Common quotation ask

# *1* Internal working of Spring Boot application annotation:

* Spring Boot annotations encapsulate common configurations and behaviors, reducing boilerplate code.
* Annotations trigger various actions during the application lifecycle, such as component scanning, auto-configuration, and bean registration.
* Annotations like @SpringBootApplication combine multiple annotations, enabling cohesive setup and initialization.
* These annotations leverage Spring's reflection capabilities to dynamically configure and manage the application context.
* They enhance developer productivity by promoting convention over configuration and reducing the need for explicit setup.

1. Internal working of @Autowired annotation:

* @Autowired is part of Spring's dependency injection mechanism, facilitating the wiring of beans.
* Internally, Spring searches for candidate beans matching the type of the annotated field, constructor, or method parameter.
* It resolves dependencies based on the type, ensuring type safety and reducing manual configuration.
* @Autowired can be applied at the field, constructor, setter method, or parameter level, offering flexibility in injection points.
* Spring internally uses reflection and proxies to inject dependencies, decoupling components and promoting modularity.

# Internal working of @Configuration annotation:

* @Configuration marks a class as a source of bean definitions within the application context.
* Internally, Spring registers beans declared within @Configuration classes during context initialization.
* It provides a declarative way to define beans and their dependencies, enhancing maintainability and readability.
* @Configuration classes are processed by Spring's container at startup, generating bean definitions and wiring dependencies.
* Spring internally caches and manages configurations annotated with @Configuration, optimizing performance and resource utilization.

# Internal working of @Service annotation:

* @Service is a specialization of the @Component annotation, indicating that a class serves as a service component.
* Internally, Spring treats classes annotated with @Service similarly to @Component, facilitating component scanning and bean management.
* It enhances code organization and readability by explicitly marking classes as service components.
* @Service annotations enable the use of stereotypes, allowing Spring to apply specific behaviors and optimizations for service components.
* Spring internally registers @Service beans within the application context, making them available for dependency injection and other Spring features.

# What is a bean in Java?:

* In Java, a bean is a reusable software component that conforms to specific conventions, typically representing application data or functionality.
* Beans are Java objects managed by a container, such as Spring's IoC container, and are typically configured and assembled at runtime.
* Beans are often POJOs (Plain Old Java Objects) annotated with Spring annotations like @Component, @Service, or @Repository.
* Beans can represent various types of components, including domain objects, data access objects, services, and controllers.
* Spring beans are singleton by default, but their scope can be customized to match different application requirements.

# What is @Bean in Java in detail?:

* + @Bean is a method-level annotation used within classes annotated with @Configuration in Spring.
  + It serves as a factory method for creating and configuring beans explicitly.
  + Internally, Spring invokes @Bean methods during application context initialization to instantiate and configure the associated beans.
  + @Bean methods can be parameterized, allowing dependencies to be injected into the created beans.
  + @Bean annotations provide a way to define beans with custom initialization logic, non-default scopes, and other customizations.

# What is dependency injection in Java?:

* + Dependency Injection (DI) is a design pattern aimed at reducing coupling between classes by externalizing dependencies.
  + In Java, DI involves providing a class's dependencies from an external source, typically a container like Spring's IoC container.
  + Spring supports various forms of DI, including constructor injection, setter injection, and field injection, achieved through annotations like @Autowired.
  + DI promotes modular, maintainable code by allowing classes to focus on their primary responsibilities without being concerned about how their dependencies are created or managed.
  + It simplifies unit testing by facilitating the substitution of dependencies with mock objects or stubs, enabling isolated testing of components.

# How does a POST request work in microservices architecture?:

 Client sends a POST request:

* The process begins when a client sends a POST request to an endpoint exposed by an API gateway.

 API Gateway routes the request:

* The API gateway receives the incoming POST request and routes it to the appropriate microservice based on the request URL.

 Microservice receives the request:

* The targeted microservice, which is built using Spring Boot, receives the POST request.

 Spring MVC Controller handles the request:

* Within the microservice, the Spring MVC controller maps the incoming POST request to a handler method annotated with @PostMapping or @RequestMapping.

