Requerimiento: Sistema de Búsqueda Dinámica de Productos

JIRA Task: ECOM-2024 – Implementar Búsqueda Avanzada de Productos

Descripción

Como **usuario del e-commerce**, necesito poder buscar productos aplicando múltiples filtros opcionales de manera dinámica, para encontrar exactamente los productos que cumplan con mis criterios específicos.

Criterios de Aceptación (BDD)

```
Feature: Búsqueda dinámica de productos con filtros opcionales

  Background:
    Given el catálogo contiene productos de múltiples marcas
    And cada producto tiene precio, rating, estado de descuento y categoría
    And las categorías disponibles son HOME y OFFICE

  Scenario: Búsqueda por marca única
    Given soy un usuario en la página de búsqueda
    When busco productos con marca "Samsung"
    Then debería ver todos los productos Samsung
    And los resultados deben estar ordenados por rating descendente y precio ascendente

  Scenario: Búsqueda con múltiples criterios
    Given soy un usuario en la página de búsqueda
    When busco productos con los siguientes criterios:
      | Campo          | Valor    |
      | marca          | Amazon   |
      | precioMínimo   | 100      |
      | precioMáximo   | 1000     |
      | ratingMínimo   | 4        |
      | tieneDescuento | true     |
    Then debería ver solo productos Amazon entre $100-$1000 con rating >= 4 en descuento
    And los resultados deben estar paginados

  Scenario: Búsqueda por categoría
    Given soy un usuario en la página de búsqueda
    When busco productos de la categoría "OFFICE"
    Then debería ver solo productos asociados a la categoría OFFICE
    And debo poder combinar este filtro con otros criterios

  Scenario: Búsqueda sin criterios
    Given soy un usuario en la página de búsqueda
    When realizo una búsqueda sin aplicar ningún filtro
    Then debería ver todos los productos del catálogo
    And los resultados deben estar ordenados y paginados
```

Definición de "Lista" (DoD)

- [ ] API REST implementada con endpoint `/filter/products`
- [ ] Soporte para todos los parámetros opcionales
- [ ] Paginación implementada
- [ ] Ordenamiento por rating (DESC) y precio (ASC)
- [ ] Combinaciones complejas

Problema a Resolver

Contexto

El sistema actual requiere crear un método específico en el repositorio para cada combinación de búsqueda. Con 7 criterios de búsqueda opcionales, esto resultaría en **2^7 = 128** métodos diferentes.

Solución: Spring Data JPA Specifications

Implementar un `ProductSpecificationBuilder` que construya dinámicamente las consultas basándose en los criterios proporcionados.

Ejemplos de Consultas SQL Generadas

1. Query con criterio simple (solo marca)

```
-- Parámetros: brand = 'Samsung'
SELECT DISTINCT pc.*
FROM products_catalog pc
WHERE pc.brand_name = 'Samsung'
ORDER BY pc.rating DESC, pc.price ASC
LIMIT 20 OFFSET 0;

-- Resultado esperado: 3 productos
-- • Monitor 24 (99.89, rating: 10, descuento: true)
-- • Monitor 27 (200.89, rating: 9, descuento: false)
-- • TV QLED 55" (1200.99, rating: 9, descuento: false)
```

2. Query con marca y estado de descuento

```
-- Parámetros: brand = 'Amazon', hasDiscount = true
SELECT DISTINCT pc.*
FROM products_catalog pc
WHERE pc.brand_name = 'Amazon'
    AND pc.is_discount = true
ORDER BY pc.rating DESC, pc.price ASC
LIMIT 20 OFFSET 0;

-- Resultado esperado: 2 productos
-- • Alexa super (200.89, rating: 4)
-- • Alexa small (500.89, rating: 1)
```

3. Query con rango de precio y rating mínimo

```
-- Parámetros: minPrice = 20, maxPrice = 500, minRating = 8
SELECT DISTINCT pc.*
FROM products_catalog pc
WHERE pc.price >= 20
    AND pc.price <= 500
    AND pc.rating >= 8
ORDER BY pc.rating DESC, pc.price ASC
LIMIT 20 OFFSET 0;

-- Resultado esperado: Múltiples productos incluyendo
-- • Samsung Monitor 24 (99.89, rating: 10)
-- • Microsoft office home (100.00, rating: 10)
-- • Puma backpack P (40.20, rating: 10)
-- • Nike backpack N (30.57, rating: 9)
-- • Samsung Monitor 27 (200.89, rating: 9)
```

