

2. Backend con Angular 14

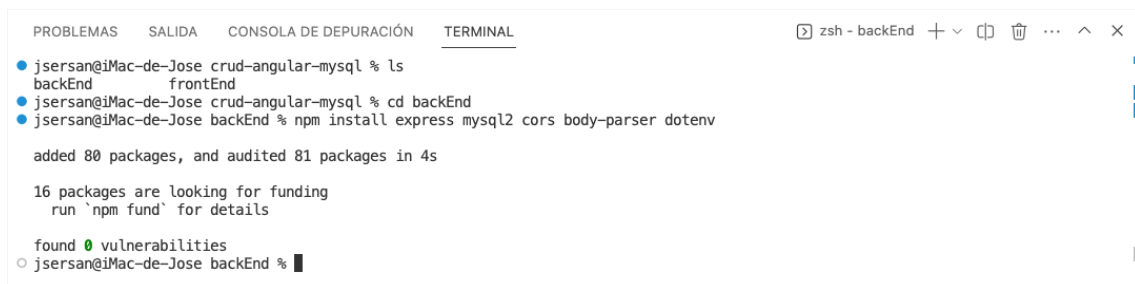
2.1 Crear la estructura del proyecto.

- Movemos todo a una carpeta frontEnd.
- Creamos la carpeta backEnd. Nos movemos dentro de ella:

```
jrsersan@iMac-de-Jose backEnd % pwd
/Applications/MAMP/htdocs/angular16/crud-angular-mysql/backEnd
jrsersan@iMac-de-Jose backEnd % npm init -y
Wrote to /Applications/MAMP/htdocs/angular16/crud-angular-mysql/backEnd/package.json:

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

2.2 Instalar dependencias necesarias:



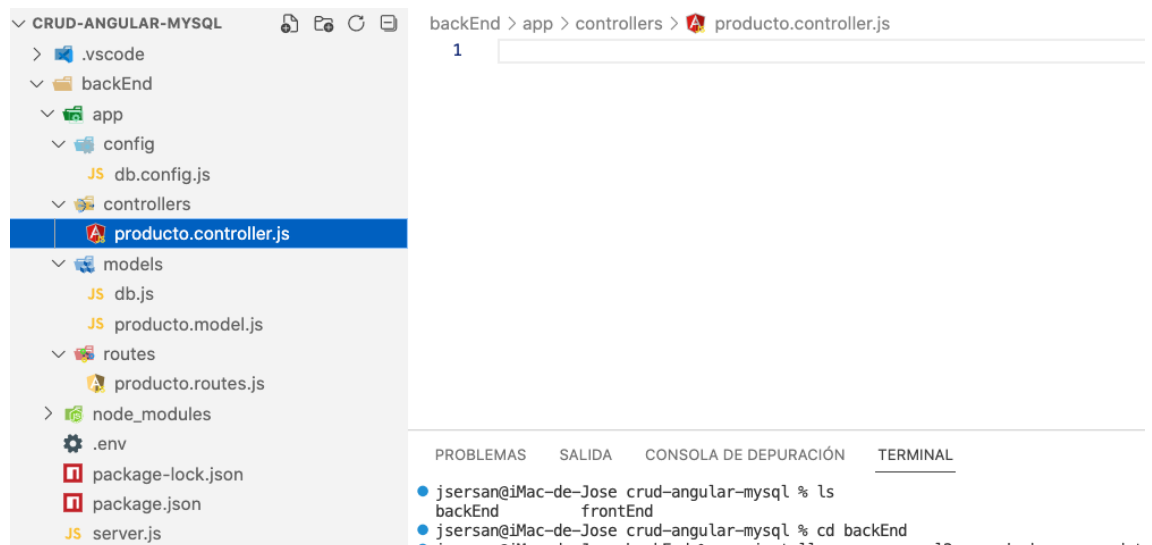
```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
jrsersan@iMac-de-Jose crud-angular-mysql % ls
backEnd      frontEnd
jrsersan@iMac-de-Jose crud-angular-mysql % cd backEnd
jrsersan@iMac-de-Jose backEnd % npm install express mysql2 cors body-parser dotenv

added 80 packages, and audited 81 packages in 4s

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

found 0 vulnerabilities
jrsersan@iMac-de-Jose backEnd %
```

2.3 Configurar la estructura de carpetas:



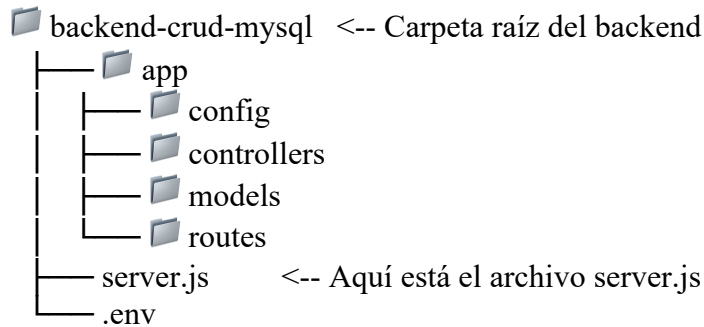
```
CRUD-ANGULAR-MYSQL
├── .vscode
├── backEnd
│   ├── app
│   │   ├── config
│   │   │   └── db.config.js
│   │   ├── controllers
│   │   │   └── producto.controller.js
│   │   ├── models
│   │   │   ├── db.js
│   │   │   └── producto.model.js
│   │   └── routes
│   │       └── producto.routes.js
├── node_modules
├── .env
├── package-lock.json
├── package.json
└── server.js

backEnd > app > controllers > producto.controller.js
1
```

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL
jrsersan@iMac-de-Jose crud-angular-mysql % ls
backEnd      frontEnd
jrsersan@iMac-de-Jose crud-angular-mysql % cd backEnd
jrsersan@iMac-de-Jose backEnd % npm install express mysql2 cors body-parser dot
```

2.3.1 server.js

Crear el archivo principal del servidor server.js. El archivo server.js se encuentra en la carpeta raíz del proyecto backend. Según la estructura que te mostré previamente, estaría ubicado así:



Este archivo contiene el punto de entrada principal para la aplicación Node.js/Express y es responsable de configurar el servidor Express, conectar middleware como CORS y body-parser, definir rutas básicas, e iniciar el servidor en el puerto especificado. También importa las rutas específicas de la aplicación desde la carpeta app/routes.

```

backEnd > JS server.js > ...
1  const express = require('express');
2  const cors = require('cors');
3  const bodyParser = require('body-parser');
4  require('dotenv').config();
5
6  const app = express();
7
8  // Configuración de CORS
9  app.use(cors({
10 |   origin: 'http://localhost:4200' // URL de tu aplicación Angular
11 | }));
12
13 // Parsear requests como JSON
14 app.use(bodyParser.json());
15 app.use(bodyParser.urlencoded({ extended: true }));
16
17 // Ruta simple para comprobar que el servidor está funcionando
18 app.get('/', (req, res) => {
19 |   res.json({ message: 'Bienvenido a la API de productos.' });
20 | });
21
22 // Importar rutas de productos
23 require('./app/routes/producto.routes')(app);
24
25 // Establecer puerto y escuchar
26 const PORT = process.env.PORT || 3000;
27 app.listen(PORT, () => {
28 |   console.log(`Servidor corriendo en el puerto ${PORT}.`);
29 | });
30
  
```

2.3.2 Configuración de la base de datos: app/config/db.config.js


```
backEnd > app > config > JS db.config.js > ...  
1  module.exports = {  
2    ...  
3    HOST: process.env.DB_HOST || 'localhost',  
4    USER: process.env.DB_USER || 'root',  
5    PASSWORD: process.env.DB_PASSWORD || '',  
6    DB: process.env.DB_NAME || 'crud_angular_mysql',  
7    dialect: 'mysql',  
8    pool: {  
9      max: 5,  
10     min: 0,  
11     acquire: 30000,  
12     idle: 10000  
13   }  
};
```

El archivo app/config/db.config.js es un componente crucial del backend ya que contiene la configuración para la conexión a la base de datos MySQL. Vamos a analizar su contenido detalladamente:

1. `module.exports = { ... }:` Exporta un objeto de configuración que será importado por otros módulos que necesiten acceder a la base de datos.
2. Parámetros de conexión básicos:
 - `HOST:` Dirección del servidor MySQL. Utiliza la variable de entorno `DB_HOST` si está disponible, o usa 'localhost' como valor predeterminado.
 - `USER:` Nombre de usuario para la conexión MySQL. Usa la variable de entorno `DB_USER` o el valor predeterminado 'root'.
 - `PASSWORD:` Contraseña para la conexión MySQL. Usa la variable de entorno `DB_PASSWORD` o una cadena vacía como valor predeterminado.
 - `DB:` Nombre de la base de datos. Usa la variable de entorno `DB_NAME` o 'crud_angular_mysql' como valor predeterminado.
 - `dialect:` Especifica que estamos usando MySQL como sistema de base de datos.
3. Configuración del pool de conexiones:
 - `pool:` Un objeto que configura el grupo (pool) de conexiones a la base de datos, lo que permite reutilizar conexiones y mejorar el rendimiento.
 - `max: 5:` Número máximo de conexiones en el pool.
 - `min: 0:` Número mínimo de conexiones en el pool.
 - `acquire: 30000:` Tiempo máximo en milisegundos que el pool intentará obtener una conexión antes de lanzar un error (30 segundos).
 - `idle: 10000:` Tiempo máximo en milisegundos que una conexión puede estar inactiva antes de ser liberada (10 segundos).

La configuración del pool de conexiones es importante para aplicaciones con múltiples usuarios simultáneos, ya que permite reutilizar conexiones existentes en lugar de crear nuevas para cada solicitud, mejorando así el rendimiento y la escalabilidad de la aplicación.

2.3.3 Archivo de variables de entorno .env:

```
backEnd >  .env
1  PORT=3000
2  DB_HOST=localhost
3  DB_USER=root
4  DB_PASSWORD=root
5  DB_NAME=crud_angular_mysql
```

2.3.4 Modelo de la base de datos app/models/db.js:

```
backEnd > app > models > JS db.js > ...
1  const mysql = require('mysql2');
2  const dbConfig = require('../config/db.config');
3
4  // Crear conexión con la base de datos
5  const connection = mysql.createConnection({
6    host: dbConfig.HOST,
7    user: dbConfig.USER,
8    password: dbConfig.PASSWORD,
9    database: dbConfig.DB
10 });
11
12 // Abrir la conexión MySQL
13 connection.connect(error => {
14   if (error) throw error;
15   console.log('Conexión a la base de datos establecida con éxito.');
```

