SPRING BOOT INTERVIEW QUESTIONS

Q1. What is Spring Boot?

Ans: Spring Boot is used to create stand-alone, production ready Spring based applications that can just run.

Spring boot internally uses Spring framework which helps in making the application development easy and faster. It is built on top of Spring framework.

It helps in developing the microservice based application.

Q2. Advantages of Spring Boot?

Ans: The main advantages of Spring Boot are:

Auto Configuration – It helps in automatically configuring the application based on the dependencies added in the classpath. When using spring, to configure a datasource , a lot of configuration is need to configure entity manager ,transaction manager,etc. But Spring boot reduces to minimal configuration and uses the existing configuration.

Starter POMS – It consists of multiple starter POMS which are mainly used to reduce maven configuration. It helps in maintaining the POM more easily as number of dependencies are reduced.

Actuators – This helps in providing the production ready features of the application such as health check, metrics, classes loaded by the application,etc.

Rapid Application Development – Spring Boot provides the infrastructure support which is required for our application and hence the application can be developed in the quickest manner.

Embedded Servers – It comes with embedded servers such as Tomcat, Jetty etc without the need to set up an external server.

Embedded Database Integration – It also supports integration with the embedded database such as H2 database.

Q3. What is the internal working of @SpringBootApplication annotation?

Ans: It is a combination of three annotations @ComponentScan, @EnableAutoConfiguration and @Configuration.

@Configuration: It is a class level annotation which indicates that a class can be used by the Spring IOC container as a source of bean definition. The method annotated with @Bean annotation will return an object that will be registered as a Spring Bean in IOC.

@ComponentScan: It is used to scan the packages and all of its sub-packages which registers the classes as spring bean in the IOC container.

@EnableAutoConfiguration: This annotation tells how Spring should configure based on the jars in the classpath. For eg , if H2 database jars are added in classpath , it will create datasource connection with H2 database.

Q4. What is Spring Initializer.

Ans: Spring initializer is a web - based tool which is used to create a project structure for spring based applications. It does not generate any source code but helps in creating a project structure by providing maven or gradle build tool for building the application.

Q5. What is the default port number of tomcat in Spring Boot? Is it possible to change the port number?

Ans: The default port number of tomcat is 8080, yet it is possible to override it using the property server.port = port_number in application.properties or application.yml.

Q6. What are the spring boot starters?

Ans: Spring Boot provides multiple starter projects which are needed to develop different types of web application. For e.g., if we add spring-boot-starter-web as a dependency in pom.xml, we can develop mvc applications. All the dependency jars will be added to the classpath which are required for the application development using MVC.

Some of the starter POMS are:

- *Spring-boot-starter – It helps in developing stand-alone applications.*
- *Spring-boot-starter-web - It helps in designing web based and distributed applications.*
- *Spring-boot-starter-data-jpa – used in designing the persistence layer.*

It reduces the code for maven configuration.

Q7. Explain the need of dev-tools dependency.

Ans: The main aim of adding dev-tools dependency is to improve the development time. When this dependency is included in the project, it automatically restarts the server when there are any modifications made to the source code which helps in reducing the effort of a developer to manually build and restart the server.

Q8. Difference between Spring and Spring Boot.

Ans: Spring – It is an opensource J2EE framework which helps in developing web based and enterprise applications easily. Its main feature is dependency injection by which we can achieve loosely coupling while developing the application. In order to develop a web application, developer needs to write a lot of code for configuring the dispatcher servlet in web.xml, configuring the database,etc. This can be avoided while using Spring Boot.It does not support embedded servers and embedded database integration.

Spring Boot – Spring Boot is built on top of Spring framework which provides flexibility to design applications in a rapid and easier approach. It provides auto configuration feature through which it reduces the developer's effort to write large xml configuration. It provides support for embedded server without the need to install it explicitly.

Q9. How to create a spring boot project using Spring Initializer.

Ans: Spring initializer is a web-based tool developed by Pivotal. With the use of it, we can easily create the project structure needed to develop Spring based application.

- *Go to the official Spring Initialize website:* ***https://start.spring.io***

- *Select the project details such as language, spring boot version, build tool which is needed for application development as:*
    - *Language: Java*
    - *Java Version: 1.8*
    - *Spring Boot: 2.1.4*
- *Add the required dependencies and click on Generate project. It shall the download the project in your system.*
- *Import the zip file into the eclipse.*

Q10. What is component scanning?

Ans: Component scanning is the process of identifying the Spring beans in the packages and its sub packages. In a spring boot application, the packages which contains the SpringBootApplication class is called as base package and will be scanned implicitly.

@ComponentScan is used to scan the packages and detect the spring beans which will be managed by the IOC container.

If we have more than one base package, then we need to scan the base package using @ComponentScan in Spring start class.

Syntax:

@ComponentScan(basePackages = "in.ashokit.service")

Q11. What is a Spring Bean?

Ans: A java class which is managed by the IOC container is called as Spring Bean.The life cycle of the spring bean are taken care by the IOC container.

A spring bean can be represented by using the below annotations.

- *@Component*
- *@Service*
- *@Repository*
- *@Configuration*
- *@Bean*
- *@Controller*
- *@RestController*

Q12. What is the use of @Configuration annotation?

Ans: A class which is used to provide few configurations such as Swagger configuration, Kafka configuration,etc can be represented using @Configuration annotation. This class contains Bean methods to customize the object creation and returns the object which can be respresented as a Spring Bean by the IOC container.

```java
@Configuration
public class AppConfig {

    public AppConfig() {
        System.out.println("AppConfig Constructor");
    }

    @Bean
    public PwdUtils getInstance(){
        System.out.println("Get Instance Method called");
        PwdUtils pwdUtils = new PwdUtils("SHA-1");
        return pwdUtils;
    }

}
```

Q13. What is Auto-wiring?

Ans: Autowiring is the process of injecting one class object into another class. It cannot be implied on primitive types and String type.

Autowiring can be done in 3 ways:

- *Constructor Injection*
- *Setter Injection*
- *Field Injection*

Q14. What is a Runner and its use?

Ans: Runner classes are used to execute the piece of code as soon as the application starts. The code inside the runner classes will execute once on bootstrap of the application. There are mainly used to setup a data source, load the data into cache, etc. These runners will be called from SpringApplication.run() method.

