Spring Boot

**1.What is Spring Boot?**  
First of all Spring Boot is not a framework, it is a way to ease to create stand-alone application with minimal or zero configurations. It is approach to develop spring based application with very less configuration. It provides defaults for code and annotation configuration to quick start new spring projects within no time. It leverages existing spring projects as well as Third party projects to develop production ready applications. It provides a set of Starter Pom’s or gradle build files which one can use to add required dependencies and also facilitate auto configuration.

Spring Boot automatically configures required classes depending on the libraries on its classpath. Suppose your application want to interact with DB, if there are Spring Data libraries on class path then it automatically sets up connection to DB along with the Data Source class.

**2. What are the advantages of using Spring Boot?**

* It is very easy to develop Spring Based applications with Java or Groovy.
* It reduces lots of development time and increases productivity.
* It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
* It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.
* It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort
* It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.
* It provides CLI (Command Line Interface) tool to develop and test Spring Boot (Java or Groovy) Applications from command prompt very easily and quickly.
* It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle
* It provides lots of plugins to work with embedded and in-memory Databases very easily.

**3. What are the disadvantages of using Spring Boot?**  
It is very tough and time consuming process to convert existing or legacy Spring Framework projects into Spring Boot Applications. It is applicable only for brand new/Greenfield Spring Projects.

**4. Why is it “opinionated”?**  
It follows “Opinionated Defaults Configuration” Approach to reduce Developer effort. Due to opinionated view of spring boot, what is required to get started but also we can get out if not suitable for application.  
• Spring Boot uses sensible defaults, “opinions”, mostly based on the classpath contents.  
• For example  
– Sets up a JPA Entity Manager Factory if a JPA implementation is on the classpath.  
– Creates a default Spring MVC setup, if Spring MVC is on the classpath.  
• Everything can be overridden easily  
– But most of the time not needed

**5. How does it work? How does it know what to configure?**  
• Auto-configuration works by analyzing the classpath  
– If you forget a dependency, Spring Boot can’t configure it  
– A dependency management tool is recommended  
– Spring Boot parent and starters make it much easier  
• Spring Boot works with Maven, Gradle, Ant/Ivy  
– Our content here will show Maven

**6. How are properties defined? Where?**

In spring boot, we have to define properties in the application.properties file exists in classpath of application as follow.  
Example: configure default DataSource bean

Database.host=localhost

Databse .user=admin

**7. What is the difference between an embedded container and a WAR?**  
There is no force to go container less  
– Embedded container is just one feature of Spring Boot  
• Traditional WAR also benefits a lot from Spring Boot  
– Automatic Spring MVC setup, including DispatcherServlet  
– Sensible defaults based on the classpath content  
– Embedded container can be used during development

**8. What embedded containers does Spring Boot support?**  
Spring Boot includes support for embedded Tomcat, Jetty, and Undertow servers.  
By default the embedded server will listen for HTTP requests on port 8080.

**9. What does @EnableAutoConfiguration do? What about @SpringBootApplication?**  
**@EnableAutoConfiguration annotation** on a Spring Java configuration class  
– Causes Spring Boot to automatically create beans it thinks you need  
– Usually based on classpath contents, can easily override

@Configuration

@EnableAutoConfiguration

public class MyAppConfig {

public static void main(String[] args) {

SpringApplication.run(MyAppConfig.class, args);

}

}

**@SpringBootApplication**was available from Spring Boot 1.2  
It is very common to use @EnableAutoConfiguration, @Configuration, and @ComponentScan together.

**With @SpringBootApplication annotation**

@Configuration

@ComponentScan

@EnableAutoConfiguration

public class MyAppConfig {

...

}

@SpringBootApplication

public class MyAppConfig {

...

}

**10. What is a Spring Boot starter POM? Why is it useful?**  
Starters are a set of convenient dependency descriptors that you can include in your application. The starters contain a lot of the dependencies that you need to get a project up and running quickly and with a consistent, supported set of managed transitive dependencies.

The starter POMs are convenient dependency descriptors that can be added to your application’s Maven. In simple words, if you are developing a project that uses Spring Batch for batch processing, you just have to include spring-boot-starter-batch that will import all the required dependencies for the Spring Batch application. This reduces the burden of searching and configuring all the dependencies required for a framework.

