* **What is Web service ?**
* Web services are **XML-based information exchange systems that use the Internet for direct application-to-application interaction**. These systems can include programs, objects, messages, or documents. **A web service is a collection of open protocols and standards used for exchanging data between applications or systems.**
* **What is XML ?**
* XML (Extensible Markup Language) is **a markup language similar to HTML, but without predefined tags to use**. Instead, you define your own tags designed specifically for your needs. This is a powerful way to store data in a format that can be stored, searched, and shared.
* **What is JSON ?**
* JSON stands for Javascript Object Notation. JSON is **a text-based data format that is used to store and transfer data**. In JSON, the data are in key/value pairs separated by a comma .
* **Which is better json or xml ?**
* **JSON is best for simple applications**, developed to satisfy simple requirements surrounding data interchange. XML is best for applications with complex requirements surrounding data interchange, such as in enterprise.
* **What is API and RestAPI ?**
* The primary goal of API is to standardize data exchange between web services. Depending on the type of API, the choice of protocol changes. On the other hand, **REST API is an architectural style for building web services that interact via an HTTP protocol.**
* **What is dispatcherServlet ?**
* **DispatcherServlet** is a class that’s works as the front controller and receives the incoming request and maps it to the right resource such as controllers, models, and views.
* **What is @Component ?**
* **@Component is an annotation that** **allows Spring to automatically detect our custom beans**. In other words, without having to write any explicit code, Spring will: Scan our application for classes annotated with @Component. Instantiate them and inject any specified dependencies into them
* **What is @Service and @Repository different ?**
* **@Service** annotates classes at the service layer**.** **@Repository** annotates classes at the persistence layer, which will act as a database repository.
* **What is @ControllerAdvice ?**
* @ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component.
* **What is @ExceptionHandler ?**
* @ExceptionHandler is an annotation used to handle the specific exceptions and sending the custom responses to the client. Example : @ExceptionHandler(EmptyException.class)
* **How to create a custom exception response ?**
* Create a template of your custom exception output(ExceptionResponse class) and the exception class by extending RunTimeException. Create a global exception handling class using @ControlllerAdvice and @RestController with extending ResponseEntityExceptionHandler. Create method like example :

*@ControllerAdvice*

*@RestController*

*public class CustomizedResponseEntityExceptionHandler extends ResponseEntityExceptionHandler {*

*@ExceptionHandler(EmptyException.class)*

*public final ResponseEntity<ExceptionResponse> handleUserEmptyException(EmptyException ex,WebRequest request){*

*ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),ex.getMessage(),*

*request.getDescription(false));*

*return ResponseEntity.badRequest().body(exceptionResponse);*

*}*

*}*

* **What is Validation in RestAPI ?**
* In RestAPI, validation is use to add restrict the data store in variable. @Valid annotation is use to indicate that validation should be use in this request. Example:

*public ResponseEntity<Object> addUserDetail(@Valid @RequestBody UserEntity user)*

There many annotation put on variable , some are:

1. @NotNull: to say that a field must not be null.
2. @NotEmpty: to say that a list field must not empty.
3. @NotBlank: to say that a string field must not be the empty string (i.e. it must have at least one character).
4. @Min and @Max: to say that a numerical field is only valid when it’s value is above or below a certain value.
5. @Pattern: to say that a string field is only valid when it matches a certain regular expression.
6. @Email: to say that a string field must be a valid email address.

Example:

*@Size(min = 2,message = "Name should be at least 2 characters")*

*private String name;*

*@Past(message = "birthday date should be in past")*

*private Date birthDate;*

@Size and @Past , both are validation annotation , message is the default message send/pass as exception message.

* **What exception does validation throw ?**
* When the target argument fails to pass the validation, Spring Boot throws a [MethodArgumentNotValidException](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/MethodArgumentNotValidException.html) exception which present in ResponseEntityExceptionHandler class

In example we are just override it and sending our exception template. getBindingResult method holds the result of a validation and binding and contains errors that may have occurred . getFieldError()).getDefaultMessage(), this is added to filter the message and only show the default message pass by programmer at validation annotation.