 Request processing in the microservice:

* Spring processes the incoming request data, which may include request parameters, form data, or JSON payloads, based on the method signature of the handler method.

 Service Layer interaction:

* The controller method invokes the appropriate service layer components to perform business logic, data validation, and interaction with databases or other microservices.
* Spring's @Autowired annotation or constructor injection is commonly used to inject service dependencies into the controller.

 Data persistence:

* If required, the service layer interacts with a database, such as MySQL or MongoDB, to store or retrieve data.

 Service discovery with Eureka:

* Before sending the response, the microservice may need to communicate with other microservices. In a microservices architecture utilizing service discovery, the microservice looks up the network location of other microservices using a service registry like Eureka.

 Invoking other microservices:

* If necessary, the microservice may make requests to other microservices to fulfill the business logic of the POST request. This interaction can be synchronous or asynchronous, depending on the requirements.

 Response construction:

* Once the business logic is executed and any necessary interactions with databases or other microservices are completed, the microservice constructs an HTTP response.
* Spring uses @ResponseBody to serialize the response data into JSON or another format suitable for the client.

 Response sent to the client:

* The constructed response is sent back to the client through the API gateway, which forwards the response to the original requester.

# Difference between @ResponseBody and @RequestMapping:

* + @ResponseBody is an annotation used to indicate that the return value of a method should be bound to the web response body in Spring MVC.
  + @RequestMapping is used to map HTTP requests to handler methods in Spring MVC controllers, specifying the URL path and HTTP method.
  + While @ResponseBody specifies the content of the response body, @RequestMapping defines the mapping of requests to handler methods.
  + @ResponseBody is typically used in conjunction with @RequestMapping or @GetMapping, @PostMapping, etc., to produce the response body.
  + @RequestMapping can be applied at the method or class level to define request mappings for multiple methods within a controller.

# What is the Spring context?:

* The Spring context is the core of the Spring Framework, providing the runtime environment for managing application components and their dependencies.
* It represents the application context or container that initializes, configures, and manages beans throughout the application lifecycle.
* The Spring context loads bean definitions, resolves bean dependencies, and applies configuration settings specified in XML files, Java configuration classes, or annotations.
* Spring contexts can be hierarchical, allowing for the creation of parent and child contexts to organize and modularize application components.
* The Spring context provides various services such as dependency injection, internationalization, event propagation, and resource loading to facilitate application development.

# Spring security annotation

Spring Security provides several annotations that you can use to secure your Spring-based applications. These annotations help in applying security rules to methods and endpoints. Here are some of the key Spring Security annotations:

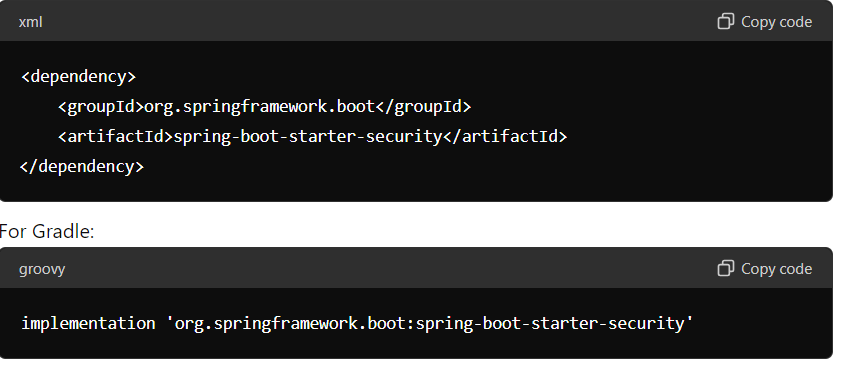
1. **@Secured**: // Method accessible only to users with ROLE\_USER authority
   * @Secured is a method-level security annotation used to specify which roles are allowed to access a method.
   * You can annotate a method with @Secured("ROLE\_USER") to ensure that only users with the "ROLE\_USER" authority can invoke that method.
   * It requires method security to be enabled in the Spring Security configuration.
2. **@PreAuthorize** // Method accessible only to users with ROLE\_ADMIN authority
3. **and @PostAuthorize**: // Method returns an object, and only objects created by the current user are allowed
   * @PreAuthorize and @PostAuthorize are method-level security annotations that allow you to specify complex security expressions using SpEL (Spring Expression Language).
   * @PreAuthorize is evaluated before the method invocation to determine whether the user is allowed to access the method.
   * @PostAuthorize is evaluated after the method invocation to filter the results based on the specified security expression.
   * These annotations provide more flexibility compared to @Secured.
4. **@RolesAllowed**: // Method accessible only to users with ROLE\_ADMIN authority
   * @RolesAllowed is similar to @Secured, but it's a standard Java EE annotation.
   * You can use @RolesAllowed("ROLE\_ADMIN") to restrict access to a method to users with the "ROLE\_ADMIN" authority.
   * Spring Security provides support for @RolesAllowed through its integration with Java EE security annotations.
5. **@PreFilter** // Filters the list of objects based on the owner field
6. **and @PostFilter**: // Retrieves a list of objects and filters based on the owner field
   * @PreFilter and @PostFilter are annotations used for method-level security filtering.
   * @PreFilter allows you to filter method parameters before the method invocation based on a specified security expression.
   * @PostFilter filters the return value of a method after the method invocation based on a specified security expression.
7. **@Secured, @PreAuthorize, @PostAuthorize, @PreFilter, and @PostFilter**:
   * These annotations provide powerful tools for securing methods and endpoints in Spring applications.
   * They allow you to express fine-grained security rules using SpEL expressions and annotations.
   * By combining these annotations, you can implement complex security requirements with ease.

These annotations offer a declarative way to define security rules in your Spring applications, making it easier to manage and maintain security configurations. You can use them to secure your controllers, services, and other components effectively.

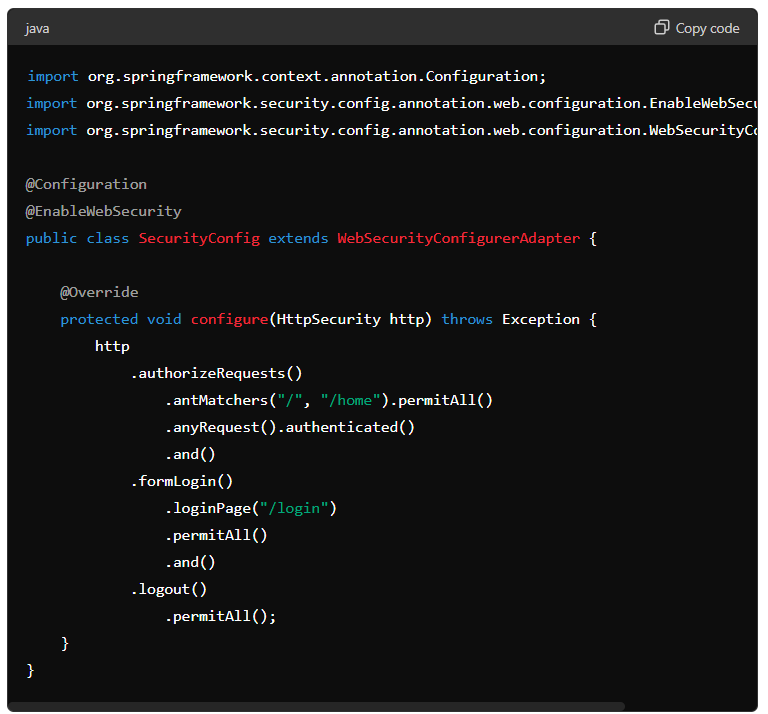
# Implement security spring boot application

Implementing Spring Security in a Spring Boot application involves several steps. Here's a guide to help you get started:

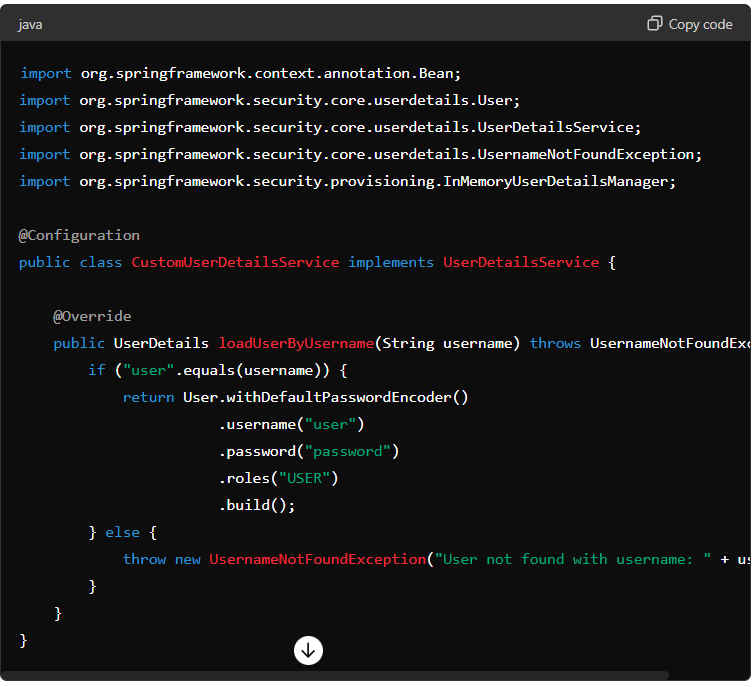
1. **Add Spring Security Dependency**:
   * Add the Spring Security dependency to your pom.xml (for Maven) or build.gradle (for Gradle) file.



**Configure Spring Security**:

* Create a configuration class to customize Spring Security settings.
* You can extend WebSecurityConfigurerAdapter to configure security settings.
* Override the configure(HttpSecurity http) method to define security rules.
* Example:

**Create UserDetailsService**:

* Implement a UserDetailsService to load user details from a data source.
* 

**Secure Endpoints**:

* Secure endpoints based on roles or specific authentication requirements.
* Use annotations such as @Secured, @PreAuthorize, or @RolesAllowed to secure methods or controllers.



**Testing**:

* Test your application to ensure that security configurations are working as expected.
* Verify that unauthorized access to protected endpoints is restricted.
* Test authentication and authorization with different user roles.

# Authentication and authorization

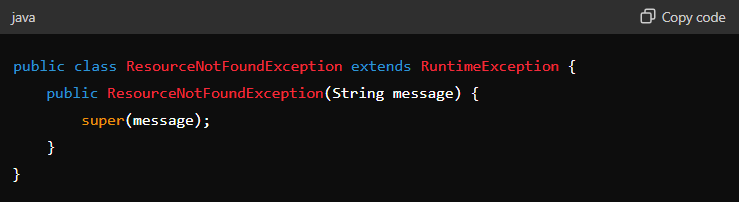
Authentication and authorization are two fundamental concepts in security, particularly in the context of software applications. Here's a breakdown of each concept:

1. **Authentication**:
   * Authentication is the process of verifying the identity of a user or system.
   * It ensures that the entity claiming to be a particular user or system is indeed that entity.
   * Authentication mechanisms include:
     + Username and password authentication: Users provide their credentials (username and password) to prove their identity.
     + Token-based authentication: Users present a token (e.g., JWT) generated after successful login to authenticate subsequent requests.
     + Certificate-based authentication: Users present a digital certificate issued by a trusted authority to authenticate themselves.
   * In Java, authentication is often implemented using frameworks like Spring Security, which provides authentication providers, filters, and mechanisms to authenticate users based on various methods.
2. **Authorization**:
   * Authorization is the process of determining whether an authenticated user or system has the necessary permissions to access a particular resource or perform a specific action.
   * It ensures that users can only access resources and perform actions that they are authorized to access or perform.
   * Authorization mechanisms include:
     + Role-based access control (RBAC): Users are assigned roles, and access to resources is based on the roles assigned to them.
     + Attribute-based access control (ABAC): Access decisions are based on attributes of the user, resource, and environment.
     + Rule-based access control: Access decisions are determined by rules defined by administrators or policies.
   * In Java, authorization is often implemented in conjunction with authentication using frameworks like Spring Security, which provides mechanisms to define access control rules, roles, and permissions.

