

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

Backend Node.js + Express + MySQL

- CRUD de productos, clientes, ventas
 - Control de stock y transacciones
 - Autenticación con JWT
 - Roles de usuario (**admin**, **vendedor**)
 - Protección de rutas
 - Exportación a **Excel** y **PDF** de informes
-

Frontend HTML + Bootstrap

- Formularios modernos y responsivos
 - Panel de control personalizado según el rol
 - Acciones con feedback visual
 - Módulos separados para productos, clientes, ventas, estadísticas y usuarios
-

Seguridad y control de acceso

- Login con validación
 - Token guardado en **localStorage**
 - Interfaz adaptada al tipo de usuario
 - Restricción de rutas sensibles (usuarios, estadísticas)
-

ESTRUCTURA GENERAL DEL PROYECTO

pgsql

```
ventas-app/
├── backend/
│   ├── db.js
│   ├── server.js
│   ├── routes/
│   │   ├── clientes.js
│   │   ├── productos.js
│   │   └── ventas.js
│   └── controllers/
│       ├── clientesController.js
│       ├── productosController.js
│       └── ventasController.js
├── frontend/
│   ├── index.html
│   ├── clientes.html
│   ├── productos.html
│   └── ventas.html
├── .env
├── package.json
└── README.md
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

PASO 1: CREAR LA BASE DE DATOS

Conéctate a MySQL y ejecuta:

sql

```
CREATE DATABASE IF NOT EXISTS ventas_comercio;  
USE ventas_comercio;
```

```
CREATE TABLE clientes (  
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100),  
    email VARCHAR(100),  
    telefono VARCHAR(20)  
);
```

```
CREATE TABLE productos (  
    id_producto INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100),  
    precio DECIMAL(10,2),  
    stock INT,  
    stock_minimo INT DEFAULT 5  
);
```

```
CREATE TABLE ventas (  
    id_venta INT AUTO_INCREMENT PRIMARY KEY,  
    id_cliente INT,  
    fecha DATETIME DEFAULT CURRENT_TIMESTAMP,  
    FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente)  
);
```

```
CREATE TABLE detalle_venta (  
    id_detalle INT AUTO_INCREMENT PRIMARY KEY,  
    id_venta INT,  
    id_producto INT,  
    cantidad INT,  
    precio_unitario DECIMAL(10,2),  
    FOREIGN KEY (id_venta) REFERENCES ventas(id_venta),  
    FOREIGN KEY (id_producto) REFERENCES productos(id_producto)  
);
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.



PASO 2: INICIALIZAR BACKEND

2.1. Estructura

bash

```
mkdir backend
cd backend
npm init -y
npm install express mysql2 dotenv cors
```

2.2. Archivos básicos

bash

```
touch server.js db.js .env
mkdir routes controllers
```

2.3. .env

env

```
DB_HOST=localhost
DB_USER=tu_usuario
DB_PASS=tu_contraseña
DB_NAME=ventas_comercio
PORT=3000
```

2.4. db.js

js

```
const mysql = require('mysql2/promise');
require('dotenv').config();

const pool = mysql.createPool({
  host: process.env.DB_HOST,
  user: process.env.DB_USER,
  password: process.env.DB_PASS,
  database: process.env.DB_NAME
});

module.exports = pool;
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

2.5. **server.js**

js

```
const express = require('express');
const cors = require('cors');
require('dotenv').config();

const app = express();
app.use(cors());
app.use(express.json());

// Rutas
app.use('/api/clientes', require('./routes/clientes'));
app.use('/api/productos', require('./routes/productos'));
app.use('/api/ventas', require('./routes/ventas'));

app.listen(process.env.PORT, () => {
  console.log(`Servidor corriendo en
http://localhost:${process.env.PORT}`);
});
```



PASO 3: FRONTEND BASE CON BOOTSTRAP

Creamos carpeta y archivo:

bash

```
mkdir ../frontend
touch ../frontend/productos.html
```

productos.html con Bootstrap

html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
initial-scale=1" />
  <title>Gestión de Productos</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstra
p.min.css" rel="stylesheet" />
</head>
<body class="bg-light">

  <div class="container py-4">
    <h1 class="mb-4">Gestión de Productos</h1>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
<form id="form-producto" class="card p-3 mb-4 shadow-sm">
  <div class="mb-3">
    <label for="nombre" class="form-label">Nombre</label>
    <input type="text" class="form-control" id="nombre" required
/>
  </div>
  <div class="mb-3">
    <label for="precio" class="form-label">Precio</label>
    <input type="number" class="form-control" id="precio"
required />
  </div>
  <div class="mb-3">
    <label for="stock" class="form-label">Stock</label>
    <input type="number" class="form-control" id="stock"
required />
  </div>
  <div class="mb-3">
    <label for="stock_minimo" class="form-label">Stock
Mínimo</label>
    <input type="number" class="form-control" id="stock_minimo"
value="5" />
  </div>
  <button type="submit" class="btn btn-primary">Guardar</button>
</form>

<h2>Lista de Productos</h2>
<table class="table table-bordered table-striped"
id="tabla-productos">
  <thead class="table-dark">
    <tr>

<th>ID</th><th>Nombre</th><th>Precio</th><th>Stock</th><th>Mínimo</t
h>
    </tr>
  </thead>
  <tbody></tbody>
</table>
</div>

<script>
const API = 'http://localhost:3000/api/productos';

const cargarProductos = async () => {
  const res = await fetch(API);
  const datos = await res.json();
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        const tbody = document.querySelector('#tabla-productos
tbody');
        tbody.innerHTML = '';
        datos.forEach(p => {
            tbody.innerHTML += `
            <tr>
                <td>${p.id_producto}</td>
                <td>${p.nombre}</td>
                <td>${p.precio}</td>
                <td>${p.stock}</td>
                <td>${p.stock_minimo}</td>
            </tr>`;
        });
    };

document.getElementById('form-producto').addEventListener('submit',
async e => {
    e.preventDefault();
    const nuevo = {
        nombre: document.getElementById('nombre').value,
        precio: parseFloat(document.getElementById('precio').value),
        stock: parseInt(document.getElementById('stock').value),
        stock_minimo:
parseInt(document.getElementById('stock_minimo').value)
    };
    const res = await fetch(API, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(nuevo)
    });
    const result = await res.json();
    alert(result.message);
    e.target.reset();
    cargarProductos();
});

    cargarProductos();
</script>
</body>
</html>
```

backend real para productos, que conectará el formulario Bootstrap con la base de datos MySQL.

