Spring-boot : REST API

Chaouki Bayoudhi

Spring-boot Presentation

Spring is widely used for creating scalable applications. For web applications Spring provides

Spring MVC which is a widely used module of spring which is used to create scalable web applications.

But main disadvantage of spring projects is that configuration is really time-consuming and can be a bit overwhelming for the new developers. Making the application production-ready takes some time if you are new to the spring.

=>Solution to this is Spring Boot.

Spring Boot is built on the top of the spring and contains all the features of spring. And is becoming favourite of developer's these days because of it's a rapid production-ready environment which enables the developers to directly focus on the logic instead of struggling with the configuration and set up.

Spring-boot Presentation

- **Spring Boot is a modern and extensible development framework that aims (and succeeds!) to take as much pain as possible out of developing with Java.**
- **With just a few Maven dependencies, new or existing programs become runnable, init.d-compliant uber-JARs or uber-WARs with embedded web-servers and virtually zero-configuration, code or otherwise.**
- **As an added freebie, Spring Boot Actuator will provide your programs with amazing configuration-free production monitoring facilities that let you have RESTFUL endpoints serving live stack-traces, heap and GC statistics, database statuses, spring-bean definitions, and password-masked configuration file audits.**

Spring-boot Presentation

Spring Boot allows to avoid heavy configuration of XML which is present in spring:

For example:

If we want to use hibernate(ORM) then we can just add @Table annotation above model/entity class(discussed later) and add @Column annotation to map it to table and columns in the database

It provides easy maintenance and creation of REST end points:

Creating a REST API is very easy in Spring Boot. Just the annotation @RestController and @RequestMapping(/endPoint) over the controller class does the work.

It includes embedded Tomcat-server so that the applications can be hosted on it.

Spring-boot Presentation

Deployment is very easy, war and jar file can be easily deployed in the tomcat server:

war or jar files can be directly deployed on the Tomcat Server and Spring Boot provides the facility to convert our project into war or jar files. Also, the instance of Tomcat can be run on the cloud as well.

Microservice Based Architecture:

Microservice, as the name suggests is the name given to a module/service which focuses on a single type of feature, exposing an API(application peripheral interface).

Let us consider an example of a hospital management system.

Spring-boot Presentation

In case of monolithic systems, there will be a single code containing all the features which are very tough to maintain on a huge scale.

But in the microservice-based system, each feature can be divided into smaller subsystems like service to handle patient registration, service to handle database management, service to handle billing etc.

Microservice based system can be easily migrated as only some services need to be altered which also makes debugging and deployment easy. Also, each service can be integrated and can be made in different technologies suited to them.

Spring-boot Architecture

Layers in Spring Boot:

There are four main layers in Spring Boot:

Presentation Layer: As the name suggests, it consists of views(i.e. frontend part)

Data Access Layer: CRUD (create, retrieve, update, delete) operations on the database comes under this category.

Service Layer: This consist of service classes and uses services provided by data access layers.

Spring-boot Architecture

Integration Layer: It consists of web different web services(any service available over the internet and uses XML messaging system).

Then we have utility classes, validator classes and view classes.

All the services provided by the classes are implemented in their corresponding classes and are retrieved by implementing the dependency on those interfaces.

Use of Spring-boot

Easy Dependency Management :

In order to speed up the dependency management process, Spring Boot implicitly packages the required compatible third-party dependencies for each type of Spring application and exposes them to the developer using starters.

Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors.

Use of Spring-boot

For example, if you want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go (no need to hunt for compatible database drivers and Hibernate libraries).

Also, if you want to create a Spring web application, just add spring-boot-starter-web dependency, and, by default, this will pull all the commonly used libraries for developing Spring MVC applications such as spring-webmvc, jackson-json, validation-api, and Tomcat.

Use of Spring-boot

In other words, Spring Boot gathers all the common dependencies and defines them in one place and allows the developer to use them instead of reinventing the wheel each time they create a new application.

