Spring Boot:

1) For rapid development.

2) It has starter dependency: it will download with transitive dependency

3) It has autoconfigration

4)Embedded server.

5)Dev tool: automatically build and deploy it

What is parent starter and starter dependency?

Ans: if we open starter dependency again it open pom.xml which has all required dependency. Spring boot has added all dependency in one starter .

Spring Security:

1)In Lumis Project , we get Ldap token in home.html and validate it with J-check (Form based authentication) , we use filter here .

2)OAuth security: case search , we have used Oauth .

Case search is used to get all cases data and agent info. It used for producer BGA . We have used to get all the data from DB.

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.

How Spring security:

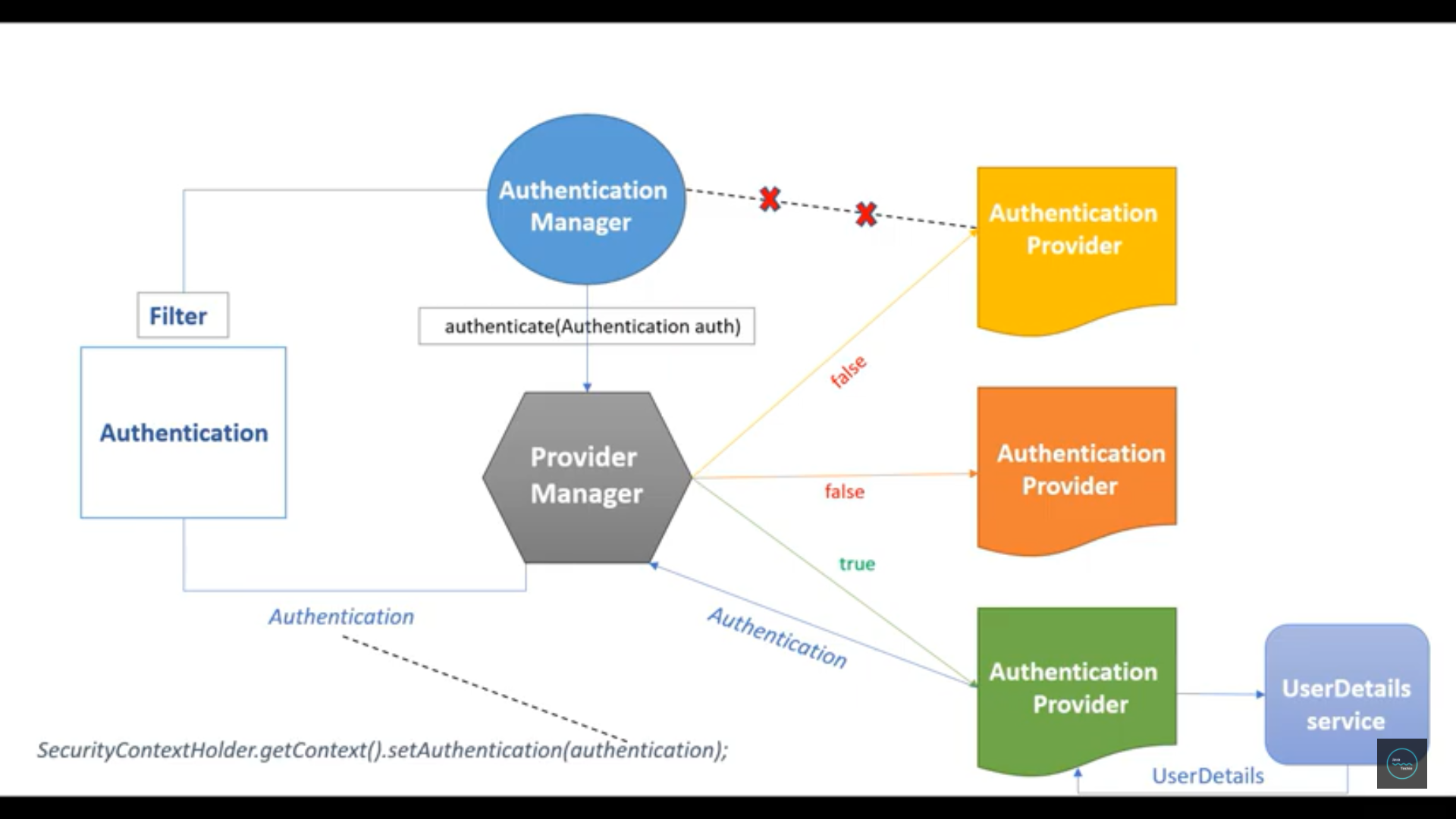
Which types of authentication provide when we add spring-boot-starter-security.

