Estructura del código

Atendiendo a las convenciones, el nombre del package principal es com.acme.jfc [*Java's package naming conventions with a reverse domain name*[1]].

Estructura de diseño:

He optado por una estructura por capa [*Structure by Layer*[1]], en lugar de una estructura por característica [*Structure by Feature*[1]], donde todos los controladores estarán ubicados en el package controller, los servicios en package service, etc.

```
com
 +- acme
      +- jfc
            +- Application.java
            |
            +- entity / domain
            |     - Categoria.java
            |     - Prenda.java
            |     - Promocion.java
            +- repository / dao
            |     - IPrendaRepository.java
            |     - IPromocionRepository.java
            +- service
            |     - IPrendaService.java
            |     - IPromocionService.java
            |------+- impl
                   |   - PrendaServiceImpl.java
                   |   - PromocionServiceImpl.java
            +- controller
            |     - PrendaController.java
            |     - PromocionController.java
            +- dto / payload / model
            |     - PrendaDto.java
            |     - PrendaPromocionadaDto.java
            |     - PrendaRebajasDto.java
            |     - PromocionDto.java
            |     - PromocionRebajasDto.java
            +- exception
            |     - ErrorResponse.java
            |     - GlobalExceptionHandler.java
            |     - ResourceNotFoundException.java
            +- configuration
            |     - DataLoader.java
            |     - ModelMapperConfig.java
            |_    - SwaggerConfig.java
```

[1] https://www.geeksforgeeks.org/spring-boot-code-structure/ - (Spring Boot – Code Structure)

Entidades: package → entity

Para ambas entidades, he usado el nombre **id** como atributo `@Id` de clase en vez del sugerido por el endpoint. Serán los DTO quienes mostrarán en su lugar prendaId / promocionId, y evitamos de este modo exponer la estructura interna de la BD.

`Prenda.java`

"Identificadas por una referencia alfanumérica de 10 caracteres. El primer carácter es la talla de la prenda (S/M/L)"

- Con `@Pattern` (javax.validation) comprobaremos que se cumpla con el formato y el largo.

"Están asociadas a cero o más categorías (escritas exactamente así)"

- He movido las categorías a una clase externa de tipo enumerado.
- Las relaciones con `Categoria.java` se guardarán en una tabla diferente que vincula a ambas.

La relación de 0 a muchos entre Prendas y Promociones también se registrará en una tabla distinta.
El cálculo de precio promocional resultante de aplicar las distintas promociones activas será un método asociado a un DTO, y mantendremos la clase prenda lo más simple posible (POJO).

`Promocion.java`

"... aplicarán a las prendas a las que estén asociadas. Son acumulativas (una prenda puede tener varias promociones)"

- Misma relación (ManyToMany) definida en Prendas, con tabla que vincula a ambas.

Repositorios: package → repository

`IPrendaRepository.java / IPromocionRepository.java`

- Interfaces que heredan de JpaRepository.
- `IPromocionRepository`: una `@Query` para controlar si existe un id de Prenda.
- `@Query` nativa en ambas interfaces que facilita borrar las relaciones ManyToMany de la tabla vinculante.

Servicios: package → service

`IPrendaService.java / IPromocionService.java`

Ambas interfaces definen los métodos CRUD básicos para sus entidades.
Incluyen un *`boolean findById`*, que dará soporte a los CRUD (antes de un create por si ya existe, o por si no existe para el resto de métodos), y facilita el manejar los errores.

