**What is the difference in spring security oauth2 flow when requesting a default access token vs when requesting a jwt (Json Web Token) access token?**

The client authentication and authorization process is all the same in both scenarios.

The difference is in what happens during an access token generation mehcanism is trigger. In case of the default access token, an entry is made in oauth\_access\_token and oauth\_refresh\_token tables, whereas, in case of the jwt access token these tables are not touched, in fact, no such entries are made in any of the tables.

Since, in case of a jwt access token, all details are stored in the token itself, nothing needs to be stored in the db – one of the reasons why a jwt token is so huge. So, everytime a jwt access token is sent by the client, the details in the token are verified and updated by the auth server but no db calls are made, which is not the case with a default access token.

**What is the difference in spring security oauth2 flow when a protected resource request is done with a default access token vs a jwt access token?**

Here, the procedure to validate the token is different in these two mechanisms. For a default access token based protected resource request, the token gets authenticated after a db call is made to the oauth\_access\_token table to fetch token details and then validate it with the input token, whereas, in jwt based access token protected resource request, for this validation to happen no such db calls are made since no data regarding the token is stored in the db, all the data is part of the token itself. So, this is where the difefrence is.

After this whatever the system does to fetch the protected resource is all the same, no difference there.

**What are Maven and Gradle?**

Maven and Gradle are build automation tools. They are like Ant, but Ant does not have great support for spring boot.

**What is the use of Maven or Gradle in spring?**

They are used for configuration and dependency management. They have great support for spring boot.

**What is the basic difference between Maven, Gradle and Ant?**

These are all build automation tools and Ant was the first of these 3. Ant and Maven both use xml for build configuration whereas Gradle uses Groovy.

**What is pom.xml?**

Pom.xml is the build and dependency management file for maven.

**What is the use of <parent> tag in pom.xml?**

The ‘parent’ tag allows us to maintain the versions of different artefacts and depdendencies that will be used in building and implementing the application. The version numbers of other dependent aretifacts need not be explicitly mentioned since maven will automatically configure them on the basis of the parent.

**What specific problem does the <parent> tag pom.xml solve?**

It solves the problem of maintaining the versions of different aretefacts and dependencies that are required for building and implementing the application. Maven automatically configures the versions of all dependencies managed by the parent artifact.

**What specific problem does the spring-boot-starter-parent artifact solve?**

It provides default configurations for our application and a complete dependency tree that we can use to quickly build our Spring Boot application.

Once, we’ve declared the starter parent as our project’s parent, we can pull any dependency from the parent by just declaring it in our dependencies tag.

**What is the parent artifact which helps us in providing a default build and dependency configurations for our spring boot applications?**

spring-boot-starter-parent artifact

**What is the use of spring-boot-starter-web artifact?**

‘spring-boot-starter-web’, is a starter for building web, including RESTful applications using Spring MVC.

**What does spring-boot-starter-web artifact offer to spring boot applications? And what other important dependencies does it have?**

This artifact uses Tomcat as the default embedded container. And hence we get the libraries supporting all these (including jackson related).

**What is the use of spring-boot-starter-data-jpa artifact?**

This is a starter for using Spring Data Jpa with Hibernate.

**Name some of it’s dependencies of spring-boot-starter-data-jpa artifact?**

Hibernate Core, JDBC and other related stuff with this.

**What is the use of spring-boot-starter-security artifact?**

This is a starter for using Spring Security.

**What is the programming paradigm that spring security depends on? And why?**

Spring Security is implemented/based on Aspect-Oriented Programming paradigm. Because, AOP as a paradigm aims to increase the modularity of a program by separating the cross-cuttin concerns and since Security is a cross-cutting concern, Spring Security is implemented using AOP.

**What are some of the dependencies of spring-boot-starter-security artifact?**

This depends on AOP and hence brings in its libraries also.

**What is a cross-cutting concern?**

A cross-cutting concern is an aspect of a program which is not core to the business logic of the application but it is an integral and important part of every layer of the application. Logging and Security are examples of cross-cutting concerns, which are not necessarily related to the business logic yet they have be managed across the application at all layers.

**Why Security in Spring is considered a cross-cutting concern?**

Becuase it is not generally part of the business logic of the application but it is an aspect of porogram which needs to be handled across all layers of the application and If such an aspect is not treated as a cross-cutting concern then it will lead to redundancy and poor modularity of the program.

**What is the purpose of spring-security-oauth2 artifact? Is this artifact also managed by Spring?**

This is not managed by Spring. This is starter for using Oauth2 with Spring. You need to mention the version for this dependency in your pom.xml for it to work.

**What is the purpose of spring-boot-starter-logging?**

This is a starter for logging with Logback, which is the default logging starter for sprongboot.

**What is the default logging starter for spring boot?**

Logback is the default logging starter for springboot.

**What is the size of the integer, text and Date data types in postgresql?**

integer is a signed 4 byte integer and text is a variable length character string. Date also takes 4 bytes.

**What is the java equivalent of postgresql’s integer and text data types?**

Integer and String.

**What is the meaning and purpose of @Entity annotation?**

It basically means that this class is an entity and has to be persisted, meaning there should be table mapping for this model in the database.

**What is the meaning of an Entity?**

Basically, an Entity is something that is distint and can have independent existence. That is, it has some certain attributes of its own.

**What is the difference between having an generation strategy of AUTO vs IDENTITY i.e. @GeneratedValue(strategy=GenerationType.AUTO) vs @GeneratedValue(strategy=GenerationType.IDENTITY)?**

**When do you use @Temporal?**

It seems, according to the Java Persistence Sepcs, any field or property of type java.util.Date or java.util.Calendar has to be annot. With @Temporal.

**When do you use @Column annotation?**

This is to tell the jpa provider about the mapping column for this particular attribute in the database table, since the column name of the table in the db des not match with the attribute name here, we have to explcitly call it out using this property.

**what is the difference between a field and a property?**

According to the official oracle documentation of java, it seems, a field is a data memeber of a class and a property is something whose value you can set/change. So, basically, if a field is settable, it can be called a property.

**What is the easiest way to configure spring related components to logging in application.properties?**

The easiest way to do this is to look at your pom.xml and pick up the group ids of all the dependencies and prefix logging.level to them and add them to your application.properties file. These group ids are nothing but top-level packages of the various libraries/dependencies that have been added to the project. Only thing is, these dependencies themselves depend on other libraries (you can check the dependency hierrachy from eclipse by opening you pom.xml and then look for the Dependency Hierarchy tab) which may not be configured for logging

**where do you put your application.properties file in your project structure?**

In src/main/resources folder.

**What is connection pooling? Why is it used?**

A connection pool is bascially a cache of database connections. Since acquiring a new conenction is a costly affair, whenever a connection is created, it is added to the pool and applications are supplied these pooled connections as and when required and when an application is done with a connection, it is released back to the pool, ready for use by some other piece of the application. By this, the adhoc, dynamic and costly affair of creating connections and executing commands all happens faster.

**What is the default connection pool library used by spring?**

By default when you use either spring-boot-starter-jdbc or spring-boot-starter-data-jpa, you automatically get a dependency to HikariCP.

