**Spring Security Введение**

**Spring security это:**

**1.** Компонент Spring Framework

**2.** Является стандартом для обеспечения безопасноти в Spring приложениях.

**3.** Помогает реализовать функционал аутентификации и авторизации.

**4.** Может использоваться как в монолитных, так и в микросервисных приложениях.

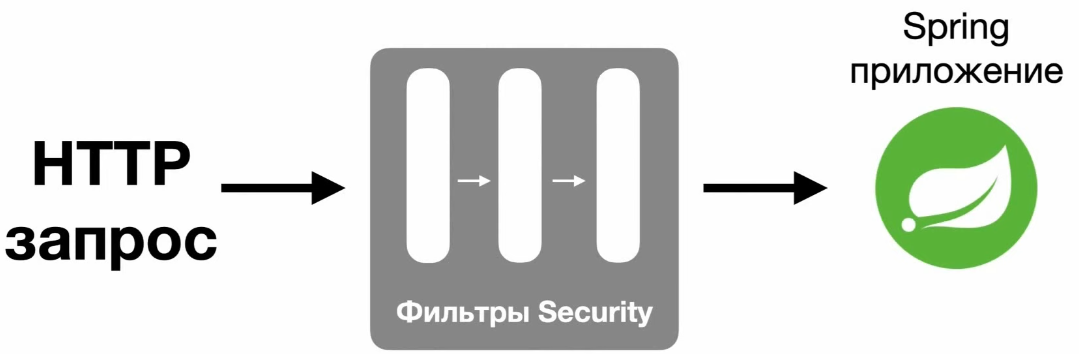
**Аутентификация и авторизация**

**Аутентификация** — это когда мы доказываем сервису или приложению, что это мы (посредствам ввдения логина и пароля, например).

**Авторизация** — после прохождения аутентификации мы зашли на какой нибудь сайт или сервис и после этого у нас появляется какая то роль в этом сервисе (можем быть обычными пользователями, администраторами и т. д.). Это разделение на роли это и есть авторизация тоесть авторизация говорит нам, что мы можем делать на сервисе после того как мы прошли аутентификацию. Одни роли авторизованы на одни дейтсвия, а другие роли на другие действия.

**Как Spring Security встраивается в приложение ?**

Spring Security работает с помощью так называемых фильтров.

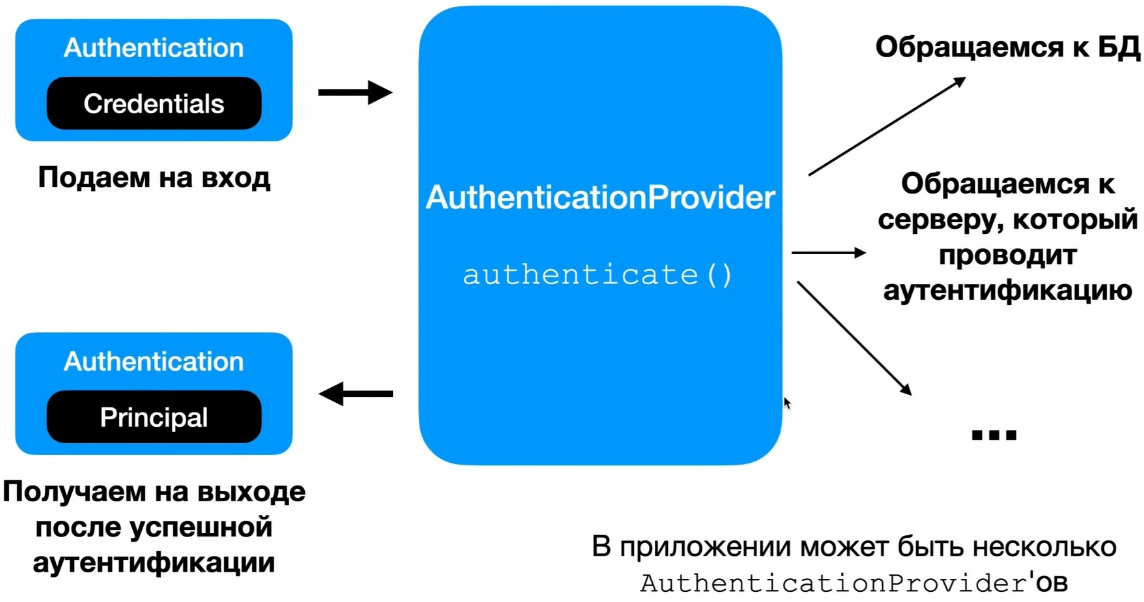
**Фильтр** — это объект который перехватывает все входящие HTTP запросы.  
  
