliente
2	10	Rosita	Rosy@gmail.com	123	cliente
3	8	Admin	paolaan766@gmail.com	123	admin
4	9	ingrid	ingrid@gmail.com	456	cliente

## GetClientes

The screenshot shows the Insomnia REST Client interface. In the top navigation bar, the URL `GET http://localhost:3000/clientes` is highlighted with a red box. The status bar indicates `200 OK`, `221 ms`, and `397 B`. The response body contains a JSON array of client data:

```
1+ [
2+   {
3+     "idCliente": 7,
4+     "nombres": "paola",
5+     "correo": "paolaan2903@gmail.com",
6+     "contrasena": "456",
7+     "rol": "cliente"
8+   },
9+   {
10+    "idCliente": 8,
11+    "nombres": "admin",
12+    "correo": "paolaan766@gmail.com",
13+    "contrasena": "123",
14+    "rol": "admin"
15+  },
16+  {
17+    "idCliente": 9,
18+    "nombres": "ingrid",
19+    "correo": "ingrid@gmail.com",
20+    "contrasena": "456",
21+    "rol": "cliente"
22+  },
23+  {
24+    "idCliente": 10,
25+    "nombres": "Rosita",
26+    "correo": "Rosy@gmail.com",
27+    "contrasena": "123",
28+    "rol": "cliente"
29+  }
30+ ]
```

The bottom right corner of the window displays the message `Made with ❤ by Kong`.

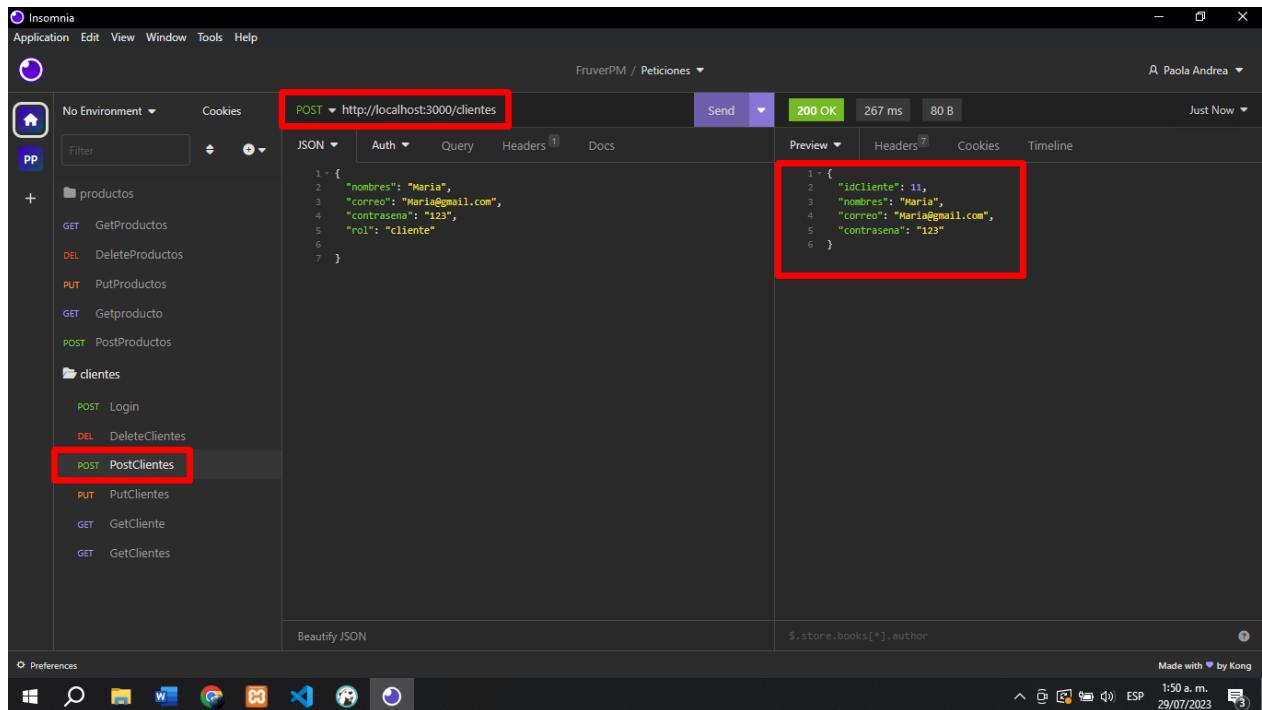
## Verificación en DBeaver

The screenshot shows the DBeaver 23.1.0 interface. The left sidebar shows the database structure for the `fruverpm` schema, including tables like `clientes`, `compras`, and `productos`. The `clientes` table is selected and its data is displayed in the main grid. The data is highlighted with a red box:

	idCliente	nombres	correo	contrasena	rol
1	7	paola	paolaan2903@gmail.com	456	cliente
2	10	Rosita	Rosy@gmail.com	123	cliente
3	8	Admin	paolaan766@gmail.com	123	admin
4	9	ingrid	ingrid@gmail.com	456	cliente

