# Security With Spring

## Introduction

Spring Security is a framework which provides various security features like authentication, authorization to create secure Java Enterprise Applications.

In addition to providing various inbuilt authentication and authorization options, Spring Security allows us to customize our authentication process as much as we want. Starting from a custom login page to our very own customized authentication providers and authentication filters, we can pretty much customize every aspect of the authentication process. We can define our own authentication process which can range from basic authentication using a username and a password to a complex one such as two-factor authentication using tokens and OTP’s. Also, we can use various databases – both relational and non-relational, use various password encoders, lock malicious users out of their accounts, and so on.

## Spring Security Features

* LDAP (Lightweight Directory Access Protocol)
* Single sign-on
* JAAS (Java Authentication and Authorization Service) LoginModule
* Basic Access Authentication
* Digest Access Authentication
* Remember-me
* Web Form Authentication
* Authorization
* Software Localization
* HTTP Authorization

### **LDAP (Lightweight Directory Access Protocol)**

It is an open application protocol for maintaining and accessing distributed directory information services over an Internet Protocol.

### **Single sign-on**

This feature allows a user to access multiple applications with the help of a single account(username and password).

### **JAAS (Java Authentication and Authorization Service) LoginModule**

It is a Pluggable Authentication Module implemented in Java. Spring Security supports it for its authentication process.

### **Basic Access Authentication**

Spring Security supports Basic Access Authentication that is used to provide username and password while making requests over the network.

### **Digest Access Authentication**

This feature allows us to make the authentication process more secure than Basic Access Authentication. It asks the browser to confirm the identity of the user before sending sensitive data over the network.

### **Remember-me**

Spring Security supports this feature with the help of HTTP Cookies. It remembers the user and avoids login again from the same machine until the user logs out.

### **Web Form Authentication**

In this process, web forms collect and authenticate user credentials from the web browser. Spring Security supports it while we want to implement web form authentication.

### **Authorization**

Spring Security provides this feature to authorize the user before accessing resources. It allows developers to define access policies against the resources.

### **Software Localization**

This feature allows us to make application user interfaces in any language.

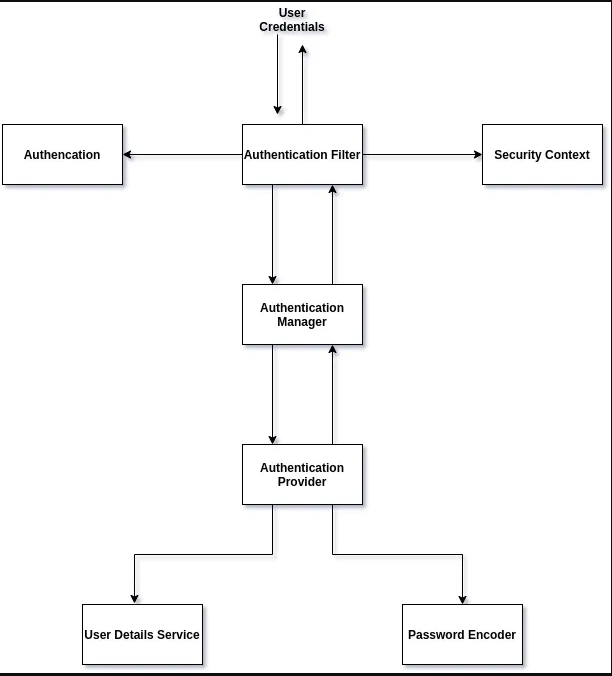
### **HTTP Authorization**

Spring provides this feature for HTTP authorization of web request URLs using Apache Ant paths or regular expressions.

## Difference between Authentication and Authorization

| Authentication | Authorization |
| --- | --- |
| In the authentication process, the identity of users are checked for providing access to the system. | While in the authorization process, person’s or user’s authorities are checked for accessing the resources. |
| In the authentication process, users or persons are verified. | While in this process, users or persons are validated. |
| It usually needs the user’s login details. | While it needs user’s privilege or security levels. |
| Authentication determines whether the person is a user or not. | While it determines **What permission do users have?** |
| **Example** : Employees in a company are required to authenticate through the network before accessing their company email. | **Example :** After an employee successfully authenticates, the system determines what information the employees are allowed to access. |

## Workflow Of Spring Security



### **AuthenticationFilter**

This is the filter that intercepts requests and attempts to authenticate it. In Spring Security, it converts the request to an Authentication Object and delegates the authentication to the AuthenticationManager.

### 

### **AuthenticationManager**

It is the main strategy interface for authentication. It uses the lone method authenticate() to authenticate the request. The authenticate() method performs the authentication and returns an Authentication Object on successful authentication or throws an AuthenticationException in case of authentication failure. If the method can’t decide, it will return null. The process of authentication in this process is delegated to the AuthenticationProvider which we will discuss next.

### **AuthenticationProvider**

The AuthenticationManager is implemented by the ProviderManager which delegates the process to one or more AuthenticationProvider instances. Any class implementing the AuthenticationProvider interface must implement the two methods – authenticate() and support(). First, let us talk about the supports() method. It is used to check if the particular authentication type is supported by our AuthenticationProvider implementation class. If it is supported it returns true or else false. Next, the authenticate() method. Here is where the authentication occurs. If the authentication type is supported, the process of authentication is started. Here this class can use the loadUserByUsername() method of the UserDetailsService implementation. If the user is not found it throws a UsernameNotFoundException.

On the other hand, if the user is found, then the authentication details of the user are used to authenticate the user. For example, in the basic authentication scenario, the password provided by the user may be checked with the password in the database. If they are found to match with each other, it is a success scenario. Then we can return an Authentication object from this method which will be stored in the Security Context, which we will discuss later.

### **UserDetailsService**

It is one of the core interfaces of Spring Security. The authentication of any request mostly depends on the implementation of the UserDetailsService interface. It is most commonly used in database backed authentication to retrieve user data. The data is retrieved with the implementation of the lone loadUserByUsername() method where we can provide our logic to fetch the user details for a user. The method will throw a UsernameNotFoundException if the user is not found.

### **PasswordEncoder**

Until Spring Security 4, the use of PasswordEncoder was optional. The user could store plain text passwords using in-memory authentication. But Spring Security 5 has mandated the use of PasswordEncoder to store passwords. This encodes the user’s password using one of its many implementations. The most common of its implementations is the BCryptPasswordEncoder. Also, we can use an instance of the NoOpPasswordEncoder for our development purposes. It will allow passwords to be stored in plain text. But it is not supposed to be used for production or real-world applications.

### **Spring Security Context**

This is where the details of the currently authenticated user are stored on successful authentication. The authentication object is then available throughout the application for the session. So, if we need the username or any other user details, we need to get the SecurityContext first. This is done with the SecurityContextHolder, a helper class, which provides access to the security context. We can use the setAuthentication() and getAuthentication() methods for storing and retrieving the user details respectively.

# **JWT AUTHENTICATION**

## What is a JSON Web Token?

JSON Web Token (JWT) is an open standard ([RFC 7519](https://tools.ietf.org/html/rfc7519)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

## What is the JSON Web Token structure?

#### In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

#### Header

#### Payload

#### Signature

## Header

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

For example:

{

"alg": "HS256",

"typ": "JWT"

}

## Payload

The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data. There are three types of claims: registered, public, and private claims.

An example payload could be:

{

"sub": "1234567890",

"name": "John Doe",

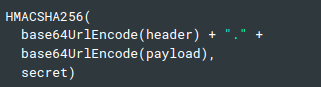
"admin": true

}

## Signature

To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:



The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

