

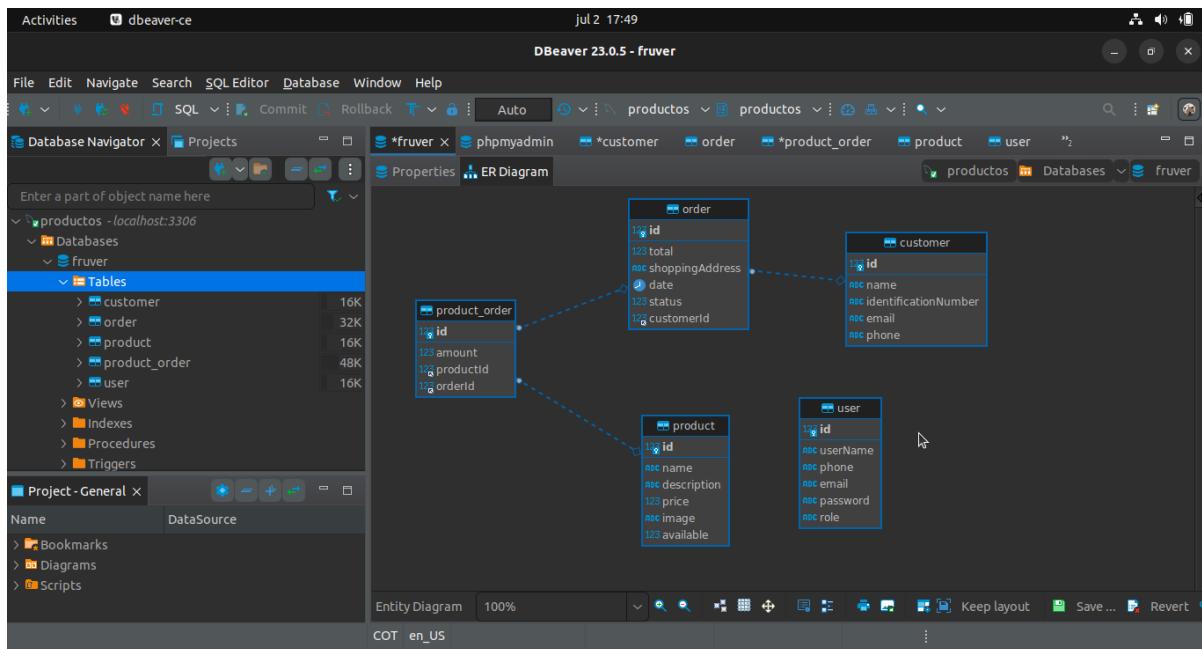
**Universidad de Nariño
Ingeniería de Sistemas**

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

**Taller Unidad 2 Backend
Estudiante: Juan José Revelo Jojoa**

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 llevar el registro de un FRUVER, se creó una base de datos con la siguiente estructura.



Como se puede observar en la anterior imagen, se cuenta con 5 tablas que permitirán llevar el control del fruver.

Existe una tabla de usuarios (user), la cual permitirá la autenticación de una persona al sistema para poder gestionar los productos y los pedidos de los clientes. Dicha tabla cuenta con 6 campos: los cuales son el id, el nombre de usuario, el teléfono, el email, el password y el rol. El rol en este caso se utilizará con el valor de “admin”, ya

que solo él puede entrar al dashboard de administración del sistema, mediante el usuario y la contraseña.

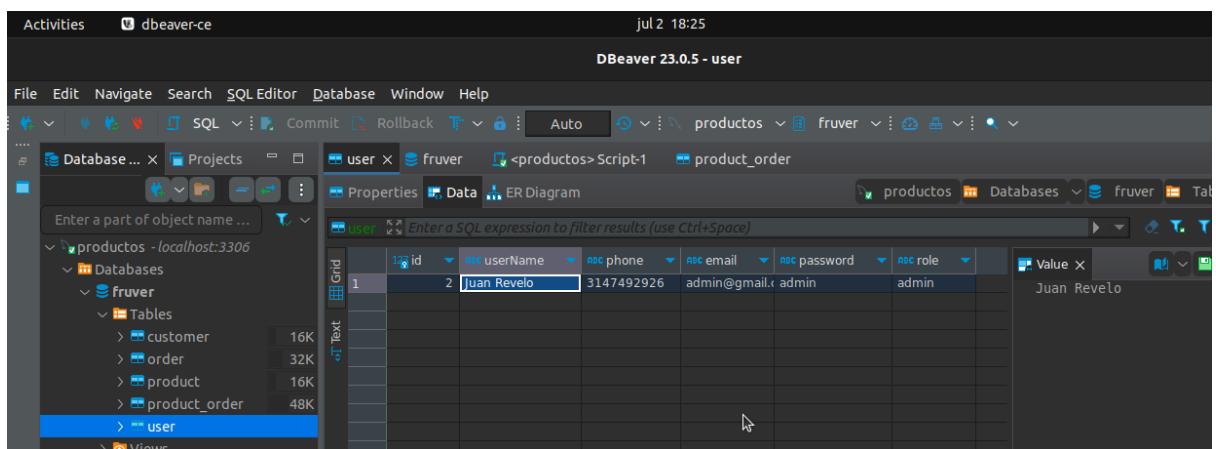
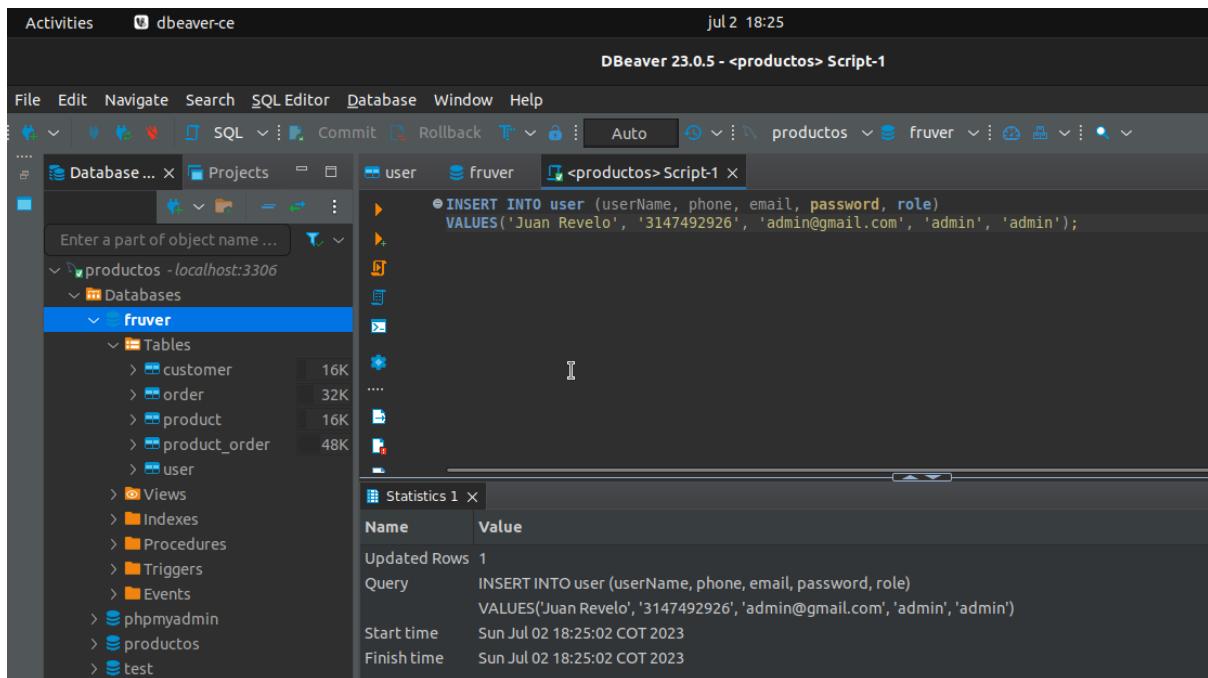
Por otro lado, la tabla de productos (product) permitirá la administración de los productos (frutas y verduras), la cual cuenta con 6 campos: el id, el nombre, la descripción, el precio, la imagen (base64) y la disponibilidad, es decir, para saber si el producto se encuentra disponible o está agotado.

La tabla de clientes (customer) permitirá el registro de un cliente cuando se realice la solicitud de un pedido, la cual cuenta con 5 campos: el id, el nombre, el número de identificación, el email (importante, ya que a este email se notificará la consolidación y envío del pedido cuando el administrador atienda dicha solicitud), y el teléfono.

La tabla de pedidos (order), permitirá controlar el proceso de solicitud de compra de unos respectivos clientes. Dicha tabla cuenta con 6 campos, el id, el total del pedido, la dirección de envío, la fecha de solicitud del pedido, el estado (es decir, para controlar si el pedido ya fue atendido o todavía no), y el id del cliente, para referenciar ese pedido a un cliente.

Finalmente, la tabla intermedia (product_order) entre pedidos (order) y productos (product), sirve para gestionar el detalle de un respectivo pedido, ya que en un pedido pueden haber muchos productos. Dicha tabla cuenta con cuatro campos: el id, la cantidad de un respectivo producto, el id del producto, y el id del pedido.

Tener en cuenta que se debe registrar un usuario desde la base de datos, ya que a este usuario se le permitirá la gestión de los productos y pedidos, ingresando al dashboard de la aplicación. La instrucción para ingresar un administrador correctamente, se muestra en la siguiente imagen.



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).

Una vez realizada la base de datos anterior, se continuó con el proyecto backend que se iba realizando en clase junto con el profesor, donde se implementó la siguiente estructura.

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

- File Explorer:** On the left, it shows the project structure under "FRUVER-BE". The "token" folder is currently selected.
- Editor:** The main area displays the content of the "database.js" file. The code uses Sequelize to connect to a MySQL database named "fruver".
- Status Bar:** At the bottom, it shows "Ln 7, Col 1" and other standard status bar information.

```
database > JS database.js > ...
1 import { Sequelize } from 'sequelize';
2
3 export const sequelize = new Sequelize('fruver', 'root', '', {
4   host: 'localhost',
5   dialect: 'mysql'
6 });
7
```

Como se puede observar en la anterior imagen, se cuenta con las siguientes carpetas:

controllers: en esta carpeta se encuentran los módulos o archivos JavaScript que contienen la lógica del manejo de las solicitudes HTTP y las respuestas correspondientes. Gracias a esta carpeta, se puede organizar el código y definir responsabilidades, ya que cada controlador se encarga de manejar un conjunto específico de rutas y funcionalidades relacionadas. Para el presente proyecto, se contará con tres archivos: uno para los productos, otro para usuarios y otro para los pedidos, los cuales se explicarán posteriormente.

database: el presente proyecto se lo trabajó con Sequelize, es decir, una biblioteca de Node.js que se utiliza como un ORM, por cuanto facilita la interacción y manipulación de bases de datos relacionales. Esta carpeta contiene el archivo database.js, el cual contiene la configuración para conectarse a la base de datos de mysql como se muestra en la siguiente imagen.

```
database.js - fruver-be - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
FRUVER-BE
  controllers
    products.js
    users.js
  database
    database.js
  models
    product.js
    user.js
  routes
    products.js
    user.js
  package.json
  prod ...
```

```
database > database.js > ...
1 import { Sequelize } from 'sequelize';
2
3 export const sequelize = new Sequelize('fruver', 'root', '', {
4   host: 'localhost',
5   dialect: 'mysql'
6 });
7
```

models: en esta carpeta se encuentran los modelos, los cuales representan las tablas en la base de datos, y cada instancia de un modelo representa una fila en dicha tabla. Los modelos definen la estructura de los datos, las relaciones entre tablas y validaciones. Para el presente caso, se contará con 5 modelos respectivamente: uno para los productos, otro para los usuarios, otro para los pedidos, otro para los clientes, y el último para la tabla intermedia entre los productos y los pedidos. Dichos modelos se explicarán posteriormente.

routes: dicha carpeta se utiliza para organizar y gestionar las rutas de la aplicación, es decir, se manejan los endpoints que una aplicación expone para recibir solicitudes HTTP y devolver respuestas. Para el presente caso, se contará con 3 archivos de rutas: uno para los productos, otro para los usuarios y el último para los pedidos. Dichos archivos se explicarán posteriormente.

token: dicha carpeta contiene un archivo denominado token.js, el cuál se encarga de validar y autenticar los tokens JWT en las solicitudes entrantes, es decir, se utiliza para proteger ciertas rutas o recursos de la aplicación que más adelante se mostrarán. Como se muestra en la siguiente imagen, el archivo contiene la función verifyToken, la cual utiliza el encabezado “Bearer” como parte de la autenticación basada en tokens, indicando que se está enviando un token de acceso para autenticar la solicitud, donde si dicho token es correcto, se permite continuar a una siguiente función (consumir un endpoint), o de lo contrario no permitir dicho acceso a un endpoint.

The screenshot shows the Visual Studio Code interface with the title bar "Activities Visual Studio Code" and "token.js - fruver-be - Visual Studio Code". The status bar indicates "Jul 2 19:38". The Explorer sidebar on the left shows a project structure for "FRUVER-BE" with files like products.js, users.js, database.js, models.js, routes.js, index.js, package-lock.json, package.json, and server.js. The token.js file is selected in the Explorer. The main editor area displays the following code:

```
token > JS token.js > verifyToken
1 import jwt from 'jsonwebtoken';
2
3 // Authorization: Bearer <token>
4 export function verifyToken(req, res, next) {
5   const bearerHeader = req.headers['authorization'];
6
7   // Verificamos que el encabezado tenga la propiedad authorization
8   if (!bearerHeader) {
9     return res.status(401).json("No autorizado");
10  }
11
12  const bearerToken = bearerHeader.split(" ")[1];
13  // verificamos que el token sea válido
14  jwt.verify(bearerToken, "SECRET", (err, response) => {
15    if (err) {
16      return res.sendStatus(403);
17    } else {
18      res.locals = response;
19      // Permite avanzar a la siguiente función
20      next();
21    }
22  });
23 }
```

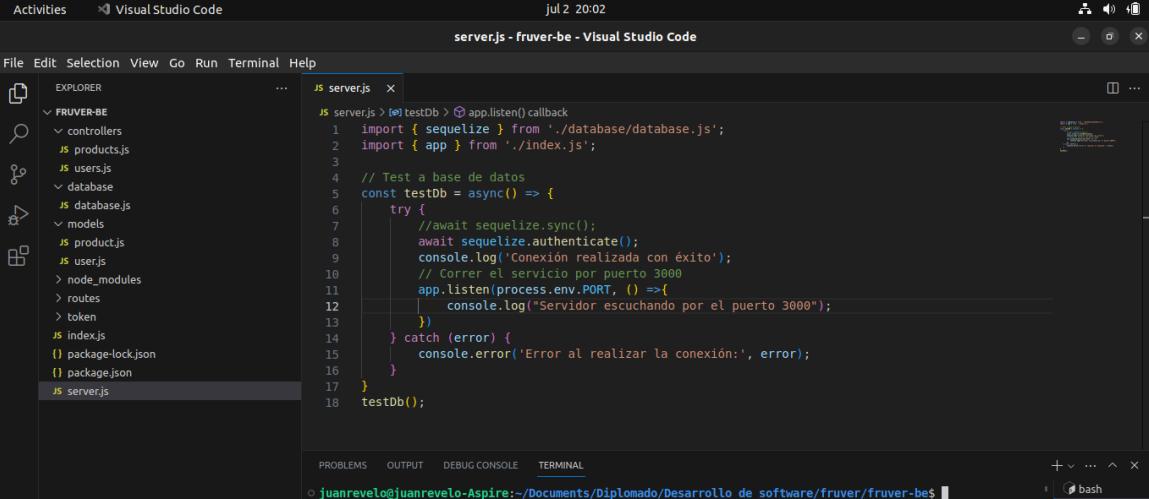
También se cuenta con un archivo denominado **index.js**, el cual se utiliza para la configuración de middleware. La presente aplicación se está trabajando con express, y express utiliza middleware para manejar solicitudes y respuestas. La siguiente imagen muestra la configuración y uso del middleware para cors, donde se utiliza para habilitar las solicitudes HTTP desde un origen cruzado; otro para analizar las solicitudes con formato de formulario; otro para analizar las solicitudes con formato JSON, y los que siguen para las rutas, es decir, para ejecutar código específico en medio del procesamiento de una solicitud antes de que se maneje una ruta en particular.