## PostClientes

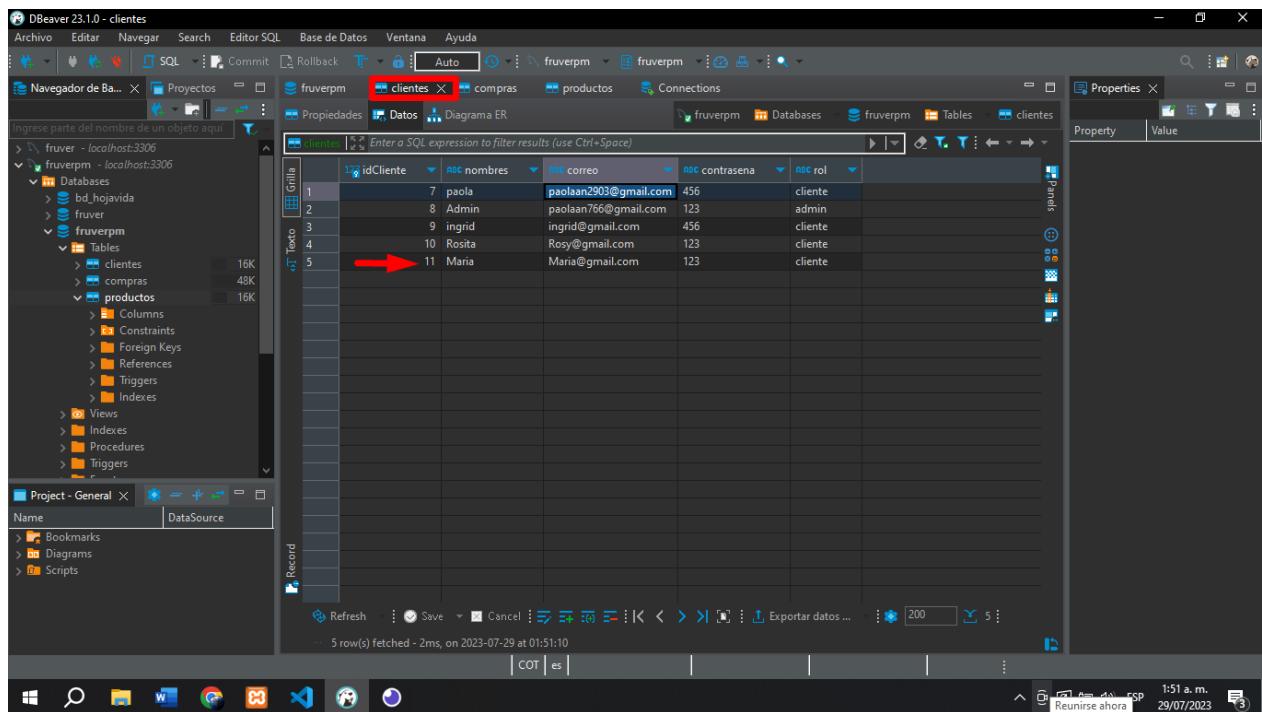
se prueba agregando la cliente María



The screenshot shows the Insomnia REST Client interface. The URL is set to `POST http://localhost:3000/clientes`. The response status is `200 OK` with a time of `267 ms` and a size of `80 B`. The response body is highlighted with a red box and contains the following JSON:

```
1 {  
2   "nombres": "Maria",  
3   "correo": "Maria@gmail.com",  
4   "contrasena": "123",  
5   "rol": "cliente"  
6 }
```

