# Introducción

El proyecto EcoMerch API es una aplicación web desarrollada para gestionar productos y clientes de una tienda en línea. La aplicación está construida utilizando tecnologías modernas como Jakarta EE, Jersey para la implementación de servicios RESTful, y MySQL como base de datos. Este informe detalla la estructura del proyecto, su funcionamiento, y justifica cada parte del mismo.

**Estructura del Proyecto**

La estructura del proyecto es la siguiente:

.project

.classpath

.factorypath

.settings/

.jsdtscope

org.eclipse.core.resources.prefs

org.eclipse.jdt.apt.core.prefs

org.eclipse.jdt.core.prefs

org.eclipse.m2e.core.prefs

org.eclipse.wst.common.component

org.eclipse.wst.common.project.facet.core.prefs.xml

org.eclipse.wst.common.project.facet.core.xml

org.eclipse.wst.jsdt.ui.superType.container

org.eclipse.wst.jsdt.ui.superType.name

org.eclipse.wst.validation.prefs

.vscode/

settings.json

pom.xml

sql/

ecomerch.sql

src/

main/

java/

tfg/

App.java

conexion/

ConexionBdd.java

dao/

ClienteDAO.java

ProductoDAO.java

modulos/

Cliente.java

Producto.java

recursos/

RecursosCliente.java

RecursosProducto.java

resources/

webapp/

css/

styles.css

js/

app.js

WEB-INF/

web.xml

index.html

test/

java/

tfg/

AppTest.java

target/

classes/

test-classes/

maven-archiver/

maven-status/

m2e-wtp/

ecomerch-rest-1.0-SNAPSHOT/

# Archivos y Directorios Clave

* **.classpath, .factorypath, .project, .settings/**: Archivos y directorios de configuración del entorno de desarrollo Eclipse.
* **pom.xml**: Archivo de configuración de Maven que gestiona las dependencias y el ciclo de vida del proyecto.
* **sql/**: Contiene scripts SQL para la base de datos.
* **src/**: Contiene el código fuente de la aplicación.
* **target/**: Directorio generado por Maven que contiene los archivos compilados y empaquetados.

# Configuración del Proyecto

## pom.xml

El archivo pom.xml es fundamental para la gestión del proyecto. Define las dependencias necesarias, las propiedades del proyecto y los plugins de Maven utilizados.

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

    <modelVersion>4.0.0</modelVersion>

    <groupId>tfg</groupId>

    <artifactId>ecomerch-rest</artifactId>

    <version>1.0-SNAPSHOT</version>

    <packaging>war</packaging>

    <properties>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

        <maven.compiler.source>17</maven.compiler.source>

        <maven.compiler.target>17</maven.compiler.target>

        <jersey.version>3.1.3</jersey.version>

        <jakarta.version>10.0.0</jakarta.version>

    </properties>

    <dependencies>

        <!-- Dependencias de Jakarta EE y Jersey -->

        ...

        <!-- Dependencias de la base de datos -->

        <dependency>

            <groupId>mysql</groupId>

            <artifactId>mysql-connector-java</artifactId>

            <version>8.0.33</version>

        </dependency>

        <!-- Dependencias de testing -->

        <dependency>

            <groupId>org.junit.jupiter</groupId>

            <artifactId>junit-jupiter</artifactId>

            <version>5.10.1</version>

            <scope>test</scope>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-war-plugin</artifactId>

                <version>3.4.0</version>

            </plugin>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.11.0</version>

            </plugin>

        </plugins>

    </build>

</project>

# Base de Datos

El proyecto utiliza MySQL como base de datos. El archivo ecomerch.sql contiene un script SQL para seleccionar todos los productos de la base de datos.

