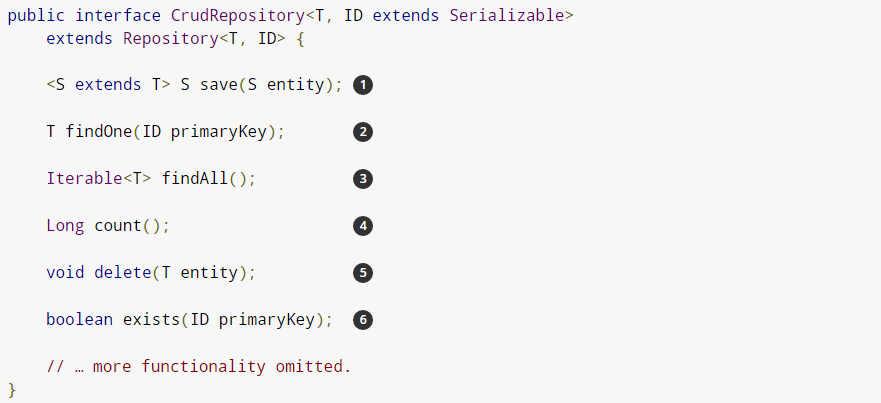
# Spring Data JPA

El objetivo de los repositorios de Spring Data es reducir significativamente la cantidad de código necesario para implementar capas de acceso a datos para diversos almacenes de persistencia.

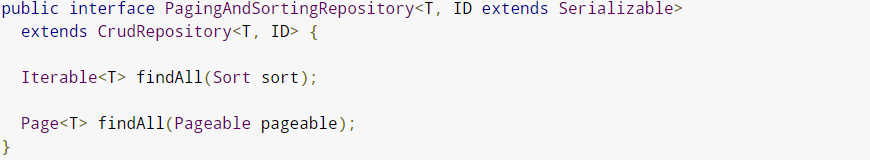
# Conceptos Clave

La interface central en la abstracción de los repositorios de Spring Data es ***Repository.*** Recibe la clase de dominio (entidad) a manejar así como el tipo de id de la clase. Esta interface actúa principalmente como un calificador que captura los tipos de datos con los que trabajará y que ayuda a descubrir interfaces que heredan de esta. La interface ***CrudRepository*** proporciona funcionalidad CRUD sofisticada para la entidad que está siendo manejada.



1. Guarda la entidad dada.
2. Retorna una entidad identificada por el id dado.
3. Retorna todas las entidades.
4. Retorna el número total de entidades.
5. Elimina la entidad dada.
6. Indica si una entidad con el id dado existe.

Construida sobre ***CrudRepository*** se encuentra la interface ***PagingAndSortingRepository*** que añade métodos adicionales para facilitar el acceso paginado a las entidades.



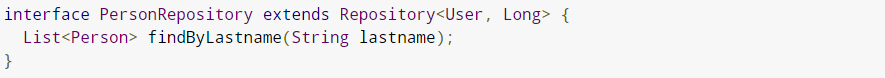
# Métodos de Consulta

Los repositorios CRUD estándar usualmente tienen consultas en el almacén de datos (base de datos). Con Spring Data, declarar estas consultas se convierte en un proceso de cuatro pasos:

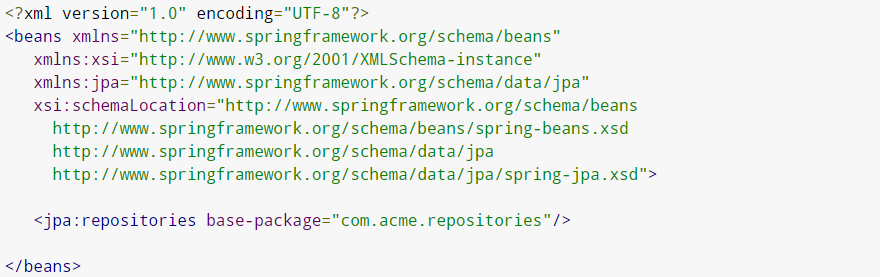
1. Declarar una interface heredando de ***Repository*** o alguna de sus subinterfaces, y tiparla con la clase de dominio y el tipo de id que ha de manejar.