✓ PASO 4: BACKEND – Rutas y controladores para productos

♦ 4.1. Crear archivo `routes/productos.js`

En la carpeta `backend/routes/`, crea el archivo:

bash

```
touch routes/productos.js
```

Contenido de `routes/productos.js`:

js

```
const express = require('express');
const router = express.Router();
const productosController =
  require('../controllers/productosController');

// Rutas
router.get('/', productosController.getAll);
router.post('/', productosController.create);

module.exports = router;
```

♦ 4.2. Crear archivo `controllers/productosController.js`

En la carpeta `backend/controllers/`, crea el archivo:

bash

```
touch controllers/productosController.js
```

Contenido de `productosController.js`:

js

```
const db = require('../db');

// Obtener todos los productos
exports.getAll = async (req, res) => {
  try {
    const [rows] = await db.query('SELECT * FROM productos');
    res.json(rows);
  } catch (error) {
    console.error('Error al obtener productos:', error);
    res.status(500).json({ error: 'Error al obtener productos' });
  }
};

// Crear nuevo producto
exports.create = async (req, res) => {
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
const { nombre, precio, stock, stock_minimo } = req.body;

if (!nombre || precio == null || stock == null) {
  return res.status(400).json({ error: 'Faltan campos obligatorios' });
}

try {
  await db.query(
    'INSERT INTO productos (nombre, precio, stock, stock_minimo) VALUES (?, ?, ?, ?)',
    [nombre, precio, stock, stock_minimo ?? 5]
  );
  res.json({ message: 'Producto creado correctamente' });
} catch (error) {
  console.error('Error al crear producto:', error);
  res.status(500).json({ error: 'Error al crear producto' });
}
};
```

♦ 4.3. Verifica que **server.js** incluye la ruta

En `backend/server.js`:

```
app.use('/api/productos', require('./routes/productos'));
```

♦ 4.4. Inicia el servidor

En el terminal:

```
bash

node server.js
```

Deberías ver:

```
bash

Servidor corriendo en http://localhost:3000
```

♦ 4.5. Prueba desde el navegador

1. Abre `frontend/productos.html`.
2. Usa el formulario para agregar un producto.
3. Verifica que aparece automáticamente en la tabla.

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

PASO 5: Gestión de Clientes, usando el mismo enfoque:

- Backend con rutas y controlador
- Frontend con Bootstrap: formulario + tabla

PASO 5: CLIENTES

5.1. BACKEND DE CLIENTES

a) Crear archivo **routes/clientes.js**

bash

```
touch routes/clientes.js
```

Contenido:

js

```
const express = require('express');
const router = express.Router();
const clientesController =
require('../controllers/clientesController');

router.get('/', clientesController.getAll);
router.post('/', clientesController.create);

module.exports = router;
```

b) Crear archivo **controllers/clientesController.js**

bash

```
touch controllers/clientesController.js
```

Contenido:

js

```
const db = require('../db');

// Obtener todos los clientes
exports.getAll = async (req, res) => {
  try {
    const [rows] = await db.query('SELECT * FROM clientes');
    res.json(rows);
  } catch (error) {
    console.error('Error al obtener clientes:', error);
    res.status(500).json({ error: 'Error al obtener clientes' });
  }
}
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
};

// Crear cliente nuevo
exports.create = async (req, res) => {
  const { nombre, email, telefono } = req.body;

  if (!nombre) {
    return res.status(400).json({ error: 'El nombre es obligatorio' });
  }

  try {
    await db.query(
      'INSERT INTO clientes (nombre, email, telefono) VALUES (?, ?, ?)',
      [nombre, email, telefono]
    );
    res.json({ message: 'Cliente creado correctamente' });
  } catch (error) {
    console.error('Error al crear cliente:', error);
    res.status(500).json({ error: 'Error al crear cliente' });
  }
};
```

c) Enlazar en **server.js**

En **backend/server.js**, asegúrate de tener:

```
app.use('/api/clientes', require('./routes/clientes'));
```

5.2. FRONTEND: **clientes.html** con Bootstrap

Crea:
bash

```
touch ../frontend/clientes.html
```

Contenido:
html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width,
initial-scale=1" />
  <title>Gestión de Clientes</title>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstra
p.min.css" rel="stylesheet" />
</head>
<body class="bg-light">
  <div class="container py-4">
    <h1 class="mb-4">Gestión de Clientes</h1>

    <form id="form-cliente" class="card p-3 mb-4 shadow-sm">
      <div class="mb-3">
        <label for="nombre" class="form-label">Nombre</label>
        <input type="text" class="form-control" id="nombre" required
/>
      </div>
      <div class="mb-3">
        <label for="email" class="form-label">Email</label>
        <input type="email" class="form-control" id="email" />
      </div>
      <div class="mb-3">
        <label for="telefono" class="form-label">Teléfono</label>
        <input type="text" class="form-control" id="telefono" />
      </div>
      <button type="submit" class="btn btn-success">Guardar</button>
    </form>

    <h2>Lista de Clientes</h2>
    <table class="table table-bordered table-striped"
id="tabla-clientes">
      <thead class="table-dark">
        <tr>
          <th>ID</th><th>Nombre</th><th>Email</th><th>Teléfono</th>
        </tr>
      </thead>
      <tbody></tbody>
    </table>
  </div>

  <script>
    const API = 'http://localhost:3000/api/clientes';

    async function cargarClientes() {
      const res = await fetch(API);
      const clientes = await res.json();
      const tbody = document.querySelector('#tabla-clientes tbody');
      tbody.innerHTML = '';
      clientes.forEach(c => {
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
tbody.innerHTML += `
    <tr>
      <td>${c.id_cliente}</td>
      <td>${c.nombre}</td>
      <td>${c.email || '-'}</td>
      <td>${c.telefono || '-'}</td>
    </tr>`;
  });
}

document.getElementById('form-cliente').addEventListener('submit',
async e => {
  e.preventDefault();
  const data = {
    nombre: document.getElementById('nombre').value,
    email: document.getElementById('email').value,
    telefono: document.getElementById('telefono').value
  };

  const res = await fetch(API, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });

  const result = await res.json();
  alert(result.message);
  e.target.reset();
  cargarClientes();
});

cargarClientes();
</script>
</body>
</html>
```

✓ ¿Qué has conseguido?

- Frontend estilizado con Bootstrap.
- Formulario funcional para agregar clientes.
- Tabla automática que muestra los datos actuales.

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

PASo 6: Registro de ventas, el núcleo del sistema.

Implementaremos:

- ✓ Selección de cliente
- ✓ Selección de varios productos con cantidad
- ✓ Control de stock
- ✓ Inserción en las tablas `ventas` y `detalle_venta`
- ✓ Actualización del stock
- ✓ Interfaz con Bootstrap

PASO 6: BACKEND - REGISTRO DE VENTAS

6.1. Ruta `routes/ventas.js`

bash

```
touch routes/ventas.js
```

Contenido:

js

```
const express = require('express');
const router = express.Router();
const ventasController = require('../controllers/ventasController');

router.post('/', ventasController.crearVenta);
router.get('/', ventasController.getAll); // opcional para ver
ventas

module.exports = router;
```

6.2. Controlador `controllers/ventasController.js`

bash

```
touch controllers/ventasController.js
```

Contenido:

js

```
const db = require('../db');

exports.crearVenta = async (req, res) => {
  const { id_cliente, productos } = req.body;
  // productos = [{ id_producto, cantidad }]

  if (!id_cliente || !Array.isArray(productos) || productos.length
=== 0) {
    return res.status(400).json({ error: 'Faltan datos' });
  }

  const conn = await db.getConnection();
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
try {
  await conn.beginTransaction();

  const [ventaRes] = await conn.query(
    'INSERT INTO ventas (id_cliente) VALUES (?)',
    [id_cliente]
  );
  const id_venta = ventaRes.insertId;

  for (const item of productos) {
    const [[prod]] = await conn.query(
      'SELECT stock, precio FROM productos WHERE id_producto = ?',
      [item.id_producto]
    );

    if (!prod || prod.stock < item.cantidad) {
      throw new Error(`Stock insuficiente para producto ${item.id_producto}`);
    }

    await conn.query(
      `INSERT INTO detalle_venta (id_venta, id_producto, cantidad,
precio_unitario)
      VALUES (?, ?, ?, ?)`,
      [id_venta, item.id_producto, item.cantidad, prod.precio]
    );

    await conn.query(
      `UPDATE productos SET stock = stock - ? WHERE id_producto =
?`,
      [item.cantidad, item.id_producto]
    );
  }

  await conn.commit();
  res.json({ message: 'Venta registrada correctamente' });
} catch (error) {
  await conn.rollback();
  console.error('Error al registrar venta:', error);
  res.status(500).json({ error: error.message });
} finally {
  conn.release();
}
};

exports.getAll = async (req, res) => {
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
try {
  const [rows] = await db.query(`
    SELECT v.id_venta, c.nombre AS cliente, v.fecha
    FROM ventas v
    JOIN clientes c ON v.id_cliente = c.id_cliente
    ORDER BY v.id_venta DESC
  `);
  res.json(rows);
} catch (error) {
  res.status(500).json({ error: 'Error al obtener ventas' });
}
};
```



PASO 7: FRONTEND CON BOOTSTRAP - ventas.html

bash