  
  
  
  
  
  
  
Когда мы подключаем **Spring Security** в наше приложение у нас начинают работать фильтры и теперь каждый HTTP запрос к нашему **Spring** приложению всегда будет проходить через эти фльтры. Если в **Spring Security** мы настроим какие-то настройки безопасности, например скажем, что пользователь не должен получить доступ к какой-то странице если он не прошел аутентификацию, то теперь никак не возможно будет получить доступ к этой странице не пройдя аутентификацию потому-что HTTP запрос попадёт в фильтры **Spring Security** и если эти фильтры поймут, что пользователь не прошел аутентификацию, то они его не допустят к нашему **Spring** приложению. Таким образом наше **Spring** приложение получается полностью защищеном потому-что всё теперь зависит от этих фильтров, они могут допустить HTTP запрос к нашему приложению, а могут его не допускать.

**Как устроена аутентификация в Spring Security**

У нас есть интерфейс **AuthenticationProvider** и есть объект **Authentication**. Когда мы хотим реализовать аутентификацию в Spring Security – мы должны создать класс, который реализует интерфейс **AuthenticationProvider.** В этом интерфейсе есть одна сигнатура ***authenticate()***. Когда мы реализуем этот метод мы описываем в нём логику аутентификации пользователя. В этот метод на вход подается объект **Authentication** и этот метод тоже возвращает объект **Authentication**. Тоесть этот объект несет в себе логин и пароль пользователя когда этот пользователь пытается аутентифицироваться. Внутри объекта **Authentication** есть **Credentials** – это как раз и есть логин и пароль нашего пользователя.

Тоесть объект **Authentication** с **Credentials** подается на вход в качестве аргумента в метод ***authenticate()***, внутри этого метода проводится аутентификация. Аутентификация может проводится разными способами, можно например, обратиться к базе данных и посмотреть, что тот логин и пароль который передается в объекте **Authentication** совпадает с логином и паролем из таблицы пользователей. Так же в методе ***authenticate()*** мы можем обращаться к какому-то серверу, который проводит аутентификацию. Такая модель тоже используется когда есть специальный отдельный сервер, который проводит аутентификацию, на него мы посылаем **Credentials (логин и пароль)**, сервер сам проверяет, что логин и пароль подходящий и в ответ например посылает сообщение: «аутентификаия прошла успешно или нет». Когда аутентификация прошла или не прошла, то из метода ***authenticate()*** тоже возвращается объект **Authentication.** Внутри этого объекта помимо того, что в нём могут хранится логин и пароль еще может хранится объект **Principal**. **Principal –** это тот объект, который получается на выходе после успешной аутентификации. В **Principal** мы можем положить что угодно, но чаще всего в этот объект кладется сам пользователь, который был аутентифицирован и найден в базе данных.

И так как у нас могут быть разные способы аутентификации, то в приложении могут быть несколько **AuthenticationProvider`**ов, каждый со своей реализацией метода ***authenticate().*** Но всегда остается неизменным то, что у нас приходят какие -то **Credentials**, а на выходе у нас получается **Principal**.



**После успешной аутентификации** каждый раз, когда пользователь будет делать запрос к нашему приложению, мы будем иметь доступ к его объекту. Тоесть мы будем получать доступ к объекту **Authentication**, а внутри него будет лежать объект **Pricipal** и этот объект будет давать нам данные об аутентифицированном пользователе. Именно поэтому нам не нужно каждый раз вводить данные для аутентификации при следующих запросах если первый раз мы ввели верные и данные и смогли авторизоваться. Сервер нас запонмил.

**Как это работает ?** Дело в том, что когда проходит успешно аутентификация, то объект с **Principal**`ом помещается в сессию пользователя и при каждом следующем запросе к этому серверу этот объект с данными подгружается из сессии. За то, чтобы при каждом запросе этот объект подгружался из сессии отвечает фильтр Spring Security. И пока сессия пользователя действительна — мы можем получать доступ к объекту **Authentication** и **Principal** и не заставлять пользователя приходить аутентификацию еще раз.

**Сессия и cookies**

Именно с помощью **сессии** и **cookies** реализована базовая аутентификация в **Spring Security**.

Когда человек прошёл аутентификацию - у него на браузере назначаются так называемые **cookies**.  
**Cookies** – это пара ключ значение, которая находится в памяти браузера и которая посылается к нашему серверу при каждом запросе.

После успешной аутентификации мы назначаем какой-то **id** для пользователя. При этом одовременно в нашем **Spring** приложении создается сессия, сессию можно воспринимать как отдельный участок памяти, который предназначается для конкретного пользователя. В сессии хранятся данные этого пользователя (**Authentication с Principal**`ом).

Тоесть аутентифицированный пользователь сделал запрос к нашему приложению, дальше мы видим из его браузера(из cookie) пришел id, дальше мы ищем такую сессию в **Spring** приложении и из нее извлекаем **Principal.** Те данные которые мы извлекли из сессии кладутся в поток текущего пользователя (**ThreadLocal**).

Эти данные доступны только в этом потоке. При каждом запросе пользвателя к нашему приложению, в нашем приложении создается отдельный java поток для этого пользователя, и в этом потоке если сессия такая существует, то из сессии будут подгружены данные этого пользователя.

У **cookie** есть срок действия, по истечению срока действия эти **cookie** будут стёрты из памяти браузера.

