Prepared by Radha V Krishna

SPRING BOOT

- Spring Boot is a brand new framework from the team at Pivotal, designed to simplify the bootstrapping and development of a new Spring application. The framework takes an opinionated approach to configuration, freeing developers from the need to define boilerplate configuration. In that, Boot aims to be a front-runner in the ever-expanding rapid application development space.

ADVANTAGES OF SPRING BOOT

- Create stand-alone Spring applications that can be started using java -jar.
- Embed Tomcat, Jetty or Undertow directly. You don't need to deploy WAR files.
- It provides opinionated 'starter' POMs to simplify your Maven configuration.
- It automatically configure Spring whenever possible.
- It provides production-ready features such as metrics, health checks and externalized configuration.
- Absolutely no code generation and no requirement for XML configuration.

PREREQUISITE OF SPRING BOOT

- To create a Spring Boot application following are the prerequisites. In this tutorial, we will use Spring Tool Suite IDE.
- Java 1.8
- Gradle 2.3+ or Maven 3.0+
- Spring Framework 5.0.0.BUILD-SNAPSHOT
- An IDE (Spring Tool Suit) is recommended.
- Or Eclipse with Spring boot plugin

SPRING BOOT FEATURES

- Web Development
- SpringApplication
- Application events and listeners
- Admin features
- Externalized Configuration
- Properties Files
- YAML Support
- Type-safe Configuration
- Logging
- Security

SPRING BOOT PROJECT

- There are multiple approaches to create Spring Boot project. We can use any of the following approach to create application.
- Spring Maven Project
- Spring Starter Project Wizard
- Spring Initializer – spring.start.io
- Spring Boot CLI

Bootstrapping a Spring boot Application

start.spring.io

Apps | Gmail | YouTube | Maps | Navbar | New Tab | AFFABLE | HOME | Overview (Java Platf... | nagappatinam dd | » | Other bookmarks

spring initializr

Project
- (●) Maven Project
- ( ) Gradle Project

Language
- (●) Java
- ( ) Kotlin
- ( ) Groovy

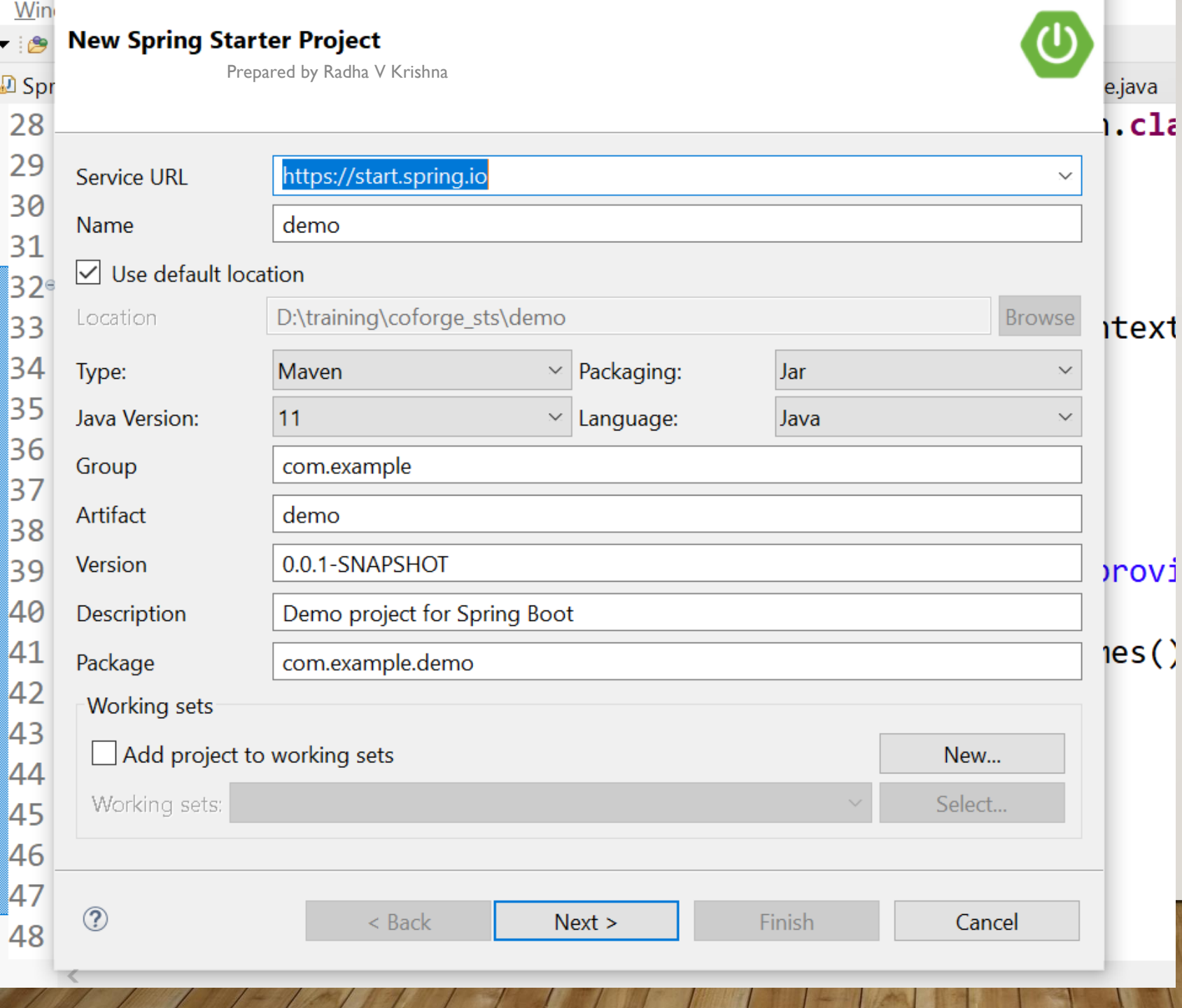
Dependencies — ADD DEPENDENCIES... CTRL + B