2.3.5 Modelo para los productos app/models/producto.model.js:

```
1  // app/models/producto.model.js
2
3  // Importa la conexión a la base de datos configurada en db.js
4  const sql = require('./db');
5
6  /**
7   * Constructor para el modelo Producto
8   * Define la estructura de un objeto Producto para la aplicación
9   * @param {Object} producto - Objeto con las propiedades del producto
10  */
11  const Producto = function(producto) {
12    this.nombre = producto.nombre; // Nombre del producto
13    this.descripcion = producto.descripcion; // Descripción del producto
14    this.precio = producto.precio; // Precio del producto
15    this.stock = producto.stock; // Cantidad en stock
16  };
17
18  /**
19   * Método para crear un nuevo producto en la base de datos
20   * @param {Object} newProducto - Objeto con los datos del nuevo producto
21   * @param {Function} result - Función callback para manejar el resultado
22  */
23  Producto.create = (newProducto, result) => {
24    // Ejecuta una consulta SQL INSERT para agregar un nuevo producto
25    // El signo '?' es un marcador de posición que será reemplazado por newProducto
26    sql.query('INSERT INTO productos SET ?', newProducto, (err, res) => {
27      // Si hay un error durante la consulta
28      if (err) {
29        console.log('error:', err);
30        result(err, null); // Devuelve el error al controlador
31        return;
32      }
33    });
34  };
35
```

```

34     // Log de éxito y creación del producto
35     console.log('Producto creado:', { id: res.insertId, ...newProducto });
36     // Devuelve el producto creado con su ID asignado
37     result(null, { id: res.insertId, ...newProducto });
38   });
39 };
40
41 /**
42  * Método para buscar un producto por su ID
43  * @param {Number} id - ID del producto a buscar
44  * @param {Function} result - Función callback para manejar el resultado
45  */
46 Producto.findById = (id, result) => {
47   // Ejecuta una consulta SQL SELECT para encontrar un producto específico por ID
48   sql.query(`SELECT * FROM productos WHERE id = ${id}`, (err, res) => {
49     // Si hay un error durante la consulta
50     if (err) {
51       console.log('error:', err);
52       result(err, null);
53       return;
54     }
55
56     // Si se encontró al menos un registro (debería ser solo uno)
57     if (res.length) {
58       console.log('Producto encontrado:', res[0]);
59       result(null, res[0]); // Devuelve el primer resultado
60       return;
61     }
62
63     // Si no se encontró ningún producto con ese ID
64     result({ kind: 'not_found' }, null);
65   });
66 };
67
68 /**
69  * Método para obtener todos los productos, con filtro opcional por nombre
70  * @param {String} nombre - Nombre (parcial) para filtrar productos (opcional)
71  * @param {Function} result - Función callback para manejar el resultado
72  */
73 Producto.getAll = (nombre, result) => {
74   // Inicia con la consulta básica
75   let query = 'SELECT * FROM productos';
76
77   // Si se proporciona un nombre, agrega una cláusula WHERE para filtrar
78   if (nombre) {
79     // Usa LIKE para buscar coincidencias parciales (contiene)
80     query += ` WHERE nombre LIKE '%${nombre}%'`;
81   }
82
83   // Ejecuta la consulta SQL
84   sql.query(query, (err, res) => {
85     if (err) {
86       console.log('error:', err);
87       result(err, null);
88       return;
89     }
90
91     console.log('productos:', res);
92     result(null, res); // Devuelve todos los productos encontrados
93   });
94 };
95
96 /**
97  * Método para actualizar un producto existente por su ID
98  * @param {Number} id - ID del producto a actualizar
99  * @param {Object} producto - Objeto con los nuevos datos del producto
100  * @param {Function} result - Función callback para manejar el resultado
101  */

```

```

102 Producto.updateById = (id, producto, result) => {
103   // Ejecuta una consulta SQL UPDATE para modificar un producto
104   // Usa múltiples marcadores de posición '?' para los valores
105   sql.query(
106     'UPDATE productos SET nombre = ?, descripcion = ?, precio = ?, stock = ? WHERE id = ?',
107     [producto.nombre, producto.descripcion, producto.precio, producto.stock, id],
108     (err, res) => {
109       if (err) {
110         console.log('error:', err);
111         result(err, null);
112         return;
113       }
114
115       // Si no se actualizó ninguna fila (affectedRows = 0) significa que no se encontró el producto
116       if (res.affectedRows == 0) {
117         result({ kind: 'not_found' }, null);
118         return;
119       }
120
121       console.log('Producto actualizado:', { id: id, ...producto });
122       result(null, { id: id, ...producto });
123     }
124   );
125 };
126
127 /**
128  * Método para eliminar un producto por su ID
129  * @param {Number} id - ID del producto a eliminar
130  * @param {Function} result - Función callback para manejar el resultado
131  */
132 Producto.remove = (id, result) => {
133   // Ejecuta una consulta SQL DELETE para eliminar un producto
134   sql.query('DELETE FROM productos WHERE id = ?', id, (err, res) => {
135     if (err) {
136       console.log('error:', err);
137       result(err, null);
138       return;
139     }
140
141     // Si no se eliminó ninguna fila, significa que no se encontró el producto
142     if (res.affectedRows == 0) {
143       result({ kind: 'not_found' }, null);
144       return;
145     }
146
147     console.log('Producto eliminado con ID:', id);
148     result(null, res);
149   });
150 };
151
152 /**
153  * Método para eliminar todos los productos
154  * @param {Function} result - Función callback para manejar el resultado
155  */
156 Producto.removeAll = result => {
157   // Ejecuta una consulta SQL DELETE sin cláusula WHERE para eliminar todos los productos
158   sql.query('DELETE FROM productos', (err, res) => {
159     if (err) {
160       console.log('error:', err);
161       result(err, null);
162       return;
163     }
164
165     console.log(`${res.affectedRows} productos fueron eliminados`);
166     result(null, res);
167   });
168 };
169
170 // Exporta el modelo Producto para que pueda ser utilizado en otros archivos
171 module.exports = Producto;

```

El archivo **producto.model.js** es fundamental en la arquitectura del backend, ya que funciona como interfaz entre la aplicación y la base de datos para todas las operaciones relacionadas con los productos. Vamos a analizar sus componentes principales:

Define la estructura básica de un objeto Producto con sus propiedades: nombre, descripción, precio y stock. Este constructor se usa para crear nuevas instancias de Producto antes de guardarlas en la base de datos.

El modelo incluye varios métodos estáticos para realizar operaciones en la base de datos:

- **create**: Inserta un nuevo producto en la base de datos.
- **findById**: Busca un producto específico por su ID.
- **getAll**: Obtiene todos los productos, con la opción de filtrar por nombre.
- **updateById**: Actualiza un producto existente.
- **remove**: Elimina un producto específico.
- **removeAll**: Elimina todos los productos.

○ Consultas SQL y Seguridad

El modelo utiliza consultas SQL parametrizadas (con marcadores `?`) para proteger contra ataques de inyección SQL en la mayoría de los métodos.

Sin embargo, en `getAll` cuando se filtra por nombre, la concatenación directa podría ser vulnerable a inyección SQL. En una aplicación de producción, esto debería mejorarse.