# Exception Handel in spring boot application

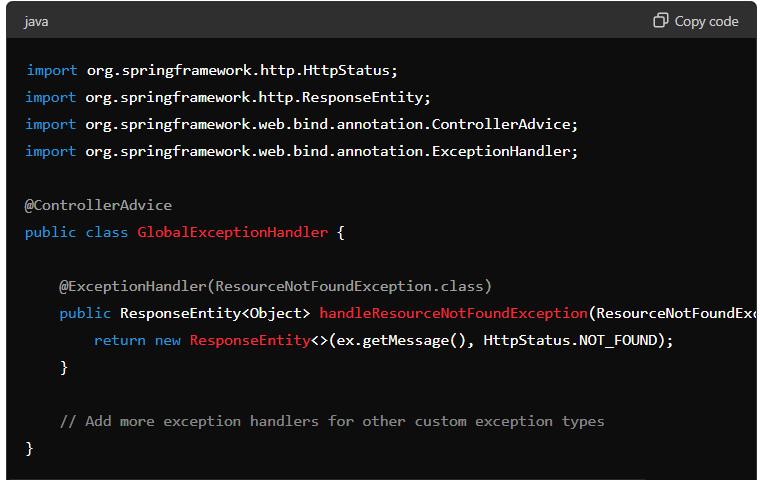
Implementing exception handling in a Spring Boot application involves configuring global exception handling to manage exceptions gracefully. Here's a guide on how to do it:

1. **Create Custom Exception Classes**:
   * Define custom exception classes that extend RuntimeException or its subclasses to represent specific types of errors in your application.



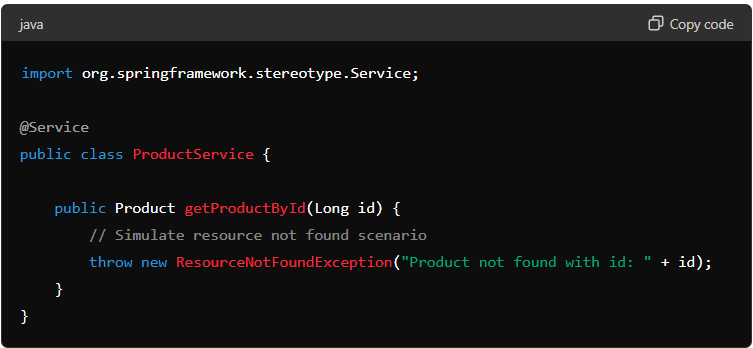
**Define Exception Handler**:

* Create a global exception handler class annotated with @ControllerAdvice to handle exceptions thrown by controllers.
* Annotate methods within this class with @ExceptionHandler to handle specific types of exceptions.



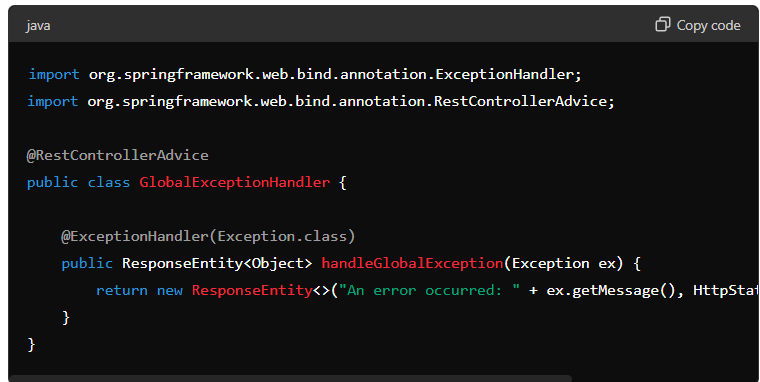
**Throw Custom Exceptions**:

* Within your controller or service classes, throw custom exceptions when specific error conditions occur.



