

DIPLOMADO DE ACTUALIZACIÓN EN NUEVAS TECNOLOGÍAS PARA EL
DESARROLLO DE
SOFTWARE

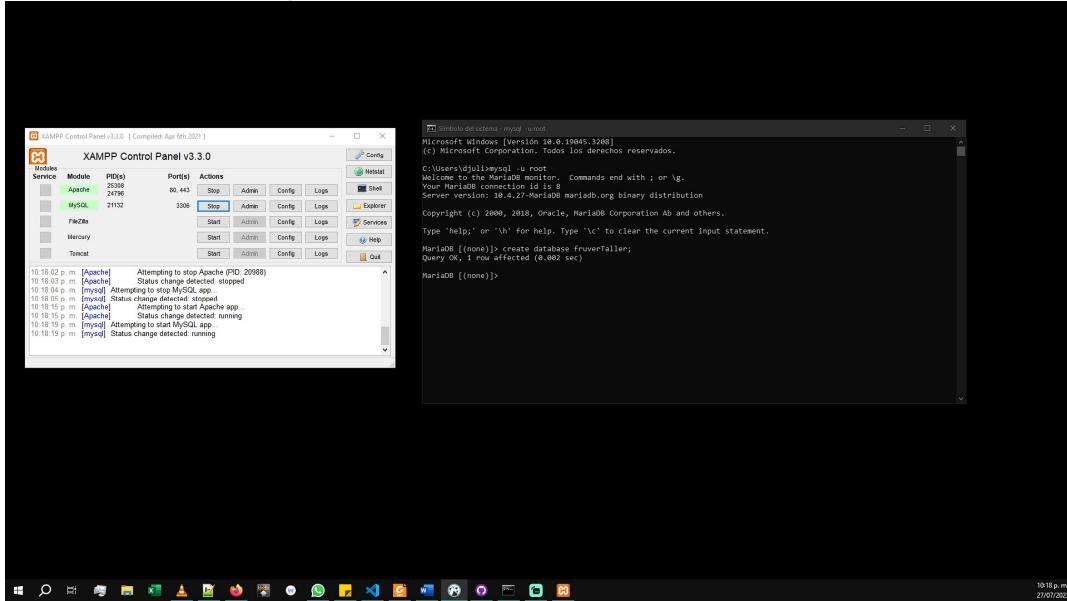
TALLER UNIDAD 2 BACKEND

DAVID JULIAN CRIOLLO GÓMEZ

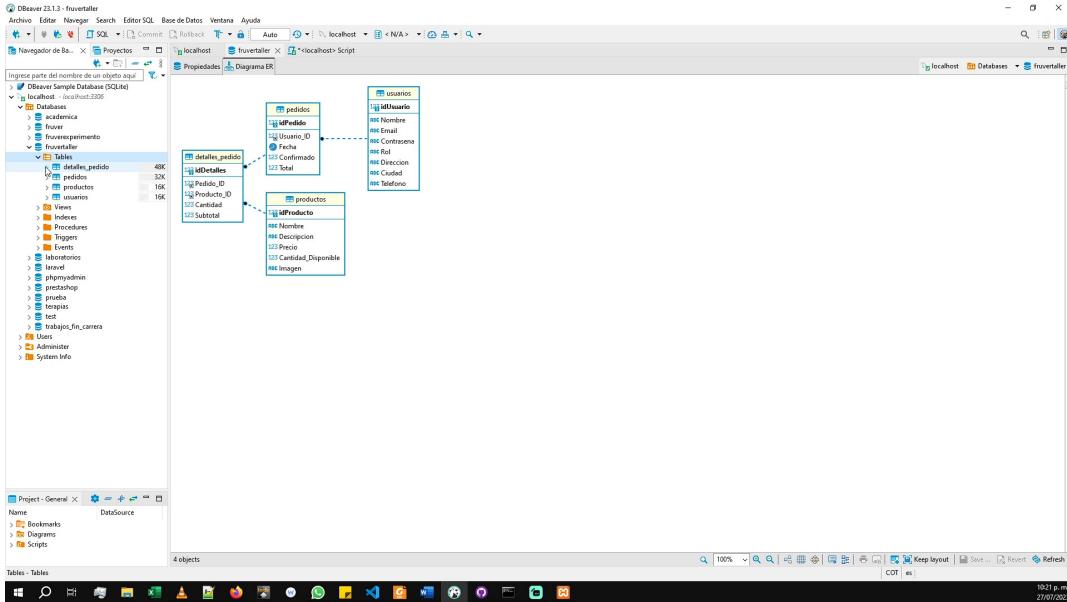
UNIVERSIDAD DE NARIÑO
INGENIERIA DE SISTEMAS
SAN JUAN DE PASTO, 2023

- 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.

Se crea la base de datos, en este caso se llamará fruvertaller



En DBeaver confirmamos que aparece la base de datos y creamos las tablas usuarios, pedidos, productos y detalles_pedido, en la imagen se observan como se ha configurada cada tabla y sus relaciones con la demás.



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

Configuramos el archivo package.json

```

{
  "name": "taller2_backend",
  "version": "1.0.0",
  "description": "Proyecto Fruver",
  "main": "server.js",
  "type": "module"
}

"scripts": {
  "test": "node node server.js"
},
"keywords": [],
"author": "",
"license": "ISC",
"dependencies": {
  "express": "^4.18.7",
  "mysql": "^2.18.1",
  "mysql2": "^3.5.2",
  "sequelize": "^6.32.1"
},
"devDependencies": {
  "nodemon": "^3.0.1"
}
}

```

The screenshot shows the Visual Studio Code interface with the package.json file open in the center editor. The left sidebar shows a tree view of the project structure with 'TALLER_BACKEND' expanded, containing 'Database', 'node_modules', and 'package-lock.json'. Below these are 'package.json' and 'server.js'. The bottom status bar shows the path 'E:\0_Programas\DIPLOMADO\MD\taller2_Backend' and the terminal command 'npm run start'.

Creamos el directorio Database y el archivo database.js para configurar la conexión con la base de datos “fruvertaller”

```

import { Sequelize } from 'sequelize';

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

export {
  sequelize
}

```

The screenshot shows the Visual Studio Code interface with the database.js file open in the center editor. The left sidebar shows the same project structure as the previous screenshot. The bottom status bar shows the path 'E:\0_Programas\DIPLOMADO\MD\taller2_Backend' and the terminal command 'npm run start'.

Creamos el directorio Routes y creamos el archivo routes.js para definir las rutas que usaran la funciones del backend

```

    Routes > routes.js
    1 import { Router } from 'express';
    2 import { getProductos, getProductosId, postProductos, putProductos, deleteProductos,
    3     getPedidos, getPedidosId, postPedidos, putPedidos, deletePedidos,
    4     getDetalles, getDetallesId, postDetalles, putDetalles, deleteDetalles,
    5     } from './Controllers/controller.js';
    6
    7 const router = Router();
    8
    9 // Ruta de la página principal (Ejemplo)
    10 router.get('/', (req, res) => {
    11     [ ... ] res.send("GET Pagina Principal Express");
    12 });
    13
    14
    15 // Rutas relacionadas con Productos
    16 router.get('/productos', getProductos);
    17 router.get('/productos/:idProducto', getProductosId);
    18 router.post('/productos', postProductos);
    19 router.put('/productos/:idProducto', putProductos);
    20 router.delete('/productos/:idProducto', deleteProductos);
    21
    22 // Rutas relacionadas con Usuarios
    23 router.get('/usuarios', getUsuarios);
    24 router.get('/usuarios/:idUsuario', getUsuariosId);
    25 router.post('/usuarios', postUsuarios);
    26 router.put('/usuarios/:idUsuario', putUsuarios);
    27 router.delete('/usuarios/:idUsuario', deleteUsuarios);
  
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

> taller2-backend@0.0.0 start
> node server.js
file:///E:/0_Programas/DIPLOMADO/M2/Taller2_Backend/server.js:12
app.use(router);
>
ReferenceError: router is not defined
at file:///E:/0_Programas/DIPLOMADO/M2/Taller2_Backend/server.js:12:9
at ModuleJob.run (node:internal/modules/esm/module_job:194:29)

Node.js v18.16.0
PS E:\0_Programas\DIPLOMADO\M2\Taller2_Backend

lin. 43, col. 23 Espacios: 4 UTF-8 CR LF ↵ Go Live ✅ Prettier 1036 p. m.
27/07/2023

Creamos el archivo server.js

```

    server.js
    1 import express from 'express';
    2 import router from './Routes/routes.js';
    3 import { Sequelize } from './Database/database.js';
    4 // Import ( Producto ) from './Models/productos.js';
    5
    6 //Crear Instancia
    7 const app = express();
    8
    9 //Montando el enrutador en la app principal
    10 app.use(express.json());
    11 app.use(router);
    12 app.set('port', 3000);
    13
    14 //Test a La Base de datos
    15 const testDB = async() => {
    16     try {
    17         await Sequelize.authenticate();
    18         console.log('Conexion realizada con éxito');
    19         //Correr servicio por el puerto 3000
    20         app.listen(app.get('port'), () => {
    21             [ ... ] console.log(`Servidor Escuchando por el puerto ${app.get('port')}`);
    22         });
    23     } catch (error) {
    24         console.error(`Error al realizar conexión ${error}`);
    25     }
    26 }
  
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

at file:///E:/0_Programas/DIPLOMADO/M2/Taller2_Backend/server.js:12:9
at ModuleJob.run (node:internal/modules/esm/module_job:194:29)

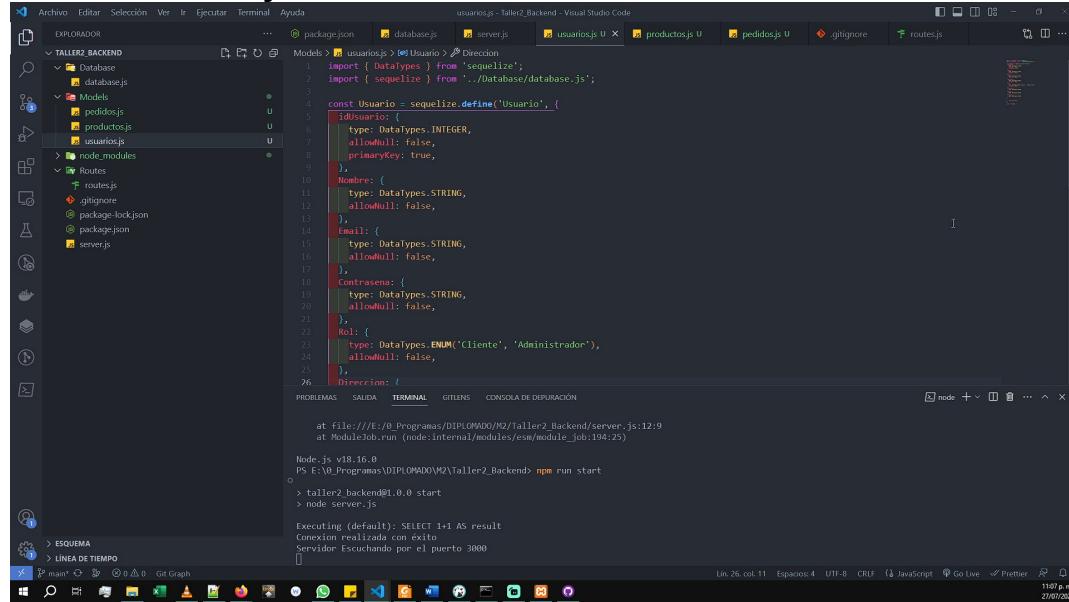
Node.js v18.16.0
PS E:\0_Programas\DIPLOMADO\M2\Taller2_Backend