## Verificación en DBeaver

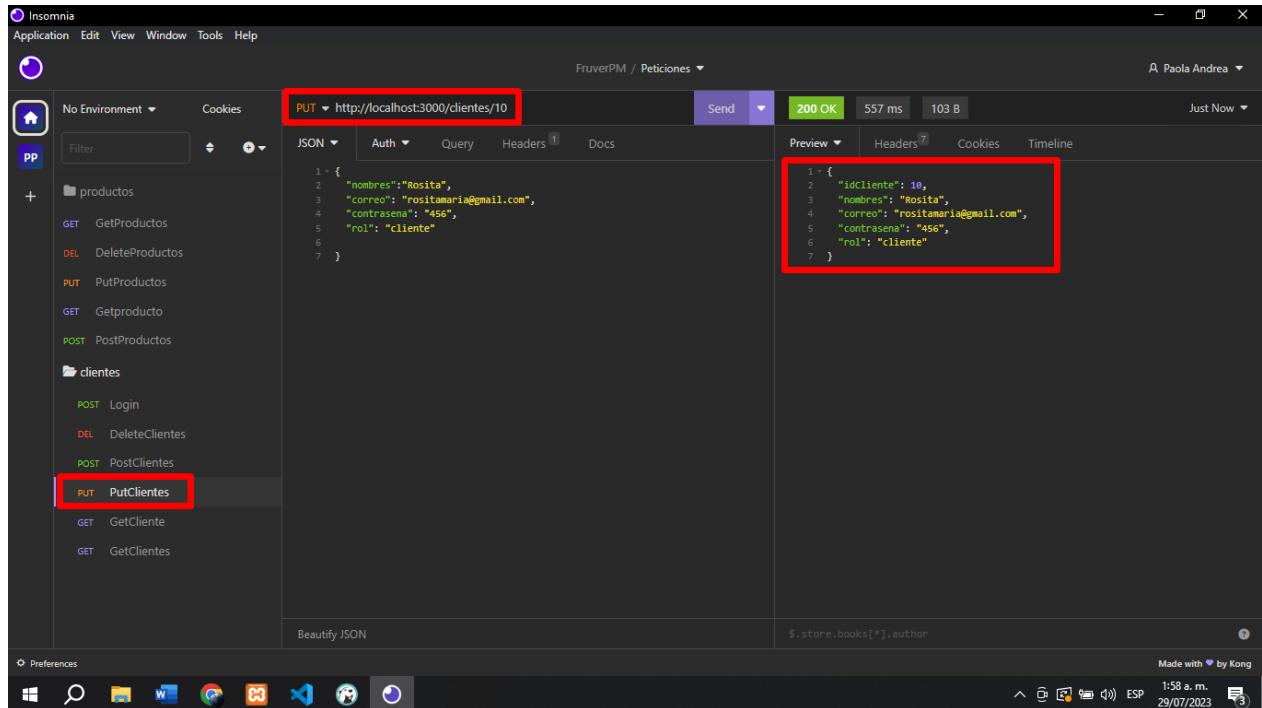


The screenshot shows the DBeaver interface connected to the `fruverpm` database. The `clients` table is selected. A red arrow points to the last row of the grid, which represents the newly added client `Maria`.

	idCliente	nombres	correo	contrasena	rol
1	7	paola	paoaan2503@gmail.com	456	cliente
2	8	Admin	paoaan766@gmail.com	123	admin
3	9	ingrid	ingrid@gmail.com	456	cliente
4	10	Rosita	Rosy@gmail.com	123	cliente
5	11	Maria	Maria@gmail.com	123	cliente

## PutClientes

Actualizamos el correo de la cliente Rosita y su contraseña

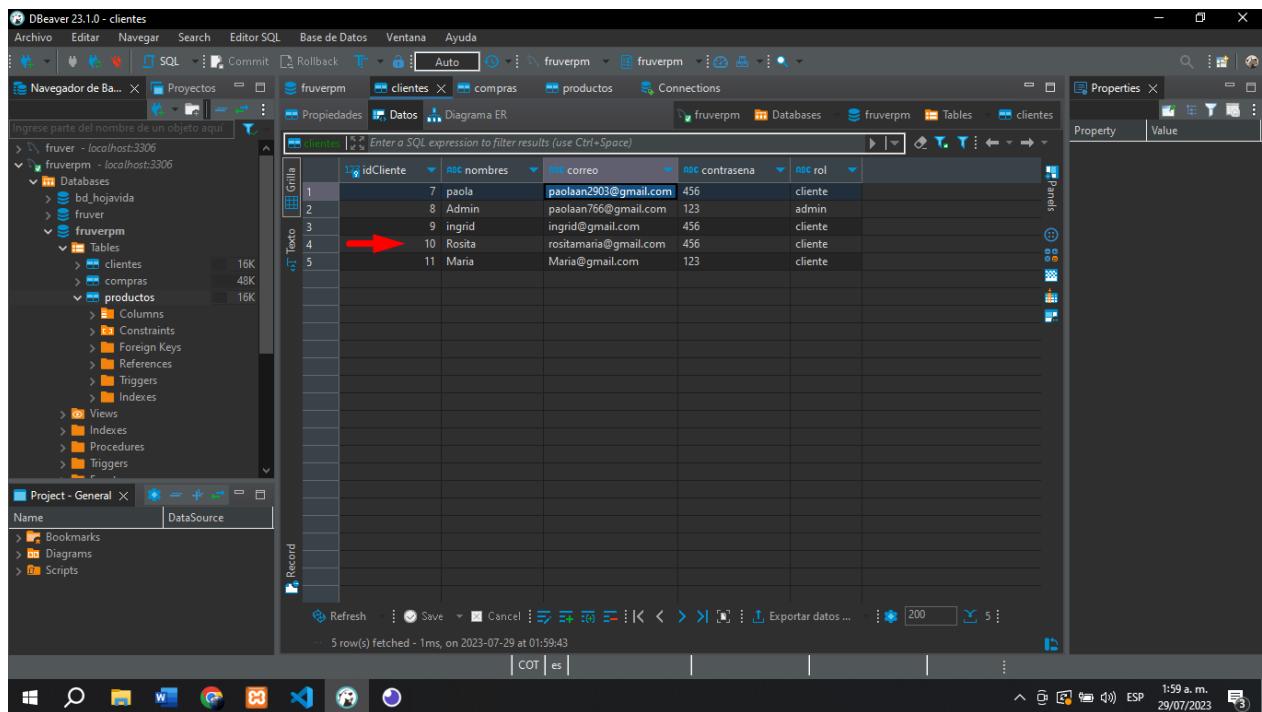


The screenshot shows the Insomnia REST Client interface. The left sidebar lists API endpoints under 'FruverPM / Peticiones'. The main area shows a 'PUT' request to `http://localhost:3000/clientes/10`. The 'JSON' tab displays the following payload:

```
1: {
2:   "idCliente": 10,
3:   "nombres": "Rosita",
4:   "correo": "rositamaria@gmail.com",
5:   "contrasena": "456",
6:   "rol": "cliente"
7: }
```

The response status is '200 OK' with a duration of '557 ms' and a size of '103 B'. The 'Preview' tab shows the updated JSON object with the new email address.

