**LINKS**

**Java8 Features:** <https://www.interviewbit.com/blog/java-8-features/>

**Java 11 Features:** <https://www.geeksforgeeks.org/java-11-features-and-comparison/>

**Spring Data Jpa interview questions:** <https://www.stackchief.com/blog/Spring%20Data%20JPA%20Interview%20Questions>

<https://www.stackchief.com/blog/Spring%20Data%20JPA>

<https://www.stackchief.com/tutorials/Spring%20Data%20JPA%20Tutorial>

<https://www.stackchief.com/blog/Spring%20Data%20JPA%20Example>

<https://www.stackchief.com/blog/Spring%20Data%20JPA%20CRUD%20Example>

<https://www.stackchief.com/blog/Spring%20Data%20JPA%20vs%20Hibernate>

**Spring vs Springboot:** <https://www.interviewbit.com/blog/spring-vs-spring-boot/#:~:text=In%20the%20Spring%20framework%2C%20you,configurations%20that%20allow%20faster%20bootstrapping.&text=Spring%20Framework%20requires%20a%20number,working%20with%20just%20one%20dependency.>

**SPRINGBOOT**

**What is Spring and spring boot?**

Spring is a java Framework built to create an enterprise ready application. When you create a java application there are a lot more things that you have to do. A lot of configurations, lot of property additions and lot of Technologies and a lot of packages and lot of modulus and JAR file that you need to add to create your applications. Spring is a Framework that allows us as a Java developer to do lot of the things. There are a lot of modules available for different kind of stuffs that we can use as a part of spring framework. That is Spring Core or spring MVC, spring web,  spring batch, spring data, spring data jpa, there are a lot of stuffs available in the spring framework.

We can use all those stuffs for our different activities. So we can only concentrate about actual coding that we have to do. But when you are creating the spring application, there is a lot of configurations that you have to do.

If you are working with spring framework and you want to include Hibernate and you have to do the configuration for that. If you want to include any caching mechanism then you have to do configuration for that. If you want to use any messaging queue you have to do the configuration for that as well.

So as a Java developer, as a spring user you have to do a lot of configuration to get ready with your application. After that once you create your application you have to deploy application to any of the application server or web server. So there are a lot of moving parts when you are working with the java application.

Spring framework  makes it easy for us to create all the application but that is a lot of configurations that we have to do. With this in mind spring developer thought of ok Let's create something easier so all the developers can really concentrate on the actual work. Only concentrate on the convention not the configuration part. For that they created the Spring boot.

Spring boot is just the extension of a spring framework. It’s not a different framework.

Spring boot provides key benefits for our Java developer,

Firstly the main thing is **Rapid application development**: Spring boot provides us to do the Rapid application development. So whatever application you want to create, you can create using spring boot very easily.

The other thing is **managing the dependencies**.When you are working with the spring framework, At that time also we have to add a lot of dependencies that you are using for your application.

Spring boot provides a way to group all those dependencies into a different starter templates. So spring boot provides different starter templates that includes all the dependencies that are required to do that particular task. Suppose if you want to work with the jdbc and there is a spring-boot-starter-jdbc template available that will include all the dependencies required to do that particular thing. If you want to work with JPA there is a spring-boot-starter-jpa template available that will include all the dependencies required to do that particular task. There is a spring-boot-starter-test available to do all our spring boot unit testing using the Junit and Mockito. So there are a lot of different templates available that you can directly used as a starter template and we can directly work with it.

The other thing spring boot provide is the **auto configuration**, so whenever we implement any of the things we have to do the configuration for all the stuffs. Suppose if we implement Hibernate we have to do the Hibernate configuration. Suppose if we implement any messaging queue we have to do the Hibernate configuration for that messaging queue.

If we implement any of the Other libraries there might be one or the other configurations  that we have to do to work with the spring framework. What spring boot does is, use auto configuration for all those dependencies, all those libraries that we can use.

So if we want to implement hibernate, just add the hibernate dependencies using spring-boot-starter, all those configurations will be added automatically using the spring-boot-auto-configurer template.

Next is the **Embedded server**: If we see the traditional way of deploying the java application is like, we create the application, we create the war file of the entire application and deploy that war file in any of the application server or the web server. Either JBoss, Tomcat, web sphere oranything. You will deploy war file to that particular server

 but with spring boot that particular server will be embedded to that particular entire application. so we won’tl be creating the war file, we will be creating the jar file and in that particular JAR file our server will be embedded. So we can directly run the JAR file in any of the environment. So it’s always production-ready. There are a lot more benefits to work with the spring application and the main thing is that market is moving towards the microservices architecture, rather than monolith.