Spring prefers HiakriCP for its performance and accuracy even for production databases.

**What is the algorithm used by spring to decide which connection pool provider/library to use in case of availability of multiple connection pool options?**

The algorithm for choosing a connection pool which is used by Spring is as follows:

1.If HikariCP is available, we always choose it.

2.Otheriwse, If Tomcat pooling DataSource is available, then this chosen.

3. If none of these are there, then Commons DBCP2 is chosen.

You can also explicitly mention what to want to use in the pom.xml.

**What is the central core interface of spring data?**

Repository interface is the central core interface in spring data and all other specialised repositories like CrudRepository, JpaRepository etc. Extend this.

**Name some interfaces that extend the central core interface of spring data?**

Specialised repositories like CrudRepository, JpaRepository etc. Extend this core Repository interface.

**What are the parameters required by the central core interface of spring data?**

The Repository interface of spring data basically takes 2 parameters – one is the domain type or the entity/model name to manage and it’s Id type.

**What is the purpose of @Repository?**

This is to indicate that the annotated component is a Persistence Layer component and it’s need to be auto-detected during classpath scanning by spring.

**@Repository is a specialization of which annot. In spring?**

This is one of the specilizations of @Component.

**What is the purpose of @Component in spring?**

The purpose of @Component is to identify the components marked with this or its specialized versions to be auto-detected during classpath scanning.

**What are the specialized versions of @Component in spring?**

@Controller, @Service and @Repository

**The following is a custom method in one of interfaces that extend the spring data repository interface:**

Company findByName(String companyname);

**Explain how spring parses this stuff?**

The query creation mechanism of Spring Data is such that, it basically strips the find..By, get..By, query..By and count..by and starts parsing from there. The first ‘By’ encountered becomes the delimiter and then begins the actual criteria.

So, in our case, when Spring Data encounters findByName, it first strips the findBy and it then starts looking for ‘name’ property in the entity that this Repository interface is managing and then it starts making a criteria accrodingly.

**When to use JpaRepository vs CrudRepository?**

**What is a Service Layer in an enterprise application?**

A Service Layer basically acts as an intermediary between your Repository and Controller where you can set up your business logic.

**What is the purpose of Service Layer in an enterprise application?**

It helps in neatly separating the persistence layer and the view/controller layer from the core business logic.

**What is the purpose of @Service annot in spring?**

This is basically to enable the component annotated with this annotation to be auto-detected during classpath scanning by spring. This is a service layer annotation of @Compoenent.

**What is the purpose of @Autowired annot in spring?**

This annot, basically, injects the dependencies.

**What is the use of @RestCOntroller in spring?**

This is a special case of @Controller. And the @RestController helps us in creatig RESTful web services controllers.

**What is the use of @Controller in spring?**

The @Controller allows us to create Spring MVC Controller based applications.

@Controller, is a specialization of @Component for the view layer.

**What is the difference between @RestController and @Controller in spring?**

The main difference between these two is in the way the HTTP Response body is created. While the Spring MVC Controller depends on the view technology, the RESTful web service controllers simply returns the object and the object data is written to the HTTP Stream as a json/xml.

So, when you annot. Your class with @RestController, all the return values/objects are then directly returned as part of the HTTP Stream as json/xml.

**When do you use @Controller vs @RestController in spring?**

For a Spring MVC Based Application, that is, when your application has a view configured in the view layer, you annotate your controller with @Controller whereas for a RESTful Web Services based application, you annotate your controller with @restController.

**What is the primary difference in Spring MVC Application vs a RESTful WebServices Application in Spring?**

The main difference between these two is in the way the HTTP Response body is created. While the Spring MVC Controller depends on the view technology, the RESTful web service controllers simply returns the object and the object data is written to the HTTP Stream as a json/xml.

**What is use of @GetMapping in spring?**

The purpose of @GetMapping is to map the http GET requests onto specific handler methods.

**What is use of @RequestMapping in spring?**

The purpose of @RequestMapping is to map the http requests onto specific handler methods.

**What is the difference between @RequestMapping and @GetMapping in spring?**

@GetMapping is equivalent to a @RequestMapping with type GET. There is no other difference.

**What is the purpose of @PathVariable in spring?**

@PathVariable, basically indicates that this method param of a method in which it is annotated, is bound to the request URI template variable.

So, basically, the request uri, for e.g. <http://localhost:8080/showCompany/xyz>, the ‘xyz’ in this request uri will be mapped to that method param of this method which is annotated with this annotation.

**What is the difference between @PostMapping and @RequestMapping in spring?**

The @PostMapping is equivalent to @RequestMapping with request type POST. So, basically, this says, that this particular method handles only POST type requests.

**What is the purpose of @PostMapping in spring?**

This is to map http POST requests onto specific handler methods.

**What is the purpose of @RequestBody annot. in spring?**

The @RequestBody anot.basically indicates that the method parameter to which this annotation is applied to, is to be bound to the body of the web i.e. http request and not to any uri template variable. That is, the value of method parameter, will not be part of the URI rather will be part of the body of the request.

**How do you configure or map the method parameter of a method in a controller to the body of the Http Request in spring?**

We annotate the method parameter with @RequestBody annot. Which basically binds the method param to the body of the http request and then the HttpMessageConverter will convert and map the method param to the body of the request.

**How do you configure the custom sequence generation strategy for the primary key field of your entity in spring?**

Use the GenerationType as SEQUENCE in the @GeneratedValue annot. And also pass in the sequence generator name in it.

And, then use the @SequenceGenerator, which basically informs our jpa about the sequence to use (by its sequenceName attribute) and also mentions the allocation size i.e. by how much should the sequence values be incremented.

**what is the purpose of nextval function in postgresql?**

**What are the various HttpStatus s and what do they mean?**

**What is the difference between 200 and 201 httpstatus?**

**What is the purpose of ResponseEntity in Spring?**

ResponseEntity is an extenstion of HttpEntity which allows us to add a HttpStatus code in the response.

**How do you add a HttpStatus code in the response of your RESTful Web Service Controller in spring?**

We can do that using ResponseEntity, which is basically an extension of HttpEntity, which allows us to add HttpStatus code in the response.

**What is the purpose of AuthorizationServerConfigurerAdapter class in spring?**

The AuthorizationServerConfigurerAdapter class implements AuthorizationServerConfigurer interface and this interface is basically an convinience interface for configuring Oauth2 Authorization Server.

**In what high-level package is AuthorizationServerConfigurerAdapter class part of in spring?**

This is part of the springframework security oauth2 package.

**What configurations can you make by extending**  **AuthorizationServerConfigurerAdapter class in spring?**

This class implements AuthorizationServerConfigurer interface.

And this interface, has 3 overloaded asbtract methods by the name of ‘configure’.

One is to configure the security of the authorization server i.e. the /oauth/token endpoint. And, the next one, is to configure the client details. And, the third one, is to configure the non-security features of the authorization server, like token stores, grant types etc.

**What is the purpose of @EnableAuthorizationServer?**

This is a convinience annot. Which basically enabling an Authorization Server.