Las implementaciones se encuentran en un **subpackage → impl**. Lanzarán un *`ResourceNotFoundException`* ante excepciones en tiempo de ejecución (JSON).
La implementación de `PromociónServiceImpl` es la encargada de gestionar las peticiones relacionadas a los requisitos del Statement (III). E incluye un método *`verify(prendaId)`* que ejecuta la `@Query` del repositorio cuando se interactúa con alguna Prenda. (Y no inyectamos el repositorio de prenda)

Controladores: package → controllers

(Anexo con Tabla que contiene todas las respuestas desde los ENDPOINTS)

`PrendaController.java`

GET	/prendas	List<PrendaDto>
GET	/prendas?full=true	List<PrendaPromocionadaDto>
GET	/prendas/{prendaId}	PrendaPromocionadaDto
POST	/prendas	PrendaDto
PUT	/prendas/{prendaId}	PrendaDto
DELETE	/prendas/{prendaId}	String

Gestiona las peticiones al EndPoint "/prendas"
Todas las respuestas que se generan desde el controlador son en formato Json. Los String son convertidos a objetos Json <dependencia *com.google.code.gson.*>
Además se trabaja únicamente con DTOs, siendo `PrendaPromocionadaDto` el más detallado.

`PromocionController.java`

GET	/promociones	List<PromocionDto>
GET	/promociones?full=true	List<PromocionRebajasDto>
GET	/promociones?search=texto	List<PromocionDto>
GET	/promociones?search=texto&full=true	List<PromocionRebajasDto>
GET	/promociones/{promocionId}	PromocionRebajasDto
POST	/promociones	PromocionDto
PUT	/promociones/{promocionId}	PromocionDto
DELETE	/promociones/{promocionId}	String
PUT	/promociones/{promocionId}/prendas	String STATEMENT(III)
DELETE	/promociones/{promocionId}/prendas	String STATEMENT(III)
PUT	/promociones/**aplicar**?promocion={promocionId} &prenda={prendaId}	PrendaPromocionadaDto
PUT	/promociones/**remover**?promocion={promocionId} &prenda={prendaId}	String

Gestiona las peticiones al EndPoint "/promociones"
Todas las respuestas que se generan desde el controlador son en formato Json. Los String son convertidos a objetos Json <dependencia *com.google.code.gson.*>
Además se trabaja únicamente con DTOs, siendo `PromocionRebajasDto` el más detallado.

En azul son implementaciones extra que utilizan un `@RequestParam` opcional. Los GET para generar la salida en uno u otro DTO, y los PUT de promociones, para aplicar/remover relaciones entre UNA promoción y UNA prenda.

Configuraciones: package → configuration

`ModelMapperConfig.java` Define el `@Bean` del mapeador de Entidades-DTOs.
`SwaggerConfig.java` Configura valores de Swagger. (http://localhost:8080/swagger-ui/)
`DataLoader.java` Carga uno datos iniciales para trabajar con la DB H2 en memoria.
http://localhost:8080/h2-console para acceder al gestor embebido.

Excepciones: package → exception

`GlobalExceptionHandler` manejará las excepciones de forma centralizada (`@RestControllerAdvice`). Los mensajes devueltos serán siempre un JSON con el mismo formato que los que genera el controlador para mantener una coherencia en el formato. Incluirán "status" y "message". Ejemplos:

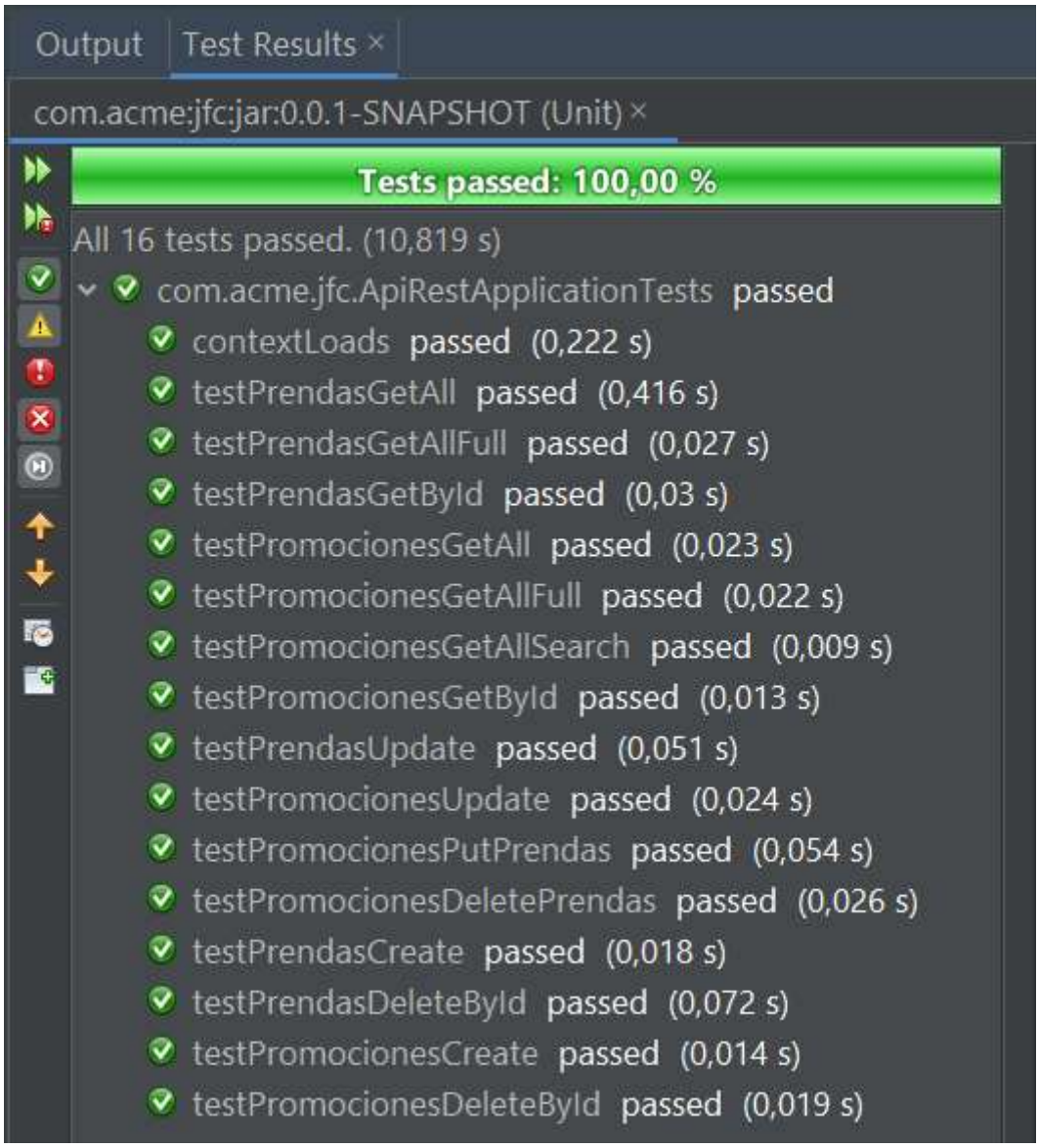
```
{
    "status": 200,
    "message": "Prenda borrada con éxito."
}

{
    "status": 404,
    "message": "Elemento no encontrado: S1234567891"
}

{
    "status": 400,
    "message": "Fallo de validación detectado. Compruebe los valores introducidos:
      [Field=precio][RejectedValue=-10.0] [Error=debe ser mayor que o igual a 0.0]"
}
```

DTOs: package → dto

`PrendaDto.java`
`PrendaPromocionadaDto.java`
`PrendaRebajasDto.java`
`PromocionDto.java`
`PromocionRebajasDto.java`

Tests unitarios

Método	URL	Nombre del Test	Orden
GET	/prendas	testPrendasGetAll	2
GET	/prendas?full=true	testPrendasGetAllFull	3
GET	/prendas/{prendaId}	testPrendasGetById	4
POST	/prendas	testPrendasCreate	13
PUT	/prendas/{prendaId}	testPrendasUpdate	9
DELETE	/prendas/{prendaId}	testPrendasDeleteById	14
GET	/promociones	testPromocionesGetAll	5
GET	/promociones?full=true	testPromocionesGetAllFull	6
GET	/promociones?search=texto	testPromocionesGetAllSearch	7
GET	/promociones/{promocionId}	testPromocionesGetById	8
POST	/promociones	testPromocionesCreate	15
PUT	/promociones/{promocionId}	testPromocionesUpdate	10
DELETE	/promociones/{promocionId}	testPromocionesDeleteById	16
PUT	/promociones/{promocionId}/prendas	testPromocionesPutPrendas	11
DELETE	/promociones/{promocionId}/prendas	testPromocionesDeletePrendas	12



