

R.M.K GROUP OF ENGINEERING INSTITUTIONS



R.M.K
GROUP OF
INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS



R.M.K
GROUP OF
INSTITUTIONS



Please read this disclaimer before proceeding:

This document is confidential and intended solely for the educational purpose of RMK Group of Educational Institutions. If you have received this document through email in error, please notify the system manager. This document contains proprietary information and is intended only to the respective group / learning community as intended. If you are not the addressee you should not disseminate, distribute or copy through e-mail. Please notify the sender immediately by e-mail if you have received this document by mistake and delete this document from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.

22CS402

Web Development Frameworks

Department : CSE & AI&DS

Batch / Year : 2022 - 2026 / II

Created by : Dr. K Manikannan & Mr. S Vijayakumar
Mr.L.MariaMichael Visuwasam
Ms.J.Bhuvaneswari
Ms.SaranyaR
R.SivaSubramanian

Date : 03.01.2024

1. Contents

S. No.	Contents
1	Contents
2	Course Objectives
3	Prerequisites
4	Syllabus
5	Course Outcomes
6	CO-PO Mapping
7	Lecture Plan
8	Activity Based Learning
9	Lecture Notes
10	Assignments
11	Part- A Questions & Answers
12	Part-B Questions
13	Supportive Online Courses
14	Real Time Applications
15	Content beyond the Syllabus
16	Assessment Schedule
17	Prescribed Text books & Reference Books
18	Mini Project Suggestions

Content – Unit I

S.No.	Contents
1	Introducing Spring Boot - Features
2	Introducing Spring Boot - Components
3	Getting started with Spring Boot
4	Common Spring Boot task-Managing configuration
5	Creating Custom Properties
6	Executing code on Spring Boot application startup
7	Database access with Spring data
8	Securing Spring Boot Application
9	Securing Spring Boot Application

2. Course Objectives

The Course will enable learners to:

- ❖ Simplify website development using Spring Boot as server-side technologies.
- ❖ Build single page applications using REACT as a reusable UI component technology as client-side technology.
- ❖ Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications
- ❖ Develop a scalable and responsive web application
- ❖ Develop an industry ready application web enterprise feature

3. Prerequisites

22CS402
Web Development Frameworks



22CS301
Advanced Java Programming



22CS202
Java Programming



22CS101 Problem Solving Using C++
22CS102 Software Development Practices

4. Syllabus

22CS301	Web Development Frameworks	L	T	P	C
		3	0	2	4
OBJECTIVES: The Course will enable learners to: <ul style="list-style-type: none">❖ Simplify website development using Spring boot as server-side technologies.❖ Build single page applications using REACT as a reusable UI component technology as client-side technology.❖ Assemble REACT as a front end technology and Node js as a server side technology to develop enterprise applications❖ Develop a scalable and responsive web application❖ Develop an industry ready application web enterprise feature					
UNIT I	SPRING BOOT AND STRUTS	9 +6			
Spring Boot: Introducing Spring Boot, getting started with spring Boot, Common Spring Boot task-Managing configuration, creating custom properties, executing code on Spring Boot application startup, Database access with Spring data, Securing Spring Boot application. List of Exercise/Experiments: <ol style="list-style-type: none">1. Use Spring Boot to build a Web Application2. Create REST Service for an Education Site					
UNIT II	JAVA REACT	9 +6			
React: Introduction to React, Pure React- The Virtual DOM, React Elements, React with JSX, Props, State, and the Component Tree, Enhancing Components- Flux. List of Exercise/Experiments: <ol style="list-style-type: none">1. Build Search filter in React2. Display a list in React3. Create Simple Login form in React					
UNIT III	Node JS	9 +6			
Node JS: Introduction to Node JS, Setting up Node.js, Node.js Modules- Finding and loading CommonJS and JSON modules using require, Hybrid CommonJS/Node.js/ES6 module scenarios, npm - the Node.js package management system. List of Exercise/Experiments: <ol style="list-style-type: none">1. Write a node.js program for making external http calls2. Write a program in node.js to parse the given url.					

4. Syllabus Contd...

UNIT IV	WEB FRAMEWORK (ANGULAR) – I	9+6
<p>Introduction- Angular First App, Angular UI with Bootstrap CSS Authentication, Authentication Service, Unsubscribe, Logout and Route Guard Cleanup, Customer Service ,Http Service, Token Interceptor, Multi Provider, Compile-time Configuration, Runtime Configuration, Error Handling.</p> <p>List of Exercise/Experiments:</p> <ol style="list-style-type: none">1. Create a Dropdown using Angular UI bootstrap2. Modify existing components and generating new components using Angular		
UNIT V	WEB FRAMEWORK (ANGULAR) – II	9+6
<p>Dependancy injection in Angular,Reactive programming in Angular, Laying out pages with Flex Layout, Implementing component communications, Change detection and component lifecycle.</p> <p>List of Exercise/Experiments:</p> <ol style="list-style-type: none">1. Launching your app with Angular root module		

4. Syllabus Contd...

OUTCOMES:

Upon completion of the course, the students will be able to:

- CO1:** Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.
- CO2:** Build single page applications using REACT as a reusable UI component technology as client side technology
- CO3:** Build applications using Node Js as server side technologies
- CO4:** Able to develop a web application using latest Angular Framework
- CO5:** Apply various Angular features including directives, components, and services.

TEXT BOOK:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (<https://www.manning.com/books/spring-boot-in-practice>)
2. Alex Banks, Eve Porcello, "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (<https://www.oreilly.com/library/view/learning-react/9781491954614/>)
3. David Herron, "Node.js Web Development - Fourth Edition", 2018, Packt Publishing, ISBN: 9781788626859
4. Sukesh Marla, "A Journey to Angular Development Paperback", BPB Publications. (https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition. (<https://www.manning.com/books/angular-development-with-typescript-Second-Edition>)

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers||, 1st Edition. Elsevier 2002

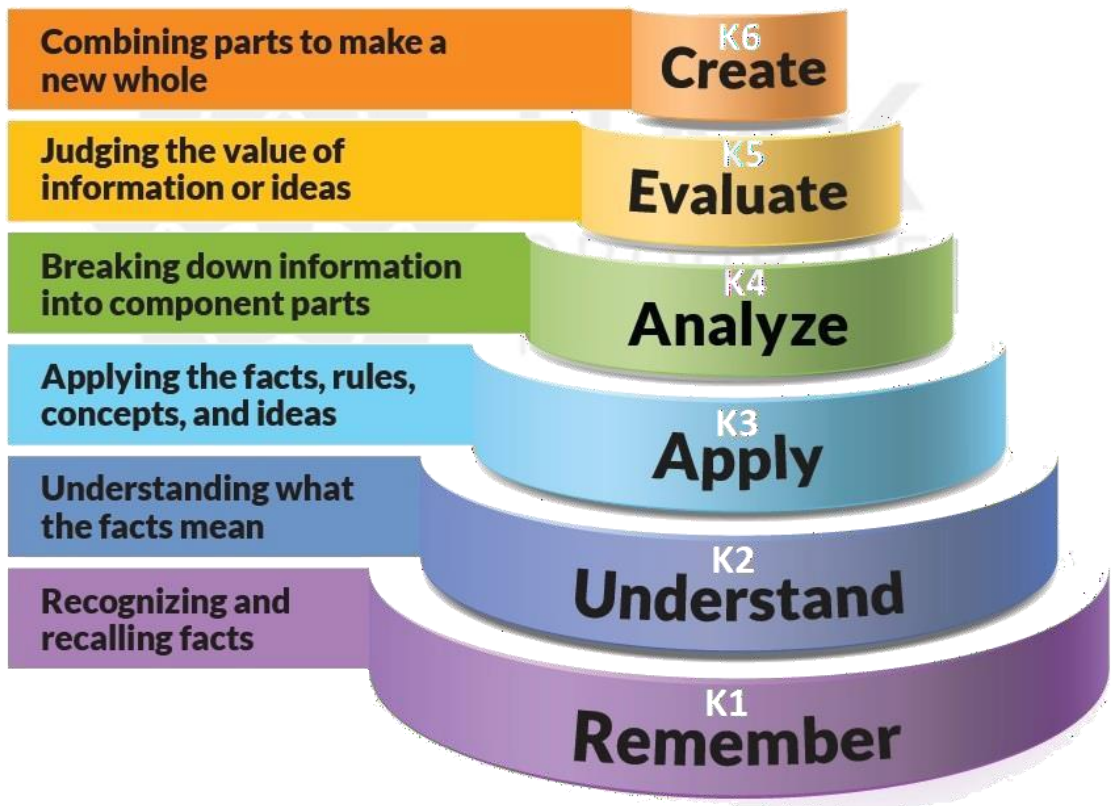
LIST OF EQUIPMENTS:

VSCode, Angular JS, React JS, Node JS, Ruby, Django

5. Course Outcomes

Upon completion of the course, the students will be able to:

- CO1:** Write Web API/RESTful API application programming interface to communicate with Spring boot as a serverside technology.
- CO2:** Build single page applications using REACT as a reusable UI component technology as client side technology
- CO3:** Build applications using Node Js as server side technologies
- CO4:** Able to develop a web application using latest Angular Framework
- CO5:** Apply various Angular features including directives, components, and services.



6. CO - PO Mapping

	POs and PSOs														
COs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	3	2	2	2	3	2	1	1	3	2	2	3	2	2
CO2	2	3	2	2	2	3	1			3	2	2	3	2	2
CO3	2	3	2	2	2	1	2	1	1	3	2	2	3	2	2
CO4	2	3	2	2	1	3	2	1	1	3	2	2	3	2	2
CO5	2	3	2	2	2	3	2	1	1	3	2	2	3	2	2



R.M.K.
GROUP OF
INSTITUTIONS

7. Lecture Plan - Unit I

S. No.	Topic	No. of Periods	Proposed Date	Actual Lecture Date	Pertaining CO	Taxonomy Level	Mode of Delivery
1	Introducing Spring Boot - Features	1			CO1	K2	Chalk & Talk
2	Introducing Spring Boot - Components	1			CO1	K3	Chalk & Talk
3	Getting started with Spring Boot	1			CO1	K2	Chalk & Talk
4	Common Spring Boot task-Managing configuration	1			CO1	K2	Chalk & Talk
5	Creating Custom Properties	1			CO1	K2	Chalk & Talk
6	Executing code on Spring Boot application startup	1			CO1	K2	Chalk & Talk
7	Database access with Spring data	1			CO1	K3	Chalk & Talk
8	Securing Spring Boot Application	1			CO1	K2	Chalk & Talk
9	Securing Spring Boot Application	1			CO1	K2	Chalk & Talk

8. Activity Based Learning

Learning Method	Activity
Learn by Solving Problems	Tutorial Sessions available in iamneo Portal
Learn by Questioning	Quiz / MCQ Using RMK Nextgen App and iamneo Portal
Learn by doing Hands-on	Practice available in iamneo Portal

iam**neo**

RMK **Nextgen**[®]



R.M.K.
GROUP OF
INSTITUTIONS

9. Lecture Notes

Spring Boot: Introducing Spring Boot, getting started with spring Boot, Common Spring Boot task-Managing configuration, creating custom properties, executing code on Spring Boot application startup, Database access with Spring data, Securing Spring Boot application.



R.M.K.
GROUP OF
INSTITUTIONS

9. Lecture Notes

Spring Boot Introducing Spring Boot

Introduction

- ❑ Spring Boot is an open source extension of the Spring Framework designed to simplify the Spring application development.
- ❑ Spring Boot is Java-based framework used to create a Micro Service.
- ❑ It is developed by Pivotal Team and is used to build stand-alone and production ready spring Applications.