○ Manejo de Callbacks

Cada método sigue un patrón consistente para el manejo de errores:

- Ejecuta una consulta SQL
- Verifica si hay errores
- Procesa los resultados
- Llama al callback pasando el error y/o los datos

Este patrón permite que los controladores manejen los resultados de manera uniforme.

○ Integración con el Resto del Sistema

Este modelo es usado por el controlador (`producto.controller.js`) para procesar las peticiones HTTP y enviar respuestas al cliente. La separación de responsabilidades es clara:

- El modelo se encarga de las operaciones de base de datos
- El controlador maneja la lógica de negocio y las respuestas HTTP
- Las rutas dirigen las peticiones HTTP al controlador adecuado

El modelo es un ejemplo de buenas prácticas en desarrollo backend, implementando una abstracción clara para la comunicación con la base de datos y proporcionando una API consistente para el resto de la aplicación.

2.3.6 Módulo controlador app/controllers/producto.controller.js:

```

1  // app/controllers/producto.controller.js
2
3  // Importa el modelo Producto para interactuar con la base de datos
4  const Producto = require('../models/producto.model');
5
6  /**
7   * Controlador para manejar las operaciones relacionadas con los productos
8   * Actúa como intermediario entre las rutas HTTP y el modelo de datos
9   */
10
11 /**
12  * Función para crear y guardar un nuevo producto
13  * Maneja peticiones POST a /api/productos
14  * @param {Object} req - Objeto de solicitud HTTP de Express
15  * @param {Object} res - Objeto de respuesta HTTP de Express
16  */
17 exports.create = (req, res) => {
18   // Valida que el cuerpo de la petición no esté vacío
19   if (!req.body) {
20     res.status(400).send({
21       message: 'El contenido no puede estar vacío!'
22     });
23     return;
24   }
25
26   // Crea un nuevo objeto Producto usando el constructor y los datos de la petición
27   const producto = new Producto({
28     nombre: req.body.nombre,
29     descripcion: req.body.descripcion,
30     precio: req.body.precio,
31     stock: req.body.stock
32   });
33
34   // Llama al método create del modelo para guardar en la base de datos
35   Producto.create(producto, (err, data) => {
36     if (err)
37       // Si hay un error, envía una respuesta con código 500 (Error del servidor)
38       res.status(500).send({
39         message:
40           err.message || 'Ocurrió un error al crear el producto.'
41       });
42     else
43       // Si la creación fue exitosa, envía los datos del producto creado
44       res.send(data);
45   });
46 };
47
48 /**
49  * Función para obtener todos los productos de la base de datos
50  * Opcionalmente puede filtrar por nombre si se proporciona un parámetro de consulta
51  * Maneja peticiones GET a /api/productos y /api/productos?nombre=xxx
52  * @param {Object} req - Objeto de solicitud HTTP
53  * @param {Object} res - Objeto de respuesta HTTP
54  */
55 exports.findAll = (req, res) => {
56   // Extrae el parámetro de consulta 'nombre' si existe
57   const nombre = req.query.nombre;
58
59   // Llama al método getAll del modelo para buscar productos
60   Producto.getAll(nombre, (err, data) => {
61     if (err)
62       // Si hay un error, envía una respuesta con código 500
63       res.status(500).send({
64         message:
65           err.message || 'Ocurrió un error al obtener los productos.'
66       });
67     else
68       // Si la búsqueda fue exitosa, envía los datos encontrados
69       res.send(data);
70   });
71 };

```



```
73  /**
74  * Función para encontrar un único producto por su ID
75  * Maneja peticiones GET a /api/productos/:id
76  * @param {Object} req - Objeto de solicitud HTTP con parámetro id
77  * @param {Object} res - Objeto de respuesta HTTP
78  */
79  exports.findOne = (req, res) => {
80    // Llama al método findById del modelo para buscar un producto específico
81    Producto.findById(req.params.id, (err, data) => {
82      if (err) {
83        if (err.kind === 'not_found') {
84          // Si no se encontró el producto, envía un código 404 (No encontrado)
85          res.status(404).send({
86            message: 'No se encontró el producto con id ${req.params.id}.'
87          });
88        } else {
89          // Para otros errores, envía un código 500
90          res.status(500).send({
91            message: 'Error al obtener el producto con id ' + req.params.id
92          });
93        }
94      } else
95        // Si se encontró el producto, envía los datos
96        res.send(data);
97    });
98  };
99
100 /**
101 * Función para actualizar un producto existente por su ID
102 * Maneja peticiones PUT a /api/productos/:id
103 * @param {Object} req - Objeto de solicitud HTTP con parámetro id y datos actualizados
104 * @param {Object} res - Objeto de respuesta HTTP
105 */
106 exports.update = (req, res) => {
107   // Valida que el cuerpo de la petición no esté vacío
108   if (!req.body) {
109     res.status(400).send({
110       message: 'El contenido no puede estar vacío!'
111     });
112     return;
113   }
114
115   // Llama al método updateById del modelo para actualizar el producto
116   Producto.updateById(
117     req.params.id,
118     new Producto(req.body),
119     (err, data) => {
120       if (err) {
121         if (err.kind === 'not_found') {
122           // Si no se encontró el producto, envía un código 404
123           res.status(404).send({
124             message: 'No se encontró el producto con id ${req.params.id}.'
125           });
126         } else {
127           // Para otros errores, envía un código 500
128           res.status(500).send({
129             message: 'Error al actualizar el producto con id ' + req.params.id
130           });
131         }
132       } else
133         // Si la actualización fue exitosa, envía los datos actualizados
134         res.send(data);
135     }
136   );
137 };
```

```

139  /**
140   * Función para eliminar un producto por su ID
141   * Maneja peticiones DELETE a /api/productos/:id
142   * @param {Object} req - Objeto de solicitud HTTP con parámetro id
143   * @param {Object} res - Objeto de respuesta HTTP
144   */
145  exports.delete = (req, res) => {
146    // Llama al método remove del modelo para eliminar el producto
147    Producto.remove(req.params.id, (err, data) => {
148      if (err) {
149        if (err.kind === 'not_found') {
150          // Si no se encontró el producto, envía un código 404
151          res.status(404).send({
152            message: `No se encontró el producto con id ${req.params.id}.`
153          });
154        } else {
155          // Para otros errores, envía un código 500
156          res.status(500).send({
157            message: `No se pudo eliminar el producto con id ` + req.params.id
158          });
159        }
160      } else {
161        // Si la eliminación fue exitosa, envía un mensaje de confirmación
162        res.send({ message: `El producto fue eliminado con éxito!` });
163      }
164    });
165  };
166
167  /**
168   * Función para eliminar todos los productos de la base de datos
169   * Maneja peticiones DELETE a /api/productos
170   * @param {Object} req - Objeto de solicitud HTTP
171   * @param {Object} res - Objeto de respuesta HTTP
172   */
173  exports.deleteAll = (req, res) => {
174    // Llama al método removeAll del modelo para eliminar todos los productos
175    Producto.removeAll((err, data) => {
176      if (err) {
177        // Si hay un error, envía una respuesta con código 500
178        res.status(500).send({
179          message:
180            err.message || 'Ocurrió un error al eliminar todos los productos.'
181        });
182      } else {
183        // Si la eliminación fue exitosa, envía un mensaje de confirmación
184        res.send({ message: `Todos los productos fueron eliminados con éxito!` });
185      }
186    });
187  };

