Spring Boot

Spring Boot is a project that is built on the top of the Spring Framework. It provides an easier and faster way to set up, configure, and run both simple and web-based applications. **Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run**”

### Problems with Spring

If you have ever written a Spring application, you will know that it needs a lot of configuration. So suppose you are creating a web-based application via Spring that also accesses a database. You will need to use Spring MVC and Spring Data JPA.  So in order to create such an application via Maven, you will typically need to do the following:

1. Create a Maven project and add all the dependencies for  the necessary components like Spring MVC, Spring Data JPA, etc within the pom file
2. Do configuration for MVC support like setting up ViewResolver, DispatcherServlet, component scan
3. Do configuration for database support like setting up DataSource, TransactionManager, EntityManager, etc
4. Add all the properties like Database URL, etc within the properties file
5. Finally, write the code

So you can see that you need to do a lot of configuration in order to set up an application via Spring.

So in order to develop a SpringBoot application via Maven, you just need to do the following:

1. Add the starter packages for the type of application you are developing to the Maven pom file
2. Add all the properties like Database URL, etc within the properties file
3. Writer code

So compared with the Spring approach, this approach is much straightforward.

Advantages or features

1 Rapid application development

How all jars are available?

By starter dependency

We mention spring-boot-starter-web this will include all other jars like (spring-web mvc,spring-web,spring-boot-tomcat,spring-boot-starter-json)

So how our requests are handled we have dispatcher servlet which will be taking all our requests by default bcz by autoconfiguration dispatcherservletautoconfiguration. Request with **/** will be taken and based on that will be map to the respective controllers

How does Bean i.e Java object is converted into respective Json in output ?

When we mention @RestController it has @ResponseBody included.

By ResponseBody+JacksonHttpMessageConverters it is possible

spring-boot-starter-json will be present in spring-boot-starter-web and by autoconfiguration JacksonHttpMessageConvertersConfiguration it will convert

who is configuring error mapping?

When we enter url which is not present in our application then whitelabbel error page will be displayed that this request is not present. So by ErrorMVCAutoConfiguration it is done the information displayed is present in code by that.

2 If you are working with Java applications with Spring you need to do lot of configuration like if you use hibernate you need to do lot of configuration. So, spring boot manages dependecies like it groups all dependies as starter dependencies if you need to work jpa there will be Spring boot starter jpa. For unit testing etc.. So it also uses Auto configuration. All the configuration for hibernate is not needed as spring boot provied auto config use just hibernate starter

Spring boot starter web

This tells SpringBoot that you are developing a web application, so it will add the necessary jar files. It will also include the embedded Tomcat server by default.

Spring boot starter JPA

This tells SpringBoot that you are developing a database application, so it will add the necessary jar files.

3 **Auto-configuration**

Based on the dependencies in the classpath, SpringBoot tries to configure your application by providing suitable defaults. This is known as auto-configuration. So suppose you are building a web application and add the starter for it to the Maven pom file. SpringBoot automatically configures the DispatcherServlet, ViewResolver, etc. Or if you have added the spring-data-jpa-starter, SpringBoot will automatically configure the necessary beans like the TransactionManager, EntityManager, etc

4 Embedded Server (Tomcat) Use Jar Don’t WAR. In normal spring we use Tomcat seperately to deploy applications by creating war files and you deploy those war files into any servers like tomcat. So here it is not required we have Jar file, will be created and jar directly we can use . So the jar files for the server are included within your web application. This allows you to run your web application as a normal Java application.

**Difference between WAR & JAR**

Their **purpose and the way they function**. JAR files allow us to package multiple files in order to use it as a library, plugin, or any kind of application. In other words A file that encapsulates one or more Java classes, a manifest, and descriptor is called JAR file.

On the other hand, WAR files are used only for web applications. **WAR** stands for **Web Archive.**  Web module contains servlet classes, JSP files, HTML files, JavaScripts, etc. are packaged as a JAR file with .**war** extension. It contains a special directory called **WEB-INF**.

WAR is a module that loads into a web container of the Java Application Server

**The structure of the archives is also different.** We can create a JAR with any desired structure. In contrast, WAR has a predefined structure with WEB-INF and META-INF directories.

Finally, we can **run a JAR from the command line** if we build it as an [executable JAR](https://www.baeldung.com/executable-jar-with-maven) without using additional software. Or, we can use it as a library. In contrast, we **need a server to execute a WAR**.