There are two types of Runner Interfaces.

- *ApplicationRunner*
- *CommandLineRunner*

Q15. What is ApplicationRunner in SpringBoot?

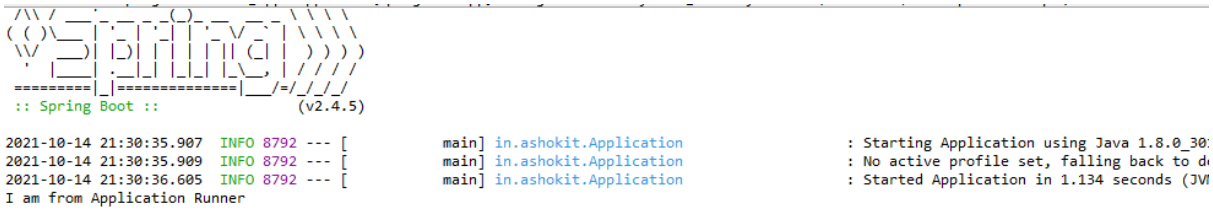
Ans: Application Runner is a functional interface which contains only one abstract method run().

When there is a need to execute some piece of code during the bootstrap of the spring boot application, then we need to write a Runner class to override the run method and provide the implementation.

Example:

```java
@Component
public class AppRunner  implements ApplicationRunner{

    @Override
    public void run(ApplicationArguments args) throws Exception {
        System.out.println("I am from Application Runner");

    }

}
```

Output:



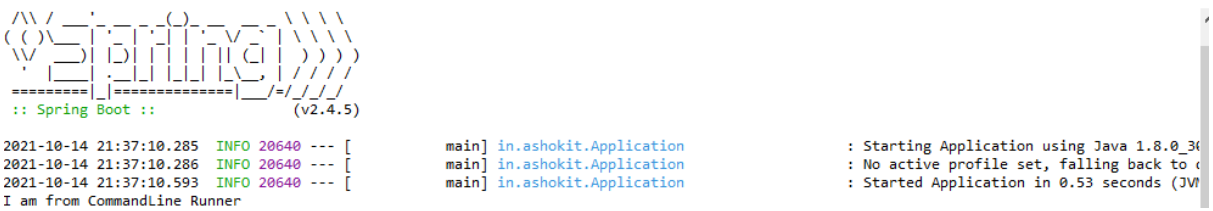
Q16. What is CommandLineRunner in SpringBoot?

Ans: CommandLineRunner is similar to the ApplicationRunner interface which is also used to execute the logic only once during application startup. The only difference between CommandLineRunner and ApplicationRunner is that ApplicationRunner accepts the arguments in the form of ApplicationArguments where as CommandLineRunner accepts in String[] array.

Example: A class implementing CommandLineRunner interface.

```java
import org.springframework.boot.CommandLineRunner;

@Component
public class CmdRunner implements CommandLineRunner {

    @Override
    public void run(String... args) throws Exception {
        System.out.println("I am from CommandLine Runner");

    }

}
```

Output :



Q17. Explain Constructor Injection.

Ans: Constructor Injection is the process of injecting the dependent bean object into the target bean using the target class construction.

E.g. : If a class Car is dependent on the Engine object which is needed for Car to run, in this case Engine object will be created first and dependency will be injected into Car class. If the dependency is achieved using the target class constructor, it is referred to as Constructor injection.

```
@Component
public class Car {

    private Engine engine;

    public Car(Engine engine) { //engine object is injected into Car class constructc
        this.engine = engine;
    }

    public Engine getEngine() {
        return engine;
    }

    public void setEngine(Engine engine) {
        this.engine = engine;
    }

}
```

```
import org.springframework.stereotype.Component;

@Component
public class Engine {

    private String engineName;

    private String mileage;

}
```

It is not mandatory to give @Autowired annotation if there is only one constructor.

Q18.Explain Setter Injection.

Ans: Setter injection is another mechanism to perform dependency injection. In this approach, the dependent object is injected into target class using target class setter methods. Setter injection can override the constructor injection.

@Autowired annotation is used on the setter methods.

Eg:

```java
@Component
public class Pizza {

    private String pizzaType;

    private boolean isVeg;

    private String size;

    private Topping topping;

    public Topping getTopping() {
        return topping;
    }

    @Autowired
    public void setTopping(Topping topping) {
        this.topping = topping;
    }

}
```

```java
@Component
public class Topping {


    private boolean isCheeseBurst;

    private boolean isPanCrust;

}
```

In setter injection, target class object should be created first followed by dependent object.

Q19.Explain Field Injection.

Ans: Field injection is a mechanism where the dependent object is injected into the target object using target class variable directly with the use of @Autowired annotation. It internally uses Reflection API to perform field injection

Eg:

```java
@Component
public class Icecream {

    @Autowired
    private Topping topping;  //field injection

    private String flavor;

    private String size;

}
```

Q20. Can you override the default web server in Spring Boot.

Ans: By default, Spring Boot provides Tomcat as the embedded server. This can be changed, as we can configure Jetty, Netty as embedded servers in Spring boot. This can be done in a convenient way by adding the starter dependencies in the maven pom.xml.

Example: Adding Jetty as dependency to pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

SPRING - JPA

Q21. What is Spring Data JPA?

Ans: Spring data JPA is used for designing the persistence layer of the web application. It is used for managing the relational data in a java application. It acts as an intermediate between the java object and relational database.

Spring data JPA is mainly built on top of JDBC API and it helps in reducing the boilerplate code.

Spring boot provides a starter-POM "spring-boot-starter-data-jpa" which is used to design the DAO layer. All the required jars are added to the classpath after adding the starter pom in dependency management configuration file.

It provides predefined interfaces which has methods to perform CRUD operations.

Q22. Explain features of Spring Data JPA?

Ans: The main advantages of using Spring Data are:

No-code repositories: *Spring data provides predefined repository interfaces which should be extended to create a repository for the entity class. It has the built-in methods to perform the CRUD operation.*

Reduces Boilerplate code: *It reduces a lot of boiler plate such as creating a connection object, creating a statement and executing the query,closing the resources,etc.*

Spring data provides predefined methods which are already implemented in the Repository interfaces, and by just calling those methods, we can perform CRUD operations.

