Paginación de resultados

Cómo hacer las consultas

Curso intermedio de Thymeleaf

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

```
@Controller
public class ProductoController {
     …
     @GetMapping("/")
     public String index(Model model, Pageable pageable) {
        model.addAttribute("productos",
              productoService.findAllPaginated(pageable));
        return "admin/list-producto";
     }
     …
}
```

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

Algunas *properties*

- Podemos configurar algunos parámetros de *Spring Data Web Support* a través de propiedades (con prefijo *spring.data.web.xxxx*).
- Algunas de ellas
  - `s.d.w.pageable.default-page-size=20`
    Tamaño de página por defecto
  - `s.d.w.pageable.max-page-size=2000`
    Tamaño de página máximo
  - `s.d.w.pageable.one-indexed-parameters=false`
    Si es falso, los números de página empiezan en 0; si es verdadero en 1.

Parámetros por defecto a nivel de método

- La anotación *@PageableDefault* nos permite establecer unos valores por defecto a nivel de método.

```
@GetMapping("/")
public String index(Model model,
     @PageableDefault(page=1, size=5) Pageable pageable) {
   model.addAttribute("productos",
         productoService.findAllPaginated(pageable));
   return "admin/list-producto";
}
```

En nuestro ejemplo

- Cambios en la persistencia (en nuestro caso, el servicio)
- Añadimos properties
- Refactorización del controlador

- ***¿Qué nos falta?*** La navegación, que la implementamos en la próxima lección.

Reto

- Implementar la paginación de las categorías
- Implementar la paginación de productos en el listado de la página principal, tanto por categorías como de productos aleatorios (*esta última requerirá algo más de trabajo*).