# Práctica 2 - E-commerce con GraphQL y Gestión de Pedidos

**Universidad Europea del Atlántico**


ENLACE AL REPOSITORIO:

**https://github.com/DavidGarciaCosta/Practica2-ProgramacionWeb**

**En el repositorio hay un archivo .md con la documentación actualizada con las instrucciones de las Queries/Mutations de GraphQL.**

**David García Costa**

**09 de enero de 2026**

# Table of Contents

## Tabla de Contenidos



# 1. Introducción

## 1.1 Descripción del Proyecto

Este proyecto implementa un portal de productos evolucionado a E-commerce completo, incorporando GraphQL para la gestión de datos, un sistema de carrito de compras con persistencia, y un panel de administración robusto para la gestión de usuarios y pedidos.

La aplicación cumple con todos los requisitos establecidos en la Práctica 2, incluyendo:

- Sistema completo de carrito de compras
- Gestión de pedidos vía GraphQL
- Panel de administrador con CRUD de usuarios
- Integración de GraphQL con API REST existente
- Persistencia de datos con MongoDB

## 1.2 Objetivos Cumplidos

**Ampliación del panel de administrador:**

- Gestión completa de usuarios (CRUD)

- Visualización y filtrado de pedidos

- Estadísticas de ventas e ingresos

**Área de usuario (cliente):**

- Carrito de compra funcional

- Persistencia en LocalStorage

- Simulación de compra con formulario completo

**Integración con GraphQL:**

- Servidor Apollo Server Express

- Queries y Mutations completas

- Convivencia con API REST

**Persistencia y modelos de datos:**

- Modelo Order completo con todos los campos requeridos

- Relaciones entre User, Product y Order

# 2. Instalación y Configuración

## 2.1 Requisitos Previos

- **Node.js:** v18 o superior
- **MongoDB:** v6 o superior
- **NPM:** v9 o superior

## 2.2 Pasos de Instalación

**Paso 1: Clonar el repositorio**

```
git clone https://github.com/DavidGarciaCosta/Practica2-ProgramacionWeb
cd Practica2
```

**Paso 2: Instalar dependencias**

```
npm install
```

**Paso 3: Configurar variables de entorno**

Crear archivo `.env` en la raíz del proyecto:

```
MONGODB_URI=mongodb://localhost:27017/practica2-ecommerce
JWT_SECRET=clave_secreta_super_segura
PORT=3000
```

**Paso 4: Iniciar MongoDB**

**Windows:**

```
net start MongoDB
```

**Linux/Mac:**

```
sudo systemctl start mongod
```

**Paso 5: Ejecutar el proyecto**

```
# Modo desarrollo
npm run dev

# Modo producción
npm start
```

**Paso 6: Acceder a la aplicación**

- **Frontend:** http://localhost:3000
- **GraphQL Playground:** http://localhost:3000/graphql

# 3. Tecnologías Utilizadas

## 3.1 Backend

| Tecnología | Versión | Propósito |
|---|---|---|
| Node.js | 18+ | Runtime JavaScript |
| Express | 4.18.2 | Framework web |
| Apollo Server Express | 3.13.0 | Servidor GraphQL |
| MongoDB | 6+ | Base de datos NoSQL |
| Mongoose | 8.0.3 | ODM para MongoDB |
| JWT | 9.0.2 | Autenticación |
| Socket.IO | 4.6.0 | Comunicación en tiempo real |
| bcryptjs | 2.4.3 | Encriptación de passwords |

## 3.2 Frontend

| Tecnología | Propósito |
|---|---|
| HTML5 | Estructura |
| CSS3 | Estilos y diseño responsive |
| JavaScript ES6+ | Lógica del cliente |
| LocalStorage API | Persistencia del carrito |
| Fetch API | Comunicaciones HTTP |

## 3.3 GraphQL

- **Cliente:** Implementación personalizada con Fetch API
- **Schema:** TypeDefs con 7 Queries y 8 Mutations
- **Autenticación:** Context con JWT

# 4. Funcionalidades Implementadas

## 4.1 Funcionalidades de Práctica 1 (Mantenidas)

- Autenticación JWT (registro y login)
- Sistema de roles (user y admin)
- CRUD completo de productos
- Chat en tiempo real con Socket.IO