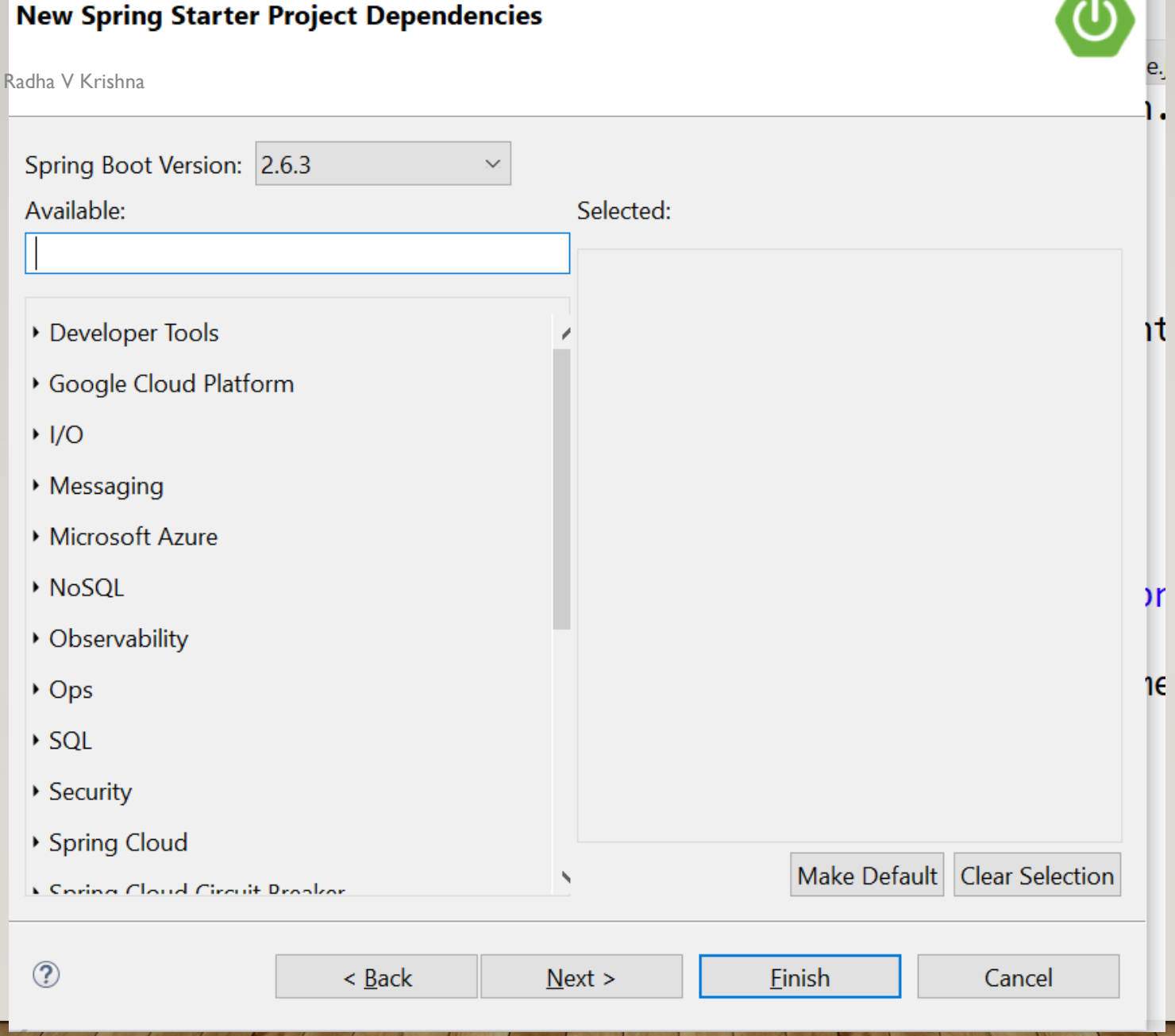
No dependency selected

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	(●) Jar ( ) War
Java	( ) 17 (●) 11 ( ) 8

