Paginación de resultados

Elementos avanzados en tu API REST con Spring Boot

Paginación de resultados

- Cuando el número de resultados de una consulta o pantalla es grande, es necesario dividirlo.
- ¿Por qué?
  - Eficiencia: tiempo de procesamiento y carga muy grande
  - Experiencia de usuario.
- ¿Te imaginas que twitter cargara todos los tuits de tu timeline desde que te diste de alta cada vez que entras en la app?

¿Qué nos ofrece Spring?

- Spring Data JPA
  - Los repositorios nos ofrecen la posibilidad de realizar *consultas paginadas*.
  - *JpaRepository<T,ID>* extiende a *PagingAndSortingRepository<T,ID>*, que tiene capacidades de paginación.

¿Qué nos ofrece Spring?

- Spring Data JPA
  - Tenemos disponible el método ***Page<T> findAll(Pageable pageable)***.
  - Todas nuestras consultas pueden ser del tipo ***Page<T> findByAlgo(Algo algo, Pageable pageable)***.

Pageable

- Elemento de entrada en una consulta paginada.
- Interfaz con la información necesaria para extraer una *página* (un subconjunto) de resultados.
- Métodos
  - ***int getPageNumber()*** //número de la página
  - ***int getPageSize()*** // tamaño de página
  - ***Sort getSort()*** // parámetros para ordenar

Page<T>

- Resultado en una consulta paginada.
- Se trata de una sublista de una lista de objetos.
- Tiene además información sobre la posición en la lista completa.
- Extiende a la interfaz *Slice<T>*

Page<T> extends Slices<T>

- Métodos
  - ***List<T> getContent()*** // elementos en la página
  - ***int getTotalPages()*** // nº total de páginas
  - ***int getTotalElements()*** // nº total de elementos
  - ***int getNumber()*** // nº de página actual
  - ***int getSize()*** // tamaño de página actual
  - ***int getNumberOfElements()*** // nº de elementos en la página

Spring Data Web Support

- La paginación debe ser soportada por la capa de persistencia subyacente.
- Las clases *Page* y *Pageable* pertenecen a Spring Data, no a Spring Web.
- Spring Boot configura automáticamente el soporte web para Spring Data (Spring Data Web Support).
- Esto nos permite usar *Pageable* en un controlador.

Spring Data Web Support

```
@RestController
public class ProductoController {
     …
     @GetMapping("/producto")
     public String index(Model model,
     @PageableDefault(page=0, size=5) Pageable pageable) {
          // ...
     }
     …
}
```

*http://www.miaplicacion.com/?**page**=1&**size**=10&**sort**=prop*

Modelo para paginación

- Nuestro resultado ya no será un List<?>.
- Debemos darle facilidades a quién consuma del API para poder navegar a las demás páginas.
- A falta de un modelo mejor usamos la propia interfaz *Page* (en concreto, la clase *PageImpl*).

Paginación según RFC 5988

- https://tools.ietf.org/html/rfc5988#section-5
- Nos indica que tenemos que, en caso de paginación, debemos añadir un encabezado llamado *link*.
- Debe incluir los enlaces a las páginas primera, anterior, siguiente y última (siempre que aplique).
- Tiene esta estructura:

```
<http://localhost:8080/producto/?page=2&size=10>; rel="next",
<http://localhost:8080/producto/?page=0&size=10>; rel="prev",
<http://localhost:8080/producto/?page=0&size=10>; rel="first",
<http://localhost:8080/producto/?page=2&size=10>; rel="last"
```

Reto

- El modelo que hemos usado para la paginación es algo “pesado”. Puedes crear uno propio, que incluya la información estrictamente necesaria y hacer las transformaciones con *ModelMapper*.
- El error que devolvemos indica el mensaje **No hay productos registrados**. Con la nueva implementación, también se devuelve si accedemos a una página que no existe (por ejemplo, la página 234). ¿Eres capaz de diferenciar ambos errores y devolver un mensaje más conveniente para una página errónea?