Class Notes: https://tinyurl.com/springboot8amnotes
Contact To : 7386095600 [Whatsapp Number]
Main To : durgasoftonlinetraining@gmail.com

Spring Boot:
—------------
Spring Boot is an open source Java based framework developed by Pivotal Team.

Spring Boot makes it easy to create standalone, Production Grade Spring based applications that we can “just run”.

Spring Boot allows us to create a standalone application in order to create any type of enterprise application like Web applications, Distributed applications,....., where it is very simple to execute a standalone application.

Spring Boot is able to provide Production Ready features like metrics, Health checks , Externalized COnfigurations,.....

Spring Boot Simplifies Bootstrapping the Spring application.

Spring Boot was designed on the basis of the existing Spring Framework in order to simplify and speedup the Spring based applications Development.

Spring Boot simplifies the Spring application development by making all the configurations as “Auto-Configurations”, where developers are not required to focus more on configurations rather than concentrating on business logic.

Spring Framework requires a Spring Configuration file , but Spring Boot does not require Spring Configuration file, to provide configurations through properties file or yaml file.

When compared with Spring framework, Spring Boot reduces application execution time, it will improve application performance.

When compared with Spring framework, Spring Boot is able to improve Productivity.

Spring boot follows “Opinionated Default Configurations” approach to reduce developers efforts, that is Spring Boot avoids writing lots of boilerplate code , Annotations and XML configurations.

Spring Boot provides a simplified environment to integrate Spring Boot applications with Spring JDBC, Spring ORM, Spring MVC,.....

Spring Framework does not have the internal embedded servers, but Spring Boot has the internal embedded servers like Tomcat, Jetty, Undertow,.... To prepare and execute Spring based web applications in a simplified manner.
Spring Framework does not have the internal embedded databases , but Spring Boot has the internal embedded In Memory databases like H2,....

Spring Boot has provided internal tools like Command Line Interfaces to run and test spring boot applications in a simplified manner.

Spring Boot allows us to develop applications by using Java or by using Groovy.

Spring Boot has the internal build tools like MAVEN and GRADLE to develop the spring boot applications in a simplified manner.

```
Spring Boot: Extension to Spring + Embedded Servers + Embedded In memory
             databases + No XML Configurations + Less Annotations + Auto
             Configurations + internal MAVEN and GRADLE.
```

Spring Boot Features:
—---------------------
1. SpringApplication:
It is a class, it has a run() method to bootstrap the Spring application.

2. Externalized Configurations:
Spring boot is giving an option to the developers to externalize the application configurations in order to with the same project in multiple environments or by multiple developers.

3. Profiles:
It is able to provide different configurations for the different environments of the project , where the different configurations for the project are including Server configurations, database configurations,..... And where the different environments are including dev, QA and prod.

4. Logging:
Spring Boot uses Commons Logging internally and it is supporting the Logging implementations like JAVA provided Logging, Log4j, Slf4,.....

5. Internationalization:
Spring Boot is able to support Internationalization in order to provide Services on the basis of the local conventions.
6. JSON:

Spring Boot is able to support JSON libraries in order to represent request and response data in JSON format. Spring Boot is able to support JSON implementations like Gson, Jackson, JSON-B,.....

7. Developing Web Applications:

Spring boot is able to provide a very good environment to prepare web applications by having the embedded Servers like Tomcat, Jetty, Undertow....

8. Working With SQL databases:

Spring Boot is able to support SQL databases in order to perform database operations
EX: Oracle, MySQL,.....

9. Working With NoSQL databases:

Spring Boot is able to support No SQL databases to perform database operations.

EX:
Mongo DB
Neo4j
Cassandra
Elasticsearch
--
---

10. Cache :

Spring boot is able to support Cache mechanisms in order to store the results for future utilizations.

11. Messaging:

Spring Boot is able to support the Messaging Services in order to submit messages between services, to provide messaging in Spring boot applications it will use internally JMS and its implementations.

12. Accessing REST Services :

Spring boot is able to support accessing Restful services by using RestTemplate and WebClient,....

13. Validations:

Spring Boot is able to support Validations over the data by using bean validations provided by Hibernate.

14. Sending Mails:

Spring Boot is able to support sending mails and to receive mails by using Java Mail API.

15. Distributed Transitions:

Spring Boot is able to support both local transactions and Distributed transactions by using JTA implementations.

16. Application Events and Listeners:

Spring Boot is able to support Event Handling in order to perform sme activities when the application events are raised.

17. Admin Features:

Spring boot is able to support the admin features like checking application health, metrics, ….. Through Spring Boot Actuators.

18. Properties File and Yaml Files:

Spring boot is able to support properties files or Yaml files to provide application configuration details like Server port numbers, database configurations,......

19. Internal Build tools:

Spring supports two Build tools internally to build and execute the applications.

EX: MAVEN and GRADLE .

20. Spring Security:

Spring Boot is able to support almost all the Security libraries like Authentication and Authorization mechanisms.

Key Components in Spring Boot Applications:

Spring Boot Framework has the following four components mainly.

1. Spring Boot Starters.
2. Spring Boot AutoConfigurator
3. Spring Boot CLI
4. Spring Boot Actuators

Spring Boot Starters:

In Spring based applications , if we want to prepare a web application which contains the database interaction then we have to use the following dependencies.

```
Maven: com.google.protobuf:protobuf-java:3.21.9
Maven: com.mysql:mysql-connector-j:8.0.32
Maven: commons-logging:commons-logging:1.2
Maven: javax.servlet:javax.servlet-api:4.0.1
Maven: org.apiguardian:apiguardian-api:1.1.2
Maven: org.junit.jupiter:junit-jupiter-api:5.9.2
Maven: org.junit.jupiter:junit-jupiter-engine:5.9.2
Maven: org.junit.platform:junit-platform-commons:1.9.2
Maven: org.junit.platform:junit-platform-engine:1.9.2
Maven: org.opentest4j:opentest4j:1.2.0
Maven: org.springframework:spring-aop:4.3.14.RELEASE
Maven: org.springframework:spring-beans:4.3.14.RELEASE
Maven: org.springframework:spring-context:4.3.14.RELEASE
Maven: org.springframework:spring-core:4.3.14.RELEASE
Maven: org.springframework:spring-expression:4.3.14.RELEASE
Maven: org.springframework:spring-jdbc:4.3.14.RELEASE
Maven: org.springframework:spring-tx:4.3.14.RELEASE
Maven: org.springframework:spring-web:4.3.14.RELEASE
Maven: org.springframework:spring-webmvc:4.3.14.RELEASE
```

All the above dependencies we are able to get by providing more dependencies configuration in the pom.xml file, it will increase the number of configurations in the pom file, it is very much difficult to manage the pom file.

To overcome the above problem , Spring Boot has provided a solution in the form of Starter components.

In the above context, Spring Boot Starters are able to combine all the above dependent jars into a single jar file and all the above dependencies configurations into a single dependency in the form of a starter dependency.

“Spring-boot-starter-web”

If we add the above starter dependency in the pom.xml file the Spring Boot will download all the dependencies and spring boot will add all these dependencies to the Spring Boot project.

If we provide the above “spring-boot-starter-web” in the pom.xml file then we are able to get the following jars into the spring boot project.

1. Spring-boot-starter
    a. spring-boot
    b. spring-boot-autoconfigure
    c. spring-boot-starter-logging
2. spring-web
3. spring-webmvc
4. Spring-boot-starter-tomcat
    a. tomcat-embedded-core
    b. tomcat-embedded-login-juli

   —-------
   —-------

2. Spring Boot Autoconfigurator:
—-------------------------------
If we prepare Spring based web applications then we have to provide the following configurations in the spring configuration file.

Handler Mappings
View Resolvers
—---

After providing all the above configurations we have to use the annotations like @Configuration , @ComponentScan , @EnableAutoConfiguration,.....

In the above context, providing Spring Configuration file and its configurations and using the above annotations explicitly is a bit burdenful thing to the developers, where to reduce burden to the developers and to make spring applications as simple applications, Spring Boot has provided a feature in the form of “Spring Boot AutoConfigurator”.

In Spring Boot Applications, when we add “spring-boot-starter-web” dependency in the pom.xml file , automatically Spring Boot Autoconfigurator will come and it will resolve all the views and ViewResolvers in spring web mvc applications without having Spring configuration file and if we use @SpringBootApplication annotations at main class then it will avoid the explicit utilization of @Configuration, @ComponentScan, @EnableAutoConfiguration

@SpringBootApplication = @Configuration + @ComponentScan + @EnableAutoConfiguration

3. Spring Boot CLI:

Spring Boot CLI[Command line Interface] is a Spring Software, it will provide a very good environment to run and test Spring boot applications from command prompt.

When we run Spring Boot applications by using CLI , then internally CLI will use Spring Boot Starters and Spring boot AutoConfigurator components to resolve all the dependencies and execute Spring boot applications.

To execute Spring Boot applications through Command prompt we have to use “spring” command in command prompt.

```
Spring run Hello.groovy
```

4. Spring Boot Actuators:

Spring Boot Actuators are providing number of “Production-Grade” or “Production-Ready” features like

Spring boot Actuators are able to provide a very good environment to manage and monitor our application by using the predefined Http endpoints.[url], here these end points are able to provide the details of our application like health, auditing, metrics automatically to our application.

If we want to get Spring boot Actuators support in Spring boot applications then we have to add the following starter dependency in the pom.xml

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

Approaches to prepare Spring Boot Applications
----

1. Using Simple MAVEN project in Eclipse Or in Intellij IDEA
2. Using STS
3. Using Spring Boot Initializr
4. Spring Boot CLI[Command Line Interface]

Using Simple MAVEN project in Eclipse Or in Intellij IDEA
----

1. Download and Install MAVEN Software.
2. Open Eclipse IDE and Create MAVEN project.
3. Provide the configurations in the pom.xml file.
   a. Change Java version.
   b. Provide “Spring boot starter web” dependency in the pom.xml file.
4. Prepare Spring Boot Application class.
   a. Declare a class with main() method.
   b. Provide @SpringBootApplication annotation over the main class
   c. Inside the main() method access SpringApplication.run() method.
5. Write application logic in the main() method.
6. Execute Spring Boot application.

pom.xml

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>com.durgasoft</groupId>
      <artifactId>app01</artifactId>
      <version>0.0.1-SNAPSHOT</version>
```

```
	<packaging>jar</packaging>
	<name>app01</name>
	<url>http://maven.apache.org</url>
	<properties>
		
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<maven.compiler.source>1.17</maven.compiler.source>
		<maven.compiler.target>1.17</maven.compiler.target>
	</properties>
	<dependencies>
		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>3.8.1</version>
			<scope>test</scope>
		</dependency>
		<!--
		
https://mvnrepository.com/artifact/org.springframework.boot/spring-boo
t-starter-web -->
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
			<version>3.0.6</version>
		</dependency>
	</dependencies>
</project>
```

App.java

```
package com.durgasoft.app01;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class App {
	public static void main(String[] args) {
		SpringApplication.run(App.class, args);
		System.out.println("Welcome To Spring Boot Application In 
Elcipse");
	}
}
```

Spring Boot Application Using STS:
—--
STS: Spring tool Suite.
Objective: To simplify the preparation of Spring based Applications.
Designed on : Eclipse IDE

If we want to use STS for Spring Boot applications then we have to use the following steps.

1. Download And install STS
2. Create Spring Starter Project
3. Prepare Spring Boot Main class and provide application logic.
4. Execute Spring boot Application.

1. Download And install STS:
   a. Open https://spring.io/tools in browser
   b. Select 4.18.1-Windows 64 button.
   c. We will get the "spring-tool-suite-4-4.18.1.RELEASE-e4.27.0-win32.win32.x86_64.self-extracting.jar" file and copy this jar file to C driver.
   d. Double click on the jar file and get sts-4.18.1.RELEASE folder .
   e. Open sts-4.18.1.RELEASE folder and double click on "SpringToolSuite4" Application
   f. Select Workspace
   g. Click on launch button.

2. Create Spring Starter Project:
   a. File—>New—->Spring Starter project
   b. Provide the following details
      Name: springbootapp03
      Type: MAVEN
      Group: com.durgasoft
      —----
      —-----
      Package: com.durgasoft
   c. Click on "Next" button
   d. Select the dependency "Spring-Web" under "web".
   e. Click on the "Next" button.
   f. Click on the "Finish" button.

3. Prepare Spring Boot Main class and provide application logic:
      Open springbootapp03Application.java file and provide application logic.

4. Execute Spring boot Application.
   a. Select the “run” icon in the menu.
   b. Select “Spring Boot App”.

springbootapp03Application.java

```
package com.durgasoft;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbooptapp03Application {

      public static void main(String[] args) {
            SpringApplication.run(Springbooptapp03Application.class, args);
            System.out.println("Welcome To Spring Boot Application with
STS");
      }

}
```

Using Spring Boot Initializr

Spring Boot Initializr is a web tool, it will provide an option to generate Spring Boot projects as per our requirement.

Spring Boot Initializr is giving an option for us to download the Spring boot project from the web and we can open it in IDEs like Eclipse or Intellij IDEA,....

1. Create Spring boot Project and download from the web.
   a. Open “https://start.spring.io/”in the browser.
   b. Provide the following details.
      Project: MAVEN
      Language : JAVA
      Spring Boot: 3.1.0
      Project Metadata:
            Group: com.durgasoft
            Artifact: springbootapp03
            Package name: com.durgasoft.springbootapp03
      Packaging: jar
      Java: 17

c. Add the required Dependencies
   web
d. Click on “Generate” button

If we do all the above steps then a spring boot project will be downloaded in the form of zip as per the selections which we made in the form.

2. Import or open the Spring Boot Project in IDEs like Eclipse, Intellij IDEA:

a. Unzip the project zip file
b. Open Eclipse IDE
c. Right Click in Package Explorer and select “import”.
d. Select “Maven”.
e. Select “Existed Maven project”.
f. Click on the “Next” button.
g. Select the downloaded project by clicking on the “Browse” button.
h. Click on the “Finish” button.

If we do all the above steps , the downloaded Spring Boot project in the STS or in Eclipse.

3. Provide Application logic and execute Spring application.

We are able to use all Stereo type annotations in the Spring Boot Application.

In Stereo type annotations we are able to use @Controller to make a component as Controller, If we want to get response directly into the Browser from the Controller component then we have to use @RestController instead of @Controller.

EX:
springbootapp05Application.java
```
package com.durgasoft.springbootapp05;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp05Application {
   public static void main(String[] args) {
      SpringApplication.run(Springbootapp05Application.class, args);
```

```
    }
}
```

HelloController.java

```
package com.durgasoft.springbootapp05.controller;


import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {
    @RequestMapping("/hello")
    public String sayHello(){
        return "<h1>Hello User!</h1>";
    }
}
```

pom.xml

```
<?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>3.1.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>springbootapp05</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springbootapp05</name>
    <description>Demo project for Spring Boot</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>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>

</project>
```

Execute Spring Boot application like Standalone Application and access the application by using the following url at browser.