## Verificación en DBeaver

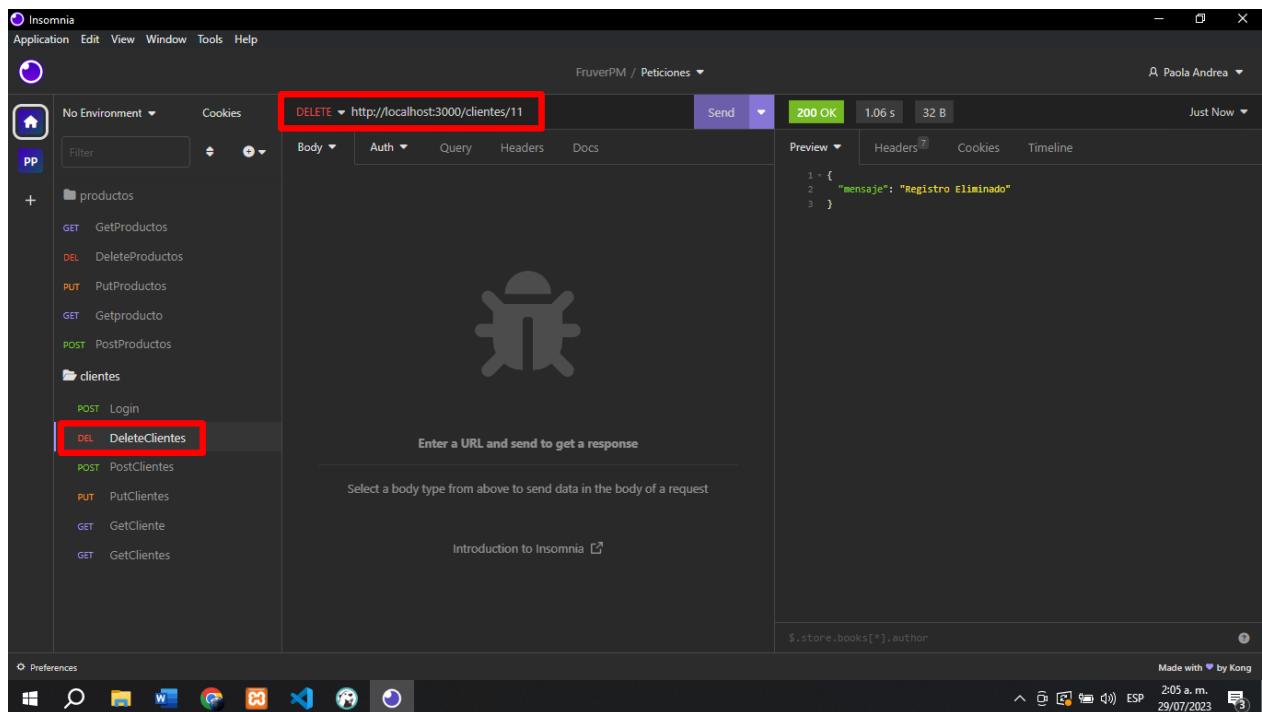


The screenshot shows the DBeaver 23.1.0 interface connected to the 'fruverpm' database. The left sidebar shows the database structure. The main area displays the 'clients' table in a grid view. An arrow points to the row where 'Rosita' is listed with the ID 10, showing the updated email address 'rositamaria@gmail.com'.

	idCliente	nombres	correo	contrasena	rol
1	7	paoala2503@gmail.com	456	cliente	
2	8	paoala766@gmail.com	123	admin	
3	9	ingrid	456	cliente	
4	10	Rosita	rositamaria@gmail.com	456	cliente
5	11	Maria	Maria@gmail.com	123	cliente