The screenshot shows the Visual Studio Code interface with the title bar "Activities Visual Studio Code" and "index.js - fruver-be - Visual Studio Code". The status bar indicates "Jul 2 19:56". The Explorer sidebar on the left shows a project structure for "FRUVER-BE" with files like products.js, users.js, database.js, models.js, routes.js, index.js, package-lock.json, package.json, and server.js. The index.js file is selected in the Explorer. The main editor area displays the following code:

```
index.js > ...
1 import express from 'express';
2 import cors from 'cors';
3 import { userRoutes } from './routes/users.js';
4 import { productRoutes } from './routes/products.js';
5 export const app = express();
6
7 app.use(cors());
8 // Middleware para analizar las solicitudes con formato de formulario
9 app.use(express.urlencoded({extended: true, limit: '20mb'})); // Se aumenta el límite a 20mb
10 // Middleware para analizar las solicitudes con formato JSON
11 app.use(express.json({limit: '20mb'})); // Se aumenta el límite a 20mb
12 app.use('/users', userRoutes);
13 app.use('/products', productRoutes);
```

Finalmente se cuenta con el archivo **server.js**, donde se realiza un test utilizando sequelize para probar si la conexión se encuentra correctamente establecida, y

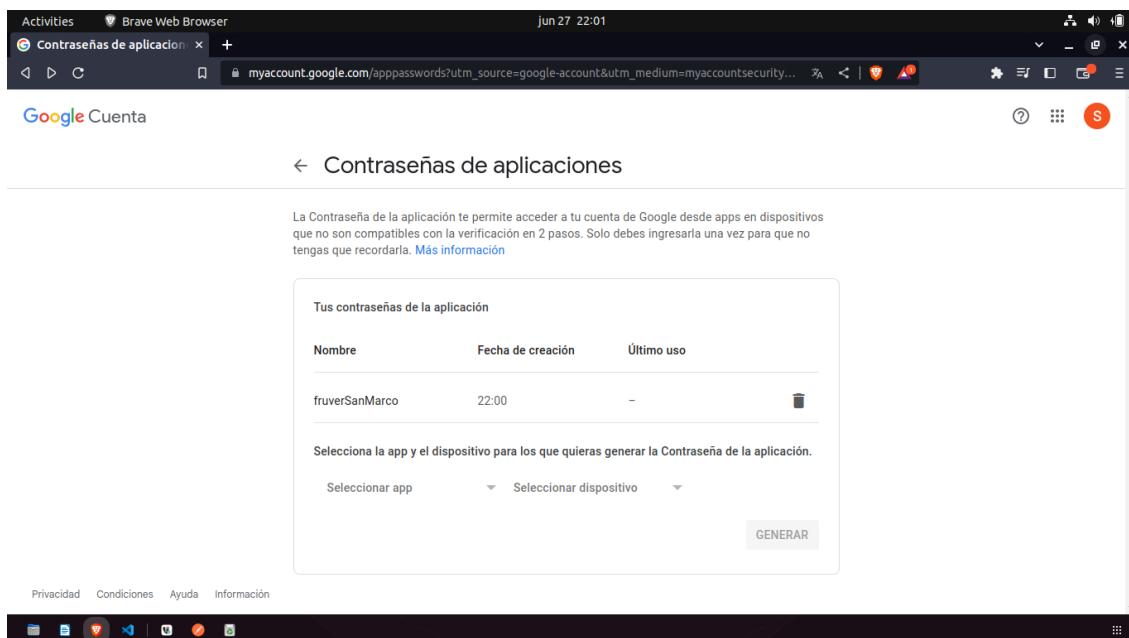
dependiendo de eso, se corre el servicio por el puerto 3000 como se muestra a continuación.



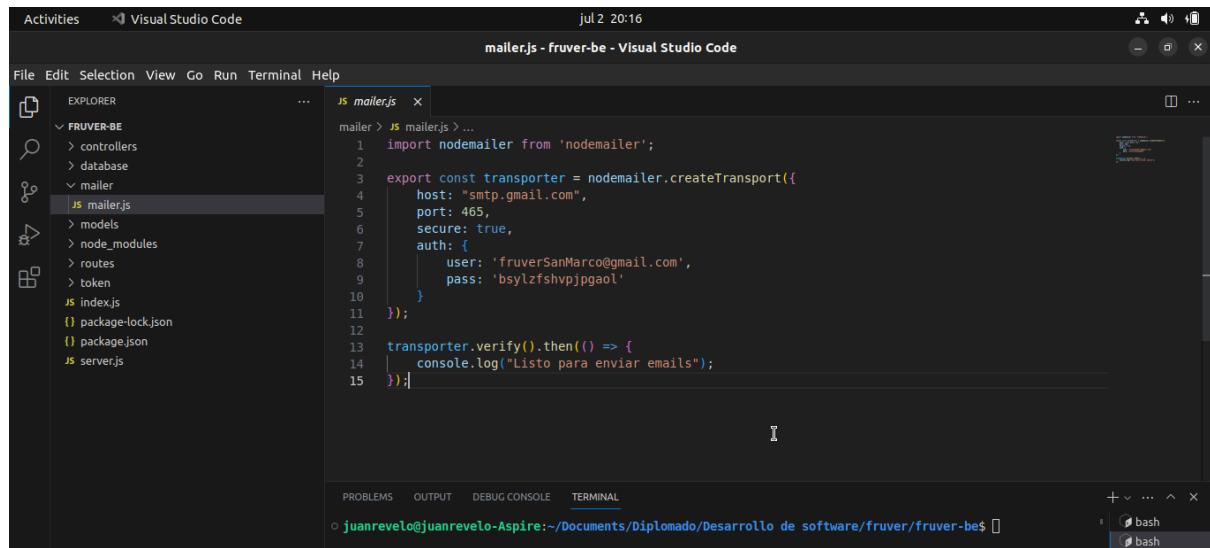
The screenshot shows the Visual Studio Code interface with the 'server.js' file open in the editor. The code is written in JavaScript and defines a server setup. It imports Sequelize and app from database/database.js and index.js respectively. It then sets up a test database (testDb) and attempts to sync it. If successful, it logs a message and starts the application on port process.env.PORT. If there's an error, it logs the error message. Finally, it calls testDb(). The code is as follows:

```
js server.js
1 import { Sequelize } from './database/database.js';
2 import { app } from './index.js';
3
4 // Test a base de datos
5 const testDb = async() => {
6   try {
7     await sequelize.sync();
8     await sequelize.authenticate();
9     console.log('Conexión realizada con éxito');
10    // Correr el servicio por puerto 3000
11    app.listen(process.env.PORT, () => {
12      console.log("Servidor escuchando por el puerto 3000");
13    })
14  } catch (error) {
15    console.error('Error al realizar la conexión:', error);
16  }
17}
18 testDb();
```

Como es necesario notificar al cliente que su pedido fue consolidado y enviado con éxito a través de un correo electrónico, se utilizó el módulo de nodemailer, el cual permite el envío de correos electrónicos desde una aplicación de node.js. Para el presente caso, se creó una cuenta de gmail denominada fruverSanMarco@gmail.com para enviar los respectivos correos a los clientes, donde se tuvo que crear una contraseña de aplicaciones para permitir la autenticación desde node.js como se muestra en la siguiente imagen.



Una vez realizado lo anterior, se configuró nodemailer de la siguiente manera para enviar correos electrónicos, dentro del archivo **mailer.js**.

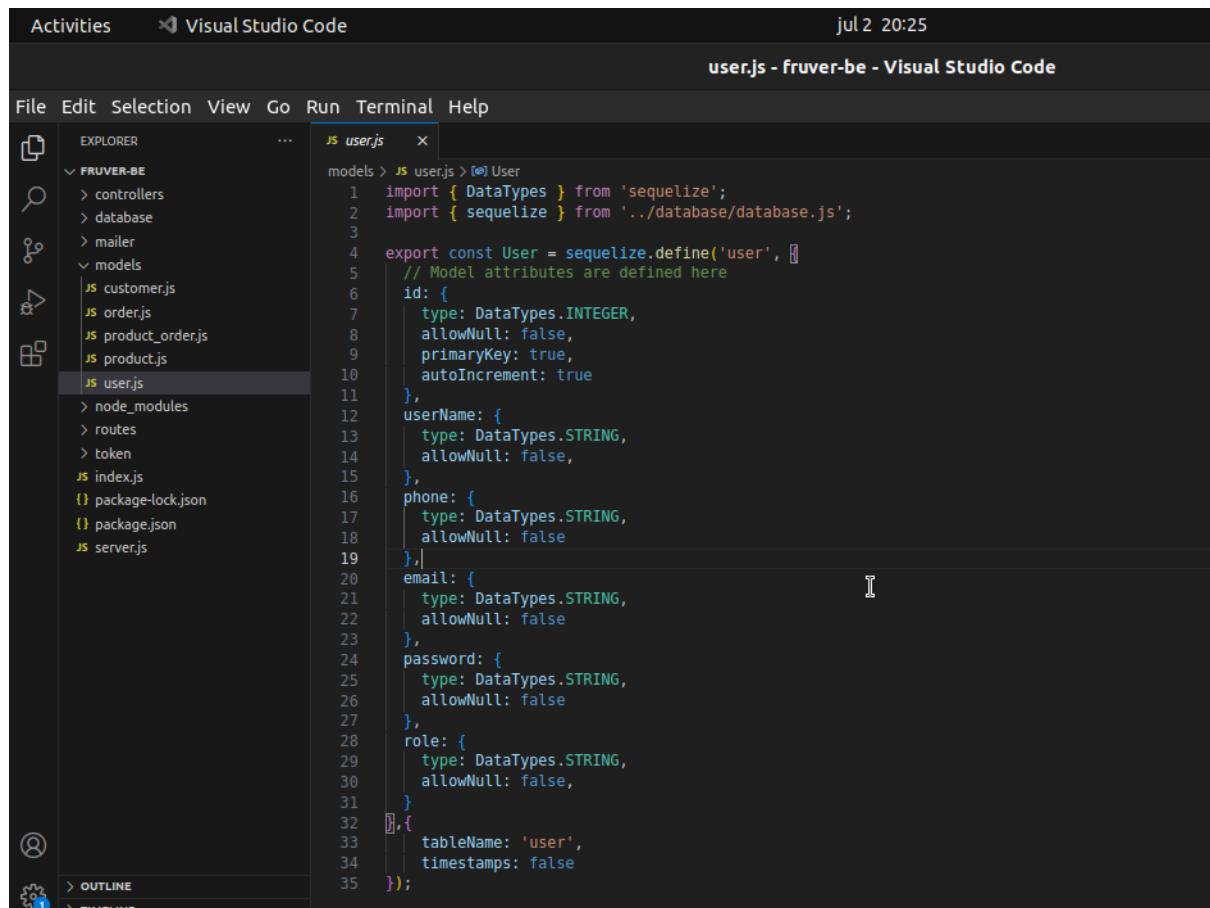


The screenshot shows the Visual Studio Code interface with the title bar "Activities Visual Studio Code" and "jul 2 20:16". The active tab is "mailer.js - fruver-be - Visual Studio Code". The code editor contains the following JavaScript code:

```
1 import nodemailer from 'nodemailer';
2
3 export const transporter = nodemailer.createTransport({
4   host: "smtp.gmail.com",
5   port: 465,
6   secure: true,
7   auth: {
8     user: 'fruverSanMarco@gmail.com',
9     pass: 'bsylzfshvpjrgao1'
10   }
11 });
12
13 transporter.verify().then(() => {
14   console.log("Listo para enviar emails");
15 });
```

The Explorer sidebar on the left shows the project structure: "FRUVER-BE" with "controllers", "database", "mailer" (containing "mailer.js"), "models", "node_modules", "routes", "token", "index.js", "package-lock.json", "package.json", and "server.js". The bottom status bar shows the terminal command: "juanrevelo@juanrevelo-Aspire:~/Documents/Diplomado/Desarrollo de software/fruver/fruver-be\$".

Como se mencionó anteriormente, se definieron 5 modelos. Uno de ellos el de usuario, el cual se muestra a continuación.