```
http://localhost:8080/hello
Hello User!
```

Spring Boot Applications Development Using CLI:

Spring Boot CLI[Command Line Interface] is a spring boot software to run and test Spring Boot applications from Command prompt.

If we run Spring Boot applications by using CLI , internally CLI uses Spring Boot Starter and Spring Boot Autoconfigurator components to resolve all the dependencies and to execute the spring boot application.

Note: In Spring Boot Applications, to get all the defaults and to resolve all the dependencies, Spring Boot uses Groovy and GRAPE[Jar Dependency Manager] internally.

To prepare and execute Spring Boot applications by using CLI we have to use the following steps.

1. Download and Install Spring Boot CLI.
2. Provide Groovy Script code.
3. Run Groovy Code throw Spring boot CLI.

Download and Install Spring Boot CLI:
—---------------------------------

   1. Use the following url in the web browser or search in googl for Spring boot CLI Download.
      https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started.installing.cli

   2. Select “Spring-Boot-Cli-3.1.0-bin.zip” .
   3. Copy the downloaded “Spring-Boot-Cli-3.1.0-bin.zip” in C driver and Extract it. And get the Spring-3.1.0 folder.
   4. Set “C:\spring-3.1.0\bin” to the path environment variable.
   5. Check the spring command on command prompt by using the following commands.

      1. spring – -version: To get Spring boot CLI version.
      2. Spring – -help: to get help for the Spring Boot CLI commands.
      3. spring run groovyscript.groovy

2. Prepare Groovy Script:
HelloController.groovy

```
@RestController
public class HelloController{
      @RequestMapping("/hello")
      public String sayHello(){
            return "Hello User!, Welcome To Spring Boot Groovy Script";
      }
}
```

3. Execute groovy Script:
On COmmand Prompt:

```
spring run HelloController.groovy
```

4. Access the application:
http://localhost:8080/hello

Note: To run the above application use Spring Boot CLI V 2.1.8 version.

Properties File in Spring Boot:

In Spring Boot, the main purpose of the properties file is

1. To provide application logical Names
2. To provide Server configurations like port numbers,....
3. To provide database configuration details like datasource names, driver class name, driver url, db user name, db password,....
4. To provide Security Configuration details
5. To provide Administrative configuration details
6. To provide profiles configurations

In general, properties files are able to provide the data in the form of key-value pairs.

In Spring Boot applications, in general, we will provide the properties file under resources folder.

In Spring Boot, The default name of the properties file is "application.properties".

Changing Server Port number:

If we want to change the port number to the embedded tomcat server we have to use the following property in the properties file.

```
server.port=1234
```

EX:
application.properties
```
server.port=1234
```

WelcomeController.java
```java
package com.durgasoft.springbootapp07.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WelcomeController {
   @RequestMapping("/welcome")
   public String welcomeUser(){
       return "<h1>Welcome to Durga Software Solutions!</h1>";
   }
}
```

SpringbootApp07Application.java

```
package com.durgasoft.springbootapp07;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp07Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp07Application.class, args);
   }
}
```

pom.xml

```
<?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>3.1.0</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>springbootapp07</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp07</name>
   <description>properties files configuration demo</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>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Changing Spring Boot Banner:
In Spring Boot we are able to disable the Spring Boot Banner by using the following property in the application.properties file.

```
spring.main.banner-mode=off
```

In Spring Boot applications, we can provide our own banner in place of the default banner SPRING.

To provide a custom banner in Spring boot applications we have to use the following steps.

1. Create a Spring Boot Application
2. Create banner.txt file under the resources folder.
3. Generate a Custom banner by using any online tool[search "Spring Boot banner Generator in Google].
   EX: https://devops.datenkollektiv.de/banner.txt/index.html
4. Copy the custom banner which we generated to the banner.txt file.
5. Provide the required banner properties in the application.properties file.
   EX:
   WelcomeController.java
   ```
   package com.durgasoft.springbootapp07.controller;

   import org.springframework.web.bind.annotation.RequestMapping;
   ```

```java
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WelcomeController {
   @RequestMapping("/welcome")
   public String welcomeUser(){
       return "<h1>Welcome to Durga Software Solutions!</h1>";
   }
}
```

banner.txt
```
,------.  ,--. ,--.,------.  ,----.     ,---.
|  .-.  \ |  | |  ||  .--. ''  .-./    /  O  \
|  |  \  :|  | |  ||  '--'.'|  | .---.|  .-.  |
|  '--'  /'  '-'  '|  |\  \ '  '--'  ||  | |  |
`-------'  `-----' `--' '--' `------' `--' `--'

${application.title} ${application.version}
Powered by Spring Boot ${spring-boot.version}
```

application.properties
```
server.port=1234
#spring.main.banner-mode=off

#Banner Details
application.title=Welcome Controller Application
application.version=V1.0
spring-boot.version=Spring Boot V3.10
```

Springbootapp07Application.java
```java
package com.durgasoft.springbootapp07;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp07Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp07Application.class,
args);
   }
}
```

pom.xml

```
<?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>3.1.0</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>springbootapp07</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp07</name>
   <description>properties files configuration
demo</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>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
```

```
    </build>

</project>
```

If we run the above application then we are able to get the following banner in the Console.

```
,------.  ,--. ,--.,------.  ,----.     ,---.
|  .-.  \ |  | |  ||  .--. ''  .-./    /  O  \
|  |  \  :|  | |  ||  '--'.'|  | .---.|  .-.  |
|  '--'  /'  '-'  '|  |\  \ '  '--'  ||  | |  |
`-------'  `-----' `--' '--' `------' `--' `--'

Welcome Controller Application V1.0
Powered by Spring Boot Spring Boot V3.10
```

In Spring Boot applications, we can provide our own banner in place of SPRING, but here we have to prepare banner.txt file with the new banner under resources folder, in this context banner.txt file name is not fixed, we can use any name but we must provide the new banner file name and its location to the Spring Boot framework by using the following property in the properties file.

```
spring.banner.location=classpath:fileName.txt
```

In Spring Boot applications, we can provide an image as a banner but the consoles must support the image banners, there is no guarantee whether the IDEs consoles are able to print the image banners.

Note: Image banners feature is valid up to Spring Boot 2.x version, Image banners support was removed in Spring Boot 3.x version.

If we want to provide any image as a banner in spring boot application then we have to use the following steps.

1. Keep the image file under resource folder[.jpg, .png, .gif]
2. Provide the image location in the properties file.
3. Use Spring Boot 2.x version

EX:
Springbootapp07Application.java

```java
package com.durgasoft.springbootapp07;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp07Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp07Application.class, args);
   }
}
```

pom.xml

```xml
<?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.durgasoft</groupId>
   <artifactId>springbootapp07</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp07</name>
   <description>properties files configuration demo</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>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

application.properties

```
server.port=1234
#spring.main.banner-mode=off

spring.image.banner.location=classpath:banner.gif
spring.banner.image.pixelmode=block
#Banner Details
#application.title=Welcome Controller Application
#application.version=V1.0
#spring-boot.version=Spring Boot V3.10
```

Note: Keep banner.gif file under resources.

Execute the application and get an image as banner in ASCII representation.

In Spring boot applications, we can provide our own properties in the application.properties and we are able to access our own properties data in the Spring boot application.

If we want to get the value from the properties file to a property in the bean component then we have to use @Value annotation.

@Value("${propertyName}")

EX:
WishController.java

```java
package com.durgasoft.springbootapp08.controller;

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

@RestController
public class WishController {
   @Value("${name}")
   private String userName;

   public void setUserName(String userName) {
       this.userName = userName;
   }

   @RequestMapping("/wish")
   public String sayWish(){
       return "Hello "+userName+"!, Good morning";
   }
}
```

EmployeeController.java

```java
package com.durgasoft.springbootapp08.controller;

import com.durgasoft.springbootapp08.beans.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {
   @Autowired
   private Employee employee;
   @RequestMapping("/emp")
   public String getEmployee(){
       return
"<h1>Employee[ENO:"+employee.getEno()+",ENAME:"+employee.getEname()+"
,ESAL:"+employee.getEsal()+",EADDR:"+employee.getEaddr()+"]</h1>";
   }
}
```

Employee.java

```java
package com.durgasoft.springbootapp08.beans;
```

```java
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class Employee {
   @Value("${employee.eno}")
   private int eno;
   @Value("${employee.ename}")
   private String ename;
   @Value("${employee.esal}")
   private float esal;
   @Value("${employee.eaddr}")
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }
   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

application.properties

```
name=Durga

employee.eno=111
employee.ename=Durga
employee.esal=50000.0f
employee.eaddr=Hyd
```

Springbootapp08Application.java

```java
package com.durgasoft.springbootapp08;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp08Application {
   public static void main(String[] args) {
      SpringApplication.run(Springbootapp08Application.class, args);
   }

}
```

If the properties file provided properties names and the bean component properties names are matched then we are able to copy the data from the properties file to the bean component property directly without using @Value annotation but we have to use the following annotation at bean class.
@ConfigurationProperties(prefix=”---”)

EX:
WishCotroller.java

```java
package com.durgasoft.springbootapp08.controller;

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

@RestController
```

```
public class WishController {
   @Value("${name}")
   private String userName;

   public void setUserName(String userName) {
       this.userName = userName;
   }

   @RequestMapping("/wish")
   public String sayWish(){
       return "Hello "+userName+"!, Good morning";
   }
}
```

EmployeeController.java

```
package com.durgasoft.springbootapp08.controller;

import com.durgasoft.springbootapp08.beans.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {
   @Autowired
   private Employee employee;
   @RequestMapping("/emp")
   public String getEmployee(){
       return
"<h1>Employee[ENO:"+employee.getEno()+",ENAME:"+employee.getEname()+"
,ESAL:"+employee.getEsal()+",EADDR:"+employee.getEaddr()+"]</h1>";
   }
}
```

Employee.java

```
package com.durgasoft.springbootapp08.beans;

import org.springframework.beans.factory.annotation.Value;
import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
```

```java
@ConfigurationProperties(prefix = "employee")
public class Employee {
   //@Value("${employee.eno}")
   private int eno;
   //@Value("${employee.ename}")
   private String ename;
   //@Value("${employee.esal}")
   private float esal;
   //@Value("${employee.eaddr}")
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }

   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

application.properties

```
name=Durga

employee.eno=222
employee.ename=Nag
employee.esal=60000.0f
employee.eaddr=Chennai
```

Springbootapp08Application.java

```
package com.durgasoft.springbootapp08;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp08Application {
   public static void main(String[] args) {
      SpringApplication.run(Springbootapp08Application.class, args);
   }

}
```

If we provide data in the properties file/files then the provided data will be stored in the form of an object like "Environment" , though with the Environment object we are able to access the Properties file data.

Steps:

1. Provide data in the application.properties
2. In The controller class or in the bean class declare Environment property and marked with @Autowired Annotation.
3. Get the properties data from the Environment object by using the following method.
   public String getProperty(String propName)

EX:

application.properties

```
customer.cid=C-111
customer.cname=Durga
customer.caddr=Hyd
```

CustomerController.java

```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerController {
   @Autowired
   private Environment environment;
   @RequestMapping("/cust")
   public String getCustomerDetails(){
       String data = "<h1>";
       data = data + environment.getProperty("customer.cid")+",";
       data = data + environment.getProperty("customer.cname")+",";
       data = data +
environment.getProperty("customer.caddr")+"</h1>";

       return data;
   }
}
```

Springbootapp09Application.java

```
package com.durgasoft.springbootapp09;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp09Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp09Application.class, args);
   }


}
```

Q)Is it possible to use more than one properties file in a single Spring Boot application?

Ans:

Yes, it is possible to use more than one properties file in a single spring boot application with different names, but we have to provide all these properties files by using either @PropertyResource annotation or @PropertyResources annotation.

EX:
student.properties
```
student.sid=S-111
student.sname=Anil
student.saddr=Hyd
```

employee.properties
```
employee.eid=E-111
employee.ename=Ramesh
employee.eaddr=Hyd
```

Customer.properties
```
customer.cid=C-111
customer.cname=Durga
customer.caddr=Hyd

```
StudentController.java
```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySource("classpath:student.properties")
@RestController
public class StudentController {
   @Autowired
   private Environment environment;
   @RequestMapping("/std")
   public String getStudentDetails(){
       String data = "<h1>";
```

```
        data = data + environment.getProperty("student.sid")+",";
        data = data + environment.getProperty("student.sname")+",";
        data = data +
environment.getProperty("student.saddr")+"</h1>";
        return data;
    }
}
```

EmployeeController.java

```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@PropertySource("classpath:employee.properties")
@RestController
public class EmployeeController {
   @Autowired
   private Environment environment;
   @RequestMapping("/emp")
   public String getEmployeeDetails(){
       String data = "<html>";
        data = data + environment.getProperty("employee.eid")+",";
        data = data + environment.getProperty("employee.ename")+",";
        data = data +
environment.getProperty("employee.eaddr")+"</h1>";
        return data;
    }
}
```

CustomerController.java

```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@PropertySource("classpath:customer.properties")
@RestController
public class CustomerController {
   @Autowired
   private Environment environment;
   @RequestMapping("/cust")
   public String getCustomerDetails(){
       String data = "<h1>";
       data = data + environment.getProperty("customer.cid")+",";
       data = data + environment.getProperty("customer.cname")+",";
       data = data +
environment.getProperty("customer.caddr")+"</h1>";

       return data;
   }
}
```

MyController.java

```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

//@PropertySource({"classpath:customer.properties",
"classpath:student.properties", "classpath:employee.properties"})
@PropertySources({
       @PropertySource("classpath:customer.properties"),
       @PropertySource("classpath:student.properties"),
       @PropertySource("classpath:employee.properties")
       }
)
@RestController
public class MyController {
   @Autowired
   private Environment environment;
   @RequestMapping("/all")
   public String getAllDetails(){
       String customerData = "<h1>Customer:[";
       customerData = customerData +
environment.getProperty("customer.cid")+",";
       customerData = customerData +
environment.getProperty("customer.cname")+",";
```

```
       customerData = customerData + 
environment.getProperty("customer.caddr")+"]</h1>";

       String employeeData = "<h1>Employee:[";
       employeeData = employeeData + 
environment.getProperty("employee.eid")+",";
       employeeData = employeeData + 
environment.getProperty("employee.ename")+",";
       employeeData = employeeData + 
environment.getProperty("employee.eaddr")+"]</h1>";