Ans:It provides basic authentication i.e form based authentication

When we enter user and password, request goes to DelegatingFilterProxy filter but filter does not know how to authenticate then filter will delegate this request to authentication manager. Now authenticate manager will take parameter as authenticate and will call authenticate () method but again authenticate manager don’t know how to authenticate. So authenticate manager will simply delegate the request to authenticate provider and it has support () e.g oAuth ,LDAP and JWT these are three types of authentication provider but authenticate manager cannot request directly to the authentication provide so authentication manager will delegate request to provider manager and provider manager will check appropriate authentication with authentication provider e.g is it JWT if support method returns true

Now how authentication provider will get to know who is user and password so appropriate authentication provider will call UserDetailService and service will return user name and password from DB to authentication provider.

Authentication provider will authenticate and will return valid authentication object to Provider Manager and Provider manager will return same authentication object to filter and filter will send this authenticate object to SecurityContext.



To use Spring security ,we need to use @EnableWenSecurity annotation and if we want to use role based security then add @PreAuthorize("hasAnyRole('admin')") annotation also we need to tell to config () to allow access when the url start with “secure/rest” and provide the role access for admin.

**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

What is spring-boot parent in pom.xml?

**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**

@RequestMapping: to map the request from client .

@RequestParam: spring will read param from request and bind it.

e.g http://localhost:8080/SpringMvcDemo/submitFormData?studentName=teju

@ModelAttribute: to bind the form data

Spring and Hibernate

1. Hibernate: the framework for persisting and saving object into DB.
2. It maps ORM mapping

What is entity: It maps the java class with db

Evict and clear() difference:

**evict():** evict() method is used to remove a specified object from the 1st level cache.

**Clear():**Clear is used to remove all the objects from 1st level cache

Save and Persist Diff:

**1)Save**: returns identifier and also return type of get is Serializable

**persist:** It does not return anything . Return type is void.

2) persist() method guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. save() method does not guarantee the same, it returns an identifier,

**Save and SaveOrUpdate():**

SaveOrUpdate: can either INSERT or UPDATE based upon the existence of a record. Clearly, saveOrUpdate is more flexible in terms of use but it involves extra processing to find out whether a record already exists in the table or not.  
  
Query :

New/Transient

commit

Save/persist

Rollback/new

Delete/remove

Persistent/Managed

Removed

Persist/rollback

Commit/ Merge

Detached

Rollback/ rollback

Close

Lazy and Eager loading:

Lazy: whenever you request, it means on demand

Eager: it always gets executed

Mapping Annotations:

1)@JoinColumn: it is used to map foreign key

2)Mapped by: when there is object ref name which your mapping into parent table with detail table.

ManyToMany: JoinTables in owner side and mappedBy in child table.

We need to create third table to map both the tables column.

Spring Security:

XMl to Java Config:

1)To initialize the spring dispatcher servlet, need to extend AbstractAnnotationConfigDispatcherServlet class and <url-pattern>/<url-pattern> , need to use getServletMapping()

@EnableMVCAnnotation: it is equivalent to <mvc:annotation>

Spring data –JPA and Hibernate :

JPA is a specification and Hibernate is a JPA provider or implementation

What is JPA:

The Java Persistence API (JPA) is a specification for mapping Java objects to database tables. Annotations like @Entity point to JPA provider (Hibernate) implementations of JPA specifications

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.

Hibernate is a JPA implementation, while Spring Data JPA is a JPA data access abstraction. Spring Data JPA cannot work without a JPA provider.

Spring Data JPA:

spring Data JPA is not an implementation or JPA provider

Spring Data JPA is an abstraction that makes it easier to work with a JPA provider. Specifically Spring Data JPA provides a set of interfaces for easily creating data access repositories like CrudRepository and JpaRepository.

Spring Data JPA adds a layer on top of JPA. That means it uses all features defined by the JPA specification, especially the entity and association mappings, the entity lifecycle management, and JPA’s query capabilities. On top of that, Spring Data JPA adds its own features like a no-code implementation of the repository pattern and the creation of database queries from method names.

Hibernate is a JPA provider and ORM that maps Java objects to relational database tables. Spring Data JPA is an abstraction that makes working with the JPA provider less verbose. Using Spring Data JPA you can eliminate a lot of the boilerplate code involved in managing a JPA provider like Hibernate.