# CREATE DATABASE ecomerch;

# USE ecomerch;

# CREATE TABLE productos (

# id BIGINT PRIMARY KEY AUTO\_INCREMENT,

# nombre VARCHAR(100) NOT NULL,

# descripcion TEXT,

# precio DECIMAL(10,2) NOT NULL,

# stock INT NOT NULL

# );

# CREATE TABLE clientes (

# id BIGINT PRIMARY KEY AUTO\_INCREMENT,

# nombre VARCHAR(100) NOT NULL,

# email VARCHAR(100) NOT NULL,

# telefono VARCHAR(20),

# direccion TEXT

# );

# Código Fuente

## Paquete [tfg.modulos](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Este paquete contiene las clases de modelo Cliente y Producto.

## [Cliente.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.modulos;

public class Cliente {

    private Long id;

    private String nombre;

    private String email;

    private String direccion;

    private String telefono;

    // Constructor, Getters y Setters

}

## [Producto.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.modulos;

public class Producto {

    private Long id;

    private String nombre;

    private String descripcion;

    private double precio;

    private int stock;

    // Constructor, Getters y Setters

}

## Paquete [tfg.conexion](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Este paquete contiene la clase ConexionBdd que gestiona la conexión a la base de datos.

## [ConexionBdd.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.conexion;

import java.sql.Connection;

import java.sql.DriverManager;

public class ConexionBdd {

    private static final String URL = "jdbc:mysql://localhost:3306/ecomerch";

    private static final String USER = "root";

    private static final String PASSWORD = "Tcachuk93";

    public static Connection getConnection() {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            return DriverManager.getConnection(URL, USER, PASSWORD);

        } catch (Exception e) {

            throw new RuntimeException("Error conectando a la base de datos", e);

        }

    }

}

## Paquete [tfg.dao](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Este paquete contiene las clases DAO (Data Access Object) para Cliente y Producto.

## [ClienteDAO.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.dao;

import tfg.modulos.Cliente;

import tfg.conexion.ConexionBdd;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class ClienteDAO {

    private Connection connection;

    public ClienteDAO() {

        this.connection = ConexionBdd.getConnection();

    }

    public Cliente crear(Cliente cliente) throws SQLException {

        String sql = "INSERT INTO clientes (nombre, email, direccion, telefono) VALUES (?, ?, ?, ?)";

        try (PreparedStatement statement = connection.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS)) {

            statement.setString(1, cliente.getNombre());

            statement.setString(2, cliente.getEmail());

            statement.setString(3, cliente.getDireccion());

            statement.setString(4, cliente.getTelefono());

            statement.executeUpdate();

            ResultSet rs = statement.getGeneratedKeys();

            if (rs.next()) {

                cliente.setId(rs.getLong(1));

            }

        }

        return cliente;

    }

    public Cliente obtenerPorId(Long id) throws SQLException {

        String sql = "SELECT \* FROM clientes WHERE id = ?";

        try (PreparedStatement statement = connection.prepareStatement(sql)) {

            statement.setLong(1, id);

            ResultSet rs = statement.executeQuery();

            if (rs.next()) {

                return extraerClienteDeResultSet(rs);

            }

        }

        return null;

    }

    private Cliente extraerClienteDeResultSet(ResultSet rs) throws SQLException {

        Cliente cliente = new Cliente();

        cliente.setId(rs.getLong("id"));

        cliente.setNombre(rs.getString("nombre"));

        cliente.setEmail(rs.getString("email"));

        cliente.setDireccion(rs.getString("direccion"));

        cliente.setTelefono(rs.getString("telefono"));

        return cliente;

    }

    public List<Cliente> getAllClientes() {

        List<Cliente> clientes = new ArrayList<>();

        String query = "SELECT \* FROM clientes";

        try (PreparedStatement ps = connection.prepareStatement(query);

             ResultSet rs = ps.executeQuery()) {

            while (rs.next()) {

                Cliente cliente = new Cliente();

                cliente.setId(rs.getLong("id"));

                cliente.setNombre(rs.getString("nombre"));

                cliente.setEmail(rs.getString("email"));

                cliente.setDireccion(rs.getString("direccion"));

                cliente.setTelefono(rs.getString("telefono"));

                clientes.add(cliente);

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

        return clientes;

