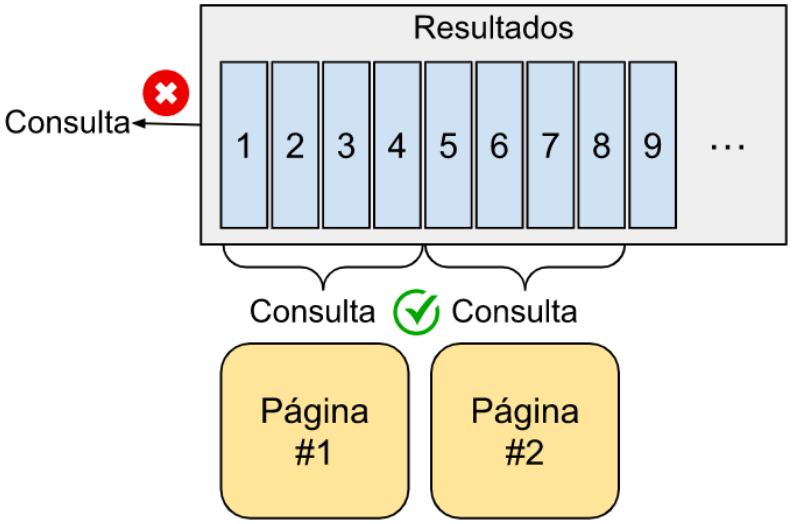
Ejercicio 16 – Paginación y ordenación de resultados

En la práctica anterior destacamos la importancia de recuperar de la base de datos solo la información imprescindible mediante el uso de proyecciones empleando interfaces, DTOs y *records* frente a entidades.

En esta práctica, vamos trabajar sobre el mismo concepto, pero desde la cantidad de registros a recuperar. Cuanta más información recupere una consulta, mayor lentitud de la operación y consumo de memoria, pudiendo, en casos extremos, venirse abajo el sistema por un *OutMemoryException*.

Paginación

La paginación es una técnica comúnmente utilizada en aplicaciones web y sistemas de manejo de datos para gestionar grandes conjuntos de datos de manera eficiente. Para ello, divide estos conjuntos de datos en lotes más pequeños, llamados páginas, consultando y devolviendo al usuario o cliente solo uno de ellos.



Como usuarios, estamos acostumbrados a ver paginación en cualquier programa o sitio web que muestre un listado de datos, como un buscador.

La paginación utiliza parámetros como el número de página y el tamaño de la página para indicar qué página de resultados y cuántos resultados por página se deben devolver. Para que el reparto de los datos en páginas sea coherente, en la consulta que va obteniendo las páginas siempre debe aplicarse el mismo tamaño de página y, algo menos evidente, criterio de ordenación.

Por último, destacar un inconveniente en esta técnica. Si se añaden nuevos registros mientras navegamos por las páginas y estos deberían aparecer en las páginas precedentes a las que estamos solicitando, los resultados pueden ser inconsistentes. En general, podemos ignorar este pequeño defecto.