       String studentData = "<h1>Student:[";
       studentData = studentData + 
environment.getProperty("student.sid")+",";
       studentData = studentData + 
environment.getProperty("student.sname")+",";
       studentData = studentData + 
environment.getProperty("student.saddr")+"]</h1>";
       return customerData + studentData + employeeData;
    }
}
```

MyController.java

```
package com.durgasoft.springbootapp09.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

//@PropertySource({"classpath:customer.properties", 
"classpath:student.properties", "classpath:employee.properties"})
@PropertySources({
       @PropertySource("classpath:customer.properties"),
       @PropertySource("classpath:student.properties"),
       @PropertySource("classpath:employee.properties")
       }
)
@RestController
public class MyController {
   @Autowired
   private Environment environment;
   @RequestMapping("/all")
   public String getAllDetails(){
       String customerData = "<h1>Customer:[";
       customerData = customerData +
environment.getProperty("customer.cid")+",";
       customerData = customerData +
environment.getProperty("customer.cname")+",";
       customerData = customerData +
environment.getProperty("customer.caddr")+"]</h1>";

       String employeeData = "<h1>Employee:[";
       employeeData = employeeData +
environment.getProperty("employee.eid")+",";
       employeeData = employeeData +
environment.getProperty("employee.ename")+",";
       employeeData = employeeData +
environment.getProperty("employee.eaddr")+"]</h1>";

       String studentData = "<h1>Student:[";
       studentData = studentData +
environment.getProperty("student.sid")+",";
       studentData = studentData +
environment.getProperty("student.sname")+",";
       studentData = studentData +
environment.getProperty("student.saddr")+"]</h1>";
       return customerData + studentData + employeeData;
   }
}
```

In the properties files we are able to provide place holders and we are able to provide values to the place holders.

EX:
application.properties

```
name=Durga
address=Hyderabad

employee.eno=111
employee.ename=${name}
employee.eaddr=${address}

student.sid=S-111
student.sname=${name}
student.saddr=${address}
```

EmployeeController.java

```java
package com.durgasoft.springbootapp10.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {

   @Autowired
   private Environment environment;
   @RequestMapping("/emp")
   public String getEmployeeDetails(){
       String data = "<h1>Employee[";
       data = data + environment.getProperty("employee.eno")+",";
       data = data + environment.getProperty("employee.ename")+",";
       data = data +
environment.getProperty("employee.eaddr")+"]</h1>";
       return data;
   }
}
```

StudentController.java

```java
package com.durgasoft.springbootapp10.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StudentController {
   @Autowired
   private Environment environment;
   @RequestMapping("/std")
   public String getStudentDetails(){
       String data = "<h1>Student[";
```

```
        data = data + environment.getProperty("student.sid")+",";
        data = data + environment.getProperty("student.sname")+",";
        data = data +
environment.getProperty("student.saddr")+"]</h1>";
        return data;
    }
}
```

Springapp10Application.java

```
package com.durgasoft.springbootapp10;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp10Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp10Application.class, args);
   }

}
```

In Spring boot applications, we are able to provide the properties files in the following locations.

1. Directly Under resources folder.
2. Under resources / config folder.
3. Directly under the project folder
4. Under the project / config folder

In the Spring Boot project, resources and its internal folders exist in the classpath directly, but the project and its internal folders except resources are not in the classpath.

If we want to get the properties files which exist in the resources and its internal folders to the controller class we have to use the @PropertySource or @propertySources annotations like below.

```
 @PropertySource({“classpath:abc.properties”,
“classpath:./config/xyz.properties”})
```

                 Or

```
@PropertySources(
      @PropertySource(“classpath:abc.properties”),
      @PropertySource(“classpath:./config/xyz.properties”)
)
```

If we want to get the properties files which are not in the classpath, which are available in the Folder structure to the Controller class then we have to use @PropertySource and @PropertySources annotations like below.

```
 @PropertySource({“file:abc.properties”, “file:./config/xyz.properties”})
```

                 Or

```
@PropertySources(
      @PropertySource(“file:abc.properties”),
      @PropertySource(“file:./config/xyz.properties”)
)
```

Note: In the above context, we can provide place holders in one properties file and we can define values to the place holders in other properties. file.

EX:
springbootapp11\src\main\\resources\employee.properties

```
name=Durga
employee.empNo=111
employee.empName=${name}
employee.empAddress=Hyd
```

springbootapp11\src\main\resources\config\student.properties

```
student.stdNo=333
student.stdName=${name}
student.stdAddress=Delhi
```

springbootapp11\customer.properties:

```
customer.custId=222
customer.custName=${name}
customer.custAddress=Pune
```

springbootapp11\config\client.properties

```
client.clientNo=555
client.clientName=${name}
client.clientAddress=Chennai
```

EmployeeController.java

```java
package com.durgasoft.springbootapp11.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySource("classpath:employee.properties")
@RestController
public class EmployeeController {
   @Autowired
   private Environment environment;
   @RequestMapping("/emp")
   public String getEmpDetails(){
       String data = "<h1>Employee[";
       data = data + environment.getProperty("employee.empNo")+",";
       data = data + environment.getProperty("employee.empName")+",";
       data = data +
environment.getProperty("employee.empAddress")+"]</h1>";
       return data;
   }
}
```

StudentController.java

```java
package com.durgasoft.springbootapp11.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySource("classpath:config/student.properties")
@RestController
public class StudentController {
   @Autowired
   private Environment environment;
   @RequestMapping("/std")
   public String getStdDetails(){
       String data = "<h1>Student[";
       data = data + environment.getProperty("student.stdNo")+",";
       data = data + environment.getProperty("student.stdName")+",";
       data = data +
environment.getProperty("student.stdAddress")+"]</h1>";
       return data;
   }
}
```

CustomerController.java

```
package com.durgasoft.springbootapp11.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySource("file:./customer.properties")
@RestController
public class CustomerController {
   @Autowired
   private Environment environment;
   @RequestMapping("/cust")
   public String getCustDetails(){
       String data = "<h1>Customer[";
       data = data + environment.getProperty("customer.custId")+",";
       data = data +
environment.getProperty("customer.custName")+",";
       data = data +
environment.getProperty("customer.custAddress")+"]</h1>";
       return data;
   }

}
```

ClientController.java

```
package com.durgasoft.springbootapp11.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
```

```java
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySource("file:./config/client.properties")
@RestController
public class ClientController {
   @Autowired
   private Environment environment;
   @RequestMapping("/client")
   public String getStdDetails(){
       String data = "<h1>Client[";
       data = data + environment.getProperty("client.clientNo")+",";
       data = data +
environment.getProperty("client.clientName")+",";
       data = data +
environment.getProperty("client.clientAddress")+"]</h1>";
       return data;
   }
}
```

SpringbootApp11Application.java

```java
package com.durgasoft.springbootapp11;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp11Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp11Application.class, args);
   }

}
```

IN Spring boot applications, we are able to provide the same properties file with different data at different locations and we are able to get the data from all the properties files.
EX:
springbootapp12/src/main/resources/user.properties

```
user.name=Durga
```

springbootapp12/src/main/resources/config/user.properties

```
user.qual=MTech
```

springbootapp12/user.properties

```
user.email=durga@durgasoft.com
```

springbootapp12/config/user.properties

```
user.mobile=9988776655
```

UserController.java

```
package com.durgasoft.springbootapp12.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySources({
       @PropertySource("classpath:user.properties"),
       @PropertySource("classpath:/config/user.properties"),
       @PropertySource("file:./user.properties"),
       @PropertySource("file:./config/user.properties")
   }
)
@RestController
public class UserController {
   @Autowired
   private Environment environment;
   @RequestMapping("/user")
   public String getUserDetails(){
       String data = "<h1>User[";
       data = data + environment.getProperty("user.name")+",";
       data = data + environment.getProperty("user.qual")+",";
       data = data + environment.getProperty("user.email")+",";
       data = data + environment.getProperty("user.mobile")+"]</h>";
       return data;
   }
}
```

Springbootapp12Application.java

```
package com.durgasoft.springbootapp12;
```

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp12Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp12Application.class, args);
   }

}
```

Q)In the Spring boot application, if we provide the same properties file at multiple locations with the same data [Key is same but value is different] then which properties data will come to the Environment object?

Ans:
--
In the Spring Boot application, if we provide the same properties files at multiple locations with the same data then the Environment object is able to get all the properties files data as per the order in which we have provided the properties files configuration with @PropertySource annotations in @PropertySources annotation.

EX:
resources/user.properties

```
message=Message from resources/user.properties
```

resources/config/user.properties

```
message=Message from resources/config/user.properties
```

springbootapp12/user.properties

```
message=Message from springbootapp12/user.properties
```

springbootapp12/config/user.properties

```
message=Message from springbootapp12/config/user.properties
```

UserController.java

```java
package com.durgasoft.springbootapp12.controller;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@PropertySources({

       @PropertySource("file:./config/user.properties"),
       @PropertySource("classpath:/config/user.properties"),
       @PropertySource("file:./user.properties"),
       @PropertySource("classpath:user.properties")

   }
)
@RestController
public class UserController {
   @Autowired
   private Environment environment;
   @RequestMapping("/user")
   public String getUserDetails(){
       String data =
"<h1>"+environment.getProperty("message")+"</h1>";

       return data;
   }
}
```

Springbootapp12Application.java

```
package com.durgasoft.springbootapp12;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp12Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp12Application.class, args);
   }
}
```

OP: **Message from resources/user.properties**

Yaml Files:

Yaml: Yet Another markup Language [Old]
Yaml: Yaml Ain’t Markup Language [New]

Yaml file is a text represented file to represent data and to transport data over the network , it is the same as XML , JSON,.... Files.

In General, Yaml files are created by using the extension “.yml”.

Inside the yaml file we are able to provide data in the form of key-value pairs with the indentation operator or levels.

In Spring Boot Applications we are able to use the yaml file as an alternative to the properties file.

In Spring boot applications, the default name of the yaml file is “application.yml” under the resources folder.

EX: application.yml

```
Server:
      port: 8080

application:
      name: Login Application
```

In the properties file:

```
server.port=8080
application.name=Login Application
```

```
EX:
durgasoft:
      courses:
            name: JAVA
```

In properties file:
durgasoft.courses.name=JAVA

Note: In Spring Boot all the conventions of the properties files are applicable to the yaml files.

EX:
application.yml

```yaml
server:
port: 1234

employee:
eno: 111
ename: Durga
esal: 50000
eaddr: Hyd

student:
sid: S-111
sname: Durga
semail: durga@durgasoft.com
smobile: 91-9988776655

customer:
cid: C-111
cname: Durga
cemail: durga@dss.com
caddr: Hyd
```

Student.java

```java
package com.durgasoft.springbootapp13.beans;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

```

```java
@Component
public class Student {
@Value("${student.sid}")
private String sid;
@Value("${student.sname}")
private String sname;
@Value("${student.semail}")
private String semail;
@Value("${student.smobile}")
private String smobile;

public String getSid() {
return sid;
}

public void setSid(String sid) {
this.sid = sid;
}

public String getSname() {
return sname;
}

public void setSname(String sname) {
this.sname = sname;
}

public String getSemail() {
return semail;
}

public void setSemail(String semail) {
this.semail = semail;
}

public String getSmobile() {
return smobile;
}

public void setSmobile(String smobile) {
this.smobile = smobile;
}
}
```

Customer.java

```
package com.durgasoft.springbootapp13.beans;

import
org.springframework.boot.context.properties.ConfigurationPropert
ies;
import org.springframework.stereotype.Component;

@ConfigurationProperties(prefix = "customer")
@Component
public class Customer {
private String cid;
private String cname;
private String cemail;

private String caddr;

public void setCemail(String cemail) {
this.cemail = cemail;
}

public String getCemail() {
return cemail;
}



public String getCid() {
return cid;
}

public void setCid(String cid) {
this.cid = cid;
}

public String getCname() {
return cname;
}
public void setCname(String cname) {
this.cname = cname;
}

public String getCaddr() {
return caddr;
}

public void setCaddr(String caddr) {
this.caddr = caddr;
}
}
```

EmployeeController.java

```
package com.durgasoft.springbootapp13.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeController {
@Autowired
private Environment environment;
@RequestMapping("/emp")
public String getEmployeeDetails(){
String data = "<h1>Employee[";
data = data + environment.getProperty("employee.eno")+",";
data = data + environment.getProperty("employee.ename")+",";
data = data + environment.getProperty("employee.esal")+",";
data = data +
environment.getProperty("employee.eaddr")+"]</h1>";
return data;
}
}
```

StudentController.java

```
package com.durgasoft.springbootapp13.controller;
```

```
import com.durgasoft.springbootapp13.beans.Student;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StudentController {
@Autowired
private Student student;
@RequestMapping("/std")
public String getStudentDetails(){
String data = "<h1>Student[";
data = data + student.getSid()+",";
data = data + student.getSname()+",";
data = data + student.getSemail()+",";
data = data + student.getSmobile()+"]</h1>";
return data;
}
}
```

CustomerController.java

```
package com.durgasoft.springbootapp13.controller;

import com.durgasoft.springbootapp13.beans.Customer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CustomerController {
@Autowired
private Customer customer;
@RequestMapping("/cust")
public String getCustomerDetails(){
String data = "<h1>Customer[";
data = data + customer.getCid()+",";
data = data + customer.getCname()+",";
data = data + customer.getCemail()+",";
data = data + customer.getCaddr()+"]</h1>";
return data;
}
}
```