Generation of the queries: *Another feature is queries are automatically generated based on the method names.*

Eg: If there is a method in EmployeeRepository as:

public List<Employee> findByName(String empName);

Spring data jpa will create a query as below :

select e.empid,e.empname,e.esalary from employee e where e.name = ? ;

Pagination and Sorting support: *It supports pagination and sorting using predefined interface PagingAndSortingRepository.*

Q23. How to create a custom Repository class in Spring JPA?

Ans: We can create custom repository by extending one of the interfaces as below:

- *CrudRepository*
- *JpaRepository*
- *PagingAndSortingRepository*

```
import java.io.Serializable;

import org.springframework.data.jpa.repository.JpaRepository;

import in.ashokit.entities.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Serializable> {

}
```

Q24. Difference between CRUDRepository and JPARepository.

Ans: CrudRepository interface provides method to perform only crud operations. It allows to create , read, update and delete records without creating own methods.

JPARepository extends PagingAndSortingRepository which provides methods to retrieve records using pagination and also to sort the records.

PagingAndSortingRepository extends CrudRepository which allows to do CRUD operations.

Q25. Write a custom query in Spring JPA?

Ans: A custom query can be written using @Query annotation in the Repository interface. Using this annotation, we can write HQL queries and native SQL queries.

HQL queries can be written as below to fetch Emp Salary based on name.

Eg :

```
@Query("select empSal from Employee where empName =:names")
public Double getEmpSalByName(String names);
```

Example for a native sql query is,

```
@Query(value = "select count(*) from emp_tbl",nativeQuery = true)
public Integer getCount();
```

Q26. What is the purpose of save () method in CrudRepository.

Ans: An entity can be saved into the database using save () method of CrudRepository. It will persist or merge the entity by using JPA Entity Manager. If the primary id is empty, it will call entityManager.persist(…) method, else it will merge the existing record by making a call to entityManager.merge(…) method.

Q27. Difference between findById() and getOne().

Ans: The findById() method is available in CrudRepository while getOne() is available in JpaRepository.

The findById() returns null if record does not exist while the getOne() will throw an exception called EntityNotFoundException.

getOne() is a lazy operation where it will return a proxy without even hitting the database.

findById() – will retrieve the row record by directly hitting the database.

Q28. Use of @Temporal annotation.

Ans: Prior to Java 8, @Temporal annotation is mainly used to convert the date and time values of an object to the compatible database type.

Generally, when we declare a Date field in the class and try to store it.It will store as TIMESTAMP in the database.

Eg:

```
@Temporal
private Date DOJ;
```

Above code will store value looks like 08-07-17 04:33:35.52000000 PM.

We can use TemporalType to DATE if the requirement is to store only date.

```
@Temporal (TemporalType.DATE)
private Date DOJ;
```

But after Java 8, there is no need to use @Temporal due to the introduction of LocalDate and LocalTime Api.

Q29. Write a query method for sorting in spring data jpa.

Ans: Spring data JPA provides two ways to sort the records in ascending and descending manner.

Approach 1: Using OrderBy method

E.g.: If there is an Entity class as:

```java
@Data
@Entity
@Table(name = "EMP_TBL")
public class Employee {

    @Id
    @Column(name = "EMP_ID")
    private Integer empId;

    @Column(name = "EMP_NAME")
    private String empName;

    @Column(name = "EMP_SAL")
    private Double empSal;


}
```

And the requirement is to fetch all the employees and sort by name in ascending order, then write a custom method as:

```java
public interface EmployeeRepository extends JpaRepository<Employee, Serializable> {

    List<Employee> findAllByOrderByEmpNameAsc();

}
```

Approach 2: Using Sort.by method

To sort the same above requirement, we can write the method as below

List<Employee> employees = empRepo.findAll(Sort.by(Sort.Direction.ASC, "empName"));

Q30. Explain @Transactional annotation in Spring.

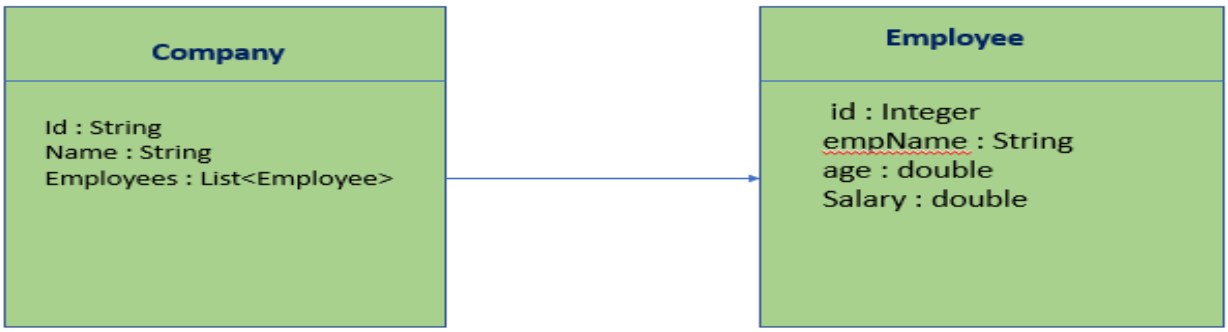
Ans: A database transaction is a sequence of statements/actions which are treated as a single unit of work. These operations should execute completely without any exception or should show no changes at all. The method on which the @Transactional annotation is declared, should execute the statements sequentially and if any error occurs, the transaction should be rolled back to its previous state. If there is no error, all the operations need to be committed to the database. By using @Transactional, we can comply with ACID principles.

E.g.: If in a transaction, we are saving entity1, entity2 and entity3 and if any exception occurs while saving entity3, then as enitiy1 and entity2 comes in same transaction so entity1 and entity2 should be rolledback with entity3.

A transaction is mainly implied on non-select operations (INSERT/UPDATE/DELETE).

Q31. What is the difference between FetchType.Eager and FetchType.Lazy?

Ans: If there exists a relationship between two entity classes, in this case for eg: Company Entity and an Employee entity as shown in the diagram.



In the above diagram, there exists a one-to-many relationship between Company Entity and Employee Entity.

