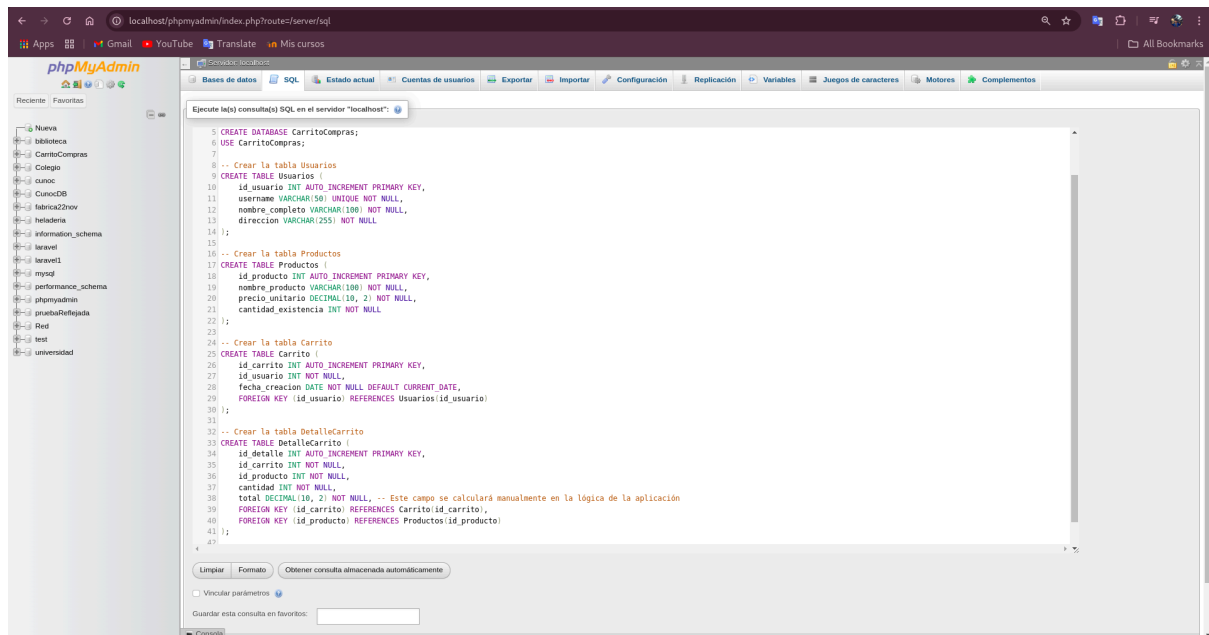


Eleazar Neftalí Colop Colop

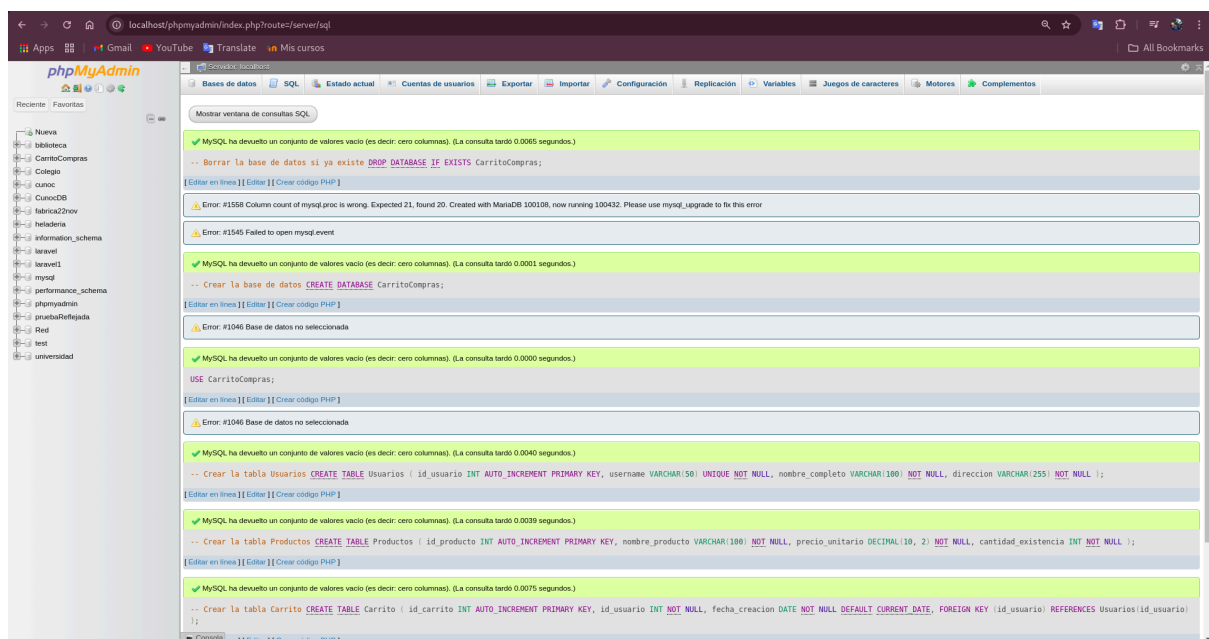
3198935960914

primero crearemos nuestra base de datos:



The screenshot shows the phpMyAdmin interface with the SQL editor open. The SQL code is as follows:

```
5 CREATE DATABASE CarritoCompras;
6 USE CarritoCompras;
7
8 -- Crear la tabla Usuarios
9 CREATE TABLE Usuarios (
10   id_usuario INT AUTO_INCREMENT PRIMARY KEY,
11   username VARCHAR(50) UNIQUE NOT NULL,
12   nombre_completo VARCHAR(100) NOT NULL,
13   direccion VARCHAR(255) NOT NULL
14 );
15
16 -- Crear la tabla Productos
17 CREATE TABLE Productos (
18   id_producto INT AUTO_INCREMENT PRIMARY KEY,
19   nombre_producto VARCHAR(100) NOT NULL,
20   precio_unitario DECIMAL(10, 2) NOT NULL,
21   cantidad_existencia INT NOT NULL
22 );
23
24 -- Crear la tabla Carrito
25 CREATE TABLE Carrito (
26   id_carrito INT AUTO_INCREMENT PRIMARY KEY,
27   id_usuario INT NOT NULL,
28   fecha_creacion DATE NOT NULL DEFAULT CURRENT_DATE,
29   FOREIGN KEY (id_usuario) REFERENCES Usuarios(id_usuario)
30 );
31
32 -- Crear la tabla DetalleCarrito
33 CREATE TABLE DetalleCarrito (
34   id_detalle INT AUTO_INCREMENT PRIMARY KEY,
35   id_carrito INT NOT NULL,
36   id_producto INT NOT NULL,
37   cantidad INT NOT NULL,
38   total DECIMAL(10, 2) NOT NULL, -- Este campo se calculará manualmente en la lógica de la aplicación
39   FOREIGN KEY (id_carrito) REFERENCES Carrito(id_carrito),
40   FOREIGN KEY (id_producto) REFERENCES Productos(id_producto)
41 );
```