Therefore, pom.xml becomes much smaller than the one used with traditional Spring applications.

Use of Spring-boot

Auto Configuration :

The second awesome feature of Spring Boot is the auto-configuration.

After you select the appropriate starter, Spring Boot attempts to automatically configure your Spring application based on the jar dependencies that you have added.

For example, if you add spring-boot-starter-web, Spring Boot automatically configures the commonly registered beans like DispatcherServlet, ResourceHandlers, MessageSource.

Use of Spring-boot

If you’re using spring-boot-starter-jdbc, Spring Boot automatically registers the DataSource, EntityManagerFactory, and TransactionManager beans and reads the connection details from the application.properties file.

In case you don't intend to use a database and you don't provide any manual connection details, Spring Boot will auto-configure an in-memory database without any further configuration on your part whenever it finds an H2 or HSQL library on the build path.

This is totally configurable and can be overridden anytime by custom configuration.

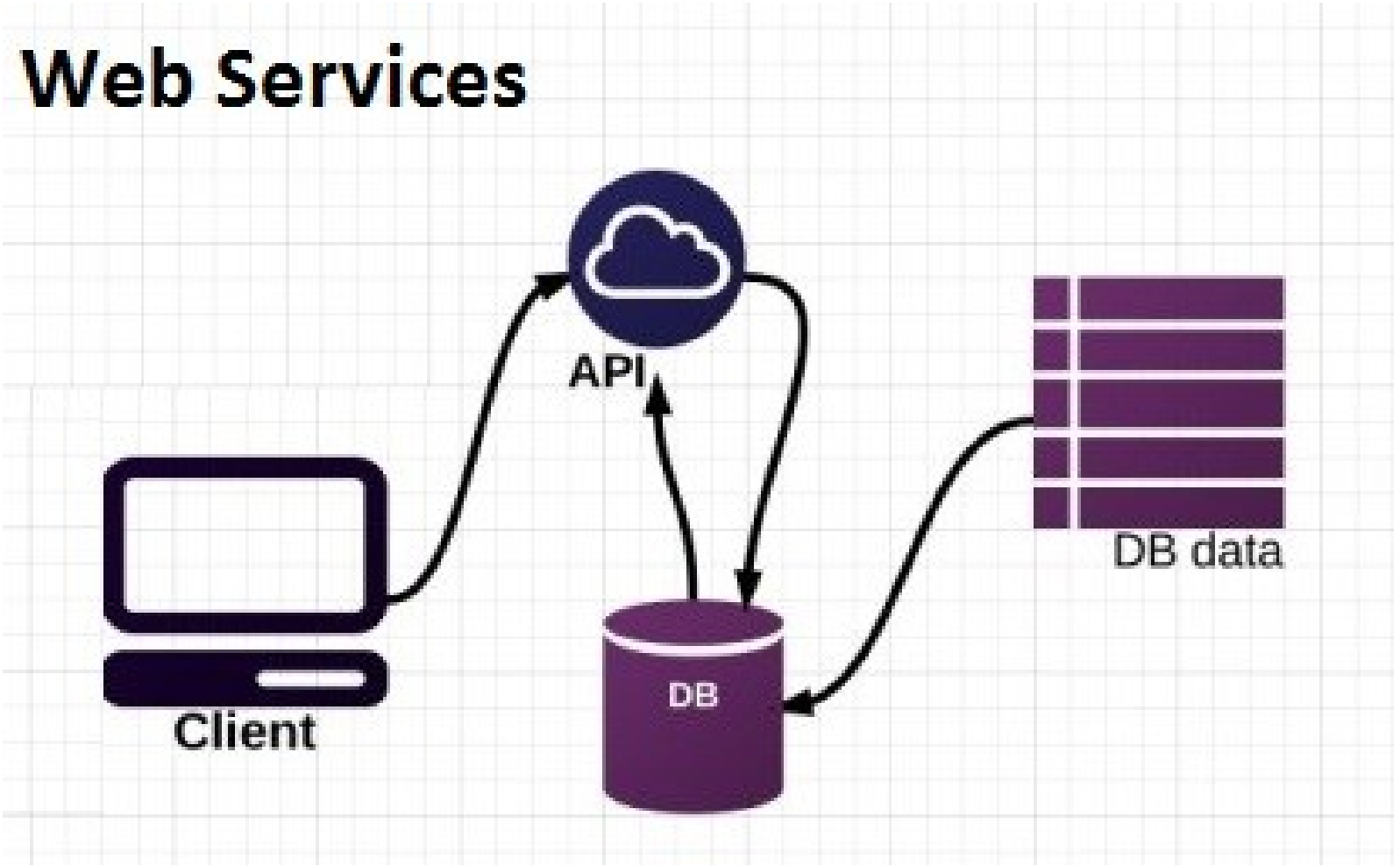
REST API Presentation

Every application on the internet does need to provide interoperability as a basic feature. At any given moment, applications are communicating with other applications (for example, a mobile application communicating with a web application).

It is important that all applications should be able to communicate with other applications without depending on the underlying operating system and the programming languages.

Web services are used to create such applications.

REST API Presentation



REST API : Web Service

A web service is a collection of standards and protocols that applications and systems use for exchanging data over the internet. A web service could be written in any programming language and is OS-independent.

For instance, an application built in PHP running on a Linux server can communicate with an Android application built using Java and running on an Android operating system.

REST API Presentation

REST stands for Representational State Transfer. It is a stateless software architecture that provides many underlying characteristics and protocols that govern the behavior of clients and servers.

What Is a RESTful API?

A RESTful API (also known as a RESTful web service) is a web service implemented using HTTP protocol and the principles of REST. It is a collection of resources that employ HTTP methods (GET, PUT, POST, DELETE).

What benefits?

The web and its core protocol, HTTP, provide a stack of features:

- **Suitable actions (GET, POST, PUT, DELETE, …)**
- **Caching**
- **Redirection and forwarding**
- **Security (encryption and authentication)**

REST API Presentation

These are all critical factors on building resilient services.

But that is not all. The web is built out of lots of tiny specs, hence it’s been able to evolve easily, without getting bogged down in "standards wars".