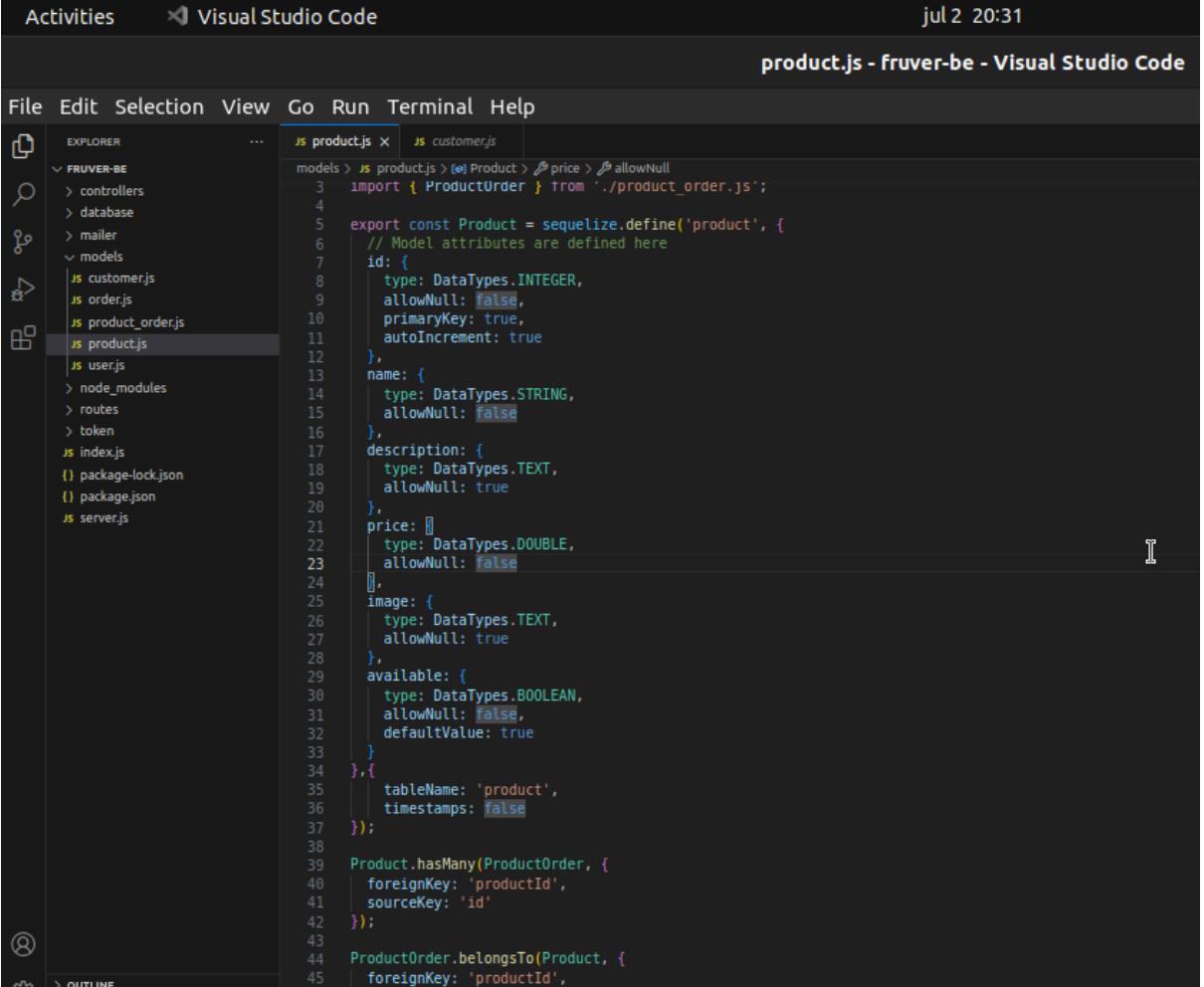
The screenshot shows the Visual Studio Code interface with the title bar "Activities Visual Studio Code" and "jul 2 20:25". The active tab is "user.js - fruver-be - Visual Studio Code". The code editor contains the following JavaScript code:

```
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from '../database/database.js';
3
4 export const User = sequelize.define('user', [
5   // Model attributes are defined here
6   id: {
7     type: DataTypes.INTEGER,
8     allowNull: false,
9     primaryKey: true,
10    autoIncrement: true
11   },
12   userName: {
13     type: DataTypes.STRING,
14     allowNull: false,
15   },
16   phone: {
17     type: DataTypes.STRING,
18     allowNull: false
19   },
20   email: {
21     type: DataTypes.STRING,
22     allowNull: false
23   },
24   password: {
25     type: DataTypes.STRING,
26     allowNull: false
27   },
28   role: {
29     type: DataTypes.STRING,
30     allowNull: false,
31   }
32 ],
33 {
34   tableName: 'user',
35   timestamps: false
36 });
37
```

The Explorer sidebar on the left shows the project structure: "FRUVER-BE" with "controllers", "database", "mailer", "models" (containing "customer.js", "order.js", "product_order.js", "product.js", and "user.js"), "node_modules", "routes", "token", "index.js", "package-lock.json", "package.json", and "server.js". The bottom status bar shows the terminal command: "juanrevelo@juanrevelo-Aspire:~/Documents/Diplomado/Desarrollo de software/fruver/fruver-be\$".

Como se puede observar en la anterior imagen, se define un modelo denominado 'User' con los campos que se muestran respectivamente. Cada campo tiene un tipo de dato y se puede especificar opciones adicionales como las que se muestran en la imagen.

Otro modelo es el de Product, como se muestra a continuación.



The screenshot shows the Visual Studio Code interface with the title bar "Activities" and "Visual Studio Code" and the date "jul 2 20:31". The tab bar shows "product.js - fruver-be - Visual Studio Code". The code editor displays the following JavaScript code for the "product" model:

```
models > js product.js > (e) Product > price > allowNull
3   import { ProductOrder } from './product_order.js';
4
5   export const Product = sequelize.define('product', {
6     // Model attributes are defined here
7     id: {
8       type: DataTypes.INTEGER,
9       allowNull: false,
10      primaryKey: true,
11      autoIncrement: true
12    },
13    name: {
14      type: DataTypes.STRING,
15      allowNull: false
16    },
17    description: {
18      type: DataTypes.TEXT,
19      allowNull: true
20    },
21    price: [
22      type: DataTypes.DOUBLE,
23      allowNull: false
24    ],
25    image: {
26      type: DataTypes.TEXT,
27      allowNull: true
28    },
29    available: {
30      type: DataTypes.BOOLEAN,
31      allowNull: false,
32      defaultValue: true
33    }
34  },
35  {
36    tableName: 'product',
37    timestamps: false
38  });
39 ProducthasMany(ProductOrder, {
40   foreignKey: 'productId',
41   sourceKey: 'id'
42 });
43
44 ProductOrder.belongsTo(Product, {
45   foreignKey: 'productId',
46   targetKey: 'id'
47 });
```

Con diferencia al anterior, el modelo de producto cuenta con asociaciones, es decir, para representar relaciones comunes con otras tablas de la base de datos.

El producto tiene una asociación de uno a muchos con el modelo ProductOrder, y a su vez, el modelo ProductOrder pertenece a Product, por tal razón, la llave foránea va en la tabla de product_order.

De la misma manera sucede con el modelo Order, como se muestra a continuación.

Activities Visual Studio Code jul 2 20:34

order.js - fruver-be - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXPLORER ... JS product.js JS order.js

models > JS order.js > (e) Order > JS date

```
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from './database/database.js';
3 import { ProductOrder } from './product_order.js';
4
5 export const Order = sequelize.define('order', {
6   // Model attributes are defined here
7   id: {
8     type: DataTypes.INTEGER,
9     allowNull: false,
10    primaryKey: true,
11    autoIncrement: true
12  },
13  total: {
14    type: DataTypes.DOUBLE,
15    allowNull: false,
16  },
17  shoppingAddress: {
18    type: DataTypes.STRING,
19    allowNull: false,
20  },
21  date: [
22    type: DataTypes.DATEONLY,
23    allowNull: false,
24  ],
25  status: {
26    type: DataTypes.BOOLEAN,
27    allowNull: false,
28    defaultValue: false,
29  }
30 }, {
31   tableName: 'order',
32   timestamps: false
33 });
34
35 Order.hasMany(ProductOrder, {
36   foreignKey: 'orderId',
37   sourceKey: 'id'
38 });
39
40 ProductOrder.belongsTo(Order, {
41   foreignKey: 'orderId',
42   targetKey: 'id'
43 });
```

OUTLINE

Igual que el anterior, el modelo Order cuenta con una asociación de uno a muchos con el modelo ProductOrder, y de la misma manera, el modelo ProductOrder pertenece al modelo Order, por tal razón, la llave foránea va en la tabla product_order.

La siguiente imagen muestra la definición del modelo ProductOrder.

Activities Visual Studio Code jul 2 20:37

product_order.js - fruver-be - Visual Studio Code

File Edit Selection View Go Run Terminal Help

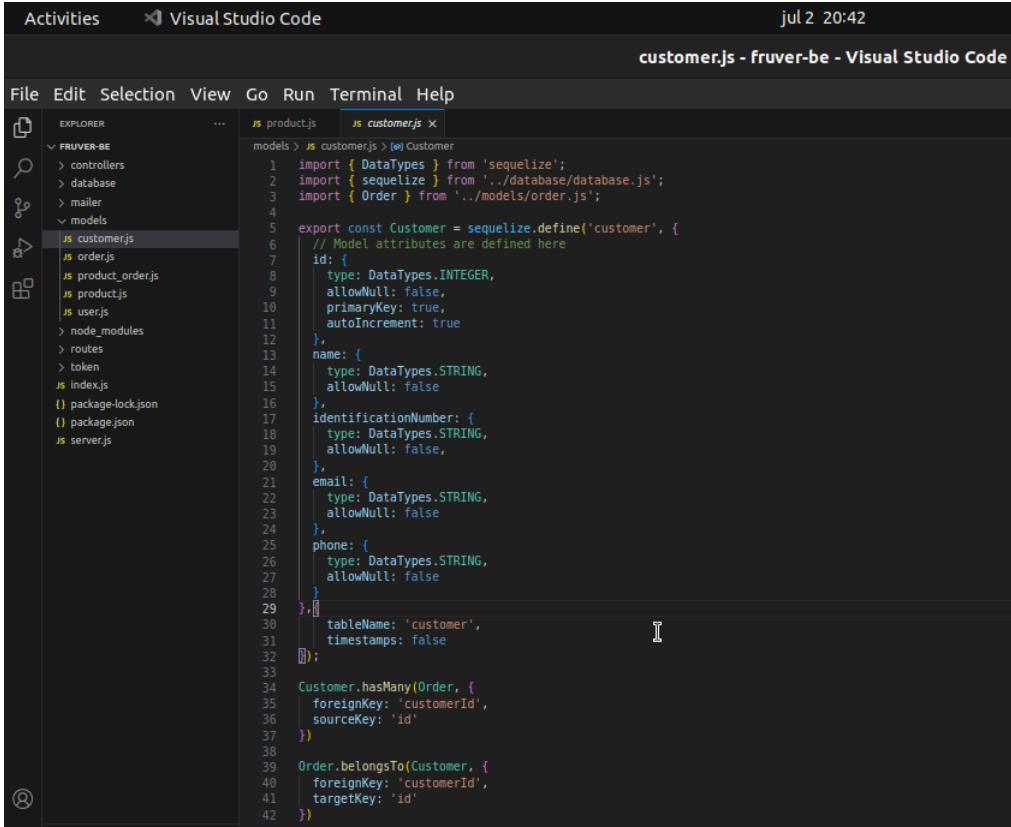
EXPLORER ... JS product.js JS product_order.js

models > JS product_order.js > ...

```
1 import { DataTypes } from 'sequelize';
2 import { sequelize } from '../database/database.js';
3
4 export const ProductOrder = sequelize.define('product_order', {
5   // Model attributes are defined here
6   id: {
7     type: DataTypes.INTEGER,
8     allowNull: false,
9     primaryKey: true,
10    autoIncrement: true
11  },
12  amount: {
13    type: DataTypes.INTEGER,
14    allowNull: false,
15  }
16 }, {
17   tableName: 'product_order',
18   timestamps: false
19 });
```

En dicha definición, no se establece la asociación, ya que este modelo se lo referenció en los modelos de Order y Product respectivamente.

Finalmente se muestra el modelo Customer.



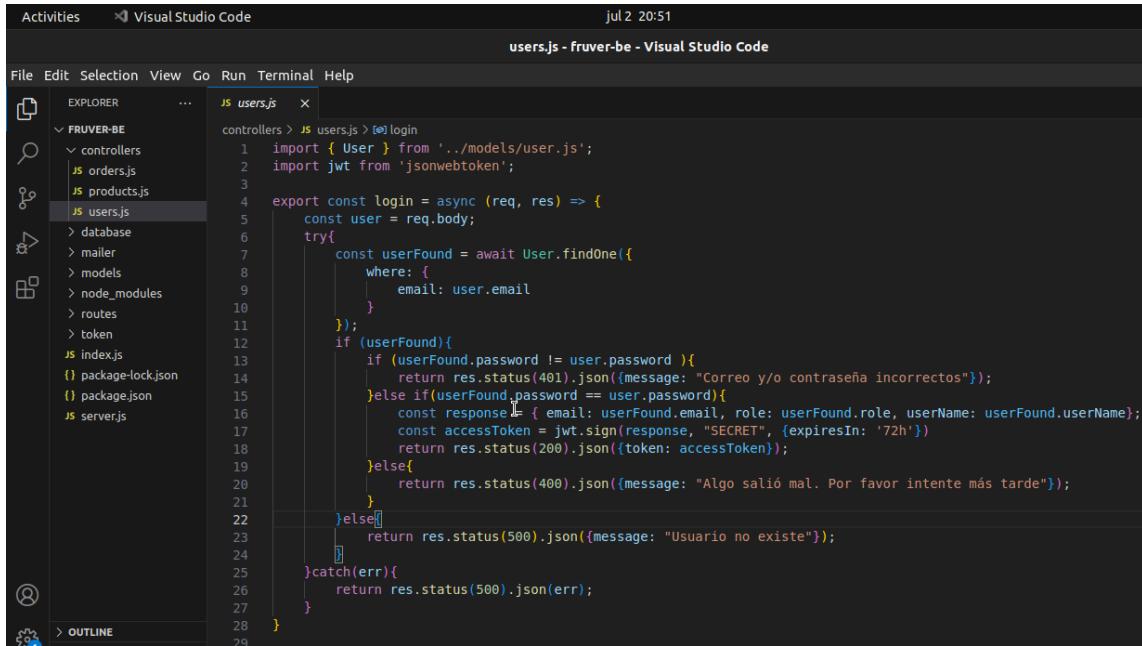
```
customer.js - fruver-be - Visual Studio Code
File Edit Selection View Go Run Terminal Help
File Explorer Terminal
customer.js - fruver-be - Visual Studio Code
customer.js
models > customer > (e) Customer
1 import { DataTypes } from 'sequelize';
2 import { Sequelize } from '../database/database.js';
3 import { Order } from '../models/order.js';
4
5 export const Customer = sequelize.define('customer', {
6   // Model attributes are defined here
7   id: {
8     type: DataTypes.INTEGER,
9     allowNull: false,
10    primaryKey: true,
11    autoIncrement: true
12  },
13  name: {
14    type: DataTypes.STRING,
15    allowNull: false
16  },
17  identificationNumber: {
18    type: DataTypes.STRING,
19    allowNull: false,
20  },
21  email: {
22    type: DataTypes.STRING,
23    allowNull: false
24  },
25  phone: {
26    type: DataTypes.STRING,
27    allowNull: false
28  }
29 });
30
31 Customer.hasMany(Order, {
32   foreignKey: 'customerId',
33   sourceKey: 'id'
34 })
35
36 Order.belongsTo(Customer, {
37   foreignKey: 'customerId',
38   targetKey: 'id'
39 })
40
41
42 }
```

Dicho modelo cuenta con una asociación de uno a muchos con el modelo Order, y de la misma manera, el modelo Order pertenece al modelo Customer, por tal razón, la llave foránea va en la tabla order.

