Spring Security:

JWT (Json web token):

* Authentication: Process through which a client confirms their identity. A simple example would be the use of a username and password.
* Authorization: Process through which it is determined if a client has authority or authorization to access certain protected resources.

What is JWT?

JSON Based Token (JWT https://jwt.io/) is a JSON-based open source standard for creating access tokens that allow us to secure communications between client and server.

If the username and password sent by the user is successfully authenticated, server will then be generating a JSON Web Token and returning it to the client.

**what is the difference between responsebody and responseentity**

ResponseEntity represents an HTTP response, including headers, body, and status. While @ResponseBody puts the return value into the body of the response, ResponseEntity also allows us to add headers and status code.

e.g return ResponseEntity.accepted().headers(headers).body(c);

**what is difference between @Component @Bean**

**@Component** annotation (within the given package) and create the beans of such classes and register them in the **ApplicationContext**. **@Component** is a class level annotation and its purpose it to make the class as spring managed component and auto detectable bean for classpath scanning feature.

**@Bean** is a method level annotation and it is used within a class that is annotated with **@Configuration**. Simply, **@Bean** annotation is used to register the bean returned by a method as a spring configuration bean in IOC Container.  **@Bean** is only a method level annotation and it cannot be used with classes and object declaration.

**Java based Configuration :**

The official Spring documentation refers to configuring your beans using a Java class annotated with @Configuration

and containing @Bean methods as 'Java Configuration'. This allows you to be absolutely free of all XML in your

application (at least as far as Spring goes).

**<context:annotation-config> vs <context:component-scan>**

) First big difference between both tags is that <context:annotation-config> is **used to activate applied annotations in already registered beans in application context**. Note that it simply does not matter whether bean was registered by which mechanism e.g. using <context:component-scan> or it was defined in application-context.xml file itself.

2) Second difference is driven from first difference itself. It does **register the beans in context + it also scans the annotations inside beans and activate them**. So <context:component-scan>; does what <context:annotation-config> does, but additionally it scan the packages and register the beans in application context.

**REST support in Spring3MVC:**  
***@Controller:-***  
Use the ***@Controller*** annotation to annotate the class that will be the controller in MVC and handle the HTTP request.

***@RequestMapping:-***  
Use the ***@RequestMapping*** annotation to annotate the function that should handle certain HTTP methods, URIs, or HTTP headers. This annotation is the key to the Spring REST support. You change the method parameter to handle other HTTP methods.  
For example:  
***@RequestMapping(method=RequestMethod.GET, value=”/emps”,  
headers=”Accept=application/xml, application/json”)***  
   
***@PathVariable:-***  
A path variable in the URI could be injected as a parameter using the @PathVariable annotation.  
For example:  
***@RequestMapping(method=RequestMethod.GET, value=”/emp/{id}”)  
public ModelAndView getEmployee(@PathVariable String id) { … }***

Other useful annotations  
Use ***@RequestParam*** to inject a URL parameter into the method.  
Use ***@RequestHeader*** to inject a certain HTTP header into the method.  
Use ***@RequestBody*** to inject an HTTP request body into the method.  
Use ***@ResponseBody***to return the content or object as the HTTP response body.  
Use HttpEntity to inject into the method automatically if you provide it as a parameter.  
Use ResponseEntity to return the HTTP response with your custom status or headers.  
**For example:**  
***public @ResponseBody Employee getEmployeeById(@RequestParam(“name”)  
String name, @RequestHeader(“Accept”) String accept, @RequestBody String body) {…}  
public ResponseEntity method(HttpEntity entity) {…}***

**Spring Boot:**

Dev tools: it is provided by spring boot which helps to restart the things automatically when we add anything into particular package or resource folder

What is spring boot?

1)faster

2)compact

3)less configuration

**Bean scopes:**

**1) Singleton:**

**I) It is cached in memory**

**ii) Will return shared reference to the same bean.**

**2) Prototype: creates new bean instance for each container.**

**3) request: scoped to an http web request. It is used in web app.**

**4) Session: scoped to an http web session. It is used in web app.**

**5) Global-session : Scoped to an global http web session . It is used in web app.**

**Bean life cycle:**

**1) Container started ->Bean instantiated ->dependencies injected**

**->Internal spring processing ->your custom init()-> bean is ready for use->container shutdown ->your custom destroy method**

**2) We can call custom code during bean initiation. E.g db**

**3) We can add custom code in bean destruction.**

**Spring Annotation configurations:**

1. **It automatically scan the classes from defined packages from ApplicationContext.xml file mentioned package**
2. **By default spring provides default bean as byname .**

**Spring Auto Wiring:**

1)Constructor Injection:

2) Setter injections:

Spring configuration types:

1) XML

2) Annotation based: we can use component scan

3) Java based Configrations:No XML

i)Use @Configration annotations

ii)User @ComponentSscan(“com.java”) annotation

iii)@Bean is equivalent to the <bean id > from xml

iv)@Bean: When JavaConfig encounters such a method, it will execute that method and register the return value as a bean within a BeanFactory. By default, the bean name will be the same as the method name

\*We can inject property using @PropertySource(ClassPath:Sports.properties) and inject the values we can use @Value({foo.email})

**Spring mvc:**

**@Controller : it inherits from @component**