**11. Spring Boot supports both Java properties and YML files. Would you recognize and understand them if you saw them?**  
spring boot application java property file name is application.properties  
spring boot application YML file name is application.yml

**12. Can you control logging with Spring Boot? How?**  
Yes, we can control logging with spring boot.  
 **Customizing default Configuration for Logging:**  
By adding logback.xml file to the application we can override the default logging configuration providing by the Spring Boot. This file place in the classpath (src/main/resources) of the application for Spring Boot to pick the custom configuration.

**Spring Boot can control the logging level**  
– Just set it in application.properties  
• Works with most logging frameworks  
– Java Util Logging, Logback, Log4J, Log4J2

**13. How to reload my changes on Spring Boot without having to restart server?**  
Include following maven dependency in the application.

logging.level.org.springframework=DEBUG

logging.level.com.acme.your.code=INFO

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.6.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

**Automatic restart**  
Applications that use spring-boot-devtools will automatically restart whenever files on the classpath change. This can be a useful feature when working in an IDE as it gives a very fast feedback loop for code changes. By default, any entry on the classpath that points to a folder will be monitored for changes.

This can be achieved using DEV Tools. With this dependency any changes you save, the embedded tomcat will restart. Spring Boot has a Developer tools (DevTools) module which helps to improve the productivity of developers. One of the key challenge for the Java developers is to auto deploy the file changes to server and auto restart the server. Developers can reload changes on Spring Boot without having to restart my server. This will eliminates the need for manually deploying the changes every time. Spring Boot doesn’t have this feature when it has released it’s first version. This was a most requested features for the developers. The module DevTools does exactly what is needed for the developers. This module will be disabled in the production environment.

**14. What is Actuator in Spring Boot?**  
pring Boot Actuator is a sub-project of Spring Boot. It adds several production grade services to your application with little effort on your part. There are also has many features added to your application out-of-the-box for managing the service in a production (or other) environment. They’re mainly used to expose different types of information about the running application – health, metrics, info, dump, env etc.

**15. How to run Spring boot application to custom port ?**  
In application.properties, add following property.

server.port = 8181

**16. How to implement security for Spring boot application ?**

Add spring security starter to the boot application

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**17. What is the configuration file name used by Spring Boot?**

The configuration file used in spring boot projects is application.properties. This file is very important where we would over write all the default configurations. Normally we have to keep this file under the resources folder of the project.

**18. How to implement Spring web using Spring boot?**

Web Application Convenience

• Boot automatically configures

– A DispatcherServlet & ContextLoaderListener

– Spring MVC using same defaults as @EnableWebMvc

• Plus many useful extra features:

– Static resources served from classpath

• /static, /public, /resources or /META-INF/resources

– Templates served from /templates

• If Velocity, Freemarker, Thymeleaf, or Groovy on classpath

– Provides default /error mapping

• Easily overridden

– Default MessageSource for I18N

**19. How to configure datasource using Spring boot?**

• Use either spring-boot-starter-jdbc or spring-boot-starterdata-jpa and include a JDBC driver on classpath

• Declare properties

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=dbuser

spring.datasource.password=dbpass

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

– Spring Boot will create a DataSource with properties set

– Will even use a connection pool if the library is found on the classpath!

**20. What is YAML?**

Yaml Ain’t a Markup Language

– Recursive acronym

• Created in 2001

• Alternative to .properties files

– Allows hierarchical configuration

• Java parser for YAML is called SnakeYAML

– Must be in the classpath

– Provided by spring-boot-starters

YAML for Properties

• Spring Boot support YAML for Properties

– An alternative to properties files

application.properties

database.host = localhost

database.user = admin

application.yml

database:

host: localhost

user: admin

• YAML is convenient for hierarchical configuration data

– Spring Boot properties are organized in groups

– Examples: server, database, etc

Q => What is Spring Boot?

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.

You can use Spring Boot to create Java applications that can be started using java -jar or more traditional war deployments.

Q => What are the advantages of spring boot application?

Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).

Opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss.

Absolutely no code generation and no requirement for XML configuration.

Provide a radically faster and widely accessible getting started experience for all Spring development.