**What annotation would you use to configure and enable an Authorization server in spring boot?**

@EnableAuthorizationServer

**How would you configure client details in AuthorizationServerConfigurerAdapter i.e. in Oauth2 Spring?**

We override the configure(ClientDetailsServiceConfigurer) method.

And, configure the jdbc datasource which needs to be used to fetch client details from the db.

**How would you configure the client details in an Oauth2 Spring Application?**

Create a class and make it extend AuthorizationServerConfigurerAdpater class (which implements AuthorizationServerCOnfigurer interface) and override it’s configure method which takes ClientDetailsServiceConfigurer as a method param to set the jdbc datasource which needs to be invoked to fetch the oauth client details from the database.

**What are the tables that are necessary for implementing an oauth2 based spring implementation?**

As per Spring Security Oauth2 requirements, we need OAUTH\_CLIENT\_DETAILS, OAUTH\_ACCESS\_TOKEN, OAUTH\_REFRESH\_TOKEN for password grant flow.

Apart from the above, we need OAUTH\_CLIENT\_TOKEN for Client Password/Token grant flow and

OAUTH\_CODE, OAUTH\_APPROVALS for Authorization Code grant type flow.

**What all details are stored in oauth2 tables?**

OAUTH\_CLIENT\_DETAILS stores:

client id, the username of the client.

Resource id, the id of the resource to which this client has access to. This is to be configured in the resource server config.

Client secret – the client password.

Scope – read and/or write scope

authorized grant types – all grant types this client will allow.

Authorities – authority of the client

access and refresh token validity – in seconds

And, OAUTH\_ACCESS\_TOKEN stores the access token, refresh token, the client id to which this access token belongs to.

And, OAUTH\_REFRESH\_TOKEN stores only the refresh token.

**what is bytea data type in postgresql?**

The bytea data type in postgresql is for storing binary strings. You prefer binary strings vs character strings when your data is to be stored as ‘raw bytes’ as against ‘text’.

**What is the data type in postgresql for storing binary string?**

bytea

**What do you have to do/configure in a spring boot oauth2 application to store/retrieve access and refresh tokens that are generated in the database (assumming we have already created the necessary oauth tables)?**

**How and where are you going to configure the jdbc token generation mechanism?**

In the Authorization Server Config class, that is, the class that extends AuthorizationServerConfigurerAdapter and which is annotated with @EnableAuthorizationServer, we have to create a @Bean method to expose a JdbcTokenStore and configure a datasource in it.

**What is a JdbcTokenStore?**

JdbcTokenStore, is a class in springframework security oauth2 package which implements the token services for storing tokens in database. Without configuring this, the tables oauth\_access\_token and oauth\_refresh\_token won’t even be touched. This is the class which inserts the tokens into these tables.

**What is the purpose of @Bean?**

@Bean annot. On the method basically means that the method produces a bean which needs to be managed by the Spring container.

**What is the purpose of @Configuration?**

And, the @Configurationn annot at class-level is basically to indicate that the class has one or more @Bean methods and they have to be managed by the spring container.

**What is the annot. You use to inform spring that the class contains one or more @Bean method and they have to be managed by spring container?**

@Configuration

**What is a PasswordEncoder?**

PasswordEncoder is basically an interface in spring security framework, which defines abstract methods to encode passwords with some hash function (which you have to tell spring which one to use or you can even build your own hashing function), to which it applies some random salt as well.

**What is a BcryptPasswordEncoder?**

And, BcryptPasswordEncoder, is an implementation of PasswordEncoder that uses Bcrypt hashing function.

**What is the hashing function prefered by spring for encoding passwords?**

Spring favours using Bcrypt hashing function for hashing purposes. It allows you to mention the no.of log rounds to be used in Bcrypt hashing, default is 10.

**what is the purpose of WebSecurityConfigurerAdapter?**

This is a convinient class that allows customization to the Web Security. This class enables you to do customizations to AuthenticationManager. We can override the methods in this class to create an AuthenticationManager and expose it as a @Bean.

**What is that you have to do/configure in a spring boot oauth2 application to customize/provide our own AuthenticationManager?**

We have to create a class and make it extend WebSecurityConfigurerAdapter class and override it’s authenticationManagerBean method and configure it @Bean to return AuthenticationManager as a bean and define the exposed bean by overridding the method which takes AuthenticationManagerBuilder to create/customize the authentication manager.

**What is an AuthenticationManagerBuilder?**

The AuthenticationManagerBuilder is used to create an AuthenticationManager

**what is bigserial in postgresql?**

BIGSERIAL in postgresql is an 8-byte unsigned auto-incrementing integer.

**What is the size of bigserial data type in postgresql?**

8-byte unsigned.

**What is the java equivalent of postgresql’s bigserial?**

Long.

**what is the difference between authority and role in terms of Spring Security and Oauth2?**

There is nothing called role when it comes to oauth2. Oauth2 only talks authorities, specifically called as granted authorities.

And, when it comes to Spring Security, we have ROLES.

**What is int8 in postgresql? What is its size? What is its java equiavalent?**

The int8 is an alias for bigint, which is an 8 byte signed integer.

Java equivalent is Long.

**What is the purpose of UserDetails?**

UserDetails is an interface and it provides core user details.

A UserServiceDetails implementation, that is, the loadUserByUsername method returns an instance of this interface and it basically contains the list of authorities that user is assigned to, and other details like username, password, whether the account is enabled, locked or account or password expired or not.

**What is the purpose of @Configuration?**

Basically means, there are one or more methods in the class which declare @Bean methods and they have to be managed by the spring container.

**In an Spring boot security oauth2 application, how do you configure the password encoder to encode/decode the password of the client?**

Create a class and make it extend AuthorizationServerConfigurerAdapter class and override the configure(AuthorizationServerSecurityConfigurer oauthServer) method to provide the encoder for the client password.

The above method is to be overridden when you are configuring security for the token endpoint i.e./ oauth/token

**In a spring boot security oauth2 application, how do you configure the security for the token endpoint?**

Create a class and make it extend AuthorizationServerConfigurerAdapter class and override the configure(AuthorizationServerSecurityConfigurer oauthServer) method to provide the encoder for the client password.

The above method is to be overridden when you are configuring security for the token endpoint i.e./ oauth/token

**why would you override configure(AuthorizationServerEndpointsConfigurer endpoints) for?**

This is overridden only in case of password grant flow otherwise out-of-the-box config is enough to ensure security. But, for password grant type flow, it is mandatory to override this method and provide an AuthenticationManager otherwise we will be getting an ‘unsupported operation – password grant type’ error.

**What do you think is the main reason when you get an error - ‘unsupported operation – password grant type’?**

This is probably because in the Auth Server Config class we did not override one of the configure methods, which takes an AuthorizationServerEndpointsConfigurer param, and provide an AuthenticationManager. In password grant flow, it is mandatory for us to override this method an provide an AuthenticationManager.

**Why would you override configure(AuthenticationManagerBuilder) for?**

To provide UserDetailsService and provide a PasswordEncoder for it. This password encoder will be used to encode user passwords.