## DeleteClientes

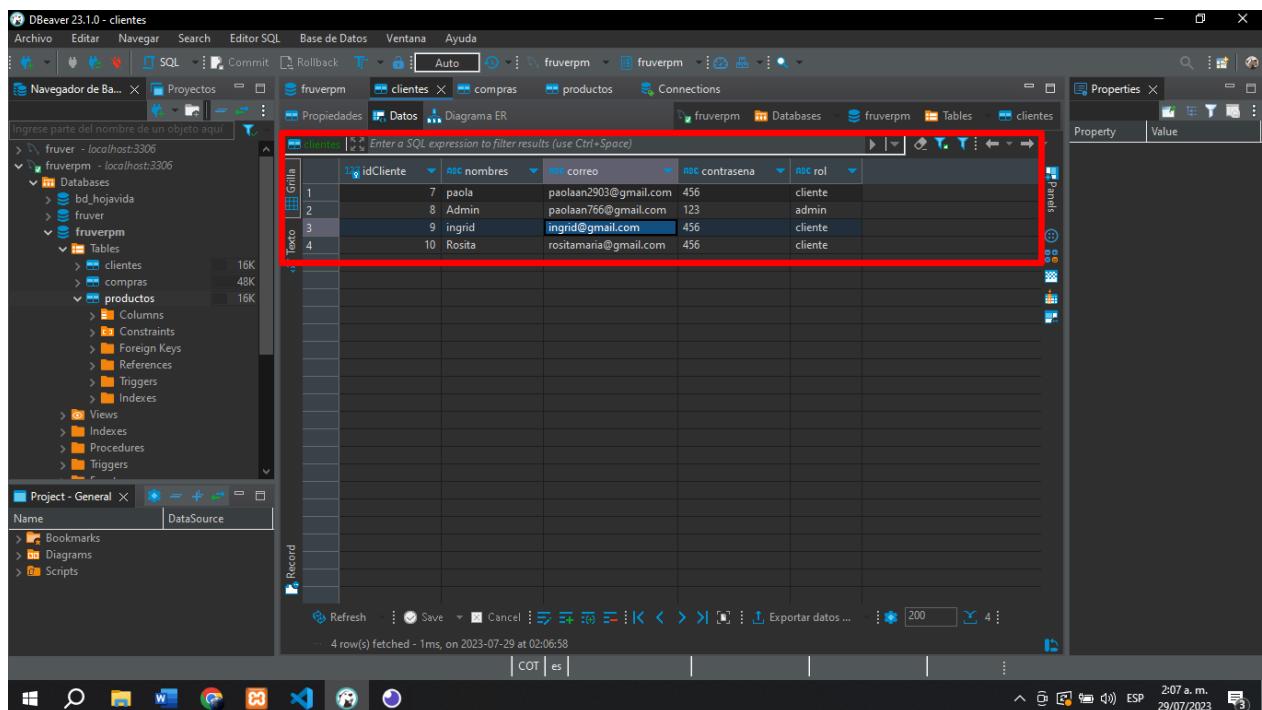
Se eliminará el cliente María con id: 11



The screenshot shows the Insomnia REST Client interface. In the left sidebar, under the 'clients' section, there is a 'DeleteClients' endpoint highlighted with a red box. The main area shows a DELETE request to 'http://localhost:3000/clientes/11'. The response status is '200 OK' with a response time of '1.06 s' and a size of '32 B'. The response body contains the JSON object: { "mensaje": "Registro Eliminado" }.

## Verificación en DBeaver

Podemos verificar que efectivamente se eliminó



The screenshot shows the DBeaver database interface. On the left, the 'Navegador de Bases de Datos' (Database Navigator) shows the 'fruverpm' database with its tables: 'clientes', 'compras', and 'productos'. The 'clientes' table is selected. The main panel displays the 'clientes' table data grid. A red box highlights the data grid. The columns are labeled: idCliente, nro\_nombres, nro\_correo, nro\_contraseña, and nro\_rol. The data shows 10 rows of client information. The last row, which corresponds to the deleted client, has the email address 'ingrid@gmail.com'.

idCliente	nro_nombres	nro_correo	nro_contraseña	nro_rol
1	7	paoala2903@gmail.com	456	cliente
2	8	paoala766@gmail.com	123	admin
3	9	ingrid@gmail.com	456	cliente
4	10	rositamaria@gmail.com	456	cliente

### 3.3 Login

Probamos la conexión de login con un correo y contraseña de la tabla clientes y verificamos que efectivamente se creo el token de autenticación.

The screenshot shows the Insomnia REST client interface. On the left sidebar, under the 'clientes' folder, there is a 'POST' method named 'Login'. This method is highlighted with a red box. In the main panel, a POST request is being made to 'http://localhost:3000/login'. The request body contains the following JSON:

```
1: {  
2:   "correo": "paolaan766@gmail.com",  
3:   "contraseña": 123  
4: }
```

The response status is '200 OK' with a response time of '20.5 ms' and a size of '183 B'. The response body is also highlighted with a red box and contains a JSON token:

```
1: {  
2:   "token":  
3:     "eyJhbGciOiJIUzI1NiJ9.eyJpZENgawVvdGUiojgsIm5vbWJyZXNhbG1pbisInVnJlb1yIG  
4:     InRho2xhv43NjZA2z2ihawwvY29tTlwiMc9sijojYwRtaW4ifQ.yfZtnpopzeZE.Jmzb1E8pCZ2vp.  
5:     Jy12amxyL42peqM"  
6: }
```