When you are trying to load the company details, it will load the id, name columns, etc. But it will not load the employee details. Employee details can be loaded in two ways.

FetchType.LAZY – It will not load the Employee details while firing the query to get Company Data. This is called as lazy loading where in the employee details will be loaded on demand.

FetchType.EAGER – This will load all the employee details while loading the Company data.

Q32. Use of @Id annotation.

Ans: @Id annotation is used on a field of an Entity class to mark the property/column as a primary key in the database. It is used along with @GeneratedValue which is used to generate the unique primary keys.

Q33. How will you create a composite primary key in Spring JPA.

Ans: A composite primary is a combination of two or more primary keys in a database table.We can create composite primary keys in 2 ways in spring data jpa.

1. *Using @IdClass annotation – Suppose there is a CustomerAccount class which has two primary keys(account Id, account type)*
   Then we need to create an AccountPK class which must be public and should implements the serializable interface.
   Example:

```
@Data
public class AccountPK implements Serializable {

    private Integer accId;
    private String accType;

}
```

   Associate this class with the CustomerAccount Entity. In order to do that, we need to annotate the entity with the @IdClass annotation. Also the declare the primary key columns in entity with @Id annotation.

```java
@Entity
@Table(name = "BANK_ACCOUNTS")
@Data
@IdClass(AccountPK.class)
public class CustomerAccount {

    @Id
    private Integer accId;
    @Id
    private String accType;

    @Column(name = "BRANCH_NAME")
    private String branchName;

    @Column(name = "MIN_BALANCE")
    private Double minBalance;

}
```

2. *Using @EmbeddedId annotation –*
   Create a class which implements Serializable interface which contains all primary keys in it. Annotate the class with @Embeddable annotation.

```java
@Data
@Embeddable
public class AccountPK  implements Serializable{

    private Integer accId;
    private String accType;
    private String holderName;


}
```

Then embed this class in entity class CustomerAccount using @EmbeddedId annotation.

```java
@Entity
@Table(name = "BANK_ACCOUNTS")
@Data
public class CustomerAccount {

    @Column(name="BRANCH_NAME")
    private String branchName;

    @Column(name = "MIN_BALANCE")
    private Double minBalance;

    @EmbeddedId
    private AccountPK accPk;

}
```

Q34. What is the use of @EnableJpaRepositories method?

Ans: If the repositories classes belong to the sub package of the Spring Boot Main class, then @SpringBootApplication is enough as it scans the package using @EnableAutoConfiguration.

If the repository classes are not part of the sub package of the Main class, in that case, it will not scan the repository classes, we need to use @EnableJpaRepositories. This needs to be provided in Configuration class or SpringBootApplication class.

Q35. What are the rules to follow to declare custom methods in Repository.

Ans: We need to follow certain rules to declare custom methods to retrieve the data as below.

The fetch methods should start with findByXXXXX followed by property name.

Eg:

If you want to retrieve list of employees based on name, then write the custom method in Repository as:

```
public interface EmpRepository extends CrudRepository<Employee, Integer> {

    // abstract method
    public List<Employee> findByEmpName(String name);

}
```

Here the property name in entity class is empName which should be in camel case while appending to the findBy method.

Q36. Explain QueryByExample in spring data jpa.

Ans: It is another way to pass the search criteria in the where clause where the requirement is to retrieve the data based on multiple conditions.

It allows us to generate the queries based on Example instance.

Example instance is created as

Example<Employee> empExample = Example.of(emp);

Where emp obj holds the search criteria.

Eg: Search for an employee whose empId is 102, name is Swathi and salary is 14000.

```java
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        ConfigurableApplicationContext context = SpringApplication.run(Application.class, args);

        EmployeeRepository empRepository = context.getBean(EmployeeRepository.class);

        Employee emp = new Employee();

        //if emp id selected
        emp.setEmpId(102);

        // if emp name selected
        emp.setEmpName("Swathi");

        emp.setEmpSal(14000.00);

        Example<Employee> empExample = Example.of(emp);

        List<Employee> findAll = empRepository.findAll(empExample);

        for(Employee e : findAll) {
            System.out.println(e);
        }
    }

}
```

Q37. What is pagination and how to implement pagination in spring data?

Ans: It is the process of displaying the records in small chunks into multiple pages. Eg: in an ecommerce application, there are several products available, but all of them will not be loaded on first page, if the client clicks on second page, few of them will be loaded and so on. This is mainly to avoid the overload on the application.

Pagination contains two fields – pageSize and pageNumber.

It can be implemented using PagingAndSortingRepository which provides methods to retrieve data using pagination.

***Page findAll(Pageable pageable)** – it returns the n records based on the pageSize to be displayed on each page.*

To apply pagination on the records fetched from database, we need to create Pageable object as :

PageRequest pageReq = PageRequest.of(pgNo,pageSize);

And then pass to the find method as:

Page<Employee> pageData = repository.findAll(pageReq);

Q38. Explain few CrudRepository methods.

Ans: Some of the methods to perform DML operations are :

***findById** – to retrieve record based on the primary key.*

***findAll** – to retrieve all records from the database.*

***existsById** – to check if the record exists by passing primary key*

***count** – to check the total number of records.*

Save *– to insert a record into the database*

deleteById *– to delete a record using primaryKey*

deleteAll *– to delete all records from the table*

Q39. Difference between delete () and deleteInBatch() methods.

Ans: delete() – It is used to delete a single record at a time. It internally uses remove method of entitymanager.

deleteInBatch() – it can delete multiple records at a time, it internally calls executeUpdate() method.It is much faster than delete method.

Q40. What is the use of @Modifying annotation?

Ans: It indicates that a query method should be considered as a modifying query. It can be implied only on non-select queries (INSERT, UPDATE, DELETE). This annotation can be used only on the query methods which are defined by @Query annotation.

SPRING MVC

Q41. What is Spring MVC?

Ans: Spring MVC is one of the modules in Spring framework which helps in building web and distributed applications. It supports two design patterns

a. *MVC Design Pattern*
b. *Front Controller Design Pattern*

MVC stands for Model, View and Controller.

The major role is played by DispatcherServlet in Spring MVC which acts as a front controller which receives the incoming request and maps it to the right resource.

The main advantage of Spring MVC is that it helps in the separation of the presentation and business layer.

