# 1. What are Authentication and Authorization ?

Authentication is when an entity proves its own identity. In other words, Authentication proves that you are who you say you are. We can liken it to an identity card that tells a person making a request by a trusted authority, like a police officer, who you really are and can be used as proof.

Authorization is when an entity proves its access right, its identity. In other words, Authorization proves that you have the right to make requests. For example, you can be in the areas where only the personnel can enter at your workplace and no one will question you because you are already an employee. However, a customer will not be able to enter these areas because he is not authorized.

# 2. What is Hashing in Spring Security ?

Hashing is the name given to the process of creating a fixed-size output from different-sized inputs. This is done using mathematical formulas (implemented as hashing algorithms) known as hash functions.

Although not all hash functions involve the use of cryptography, cryptographic hash functions form the basis of cryptocurrencies. With these functions, blockchains and other distributed systems can achieve a high level of data integrity and security.

# 3. What is Salting and why do we use the process of Salting ?

It is a concept that denotes password hashing. To do this, a different hash value is added to the end of the password. In this way, while the security layer is created in the hashing process, especially brute force attacks are prevented. Brute Force attack is the process of a computer or botnet network trying all possibilities with numbers and letters until it finds the password.

In the salting operation, an additional value of 'salt' is added. The goal is to add the value 'salt' to the end of the password, then hash it.

# 4. What is “intercept-url” pattern ?

The intercept-url element defines a pattern which is matched against the URLs of incoming requests using an ant path style syntax.The access attribute defines the access requirements for requests matching the given pattern.

# 5. What do you mean by session management in Spring Security ?

Session management is the process of securely handling multiple requests to a web-based application or service from a single user or entity. HTTP is used to communicate between websites and browsers, and a session is a series of HTTP requests and transactions created by the same user. The session management implementation specifies the process for sharing and continually exchanging the session ID between the user and the web application.

We can control exactly when our session gets created and how Spring Security will interact with it:

* always – A session will always be created if one doesn't already exist.
* ifRequired – A session will be created only if required (default).
* never – The framework will never create a session itself, but it will use one if it already exists.
* stateless – No session will be created or used by Spring Security.

# 6. Why we need Exception Handling ?

Exception runtime, that is, the errors that occur while the application is running. Some of these errors can be tolerated, while others cause the application to stop completely.

In Java, all errors are represented by classes, so when an error occurs an error object is created. The ancestor of all errors in Java is the Throwable class. The Throwable class has two direct subclasses, Exception and Error. The Error class is used for JVM related errors that are not directly related to our code and these errors are beyond our control.

Checked / Unchecked exception

# 7. Explain what is AuthenticationManager in Spring security ?

AuthenticationManager allows us to plug in other authentication schemes. It specifies how the authentication is performed by Spring Security Filters. The returned Authentication is then set on the SecurityContextHolder by Spring Security Filters which called the AuthenticationManager. Although the implementation of AuthenticationManager could be anything, the most used implementation is ProviderManager.

We can set the SecurityContextHolder directly and are not required to use an AuthenticationManager, if we are not integrating with Spring Security Filters .

# 8. What is Spring Security Filter Chain ?

Spring Security's web infrastructure is based entirely on standard servlet filters. It doesn't use servlets or any other servlet-based frameworks (such as Spring MVC) internally, so it has no strong links to any particular web technology. It deals in HttpServletRequests and HttpServletResponses and doesn't care whether the requests come from a browser, a web service client, an HttpInvoker or an AJAX application.

Spring Security maintains a filter chain internally where each of the filters has a particular responsibility and filters are added or removed from the configuration depending on which services are required. The ordering of the filters is important as there are dependencies between them. If you have been using namespace configuration, then the filters are automatically configured for you and you don't have to define any Spring beans explicitly but here may be times when you want full control over the security filter chain, either because you are using features which aren't supported in the namespace, or you are using your own customized versions of classes.

# 9. What are the differences between OAuth2 and JWT ?

The OAuth2 specification defines a delegation protocol that provides clients with secure access to the user resources on a service provider. Such an approach prevents the user from the necessity to enter his password out of the service provider: the whole process is curtailed to clicking the «I agree to provide access to ...» button. The idea is that having one secure account, the user can use it for identity verification on other services, without disclosing his password.

JWT tokens are JSON encoded data structures contains information about issuer, subject (claims), expiration time etc. It is signed for tamper proof and authenticity and it is supported by all devices.

# 10. What is method security and why do we need it ?

This tutorial will explore two ways to configure authentication and authorization in Spring Boot using Spring Security. One method is to create a WebSecurityConfigurerAdapter and use the fluent API to override the default settings on the HttpSecurity object. Another is to use the @PreAuthorize annotation on controller methods, known as method-level security or expression-based security. The latter will be the main focus of this tutorial. However, I will present some HttpSecurity code and ideas by way of contrast.

The first authentication method is HttpSecurity, which is global and is by default applied to all requests. Finer-grained control is possible, however, using pattern matching for endpoints, and the fluent API exposed by the HttpSecurity is quite powerful. This is where configuration options such as OAuth 2.0, Form Login, and HTTP Basic are exposed. It is a great place to set global authentication policies.

Method-level security is implemented by placing the @PreAuthorize annotation on controller methods (actually one of a set of annotations available, but the most commonly used). This annotation contains a Spring Expression Language (SpEL) snippet that is assessed to determine if the request should be authenticated. If access is not granted, the method is not executed and an HTTP Unauthorized is returned. In practice, using the @PreAuthorize annotation on a controller method is very similar to using HttpSecurity pattern matchers on a specific endpoint. There are some differences, however.