```

El archivo `app/controllers/producto.controller.js` es crucial en la arquitectura del backend pues actúa como intermediario entre las rutas HTTP y el modelo de datos. Vamos a analizar su estructura y funcionalidad:

Estructura general del controlador

Este controlador implementa el patrón MVC (Modelo-Vista-Controlador) donde:

- El **modelo** (`producto.model.js`) maneja la interacción con la base de datos.
- El **controlador** (este archivo) maneja la lógica de la aplicación y las respuestas HTTP.
- La **vista** es manejada por el frontend en Angular.

Métodos del controlador

El controlador expone seis métodos principales que corresponden a las diferentes operaciones CRUD:

1. **create**: Maneja la creación de nuevos productos (POST)
 - Valida que el cuerpo de la petición no esté vacío
 - Crea un nuevo objeto Producto con los datos recibidos
 - Llama al modelo para guardar en la base de datos
 - Devuelve respuesta exitosa o error según corresponda
2. **findAll**: Obtiene todos los productos con filtro opcional por nombre (GET)
 - Extrae el parámetro de consulta nombre si existe
 - Llama al modelo para buscar los productos
 - Devuelve la lista de productos o un mensaje de error
3. **findOne**: Busca un producto específico por su ID (GET)
 - Obtiene el ID del producto de los parámetros de ruta
 - Llama al modelo para buscar el producto
 - Maneja el caso específico de "producto no encontrado" con estado 404
 - Devuelve el producto o mensajes de error apropiados
4. **update**: Actualiza un producto existente (PUT)
 - Valida que el cuerpo de la petición no esté vacío
 - Obtiene el ID del producto de los parámetros de ruta
 - Crea un objeto Producto con los datos actualizados
 - Llama al modelo para realizar la actualización
 - Maneja diferentes tipos de errores con códigos HTTP apropiados
5. **delete**: Elimina un producto específico (DELETE)
 - Obtiene el ID del producto de los parámetros de ruta
 - Llama al modelo para eliminar el producto
 - Maneja el caso de "producto no encontrado" con estado 404
 - Devuelve confirmación de éxito o mensajes de error
6. **deleteAll**: Elimina todos los productos (DELETE)
 - Llama al método del modelo para eliminar todos los registros
 - Devuelve confirmación de éxito o mensaje de error

Manejo de errores

El controlador implementa un manejo de errores consistente:

- Errores 400 (Bad Request) para peticiones inválidas
- Errores 404 (Not Found) cuando no se encuentra un recurso
- Errores 500 (Internal Server Error) para problemas de servidor

Integración con Express

Cada método del controlador recibe los objetos req (request) y res (response) de Express, lo que permite:

- Acceder a parámetros de ruta, consulta y cuerpo de la petición
- Enviar respuestas HTTP con códigos de estado apropiados
- Formatear las respuestas como JSON

Este controlador sigue buenas prácticas de desarrollo backend como:

- Separación clara de responsabilidades
- Manejo adecuado de errores
- Respuestas HTTP estandarizadas
- Validación de datos de entrada

En resumen, este archivo es el cerebro que procesa todas las solicitudes relacionadas con productos en la API, asegurando que las operaciones en la base de datos se realicen correctamente y devolviendo respuestas adecuadas al cliente.

2.3.8 Archivo de rutas app/routes/producto.routes.js:

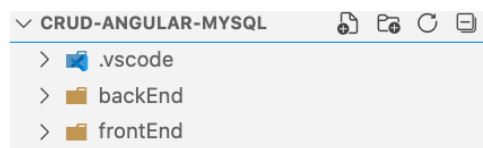
```
backEnd > app > routes >  producto.routes.js > ...
1  // Creamos las rutas para la API
2
3  module.exports = app => {
4    const productos = require('../controllers/producto.controller');
5
6    // Crear un nuevo producto
7    app.post('/api/productos', productos.create);
8
9    // Obtener todos los productos
10   app.get('/api/productos', productos.findAll);
11
12   // Obtener un solo producto por id
13   app.get('/api/productos/:id', productos.findOne);
14
15   // Actualizar un producto por id
16   app.put('/api/productos/:id', productos.update);
17
18   // Eliminar un producto por id
19   app.delete('/api/productos/:id', productos.delete);
20
21   // Eliminar todos los productos
22   app.delete('/api/productos', productos.deleteAll);
23 };
```

Guía Completa para CRUD MySQL

Requisitos previos

- Node.js (v14+) y npm instalados
- Angular CLI (v14) instalado: `npm install -g @angular/cli@14`
- MySQL (v5.7+) instalado y en ejecución