Paginación en Spring Data

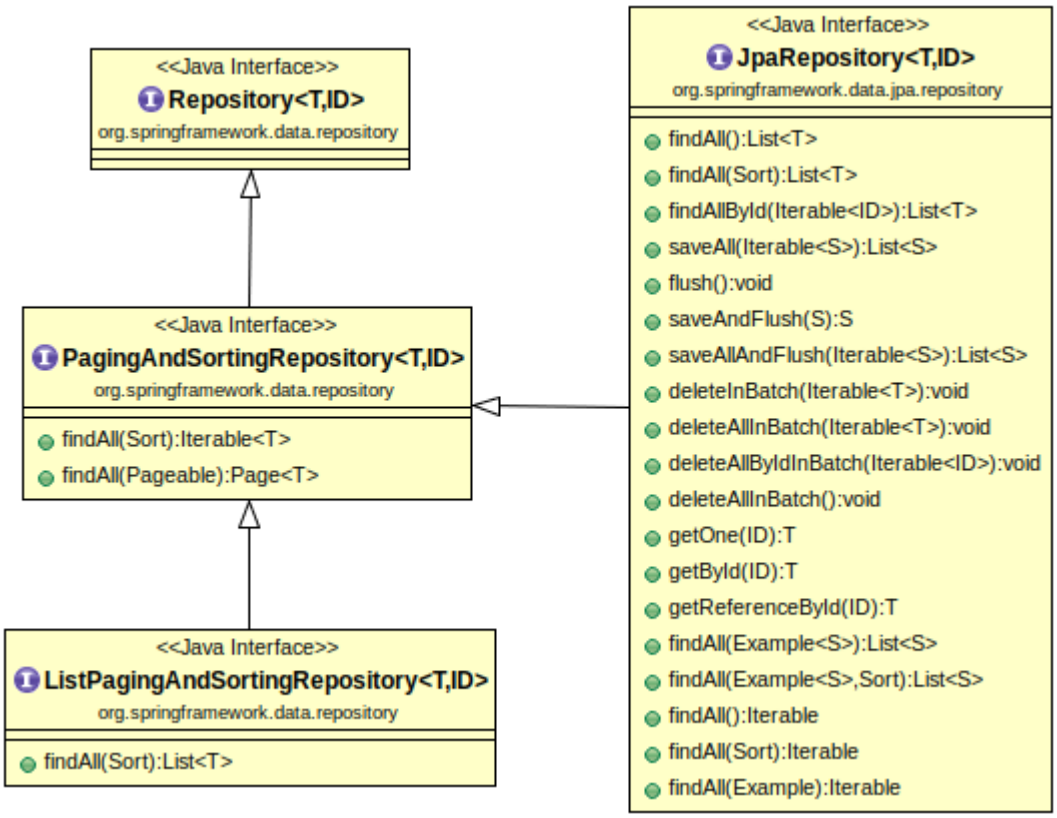
Para ilustrar el concepto de paginación y ver cómo implementarlo en Spring, vamos a tomar como referencia el siguiente código de ejemplo:

```
public class Customer {
    @Id Long id;
    String firstname, lastname;
    String password;
    LocalDate regDate;
    Address address;
}
```

Ahora, creamos un repositorio que extiende ***JpaRepository*** y usa el tipo ***Page*** como tipo de retorno:

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {
    Page<Customer> findAllProjectedBy(Pageable pageable);
}
```

Utilizamos el repositorio ***JpaRepository*** porque extiende ***PaginaAndSortingRepository***, que permite trabajar con el concepto de paginación y ordenación.



La interfaz Pageable contiene información sobre el número de página, el tamaño de página o el criterio de ordenación aplicado.

Por otro lado, Page es una interfaz y sublista de una lista de objetos.

Uso de parámetros Page y Sort

Generalmente, los parámetros de paginación y ordenación son opcionales y, por lo tanto, forman parte de la URL de solicitud como parámetros de consulta. Debemos proporcionar unos valores predeterminados para los casos donde el cliente no especifique ninguna preferencia.

Estos valores predeterminados de paginación y clasificación deben estar documentados, de forma que sea conocedor de cómo funciona su uso.

En el siguiente controlador aceptamos parámetros de paginación y ordenación, utilizando los siguientes parámetros de consulta:

- page: El número de página actual. La primera página es la número 0.
- size: El tamaño máximo de la página, es decir, los resultados que puede contener.
- sortBy: El campo por el que aplicar la ordenación
- sortDirection: El orden en que se muestran los registros.

Un ejemplo de controlador podría ser el siguiente:

```
@GetMapping("/customers")
public ResponseEntity<List<Customer>> getAllCustomers(
                    @RequestParam(defaultValue = "0") Integer page,
                    @RequestParam(defaultValue = "10") Integer size,
                    @RequestParam(defaultValue = "id") String sortBy,
                    @RequestParam(defaultValue = "ASC") String sortDirection)
{
     Page<Customer> list = service.getAll(page, size, sortBy, sortDirectiony);
     return ResponseEntity.ok(list.getContent());
}
```

De forma predeterminada, se recuperarán los 10 primeros (página 0) customers de la base de datos, ordenados de forma ascendente según el campo id.

En el servicio, se construye un objeto ***PageRequest***, que toma los valores de los parámetros de paginación y ordenación enviados por el cliente de la API. El objeto ***PageRequest*** debe enviarse al repositorio.

```
@Override
public Page<Customer> getAll(int page, int size, String sortBy, String sortDirection) {
    Sort sort = Sort.by(Sort.Direction.fromString(sortDirection), sortBy);
    Pageable pageable = PageRequest.of(page, size, sort);
    return customerRepository.findAllProjectedBy(pageable);
}
```

Dado el código mostrado, podríamos invocar diversas URL para probar las salidas:

- http://localhost:8080/customers?size=5
- http://localhost:8080/customers?size=5&page=1
- http://localhost:8080/customers?size=5&page=2
- http://localhost:8080/customers?size=5&page=1&sortBy=lastname
- http://localhost:8080/customers?size=5&page=1&sortBy=lastname&sortDirection=desc