What is Spring boot?

- ❑ Sprint boot is a Java-based spring framework used for Rapid Application Development to build stand-alone micro services.
- ❑ It has extra support of auto-configuration and embedded application servers like tomcat, jetty, etc to support minimum configurations without the need for an entire Spring configuration setup.
- ❑ Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring based application.

What is Micro Service?

- ❑ Micro Service is an architecture that allows the developers to develop and deploy services independently.
- ❑ Each service running has its own process and this achieves the lightweight model to support business applications.

Micro services offers the following advantages to its developers

- ❑ Easy deployment
- ❑ Simple scalability
- ❑ Compatible with Containers
- ❑ Minimum configuration
- ❑ Lesser production time

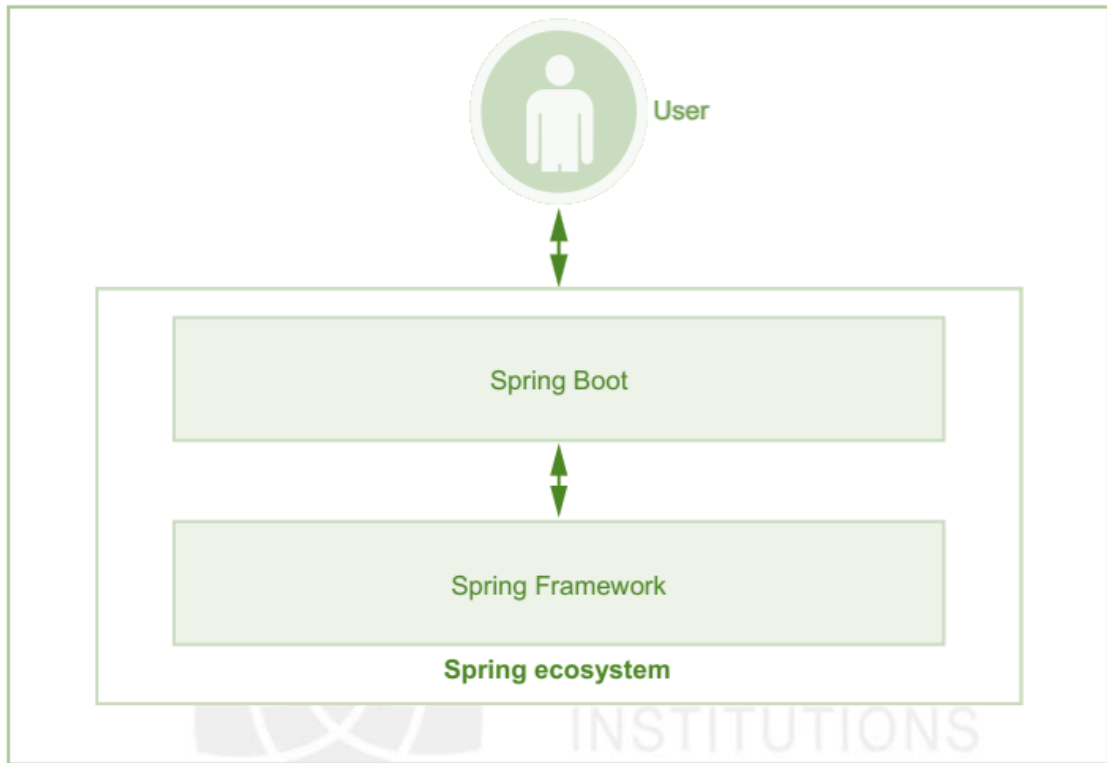
9. Lecture Notes

Spring Boot offers the following advantages to its developers:

- ❑ The Spring Framework started its journey to simplify the Java Enterprise application development.
- ❑ Support to the developers in focusing only on solving their business problems
- ❑ Allows developers to focus more on the business logic rather than the technical code and associated configurations.
- ❑ Spring Boot intends to create Spring-based, production-ready, standalone applications.
- ❑ Easy to understand and develop spring applications
- ❑ Increases productivity
- ❑ Reduces the Development Time
- ❑ To avoid complex XML configuration in Spring.
- ❑ Spring Boot is nothing but an existing framework with the addition of an embedded HTTP server and annotation configuration.
- ❑ It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- ❑ It provides a powerful batch processing and manages REST endpoints.
- ❑ In Spring Boot, everything is auto configured; no manual configurations are needed.
- ❑ It offers annotation-based spring application
- ❑ Eases dependency management
- ❑ It includes Embedded Servlet Container

9. Lecture Notes

Figure 1.1 : Developer view of Spring Boot. It sandwiches itself between the developer and the Spring Framework. Several Spring Framework components are automatically configured by Spring Boot based on the Spring components a developer uses



Spring Boot Core Features

Fast bootstrapping

- ❑ Spring Boot is to provide a fast startup experience in Spring application development.
- ❑ With Spring Boot, can generate an application by specifying the dependencies we need in our application, and Spring Boot takes care of the rest.

Autoconfiguration

- ❑ Spring Boot automatically configures the bare minimum components of a Spring application.
- ❑ For instance, if Spring Boot detects the presence of a database driver JAR file (e.g., H2 in-memory database JAR) in the classpath, it automatically configures the corresponding data source to connect to the database.

9. Lecture Notes

Standalone

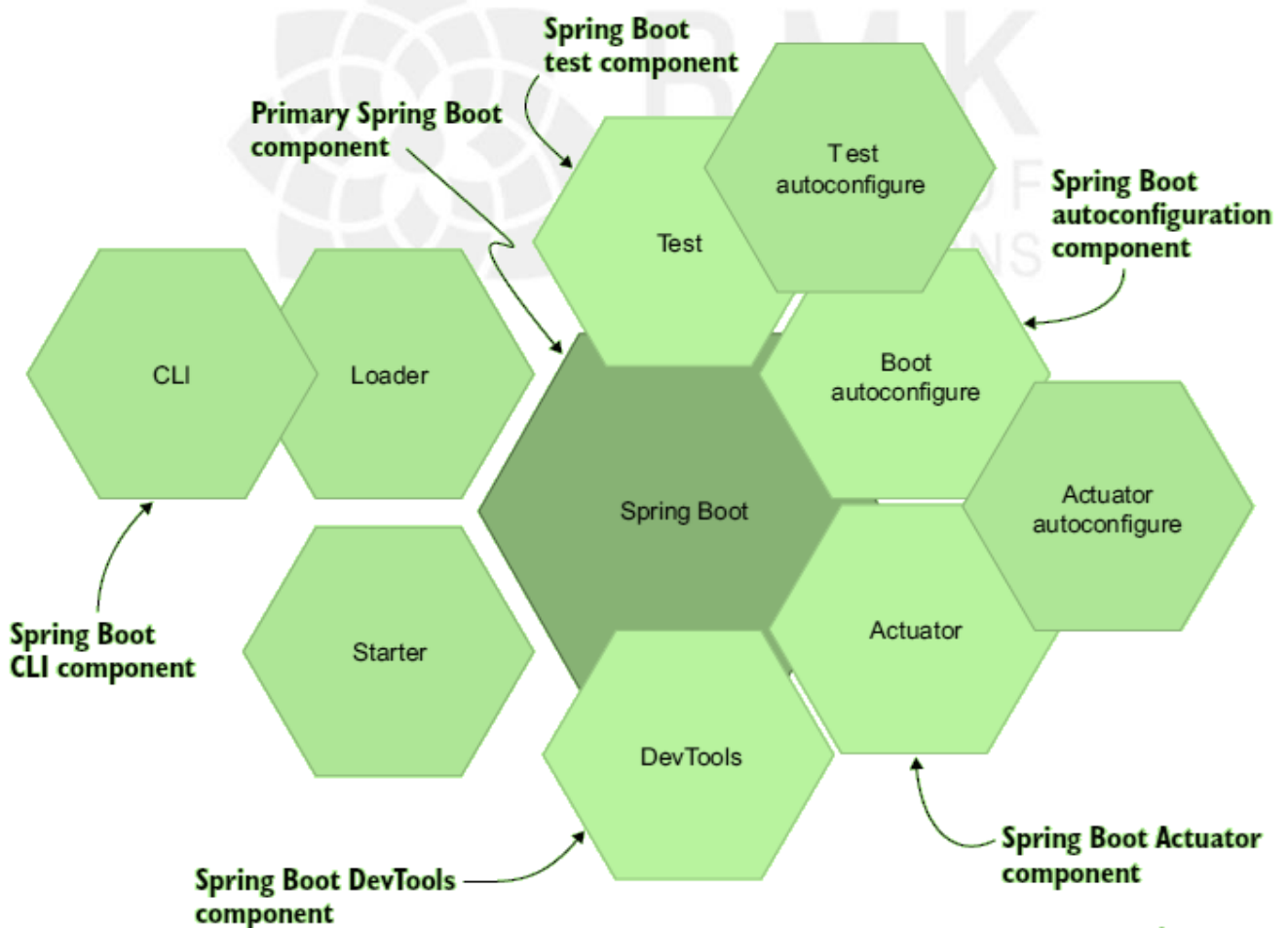
- ❑ Spring Boot applications embed a web server, so they can run standalone
- ❑ And do not necessarily require an external web or application server.

Production-ready

- ❑ Spring Boot provides several useful production-ready features out of the box to monitor and manage the application once it is pushed to production.

Spring Boot Components

Spring Boot consists of several components with each component focusing on a specific area of the application development. **Figure 1.2** below shows the Spring Boot components,



