PLAN DE PRUEBAS

Proyecto: Sistema de Gestión de Inventario

Fecha: Diciembre 2025

Autor: John Sebastian Castañeda Hoyos

1. OBJETIVOS DE PRUEBA

Los objetivos principales de este plan de pruebas son:

• Verificar el correcto funcionamiento de la lógica de negocio en los servicios de categorías y productos.

• Validar la integración entre la API REST y la base de datos PostgreSQL.

• Asegurar que las operaciones CRUD funcionen correctamente para categorías y productos.

• Detectar errores de validación y manejo de excepciones.

• Garantizar la calidad del software antes del despliegue.

2. ALCANCE

2.1 Módulos a Probar

• Servicio de Categorías (CategoryService): Creación, actualización, eliminación y consulta de categorías.

• Servicio de Productos (ProductService): Creación, actualización, eliminación, consulta y validación de productos.

• API REST: Endpoints para categorías (/api/categories) y productos (/api/products).

• Base de Datos: Persistencia y relaciones entre tablas categories y products.

2.2 Tipos de Pruebas

• Pruebas Unitarias: Validación de la lógica interna de los servicios de manera aislada.

• Pruebas de Integración: Validación del comportamiento de la API en conjunto con la base de datos.

3. RECURSOS

3.1 Herramientas

• Jest: Framework de pruebas para JavaScript.

• Supertest: Librería para pruebas de APIs HTTP.

• PostgreSQL: Base de datos relacional.

• GitHub Actions: Plataforma de integración continua.

• Node.js v18: Entorno de ejecución.

3.2 Personal Involucrado

• Desarrollador/Tester: Responsable de la creación y ejecución de las pruebas.

4. CASOS DE PRUEBA

4.1 Pruebas Unitarias

Las pruebas unitarias validan la lógica interna de los servicios de manera aislada, utilizando mocks para simular las dependencias.

ID	Descripción	Precondiciones	Pasos	Resultado Esperado	Resultado Obtenido
UT-001	Crear categoría exitosamente	Servicio CategoryService disponible. Mock de Category configurado.	1. Llamar createCategory con nombre válido. 2. Verificar que se llame a Category.create.	Retorna objeto categoría con id y nombre.	✓ PASÓ
UT-002	Error al crear categoría con nombre vacío	Servicio CategoryService disponible.	1. Llamar createCategory con nombre vacío. 2. Capturar excepción.	Lanza error 'Category name is required' con status 400.	✓ PASÓ
UT-003	Error al crear categoría duplicada	Mock retorna categoría existente con mismo nombre.	1. Llamar createCategory con nombre existente. 2. Capturar excepción.	Lanza error 'Category with this name already exists' con status 409.	✓ PASÓ
UT-004	Error al eliminar categoría con productos	Mock indica que categoría tiene productos asociados.	1. Llamar deleteCategory. 2. Verificar validación de productos.	Lanza error 'Cannot delete category with associated products' con status 400.	✓ PASÓ
UT-005	Crear producto exitosamente	Mock de Product y Category configurados.	1. Llamar createProduct con datos válidos. 2. Verificar llamada a Product.create.	Retorna objeto producto con todos los campos.	✓ PASÓ
UT-006	Error al crear producto con nombre vacío	Servicio ProductService disponible.	1. Llamar createProduct sin nombre. 2. Capturar excepción.	Lanza error 'Product name is required' con status 400.	✓ PASÓ
UT-007	Error al crear producto con precio negativo	Servicio ProductService disponible.	1. Llamar createProduct con precio -10. 2. Capturar excepción.	Lanza error 'Product price must be a positive number' con status 400.	✓ PASÓ
UT-008	Error al crear producto con stock negativo	Servicio ProductService disponible.	1. Llamar createProduct con stock -5. 2. Capturar excepción.	Lanza error 'Product stock must be a non-negative	✓ PASÓ

ID	Descripción	Precondiciones	Pasos	Resultado Esperado	Resultado Obtenido
				integer' con status 400.	
UT-009	Error por stock insuficiente	Mock de producto con stock 5.	1. Llamar updateProductStock con cantidad -10. 2. Capturar excepción.	Lanza error 'Insufficient stock' con status 400.	✓ PASÓ
UT-010	Eliminar categoría sin productos	Mock indica que categoría no tiene productos.	1. Llamar deleteCategory. 2. Verificar llamada a Category.delete.	Retorna categoría eliminada correctamente.	✓ PASÓ

4.2 Pruebas de Integración

Las pruebas de integración validan el comportamiento de la API REST en conjunto con la base de datos PostgreSQL.

ID	Descripción	Precondiciones	Pasos	Resultado Esperado	Resultado Obtenido
IT-001	POST /api/categories - Crear categoría y persistir en BD	Servidor corriendo. BD conectada. Tablas creadas y vacías.	1. Enviar POST con {name: 'Electronics'}. 2. Consultar BD directamente.	Status 201. Categoría existe en BD con nombre 'Electronics'.	✓ PASÓ
IT-002	POST /api/products - Crear producto con relación a categoría	Categoría 'Computers' creada previamente en BD.	1. Crear categoría. 2. Crear producto con category_id. 3. Verificar JOIN en BD.	Status 201. Producto en BD con category_id correcto. JOIN retorna category_name.	✓ PASÓ
IT-003	GET /api/products - Listar productos con nombres de categoría	Categoría y producto creados en BD.	1. Crear categoría 'Phones'. 2. Crear producto 'iPhone 15'. 3. GET /api/products.	Status 200. Array contiene producto con category_name 'Phones'.	✓ PASÓ
IT-004	PUT /api/products/:id - Actualizar producto y reflejar en BD	Producto existente en BD.	1. Crear producto. 2. PUT con nuevos datos. 3. Consultar BD.	Status 200. BD contiene valores actualizados (name, price).	✓ PASÓ
IT-005	DELETE /api/products/:id - Eliminar producto de BD	Producto existente en BD.	1. Crear producto. 2. Verificar existencia. 3. DELETE. 4. Verificar eliminación.	Status 200. Consulta a BD retorna 0 filas para ese ID.	✓ PASÓ

5. RESUMEN DE RESULTADOS

Tipo de Prueba	Total	Pasaron	Fallaron
Pruebas Unitarias	10	10	0
Pruebas de Integración	5	5	0
TOTAL	**15**	**15**	**0**

6. CONCLUSIONES

• Todas las pruebas unitarias y de integración fueron ejecutadas correctamente.

• La lógica de negocio en los servicios funciona correctamente según lo que se pedía.

• La integración entre la API REST y PostgreSQL trabaja sin errores.

• Las validaciones de datos (nombre requerido, precio positivo, stock no negativo) funcionan correctamente.

• El pipeline de CI/CD en GitHub Actions ejecuta todas las pruebas automáticamente.

• El sistema de inventario está listo para desplegar en producción