```
touch frontend/ventas.html
```

Contenido inicial (interfaz básica funcional):
html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Registrar Venta</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap
p.min.css" rel="stylesheet" />
</head>
<body class="bg-light">
  <div class="container py-4">
    <h1 class="mb-4">Registrar Venta</h1>

    <form id="form-venta" class="card p-3 shadow-sm mb-4">
      <div class="mb-3">
        <label for="cliente" class="form-label">Cliente</label>
        <select id="cliente" class="form-select" required></select>
      </div>

      <div id="productos-container" class="mb-3">
        <label class="form-label">Productos</label>
        <div class="row align-items-end mb-2">
          <div class="col">
            <select class="form-select producto"></select>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        </div>
        <div class="col">
            <input type="number" class="form-control cantidad"
placeholder="Cantidad" min="1">
        </div>
    </div>
</div>

    <button type="button" class="btn btn-secondary mb-2"
onclick="agregarProducto()">+ Añadir otro producto</button>
    <button type="submit" class="btn btn-success">Registrar
Venta</button>
</form>
</div>

<script>
    const API_CLIENTES = 'http://localhost:3000/api/clientes';
    const API_PRODUCTOS = 'http://localhost:3000/api/productos';
    const API_VENTAS = 'http://localhost:3000/api/ventas';

    async function cargarClientes() {
        const res = await fetch(API_CLIENTES);
        const data = await res.json();
        const select = document.getElementById('cliente');
        data.forEach(c => {
            select.innerHTML += `<option
value="${c.id_cliente}">${c.nombre}</option>`;
        });
    }

    async function cargarProductos() {
        const res = await fetch(API_PRODUCTOS);
        const data = await res.json();
        document.querySelectorAll('.producto').forEach(select => {
            select.innerHTML = '';
            data.forEach(p => {
                select.innerHTML += `<option
value="${p.id_producto}">${p.nombre}</option>`;
            });
        });
    }

    function agregarProducto() {
        const contenedor = document.createElement('div');
        contenedor.className = 'row align-items-end mb-2';
        contenedor.innerHTML = `
```


Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        <div class="col">
            <select class="form-select producto"></select>
        </div>
        <div class="col">
            <input type="number" class="form-control cantidad"
placeholder="Cantidad" min="1" />
        </div>
    `;

document.getElementById('productos-container').appendChild(contenedo
r);
    cargarProductos();
}

    document.getElementById('form-venta').addEventListener('submit',
async e => {
    e.preventDefault();

    const id_cliente = document.getElementById('cliente').value;
    const productos =
Array.from(document.querySelectorAll('.producto')).map((sel, i) => {
    return {
        id_producto: parseInt(sel.value),
        cantidad:
parseInt(document.querySelectorAll('.cantidad')[i].value)
    };
});

    const res = await fetch(API_VENTAS, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ id_cliente, productos })
    });

    const result = await res.json();
    alert(result.message || result.error);
    location.reload();
});

    cargarClientes();
    cargarProductos();
</script>
</body>
</html>
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

✓ ¿Qué has conseguido?

- Formulario completo con Bootstrap para registrar ventas.
 - Control total desde backend con transacciones.
 - Validación de stock.
 - Lógica para añadir múltiples productos dinámicamente.
-

✓ PASO 8: Listado de ventas realizadas (con detalles)

Backend: GET /api/ventas/detalles

🔧 1. Añadir ruta a **routes/ventas.js**

Edita **backend/routes/ventas.js**:
js

```
router.get('/detalles', ventasController.getDetalles); // NUEVA RUTA
```

🔧 2. Añadir función a **ventasController.js**

Edita **backend/controllers/ventasController.js** y agrega:
js

```
exports.getDetalles = async (req, res) => {
  try {
    const [ventas] = await db.query(`
      SELECT v.id_venta, v.fecha, c.nombre AS cliente
      FROM ventas v
      JOIN clientes c ON v.id_cliente = c.id_cliente
      ORDER BY v.fecha DESC
    `);

    for (let venta of ventas) {
      const [detalles] = await db.query(`
        SELECT p.nombre, d.cantidad, d.precio_unitario
        FROM detalle_venta d
        JOIN productos p ON d.id_producto = p.id_producto
        WHERE d.id_venta = ?
      `, [venta.id_venta]);

      venta.detalles = detalles;
    }

    res.json(ventas);
  } catch (error) {
    console.error('Error al obtener detalles:', error);
  }
}
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    res.status(500).json({ error: 'Error al obtener ventas con  
detalles' });  
  }  
};
```



Frontend: **ventas.html** – añadir listado

Edita el archivo `frontend/ventas.html`, **después del formulario**, y añade esta sección: `html`

```
<h2 class="mt-5">Ventas Realizadas</h2>  
<div id="listado-ventas" class="accordion"></div>
```

Y justo antes del `</script>`:
`js`

```
async function mostrarVentas() {  
  const res = await fetch(`${API_VENTAS}/detalles`);  
  const data = await res.json();  
  const container = document.getElementById('listado-ventas');  
  
  container.innerHTML = '';  
  data.forEach((venta, index) => {  
    const total = venta.detalles.reduce((acc, d) => acc +  
d.precio_unitario * d.cantidad, 0);  
    container.innerHTML += `  
    <div class="accordion-item">  
      <h2 class="accordion-header" id="heading${index}">  
        <button class="accordion-button collapsed" type="button"  
data-bs-toggle="collapse" data-bs-target="#collapse${index}">  
          Venta #${venta.id_venta} - ${venta.cliente} - ${new  
Date(venta.fecha).toLocaleString()}  
        </button>  
      </h2>  
      <div id="collapse${index}" class="accordion-collapse  
collapse" data-bs-parent="#listado-ventas">  
        <div class="accordion-body">  
          <ul class="list-group">  
            ${venta.detalles.map(d => `  
              <li class="list-group-item d-flex  
justify-content-between">  
                ${d.nombre} (x${d.cantidad})  
                <span>${(d.precio_unitario *  
d.cantidad).toFixed(2)} €</span>  
              </li>  
            `).join('')}`  
          </ul>  
        </div>  
      </div>  
    </div>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
                <li class="list-group-item fw-bold d-flex
justify-content-between">
                    Total
                    <span>${total.toFixed(2)} €</span>
                </li>
            </ul>
        </div>
    </div>
</div>
`
    });
}
```

```
mostrarVentas();
```

Y al final del `<body>` (opcional, pero necesario para Bootstrap Accordion):
html