**In which class is configure(AuthenticationManagerBuilder) for?**

This is part of WebSecurityConfigurerAdapter class.

**What is ResourceServerConfigurerAdapter? What methods does it have? When do you override them and what for?**

This is a class in springframework security oauth2 package, which implements ResourceServerConfigurer interface, and the purpose of this interface is to enable a resource server configuration.

**What configurations do you make in a resource server config class in a spring boot oauth2 application?**

We create a class and make it extend ResourceServerConfigurerAdapter, which implmenets ResourceServerConfigurer interface, which is basically a convinience interface for enabling resource server config.

The interface has two methods configure(ResourceServerSecurityConfigurer resources) and configure(HttpSecurity http).

Former is overridden to mention a resource id, this has to match with the resource\_id in the oauth\_client\_details table. And, the latter is overridden to configure access rules, that is to ensure that client with right scope perform right ops. Otherwise, a read client wlll also be able to perform a write op.

**What is the purpose of @EnableResourceServer?**

This is to enable a resource server configuration.

**What all class-level configs you do in terms of Auth Server and Resource Server in order to make a password grant flow work in a spring boot oauth2 application?**

First, we config a Auth Server by creating a class and extending AuthorizationServerConfigurerAdapter class. Now, we configure the client by informing the auth server to fetch them from the configured datasource. We do this by dependency injecting a datasource and then overridding the configure(ClientDetailsServiceConfigurer) method and then configuring the jdbc datasource in it. And, then we configure the password encoder in order to encode/decode client passwords. This we do by first dependency injecting an already created password encoder bean and then we override the configure(AuthorizationServerSecurityConfigurer) method and configure this di’d passwordencoder in it. Next, we need to define the token generation mechanism, that is, espeically about where to store the tokens, so we configure a jdbctokenstore by configuring it as a @bean. And, since this is a password grant flow, we need to provide an AuthenticationManager, for which we shall first dependency inject the AuthenticationManager and then override the configure(AuthorizationServerEndpointsConfigurer) method and provide the AuthenticationManager from here.

Now, the dependency injected authenticatiomanager here has to be define as a bean somewhere, for which, we create a class and extend WebSecurityConfigurerAdapter and override its authenticationManagerBean method and configure it as a bean. Now, since this methods returns a bean, we will have to customize that bean according to our requirements by a userdetailsservice and passwordencoder to encode user passwords, which we do by overriddin configure(AuthenticationManagerBuilder) method which allows us to create an AuthenticationManager. Before overriddding this method, we first dependency inject the userDetailsService and the already define password encoder and then we provide these configs in the overidden method. Now, the dependency injected userdetailsservice also has to be implemented, but I am not covering it in this question because this question is basically about auth and resource server.

Now, for the client to be able to access the resource id for which he has access to (according to the resource id mapped to the client in the oauth\_client\_details table), we need to create a class and extend ResourceServerConfigurerAdapter and override its configure(ResourceServerSecurityConfigurer) method to configure the resource id and override its configure(HttpSecurity) to config access controls for the client, which is to ensure that client with right scope can perform right ops i.e. a client with read scope should not be able to perform a write op.

**What is a CSRF attack?**

Cross-site request forgery, also known as one-click attack or session riding or XSRF, is a type of malicious exploit of a website where unauthorized commands are transmitted from a user that the web application trusts.

**Give an example of CSRF attack?**

As an example, my banking website, example.com, does not protect itself against CSRF. You, an unsuspecting example.com user, also happened to be logged in to example.com. Now, malicious user Mallory sends you (and millions of other example.com users, of course) an HTML e-mail including a malicious html tag, which has an instruction to transfer certain amount to Mallory, embedded in an image. If you have a webmail client that loads images automatically, the transfer request will be made from your browser using your IP address and your example.com session cookies, exactly as if you made the request yourself. My bank website, therefore, treats this like a legitimate request and sends $1000 from your account to Mallory’s account. All evidence suggests you legitimately made this transaction from your logged-in browser.

**What is an XSS attack?**

This is different from Cross-Site Scripting (XSS) attacks. XSS allows malicious users to inject client-side code (mainly Javascript) into web pages to be run by other unsuspecting users.

**What is SQL Injection attacks?**

Most modern websites are powered by a web server that communicates with a database. That database is used to store anything that’s provided or generated by the website’s users, including private information like login credentials and credit card numbers. The web server accesses this information using SQL (Structured Query Language). An SQL statement essentially asks the database to retrieve or store some information, using plain English sentences. These natural language-like statements are sent from the web server to the database as raw strings that the database driver then has to parse and turn into actionable commands. These SQL commands can then be manipulated by an attacker to replace the values in the sql string with some malicious values and this results in an SQL-INjection attack.

**How do you prevent SQL-Injection attacks?**

As web developers, we thankfully have a tool that easily and completely prevents such attacks: the parameterized statement. A parameterized statement is a SQL statement with placeholders, such as ?, that are substituted with real values during execution. the variables are sent to the database separately from the statement string and thus are never treated as commands.

**Since what version of spring security is CSRF protection enabled by default?**

Since Spring Security 4.0.

**What is CORS?**

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin.

Cross-Origin Resource Sharing (CORS) is a W3C specification implemented by most browsers that lets you specify what kind of cross-domain requests are authorized.

**What is the purpose of @EnableGlobalMethodSecurity?**

This enables enables annotation based method-level security. This is a class-level annotation. This can be applied on an @Configuration instance.

**what are the different options that you can apply with @EnableGlobalMethodSecurity?**

This annotation comes with enabling 3 kinds of method-level security annotations:

@EnableGlobalMethodSecurity(securedEnabled = true)

@EnableGlobalMethodSecurity(jsr250Enabled = true)

@EnableGlobalMethodSecurity(prePostEnabled = true)

**What does @EnableGlobalMethodSecurity(securedEnabled = true) do?**

This enables Spring’s @Secured annotation, which is used to specify a list of roles on a method. Hence, a user only can access that method if she has at least one of the specified roles.

The @Secured annotation doesn’t support Spring Expression Language (SpEL).

**What does @EnableGlobalMethodSecurity(jsr250Enabled = true) do?**

This will enable jsr-250’s @RoleAllowed annotation. This equivalent annotation of the @Secured annotation.

Basically, we can use the @RoleAllowed annotation in a similar way as @Secured

**What does @EnableGlobalMethodSecurity(prePostEnabled = true) do?**

This enables Spring Security's pre post annotations.

These annot. can be written in SpEL. These annotations are @PreAuthorize, @PostAuthorize, @PreFilter and @PostFiler.

**How do you enable @Secured annotation in spring security?**

To enable this annotation, we have to annotate any @Configuration class with @EnableGlobalMethodSecurity(securedEnabled = true).

**How do you enable @RoleAllowed annotation in spring ?**

To enable this annotation, we have to annotate any @Configuration class with @EnableGlobalMethodSecurity(jsr250Enabled = true).

**How do you enable @PreAuthorize annotation in spring ?**

To enable this annotation, we have to annotate any @Configuration class with @EnableGlobalMethodSecurity(prePostEnabled = true).