Springbootapp13Application.java

```java
package com.durgasoft.springbootapp13;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp13Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp13Application.class, args);
System.out.println("YML file Application");
}
}
```

pom.xml

```xml
<?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>3.1.0</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.durgasoft</groupId>
<artifactId>springbootapp13</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootapp13</name>
<description>springbootapp13</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>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

In Spring Boot applications, we are able to provide multiple yml files as per the requirement at the following locations.

1. resources/a.yml
2. resources/config/a.yml
3. application/a.yml
4. application/config/a.yml

In the above context, Spring Boot will get the message from all the yml files in the following order.

1. resources/a.yml
2. resources/config/a.yml
3. application/a.yml
4. application/config/a.yml

EX:
resources/application.yml
message: AAA

resources/config/application.yml
message: BBB

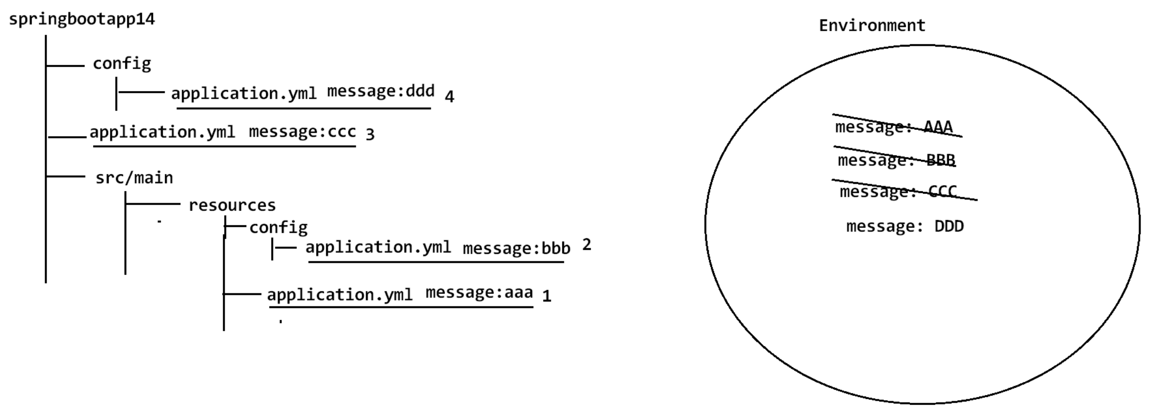
application/application.yml
message: CCC

application/config/application.yml
message: DDD

AppController.java
```
package com.durgasoft.springbootapp14;

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

@RestController
public class AppController {
@Value("${message}")
public String message;

public void setMessage(String message) {
this.message = message;
}

@RequestMapping("/msg")
public String getMessage() {
return message;
}


}
```

Springbootapp14Application.java
```
package com.durgasoft.springbootapp14;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp14Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp14Application.class, args);
}


}
```

Output:
DDD


springbootapp14
config
application.yml message:ddd 4
application.yml message:ccc 3
src/main
resources
config
application.yml message:bbb 2
application.yml message:aaa 1
Environment
~~message: AAA~~
~~message: BBB~~
~~message: CCC~~
message: DDD

If we want to change the search order then we must not use the yml file name application.yml, we have to use different names.

EX:
resources/a.yml
message: AAA

resources/config/b.yml
message: BBB

application/c.yml
message: CCC

application/config/d.yml
message: DDD

AppController.java

```
package com.durgasoft.springbootapp14;

import org.springframework.beans.factory.annotation.Value;
```

```java
import org.springframework.context.annotation.PropertySource;
import org.springframework.context.annotation.PropertySources;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@PropertySources({
@PropertySource("classpath:a.yml"),
@PropertySource("file:./c.yml"),
@PropertySource("file:./config/d.yml"),
@PropertySource("classpath:config/b.yml")

})
public class AppController {
@Value("${message}")
public String message;

public void setMessage(String message) {
this.message = message;
}

@RequestMapping("/msg")
public String getMessage() {
return message;
}

}
```

Springbootapp14Application.java

```java
package com.durgasoft.springbootapp14;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp14Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp14Application.class, args);
}


}
```

```
OP:
BBB
```

In Spring Boot applications, in the properties file we can provide more than one value for a single property and we will get it into the Spring Boot application.

In the properties file , if we provide ‘,’ separated values to a single property then Spring Boot will convert all these values into the String[] or a List ,... as per the variable declaration in the Bean component.

EX:
application.properties

```
user.qual=BTech,MTech,PHD
```

User.java

```
@Component
public class User{
      @Value(“${user.qual}”)
      private String[] qual;

}
```

EX:
application.properties

```
server.port=1234
user.uname=Durga
user.uaddress=Chennai
user.uquals=BSC,MTech,PHD
user.utechs=JAVA,PYTHON,DEVOPS

```

User.java

```
package com.durgasoft.springbootapp15.beans;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.List;
```

```
@Component
public class User {
@Value("${user.uname}")
private String name;
@Value("${user.uaddress}")
private String address;
@Value("${user.uquals}")
private String[] quals;
@Value("${user.utechs}")
private List<String> techs;

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getAddress() {
return address;
}

public void setAddress(String address) {
this.address = address;
}

public String[] getQuals() {
return quals;
}

public void setQuals(String[] quals) {
this.quals = quals;
}

public void setTechs(List<String> techs) {
this.techs = techs;
}

public List<String> getTechs() {
return techs;
}
}
```

USerController.java

```
package com.durgasoft.springbootapp15.controller;

import com.durgasoft.springbootapp15.beans.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
@Autowired
private User user;
@RequestMapping("/user")
public String getUserDetails(){
String data = "<h1>User[";
data = data + "Name : "+user.getName()+",";
data = data + "Address : "+user.getAddress()+",";
data = data + "Qualifications:";
for(String qual: user.getQuals()){
data = data + qual+" ";
}
data = data+",Technologies: "+user.getTechs();
data = data +"]</h1>";
return data;
}
}
```

springbootapp15Application.java

```
package com.durgasoft.springbootapp15;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp15Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp15Application.class, args);
}


}
```

In Spring Boot applications, if we want to provide multiple values to a single property then we have to use the following notation in the yml file.

```
application.yml
user:
  qual:
   -  BTECH
   -  MTECH
   -  PHD
```

The above values will be converted to the Sting[] or a List as pert the property declaration.

EX:
application.yml

```
server:
     port: 1234
User:
     name: Anil
     address: Delhi
     quals:
          - BTech
          - MTech
          - PHD
     techs:
          - JAVA
          - PYTHON
          - DEVOPS
```

User.java

```
package com.durgasoft.springbootapp15.beans;

import org.springframework.beans.factory.annotation.Value;
import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import java.util.List;
```

```java
@ConfigurationProperties(prefix = "user")
@Component
public class User {
//@Value("${user.name}")
private String name;
//@Value("${user.address}")
private String address;
//@Value("${quals}")
private String[] quals;
//@Value("${techs}")
private List<String> techs;

public String getName() {
return name;
}

public void setName(String name) {
this.name = name;
}

public String getAddress() {
return address;
}

public void setAddress(String address) {
this.address = address;
}

public String[] getQuals() {
return quals;
}

public void setQuals(String[] quals) {
this.quals = quals;
}

public void setTechs(List<String> techs) {
this.techs = techs;
}

public List<String> getTechs() {
return techs;
}
}
```

USerController.java

```
package com.durgasoft.springbootapp15.controller;

import com.durgasoft.springbootapp15.beans.User;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
@Autowired
private User user;
@RequestMapping("/user")
public String getUserDetails(){
String data = "<h1>User[";
data = data + "Name : "+user.getName()+",";
data = data + "Address : "+user.getAddress()+",";
data = data + "Qualifications:";
for(String qual: user.getQuals()){
data = data + qual+" ";
}

data = data+",Technologies: "+user.getTechs();
data = data +"]</h1>";
System.out.println(user.getQuals().length);
System.out.println(user.getTechs().size());
return data;
}
}
```

Springbootapp15Application.java

```
package com.durgasoft.springbootapp15;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp15Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp15Application.class, args);
}


}
```
Q)In Spring Boot application, if we provide the same property in both application.properties and application.yml files then which value would be taken by the Spring Boot Application?
——

Ans:

If we provide properties in both application.properties and application.yml files then Spring Boot will search for the properties first in the application.properties, if the property does not exist in the application.properties file then Spring Boot will search for the property in application.yml file.

EX:
--
application.properties
```
message=Hello User, This is from application.properties
```

application.yml
```
message: Hello User, This is from application.yml
```

USerController.java
```
package com.durgasoft.springbootapp15.controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {
@Value("${message}")
private String message;
@Autowired
private User user;
@RequestMapping("/test")
public String getUserDetails(){
return message;
}
}
```

springbootapp15APplication.java

```
package com.durgasoft.springbootapp15;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp15Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp15Application.class, args);
}

}
```

Spring Boot Runners:

The main purpose of Spring Boot Runner is to execute a block of code once immediately after loading Spring Boot Framework automatically.

In Spring Boot applications, Spring Boot Runners are executed only one time.

In general, in Spring Boot applications, if we want to test our Code instantly we will use Spring Boot Runners.

If we want to override the configuration data which we have provided in the properties file or if we want to override the default configuration details in Spring Boot Applications then we have to use Runners and Command Line Arguments.

Spring Boot has provided the following two types of Runners.

1. CommandLineRunner
2. ApplicationRunner

CommandLineRunner:

CommandLineRunner is a Runner class , it was provided in SpringBoot 1.x version and it is able to take any type of command line arguments including Option Arguments and Non Option Arguments.

Where Option ARguments are single values like - -server.port=1234
Where Non Option Arguments are Key-Value pairs start, stop, edit,..
CommandLineRunner is a Functional interface, it has only one method .

```
      public void run(String[] args)
```

If we want to Use CommandLineRunner in Spring boot applications then we have to use the following steps.

1. Prepare a user defined class.
2. Provide @SpringBootApplication or @Component annotation over the User defined class.
3. Implement CommandLineRunner interface in user defined class.
4. Implement run() method in user defined class.

EX:

```
@SpringBootApplication
public class EditorRunner implements COmmandLineRunner{
      public void run(String[] args){
            ------
      }
}
```

EX:
EditorRunner.java

```
package com.durgasoft.springbootapp16.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EditorRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("Hello User, this is from Editor Runner from
CommandLineRunner....");
}
}
```

Springbootapp16Application.java

```
package com.durgasoft.springbootapp16;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp16Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp16Application.class, args);
System.out.println("Hello User, This is from Spring Boot Entry
class....");
}

}
```

```
Hello User, this is from Editor Runner from CommandLineRunner....
Hello User, This is from Spring Boot Entry class....
```

Internal Flow:
When we execute the Spring boot Project,

1. Spring Boot Project will be loaded
2. Spring Boot Framework will execute the main() method of the SpringBoot Entry class.
3. In the main() method Spring Boot Framework will execute SpringApplication.run() method.
4. SpringApplication.run() method will recognize all the runner classes and it will execute all the Runner classes.

In CommandLineRunner we are able to provide both Option Arguments and Non Option arguments as the command line arguments.

EditorRunner.java

```
package com.durgasoft.springbootapp16.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

```

```java
@SpringBootApplication
public class EditorRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("Hello User, this is from Editor Runner from
CommandLineRunner....");
for(String str: args){
System.out.println(str);
}

}
}
```

Springbootapp16Application.java

```java
package com.durgasoft.springbootapp16;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp16Application {

public static void main(String[] args) {

SpringApplication.run(Springbootapp16Application.class, args);
System.out.println("Hello Suer, this is from Spring Boot Entry
class....");
}

}
```

Execute the Spring Boot application by providing Command line arguments.

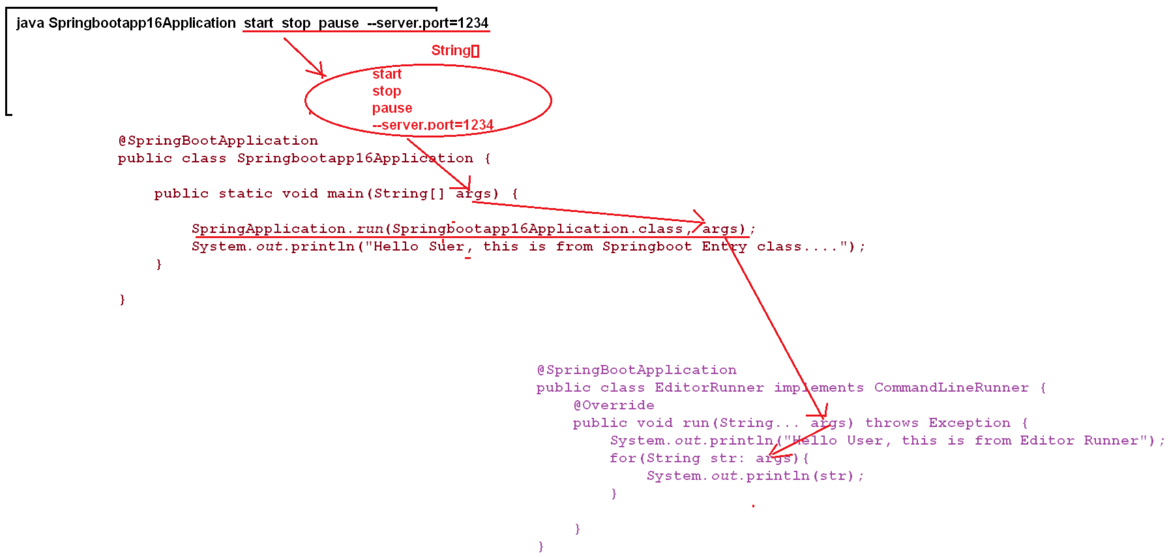
1. Open “Edit the run configuration” by clicking on three dots at run option.
2. Select More Actions.
3. Select Program Arguments
4. Provide Command Line Arguments with the space separator.
5. Click on run button

```
Hello User, this is from Editor Runner from CommandLineRunner....
```

```
start
stop
pause
--server.port=1234
Hello Suer, this is from Spring Boot Entry class....
```

If we pass the command line input to the Spring Boot application then the following actions will take place.

1. JVM will read all the command line Arguments from command prompt, JVM will create a String[] with all the command line arguments.
2. JVM will access Spring Boot main class provided main() method bypassing the Command line input String[] as parameter.
3. In the main() method, main() method will access SpringApplication.run() method by passing the same Command Line Arguments String[].
4. Where SpringApplication.run() method will recognize all the Runner classes and access runner classes run() method bypassing the command line arguments String[].