So to create all the microservices using Java, Spring boot is the default option. So now with this it is pretty much clear that spring boot is the way to go.

**Dependency injection:**

Whenever we talk about the Spring Framework or spring boot, dependency injection is the topic that everyone discuss. Let’s understand why dependency injection is so hype over here. So if we take any of the programming languages or any of the frameworks for that particular programming languages dependency injection pattern is a default way they will go for creating the different objects in the application.

If we take the traditional example to create any of the class in your Java applications suppose student. now to create the object of the student what you will do student  s = new Student();  This means you yourself creating the object of that particular class over here. And when there is  a lot of hundreds and thousands of classes created and all those classes are interrelated to each other then it's not a good idea to use that approach to create the class manually

So in this type of scenarios inversion of control comes into picture.

what is inversion of control

 inversion of control is nothing but to give the control from yourself as a developer to the Framework that you are using Suppose if you are using Spring Framework that means we are giving the control to spring to create the object for us.

Now to implement that thing, dependency injection is the pattern that we use.

Dependency injection means suppose you have created one of the class that is student and in that particular class there is a another class that is subject available so you don't have to create that particular object for that dendency injection pattern will create the particular object for you whenever you want, so what will happen is when a spring boot application start, it have the factory that will create all the objects or all the beans of your application

Suppose if we have Hundreds and thousands of classes is in your application, all those particular classes object will be created when that particular spring boot context  is started. And all those particular beans will be stored in one of the springs container and whenever you want that particular bean or particular object, we will tell ok spring I want this particular class and spring will give you directly. You don't have to create the objects, spring already has all the objects you can directly use it.

So this is very important part to understand when we do the demo will be using it everywhere that the time I will show you how to do this but this is the theoretical part that you should understand what is IOC and what is dependency injection in which everything will be working all about inversion of control and penalties injection in Spring Framework now let's go ahead and start our applications

**Spring initializer:**

IF we go to start.spring.io, with this particular tool we can create the barebone spring boot project.

Here we can choose Project build tools like Maven Project or Gradle Project. We can choose Languages like Java, Kotlin, Groovy. We can choose the different spring boot versions available. I always go with the latest stable version.

We also have Packaging Jar, War. We can select the Java version 8, 11, 16.

We also have project metadata like Group, Artifact, Name, Description, Package name.

Now we can add the different dependencies for example to create restful applications, we can add the spring web dependency.

**Spring boot IDE's**

We can open the project in any of the IDE's like STS, Intellij IDEA, Eclipse, VS Code, etc.

There are a lot of different plugins available for all the different editors. We can add those plugins and it will be very useful for java applications as well.

**Spring boot starters:**

In pom.xml we have dependencies like spring-boot-starter-parent which takes care of the versions of other dependencies like spring-boot-starter-web, spring-boot-starter-test, etc.

Inside spring-boot-starter-web, we have spring-boot-starter, spring-boot-starter-json,spring-boot-starter-tomcat,spring-web, etc.

Similarly for other dependencies.

**Creating Simple API:**

The file which has the main method is the starting point for our spring boot application.

and this class is Annotated with @SpringBootApplication. This is the key annotation for our spring boot application.

@SpringBootApplication annotation is equalent to

@SpringBootConfiguration, @EnableAutoConfiguration, @ComponentScan.

@SpringBootConfiguration: It tells the application that this is the main spring boot configuration file.

@EnableAutoConfiguration: All the autoconfigurations for our application will be added to it. Suppose if we want some of the classes or some of the configurations not add automatically, we can also exclude them.

Suppose if we don't want to add the autoconfiguration for hibernate,we can exclude by adding the classes.

@ComponentScan:It will scan all the components available in your spring boot application and all those components will be added to the spring container when the application starts.

The SpringApplication.run method starts our application.

The application will run on Tomcat port 8080.

Go to localhost:8080, now we will get a Whitelabel Error Page because we have not created any of the API's now.

We can create a controller class inside controller package. To make this controller class as the component of spring so that whenever the application is started, this particular class is added to the spring container for that we need to annotate the controller class with @Component. But this controller class is not a simple component, its a Resouce, so we can use @Controller.