**What is the difference between @Secured and @RoleAllowed annotation?**

@Secured is a Spring Security’s annotation for implementing method-level security whereas @RoleAllowed is a jsr-250 annotation for the same. In terms of functionality, they both work in the same way, which is allow access to the methods (on which they are declared) to only those roles that are configured in these annotations.

**How are Spring Security’s PrePost annotations more powerful and flexible than @Secured and @RoleAllowed annotations?**

The PrePost annotations can be configured using SpEL and not just roles but these annotations can be configured with method parameters as well.

For e.g., we can configure a method which takes a username as method param with @PreAuthorize(“#username == authentication.princiapl.username”), which basically means this method can be accessed only by the current authenticated user.

And this flexibility makes these prepost annotations very comfortable.

**Why is an mvcMatcher more secure than an antMatcher?**

As an example antMatchers("/services") only matches the exact “/services” URL while mvcMatchers("/services") matches “/services” but also “/services/”, “/services.html” and “/services.abc”.

Thus the mvcMatcher matches more than the antMatcher and is more forgiving as far as configuration mistakes are concerned.

In addition, the mvcMatchers API uses the same matching rules as used by the @RequestMapping annotation.

**What is password hashing?**

Password hashing is the process of calculating a hash-value for a password. The hash-value is stored, for instance in a database, instead of storing the password itself. Later when a user attempts to log in, a hash-value is calculated for the password supplied by the user and compared to the stored hash-value. If the hash-values does not match, the user has not supplied the correct password.

**What in Spring Security is responsible for implementing the password hashing?**

In Spring Security, this process is referred to as password encoding and is implemented using the PasswordEncoder interface.

**What is salting?**

A salt used when calculating the hash-value for a password is a sequence of random bytes that are used in combination with the cleartext password to calculate a hash-value. The salt is stored in cleartext alongside the password hash-value and can later be used when calculating hash-values for user-supplied passwords at login.

The reason for salting is to avoid always having the same hash-value for a certain word, which would make it easier to guess passwords using a dictionary of hash-values and their corresponding passwords.

**Why do you need method security? What type of object is typically secured at the method level?**

Method level security in spring is nothing but enabling authorization at method level.

To enable support for spring method level security, use @EnableGlobalMethodSecurity in a @Configuration class.

This is basically for securing our service layer.

We could secure our service layer by, for example, restricting which roles are able to execute a particular method

**What do @PreAuthorize and @RolesAllowed do? What is the difference between them?**

The @PreAuthorize and @RolesAllowed annotations are annotations with which method security can be configured either on individual methods or on class level.

The @PreAuthorize annotation allows for specifying access constraints to a method using the Spring Expression Language (SpEL).

The @PreAuthorize annotation is part of the Spring Security framework.

The @RolesAllowed annotation is the JSR-250 equivalent of @Secured annot.

This annotation is more limited than the @PreAuthorize annotation in that it only supports role-based security.

In order to use the @RolesAllowed annotation the library containing this annotation needs to be on the classpath, as it is not part of Spring Security. In addition, the jsr250Enabled attribute of the @EnableGlobalMethodSecurity annotation need to be set to true.

**What does Spring’s @Secured do?**

The @Secured annotation is a legacy Spring Security 2 annotation that can be used to configured method security.

It does not support using Spring Expression Language (SpEL) to specify security constraints.

It is recommended to use the @PreAuthorize annotation in new applications over this annotation.

In terms of functionality, this is similar to jsr-250’s @RoleAllowed annotation and only let’s you configure role based authorization.

**In which security annotation are you allowed to use SpEL?**

@PreAuthorize, @PostAuthorize, @PreFilter, @PostFilter

**What is OAuth2?**

OAuth2 is Authorization protocol.

It allows users to share their private resources with a third-party while keeping their credentials secret. These resources could be photos, videos etc.

**How do you access restricted resources in traditional client-server model i.e. w/o oauth2?**

The clients requests access to a protected resource on the server by authenticating with the server using the resource owner's creds. For this to happen, the resource owner must share it's creds with the third-party appi i.e. the client.

**What are the Roles in OAuth2?**

There are 4 roles in OAuth2.

Resource Owner: an entity capable of granting access to protected resources which it owns, basically by issuing authorizations grants to the client. When a resource owner is a person, it is referred to as end-user.

Resource Server: the server hosting the protected resources

and capable of accepting and responding to protected resource requests using access tokens.

Client: an application making protected resurce requests on behalf of the resource owner and with its authorization.

Authorization Server: the server issuing access tokens to client after sucessfully authenticating the resource owner and obtaining authorization.

**Define OAuth2 Asbtract Protocol Flow.**

Following is the protocol flow:

a.the client request authorization from resource owner. The authorization request can be made directly to the resource owner, or preferably indirectly via the authorization server.

b.the client receives the authorization grant, which is a credential represential resource owner's authorization, expressed using one of four grant types or using an extension grant type. The grant type dependens upon the methd used by the client to request authorization and the types supported by the auth. server.

c.the client requests an access token by authenticating with the auth server and presenting the auth grant.

the auth server authenticates the client and validates the auth grant, d.if valid, issues an access token.

the client requests the protected resource from the resource server and authenticates by presenting the access token.

the resource server validates the access token, if valid, serves the request.

**What are the different OAuth2 Authorization grant types?**

Following are the grant types:

a.Auth. Code: Here the auth. code is obtained by using an auth. server as an intermediary b/w the client and the resource owner.

Instead of requesting auth. directly from the resource owner, the client directs the resource owner to an auth. server which in turn directs the resource owner back to the client w/ the auth. code but before directing the resource owner back to the client with auth. code, the auth. server authenticates the resource owner and obtains auth.

This is generally used with server-side applications.

b.Implicit:

Here instead of issuing the auth. code, the client is issued an access token directly.

c.Resource Owner password Creds (also known as Password grant type):

the resource owner pwd creds (i.e. username and pwd) can be used directly as an auth. grant to obtain an access token.

This should only be used when there is a high degree of trust b/w the resource owner and tthe client.

But even though the resource owner creds are with client, they are with the client for a single reqest and are then exchanged for an access token.

This grant eliminates the need for the client to store the creds for future use, by exchanging creds with a long-lived access token.

this is used with trusted applications, such as those owned by the service itself.

d.Client creds:

Here, the client creds are used as auth. grant typically when the client is acting on its own behalf i.e. the client is also the resource owner.

used for machine-to-machine interaction.

used with Applications API access.

**What is an OAuth2 Access Token?**

Access tokens are strings representing an auth. issued by the auth. server in exchange of the auth. grant to the client to enable the client to access protected resources.

These tokens repesent specific scopes and duration of access.

**What is the purpose of Oauth2 Access Token?**

Access tokens are strings representing an auth. issued by the auth. server in exchange of the auth. grant to the client to enable the client to access protected resources.

These tokens repesent specific scopes and duration of access.

**what is an OAuth2 Refresh Token?**

Refresh tokens are creds issued to the clients by the auth. server which are used to obtain fresh/new access tokens when the existing ones become invalid/expire.

