**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

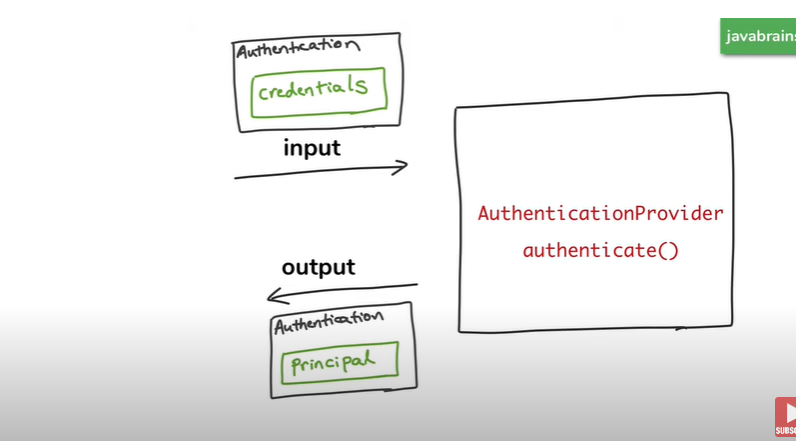
Description automatically generated

**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

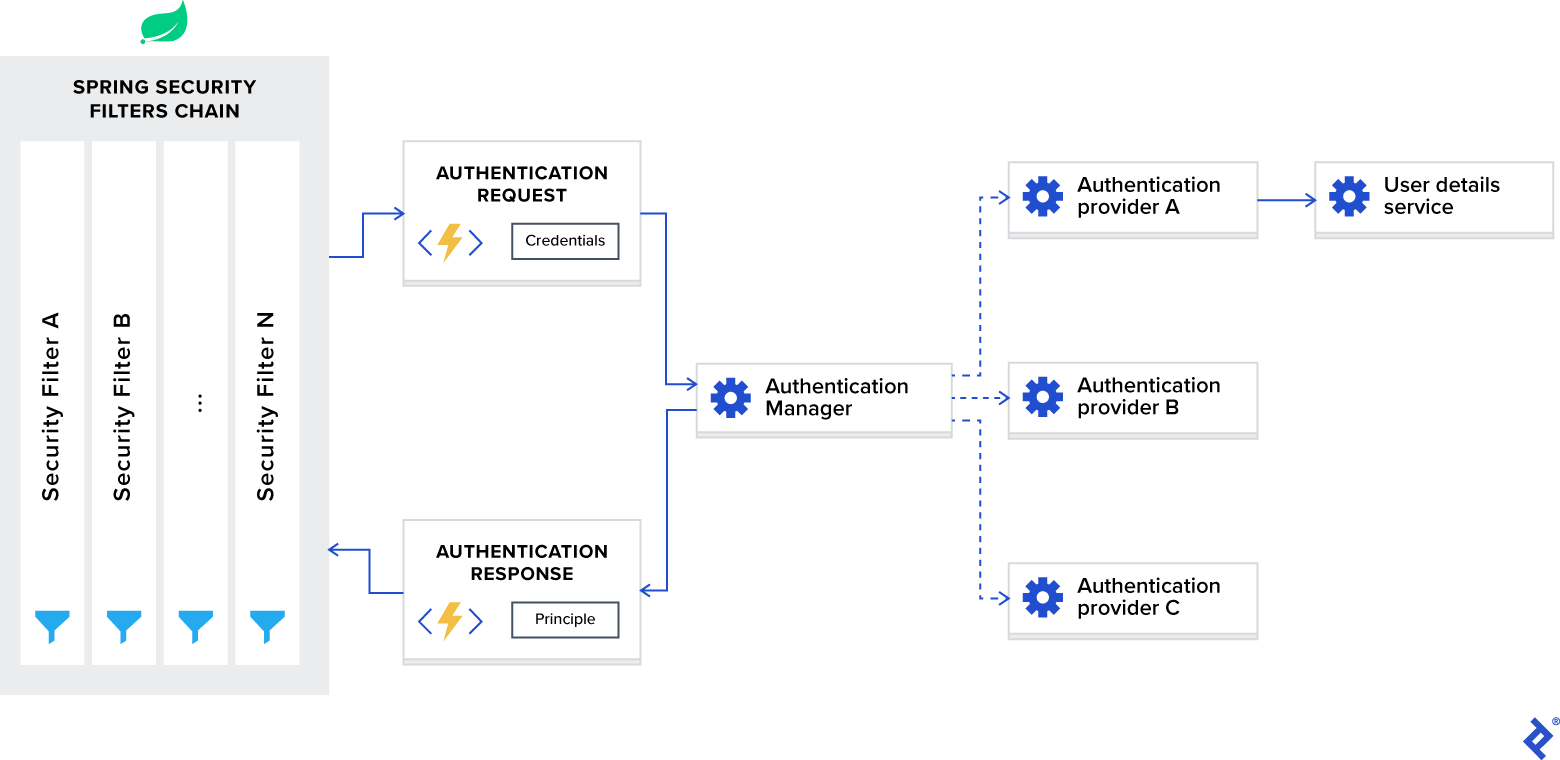
Description automatically generated

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

**OAUTH**

To begin at a high level, OAuth is not an API or a service: it’s an open standard for authorization and anyone can implement it.

More specifically, OAuth is a standard that apps can use to provide client applications with “secure delegated access”. OAuth works over HTTPS and authorizes devices, APIs, servers, and applications with access tokens rather than credentials.

There are two versions of OAuth: [OAuth 1.0a](https://tools.ietf.org/html/rfc5849) and [OAuth 2.0](https://tools.ietf.org/html/rfc6749). These specifications are completely different from one another, and cannot be used together: there is no backwards compatibility between them.

A picture containing text, font, screenshot, line

Description automatically generated

A picture containing text, font, screenshot, design

Description automatically generated

What the OAuth is basically explained in following video with interesting real-life example.

<https://youtu.be/t4-416mg6iU>

its important to learn about OAuth terminologies.

**RESOURCE SERVER:**

Resource Server is an application that provides the access token to the clients to access the Resource Server HTTP Endpoints. It is collection of libraries which contains the HTTP Endpoints, static resources, and Dynamic web pages.

**RESOURCE OWNER:**

A close-up of a sign

Description automatically generated with medium confidence

Suppose let’s assume we have two services PPS and Google Drive. PPS is used to print the photos of user. But user wants PPS to print the photos which are in his google account. User asks PPS to print some of photos in google drive. When PPS contacts Google Drive to give access to user’s photos, it won’t accept.   
and it also not a good way to ask user to provide his google account credentials. They may not provide.

Pls refer the above utube video to get more better insight.