Estructura del proyecto



Paso 1: Configurar la base de datos MySQL

1. Inicia MySQL en tu sistema
2. Ejecuta el siguiente script SQL para crear la base de datos y la tabla:

```

backEnd > crud.sql
1  -- Crear la base de datos
2  CREATE DATABASE IF NOT EXISTS crud_angular_mysql;
3  USE crud_angular_mysql;
4
5  -- Crear la tabla de productos
6  CREATE TABLE IF NOT EXISTS productos (
7      id INT AUTO_INCREMENT PRIMARY KEY,
8      nombre VARCHAR(100) NOT NULL,
9      descripcion TEXT,
10     precio DECIMAL(10, 2) NOT NULL,
11     stock INT NOT NULL,
12     createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
13 );
14
15 -- Insertar algunos datos de ejemplo
16 INSERT INTO productos (nombre, descripcion, precio, stock) VALUES
17 ('Laptop Dell XPS 13', 'Laptop de alta gama con pantalla 4K', 1299.99, 15),
18 ('iPhone 15 Pro', 'Último modelo con cámara avanzada', 999.99, 25),
19 ('Monitor LG UltraWide', 'Monitor de 34 pulgadas curvo', 499.99, 10),
20 ('Teclado Mecánico Logitech', 'Teclado con switches Cherry MX', 129.99, 30),
21 ('Auriculares Sony WH-1000XM5', 'Cancelación de ruido premium', 349.99, 20);
  
```

Resultado:

✓ Mostrando filas 0 - 4 (total de 5, La consulta tardó 0.0006 segundos.)

`SELECT * FROM `productos``

☐ Perfilando [\[Editar en línea \]](#) [\[Editar \]](#) [\[Explicar SQL \]](#) [\[Crear código PHP \]](#) [\[Actualizar \]](#)

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla | Ordenar según la clave: Ninguna

Opciones extra

			id	nombre	descripcion	precio	stock	createdAt
<input type="checkbox"/>	Editar	Copiar	Borrar	1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	1299.99	15 2025-04-17 17:44:45
<input type="checkbox"/>	Editar	Copiar	Borrar	2	iPhone 15 Pro	Último modelo con cámara avanzada	999.99	25 2025-04-17 17:44:45
<input type="checkbox"/>	Editar	Copiar	Borrar	3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	499.99	10 2025-04-17 17:44:45
<input type="checkbox"/>	Editar	Copiar	Borrar	4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	129.99	30 2025-04-17 17:44:45
<input type="checkbox"/>	Editar	Copiar	Borrar	5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	349.99	20 2025-04-17 17:44:45

Paso 2: Configurar el Backend (Node.js + Express)

1. Creamos la carpeta backEnd. Nos movemos dentro de ella:

```

[jsersan@iMac-de-Jose backEnd % pwd
/Applications/MAMP/htdocs/angular16/crud-angular-mysql/backEnd
[jsersan@iMac-de-Jose backEnd % npm init -y
Wrote to /Applications/MAMP/htdocs/angular16/crud-angular-mysql/backEnd/package.json:

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

```

2. Instalar dependencias necesarias:

```

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
● jsersan@iMac-de-Jose crud-angular-mysql % ls
backEnd      frontEnd
● jsersan@iMac-de-Jose crud-angular-mysql % cd backEnd
● jsersan@iMac-de-Jose backEnd % npm install express mysql2 cors body-parser dotenv

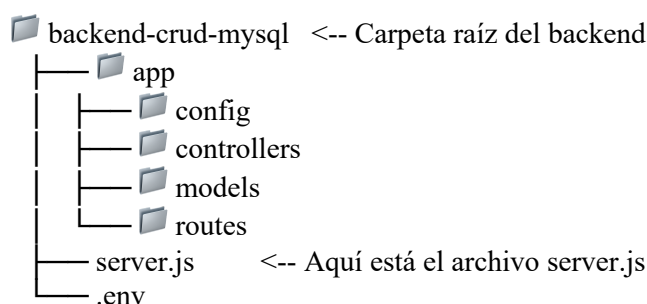
added 80 packages, and audited 81 packages in 4s

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

found 0 vulnerabilities
○ jsersan@iMac-de-Jose backEnd % █

```

3. Crea la siguiente estructura de directorio:



4. Crea el Archivo de variables de entorno .env:

```

backEnd > .env
1  PORT=3000
2  DB_HOST=localhost
3  DB_USER=root
4  DB_PASSWORD=root
5  DB_NAME=crud-angular-mysql

```

5. Crea los archivos según los códigos proporcionados en el artefacto anterior:

- app/config/db.config.js
- app/models/db.js
- app/models/producto.model.js
- app/controllers/producto.controller.js
- app/routes/producto.routes.js
- server.js

6. Inicia el servidor backend:

```
$ node server.js
```

Nos sale este error:

```
Node.js v18.20.4
jrsersan@iMac-de-Jose backEnd % node server.js
Servidor corriendo en el puerto 3000.
/Applications/MAMP/htdocs/angular16/crud-angular-mysql/backEnd/app/models/db.js:14
    if (error) throw error;
                ^
Error: connect ECONNREFUSED 127.0.0.1:3306
    at TCPConnectWrap.afterConnect [as oncomplete] (node:net:1278:16) {
  errno: -61,
  code: 'ECONNREFUSED',
  syscall: 'connect',
  address: '127.0.0.1',
  port: 3306,
```

Para solucionarlo en el fichero .env añado una entrada: DB_PORT

```
backEnd > .env
1  PORT=3000
2  DB_HOST=localhost
3  DB_PORT=8889
4  DB_USER=root
5  DB_PASSWORD=root
6  DB_NAME=crud_angular_mysql
```