Una vez lista la definición de todos los modelos, se realiza la lógica del manejo de las solicitudes HTTP y las respuestas correspondientes, dentro de la carpeta controllers. Como se mencionó en un inicio, para el presente proyecto se contará con tres archivos: uno para los productos, otro para los usuarios y otro para los pedidos.

El archivo **users.js** cuenta con 3 funciones. La primera de ellas es la función de "login", la cual encuentra un usuario mediante su correo electrónico, donde si el usuario existe, compara si las contraseñas son iguales, y si es así, retornará un

token y la información del respectivo usuario, de lo contrario retornará un mensaje informando el respectivo error como se muestra a continuación.

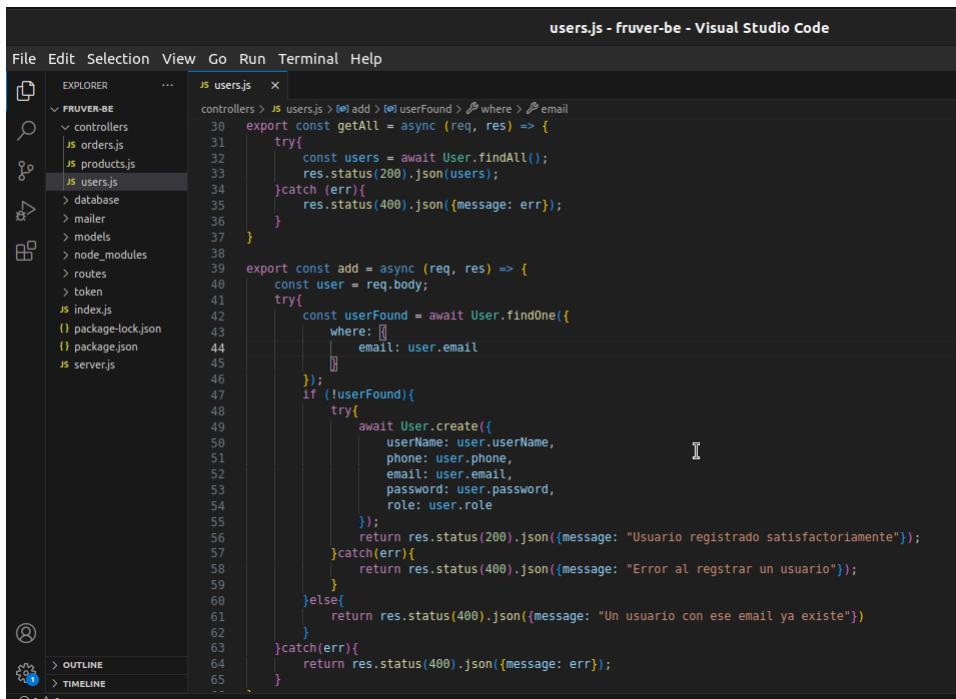


The screenshot shows the Visual Studio Code interface with the 'users.js' file open in the editor. The code implements a login function that checks if a user exists in the database. If the password is correct, it generates a JWT token and returns it. If the user is not found or the password is incorrect, it returns an error message. If an unexpected error occurs, it returns a generic error message.

```
import { User } from '../models/user.js';
import jwt from 'jsonwebtoken';

export const login = async (req, res) => {
  const user = req.body;
  try{
    const userFound = await User.findOne({
      where: [
        { email: user.email }
      ]
    });
    if (userFound){
      if (userFound.password != user.password ){
        return res.status(401).json({message: "Correo y/o contraseña incorrectos"});
      }else if(userFound.password == user.password){
        const response = { email: userFound.email, role: userFound.role, userName: userFound.userName };
        const accessToken = jwt.sign(response, "SECRET", {expiresIn: '72h'});
        return res.status(200).json({token: accessToken});
      }else{
        return res.status(400).json({message: "Algo salió mal. Por favor intente más tarde"});
      }
    }else{
      return res.status(500).json({message: "Usuario no existe"});
    }
  }catch(err){
    return res.status(500).json(err);
  }
}
```

La otra función es “getAll”, la cual trae todos los usuarios registrados en la base de datos. Finalmente, se encuentra la función “add”, la cual registra un nuevo usuario dependiendo si el correo ya existe o no, por cuanto este debe ser único. La implementación de estas dos funciones se muestra en la siguiente imagen.

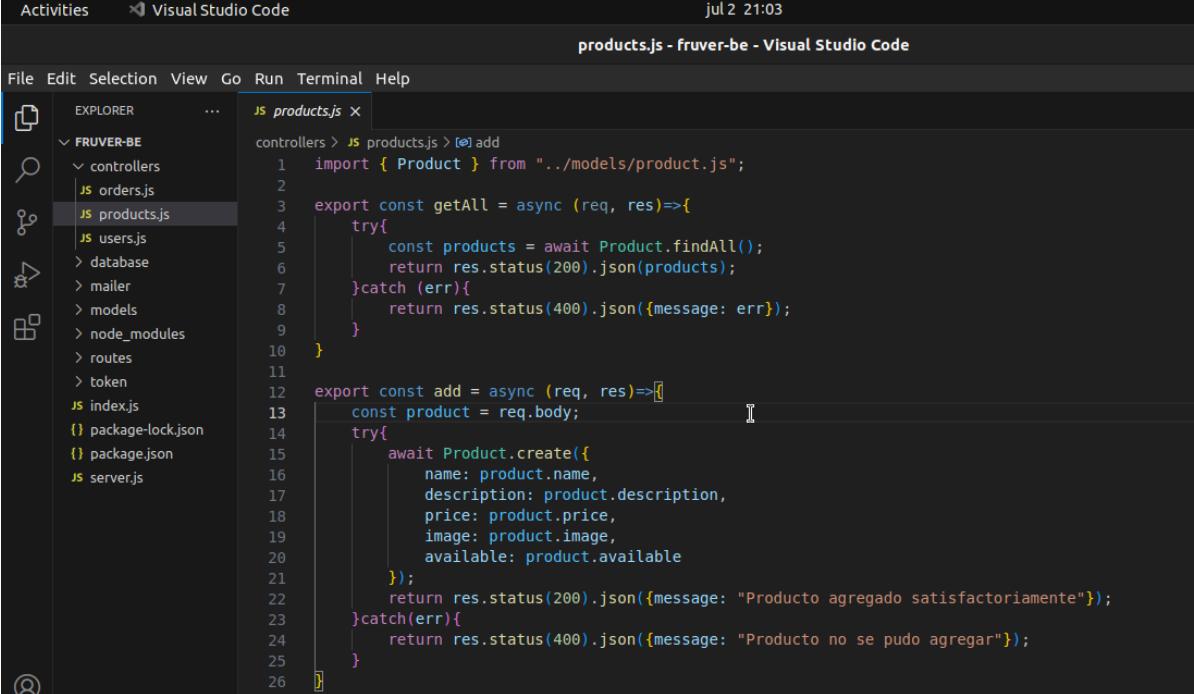


The screenshot shows the Visual Studio Code interface with the 'users.js' file open in the editor. It includes two new functions: 'getAll', which retrieves all users from the database, and 'add', which creates a new user. The 'add' function checks if a user with the same email already exists. If not, it creates a new user with the provided information. If an error occurs during creation, it returns an error message. If the user already exists, it returns a message indicating that an email must be unique.

```
export const getAll = async (req, res) => {
  try{
    const users = await User.findAll();
    res.status(200).json(users);
  }catch (err){
    res.status(400).json({message: err});
  }
}

export const add = async (req, res) => {
  const user = req.body;
  try{
    const userFound = await User.findOne({
      where: [
        { email: user.email }
      ]
    });
    if (!userFound){
      try{
        await User.create({
          userName: user.userName,
          phone: user.phone,
          email: user.email,
          password: user.password,
          role: user.role
        });
        return res.status(200).json({message: "Usuario registrado satisfactoriamente"});
      }catch(err){
        return res.status(400).json({message: "Error al registrar un usuario"});
      }
    }else{
      return res.status(400).json({message: "Un usuario con ese email ya existe"});
    }
  }catch(err){
    return res.status(400).json({message: err});
  }
}
```

El otro archivo es **products.js**, el cual cuenta con 5 funciones. La primera de ellas es la función “**getAll**”, la cual obtiene todos los productos registrados hasta el momento. La segunda función es “**add**”, la cual registra un nuevo producto con sus respectivos valores como se muestra en la siguiente imagen.



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

- Title Bar:** Activities > Visual Studio Code | jul 2 21:03 | products.js - fruver-be - Visual Studio Code
- File Menu:** File Edit Selection View Go Run Terminal Help
- Explorer:** Shows the project structure under "FRUVER-BE": controllers (orders.js, products.js), database, mailer, models, node_modules, routes, token, index.js, package-lock.json, package.json, server.js. "products.js" is selected.
- Code Editor:** The "products.js" file is open, displaying the following code:

```
1 import { Product } from "../models/product.js";
2
3 export const getAll = async (req, res) => {
4     try{
5         const products = await Product.findAll();
6         return res.status(200).json(products);
7     }catch (err){
8         return res.status(400).json({message: err});
9     }
10
11 export const add = async (req, res) =>{
12     const product = req.body;
13     try{
14         await Product.create({
15             name: product.name,
16             description: product.description,
17             price: product.price,
18             image: product.image,
19             available: product.available
20         });
21         return res.status(200).json({message: "Producto agregado satisfactoriamente"});
22     }catch(err){
23         return res.status(400).json({message: "Producto no se pudo agregar"});
24     }
25 }
26
27 }
```

La tercera función es “**getById**”, la cual obtiene un producto dado su id. La cuarta función es **update**, la cual actualiza un producto dado su id. Finalmente, la quinta función es “**del**”, la cual elimina un determinado producto dado su id. Dichas funciones se muestran a continuación.

Activities Visual Studio Code jul 2 21:06

products.js - fruver-be - Visual Studio Code

```

File Edit Selection View Go Run Terminal Help
EXPLORER ... JS products.js x
FRUVER-BE
  controllers
    orders.js
  products.js
  users.js
  database
  mailer
  models
  node_modules
  routes
  token
  index.js
  package-lock.json
  package.json
  server.js

products.js
1 /
2 export const getById = async (req, res) =>{
3   const id = req.params.id;
4   try{
5     const product = await Product.findById(id);
6     return res.status(200).json(product);
7   }catch{
8     return res.status(400).json({message: "El producto con ese id no existe"});
9   }
10
11 export const update = async (req, res) =>{
12   const product = req.body;
13   try{
14     await Product.update({
15       name: product.name,
16       description: product.description,
17       price: product.price,
18       image: product.image,
19       available: product.available
20     },{
21       where: {
22         id: product.id
23       }
24     });
25     return res.status(200).json({message: "Producto actualizado satisfactoriamente"});
26   }catch(err){
27     return res.status(400).json({message: "Producto no se pudo actualizar"});
28   }
29
30 export const del = async (req, res) =>{
31   const id = req.params.id;
32   try{
33     await Product.destroy({
34       where: {
35         id: id
36       }
37     });
38     return res.status(200).json({message: "Producto eliminado satisfactoriamente"});
39   }catch{
40     return res.status(400).json({message: "Producto no eliminado"});
41   }
42
43 }

```

OUTLINE

El último archivo es **orders.js**, el cual cuenta con 3 funciones. La primera de ellas es la función “getAll”, la cual obtiene todos los pedidos que no se han atendido hasta el momento. En dicha función se hace uso de ‘include’ (inner join en sql) para recuperar los datos relacionados al modelo Customer, al modelo ProductOrder y al modelo Product como se muestra a continuación.

Activities Visual Studio Code jul 2 21:11

orders.js - fruver-be - Visual Studio Code

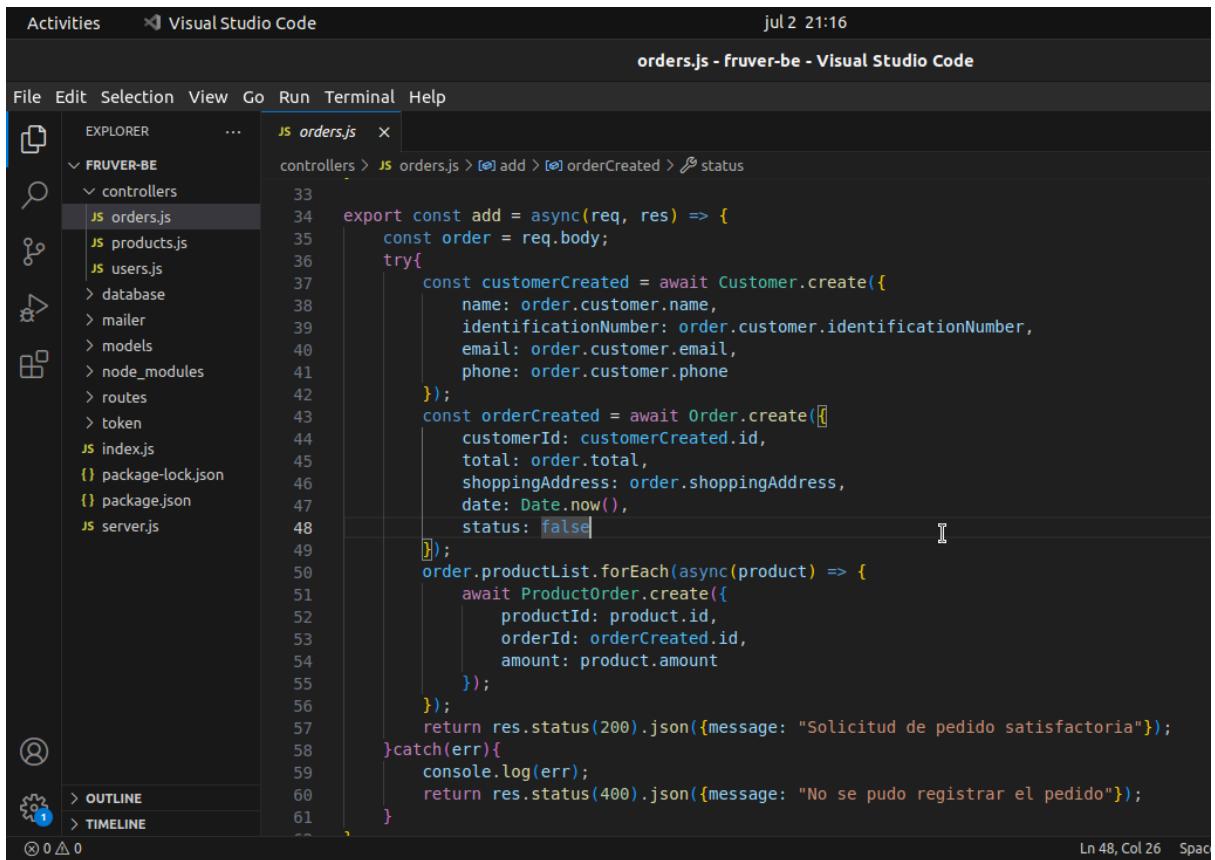
```

File Edit Selection View Go Run Terminal Help
EXPLORER ... JS orders.js x
FRUVER-BE
  controllers
    orders.js
  products.js
  users.js
  database
  mailer
  models
  node_modules
  routes
  token
  index.js
  package-lock.json
  package.json
  server.js

orders.js
1 import { Order } from "../models/order.js";
2 import { Customer } from "../models/customer.js";
3 import { ProductOrder } from "../models/product_order.js";
4 import { Product } from "../models/product.js";
5 import { transporter } from "../mailer/mailer.js";
6
7 export const getAll = async (req, res) =>{
8   try{
9     const orders = await Order.findAll({
10       attributes: ['id', 'total', 'shoppingAddress', 'date', 'status'],
11       where: {status: false},
12       include: [
13         {
14           model: Customer,
15           required: true,
16         },
17         {
18           model: ProductOrder,
19           attributes: ['id', 'amount'],
20           required: true,
21           include: [
22             {
23               model: Product,
24               required: true
25             }
26           ]
27         }
28       ],
29     );
30     return res.status(200).json(orders);
31   }catch (err){
32     return res.status(400).json({message: err});
33   }
34
35 }

```

La segunda función es “add”, la cual se encarga de registrar la solicitud de un pedido por parte de un cliente, por tal razón, lo primero que se hace es registrar el cliente, luego con el id de ese cliente se registra un pedido, y finalmente, con el id de ese pedido se registra en la tabla product_order el detalle de ese pedido, que serían todos los productos y sus respectivas cantidades a comprar. La implementación de dicha función se muestra a continuación.