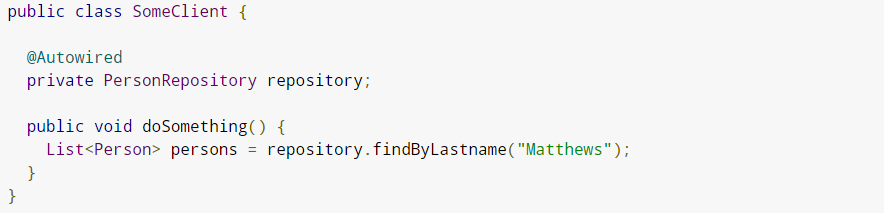
1. Declarar métodos de consulta en la interface.



1. Configurar Spring para crear instancias proxy de las interfaces.

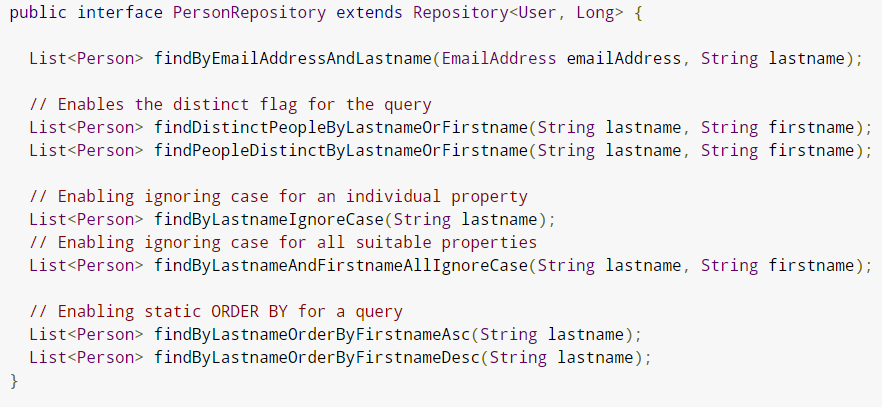


1. Inyectar la instancia del repositorio y usarlo



# Creación de Consultas

El mecanismo de construcción de consultas de Spring Data es útil para construir consultas para entidades del repositorio. El mecanismo separa los prefijos **find..By, read..By, query..By, count..By** y **get..By** del nombre del método y comienza a interpretar el resto. La clausula introductoria puede contener otras expresiones como **Distinct** para establecer que la consulta a ser generada debe contener distinct. Sin embargo, el primer **By** actúa como separador para indicar el inicio de las condiciones. En un nivel muy básico, puedes definir condiciones sobre las propiedades de las entidades y concatenarlas con **And** y **Or**.



El resultado actual de interpretar los métodos depende del almacén de persistencia para el cual se crea la consulta. Sin embargo, hay algunas cosas generales que notar:

* Las expresiones son usualmente propiedades combinadas con operadores que pueden ser concatenados. Puedes combinar propiedades con **And** y **Or**. También puedes hacer usos de operadores como **Between**, **LessThan**, **GreaterThan**, **Like**, etc. Los operadores soportados pueden variar por base de datos, así que es recomendable consultar la documentación de la base de datos.
* El interprete de métodos soporta indicar una bandera **IgnoreCase** para propiedades individuales (por ejemplo, **findByLastnameIgnoraCase**), o para todas las propiedades de un tipo que soporten **IgnoraCase** (normalmente instancias de String, por ejemplo **findByLastnameAndFirstnameAllIgnoraCase**).
* Puedes aplicar ordenamiento estático añadiendo la clausula **OrderBy** al método de consulta y proporcionando una dirección de ordenamiento (**Asc** o **Desc**).

# Spring Security

Spring Security es un framework para el control de autenticación y autorización de usuarios robusto y altamente configurable. Es el estándar de-facto para implementar seguridad en aplicaciones construidas en Spring Framework.