    }

    public boolean updateCliente(Cliente cliente) {

        String query = "UPDATE clientes SET nombre=?, email=?, direccion=?, telefono=? WHERE id=?";

        try (Connection conn = ConexionBdd.getConnection();

             PreparedStatement ps = conn.prepareStatement(query)) {

            ps.setString(1, cliente.getNombre());

            ps.setString(2, cliente.getEmail());

            ps.setString(3, cliente.getDireccion());

            ps.setString(4, cliente.getTelefono());

            ps.setLong(5, cliente.getId());

            return ps.executeUpdate() > 0;

        } catch (SQLException e) {

            e.printStackTrace();

            return false;

        }

    }

    public boolean deleteCliente(int id) {

        String query = "DELETE FROM clientes WHERE id=?";

        try (PreparedStatement ps = connection.prepareStatement(query)) {

            ps.setInt(1, id);

            return ps.executeUpdate() > 0;

        } catch (SQLException e) {

            e.printStackTrace();

            return false;

        }

    }

}

[ProductoDAO.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.dao;

import tfg.modulos.Producto;

import tfg.conexion.ConexionBdd;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class ProductoDAO {

    public List<Producto> getAllProductos() throws SQLException {

        List<Producto> productos = new ArrayList<>();

        String query = "SELECT \* FROM productos";

        try (Connection conn = ConexionBdd.getConnection();

             Statement stmt = conn.createStatement();

             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {

                Producto producto = new Producto(

                    rs.getLong("id"),

                    rs.getString("nombre"),

                    rs.getString("descripcion"),

                    rs.getDouble("precio"),

                    rs.getInt("stock")

                );

                productos.add(producto);

            }

        }

        return productos;

    }

    public Producto getProductoById(long id) throws SQLException {

        Producto producto = null;

        String query = "SELECT \* FROM productos WHERE id = ?";

        try (Connection conn = ConexionBdd.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setLong(1, id);

            try (ResultSet rs = pstmt.executeQuery()) {

                if (rs.next()) {

                    producto = new Producto(

                        rs.getLong("id"),

                        rs.getString("nombre"),

                        rs.getString("descripcion"),

                        rs.getDouble("precio"),

                        rs.getInt("stock")

                    );

                }

            }

        }

        return producto;

    }

    public void createProducto(Producto producto) throws SQLException {

        String query = "INSERT INTO productos (nombre, descripcion, precio, stock) VALUES (?, ?, ?, ?)";

        try (Connection conn = ConexionBdd.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setString(1, producto.getNombre());

            pstmt.setString(2, producto.getDescripcion());

            pstmt.setDouble(3, producto.getPrecio());

            pstmt.setInt(4, producto.getStock());

            pstmt.executeUpdate();

        }

    }

    public void updateProducto(long id, Producto producto) throws SQLException {

        String query = "UPDATE productos SET nombre = ?, descripcion = ?, precio = ?, stock = ? WHERE id = ?";

        try (Connection conn = ConexionBdd.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setString(1, producto.getNombre());

            pstmt.setString(2, producto.getDescripcion());

            pstmt.setDouble(3, producto.getPrecio());

            pstmt.setInt(4, producto.getStock());

            pstmt.setLong(5, id);

            pstmt.executeUpdate();

        }

    }

    public void deleteProducto(long id) throws SQLException {

        String query = "DELETE FROM productos WHERE id = ?";

        try (Connection conn = ConexionBdd.getConnection();

             PreparedStatement pstmt = conn.prepareStatement(query)) {

            pstmt.setLong(1, id);

            pstmt.executeUpdate();

        }

    }

}

## Paquete [tfg.recursos](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

Este paquete contiene las clases de recursos RESTful para Cliente y Producto.