Since we are creating Restful APIs, this controller has to a Rest controller,for that we need to annotate the controller class with @RestController

@RestController defines it is a controller and also returns a responseBody

We can create method inside this controller.

Now whenever I hit an endpoint, this particular method should be executed for that we annotate with @RequestMapping(value="/", method = RequestMethos.GET)

Instead of using RequestMapping we can use GetMapping("/")

Now suppose if we want to run the application in port 8082, we can add the server.port in application.properties file.

We can run the application using mvn spring-boot:run

**Springboot Devtools:**

Whenever we are developing the application, we need to do a lot of changes and we need to start our servers. To stop and start the server every time is a big task. For that spring boot provides the spring boot devtools dependency that we can add into our project. Whenever there is any changes spring boot will detect the changes and it will restart the application.

We need to add the spring-boot-devtools dependency in pom.xml and Save and reload the maven configuration

**Architecture and example:**

From Frontend technologies any thing call this Restful APIs.

If we consider Department,

In Department controller class inside controller package,

This controller layer handles all our request,

Controller layer is just to get the request and send the response back. Just for the routing purpose.

We can create post api to save the department.

we can create an api to get all the departments,

we can create an api to get that particular department, we can create an api to delete the particulardepartment. we can create an api to update that department.

Service layer will be our business layer. All the business logic that we want to add to handle our department will be adding in our dept layer.

From the service layer we will be calling the DataAccess/Repository layer. This layer is responsible to intract with the database.

This layer will handle all the database operations.

For this layer we will be using spring-data-jpa dependency

We will also have a database.

**Adding Dependency H2 & JPA:**

Suppose if we want to use H2 database, we need to add h2 dependency in pom.xml and for JPA, we can add spring-boot-starter-data-jpa dependency and some configurations, spring.h2.console.enabled=true, spring.datasource.url=jdbc:h2:mem:dbname,spring.datasource.driverClaassName=org.h2.Driver, spring.datasource.username,spring.datasource.password,spring.datasource.platform=org.hibernate.dialect.H2Dialect

With this particular properties, our entire h2 database is configured for jpa and hibernate

We can go to localhost:8080/h2-console

**Creating components:**

entity:

Inside entity package we can create Department entity, Inside this Department class we can create properties: deptId, deptName, deptAdd,deptCode, etc. Create getters and setters for these properties. We can also create the no-args and all-args constructor and toString method.

Now to make this particular class as an entity so that it can interact with our database using jpa, we need to annotate this class with @Entity using the javax.persistance.Entity package.

Now this Department entity can interact with the database.

In the relational database all the tables will have a primary key. So for this entity also we need to create one. Here deptId will be the primary key, so to make this as primary key we need to annotate it with @Id.

To generate the primary key automatically, we add @GeneratedValue(strategy = GeneratedType.AUTO).

Controller:

Inside controller package we can create DepartmentController. We will annotate this class with @RestController. We can create Rest apis here.

Inside service package, we can create the DepartmentService interface and DepartmentServiceImpl class which implements DepartmentService. Annote the DepartmentServiceImpl class with @Service.

Inside repository package, we can create the DepartmentRepository interface and we annotate with @Repository.

This DepartmentRepository extends JpaRepository<Department, Long>()

For this JpaRepository, we need to pass the entity and primary key type.

This JpaRepository extends PagingAndSortingRepository and QueryByExampleExecutor.

This PagingAndSortingRepository extends CrudRepository.

**Department save API:**

Inside the DepartmentController, create a method to save the Department and annotate with @PostMapping and I need to call this particular post with "/departments".

As I am creating the rest api using the postmapping, ie, the post request that I will do from any of the rest client and I will be passing the RequestBody as well, so that will be entire json object. I want the entire json object to be coming over here and to convert the entire json object to my Department entity,for that we add @RequestBody

@RequestBody: Get the particular json and convert to my Department object.

Now we have to call the service layer to pass this data and in this service layer we do the business logic. For this we create private DepartmentService deptService and autowire it using @Autowired

Types of Autowiring/ dependency injection:

Constructor based and setter based

Property based

then we use deptService.saveDepartment(dept) to create an abstract method to save the department.

Now we have to implement this method in DepartmentServiceImpl. Now we have to call the repository layer to pass the data from Impl class.

For this we create private DepartmentRepository deptRepository and autowire it using @Autowired

then we use deptRepository.save(dept)

For testing this we can using any of the Rest client like Postman, Fiddler, Insomnia, etc..