Spring data-JPA features:

1) Removed boilerplate code.

2) Generated queries

Thymeleaf is a Java Template Engine which enables creation of web applications. It enables interaction between Java Classes and HTML/XML templates.

Thymeleaf Provides below Template Modes:

* HTML
* XML
* TEXT
* JAVASCRIPT
* CSS
* RAW

Spring Data JPA:

When we create spring mvc , we can use thymeleaf and when we use spring data JPA , we just need to extend the repository to use method like findById so it will create query on the basis passed parameter and with mentioned method name.

In Case search Project Spring Data JPA Overview:

In this project, we are finding the agent policy based on agent id or agency id or case number. So we have extended JPARepository into Repository interface which will reduce the lots of code.

Apart from this when we pull the data from rest service call , we show it into Page format e.g Page<CaseSearchView>.

The Exceptions which handled in CaseSearch Project with @ExceptionHandler : which handles multiple exception at handler level but if we want to handle exception at application level, we can do it with @ControllerAdvice and we can add multiple @ExceptionHandler for different http status code(e.g ResourceNotFoundException, Exception.class)

Spring Boot Rest Services:

1. The @RequestParam is used to extract query parameters while @PathVariable is used to extract data right from the URI.

Post and Put Difference:

If you send the same PUT request multiple times, the result will remain the same but if you send the same POST request multiple times, you will receive different results.

PUT method is idempotent whereas POST method is not idempotent.

GET and Post Diff:

1)Get method is not secured bz we send data in url and post is secured bz data is send in body.

2)GET is idempotent and post is non-idempotent

What is the diff between ResponseEntity<> and @ResponseBody in rest services:

As we know that @RestController=@ResponseBody+@Controller

ResponseEntity represents an HTTP response, including headers, body, and status. While @ResponseBody puts the return value into the body of the

The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object.response,

ResponseEntity also allows us to add headers and status code.

Spring boot –starter-json : if we return any object in rest services , it automacatlly converts into JSON . By default, Spring Boot RESTful Web service endpoints consume and produce JSON representation

ResponseEntity can return any object as JSON automatically using Jackson library internally.

e.g : **if return** **new** ResponseEntity<User>(obj, HttpStatus.***OK***);

else return **new** ResponseEntity(**new** CustomErrorType("User does not found"),HttpStatus.***NOT\_FOUND***);

How to exclude he embedded tomcat?

Ans : add this into pom.xml inside te spring-boot-starter-web dependancies

<exclusions>

<exclusion>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

Same tag need to use to exclude the dependency from pm.xml

HTTP Status:

|  |  |  |
| --- | --- | --- |
| 200 | OK | The request was successfully completed. |
| 201 | Created | A new resource was successfully created. |
| 400 | Bad Request | The request was invalid. |
| 401 | Unauthorized | The request did not include an authentication token or the authentication token was expired. |
| 403 | Forbidden | The client did not have permission to access the requested resource. |
| 404 | Not Found | The requested resource was not found. |
| 405 | Method Not Allowed | The HTTP method in the request was not supported by the resource. For example, the DELETE method cannot be used with the Agent API. |
| 500 | Internal Server Error | The request was not completed due to an internal error on the server side. |
| 503 | Service Unavailable | The server was unavailable. |

Spring Boot Actuator: module helps you monitor and manage your Spring Boot application by providing production-ready features like health check-up, auditing, metrics gathering, HTTP tracing etc.

### ****@EnableAutoConfiguration****

This annotation enables the magical auto-configuration feature of Spring Boot, which can automatically configure a lot of stuff for you.

For example, if you are writing a Spring MVC application and you have Thymeleaf JAR files on the application classpath, then Spring Boot auto-configuration can automatically configure the Thymeleaf template resolver, view resolver, and other settings automatically.

We can exclude the classes which we do not want to autoconfigure

**@ComponentScan: It will scan all base packages and will register the beans.**

[**@Configuration**](http://www.javaguides.net/2018/09/spring-configuration-annotation-with-example.html)**:** allow to register extra beans in the context

How to exclude the any class from package:

ComponentScan(basePackages = "com.logicbig.example.client;com.logicbig.example.service",

excludeFilters = @ComponentScan.Filter(

type = FilterType.ASSIGNABLE\_TYPE,

classes = {WholeSaleOrderService.class, Wholesaler.class})