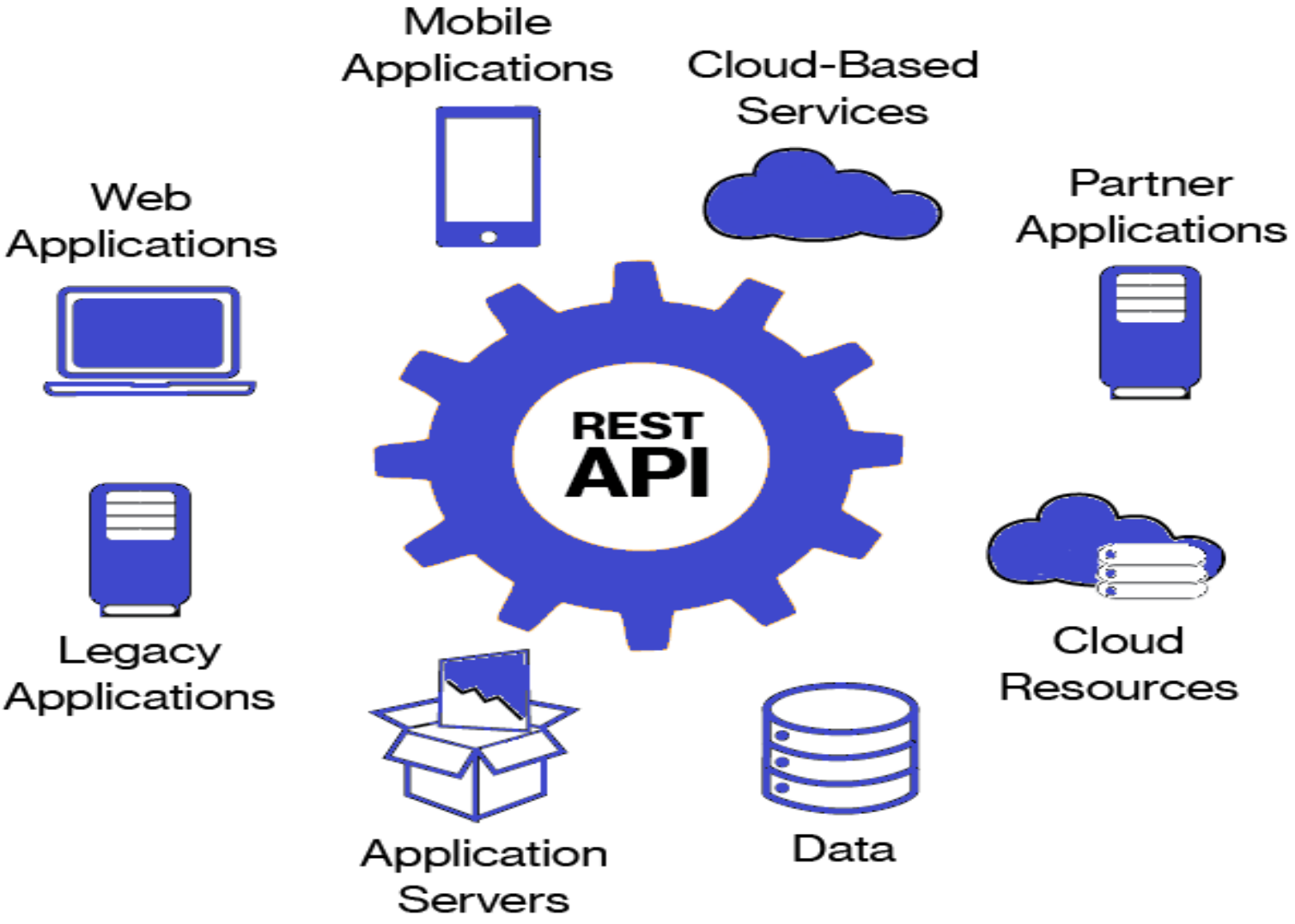
The collection of the resources is then represented in a standardized form (usually XML) that can be any valid Internet media type, provided that it is a valid hypertext standard.

REST API Presentation

Stateless: No data should be stored on the server during the processing of the request transfer. The state of the session should be saved at the client's end.

Cacheable: The client should have the ability to store responses in a cache. This greatly improves the performance of the API.

REST API Presentation



REST API Presentation

Popular RESTful API request formats : REST, XML-RPC, SOAP

Popular RESTful API response formats : REST, XML-RPC, SOAP, JSON, PHP

Why Use a RESTful API?

A RESTful API is used to make applications distributed and independent over the internet with the aim of enhancing the performance, scalability, simplicity, modifiability, visibility, portability, and reliability of the application.

REST Controller

In Spring, a controller class, which is capable of serving REST API requests, is called rest controller. It should be annotated with @RestController annotation.

The resource uris are specified in @RequestMapping annotations. It can be applied at class level and method level both. Complete URI for an API is resolved after adding class level path and method level path.

We should always write produces and consumes attributes to specify the mediatype attributes for the API. Never reply on assumptions.

In given controller, we have two API methods. Feel free to add more methods as needed.

HTTP GET /employees – Returns list of the employees.

HTTP POST /employees – Add an employee in the employees collection.

REST Controller

```
@RestController
@RequestMapping(path = "/employees")
public class EmployeeController
{
   @Autowired
   private EmployeeREpository empRepos;

   @GetMapping(path="/", produces =
"application/json")
   public Employees getEmployees()
   {
      return empRepos.findAll();
   }
```

```
@PostMapping(path= "/", consumes =
"application/json", produces =
"application/json")
   public ResponseEntity<Object>
addEmployee(@RequestBody Employee
employee) {
   Integer id = empRepos.findAll().size() + 1;
    employee.setId(id);
    empRepos.addEmployee(employee);

      URI location =
ServletUriComponentsBuilder.fromCurrentRe
quest().path("/{id}")
 .buildAndExpand(employee.getId()).toUri();

return ResponseEntity.created(location)
                                .build();
   }
}
```

Spring-boot Most used Annotations

Spring Annotations

Spring framework implements and promotes the principle of control inversion (IOC) or dependency injection (DI) and is in fact an IOC container.