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

- Title Bar:** Activities > Visual Studio Code | jul 2 21:16 | orders.js - fruver-be - Visual Studio Code
- File Menu:** File Edit Selection View Go Run Terminal Help
- Explorer:** Shows the project structure under "FRUVER-BE": controllers (orders.js, products.js, users.js), database, mailer, models, node_modules, routes, token, index.js, package-lock.json, package.json, server.js.
- Editor:** The "orders.js" file is open, showing the implementation of the "add" function. The code uses async/await to handle promises for creating a customer, an order, and individual order items.

```
33  export const add = async(req, res) => {
34    const order = req.body;
35    try{
36      const customerCreated = await Customer.create({
37        name: order.customer.name,
38        identificationNumber: order.customer.identificationNumber,
39        email: order.customer.email,
40        phone: order.customer.phone
41      });
42      const orderCreated = await Order.create([
43        customerId: customerCreated.id,
44        total: order.total,
45        shoppingAddress: order.shoppingAddress,
46        date: Date.now(),
47        status: false
48      ]);
49      order.productList.forEach(async(product) => {
50        await ProductOrder.create({
51          productId: product.id,
52          orderId: orderCreated.id,
53          amount: product.amount
54        });
55      });
56      return res.status(200).json({message: "Solicitud de pedido satisfactoria"});
57    }catch(err){
58      console.log(err);
59      return res.status(400).json({message: "No se pudo registrar el pedido"});
60    }
61  }
```

- Status Bar:** Ln 48, Col 26 Spac

Finalmente, la quinta función es “dispatchOrder”, la cual atiende la solicitud de compra de un cliente. Esta función recibe el id del pedido, donde hace una consulta para traer todo el pedido, incluyendo el cliente y el detalle de ese pedido (productos y su cantidad). Después actualiza el estado del pedido a true, lo que quiere decir que dicha solicitud ya fue atendida, para finalmente notificar al cliente, mediante un correo electrónico, que su solicitud ya fue atendida y que el pedido fue despachado (con su respectivo detalle), utilizando nodemailer como se muestra a continuación.

Activities Visual Studio Code jul 2 21:24

orders.js - fruver-be - Visual Studio Code

File Edit Selection View Go Run Terminal Help

```

EXPLORER js orders.js
FRUVER-BE
  controllers
    js orders.js
    js products.js
    js users.js
  database
  mailer
  models
  node_modules
  routes
  token
  index.js
  package-lock.json
  package.json
  server.js

62  }
63
64 export const dispatchOrder = async(req, res)=>{
65   const id = req.body.id;
66   try{
67     const order = await Order.findOne({
68       attributes: ['id', 'total', 'shoppingAddress', 'date'],
69       where: {
70         id: id
71       },
72       include: [
73         {
74           model: Customer,
75           required: true,
76         },
77         {
78           model: ProductOrder,
79           attributes: ['id', 'amount'],
80           required: true,
81           include: [
82             {
83               model: Product,
84               required: true
85             }
86           ]
87         }
88       ],
89     });
90     await Order.update({
91       status: true
92     },
93     {
94       where: {
95         id: id
96       }
97     });
98   let products = "";
99
100   order.product_orders.forEach(productOrder => {
101     products += `<tr>
102       <td style="padding:2px; text-align:center; border: lpx solid #000;">${productOrder.product.name}</td>
103       <td style="padding:2px; text-align:center; border: lpx solid #000;">${productOrder.amount}</td>
104       <td style="padding:2px; text-align:center; border: lpx solid #000;">${productOrder.product.price}</td>
105     </tr>
`);
```

Formato del detalle y envío del email:

Activities Visual Studio Code jul 2 21:24

orders.js - fruver-be - Visual Studio Code

File Edit Selection View Go Run Terminal Help

```

EXPLORER js orders.js
FRUVER-BE
  controllers
    js orders.js
    js products.js
    js users.js
  database
  mailer
  models
  node_modules
  routes
  token
  index.js
  package-lock.json
  package.json
  server.js

107   await transporter.sendMail([
108     from: "Pedido Envíado<fruverSanMarco@gmail.com>",
109     to: "reveltojuan8@gmail.com",
110     subject: "Pedido Envíado",
111     html:
112       <div style="width:100%; margin-bottom: 10px;">
113         <h3 style="margin-bottom: 5px; color: black;">Detalles del pedido</h3>
114         <table style="width:100%; color: black; margin-bottom: 10px; border-collapse: collapse">
115           <tr>
116             <td style="width: 300px;">
117               <strong style="margin-right: 4px">Cliente:</strong> ${order.customer.name}
118             </td>
119             <td>
120               <strong style="margin-right: 4px">Número de identificación:</strong> ${order.customer.identificationNumber}
121             </td>
122           </tr>
123           <tr>
124             <td style="width: 300px;">
125               <strong style="margin-right: 4px">Email:</strong> ${order.customer.email}
126             </td>
127             <td>
128               <strong style="margin-right: 4px">Número de celular:</strong> ${order.customer.phone}
129             </td>
130           </tr>
131           <tr>
132             <td style="width: 300px;">
133               <strong style="margin-right: 4px">Fecha de solicitud:</strong> ${order.date}
134             </td>
135             <td>
136               <strong>Dirección de envío:</strong> ${order.shoppingAddress}
137             </td>
138           </tr>
139         </table>
140         <table style="width:60%; border-collapse: collapse; border: 1px solid #000; color: black; text-align:center">
141           <thead>
142             <tr>
143               <th style="padding:2px; text-align:center; border: lpx solid #000;">Productos</th>
144               <th style="padding:2px; text-align:center; border: lpx solid #000;">Cantidad</th>
145               <th style="padding:2px; text-align:center; border: lpx solid #000;">Precio unitario</th>
146               <th style="padding:2px; text-align:center; border: lpx solid #000;">Subtotal</th>
147             </tr>
148           </thead>
149           <tbody>
```

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

- Title Bar:** Activities, Brave Web Browser, jul 2 21:24, orders.js - fruver-be - Visual Studio Code
- Menu Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Explorer View:** Shows the project structure under "FRUVER-BE": controllers (orders.js, products.js, users.js), database, mailer, models, routes, token, index.js, package-lock.json, package.json, server.js.
- Code Editor:** The "orders.js" file is open. The code defines a controller for handling orders. It includes a template for displaying order details (customer email, phone, date, shipping address) and a table for listing products with columns for product name, quantity, unit price, and subtotal. It also handles the total purchase amount and sends a thank you message to the customer.
- Bottom Status Bar:** OUTLINE, TIMELINE, 0 △ 0

```
controllers > JS orders.js > (e) dispatchOrder
122     </td>
123     </tr>
124     <tr>
125         <td style="width: 300px;">
126             <strong style="margin-right: 4px">Email:</strong> ${order.customer.email}
127         </td>
128         <td>
129             <strong style="margin-right: 4px">Número de celular:</strong> ${order.customer.phone}
130         </td>
131     </tr>
132     <tr>
133         <td style="width: 300px;">
134             <strong style="margin-right: 4px">Fecha de solicitud:</strong> ${order.date}
135         </td>
136         <td>
137             <strong>Dirección de envío:</strong> ${order.shoppingAddress}
138         </td>
139     </tr>
140 </table>
141 <table style="width:60%; border-collapse: collapse; border: 1px solid #000; color: black; text-align:center">
142     <thead>
143         <tr>
144             <th style="padding:2px; text-align:center; border: 1px solid #000;">Producto</th>
145             <th style="padding:2px; text-align:center; border: 1px solid #000;">Cantidad</th>
146             <th style="padding:2px; text-align:center; border: 1px solid #000;">Precio unitario</th>
147             <th style="padding:2px; text-align:center; border: 1px solid #000;">Subtotal</th>
148         </tr>
149     </thead>
150     <tbody>
151         ${products}
152     </tbody>
153 </table>
154 <div style="color: black;">
155     <p><strong>Total Compra:</strong> ${order.total}</p>
156 </div>
157 <div style="color: black;">
158     <p>Muchas Gracias por confiar en <strong>FRUVER SAN MARCO</strong></p>
159 </div>
160     });
161     return res.status(200).json({message: "Pedido consolidado y enviado exitosamente: Se notificó al cliente el envío de su pedido"});
162 }catch(err){
163     return res.status(400).json({message: "Pedido no se pudo despachar"});
164 }
165 }
```

Una vez lista la definición de los controladores, se define la estructura para organizar y gestionar las rutas de la aplicación. Como se mencionó en un inicio, para el presente proyecto se contará con tres archivos: uno para los usuarios, otro para los productos y otro para los pedidos.

En el archivo **users.js** dentro del archivo routes, se definen tres rutas utilizando los respectivos verbos Http. La primera de ellas utiliza el verbo post para loguearse y no cuenta con autenticación. La segunda utiliza el verbo post para registrar un nuevo usuario y no necesita autenticación. La última utiliza el verbo get para obtener todos los usuarios y necesita autenticación, haciendo uso de la función verifyToken como se explicó anteriormente. La definición de estas rutas se muestra en la siguiente imagen.

The screenshot shows the Visual Studio Code interface. The title bar reads "Activities" and "Visual Studio Code" with the date "jul 2 21:40". The tab bar shows "users.js - fruver-be - Visual Studio Code". The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The Explorer sidebar on the left shows a project structure under "FRUVER-BE": controllers, database, mailer, models, node_modules, routes (which contains orders.js, products.js, and users.js), token, index.js, package-lock.json, package.json, and server.js. The main editor area displays the "users.js" file:

```
1 import express from 'express';
2 import { verifyToken } from '../token/token.js';
3 import { login, getAll, add } from '../controllers/users.js';
4
5 export const userRoutes = express.Router();
6
7 userRoutes.post('/login', login);
8
9 userRoutes.post('/signup', add);
10
11 userRoutes.get('/', verifyToken, getAll);
12
```

Por otro lado, en el archivo **products.js** dentro del archivo routes, se definen cinco rutas utilizando los respectivos verbos Http. La primera de ellas utiliza el verbo get para obtener todos los productos y no necesita autenticación. La segunda utiliza el verbo post para registrar un nuevo producto y necesita autenticación. La tercera utiliza el verbo get para obtener un respectivo producto según su id, y necesita autenticación. La cuarta utiliza el verbo put para actualizar un respectivo producto según su id, y necesita autenticación. La última utiliza el verbo delete para eliminar un respectivo producto según su id, y necesita autenticación. Las cuatro últimas necesitan autenticación porque sólo un administrador puede gestionar los productos, y la primera no, porque los clientes podrán visualizar los productos para poder comprarlos. La definición de estas rutas se muestra en la siguiente imagen.

The screenshot shows the Visual Studio Code interface with the title bar "Activities" and "Visual Studio Code" and the date "jul 2 21:47". The active tab is "products.js". The left sidebar shows a project structure under "FRUVER-BE" with files like controllers, database, mailer, models, node_modules, routes, users.js, token, index.js, package-lock.json, package.json, and server.js. The "routes" folder is expanded, and "orders.js" is selected. The main editor area displays the following code:

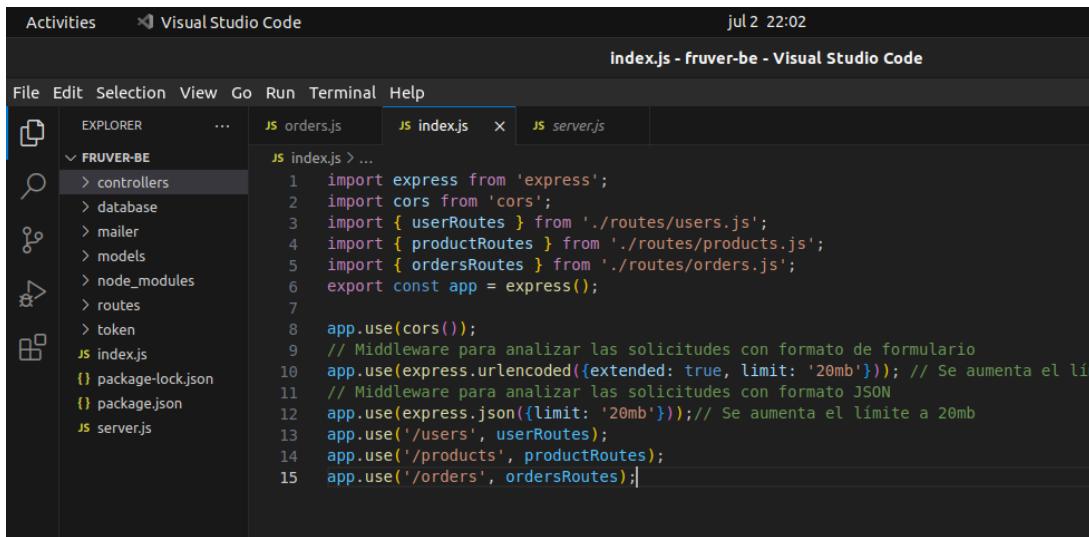
```
1 import express from 'express';
2 import { getAll, add, update, getById, del } from '../controllers/products.js';
3 import { verifyToken } from '../token/token.js';
4
5 export const productRoutes = express.Router();
6
7 productRoutes.get("/", getAll);
8
9 productRoutes.post("/", verifyToken, add);
10
11 productRoutes.get("/getById/:id", verifyToken, getById);
12
13 productRoutes.put("/", verifyToken, update);
14
15 productRoutes.delete("/:id", verifyToken, del);
16
```

Finalmente, en el archivo **orders.js** dentro del archivo routes, se definen tres rutas utilizando los respectivos verbos Http. La primera de ellas utiliza el verbo get para obtener todos los pedidos que no se han atendido hasta el momento, y no cuenta con autenticación. La segunda utiliza el verbo post para registrar una nueva solicitud de compra de un respectivo cliente y no necesita autenticación. La última utiliza el verbo put para consolidar y actualizar el estado de un pedido, notificando al correo electrónico del cliente el envío del mismo, y necesita autenticación. La primera ruta y la tercera necesitan autenticación, ya que solo el administrador podrá gestionar los pedidos. La definición de estas rutas se muestra en la siguiente imagen.