*@Override*

*protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus status, WebRequest request) {*

*ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(), "Validation fail",*

*Objects.requireNonNull(ex.getBindingResult().getFieldError()).getDefaultMessage());*

*return ResponseEntity.badRequest().body(exceptionResponse);*

*}*

* **What is HATEOAS ?**
* The Spring HATEOAS project is a library of APIs that we can use to easily create REST representations that follow the principle of HATEOAS (Hypertext as the Engine of Application State).

**Generally speaking, the principle implies that the API should guide the client through the application by returning relevant information about the next potential steps, along with each response.**

**Spring HATEOAS offers three abstractions for creating the URI – RepresentationModel, Link, and WebMvcLinkBuilder**

Example 1:

*//creating a EntityModel of UserEntity*

*EntityModel<UserEntity> userEntityEntityModel = EntityModel.of(user);*

*//creating a link of get all user details*

*WebMvcLinkBuilder linkToGetAllUser = WebMvcLinkBuilder.linkTo(*

*WebMvcLinkBuilder.methodOn(Controller.class)*

*.getAllUser());*

*//adding the link to EntityModel*

*userEntityEntityModel.add(linkToGetAllUser.withRel("to get all users"));*

*Example 2:*

*//userList contain EntityModel of UserEntity*

*userList.forEach(user ->*

*user.add(WebMvcLinkBuilder.linkTo( // creating link with linkTo() method*

*WebMvcLinkBuilder.methodOn(Controller.class) /\*selecting the class where that link*

*\*(request map) is declare\*/*

*.getUserById( //method of that link*

*Objects.requireNonNull( //it is added so id of user is not null*

*user.getContent()) /\*getContent() method will give the*

*\*UserEntity object from EntityModel\*/*

*.getId())) /\*id of use to pass as argument of*

*\* getUserById( int id) method \*/*

*.withRel("link to get user by id"))); //to inform client what is this link for ?*

* **How to send response in XML format ?**
* By adding dependency and using produce in request mapping annotation we can return the response in xml format. We can also change accept tag in header of request to get xml format instead of using produce in annotation.

Example:

***<dependency>***

***<groupId>com.fasterxml.jackson.dataformat</groupId>***

***<artifactId>jackson-dataformat-xml</artifactId>***

***</dependency>***

*@GetMapping(path = "/user/{id}/{name}",produces = "application/xml")*

* **What is openapi ?**
* springdoc-openapi java library helps to automate the generation of API documentation using spring boot projects. springdoc-openapi works by examining an application at runtime to infer API semantics based on spring configurations, class structure and various annotations.

Automatically generates documentation in JSON/YAML and HTML format APIs. This documentation can be completed by comments using swagger-api annotations.

Just by adding the Dependency in pom file swagger-api documentation is generate:

***<dependency>***

***<groupId>org.springdoc</groupId>***

***<artifactId>springdoc-openapi-ui</artifactId>***

***<version>1.6.6</version>***

***</dependency>,***

* **What is HAL ?**
* JSON Hypertext Application Language, or HAL, is a simple format that gives a consistent and easy way to hyperlink between resources in our API. Including HAL within our REST API makes it much more explorable to users as well as being essentially self-documenting.It works by returning data in JSON format which outlines relevant information about the API.
* **What is Actuator ?**
* **Spring Boot Actuator** is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live). We can use **HTTP** and **JMX** endpoints to manage and monitor the Spring Boot application. If we want to get production-ready features in an application, we should use the S**pring Boot actuator.**
* **What is static filtering in spring boot ?**
* If we do not want to show some class members in the response. We use a feature to remove those class member from response , this process is called filtering. Jackson has two annotations that are used in filtering are:

1. **@JsonIgnore :** JsonIgonre is a member or method level annotation. It expects that the properties to be excluded are marked one by one. If we want to eliminate a member from the process of serialization and deserialization. It is use in entity class above a variable. Example:

*@JsonIgnore*

*private String salary;*

1. **@JsonIgnoreProperties** : **It** is a class-level annotation. It ignores the logical properties in JSON serialization and deserialization (*JsonIgrone annotation is better because if the variable name change or any spell mistake it will not work*). It is use in entity class above the class. Example:

*@JsonIgnoreProperties({"name", "phone"})*

*public class SomeBean{}*

* **What is Dynamic filtering in spring boot and what is different from static filtering?**
* We can define ***different filters for different services***, according to need for example there are three fields: name, phone, and salary. We want to send two fields: name and salary for the first service and name and phone for the second service. Here we use ***dynamic filtering*** but in ***static filtering*** we can’t do that once filtering is applied it will ***work same for all response***. Dynamic filtering is use in controller class in the response method.
* **What is the process of applying dynamic filtering in spring boot ?**
* **Step 1:** Open Controller.java file.

**Step 2:** Create a constructor of MappingJacksonValue class and pass a bean (someBean) as a constructor argument. We want to create a mapping Jackson value for this particular bean.

***MappingJacksonValue mapping = new MappingJacksonValue (someBean);***

**Step 3:** To configure the filters, we need to create them. To create a filter, declare local variable filters of type FilterProvider. FilterProvider is an abstract class. It has a single implementation of the SingleFilterProvider method. Invoke the addFilter() method that has two parameters String id and SimpleBeanPropretyFilter filter.

***FilterProvider filters=new SimpleFilterProvider().addFilter("SomeBeanFilter", filter);***

**Step 4:** Invoke the static method filterOutAllExcept() of the class SimpleBeanPropertyFilter class. It filters all the fields in response except the fields which we have specified. We want to send the name and salary field in the response, so we have specified these two fields.

***SimpleBeanPropertyFilter filter=SimpleBeanPropertyFilter.filterOutAllExcept("name", "salary");***

**Step 5:** Configure the filters.

***mapping.setFilters(filters);***

**Step 6:** Instead of returning the someBean return mapping.

***return mapping;***

**Step 7:** We have returned mapping, so we are required to change the return type of the method to MappingJacksonValue.

**Step 8:** Open SomeBean.java file and define a filter by using the annotation **@JsonFilter**. It is used at class level. It defines a filter name which we filter out properties in JSON serialization.

***@JsonFilter("SomeBeanFilter")***

* **How to create a versioning request ?**
* There are many ways :

1. **URL Versioning** : Here we change the URL of the request for different versions by mentioning the version in URL of request.

**Example**: *GetMapping(“/v1/user) and GetMapping(“/v2/user”),*

1. **Request Parameter Versioning**: Here we add param in the URL and check the param in receiving request method.

**Example**: http://localhost:8080/person?version=1

*@GetMapping(value = "/person", params = "version=1")*

1. **Header Versioning** : Here we have to pass key pair value in header which will be check by the receiving request method. Key = X-API-VERSION and value = 1 key pair value is depend on headers = "X-API-VERSION=1"

**Example**: *http://localhost:8080/person/header*

*@GetMapping(value = "/person/header", headers = "X-API-VERSION=1")*

1. **Media Type Versioning**: Here we pass key pair value same as header versioning but we use produce instead of header and key will be always Accept and value is the produce. key = Accept and value = application/vnd.company.app-v1+json

**Example**: [*http://localhost:8080/person/produces*](http://localhost:8080/person/produces)

*@GetMapping(value = "/person/produces", produces = "application/vnd.company.app-v1+json")*

* **What are factors different type of versioning follow?**
* The factors are:

1. **URl Pollution** – **URL** and **Param versioning** as we are mentioning version in URL.
2. **Misuse of Header** – **Media type** and **Header versioning** as header where never induce for versioning.
3. **Caching Problem** – **Media type** and **Header versioning** as URL is same for all versions.
4. **Execute of Request** – **Media type** and **Header versioning** because we have to pass information in the header during sending request, which is not possible in normal bowsers. We have to use special bowsers or plugins (only for get request).

* **How to add security in spring boot ?**