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.
bundle.min.js"></script>
```

✓ Resultado

Ahora verás en la parte inferior de `ventas.html` un acordeón elegante con:

- Lista de todas las ventas realizadas
- Productos comprados en cada una
- Cantidades y totales calculados

✓ PASO 9: Editar y eliminar productos (cliente será igual después)

Trabajaremos en tres partes:

1. Agregar **botones Editar / Eliminar** en la tabla de productos.
 2. Crear **modal Bootstrap** para editar productos.
 3. Implementar rutas backend para **PUT** (editar) y **DELETE** (eliminar).
-

1. BACKEND - Actualizar y eliminar productos

routes/productos.js

Agrega estas líneas:

js

```
router.put('/:id', productosController.update);
router.delete('/:id', productosController.remove);
```

controllers/productosController.js

Agrega al final del archivo:

js

```
// Actualizar producto
exports.update = async (req, res) => {
  const { nombre, precio, stock, stock_minimo } = req.body;
  const { id } = req.params;

  try {
    await db.query(
      `UPDATE productos SET nombre = ?, precio = ?, stock = ?,
stock_minimo = ? WHERE id_producto = ?`,
      [nombre, precio, stock, stock_minimo, id]
    );
    res.json({ message: 'Producto actualizado correctamente' });
  } catch (error) {
    console.error('Error al actualizar producto:', error);
    res.status(500).json({ error: 'Error al actualizar producto' });
  }
};

// Eliminar producto
exports.remove = async (req, res) => {
  const { id } = req.params;
  try {
    await db.query('DELETE FROM productos WHERE id_producto = ?',
[id]);
    res.json({ message: 'Producto eliminado correctamente' });
  } catch (error) {
    console.error('Error al eliminar producto:', error);
    res.status(500).json({ error: 'Error al eliminar producto' });
  }
};
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

2. FRONTEND - productos.html

A) Agrega columna de acciones en la tabla:

Modifica el `<thead>`:

html

```
<thead class="table-dark">
  <tr>

<th>ID</th><th>Nombre</th><th>Precio</th><th>Stock</th><th>Mínimo</th>
<th>Acciones</th>
  </tr>
</thead>
```

Modifica el `tbody.innerHTML +=` en `cargarProductos()`:

js

```
<tbody id="tabla-productos">
  productos.forEach(p => {
    tbody.innerHTML += `
      <tr>
        <td>${p.id_producto}</td>
        <td>${p.nombre}</td>
        <td>${p.precio}</td>
        <td>${p.stock}</td>
        <td>${p.stock_minimo}</td>
        <td>
          <button class="btn btn-sm btn-warning"
onclick="editarProducto(${p.id_producto}, '${p.nombre}',
${p.precio}, ${p.stock}, ${p.stock_minimo})">Editar</button>
          <button class="btn btn-sm btn-danger"
onclick="eliminarProducto(${p.id_producto})">Eliminar</button>
        </td>
      </tr>`;
    });
```

B) Modal Bootstrap para editar producto

Agrega al final del `<body>`:

html

```
<!-- Modal -->
<div class="modal fade" id="modalEditor" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <form id="form-editor">
        <div class="modal-header">
          <h5 class="modal-title">Editar Producto</h5>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        <button type="button" class="btn-close"
data-bs-dismiss="modal"></button>
    </div>
    <div class="modal-body">
        <input type="hidden" id="edit-id">
        <div class="mb-3">
            <label class="form-label">Nombre</label>
            <input type="text" class="form-control" id="edit-nombre"
required>
        </div>
        <div class="mb-3">
            <label class="form-label">Precio</label>
            <input type="number" class="form-control"
id="edit-precio" required>
        </div>
        <div class="mb-3">
            <label class="form-label">Stock</label>
            <input type="number" class="form-control"
id="edit-stock" required>
        </div>
        <div class="mb-3">
            <label class="form-label">Stock Mínimo</label>
            <input type="number" class="form-control"
id="edit-minimo" required>
        </div>
    </div>
    <div class="modal-footer">
        <button type="submit" class="btn
btn-primary">Guardar</button>
    </div>
</form>
</div>
</div>
</div>
```

Y al final:
html

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.
bundle.min.js"></script>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

C) Funciones JS para editar y eliminar

Agrega en el `<script>` de `productos.html`:

js

```
let modal;
document.addEventListener('DOMContentLoaded', () => {
  modal = new
bootstrap.Modal(document.getElementById('modalEditar'));
});

function editarProducto(id, nombre, precio, stock, stock_minimo) {
  document.getElementById('edit-id').value = id;
  document.getElementById('edit-nombre').value = nombre;
  document.getElementById('edit-precio').value = precio;
  document.getElementById('edit-stock').value = stock;
  document.getElementById('edit-minimo').value = stock_minimo;
  modal.show();
}

document.getElementById('form-editar').addEventListener('submit',
async (e) => {
  e.preventDefault();
  const id = document.getElementById('edit-id').value;
  const data = {
    nombre: document.getElementById('edit-nombre').value,
    precio:
parseFloat(document.getElementById('edit-precio').value),
    stock: parseInt(document.getElementById('edit-stock').value),
    stock_minimo:
parseInt(document.getElementById('edit-minimo').value)
  };

  const res = await
fetch(`http://localhost:3000/api/productos/${id}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data)
  });

  const result = await res.json();
  alert(result.message);
  modal.hide();
  cargarProductos();
});

async function eliminarProducto(id) {
```


Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

```
    if (confirm('¿Estás seguro de eliminar este producto?')) {
      const res = await
fetch(`http://localhost:3000/api/productos/${id}`, {
      method: 'DELETE'
    });
    const result = await res.json();
    alert(result.message);
    cargarProductos();
  }
}
```

✓ Resultado

- Puedes **editar cualquier producto** desde el modal con Bootstrap.
 - Puedes **eliminar productos** con confirmación.
 - Todo se actualiza al instante con `cargarProductos()`.
-

✓ PASO 10: Editar y eliminar clientes

✚ 1. BACKEND - Nuevas rutas

🔧 `routes/clientes.js`

Agrega al final:

js

```
router.put('/:id', clientesController.update);
router.delete('/:id', clientesController.remove);
```

🔧 `controllers/clientesController.js`

Agrega al final del archivo:

js

```
// Actualizar cliente
exports.update = async (req, res) => {
  const { nombre, email, telefono } = req.body;
  const { id } = req.params;

  try {
    await db.query(
      `UPDATE clientes SET nombre = ?, email = ?, telefono = ? WHERE
id_cliente = ?`,
      [nombre, email, telefono, id]
    );
    res.json({ message: 'Cliente actualizado correctamente' });
  }
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    } catch (error) {
      console.error('Error al actualizar cliente:', error);
      res.status(500).json({ error: 'Error al actualizar cliente' });
    }
  };

// Eliminar cliente
exports.remove = async (req, res) => {
  const { id } = req.params;

  try {
    await db.query('DELETE FROM clientes WHERE id_cliente = ?',
[id]);
    res.json({ message: 'Cliente eliminado correctamente' });
  } catch (error) {
    console.error('Error al eliminar cliente:', error);
    res.status(500).json({ error: 'Error al eliminar cliente' });
  }
};
```



2. FRONTEND - **clientes.html**



A) Agrega columna de acciones a la tabla

Modifica el `<thead>`:

html

```
<thead class="table-dark">
  <tr>

<th>ID</th><th>Nombre</th><th>Email</th><th>Teléfono</th><th>Accione
s</th>
  </tr>
</thead>
```