The screenshot shows the Visual Studio Code interface with the title bar "Activities" and "Visual Studio Code" and the date "jul 2 21:53". The active tab is "orders.js". The left sidebar shows a project structure under "FRUVER-BE" with files like controllers, products.js, users.js, database, mailer, models, node_modules, routes, products.js, users.js, token, index.js, package-lock.json, package.json, and server.js. The "routes" folder is expanded, and "orders.js" is selected. The main editor area displays the following code:

```
1 import express from 'express';
2 import { getAll, add, dispatchOrder } from '../controllers/orders.js';
3 import { verifyToken } from '../token/token.js';
4
5 export const ordersRoutes = express.Router();
6
7 ordersRoutes.get("/", verifyToken, getAll);
8
9 ordersRoutes.post("/", add);
10
11 ordersRoutes.put("/", verifyToken, dispatchOrder);
```

Una vez realizadas las configuraciones anteriores, se actualiza el archivo **index.js** con el middleware para las rutas de los pedidos, como se muestra a continuación.

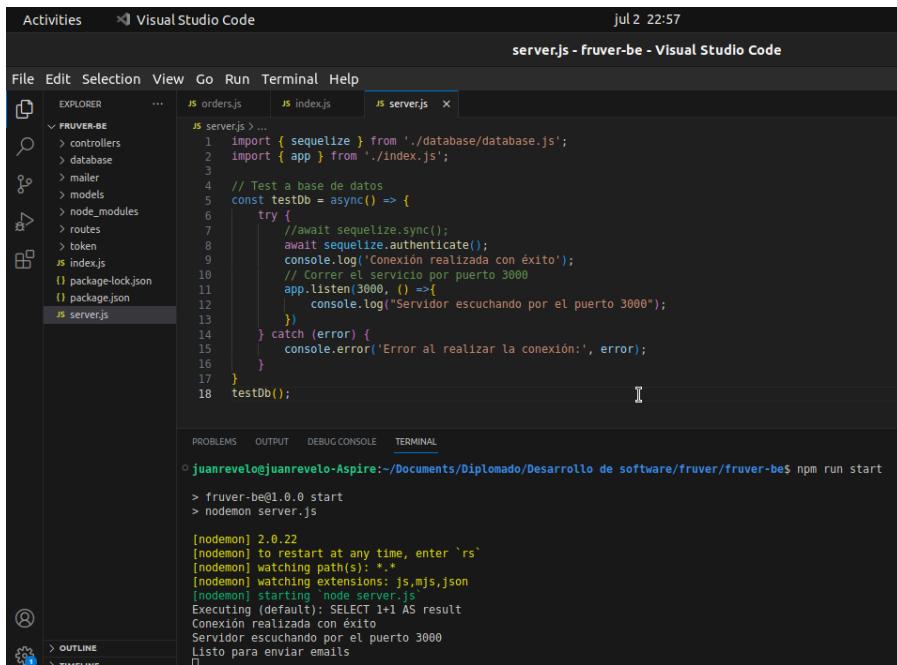


The screenshot shows the Visual Studio Code interface with the title bar "Activities" and "Visual Studio Code" and the date "jul 2 22:02". The tab bar shows "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". The active tab is "index.js". The code editor displays the following content:

```
index.js - fruver-be - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER ... JS orders.js JS index.js JS server.js
JS index.js > ...
1 import express from 'express';
2 import cors from 'cors';
3 import { userRoutes } from './routes/users.js';
4 import { productRoutes } from './routes/products.js';
5 import { ordersRoutes } from './routes/orders.js';
6 export const app = express();
7
8 app.use(cors());
9 // Middleware para analizar las solicitudes con formato de formulario
10 app.use(express.urlencoded({extended: true, limit: '20mb'})); // Se aumenta el lím
11 // Middleware para analizar las solicitudes con formato JSON
12 app.use(express.json({limit: '20mb'})); // Se aumenta el límite a 20mb
13 app.use('/users', userRoutes);
14 app.use('/products', productRoutes);
15 app.use('/orders', ordersRoutes);
```

Finalmente, el aplicativo está listo para ser iniciado. Tener en cuenta que para descargar todos los módulos de los cuales hace uso el aplicativo, se debe ejecutar inicialmente el comando `npm install`.

Con el comando `npm run start` se inicia el aplicativo, como se muestra a continuación.



The screenshot shows the Visual Studio Code interface with the title bar "Activities" and "Visual Studio Code" and the date "jul 2 22:57". The tab bar shows "File", "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help". The active tab is "server.js". The code editor displays the following content:

```
server.js - fruver-be - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER ... JS orders.js JS index.js JS server.js
JS server.js > ...
1 import { Sequelize } from './database/database.js';
2 import { app } from './index.js';
3
4 // Test a base de datos
5 const testDb = async () => {
6   try {
7     //await sequelize.sync();
8     await sequelize.authenticate();
9     console.log('Conexión realizada con éxito');
10    // Correr el servicio por puerto 3000
11    app.listen(3000, () =>{
12      console.log("Servidor escuchando por el puerto 3000");
13    });
14  } catch (error) {
15    console.error('Error al realizar la conexión:', error);
16  }
17}
18 testDb();
```

The terminal below shows the command being run and its output:

```
juanrevelo@juanrevelo-Aspire:~/Documents/Diplomado/Desarrollo de software/fruver-be$ npm run start
> fruver-be@1.0.0 start
> nodemon server.js
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *
[nodemon] watching extensions: js,mjs,json
[nodemon] starting 'node server.js'
Executing (default): SELECT 1+1 AS result
Conexión realizada con éxito
Servidor escuchando por el puerto 3000
Listo para enviar emails
```

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

Una vez iniciado el aplicativo, se utiliza el cliente Postman para realizar solicitudes y probar el aplicativo construido.

Desde postman se tiene la siguiente estructura para probar los endpoints construidos.

The screenshot shows the Postman application interface. At the top, there's a header bar with 'Activities' and 'Postman' logo, the date 'jul 2 22:54', and a search bar 'Search Postman'. Below the header is a navigation bar with 'File', 'Edit', 'View', 'Help', and links for 'Home', 'Workspaces', 'API Network', 'Explore', and 'Invite'. The main area is titled 'Fruver - My Workspace'. On the left, there's a sidebar with 'Collections' (selected), 'Environments', and 'History'. The 'Collections' section shows a tree view of API endpoints under 'Fruver': 'Users' (GET GetUsers, POST login, POST PostUser), 'Products' (GET GetProducts, POST PostProduct, PUT PutProduct, DEL DeleteProduct, GET GetProductById), and 'Orders' (GET GetOrders, POST PostOrder, PUT DispatchOrder). The right side of the interface shows the 'Overview' tab for the 'Fruver' collection, which includes sections for 'Add collection description...', 'View complete collection documentation →', 'Created by You', and 'Created on 17 Jun 2023, 10:30 AM'.

- Prueba endpoints del usuario.

1. Registrar usuario

Como se puede observar en la siguiente imagen, se consumió el endpoint para registrar un nuevo usuario, y la solicitud fue exitosa.

The screenshot shows the Postman interface. On the left, the sidebar displays collections like 'Category', 'DocumentA', 'Fruber', and 'Products'. Under 'Fruber', there are 'Users' (with 'GetUsers', 'login', and 'PostUser'), 'Products' (with 'GetProducts', 'PostProduct', 'PutProduct', 'DeleteProduct', and 'GetProductByid'), and 'Orders' (with 'GetOrders', 'PostOrder', and 'DispatchOrder'). The main workspace shows a POST request to 'http://localhost:3000/users/signup'. The 'Body' tab contains JSON data:

```
1 "userName": "Pedro Gonzales",
2 "phone": "315676567",
3 "email": "pedro@gmail.com",
4 "password": "admin",
5 "role": "admin"
```

The response status is 200 OK, with the message: "Usuario registrado satisfactoriamente".

Y en la base de datos se puede visualizar el nuevo registro.

The screenshot shows the DBeaver interface connected to a database named 'user'. The left sidebar shows databases like 'Fruber', 'productos', and 'test'. The 'user' table is selected, displaying two rows of data:

id	user_name	phone	email	password	role
1	Juan Revelo	3147492926	admin@gmail.com	admin	admin
2	Pedro Gonzales	315676567	pedro@gmail.com	admin	admin

A continuación, se consume el mismo endpoint pero queriendo registrar un usuario con un email que ya existe.

The screenshot shows the Postman interface with a collection named "My Workspace". The "Users" folder contains a "PostUser" endpoint. The request method is POST, and the URL is `http://localhost:3000/users/signup`. The "Body" tab is selected, showing a JSON payload:

```
1 "userName": "David Sanchez",
2 ... "phone": "315676567",
3 ... "email": "pedro@gmail.com",
4 ... "password": "admin",
5 ... "role": "admin"
```

The response status is 400 Bad Request, and the message is "Un usuario con ese email ya existe".

Como es de esperar, un usuario con ese email ya existe, y por lo tanto no se puede registrar al nuevo usuario.

2. Iniciar sesión

Como se puede observar en la siguiente imagen, se consumió el endpoint para iniciar sesión en el aplicativo.

The screenshot shows the Postman interface with a collection named "My Workspace". The "Users" folder contains a "login" endpoint. The request method is POST, and the URL is `http://localhost:3000/users/login`. The "Body" tab is selected, showing a JSON payload:

```
1 ... "email": "admin@gmail.com",
2 ... "password": "admin"
```

The response status is 200 OK, and the token is displayed in the response body:

```
1 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
2 ey31bWFpbC161mFkbWluQ0dtyW1sLmNvbSisInJvbGUiO1JhZGlpbihIsInVzZXJOYW1lIjoiSnVhbIBSZXZibG81LCjpxYXQiOjE20DgzNTc3N
3 DAsImV4cCI6MTY4ODYxNjk0MH0.bUygUB8G213bg0m1DNaGP-uJe4TC_3tQWKLc1-5p9Cw"
```

El resultado es un token que va a permitir obtener la información del usuario y su respectiva autenticación.

A continuación, se intenta iniciar sesión con un usuario incorrecto.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Category', 'DocumentA', 'Fruber' (which has 'Users' and 'Products' sub-collections), and 'Orders'. In the main area, a request for 'POST /login' is selected. The URL is set to 'http://localhost:3000/users/login'. The 'Body' tab is active, showing JSON input:

```
1 ... "email": "admin@gmail.com",
2 ... "password": "admin2"
```

Below the body, the response status is shown as 'Status: 401 Unauthorized' with a 'Pretty' JSON view:

```
1 {
2   "message": "Correo y/o contraseña incorrectos"
3 }
```

Como es de esperar, la respuesta indica que el usuario y/o contraseña son incorrectos.

3. Obtener usuarios

Como se puede observar en la siguiente imagen, se consumió el endpoint para obtener todos los usuarios registrados hasta el momento. Dicho endpoint requiere autenticación, por tal razón, si se intenta realizar una petición y no se envía el respectivo token, se retornará una respuesta de no autorizado, como se muestra en la siguiente imagen.

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Category', 'DocumentA', and 'Fruber' which has sub-collections 'Users', 'Products', 'Orders', and 'Metrics'. The main area shows a request for 'GET /Fruber / Users / GetUsers' with a status of 401 Unauthorized. The 'Body' tab is selected, showing the response body: "No autorizado".

Para poder consumir dicho endpoint desde postman, se debe ingresar a la pestaña “Authorization” y seleccionar la opción Bearer Token, enviando el respectivo token como se muestra a continuación.

This screenshot of the Postman interface shows the 'Authorization' tab selected under the 'Headers' section for the 'GetUsers' request. The 'Type' dropdown is set to 'Bearer Token'. A tooltip explains that this is sensitive data and recommends using variables. The 'Token' field contains a long string of characters: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ... . Below the request, a small illustration of a rocket launching is visible.

Una vez ingresado el token, ya se puede consumir el endpoint de manera exitosa.

The screenshot shows the Postman application interface. The top navigation bar includes File, Edit, View, Help, Home, Workspaces, API Network, Explore, and a search bar labeled "Search Postman". On the left, there's a sidebar titled "My Workspace" with sections for Collections, Environments, History, and a main area for API requests. The main workspace shows a "POST PostUser" request and a "GET GetUsers" request. The "GET GetUsers" request is selected, showing its details. The URL is set to "http://localhost:3000/users". The "Params" tab indicates that tokens are automatically generated for authorization. The "Body" tab shows a JSON response with two user objects:

```
[{"id": 2, "userName": "Juan Revelo", "phone": "3147492926", "email": "admin@mail.com", "password": "admin", "role": "admin"}, {"id": 3, "userName": "Pedro Gonzales", "phone": "3156765657", "email": "pedro@gmail.com", "password": "admin"}]
```

- Prueba endpoints del producto.

1. Registrar producto

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para registrar un nuevo producto, y la solicitud fue exitosa.

Y en la base de datos se puede visualizar el nuevo registro.

The screenshot shows the DBBeaver interface with the following details:

- File Edit Navigate Search SQL Editor Database Window Help**
- Auto** tab selected in the top bar.
- productos** database selected in the left sidebar.
- product** table selected in the central pane.
- Properties Data ER Diagram** tabs are visible at the top of the central pane.
- Grid** view is active, showing one record:

#	name	description	price	image
1	Manzana	Manzana roja	1,300	data:image/jpeg;base64,/9j/4AAI

- Text** view is also present on the left.
- Record** view is at the bottom left.
- Bottom Bar Buttons:** Refresh, Save, Cancel, Export data, and others.
- Status Bar:** 1 row(s) fetched - 1ms, on 2023-07-02 at 23:30:43

Ingresamos un nuevo producto como se muestra en la siguiente imagen.

Y en la base de datos se puede visualizar el nuevo registro.