> taller2-backend@0.0.0 start
> node server.js
Executing (default): SELECT 1+1 AS result
Conexion realizada con éxito
Servidor Escuchando por el puerto 3000

lin. 10, col. 25 Espacios: 4 UTF-8 CR LF ↵ Go Live ✅ Prettier 1637 p. m.
27/07/2023

Creamos los Modelos

Modelo usuarios.js



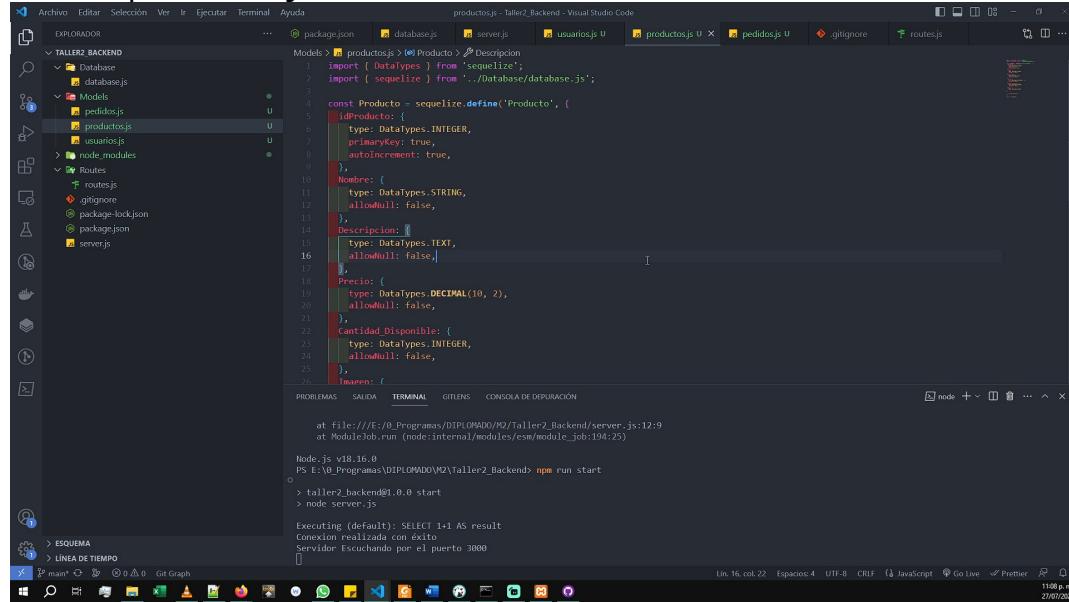
```
import { DataTypes } from 'sequelize';
import { sequelize } from '../Database';

const Usuario = sequelize.define('Usuario', {
  idUsuario: {
    type: DataTypes.INTEGER,
    allowNull: false,
    primaryKey: true,
  },
  Nombre: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  Email: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  Contrasena: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  Rol: {
    type: DataTypes.ENUM('Cliente', 'Administrador'),
    allowNull: false,
  },
}, {
  Direccion: {
    type: DataTypes.TEXT,
    allowNull: false,
  }
});

at file:///E:/0_Programas/DIPLOMADO/M2/taller2_Backend/server.js:12:9
at ModuleJob.run (node:internal/modules/esm/module_job:194:25)

Node.js v18.16.0
PS E:\0_Programas\DIPLOMADO\M2\taller2_Backend> npm run start
> taller2-backend@1.0.0 start
> node server.js
Executing (default): SELECT 1+1 AS result
Conexión realizada con éxito
Servidor Escuchando por el puerto 3000
[{"id": 1, "Nombre": "Juan", "Email": "juan@example.com", "Contrasena": "123456", "Rol": "Administrador", "Direccion": "Calle 123, Ciudad Centro"}, {"id": 2, "Nombre": "Ana", "Email": "ana@example.com", "Contrasena": "123456", "Rol": "Cliente", "Direccion": "Calle 456, Ciudad Centro"}]
```

Modelo productos.js



```
import { DataTypes } from 'sequelize';
import { sequelize } from '../Database';

const Producto = sequelize.define('Producto', {
  idProducto: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true,
  },
  Nombre: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  Descripcion: {
    type: DataTypes.TEXT,
    allowNull: false,
  },
  Precio: {
    type: DataTypes.DECIMAL(10, 2),
    allowNull: false,
  },
  Cantidad_Disponible: {
    type: DataTypes.INTEGER,
    allowNull: false,
  },
  Imagen: {
    type: DataTypes.TEXT,
    allowNull: false,
  }
});

at file:///E:/0_Programas/DIPLOMADO/M2/taller2_Backend/server.js:12:9
at ModuleJob.run (node:internal/modules/esm/module_job:194:25)

Node.js v18.16.0
PS E:\0_Programas\DIPLOMADO\M2\taller2_Backend> npm run start
> taller2-backend@1.0.0 start
> node server.js
Executing (default): SELECT 1+1 AS result
Conexión realizada con éxito
Servidor Escuchando por el puerto 3000
[{"id": 1, "Nombre": "Camisa", "Descripcion": "Camisa de algod\u00f3n", "Precio": 50.0, "Cantidad_Disponible": 100, "Imagen": "data:image/jpeg;base64,/9j/4AAQSkQ..."}, {"id": 2, "Nombre": "Pantal\u00f3n", "Descripcion": "Pantal\u00f3n de mezclilla", "Precio": 75.0, "Cantidad_Disponible": 50, "Imagen": "data:image/jpeg;base64,/9j/4AAQSkQ..."}, {"id": 3, "Nombre": "Zapato", "Descripcion": "Zapato deportivo", "Precio": 120.0, "Cantidad_Disponible": 30, "Imagen": "data:image/jpeg;base64,/9j/4AAQSkQ..."}]
```

Modelo pedidos.js

```

1 import ( DataTypes ) from 'sequelize';
2 import ( sequelize ) from './database/database.js';
3 import ( Usuario ) from '../Models/usuarios.js';
4
5 const Pedido = sequelize.define('Pedido', {
6   idPedido: {
7     type: DataTypes.INTEGER,
8     primaryKey: true,
9     autoIncrement: true,
10   },
11   Fecha: {
12     type: DataTypes.DATE,
13     allowNull: false,
14   },
15   Confirmado: {
16     type: DataTypes.BOOLEAN,
17     allowNull: false,
18     defaultValue: false,
19   },
20   Total: {
21     type: DataTypes.DECIMAL(10, 2),
22     allowNull: false,
23   },
24   {
25     timestamps: false,
26   });
27 });
28

```

ReferenceError: Pedido is not defined
at file:///E:/0_Programas/DIPLOMADO/M2/Taller2_Backend/Models/detalles_pedido.js:24:27
at ModuleJob.run (node:internal/modules/esm/module_job:194:25)

Modelo detalles_pedido.js

```

1 You have 9 segundos || autor (You)
2 import ( DataTypes ) from 'sequelize';
3 import ( sequelize ) from './Database/database.js';
4 import ( Producto ) from '../Models/productos.js';
5 import ( Pedido ) from '../Models/pedidos.js';
6
7 const Detalles_Pedido = sequelize.define('Detalles_Pedido', {
8   idDetalle: {
9     type: DataTypes.INTEGER,
10    primaryKey: true,
11    autoIncrement: true,
12  },
13  Cantidad: {
14    type: DataTypes.INTEGER,
15    allowNull: false,
16  },
17  Subtotal: {
18    type: DataTypes.DECIMAL(10, 2),
19    allowNull: false,
20  },
21  {
22    timestamps: false,
23  });
24
25 // Establecer la relación con Pedidos y Productos
26 Detalles_Pedido.belongsTo(Pedido, { foreignKey: 'Pedido_ID' });

```

Executing (default): SELECT `ID`, `Cantidad`, `Subtotal`, `Pedido_ID`, `Producto_ID` FROM `Detalles_Pedidos` AS 'Detalles_Pedido';
Executing (default): SELECT `ID`, `Cantidad`, `Subtotal`, `Pedido_ID`, `Producto_ID` FROM `Detalles_Pedidos` AS 'Detalles_Pedido';
Executing (default): SELECT `ID`, `Cantidad`, `Subtotal`, `Pedido_ID`, `Producto_ID` FROM `Detalles_Pedidos` AS 'Detalles_Pedido';
Executing (default): SELECT `ID`, `Cantidad`, `Subtotal`, `Pedido_ID`, `Producto_ID` FROM `Detalles_Pedidos` AS 'Detalles_Pedido';
Executing (default): SELECT `ID` AS '1' AS result
PS E:\0_Programas\DIPL0MADO\M2\Taller2_Backend> node .\server.js
Conexión realizada con éxito
Servidor Escuchando por el puerto 3000

Creamos el directorio Controllers y el archivo controllers.js

Funciones para producto

- get

The screenshot shows the Visual Studio Code interface with the file 'controller.js' open. The code defines two functions: 'getProductos' and 'getProductosId'. Both functions use await to call 'findall' or 'findByIdPk' from the 'Producto' model. Error handling is included, and the functions return JSON responses. The code is part of an export block.

```
1 import { Producto } from "../Models/productos.js";
2
3 //Cuando hago esto puedo modificar las rutas
4 const getProductos = async (req,res) =>{
5   try {
6     const productos=await Producto.findall();
7     res.status(200).json(productos);
8   } catch (error) {
9     res.status(400).json({mensaje: error});
10 }
11
12 const getProductosId = async (req,res) => {
13   const {idProducto} = req.params;
14   try {
15     const producto = await Producto.findByIdPk(idProducto);
16     res.status(200).json({producto});
17   } catch (error) {
18     res.status(400).json({ mensaje: error });
19   }
20 }
21
22 export {getProductos, getProductosId, postProductos, putProductos, deleteProductos}
```

- post

The screenshot shows the Visual Studio Code interface with the file 'controller.js' open. The code defines a 'postProductos' function that creates a new product using the 'create' method of the 'Producto' model. It takes 'Nombre', 'Descripcion', 'Precio', and 'Cantidad_Disponible' from the request body. Error handling is included, and the function returns a JSON response. The code is part of an export block.

```
17 const postProductos = async (req,res) => {
18   const { Nombre, Descripcion, Precio, Cantidad_Disponible } = req.body;
19   try {
20     const newProducto = await Producto.create([
21       Nombre,
22       Descripcion,
23       Precio,
24       Cantidad_Disponible,
25       Imagen,
26     ]);
27     res.status(200).json(newProducto);
28   } catch (error) {
29     res.status(400).json({ mensaje: error.message });
30   }
31 }
32
33 const putProductos = async (req,res) => {
34   const { idProducto } = req.params;
35   const { nombre,descripcion,precio,cantidad_disponible,imagen } = req.body;
36   try {
37     const oldProducto = await Producto.findByIdPk(idProducto);
38     oldProducto.nombre = nombre;
39     oldProducto.descripcion = descripcion;
40     oldProducto.precio = precio;
41     oldProducto.cantidad_disponible = cantidad_disponible;
42     oldProducto.imagen = imagen;
43     await oldProducto.save();
44     res.status(200).json(oldProducto);
45   } catch (error) {
46     res.status(400).json({ mensaje: error.message });
47   }
48 }
```

● put

The screenshot shows the Visual Studio Code interface with the 'controller.js' file open. The code implements a PUT endpoint to update a product. It first checks if the product exists. If it does, it updates its properties (Nombre, Descripcion, Precio, Cantidad_Disponible, Imagen) and saves the changes. If the product is not found, it returns a 404 error. Finally, it returns a 200 OK response with the updated product.

```
const putProductos = async (req, res) => {
  const { idProducto } = req.params;
  const { Nombre, Descripcion, Precio, Cantidad_Disponible, Imagen } = req.body;
  try {
    const oldProducto = await Producto.findByIdPk(idProducto);

    if (!oldProducto) {
      return res.status(404).json({ mensaje: "Producto no encontrado" });
    }

    oldProducto.Nombre = Nombre;
    oldProducto.Descripcion = Descripcion;
    oldProducto.Precio = Precio;
    oldProducto.Cantidad_Disponible = Cantidad_Disponible;
    oldProducto.Imagen = Imagen;

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

● delete

The screenshot shows the Visual Studio Code interface with the 'controller.js' file open. The code implements a DELETE endpoint to delete a product. It finds the product by ID and calls its 'destroy' method. If successful, it returns a 200 OK response with a message indicating the product was deleted. If there's an error or the product is not found, it returns a 400 Bad Request response with an error message.