## [RecursosCliente.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.recursos;

import tfg.dao.ClienteDAO;

import tfg.modulos.Cliente;

import java.util.List;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

@Path("/clientes")

public class RecursosCliente {

    private ClienteDAO clienteDAO = new ClienteDAO();

    @GET

    @Path("/test")

    @Produces(MediaType.APPLICATION\_JSON)

    public String test() {

        return "{\"mensaje\":\"API funcionando\"}";

    }

    @POST

    @Consumes(MediaType.APPLICATION\_JSON)

    @Produces(MediaType.APPLICATION\_JSON)

    public Response crearCliente(Cliente cliente) {

        try {

            Cliente nuevoCliente = clienteDAO.crear(cliente);

            return Response.status(Response.Status.CREATED)

                         .entity(nuevoCliente)

                         .build();

        } catch (Exception e) {

            return Response.status(Response.Status.INTERNAL\_SERVER\_ERROR)

                         .entity("Error al crear el cliente: " + e.getMessage())

                         .build();

        }

    }

    @GET

    @Path("/{id}")

    @Produces(MediaType.APPLICATION\_JSON)

    public Response obtenerCliente(@PathParam("id") Long id) {

        try {

            Cliente cliente = clienteDAO.obtenerPorId(id);

            if (cliente != null) {

                return Response.ok(cliente).build();

            } else {

                return Response.status(Response.Status.NOT\_FOUND).build();

            }

        } catch (Exception e) {

            return Response.status(Response.Status.INTERNAL\_SERVER\_ERROR)

                         .entity("Error al obtener el cliente: " + e.getMessage())

                         .build();

        }

    }

    @GET

    @Path("/all")

    @Produces(MediaType.APPLICATION\_JSON)

    public Response getAllClientes() {

        try {

            List<Cliente> clientes = clienteDAO.getAllClientes();

            if (clientes != null && !clientes.isEmpty()) {

                return Response.ok(clientes).build();

            } else {

                return Response.status(Response.Status.NOT\_FOUND)

                             .entity("{\"mensaje\":\"No se encontraron clientes\"}")

                             .build();

            }

        } catch (Exception e) {

            return Response.serverError()

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

    @PUT

    @Path("/{id}")

    @Consumes(MediaType.APPLICATION\_JSON)

    @Produces(MediaType.APPLICATION\_JSON)

    public Response updateCliente(@PathParam("id") long id, Cliente cliente) {

        cliente.setId(id);

        boolean actualizado = clienteDAO.updateCliente(cliente);

        if (actualizado) {

            return Response.ok(cliente).build();

        }

        return Response.status(404).build();

    }

    @DELETE

    @Path("/{id}")

    @Produces(MediaType.APPLICATION\_JSON)

    public Response deleteCliente(@PathParam("id") int id) {

        boolean eliminado = clienteDAO.deleteCliente(id);

        if (eliminado) {

            return Response.ok()

                          .entity("{\"mensaje\":\"Cliente eliminado correctamente\"}")

                          .build();

        }

        return Response.status(404)

                      .entity("{\"error\":\"Cliente no encontrado\"}")

                      .build();

    }

}

[RecursosProducto.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg.recursos;

import java.util.List;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

import tfg.dao.ProductoDAO;

import tfg.modulos.Producto;

@Path("/productos")

public class RecursosProducto {

    private ProductoDAO productoDAO = new ProductoDAO();

    @GET

    @Path("/test")

    @Produces(MediaType.APPLICATION\_JSON)

    public String test() {

        return "{\"mensaje\":\"API de productos funcionando\"}";

    }

    @GET

    @Path("/all")

    @Produces(MediaType.APPLICATION\_JSON)

    public Response getAllProductos() {

        try {

            List<Producto> productos = productoDAO.getAllProductos();

            if (!productos.isEmpty()) {

                return Response.ok(productos).build();

            } else {

                return Response.status(Response.Status.NOT\_FOUND)

                             .entity("{\"mensaje\":\"No hay productos disponibles\"}")

                             .build();

            }

        } catch (Exception e) {

            return Response.serverError()

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

    @GET

    @Path("/{id}")

    @Produces(MediaType.APPLICATION\_JSON)

    public Response getProducto(@PathParam("id") long id) {

        try {

            Producto producto = productoDAO.getProductoById(id);

            if (producto != null) {

                return Response.ok(producto).build();

            } else {

                return Response.status(Response.Status.NOT\_FOUND)