The components of Spring MVC are:

Model *– A model represents the data which can be an object or a group of objects.*

View *– A view represents an UI to display the data. It can be a JSP, or a Thymeleaf page.*

Controller *– It acts as an intermediate between model and view components and is responsible to handle the incoming requests.*

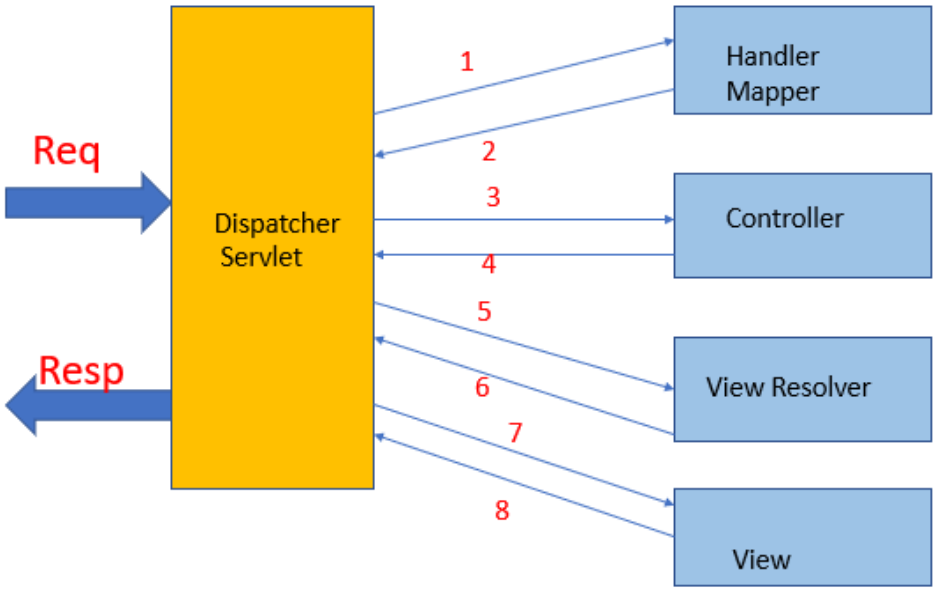
Front Controller *– Dispatcher servlet serves the main purpose of redirecting the request to the respective controller methods.*

Q42. Explain the flow of Spring MVC.

Ans: When a client request comes in, it is intercepted by the Dispatcher Servlet which acts as a Front Controller. The dispatcher servlet is responsible for pre – processing of the request and calls the handler methods to decide which controller should handle the request. It uses BeanNameUrlHandlerMapping and SimpleUrlHandlerMapping to map the request to the

corresponding controller method. The controller then processes the request and sends the response as ModelAndView back to the DispatcherServlet. Model represents the data to be displayed and view represents the component in which data should be rendered on the browser.

Front Controller is also responsible in manipulating the response data(post-processing) before sending back to client.



Q43. What is Dispatcher Servlet.

Ans: Dispatcher Servlet acts as a central servlet which handles all the incoming HTTP Requests and Responses. Once a client request is sent, it is received by the DispatcherServlet and it forwards the request to handler mapper to identify the corresponding controller class to handle the request. The controller performs all the business logic and hands over the response back to DispatcherServlet. The servlet then prepares the view component by looking for the view resolver in properties file and sends the data to be rendered on view page.

Q44. What is the use of @Controller annotation.

Ans: A class can be represented as a Controller class which is used to handle one or more HTTP requests. It is represented as a controller class using @Controller annotation. It is one of the stereotype annotations.

```
@Controller
public class HomeController {

    @GetMapping("/home")
    public String homePage() {
        return "home";
    }

}
```

*In the example above, whenever a client sends a request with the url as **localhost:8090/home**, the homepage method is invoked and view is returned from the method.*

Q45. What is the use of InternalResourceViewResolver?

Ans: InternalResourceViewResolver is the implementation of View Resolver interface which is used to resolve logical view names returned by the controller to a physical location where file actually exists. It is also a subclass of UrlBasedViewResolver which uses "prefix" and "suffix" to convert the logical view into physical view.

```
@Controller
public class HomeController {

    @GetMapping("/home")
    public String homePage() {
        return "home";
    }

}
```

For example, if a user tries to access /home URL and HomeController returns "home" then DispatcherServlet will check with InternalResourceViewResolver and it will use prefix and suffix to find the actual physical view.

Iif prefix is "/WEB-INF/views/" and suffix is ".jsp" then "home" will be resolved to "/WEB-INF/views/home.jsp" by InternalResourceViewResolver.

Q46. Difference between @RequestParam and @PathVariable.

Ans: @RequestParam is used to access the parameter values which are passed as part of request URL.

URL: ***http://localhost:8090/fee?cname=SBMS&tname=Savitha***

In the above example, we can access the parameter values of courseName and trainerName using @RequestParam annotation. In case of @RequestParam, if the parameter value is empty , it can take default value using attribute defaultValue=XXXXX

```
@GetMapping(value = "/fee")
public String getCourseFee(@RequestParam("cname")  String courseName, @RequestParam("tname") String trainerName) {
    String msg = courseName + " By " + trainerName + " is 5000 INR only";

    return msg;
}
```

@PathVariable – It is used to extract data from the request URI. Eg : if the URL is as : ***http://localhost:8090/carPrice/{carName}*** *– the value for the placeholder {carName} can be accessed using @PathVariable annotation. In order to access the carName, we need to write code as below:*

```
@GetMapping("/carPrice/{carName}")
@ResponseBody
public String getCarPrice(@PathVariable("carName") String carName) {
    String msg = carName + " Price is 7.8 lakhs";
    return msg;

}
```

Q47. Explain @Service and @Repository annotations.

Ans: A class which is annotated with @Repository annotation is where the data is stored. It is a stereotype annotation for the persistence layer.

@Service – It indicates that a java class contains the business logic. It is also a part of @Component stereotype annotation.

Q48. What is the purpose of @Model Attribute?