9. Lecture Notes

1. **Spring Boot** is the primary component used in almost every Spring Boot project.
 - ❑ It contains the `SpringApplication` class, which contains several static methods to create a standalone Spring Boot application (not required internet connection or any server access).
 - ❑ It also provides support for embedded web servers (e.g., Tomcat) and supports externalized application configurations (e.g., database details of application), etc.
 - ❑ `spring-boot-autoconfigure` component provides the necessary support for the automatic configuration of a Spring Boot application.
 - ❑ Spring Boot autoconfiguration guesses and configures the spring beans based on the dependencies.
 - ❑ However, autoconfiguration backs away from the default configuration if it detects user-configured beans with custom configurations.
 - ❑ Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project.
 - ❑ For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.
 - ❑ For this purpose, you need to add `@EnableAutoConfiguration` annotation or `@SpringBootApplication` annotation to your main class file. Then, your Spring Boot application will be automatically configured.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
```

@EnableAutoConfiguration

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

9. Lecture Notes

3. Spring-boot-starters are a set of **prepackaged** dependency descriptors provided for **developer convenience**.

- ❑ A Spring Boot starter assists in **providing** a set of Spring and **related technologies** to the developer.
- ❑ Which otherwise, the developer needs to manage **themselves**.

Spring Boot Starters

- ❑ Spring Boot Starters are dependency descriptors that can be added under the <dependencies> section in pom. xml.
- ❑ Handling dependency management is a difficult task for big projects.
- ❑ Spring Boot resolves this problem by providing a set of dependencies for developers convenience.
- ❑ For example, if you want to use Spring and JPA for database access, it is sufficient if you include spring-boot-starter-data-jpa dependency in your project.
- ❑ Note that all Spring Boot starters follow the same naming pattern spring-boot-starter- *, where * indicates that it is a type of the application.

Spring Boot starters explained below for a better understanding .

Spring Boot Starter Actuator dependency is used to monitor and manage your application. Its code is shown below.

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

Spring Boot Starter Security dependency is used for Spring Security.

Its code is shown below –

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

9. Lecture Notes

Spring Boot Starter web dependency is used to write a Rest Endpoints. Its code is shown below

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

Spring Boot Starter Thyme Leaf dependency is used to create a web application. Its code is shown below

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

Spring Boot Starter Test dependency is used for writing Test cases. Its code is shown below

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

4. spring-boot-CLI is a developer-friendly command-line utility that compiles and runs groovy codes.
- ❑ It can also watch files for changes, so you do not need to restart your application on modifications.
 - ❑ This CLI tool exempts from the need for dependency management tools, such as Maven or Gradle.
 - ❑ Also, it lets developers quickly prototype Spring applications without worrying much about dependency management and other builds-related issues

9. Lecture Notes

5. **spring-boot-actuator** component provides the actuator endpoints to interact with, monitor, and audit a Spring Boot application.
- ❑ An actuator in Spring Boot can be managed through JMX or HTTP endpoints.
 - ❑ Spring Boot provides a predefined list of actuator endpoints that cover a range of application aspects.
 - ❑ If that does not satisfy your need, you can also create your custom actuator endpoints specific to your application.
6. **spring-boot-actuator-autoconfigure** component provides support to autoconfigure the actuator endpoints based on the classpath.
- ❑ For instance, if the Micrometer (<https://micrometer.io>) dependency is present in the classpath, Spring Boot automatically configures the MetricsEndpoint.
7. **spring-boot-test** module contains **annotations** and **methods** to write **test cases** for the Spring Boot.
8. **spring-boot-test-autoconfigure** component supports the **autoconfiguration** of the **test cases** of your application.
9. **spring-boot-loader** component allows a Spring Boot application to be **packaged** as a single fat JAR file, **including all dependencies and the embedded web servers** that can be run standalone.
10. **spring-boot-devtools** component contains an additional developer toolkit for a smooth development experience of Spring Boot applications.

Spring Boot Coding Details

Build System

- ❑ Spring Boot lets you create a Spring Boot project with either **Apache Maven** ([https:// maven.apache.org/](https://maven.apache.org/)) or **Gradle** (<https://gradle.org/>) build tools.
- ❑ In the **Spring Initializr** (<https://start.spring.io/>) tool, you can choose the build system of your choice and generate the project.

Programming Language

- ❑ Both Java and Kotlin (<https://kotlinlang.org/>) programming languages in your Spring Boot project.

9. Lecture Notes

Database

- ❑ Spring Boot extends support to an array of SQL and NoSQL databases

Lombok

Lombok (<https://projectlombok.org/>) is a Java library that automatically generates the constructors, getter, setter, toString, and others based on the presence of a few annotations in the plain old Java object (POJO) class.

Spring Vs Spring Boot

Below are some key points which spring boot offers but spring doesn't:

- ❑ Starter POM.
- ❑ Version Management.
- ❑ Auto Configuration.
- ❑ Component Scanning.
- ❑ Embedded server.
- ❑ InMemory DB.
- ❑ Actuators

9. Lecture Notes

Getting started with spring Boot

- ❑ Spring boot is a backend development framework based on the Spring framework.
- ❑ Spring Boot is an opinionated addition to the Spring platform, focused on convention over configuration — highly useful for getting started with minimum effort and creating standalone, production-grade applications.
- ❑ We don't have to do much configuration ourselves, as it auto-configures based on the dependencies we add while creating a project so we can focus on the actual project.
- ❑ This will show you how to set up Spring boot and print your first "Hello World!" in your browser.
- ❑ First, you should have some basic understanding of Java and have Java 8 (JDK 1.8) or higher installed on your PC. Also need an IDE or text editor.

Create your Application

There are 3 ways to do this:

1. Spring Initializr
2. Spring Tool Suite (STS)
3. Spring CLI

Spring Initializr

- ❑ Spring Boot provides a tool called Spring Initializr that lets to generate a skeleton Spring Boot project.
- ❑ We can access the Spring Initializr tool at <https://start.spring.io>.
- ❑ The Initializr layout features basically looks like this:

The screenshot displays the Spring Initializr web interface. At the top, there's a navigation bar with the Spring logo and 'spring initializr' text. Below this, the interface is divided into several sections:

- Project:** Includes radio buttons for 'Maven Project' (selected) and 'Gradle Project'.
- Language:** Includes radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'.
- Spring Boot:** Includes radio buttons for '2.3.0 RC1', '2.3.0 (SNAPSHOT)', '2.2.7 (SNAPSHOT)' (selected), and '2.1.13'.
- Project Metadata:** Includes input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo).
- Packaging:** Includes radio buttons for 'Jar' (selected) and 'War'.
- Dependencies:** Includes a text input field and a button 'ADD DEPENDENCIES... CTRL + B'. Below it, it says 'No dependency selected'.

At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'. The R.M.K. GROUP OF INSTITUTIONS logo is visible in the bottom right corner.

9. Lecture Notes

Spring Boot project structure

A generated Spring Boot project structure is relatively **simple** and consists of **only the component required** to proceed with Spring Boot application development.

It contains the following components:

1. A **pom.xml** file that contains the dependencies selected during project generation.
2. A **Maven wrapper** file that lets to build the project without installing Maven in local machine.
3. A package structure that contains the **source and tests Java files**. The source package contains a **Java class** with **the main method**, and the **test package** has an **empty test class**.
4. A resources folder to maintain additional project **artifacts and an empty application.properties file**.

The **pom.xml** file of generated Spring Boot project

There are three segments of the pom.xml:

1. **The parent tag**
2. **The dependencies section**
3. **The Spring Boot Maven plugin**

The spring-boot-starter-parent is the parent dependency for all Spring Boot starter dependencies.

A spring-boot-starter-parent is a special type of starter dependency that provides several default configurations, such as the default java version and default configurations

Further, spring-boot-starter-parent also assists in dependency management for several Maven plugins to a Spring Boot project.

Building an Application with Spring Boot

<https://spring.io/guides/gs/spring-boot/>

Spring Boot project The pom.xml file of generated Spring Boot project is Illustrated in Figure 1.3



9. Lecture Notes

Figure 1.3 The pom.xml file of generated Spring Boot project

Current project declares Spring Boot starter parent as its parent to indicate that this project is a child Spring Boot project. This ensures several features of the application, such as plugin and dependency management, can be managed by Spring Boot.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.3</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.manning.sbp.ch01</groupId>
  <artifactId>spring-boot-app-demo</artifactId>
  <version>1.0.0</version>
  <name>spring-boot-app-demo</name>
  <description>Spring Boot Demo Application</description>
  <properties>
    <java.version>17</java.version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
      <exclusions>
        <exclusion>
          <groupId>org.junit.vintage</groupId>
          <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Current project's
artifact details

Spring Boot starter test dependency provides necessary support to perform testing Spring Boot applications with popular testing libraries, such as Junit, Hamcrest, and Mockito. This dependency excludes junit-vintage-engine dependency to leverage Junit 5 features with junit-jupiter-engine.

List of
declared Maven
dependencies:
Spring Boot
starter web and
Spring Boot
starter test

Spring Boot Maven plugin is a Maven plugin that provides useful goals to perform several application management activities. For instance, you can quickly start the Spring Boot application with this plugin using mvn spring-boot:run command.

Activ
Go to

9. Lecture Notes

To run the current Spring Boot application, you can execute the following command in command-line or terminal from the same directory where pom.xml is located: `mvn spring-boot:run`. You'll see the application starts and runs on default HTTP port 8080, as shown in **Figure 1.4**

```
C:\sbip\repo\ch01\spring-boot-app-demo>mvn spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----< com.manning.sbip.ch01:spring-boot-app-demo >-----
[INFO] Building spring-boot-app-demo 0.0.1-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.6.3:run (default-cli) > test-compile @ spring-boot-app-demo >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ spring-boot-app-demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ spring-boot-app-demo ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ spring-boot-app-demo ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory C:\sbip\repo\ch01\spring-boot-app-demo\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ spring-boot-app-demo ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\sbip\repo\ch01\spring-boot-app-demo\target\test-classes
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.6.3:run (default-cli) < test-compile @ spring-boot-app-demo <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:2.6.3:run (default-cli) @ spring-boot-app-demo ---
[INFO] Attaching agents: []
```

The Spring Boot Main Class

In the generated project, you can find that **Spring Initializr** has generated a Java class with a **Java main() method** in it. The following listing shows this.

```
package com.manning.sbip.ch01;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringBootApplication
{
    public static void main(String[] args)
    {
        SpringApplication.run(SpringBootApplication.class, args);
    }
}
```

Let's examine the following components of the generated Java file:

1. Using the **main()** method
2. Using the **@SpringBootApplication** annotation
3. The role of **SpringApplication class**



9. Lecture Notes

Common Spring Boot task-Managing Configuration

Annotations

- ❑ Spring Boot automatically configures your application based on the dependencies you have added to the project by using `@EnableAutoConfiguration` annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.
- ❑ The entry point of the spring boot application is the class contains `@SpringBootApplication` annotation and the main method.
- ❑ Spring Boot automatically scans all the components included in the project by using `@ComponentScan` annotation.

What does the `@SpringBootApplication` annotation do internally?

The `@SpringBootApplication` annotation is equivalent to using `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` with their default attributes. Spring Boot enables the developer to use a single annotation instead of using multiple. Just like any other Java program, a Spring Boot application must have a main method. This method serves as an entry point, which invokes the `SpringApplication.run` method to bootstrap the application.

@SpringBootApplication