```
java Springbootapp16Application  start  stop  pause  --server.port=1234
```

```
@SpringBootApplication
public class Springbootapp16Application {

    public static void main(String[] args) {

        SpringApplication.run(Springbootapp16Application.class, args);
        System.out.println("Hello Suer, this is from Springboot Entry class....");
    }

}
```

```
@SpringBootApplication
public class EditorRunner implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("Hello User, this is from Editor Runner");
        for(String str: args){
            System.out.println(str);
        }

    }
}
```

In Spring Boot Applications we are able to provide more than one Runner class and they will be executed in alphabetic order as part of loading Spring Boot application.

EX:

StudentCRUDRunner.java

```java
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class StudentCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from StudentCRUDRunner");
}
}
```

EmployeeCRUDRunner.java

```java
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class EmployeeCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from EMployeeCRUDRunner");
}
}
```

AccountCRUDRunner.java

```java
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AccountCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from AccountCRUDRunner");
}
}
```

Springbootapp17Application.java

```java
package com.durgasoft.springbootapp17;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp17Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp17Application.class, args);
System.out.println("This is from Spring Boot Main Class");
}


}
```

```
This is from AccountCRUDRunner
This is from EMployeeCRUDRunner
This is from StudentCRUDRunner
This is from Spring Boot Main Class
```

If we want to execute the above runners as per our own order then have to declare all the Runner classes with @Order annotation.

@Order : Still it executes in Alphabetic Order.
@Order(pripritYValue) : It will execute Runner as priority.

Where the priorityValue is less then the runner class will get more priority and it will be executed earlier.

Where the priorityValue is more then the Runner class will get less priority and it will be executed later.

EX:
StudentCRUDRunner.java

```java
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.core.annotation.Order;

```

```
@Order(100)
@SpringBootApplication
public class StudentCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from StudentCRUDRunner");
}
}
```

EmployeeCRUDRunner.java

```
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.core.annotation.Order;

@Order(200)
@SpringBootApplication
public class EmployeeCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from EMployeeCRUDRunner");
}
}
```

AccountCRUDRunner.java

```
package com.durgasoft.springbootapp17.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.core.annotation.Order;

@Order(300)
@SpringBootApplication
public class AccountCRUDRunner implements CommandLineRunner {
@Override
public void run(String... args) throws Exception {
System.out.println("This is from AccountCRUDRunner");
}
}
```

Springbootapp17Application.java

```
package com.durgasoft.springbootapp17;
```

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

@SpringBootApplication
public class Springbootapp17Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp17Application.class, args);
System.out.println("This is from Spring Boot Main Class");
}

}
```

```
This is from StudentCRUDRunner
This is from EMployeeCRUDRunner
This is from AccountCRUDRunner
This is from Spring Boot Main Class
```

ApplicationRunner:

ApplicationRunner is a Spring Boot Runner, it is able to take both Option ARguments and Non Option Arguments, but they must be maintained separately.

ApplicationRunner is a functional interface, it has only method that is

public void run(ApplicationArguments ar)

Where ApplicationArguments is able to manage all the command line arguments separately like Option Arguments separately and Non Option Arguments Separately.

If we want to get all the names of the Option ARguments:
public Set getOptionNames()

If we want to get all the Non Option Arguments then we have to use the following method.
public List getNonOptionArguments()

If we want to get the value of an Option Argument on the basis of the name then we have to use the following method.
public List getOptionValues(String name)

If we want to check whether the argument name is available or not in Option Argument then we have to use the following method.

public boolean containsOption(String name)

CRUDRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CRUDRunner implements ApplicationRunner {
@Override
public void run(ApplicationArguments args) throws Exception {
System.out.println("This is from CRUDRunner");
System.out.println(args.getOptionNames());
System.out.println(args.getNonOptionArgs());
System.out.println(args.getOptionValues("application.name"));
System.out.println(args.containsOption("server.port"));
}
}
```

Springbootapp18Application.java

```java
package com.durgasoft.springbootapp18;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp18Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp18Application.class, args);
System.out.println("This is from Spring Boot Entry Class");
}
}
```

If we execute the above application with the command line arguments like
start stop pause --server.port=1234 --application.name=runnerapp

```
Output:
This is from CRUDRunner
[server.port, application.name]
[start, stop, pause]
[runnerapp]
true
This is from Spring Boot Entry Class
```

We can provide more than one Application Runner in a single Spring Boot application and they will be executed as per the Alphabetical Order.

If we want to customize the Order then we have to use @Order(val) annotation.

EX:
CRUDRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Order(3)
@Component
public class CRUDRunner implements ApplicationRunner {
@Override
public void run(ApplicationArguments args) throws Exception {
System.out.println("This is from CRUDRunner");
}
}
```

PoolRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Order(2)
@Component
public class PoolRunner implements ApplicationRunner {
@Override
public void run(ApplicationArguments args) throws Exception {
System.out.println("This is from Pool Runner");
}
}
```

RepositoryRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Order(1)
@Component
public class RepositoryRunner implements ApplicationRunner {
@Override
public void run(ApplicationArguments args) throws Exception {
System.out.println("This is from RepositoryRunner");
}
}
```

Springbootapp18Application.java

```java
package com.durgasoft.springbootapp18;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp18Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp18Application.class, args);
System.out.println("This is from Spring Boot Entry Class");
}
}
```

```
This is from RepositoryRunner
This is from Pool Runner
This is from CRUDRunner
This is from Spring Boot Entry Class
```

IN Spring Boot applications, we can use both CommandLineRunner and ApplicationRunner in a single Spring Boot Application.

EX:
CRUDRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Order(2)
@Component
public class CRUDRunner implements ApplicationRunner {
@Override
public void run(ApplicationArguments args) throws Exception {
System.out.println("This is from CRUDRunner");
}
}
```

RepositoryRunner.java

```java
package com.durgasoft.springbootapp18.runners;

import org.springframework.boot.CommandLineRunner;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Order(1)
@Component
public class RepositoryRunner implements CommandLineRunner {
@Override
public void run(String ... args) throws Exception {
System.out.println("This is from RepositoryRunner");
}
}
```

Springbootapp18Application.java

```
package com.durgasoft.springbootapp18;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp18Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp18Application.class, args);
System.out.println("This is from Spring Boot Entry Class");
}
}
```

Java Based Configuration for the Runners:

```
@Configuration
public class AppConfig{
	@Bean
	public CommandLineRunner commandLineRunner(){
		return args-> System.out.println("This is from Command Line
						Runner");
	}
	@Bean
	public ApplicationLineRunner applicationRunner(){
		return args-> System.out.println("This is from Application
						Runner");
	}
}
```

EX:
AppConfig.java

```
package com.durgasoft.springapp19.config;

import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig{
@Bean
public CommandLineRunner commandLineRunner(){
```

```
return args -> System.out.println("This is from CommandLineRunner");
}



@Bean
public ApplicationRunner applicationRunner(){
return args -> System.out.println("This is from ApplicationRunner");
}

}
```

Springbootapp18Application.java

```
package com.durgasoft.springapp19;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springapp19Application {

public static void main(String[] args) {
SpringApplication.run(Springapp19Application.class, args);
System.out.println("This is from Spring Boot Entry class");
}


}
```

Q)What are the differences between CommandLineRunner and ApplicationRunner?

Ans:
----

1. CommandLineRunner was introduced in Spring Boot 1.0 version.
   ApplicationRunner was introduced in Spring Boot 1.3 version.

2. CommandLineRunner has the following method.
   public void run(String … args)

   ApplicationRunner has the following method.
   public void run(ApplicationArguments args)

3. CommandLineRunner is able to represent all the Option arguments and Non option arguments in the form of String[] without having a difference.

ApplicationRunner is able to represent Option arguments and Non option arguments separately.

Spring boot Profiles:

In general, a software project will have the following lifecycle stages.

Development
Testing
UAT
Pre-Production
Production

In the software applications implementations, we have to provide a separate environment for each and every lifecycle phase, here the environment means the configurations like Server configurations, Database configurations,....

In the above context, configurations are varied from one environment to another environment, to switch the project from one Environment to another environment and to provide all the configurations as per the environment if we provide all the configurations manually then it is very much difficult for the developers and it is not giving guarantee for the configurations.

To overcome all the above problems Spring Boot has provided an automated solution in the form of Spring Boot Profiles.
To use profiles in Spring boot applications we have to use the following steps.

1. Create a separate properties file for each and every environment:
   Properties file name format: application-profileName.properties
   EX:
   application.properties for Default Environment.
   application-dev.properties for Development Environment.
   application-test.properties for QA Environment
   application-prod.properties for Production Environment

2. Provide the active profile :
   To provide the active profile we have to use the following approaches.
   1. By Using application.properties:
      application.properties
      spring.profiles.active=dev

2. By Using System Properties:
   In Spring Boot Entry class, in main(), before SpringApplication.run() method we have to use the following instruction.
   System.setProperty("spring.profiles.active", "dev");

3. By using Application Arguments in run configurations:
   Application Arguments: – -spring.profiles.active=dev

4. By Using VM Arguments in Run configurations:
   VM Arguments: - -spring.profiles.active=dev

3. Run the Spring Boot Application

EX:
application.properties

```
spring.profiles.active=prod
```

application-dev.properties

```
server.port=1234
spring.application.name=ICICI Application- DEV Environment
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/icicidb
spring.datasource.username=icici-dev
spring.datasource.password=icici-dev
```

application-test.properties

```
server.port=2345
spring.application.name=ICICI Application- TEST Environment
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/icicidb
spring.datasource.username=icici-test
spring.datasource.password=icici-test
```

application-prod.properties

```
server.port=3456
spring.application.name=ICICI Application- PROD Environment
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/icicidb
spring.datasource.username=icici-prod
spring.datasource.password=icici-prod
```

ProfileEnvironment.java

```java
package com.durgasoft.springbootapp19.beans;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class ProfilesEnvironment {
@Value("${spring.application.name}")
private String applicationName;
@Value("${spring.datasource.driver-class-name}")
private String driverClassName;
@Value("${spring.datasource.url}")
private String url;
@Value("${spring.datasource.username}")
private String userName;
@Value("${spring.datasource.password}")
private String password;

public String getApplicationName() {
return applicationName;
}

public void setApplicationName(String applicationName) {
this.applicationName = applicationName;
}

public String getDriverClassName() {
return driverClassName;
}

public void setDriverClassName(String driverClassName) {
this.driverClassName = driverClassName;
}

public String getUrl() {
return url;
}
public void setUrl(String url) {
this.url = url;
}

public String getUserName() {
return userName;
}

public void setUserName(String userName) {
this.userName = userName;
}

public String getPassword() {
return password;
}

public void setPassword(String password) {
this.password = password;
}
}
```

ProfilesController

```
package com.durgasoft.springbootapp19.controller;

import com.durgasoft.springbootapp19.beans.ProfilesEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProfilesController {
@Autowired
private ProfilesEnvironment profilesEnvironment;
@RequestMapping("/profile")
public String getProfilesDetails(){
String data = "Application Name :
"+profilesEnvironment.getApplicationName()+"<br>";
data = data + "Driver Class Name :
"+profilesEnvironment.getDriverClassName()+"<br>";
data = data + "Driver URL : "+profilesEnvironment.getUrl()+"<br>";
data = data + "Database User Name :
"+profilesEnvironment.getUserName()+"<br>";
data = data + "Database password :
"+profilesEnvironment.getPassword();
return data;
}
}
```

Springbootapp19Application.java

```
package com.durgasoft.springbootapp19;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class Springbootapp19Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp19Application.class, args);



}


}
```

http://localhost:1234/profiles
—- Deve details—---
http://localhost:2345/profiles
—- Test details—---
http://localhost:3456/profiles
—- Prod details—---

Spring Boot Profiles with yml file:
—-----------------------------------
If we want to provide spring boot profiles with YML files we have to use the following steps.

application.yml
—----------------
```
spring:
      profiles:
            active: prod
- - -
spring:
      config:
            activate:
                  profiles-on: dev
------
------

- - -
Spring:
      config:
            activate:
                  profiles-on: test

-----
-----

- - -
spring:
      config:
            activate:
                  profiles-on: prod
----
----
```

EX:
application.yml

```
spring:
profiles:
active: prod
---
server:
port: 1234
spring:
application:
name: ICICI Application- DEV Environment
datasource:
driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://localhost:3306/icicidb
username: icici-dev
password: icici-dev
```

```
config:
activate:
on-profile: dev

---

server:
port: 2345
spring:
application:
name: ICICI Application- TEST Environment
datasource:
driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://localhost:3306/icicidb
username: icici-test
password: icici-test
config:
activate:
on-profile: test
---
server:
port: 3456
spring:
application:
name: ICICI Application- PROD Environment
datasource:
driver-class-name: com.mysql.cj.jdbc.Driver
url: jdbc:mysql://localhost:3306/icicidb
username: icici-prod
password: icici-prod
config:
activate:
on-profile: prod
```

ProfilesEnvironment.java

```
package com.durgasoft.springbootapp19.beans;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component
public class ProfilesEnvironment {
@Value("${spring.application.name}")
```

```
private String applicationName;
@Value("${spring.datasource.driver-class-name}")
private String driverClassName;
@Value("${spring.datasource.url}")
private String url;
@Value("${spring.datasource.username}")
private String userName;
@Value("${spring.datasource.password}")
private String password;

public String getApplicationName() {
return applicationName;
}

public void setApplicationName(String applicationName) {
this.applicationName = applicationName;
}

public String getDriverClassName() {
return driverClassName;
}

public void setDriverClassName(String driverClassName) {
this.driverClassName = driverClassName;
}

public String getUrl() {
return url;
}

public void setUrl(String url) {
this.url = url;
}

public String getUserName() {
return userName;
}

public void setUserName(String userName) {
this.userName = userName;
}

public String getPassword() {
```

```
return password;
}

public void setPassword(String password) {
this.password = password;
}
}
```

ProfilesController.java

```
package com.durgasoft.springbootapp19.controller;

import com.durgasoft.springbootapp19.beans.ProfilesEnvironment;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProfilesController {
@Autowired
private ProfilesEnvironment profilesEnvironment;
@RequestMapping("/profile")
public String getProfilesDetails(){
String data = "Application Name :
"+profilesEnvironment.getApplicationName()+"<br>";
data = data + "Driver Class Name :
"+profilesEnvironment.getDriverClassName()+"<br>";
data = data + "Driver URL : "+profilesEnvironment.getUrl()+"<br>";
data = data + "Database User Name :
"+profilesEnvironment.getUserName()+"<br>";
data = data + "Database password :
"+profilesEnvironment.getPassword();
return data;
}
}
```

Springbootapp19Application.java

```
package com.durgasoft.springbootapp19;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class Springbootapp19Application {

```

```
public static void main(String[] args) {
//System.setProperty("spring.profiles.active", "dev");
SpringApplication.run(Springbootapp19Application.class, args);
}

}
```

Spring Boot Data JDBC:

In Spring, If we want to prepare Spring JDBC Applications then we have to use the following steps.

1. Provide all the Spring dependencies.
2. Create Dao Class.
3. Declare JdbcTemplate as property.
4. Perform the database operations in Dao.
5. Provide DataSource configurations, JdbcTemplate Configurations, Dao Configurations
6. In Main class or in service class we have to access the Dao methods.

In Spring Boot, if we want to prepare a Spring JDBC application then we have to use the following steps.

1. Create Spring Boot project with Spring Initializr by providing the Dependencies Spring-data-jdbc, mysql-connector, web.
2. In the Properties file, provide all the Datasource parameters.
    a. spring.datasource.driver-class-name
    b. spring.datasource.url
    c. spring.datasource.username
    d. spring.datasource.password
3. Prepare Dao interface and Dao class with @Repository annotation. In DAO class , declare JdbcTemplate property with @Autowired annotation.
4. Prepare Service interface and Service class with @Service annotation and access Dao methods.
5. Prepare Controller class with @Controller and access Service methods
6. Prepare Runner class and access controller methods.

EX:
application.properties