GENERATE CTRL + ⏎ | EXPLORE CTRL + SPACE | SHARE...


New Spring Starter Project
Prepared by Radha V Krishna
Service URL
https://start.spring.io
Name
demo
Use default location
Location
D:\training\coforge_sts\demo
Browse
Type:
Maven
Packaging:
Jar
Java Version:
11
Language:
Java
Group
com.example
Artifact
demo
Version
0.0.1-SNAPSHOT
Description
Demo project for Spring Boot
Package
com.example.demo
Working sets
Add project to working sets
New...
Working sets:
Select...
< Back
Next >
Finish
Cancel



Include @Configuration, @EnableAutoConfiguration and @ComponentScan

```
@SpringBootApplication
public class SpringBootDemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootDemoApplication.class, args);
    }
}
```

Adding a server to Spring Boot App

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

Command Line Runner


```
//java 8
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

    @Bean
    public CommandLineRunner
commandLineRunner(ApplicationContext ctx) {
        return args -> {

// code
}     };   }
}
```

```
//older versions of java
@SpringBootApplication
public class Application implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

public void run(String... args) throws Exception {

}
}
```

APPLICATION CONFIGURATION WITH SPRING BOOT APPLICATION.PROPERTIES

- Spring Boot allows you to configure your application configuration using a file named application.properties
- application.properties can reside anywhere in the classpath of the application.
- Following can be set
  - server.port
  - Spring.application.name
  - logging.level.org.springframework.web.servlet: DEBUG
  - <userdefined-properties>.<value>

CHANGING THE SERVER PORT

- In application.resources
  - server.port=8888 or <any port number>

```
@Component
@ConfigurationProperties("limits-service")
public class Configuration {
public int getMaximum() {
return maximum;
……….
}
```

Or @Value(“${message}”)

In application.properties:

limits-service.maximum=100
Message=hello

Using Yaml for configuration

```
//using application.yaml
server:
  port:
    8088

name:
  Spring Yaml Example
```

```
@Configuration
@EnableConfigurationProperties
@ConfigurationProperties
public class YamlConfig {

private String name;
//getters and setters..
```

```
@SpringBootApplication
public class YamlSupoortApplication implements
CommandLineRunner{
@Autowired
private YamlConfig config;
…main()
@Override
public void run(String... args) throws Exception {
// TODO Auto-generated method stub
System.out.println("name: " + config.getName());
System.out.println(("name1:"+config.getName1()));
}

}
```

YAML is a convenient format for specifying hierarchical configuration data.

HANDSON

- Create a RestController that reads properties defined in application.resources
- Like server.port.
- Limits-service.minimum
- Limits-service.maximum
- welcome.message
- logfile:text.txt

Lombok Dependency for autogeneration of bean methods

```
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.12</version>
    <scope>provided</scope>
</dependency>
```

THYMELEAF

- **Thymeleaf** is a modern server-side Java template engine for both web and standalone environments.
- Thymeleaf's main goal is to bring elegant *natural templates* to your development workflow — HTML that can be correctly displayed in browsers and also work as static prototypes, allowing for stronger collaboration in development teams.

Refer:
https://www.thymeleaf.org/doc/articles/springmvcaccessdata.html

MVC app Sample


```
@Controller
public class TestController {

@GetMapping("/")
//@ResponseBody
public String start()
{
return "login";
}

@PostMapping("/login")
public String
sayHello(@RequestParam("username") String
username,
@RequestParam("password") String password)
{
return "hello";
}
}
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
thymeleaf</artifactId>
</dependency>
```

```
// login.html – to be placed inside templates folder
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
      xmlns:sec="https://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
    <head>
        <title>Spring Boot Example </title>
    </head>
    <body>
        <div th:if="${param.error}">
            Invalid username and password.
        </div>
        <div th:if="${param.logout}">
            You have been logged out.
        </div>
        <form th:action="@{/login}" method="post">
            <div><label> User Name : <input type="text" name="username"/> </label></div>
            <div><label> Password: <input type="password" name="password"/>
</label></div>
            <div><input type="submit" value="Sign In"/></div>
        </form>
    </body>
</html>
```

```
//hello.html
<h1> This is hello from html file </h1>

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="https://www.thymeleaf.org"
xmlns:sec="https://www.thymeleaf.org/thymeleaf-
extras-springsecurity3">
    <head>
        <title>Spring Boot Example</title>
    </head>
    <body>
        <h1>Welcome!</h1>
        <p th:text="${param.username}"></p>

    </body>
</html>
```