## 4.2 Nuevas Funcionalidades de Práctica 2

### 4.2.1 Panel de Administrador

**Gestión de Usuarios:**

- Listar todos los usuarios registrados

- Cambiar rol de usuarios (user - admin)

- Eliminar usuarios con protección anti-auto-eliminación

- Visualización de información completa (username, email, rol, fecha)

**Gestión de Pedidos:**

- Visualizar todos los pedidos de la plataforma

- Filtrar pedidos por estado (pending, completed, cancelled)

- Ver detalle completo de cada pedido

- Estadísticas: total de pedidos, ingresos, pedidos por estado

### 4.2.2 Área de Usuario

**Sistema de Carrito:**

- Botón “Añadir al Carrito” en cada producto

- Visualización completa del carrito con:

- Lista de productos con imagen, nombre, precio, cantidad

- Cálculo automático de subtotal y total

- Opciones para modificar cantidades - Botón para eliminar productos

- Persistencia en LocalStorage (se mantiene entre sesiones)

- Badge visual mostrando cantidad de items

**Proceso de Compra:**

- Formulario completo de dirección de envío: - Dirección - Ciudad - Código postal - País - Notas adicionales (opcional)

- Validación de formulario

- Creación de pedido vía GraphQL

- Vaciado automático del carrito tras compra exitosa

- Redirección a página “Mis Pedidos”

**Mis Pedidos:**

- Visualización de historial de pedidos

- Estado de cada pedido

- Detalle completo de items comprados

- Número de pedido único generado automáticamente

#### 4.2.3 Integración GraphQL

**Servidor Apollo Server:**

- Endpoint `/graphql` configurado

- GraphQL Playground para pruebas

- Context con autenticación JWT

- Manejo de errores robusto

**Queries Implementadas:**

- `products`

- Lista de productos con filtros

- `product(id)`

- Producto individual

- `orders(status)`

- Todos los pedidos (admin)

- `order(id)`

- Detalle de pedido - `myOrders` - Pedidos del usuario - `users` - Lista de usuarios (admin) - `orderStats` - Estadísticas (admin)

**Mutations Implementadas:** - `createOrder` - Crear pedido - `updateOrderStatus` - Cambiar estado (admin) - `cancelOrder` - Cancelar pedido - `updateUserRole` - Cambiar rol (admin) - `deleteUser` - Eliminar usuario (admin) - `createProduct` - Crear

producto (admin) - `updateProductStock` - Actualizar stock (admin) - `deleteProduct` - Eliminar producto (admin)

# 5. Documentación de GraphQL

## 5.1 Endpoint y Autenticación

### Endpoint:

```
POST http://localhost:3000/graphql
```

### Header de Autenticación:

```
Authorization: Bearer <token_jwt>
```

## 5.2 Queries Principales

#### Query 1: Obtener Productos

```
query GetProducts {
  products {
    id
    name
    description
    price
    category
    image
    stock
    createdAt
  }
}
```

### Con filtros:

```
query GetProducts($category: String, $search: String) {
  products(category: $category, search: $search) {
    id
    name
    price
    stock
  }
}
```

### Variables:

```
{
  "category": "Electrónica",
  "search": "laptop"
}
```

**Query 2: Obtener Mis Pedidos**

```
query GetMyOrders {
  myOrders {
    id
    items {
      name
      price
      quantity
      image
    }
    total
    status
    shippingAddress {
      address
      city
      postalCode
      country
    }
    orderNumber
    itemCount
    createdAt
  }
}
```

**Requiere:** Token JWT del usuario autenticado

**Query 3: Obtener Todos los Pedidos (Admin)**

```
query GetOrders($status: String) {
  orders(status: $status) {
    id
    user {
      username
      email
    }
    items {
      name
      price
      quantity
    }
    total
    status
    orderNumber
    createdAt
  }
}
```

**Variables (opcional):**

```
{
  "status": "pending"
}
```

**Requiere:** Token JWT de administrador

## 5.3 Mutations Principales

**Mutation 1: Crear Pedido (FUNCIONALIDAD PRINCIPAL)**

```
mutation CreateOrder($input: CreateOrderInput!) {
  createOrder(input: $input) {
    success
    message
    order {
      id
      orderNumber
      total
      status
      createdAt
    }
  }
}
```

**Variables:**

```
{
  "input": {
    "items": [
      {
        "product": "507f1f77bcf86cd799439011",
        "name": "Laptop HP Pavilion",
        "price": 799.99,
        "quantity": 1,
        "image": "https://example.com/laptop.jpg"
      }
    ],
    "total": 799.99,
    "shippingAddress": {
      "address": "Calle Mayor 123",
      "city": "Bilbao",
      "postalCode": "48001",
      "country": "España"
    },
    "notes": "Dejar en portería"
  }
}
```

**Validaciones automáticas:** - Verifica stock disponible - Valida que los precios no hayan cambiado - Valida el total calculado - Actualiza stock de productos automáticamente

**Mutation 2: Actualizar Estado de Pedido (Admin)**

```
mutation UpdateOrderStatus($orderId: ID!, $status: String!) {
  updateOrderStatus(orderId: $orderId, status: $status) {
```

```
    success
    message
    order {
      id
      status
      orderNumber
    }
  }
}
```

**Variables:**

```
{
  "orderId": "67def123456789abcdef0123",
  "status": "completed"
}
```

**Estados válidos:** `pending`, `completed`, `cancelled`

**Mutation 3: Cambiar Rol de Usuario (Admin)**

```
mutation UpdateUserRole($userId: ID!, $role: String!) {
  updateUserRole(userId: $userId, role: $role) {
    success
    message
    user {
      id
      username
      role
    }
  }
}
```

**Variables:**

```
{
  "userId": "507f1f77bcf86cd799439020",
  "role": "admin"
}
```

**Roles válidos:** `user`, `admin`

# 6. Decisiones Técnicas

## 6.1 Arquitectura General

### ¿Por qué Apollo Server Express?

**Ventajas:**

- Integración perfecta con Express existente

- GraphQL Playground integrado para desarrollo

- Excelente manejo de context para autenticación

- Schema-first approach facilita el mantenimiento

- Amplia comunidad y documentación

**Implementación:** El servidor Apollo convive con las rutas REST existentes, permitiendo una migración gradual y manteniendo compatibilidad con funcionalidades anteriores.

#### ¿Por qué LocalStorage para el carrito?

**Ventajas:**

- Persistencia entre sesiones del navegador

- No requiere autenticación para navegar productos

- Operaciones síncronas y rápidas

- Reduce carga en el servidor

- Experiencia de usuario fluida

**Consideraciones:**

- Límite de 5-10MB suficiente para un carrito típico

- Se sincroniza con el servidor al crear el pedido

- Validaciones de stock en servidor al finalizar compra

#### ¿Por qué convivencia REST + GraphQL?

**REST mantiene:** - Endpoints de autenticación (`/api/auth/login`, `/api/auth/register`) - Rutas administrativas REST como fallback - Compatibilidad con código existente de Práctica 1

**GraphQL maneja:** - Queries complejas de productos y pedidos - Mutations para crear y gestionar pedidos - Operaciones que requieren múltiples recursos relacionados

**Beneficios:** - Transición gradual sin romper funcionalidad - Cada tecnología usada donde más brilla - Flexibilidad para elegir el mejor enfoque por caso de uso

## 6.2 Seguridad

#### Autenticación y Autorización

**JWT (JSON Web Tokens):** - Tokens firmados con clave secreta - Expiración configurable - Incluye información del usuario (id, role)

**Context en GraphQL:**

```
context: ({ req }) => {
  const token = req.headers.authorization?.replace('Bearer ', '');
  if (token) {
    const user = jwt.verify(token, config.jwtSecret);
    return { user };
  }
  return { user: null };
}
```

**Middleware de autorización:**

- `authenticateJWT`: Verifica token válido

- `requireAdmin`: Verifica rol de administrador

- Validaciones en resolvers de GraphQL

#### Validaciones de Datos

**En creación de pedidos:**

1. Verificar stock disponible para cada producto

2. Validar que los precios no hayan cambiado

3. Calcular total en servidor (no confiar en cliente)

4. Verificar que el usuario está autenticado

**En gestión de usuarios:**

- Impedir auto-eliminación de admin

- Validar roles permitidos

- Hash de passwords con bcrypt

## 6.3 Modelo de Datos

#### Modelo Order

**Decisión:** Guardar snapshot de productos en el pedido

**Razones:**

- Preservar información histórica (nombre, precio)

- Independencia de cambios futuros en productos

- Auditoría correcta de precios aplicados

- Facilita reportes históricos

**Estructura:**

```
items: [{
  product: ObjectId,      // Referencia para stock
  name: String,           // Snapshot
  price: Number,          // Snapshot
  quantity: Number,
  image: String           // Snapshot
}]
```

#### Virtuals en Mongoose

**orderNumber:** Genera identificador único legible

```
orderNumber.get(function() {
  const year = this.createdAt.getFullYear().toString().slice(-2);
  const month = (this.createdAt.getMonth() + 1).toString().padStart(2,
'0');
  const idPart = this._id.toString().slice(-6).toUpperCase();
  return `ORD-${year}${month}-${idPart}`;
});
```

**itemCount:** Calcula cantidad total de items

```
itemCount.get(function() {
  return this.items.reduce((sum, item) => sum + item.quantity, 0);
});
```

## 6.4 Performance

#### Indexes en MongoDB

```
// Order model
orderSchema.index({ user: 1, createdAt: -1 });
orderSchema.index({ status: 1, createdAt: -1 });
```

**Beneficios:**

- Queries rápidas por usuario

- Filtrado eficiente por estado

- Ordenamiento optimizado por fecha

#### Populate Selectivo

```
// Solo traer campos necesarios
.populate('user', 'username email')
.populate('items.product', 'name category')
```

Reduce tamaño de respuestas y carga de red.

## 7. Estructura del Proyecto

```
Practica2/
├── src/
│   ├── graphql/
│   │   ├── schema.js           # TypeDefs de GraphQL
│   │   └── resolvers.js        # Lógica de Queries y Mutations
│   │
│   ├── models/
│   │   ├── User.js             # Usuario con roles
│   │   ├── Product.js          # Producto del catálogo
│   │   ├── Order.js            # Pedido (NUEVO)
│   │   └── Message.js          # Mensaje de chat
│   │
│   ├── routes/
│   │   ├── authRoutes.js       # Login y registro (REST)
│   │   ├── productRoutes.js    # CRUD productos (REST)
│   │   └── adminRoutes.js      # Gestión admin (REST + NUEVO)
│   │
│   ├── middleware/
│   │   └── authenticateJWT.js  # Middleware de autenticación
│   │
│   └── public/
│       ├── index.html          # Página principal
│       ├── login.html          # Login
│       ├── register.html       # Registro
│       ├── products.html       # Catálogo de productos
│       ├── cart.html           # Carrito de compras (NUEVO)
│       ├── my-orders.html      # Mis pedidos (NUEVO)
│       ├── admin.html          # Panel administrador (MEJORADO)
│       ├── chat.html           # Chat en tiempo real
│       ├── cart.js             # Lógica del carrito (NUEVO)
│       ├── graphql-client.js   # Cliente GraphQL (NUEVO)
│       ├── client.js           # Lógica general
│       └── styles.css          # Estilos
│
├── server.js                   # Configuración del servidor
├── config.js                   # Variables de configuración
├── package.json                # Dependencias
├── .env                        # Variables de entorno
├── .gitignore
└── README.md                   # Documentación
```

### Archivos Clave Nuevos en Práctica 2

1. **src/graphql/schema.js** - Definición de tipos GraphQL
2. **src/graphql/resolvers.js** - Lógica de queries y mutations
3. **src/models/Order.js** - Modelo de pedido
4. **src/routes/adminRoutes.js** - Rutas de administración
5. **src/public/cart.html** - Interfaz del carrito

6. **src/public/cart.js** - Clase ShoppingCart
7. **src/public/graphql-client.js** - Cliente GraphQL
8. **src/public/my-orders.html** - Vista de pedidos del usuario

# 8. Repositorio Git

## 8.1 URL del Repositorio

**https://github.com/DavidGarciaCosta/Practica2-ProgramacionWeb**
8.2 Estructura de Commits

El repositorio contiene el historial completo de desarrollo, incluyendo:

- Configuración inicial del proyecto
- Implementación de GraphQL (schema y resolvers)
- Sistema de carrito de compras
- Integración de pedidos
- Panel de administración
- Pruebas y refinamientos

## 8.3 Instrucciones para Clonar

```
git clone [URL_DEL_REPOSITORIO]
cd Practica2
npm install
```

Seguir las instrucciones de instalación en la Sección 2.