```
public class MyApplication
{
    public static void main(String[] args)
    { SpringApplication.run(MyApplication.class);
    // other statements
    }
}
```

What is the purpose of using `@ComponentScan` in the class files?

Spring Boot application scans all the beans and package declarations when the application initializes. You need to add the `@ComponentScan` annotation for your class file to scan your components added to your project.

9. Lecture Notes

What is the purpose of using @EnableAutoConfiguration?

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 developer.

System Requirements

- ❑ Spring Boot 3.2.1 requires Java 17 and is compatible up to and including Java 21.
- ❑ Spring Framework 6.1.2 or above.

Build Tool	Version
------------	---------

- | | | |
|---|--------|----------------------------|
| ❑ | Maven | 3.6.3 or later |
| | Gradle | 7.x (7.5 or later) and 8.x |

- ❑ . Spring Boot supports the following embedded servlet containers

Name	Servlet Version
------	-----------------

Tomcat 10.1	6.0
Jetty 12.0	6.0
Undertow 2.3	6.0

- ❑ Maven and Gradle are build tools used by Spring. Selecting either of them will automatically add the build file to project, and when you add Dependencies (right of the screen), it would auto-configure.
- ❑ Select the Spring boot version. I'd advice you leave whichever one is selected already. It's likely the latest and most compatible.
- ❑ Add your site in Group(or simply leave it as example.com), then give it any name you'd like (same for Artifact).
- ❑ Leave the Packaging as Jar. This can be related to a zip file for packaging the project.
- ❑ Select the Java version that is installed on your PC.
- ❑ Add dependencies. Since we're working on a simple web application, a web dependency would be fine. Click on "add dependency" and search for "Spring Web". This would automatically be added to your files.
- ❑ Click "Generate". This will create a zip folder containing all necessary files including dependencies you have chosen.
- ❑ Unzip it and add it to your IDE workspace and you're all set!

9. Lecture Notes

Creating Custom Properties

Managing application configuration is a key part of any application. Depending on how you develop and manage applications, you can have multiple environments (e.g., dev, test, staging, and prod) for an application in your organization.

For instance, you can have one environment for development, one for testing, one for staging, and one for production. For all these environments, your application code mostly remains the same, and you need to manage many different configurations based on the environment.

As an example, the database configurations or the security configurations are different in all these environments. Besides, as the application grows, and you incorporate new features, it becomes more tedious to manage the configurations.

Spring Boot provides several approaches to let you externalize application configurations without altering the application source code. The various approaches include property files, YAML files, environment variables, and command-line arguments.

CONFIGURATION MANAGEMENT WITH THE APPLICATION PROPERTIES FILE

- ❑ Spring Boot provides a power and flexible mechanism for application configuration using the application.properties file.
- ❑ This mechanism provides the flexibility to configure and change the application behaviour without changing the code in our application.
- ❑ The default application.properties comes up with a large list of configurations to boot strap our application.
- ❑ Spring Boot provide the option to change or override the application behaviour by overriding these configuration properties.
- ❑ Also provides a powerful mechanism to inject the custom properties in our application using the **application.properties** file.

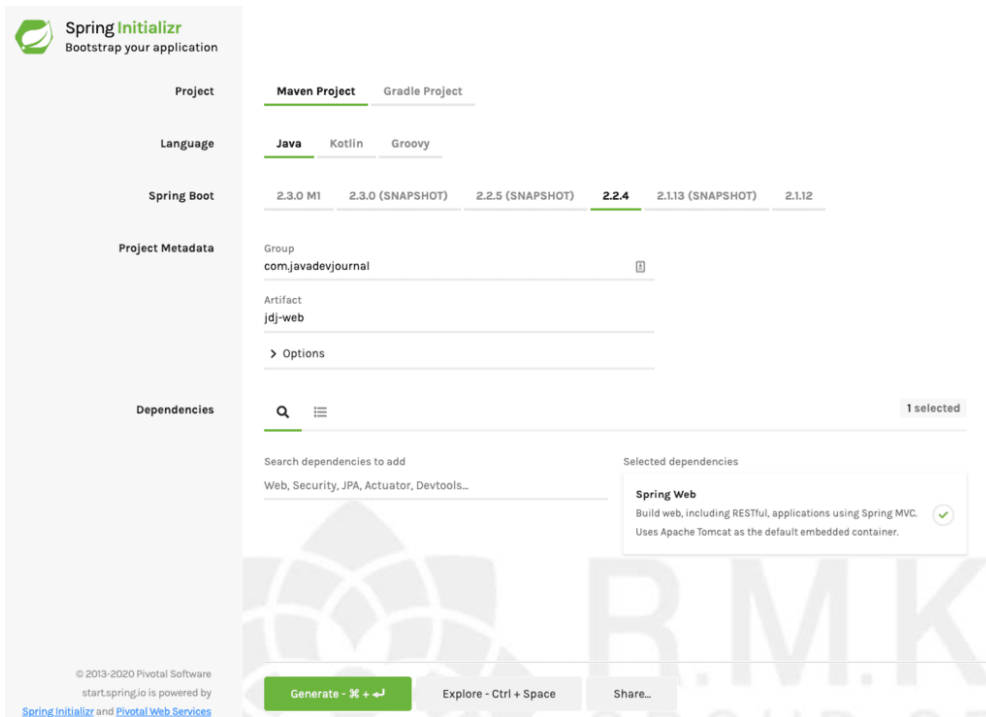
1. application.properties File.

- ❑ The **application.properties** file is a simple property file with a key-value information to configure or externalize our application properties. Spring Boot provides multiple options to bundle this file in the application.
 - Bundle it with the jar.
 - Load from the file system on startup.
- ❑ Think of the property file as the central control unit for your application. This file is useful for:
 - Customize or override the default Spring framework behaviour (e.g. changing the server port, or timeout or caching).
 - Custom properties to control our application (defining the username and password for API integration).

9. Lecture Notes

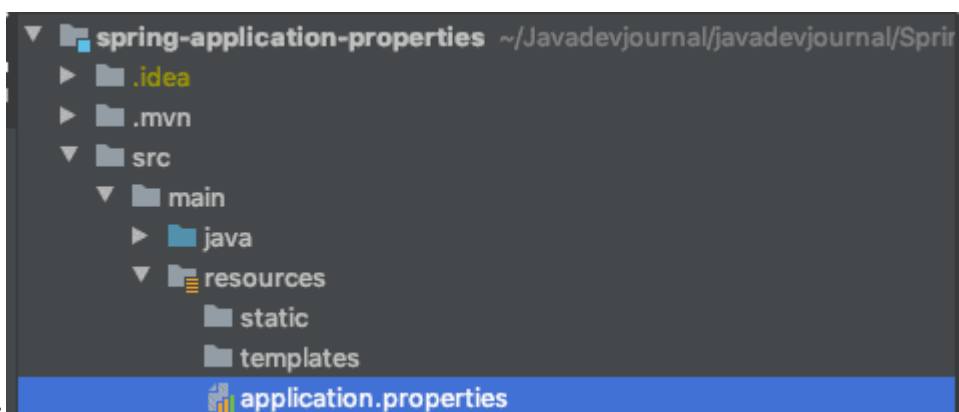
2. Setting up Application

Let's start our journey by creating a simple web application. We can use the IDE or Spring Initializr to bootstrap our application.



The image shows the Spring Initializr web interface. On the left, there's a sidebar with sections: Project, Language, Spring Boot, Project Metadata, and Dependencies. The main area is titled 'Maven Project' and 'Java'. Under 'Spring Boot', version '2.2.4' is selected. In 'Project Metadata', the group is 'com.javadevjournal' and the artifact is 'jdj-web'. Under 'Dependencies', 'Spring Web' is selected. At the bottom, there's a 'Generate' button, an 'Explore' button, and a 'Share' button. A watermark 'B.M.K GROUP OF INSTITUTIONS' is visible in the background.

Click on the “Generate” button to download the project structure in your local machine. The next step is to import the project in the Java editor. Our Spring Boot configuration file will be available under the **src/main/resources** directory.



By default, this file will be empty (values need to be added). Spring also support the property configuration using the **.yml** file. If you prefer the **.yml**, create **application.yml** file in the same file location. We are using the **.properties** type **Don't mix property and yml convention. Choose one and stick to that.**

9. Lecture Notes

Let's add a custom property to in the **application.properties** file

javadevjournals.welcome.message= A warm greeting from Javadevjournals Team!!:

3. Property Injection Using @Value Annotation

The most common way to inject these properties are through the **@Value** annotation. We have the option to use this annotation in

In the constructors

On the Bean fields.

Let's create a REST controller and provide a configurable welcome message to all customer:

```
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WelcomeController {
```

```
    // We are injecting the configuration message from the application.properties file
    using @Value annotation
```

```
    @Value("${javadevjournals.welcome.message}")
    private String welcomeMsg;

    /**
     * Our Welcome display message which will use the welcome message property
    injected through the
     * @Value annotation.welcome
     * @return welcome message
     */
    @GetMapping("/welcome")
    public String displayWelcomeMsg() {
        return welcomeMsg;
    }
}
```

When we run our application, our welcome controller will return the property injected from the application.properties file through @Value annotation.

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/welcome
- Params:** (empty)
- Buttons:** Send, Save
- Authorization:** No Auth
- Body:** A warm greeting from Javadevjournals Team!!
- Status:** 200 OK
- Time:** 243 ms
- Headers:** (5)
- Test Results:** (empty)
- Footer:** R.M.K. GROUP OF INSTITUTIONS

9. Lecture Notes

3.1. Using Constructor Argument.

We have the option to use the @Value annotation to pass as the constructor argument. Let's take an example, where we want to pass a default value to the constructor:

```
public void DefaultWelcomeService(@Value("${javadevjournal.init.secret.key}") String
secretKey) {

    this.secretKey = secretKey;

    LOG.info("@Value annotation is working for our secret key {}", secretKey);

}
```

If Spring doesn't find the key you want to inject, it'll throw IllegalArgumentException

Spring Boot is flexible enough to provide an option to handle the IllegalArgumentException in case the property is missing. We can pass an option value in case the property is missing in the application.properties file. We can pass the default value by adding colon (:) after the key followed by the default value.

```
@Value("${javadevjournal.welcome.message: Welcome!!!}")
private String welcomeMsg;
```

4. Using @ConfigurationProperties

This annotation is helpful if our properties have some common context. Consider following entries in the property file.

```
user.firstName = Demo
user.lastName = User
user.greeting = Hello Stranger
user.blogName = javadevjournal.com
```

If need to use these property files in the Spring application.

```
public class SimpleSpringPropertyTest
{
    @Value("${user.firstName}") private String firstName;
    @Value("${user.lastName}") private String lastName;
}
```

9. Lecture Notes

@Value("\${properties}") annotation is handy and easy to use, but it will really be a very tedious process if we have several properties. Spring Boot has introduced the @ConfigurationProperties annotation to handling these properties in a more clean way with an option to validate these configurations value.

#Database Configuration

db.driver =org.hsqldb.jdbcDriver

db.username =test

db.password =test

db.tablePrefix =_prefix

#SMTP Configuration

mail.from =test@test.com

mail.host =test@test.com

mail.port =25

mail.security.userName =test

mail.security.password =test

#Server Configurations

server.tomcat.httpPort =80

server.tomcat.sslPort =443

server.tomcat.ajpPort =444

server.tomcat.jmxPort =445

Let's see how to set up email configurations without injecting individual properties:

@Configuration

@ConfigurationProperties(prefix = "mail")

public class ApplicationConfigurationProp

{

private String from;

private String host;

private int port;

//getter and setter

public static class Security

{

private String userName;

private String password;

//getter and setter

}

}

Once we run above application, all properties defined in the property files with prefix "mail" will automatically be bind /assigned to this object

9. Lecture Notes

5. Overriding Default Properties

To override the properties defined in the default application.properties file, we just need to define the property in our project configuration file with custom value.

Spring Boot load these property files in certain order and it will make sure that the configuration defined in project application.properties file take precedence.

Let's take an example, where we like to change the default port of the tomcat, add the following property in the project configuration file:

```
server.port = 8090
```

6. Multiple Lines in Property File

If our property has a long value, we can add backslash character to break it in multi-line and improve the overall readability of the property. Let's see how to do this in the application.properties file:

```
javadevjournal.welcome.message = A warm and long greeting from Javadevjournal \
    Team!! to show how we can use the backslash character to improve the overall \
    readability of the file.
```

7. Custom Properties Type Conversion

All properties defined in the application.properties file is of type String (it's a text file). Spring framework comes a long list of type convertors to convert string to other types based on the type declared in the application. Let's look at the following example:

```
javadevjournal.max.login.retry=3
javadevjournal.enable.guest.checkout=true
```

Spring detects variable type automatically and will perform type conversion before injection;

