**SPRING SECURITY**

Graphical user interface, text, application

Description automatically generated

A picture containing text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**AUTHENTICATION:**

Graphical user interface

Description automatically generated with medium confidence

Identification of user based on his identity.

Most cases it will be username and password.

**KNOWLEDGE BASED AUTHENTICATION:**

­­­­­­When a user provides a password and username form his memory .

It is risky becoz if some one steals ur password then he can easily access the account.

**POSSESSION BASED AUTHENTICATION:**

Mobile : authenticated by text messages sent to user mobile phone.

Access cards/token cards: which we need to swap to enter buildings.

The idea is it is difficult to steal mobiles and cards from user is difficult than stealing password.

**MULTIFACTOR AUTHENTICATION =**

**KNOWLEDGE BASED AUTHENTICATION + POSSESSION BASED AUTHENTICATION**

**AUTHORISATION:**

Graphical user interface, text, application, chat or text message

Description automatically generated

**For authorization we need to do authentication first and then decide whether this user is allowed/authorized to do/access a particular task in system.**

Graphical user interface, text, application, chat or text message

Description automatically generated

**PRINCIPAL:**

**­­­**The currently logged in user into the system.

Graphical user interface, text

Description automatically generated

**AUTHORITY:**

Application grants permission to access a task if only that user is authorized to do that. This is called authority in terms of application security.

It is called **GRANTED AUTHORITY** in terms of spring security.

Authorities are always fine-grained.

**ROLES:**

In some cases we need to give same authorities to many users. Repeatedly providing authority to all persons is a tedious task. We can avoid that by using roles.

**ROLES ARE A GROUP OF AUTHORITIES.**

Suppose a retail store has some employees. Store clerk has some authorities . Department manager might be having some additional authorities along with store clerk authorities. Store manager may have still more additional authorities along with authorities of lower grade workers. We can reduce the redundancy of all above authorities by using roles.

**ROLES ARE MORE OF COARSE GRAINED.**

Diagram

Description automatically generated

**SPRING BOOT SECURITY:**

**PRACTICAL SCENARIO FROM BASICS:**

I am going to experiment mostly on TestingBoot application.

For starting just add a spring boot starter security in pom.xml

Mostly I am going to add notes in the application code itself.

Graphical user interface, text, application

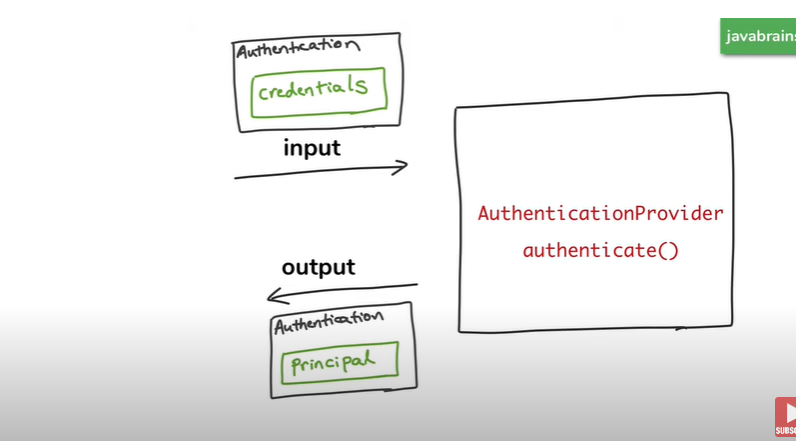
Description automatically generated

**HOW SPRING SECURITY AUTHENTICATION WORKS**

Diagram

Description automatically generated

**This what authenticate inside really.**



Authentication object have some internal method which are useful, u should explore them

Before authentication of the app, the Authentication object holds the credentials of the user and then after it only holds the principal.

Whatever the type of authentication (db , ldap ,oauth) , the authentication provider has to get details from somewhere to verify.

Diagram

Description automatically generated

Spring Security’s Authentication provider achieves it by using UserDetailsService to get the authentication details. UserDetailsService gets the auth details and stores in it. It just takes a user name and loads the UserDetails object which we will have all details of user. Instead of directly connecting to DB , the AuthenticationPorvider connects to UserDetailsService.

Diagram

Description automatically generated

The application may have different type of athentications to it. i.e DB , OAuth , LDAP etc.. We need different types of AuthenticationProviders to manage them. Each manages single type only.

Diagram

Description automatically generated

These different types of AuthenticationProviders are managed by other Object and it redirects request to different AuthenticationProviders based on the type of Auth user is using.