Q => What are the requirements of Spring boot System?

Spring Boot 1.5.9.RELEASE requires

Java 7 +

Spring 4.3.13 +

For build support

Maven 3.2+

Gradle 2.9+

Container Support

Tomcat 7+

Jetty 8+ (Jetty 9.3 requires JDK 8 +)

Read more Spring boot system requirements

Q => What are the use of @EnableAutoConfiguration annotation?

This annotation tells Spring Boot to “guess” how you will want to configure Spring, based on the jar dependencies that you have added.

Read more @enableautoconfiguration spring boot example

Q => What is Spring Boot Starter?

Spring Boot provides a number of “Starters” that make easy to manage dependencies for your project.

Read More Spring boot starter

Q => What is spring-boot-starter-parent?

The spring-boot-starter-parent is a special starter that makes Maven or Gradle dependency-management easier by adding jars to your classpath.

Read More spring-boot-starter-parent

Q => What is spring-boot-starter-web?

This starter will add Tomcat and Spring MVC dependency to our application and its default configuration.

Read More spring-boot-starter-web

Q => How to create an executable jar using spring boot?

Add this below plugin to pom.xml

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

Use mvn clean package to create executable jar file

Read More spring boot executable jar example

Q => How do you run and stop spring boot executable jar?

Open cmd or shell window and use java -jar as shown below

$ java -jar myproject-0.0.1-SNAPSHOT.jar

To stop use ctrl+C

Read More run spring boot jar from command line

Q => How do you change JDK version in spring boot?

Java 1.6 as the default compiler level.

You can overwrite it by adding java.version property tag as shown below

<properties>

<java.version>1.8</java.version>

</properties>

Q => How to disable specific auto-configuration in spring boot?

You can use exclude property as shown below to disable specific auto configuration

@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})

Q => What is the use of @SpringBootApplication annotation?

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes

Since many spring Boot developers have their main class annotated with @Configuration, @EnableAutoConfiguration and @ComponentScan, spring boot provides you a new annotation @SpringBootApplication as replacement.

Q => What is spring-boot-devtools ?

Applications that use spring-boot-devtools will automatically restart whenever files on the classpath change. This can be a useful feature when working in an IDE as it gives a very fast feedback loop for code changes.

Q => What is LiveReload?

The spring-boot-devtools module includes an embedded LiveReload server that can be used to trigger a browser refresh when a resource is changed. LiveReload browser extensions are freely available for Chrome, Firefox and Safari from livereload.com.

Read More spring boot devtools

Q => How to exclude auto restart for static files?

By default changing resources in /META-INF/maven, /META-INF/resources, /resources, /static, /public or /templates will not trigger a restart

But If you want to customize these exclusions you can use the spring.devtools.restart.exclude property

if you want to keep those defaults and add additional exclusions, use the spring.devtools.restart.additional-exclude property instead

Read More spring boot devtools

Q => How to start spring boot application in debug mode?

java -jar myproject-0.0.1-SNAPSHOT.jar –debug

Q => What are the advantages of YAML file than Properties file?

YAML is a superset of JSON, and as such is a very convenient format for specifying hierarchical configuration data. The SpringApplication class will automatically support YAML as an alternative to properties whenever you have the SnakeYAML library on your classpath.

Q => What are the different ways to load YAML file in Spring boot?

1. YamlPropertiesFactoryBean will load YAML as Properties

2. YamlMapFactoryBean will load YAML as a Map

Q => What are the advantages of spring Externalized Configuration?

Externalize your configuration to work with the same application code in different environments. You can use properties files, YAML files, environment variables and command-line arguments to externalize configuration.

Q => What are Profiles in spring boot?

Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. Any @Component or @Configuration can be marked with @Profile to limit when it is loaded

Q => How to write custom log configuration in spring boot?

You can force Spring Boot to use a particular logging system using the org.springframework.boot.logging.LoggingSystem system property. The value should be the fully-qualified class name of a LoggingSystem implementation. You can also disable Spring Boot’s logging configuration entirely by using a value of none.

Q => How do you customize Favicon in spring boot web application?

Spring Boot looks for a favicon.ico in the configured static content locations and the root of the classpath (in that order). If such file is present, it is automatically used as the favicon of the application.