4. Query con filtro de categoría

```
-- Parámetros: categoryCode = 'HOME', minPrice = 100, maxPrice = 1000
SELECT DISTINCT pc.*
FROM products_catalog pc
WHERE pc.price >= 100
    AND pc.price <= 1000
    AND pc.id IN (
        SELECT pjc.id_product
        FROM product_join_category pjc
        JOIN categories c ON pjc.id_category = c.id
        WHERE c.code = 'HOME'
    )
ORDER BY pc.rating DESC, pc.price ASC
LIMIT 20 OFFSET 0;

-- Resultado esperado: Productos de categoría HOME en el rango de precio
-- • Samsung Monitor 24 (99.89)
-- • Samsung Monitor 27 (200.89)
-- • LG TV 65 (900.89)
-- • Amazon Alexa small (500.89)
```

5. Query completa con todos los parámetros

```
-- Parámetros completos:
-- brand = 'Samsung'
-- minPrice = 50
-- maxPrice = 1500
-- minRating = 9
-- hasDiscount = false
-- categoryCode = 'HOME'
-- launchedAfter = '2022-01-01'

SELECT DISTINCT pc.*
FROM products_catalog pc
WHERE pc.brand_name = 'Samsung'
    AND pc.price >= 50
    AND pc.price <= 1500
    AND pc.rating >= 9
    AND pc.is_discount = false
    AND pc.id IN (
        SELECT pjc.id_product
        FROM product_join_category pjc
        JOIN categories c ON pjc.id_category = c.id
        WHERE c.code = 'HOME'
    )
    AND pc.launching_date > '2022-01-01'
ORDER BY pc.rating DESC, pc.price ASC
LIMIT 20 OFFSET 0;

-- Resultado esperado: Productos Samsung premium sin descuento
-- • Samsung Monitor 27 (200.89, rating: 9)
-- • Samsung TV QLED 55" (1200.99, rating: 9)
```

Ventajas de la Solución con Specifications

1. Flexibilidad Total

- Cualquier combinación de criterios funciona sin código adicional
- Fácil agregar nuevos criterios de búsqueda

2. Mantenibilidad

- Un solo método maneja todas las combinaciones
- Lógica centralizada en el builder

3. Type Safety

- Validación en tiempo de compilación
- Evita errores de SQL injection

4. Integración Perfecta

- Funciona nativamente con Spring Data JPA
- Soporta paginación y ordenamiento automáticamente

Spring controller

```
@RestController
@RequestMapping("/filter/products")
@RequiredArgsConstructor
public class ProductController {

    private final ProductCatalogRepository repository;
    private final ProductSpecificationBuilder specBuilder;

    @GetMapping
    public Page<ProductCatalogEntity> search(@ModelAttribute
ProductSearchCriteria criteria, Pageable pageable) {
        Specification<ProductCatalogEntity> spec =
specBuilder.build(criteria);
        return repository.findAll(spec, pageable);
    }
```

Criterio de busqueda

```
@Data
public class ProductSearchCriteria {

    private String brand;
    private BigDecimal minPrice;
    private BigDecimal maxPrice;
    private Integer minRating;
    private Boolean hasDiscount;
    private String categoryCode;
    private LocalDate launchedAfter;
}
```

URLs de Ejemplo

```
# Búsqueda simple por marca
GET /filter/products/search?brand=Samsung

# Búsqueda con múltiples criterios
GET /filter/products/search?
brand=Amazon&minPrice=100&maxPrice=1000&hasDiscount=true&minRating=4

# Búsqueda por categoría con paginación
GET /filter/products/search?categoryCode=OFFICE&page=0&size=10

# Búsqueda completa
GET /filter/products?
brand=Samsung&minPrice=50&maxPrice=1500&minRating=9&hasDiscount=false&cate
goryCode=HOME&launchedAfter=2022-01-01
```

Conclusión

El patrón Specification resuelve elegantemente el problema de búsquedas dinámicas complejas, evitando la explosión combinatoria de métodos en el repositorio y proporcionando una solución mantenible y extensible.