# 11. What Proxy means and how and where can be used ?

Proxy is a structural design pattern that provides an object that acts as a substitute for a real service object used by a client. A proxy receives client requests, does some work (access control, caching, etc.) and then passes the request to a service object.

The proxy object has the same interface as a service, which makes it interchangeable with a real object when passed to a client. Some examples of proxies in standard Java libraries:

*java.lang.reflect.Proxy*

*java.rmi.\**

*javax.ejb.EJB (see comments)*

*javax.inject.Inject (see comments)*

*javax.persistence.PersistenceContext*

# 12. What is Wrapper Class and where can be used ?

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

**Need of Wrapper Classes**

They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).

The classes in java.util package handles only objects and hence wrapper classes help in this case also.

Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

An object is needed to support synchronization in multithreading.

# 13. What is SSL ? What is TLS ? What is the difference ? How can we use them ?

## 13.1. SSL

Netscape developed SSL in the year 1994. It was envisioned as a system that will ensure secure communication between client and server systems on the web. Gradually, the IETF (the Internet Engineering Task Force) picked up the protocol and standardized it as a protocol. Two versions of SSL followed that ironed out the vulnerabilities found in version 1. The current SSL version is SSL 3.0. If we look at below history, we can assume that IETF seriously attempted to secure online data with robust security at its best.

SSL 1.0 Due to security flaw, SSL 1.0 was not released.

SSL 2.0 SSL v2.0 was the first public release of SSL by Netscape. It was released in February 1995 but there were design flaws that compelled Netscape to release SSL v.3. However, SSL v.2.0 was deprecated in 2011.

SSL 3.0 SSL v3 was an upgrade version of earlier version SSL v2.0 that fixed few security design flaws of SSL v2.0 However, SSL v3.0 deemed insecure in 2004 due to the POODLE attack.

## 13.2.TLS

TLS means Transport Layer Security, which is a cryptographic protocol successor of SSL 3.0, which was released in 1999.

TLS 1.0 TLS 1.0 which was upgrade of SSL v.3.0 released in January 1999 but it allows connection downgrade to SSL v.3.0.

TLS 1.1 After that, TLS v1.1 was released in April 2006, which was an update of TLS 1.0 version. It added protection against CBC (Cipher Block Chaining) attacks. In March 2020, Google, Apple, Mozilla and Microsoft has announced for deprecation of TLS 1.0 and 1.1 versions.

TLS 1.2 TLS v1.2 was released in 2008 that allows to specification of hash and algorithm used by the client and server. It allows authenticated encryption, which was added more support with extra data modes. TLS 1.2 was able to verify length of data based on cipher suite.

TLS 1.3 TLS v1.3 was released in August 2018 and had major features that differentiate it with its earlier version TLS v1.2 like removal of MD5 and SHA-224 support, require digital signature when earlier configuration used, compulsory use of Perfect forward secrecy in case of public-key based key exchange, handshake messages will now be encrypted after “Server Hello”.

## 13.3. Differences between SSL and TLS

However, the differences between SSL and TLS are very minor. In fact, only a technical person will be able to spot the differences. The notable differences include:

**Cipher suites**

SSL protocol offers support for Fortezza cipher suite. TLS does not offer support. TLS follows a better standardization process that makes defining of new cipher suites easier like RC4, Triple DES, AES, IDEA, etc.

**Alert messages**

SSL has the “No certificate” alert message. TLS protocol removes the alert message and replaces it with several other alert messages.

**Record Protocol**

SSL uses Message Authentication Code (MAC) after encrypting each message while TLS on the other hand uses HMAC — a hash-based message authentication code after each message encryption.

**Handshake process**

In SSL, the hash calculation also comprises the master secret and pad while in TLS, the hashes are calculated over handshake message.

**Message Authentication**

SSL message authentication adjoins the key details and application data in ad-hoc way while TLS version relies on HMAC Hash-based Message Authentication Code.

These are the essentially differences between an SSL and TLS certificate. Like I mentioned before, it takes a trained eye to understand the differences.

# 14. Why do you need the intercept-url ?

Most web applications using Spring Security only have a couple of intercept-urls because they only have very basic security requirements. You need to have unauthenticated access to the login and login-error screens and usually some aspect of the public site, so that can be a few URL patterns. Then there's often an admin section, and then everything else is ROLE\_USER.

If you need more roles, it's customary to associate them with top level URL path components. Although it's not required, it makes it easier to be sure that resources are appropriately protected.

*<http realm="Contacts Realm" use-expressions="false">*

*<intercept-url pattern="/index.jsp" access="IS\_AUTHENTICATED\_ANONYMOUSLY"/>*

*<intercept-url pattern="/login.jsp\*" access="IS\_AUTHENTICATED\_ANONYMOUSLY"/>*

*<intercept-url pattern="/admin/\*" access="ROLE\_ADMIN"/>*

*<intercept-url pattern="/secret/\*" access="ROLE\_SECRET"/>*

*<intercept-url pattern="/\*\*" access="ROLE\_USER,ROLE\_ADMIN,ROLE\_SECRET"/>*

*<http-basic/>*

*</http>*

You have to decide, based on your use cases, whether you want to allow people to have multiple roles. It's a bit harder to manage in the app, so most people with simple security set it up so that users have exactly one role and then they allow multiple roles to access protected content. The other way to do it, of course, is one role per URL pattern and give people multiple roles.