The screenshot shows the phpMyAdmin interface with the SQL editor open, displaying the execution results of the SQL code. The results are as follows:

```
MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0065 segundos.)
-- Borrar la base de datos si ya existe DROP DATABASE IF EXISTS CarritoCompras;
[Editar en línea] [Editar] [Crear código PHP]

Error: #1558 Column count of mysql.proc is wrong. Expected 21, found 20. Created with MariaDB 100108, now running 100432. Please use mysql_upgrade to fix this error.

Error: #1545 Failed to open mysql event.

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0001 segundos.)
-- Crear la base de datos CREATE DATABASE CarritoCompras;
[Editar en línea] [Editar] [Crear código PHP]

Error: #1046 Base de datos no seleccionada

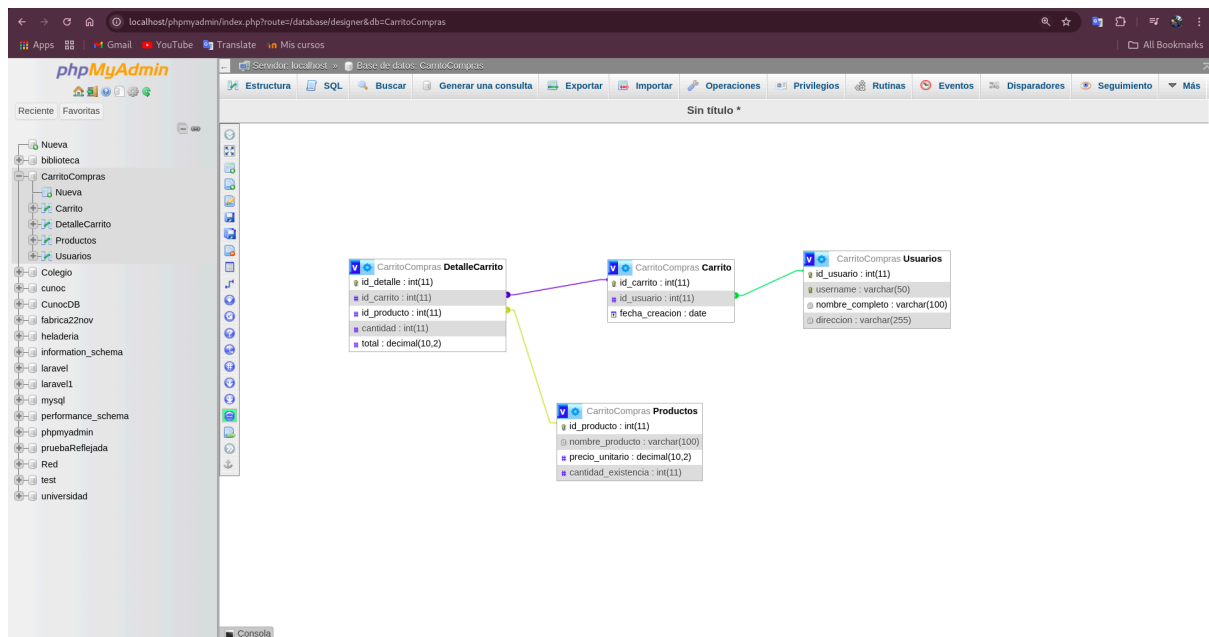
MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0000 segundos.)
USE CarritoCompras;
[Editar en línea] [Editar] [Crear código PHP]

Error: #1046 Base de datos no seleccionada

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0040 segundos.)
-- Crear la tabla Usuarios CREATE TABLE Usuarios ( id_usuario INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, nombre_completo VARCHAR(100) NOT NULL, direccion VARCHAR(255) NOT NULL );
[Editar en línea] [Editar] [Crear código PHP]

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0039 segundos.)
-- Crear la tabla Productos CREATE TABLE Productos ( id_producto INT AUTO_INCREMENT PRIMARY KEY, nombre_producto VARCHAR(100) NOT NULL, precio_unitario DECIMAL(10, 2) NOT NULL, cantidad_existencia INT NOT NULL );
[Editar en línea] [Editar] [Crear código PHP]

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0.0075 segundos.)
-- Crear la tabla Carrito CREATE TABLE Carrito ( id_carrito INT AUTO_INCREMENT PRIMARY KEY, id_usuario INT NOT NULL, fecha_creacion DATE NOT NULL DEFAULT CURRENT_DATE, FOREIGN KEY (id_usuario) REFERENCES Usuarios(id_usuario) );
[Editar en línea] [Editar] [Crear código PHP]
```



en nuestro caso la creamos con phpmy admin.
Luego creamos una carpeta para nuestro proyecto
y utilizamos el comando para crear nuestro proyecto:

```
eleazar@kaliNeta: ~/intecapcito/tareas/DPW_examen/API
(eleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$ npm init -y
Wrote to /home/eleazar/intecapcito/tareas/DPW_examen/API/package.json:

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

(eleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$
```