5 It helps in reducing all the manual work of writing annotations, boilerplate codes and XML configurations.

6 It has inmemory database

7 It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

|  |  |  |
| --- | --- | --- |
| S.No. | **Spring VERSUS** | **Spring Boot** |
| 1. | Spring is an open-source lightweight framework widely used to develop enterpriseapplications. | Spring Boot is built on top of the conventional spring framework, widely used to develop REST APIs. It is a module of Spring |
| 2. | The most important feature of the Spring Framework is dependency injection. | The most important feature of the Spring Boot is Autoconfiguration. |
| 3. | It helps to create a loosely coupled application. | It helps to create a stand-alone application. |
| 4. | To run the Spring application, we need to set the server explicitly. | Spring Boot provides embedded servers such as Tomcat and Jetty etc. |
| 5. | To run the Spring application, a deployment descriptor is required. | There is no requirement for a deployment descriptor. |
| 6. | To create a Spring application, the developers write lots of code. | It reduces the lines of code. |
| 7. | It doesn’t provide support for the in-memory database. | It provides support for the in-memory database such as H2. |
| 8. | Developers need to write boilerplate code for smaller tasks. | In Spring Boot, there is reduction in boilerplate code. |
| 9. | Developers have to define dependencies manually in the  pom.xml file. | pom.xml file internally handles the required dependencies. |

As Spring boot provides autoconfiguration, you can also modify those configuration b y application.properties like servert.port =8082 by defulat spring will be configured port as 8080

# **Spring Boot Annotations:**

Annotations are a form of metadata that provides data about the program. They are not part of the program itself. Annotations do not have a direct effect on the operation of the code they annotate.

While we create springboot application there will be app.java class and on top of that there will be an annotation @SpringBootApplication. If we open that annotation you can see like there will be @EnableAutoConfiguration,@ComponentScan, @Configuration etc..

@SpringBootApplication is used to annotate the **main class** of the spring boot application. The SpringApplication.run() method is mandatory because it helps to initiate the Spring framework.

The @SpringBootApplication annotation is a convenience annotation that combines the **@EnableAutoConfiguration**, **@Configuration** and the **@ComponentScan** annotations in a [Spring Boot](https://click.linksynergy.com/deeplink?id=MnzIZAZNE5Y&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fspringbootfundamentals%2F) application. These annotations do the following:

Before understanding these three annotations, let us first know about @Bean. A **Spring IoC container** is at the core of the spring framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. These objects are known as **Spring Beans**.

**@Bean**annotation signifies that the given method creates a bean which is managed by the Spring Container. **@Configuration** annotation indicates that a class declares one or more **@Bean** methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime.

**@Configuration** – Designates the class as a configuration class for [Java configuration](https://learnjava.co.in/how-spring-works-under-the-hood/#Configuration_metadata). In addition to beans configured via component scanning, an application may desire to configure some additional beans via the @Bean annotation as demonstrated [here](https://learnjava.co.in/spring-java-configuration-example/). Thus, the return value of methods having the @Bean annotation in this class are registered as beans.

**@ComponentScan** – Typically, in a [Spring](https://click.linksynergy.com/deeplink?id=MnzIZAZNE5Y&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fspring-tutorial-for-beginners%2F) application, annotations like @Component, @Configuration, @Service, @Repository are specified on classes to mark them as Spring beans. The @ComponentScan annotation basically tells Spring Boot to scan the current package and its sub-packages in order to identify annotated classes and configure them as Spring beans. Thus, it designates the current package as the root package for component scanning.

**@EnableAutoConfiguration** – This enables Spring Boot’s [autoconfiguration mechanism](https://learnjava.co.in/springboot-what-and-why/#Autoconfiguration). Auto-configuration refers to creating beans automatically by scanning the classpath. It tries to configure automatically your Spring application based upon the JAR dependency you have added in the classpath.

The @EnableAutoConfiguration annotation enables Spring Boot to auto-configure the application context. Therefore, it automatically creates and registers beans based on both the included jar files in the classpath and the beans defined by us.

When we define the spring-boot-starter-web dependency in our classpath/pom, Spring boot auto-configures Tomcat and Spring MVC. However, this auto-configuration has less precedence in case we define our own configurations.

**@Controller**

By using this annotation we do two things, first, (i.e inside this annotation @ Component is included)we declare that this class is a Spring bean and should be created and maintained by SpringApplicationContext, second it indicates that it’s a controller in MVC setup. When a request is made, this annotation tells the dispatcher servlet to look for the required endpoint request path in the components marked with @Controller.