Ans: It is part of Spring MVC module and can be used in two scenarios:

@ModelAttribute at method level: *When used at method level, it indicates that a method will return one or more model attributes.*

```
@ModelAttribute
public void addAttributes(Model model) {
    Employee emp = new Employee();
    emp.setEmpId(101);
    emp.setEmpName("Raju");
    emp.setEmpSal(40000.00);
    model.addAttribute("employee", emp);
}
```

@ModelAttribute at method argument: *When it is used at method argument, it indicates that the argument should be retrieved from the form data. It binds the form data to the bean object.*

```
@PostMapping("/saveEmp")
public String save(@ModelAttribute("employee") Employee emp, Model model) {
    model.addAttribute("employee", emp);
    return "success";
}
```

Q49. Explain @RequestBody annotation.

Ans: This annotation indicates that the method parameter should be bound to the body of the HTTP request.

Eg:

```
@PostMapping(value = "/saveUser",consumes = {"application/json","application/xml"},produces = "text/plain")
public String addUser(@RequestBody User user) {
    System.out.println(user);
    String msg = "User Saved Successfully";

    return msg;
}
```

Q50. What is the use of Binding Result.

Ans: BindingResult holds the result of the validation and binding and contains errors that have occurred. The BindingResult is a spring's object which must come right after the model object that is validated or else Spring will fail to validate the object and throw an exception.

```java
@PostMapping("/saveUser")
public String saveUser(@Valid User usr,BindingResult result, Model model) {

    if(result.hasErrors()) {
        return "index";
    }
    System.out.println(usr);
    model.addAttribute("msg", "User Saved Successfully");
    return "dashboard";
}
```

Q51. How does Spring MVC support for validation.

Ans: It is used to restrict the user input provided by the user. Spring provides the validation API where the BindingResult class is used to capture the errors raised while validating the form.

We need to add spring-boot-starter-validation in pom.xml.

Q52. Explain @GetMapping and @PostMapping.

Ans: @GetMapping is an alternative for @RequestMapping (method = RequestMethod.GET)

It handles the HTTP Get methods matching with the given URI.

```java
@GetMapping("/course")
public String getCourseDetails(String cname,String trainer) {

    if(cname.equals("SBMS")) {
        String msg = cname + " By " + trainer + " Starting from 23-Jun-2021";
        return msg;
    }else if(cname.equals("JRTP")) {
        String msg = cname + " By " + trainer + " Starting from 30-Jun-2021";
        return msg;
    }
    return "Contact Admin";

}
```

@ PostMapping is an alternative for @RequestMapping (method = RequestMethod.POST)

It handles the HTTP Post methods matching with the given URI.

```java
@PostMapping("/saveBook")
public String handleSaveBtn(Book book, Model model) {
    System.out.println(book);

    return "bookDtls";
}
```

Q53. How to send the data from Controller to UI.

Ans: In spring mvc, controller is responsible to send the data to UI. We have Model object and ModelAndView to send the data from controller to UI.

The data in the model object is represented in key-value format as below.

model.addAttribute("key","value");

```java
@GetMapping("/greet")
public String getWishMsg(Model model) {

    model.addAttribute("msg", "God Bless You");
    return "index";
}
```

Q54. What is the purpose of query parameter?

Ans: QueryParameters are used to send the data from UI to Controller.The query parameters are passed in the URL and starts with ?.

Multiple query parameters will be represented with ampersand operator (&).

Query Parameters are appended at the end of the URL and this can be retrieved from the url using @RequestParam annotation.

Eg: http://localhost:8090/studentapp?sid=100&sname=Raju

Q55. Describe the annotations to validate the form data.

Ans: Some annotations to validate the form data.

@NotNull – It is used to check if the value entered is not null.

@NotEmpty – It checks whether the annotated element is not null nor empty.

@Email – It checks whether the given value is a valid email address.

@Size – It is used to determine that size of the value must be equal to the mentioned size.

@Null – It checks that the value is null.

Q56. What do you know about Thymeleaf?

Ans: Thymeleaf is used to design the presentation logic. It is a java template engine that helps in processing and creating HTML,javascript,etc.

It reads the template file and parses it and produces web content directly on the browser.

It helps in developing dynamic web content.

In spring mvc, we need to add the dependency as

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Q57. Explain the use of @ResponseBody annotation.

Ans: @ResponseBody on a method indicates that the return type should be directly written to the response object.

```java
@GetMapping("/plandata")
@ResponseBody
public InsurancePlan getPlanData() {
    InsurancePlan plan = new InsurancePlan();
    plan.setPlanId(101);
    plan.setPlanName("Jeevan Anand");
    plan.setPlanStatus("Approved");

    return plan; //method return type is object
}
```

Q58. What is the role of Handler Mapper in Spring MVC.

Ans: Handler mapper is used to map the incoming request to the respective controller method.DispatcherServlet forwards the request received to handler mapper.By default ,It uses BeanNameUrlHandlerMapping and DefaultAnnotationHandlerMapping to map the request to the controller.

Q59. How will you map the incoming request to a Controller method.

Ans: When a client request is received by the dispatcher servlet with the URI , for example as ***http://localhost:8090/home*** *, the central servlet forwards the request to handler mapper which checks for the controller method which matches the url pattern, in this case /home and returns the name of the controller. The front controller then sends the request to the appropriate Controller which processes the business logic and sends back the response to the client.*

Q60. How to bind the form data to Model Object in Spring MVC.

Ans: In order to create a form in spring, we need to use <form:form> tag. Eg: to store the form data into model object.

Create a POJO class:

```java
@Data
public class Product {

    private Integer productId;

    private String productName;

    private Double productPrice;


}
```

Create a product.jsp which contains the form fields using spring mvc form tag library.

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <h3>Product Form</h3>
    <form:form action="saveProduct" modelAttribute="product" method="POST">
        <table>
        <tr>
            <td>ProductId:</td>
            <td><form:input path="productId"/></td>
        </tr>
        <tr>
            <td>Product Name :</td>
            <td><form:input path="productName"/></td>

        </tr>
        <tr>

            <td>Product Price</td>
            <td><form:input path="productPrice"/></td>
        </tr>
        <tr>
            <td>        </td>
            <td><input type="submit" value="Save"></td>
        </tr>
        </table>
    </form:form>
</body>
</html>
```

In order to bind the form data , we have an attribute called as "modelAttribute", which specifies the name of the bean class to which form data should be binded.

The form data can be retrieved in the controller class using @ModelAttribute annotation as below. The modelAttribute object specified in the JSP should match with the one in controller class, else form binding won't work.