luego instalamos mysql2 para la database de nuestro proyecto

```
eleazar@kaliNeta: ~/intecapcito/tareas/DPW_examen/API
(aleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$ npm init -y
Wrote to /home/eleazar/intecapcito/tareas/DPW_examen/API/package.json:

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

(aleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$ npm i mysql2 express nodemon body-parser
```

```
eleazar@kaliNeta: ~/intecapcito/tareas/DPW_examen/API
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
"keywords": [],
"author": "",
"license": "ISC",
"description": ""
}

(aleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$ npm i mysql2 express nodemon body-parser

added 107 packages, and audited 108 packages in 36s

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

found 0 vulnerabilities

(aleazar@kaliNeta)-[~/intecapcito/tareas/DPW_examen/API]
$
```

configuramos nuestra conexión en nuestro archivo index.js que crearemos:

```
JS index.js > @ app.use("/") callback
1 import mysql2 from "mysql2";
2 import express from "express";
3 import bodyParser from "body-parser";
4
5 const connection = mysql2.createConnection({
6   host: "localhost",
7   database: "CarritoCompras",
8   user: "root",
9   password: "",
10 });
11
12 const app = express();
13
14 app.use(bodyParser.json());
15
16 const PORT = 5000;
17
18 // Conectar a la base de datos
19 connection.connect((err) => {
20   if (err) {
21     console.error("Error al conectar a la base de datos:", err.message);
22     return;
23   }
24   console.log("Base de datos conectada!");
25 });
26
27
28 // Ruta de prueba
29 app.use("/", (req, res) => {
30   res.send("Hola esta es la API de nuestro carrito de compras");
31 });
32
33 // Iniciar el servidor
34 app.listen(PORT, () => {
35   console.log(`Corriendo en: http://localhost:${PORT}`);
36 });
37
```