En `cargarClientes()`, modifica cada fila:

js

```
clientes.forEach(c => {
  tbody.innerHTML += `
    <tr>
      <td>${c.id_cliente}</td>
      <td>${c.nombre}</td>
      <td>${c.email || '-'}</td>
      <td>${c.telefono || '-'}</td>
      <td>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        <button class="btn btn-sm btn-warning"
onclick="editarCliente({c.id_cliente}, '{c.nombre}', '{c.email}',
'{c.telefono}')">Editar</button>
        <button class="btn btn-sm btn-danger"
onclick="eliminarCliente({c.id_cliente})">Eliminar</button>
    </td>
</tr>`;
});
```

B) Modal Bootstrap para editar cliente

Agrega al final del `<body>`:
html

```
<!-- Modal Cliente -->
<div class="modal fade" id="modalEditarCliente" tabindex="-1">
  <div class="modal-dialog">
    <div class="modal-content">
      <form id="form-editar-cliente">
        <div class="modal-header">
          <h5 class="modal-title">Editar Cliente</h5>
          <button type="button" class="btn-close"
data-bs-dismiss="modal"></button>
        </div>
        <div class="modal-body">
          <input type="hidden" id="edit-cliente-id">
          <div class="mb-3">
            <label class="form-label">Nombre</label>
            <input type="text" class="form-control"
id="edit-cliente-nombre" required>
          </div>
          <div class="mb-3">
            <label class="form-label">Email</label>
            <input type="email" class="form-control"
id="edit-cliente-email">
          </div>
          <div class="mb-3">
            <label class="form-label">Teléfono</label>
            <input type="text" class="form-control"
id="edit-cliente-telefono">
          </div>
        </div>
        <div class="modal-footer">
          <button type="submit" class="btn
btn-primary">Guardar</button>
        </div>
      </form>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    </div>
  </div>
</div>
```

Y no olvides el JS de Bootstrap:
html

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.
bundle.min.js"></script>
```

C) Funciones JavaScript

Agrega al `<script>`:
js

```
let modalCliente;
document.addEventListener('DOMContentLoaded', () => {
  modalCliente = new
bootstrap.Modal(document.getElementById('modalEditarCliente'));
});

function editarCliente(id, nombre, email, telefono) {
  document.getElementById('edit-cliente-id').value = id;
  document.getElementById('edit-cliente-nombre').value = nombre;
  document.getElementById('edit-cliente-email').value = email;
  document.getElementById('edit-cliente-telefono').value = telefono;
  modalCliente.show();
}

document.getElementById('form-editar-cliente').addEventListener('sub
mit', async (e) => {
  e.preventDefault();
  const id = document.getElementById('edit-cliente-id').value;
  const data = {
    nombre: document.getElementById('edit-cliente-nombre').value,
    email: document.getElementById('edit-cliente-email').value,
    telefono: document.getElementById('edit-cliente-telefono').value
  };

  const res = await
fetch(`http://localhost:3000/api/clientes/${id}`, {
  method: 'PUT',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(data)
});
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    const result = await res.json();
    alert(result.message);
    modalCliente.hide();
    cargarClientes();
  });

  async function eliminarCliente(id) {
    if (confirm('¿Estás seguro de eliminar este cliente?')) {
      const res = await
    fetch(`http://localhost:3000/api/clientes/${id}`, {
        method: 'DELETE'
      });
      const result = await res.json();
      alert(result.message);
      cargarClientes();
    }
  }
}
```

Resultado

Ahora puedes:

- Editar clientes con un modal cómodo y Bootstrap.
 - Eliminar clientes con confirmación.
 - Visualizar cambios al instante.
-

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

✓ PASO 11: Estadísticas e Informes

Vamos a implementar dos informes:

1.  **Productos más vendidos**
 2.  **Productos con stock por debajo del mínimo**
-

1. BACKEND

routes/productos.js – Añadir rutas estadísticas

Agrega:
js

```
router.get('/mas-vendidos', productosController.masVendidos);  
router.get('/stock-bajo', productosController.stockBajo);
```

controllers/productosController.js – Añadir funciones

Agrega al final del archivo:
js

```
// Productos más vendidos  
exports.masVendidos = async (req, res) => {  
  try {  
    const [rows] = await db.query(`  
      SELECT p.nombre, SUM(dv.cantidad) AS total_vendido  
      FROM detalle_venta dv  
      JOIN productos p ON dv.id_producto = p.id_producto  
      GROUP BY p.nombre  
      ORDER BY total_vendido DESC  
      LIMIT 10  
    `);  
    res.json(rows);  
  } catch (error) {  
    console.error('Error al obtener productos más vendidos:',  
error);  
    res.status(500).json({ error: 'Error en consulta' });  
  }  
};  
  
// Productos con stock por debajo del mínimo  
exports.stockBajo = async (req, res) => {  
  try {  
    const [rows] = await db.query(`  
      SELECT id_producto, nombre, stock, stock_minimo  
      FROM productos  
    `);  
    res.json(rows);  
  } catch (error) {  
    console.error('Error al obtener productos con stock bajo:',  
error);  
    res.status(500).json({ error: 'Error en consulta' });  
  }  
};
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        WHERE stock < stock_minimo
    `);
    res.json(rows);
  } catch (error) {
    console.error('Error al obtener productos con stock bajo:',
error);
    res.status(500).json({ error: 'Error en consulta' });
  }
};
```



2. FRONTEND: Nueva vista estadísticas.html

Crea el archivo:
bash

```
touch frontend/estadisticas.html
```

Contenido:
html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Estadísticas de Ventas</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body class="bg-light">
  <div class="container py-4">
    <h1 class="mb-4">Estadísticas e Informes</h1>

    <h3><img alt="Bar chart icon" data-bbox="208 665 232 682"/> Productos más vendidos</h3>
    <table class="table table-striped">
      <thead class="table-dark">
        <tr>
          <th>Producto</th><th>Total Vendido</th>
        </tr>
      </thead>
      <tbody id="mas-vendidos"></tbody>
    </table>

    <h3 class="mt-5"><img alt="Line chart icon" data-bbox="352 838 376 855"/> Productos con stock bajo</h3>
    <table class="table table-striped">
      <thead class="table-dark">
        <tr>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        <th>ID</th><th>Nombre</th><th>Stock</th><th>Mínimo</th>
    </tr>
</thead>
    <tbody id="stock-bajo"></tbody>
</table>
</div>

<script>
    async function cargarMasVendidos() {
        const res = await
fetch('http://localhost:3000/api/productos/mas-vendidos');
        const data = await res.json();
        const tbody = document.getElementById('mas-vendidos');
        tbody.innerHTML = '';
        data.forEach(p => {
            tbody.innerHTML +=
`<tr><td>${p.nombre}</td><td>${p.total_vendido}</td></tr>`;
        });
    }



    async function cargarStockBajo() {
        const res = await
fetch('http://localhost:3000/api/productos/stock-bajo');
        const data = await res.json();
        const tbody = document.getElementById('stock-bajo');
        tbody.innerHTML = '';
        data.forEach(p => {
            tbody.innerHTML += `<tr>
                <td>${p.id_producto}</td>
                <td>${p.nombre}</td>
                <td>${p.stock}</td>
                <td>${p.stock_minimo}</td>
            </tr>`;
        });
    }

    cargarMasVendidos();
    cargarStockBajo();
</script>
</body>
</html>
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

Resultado

Al abrir `estadisticas.html` verás:

-  Una tabla de los **10 productos más vendidos** ordenados por ventas.
 -  Una tabla con **productos que necesitan reposición** por estar por debajo del stock mínimo.
-

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

FASE 1: EXPORTACIÓN DE INFORMES A PDF/EXCEL

Backend

Usaremos las librerías:

bash