```
public void DefaultWelcomeService(@Value("${javadevjournal.init.secret.key}") String
secretKey, @Value("${javadevjournal.max.login.retry}") int retry,
@Value("${javadevjournal.enable.guest.checkout}") boolean enableGuestCheckout) {
    this.secretKey = secretKey;
    LOG.info("@Value annotation is working for our secret key {}", secretKey);
}
```

9. Lecture Notes

8. Spring Profile (Environment Specific Files)

Spring Profiles provides a powerful and easy way to control code and configuration based on the environment. Using Spring Profiles its possible to segregate parts of our application and make it only available in certain environments. One of the most interesting and powerful features provided by Spring Boot is the ability to define profile specific application.properties file and active these by main application.properties file.

To use profile specific configuration files, we need to the naming convention of application-{profile}.properties where profile defines the name of the intended profile. It will load profile files from the same location as application.properties file.

application-local.properties

application-dev.properties

application-staging.properties

application-prod.properties

You can define properties as per your requirements. Use the spring.profiles.active property to help Spring Boot choose the correct configurations for us.

spring.profiles.active=staging

We are setting the active profile as staging. With above setting,, Spring Boot will load the properties defined in the application-staging.properties besides the main application.properties file. For more detail, read Spring Profiles

Reference : <https://www.javadevjournals.com/spring-boot/spring-boot-configuration-properties/>

9. Lecture Notes

Executing code on Spring Boot application startup

Spring Boot is a popular framework for building web applications in Java. It simplifies the development process by providing a range of features and functionalities out of the box. One such feature is the ability to run code on application startup. This can be useful for initializing resources, loading data, and performing other tasks that need to be executed once the application has started.

Spring Boot offers at least 5 different ways of executing code on startup.

1. CommandLineRunner - Implementing CommandLineRunner Interface
2. ApplicationRunner - Implementing ApplicationRunner Interface
3. ApplicationListener - Using EventListener Annotation
4. @PostConstruct - Using PostConstruct Annotation
5. InitializingBean - Implementing InitializingBean Interface

6. CommandLineRunner - Implementing CommandLineRunner Interface

CommandLineRunner is a simple interface we can implement to execute some code after the Spring application has successfully started up:

```
@Component
```

```
@Order(1)
```

```
class MyCommandLineRunner implements CommandLineRunner
```

```
{
```

```
private static final Logger logger = LoggerFactory.getLogger(MyCommandLineRunner.class);
```

```
    @Override
```

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

```
        {
```

```
            if(args.length > 0)
```

```
            {
```

```
                logger.info("first command-line parameter: '{}'", args[0]);
```

```
            }
```

```
        }
```

```
    }
```

9. Lecture Notes

Executing code on Spring Boot application startup

When Spring Boot finds a `CommandLineRunner` bean in the application context, it will call its `run()` method after the application has started up and pass in the command-line arguments with which the application has been started.

We can now start the application with a command-line parameter like this:

```
java -jar application.jar --foo=bar
```

This will produce the following log output:

```
first command-line parameter: '--foo=bar'
```

As we can see, the parameter is not parsed but instead interpreted as a single parameter with the value `--foo=bar`. We'll later see how an `ApplicationRunner` parses arguments for us.

Note the Exception in the signature of `run()`. Even though we don't need to add it to the signature in our case, because we're not throwing an exception, it shows that Spring Boot will handle exceptions in our `CommandLineRunner`. Spring Boot considers a `CommandLineRunner` to be part of the application startup and will abort the startup when it throws an exception.

Multiple `CommandLineRunner` beans can be defined within the same application context and can be ordered using the `@Ordered` interface or `@Order` annotation.

2. ApplicationRunner - Implementing ApplicationRunner Interface

We can use an `ApplicationRunner` instead if we want the command-line arguments parsed:

```
@Component
```

```
@Order(2)
```

```
class MyApplicationRunner implements ApplicationRunner
```

```
{
```

```
    private static final Logger logger = LoggerFactory.getLogger(MyApplicationRunner.class);
```

```
    @Override
```

```
    public void run(ApplicationArguments args) throws Exception
```

```
{
```

```
    logger.info("ApplicationRunner#run()");
```

```
    logger.info("foo: {}", args.getOptionValues("foo"));
```

```
}
```

```
}
```


9. Lecture Notes

Executing code on Spring Boot application startup

The `ApplicationArguments` object gives us access to the parsed command-line arguments. Each argument can have multiple values because they might be used more than once in the command-line. We can get an array of the values for a specific parameter by calling `getOptionValues()`.

Let's start the application with the `foo` parameter again:

```
java -jar application.jar --foo=bar
```

The resulting log output looks like this:

```
foo: [bar]
```

As with `CommandLineRunner`, an exception in the `run()` method will abort application startup and several `ApplicationRunners` can be put in sequence using the `@Order` annotation. The sequence created by `@Order` is shared between `CommandLineRunners` and `ApplicationRunners`.

3. `ApplicationListener` - Using `EventListener` Annotation

If we don't need access to command-line parameters, we can tie our startup logic to Spring's `ApplicationReadyEvent`:

```
@Component
```

```
@Order(0)
```

```
class MyApplicationListener
```

```
    implements ApplicationListener<ApplicationReadyEvent> {
```

```
    private static final Logger logger = ...;
```

```
    @Override
```

```
    public void onApplicationEvent(ApplicationReadyEvent event) {
```

```
        logger.info("ApplicationListener#onApplicationEvent()");
```

```
    }
```

```
}
```

The `ApplicationReadyEvent` is fired only after the application is ready (duh) so that the above listener will execute after all the other solutions described have done their work.

9. Lecture Notes

Executing code on Spring Boot application startup

Multiple `ApplicationListeners` can be put in an order with the `@Order` annotation. The order sequence is shared only with other `ApplicationListeners` and not with `ApplicationRunners` or `CommandLineRunners`.

An `ApplicationListener` listening for the `ApplicationReadyEvent` is the way to go if we need to create some global startup logic without access to command-line parameters. We can still access environment parameters by injecting them with Spring Boot's support for configuration properties.

4. `@PostConstruct` - Using `PostConstruct` Annotation

Another simple solution to create startup logic is by providing an initializing method that is called by Spring during bean creation. All we have to do is to add the `@PostConstruct` annotation to a method:

```
@Component
@DependsOn("myApplicationListener")
class MyPostConstructBean
{
    private static final Logger logger = ...;
    @PostConstruct
    void postConstruct()
    {
        logger.info("@PostConstruct");
    }
}
```

This method will be called by Spring once the bean of type `MyPostConstructBean` has been successfully instantiated. The `@PostConstruct` method is called right after the bean has been created by Spring, so we cannot order it freely with the `@Order` annotation, as it may depend on other Spring beans that are `@Autowired` into our bean.

Instead, it will be called after all beans it depends on have been initialized. If we want to add an artificial dependency, and thus create an order, we can use the `@DependsOn` annotation (same warnings apply as for the `@Order` annotation!).

9. Lecture Notes

Executing code on Spring Boot application startup

A `@PostConstruct` method is inherently tied to a specific Spring bean so it should be used for the initialization logic of this single bean only. For global initialization logic, a `CommandLineRunner`, `ApplicationRunner`, or `ApplicationListener` provides a better solution.

5. InitializingBean - Implementing InitializingBean Interface

Very similar in effect to the `@PostConstruct` solution, we can implement the `InitializingBean` interface and let Spring call a certain initializing method:

`@Component`

`class MyInitializingBean implements InitializingBean`

```
{
    private static final Logger logger = ...;
    @Override
    public void afterPropertiesSet() throws Exception
    {
        logger.info("InitializingBean#afterPropertiesSet()");
    }
}
```

Spring will call the `afterPropertiesSet()` method during application startup. As the name suggests, we can be sure that all the properties of our bean have been populated by Spring. If we're using `@Autowired` on certain properties (which we shouldn't - we should use constructor injection instead), Spring will have injected beans into those properties before calling `afterPropertiesSet()` - same as with `@PostConstruct`.

With both `InitializingBean` and `@PostConstruct` we must be careful not to depend on state that has been initialized in the `afterPropertiesSet()` or `@PostConstruct` method of another bean. That state may not have been initialized yet and cause a `NullPointerException`.

9. Lecture Notes

Database access with Spring data

Spring Data (<https://spring.io/projects/spring-data>) lets to access data from a variety of data sources (e.g., relational and non-relational databases, MapReduce databases, and cloud-based data services). It attempts to provide a uniform, easy-to-use, and familiar programming model through the Spring Framework.

It is an umbrella project under the Spring Framework that contains several subprojects, each of which targeting a specific database. For instance, the Spring Data JPA module is specific to relational databases (e.g., H2, MySQL, PostgreSQL). Similarly, Spring Data MongoDB aims to provide support for the MongoDB database.

Java Persistence API (JPA)

Most applications in today's world need to communicate with the database to store and retrieve application data. And to achieve this interaction developers generally need to write a lot of boilerplate code. For instance, in the standard Java Database Connectivity (JDBC) approach, you need to obtain a database connection, define a PreparedStatement, set the bind variables, execute the query, and perform resource management.

The Java Persistence API (JPA) takes away most of these burdens and provides the developers with a bridge between the Java object model (e.g., business objects) and the relational database model (e.g., database tables). This mapping between Java objects and the relational model is popularly known as object-relational mapping (ORM) as illustrated in figure 1.5

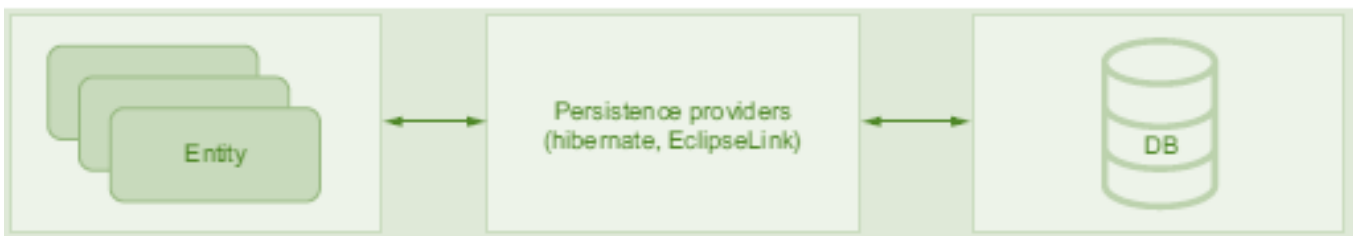


Figure 1.5: An overview of object-relational mapping. An entity represents a business object to be persisted. A persistence provider implements the JPA specification.

JPA is a specification that provides a set of interfaces, classes, and annotations to persist and retrieve application objects easily and concisely. Note that it is just a specification and outlines the standards for the ORM techniques. There are several third-party vendors, such as Hibernate and EclipseLink that provide a concrete implementation of this specification.

9. Lecture Notes

Database access with Spring data

Why Spring Data?

One of the core themes of Spring Data is to provide a consistent programming model to access various data sources. Thus, it provides a convenient API that lets to specify the metadata to the domain objects that need to be persisted and ensures that business domain objects are eligible to be persisted in the specific datastore.

For instance, we can use a relational database and Spring Data JPA to manage business objects. we can provide the JPA annotations in business objects, and Spring Data JPA ensures the domain object is persisted in the database table.

Spring Data modules

Spring Data is an umbrella project that provides support for several mainstream data stores.

Spring Data modules and their purposes

Spring Data Commons It contains the foundational components used in all Spring Data projects.

1. **Spring Data JDBC** This module provides repository support for JDBC.
2. **Spring Data JPA** It provides repository support for JPA.
3. **Spring Data MongoDB** It provides support for documents-based MongoDB database.
4. **Spring Data REDIS** It provides the necessary support for Redis datastore.
5. **Spring Data REST** It lets you access Spring data repositories as REST resources.
6. **Spring Data for Apache Cassandra** This module provides the necessary support for Apache Cassandra.

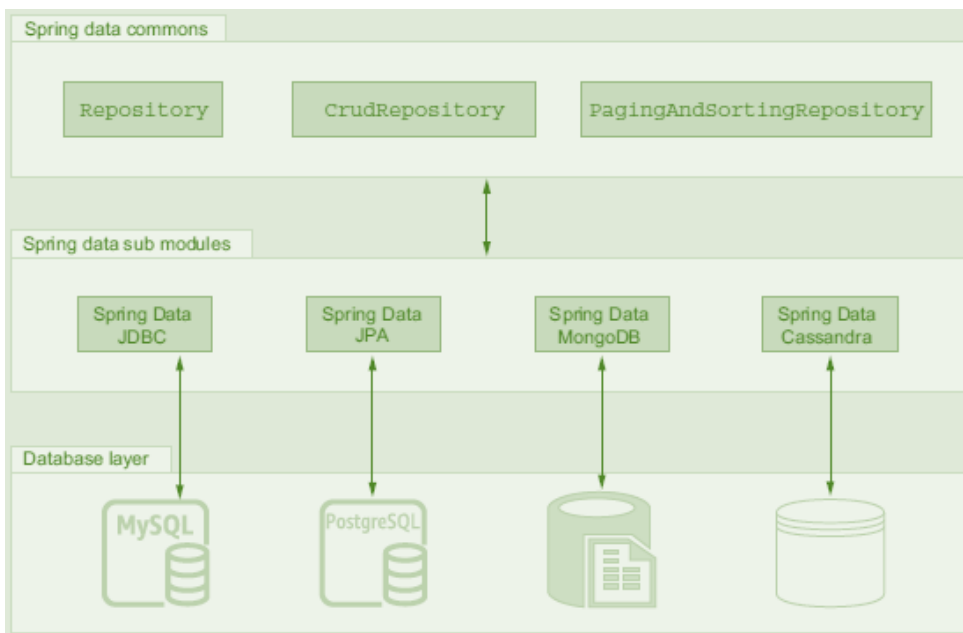


Figure 1.6 : Spring Data modules. The Spring Data Commons module provides a foundation upon which other submodules are based. Each submodule targets a specific type of database. The `Repository`, `CrudRepository`, and `PagingAndSortingRepository` are interfaces of the Spring Data Commons module.

9. Lecture Notes

Database access with Spring data

Spring Boot Data Access with Spring Data JPA and MySQL

Spring Data JPA is a powerful tool that simplifies database access in Spring Boot applications. In this tutorial, we'll create a Spring Boot application that uses Spring Data JPA to perform CRUD (Create, Read, Update, Delete) operations on a MySQL database.

Prerequisites

- ☐ Basic understanding of Java programming
- ☐ Familiarity with Spring Boot basics
- ☐ MySQL database installed and running

Step 1: Setting Up Your Spring Boot Project

1. Visit the Spring Initializer.
2. Choose your project's metadata (e.g., group, artifact, package name).
3. Select the latest stable version of Spring Boot.
4. Add the following dependencies:
 - Spring Web
 - Spring Data JPA
 - MySQL Driver
5. Click "Generate" to download the project structure as a ZIP file.

Step 2: Setting Up the Database

1. Create a MySQL database named library.
2. Update the application.properties (or application.yml) file in the src/main/resources directory with the database configuration:

`spring.datasource.url=jdbc:mysql://localhost:3306/library`

`spring.datasource.username=root`

`spring.datasource.password=your_password`

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

`spring.jpa.hibernate.ddl-auto=update`

`spring.jpa.show-sql=true`

Replace your_password with your actual MySQL password.

9. Lecture Notes

Database access with Spring data

Step 3: Creating the Entity

Create a new Java class named Book in the com.example.demo package:

```
package com.example.demo;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;

    // Getters and setters
}
```

This entity represents a book with id, title, and author properties

Step 4: Creating the Repository

Create a new Java interface named BookRepository in the com.example.demo package:

```
package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long>

{

}
```

JpaRepository provides basic CRUD operations for the Book entity.

9. Lecture Notes

Database access with Spring data

Step 5: Creating the Controller

Create a new Java class named BookController in the com.example.demo package:

```
package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@RestController
@RequestMapping("/books")
public class BookController {
    private final BookRepository bookRepository;

    @Autowired
    public BookController(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }

    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Long id) {
        return bookRepository.findById(id).orElse(null);
    }
}
```


9. Lecture Notes

Database access with Spring data

@PutMapping("/{id}")

```
public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {  
    if (bookRepository.existsById(id)) {  
        updatedBook.setId(id);  
        return bookRepository.save(updatedBook);  
    }  
    return null;  
}
```

@DeleteMapping("/{id}")

```
public void deleteBook(@PathVariable Long id) {  
    bookRepository.deleteById(id);  
}
```

This controller defines endpoints for CRUD operations on books.

Step 6: Running the Application

Open the DemoApplication class and run the main method to start the Spring Boot application.

Step 7: Testing the Application

Use a tool like Postman or a web browser to interact with the endpoints:

Get all books: GET <http://localhost:8080/books>

Add a book: POST <http://localhost:8080/books> with JSON body:

```
{  
    "title": "Spring Boot in Action",  
    "author": "Craig Walls"  
}
```

Get a book by ID: GET <http://localhost:8080/books/{id}>

Update a book: PUT <http://localhost:8080/books/{id}> with JSON body:

```
{  
    "title": "Updated Title",  
    "author": "Updated Author"  
}
```

Delete a book: DELETE <http://localhost:8080/books/{id}>

9. Lecture Notes

Database access with Spring data

@Query

@Query Annotation is used for defining custom queries in Spring Data JPA. When you are unable to use the query methods to execute database operations then you can use @Query to write a more flexible query to fetch data. Some of the points to be remembered in @Query Annotation are mentioned below

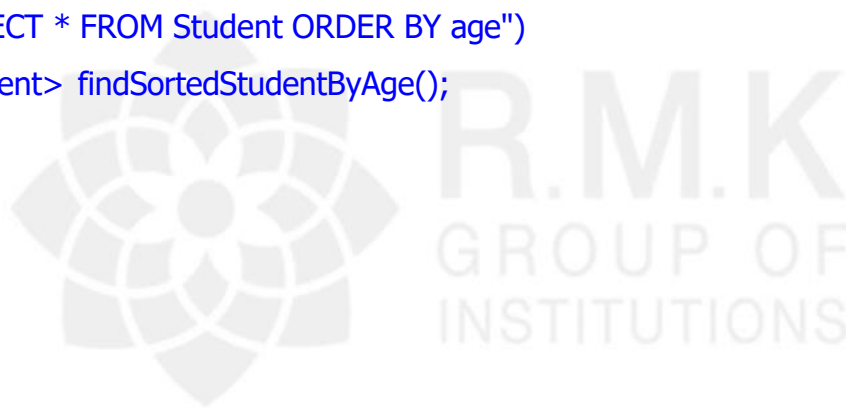
- ❑ @Query Annotation supports both JPQL and native SQL queries.
- ❑ It also supports SpEL expressions.
- ❑ @Param in method arguments to bind query parameter.

Examples:

JPQL:

```
@Query("SELECT * FROM Student ORDER BY age")
```

```
Optional<Student> findSortedStudentByAge();
```



9. Lecture Notes

Securing Spring Boot application.

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

Features

- ❑ Comprehensive and extensible support for both Authentication and Authorization
- ❑ Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
- ❑ Servlet API integration
- ❑ Optional integration with Spring Web MVC

Security features offered by Spring Security in a Spring Boot application:

- ❑ Spring Security enforces the application users to be authenticated before accessing it.
- ❑ If the application does not have a login page, Spring Security generates a default login page for user login and allows the user to log out from the application.
- ❑ Spring Security provides a default user named user and generates a default password (printed in the console log) for form-based login.
- ❑ Spring Security provides several password encoders to encode the plain-text password and store it in the persistence storage.
- ❑ Spring Security prevents session fixation attacks by changing the session ID after successful user authentication.
- ❑ Spring Security provides default protection from cross-site request forgery (CSRF) attacks. It does so by including a randomly generated token in the HTTP response. It expects this token to be available in all subsequent formbased requests that intend to perform a state-changing operation in the application. A malicious user won't have access to the token and, thus, can't make CSRF attacks. Figure 1.7 demonstrates the CSRF protection with Spring Security.
- ❑ By default, Spring Security includes several HTTP response headers that prevent many common types of attacks.

9. Lecture Notes

Securing Spring Boot application.

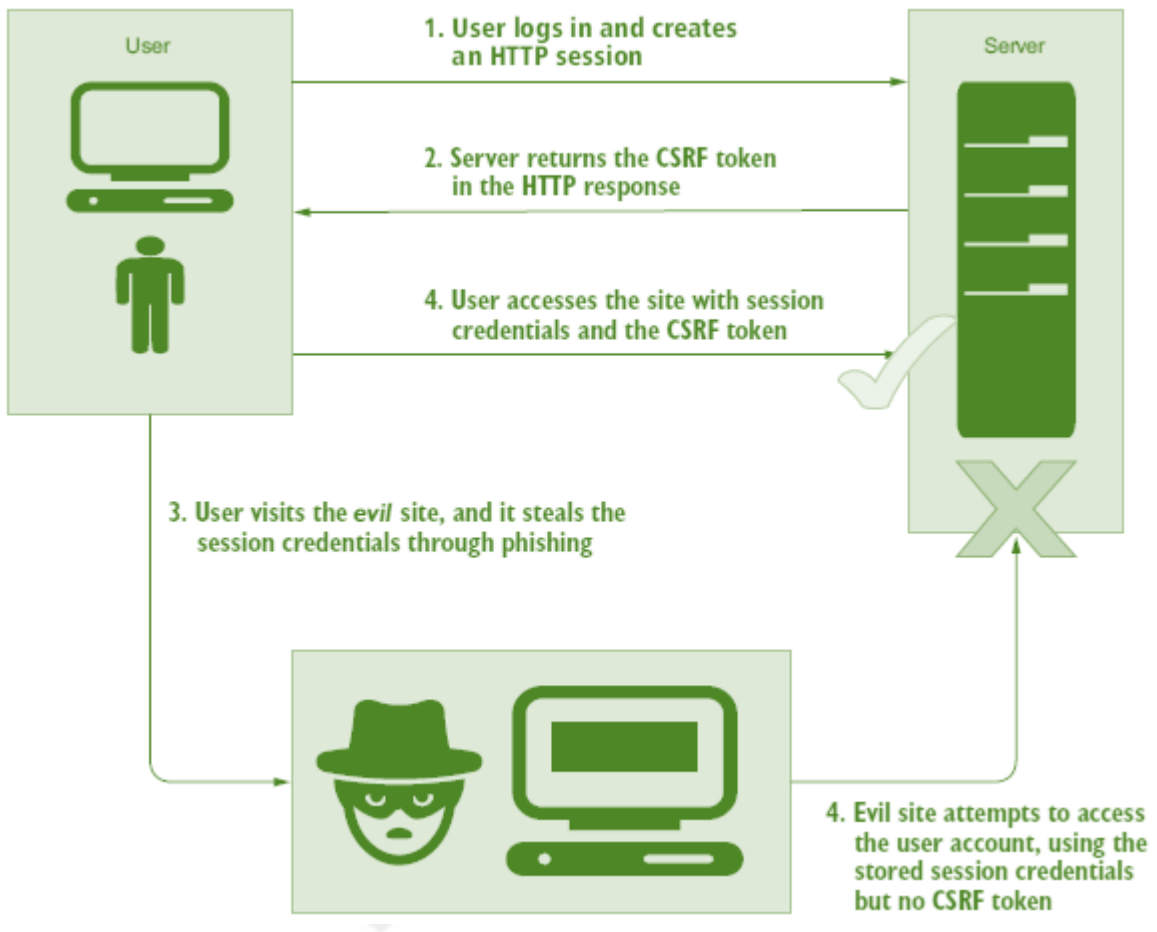


Figure 1.7 CSRF protection in a Spring Security application

9. Lecture Notes

Securing Spring Boot application.

Before we make ourselves familiar with the internal workings of Spring Security, let's provide a very high-level overview of the authentication process in a typical Web application. The sequence diagram in figure 5.5 provides the sequence of steps:

- 1 You attempt to access the home page of an application by accessing a Web URL (e.g., `http://localhost:8080/` in the course tracker application).
- 2 The request reaches the server, and it finds that you are trying to access a protected resource.
- 3 As you are not presently authenticated, the server responds indicating that you need to be authenticated. The response could be an HTTP response code or redirect to a Web page based on the security implementation at the server.
- 4 Based on the authentication mechanisms implemented in the server, the browser will either redirect you to a login page or retrieve the credentials through other modes, such as the HTTP basic authentication dialogue box or a cookie. You'll learn how to configure the authentication mechanisms in the server in later techniques.
- 5 The credentials are then sent back to the server. Browsers can either use an HTTP POST request (e.g., for a login page) or an HTTP header (e.g., for BASIC authentication) to pass the credentials to the server.
- 6 The server validates the credentials. If the credentials are valid, the login is considered successful, and the server moves to the next step. However, if the credentials are invalid, the browser typically asks to try again, so you return to step 3.
- 7 If the login is successful and logged in with sufficient authorities, then the request will be successful. Otherwise, the server returns with an HTTP error code 403, which indicates forbidden.
- 8 If the user logs out from the application, the server clears the session and other login credentials from the server and logs out the user. It then redirects the user to the login page or the index page of the application based on the security configuration of the server.

9. Lecture Notes

Securing Spring Boot application.

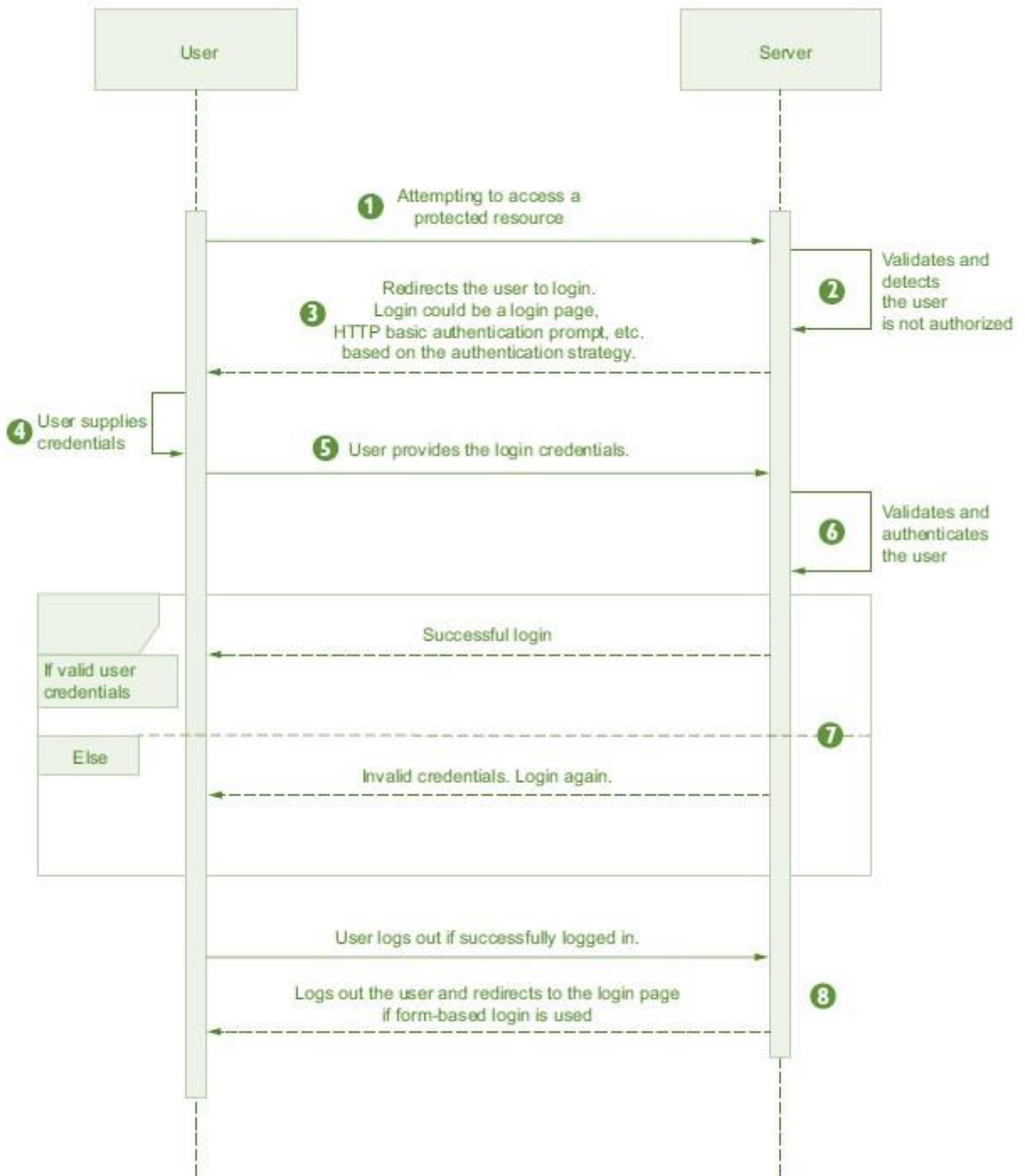


Figure 1.8 Sequence diagram of user authentication in a Web application

9. Lecture Notes

Securing Spring Boot application.

Authentication process is implementation in Spring Security.

Figure 1.9 shows this process through a block diagram. The following are the high level steps:

- 1 The initial request is handled by the authentication filters. Based on the security strategy configured in the server (you'll see how you can configure this shortly), an appropriate authentication filter handles the request. For instance, the `BasicAuthenticationFilter` processes the request if the HTTP basic authentication is configured.
- 2 The authentication filter creates an authentication token from the incoming request.
- 3 It then invokes an `AuthenticationManager` to authenticate the request.
- 4 The `AuthenticationManager` contains a list of `AuthenticationProvider` instances. An `AuthenticationProvider` has two methods: `supports(..)` and `authenticate(..)`. The `supports(..)` method decides whether the `AuthenticationProvider` supports the authentication type. The `authenticate(..)` performs the actual authentication.
- 5 The `AuthenticationProvider` uses the `UserDetailsService` implementation to perform the authentication. The `UserDetailsService` loads the User- Details from an identity store that contains user account details, such as user authorities, username, password, and other account-related statistics. The `AuthenticationProvider` uses the loaded `UserDetails` instance and performs the actual authentication. The authenticated principal is then returned to the `AuthenticationManager`, and the returned `Authentication` object is stored in the `SecurityContext` for later usage by other filters.

9. Lecture Notes

Securing Spring Boot application.

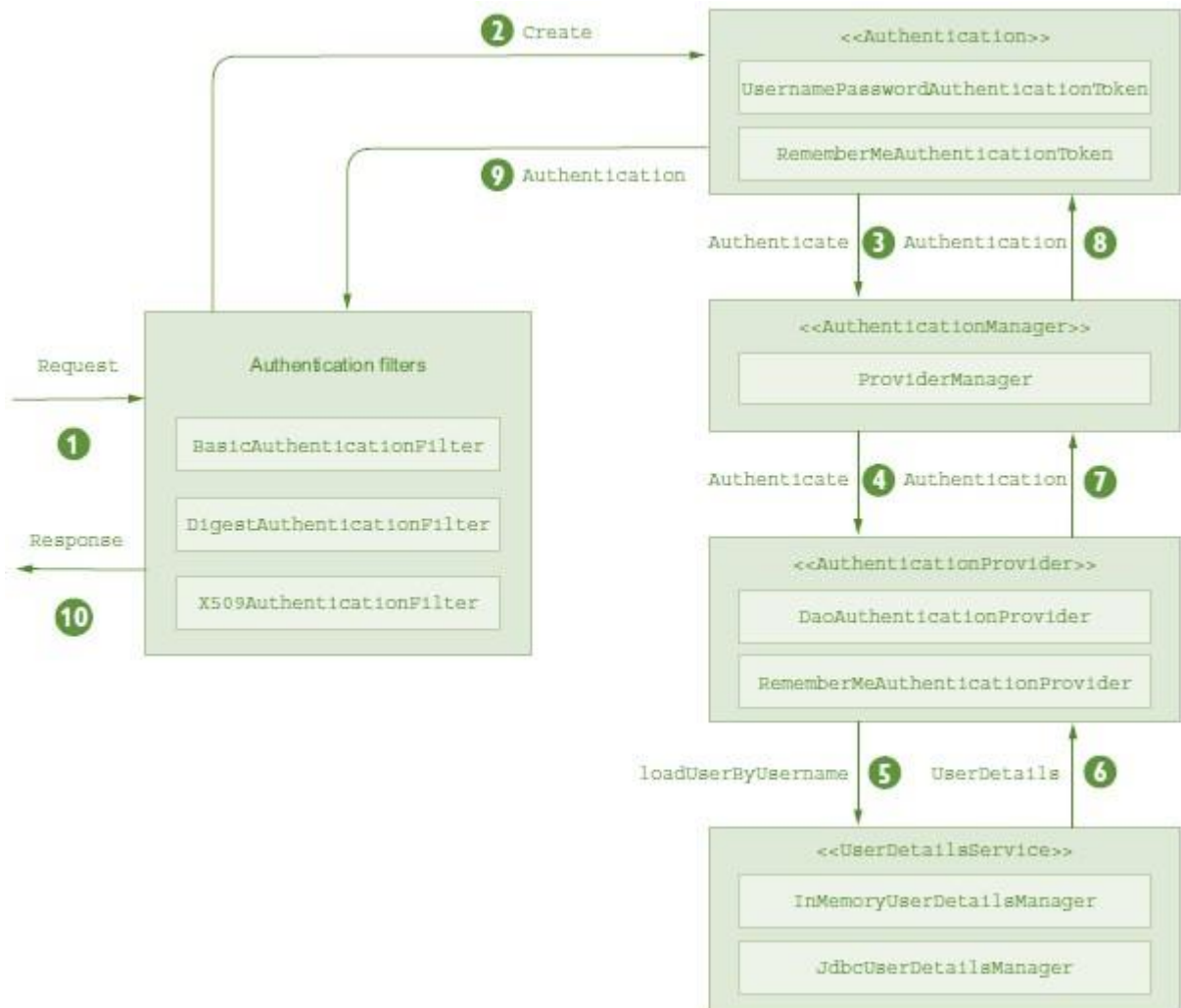


Figure 1.9 High-level overview of the Spring Security authentication steps

10. Assignment Questions

Category - 1

S.No .	Write Programs for the following	K - Level	COs
1	Use Spring Boot to build a Web Application	K2	CO1
2	Create REST Service for an Education Site	K3	CO1

Category - 2

S.No .	Write Programs for the following	K - Level	COs
1	Create an RESTFUL API service using Spring Boot	K5	CO1
2	Develop an application to demonstrate department website	K3	CO1

Category - 3

S.No .	Write Programs for the following	K - Level	COs
1	Develop a shopping cart using Spring Boot	K5	CO1
2	Develop a Image Loading application using spring boot	K5	CO1

10. Assignment Questions

Category - 4

S.No .	Write Programs for the following	K - Level	COs
1	Integrate Spring Boot and Database	K5	CO1
2	Design a calculator using Spring Boot	K5	CO1

Category - 5

S.No .	Write Programs for the following	K - Level	COs
1	Develop an spring boot application to store and retrieve	K5	CO1
2	Sort the employee names in ascending order	K5	CO1

11. Part A

Question & Answer



R.M.K.
GROUP OF
INSTITUTIONS

Part A

1. What is Spring Boot? CO1

Spring Boot is a framework that simplifies the development of Java applications by providing default configurations and conventions for building production-ready applications quickly.

2. How does Spring Boot simplify the configuration of applications? CO1

Spring Boot uses sensible defaults and provides a set of conventions, reducing the need for explicit configuration. It also offers a wide range of auto-configurations for common tasks.

3. What is the purpose of application.properties (or application.yml) in Spring Boot? CO1

It is used for managing configuration properties in a Spring Boot application. These properties can control various aspects of the application, such as server port, database connection, etc.

4. How can custom properties be defined in a Spring Boot application? CO2

Custom properties can be defined in the application.properties (or application.yml) file or as Java properties using the `@Value` annotation.

5. What is the purpose of the `@ConfigurationProperties` annotation in Spring Boot? CO2

`@ConfigurationProperties` is used to bind external properties defined in the application.properties (or application.yml) file to Java objects.

6. How can code be executed on Spring Boot application startup? CO2

By using the `@PostConstruct` annotation on a method in a Spring bean, the method will be executed after the bean is initialized.

7. What is the role of Spring Data in database access with Spring Boot? CO1

Spring Data simplifies database access by providing a consistent and high-level abstraction over different data stores, making it easier to perform CRUD operations.

8. How does Spring Boot support database initialization? CO2

Spring Boot can initialize databases using the `spring.datasource.initialization-mode` property, which allows for schema creation and data insertion during application startup.

9. What is the purpose of the `@Entity` annotation in Spring Boot? CO1

`@Entity` is used to indicate that a class is a JPA entity, representing a table in the database.

10. How can you secure a Spring Boot application? CO2

Spring Boot supports various security mechanisms, such as securing endpoints with roles, using Spring Security, and integrating OAuth 2.0 for authentication and authorization.

Part A

11. Explain the role of `@EnableAutoConfiguration` in Spring Boot. CO2
`@EnableAutoConfiguration` triggers the automatic configuration of the Spring context based on the dependencies present in the classpath.
12. What is the purpose of the `@SpringBootApplication` annotation?
It is a convenience annotation that combines `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`, simplifying the configuration of a Spring Boot application.
13. How can you enable Cross-Origin Resource Sharing (CORS) in a Spring Boot application? CO2
By using the `@CrossOrigin` annotation on the controller or by configuring CORS in the application properties using `spring.mvc.cors.*` properties.
14. Explain the use of the `@Scheduled` annotation in Spring Boot. CO2
`@Scheduled` is used to trigger methods at specified intervals, providing a way to schedule tasks in a Spring Boot application.
15. What is the purpose of the `@RestController` annotation?