A screenshot of a video

Description automatically generated with low confidence

**CLIENT SERVER:**

A close-up of a white background

Description automatically generated with low confidence

**AUTHORISATION SERVER:**

**It is the middlemen who helps different services communicate together. The resource server must have implemented it for client to access it. Resource owner provides the details for client to access the resource server via authorization server.**

Authorization Server is a supreme architectural component for Web API Security. The Authorization Server acts a centralization authorization point that allows your apps and HTTP endpoints to identify the features of your application.

A close-up of a sign

Description automatically generated with low confidence

**OAuth Flow 1:**

As in the above scenario, let’s continue.

* So now the client approaches the Resource server (which has the Authorization server in it) and asks to access the photos.
* The Authorization server doesn’t believe the client PPS and wants to verify the claim with the user (Resource owner) itself.
* So the OAuth server goes to user and asks him regarding the claim along with listing down what are all the authorizations that client server (PPS) can do with Resource server (Google Drive).
* If the user accepts and confirms to trust the client service (PPS) then authorization server sends the authorization token (short lived) to the client server.
* Now the client server (PPS) sends this authorization token again to Authorization server to get the access token.
* Now the client can use this access token to access resource server (Google Drive).
* Now the resource server takes the access token and validates it with Auth server.
* If everything goes fine then now the client can access the resources server according to authorities set by user(Resource owner).

# OAuth Flow 2: IMPLICIT FLOW:

This is same as above flow except for a single change.  
In this flow the granting authorization token is skipped and instead the Authorization server directly gives access token to client server and remaining same.

A screenshot of a computer

Description automatically generated with low confidence