```java
@PostMapping("/saveProduct")
public String handleSaveBtnClick(@ModelAttribute Product product,Model model) {
    System.out.println(product);

    model.addAttribute("msg", "Product saved successfully");

    return "dashboard";

}
```

Q61. What is Spring MVC Tag library.

Ans: Spring provides a tag library which is used in creating view component. It provides tags to create HTML fields, error messages, etc. It is a predefined library which can be used in JSP by using the tag as:

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
```

After adding the above tag, we can create HTML form input text by using prefix as "form".

```
<td><form:input path="productId"/></td>
```

Q62. Difference between @Controller and @RestController annotations.

Ans: A class which is annotated with @Controller indicates that it is a controller class which is responsible to handle the requests and forwards the request to perform business logic. It returns a view which is then resolved by ViewResolver after performing business operations.

@RestController is used in REST Webservices and is combination of @Controller and @ResponseBody.It returns the object data directly to HTTP Response as a JSON or XML.

Q63. Explain few tags in Spring MVC tag library.

Ans: Few of the spring mvc tags are :

<form:form> - It is used to create a HTML form.

<form:input> - It is used to create input text field.

<form:radiobutton> - It is used to create a radio button.

<form:select> - It is used to create a dropdown list.

<form:hidden> - It is used to create a hidden field.

<form:checkbox> - It is used to create a checkbox.

<form:option> - It is used to create a single Html option inside select tag.

Q64. Explain the use of @ResponseEntity annotation.

Ans: @ResponseEntity is used to represent the entire HTTP response such as status code, headers and response body.

We can return the entire response from the endpoint. When using @ResponseBody , it returns the value into the body of the http response.

Example:

```
@GetMapping("/welcome")
public ResponseEntity<String> getWelcomeMsg(){
    String msg = "Welcome To Ashok it";

    return new ResponseEntity<String>(msg, HttpStatus.OK);
}
```

Q65. How to handle exceptions in Spring MVC.

Ans: In Spring Boot, exceptions can be handled using below two annotations:

- *@ExceptionHandler -specific to a controller class*
- *@ControllerAdvice – common to all controllers.*

Q66. Explain @ControllerAdvice in Spring Boot.

Ans: @ControllerAdvice is a specialization of @Component annotation which is used to handle the exceptions across the whole application by providing a global code which can be applied to multiple controllers.

Spring Actuator

Q67. What is Spring actuator and its advantages.

Ans: An actuator is mainly used to provide the production ready features of an application. It helps to monitor and manage our application. It provides various features such as healthcheck, auditing, beans loaded into the application,etc.

Q68. How will you enable actuator in spring boot application.

Ans: An actuator can be enabled by adding the starter pom into the pom.xml.

```
<dependency>
<groupId> org.springframework.boot</groupId>
<artifactId> spring-boot-starter-actuator </artifactId>
</dependency>
```

Q69. What are the actuator endpoints which are needed to monitor the application.

Ans: Actuators provide below pre-defined endpoints to monitor our application.

- *Health*
- *Info*
- *Beans*
- *Mappings*
- *Configprops*
- *Httptrace*
- *Heapdump*
- *Threaddump*
- *Shutdown*

Q70. How to get the list of beans available in spring application.

Ans: Spring Boot Actuator provides an endpoint url /beans to load all the spring beans of the application.

Q71. How to enable all endpoints in actuator?

Ans: In order to expose all endpoints of actuator, we need to configure it in application properties/yml file as:

```yaml
management:
  endpoints:
    web:
      exposure:
        include: '*'
```

Q72. What is a shutdown in the actuator?

Ans: A shutdown is an endpoint that helps application to shut down properly. This feature is not enabled by default. We can enable it by giving the below command in properties file.

management.endpoint.shutdown.enabled=true

Spring Security

Q73. What is Spring Security?

Ans: Spring security is a powerful access control framework. It aims at providing authentication and authorization to java applications. It enables the developer to impose security restrictions to save from common attacks.

Q74. What are the features of Spring Security?

Ans: Spring security provides many features as below:

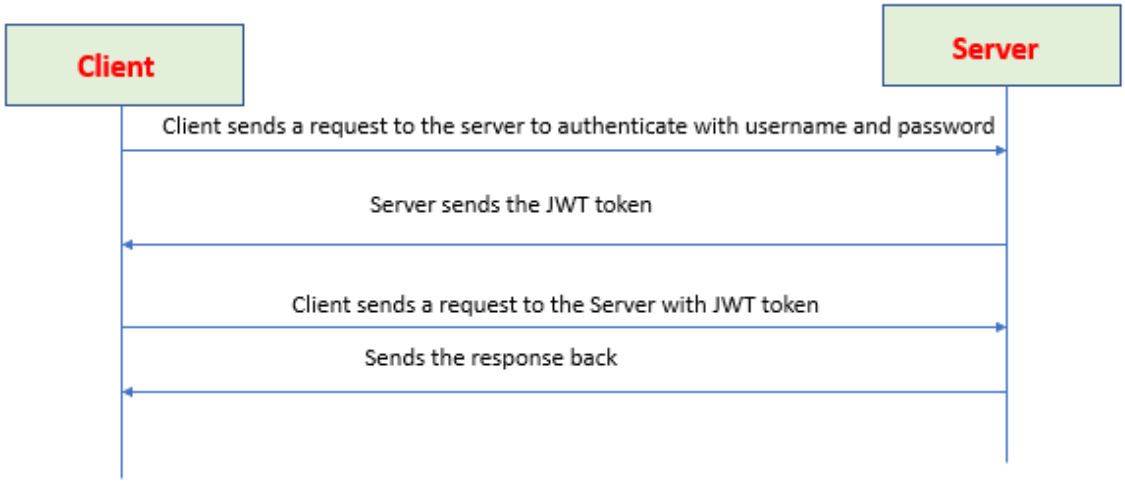
- *Authentication and Authorization.*
- *Supports Basic and Digest Authentication.*
- *Supports CSRF Implementation.*
- *Supports Single Sign-on.*

Q75. How to implement JWT?