Traditionally, Spring allows a developer to manage bean dependencies by using XML-based configuration.

There is an alternative way to define beans and their dependencies. This method is a Java-based configuration.

Unlike the XML approach, Java-based configuration allows you to manage bean components programmatically. That’s why Spring annotations were introduced.

Spring-boot Most used Annotations

@SpringBootApplication

Our REST APIs skeleton is ready. Now we need to configure Spring to detect our rest controller (using auto scanning) and deploy apis in embedded tomcat server. Thankfully, Spring boot makes all these things very easy by using the concept of auto configuration.

Auto-configuration attempts to guess and configure beans we you are likely to need. Auto-configuration classes are usually applied based on the jars in application classpath and the beans we have defined additionally in @Configuration classes.

In this case, it does following things.

It detects spring-webmvc so configure default spring mvc application beans. It help in scan and configure @RestController and similar annotations.

It detects embed tomcat jars so configure embedded tomcat for us.

It detects JSON jars so configure JSON support to APIs.

Spring-boot Most used Annotations

@Controller

@RequestMapping

@PathVariable

@RequestParam

@ModelAttribute

@RequestBody and @ResponseBody

@RequestHeader and @ResponseHeader

Spring-boot Most used Annotations

@Bean: Indicates that a method produces a bean to be managed by the Spring container. This is one of the most used and important spring annotation.

@Bean annotation also can be used with parameters like name, initMethod and destroyMethod.

name – allows you give name for bean

initMethod – allows you to choose method which will be invoked on context register

destroyMethod – allows you to choose method which will be invoked on context shutdown

Spring-boot Most used Annotations

@ComponentScan: Configures component scanning directives for use with @Configuration classes. Here we can specify the base packages to scan for spring components.

@Component: Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.

Spring-boot Most used Annotations

@PropertySource: Provides a simple declarative mechanism for adding a property source to Spring's Environment. There is a similar annotation for adding an array of property source files i.e @PropertySources.

@Service: Indicates that an annotated class is a "Service". This annotation serves as a specialization of @Component, allowing for implementation classes to be autodetected through classpath scanning.

@Repository: Indicates that an annotated class is a "Repository". This annotation serves as a specialization of @Component and advisable to use with DAO classes.

@Configuration Indicates that a class is a configuration class that may contain bean definitions.

Spring-boot Most used Annotations

@RequestMapping - maps HTTP request with a path to a controller method.

@SpringBootApplication - enables Spring Boot auto-configuration and component scanning.

@Component is a generic stereotype for a Spring managed component. It turns the class into a Spring bean at the auto-scan time. Classes decorated with this annotation are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning. @Repository, @Service, and @Controller are specializations of @Component for more specific use cases.

There are also Hibernate @Entity, @Table, @Id, and @GeneratedValue annotations in the example.

Spring-boot Most used Annotations

@EnableAutoConfiguration, as its name says, enables auto-configuration. It means that Spring Boot looks for auto-configuration beans on its classpath and automatically applies them.

Note, that we have to use this annotation with @Configuration:

```
@Configuration
@EnableAutoConfiguration
class VehicleFactoryConfig {}
```

Usually, when we write our custom auto-configurations, we want Spring to use them conditionally. We can achieve this with the annotations in this section.

We can place the annotations in this section on @Configuration classes or @Bean methods.

Spring-boot Most used Annotations

@Required

This annotation is applied on bean setter methods. Consider a scenario where you need to enforce a required property. The @Required annotation indicates that the affected bean must be populated at configuration time with the required property. Otherwise an exception of type BeanInitializationException is thrown.