Unlike access tokens, refresh tokens are intended for use only with the auth. server and are never sent to the resource server.

**What is the purpose of an Oauth2 Refresh Token?**

If the access token is invalid or expired, the resource server returns an invalid token error.

The client, then, can requests a fresh access token by authenticating with the auth. server by presenting the refresh token.

The auth. server authenticates the client and validates the refresh token, and if valid, issues a new access token (and, optionally, a new refresh token).

Unlike access tokens, refresh tokens are intended for use only with the auth. server and are never sent to the resource server.

**Is it mandatory for an Authorization Server to generate a Refresh Token?**

These tokens are optional and it depends upon the auth. server whether to issue to refresh tokens or not. But if they are issued, they are issued along with the access tokens itself by the auth. Server.

**In a perfect oauth2 authorization system, who generates access token, authorization grant and refresh tokens?**

Refresh tokens are creds issued to the clients by the auth. server which are used to obtain fresh/new access tokens when the existing ones become invalid/expire.

**Describe the flow for fetching a refresh token and it’s use.**

It's use is as follows:

a.the client requests an access token by authenticating with the auth. server and presenting an auth. grant.

b.the auth. server authenticates the client and validates the auth. grant, and if valid, issues an access token and a refresh token.

c.the client makes a protected resource request to the resource server by presenting an access token.

d.the resource server validates the access token, and if valid, serves the request.

e.If the access token is invalid or expired, the resource server returns an invalid token error.

f.the client requests a new access token by authenticating with the auth. server and presenting the refresh token.

g.the auth. server authenticates the client and validates the refresh token, and if valid, issues a new access token (and, optionally, a new refresh token).

**What is Oauth2 Client Identifier?**

The authorization server issues the registered client a client identifier -- a unique string representing the registration information provided by the client.

Assume this as the client username that we use in spring oauth2 flows.

**what is http basic authentication?**

In the context of an HTTP transaction, basic access authentication is a method for an HTTP user agent (e.g. a web browser) to provide a user name and password when making a request.

**In Http Basic Authentication, how do you authenticate?**

In basic HTTP authentication, to authenticate, a request is sent that contains a header field of the form “Authorization: Basic <credentials>”, where credentials is the base64 encoding of id and password joined by a single colon (:).

**The credentials that are passes as part of the request header in a http basic authentication are encrypted with what?**

They are not encrypted, they are just base 64 encoded.

**What is the purpose of http basic authentication?**

The basic purpose of this mechanism is to provide a simple, convinient way for the user-agent (i.e. a browser) to send client credentials to the server for them then to be authenticated by the server.

**Why is HTTP Basic authentication (BA) is known as the simplest technique for enforcing access controls to web resources?**

Because it does not require cookies, session identifiers, or login pages; rather, HTTP Basic authentication uses standard fields in the HTTP header, removing the need for handshakes.

**Why is HTTP Basic authentication (BA) is not known as the most secure mechanism to enforce access controls to web resources?**

The BA mechanism provides no confidentiality protection for the transmitted credentials. They are merely encoded with Base64 in transit, but not encrypted or hashed in any way. Therefore, Basic Authentication is typically used in conjunction with HTTPS to provide confidentiality.

**what are the different OAuth2 Protocol Endpoints?**

There are 2 protocol endpoints:

Auth. endpoint:

This is used by the client to obtain authorization from the resource owner. This endpoint is basically at /aouth/authorize and in all the flows that I have used I did not touch this endpoint since for password grant flow you do no use this endpoint. This is for other grant type flows.

Token endpoint:

This is used by the client for obtaining an access token by presenting it’s auth. grant or refresh token.

The token endpoint is used with every authorization grant except for the implicit grant type (since an access token is issued directly).

This endpoint is at /oauth/token and this is what I have used in all my implementations because I used password grant type flow and this the endpoint to hit for that flow.

**What is the http request type used while making an access token/ refresh token request in oauth2?**

The client MUST use the HTTP "POST" method when making access token requests.

**Where is the scope of an oauth2 access token stored?**

Actually, the scope that we are talking about does not belong to the access token as such but it belongs to the client for whom the access token has been generated. And, since this is a client specific thing, this is stored in the scope column of oauth\_client\_details table.

**What is the use of an OAuth2 Access Token Scope?**

When the client brandishes the access token to the resource server while making a protected resource request, the resource server while validating the access token also checks to see if the client has the required scope to perform the kind of action that it is requesting to perform. For example, the resource server, makes sure that a client with a read scope does not perform a write op. On the protected resource.

**Where is the scope of a client stored in oauth2 database backed token store implementation?**

In an database backed token store implementation, the scope of every client is stored in the oauth\_client\_details.

**How is the scope of a client is generated in oauth2 implementation?**

Before making an access token request, the client has to register itself with the authorization server and during this process the client scope also gets determined. The client can request for a certain list of scopes and if the authorization server deems client to be worthy of that scope will grant it or other will grant whatever scope the auth server deems the client to be fit for and that scope and other details of the client gets stored in oauth\_client\_details table on the auth server.

**What do you get as part of the response when a client makes an Oauth2 Access Token request, assuming the access token is valid?**

You get an access\_token, a token\_type, a expires\_in, a refresh\_token and scope in the response, provided the auth server found the access token to be valid/non-expired.

**What is the type of token issued by the auth server to the client when the client requests for an access token?**

It is a Bearer Token.

**What is the procedure for an oauth2 client to access an protected resource?**

The client accesses protected resources by presenting the access token to the resource server. The resource server MUST validate the access token and ensure that it has not expired and that its scope covers the requested resource.

**what is an OAuth2 Bearer Token?**

A bearer token is a type of a token issued by the auth. server to the client to access protected resources.

**In Oauth2, let’s say, there are 2 clients – A and B. And, then what happens when Client B somehow gets hold of the access token that was issued to Client A and then makes a protected resource request with that access token to access a resource which only client A have access to?**

Since the type of token issued by auth server to the client when it requests for an access token is ‘bearer’ and the main property of a bearer token is that, any client in posession of the token can use the token in the any way that any other client in possession of it can. This basically means, in our case, the Client B will be able to access all those resources that client A have access to.

**In how many ways can you transmit an OAuth2 Bearer Token?**

There are 3 methods of sending bearer tokens in the resource request to the resource server. Clients MUST NOT use more than one method to send token in each request.

3 methods are:

Authorization Request Header Field: the client uses the "Bearer" authentication scheme to transmit the access token in this method.

Form-Encoded Body Parameter: the client adds the access token to the request-body using the "access\_token" parameter but there are many pre-conditions to use this method.

URI Query Parameter: the client adds the access token to the request URI query component using the "access\_token" parameter.

**Which is a relatively safe way to transmit an oauth2 bearer token to resource server while making an protected resource request?**

The auth request header field, where you send the token using the bearer authentication scheme, as part of the header, is relatively safe considered to sending it as pasrt of the form body and much better compared to sending it as part of uri paramater.

**What is a JSON Web Token (JWT)?**

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.