*@Controller is a @Component (just like @Service, @Repository, @Endpoint etc.)*

**@Service**

Annotates classes at the service layer. Service layer holds your business logic which is called from the controller layer. @Component is also included inside this

**@Repository**

Annotates classes at the persistence layer, which will act as a database repository. The class provides the mechanism for storage, retrieval, update, delete and search operation on objects by extending Jpa Repository. This annotation is a general-purpose stereotype annotation which very close to the [DAO pattern](https://www.geeksforgeeks.org/data-access-object-pattern/) where DAO classes are responsible for providing CRUD operations on database tables.

**@RequestMapping** annotation maps HTTP requests to handler methods of MVC and REST controllers. This annotation can be used both at the class and at the method level. For example, the syntax for @RequestMapping for GET method is:

@RequestMapping(value = “/get/{id}”, method = RequestMethod.GET)

and, it is made simpler by using @GetMapping:

@GetMapping("/get/{id}")

Similarly we can use post,delete,put mappings

When we are working with rest application we need to use @RestController rather than @Contoller

Bcz in normal Controller it tells it is a spring componenet and controller but in RestController it tells above things and also it returns @ResponseBody. i.e it will return some data in which it will not require jsp or some other thing

spring-boot-devtools dependency

We do lot of changes in our application, so to start our server and stop our server it will take lot of time . so by adding this it will reload automatically. In Intellij some settings needs to be changed after adding this dependency. It saves lot of time.

Adding config in properties file so that we can change values in future if needed

In application.properties

welcome.message = This is just to print message

In Controller class

@Value(“${welcome.message}”)

Private String welcomeMessage;

**Yaml file**

Spring in default provides application.properties but we can use yaml file

application.yaml or application.yml(yet another markup language)

yaml files is more human readable file, It reduces duplicate values. It follows hirearchichal model. Indentation should be correct.

Ex:In app.properties

spring.datasource.url=jdbc:h2:mem:testdb spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=sa spring.datasource.password=password

**In yaml file**

spring:

datasource:

url : jdbc:h2:mem:testdb

driverClassName:org.h2.Driver

username:sa

hibernate:

show-sql: true

There are different plugins also available to converts app.prop to yaml in intellij also

Spring Profiles

The development process of an application has different stages; the typical ones are development, testing, and production. Spring Boot profiles group parts of the application configuration and make it be available only in certain environments.

A *profile* is a set of configuration settings. Spring Boot allows to define profile specific property files in the form of application-{profile}.properties. It automatically loads the properties in an application.properties file for all profiles, and the ones in profile-specific property files only for the specified profile. The keys in the profile-specific property override the ones in the master property file.

We can define profiles active = dev or qa and deploy into specific environment

-Dspring.profiles.active=dev

These have to be named in the format application-{profile}.properties.

We can make different files like application-dev.properties, application-qa.properties

**Spring Boot Actuator**

Spring Actuator is a cool feature of Spring Boot with the help of which you can see what is happening inside a running application.

Provides spring boots production ready features

Monitor and manage your application in your production

Dependency to add is spring-boot-starter-actuator

Provides number of endpoints like

Beans,health,metrics,mappings, etc…

Some important are

By default when you send actuator request only health endpoint will be shown

In health status will be there UP status: UP

So to see other endpoints management.endpoints.web.exposure.include= \*  
In beans this will provide all beans that are loaded into application context

Like controller class is a bean and tells scope of bean like singleton. It tells package and dependencies like if one object is dependent on other it will show that.

In env it will show which environment our app is running and port number. Java version and other details.

In metrics you can see application start time, how many request have been sent and which different uris has been hit & http status returned

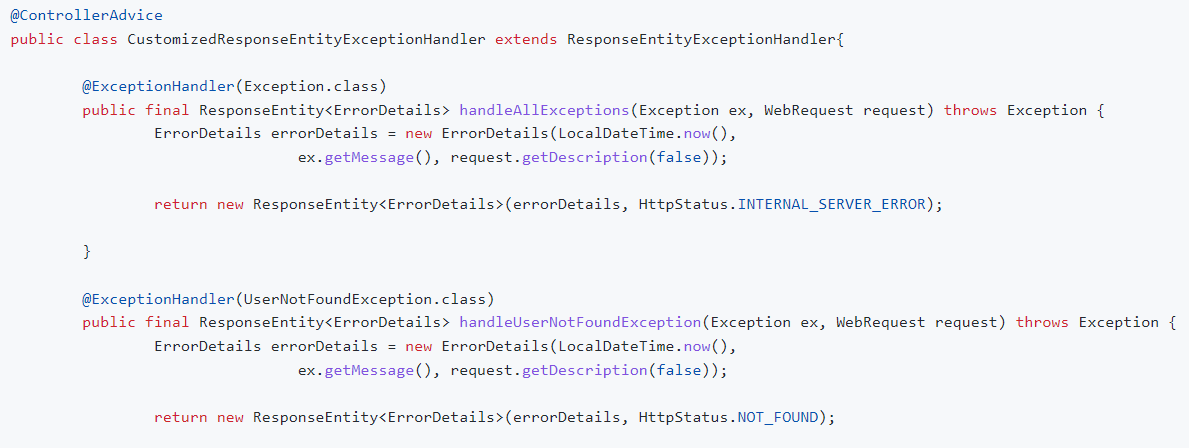
In mappings we can see all different request methods we can see & headers etc..