Q => How spring boot handles error in application?

Spring Boot provides an /error mapping by default that handles all errors in a sensible way, and it is registered as a ‘global’ error page in the servlet container.

Q => How do you Create a deployable war file in spring boot?

Step1: Extend SpringBootServletInitializer and override its configure method

Step 2: Change packing type to war in pom.xml or in build.gradle

Step 3: Mark the embedded servlet container dependency as provided

Q => What is Hot swapping in spring boot?

Reloading the changes without restarting the server is called hot swapping, Modern IDEs (Eclipse, IDEA, etc.) all support hot swapping of bytecode, so if you make a change that doesn’t affect class or method signatures it should reload cleanly with no side effects.

Q => How do you Switch off the Spring Boot security configuration?

If you define a @Configuration with @EnableWebSecurity anywhere in your application it will switch off the default webapp security settings in Spring Boot.

Q => How to execute Spring Batch jobs on startup?

Spring Batch auto-configuration is enabled by adding @EnableBatchProcessing (from Spring Batch) somewhere in your context. By default it executes all Jobs in the application context on startup

Q => Does spring boot need Logging? What is the default one?

Spring Boot has no mandatory logging dependency, except for the Commons Logging API.

Q => How do you configure Configure Logback for logging?

If you put a logback.xml in the root of your classpath it will be picked up from there

Q => How do you Configure Log4j for logging?

Spring Boot supports Log4j 2 for logging configuration if it is on the classpath. If you are using the starters for assembling dependencies that means you have to exclude Logback and then include log4j 2 instead

Q => How do you write a Write a JSON REST service in spring boot?

Any Spring @RestController in a Spring Boot application should render JSON response by default as long as Jackson2 is on the classpath

Q => How do you Write an XML REST service in spring boot?

If you have the Jackson XML extension (jackson-dataformat-xml) on the classpath, it will be used to render XML responses

Q => What is the default Multipart File Uploads size in spring boot?

By default Spring Boot configures Spring MVC with a maximum file of 1MB per file and a maximum of 10MB of file data in a single request.

Q => How do you Enable HTTP response compression in spring boot?

HTTP response compression is supported by Jetty, Tomcat, and Undertow. It can be enabled by adding server.compression.enabled=true in application.properties

Q => How do you add Add a Servlet, Filter or Listener to an application ?

There are two ways to add Servlet, Filter, ServletContextListener and the other listeners supported by the Servlet spec to your application. You can either provide Spring beans for them, or enable scanning for Servlet components.

Q => How do you Change tomcat or jetty HTTP port?

You can change the tomcat http port by changing default http property in application.properties file.

1) What is Spring Boot?

Spring Boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework. It is used to create stand alone spring based application that you can just run because it needs very little spring configuration.

2) What are the advantages of Spring Boot?

o Create stand-alone Spring applications that can be started using java -jar.

o Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.

o It provides opinionated 'starter' POMs to simplify your Maven configuration.

o It automatically configure Spring whenever possible.

3) What are the features of Spring Boot?

o Web Development

o SpringApplication

o Application events and listeners

o Admin features

4) How to create Spring Boot application using Maven?

There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.

o Spring Maven Project

o Spring Starter Project Wizard

o Spring Initializr

o Spring Boot CLI

5) How to create Spring Boot project using Spring Initializer?

It is a web tool which is provided by Spring on official site. You can create Spring Boot project by providing project details.

6) How to create Spring Boot project using boot CLI?

It is a tool which you can download from the official site of Spring Framework. Here, we are explaining steps.

7) How to create simple Spring Boot application?

To create an application. We are using STS (Spring Tool Suite) IDE and it includes the various steps that are explaining in steps.

8) What are the Spring Boot Annotations?

The @RestController is a stereotype annotation. It adds @Controller and @ResponseBody annotations to the class. We need to import

org.springframework.web.bind.annotation package in our file, in order to implement it.

9) What is Spring Boot dependency management?

Spring Boot manages dependencies and configuration automatically. You don't need to specify version for any of that dependencies.Spring Boot upgrades all dependencies automatically when you upgrade Spring Boot.

10) What are the Spring Boot properties?