# Configuración Java de Spring Security

El primera paso es crear la clase de configuración de Spring Security. Esta configuración crea un filtro de Servlet conocido como ***springSecurityFilterChain***, el cual es responsable de toda la seguridad dentro de la aplicación.

A continuación se presenta el ejemplo mas básico de una clase de configuración de Spring Security:

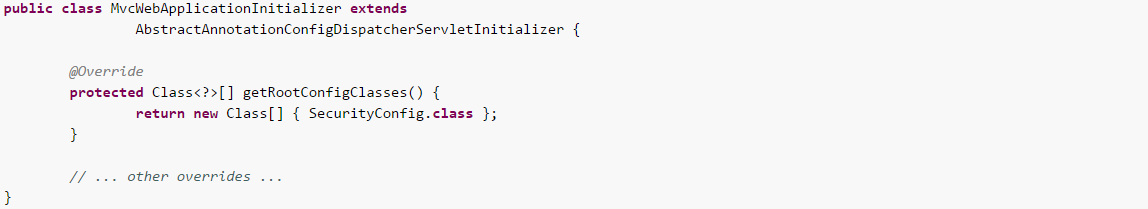


# AbstractSecurityWebApplicationInitializer con Spring MVC

El siguiente paso es registrar Spring Security dentro de nuestro contexto de Spring existente. Por ejemplo creando una clase como la que se presenta a continuación:

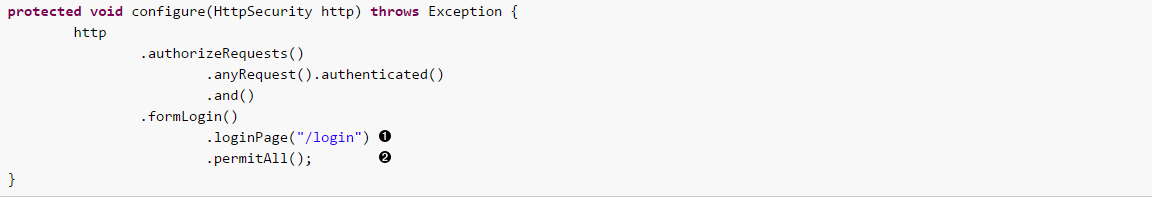


La clase anterior registrará el filtro ***springSecurityFilterChain***para cada una de las URL en nuestra aplicación. En seguida, debemos asegurarnos de que la clase de configuración de Spring Security sea cargada en nuestro ***ApplicationInitializer*** existente. A continuación se muestra un ejemplo de como registrar nuestra clase de configuración de Spring:



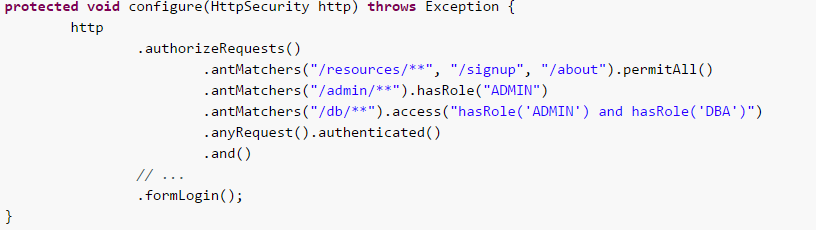
# Configuración Java y Formulario de Acceso

Para indicar que todas las urls deben requerir un usuario autenticado y en donde se encuentra el formulario de acceso de usuarios, es necesario actualizar nuestra clase de configuración de la siguiente manera:



# Autorizar Peticiones

El ejemplo anterior requiere solamente que el usuario esté autenticado. Podemos agregar requerimientos adicionales añadiendo multiples hijos al método ***http.authorizeReqeuests().*** Por ejemplo:



# Autenticación JDBC

Spring Security proporciona también soporte para autenticación basada en JDBC. El ejemplo siguiente asume que ya existe un ***dataSource*** dentro de la aplicación para implementar una autenticación basada en JDBC:

