Sustentación: API Petshop — Desarrollo de API para sistema modular
Adriana Carolina Rodríguez Caicedo Y Yuleibis Noemí Armenta Cruz

1. **Problemática que resuelve nuestro programa**

En el negocio de petshop, la información importante (clientes, mascotas, productos, proveedores, ventas, inventarios y tiendas) suele estar dispersa: registros en hojas de cálculo, notas, o dependientes en la memoria del personal. Esto produce varios inconvenientes:
• Dificultad para consultar datos rápidos (por ejemplo: “¿Qué productos están en bajo stock?” o “¿Cuál es el historial médico de la mascota X?”).
• Riesgo de duplicación y errores al ingresar información.
• Imposibilidad de controlar accesos según rol (empleado, administrador, cliente).
• Dificultad para integrar la información con una app móvil o un punto de venta.

2. **Solución que aporta nuestra aplicación**

Creamos una API REST modular (NestJS) que centraliza y valida la información del petshop. La API:
• Organiza el negocio en módulos por entidad para facilitar mantenimiento y pruebas.
• Valida entradas usando DTOs (class-validator) para asegurar datos consistentes.
• Usa Argon2 para hashing de contraseñas y JWT para autenticación stateless.
• Implementa control por roles con un decorador @Roles() y RolesGuard para proteger rutas según permisos.
Con esto se obtiene un punto único de verdad para la tienda, reducimos errores humanos y facilitamos futuras integraciones (apps, contabilidad, e-commerce).

3. **Entidades**

4. Administrador — Usuarios con permisos de gestión: nombre, email, contraseña(hasheada), role. Manejan configuración y reportes.

5. Cliente — Datos del comprador: nombre, teléfono, dirección, correo. Relacionado con mascotas y ventas.

6. Empleado — Personal del local: nombre, cargo, turno, contacto. Asociado a ventas/servicios realizados.

7. Mascota — Registros de mascotas: nombre, especie, raza, edad, observaciones, propietario (cliente). Usado para historial y reservas.

8. Producto — Items en venta: nombre, descripción, SKU, precio, stock, proveedor. Base del inventario y ventas.

9. Proveedor — Información de quien suministra productos: nombre, contacto, condiciones de pago.

10. Inventario — Movimiento de stock por producto y tienda: entradas, salidas, cantidad actual. Previene faltantes.

11. Venta — Transacción comercial: cliente, items, total, fecha, medio de pago, usuario que registró la venta.

12. Detalle-Factura — Línea de venta con producto, cantidad, precio unitario y subtotal. Relaciona Venta ↔ Producto.

13. Tienda — Local físico: nombre, dirección, horario y responsable. Si existen varias sucursales, permite controlar el stock por local.

14. Rutas y verbos (patrón por entidad)
    Para mantener uniformidad, cada módulo ofrece endpoints REST básicos:
    • GET /{entity} → obtener todos los registros.
    • GET /{entity}/:id → obtener por id.
    • POST /{entity} → crear registro.
    • PATCH /{entity}/:id → actualizar parcialmente.
    • DELETE /{entity}/:id → eliminar por id.

Con 10 entidades y este patrón obtenemos:
• Rutas por entidades: 10 × 5 = 50.
• Endpoints de autenticación: POST /auth/register, POST /auth/login, POST /auth/me → 3.

• Total de rutas: 53.
• Verbos usados: GET, POST, PATCH, DELETE (4 verbos).

5. **Seguridad y control de acceso**

• Hashing de contraseñas: Argon2 para almacenar contraseñas.
• Autenticación: JWT con Passport y passport-jwt (token en header Authorization: Bearer <token>).
• Autorización por roles: Decorador @Roles('ADMIN','EMPLEADO','CLIENTE') y RolesGuard lee metadatos y compara user.role.
• Guards de protección: JwtAuthGuard para endpoints que requieren sesión; RolesGuard para permisos finos.
Esto provee un sistema stateless, seguro y escalable.

6. **Estructura de carpetas**

```
src/
├─ auth/
│  ├─ decorators/        # Roles decorator
│  ├─ dtos/              # login.dto.ts register.dto.ts
│  ├─ guards/            # jwt-auth.guard.ts roles.guard.ts
│  ├─ strategies/        # jwt.strategy.ts
│  ├─ auth.controller.ts
│  ├─ auth.service.ts
│  └─ auth.module.ts
├─ administrador/
│  ├─ dto/
│  ├─ entities/
│  ├─ administrador.controller.ts
│  └─ administrador.service.ts
├─ cliente/
├─ empleado/
├─ mascota/
├─ producto/
├─ proveedor/
├─ inventario/
├─ venta/
├─ detalle-factura/
```

```
├─ tienda/
└─ common/              # utilidades, pipes, interceptors, base entities
```

7. **Conclusión**

La API propuesta organiza la operación del petshop y aporta control, trazabilidad y seguridad. La modularidad facilita escalar el sistema (añadir reservas de servicio, historial médico más detallado, integraciones con contabilidad). Este documento replica la lógica y el formato del PDF de ejemplo, pero aplicado a la realidad de un petshop para que la presentación sea coherente y práctica.