@Autowired

This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.

When you use @Autowired on fields and pass the values for the fields using the property name, Spring will automatically assign the fields with the passed values.

Spring-boot Most used Annotations

@RequestMapping

This annotation is used both at class and method level. The @RequestMapping annotation is used to map web requests onto specific handler classes and handler methods. When @RequestMapping is used on class level it creates a base URI for which the controller will be used. When this annotation is used on methods it will give you the URI on which the handler methods will be executed. From this you can infer that the class level request mapping will remain the same whereas each handler method will have their own request mapping.

Sometimes you may want to perform different operations based on the HTTP method used, even though the request URI may remain the same. In such situations, you can use the method attribute of @RequestMapping with an HTTP method value to narrow down the HTTP methods in order to invoke the methods of your class.

Spring-boot Most used Annotations

Here is a basic example on how a controller along with request mappings work:

```
@Controller
@RequestMapping("/welcome")
public class WelcomeController{
  @RequestMapping(method = RequestMethod.GET)
  public String welcomeAll(){
    return "welcome all";
  }
}
```

In this example only GET requests to /welcome is handled by the welcomeAll() method.

This annotation also can be used with Spring MVC and Spring WebFlux.

The @RequestMapping annotation is very versatile.

Spring-boot Most used Annotations

Composed @RequestMapping Variants

Spring framework 4.3 introduced the following method-level variants of @RequestMapping annotation to better express the semantics of the annotated methods. Using these annotations have become the standard ways of defining the endpoints. They act as wrapper to @RequestMapping.

These annotations can be used with Spring MVC and Spring WebFlux.

@GetMapping

This annotation is used for mapping HTTP GET requests onto specific handler methods. @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET)

@PostMapping

This annotation is used for mapping HTTP POST requests onto specific handler methods. @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST)

Spring-boot Most used Annotations

@PutMapping

This annotation is used for mapping HTTP PUT requests onto specific handler methods. @PutMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT)

@PatchMapping

This annotation is used for mapping HTTP PATCH requests onto specific handler methods. @PatchMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PATCH)

@DeleteMapping

This annotation is used for mapping HTTP DELETE requests onto specific handler methods. @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE)

@ExceptionHandler

This annotation is used at method levels to handle exception at the controller level. The @ExceptionHandler annotation is used to define the class of exception it will catch. You can use this annotation on methods that should be invoked to handle an exception. The @ExceptionHandler values can be set to an array of Exception types. If an exception is thrown that matches one of the types in the list, then the method annotated with matching @ExceptionHandler will be invoked.

@InitBinder

This annotation is a method level annotation that plays the role of identifying the methods which initialize the WebDataBinder - a DataBinder that binds the request parameter to JavaBean objects. To customise request parameter data binding , you can use @InitBinder annotated methods within our controller. The methods annotated with @InitBinder all argument types that handler methods support.

The @InitBinder annotated methods will get called for each HTTP request if you don’t specify the value element of this annotation. The value element can be a single or multiple form names or request parameters that the init binder method is applied to.

@Mappings and @Mapping

This annotation is used on fields. The @Mapping annotation is a meta annotation that indicates a web mapping annotation. When mapping different field names, you need to configure the source field to its target field and to do that you have to add the @Mappings annotation. This annotation accepts an array of @Mapping having the source and the target fields.

@MatrixVariable

This annotation is used to annotate request handler method arguments so that Spring can inject the relevant bits of matrix URI. Matrix variables can appear on any segment each separated by a semicolon. If a URL contains matrix variables, the request mapping pattern must represent them with a URI template. The @MatrixVariable annotation ensures that the request is matched with the correct matrix variables of the URI.

Spring-boot Most used Annotations

@PathVariable

This annotation is used to annotate request handler method arguments. The @RequestMapping annotation can be used to handle dynamic changes in the URI where certain URI value acts as a parameter. You can specify this parameter using a regular expression.

The @PathVariable annotation can be used declare this parameter.