```
npm install exceljs pdfkit
```

a) Exportar productos más vendidos a Excel

Agregaremos una ruta:

js

```
router.get('/mas-vendidos/excel',  
productosController.exportMasVendidosExcel);
```

Y en el controlador:

js

```
const ExcelJS = require('exceljs');  
  
exports.exportMasVendidosExcel = async (req, res) => {  
  try {  
    const [data] = await db.query(`  
      SELECT p.nombre, SUM(dv.cantidad) AS total_vendido  
      FROM detalle_venta dv  
      JOIN productos p ON dv.id_producto = p.id_producto  
      GROUP BY p.nombre  
      ORDER BY total_vendido DESC  
    `);  
  
    const workbook = new ExcelJS.Workbook();  
    const sheet = workbook.addWorksheet('Productos más vendidos');  
  
    sheet.columns = [  
      { header: 'Producto', key: 'nombre', width: 30 },  
      { header: 'Total Vendido', key: 'total_vendido', width: 20 }  
    ];  
  
    data.forEach(item => sheet.addRow(item));  
  
    res.setHeader('Content-Type',  
'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')  
    ;  
    res.setHeader('Content-Disposition', 'attachment;  
filename="mas_vendidos.xlsx"');  
  
    await workbook.xlsx.write(res);  
  }  
}
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

```
        res.end();  
    } catch (error) {  
        res.status(500).json({ error: 'No se pudo exportar a Excel' });  
    }  
};
```

b) Exportar a PDF

Luego podemos hacer lo mismo con **pdfkit**, generando un PDF dinámico con los datos.

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.



FASE 2: SISTEMA DE LOGIN Y ROLES



Base de datos: tabla de usuarios

sql

```
CREATE TABLE usuarios (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(50),  
  email VARCHAR(100) UNIQUE,  
  password VARCHAR(255),  
  rol ENUM('admin', 'vendedor') DEFAULT 'vendedor'  
);
```



Backend

Instalar:

bash

```
npm install bcrypt jsonwebtoken
```

Implementaremos:

- Registro/login (/api/auth/register, /login)
 - Middleware `auth.js` que valide el token
 - Verificación de rol (`admin`, `vendedor`)
-

FASE 3: PROTECCIÓN DE RUTAS

Usaremos el middleware `auth.js`:

js

```
const jwt = require('jsonwebtoken');
module.exports = (req, res, next) => {
  const token = req.headers.authorization;
  if (!token) return res.status(401).json({ error: 'Token requerido'
});

  try {
    const decoded = jwt.verify(token.split(' ')[1],
process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (err) {
    res.status(401).json({ error: 'Token inválido' });
  }
};
```

Y se usará así:

js

```
const auth = require('../middlewares/auth');
router.get('/admin', auth, (req, res) => {
  if (req.user.rol !== 'admin') return res.status(403).json({ error:
'No autorizado' });
  res.json({ message: 'Ruta de administrador' });
});
```

FASE 4: PANEL DE ADMINISTRACIÓN

Tendrás una vista `admin.html` con:

- Menú con enlaces a:
 - gestión de productos
 - gestión de clientes
 - gestión de ventas
 - estadísticas
 - usuarios
- Accesos condicionales según el rol

Sistema de Gestión de Ventas en un Comercio, con **backend** en **Node.js**, **MySQL** y **frontend** con **Bootstrap**.

- Logout y token guardado en `localStorage`

✓ FASE 1.1: EXPORTAR A EXCEL – Productos más vendidos

✖ 1. Instalar dependencia

Desde la carpeta del backend:
bash

```
npm install exceljs
```

✖ 2. Backend – Ruta y controlador

a) Añadir nueva ruta en `routes/productos.js`

```
router.get('/mas-vendidos/excel',  
productosController.exportMasVendidosExcel);
```

b) Añadir función en `controllers/productosController.js`

```
const ExcelJS = require('exceljs');  
  
exports.exportMasVendidosExcel = async (req, res) => {  
  try {  
    const [data] = await db.query(`  
      SELECT p.nombre, SUM(dv.cantidad) AS total_vendido  
      FROM detalle_venta dv  
      JOIN productos p ON dv.id_producto = p.id_producto  
      GROUP BY p.nombre  
      ORDER BY total_vendido DESC  
    `);  
  
    const workbook = new ExcelJS.Workbook();  
    const sheet = workbook.addWorksheet('Más Vendidos');  
  
    sheet.columns = [  
      { header: 'Producto', key: 'nombre', width: 30 },  
      { header: 'Total Vendido', key: 'total_vendido', width: 20 }  
    ];  
  
    data.forEach(item => {  
      sheet.addRow(item);  
    });  
  
    res.setHeader(  

```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        'Content-Type',  
  
        'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet'  
    );  
    res.setHeader(  
        'Content-Disposition',  
        'attachment; filename="productos_mas_vendidos.xlsx"'  
    );  
  
    await workbook.xlsx.write(res);  
    res.end();  
} catch (error) {  
    console.error('Error al generar Excel:', error);  
    res.status(500).json({ error: 'No se pudo generar el Excel' });  
}  
};
```

✓ 3. Probar la descarga

Abre en tu navegador:

bash

<http://localhost:3000/api/productos/mas-vendidos/excel>

Se descargará un archivo llamado `productos_mas_vendidos.xlsx` con las columnas:

Producto | Total Vendido

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

✓ FASE 1.2: EXPORTACIÓN A PDF – Productos más vendidos

✂ 1. Instalar pdfkit

Desde el backend:

bash

```
npm install pdfkit
```

✂ 2. Backend – Ruta y controlador

a) Añadir nueva ruta en `routes/productos.js`

js

```
router.get('/mas-vendidos/pdf',  
productosController.exportMasVendidosPDF);
```

b) Añadir función en `controllers/productosController.js`

Agrega al final del archivo:

js

```
const PDFDocument = require('pdfkit');  
  
exports.exportMasVendidosPDF = async (req, res) => {  
  try {  
    const [data] = await db.query(`  
      SELECT p.nombre, SUM(dv.cantidad) AS total_vendido  
      FROM detalle_venta dv  
      JOIN productos p ON dv.id_producto = p.id_producto  
      GROUP BY p.nombre  
      ORDER BY total_vendido DESC  
    `);  
  
    const doc = new PDFDocument({ margin: 50 });  
  
    // Cabeceras  
    res.setHeader('Content-Type', 'application/pdf');  
    res.setHeader('Content-Disposition', 'inline;  
filename="mas_vendidos.pdf"');  
  
    // Conectar el doc PDF al response  
    doc.pipe(res);  
  
    // Título  
    doc.fontSize(20).text('Informe: Productos Más Vendidos', {  
align: 'center' });  
    doc.moveDown();
```


Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
// Tabla
doc.fontSize(12).text('Producto', 100, doc.y, { bold: true });
doc.text('Total Vendido', 400, doc.y, { bold: true });
doc.moveDown();

data.forEach(p => {
  doc.text(p.nombre, 100, doc.y);
  doc.text(p.total_vendido.toString(), 400, doc.y);
  doc.moveDown();
});

// Finalizar
doc.end();
} catch (error) {
  console.error('Error al generar PDF:', error);
  res.status(500).json({ error: 'No se pudo generar el PDF' });
}
};
```