In other endpoints we can also see loggers, caching etc..

Exceptional Handling

In Spring Boot we can customize the response when excpetion occurs like the required fields in response body and response status to return. So that end user can easily understand and we need to send correct status codes

There is a class ResponseEntityExceptionHandler. This class has @ExceptionHandler method that handles all Spring Mvc raised exceptions by returning a response entity formatted error details in the body. We need to create a class and extend this class to override a method handleException. By @ExceptionHandler we can define for which type of exceptions we need to handle. @ControllerAdvice will tell this class is a component and also allows handling exceptions across the whole application in one global handling component. ErrorDetails below is the class we defined to customize fields.



Validation in SpringBoot

Its important to validate request bcz say you give empty filed name: “” it will take and also future date or wrong email format

We need to add spring-boot-starter-validation dependency

@Valid in method Whenever binding happen then validation which we defined on that particular object will happen.

saveStudent(@Valid @RequestBody User user)

@Size(min=2,max=15,message=”name size should be between 2 to 15”)

private String name;

@Past(message=”birthDate should be past”)

private String birthDate;

we can customize exception as below

@Override

protected ResponseEntity<Object> handleMethodArgumentNotValid(

MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatusCode status, WebRequest request) {

ErrorDetails errorDetails = new ErrorDetails(LocalDateTime.now(),

ex.getFieldError().getDefaultMessage(), request.getDescription(false));

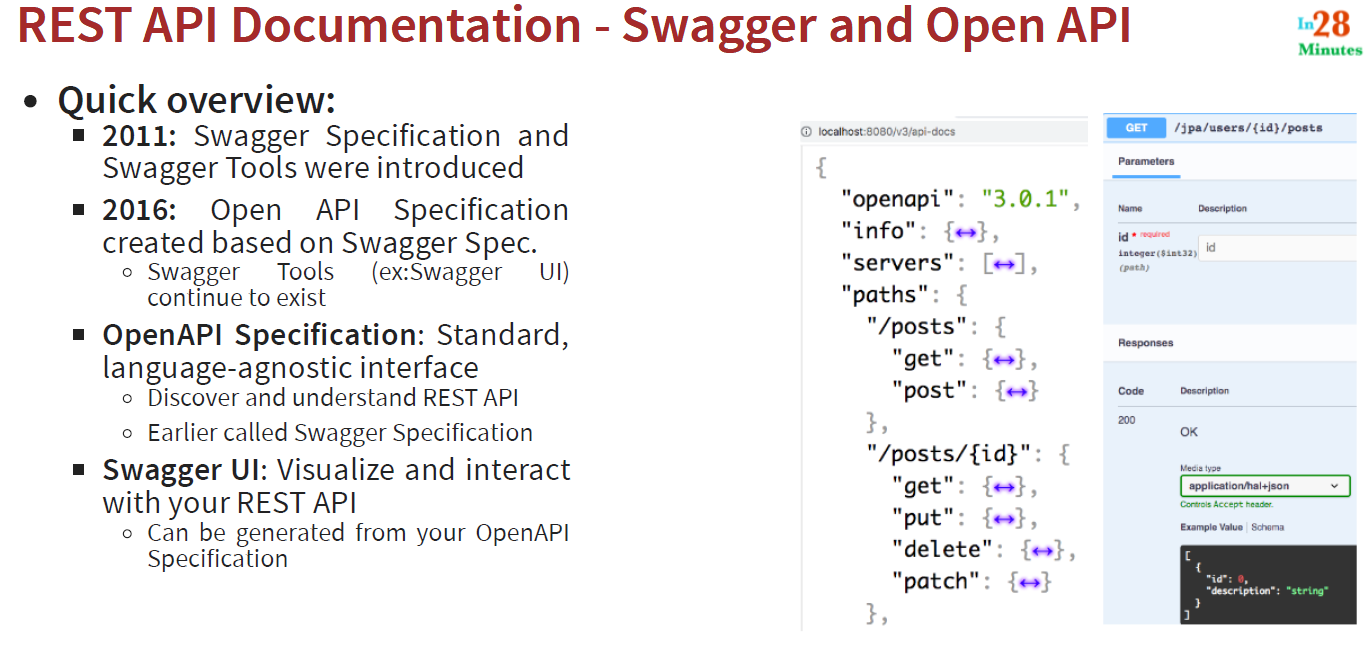
return new ResponseEntity(errorDetails, HttpStatus.BAD\_REQUEST);