**With respect to a Json Web Token, what are claims?**

Claims are the details that are shared by the client to the resource server while making a protected resource request. These claims contains details that will help the resource server validate the token and also decide if the token holder has the necessary scope to perform the requested action on the protected resource.

It can be something like the expiry duration left on token, and the scope of the token/client etc.

**What problem does a Json Web Token Solve?**

The one big advantage a Json Web Token has is that the token itself contains all the claims made by the client which are necessary for the resource server to validate the token and also to decide if the token holder has the necessary scope to perform the necessary op on the requested protected resource. And, since the resource server can find all these details in the token itself, it does not have to do any database lookups to fetch all these details.

**Why is a Json Web Token so lengthy?**

Well, it’s because it conatins all the claims to be trnasferred between parties and because of which it enables the resource server to not do any database looks and still validate the token, the token is big/lengthy.

**Is a Json Web Token encrypted? If yes, what encryption does it use?**

A Json Web Token has separate parts and each part is delimited by a period and each such part is base 64 url encoded. Not encrypted.

**Are claims, that are part of a Json Web Token, encrypted or enocded?**

JWT claims are encoded in a JSOn Web Signature (JWS) or a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or MACed or encrypted.

**What are the disadvantages of using Json Web Token isntead of a default access token in oauth2?**

a. One big disadvantage is that a JWT cannot be invalidated by an auth server once it is issued.

Since, no details of the token are stored in the auth server database, once the token is issued, it will become invalid only when it naturally expires. So, this gives less control for the auth server to invalidate such a token, if it ever wants to, before the natural expiry of the token.

b. The other disadvantage which I feel is probably that the token is too lengthy and may not be a very comfortable thing for the clients to use.

**What is an Authorization Grant in Oauth2?**

In OAuth, an authorization grant is an abstract term used to describe intermediate credentials that represent the resource owner authorization issued to a client.

**What is the use of an authorization grant in oauth2?**

An authorization grant is used by the client to obtain an access token.

A client exhcanges it’s authorization grant (issued by the resource owner) with the authorization server for an access token. This is particularly true in case of password grant type flow.

**In which particular grant type flow, an authorization grant is not required?**

In Implicit Grant Flow, an access token is issued directly to the client by the auth server with the implcit approval of resource owner without the client first having to fetch an authorization grant from the resource owner.

**Name all the key components i.e. classes/interfaces of Spring Security Architecture only with respect to Authentication?**

Principal, Authentication, UsernamePasswordUathenticationToken, AuthenticationManager, ProviderManager, AuthenticationProvider. UserDetailsService, UserDetails.

**What is a Principal in the context of security?**

A Principal in the context of security means an entity which can be authenticated by a computer system or network.

**What is the significance of an Authentication object/interface in the context of Spring Security?**

It basically represents the token for an authentication request and once the authentication is successful, the fully populated Authentication object will contain all details about the authenticated principal like the GrantedAuthorities, username, password, account validity etc.

**Which object/interface is used by Spring Security to store the details of an authenticated principal?**

Before the authentication process begins, an Authentication object (an UsernamePasswordAuthenticationToken) will contain the authentication token request i.e. the username and password passed in and once authenticated, the fully populated Authentication object will contain all details related to the authenticated principal.

**What details are stored in an Authentication instance for an authenticated principal in the context of Spring Security?**

It contains the granted authorities, the username, password and other account validity details.

**What is the main job of an AuthenticationManager in Spring Security?**

The job of this interface/object is to process an Authentication request i.e. the passed-in Authentication object and if successful, it will return a full populated Authentication object (including the granted authorities).

Hence it is called the core strategy of authentication.

**Which object/interface in Spring Security is primarily responsible for authenticating the principal that is to be authenticated?**

AuthenticationManager and that is why it is called the core strategy of authentication.

**In Spring Security, what happens when the AuthenticationManager finds the principal that it is trying to validate to be invalid?**

Will throw an AuthenticationException.

**In Spring Security, what happens after the AuthenticationManager successfully authenticates the principal that it is trying to authenticate?**

It will return a full populated Authentication object (including the granted authorities).

**In Spring Security, what all does an AuthenticationManager check while it is in the process of authenticating an principal and in what order?**

An AuthenticationManager while authenticating the passed Authentication object, it will first check if the account associated with the principal is enabled, then it will check if the account is not locked and when these checks pass, it will only then check for the credentials of the principal.

And if any of these checks fail, it will throw an AuthenticationException.

**In the context of Spring Security, what is the main job of an AuthenticationProvider?**

The main job of the AuthenticationProvider interface is to indicate if it can authenticate the passed in Authenticaton object using one of its authentication provider implementations.

**In the context of Spring Security, which is the authentication provider that handles the authentication of a database based token store and authentication mechanism?**

DaoAuthenticationProvider

**Which class in spring security implements AuthenticationManager?**

ProviderManager.

**What is the primary job of ProviderManager in spring security?**

This class implements the AuthenticationManager and it’s main job is to basically iterate the Authentication request through a list of AuthenticationProvider s. And it does so until one of the authentication providers indicate saying it has the authority to handle the authentication.

**In Spring Security, what happens when a ProviderManager finds a suitable AuthenticationProvider which can authenticate the passed-in Authentication request?**

If a match is found, then it performs the authentication by using the matched AuthenticationProvider. This class has the ability to perform the authentication because it implements AuthenticationManager.

So, in theory, it is the AuthenticationManager that performs the authentication but we can say that the main decision point is in ProviderManager class.

**What is the information stored in an UserDetails object/interface in spring security?**

This simply contains user information such as user password, username, authorities etc. Of the authenticated principal.

**What is the difference between an UserDetails and an Authentication object/interface in spring security?**

More or less both contain the same kind of information about an authenticated principal but UserDetails and it’s implementatios are not used by Spring Security in it’s internal flow. This is basically for use by clients/developers. Whereas, Spring Security uses the Authentication object for it’s purposes. That is the only major difference between these two.

**In Spring Security, in a database based token store and authentication system, how does the DaoAuthenticationProvider fetch the user details from to perform authentication?**

The DaoAuthenticationProvider uses UserDetailsService to retrieve user details from the database.

**What is the primary purpose of UserDetailsObject interface in spring security?**

It’s a core interface which loads user-specific data. This is used by the DaoAuthenticationProvider, which is the Authenticationprovider for database based authentication, to retrieve user details.

This interface has only one abstract method – loadUserByUsername and returns a UserDetails object.

**What is the interface implemented by UsernamePasswordAuthenticationToken in Spring Security?**

Authentication.

**In Spring Security, what does an instance of UsernamePasswordAuthenticationToken primarly store?**

This class implements the Authentication interface and is basically to store the username and password of the user.