Spring Boot provides various properties which can be specified inside our project's application.properties file. These properties have default values and you can set that inside the properties file. Properties are used to set values like: server-port number, database connection configuration etc.

11) What are the Spring Boot Starters?

Starters are a set of convenient dependency descriptors which we can include in our application. Spring Boot provides built-in starters which makes development easier and rapid. For example, if we want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project.

12) What is Spring Boot Actuator?

Spring Boot provides actuator to monitor and manage our application. Actuator is a tool which has HTTP endpoints. when application is pushed to production, you can choose to manage and monitor your application using HTTP endpoints.

13) What is thymeleaf?

It is a server side Java template engine for web application. It's main goal is to bring elegant natural templates to your web application. It can be integrate with Spring Framework and ideal for HTML5 Java web applications.

14) How to use thymeleaf?

In order to use Thymeleaf we must add it into our pom.xml file like:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

15) How to connect Spring Boot to the database using JPA?

Spring Boot provides spring-boot-starter-data-jpa starter to connect Spring application with relational database efficiently. You can use it into project POM (Project Object Model) file.

16) How to connect Spring Boot application to database using JDBC?

Spring Boot provides starter and libraries for connecting to our application with JDBC. Here, we are creating an application which connects with Mysql database. It includes the following steps to create and setup JDBC with Spring Boot.

17) What is @RestController annotation in Spring Boot?

The @RestController is a stereotype annotation. It adds @Controller and @ResponseBody annotations to the class. We need to import org.springframework.web.bind.annotation package in our file, in order to implement it.

18) What is @RequestMapping annotation in Spring Boot?

The @RequestMapping annotation is used to provide routing information. It tells to the Spring that any HTTP request should map to the corresponding method. We need to import org.springframework.web.annotation package in our file.

19) How to create Spring Boot application using Spring Starter Project Wizard?

There is one more way to create Spring Boot project in STS (Spring Tool Suite). Creating project by using IDE is always a convenient way. Follow the following steps in order to create a Spring Boot Application by using this wizard.

20) Spring Vs Spring Boot?

Spring is a web application framework based on Java. It provides tools and libraries to create a complete cutomized web application.Wheras Spring Boot is a spring module which is used to create spring application project that can just run.

1. Overview

Spring Boot is an opinionated, convention-over-configuration focused addition to the Spring platform – highly useful to get started with minimum effort and create stand-alone, production-grade applications.

This tutorial is a starting point for Boot – a way to get started in a simple manner, with a basic web application.

We’ll go over some core configuration, a front-end, quick data manipulation, and exception handling.

2. Setup

First, let’s use Spring Initializr to generate the base for our project.

The generated project relies on the Boot parent:

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.0.1.RELEASE</version>

<relativePath />

</parent>

The initial dependencies are going to be quite simple:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

3. Application Configuration

Next, we’ll configure a simple main class for our application:

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

Notice how we’re using @SpringBootApplication as our primary application configuration class; behind the scenes, that’s equivalent to @Configuration, @EnableAutoConfiguration, and @ComponentScan together.

Finally, we’ll define a simple application.properties file – which for now only has one property:

server.port=8081

server.port changes the server port from the default 8080 to 8081; there are of course many more Spring Boot properties available.

4. Simple MVC View

Let’s now add a simple front end using Thymeleaf.

First, we need to add the spring-boot-starter-thymeleaf dependency to our pom.xml:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

That enables Thymeleaf by default – no extra configuration is necessary.

We can now configure it in our application.properties:

spring.thymeleaf.cache=false

spring.thymeleaf.enabled=true

spring.thymeleaf.prefix=classpath:/templates/

spring.thymeleaf.suffix=.html

spring.application.name=Bootstrap Spring Boot

Next, we’ll define a simple controller and a basic home page – with a welcome message:

@Controller

public class SimpleController {

@Value("${spring.application.name}")

String appName;

@GetMapping("/")

public String homePage(Model model) {

model.addAttribute("appName", appName);

return "home";

}

}

Finally, here is our home.html:

<html>

<head><title>Home Page</title></head>

<body>

<h1>Hello !</h1>

<p>Welcome to <span th:text="${appName}">Our App</span></p>

</body>

</html>

Note how we used a property we defined in our properties – and then injected that so that we can show it on our home page.