```
spring.application.name=Spring Boot - Spring JDBC Application
server.port=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
```

Employee.java

```
package com.durgasoft.springbootapp20.beans;

public class Employee {
private int eno;
private String ename;
private float esal;
private String eaddr;

public int getEno() {
return eno;
}

public void setEno(int eno) {
this.eno = eno;
}

public String getEname() {
return ename;
}

public void setEname(String ename) {
this.ename = ename;
}

public float getEsal() {
return esal;
}

public void setEsal(float esal) {
this.esal = esal;
}
```

```java
public String getEaddr() {
return eaddr;
}

public void setEaddr(String eaddr) {
this.eaddr = eaddr;
}
}
```

EmployeeDao.java

```java
package com.durgasoft.springbootapp20.dao;

import com.durgasoft.springbootapp20.beans.Employee;

public interface EmployeeDao {
public String add(Employee employee);
public Employee search(int eno);
public String update(Employee employee);
public String delete(int eno);

}
```

EmployeeDaoImpl.java

```java
package com.durgasoft.springbootapp20.dao;

import com.durgasoft.springbootapp20.beans.Employee;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public class EmployeeDaoImpl implements EmployeeDao{

@Autowired
private JdbcTemplate jdbcTemplate;
@Override
public String add(Employee employee) {
String status = "";
String query = "insert into emp1
values("+employee.getEno()+",'"+employee.getEname()+"',"+employee.get
Esal()+",'"+employee.getEaddr()+"')";
int rowCount = jdbcTemplate.update(query);
if(rowCount == 1){
status = "SUCCESS";
}else{
status = "FAILURE";
}

return status;
}

@Override
public Employee search(int eno) {
List<Employee> employeeList = jdbcTemplate.query("select * from emp1
where ENO = "+eno, (rs, num)->{
Employee employee = new Employee();
employee.setEno(rs.getInt("ENO"));
employee.setEname(rs.getString("ENAME"));
employee.setEsal(rs.getFloat("ESAL"));
employee.setEaddr(rs.getString("EADDR"));
return employee;
} );
return employeeList.isEmpty()?null:employeeList.get(0);
}

@Override
public String update(Employee employee) {
String status = "";
Employee emp = search(employee.getEno());
if(emp == null){
status = "Employee Does not Exist";
}else{
String query = "update emp1 set ENAME = '"+employee.getEname()+"',
ESAL = "+employee.getEsal()+", EADDR = '"+employee.getEaddr()+"'
where ENO = "+employee.getEno();
int rowCount = jdbcTemplate.update(query);
if(rowCount == 1){
status = "SUCCESS";
}else{
status = "FAILURE";
}
}
return status;
}

@Override
public String delete(int eno) {
Employee employee = search(eno);
String status = "";
if(employee == null){
status = "Employee Does Not Exist";
}else{
int rowCount = jdbcTemplate.update("delete from emp1 where ENO =
"+eno);
if(rowCount == 1){
status = "SUCCESS";
}else{
status = "FAILURE";
}
}
return status;
}

}
```

EmployeeService.java

```
package com.durgasoft.springbootapp20.service;

import com.durgasoft.springbootapp20.beans.Employee;

public interface EmployeeService {
public String addEmployee(Employee employee);
public Employee searchEmployee(int eno);
public String updateEmployee(Employee employee);
public String deleteEmployee(int eno);

}
```

EmployeeServiceImpl.java

```
package com.durgasoft.springbootapp20.service;

import com.durgasoft.springbootapp20.beans.Employee;
import com.durgasoft.springbootapp20.dao.EmployeeDao;
import org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.stereotype.Service;

@Service
public class EmployeeServiceImpl implements EmployeeService{
@Autowired
private EmployeeDao employeeDao;
@Override
public String addEmployee(Employee employee) {
String status = employeeDao.add(employee);
return status;
}

@Override
public Employee searchEmployee(int eno) {
Employee employee = employeeDao.search(eno);
return employee;
}

@Override
public String updateEmployee(Employee employee) {
String status = employeeDao.update(employee);
return status;
}

@Override
public String deleteEmployee(int eno) {
String status = employeeDao.delete(eno);
return status;
}
}
```

EmployeeController.java

```java
package com.durgasoft.springbootapp20.controller;

import com.durgasoft.springbootapp20.beans.Employee;
import com.durgasoft.springbootapp20.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

@Controller
public class EmployeeController {
@Autowired
private EmployeeService employeeService;
public String addEmployee(Employee employee){
String status = "";
status = employeeService.addEmployee(employee);
return status;
}
public Employee searchEmployee(int eno){
Employee employee = employeeService.searchEmployee(eno);
return employee;
}

public String updateEmployee(Employee employee){
String status = employeeService.updateEmployee(employee);
return status;
}

public String deleteEmployee(int eno){
String status = employeeService.deleteEmployee(eno);
return status;
}
}
```

EmployeeRunner.java

```
package com.durgasoft.springbootapp20.runner;

import com.durgasoft.springbootapp20.beans.Employee;
import com.durgasoft.springbootapp20.controller.EmployeeController;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class EmployeeRunner implements ApplicationRunner {
@Autowired
private EmployeeController employeeController;
@Override
public void run(ApplicationArguments args) throws Exception {
/*
Employee employee = new Employee();
employee.setEno(111);
employee.setEname("Durga");
employee.setEsal(5000);
employee.setEaddr("Hyd");
String status = employeeController.addEmployee(employee);
System.out.println(status);
*/
/*
Employee employee = employeeController.searchEmployee(111);
if(employee == null){
System.out.println("Employee does not exist");
}else{
System.out.println("Employee Details");
System.out.println("----------------------");
System.out.println("Employee Number : "+employee.getEno());
System.out.println("Employee Name : "+employee.getEname());
System.out.println("Employee Salary : "+employee.getEsal());
System.out.println("Employee Address : "+employee.getEaddr());
}
*/
/*
Employee employee = new Employee();
employee.setEno(111);
employee.setEname("Anil");
employee.setEsal(6000);
employee.setEaddr("Chennai");
String status = employeeController.updateEmployee(employee);
System.out.println(status);
*/
String status = employeeController.deleteEmployee(111);
System.out.println(status);
}
}
```

Springbootapp20Application.java

```
package com.durgasoft.springbootapp20;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp20Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp20Application.class, args);
}

}
```

pom.xml

```
<?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>3.1.0</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.durgasoft</groupId>
<artifactId>springbootapp20</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootapp20</name>
<description>springbootapp20</description>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
```

```
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

In the above Spring Boot - Spring JDBC application, in the Dao implementation class we have to provide implementation for all the CRUD operations, these CRUD operations implementations common in all the Spring Boot applications, so Spring Boot has provided the predefined implementations for the all CRUD operations in the form of an interface that is CrudRepository.

IN the Repository or in the DAO interface just extend the CrudRepository interface automatically all CRUD methods are available to the DAO class, no need to implement crud operations explicitly.

CrudRepository interface has provided the following methods to perform the persistence operations.

save(): It will perform save / update operations.
delete(): Delete a record
findById(--): Get a record.
findAll(): Get all records
deleteById(--): Delete a record
deleteAll(): Delete All records


```
public interface EmployeeReposiroty extends CrudRepository{
      // Here we get all CRUD operation methods from CrudRepository--

}
```

If we want to use CrudRepository in Spring Boot applications then we have to use the following steps.

1. Create a Spring Boot project with the following dependencies.
   web
   Spring data jdbc
   Mysql connector

2. Create a Table in the database with the auto_increment feature to the primary key.
   sql>create table emp1(ENO int(3) primary key auto_increment, ENAME char(10), ESAL float(5), EADDR char(10));
3. Prepare a bean class or model class , it must be annotated with @Table and the id property must be annotated with @Id annotation.

   ```
   @Table(name=”emp1”)
   public class Employee{
         @Id
         private int eno;
         —---
   }
   ```

4. Prepare Repository interface by extending CrudRepository.

   ```
   @Repository
   public EmployeeRepository extends CrudRepository<Employee, Integer>
   {
   }
   ```

5. Prepare Service class with @Service annotation and every method in the Service class must be annotated with @Transactional

   ```
   @Service
   public class EmployeeServiceImpl implements EmployeeService{
         @Autowired
         private EmployeeRepository employeeRepository;

         public Employee addEmployee(Employee employee){
               Employee emp = employeeRepository.save(employee);
               return emp;
         }
   }
   ```

6. Prepare Controller class with @Controller

   ```
   @Controller
   ```

```
public class EmployeeController{
      @Autowired
      private EmployeeService employeeService;
      public Employee addEmployee(Employee employee){
            return employeeService.addEmployee(employee);
      }
}
```

7. Prepare a Runner class to test the application.

```
@Component
public class EMployeeRunner implements ApplicationRunner{
      @Autowired
      private EmployeeController employeeController;

      Employee emp = new Employee();
      emp.setEno(111);
      emp.setEname("Durga");
      emp.setEsal(5000);
      emp.setEaddr("Hyd");

      Employee e = employeeController.addEmployee(emp);
      System.out.println(e);


}
```

EX:

application.properties

```
server.port=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
```

Employee.java

```
package com.durgasoft.springbootapp21.beans;

import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;
```

```java
@Table(name="emp1")
public class Employee {
@Id
private int eno;
private String ename;
private float esal;
private String eaddr;

public int getEno() {
return eno;
}

public void setEno(int eno) {
this.eno = eno;
}

public String getEname() {
return ename;
}

public void setEname(String ename) {
this.ename = ename;
}

public float getEsal() {
return esal;
}

public void setEsal(float esal) {
this.esal = esal;
}

public String getEaddr() {
return eaddr;
}

public void setEaddr(String eaddr) {
this.eaddr = eaddr;
}

@Override
public String toString() {
return "Employee{" +
"eno=" + eno +
", ename='" + ename + '\'' +
", esal=" + esal +
", eaddr='" + eaddr + '\'' +
'}';
}
}
```

EmployeeRunner.java

```
package com.durgasoft.springbootapp21.runner;

import com.durgasoft.springbootapp21.beans.Employee;
import
com.durgasoft.springbootapp21.controller.EmployeeController;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

import java.util.Iterator;

@Component
public class EmployeeRunner implements ApplicationRunner {
@Autowired
private EmployeeController employeeController;
@Override
public void run(ApplicationArguments args) throws Exception {
/*
Employee employee = new Employee();
//employee.setEno(111);
employee.setEname("Durga");
employee.setEsal(5000);
employee.setEaddr("Hyd");
Employee emp = employeeController.addEmployee(employee);
System.out.println(emp);
*/
/*
Employee employee = employeeController.searchEmployee(3);
System.out.println(employee);
*/
/*
Iterable<Employee> iterable =
employeeController.getAllEmployees();
```

```java
Iterator<Employee> iterator = iterable.iterator();
while (iterator.hasNext()){
Employee employee = iterator.next();
System.out.println(employee);
}
*/
/*
Employee employee = new Employee();
employee.setEno(3);
employee.setEname("XXX");
employee.setEsal(9000);
employee.setEaddr("Chennai");

Employee emp = employeeController.updateEmployee(employee);
System.out.println(emp);
if(emp.getEno() == employee.getEno()){
System.out.println("SUCCESS");
}else{
System.out.println("FAILURE");
}
*/
String status = employeeController.deleteEmployee(5);
System.out.println(status);
}
}
```

EmployeeController.java

```java
package com.durgasoft.springbootapp21.controller;

import com.durgasoft.springbootapp21.beans.Employee;
import com.durgasoft.springbootapp21.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

@Controller
public class EmployeeController {
@Autowired
private EmployeeService employeeService;
public Employee addEmployee(Employee employee){
Employee employee1 = employeeService.addEmployee(employee);
return employee1;
}
public Employee searchEmployee(int eno){
```

```
Employee employee = employeeService.searchEmployee(eno);
return employee;
}
public Iterable<Employee> getAllEmployees(){
return employeeService.getAllEmployees();
}

public Employee updateEmployee(Employee employee){
return employeeService.updateEmployee(employee);
}
public String deleteEmployee(int eno){
return employeeService.deleteEmployee(eno);
}

}
```

EmployeeService.java

```
package com.durgasoft.springbootapp21.service;

import com.durgasoft.springbootapp21.beans.Employee;

import java.util.List;

public interface EmployeeService {
public Employee addEmployee(Employee employee);
public Employee searchEmployee(int eno);
public Iterable getAllEmployees();
public Employee updateEmployee(Employee employee);
public String deleteEmployee(int eno);
}
```

EmployeeServiceImpl.java

```
package com.durgasoft.springbootapp21.service;

import com.durgasoft.springbootapp21.beans.Employee;
import
com.durgasoft.springbootapp21.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import
org.springframework.transaction.annotation.Transactional;

import java.util.List;
```

```java
import java.util.Optional;

@Service
public class EmployeeServiceImpl implements EmployeeService{

@Autowired
private EmployeeRepository employeeRepository;
@Transactional
@Override
public Employee addEmployee(Employee employee) {
Employee employee1 = employeeRepository.save(employee);
return employee1;
}

@Override
public Employee searchEmployee(int eno) {
Optional<Employee> optional = employeeRepository.findById(eno);
Employee employee = optional.get();
return employee;
}

@Override
public Iterable getAllEmployees() {
Iterable<Employee> iterable = employeeRepository.findAll();
return iterable;
}

@Override
public Employee updateEmployee(Employee employee) {
Employee employee1 = employeeRepository.save(employee);

return employee1;
}

@Override
public String deleteEmployee(int eno) {
employeeRepository.deleteById(eno);
return "SUCCESS";
}



}
```

EmployeeRepository.java

```java
package com.durgasoft.springbootapp21.repository;

import com.durgasoft.springbootapp21.beans.Employee;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface EmployeeRepository extends
CrudRepository<Employee, Integer> {

}
```

Sprongbootapp21Application.java

```java
package                              ;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp21Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp21Application.class, args);
}
}
```

pom.xml

```xml
<?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>3.1.1</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.durgasoft</groupId>
<artifactId>springbootapp21</artifactId>
```

```xml
<version>0.0.1-SNAPSHOT</version>
<name>springbootapp21</name>
<description>springbootapp21</description>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

In the CrudRepository we are able to define our own methods with @Query annotation.

If we want to provide select sql query with @Query annotation then it is not required to use @Modifying annotation, but if we want to provide non select sql query along with @Query annotation then we must provide @Modifying annotation.

EX:
```
@Modifying
@Query(“update emp1 set …. “)
public int update(----);
```

Note: In the case of Non select sql queries , Repository method will return rowCount value in the form of integer value.

EX:
```
@Query(“select ….“)
public int findByEmail(----)
```

EX:
Employee.java
```
package com.durgasoft.springbootapp21.beans;

import org.springframework.data.annotation.Id;
import org.springframework.data.relational.core.mapping.Table;