```
const deleteProductos = async (req, res) => {
  try {
    const { idProducto } = req.params;
    const respuesta = await Producto.destroy({
      where: {
        id: idProducto,
      }
    });
    res.status(200).json({ mensaje: `Registro con id ${idProducto} Eliminado` });
  } catch (error) {
    res.status(400).json({ mensaje: `Registro No Eliminado ${error}` });
  }
}

export { getProductos, getProductosId, postProductos, putProductos, deleteProductos }
```

Funciones para usuario

- get

The screenshot shows the Visual Studio Code interface with the 'controller.js' file open. The code defines two asynchronous functions: `getUsuarios` and `getUsuarioId`. Both functions use the `await` keyword to call `Usuario.findAll()` and `Usuario.findById()` respectively, and then return the results as JSON objects. The `getProductos` function is also present at the bottom of the file.

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

const getUsuarioId = async (req,res) => {
    const idUsuario = req.params;
    try {
        const usuario = await Usuario.findById(idUsuario);
        res.status(200).json(usuario);
    } catch (error) {
        res.status(400).json({ mensaje: error });
    }
}

export [getProductos, getProductosId, postProductos, putProductos, deleteProductos, getUsuarios, getUsuarioId, postUsuarios, putUs
```

- post

The screenshot shows the Visual Studio Code interface with the 'controller.js' file open. The code defines a single asynchronous function `postUsuarios`. This function extracts data from the request body (`req.body`) and uses it to create a new user (`Usuario.create`). It then returns the created user as a JSON object. Error handling is included to manage cases where the creation fails or the request body is missing.

```
const postUsuarios = async (req, res) => {
    const ( idUsuario, Nombre, Email, Contraseña, Rol, Direccion, Ciudad, Telefono ) = req.body;
    try {
        const newUsuario = await Usuario.create({
            idUsuario,
            Nombre,
            Email,
            Contraseña,
            Rol,
            Direccion,
            Ciudad,
            Telefono,
        });
        res.status(200).json(newUsuario);
    } catch (error) {
        res.status(400).json({ mensaje: error.message });
    }
}

export [getProductos, getProductosId, postProductos, putProductos, deleteProductos, getUsuarios, getUsuarioId, postUsuarios, putUs
```

● put

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code implements a PUT endpoint to update a user's information. It first checks if the user exists by ID. If found, it updates the user's details (Nombre, Email, Contraseña, Rol, Dirección, Ciudad, Telefono) from the request body. If not found, it returns a 404 error. Finally, it saves the updated user to the database and returns a 200 status.

```
const putUsuarios = async (req, res) => {
  const { idUsuario } = req.params;
  const { Nombre, Email, Contraseña, Rol, Dirección, Ciudad, Telefono } = req.body;
  try {
    const oldUsuario = await Usuario.findById(idUsuario);

    if (!oldUsuario) {
      return res.status(404).json({ mensaje: "Usuario no encontrado" });
    }

    oldUsuario.Nombre = Nombre;
    oldUsuario.Email = Email;
    oldUsuario.Contraseña = Contraseña;
    oldUsuario.Rol = Rol;
    oldUsuario.Dirección = Dirección;
    oldUsuario.Ciudad = Ciudad;
    oldUsuario.Telefono = Telefono;

    You, ahora + Uncommitted changes
    const modUsuario = await oldUsuario.save();
    res.status(200).json(modUsuario);
  } catch (error) {
    res.status(400).json({ mensaje: error.message });
  }
}

PS E:\0_Programas\DIPLOMADO\02\Taller2_Backend> npm run start
> taller2_backend@1.0.0 start
> node server.js

Executing (default): SELECT 1+1 AS result
Conexion realizada con éxito
Servidor Escuchando por el puerto 3000
[...]
```

● delete

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code implements a DELETE endpoint to delete a user by ID. It first finds the user by ID. If found, it saves the user to the database and returns a 200 status. If not found, it returns a 400 error.

```
const deleteUsuarios = async (req, res) => {
  try {
    const idUsuario = req.params;
    const respuesta = await Usuario.destroy({
      where: {
        idUsuario: idUsuario,
      }
    });

    res.status(200).json({ mensaje: 'Registro con id $idUsuario Eliminado' });
  } catch (error) {
    res.status(400).json({ mensaje: 'Registro No Eliminado $error' });
  }
}

PS E:\0_Programas\DIPLOMADO\02\Taller2_Backend> node .\server.js
Desea terminar el trabajo por lotes (S/N)? s
Executing (default): SELECT 1+1 AS result
Conexion realizada con éxito
Servidor Escuchando por el puerto 3000
PS E:\0_Programas\DIPLOMADO\02\Taller2_Backend> node .\server.js
Executing (default): SELECT 1+1 AS result
Conexion realizada con éxito
Servidor Escuchando por el puerto 3000
[...]
```

Funciones para pedido

- get

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code defines two asynchronous functions: `getPedidos` and `getPedidosId`. Both functions use `await Pedido.findAll()` or `await Pedido.findByPk(idPedido)` to retrieve data from the database. The `getPedidos` function returns a JSON response with status 200, while `getPedidosId` returns a JSON response with status 400 if an error occurs. The `export` statement at the bottom includes all other functions from the file.

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

const getPedidosId = async (req, res) => {
    const {idPedido} = req.params;
    try {
        const pedido = await Pedido.findByPk(idPedido);
        res.status(200).json(pedido);
    } catch (error) {
        res.status(400).json({mensaje: error});
    }
}

export {getProductos, getProductosId, postProductos, putProductos, deleteProductos,
        getUsuarios, getUserId, postUsuarios, putUsuarios, deleteUsuarios,
        getPedidos, getPedidosId, postPedidos, putPedidos, deletePedidos}
```

- post

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code defines two asynchronous functions: `postPedidos` and `postPedidos`. The first function retrieves a pedido by ID. The second function creates a new pedido using `Pedido.create()` with fields `Usuario_ID`, `Fecha`, `Confirmado`, and `Total`. It returns a JSON response with status 200 if successful or 400 if an error occurs. The `export` statement at the bottom includes all other functions from the file.

```
const pedido = await Pedido.findByPk(idPedido);
res.status(200).json(pedido);
} catch (error) {
    res.status(400).json({mensaje: error});
}

const postPedidos = async (req, res) => {
    const { Usuario_ID, Fecha, Confirmado, Total } = req.body;
    try {
        const newPedido = await Pedido.create({
            Usuario_ID,
            Fecha,
            Confirmado,
            Total,
        });
        res.status(200).json(newPedido);
    } catch (error) {
        res.status(400).json({mensaje: error.message}); // You, hace 1 segundo * Uncommitted changes
    }
}

export {getProductos, getProductosId, postProductos, putProductos, deleteProductos,
        getUsuarios, getUserId, postUsuarios, putUsuarios, deleteUsuarios,
        getPedidos, getPedidosId, postPedidos, putPedidos, deletePedidos}
```

● put

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code implements the PUT method to update a pedido. It first checks if the pedido exists, then updates its properties (Fecha, Confirmado, Total) from the request body, saves the changes, and returns a success response. If an error occurs during the update, it returns a 400 status with an error message.

```
const putPedidos = async (req, res) => {
  const { idPedido } = req.params;
  const { Usuario_ID, Fecha, Confirmado, Total } = req.body;
  try {
    const oldPedido = await Pedido.findByPk(idPedido);

    if (!oldPedido) {
      return res.status(404).json({ mensaje: "Pedido no encontrado" });
    }

    oldPedido.Usuario_ID = Usuario_ID;
    oldPedido.Fecha = Fecha;
    oldPedido.Confirmado = Confirmado;
    oldPedido.Total = Total;

    const modPedido = await oldPedido.save();
    res.status(200).json(modPedido);
  } catch (error) {
    res.status(400).json({ mensaje: error.message });
  }
}

const deletePedidos = async (req, res) =>
```

● delete

The screenshot shows the Visual Studio Code interface with the controller.js file open. The code implements the DELETE method to delete a pedido. It finds the pedido by its ID, sends a destroy() query to remove it from the database, and returns a success response. If an error occurs during the deletion, it returns a 400 status with an error message.

```
const putPedidos = async (req, res) => {
  const { idPedido } = req.params;
  const modPedido = await oldPedido.save();
  res.status(200).json(modPedido);
}

const deletePedidos = async (req, res) =>
```

Funciones para detalle

- get

The screenshot shows the code editor in VS Code with the file `detalle_pedido.js` open. The code implements several functions:

- `getDetallesP`: Asynchronous function that retrieves all details from the database and returns them in JSON format.
- `getDetalleId`: Asynchronous function that retrieves a detail by its ID and returns it in JSON format.
- `postDetallesP`: Asynchronous function that creates a new detail in the database based on the provided body.

The code uses `async/await` and `try/catch` blocks to handle database operations and errors. It also includes validation logic for the input body.

- post

The screenshot shows the code editor in VS Code with the file `detalle_pedido.js` open. The code implements three functions:

- `getDetalleId`: Asynchronous function that retrieves a detail by its ID and returns it in JSON format.
- `postDetallesP`: Asynchronous function that creates a new detail in the database based on the provided body. This function is identical to the one in the previous screenshot.
- `putDetallesP`: Asynchronous function that updates an existing detail in the database based on its ID and the provided body.

The code uses `async/await` and `try/catch` blocks to handle database operations and errors. It includes validation logic for the input body.

● put

The screenshot shows the Visual Studio Code interface with the file `controller.js` open. The code implements a `putDetallesP` method to update details of an existing order. It first finds the old details by ID, checks if it exists, and then updates it with new values from the request body. If successful, it returns a 200 status with the updated details; otherwise, it returns a 400 status with an error message.