aca tenemos nuestra conexión a la db y configuramos nuestro puerto
luego creamos nuestros metodos para ingresar, ver, modificar y eliminar para:
usuarios

```
//rutas para los usuarios
app.post("/usuarios", (req, res) => {
  const { username, nombre_completo, direccion } = req.body;

  if (!username || !nombre_completo || !direccion) {
    return res.status(400).send("Todos los campos son obligatorios: username, nombre_completo, direccion");
  }

  const query = "INSERT INTO Usuarios (username, nombre_completo, direccion) VALUES (?, ?, ?)";
  connection.query(query, [username, nombre_completo, direccion], (err, result) => {
    if (err) {
      console.error("Error al agregar el usuario:", err.message);
      return res.status(500).send("Error al agregar el usuario");
    }
    res.status(201).send("Usuario agregado con éxito");
  });
});

app.get("/usuarios", (req, res) => {
  const query = "SELECT * FROM Usuarios";
  connection.query(query, (err, results) => {
    if (err) {
      console.error("Error al obtener los usuarios:", err.message);
      return res.status(500).send("Error al obtener los usuarios");
    }
    res.json(results);
  });
});

app.put("/usuarios/:id", (req, res) => {
  const { id } = req.params;
  const { username, nombre_completo, direccion } = req.body;

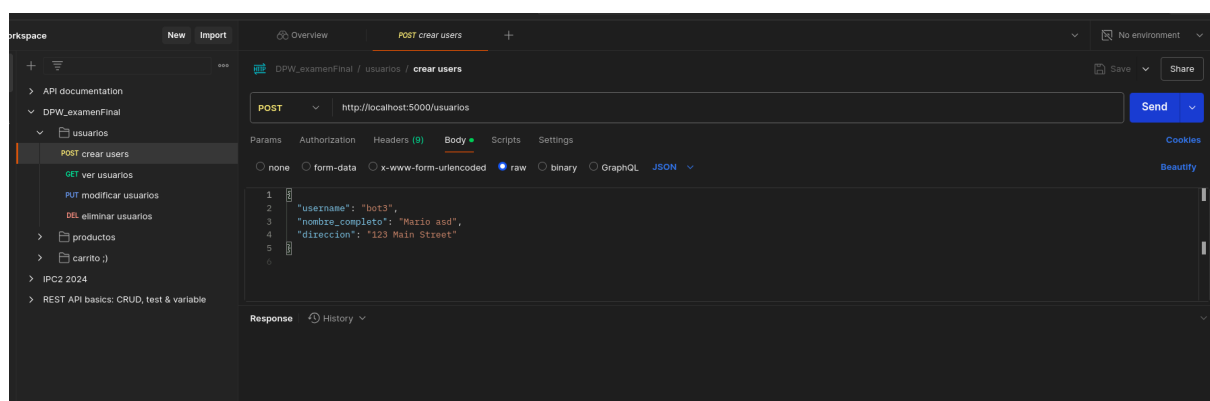
  if (!username && !nombre_completo && !direccion) {
    return res.status(400).send("Debe proporcionar al menos un campo para actualizar: username, nombre_completo o direccion");
  }

  const fieldsToUpdate = [];
  if (username) fieldsToUpdate.push(`username = '${username}'`);
  if (nombre_completo) fieldsToUpdate.push(`nombre_completo = '${nombre_completo}'`);
  if (direccion) fieldsToUpdate.push(`direccion = '${direccion}'`);

  const query = `UPDATE Usuarios SET ${fieldsToUpdate.join(", ")} WHERE id_usuario = ?`;
  connection.query(query, [id], (err, result) => {
    if (err) {
      console.error("Error al actualizar el usuario:", err.message);
      return res.status(500).send("Error al actualizar el usuario");
    }

    if (result.affectedRows === 0) {
      return res.status(404).send("Usuario no encontrado");
    }

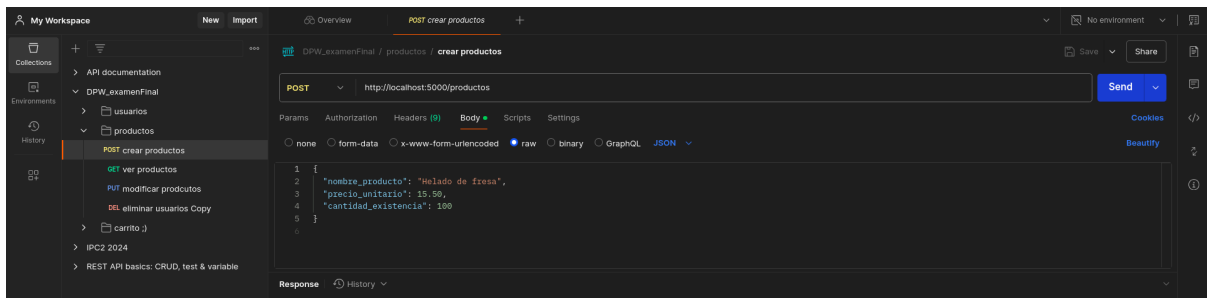
    res.send("Usuario actualizado con éxito");
  });
});
```