`@RestController` is a specialized version of `@Controller` that is used to build RESTful web services. It combines `@Controller` and `@ResponseBody`.
16. How can you externalize configuration properties in Spring Boot? CO2
By using external configuration files (e.g., `application.properties` or `application.yml`) or environment variables.
17. What is the difference between `@Component`, `@Repository`, and `@Service` annotations in Spring Boot? CO 2
They are all specialized forms of `@Component` used to indicate different types of Spring beans. `@Repository` is used for database-related operations, and `@Service` is used for service components.
18. How does Spring Boot support logging?
Spring Boot uses the Spring Boot Starter for logging, which provides a default configuration for logging using libraries like Logback or Log4j.
19. Explain the purpose of the `@Transactional` annotation. CO2
`@Transactional` is used to demarcate a method as transactional, ensuring that the method is executed within a transaction.
20. How can you customize the error handling in a Spring Boot application? CO2
By using the `@ControllerAdvice` annotation to create a global exception handler and by defining custom error pages in the application.

Part A

21. What is the purpose of the `@Qualifier` annotation in Spring Boot? CO2
`@Qualifier` is used to specify which bean should be injected when multiple beans of the same type exist.

22. How does Spring Boot support internationalization and localization? CO2
By using the `MessageSource` interface and providing localized messages in different property files.

23. Explain the purpose of the `@Value` annotation in Spring Boot. CO2
`@Value` is used to inject values from properties files or environment variables into Spring beans.

24. How can you profile Spring Boot applications for different environments? CO2
By using profiles in the `application.properties` or `application.yml` file, or by using the `spring.profiles.active` property.

25. What is the role of the Spring Boot Actuator? CO2
The Spring Boot Actuator provides production-ready features, including monitoring, health checks, and application management, through built-in endpoints.



12.PART – B Questions

1. Explain the key features of Spring Boot that make it suitable for rapidly developing production-ready applications. Provide examples of how these features enhance development efficiency. CO2
2. Describe the steps involved in creating and managing custom configuration properties in a Spring Boot application. Provide a practical example of when custom properties might be necessary. CO1
3. Examine the various approaches for implementing database access in a Spring Boot application using Spring Data. Provide examples of different repository interfaces and their use cases. CO2
4. Discuss the mechanisms provided by Spring Boot for securing a web application. Explain the role of Spring Security and how it can be configured to handle authentication and authorization. CO1
5. Explain the role of the @ConfigurationProperties annotation in managing configuration in a Spring Boot application. Provide an example scenario where using this annotation is beneficial. CO2



13. Supportive online courses

Online courses

1. <https://www.udemy.com/topic/spring-boot/>
2. <https://www.guvi.in/mlp/join-full-stack-program>
3. <https://www.codingninjas.com/careercamp/professionals/>
4. <https://www.coursera.org/learn/spring-repositories>
5. <https://www.coursera.org/learn/google-cloud-java-spring>

External Links for Additional Resources

1. <https://spring.io/guides/gs/spring-boot/>
2. <https://www.baeldung.com/spring-boot-start>
3. <https://www.interviewbit.com/spring-boot-interview-questions/>

14. Real Time Applications

**E-commerce Platform:
Social Media Dashboard
Collaborative Document Editing**



15. Content Beyond Syllabus

SpringBootstartupevents

Spring framework's event management mechanism promotes decoupling event publishers and subscribers in an application. It allows you to subscribe to the framework's built-in events as well as define your custom events.

The Spring Boot framework also provides several built-in events that you can subscribe to perform certain actions. For instance, there might be a requirement that you need to invoke an external REST API if your Spring Boot application initializes completely. In this section, we'll introduce several Spring Boot events, which are published at various stages of an application startup and initialization:

ApplicationStartingEvent—Published at the beginning of the application startup once the Listeners are registered. Spring Boot's **LoggingSystem** uses this event to perform any action that needs to be taken up before application initialization.

ApplicationEnvironmentPreparedEvent—Published when the application is starting up and the Environment is ready for inspection and modification. Spring Boot internally uses this event to preinitialize several services, such as **MessageConverter**, **ConversionService**, **Initialize Jackson**, and others.

ApplicationContextInitializedEvent—Published when the Application Context is prepared, **ApplicationContextInitializers** are executed, but none of the bean definitions are loaded. This event can be used to perform a task before beans are initialized in the **Springcontainer**.

15. Content Beyond Syllabus

SpringBootstartupevents

ApplicationPreparedEvent—Published when the

ApplicationContext is prepared, bean definitions are loaded but not refreshed. The Environment is ready for use at this stage.

ContextRefreshedEvent—Published when the **ApplicationContext** is refreshed. This event comes from Spring—not Spring Boot.

This event does not extend **SpringApplicationEvent**. The Spring Boot **ConditionEvaluationReport** **LoggingListener** listens to this event and prints the autoconfiguration report once this event is published.

WebServerInitializedEvent—Published when the webserver is

ready. This event has two variants based on the type of the

application: **ServletWebServer InitializedEvent** for Servlet-based applications and **ReactiveWebServer InitializedEvent** for reactive applications. This event does not extend **SpringApplicationEvent**.

ApplicationStartedEvent—Published when the **ApplicationContext** is refreshed but before the **ApplicationRunner** and **CommandLineRunners** are called.

ApplicationReadyEvent—Published by **SpringApplication** to indicate the application is ready to service requests. It is not advised to change the internal state of the application, as all application initialization steps are finished.

ApplicationFailedEvent—Published when there are some exceptions, and the application has failed to start. This event is useful to perform tasks like script execution or notifying startup failures.

16. Assessment Schedule

Tentative schedule for the Assessment During 2023-2024 Even Semester

S. No.	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			Unit 1
2	IAT 1	12.02.2024	17.02.2024	Unit 1 & 2
3	Unit Test 2			Unit 3
4	IAT 2	01.04.2024	06.04.2024	Unit 3 & 4
5	Revision 1			Unit 5, 1 & 2
6	Revision 2			Unit 3 & 4
7	Model	20.04.2024	30.04.2024	All 5 Units

17. Text Books & References

TEXT BOOKS:

1. Somnath Musib, Spring Boot in Practice, Manning publication, June 2022 (<https://www.manning.com/books/spring-boot-in-practice>)
2. Alex Banks, Eve Porcello , "Learning React", May 2017, O'Reilly Media, Inc. ISBN: 9781491954621. (<https://www.oreilly.com/library/view/learning-react/9781491954614/>)
3. David Herron , "Node.js Web Development - Fourth Edition", 2018, Packt Publishing, ISBN: 9781788626859
4. Suresh Marla, "A Journey to Angular Development Paperback ", BPB Publications. (https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r)
5. Yakov Fain Anton Moiseev, "Angular Development with TypeScript", 2nd Edition. ([https://www.manning.com/books/angular-development-with-typescript-Second Edition](https://www.manning.com/books/angular-development-with-typescript-Second-Edition)).

Reference Books:

REFERENCES:

1. Sue Spielman, The Struts Framework 1: A Practical guide for Java Programmers||, 1st Edition. Elsevier 2002

WEB REFERENCES:

1. <https://www.manning.com/books/spring-boot-in-practice>
2. <https://www.oreilly.com/library/view/learning-react/9781491954614>
3. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
4. https://in.bpbonline.com/products/a-journey-to-angular-development?_pos=1&_sid=0a0a0e9fb&_ss=r
5. [https://www.manning.com/books/angular-development-with-typescript-Second Edition](https://www.manning.com/books/angular-development-with-typescript-Second-Edition)

18. Mini Project Suggestions

Category-1

1. Using Spring Boot, build a Web Application to implement Student Information System (CO1, K3)
2. Using Spring Boot, build a Web Application to implement Employee Management System (CO1, K3)

Category-2

1. Using Spring Boot, build a Web Application to implement Employee Information System (CO1, K3)
2. Using Spring Boot, build a Web Application to implement Employee Management System (CO1, K3)

Category-3

1. Using Spring Boot, build a Web Application to implement College Information System (CO1, K3)
2. Using Spring Boot, build a Web Application to implement College Management System (CO1, K3)

Category-4

1. Using Spring Boot, build a Web Application to implement Movie Ticket Reservation System (CO1, K5)
2. Using Spring Boot, build a Web Application to implement College Management System (CO1, K3)

Category-5

1. Create an application using Spring Boot and Rest API(CO1, K5)
2. Create an application using spring boot framework to design and integrate a different data sources.(CO1, K3)



Thank you

Disclaimer:

This document is confidential and intended solely for the educational purpose of RMK Group of Educational Institutions. If you have received this document through email in error, please notify the system manager. This document contains proprietary information and is intended only to the respective group / learning community as intended. If you are not the addressee you should not disseminate, distribute or copy through e-mail. Please notify the sender immediately by e-mail if you have received this document by mistake and delete this document from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.