```
const putDetallesP = async (req, res) => {
  const { idDetalles } = req.params;
  const { Pedido_ID, Producto_ID, Cantidad, Subtotal } = req.body;
  try {
    const oldDetalles = await Detalles_Pedido.findByPk(idDetalles);

    if (!oldDetalles) {
      return res.status(404).json({ mensaje: "Detalles de Pedido no encontrado" });
    }

    oldDetalles.Pedido_ID = Pedido_ID;
    oldDetalles.Producto_ID = Producto_ID;
    oldDetalles.Cantidad = Cantidad;
    oldDetalles.Subtotal = Subtotal;

    const modDetalles = await oldDetalles.save();
    res.status(200).json(modDetalles);
  } catch (error) {
    res.status(400).json({ mensaje: error.message });
  }
};

const deleteDetallesP = async (req, res) => {
  try {
    const { idProducto } = req.params;
    const oldDetalles = await Detalles_Pedido.findByPk(idProducto);
    if (!oldDetalles) {
      return res.status(404).json({ mensaje: "Detalle de Pedido no encontrado" });
    }
    await oldDetalles.destroy();
    res.status(200).json({ mensaje: "Detalle de Pedido eliminado" });
  } catch (error) {
    res.status(400).json({ mensaje: "Error al eliminar el detalle de Pedido" });
  }
};
```

● delete

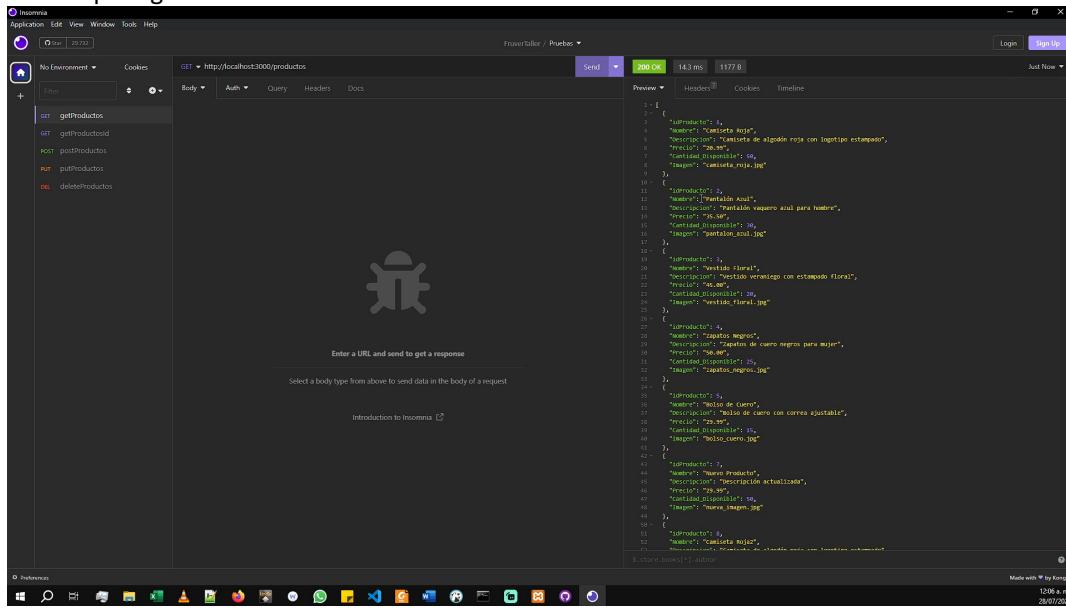
The screenshot shows the Visual Studio Code interface with the file `controller.js` open. The code implements a `deleteDetallesP` method to delete a specific detail from an order. It finds the detail by ID and then uses the `destroy` method to remove it from the database. It returns a 200 status with a success message or a 400 status with an error message if the detail is not found or an error occurs during deletion.

```
const deleteDetallesP = async (req, res) => {
  try {
    const { idDetalles } = req.params;
    const respuesta = await Detalles_Pedido.destroy({
      where: {
        idDetalles: idDetalles,
      }
    });
    if (!respuesta) {
      res.status(404).json({ mensaje: "Registro con id ${idDetalles} Eliminado" });
    } else {
      res.status(200).json({ mensaje: "Registro con id ${idDetalles} Eliminado" });
    }
  } catch (error) {
    res.status(400).json({ mensaje: "Registro No Eliminado ${error}" });
  }
};

export [getProductos, getProductosId, postProductos, putProductos, deleteProductos,
  getUsuarios, getUsuariosId, postUsuarios, putUsuarios, deleteUsuarios,
  getPedidos, getPedidosId, postPedidos, putPedidos, deletePedidos,
  getDetallesP, getDetallesId, postDetallesP, putDetallesP, deleteDetallesP]
```

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

Prueba para getProductos

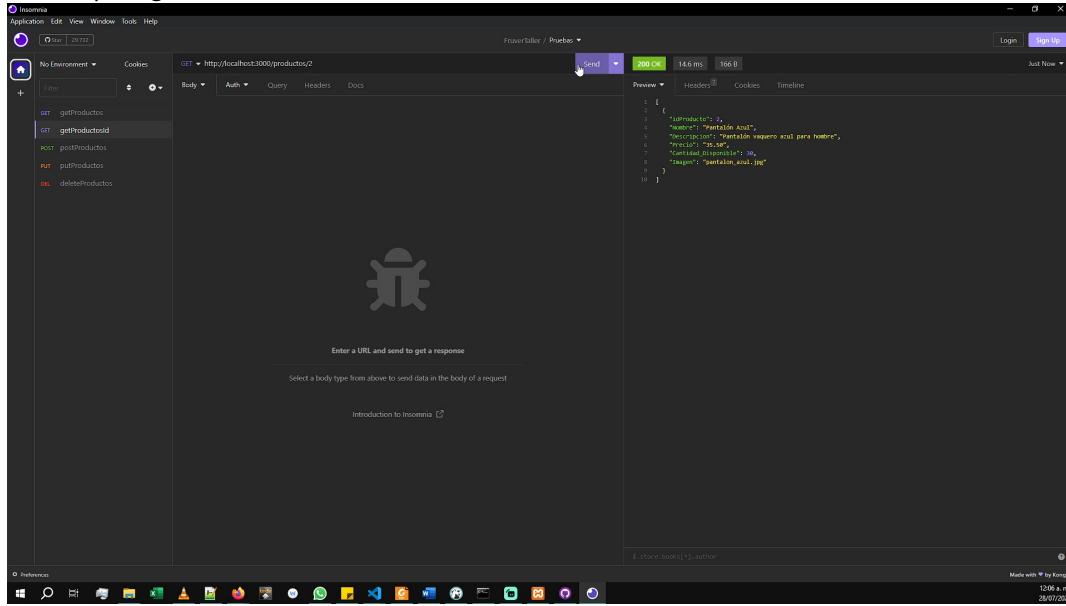


```

[{"id": 1, "name": "Camiseta azul", "description": "Camiseta azul de algod\u00f3n con logotipo estampado", "price": 25.99, "cantidad_disponible": 10, "image": "camiseta_azul.jpg"}, {"id": 2, "name": "Pantal\u00f3n azul", "description": "Pantal\u00f3n vaquero azul para hombre", "price": 45.99, "cantidad_disponible": 10, "image": "pantalon_azul.jpg"}, {"id": 3, "name": "Camiseta floral", "description": "Camiseta floral con estampado floral", "price": 22.99, "cantidad_disponible": 10, "image": "camiseta_floral.jpg"}, {"id": 4, "name": "Zapatos negros", "description": "Zapatos de cuero negros para mujer", "price": 35.99, "cantidad_disponible": 10, "image": "zapato_negro.jpg"}, {"id": 5, "name": "Bolso de cuero", "description": "Bolso de mano de cuero con correas ajustables", "price": 28.99, "cantidad_disponible": 10, "image": "bolso_cuero.jpg"}, {"id": 6, "name": "Bufanda blanca", "description": "Bufanda de algod\u00f3n blanca de color blanco", "price": 12.99, "cantidad_disponible": 10, "image": "bufanda_blanca.jpg"}, {"id": 7, "name": "Pantal\u00f3n rosado", "description": "Pantal\u00f3n vaquero rosa para hombre", "price": 30.99, "cantidad_disponible": 10, "image": "pantalon_rosado.jpg"}, {"id": 8, "name": "Camiseta rosa", "description": "Camiseta rosa de algod\u00f3n con logotipo estampado", "price": 20.99, "cantidad_disponible": 10, "image": "camiseta_rosa.jpg"}]

```

Prueba para getProductosId

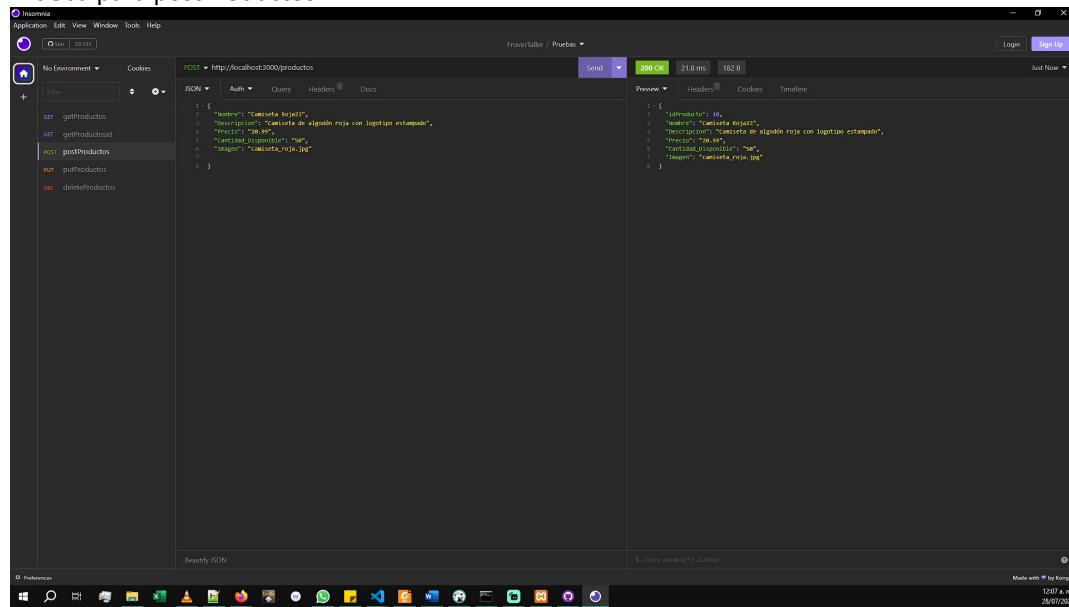


```

{
  "id": 2,
  "name": "Pantal\u00f3n azul",
  "description": "Pantal\u00f3n vaquero azul para hombre",
  "price": 25.99,
  "cantidad_disponible": 10,
  "image": "pantalon_azul.jpg"
}