# 9. Conclusiones

## 9.1 Objetivos Alcanzados

En este proyecto he logrado implementar exitosamente todas las funcionalidades requeridas para la Práctica 2:

**GraphQL Integrado:** Servidor Apollo Server Express completamente funcional con 7 queries y 8 mutations, conviviendo perfectamente con la API REST existente.

**Sistema de Carrito:** Implementación completa con persistencia en LocalStorage, validaciones de stock y experiencia de usuario fluida.

**Gestión de Pedidos:** Flujo completo desde añadir productos al carrito hasta la creación del pedido vía GraphQL, con validaciones robustas en el servidor.

**Panel de Administrador:** Herramientas completas para gestión de usuarios, pedidos y visualización de estadísticas.

**Arquitectura Sólida:** Código bien estructurado, modular y mantenible, siguiendo mejores prácticas de desarrollo web.

## 9.2 Aprendizajes Clave

### A nivel técnico

1. **GraphQL en producción:** Comprensión profunda de schemas, resolvers, context y autenticación en GraphQL.
2. **Gestión de estado en frontend:** Implementación de un sistema de carrito robusto con sincronización entre cliente y servidor.
3. **Validaciones de negocio:** Implementación de lógica compleja para verificar stock, precios y totales de forma segura.
4. **Arquitectura híbrida:** Experiencia práctica en combinar REST y GraphQL en una misma aplicación.

### A nivel conceptual

1. **Diseño de APIs:** Comprensión de cuándo usar REST vs GraphQL y cómo diseñar queries y mutations efectivas.
2. **Seguridad:** Implementación de autenticación JWT, autorización por roles y validaciones server-side.
3. **Experiencia de usuario:** Diseño de flujos intuitivos con feedback visual y manejo de errores apropiado.

## 9.3 Dificultades Encontradas y Soluciones

### 1. Persistencia del Carrito

**Dificultad:** Mantener el carrito sincronizado entre navegador y servidor.

**Solución:** Usar LocalStorage para persistencia local y validar todo en servidor al crear el pedido, garantizando que stock y precios sean correctos.

### 2. Validación de Precios

**Dificultad:** Evitar que usuarios manipulen precios en el frontend.

**Solución:** Recalcular total en servidor, comparar con productos actuales en base de datos y rechazar pedidos con discrepancias.

### 3. Context de GraphQL

**Dificultad:** Ha sido implementar autenticación en GraphQL manteniendo compatibilidad con REST.

**Solución:** Uso context de Apollo Server para parsear JWT del header Authorization, compartiendo la misma lógica de autenticación.

## 9.4 Posibles Mejoras Futuras

Pienso que puede ser alguna de estas:

1. **Paginación avanzada:** Implementar cursor-based pagination en GraphQL para mejor performance con grandes datasets.
2. **Caché:** Agregar Apollo Client con caché en frontend para reducir requests redundantes.
3. **Notificaciones en tiempo real:** Usar Socket.IO para notificar al admin cuando se crean nuevos pedidos.
4. **Tests automatizados:** Implementar tests unitarios y de integración con Jest y Supertest.
5. **Procesamiento de pagos:** Integrar con Stripe o similar para pagos reales.
6. **Búsqueda avanzada:** Implementar búsqueda full-text con índices de MongoDB.

## 9.5 Reflexión Final

Considero que es una gran práctica para aproximarnos a una página web real y me ha enseñado mucho. Me ha gustado sobre todo aprender a manejar ApolloServer GraphQL.

---

**Fecha de entrega:** 09 de enero de 2026
**Universidad Europea del Atlántico**
**Asignatura:** Programación Web