Ans: JWT stands for Json Web Token which helps in implementing token-based security. Token is generated using the secret key. We need to add below dependency in pom.xml.

```
<dependencies>
        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>
        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>
    </dependencies>
```

The following diagram depicts the working of JWT.



Q76. What is DelegatingFilterProxy in Spring Security.

Ans: It is a predefined filter class which helps in performing pre-processing of the request.It supports for both authentication and authorization. It is a proxy servlet filter which acts as an intermediator before redirecting the request to dispatcher servlet.

Q77. What is Spring Security OAuth2.

Ans: OAuth2 is an authorization framework, granting clients access to protected resources via authorization server. It allows end user's account information to be used by third party services(eg.facebook) without exposing user's password.

The oAuth token are random strings generated by the authorization server.

There are 2 types of token.

Access token – It is sent with each request, usually valid for about an hour only.

Refresh token – It is used to get a new access token, not sent with each request. It lives longer than access token.

Q78. What is the advantage of using JWT Token?

Ans: Advantages of using JWT Token:

- *The jwt token has authentication details and expire time information.*
- *It is one of the approaches to secure the application data because the parties which are interacting are digitally signed.*
- *It is very small token and is better than SAML token.*
- *It is used at internet scale level, so it is very easy to process on user's device.*

Q79.What is authentication?

Ans: Authentication is the mechanism to identify whether user can access the application or not.

Q80.What is authorization?

Ans: Authorization is the process to know what the user can access inside the application and what it cannot i.e which functionality it can access and which it cannot.

Q.81. What is filter Chain Proxy?

Ans: The filter Chain Proxy contains multiple security filter chains and a task is delegated to the filter chain based on the URI mapping. It is not executed directly but started by DelegatingFilterProxy.

Q.82. What is security context in Spring Security.

Ans: It is used to store the details of the current authenticated user, which is known as principle. So if we want to get the current username, we need to get the SecurityContext.

Q.83. Difference between has Authority and hasRole?

Ans: hasRole – defines the role of the user.It does not use the ROLE_prefix but it will automatically added by spring security as hasRole(ADMIN);

has Authority – defines the rights of the user. It uses ROLE prefix while using has Authority method as has Authority (ROLE_ADMIN)

Q.84. How to enable spring boot security in spring boot project?

Ans: If Spring boot Security dependency is added on class path, it automatically adds basic authentication to all endpoints. The Endpoint "/" and "/home" does not require any authentication. All other Endpoints require authentication.

The following dependency needs to be added.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

Q.85.What is Basic Authentication?

Ans: In Basic Authentication, we send username and password as part of the request to allow user to access the resource. The user credentials are sent as authorization request headers. This approach is used to authenticate the client requests.

Q.86.What is Digest Authentication?

Ans: Digest Authentication is more preferable when compared to basic authentication as the credentials are in encrypted format by applying hash function to username,password,etc. It does not send the actual password to the server. Spring security provides digest authentication filter to authenticate the user using digest authentication header.

Q.87. How to get current logged in user in spring security.

Ans: We can get the current logged in user by using the following code snippet.

```
User user = (User)SecurityContextHolder.getContext().getAuthentication().getPrincipal();
```

String name = user.getUsername();

Q.88.What is SSL and its use?

Ans: SSL stands for secure socket layer which is an encryption- based internet security protocol.

It is mainly used to secure client information (such as credit card number /password/ssn) to a web server.

SSL provides a secure channel between two machines or devices running over the internet. A common example is when SSL is used to secure communication between a web browser and a web server. This changes the address of the website from HTTP to HTTPS, basically 'S' stands for 'Secure'.

Q.89. What is salting?

Ans: Salting is used to generate random bytes that is hashed along with the password. Using salt , we can add extra string to password , so that hackers finds difficult to break the password. The salt is stored as it is, and need not be protected.

Q90.What is hashing in spring security.

Ans: Hashing is an approach where a string is converted into encoded format using hashing algorithm. The hashing algorithm takes input as password and returns hashed string as output. This hashed data is stored in the database instead of plain text which is easily vulnerable to hacker attacks.

Q91.How to secure passwords in a web application?

Ans: Generally, passwords should not be stored as plain text into the storage as it can easily be accessed by the hackers.

We need to use encryption techniques before storing the password.

We need to use hashing and salting techniques to prevent from security breaches.

Q92.What is AuthenticationManager in spring security?

Ans: Authentication manager is the interface that provides the authentication mechanism for any object. The most common implementation of it is the AuthenticationProvider.

If the principal of the input is valid , and authenticated , it returns an authentication instance if successful.

It checks whether the username and password is authenticated to access a specific resource.

Q93. What are the various ways to implement security in spring boot project?

Ans: There are 3 ways to implement security.

a. *In Memory Credential Security*
b. *Using Database*
c. *Using UserDetailsService*

***In Memory Credential** – In this mechanism, we configure the user credentials in the application itself, and use it when there is a request to validate the user.*

Using JDBC Credentials *– Here, the user credentials are stored into the database and when the client request comes, it is validated against it.*

Using UserDetailsService *– It is an interface provided by Spring framework. After entering the username in the form and clicking on Login button invokes a call to this service. It locates the user based on the username provided. It contains a method loadUserByUsername(String username) which returns UserDetails object.*

General :

Q94. What are the essential components of Spring Boot?

Ans: Some of the components of Spring Boot are:

- *Spring Boot Starter*
- *Spring Boot autoconfiguration*
- *Spring Boot Actuator*
- *Spring Boot CLI*

Q95. What is the use of profiles in Spring Boot?

Ans: Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. For eg, if we want to enable swagger configuration only in QA environment, it can be done using spring profiles.

Q96. How can you set active profile in Spring Boot.

Ans: We can set the active profile by using configuration properties as:

spring.profiles.active=production

Q95. What is AOP?

Ans: Aspect Oriented Programming aims at separating the cross-cutting logics from the main business logic.

Q96. What is YAML?

Ans: YAML is mainly used for configuration purpose. It is similar to properties file and provides more readability.

Q99. Use of @Profile annotation.

Ans: The @Profile annotation indicates that a component is eligible for registration when the specified profile is active. The default profile is called default, all the beans that do not have a profile set belong to this profile.

Q100. How to get current profile in Spring Boot.

Ans: String[] activeProfiles = environment. getActiveProfiles();