}

These are different Validation annotations we can use



**Advanced Rest API Features**



Content Negotiation

How user negotiate with the content he want . Say user want respone in the form of Json or Xml or else he want with different language like english,dutch etc

With Same Resource( same url) different content types or different languages based response we can achieve

For Xml response we just need to add dependency

While hitting url specify Accept Header (MIME Types-applicaton/xml or application/json you need)

Internationalization(i18n) 18 means 18 characters

To get response based on required language we use i18n

In Http Request Header pass required language as below

Accept-Language Header(en,nl,fr,…)

Coding:

We need to create files like message.properties, message\_nl.properties and define specific language message there like good.morning.message= Goedemorgen

A **Locale object represents a specific geographical, political, or cultural region**. It will be taken from header which we pass like en,nl etc..

Message Source helps to define such as environment-specific configuration, internationalization or configurable values.

@RestController

public class HelloWorldController {

private MessageSource messageSource;

public HelloWorldController(MessageSource messageSource) {

this.messageSource = messageSource;

}

@GetMapping(path = "/hello-world-internationalized")

public String helloWorldInternationalized() {

Locale locale = LocaleContextHolder.getLocale();

return messageSource.getMessage("good.morning.message", null, "Default Message", locale );

// - Example: `en` - English (Good Morning)

// - Example: `nl` - Dutch (Goedemorgen)

// - Example: `fr` - French (Bonjour)

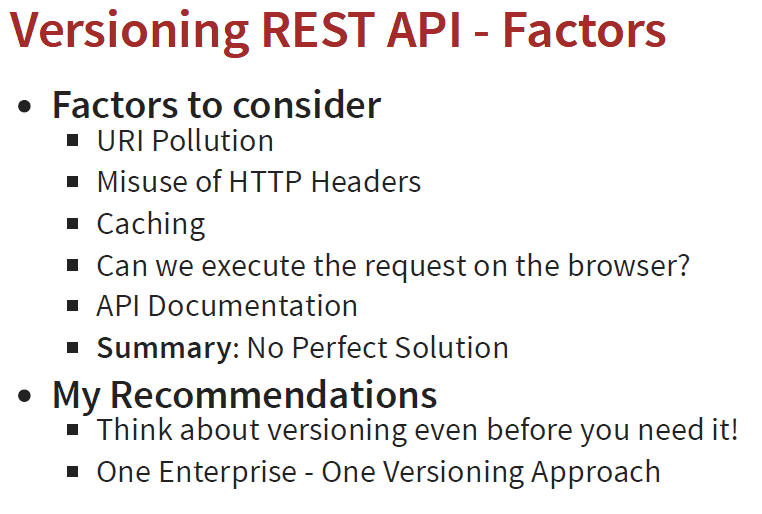
// - Example: `de` - Deutsch (Guten Morgen)

}

**Versioning**

When you need to make changes in your application the existing thing will be used by end used so better to not break that one so we need to create new version





Spring Data

**Spring Data** is a project driven by Spring that aims at providing a consistent data access layer for various data stores, right from relational to NoSQL databases.  **Spring Data is NOT a Specification or Standard, but it is an Abstraction**. It is an umbrella project with many modules (like spring-data-redis, spring-data-mongo, etc.) and a core spring-data-commons module.

### What is the name of the default H2 database configured by Spring Boot?

The name of the default H2 database is **testdb.  Refer below:**

spring.datasource.name=testdb # Name of the datasource.

spring-boot-data-jpa-starter has jpa+jdbc+hibernate+aspect+transaction

Unit Testing

@SpringBootTest

@SpringBootTest by default starts searching in the current package of the test class and then searches upwards through the package structure, looking for a class annotated with @SpringBootConfiguration from which it then reads the configuration to create an application context. This class is usually our main application class since the @SpringBootApplication annotation includes the @SpringBootConfiguration annotation. It then creates an application context very similar to the one that would be started in a production environment.

@MockBean