### 3.4 Compras

#### GetCompras

The screenshot shows the Insomnia REST client interface. On the left sidebar, under the 'Compras' folder, there is a 'GET' method named 'GetCompras'. This method is highlighted with a red box. In the main panel, a GET request is being made to 'http://localhost:3000/compras'. The response status is '200 OK' with a response time of '7.9 ms' and a size of '915 B'. The response body is highlighted with a red box and contains a JSON array of purchase items:

```
37:   {  
38:     "idCliente": 8,  
39:     "cantidad": 8,  
40:     "fechaSolicitud": "2023-07-26T19:33:03.000Z",  
41:     "estado": -1  
42:   },  
43:   {  
44:     "idCompra": 18,  
45:     "idProducto": 10,  
46:     "idCliente": 7,  
47:     "cantidad": 7,  
48:     "fechaSolicitud": "2023-07-26T19:36:30.000Z",  
49:     "estado": 1  
50:   },  
51:   {  
52:     "idCompra": 25,  
53:     "idProducto": 11,  
54:     "idCliente": 10,  
55:     "cantidad": 3,  
56:     "fechaSolicitud": "2023-07-29T17:00:20.000Z",  
57:     "estado": 1  
58:   },  
59:   {  
60:     "idCompra": 26,  
61:     "idProducto": 15,  
62:     "idCliente": 9,  
63:     "cantidad": 8,  
64:     "fechaSolicitud": "2023-07-29T05:49:19.000Z",  
65:     "estado": 0  
66:   }  
]
```

## Verificación en DBBeaver

The screenshot shows the DBBeaver interface with the 'compras' table selected in the central grid. The table has columns: idCompra, idProducto, idCliente, cantidad, fechaSolicitud, and estado. The data grid contains 10 rows of purchase records. A red box highlights the entire data grid.

	idCompra	idProducto	idCliente	cantidad	fechaSolicitud	estado
1	13	8	7	23	2023-07-22 11:14:02	1
2	25	11	10	3	2023-07-29 12:00:20	1
3	26	15	9	8	2023-07-29 00:49:19	0
4	14	7	7	2	2023-07-25 13:59:45	-1
5	15	8	7	2	2023-07-25 14:02:16	-1
6	16	11	7	2	2023-07-25 14:02:22	1
7	17	10	8	8	2023-07-26 14:33:03	-1
8	18	10	7	7	2023-07-26 14:36:30	1

## PostCompras

Para la prueba se agregará una compra

idCliente: 10 –Rosita

idProducto:16 -cebolla

cantidad: 5

The screenshot shows the Insomnia API client interface. A POST request is made to `http://localhost:3000/compras`. The request body is a JSON object with fields: idCompra, idProducto, idCliente, cantidad. The response status is 200 OK, and the response body is a JSON object with the same fields and values as the request body. A red box highlights the response body.

```
POST http://localhost:3000/compras
200 OK
223 ms | 59 B
Just Now
Preview Headers Cookies Timeline
1 = {
2   "idCompra": 30,
3   "idCliente": 10,
4   "idProducto": 16,
5   "cantidad": 5
6 }
```

## Verificación en DBBeaver

Podemos verificar que efectivamente se agrego la compra con idCompra: 30

The screenshot shows the DBBeaver interface with the 'compras' table selected. The table has columns: idCompra, idProducto, idCliente, cantidad, fechaSolicitud, and estado. A red arrow points to the last row, which contains the value 30 in the idCompra column.

	idCompra	idProducto	idCliente	cantidad	fechaSolicitud	estado
1	13	8	7	23	2023-07-22 11:14:02	1
2	14	7	7	2	2023-07-25 13:59:45	-1
3	15	8	7	2	2023-07-25 14:02:16	-1
4	16	11	7	2	2023-07-25 14:02:22	1
5	17	10	8	8	2023-07-26 14:33:03	-1
6	18	10	7	7	2023-07-26 14:36:30	1
7	25	11	10	3	2023-07-29 12:00:20	1
8	26	15	9	8	2023-07-29 02:49:19	0
9	30	16	10	5	2023-07-29 02:44:53	0

## PutCompras

Para la prueba se actualizará la compra con idCompra:25, se modificará la cantidad a 10

The screenshot shows the Insomnia API client. A red box highlights the URL 'PUT http://localhost:3000/compras/25'. The request body is a JSON object:

```
1 + {  
2     "idCompra": 25,  
3     "idProducto": 11,  
4     "idCliente": 10,  
5     "cantidad": 10  
6 }
```

The response shows a green '200 OK' status with '302 ms' and '104 B'.

## Verificación en DBBeaver

Podemos verificar que efectivamente se actualizo la compra.

	idCompra	idProducto	idCliente	cantidad	fechaSolicitud	estado
1	13	8	7	23	2023-07-22 11:14:02	1
2	14	7	7	2	2023-07-25 13:59:45	-1
3	15	8	7	2	2023-07-25 14:02:16	-1
4	16	11	7	2	2023-07-25 14:02:22	1
5	17	10	8	8	2023-07-26 14:33:03	-1
6	18	10	7	7	2023-07-26 14:36:30	1
7	25	11	10	10	2023-07-29 12:00:20	1
8	26	15	9	8	2023-07-29 02:49:19	0
9	30	16	10	5	2023-07-29 02:44:53	0

## 4. Subir el código y el informe a un repositorio remoto (GitHub)

### 4.1 Ingresamos a GitHub y Creamos un nuevo repositorio con el nombre **Taller2BackendFruvermania**

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \* / Repository name \*

PaulaAndrea20 / Taller2BackendFruvermania

Taller2BackendFruvermania is available.