5. Security

Next, let’s add security to our application – by first including the security starter:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

By now, you’re hopefully noticing a pattern – most Spring libraries are easily imported into our project with the use of simple Boot starters.

Once the spring-boot-starter-security dependency on the classpath of the application – all endpoints are secured by default, using either httpBasic or formLogin based on Spring Security’s content-negotiation strategy.

That’s why, if we have the starter on the classpath, we should usually define our own custom Security configuration by extending the WebSecurityConfigurerAdapter class:

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.anyRequest()

.permitAll()

.and().csrf().disable();

}

}

In our example, we’re allowing unrestricted access to all endpoints.

Of course, Spring Security is an extensive topic and one not easily covered in a couple of lines of configuration – so I definitely encourage you to go deeper into the topic.

6. Simple Persistence

Let’s start by defining our data model – a simple Book entity:

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

@Column(nullable = false, unique = true)

private String title;

@Column(nullable = false)

private String author;

}

And its repository, making good use of Spring Data here:

public interface BookRepository extends CrudRepository<Book, Long> {

List<Book> findByTitle(String title);

}

Finally, we need to of course configure our new persistence layer:

@EnableJpaRepositories("org.baeldung.persistence.repo")

@EntityScan("org.baeldung.persistence.model")

@SpringBootApplication

public class Application {

...

}

Note that we’re using:

@EnableJpaRepositories to scan the specified package for repositories

@EntityScan to pick up our JPA entities

To keep things simple, we’re using an H2 in-memory database here – so that we don’t have any external dependencies when we run the project.

Once we include H2 dependency, Spring Boot auto-detects it and sets up our persistence with no need for extra configuration, other than the data source properties:

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.url=jdbc:h2:mem:bootapp;DB\_CLOSE\_DELAY=-1

spring.datasource.username=sa

spring.datasource.password=

Of course, like security, persistence is a broader topic than this basic set here, and one you should certainly explore further.

7. Web and the Controller

Next, let’s have a look at a web tier – and we’ll start that by setting up a simple controller – the BookController.

We’ll implement basic CRUD operations exposing Book resources with some simple validation:

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public Iterable findAll() {

return bookRepository.findAll();

}

@GetMapping("/title/{bookTitle}")

public List findByTitle(@PathVariable String bookTitle) {

return bookRepository.findByTitle(bookTitle);

}

@GetMapping("/{id}")

public Book findOne(@PathVariable Long id) {

return bookRepository.findById(id)

.orElseThrow(BookNotFoundException::new);

}

@PostMapping

@ResponseStatus(HttpStatus.CREATED)

public Book create(@RequestBody Book book) {

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void delete(@PathVariable Long id) {

bookRepository.findById(id)

.orElseThrow(BookNotFoundException::new);

bookRepository.deleteById(id);

}

@PutMapping("/{id}")

public Book updateBook(@RequestBody Book book, @PathVariable Long id) {

if (book.getId() != id) {

throw new BookIdMismatchException();

}

bookRepository.findById(id)

.orElseThrow(BookNotFoundException::new);

return bookRepository.save(book);

}

}

Given this aspect of the application is an API, we made use of the @RestController annotation here – which equivalent to a @Controller along with @ResponseBody – so that each method marshalls the returned resource right to the HTTP response.

Just one note worth pointing out – we’re exposing our Book entity as our external resource here. That’s fine for our simple application here, but in a real-world application, you will likely want to separate these two concepts.

8. Error Handling

Now that the core application is ready to go, let’s focus on a simple centralized error handling mechanism using @ControllerAdvice:

@ControllerAdvice

public class RestExceptionHandler extends ResponseEntityExceptionHandler {

@ExceptionHandler({ BookNotFoundException.class })

protected ResponseEntity<Object> handleNotFound(

Exception ex, WebRequest request) {

return handleExceptionInternal(ex, "Book not found",

new HttpHeaders(), new HttpHeaders(), HttpStatus.NOT\_FOUND, request);

}

@ExceptionHandler({ BookIdMismatchException.class,

ConstraintViolationException.class,

DataIntegrityViolationException.class })

public ResponseEntity<Object> handleBadRequest(

Exception ex, WebRequest request) {

return handleExceptionInternal(ex, ex.getLocalizedMessage(),

new HttpHeaders(), HttpStatus.BAD\_REQUEST, request);

}

}

Beyond the standard exceptions we’re handling here, we’re also using a custom exception:

BookNotFoundException:

public class BookNotFoundException extends RuntimeException {

public BookNotFoundException(String message, Throwable cause) {

super(message, cause);

}

// ...