Diagram

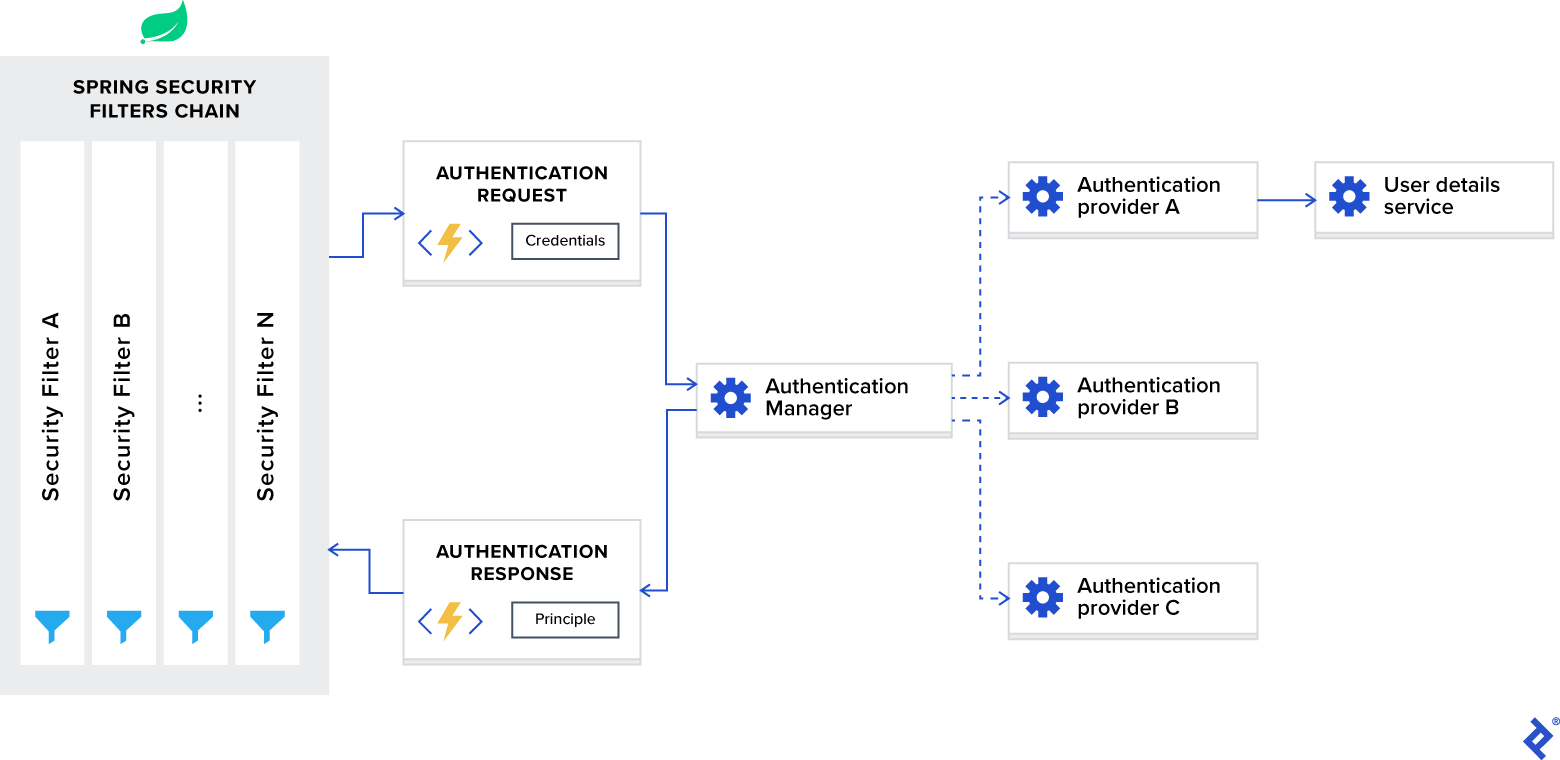
Description automatically generated

Every AuthenticationProvider has method suppots() which tells whether that object supports a particular type of Authentication and AuthenficationManager utilizes this methos and checks all the AuthenticationProviders to check whether the particular type of authentication is supported by that AuthenticationProvider. If it matches then the request is forwarded to that AuthenticationProvider and auth happens the same way as we did preivioulsy.

**OVERALL WORKILNG OF SPRING SECURITY**

Diagram

Description automatically generated



**JWT – JSON WEB TOKENS**A picture containing handwriting, text, child art, sketch

Description automatically generated

**WHY DO WE NEED JWT ?**

we process everything through HTTP protocol.

**HTTP IS A STATELESS PROTOCOL.**

**Once upon a time**

Suppose you have a [REST API](https://en.wikipedia.org/wiki/Representational_state_transfer) (e.g. GET /orders) and you want to restrict access to authorized users only.  
In the most naïve approach, the API would ask for a username and password; then it will be searched in a database for whether those credentials really exist. We check for *authenticity*. Finally, it will be checked if the *authenticated* user is also *authorized* to perform that request. If both checks pass, the real API will be executed. It seems logical.

A picture containing text, handwriting, whiteboard, diagram

Description automatically generated

A picture containing text, handwriting, screenshot, diagram

Description automatically generated

So in this way session id is stored and we don’t need to give auth details everytime we hit the request.

**A problem of state**

**The HTTP protocol is *stateless***, this means a new request (e.g. GET /order/42) won’t know anything about the previous one, **so** **we need to reauthenticate for each new request** (fig.1).

BELOW FIGURE:

Due to the stateless nature of HTTP protocol, every new API request needs a complete authentication.

A diagram of a server

Description automatically generated with low confidence

EVERTHING IS WELL WRITTEN IN THESE ARTICLES  
<https://medium.com/swlh/why-do-we-need-the-json-web-token-jwt-in-the-modern-web-8490a7284482>

A picture containing text, font, screenshot, design

Description automatically generated

**THIS IS HOW JSON WORKS**

A picture containing text, handwriting, sketch, drawing

Description automatically generated

In the above picture it explains how the token is retrieved form the application. First the user has to send the request using his credentials and then the application returns the jwt.

And then using that JWT user can access the applications api.

A picture containing text, handwriting, sketch, drawing

Description automatically generated

A picture containing text, screenshot, handwriting, diagram

Description automatically generated

**JSON WEB TOKEN STRUCTURE:**

A picture containing text, screenshot, font, design

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, handwriting, diagram, font

Description automatically generated

**This Article has good info about vulnerablilities of JWT**

**https://developer.okta.com/blog/2018/06/20/what-happens-if-your-jwt-is-stolen**