Great repository names are short and memorable. Need inspiration? How about [bookish-umbrella](#) ?

Description (optional)

Backend Tienda de Frutas y Verduras

Public  
Anyone on the internet can see this repository. You choose who can commit.

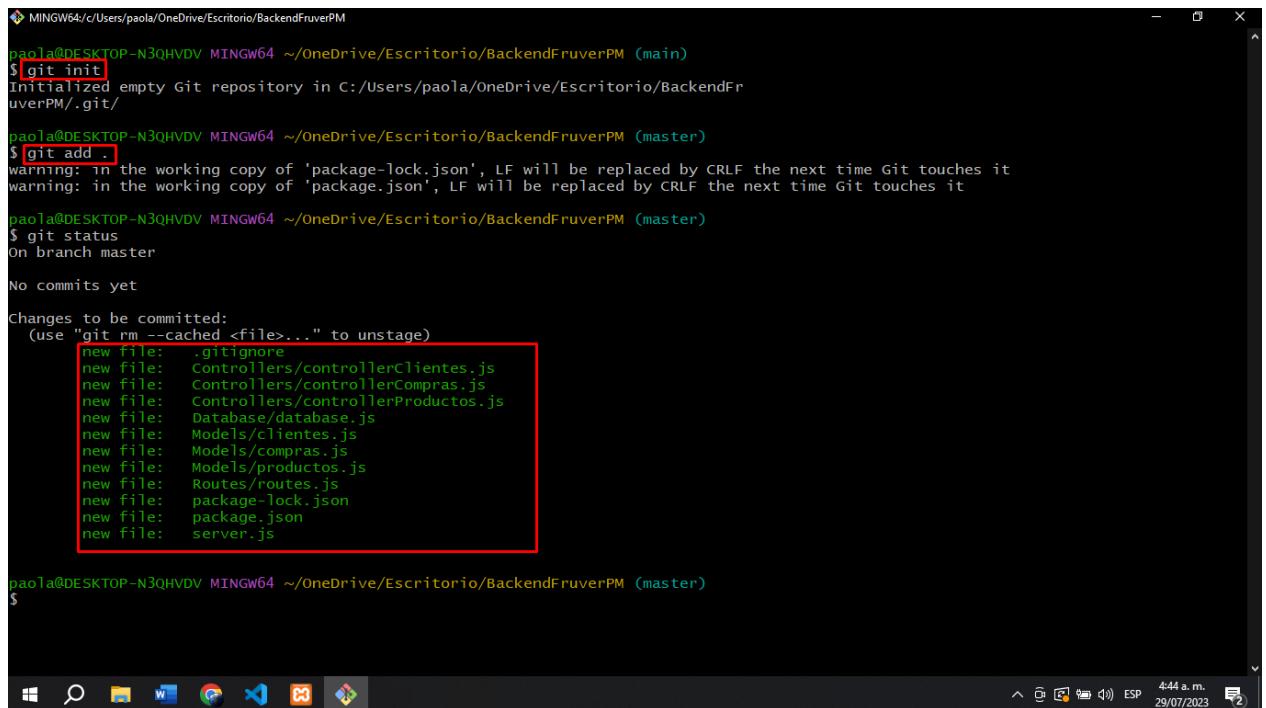
Private  
You choose who can see and commit to this repository.

Initialize this repository with:

Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

## 4.2 Inicializamos un repositorio local de nuestro directorio, agregamos todos los archivos y verificamos que efectivamente se hayan agregado.