realizamos los request en postman
De la misma manera en productos:

```
JS index.js X
JS index.js > ...

84 //ruta para los productos
85 app.post("/productos", (req, res) => {
86   const { nombre_producto, precio_unitario, cantidad_existencia } = req.body;
87
88   if (!nombre_producto || !precio_unitario || !cantidad_existencia) {
89     return res.status(400).send("Todos los campos son obligatorios: nombre_producto, precio_unitario, cantidad_existencia");
90   }
91
92   const query = "INSERT INTO Productos (nombre_producto, precio_unitario, cantidad_existencia) VALUES (?, ?, ?)";
93   connection.query(query, [nombre_producto, precio_unitario, cantidad_existencia], (err, result) => {
94     if (err) {
95       console.error("Error al agregar el producto:", err.message);
96       return res.status(500).send("Error al agregar el producto");
97     }
98     res.status(201).send("Producto agregado con éxito");
99   });
100 });
101
102 app.get("/productos", (req, res) => {
103   const query = "SELECT * FROM Productos";
104   connection.query(query, (err, results) => {
105     if (err) {
106       console.error("Error al obtener los productos:", err.message);
107       return res.status(500).send("Error al obtener los productos");
108     }
109     res.json(results);
110   });
111 });
112
113 app.put("/productos/:id", (req, res) => {
114   const { id } = req.params;
115   const { nombre_producto, precio_unitario, cantidad_existencia } = req.body;
116
117   if (!nombre_producto && !precio_unitario && !cantidad_existencia) {
118     return res.status(400).send("Debe proporcionar al menos un campo para actualizar: nombre_producto, precio_unitario o cantidad_existencia");
119   }
120
121   const fieldsToUpdate = [];
122   if (nombre_producto) fieldsToUpdate.push(`nombre_producto = '${nombre_producto}'`);
123   if (precio_unitario) fieldsToUpdate.push(`precio_unitario = ${precio_unitario}`);
124   if (cantidad_existencia) fieldsToUpdate.push(`cantidad_existencia = ${cantidad_existencia}`);
125
126   const query = `UPDATE Productos SET ${fieldsToUpdate.join(", ")} WHERE id_producto = ?`;
127   connection.query(query, [id], (err, result) => {
128     if (err) {
129       console.error("Error al actualizar el producto:", err.message);
130       return res.status(500).send("Error al actualizar el producto");
131     }
132
133     if (result.affectedRows === 0) {
134       return res.status(404).send("Producto no encontrado");
135     }
136
137     res.send("Producto actualizado con éxito");
138   });
139 });
```



The screenshot shows the VS Code interface with a REST client request configured. The request is a POST to 'http://localhost:5000/productos'. The body is raw JSON: { 'nombre_producto': 'Helado de fresa', 'precio_unitario': 15.50, 'cantidad_existencia': 100 }. The left sidebar shows a file explorer with 'My Workspace' and a list of files including 'API documentation', 'DPW_examenFinal', 'usuarios', 'productos', and 'POST crear productos'.

y luego para el carrito:

```

// Agregar producto al carrito
app.post('/carrito/agregar', (req, res) => {
  const { identificador_carrito, id_usuario, id_producto, cantidad } = req.body;

  // Obtener el precio del producto
  connection.query('SELECT precio_unitario FROM Productos WHERE id_producto = ?', [id_producto], (err, result) => {
    if (err) {
      return res.status(500).json({ error: err.message });
    }

    if (result.length === 0) {
      return res.status(404).json({ error: 'Producto no encontrado' });
    }

    const precio_unitario = result[0].precio_unitario;
    const subtotal = precio_unitario * cantidad;

    // Insertar en la tabla Carrito
    connection.query(
      'INSERT INTO Carrito (identificador_carrito, id_usuario, id_producto, cantidad, subtotal) VALUES (?, ?, ?, ?, ?)',
      [identificador_carrito, id_usuario, id_producto, cantidad, subtotal],
      (err, result) => {
        if (err) {
          return res.status(500).json({ error: err.message });
        }
        res.status(201).json({ message: 'Producto agregado al carrito', id_carrito: result.insertId });
      }
    );
  });
});

// Ver productos en un carrito
app.get('/carrito/:identificador_carrito', (req, res) => {
  const { identificador_carrito } = req.params;

  connection.query(
    'SELECT c.id_carrito, p.nombre_producto, c.cantidad, c.subtotal, c.fecha_creacion FROM Carrito c JOIN Productos p ON c.id_producto = p.id_producto WHERE c.identificador_carrito = ?',
    [identificador_carrito],
    (err, result) => {
      if (err) {
        return res.status(500).json({ error: err.message });
      }

      if (result.length === 0) {
        return res.status(404).json({ message: 'No se encontraron productos en este carrito' });
      }

      res.json(result);
    }
  );
});

// Editar cantidad de un producto en el carrito
app.put('/carrito/editar/:id_carrito', (req, res) => {
  const { id_carrito } = req.params;
  const { cantidad } = req.body;

```