**Configure Default Exception Handling** (Optional):

* If needed, configure default exception handling to catch any unhandled exceptions and provide a generic error response.



**Test Exception Handling**:

* Write unit tests to verify that exceptions are handled as expected.
* Mock service calls or controller invocations that may throw exceptions and assert the response status and content.

# Exception Handel in Anotation

**@ControllerAdvice**:

* + Used to define global exception handling for all controllers.

1. **@ExceptionHandler**:
   * Used within a @ControllerAdvice class to handle specific types of exceptions.
2. **@ResponseStatus**:
   * Annotates a custom exception class to specify the HTTP status code to be returned when the exception is thrown.
3. **@RestControllerAdvice**:
   * Similar to @ControllerAdvice, but specifically used for handling exceptions in RESTful controllers.

These annotations provide a convenient way to manage and handle exceptions in your Spring Boot applications, allowing for centralized error handling and consistent responses to clients.

>Roles and responsiblites in the project?

ans: In the project my role is used to develop the code by using client requirement and i need to send the code for testing side in that where any testers find any issues i need to resolve it and in our group we are group of 4 where we play a crucial role for development side and i am the team member and our team lead will assign the work in btw our group.

==============================================================================================

->Spring boot version?

ans:ha we have used the latest spring boot version y becaz the spring boot version will be changing every 6months like they used to remove some spring modules and they will update the new version.

==============================================================================================

->new features added in spring boot?

ans:There are many features that are added to the spring boot they are like microservices support means building and deploying microservices architecture and devops tools like jeniks and spring secuirty feautres and it support aws lambda or azure function inthe spring boot ...etc

==============================================================================================

->advantages with spring boot?

ans:1.we no need to add the dependencies manually

2.And if we do any modifications by using dev tools the server will automatically run

3.it has embedded servers like tomcat,jetty,netty.

4.microservices support

5.integration with spring eco-system like spring data jpa ,security and ...etc

==============================================================================================

->embedded servers supported by boot?

ans:1.tomcat(default server)

2.jetty

3.netty

==============================================================================================

->how to make jetty as default container?

ans:we need to add the dependency of the jetty and we need to do modification inthe application.properties and pom.xml

==============================================================================================

->what is ioc in springboot?

ans: ioc means inverse of control which is used to manage the dependency injection inthe spring boot.

============================================================================================

->start class in spring boot?

ans: the class which contains main method inour application

==============================================================================================

->what is component scaning in springboot?

ans:1.it is used to scan the project and identify the how many spring beans avaliable inthe application

2.component scaning works based on the base package.

example:in.ashokit.application.java

in.ashokit.restcontroller

in.ashokit.service

in.ashokit.dao

in.ashokit -it does not consider as component scan.

==============================================================================================

->what is the difference btw component and service(metadata)?

ans:1)In component it is used to scan and identify the how many number of spring beans are avaliable inthe application.

2) and even inthe service it is represented as spring bean class

3)but in the service it is used to encapasulate the bussiness logic

4)and inthe service we can perform crud operations also.

5)comming to the component the component are typically annotated with @component, @restcontroller,@controller...etc

6)Components are generally used to represent general-purpose beans in your application, such as data access objects (DAOs), controllers, or utility classes.

==============================================================================================->how to connect with mysql db using springboot?

ans:we need to configure the application.properites inthe resources file inthat file we need to add mysql db properties for example

spring.datasource.username=root

spring.datasource.password=PHW#84#jeor

spring.datasource.url=jdbc:mysql://localhost:3306/sbms39

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

Note:By using the above db properites we can connect with mysql db.

.And if we want to connect with oracle or mongdb we need to change url username password.

==============================================================================================

->how to connect with multiple database?

Ans:In spring boot we can connect with multiple database by using datasource configuration,jdbc template or spring data jpa,routing datasource(in routing datasource we utilize abstractroutingdatasource in which we can connect multiple databases dynamically.and the last one is custom datasource configuration.