@Table(name="emp1")
public class Employee {
@Id
private int eno;
private String ename;
private float esal;
private String eaddr;

public int getEno() {
return eno;
}

public void setEno(int eno) {
this.eno = eno;
}

public String getEname() {
return ename;
```

```java
}

public void setEname(String ename) {
this.ename = ename;
}

public float getEsal() {
return esal;
}

public void setEsal(float esal) {
this.esal = esal;
}

public String getEaddr() {
return eaddr;
}

public void setEaddr(String eaddr) {
this.eaddr = eaddr;
}

@Override
public String toString() {
return "Employee{" +
"eno=" + eno +
", ename='" + ename + '\'' +
", esal=" + esal +
", eaddr='" + eaddr + '\'' +
'}';
}
}
```

EmployeeRunner.java

```java
package com.durgasoft.springbootapp21.runner;

import com.durgasoft.springbootapp21.beans.Employee;
import com.durgasoft.springbootapp21.controller.EmployeeController;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;
import java.util.Iterator;

@Component
public class EmployeeRunner implements ApplicationRunner {
@Autowired
private EmployeeController employeeController;
@Override
public void run(ApplicationArguments args) throws Exception {
/*
Employee employee = new Employee();
//employee.setEno(111);
employee.setEname("Durga");
employee.setEsal(5000);
employee.setEaddr("Hyd");
Employee emp = employeeController.addEmployee(employee);
System.out.println(emp);
*/
/*
Employee employee = employeeController.searchEmployee(3);
System.out.println(employee);
*/
/*
Iterable<Employee> iterable = employeeController.getAllEmployees();
Iterator<Employee> iterator = iterable.iterator();
while (iterator.hasNext()){
Employee employee = iterator.next();
System.out.println(employee);
}
*/

Employee employee = new Employee();
employee.setEno(3);
employee.setEname("XXX");
employee.setEsal(9000);
employee.setEaddr("Pune");

int rowCount = employeeController.updateEmployee(employee);
System.out.println(rowCount);
if(rowCount == 1){
System.out.println("SUCCESS");
}else{
System.out.println("FAILURE");
}
/*
String status = employeeController.deleteEmployee(5);
System.out.println(status);
*/
}
}
```

EmployeeController.java

```
package com.durgasoft.springbootapp21.controller;

import com.durgasoft.springbootapp21.beans.Employee;
import com.durgasoft.springbootapp21.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

@Controller
public class EmployeeController {
@Autowired
private EmployeeService employeeService;
public Employee addEmployee(Employee employee){
Employee employee1 = employeeService.addEmployee(employee);
return employee1;
}
public Employee searchEmployee(int eno){
Employee employee = employeeService.searchEmployee(eno);
return employee;
}
public Iterable<Employee> getAllEmployees(){
return employeeService.getAllEmployees();
}

public int updateEmployee(Employee employee){
return employeeService.updateEmployee(employee);
}
public String deleteEmployee(int eno){
return employeeService.deleteEmployee(eno);
}

}
```

EmployeeService.java

```
package com.durgasoft.springbootapp21.service;
```

```java
import com.durgasoft.springbootapp21.beans.Employee;

import java.util.List;

public interface EmployeeService {
public Employee addEmployee(Employee employee);
public Employee searchEmployee(int eno);
public Iterable getAllEmployees();
public int updateEmployee(Employee employee);
public String deleteEmployee(int eno);
}
```

EmployeeServiceImpl.java

```java
package com.durgasoft.springbootapp21.service;

import com.durgasoft.springbootapp21.beans.Employee;
import com.durgasoft.springbootapp21.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

@Service
public class EmployeeServiceImpl implements EmployeeService{

@Autowired
private EmployeeRepository employeeRepository;
@Transactional
@Override
public Employee addEmployee(Employee employee) {
Employee employee1 = employeeRepository.save(employee);
return employee1;
}

@Override
public Employee searchEmployee(int eno) {
Optional<Employee> optional = employeeRepository.findById(eno);
Employee employee = optional.get();
return employee;
}
@Override
public Iterable getAllEmployees() {
Iterable<Employee> iterable = employeeRepository.findAll();
return iterable;
}

@Override
public int updateEmployee(Employee employee) {
//Employee employee1 = employeeRepository.save(employee);
Integer rowCount = employeeRepository.update(employee.getEno(),
employee.getEname(), employee.getEsal(), employee.getEaddr());
return rowCount;
}

@Override
public String deleteEmployee(int eno) {
employeeRepository.deleteById(eno);
return "SUCCESS";
}



}
```

EmployeeRepository.java

```java
package com.durgasoft.springbootapp21.repository;

import com.durgasoft.springbootapp21.beans.Employee;

import org.springframework.data.jdbc.repository.query.Modifying;
import org.springframework.data.jdbc.repository.query.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;


@Repository
public interface EmployeeRepository extends CrudRepository<Employee,
Integer> {

@Modifying
@Query("update emp1 set ENAME=:ename, ESAL=:esal, EADDR=:eaddr where
ENO=:eno")
public Integer update(int eno, String ename, float esal, String
eaddr);
}
```

application.properties

```
server.port=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
```

pom.xml

```
<?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>3.1.1</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.durgasoft</groupId>
<artifactId>springbootapp21</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootapp21</name>
<description>springbootapp21</description>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

PagingAndSortingRepository:
—-------------------------
Its main purpose is to sort all the results and generate the results as per the pagination.

In Spring Data Jdbc applications, if we want to sort all the results then we have to use the “Sort” class like below.

```
Sort sort = Sort.by(Sort.Direction.fromString("ASC"), "saddr");
Iterable<Student> iterable = studentRepository.findAll(sort);
```

The “by()” method is able to define the Sorting direction like Ascending or Descending and the parameter on which we want to perform Sorting.

To get all the elements from the Database with a particular Sorting order we have to use the following method.

public Iterable findAll(Sort sort)

In Spring Data JDBC if we want to read all the results page by page , that is as per Pagination then we have to use the predefined libraries “PageRequest” and “Page” like below.

```
PageRequest pageRequest = PageRequest.of(0,3);
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
```

Where PageRequest is able to manage the page number and number of results in the respective page by using the following static method.

public static PageRequest of(int pageNo, int numOfRecords)

Where Page object is able to manage the records which are retrieved as per the PageRequest object provided page number and number of records. To get a Page object we have to access the findAll() method from the Repository interface with the pageRequest parameter.

public Page findAll(PageRequest pageRequest)

If we want to get all Results from Page object to List we have to use the following code.
List<Student> stdList = page.stream().toList();
In Spring Data JDBC if we want to read all the results page by page , that is as per Pagination with a particular Sorting order then we have to use the following code.

```
PageRequest pageRequest = PageRequest.of(0,3,
Sort.Direction.fromString("DESC"),"sname");
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
```

EX:
Student.java

```
package com.durgasoft.springbootapp22.beans;

public class Student {
private String sid;
private String sname;
private String saddr;
public String getSid() {
return sid;
}

public void setSid(String sid) {
this.sid = sid;
}

public String getSname() {
return sname;
}

public void setSname(String sname) {
this.sname = sname;
}

public String getSaddr() {
return saddr;
}

public void setSaddr(String saddr) {
this.saddr = saddr;
}
}
```

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
```

StudentRepository.java

```
package com.durgasoft.springbootapp22.repository;

import com.durgasoft.springbootapp22.beans.Student;
import
org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentRepository extends
PagingAndSortingRepository<Student, String> {
}
```

StudentService.java

```
package com.durgasoft.springbootapp22.service;

import com.durgasoft.springbootapp22.beans.Student;

import java.util.List;

public interface StudentService {
public List<Student> getAllStudents();
}
```

StudentServiceImpl.java

```
package com.durgasoft.springbootapp22.service;

import com.durgasoft.springbootapp22.beans.Student;
import com.durgasoft.springbootapp22.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import java.util.Iterator;
import java.util.List;

@Service
public class StudentServiceImpl implements StudentService{
@Autowired
private StudentRepository studentRepository;
@Override
public List<Student> getAllStudents() {
/*
Sort sort = Sort.by(Sort.Direction.fromString("DESC"), "saddr");
Iterable<Student> iterable = studentRepository.findAll(sort);
List<Student> stdList = (List<Student>) iterable;
*/
/*
PageRequest pageRequest = PageRequest.of(0,3);
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
*/
PageRequest pageRequest = PageRequest.of(0,3,
Sort.Direction.fromString("DESC"),"sname");
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
return stdList;
}
}
```

StudentController.java

```
package com.durgasoft.springbootapp22;

import com.durgasoft.springbootapp22.beans.Student;
import com.durgasoft.springbootapp22.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import java.util.List;

@Controller
public class StudentController {
@Autowired
private StudentService studentService;
public List<Student> getAllStudents(){
return studentService.getAllStudents();
}
}
```

StudentRunner.java

```
package com.durgasoft.springbootapp22.runner;

import com.durgasoft.springbootapp22.StudentController;
import com.durgasoft.springbootapp22.beans.Student;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class StudentRunner implements ApplicationRunner {
@Autowired
```

```java
private StudentController studentController;
@Override
public void run(ApplicationArguments args) throws Exception {
List<Student> stdList = studentController.getAllStudents();
System.out.println("SID\tSNAME\tSADDR");
System.out.println("---------------------------");
for(Student std: stdList){
System.out.print(std.getSid()+"\t");
System.out.print(std.getSname()+"\t\t");
System.out.print(std.getSaddr()+"\n");
}
}
}
```

Springbootapp22Application.java

```java
package com.durgasoft.springbootapp22;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp22Application {

public static void main(String[] args) {
SpringApplication.run(Springbootapp22Application.class, args);
}


}
```

pom.xml

```xml
<?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>3.1.2</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.durgasoft</groupId>
```

```xml
<artifactId>springbootapp22</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>springbootapp22</name>
<description>springbootapp22</description>
<properties>
<java.version>17</java.version>
</properties>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>com.mysql</groupId>
<artifactId>mysql-connector-j</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>

<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>

</project>
```

```
Spring Data JPA:
-----------------
JPA: Java Persistence API
```

JPA is an API, it can be used to perform database operations in enterprise applications with ORM implementation tools.

JPA was provided by J2EE along with EJB3.0 version as a persistence mechanism.

JPA is a specification provided by SUN Microsystems and it is implemented by third party vendors like JBOSS, Eclipse Foundation, Apache, Weblogic...

JPA is following ORM rules and regulations to implement data persistence in enterprise applications and it is implemented by the following tools.

1. Hibernate
2. Eclipse Link
3. Open JPA
   —----
   —----

Note: If we want to use JPA in enterprise applications then we have to use either of the above JPA implementation tools.

If we want to use JPA in enterprise applications we have to use the following steps.

1. Create a Java project with JPA libraries including Hibernate JARs.
2. Prepare Entity class[Bean class]
3. Create Mapping File or Use Annotations in Entity class.
4. Create JPA configuration File[persistence.xml]
5. Create Test Application

Create Entity class:
Student.java

```
public class Student{
      private String sid;
      private String sname;
      private String saddr;
      —--
}
```

Create Mapping File:
It is the same as the Hibernate Mapping file.

```
Student.xml
<hibernate-mapping>
      <class name=”com.durgasoft.entities.Student” table=”student”>
            <id name=”sid” column=”SID”/>
            <property name=”sname” column=”SNAME”/>
            <property name=”saddr” column=”SADDR”/>
      </class>
</hibernate-mapping>
```

Create JPA Configuration File:
It is similarly to Hibernate configuration file, it is able to take all the JPA configuration details like

1. JDBC parameters like Driver class name, Driver URL, database user name, database password,....
2. Dialect Configuration
3. Mapping Files COnfiguration
4. Transactions configurations
   —---
   —----

The default name of the JPA configuration file is “persistence.xml”.

```
persistence.xml
<persistence>
  <persistence-unit name=”mysql”>
      <mapping-file>Student.xml</mapping-file>  or
      <class>com.durgasoft.entities.Student</class>in the case of Annotations
      <properties>
         <property name=”javax.persistence.jdbc.driver”
                  value=”com.mysql.cj.jdbc.Driver”/>
         <property name=”javax.persistence.jdbc.url”
                  value=”jdbc:mysql://localhost:3306/durgadb”/>
         <property name=”javax.persistence.jdbc.user” value=”root”>
         <property name=”javax.persistence.jdbc.password” value=”root”>
         <property name=”hibernate.dialect”
                  value=”org.hibernate.dialect.MySQLDialect”/>
                  —----
      </properties>
  </persistence-unit>
```

```
</persistence>
```

Create Test Application:

1. Create EntityManagerFactory object
2. Create EntityManager object
3. Create EntityTransaction object
4. Perform Persistence operations
5. Perform commit or rollback operations if we use EntityTransaction

Create EntityManagerFactory object:
I JPA, EntityManagerFactory object is same as SessionFactory in Hibernate, it is able to take all Jdbc parameters like driver class, driver url , Database USername and Password, dialect class,... and it is able to generate EntityManager object.

To get an EntityManagerFactory object we have to use the following method from the Persistence class.

```
public static EntityManagerFactory getEntityManagerFactory(String
persistence_Unit_Name);
EX:
EntityManagerFactory factory =
Persistence.createEntityManagerFactory("mysql");
```

2. Create EntityManager object:
EntityManager is the same as Session object in Hibernate, it is able to define predefined methods in order to perform the database operations.

To get an EntityManager object we have to use the following method from EntityManagerFactory.