@RequestAttribute

This annotation is used to bind the request attribute to a handler method parameter. Spring retrieves the named attributes value to populate the parameter annotated with @RequestAttribute.

While the @RequestParam annotation is used bind the parameter values from query string, the @RequestAttribute is used to access the objects which have been populated on the server side.

Spring-boot Most used Annotations

@RequestBody

This annotation is used to annotate request handler method arguments. The @RequestBody annotation indicates that a method parameter should be bound to the value of the HTTP request body.

The HttpMessageConveter is responsible for converting from the HTTP request message to object.

@RequestHeader

This annotation is used to annotate request handler method arguments. The @RequestHeader annotation is used to map controller parameter to request header value. When Spring maps the request, @RequestHeader checks the header with the name specified within the annotation and binds its value to the handler method parameter.

This annotation helps you to get the header details within the controller class.

Spring-boot Most used Annotations

@ResponseStatus

This annotation is used on methods and exception classes. @ResponseStatus marks a method or exception class with a status code and a reason that must be returned. When the handler method is invoked the status code is set to the HTTP response which overrides the status information provided by any other means. A controller class can also be annotated with @ResponseStatus which is then inherited by all @RequestMapping methods.

@ControllerAdvice

This annotation is applied at the class level. As explained earlier, for each controller you can use @ExceptionHandler on a method that will be called when a given exception occurs. But this handles only those exception that occur within the controller in which it is defined. To overcome this problem you can now use the @ControllerAdvice annotation. This annotation is used to define @ExceptionHandler, @InitBinder and @ModelAttribute methods that apply to all @RequestMapping methods. Thus if you define the @ExceptionHandler annotation on a method in @ControllerAdvice class, it will be applied to all the controllers.

Spring-boot Most used Annotations

@RestController

This annotation is used at the class level.

The @RestController annotation marks the class as a controller where every method returns a domain object instead of a view.

By annotating a class with this annotation you no longer need to add @ResponseBody to all the RequestMapping method. It means that you no more use view-resolvers or send html in response.

You just send the domain object as HTTP response in the format that is understood by the consumers like JSON.

@RestController is a convenience annotation which combines @Controller and @ResponseBody.

Spring-boot Most used Annotations

@RestControllerAdvice

This annotation is applied on Java classes. @RestControllerAdvice is a convenience annotation which combines @ControllerAdvice and @ResponseBody. This annotation is used along with the @ExceptionHandler annotation to handle exceptions that occur within the controller.

@SessionAttribute

This annotation is used at method parameter level. The @SessionAttribute annotation is used to bind the method parameter to a session attribute. This annotation provides a convenient access to the existing or permanent session attributes.

Spring-boot Most used Annotations

@SessionAttributes

This annotation is applied at type level for a specific handler. The @SessionAtrributes annotation is used when you want to add a JavaBean object into a session. This is used when you want to keep the object in session for short lived. @SessionAttributes is used in conjunction with @ModelAttribute.

Consider this example.

```
@ModelAttribute("person")
public Person getPerson(){}
// within the same controller as above snippet
@Controller
@SeesionAttributes(value="person", types={Person.class})
public class PersonController{}
```

The @ModelAttribute name is assigned to the @SessionAttributes as value. The @SessionAttributes has two elements. The value element is the name of the session in the model and the types element is the type of session attributes in the model.

Http Status Codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

Informational responses (100–199),

Successful responses (200–299),

Redirects (300–399),

Client errors (400–499),

and Server errors (500–599).

Http Status Codes : Successful responses

200 OK

The request has succeeded. The meaning of the success depends on the HTTP method:

GET: The resource has been fetched and is transmitted in the message body.

HEAD: The entity headers are in the message body.

PUT or POST: The resource describing the result of the action is transmitted in the message body.

TRACE: The message body contains the request message as received by the server

201 Created

The request has succeeded and a new resource has been created as a result. This is typically the response sent after POST requests, or some PUT requests.