3. Probar la descarga

Abre en tu navegador:

bash



<http://localhost:3000/api/productos/mas-vendidos/pdf>

Verás el PDF directamente en el navegador (o lo descargará, según configuración). Incluirá:

- Título centrado
- Tabla sencilla con:
 - Producto
 - Total vendido

Resultado

Ya tienes el informe de productos más vendidos exportable a:

-  **Excel** → </mas-vendidos/excel>
-  **PDF** → </mas-vendidos/pdf>

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

✓ FASE 2: LOGIN Y ROLES EN NODE.JS

🧩 1. BASE DE DATOS – Crear tabla de usuarios

Ejecuta en MySQL:
sql

```
CREATE TABLE usuarios (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  nombre VARCHAR(100) NOT NULL,  
  email VARCHAR(100) UNIQUE NOT NULL,  
  password VARCHAR(255) NOT NULL,  
  rol ENUM('admin', 'vendedor') DEFAULT 'vendedor'  
);
```

Ejemplo de roles:

- "admin": puede gestionar todo.
- "vendedor": acceso solo a ventas, productos y consultas.

🧩 2. DEPENDENCIAS

Instala en el backend:
bash

```
npm install bcrypt jsonwebtoken
```

🧩 3. BACKEND – Autenticación

🔧 a) Crear carpeta y archivo

bash

```
mkdir auth  
touch auth/authController.js auth/authRoutes.js middlewares/auth.js
```

🔧 b) authRoutes.js

js

```
const express = require('express');  
const router = express.Router();  
const controller = require('./authController');  
  
router.post('/register', controller.register);
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
router.post('/login', controller.login);
```

```
module.exports = router;
```

c) authController.js

```
const db = require('../db');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

exports.register = async (req, res) => {
  const { nombre, email, password, rol } = req.body;

  if (!nombre || !email || !password) {
    return res.status(400).json({ error: 'Faltan campos obligatorios' });
  }

  try {
    const hashed = await bcrypt.hash(password, 10);
    await db.query(
      'INSERT INTO usuarios (nombre, email, password, rol) VALUES (?, ?, ?, ?)',
      [nombre, email, hashed, rol || 'vendedor']
    );
    res.json({ message: 'Usuario registrado correctamente' });
  } catch (err) {
    res.status(500).json({ error: 'Error al registrar usuario' });
  }
};

exports.login = async (req, res) => {
  const { email, password } = req.body;

  try {
    const [[user]] = await db.query('SELECT * FROM usuarios WHERE email = ?', [email]);
    if (!user) return res.status(404).json({ error: 'Usuario no encontrado' });

    const match = await bcrypt.compare(password, user.password);
    if (!match) return res.status(401).json({ error: 'Contraseña incorrecta' });

    const token = jwt.sign(
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        { id: user.id, nombre: user.nombre, rol: user.rol },
        process.env.JWT_SECRET,
        { expiresIn: '1h' }
    );

    res.json({ token, user: { id: user.id, nombre: user.nombre, rol:
user.rol } });
    } catch (err) {
        res.status(500).json({ error: 'Error en login' });
    }
};
```



d) middlewares/auth.js

js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
    const authHeader = req.headers.authorization;
    if (!authHeader) return res.status(401).json({ error: 'Token
requerido' });

    const token = authHeader.split(' ')[1];
    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch {
        res.status(403).json({ error: 'Token inválido' });
    }
};
```



4. Integrar en server.js

En server.js:

js

```
const authRoutes = require('./auth/authRoutes');
const authMiddleware = require('./middlewares/auth');

app.use('/api/auth', authRoutes);

// Rutas protegidas (ejemplo)
app.get('/api/protegido', authMiddleware, (req, res) => {
    res.json({ message: `Hola ${req.user.nombre}, eres
${req.user.rol}` });
});
```

Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

```
});
```



5. PRUEBA CON POSTMAN

◆ Registro:

bash

```
POST http://localhost:3000/api/auth/register
```

```
Content-Type: application/json
```

```
{
  "nombre": "Óscar",
  "email": "oscar@email.com",
  "password": "123456",
  "rol": "admin"
}
```

◆ Login:

bash

```
POST http://localhost:3000/api/auth/login
```

```
Content-Type: application/json
```

```
{
  "email": "oscar@email.com",
  "password": "123456"
}
```

✓ Recibirás un **token** y los datos del usuario.

✓ FASE 3: PROTECCIÓN DE RUTAS Y CONTROL DE ACCESO

✂ 1. Guardar y usar el token (login frontend)

Creamos un archivo `login.html`:
html

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Login</title>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body class="bg-light">
  <div class="container py-5">
    <h2 class="mb-4">Iniciar Sesión</h2>
    <form id="form-login" class="card p-4 shadow-sm">
      <div class="mb-3">
        <label for="email" class="form-label">Correo</label>
        <input type="email" class="form-control" id="email" required
/>
      </div>
      <div class="mb-3">
        <label for="password" class="form-label">Contraseña</label>
        <input type="password" class="form-control" id="password"
required />
      </div>
      <button type="submit" class="btn btn-primary">Entrar</button>
    </form>
  </div>

  <script>
    document.getElementById('form-login').addEventListener('submit',
async e => {
      e.preventDefault();

      const email = document.getElementById('email').value;
      const password = document.getElementById('password').value;

      const res = await
fetch('http://localhost:3000/api/auth/login', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        body: JSON.stringify({ email, password })
    });

    const result = await res.json();
    if (result.token) {
        localStorage.setItem('token', result.token);
        localStorage.setItem('user', JSON.stringify(result.user));
        window.location.href = 'panel.html';
    } else {
        alert(result.error);
    }
    });
</script>
</body>
</html>
```

2. Panel de usuario (panel.html)

html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8" />
    <title>Panel de Usuario</title>
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="container py-5">
        <h2 class="mb-4">Panel de Control</h2>
        <p id="bienvenida" class="lead"></p>

        <div class="d-grid gap-2 col-6 mx-auto">
            <a href="productos.html" class="btn
btn-outline-primary">Gestión de Productos</a>
            <a href="clientes.html" class="btn
btn-outline-secondary">Gestión de Clientes</a>
            <a href="ventas.html" class="btn
btn-outline-success">Ventas</a>
            <a href="estadisticas.html" class="btn
btn-outline-dark">Estadísticas</a>
            <a id="admin-link" href="#" class="btn btn-outline-warning
d-none">Administrar Usuarios</a>
            <button onclick="logout()" class="btn btn-danger mt-4">Cerrar
Sesión</button>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    </div>
  </div>

  <script>
    const user = JSON.parse(localStorage.getItem('user'));
    if (!user) {
      window.location.href = 'login.html';
    }

    document.getElementById('bienvenida').textContent = `Hola,
    ${user.nombre} (${user.rol})`;

    if (user.rol === 'admin') {
      document.getElementById('admin-link').href = 'usuarios.html';
    }

    document.getElementById('admin-link').classList.remove('d-none');
  }

  function logout() {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    window.location.href = 'login.html';
  }
  </script>
</body>
</html>
```

🛡️ 3. Proteger otras páginas (ejemplo en **productos.html**)

Agrega al inicio del `<script>`:




```
const token = localStorage.getItem('token');
if (!token) {
  window.location.href = 'login.html';
}
```

Y al hacer `fetch`, añade siempre el header de autorización:

```
const res = await fetch(API_URL, {
  method: 'GET',
  headers: {
    'Authorization': 'Bearer ' + token
  }
});
```


Sistema de Gestión de Ventas en un Comercio, con **backend en Node.js, MySQL** y **frontend con Bootstrap**.

Resultado

-  Login funcional con control de acceso y token JWT
-  Panel adaptado por rol (**admin** puede ver más botones)
-  Rutas protegidas en frontend y backend

✅ FASE 4: GESTIÓN DE USUARIOS (solo admin)

¿Qué incluirá?

1. Ver listado de usuarios (nombre, email, rol)
 2. Crear nuevo usuario
 3. Editar o eliminar usuarios
 4. Proteger el acceso solo para administradores
-

🧩 1. BACKEND

🔧 routes/usuarios.js

Crea el archivo:

```
bash
CopiarEditar
touch routes/usuarios.js
```

Contenido:

```
js
CopiarEditar
const express = require('express');
const router = express.Router();
const controller = require('../controllers/usuariosController');
const auth = require('../middlewares/auth');

// Solo accesible con token y rol admin
router.use(auth);
router.get('/', controller.getAll);
router.post('/', controller.create);
router.put('/:id', controller.update);
router.delete('/:id', controller.remove);

module.exports = router;
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

controllers/usuariosController.js

bash

CopiarEditar

```
const db = require('../db');
const bcrypt = require('bcrypt');

exports.getAll = async (req, res) => {
  if (req.user.rol !== 'admin') return res.status(403).json({ error: 'No autorizado' });

  const [rows] = await db.query(`SELECT id, nombre, email, rol FROM usuarios`);
  res.json(rows);
};

exports.create = async (req, res) => {
  if (req.user.rol !== 'admin') return res.status(403).json({ error: 'No autorizado' });

  const { nombre, email, password, rol } = req.body;
  if (!nombre || !email || !password) return res.status(400).json({ error: 'Faltan datos' });

  try {
    const hashed = await bcrypt.hash(password, 10);
    await db.query(
      'INSERT INTO usuarios (nombre, email, password, rol) VALUES (?, ?, ?, ?)',
      [nombre, email, hashed, rol || 'vendedor']
    );
    res.json({ message: 'Usuario creado' });
  } catch {
    res.status(500).json({ error: 'Error al crear usuario' });
  }
};

exports.update = async (req, res) => {
  if (req.user.rol !== 'admin') return res.status(403).json({ error: 'No autorizado' });

  const { nombre, email, rol } = req.body;
  const { id } = req.params;
  try {
    await db.query(
      'UPDATE usuarios SET nombre = ?, email = ?, rol = ? WHERE id = ?',

```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        [nombre, email, rol, id]
    );
    res.json({ message: 'Usuario actualizado' });
  } catch {
    res.status(500).json({ error: 'Error al actualizar usuario' });
  }
};

exports.remove = async (req, res) => {
  if (req.user.rol !== 'admin') return res.status(403).json({ error: 'No autorizado' });

  const { id } = req.params;
  try {
    await db.query('DELETE FROM usuarios WHERE id = ?', [id]);
    res.json({ message: 'Usuario eliminado' });
  } catch {
    res.status(500).json({ error: 'Error al eliminar usuario' });
  }
};
```

Añadir en **server.js**

```
js
CopiarEditar
app.use('/api/usuarios', require('./routes/usuarios'));
```

2. FRONTEND – usuarios.html

Crea el archivo:

```
bash
CopiarEditar
touch frontend/usuarios.html
```

Contenido básico con Bootstrap:

```
html
CopiarEditar
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <title>Administración de Usuarios</title>
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body class="bg-light">
  <div class="container py-4">
    <h2 class="mb-4">Gestión de Usuarios</h2>
    <table class="table table-striped" id="tabla-usuarios">
      <thead class="table-dark">
        <tr>

<th>ID</th><th>Nombre</th><th>Email</th><th>Rol</th><th>Acciones</th>
>
        </tr>
      </thead>
      <tbody></tbody>
    </table>

    <h4 class="mt-4">Nuevo Usuario</h4>
    <form id="form-usuario" class="card p-3 shadow-sm">
      <input type="hidden" id="usuario-id" />
      <input type="text" class="form-control mb-2" id="nombre"
placeholder="Nombre" required />
      <input type="email" class="form-control mb-2" id="email"
placeholder="Correo" required />
      <input type="password" class="form-control mb-2" id="password"
placeholder="Contraseña" required />
      <select class="form-select mb-2" id="rol">
        <option value="vendedor">Vendedor</option>
        <option value="admin">Admin</option>
      </select>
      <button class="btn btn-success" type="submit">Guardar</button>
    </form>
  </div>

  <script>
    const token = localStorage.getItem('token');
    const API = 'http://localhost:3000/api/usuarios';

    async function cargarUsuarios() {
      const res = await fetch(API, {
        headers: { Authorization: 'Bearer ' + token }
      });
      const usuarios = await res.json();
      const tbody = document.querySelector('#tabla-usuarios tbody');
      tbody.innerHTML = '';
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
    usuarios.forEach(u => {
      tbody.innerHTML += `
        <tr>
          <td>${u.id}</td>
          <td>${u.nombre}</td>
          <td>${u.email}</td>
          <td>${u.rol}</td>
          <td>
            <button class="btn btn-sm btn-warning"
onclick="editar(${u.id}, '${u.nombre}', '${u.email}',
'${u.rol}')">Editar</button>
            <button class="btn btn-sm btn-danger"
onclick="eliminar(${u.id})">Eliminar</button>
          </td>
        </tr>`;
    });
  }

document.getElementById('form-usuario').addEventListener('submit',
async e => {
  e.preventDefault();
  const id = document.getElementById('usuario-id').value;
  const data = {
    nombre: document.getElementById('nombre').value,
    email: document.getElementById('email').value,
    password: document.getElementById('password').value,
    rol: document.getElementById('rol').value
  };

  const res = await fetch(id ? `${API}/${id}` : API, {
    method: id ? 'PUT' : 'POST',
    headers: { 'Content-Type': 'application/json',
Authorization: 'Bearer ' + token },
    body: JSON.stringify(data)
  });

  const result = await res.json();
  alert(result.message);
  document.getElementById('form-usuario').reset();
  cargarUsuarios();
});

function editar(id, nombre, email, rol) {
  document.getElementById('usuario-id').value = id;
  document.getElementById('nombre').value = nombre;
```

Sistema de Gestión de Ventas en un Comercio, con backend en Node.js, MySQL y frontend con Bootstrap.

```
        document.getElementById('email').value = email;
        document.getElementById('password').value = '';
        document.getElementById('rol').value = rol;
    }

    async function eliminar(id) {
        if (confirm('¿Eliminar este usuario?')) {
            const res = await fetch(`${API}/${id}`, {
                method: 'DELETE',
                headers: { Authorization: 'Bearer ' + token }
            });
            const result = await res.json();
            alert(result.message);
            cargarUsuarios();
        }
    }

    cargarUsuarios();
</script>
</body>
</html>
```

✓ Resultado

- ✓ Panel de administración exclusivo para **admin**
- ✓ Alta, edición y eliminación de usuarios
- ✓ Protección total con token y rol