En db.config.sys añado lo mismo:

```
backEnd > app > config > JS db.config.js > <unknown>
1  module.exports = {
2      HOST: process.env.DB_HOST || 'localhost',
3      USER: process.env.DB_USER || 'root',
4      PASSWORD: process.env.DB_PASSWORD || 'root',
5      DB: process.env.DB_NAME || 'crud_angular_mysql',
6      dialect: 'mysql',
7      port: 8889,
8      pool: {
9          max: 5,
10         min: 0,
11         acquire: 30000,
12         idle: 10000
13     }
14 };
```

Por último, en `models/db.js` añado una entrada para el puerto:

```
backEnd > app > models > JS db.js > ...
1  const mysql = require('mysql2');
2  const dbConfig = require('../config/db.config');
3
4  // Crear conexión con la base de datos
5  const connection = mysql.createConnection({
6    host: dbConfig.HOST,
7    user: dbConfig.USER,
8    password: dbConfig.PASSWORD,
9    database: dbConfig.DB,
10   port: dbConfig.port || 8889
11 });
12
13 // Abrir la conexión MySQL
14 connection.connect(error => {
15   if (error) throw error;
16   console.log('Conexión a la base de datos establecida con éxito.');
```

Lanzamos el servidor en el backend:



```
backEnd — node server.js — 124x53
jsersan@iMac-de-Jose backEnd % node server.js
Servidor corriendo en el puerto 3000.
Conexión a la base de datos establecida con éxito.
█
```

Paso 3: Configurar el Frontend (Angular 14)

1. Crea un nuevo proyecto Angular:

Ejecutamos los siguientes comandos:

```
$ ng new crud-angular-mysql --routing --style=scss
$ cd crud-angular-mysql
```

2. Instala las dependencias adicionales:

```
npm install bootstrap
npm install @angular/material @angular/cdk @angular/animations
```


3. Agrega los estilos de Bootstrap y Angular Material en `src/styles.scss`:

```
frontEnd > src > styles.scss > ...
1  /* You can add global styles to this file, and also import other style files */
2  @import "~bootstrap/dist/css/bootstrap.min.css";
3  @import "~@angular/material/prebuilt-themes/indigo-pink.css";
4
5  html, body { height: 100%; }
6  body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }
7
8  .mat-form-field {
9    margin-bottom: 15px;
10 }
```

4. Actualiza `angular.json` para incluir los estilos:

```
angular.json x JS db.config.js producto.controller.js JS db.js JS produ
frontEnd > angular.json > ...
116 ],
117 "styles": [
118   "src/styles.scss"
119 ],
120 "scripts": []
121 }
122 }
123 }
124 }
125 }
126 }
```

5. Crea la estructura de componentes:

```
$ ng generate component components/productos-list
$ ng generate component components/producto-detalle
$ ng generate component components/producto-form
$ ng generate service services/producto
$ ng generate interface models/producto
```

6. Implementa el código de cada archivo según los artefactos proporcionados:

- Modelo (`src/app/models/producto.model.ts`)
- Servicio (`src/app/services/producto.service.ts`)
- Módulo principal (`src/app/app.module.ts`)
- Rutas (`src/app/app-routing.module.ts`)
- Componentes (todos los archivos en las carpetas de componentes)
- Plantilla principal (`src/app/app.component.html`)

7. Inicia el servidor de desarrollo:

```
$ ng serve
```

Tenemos el servidor:

```
jrsersan@iMac-de-Jose backEnd % cd /Applications/MAMP/htdocs/angular16/crud-angular-mysql/frontEnd
jrsersan@iMac-de-Jose frontEnd % ng serve
✓ Browser application bundle generation complete.

Initial Chunk Files | Names          | Raw Size
vendor.js           | vendor         | 4.12 MB
styles.css, styles.js | styles         | 468.30 kB
polyfills.js        | polyfills      | 234.36 kB
main.js             | main           | 72.02 kB
runtime.js          | runtime        | 6.51 kB

Initial Total       | 4.88 MB

Build at: 2025-04-18T15:46:54.944Z - Hash: e281c9287783a189 - Time: 23890ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
```

Abrimos el navegador. Sale este error:

```
✖ ERROR Error: Uncaught (in promise): Error: NG0300: Multiple components match node with tagname button: MatButtonModule and MatIconButton. Find more at https://angular.io/errors/NG0300
Error: NG0300: Multiple components match node with tagname button: MatButtonModule and MatIconButton. Find more at https://angular.io/errors/NG0300
    at throwMultipleComponentError (core.mjs:8502:11)
    at findDirectiveDefMatches (core.mjs:11060:29)
    at resolveDirectives (core.mjs:10860:29)
    at elementStartFirstCreatePass (core.mjs:13543:5)
    at Module.ɵɵelementStart (core.mjs:13579:9)
    at ProductosListComponent_Template (productos-list.component.html:22:11)
    at executeTemplate (core.mjs:10441:9)
    at renderView (core.mjs:10263:13)
    at renderComponent (core.mjs:11434:5)
    at renderChildComponents (core.mjs:10122:9)
    at throwMultipleComponentError (core.mjs:8502:11)
    at findDirectiveDefMatches (core.mjs:11060:29)
    at resolveDirectives (core.mjs:10860:29)
    at elementStartFirstCreatePass (core.mjs:13543:5)
    at Module.ɵɵelementStart (core.mjs:13579:9)
    at ProductosListComponent_Template (productos-list.component.html:22:11)
    at executeTemplate (core.mjs:10441:9)
    at renderView (core.mjs:10263:13)
    at renderComponent (core.mjs:11434:5)
    at renderChildComponents (core.mjs:10122:9)
    at resolvePromise (zone.js:1214:31)
    at resolvePromise (zone.js:1168:17)
    at zone.js:1201:17
    at _ZoneDelegate.invokeTask (zone.js:409:31)
    at core.mjs:23896:55
    at AsyncStackTaggingZoneSpec.onInvokeTask (core.mjs:23896:36)
    at _ZoneDelegate.invokeTask (zone.js:408:60)
    at Object.onInvokeTask (core.mjs:24197:33)
    at _ZoneDelegate.invokeTask (zone.js:408:60)
    at Zone.runTask (zone.js:178:47)
```

Este error indica un conflicto entre dos componentes de Angular Material que intentan asociarse al mismo elemento HTML `<button>`. El mensaje específicamente menciona que tanto `MatButtonModule` como `MatIconModule` están intentando aplicarse a un botón en tu plantilla.