202 Accepted

The request has been received but not yet acted upon. It is noncommittal, since there is no way in HTTP to later send an asynchronous response indicating the outcome of the request. It is intended for cases where another process or server handles the request, or for batch processing.

Http Status Codes

203 Non-Authoritative Information

This response code means the returned meta-information is not exactly the same as is available from the origin server, but is collected from a local or a third-party copy. This is mostly used for mirrors or backups of another resource. Except for that specific case, the "200 OK" response is preferred to this status.

204 No Content

There is no content to send for this request, but the headers may be useful. The user-agent may update its cached headers for this resource with the new ones.

205 Reset Content

Tells the user-agent to reset the document which sent this request.

206 Partial Content

This response code is used when the Range header is sent from the client to request only part of a resource.

Http Status Codes : Client error responses

400 Bad Request

The server could not understand the request due to invalid syntax.

401 Unauthorized

Although the HTTP standard specifies "unauthorized", semantically this response means "unauthenticated". That is, the client must authenticate itself to get the requested response.

402 Payment Required

This response code is reserved for future use. The initial aim for creating this code was using it for digital payment systems, however this status code is used very rarely and no standard convention exists.

403 Forbidden

The client does not have access rights to the content; that is, it is unauthorized, so the server is refusing to give the requested resource. Unlike 401, the client's identity is known to the server.

404 Not Found

The server can not find requested resource. In the browser, this means the URL is not recognized. In an API, this can also mean that the endpoint is valid but the resource itself does not exist. Servers may also send this response instead of 403 to hide the existence of a resource from an unauthorized client. This response code is probably the most famous one due to its frequent occurrence on the web.

Http Status Codes

405 Method Not Allowed

The request method is known by the server but has been disabled and cannot be used. For example, an API may forbid DELETE-ing a resource. The two mandatory methods, GET and HEAD, must never be disabled and should not return this error code.

406 Not Acceptable

This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content that conforms to the criteria given by the user agent.

407 Proxy Authentication Required

This is similar to 401 but authentication is needed to be done by a proxy.

408 Request Timeout

This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome, Firefox 27+, or IE9, use HTTP pre-connection mechanisms to speed up surfing. Also note that some servers merely shut down the connection without sending this message.

409 Conflict

This response is sent when a request conflicts with the current state of the server.

Http Status Codes

410 Gone

This response is sent when the requested content has been permanently deleted from server, with no forwarding address. Clients are expected to remove their caches and links to the resource. The HTTP specification intends this status code to be used for "limited-time, promotional services". APIs should not feel compelled to indicate resources that have been deleted with this status code.

411 Length Required

Server rejected the request because the Content-Length header field is not defined and the server requires it.

412 Precondition Failed

The client has indicated preconditions in its headers which the server does not meet.

413 Payload Too Large

Request entity is larger than limits defined by server; the server might close the connection or return an Retry-After header field.

414 URI Too Long

The URI requested by the client is longer than the server is willing to interpret.

Http Status Codes

415 Unsupported Media Type

The media format of the requested data is not supported by the server, so the server is rejecting the request.

416 Requested Range Not Satisfiable

The range specified by the Range header field in the request can't be fulfilled; it's possible that the range is outside the size of the target URI's data.

417 Expectation Failed

This response code means the expectation indicated by the Expect request header field can't be met by the server.

Http Status Codes

418 I'm a teapot

The server refuses the attempt to brew coffee with a teapot.

421 Misdirected Request

The request was directed at a server that is not able to produce a response. This can be sent by a server that is not configured to produce responses for the combination of scheme and authority that are included in the request URI.

422 Unprocessable Entity (WebDAV)

The request was well-formed but was unable to be followed due to semantic errors.

Example : Competition Example

For more details see the full example source code on Githib.

The link is below :

Competition API example

Example : Product Example

For more details see the full example source code on Githib.

The link is below :

Product Management Example

Example : Toy Example

For more details see the full example source code on Githib.

The link is below :

https://github.com/ChaoukiBayoudhi/toy-store-server