                             .entity("{\"mensaje\":\"Producto no encontrado\"}")

                             .build();

            }

        } catch (Exception e) {

            return Response.serverError()

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

    @POST

    @Consumes(MediaType.APPLICATION\_JSON)

    @Produces(MediaType.APPLICATION\_JSON)

    public Response crear(Producto producto) {

        try {

            productoDAO.createProducto(producto);

            return Response.status(Response.Status.CREATED)

                         .entity(producto)

                         .build();

        } catch (Exception e) {

            return Response.status(Response.Status.BAD\_REQUEST)

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

    @PUT

    @Path("/{id}")

    @Consumes(MediaType.APPLICATION\_JSON)

    @Produces(MediaType.APPLICATION\_JSON)

    public Response updateProducto(@PathParam("id") long id, Producto producto) {

        try {

            productoDAO.updateProducto(id, producto);

            return Response.ok(producto).build();

        } catch (Exception e) {

            return Response.serverError()

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

    @DELETE

    @Path("/{id}")

    public Response deleteProducto(@PathParam("id") long id) {

        try {

            productoDAO.deleteProducto(id);

            return Response.ok()

                         .entity("{\"mensaje\":\"Producto eliminado correctamente\"}")

                         .build();

        } catch (Exception e) {

            return Response.serverError()

                         .entity("{\"error\":\"" + e.getMessage() + "\"}")

                         .build();

        }

    }

}

## Paquete [tfg](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

## [App.java](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

package tfg;

import jakarta.ws.rs.ApplicationPath;

import jakarta.ws.rs.core.Application;

@ApplicationPath("/api")

public class App extends Application {

}

# Interfaz de Usuario

La interfaz de usuario está construida utilizando HTML, CSS y JavaScript. Se utiliza Bootstrap para el diseño responsivo y estilización.

## [index.html](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

El archivo index.html define la estructura principal de la interfaz de usuario. A continuación, se detallan las secciones clave del archivo:

* **Encabezado y Metadatos**: Se incluyen metadatos esenciales como el conjunto de caracteres y la configuración de la vista para dispositivos móviles. Además, se enlazan los estilos de Bootstrap y la hoja de estilos personalizada.

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>EcoMerch API</title>

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

    <link href="css/styles.css" rel="stylesheet">

</head>

* **Barra de Navegación**: Se utiliza una barra de navegación de Bootstrap para permitir la navegación entre las secciones de productos y clientes.

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

    <div class="container">

        <a class="navbar-brand" href="#">EcoMerch API</a>

        <div class="collapse navbar-collapse">

            <ul class="navbar-nav">

                <li class="nav-item">

                    <a class="nav-link" href="#" onclick="loadProductos()">Productos</a>

                </li>

                <li class="nav-item">

                    <a class="nav-link" href="#" onclick="loadClientes()">Clientes</a>

                </li>

            </ul>

        </div>

    </div>

</nav>

* **Panel de Control**: Incluye un formulario para realizar peticiones a la API y una barra de búsqueda para buscar registros por ID.