En `product-list.component.ts` del frontend modificamos la línea 22:

```
21 | | | | | <!-- Botón de búsqueda ubicado en el sufijo del campo -->
22 | | | | | <button mat-button matSuffix mat-icon-button (click)="buscarPorNombre()">
23 | | | | | | <!-- Ícono de búsqueda (requiere MatIconModule) -->
24 | | | | | | <mat-icon>search</mat-icon>
25 | | | | | </button>
```

Por:

```
21 | | | | | <!-- Botón de búsqueda ubicado en el sufijo del campo -->
22 | | | | | <button mat-icon-button matSuffix (click)="buscarPorNombre()">
23 | | | | | | <!-- Ícono de búsqueda (requiere MatIconModule) -->
24 | | | | | | <mat-icon>search</mat-icon>
```

Recargamos:

CRUD Angular-MySQL

Productos

Lista de Productos

Nuevo Producto

ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	ed de
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	ed de
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	ed de
4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	\$129.99	30	ed de
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	ed de

Items per page: 5
1 - 5 of 5

Vemos que están mal los iconos de los botones y del input text. Es posible que tengamos problema con la carga de los iconos de Material Design. Para que los iconos edit y delete aparezcan correctamente, necesitamos asegurarnos de que tenemos configurada la fuente de iconos de Material Design en el proyecto. Tenemos importada la librería `matIconModule` en `app.module.ts` por lo que este no era el problema:

```

31 imports: [
32   BrowserModule,
33   AppRoutingModule,
34   FormsModule,
35   ReactiveFormsModule,
36   HttpClientModule,
37   BrowserAnimationsModule,
38   // Módulos de Angular Material
39   MatFormFieldModule,
40   MatInputModule, // Este es crucial para mat-form-field
41   MatButtonModule,
42   MatTableModule,
43   MatPaginatorModule,
44   MatIconModule,
45   MatDialogModule,
46   MatSnackBarModule
47 ],

```

Asegúrate de que has incluido la fuente de iconos de Material Design en tu archivo `index.html` o en `styles.scss`. Añade esto a tu `index.html` dentro de la etiqueta `<head>`:


```

frontEnd > src > index.html > html > head > link
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>CrudAngularMysql</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9   <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
10 </head>











```



Ya ahora tenemos:

Lista de Productos




[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	 
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	 
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	 
4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	\$129.99	30	 
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	 











Items per page: 5 1 - 5 of 5  



Probamos a insertar dos nuevos registros:

Lista de Productos

[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	 
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	 
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	 
4	Teclado Mecánico Logitech		\$99.99	30	 
5	Auriculares Sony WH-1000XM5		\$99.99	20	 

Items per page: 5 1 - 5 of 5  

Nuevo Producto

Nombre*
Samsung Galaxy S10 plus

Descripción*
Teléfono móvil para personas inquietas


Precio*
\$ 600

Stock*
10













[Cancelar](#) [Guardar](#)

Tras pulsar el botón Guardar:

Lista de Productos





[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	 
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	 
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	 
4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	\$129.99	30	 
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	 
6	Samsung Galaxy S10 plus	Teléfono móvil para personas inquietas	\$600.00	10	 


Items per page:

10











 1 – 6 of 6  

Cambiamos el número de ítems por página:

Lista de Productos





[+ Nuevo Producto](#)


ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	 
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	 
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	 
4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	\$129.99	30	 
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	 

Items per page:



5

 1 – 5 of 6  

Lista de Productos





[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
6	Samsung Galaxy S10 plus	Teléfono móvil para personas inquietas	\$600.00	10	 

Items per page:

5

 6 – 6 of 6  

Actualizamos el stock a 20 unidades:

Lista de Productos

Buscar por nombre

[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
6	Samsung Galaxy S10 plus	Teléfono móvil para personas inquietas	\$600.00	10	✎ 🗑

Items per page: 6 – 6 of 6 |< < > >|

Editar Producto

Nombre*

Descripción*

Precio* Stock*

Ahora:

Lista de Productos

Buscar por nombre

[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
6	Samsung Galaxy S10 plus	Teléfono móvil para personas inquietas	\$600.00	20	✎ 🗑

Items per page: 6 – 6 of 6 |< < > >|

Lo eliminamos:

← → ↺ 🔍 ☆ 🔄 🏠 48 📁 | En pausa

📁 AJAX 📁 Angular 📁 Angular Node CR... 📁 Arimazubi 📁 Mozilla 📁 Common Creative... >> | 📁 Todos los marcados

CRUD Angular-MySQL Productos

localhost:4200 dice

¿Está seguro de eliminar este producto?

Lista de Productos

Buscar por nombre


[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
6	Samsung Galaxy S10 plus	Teléfono móvil para personas inquietas	\$600.00	20	✎ 🗑











Items per page: 6 – 6 of 6 |< < > >|



Tras decirle que sí:

Lista de Productos




[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
1	Laptop Dell XPS 13	Laptop de alta gama con pantalla 4K	\$1,299.99	15	 
2	iPhone 15 Pro	Último modelo con cámara avanzada	\$999.99	25	 
3	Monitor LG UltraWide	Monitor de 34 pulgadas curvo	\$499.99	10	 
4	Teclado Mecánico Logitech	Teclado con switches Cherry MX	\$129.99	30	 
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	 



Items per page: 5 1 – 5 of 5  

Probamos el filtrado. Buscamos Sony. Y le damos a la lupa:

Lista de Productos



[+ Nuevo Producto](#)

ID	Nombre	Descripción	Precio	Stock	Acciones
5	Auriculares Sony WH-1000XM5	Cancelación de ruido premium	\$349.99	20	 

Items per page: 5 1 – 1 of 1 