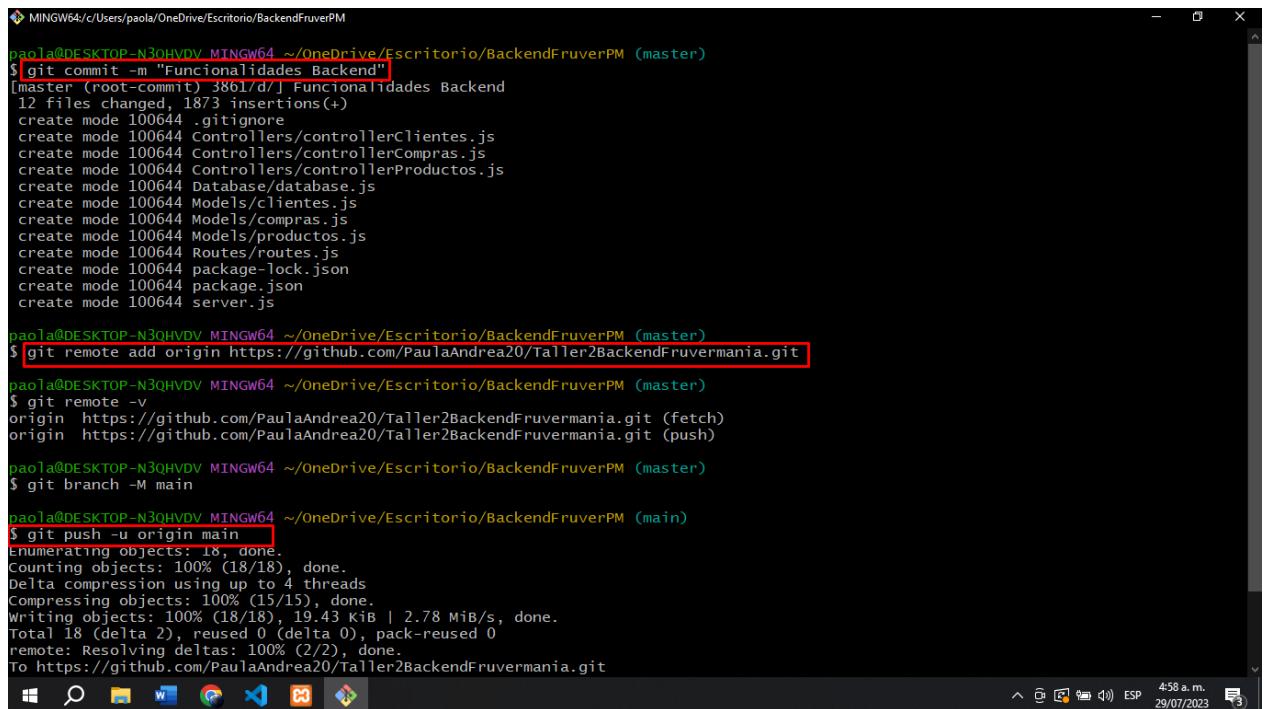
```
MINGW64:/c/Users/paola/OneDrive/Escritorio/BackendFruverPM
paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (main)
$ git init
Initialized empty Git repository in C:/Users/paola/OneDrive/Escritorio/BackendFr
uverPM/.git/
paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git add .
warning: in the working copy of 'package-lock.json', LF will be replaced by CRLF the next time Git touches it
warning: in the working copy of 'package.json', LF will be replaced by CRLF the next time Git touches it
paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file: .gitignore
    new file: Controllers/controllerClientes.js
    new file: Controllers/controllerCompras.js
    new file: Controllers/controllerProductos.js
    new file: Database/database.js
    new file: Models/clientes.js
    new file: Models/compras.js
    new file: Models/productos.js
    new file: Routes/routes.js
    new file: package-lock.json
    new file: package.json
    new file: server.js

paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$
```

Por último, realizamos un commit con el nombre **Funcionalidades Backend** para guardar los cambios en el repositorio, luego lo vinculamos con el repositorio remoto.



```
MINGW64:/c/Users/paola/OneDrive/Escritorio/BackendFruverPM
paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git commit -m "Funcionalidades Backend"
[master (root-commit) 3861/d] Funcionalidades Backend
12 files changed, 1873 insertions(+)
create mode 100644 .gitignore
create mode 100644 Controllers/controllerClientes.js
create mode 100644 Controllers/controllerCompras.js
create mode 100644 Controllers/controllerProductos.js
create mode 100644 Database/database.js
create mode 100644 Models/clientes.js
create mode 100644 Models/compras.js
create mode 100644 Models/productos.js
create mode 100644 Routes/routes.js
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 server.js

paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git remote add origin https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git
paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git remote -v
origin https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git (fetch)
origin https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git (push)

paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (master)
$ git branch -M main

paola@DESKTOP-N3QHVDV MINGW64 ~/OneDrive/Escritorio/BackendFruverPM (main)
$ git push -u origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 4 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (18/18), 19.43 KiB | 2.78 MiB/s, done.
Total 18 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git
  458 a.m.
  29/07/2023
```

Podemos verificar que efectivamente se configuro el repositorio remoto.

**Enlace de GIT:** <https://github.com/PaulaAndrea20/Taller2BackendFruvermania.git>

The screenshot shows a GitHub repository page for 'Taller2BackendFruvermania'. The repository has 1 branch and 0 tags. The main branch has 2 commits. The files listed include Controllers, Database, Models, Routes, .gitignore, README.md, package-lock.json, package.json, and server.js. All files were committed by 'PaulaAndrea20' and are labeled 'Funcionalidades Backend'. The commit times range from 21 minutes ago to 29 minutes ago. The repository has 0 stars, 1 watching, and 0 forks. It also includes sections for About, Releases, and Packages.

**About**

Backend Tienda de Frutas y Verduras

- Readme
- Activity
- 0 stars
- 1 watching
- 0 forks

**Releases**

No releases published

[Create a new release](#)

**Packages**

No packages published

[Publish your first package](#)