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<link href="css/usercss.css" rel="stylesheet">
<title>Insert title here</title>
</head>
<body>
<form th:action="@{/validateuser}" th:object="${loginBean}" method="get">

  <div class="imgcontainer">
    <img src="images/avatar.jpg" alt="Avatar" class="avatar">
  </div>
  <label id="error_label" th:if="${#fields.hasErrors('email')}" th:errors="*{email}">Email
empty</label>
<input type="text" th:field="*{email}" placeholder="Email address" />


<label id="error_label" th:if="${#fields.hasErrors('password')}" th:errors="*{password}">Empty
Password</label>
<input type="text" th:field="*{password}" placeholder="Password"/>
```

```
<button type="submit">Login</button>
<a href="/userform">New User?</a>
     <label>
      <input type="checkbox" checked="checked" name="remember">
Remember me
    </label>

<div class="container" style="background-color:#f1f1f1">
    <button type="button" class="cancelbtn">Cancel</button>
    <span class="psw">Forgot <a href="#">password?</a></span>
  </div>
</form>
</body>
</html>
```

Note: CSS in the notes section

```
package com.examples.HelloWorldApplication;
```



```
import
javax.validation.constraints.NotNull;

public class LoginBean {
@NotNull(message="Email cannot be null")
private String email;
@NotNull(message="Password cannot be null")
private String password;
public String getEmail() {
return email;
}
public void setEmail(String email) {
this.email = email;
}
public String getPassword() {
return password;
}
public void setPassword(String password) {
this.password = password;
}

}
```

HANDS ON

- Create a Collection of Book Objects .Create endpoints for Get,POST,PUT and DELETE and test with a Suitable REST Client
- Like PostMan.

```
@Component
public class BookData {

private List<Book> bookList;

public BookData()
{
bookList = new ArrayList<Book>(Arrays.asList(new
Book("7886","Alchemist",150.25,100,"Motivational"),
new Book("1234","Think like a
Monk",250.25,100,"Motivational"),
new Book("2345","Ikigai",220.25,100,"Motivational")));
}

public List<Book> getAllBooks()
{
return bookList;
}
```

```
public class Book {

private String isbn;
private String title;
private double price;
private long stock;
private String category;
…….
….
```

Creating a RestController

```
@RequestMapping("/books")
@RestController
public class BookController {

@Autowired
private BookData bookData;


@GetMapping("/hello")    //
http://localhost:8081/books/hello
public String sayHello()
{
return "Hello! This is my First Rest Service";
}
}
```

Controller is coupled to a view component created in the same application.
(based on Monolithic Architecture)

RestController returns and exposes data in JSON format to other applications and is based on SOA.
(ServiceOrientedArchitecture)

```
@GetMapping("/getbooks")
public List<Book> getBooks()
{
return bookData.getAllBooks();
}

@GetMapping("/getbook/isbn/{isbn}") //
localhost:8081/books/getbook/isbn/1234
public Book getBook(@PathVariable("isbn") String isbn)
{
return bookData.getBook(isbn);
}

@PostMapping("/addbook")
public int addBook(@RequestBody Book book)
{
return bookData.addBook(book);
}
```

```
@DeleteMapping("/deletebook/isbn/{isbn}")
public int deleteBook(@PathVariable("isbn")
String isbn)
{
return bookData.deleteBook(isbn);
}
```

Profiling 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:

```
@Configuration
@Profile("production")
public class ProductionConfiguration {

    // ...

}
```

```
spring.profiles.active=production
```

Set this in application.properties

```
public interface MyProfile {

public void process();
}
```

```
@Component
@Profile("dev")
public class DevProfile implements MyProfile
{

@Override
public void process() {

System.out.println("This is Dev Profile");

}

}
```

```
spring.profiles.active=dev
```

```
@SpringBootApplication
public class SpringBootBasicsApplication{//
implements CommandLineRunner{

@Autowired
private MyProfile profile;

public static void main(String[] args) {

SpringApplication.run(SpringBootBasicsApplicat
on.class, args);
}

@Bean
CommandLineRunner
getCommandLineRunner(ApplicationContext ctx) {

return args->{
profile.process();}}
```