DBeaver 23.0.5 - product

File Edit Navigate Search SQL Editor Database Window Help

Database ... Projects

Enter a part of object name ...

productos - localhost:3306

fruver

Tables

customer 16K
order 32K
product 48K

Views
Indexes
Procedures
Triggers
Events

phpmyadmin
productos
test

Users
Administrator
System Info

Properties Data ER Diagram

product

Enter a SQL expression to filter results (use Ctrl+Space)

id	name	description	price	image
1	Manzana	Manzana roja	1,300	data:image/jpeg;base64,/9j/4AAf
2	Mango	Mango dulce	1,600	data:image/png;base64,iVBORw

Value X
Mango

Record

Text

Refresh Save Cancel Export data 200 2

2 row(s) fetched - 1ms, on 2023-07-02 at 23:35:34

COT en US:

Finalmente ingresamos otro producto.

PostProduct - My Workspace

File Edit View Help

← → Home Workspaces API Network Explore

Search Postman

POST PostProduct GET GetUsers

Fruver / Products / PostProduct

POST http://localhost:3000/products

Params Authorization Headers (10) Body Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON Cookies

Beautify

Body

```
1   "name": "Pera",
2   "description": "Pera madura",
3   "price": 1600,
4   "available": "true",
5   "image": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAZAAAAQCAMAAAC3Ycb+AAAABGdBTUEAALGPC/xhBQAAAFAzUkdCAK7OHOKAAAMAUExURQAAAPX0vXy6Pb16+Hpw7erPb37+rz2X+Q5+v730Llt712Vpkob/j480/w2vX06Pb267CtUhxZ0vz683thSXSSSH57K+rzq2YkcWPj48KyzBmZd+fmwO3u0cf0mMTEjYzIUXqdQvj48fz7+MfbmPlxJB0qH6eS2suY3i1RgeXL9TRXeP0/u1WdUNuzS7mv1N1ztqtMc8DLMwmdbkq4bdTyo1DzbEgaOzYluNm9yN8mz13FkPhycVtK8m2iaJ5uWSJR9Y5eLY46QSjo4IExKHUpJJjoA/DOAEKv20Kv-n-0m9Pb-Oc-0c-0CnMI-W8DfSutM4-20n-0-MHUY-24-D9tuvuonc7CEDn-EDML-A-08-217h-6AO-2mnuMa-Toddna-eford/
```

Pretty Raw Preview Visualize JSON

Body Cookies Headers (8) Test Results

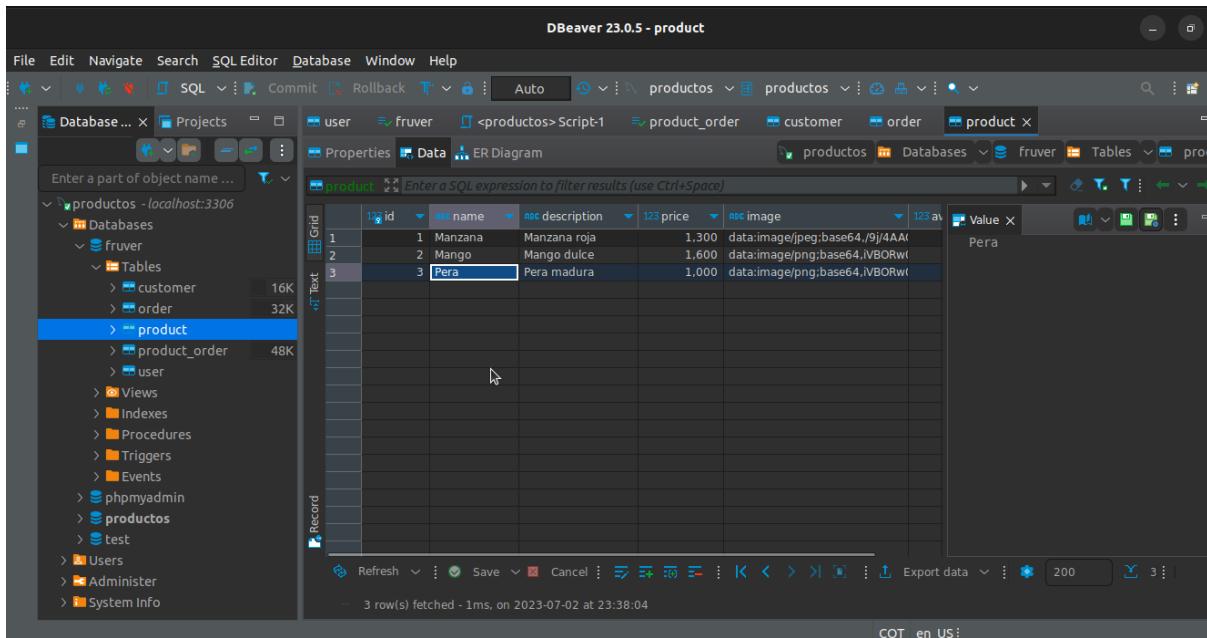
Status: 200 OK Time: 132 ms Size: 317 B Save as Example

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

1 "message": "Producto agregado satisfactoriamente"

Y en la base de datos se puede visualizar el nuevo registro.



2. Actualizar producto

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para actualizar un producto, y la solicitud fue exitosa.

File Edit View Help

← → Home Workspaces API Network Explore

Search Postman

My Workspace New Import

POST PostProduct PUT PutProduct + ...

No Environment

Save

Fruver / Products / PutProduct

PUT http://localhost:3000/products

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Send Cookies Beautify

1 ... "id": 3,
2 ... "name": "Peza Modificada",
3 ... "description": "Peza madura modificada",
4 ... "price": 1000,
5 ... "available": "true",
6 ... "image": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAZAAAGOCAMAAAC3Ycb+AAAABGdBTUEAALGPC/xhRQAAAFzIkUkC7OKh0KAAA4MAufxIRgAAAPX06vXyPh16+npw97erPb37+zr2X+0S+v130L1t712vpKob/j4B0/w2vX06Pb267cTUhxZov2683thSXSSSH57K+rzqYKCPj48KyvZmZd+3...+03u6gc10mMTejYzzUXqdQy4d8Ef7+MEbpnLx33Bo(HgesZSu31RGexKLStrXCeP0/

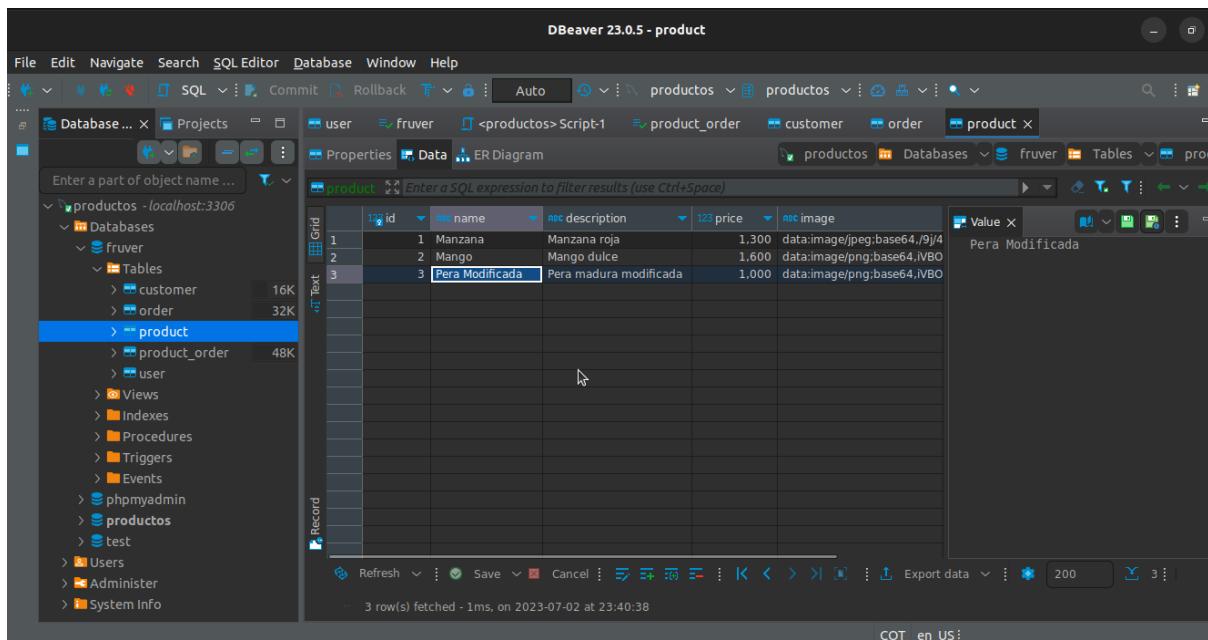
Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 114 ms Size: 320 B Save as Example

Message: "Producto actualizado satisfactoriamente"

Y en la base de datos se puede visualizar el registro actualizado.



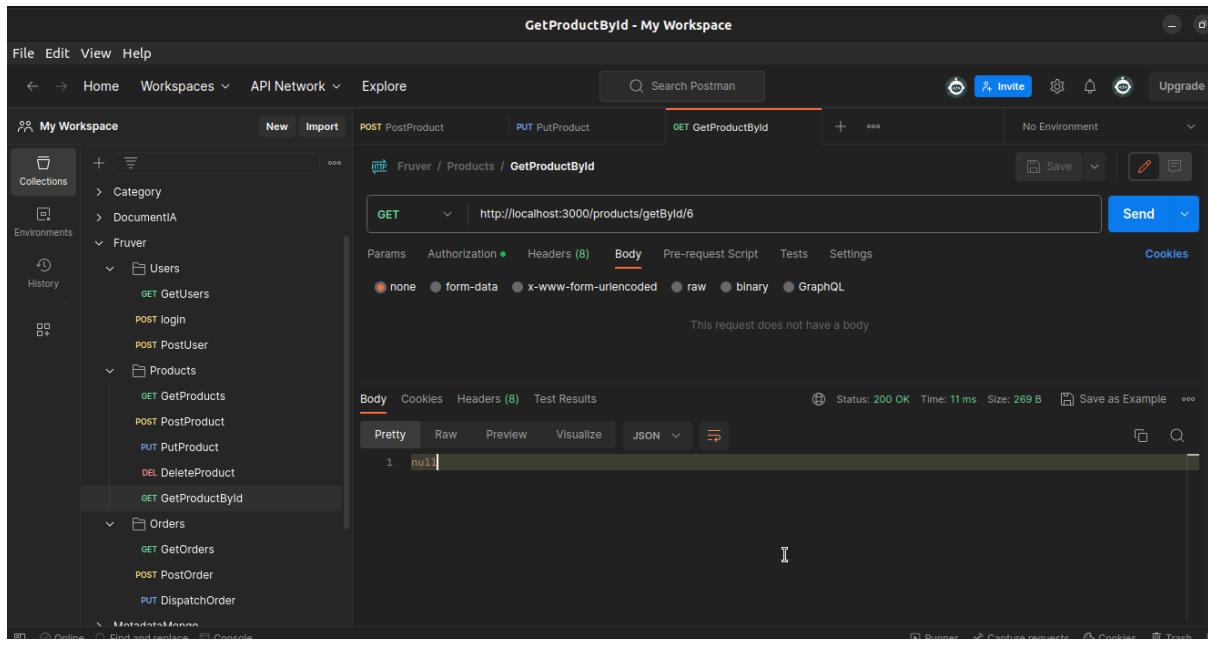
3. Obtener producto por su id

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para obtener un producto por su id, y la solicitud fue exitosa.

The screenshot shows the Postman interface with the following details:

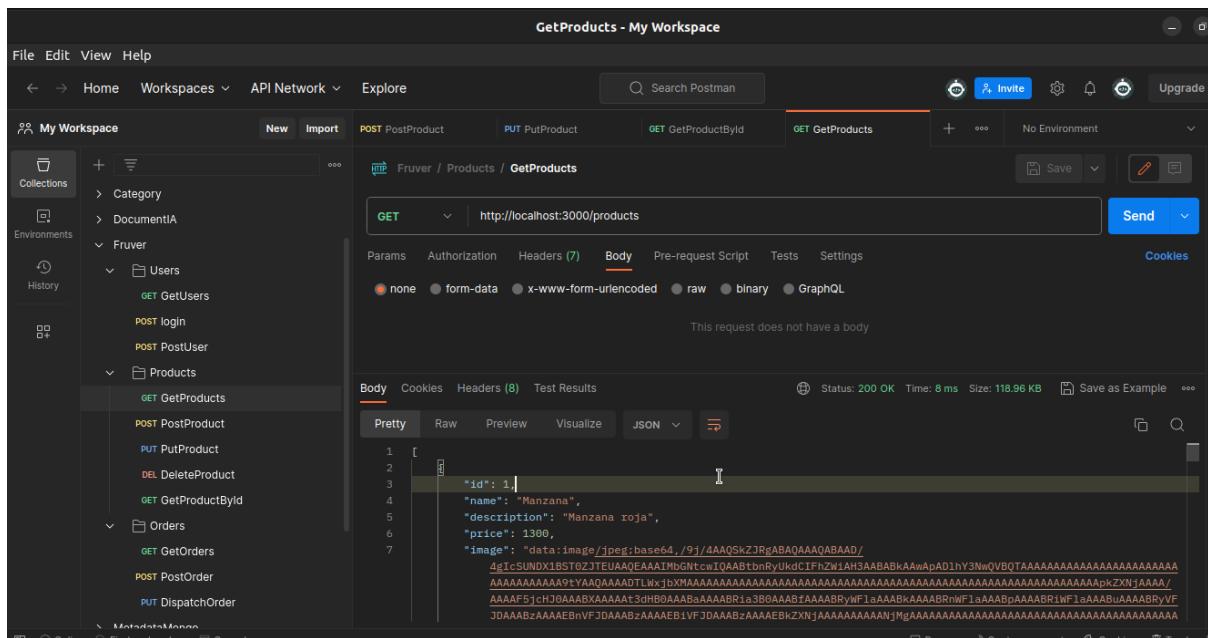
- File Edit View Help**
- Home Workspaces API Network Explore**
- Search Postman**
- Invite**, **Help**, **Upgrade**
- Collections**: My Workspace
- Environments**
- History**
- +**, **Category**, **DocumentA**, **Fruger** (selected), **Users**:
 - GET GetUsers**
 - POST login**
 - POST PostUser**
- Products**:
 - GET GetProducts**
 - POST PostProduct**
 - PUT PutProduct**
 - DELETE DeleteProduct**
 - GET GetProductById** (selected)
 - GET GetOrders**
 - POST PostOrder**
 - PUT DispatchOrder**
- Middle Panel**:
 - Method: GET**, **URL: http://localhost:3000/products/getById/1**
 - Params**, **Authorization**, **Headers (8)**, **Body** (selected), **Pre-request Script**, **Tests**, **Settings**
 - Body Options**: none, form-data, x-www-form-urlencoded, raw (selected), binary, GraphQL
 - Body Content**: This request does not have a body.
- Bottom Panel**:
 - Body**, **Cookies**, **Headers (8)**, **Test Results**
 - Status**: 200 OK, **Time**: 37 ms, **Size**: 18.22 KB
 - Save as Example**
 - Pretty**, **Raw**, **Preview**, **Visualize**, **JSON**
 - Code Snippet** (dropdown menu) and **Copy**, **Find** icons.