<div class="col-md-6">

    <!-- Barra de búsqueda -->

    <div class="card mb-3">

        <div class="card-body">

            <div class="row g-3 align-items-center">

                <div class="col-auto">

                    <label for="searchId" class="col-form-label">Buscar por ID:</label>

                </div>

                <div class="col-auto">

                    <input type="number" id="searchId" class="form-control">

                </div>

                <div class="col-auto">

                    <button class="btn btn-primary" onclick="searchById()">Buscar</button>

                    <button class="btn btn-secondary ms-2" onclick="showAll()">Ver todos</button>

                </div>

            </div>

        </div>

    </div>

    <!-- Formulario API -->

    <div class="card mb-4">

        <div class="card-header">

            <h5 class="mb-0">Request</h5>

        </div>

        <div class="card-body">

            <form id="apiForm">

                <div class="mb-3">

                    <label>Método</label>

                    <select class="form-select" id="method">

                        <option value="POST">POST</option>

                        <option value="PUT">PUT</option>

                        <option value="DELETE">DELETE</option>

                    </select>

                </div>

                <div class="mb-3">

                    <label>ID (para PUT/DELETE)</label>

                    <input type="number" class="form-control" id="entityId">

                </div>

                <div class="mb-3">

                    <label>Datos (JSON)</label>

                    <textarea class="form-control" id="requestData" rows="5"></textarea>

                </div>

                <button type="submit" class="btn btn-primary">Enviar</button>

            </form>

        </div>

    </div>

</div>

* **Panel de Respuesta**: Muestra la respuesta de la API en un área de texto.

<div class="col-md-6">

    <div class="card mb-4">

        <div class="card-header">

            <h5 class="mb-0">Response</h5>

        </div>

        <div class="card-body">

            <pre id="responseArea" class="bg-light p-3"></pre>

        </div>

    </div>

</div>

* **Tabla de Datos**: Muestra los datos obtenidos de la API en una tabla.

<div class="card">

    <div class="card-header d-flex justify-content-between">

        <h5 id="tableTitle" class="mb-0">Datos</h5>

        <button class="btn btn-primary" onclick="showAddForm()">Nuevo</button>

    </div>

    <div class="card-body">

        <div class="table-responsive">

            <table class="table" id="dataTable">

                <thead id="tableHead"></thead>

                <tbody id="tableBody"></tbody>

            </table>

        </div>

    </div>

</div>

## [styles.css](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

El archivo styles.css contiene estilos personalizados para mejorar la presentación visual de la aplicación.

.table th {

    white-space: nowrap;

}

.btn-group-xs > .btn,

.btn-xs {

    padding: .25rem .4rem;

    font-size: .875rem;

    line-height: .5;

    border-radius: .2rem;

}

pre {

    max-height: 400px;

    overflow: auto;

}

.card {

    box-shadow: 0 0.125rem 0.25rem  rgba(0, 0, 0, 0.075);

}

#response-data {

    font-family: 'Consolas', monospace;

    font-size: 0.875rem;

}

## [app.js](vscode-file://vscode-app/c:/Users/CAMPUSFP/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html)

El archivo app.js contiene la lógica JavaScript para interactuar con la API y actualizar la interfaz de usuario.

* **Variables Globales**: Define la URL base de la API y el endpoint actual.

var API\_URL = 'http://localhost:8080/ecomerch-rest/api';

var currentEndpoint = 'productos';

* **Función para Ejecutar Peticiones**: Realiza peticiones a la API utilizando el método especificado y los datos proporcionados.

function executeRequest(method, url, data = null) {

    var options = {

        method: method,

        headers: {

            'Content-Type': 'application/json',

            'Accept': 'application/json'

        }

    };

    if (data && (method === 'POST' || method === 'PUT')) {

        options.body = JSON.stringify(data);

    }

    fetch(url, options)

        .then(function(response) {

            if (!response.ok) {

                throw new Error('Error en la petición: ' + response.status);

            }

            return response.json();

        })

        .then(function(responseData) {

            updateResponsePanel(responseData);

            loadTableData();

        })

        .catch(function(error) {

            console.error('Error:', error);

            updateResponsePanel({error: error.message});

        });

}

* **Función para Cargar Datos en la Tabla**: Obtiene los datos del endpoint actual y actualiza la tabla.

function loadTableData() {

    fetch(API\_URL + '/' + currentEndpoint + '/all')

        .then(function(response) {

            return response.json();

        })

        .then(function(data) {

            updateTable(data);

        })

        .catch(function(error) {

            console.error('Error:', error);

        });

}

* **Manejador del Formulario**: Captura los datos del formulario y realiza la petición correspondiente a la API.

function handleFormSubmit(e) {

    e.preventDefault();

    var method = document.getElementById('method').value;

    var id = document.getElementById('entityId').value;

    var requestData = document.getElementById('requestData').value;

    try {

        var jsonData = requestData ? JSON.parse(requestData) : null;

        var url = API\_URL + '/' + currentEndpoint;

        if (id && (method === 'PUT' || method === 'DELETE')) {

            url += '/' + id;