Dependencias incluidas en el proyecto:

- spring-boot-starter-data-jpa
- spring-boot-starter-web
- spring-boot-starter-validation
- springfox-boot-starter
- h2database
- lombok
- modelmapper
- gson

ANEXO

ENDPOINTS: Prendas		Funcionalidad	Status
GET	/prendas	Lista completa resumida, sin precio promocionado y sin promociones (**PrendaDto**)	200
GET (extra)	/prendas?full=true	Lista completa detallada, incluye promociones y precio promocionado (**PrendaPromocionadaDto**)	200
GET	/prendas/{prendaId}	Info detallada de prenda (**PrendaPromocionadaDto**). (ok 200) Si el id no existe, devuelve un mensaje 404.	200 404
PUT	/prendas/{prendaId}	Body acepta un **PrendaDto**, y mapea a Prenda al guardar. Si el Body está mal estructurado (@Valid), devuelve un 400. Si el id no existe, devuelve un Json 404.	200 400 404
DELETE	/prendas/{prendaId}	Si existe, un Json 200. Si el id no existe, devuelve un Json 404.	200 404
POST	/prendas	Body acepta un **PrendaDto**, y mapea a Prenda al guardar. Si el Body está mal estructurado (@Valid), devuelve un 400. Si el id ya existe, devuelve un Json 400.	201 400 400

ENDPOINTS: Promociones		Funcionalidad	Status
GET	/promociones	Lista completa resumida, sin prendas asociadas. (**PromocionDto**)	200
GET (extra)	/promociones?full=true	Lista completa detallada, incluye prendas asociadas (**PromocionRebajasDto**)	200
GET	/promociones?search=texto (Mejora propuesta)	Lista que incluye las promociones que incluyan "texto". Puede ser combinada con full. *promociones?search=text&full=true*	200
GET	/promociones/{promocionId}	Info detallada de promocion (**PromocionRebajasDto**). (ok 200) Si el id no existe, devuelve un mensaje 404.	200 404
PUT	/promociones/{promocionId}	Body acepta un **PromocionDto**, y mapea a Prenda al guardar. Si el Body está mal estructurado (@Valid), devuelve un 400. Si el id no existe, devuelve un Json 404.	200 400 404
DELETE	/promociones/{promocionId}	Si existe, un Json 200 Si el id no existe, devuelve un Json 404.	200 404
POST	/promociones	Body acepta un **PromocionDto**, y mapea a Promocion al guardar. Si el Body está mal estructurado (@Valid), devuelve un 400. Si el id ya existe, devuelve un Json 400.	201 400 400

Statement(III) + extra (aplicar // remover)

PUT (extra)	/promociones/aplicar ?promocion={promocionId} &prenda={prendaId}	Con dos @RequestParam asocia una prenda y una promoción. Si algún id no existe, devuelve un Json 400.	200 400
PUT (extra)	/promociones/remover ?promocion={promocionId} &prenda={prendaId}	Elimina una relación entre una prenda y una promoción. Si algún id no existe, devuelve un Json 404.	200 404

PUT	/promociones/{promocionId} /prendas	Asocia una ARRAY de Strings [prendasId] a una promocion. Json. Si algún id no existe, no asocia ninguno y devuelve un Json 400.	200 400
DELETE	/promociones/{promocionId} /prendas	200 Si existen todos los ids contenidos en el Array de ids. Si algún id no existe, no se realiza ninguna operación. Json 404.	200 404