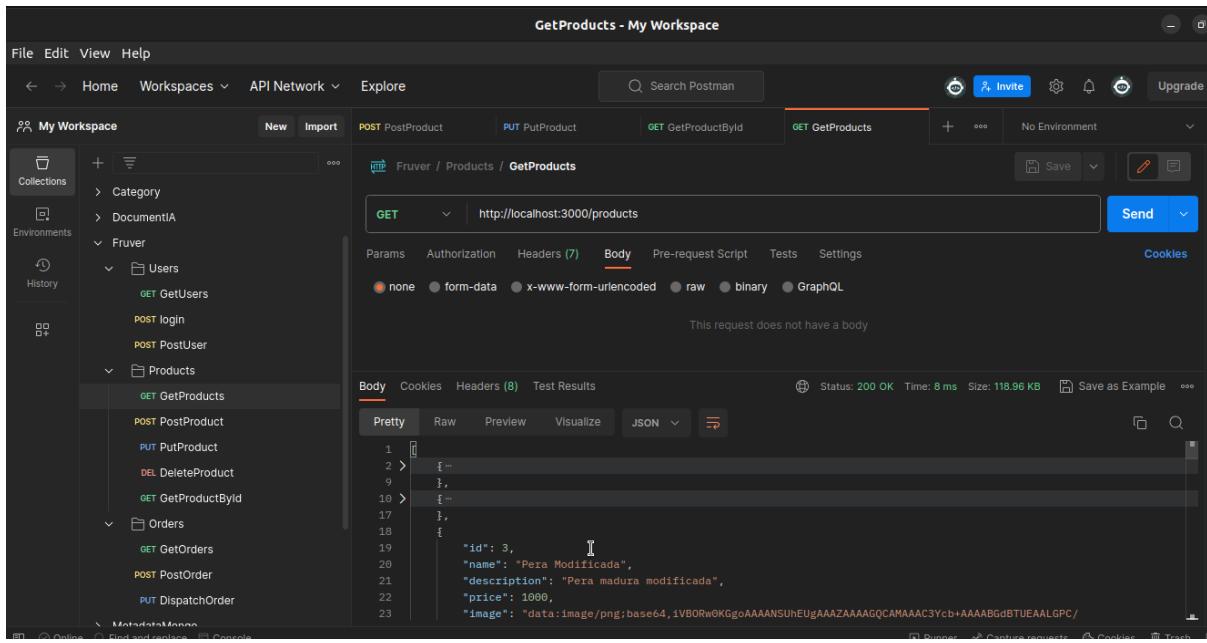
Si el id del producto no existe, se obtiene un null como resultado.



4. Obtener productos

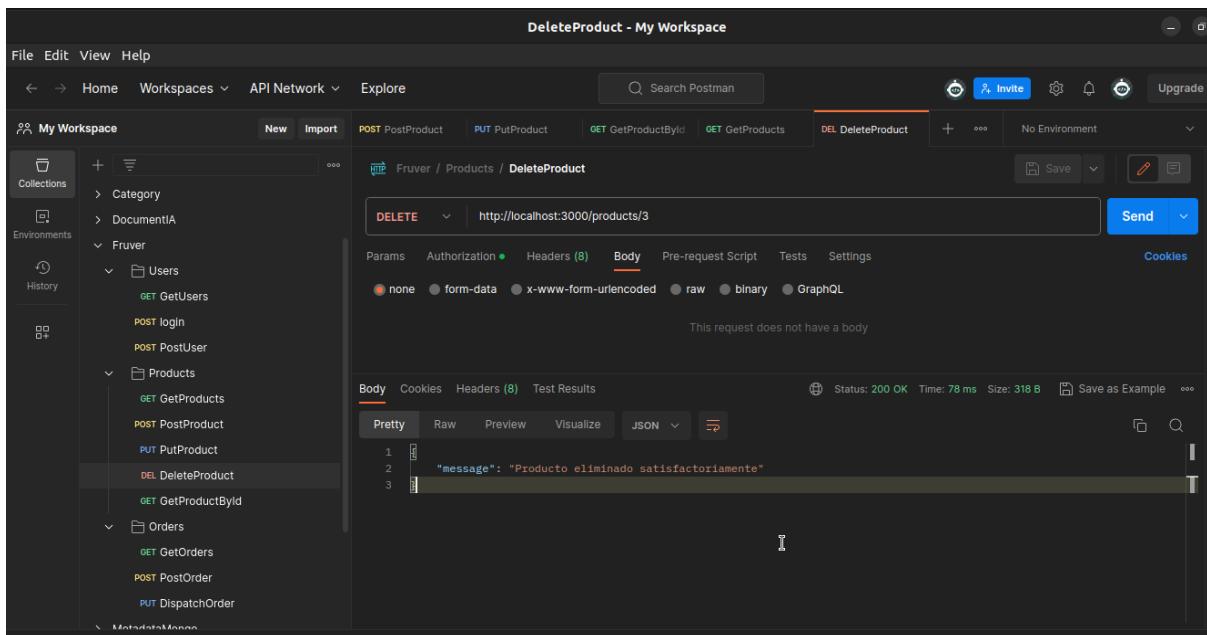
Para consumir dicho endpoint no se debe estar autenticado. Como se puede observar en la siguiente imagen, se consumió el endpoint para obtener todos los productos registrados hasta el momento.





4. Eliminar producto

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para eliminar un producto por su id, y la solicitud fue exitosa.



Y en la base de datos se puede visualizar que el registro ha sido eliminado con éxito.

DBeaver 23.0.5 - product

File Edit Navigate Search SQL Editor Database Window Help

Enter a part of object name ...

... Database X Projects X user X fruver X <productos> Script-1 X product_order X customer X order X product X

Properties Data ER Diagram

Grid

	id	name	description	price	image
1	1	Manzana	Manzana roja	1,300	data:image/jpeg;base64,/9j/4
2	2	Mango	Mango dulce	1,600	data:image/png;base64,IVBO

Value X
Manzana

Text

Record

Refresh Save Cancel Export data 200 2

2 row(s) fetched - 0ms, on 2023-07-02 at 23:52:51

COT en US:

- Prueba endpoints de los pedidos.

1. Registrar un pedido

Para consumir dicho endpoint no se debe estar autenticado. Como se puede observar en la siguiente imagen, se consumió el endpoint para registrar una solicitud de compra, la cual fue exitosa.

PostOrder - My Workspace

File Edit View Help

← → Home Workspaces API Network Explore

POST PostProduct PUT PutProduct GET GetProduct GET GetProducts DEL DeleteProduct POST PostOrder + No Environment

My Workspace New Import Fruver / Orders / PostOrder

POST http://localhost:3000/orders

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies

Body

```

2   "total": 2900,
3   "shoppingAddress": "Calle 18D #8A E Lorenzo",
4   "status": false,
5   "customer": {
6     "name": "Juan José Revelo",
7     "identificationNumber": "194543454",
8     "email": "revelojuan8@gmail.com",
9     "phone": "3145676787"
10   },
11   "productList": [
12     { "id": 1, "amount": 1 },
13     { "id": 2, "amount": 1 }

```

Send

Body Cookies Headers (8) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 150 ms Size: 314 B Save as Example

Message: "Solicitud de pedido satisfactoria"

Y en la base de datos se puede visualizar el nuevo registro.

Para el cliente.

The screenshot shows the DBeaver interface with the database 'products' selected. The left sidebar shows tables like 'customer', 'order', 'product', and 'product_order'. The main panel displays the 'customer' table with one row:

id	name	identificationNumber	email	phone
1	Juan José Revel	1194543454	revelojuan88gr	3145676787

Para el pedido.

The screenshot shows the DBeaver interface with the database 'products' selected. The left sidebar shows tables like 'customer', 'order', 'product', and 'product_order'. The main panel displays the 'order' table with one row:

id	total	shoppingAddress	date	status	customerId
1	2,900	Calle 18D #8A E Lorenzo	2023-07-02	0	1

y para el detalle del pedido.

DBeaver 23.0.5 - produktos

File Edit Navigate Search SQL Editor Database Window Help

Database Projects ...

Enter a part of object name ...

Databases

Fruver

Tables

customer

order

product

product_order

user

Views

Indexes

Procedures

Triggers

Events

phpmyadmin

productos

test

Users

Administrator

System Info

Properties Data ER Diagram

product_order

Enter a SQL expression to filter results (use Ctrl+Space)

id	amount	productId	orderId
1	1	1	1
2	1	1	2

Value

1

Grid Record

Refresh Save Cancel Export data 200 2

2 row(s) fetched - 1ms, on 2023-07-03 at 00:00:17

COT en US:

2. Obtener pedidos

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para obtener todos los pedidos que aún no han sido atendidos por un administrador, y la solicitud fue exitosa.

GetOrders - My Workspace

File Edit View Help

← → Home Workspaces API Network Explore

My Workspace New Import

POST PostProduct PUT PutProduct GET GetProductByld GET GetProducts DB DeleteProduct POST PostOrder GET GetOrders

HTTP Fruver / Orders / GetOrders

GET http://localhost:3000/orders

Params Authorization Headers (8) Body

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Body Cookies Headers (8) Test Results

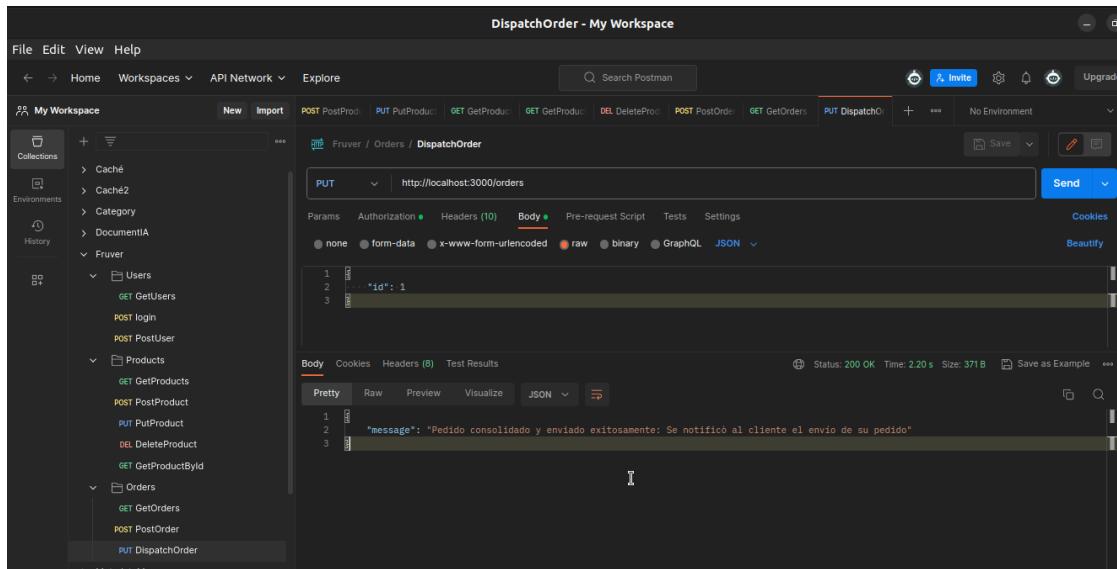
Pretty Raw Preview Visualize JSON

3	"id": 1,
4	"total": 2000,
5	"shoppingAddress": "Calle 18D #8A E Lorenzo",
6	"date": "2023-07-02",
7	"status": false,
8	"customer": [
9	{id: 1,
10	"name": "Juan José Revales",
11	"identificationNumber": "1194543454",
12	"email": "tevelojuan88@mail.com",
13	"phone": "3145676787"
14],
15	"product_orders": [
16	{id: 1,
17	"amount": 1,
18	"product": {
19	"id": 1,
20	"name": "Manzana",
21	}

Status: 200 OK Time: 14 ms Size: 6013 KB Save as Example

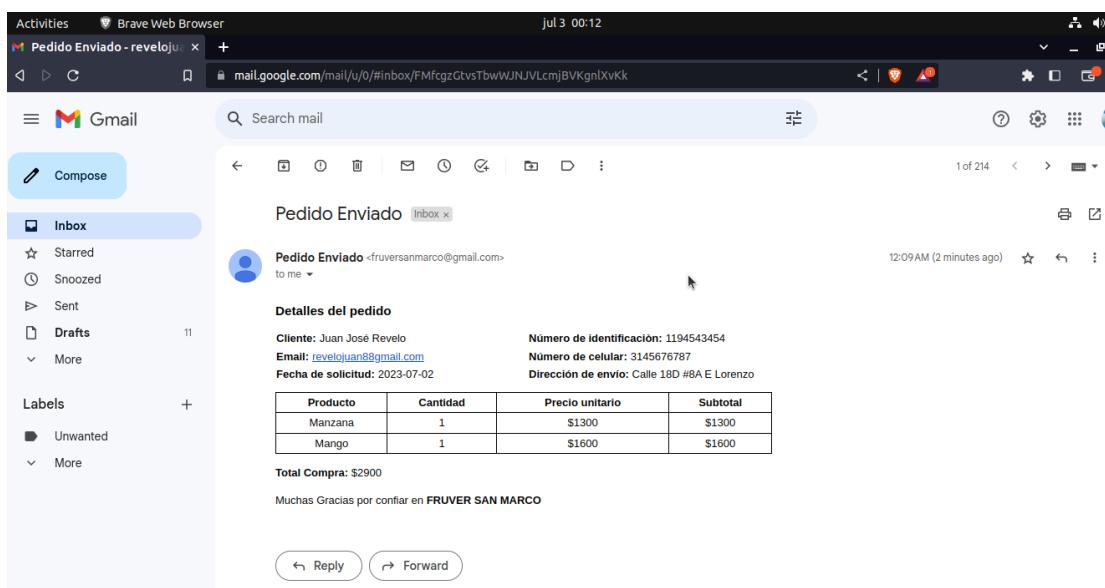
3. Consolidar y enviar pedido

Para consumir dicho endpoint se debe estar autenticado, por tal razón, se tiene que realizar el proceso mostrado anteriormente. Como se puede observar en la siguiente imagen, se consumió el endpoint para consolidar y enviar un pedido, y la solicitud fue exitosa.



The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000/orders`. The Body tab shows a JSON payload with `"id": 1`. The response status is `200 OK` with the message: `"message": "Pedido consolidado y enviado exitosamente: Se notificó al cliente el envío de su pedido"`.

Una vez consolidado y enviado el pedido, se cambia su estado a true, refiriéndose a que el pedido ya fue atendido, y se notifica al correo del cliente que el pedido fue enviado, como se muestra en la siguiente imagen.



The screenshot shows an email in the Gmail inbox. The subject is `Pedido Enviado`. The email details a purchase of two items: `Manzana` and `Mango`, both at `$1300` and `$1600` respectively, totaling `$2900`. The message concludes with a thank you from `FRUVER SAN MARCO`.