        }

        executeRequest(method, url, jsonData);

        if (method !== 'GET') {

            document.getElementById('entityId').value = '';

            document.getElementById('requestData').value = '';

        }

    } catch (e) {

        updateResponsePanel({error: 'Error en formato JSON: ' + e.message});

    }

}

* **Función para Buscar por ID**: Realiza una petición GET para obtener un registro específico por ID.

function searchById() {

    var id = document.getElementById('searchId').value;

    if (!id) {

        updateResponsePanel({error: 'Por favor, ingrese un ID'});

        return;

    }

    fetch(API\_URL + '/' + currentEndpoint + '/' + id)

        .then(function(response) {

            if (!response.ok) {

                throw new Error('No se encontró el elemento con ID: ' + id);

            }

            return response.json();

        })

        .then(function(data) {

            updateResponsePanel(data);

            var dataArray = [data];

            updateTable(dataArray);

            highlightRow(id);

        })

        .catch(function(error) {

            updateResponsePanel({error: error.message});

            loadTableData();

        });

}

* **Función para Resaltar la Fila**: Resalta la fila de la tabla correspondiente al ID buscado.

function highlightRow(id) {

    var rows = document.querySelectorAll('#tableBody tr');

    rows.forEach(function(row) {

        if (row.cells[0].textContent === id.toString()) {

            row.classList.add('table-primary');

            setTimeout(function() {

                row.classList.remove('table-primary');

            }, 2000);

        }

    });

}

* **Función para Mostrar Todos los Registros**: Restaura la vista completa de la tabla.

function showAll() {

    document.getElementById('searchId').value = '';

    loadTableData();

}

* **Función para Actualizar la Tabla**: Actualiza el contenido de la tabla con los datos obtenidos de la API.

function updateTable(data) {

    var tbody = document.getElementById('tableBody');

    var content = '';

    data.forEach(function(item) {

        content += '<tr>';

        if (currentEndpoint === 'productos') {

            content += `

                <td>${item.id || ''}</td>

                <td>${item.nombre || ''}</td>

                <td>${item.descripcion || ''}</td>

                <td>${item.precio || 0}€</td>

                <td>${item.stock || 0}</td>

            `;

        } else {

            content += `

                <td>${item.id || ''}</td>

                <td>${item.nombre || ''}</td>

                <td>${item.email || ''}</td>

                <td>${item.telefono || ''}</td>

                <td>${item.direccion || ''}</td>

            `;

        }

        content += `

            <td>

                <button class="btn btn-sm btn-primary me-2" onclick="editItem(${item.id})">

                    Editar

                </button>

                <button class="btn btn-sm btn-danger" onclick="deleteItem(${item.id})">

                    Eliminar

                </button>

            </td>

        </tr>`;

    });

    tbody.innerHTML = content;

    var table = document.getElementById('dataTable');

    table.classList.add('table-update');

    setTimeout(function() {

        table.classList.remove('table-update');

    }, 500);

}

* **Funciones CRUD**: Permiten editar y eliminar registros.

function editItem(id) {

    fetch(API\_URL + '/' + currentEndpoint + '/' + id)

        .then(function(response) {

            return response.json();

        })

        .then(function(data) {

            document.getElementById('method').value = 'PUT';

            document.getElementById('entityId').value = id;

            document.getElementById('requestData').value = JSON.stringify(data, null, 2);

            updateResponsePanel(data);

        });

}

function deleteItem(id) {

    if (confirm('¿Confirma eliminar este elemento?')) {

        executeRequest('DELETE', API\_URL + '/' + currentEndpoint + '/' + id);

    }

}

* **Funciones para Cargar Productos y Clientes**: Cambian el endpoint actual y cargan los datos correspondientes.

function loadProductos() {

    currentEndpoint = 'productos';

    document.getElementById('tableTitle').textContent = 'Productos';

    loadTableData();

}

function loadClientes() {

    currentEndpoint = 'clientes';

    document.getElementById('tableTitle').textContent = 'Clientes';

    loadTableData();

}

* **Función para Mostrar el Formulario de Añadir**: Prepara el formulario para añadir un nuevo registro.

function showAddForm() {

    document.getElementById('method').value = 'POST';

    document.getElementById('entityId').value = '';

    var template = currentEndpoint === 'productos'

        ? {

            nombre: '',

            descripcion: '',

            precio: 0,

            stock: 0

          }

        : {

            nombre: '',

            email: '',

            telefono: '',

            direccion: ''

          };

    document.getElementById('requestData').value = JSON.stringify(template, null, 2);

}

* **Inicialización**: Vincula eventos y carga los datos iniciales.

document.addEventListener('DOMContentLoaded', function() {

    document.getElementById('apiForm').addEventListener('submit', handleFormSubmit);

    loadProductos();

});

document.head.insertAdjacentHTML('beforeend', `

    <style>

        .table-update {

            animation: tableHighlight 0.5s ease-out;

        }

        @keyframes tableHighlight {

            0% { background-color: #e8f4f8; }

            100% { background-color: transparent; }

        }

    </style>

`);

# Resultado del proyecto

## Productos

### Get

* **Resultado esperado**: Lista de todos los productos en formato JSON
* **Verificación**: La tabla se actualiza mostrando todos los productos disponibles

### Get(Buscado por ID)

* **Resultado esperado:** Datos del producto específico
* **Verificación:** Los datos coinciden con el producto buscado

### Post

Content-Type: application/json

{

    "nombre": "Producto Test",

    "descripcion": "Descripción del producto de prueba",

    "precio": 29.99,

    "stock": 100

}

* **Resultado esperado**: Producto creado y respuesta 201 (Created)
* **Verificación**: El producto aparece en la tabla y se puede buscar por su ID

### Put

Content-Type: application/json

{

    "nombre": "Producto Actualizado",

    "descripcion": "Nueva descripción",

    "precio": 39.99,

    "stock": 150

}

* **Resultado esperado**: Producto actualizado y respuesta 200 (OK)
* **Verificación**: Los cambios se reflejan en la tabla

### Delete

* **Resultado esperado:** Producto eliminado y respuesta 200 (OK)
* **Verificación:** El producto ya no aparece en la tabla

## Clientes

### Get

* **Resultado esperado**: Lista de todos los clientes en formato JSON
* **Verificación**: La tabla muestra todos los clientes registrados

### Get(Buscar por ID)

* **Resultado esperado:** Datos del cliente específico
* **Verificación:** Los datos coinciden con el cliente buscado

### Post

Content-Type: application/json

{

    "nombre": "Cliente Test",

    "email": "test@email.com",

    "telefono": "666555444",

    "direccion": "Calle Test, 123"

}

* **Resultado esperado**: Cliente creado y respuesta 201 (Created)
* **Verificación**: El cliente aparece en la tabla y se puede buscar por su ID

### Put

### Content-Type: application/json

### {

### "nombre": "Cliente Actualizado",

### "email": "actualizado@email.com",

### "telefono": "666777888",

### "direccion": "Nueva Dirección, 456"

### }

* **Resultado esperado:** Cliente actualizado y respuesta 200 (OK)
* **Verificación:** Los cambios se reflejan en la tabla

### Delete

* **Resultado esperado**: Cliente eliminado y respuesta 200 (OK)
* **Verificación**: El cliente ya no aparece en la tabla

# 3. Pruebas de Errores

## 3.1 Búsqueda de ID inexistente

* Intentar obtener un producto/cliente con ID que no existe
* **Resultado esperado**: Respuesta 404 (Not Found)

## 3.2 Datos inválidos en creación/actualización

* Enviar JSON mal formado o con campos faltantes
* **Resultado esperado**: Respuesta 400 (Bad Request)

## 3.3 Operaciones sin datos requeridos

* Intentar actualizar sin proporcionar datos
* **Resultado esperado**: Mensaje de error apropiado