}

This should give you an idea of what’s possible with this global exception handling mechanism. If you’d like to see a full implementation, have a look at the in-depth tutorial.

Note that Spring Boot also provides an /error mapping by default. We can customize its view by creating a simple error.html:

<html lang="en">

<head><title>Error Occurred</title></head>

<body>

<h1>Error Occurred!</h1>

<b>[<span th:text="${status}">status</span>]

<span th:text="${error}">error</span>

</b>

<p th:text="${message}">message</p>

</body>

</html>

Like most other aspects in Boot, we can control that with a simple property:

1 server.error.path=/error2

9. Testing

Finally, let’s test our new Books API.

We’ll immediately make use of @SpringBootTest to load the application context:

@RunWith(SpringRunner.class)

@SpringBootTest(classes = { Application.class }, webEnvironment

= WebEnvironment.DEFINED\_PORT)

public class LiveTest {

private static final String API\_ROOT

= "http://localhost:8081/api/books";

private Book createRandomBook() {

Book book = new Book();

book.setTitle(randomAlphabetic(10));

book.setAuthor(randomAlphabetic(15));

return book;

}

private String createBookAsUri(Book book) {

Response response = RestAssured.given()

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.body(book)

.post(API\_ROOT);

return API\_ROOT + "/" + response.jsonPath().get("id");

}

}

First, we can try to find books using variant methods:

@Test

public void whenGetAllBooks\_thenOK() {

Response response = RestAssured.get(API\_ROOT);

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

}

@Test

public void whenGetBooksByTitle\_thenOK() {

Book book = createRandomBook();

createBookAsUri(book);

Response response = RestAssured.get(

API\_ROOT + "/title/" + book.getTitle());

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

assertTrue(response.as(List.class)

.size() > 0);

}

@Test

public void whenGetCreatedBookById\_thenOK() {

Book book = createRandomBook();

String location = createBookAsUri(book);

Response response = RestAssured.get(location);

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

assertEquals(book.getTitle(), response.jsonPath()

.get("title"));

}

@Test

public void whenGetNotExistBookById\_thenNotFound() {

Response response = RestAssured.get(API\_ROOT + "/" + randomNumeric(4));

assertEquals(HttpStatus.NOT\_FOUND.value(), response.getStatusCode());

}

Next, we’ll test creating a new book:

@Test

public void whenCreateNewBook\_thenCreated() {

Book book = createRandomBook();

Response response = RestAssured.given()

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.body(book)

.post(API\_ROOT);

assertEquals(HttpStatus.CREATED.value(), response.getStatusCode());

}

@Test

public void whenInvalidBook\_thenError() {

Book book = createRandomBook();

book.setAuthor(null);

Response response = RestAssured.given()

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.body(book)

.post(API\_ROOT);

assertEquals(HttpStatus.BAD\_REQUEST.value(), response.getStatusCode());

}

Update an existing book:

@Test

public void whenUpdateCreatedBook\_thenUpdated() {

Book book = createRandomBook();

String location = createBookAsUri(book);

book.setId(Long.parseLong(location.split("api/books/")[1]));

book.setAuthor("newAuthor");

Response response = RestAssured.given()

.contentType(MediaType.APPLICATION\_JSON\_VALUE)

.body(book)

.put(location);

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

response = RestAssured.get(location);

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

assertEquals("newAuthor", response.jsonPath()

.get("author"));

}

And delete a book:

@Test

public void whenDeleteCreatedBook\_thenOk() {

Book book = createRandomBook();

String location = createBookAsUri(book);

Response response = RestAssured.delete(location);

assertEquals(HttpStatus.OK.value(), response.getStatusCode());

response = RestAssured.get(location);

assertEquals(HttpStatus.NOT\_FOUND.value(), response.getStatusCode());

}