Select POST, pass the endpoint url localhost:8082/departments

Pass json data and click on send

**Hibernate Validtion:**

Suppose if we have 4 fields comig as an input parameter to save or update the data. We can add validations like, I want the deptName to be available. Without deptName that particular json data, particular requestbody that we are getting from the client that should be invalid

For this,

We need to add the spring-boot-starter-validation,

above deptName field, add @NotBlank(message="Please Add Department Name")

Then in the particular controller, add @Valid from where request is coming.

Similary we have more validations,

@Length(max=5,min=1), @Size(max=10,min=0), @Email, @Positive, @Negative, @PositiveOrZero, @NegativeOrZero, @Future, @FutureOrPresent, @Past, @PastOrPresent

**Loggers in Springboot:**

Springboot comes with the slf4j logging library,

In all the class add private final Logger logger = LoggerFactory.getLogger(name: DeptController.class)

then in method add logger.info()

This loggers will be helpful for debugging our application.

**Project Lombok:**

As we are working with the java application, we tend to create a lot of POJOs, properties and for all those properties we will be creating getter setters, constructors, toString method. for 4 fields, we have lot of boilerplate codes, To getrid of this particular thing alibrary is available called Lombok

Lombok allows us to remove the boilerplatecode from our java application

Add lombok dependency,We have to tell maven plugin that we are using lombok , add lomok plugin.

Go to your entity and annotate with @Data(equivalent to @Getter, @Setter, @RequiredArgsConstructor, @ToString, @EqualsAndHashCode), @NoArgsConstructor, @AllArgsConstructor, @Builder(builder pattern will be implemented)

**Exception Handling:**

Inside error package, create a custom exception DepartmentNotFoundException, this will extend Exception and override methods inside that.

We will throw this custom exception when there is no department found

Whenever an exception is occuring at the controller layer, we have to identify that this is the exception thrown. And whenever that exception is there we have to send back the data as a response that we have to configure here.

For that we create RestResponseEntityExceptionHandler class, this will extend ResponseEntityExceptionHandler.

Now whatever the class you are creating to handle all your exceptions, that particular class should be annotated with @ControllerAdvice and we define this particular class will return the response status by annotating with @ResponseStatus.

Now inside this class we have to create one method, and that method will be responsible to handl that particular kind of exception and we have to annotate this method with @ExceptionHandler(ClassName.class)

For that create a class ErrorMessage, add properties: HttpStatus status, String message

**H2 to MySQL:**

spring.datasource.url=jdbc:mysql:localhost:3306/dbname,

spring.datasource.driverClaasName=com.mysql.jdbc.Driver, spring.datasource.username,

spring.datasource.password,

spring.jpa.hibernate.ddl-auto=update(to create tables)

spring.jpa.show-sql:true

Now add the mysql-connector-java dependency.

create the schema dbname

By default, JPA databases are automatically created only if you use an embedded database (H2, HSQL, or Derby)

You can explicitly configure JPA settings by using spring.jpa.\* properties. For example, to create and drop tables you can add the following line to your application.properties:

spring.jpa.hibernate.ddl-auto=create-drop

**Adding configurations in application.properties:**

Let me add welcome.message=Welcome home, to use this in our application,

Lets create a private String welcomeMessage , above this add @Value("${welcome.message}")

**application.yml file:**

This file is more human readable format and it reduces the duplicate values here ie, we can remove the redundant part.

Create a application.yml inside resources folder

server:

port: 8082

There are different plugins available to convert properties to yml.

**Springboot Profiles:**

Consider we have an application, now this application has to be deployed in our dev server,qa server, stage server, prod server for all the different environments.

For all these different environments we will be using different configuration properties.

For example:

The database configuration properties have to be different for all the environments.We can achieve this using profiles.

We will create different profiles for our dev, qa, stage and prod servers and we will add the different configurations for it.

When we deploy our application,we will tell spring that deploy my application using this particular profile.

Now in yml we can create multiple documents in single file itself.

For that add 3 hypens for separation between profiles and default one,

spring:

profiles:dev

In default one,

spring:

profiles:

active:dev

Otherwise we can create the different configuration files for different environments.

Suppose your profile name is dev,then we can create the application-dev.yml, similarly for others.

**Running springboot with multiple profiles:**

Change the version,

mvn clean install.

jar file will be created in the target folder.

java -jar jarFileName.jar --spring.profiles.active=prod