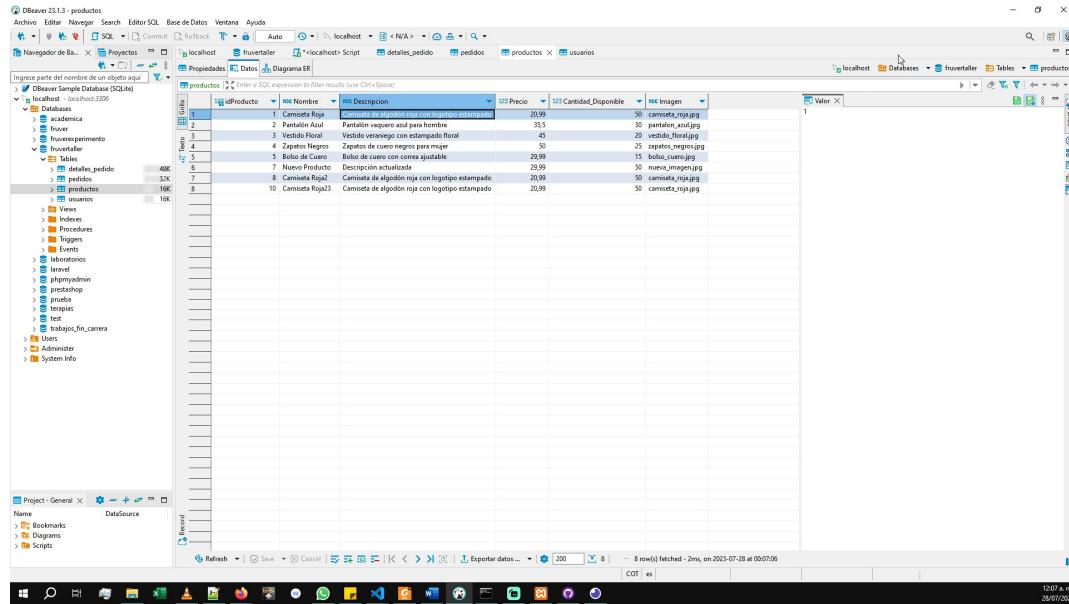
```

Prueba para postProductos



```
POST http://localhost:3000/productos
{
  "Nombre": "Camiseta Roja03",
  "Descripción": "Camiseta de algodón roja con logotipo estampado",
  "Precio": "20,99",
  "Cantidad_Disponible": "50",
  "Imagen": "camiseta_roja.jpg"
}
```

Se observa cambios en la base de datos



#	idProducto	Nombre	Descripción	Precio	Cantidad_Disponible	Imagen
1	1	Camiseta Roja	Camiseta de algodón roja con logotipo estampado	20,99	50	camiseta_roja.jpg
2	2	Pantalon Azul	Pantalon vaquero azul para hombre	31,5	30	pantalon_azul.jpg
3	3	Pantalon Rojo	Vestido de algodón rojo con estampado floral	45	20	vestido_floral.jpg
4	4	Tazón de Cereales	Tazón de cereales para desayuno	5,99	15	tazon_cereales.jpg
5	5	Bolso de Cuero	Bolso de cuero con correa ajustable	29,99	15	bolso_cuero.jpg
6	6	Nuevo Producto	Descripción actualizada	29,99	50	nueva_imagen.jpg
7	7					
8	8	Camiseta Roja2	Camiseta de algodón roja con logotipo estampado	20,99	50	camiseta_roja.jpg
9	9	Camiseta Roja3	Camiseta de algodón roja con logotipo estampado	20,99	50	camiseta_roja.jpg
10	10	Camiseta Roja3	Camiseta de algodón roja con logotipo estampado	20,99	50	camiseta_roja.jpg

Prueba para putProductos

The screenshot shows the Insomnia REST Client interface. A PUT request is made to `http://localhost:3000/products/10`. The response status is `200 OK` with a response time of `29.7 ms` and a size of `152.0`. The response body is a JSON object:

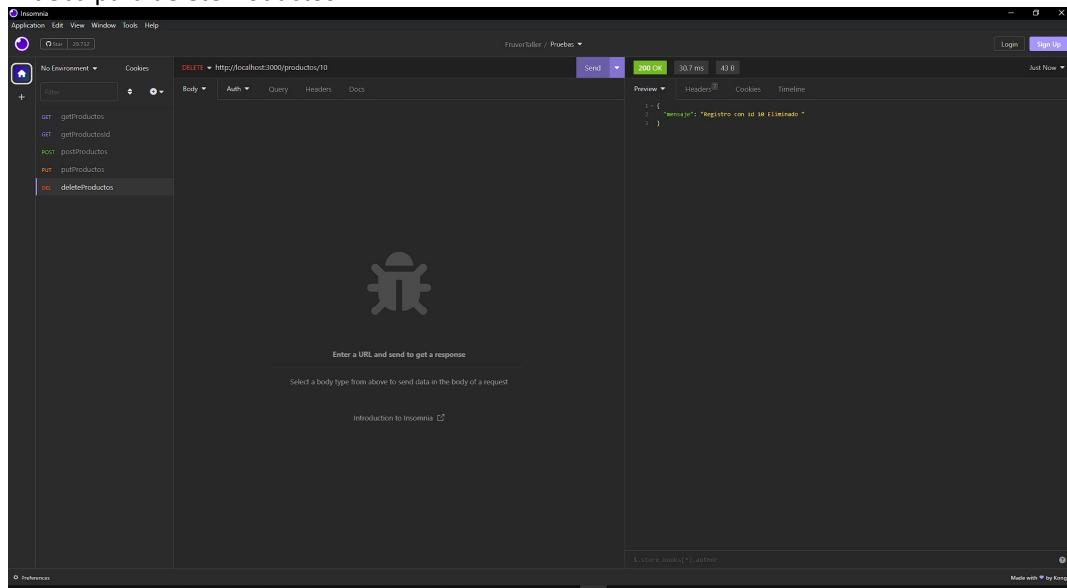
```
[{"id": 10, "Nombre": "Nuevo Producto", "Descripción": "Descripción actualizada", "Precio": 29.99, "Cantidad_Disponible": 50, "Imagen": "nueva_imagen.jpg"}]
```

Se observa cambios en la base de datos

The screenshot shows the MySQL Workbench interface. The left sidebar shows the database structure for the `storebook` database, including tables like `author`, `books`, `pedidos`, and `users`. The main area displays the `products` table. The table has columns: `idProducto`, `Nombre`, `Descripcion`, `Precio`, `Cantidad_Disponible`, and `Imagen`. The data shows several products, including one with the name "Nuevo Producto" and the image "nueva_imagen.jpg".

idProducto	Nombre	Descripcion	Precio	Cantidad_Disponible	Imagen
1	Camiseta Roja	Camiseta de algodón roja con estampado floral	20,99	50	camiseta_roja.jpg
2	Pantalon Azul	Pantalon vaquero azul para hombre	35,5	30	pantalon_azul.jpg
3	Pantalon Rosado	Vestido de fiesta rosado con estampado floral	45	20	vestido_rosado.jpg
4	Taza de Caffe	Taza de cerámica blanca para café	5,99	15	taza_caffe.jpg
5	Bolso de Cuero	Bolso de cuero con correas ajustables	29,99	15	bolso_cuero.jpg
7	Nuevo Producto	Descripción actualizada	29,99	50	nueva_imagen.jpg
8	Camiseta Roja2	Camiseta de algodón roja con logotipo estampado	20,99	50	camiseta_roja.jpg
10	Nuevo Producto	Descripción actualizada	29,99	50	nueva_imagen.jpg

Prueba para deleteProductos



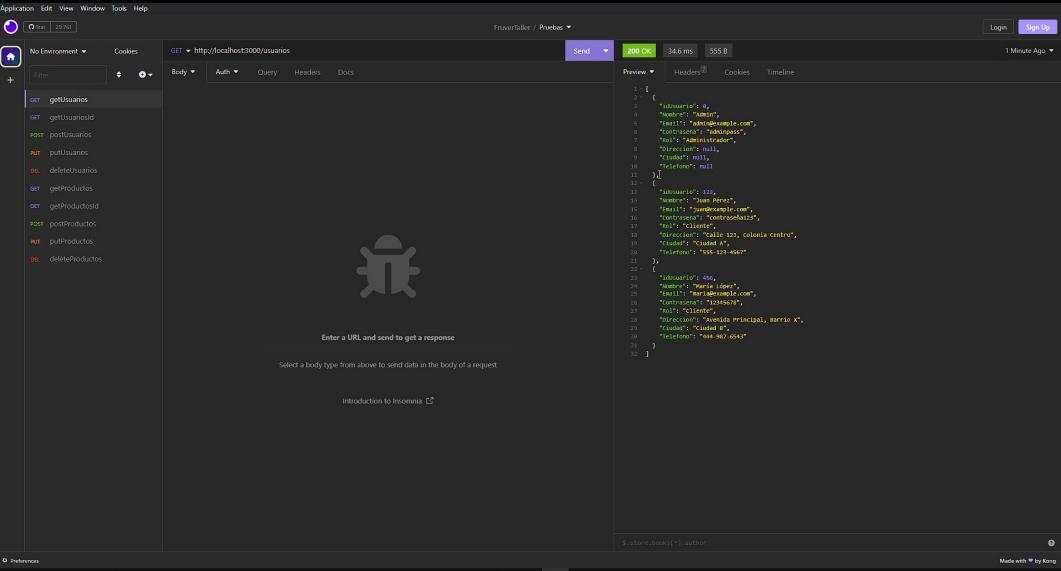
The screenshot shows the Insomnia REST Client interface. A DELETE request is made to `http://localhost:3000/products/10`. The response status is `200 OK`, time is `30.7 ms`, and size is `43 B`. The response body contains the following JSON:

```
{ "mensaje": "Registro con id 10 Eliminado" }
```

Below the client, the MySQL Workbench interface is shown. It displays the `products` table with 8 rows of data. The columns are `idProducto`, `Nombre`, `Descripcion`, `Precio`, `Cantidad_Disponible`, and `Imagen`. The data is as follows:

idProducto	Nombre	Descripcion	Precio	Cantidad_Disponible	Imagen
1	Camiseta Roja	Camiseta de algodón roja con logo estampado	20,99	50	camiseta_roja.jpg
2	Pantalon Azul	Pantalon vaquero azul para hombre	31,5	30	pantalon_azul.jpg
3	Pantalon Rosado	Vestido de fiesta de color rosado con estampado floral	45	20	vestido_rosado.jpg
4	Tacones Altos	Tacones de cuero altos para mujer	50	15	tacones_altos.jpg
5	Bolso de Cuero	Bolso de cuero con correa ajustable	29,99	15	bolso_cuero.jpg
7	Nuevo Producto	Descripción actualizada	29,99	50	nueva_imagen.jpg
8	Camiseta Roja2	Camiseta de algodón roja con logo estampado	20,99	50	camiseta_roja.jpg

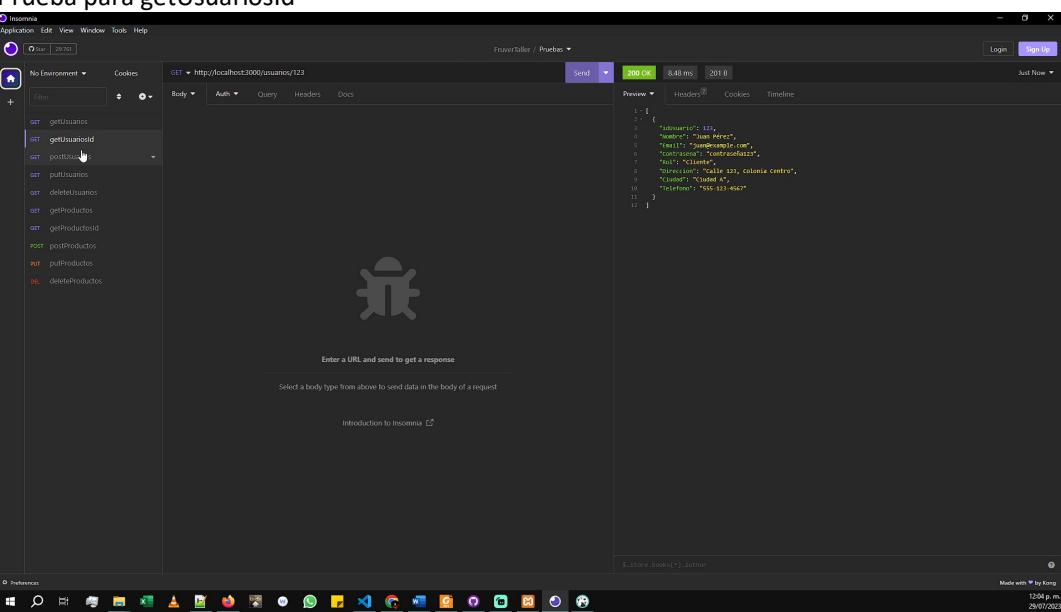
Prueba para getUsuarios



The screenshot shows the Insomnia REST Client interface. The URL is `http://localhost:3000/usuarios`. The response status is `200 OK` with a `34.6 ms` latency and `555 B` body size. The response was received `1 Minute Ago`. The response body is a JSON array:

```
[{"id": 1, "name": "Juan Perez", "email": "juan@example.com", "telefono": "555-1234567", "rol": "Administrador", "dirección": null, "cliente": null}, {"id": 2, "name": "Maria Hernandez", "email": "maria@example.com", "telefono": "555-2345678", "rol": "Cliente", "dirección": "Calle 123, colonia centro", "cliente": {"calle": "Calle 123", "colonia": "Centro", "numero": 1234, "telefono": "555-321-4567"}]
```

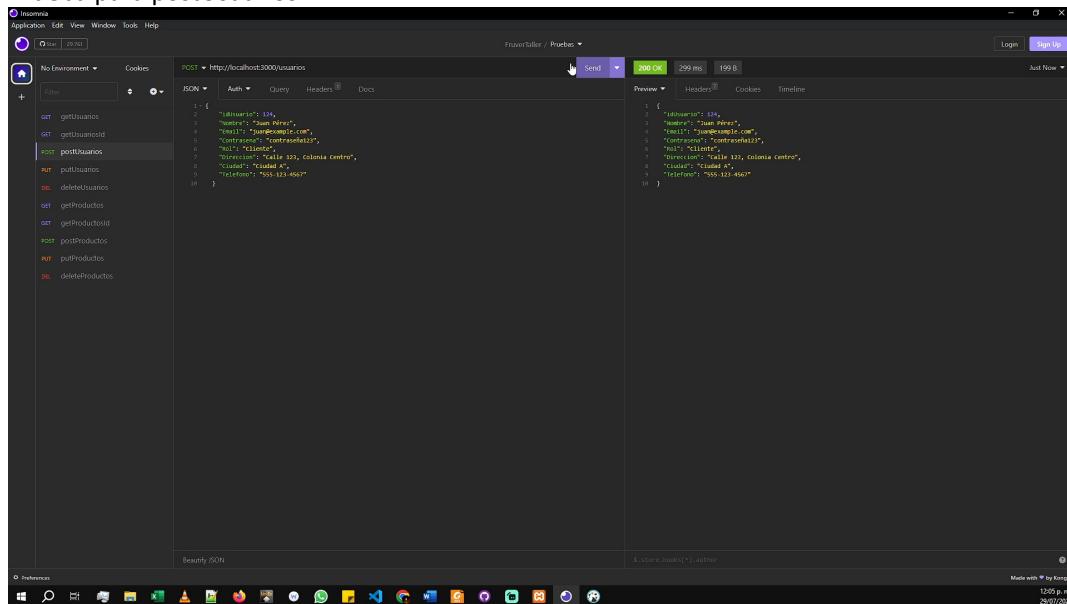
Prueba para getUsuariosId



The screenshot shows the Insomnia REST Client interface. The URL is `http://localhost:3000/usuarios/123`. The response status is `200 OK` with a `8.48 ms` latency and `201 B` body size. The response was received `Just Now`. The response body is a JSON object:

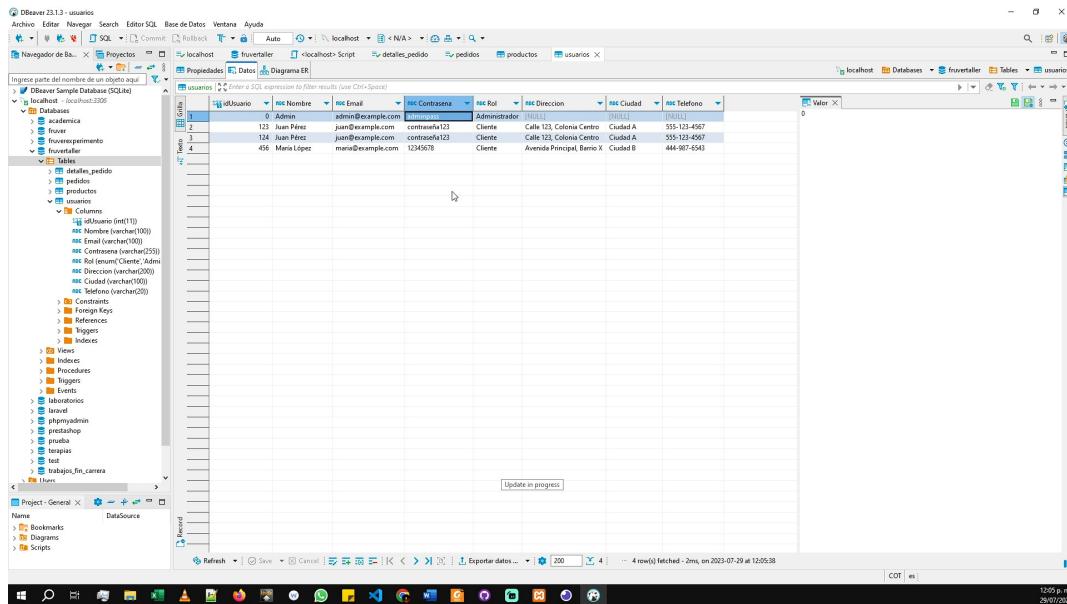
```
{"id": 2, "name": "Maria Hernandez", "email": "maria@example.com", "telefono": "555-2345678", "rol": "Cliente", "dirección": "Calle 123, colonia centro", "cliente": {"calle": "Calle 123", "colonia": "Centro", "numero": 1234, "telefono": "555-321-4567"}}
```

Prueba para postUsuarios



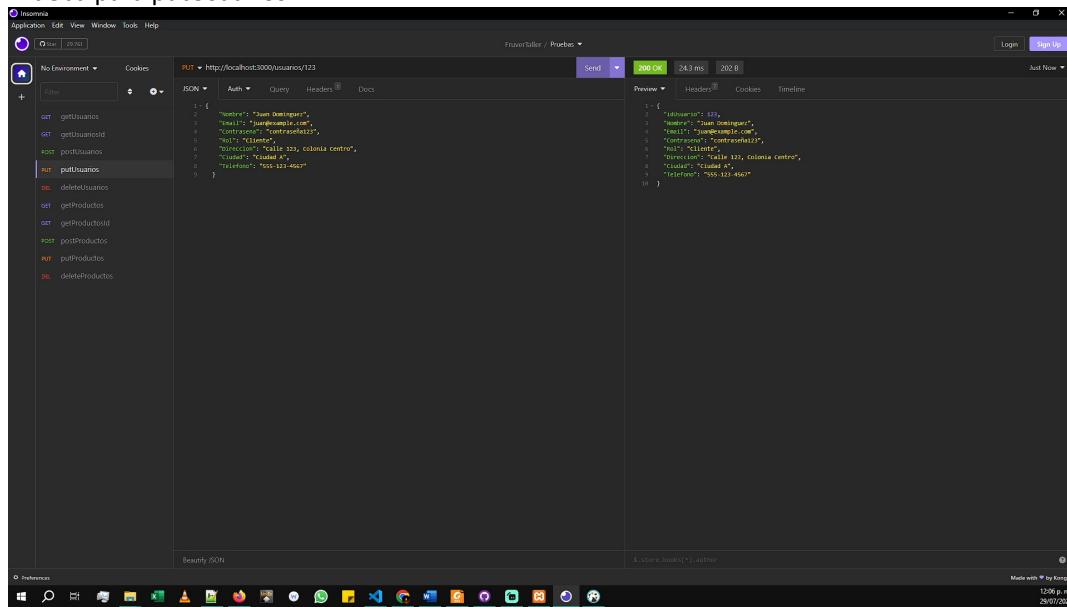
```
POST http://localhost:3000/usuarios
{
  "idUsuario": 123,
  "Nombre": "Juan Pérez",
  "Apellido": "Pérez",
  "Email": "juan@example.com",
  "Contraseña": "contraseña123",
  "Teléfono": "555-123-4567",
  "Dirección": "Calle 123, Colonia Centro",
  "Ciudad": "Ciudad A",
  "Teléfono": "555-123-4567"
}
```

Se observa cambios en la base de datos



Gris	idUsuario	Nombre	Email	Contraseña	Rol	Dirección	Ciudad	Teléfono
1	1	Admín	admin@example.com	contraseña123	Administrador	Calle 123, Colonia Centro	Ciudad A	555-123-4567
2	2	Juan Pérez	juan@example.com	contraseña123	Cliente	Calle 123, Colonia Centro	Ciudad A	555-123-4567
3	3	Juan Pérez	juan@example.com	contraseña123	Cliente	Calle 123, Colonia Centro	Ciudad A	555-123-4567
4	456	Maria López	maria@example.com	12345678	Cliente	Avenida Principal, Barrio X	Ciudad B	444-887-6543

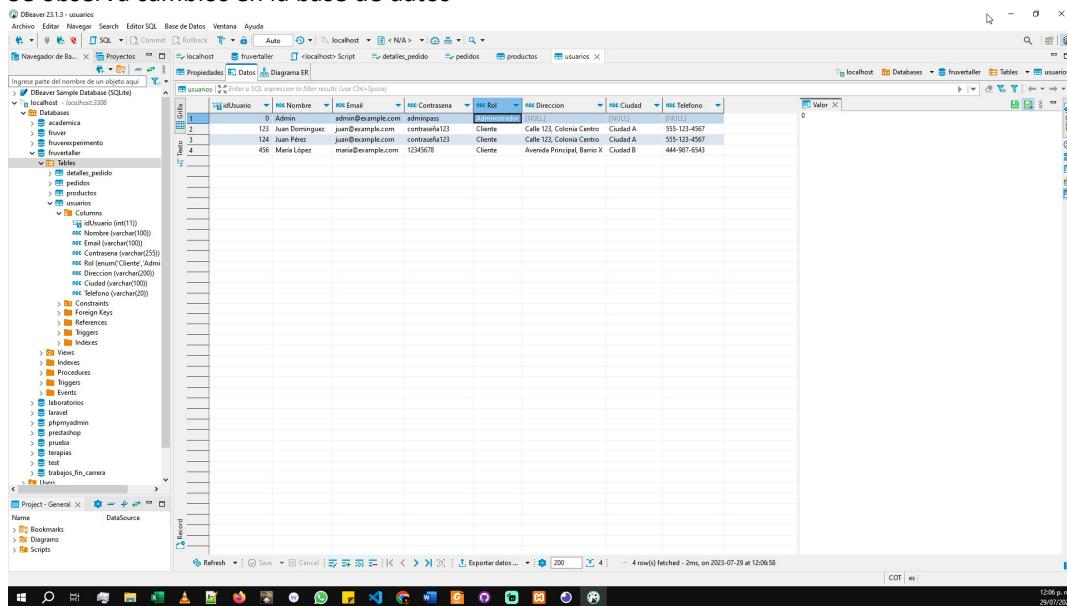
Prueba para putUsuarios



The screenshot shows a POSTMAN interface. The URL is `PUT http://localhost:3000/usuarios/123`. The request body is a JSON object:

```
1: {  
2:   "Nombre": "Juan Dominguez",  
3:   "Email": "juandominguez@example.com",  
4:   "Contrasena": "contraseña123",  
5:   "Rol": "Cliente",  
6:   "Direccion": "Calle 123, colonia centro",  
7:   "Ciudad": "Ciudad A",  
8:   "Telefono": "555-123-4567"  
9: }
```

Se observa cambios en la base de datos



The screenshot shows the MySQL Workbench interface. The left sidebar shows the database structure for 'inventario'. The 'usuarios' table is selected, displaying the following data:

ID	Nombre	Email	Contrasena	Rol	Direccion	Ciudad	Telefono
1	Admín	admin@example.com	adminpass	Administrador	Calle 123, Colonia Centro	Ciudad A	555-123-4567
2	Juan Dominguez	juan@example.com	contraseña123	Cliente	Calle 123, Colonia Centro	Ciudad A	555-123-4567
3	Juan Pérez	juan@example.com	contraseña123	Cliente	Calle 123, Colonia Centro	Ciudad A	555-123-4567
4	Maria Lopez	maria@example.com	12345678	Cliente	Avenida Principal, Barrio X	Ciudad B	444-897-6543

Prueba para deleteUsuarios

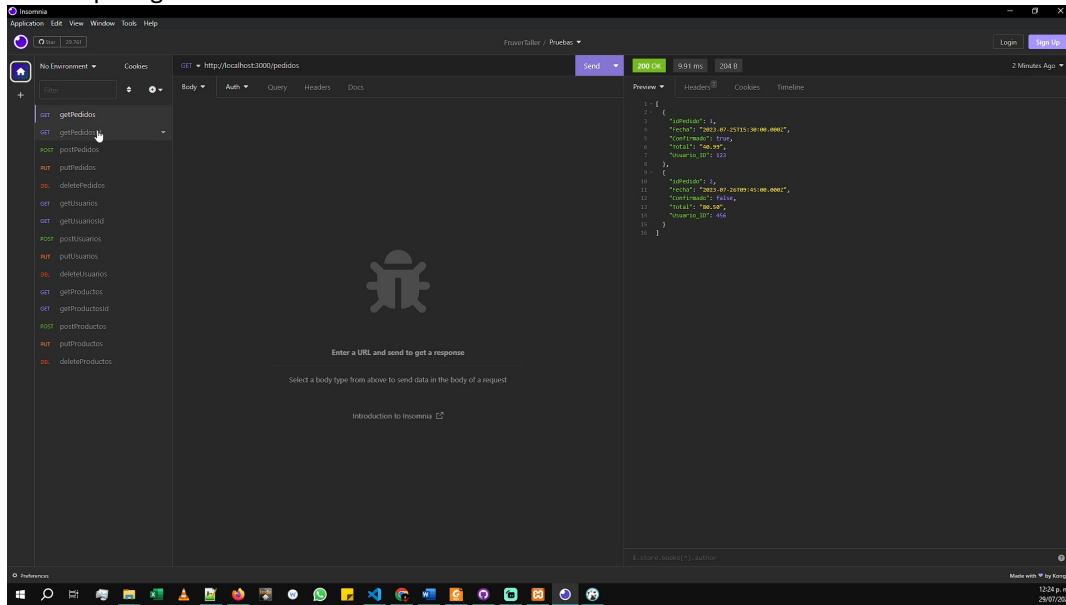
The screenshot shows the Insomnia REST Client interface. The top bar has tabs for Application, Edit, View, Window, Tools, and Help. A sidebar on the left lists various API endpoints under categories like No Environment, Cookies, and Headers. The main area shows a DELETE request to `http://localhost:3000/usuarios/124`. The response status is 200 OK with a time of 24.5 ms and a size of 44 B. The preview tab shows a JSON response with the message: `{ "Mensaje": "Registro con id 124 Eliminado" }`.

Se observa cambios en la base de datos

The screenshot shows the MySQL Workbench Data Grid interface. The left sidebar shows a database tree with a connection to 'localhost'. The central grid displays the 'usuarios' table with three rows of data. The columns are labeled: idUsuario, Nombre, Email, Contraseña, Rol, Direccion, Ciudad, and Telefono. The data is as follows:

	idUsuario	Nombre	Email	Contraseña	Rol	Direccion	Ciudad	Telefono
1	0	Admín	admin@example.com	contraseña123	Administrador	Calle 123, Colonia Centro	Ciudad A	555-123-4567
2	123	Juan Dominguez	juan@example.com	contraseña123	Cliente	Calle 123, Colonia Centro	Ciudad A	555-123-4567
3	456	Maria Lopez	maria@example.com	contraseña123	Cliente	Avenida Principal, Barrio X	Ciudad B	444-987-6543

Prueba para getPedidos



Prueba para getPedidos

GET http://localhost:3000/pedidos

200 OK 9.91 ms 204 B 2 Minutes Ago

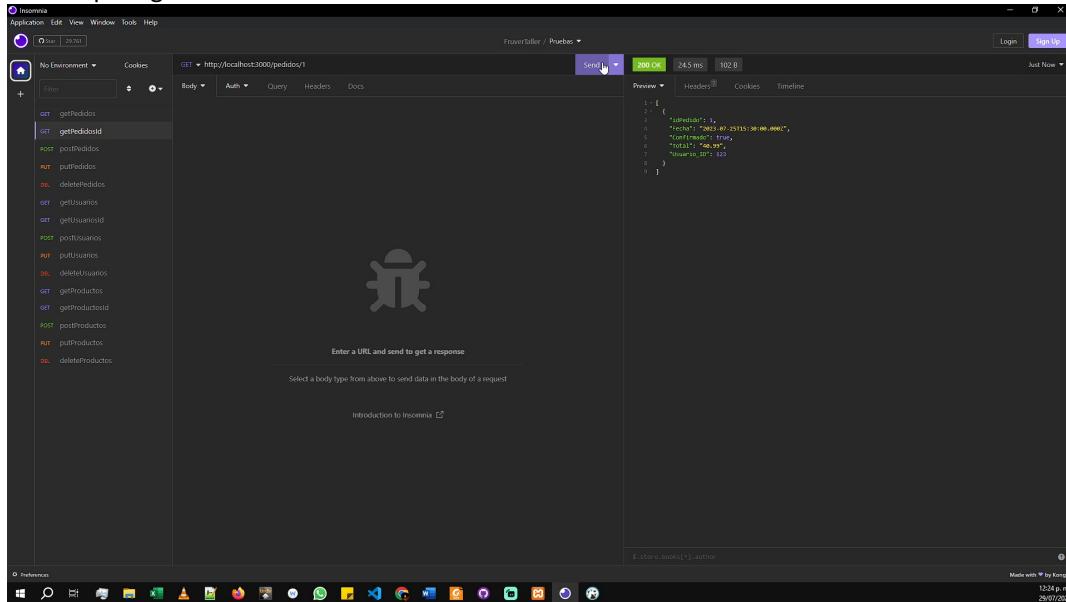
Preview Headers Cookies Timeline

```
[{"id": 1, "timestamp": "2023-07-25T15:30:00.000Z", "receipt": "2023-07-25T15:30:00.000Z", "confirmed": true, "total": "40.99", "user_id": "1"}, {"id": 2, "timestamp": "2023-07-26T09:05:00.000Z", "receipt": "2023-07-26T09:05:00.000Z", "confirmed": false, "total": "80.98", "user_id": "2"}]
```

Enter a URL and send to get a response

Introduction to Insomnia

Prueba para getPedidosId



Prueba para getPedidosId

GET http://localhost:3000/pedidos/1

200 OK 24.5 ms 102 B Just Now

Preview Headers Cookies Timeline

```
{"id": 1, "timestamp": "2023-07-25T15:30:00.000Z", "receipt": "2023-07-25T15:30:00.000Z", "confirmed": true, "total": "40.99", "user_id": "1"}
```

Enter a URL and send to get a response

Introduction to Insomnia

Prueba para postProductos

The screenshot shows a browser window with the title "Prueba para postProductos". The address bar indicates a POST request to "http://localhost:3000/pedidos". The response status is 200 OK with a response time of 58.5 ms and a body size of 100 B. The response content is a JSON object:

```
[{"idPedido": "2023-07-25T15:26:00.000Z", "idUsuario": "123", "Fecha": "2023-07-25 15:26:00", "Total": "40.99", "Confirmado": true}, {"idPedido": "2023-07-26T09:40:00.000Z", "idUsuario": "456", "Fecha": "2023-07-26 09:40:00", "Total": "80.5", "Confirmado": false}, {"idPedido": "2023-07-25T15:30:00.000Z", "idUsuario": "123", "Fecha": "2023-07-25 15:30:00", "Total": "40.99", "Confirmado": true}]
```

Below the browser, a screenshot of the MySQL Workbench interface shows the "pedidos" table. The table has columns: idPedido, idUsuario_ID, Fecha, idEstado, and Total. There are three rows of data, corresponding to the JSON objects above.

idPedido	idUsuario_ID	Fecha	idEstado	Total
1	123	2023-07-25 15:26:00	1	40.99
2	456	2023-07-26 09:40:00	0	80.5
3	123	2023-07-25 15:30:00	1	40.99

Prueba para putPedidos

The screenshot shows the Postman application interface. In the top left, there's a sidebar with various API endpoints listed under the 'pedidos' collection. The main area shows a single request: a PUT operation to 'http://localhost:3000/pedidos/4'. The response status is '200 OK' with a duration of '19.6 ms' and a size of '100 B'. The response body is displayed in JSON format:

```
[{"idPedido": 1, "Usuario_ID": 432, "Fecha": "2023-07-25 15:30:00", "Confirmado": 0, "Total": 40.99}, {"idPedido": 2, "Usuario_ID": 456, "Fecha": "2023-07-26 09:45:00", "Confirmado": 0, "Total": 80.5}, {"idPedido": 3, "Usuario_ID": 123, "Fecha": "2023-07-25 15:30:00", "Confirmado": 1, "Total": 40.99}, {"idPedido": 4, "Usuario_ID": 133, "Fecha": "2023-07-25 15:30:00", "Confirmado": 1, "Total": 90.99}], "total": 223
```

Se observa cambios en la base de datos

The screenshot shows the MySQL Workbench interface. On the left, the database structure is visible, including the 'pedidos' table. The main area displays the contents of the 'pedidos' table in a grid format. There are four rows of data, each representing a different order (idPedido 1, 2, 3, 4) placed by different users (Usuario_ID 432, 456, 123, 133) at different dates and times. The 'Confirmado' column indicates whether the order has been confirmed (0 for unconfirmed, 1 for confirmed), and the 'Total' column shows the total amount for each order.

idPedido	Usuario_ID	Fecha	Confirmado	Total
1	432	2023-07-25 15:30:00	0	40.99
2	456	2023-07-26 09:45:00	0	80.5
3	123	2023-07-25 15:30:00	1	40.99
4	133	2023-07-25 15:30:00	1	90.99

Prueba para deletePedidos

The screenshot shows the Insomnia REST Client interface. The URL is set to `http://localhost:3000/pedidos/4`. The response status is `200 OK` with a response time of `25.1 ms` and a size of `42.8`. The response body is a JSON object:

```
{ "Mensaje": "Registro con id 4 Eliminado" }
```

The left sidebar lists various API endpoints for the `pedidos` resource, including `getPedidos`, `getPedidosId`, `postPedidos`, `putPedidos`, `deletePedidos`, `getUsuarios`, `getUsuarioId`, `postUsuarios`, `putUsuarios`, `deleteUsuarios`, `getProductos`, `getProductoId`, `postProductos`, `putProductos`, and `deleteProductos`.

Se observa cambios en la base de datos

The screenshot shows the MySQL Workbench interface. The left sidebar shows the database structure for the `pedidos` schema, including tables like `pedidos`, `detalles_pedido`, `productos`, and `usuarios`. The main area displays the `pedidos` table with the following data:

ID Pedido	ID Usuario	Fecha	Estado	Total
1	1	2023-07-26 09:45:55	Confirmado	40.99
2	2	2023-07-26 09:45:00	Pendiente	80.5
3	3	2023-07-25 15:30:00	Confirmado	40.99

Prueba para getDetalles

The screenshot shows the Insomnia REST client interface. The top bar displays "Prueba para getDetalles". The main area shows a successful GET request to `http://localhost:3000/detallesP`. The response status is `200 OK`, with a response time of `12.6 ms` and a size of `478 B`. The response body is a JSON array containing five objects, each representing a detail with fields like `id`, `detallesId`, `kontrolId`, `kontrolName`, `pedidoId`, and `productoId`.

```
[{"id": 1, "detallesId": 1, "kontrolId": 1, "kontrolName": "41,48%", "pedidoId": 1, "productoId": 1}, {"id": 2, "detallesId": 2, "kontrolId": 2, "kontrolName": "40,08%", "pedidoId": 1, "productoId": 2}, {"id": 3, "detallesId": 3, "kontrolId": 3, "kontrolName": "39,08%", "pedidoId": 1, "productoId": 3}, {"id": 4, "detallesId": 4, "kontrolId": 4, "kontrolName": "38,08%", "pedidoId": 1, "productoId": 4}, {"id": 5, "detallesId": 5, "kontrolId": 5, "kontrolName": "29,08%", "pedidoId": 1, "productoId": 5}], [{"text": "\$-store.book[*].author"}]
```

Prueba para getDetallesId

The screenshot shows the Insomnia REST client interface. The top bar displays "Prueba para getDetallesId". The main area shows a successful GET request to `http://localhost:3000/detallesP/2`. The response status is `200 OK`, with a response time of `18.6 ms` and a size of `80 B`. The response body is a single JSON object representing a detail with fields like `id`, `detallesId`, `kontrolId`, `kontrolName`, `pedidoId`, and `productoId`.

```
{"id": 2, "detallesId": 2, "kontrolId": 2, "kontrolName": "40,08%", "pedidoId": 1, "productoId": 2}], [{"text": "\$-store.book[*].author"}]
```

Prueba para postDetalles

The screenshot shows the Insomnia REST client interface. A POST request is being made to the URL `http://localhost:3000/detalles/1`. The response status is 200 OK with a response time of 12.4 ms and a body size of 79 B. The response body is a JSON object:

```
[{"id": 1, "detalles": "Nuevo detalle", "resumen": "Resumen de Nuevo", "resumen_DET": 1, "cancelar": 2, "eliminar": 3, "actualizar": 4, "nuevo": 5}
```

Prueba para putDetalles

The screenshot shows the Insomnia REST client interface. A PUT request is being made to the URL `http://localhost:3000/detalles/1`. The response status is 200 OK with a response time of 25.4 ms and a body size of 79 B. The response body is a JSON object:

```
[{"id": 1, "detalles": "Nuevo detalle", "resumen": "Resumen de Nuevo", "resumen_DET": 1, "cancelar": 2, "eliminar": 3, "actualizar": 4, "nuevo": 5}
```

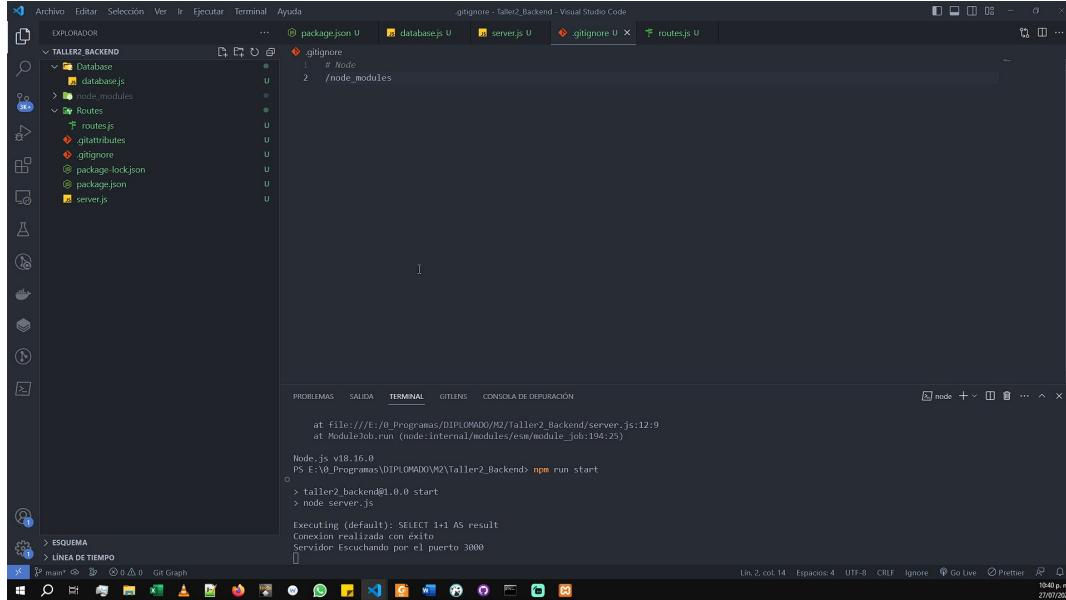
Prueba para deleteDetalles

The screenshot shows the Insomnia REST client interface. A DELETE request is being made to the URL `http://localhost:3000/detalles/1`. The response status is 200 OK with a response time of 4.55 ms and a body size of 42 B. The response body is a JSON object:

```
[{"message": "registro con id = 1 eliminado"}
```

4. Archivo en repositorio remoto

Creamos el archivo .gitignore para restringir la carga de los archivos en /node_modules

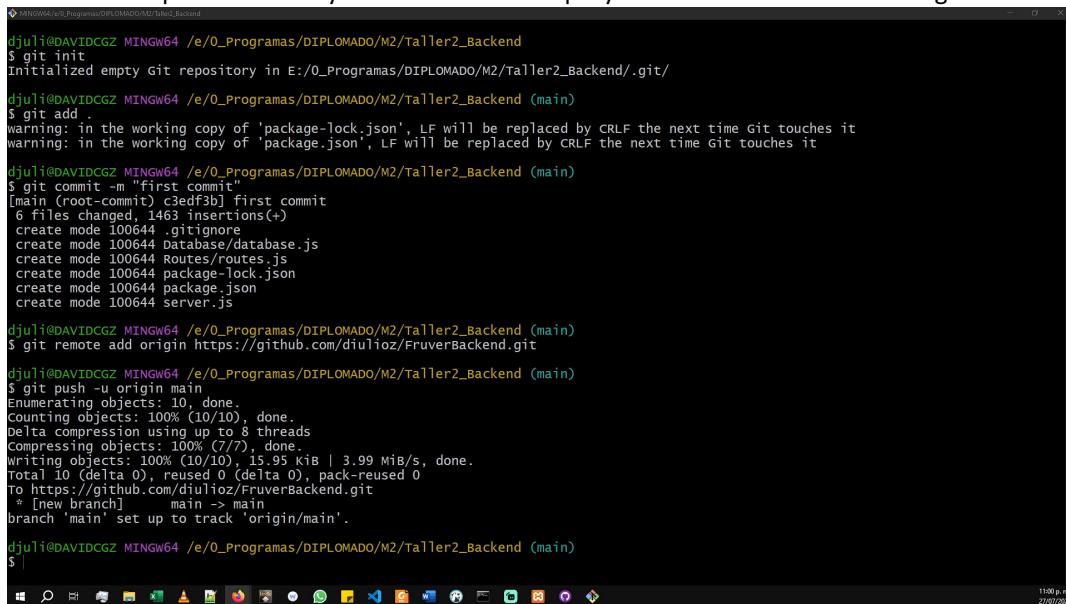


The screenshot shows the Visual Studio Code interface. The left sidebar displays a tree view of the project structure under 'EXPLORADOR'. The root folder 'TALLER2_BACKEND' contains 'Database', 'mode_modules', 'Routes', 'gitignore', 'package-lock.json', and 'server.js'. The 'gitignore' file is open in the center editor area, containing two lines: '# Node' and '/node_modules'. The bottom right panel shows a terminal window with the following output:

```
at File:///e:/0_Programas/DIPLOMADO/M2/Taller2_Backend/server.js:12:9
at ModuleJob.run (node:internal/modules/esm/module_job:194:25)
Node.js v18.16.0
PS E:\0_Programas\DIPLOMADO\M2\Taller2_Backend> npm run start
> taller2 backend@0.0.0 start
> node server.js
Executing (default): SELECT 1-1 AS result
Conexión realizada con éxito
Servidor Escuchando por el puerto 3000
```

The status bar at the bottom indicates 'Line 2, col. 14' and '1040 p. m. 27/07/2023'.

Creamos el repositorio local y lo sincronizamos al proyecto remoto FruverBackend.git



The screenshot shows a terminal window with the following session:

```
djuli@DAVIDCGZ MINGW64 /e/0_Programas/DIPLOMADO/M2/Taller2_Backend
$ git init
Initialized empty Git repository in E:/0_Programas/DIPLOMADO/M2/taller2_backend/.git/
djuli@DAVIDCGZ MINGW64 /e/0_Programas/DIPLOMADO/M2/Taller2_Backend (main)
$ 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
djuli@DAVIDCGZ MINGW64 /e/0_Programas/DIPLOMADO/M2/Taller2_Backend (main)
$ git commit -m "first commit"
[main root-commit c3edf3b] first commit
 6 files changed, 1463 insertions(+)
create mode 100644 .gitignore
create mode 100644 database/database.js
create mode 100644 Routes/routes.js
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 server.js
djuli@DAVIDCGZ MINGW64 /e/0_Programas/DIPLOMADO/M2/Taller2_Backend (main)
$ git remote add origin https://github.com/diulioz/FruverBackend.git
djuli@DAVIDCGZ MINGW64 /e/0_Programas/DIPLOMADO/M2/Taller2_Backend (main)
$ git push -u origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 15.95 Kib | 3.99 MiB/s, done.
Total 10 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/diulioz/FruverBackend.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

The status bar at the bottom indicates '1100 p. m. 27/07/2023'.