**Explain Spring Security Authentication Architecture/Flow.**

The supplied username and password of the user is obtained and combined into an instance of UsernamePasswordAuthentication token, which in itself is an intsance of Authentication, and it’s instance is passed to an instance of AuthenticationManager. Then the ProviderManager, which is an instance of AuthenticationManager gets kicked in and it passes the passed in Authentication object through a list of AuthenticationProvider s. Then the AuthenticationProvider checks the Authentication objects to see if any of its implementations has the authority to authenticate the Authentication object. Once a suitable authentication provider is found, in case of database based authentication, it will be a DaoAuthenticationProvider, which will retrieve user specific details from UserDetailsService. The class that implements UserDetailsService provides the implementation as to how and from where to fetch the user details from i.e. from what tables. Once the user details are fetched, a UserDetails object will be returned and will be validated/authenticated by the AuthenticationManager or one of its implementations like ProviderManager and once authenticated, a fully populated Authentication object (also containing grabted authorities) will be returned. More or less, this is the flow of Spring Security Authentication.

**Name the key components i.e. classes/interfaces in Spring Security Architecture/flow, with respect to Authorization?**

AccessDecisionManager, AffirmativeBased, AccessDecisionVoter and WebExpressionVoter.

**what is authorization generally known as in the context of spring security?**

It is known as ‘access control’. This is to not to confuse the term authroization with authentication.

**What is the core strategy interface in spring security authorization?**

AccessDecisionManager

**Why AccessDecisionManager is called as the core strategy interface of spring secutity authorizaton?**

Because this core interface is responsible for 2 main things – one is to indicate which of it’s implementations will be able to provide access control decisions for the indicated secured object and the other is to decide whether access control i.e. authorization can be granted or not. That is why, this is the core strategy interface for spring security authroization.

**What are the two main responsibilities of AccessDecisionManager in the context of spring security?**

And, this interface has basically has 2 types of methods and hence two main responsibilities – one is to indicate which of it’s implementations will be able to provide access control decisions for the indicated secured object and the other is to decide whether access control i.e. authorization can be granted or not.

**In Spring Security, which class implements AccessDecisionManager?**

AffirmativeBased.

**What is the main job of AffirmativeBased in spring security?**

This class implements AccessDecisionManager interface and it’s job is to poll all configured AccessDecisionVoter s and grant access if any AccessDecisionVoter voted affirmatively.

**What is an AccessDecisionVoter in spring security and what does it do?**

AccessDecisionVoter interface has 2 main responsibilities – one is to indicate which of its implementations will be able to provide access control votes for the indicated secured object and the other is to vote and return whether access is ACCESS\_GRANTED, ACCESS\_DENIED or ACCESS\_ABSTAIN.

**In Spring Security, what is the difference between an AccessDecisionVoter and AccessDecisionManager?**

AccessDecisionVoter interface has 2 main responsibilities – one is to indicate which of its implementations will be able to provide access control votes for the indicated secured object and the other is to vote and return whether access is ACCESS\_GRANTED, ACCESS\_DENIED or ACCESS\_ABSTAIN.

Whereas, an AccessDecisionManager is also responsible for 2 main things – one is to indicate which of it’s implementations will be able to provide access control decisions for the indicated secured object and the other is to decide whether access control i.e. authorization can be granted or not.

AccessDecisionManager is the core strategy interface for spring security authroization and it in the process of access control calls the AccessDecisionVoter to perform voting.

**In Spring Security, which implementation of AccessDecisionVoter will handle the access control voting in a web based or http access control request?**

WebExpressionVoter

**What is a WebExpressionVoter in spring security?**

This is one of the implementations of the AccessDecisionVoter interface and this basically will vote on the access control and return a decision. This is the implementaiton of AccessDecisionVoter which will, by default, be called in case of web based/ http access control request to do the voting.

**Explain Spring Access Control / Authorizaton Architecture/Flow.**

So, as soon as the authorization part begins, one of the implementations of the AccessDecisionManager, which is the core strategy of spring authorization, which can handle the access control for the indicated secured object will be invoked, which in all my cases has been a AffirmativeBased. The job of AffirmativeBased then, is to iterate through the list of all configured AccessDecisionVoter s and see if any of it’s implementations is returning a postivie/affirmative response.

Once again, one of the implementations of AccessDecisionVoter which can handle the access control vote for the indicated secured object will be invoked, which in all my cases has been WebExpressionVoter, which handles all web based authorizations, will vote on the indicated secured object and will return a response. If the response if ACCESS\_GRANTED then access will be granted by AffirmativeBased i.e. an instance of AccessDecisionManager and hence authorization will be successful. If not, and AccessDeniedException will be thrown.

**What is a SecurityContext with respect to Spring Security?**

Spring Security is fundamentally thread bound because it needs to make the current authenticated principal available to a wide variety of downstream consumers and The basic building block is the SecurityContext which may contain an Authentication.

This is stored in SecurityContextHolder.

**What is a SecurityContextHolder with respect to Spring Security?**

This class basically conatins a SecurityContext for the current thread of execution.

You can always access and manipulate the SecurityContext via static convenience methods in SecurityContextHolder

**What is the purpose of FilterChainProxy?**

This is a class and it’s purpose is to delegate Filter requests to spring managed filter beans.

So, basically the incoming requests will be matched against a list of SecurityFilterChains and the first matched chain will be invoked.

**In Spring Security, what is the filter that is invoked by the FilterChainProxy to perform authentication for an Http Request?**

BasicAuthenticaionFilter

**What does an BasicAuthenticationFilter do?**

This filter is responsible for processing any request that has a HTTP request header of Auth orization with an authentication scheme of Basic and a Base64-encoded username:password token.

If authentication is successful, the resulting [Authentication](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/Authentication.html) object will be placed into the SecurityContextHolder.

If authentication fails, an [AuthenticationEntryPoint](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/AuthenticationEntryPoint.html) implementation is called. Usually this should be [BasicAuthenticationEntryPoint](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/web/authentication/www/BasicAuthenticationEntryPoint.html), which will prompt the user to authenticate again via BASIC authentication.

**What is the one downside of using Basic Authentication Mechanism?**

One major downside with Basic authentication is that it transmits a password in clear text and as such is undesirable in many situations.

**What is DelegatingFilterProxy?**

It’s a proxy for standard servelt filter and it’s main and only job is to delegate filter requests to one of spring-managed bean filters in the chain of filters managed by the FilterChainProxy.

**Explain the overall flow of spring security including the filter chain mechanism.**

As soon as oauth access token request comes in, the filterchainproxy gets kicked in and it delegates the request to matching filter chains for various purposes.

And similarly the BasicAuthenticationFilter is invoked to process the authentication request.

The BasicAuthenticationFilter will then fetch the AuthenticationManager by using it’s getAuthenticationManager method (not displayed in the logs) and which then invokes a ProviderManager, which then passes control to AuthenticationProvider to decide which implementation can authenticate the request and hence the DaoAuthenticationProvider gets invoked. And then the details of user will be fetched from UserDetailsService (i mean from it’s implementation, which will tell spring how and from where to fetch user tables, i.e. from which tables).

Once authenticated, the authorization process kicks-in.

So, more or less this is how the access request is processed and handled by spring and this is how the authentication and authorization are all triggered and hanlded.

**What is the starter for using Spring Security?**

**What is a spring post processor?**

**What is the purpose of @Controller annot in Spring?**

**What is the difference between @Controller and @RestController annot in spring?**