==============================================================================================

->how to manage transcation in springboot?

ans:we can manage transcation in springboot by using spring security

1)and by enabling the @enabletranscationManagment in the springboot application

2)And by using @Transcational annotation we can manage transcation

3)And by using transcation rules like rollbackfor and norollbackfor attributes of the @transcational

===========================================================================================

->transcation propogation levels?

ans:there are different types of transcation propogation levels

1)required-required anedhi eppudu use avuthundhi ante manaki transcation context anedhi existing ga unte appudu a existing transcation anedhi excute avuthundhi with inthe method ade okvela context anedhi exist ayi lekhapothe appudu manaki new transcation anedhi create avuthundhi.

2)Not required-okavela kotha transcation anedhi create ayithe method ki appudu manaki una current transcation ni suspend chesthundhi.

3)nested-current transcation exist ayi unte manaki execution anedhi within the nested transcation lo jarguthadhi and okvela idhi jaragakapothe same required lo ela ayith exist ayi undho alaney untadhi.

4)support-support anedhi non-transcationally ga execute avuthadhi eppudu ante transcation context existing ga lekhapothey

5)not supported-not supported is used to execute non-transcationally atleast if there is one transcation exist.

6)never-Requires that no transaction context exists for the method to execute. If a transaction context exists, an exception is thrown.

7)unknown-The transaction manager should determine the propagation behavior dynamically based on the transaction context.

-=======================-=-=-=-=-=====================-=-=-=-=============-=-=========-==---==

->how to call store procedure in springboot?

ans:-so in the spring boot we can call it in three different types

1)jdbc template

2)spring datajpa

3)spring datajdbc.

By using the above three we can call store procedure inthe springboot.

==============================================================================================

->what is queryByexample(filtering the queries)?

ans:-querybyexample is used to filter the queries.

1)and it is used for the dynamic queries

==============================================================================================

->what is specification in datajpa?

ans:1)in the datajpa the specification plays a crucical role for creating dynamic queries

2)Specifications are often used when you need complex and dynamic query predicates based on various conditions.

3)a specification is an interface which extends specification<t>

==============================================================================================

->profiles in spring boot?

->what is the main difference btw properties file and yml

==============================================================================================

->acutators in spring boot

ans:acutators are used to monitor and manage the application.

===========================================================================================

->acutators endpoint inthe springboot(health,bean,loggers,metrics,env,mappings,aduitevents)

ans:there are different types of endpoints inthe acutators they are

1.health

2.bean

3.loggers

4.metrics

5.env

6.mappings

7.aduitevents

==============================================================================================

->what is microservices?

ans:microservices means architectural design pattern and it is loosely coupling and it is easy mataince and it has no single point of failure and technology independent.and now days the microservices are very usefull inthe software industry where we can not create project by using microservices.

=============================================================================================

->what is apigateway?

->how to secure microservices?

==============================================================================================

->what is containerization?

ans:converting the our java application code and dependencies as a single unit inthe container is called containerization.

==============================================================================================

->what are the annotations used in your springboot application or project?s

ans:in project we have used different types of annotations as per the requirement for example

1.restcontroller

2.entity

3.id

4.generated value

5.controller

6.service

7.component

8.bean

9.configuration

10.autoweired

=====================================================================================================================================================================================================mmockinterview questions

------------------------

->explain about yourself ?

->explain about your project?

->roles and responsibilities?

->recent task you have done in project?

->challenges faced in project?

->multi threading scenario in project?

->oops concepts scenario

->lambda expressions

->spring boot?

->auto configuration in springboot?

->what is run method in springboot?

->run method in springboot class?

->what is @springbootapplication?

->what is @configuration?

->@configuration vs @bean?

->autowiring modes(byname vs byType)

->SI vs CI vs FI

->what is scopes of the springbean

->what is runners in springboot

->how to customize banner in springboot

->what is lazy loading in ioc?

->@controller vs @restcontroller

->what is @requestbody

->query parameter in request

->what is path variable

->how to handel exception in rest api?

->explain http status code

->put vs patch methods?

Top of Form

Bottom of Form