```
public EntityManager createEntityManager()

EX: EntityManager entityManager = factory.createEntityManager();
```

3. Create EntityTransaction Object:
It is the same as the Transaction object in Hibernate, EntityTransaction is able to provide Transactions Support in the JPA applications in order to perform Persistence operations.

To get an EntityTransaction object we have to use the following method from the EntityManager class.

public EntityTransaction createEntityTransaction()

EX: EntityTransaction entityTransaction =
entityManager.createEntitTransaction();

Note: If we use EntityTransaction in JPA applications then we have to perform either commit or rollback operations at the end of the Transaction.

public void commit()
public void rollback()

Note: EntityTransaction is required for only Non select operations, not for Select operations.

4. Perform Persistence operations:
To perform Persistence operations we have to use the following methods.

1. public Object find(Class entity, Serializable pkVal)
2. public void persist(Object entity)
3. public void remove(Object entity)

Note: To perform Updations, first we have to get the Object from the database table, perform modifications in the Entity object and perform commit operation.

Note: To prepare JPA Application we have to use Hibernate-core dependency in the pom.xml file.

EX:
Employee.java

```java
package com.durgasoft.entity;

public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }

   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

Employee.xml

```xml
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="com.durgasoft.entity.Employee" table="emp2">
        <id name="eno"/>
        <property name="ename"/>
        <property name="esal"/>
        <property name="eaddr"/>
    </class>
</hibernate-mapping>
```

persistence.xml

```xml
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
```

```
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
         https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
            version="3.0">
   <persistence-unit name="emp">
       <mapping-file>Employee.xml</mapping-file>
       <properties>
           <property name="jakarta.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver"/>
           <property name="jakarta.persistence.jdbc.url"
value="jdbc:mysql://localhost:3300/durgadb"/>
           <property name="jakarta.persistence.jdbc.user"
value="root"/>
           <property name="jakarta.persistence.jdbc.password"
value="root"/>
           <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
           <property name="hibernate.show_sql" value="true"/>
           <property name="hibernate.format_sql" value="true"/>
       </properties>
   </persistence-unit>
</persistence>
```

Main.java

```
package com.durgasoft;

import com.durgasoft.entity.Employee;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;

public class Main {
   public static void main(String[] args)throws Exception {
       EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("emp");
       EntityManager entityManager =
entityManagerFactory.createEntityManager();

       /*Employee employee = new Employee();
       employee.setEno(111);
       employee.setEname("AAA");
       employee.setEsal(5000);
       employee.setEaddr("Hyd");
       EntityTransaction entityTransaction =
entityManager.getTransaction();
       entityTransaction.begin();
       entityManager.persist(employee);
       entityTransaction.commit();
       System.out.println("Employee Inserted Successfully");*/

       /*Employee employee = entityManager.find(Employee.class, 111);
       System.out.println("Employee Details");
       System.out.println("-----------------------");
       System.out.println("Employee Number     :
"+employee.getEno());
       System.out.println("Employee Name       :
"+employee.getEname());
       System.out.println("Employee Salary     :
"+employee.getEsal());
       System.out.println("Employee Address    :
"+employee.getEaddr());*/

       /*EntityTransaction entityTransaction =
entityManager.getTransaction();
       entityTransaction.begin();
       Employee employee = entityManager.find(Employee.class, 111);
       employee.setEname("XXX");
       employee.setEsal(9000);
       employee.setEaddr("Hyd");
       entityTransaction.commit();
       System.out.println("Employee Updated Successfully");*/

       EntityTransaction entityTransaction =
entityManager.getTransaction();
       Employee employee = entityManager.find(Employee.class, 111);
       entityTransaction.begin();
       entityManager.remove(employee);
       entityTransaction.commit();
       System.out.println("Employee Removed Successfully");

       entityManager.close();
       entityManagerFactory.close();
   }
}
```

pom.xml

```xml
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.durgasoft</groupId>
   <artifactId>jpaapp01</artifactId>
   <version>1.0-SNAPSHOT</version>

   <properties>
       <maven.compiler.source>17</maven.compiler.source>
       <maven.compiler.target>17</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.hibernate.orm</groupId>
           <artifactId>hibernate-core</artifactId>
           <version>6.3.0.CR1</version>
       </dependency>
       <dependency>
           <groupId>mysql</groupId>
           <artifactId>mysql-connector-java</artifactId>
           <version>8.0.33</version>
       </dependency>
   </dependencies>

</project>
```

Annotations in JPA:

If we want to use Annotations in a JPA application we have to use the following steps.

1. Use jakarta.persistence provided annotations.
   @Entity
   @Table

```
@Id
@Column
```

2. Configure Annotation class in persistence.xml

```
<persistence>
      <persistence-unit name=”emp”>
            <class>com.durgasoft.entity.Student</class>
            -----
      </persistence-unit>
</persistence>
```

EX

Employee.java

```java
package com.durgasoft.entity;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "emp2")
public class Employee {
   @Id
   @Column(name = "ENO")
   private int eno;
   @Column(name = "ENAME")
   private String ename;
   @Column(name = "ESAL")
   private float esal;
   @Column(name = "EADDR")
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
```

```java
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public float getEsal() {
        return esal;
    }

    public void setEsal(float esal) {
        this.esal = esal;
    }

    public String getEaddr() {
        return eaddr;
    }

    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }
}
```

persistence.xml

```xml
<persistence xmlns="https://jakarta.ee/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="https://jakarta.ee/xml/ns/persistence
         https://jakarta.ee/xml/ns/persistence/persistence_3_0.xsd"
             version="3.0">
   <persistence-unit name="emp">
<!--         <mapping-file>Employee.xml</mapping-file>-->
       <class>com.durgasoft.entity.Employee</class>
       <properties>
           <property name="jakarta.persistence.jdbc.driver"
value="com.mysql.cj.jdbc.Driver"/>
           <property name="jakarta.persistence.jdbc.url"
value="jdbc:mysql://localhost:3300/durgadb"/>
           <property name="jakarta.persistence.jdbc.user"
value="root"/>
           <property name="jakarta.persistence.jdbc.password"
value="root"/>
           <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect"/>
```

```
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

Main.java

```
package com.durgasoft;

import com.durgasoft.entity.Employee;
import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityManagerFactory;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Persistence;

public class Main {
   public static void main(String[] args)throws Exception {
       EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("emp");
       EntityManager entityManager =
entityManagerFactory.createEntityManager();

       /*Employee employee = new Employee();
       employee.setEno(111);
       employee.setEname("AAA");
       employee.setEsal(5000);
       employee.setEaddr("Hyd");
       EntityTransaction entityTransaction =
entityManager.getTransaction();
       entityTransaction.begin();
       entityManager.persist(employee);
       entityTransaction.commit();
       System.out.println("Employee Inserted Successfully");*/

       /*Employee employee = entityManager.find(Employee.class, 111);
       System.out.println("Employee Details");
       System.out.println("-----------------------");
       System.out.println("Employee Number      :
"+employee.getEno());
       System.out.println("Employee Name        :
"+employee.getEname());
       System.out.println("Employee Salary      :
"+employee.getEsal());
```

```
       System.out.println("Employee Address    :
"+employee.getEaddr());*/

       /*EntityTransaction entityTransaction =
entityManager.getTransaction();
       entityTransaction.begin();
       Employee employee = entityManager.find(Employee.class, 111);
       employee.setEname("XXX");
       employee.setEsal(9000);
       employee.setEaddr("Hyd");
       entityTransaction.commit();
       System.out.println("Employee Updated Successfully");*/

       EntityTransaction entityTransaction =
entityManager.getTransaction();
       Employee employee = entityManager.find(Employee.class, 111);
       entityTransaction.begin();
       entityManager.remove(employee);
       entityTransaction.commit();
       System.out.println("Employee Removed Successfully");

       entityManager.close();
       entityManagerFactory.close();
   }
}
```

pom.xml

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.durgasoft</groupId>
   <artifactId>jpaapp01</artifactId>
   <version>1.0-SNAPSHOT</version>

   <properties>
       <maven.compiler.source>17</maven.compiler.source>
       <maven.compiler.target>17</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
```

```
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.hibernate.orm</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>6.3.0.CR1</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>
    </dependencies>

</project>
```

Spring With JPA:

If we want to use JPA in the Spring application in SPring ORM module then we have to use the following steps.

1. Create Java Project with both Spring and JPA[Hibernate] dependencies.
2. Create POJO class
3. Create a Dao interface.
4. Create a DAO implementation class.
5. Create Mapping file
6. Create Spring COnfiguration File
7. Create Test Application

1. Create Java Project with both Spring and JPA[Hibernate] dependencies:
   In the Java project we have to use Spring ORM dependency and MySQL Driver dependency and Hibernate dependency.

2. Create POJO class:

```
public class Product{
      private String pid;
      private String pname;
      private int pcost;

      ----
}
```

3. Create a Dao interface:

```
public interface ProductDao{
      public String add(Product product);
      public Product search(String pid);
      public String update(Product p);
      public String delete(int pid);
}
```

4. Create a DAO implementation class:

```
@Repository
public class ProductDaoImpl implements ProductDao{
      @PersistenceContext
      Private EntityManager entityManager;
      @Transactional
      public String add(Product product){   }
      public Product search(String pid){   }
      @Transactional
      public String update(Product p){   }
      @Transactional
      public String delete(int pid){   }
}
```

5. Create Mapping file

   It is the same as the Hibernate mapping file.

```
<hibernate-mapping>
      <class name=”com.dss.entity.Product” table=”product”>
            <id name=”pid”/>
            <property name=”pname”/>
            <property name=”pcost”/>
      </class>
</hibernate-mapping>
```

8. Create Spring Configuration File:

   In Spring - JPA integration we have to use the following configuration details in the Spring configuration file.

   a. DataSource Configuration:
      driverClassName
      url
      userName
      password
   b. EntityManagerFactoryBean:
      dataSource

```
      persistenceUnitName
      jpaVendorAdaptrer
      mappingResources
      jpaProperties
   c. TransactionManager
      entityManagerFactory
   d. ProductDao

<beans>
  <context:annotation-config/> or <context:component-scan
                  base-package=”com.durgasoft.*”/>
  <tx:annotation-driven/>
  <bean name=”dataSource” class=”0rg…DriverManagerDataSource”>
      <property name=”driverClassName”
                value=”com.mysql.cj.jdbc.Driver”/>
      <property name=”url”
                value=”jdbc:mysql://localhost:3300/durgadb”/>
      <property name=”username” value=”root”/>
      <property name=”password” value=”root”/>
  </bean>
  <bean name=”entityManagerFactoryBean”
        class=”org….LocalContainerEntityManagerFactoryBean”>
        <property name=”dataSource” ref=”dataSource”/>
        <property name=”persistenceUnit” value=”emp2”/>
        <property name=”jpaVendorAdapter”
                  value=”HibernateJpaVendorAdapter”/>
        <property name=”mappingResources”>
           <list>
              <value>Product.xml</value>
           </list>
        </property>
        <property name=”jpaProperties”>
            <props>
              <prop key=”hibernate.dialect”>MySQLDialect</prop>
              <prop key=”hibernate.show_sql”>true</prop>
            </props>
        </property>
  </bean>
  <bean name=”transactionManager” class=”org..JpaTransactionManager”>
      <property name=”entityManagerFactory”
               ref=”entityManagerFactoryBean”/>
  </bean>
  <bean name=”productDao” class=”com.dss.dao.ProductDao”/>
```

```
</beans>
```

9. Create Test Application

```
public class Test{
      public static void main(String[] args){
            ApplContx ac = new ClassPathXmlApplConxt(---);
            ProductDao pdao = ac.getBean(“productDao”);
            Product p = new Product();
            --
            pdao.add(p);
      }
}
```

EX:
Employee.java

```
package com.durgasoft.entities;

public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }

   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

EmployeeDao.java

```
package com.durgasoft.dao;

import com.durgasoft.entities.Employee;

public interface EmployeeDao {
    public String add(Employee employee);
    public Employee search(int eno);
    public String update(Employee employee);
    public String delete(int eno);
}
```

EmployeeDaoImpl.java

```
package com.durgasoft.dao;

import com.durgasoft.entities.Employee;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public class EmployeeDaoImpl implements EmployeeDao{
    String status = "";

    @PersistenceContext
```

```java
   private EntityManager entityManager;
   @Transactional
   @Override
   public String add(Employee employee) {
       entityManager.persist(employee);
       status = "SUCCESS";
       return status;
   }


   @Override
   public Employee search(int eno) {
       Employee employee = entityManager.find(Employee.class, eno);
       return employee;
   }

   @Transactional
   @Override
   public String update(Employee employee) {
       Employee emp = entityManager.find(Employee.class,
employee.getEno());
       emp.setEname(employee.getEname());
       emp.setEsal(employee.getEsal());
       emp.setEaddr(employee.getEaddr());
       status = "SUCCESS";
       return status;
   }

   @Transactional
   @Override
   public String delete(int eno) {
       Employee employee = entityManager.find(Employee.class, eno);
       String status = "";
       if(employee == null){
           status = "Employee Does Not Exist";
       }else{
           entityManager.remove(employee);
           status = "SUCCESS";
       }
       return status;
   }
```

```
}
```

Main.java

```
package com.durgasoft;
import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.entities.Employee;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {
   public static void main(String[] args) {
       ApplicationContext applicationContext = new
ClassPathXmlApplicationContext("Spring-Config.xml");
       EmployeeDao employeeDao = (EmployeeDao)
applicationContext.getBean("employeeDao");
      /* Employee employee = new Employee();
       employee.setEno(111);
       employee.setEname("Durga");
       employee.setEsal(5000);
       employee.setEaddr("Hyd");

       String status = employeeDao.add(employee);
       System.out.println(status);
       */

       /*Employee employee = employeeDao.search(111);
       if(employee == null){
           System.out.println("Employee Does not exist");
       }else{
           System.out.println("Employee Details");
           System.out.println("-------------------------");
           System.out.println("Employee Number    :
"+employee.getEno());
           System.out.println("Employee Name      :
"+employee.getEname());
           System.out.println("Employee Salary    :
"+employee.getEsal());
           System.out.println("Employee Address   :
"+employee.getEaddr());
       }*/
```

```
        /*Employee employee = new Employee();
        employee.setEno(111);
        employee.setEname("XXX");
        employee.setEsal(9000);
        employee.setEaddr("Pune");
        String status = employeeDao.update(employee);
        System.out.println(status);*/

        String status = employeeDao.delete(111);
        System.out.println(status);
    }
}
```

Employee.xml

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="com.durgasoft.entities.Employee" table="emp2">
       <id name="eno"/>
       <property name="ename"/>
       <property name="esal"/>
       <property name="eaddr"/>
   </class>
</hibernate-mapping>
```

Spring-Config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

      xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:context="http://www.springframework.org/schema/context"
      xsi:schemaLocation="
       http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/tx
       http://www.springframework.org/schema/tx/spring-tx.xsd

       http://www.springframework.org/schema/context
```

```xml
http://www.springframework.org/schema/context/spring-context.xsd">
   <context:component-scan base-package="com.durgasoft.*"/>
   <tx:annotation-driven/>
   <bean name="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       <property name="driverClassName"
value="com.mysql.cj.jdbc.Driver"/>
       <property name="url"
value="jdbc:mysql://localhost:3300/durgadb"/>
       <property name="username" value="root"/>
       <property name="password" value="root"/>
   </bean>
   <bean name="entityManagerFactoryBean"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactory
Bean">
       <property name="dataSource" ref="dataSource"/>
       <property name="persistenceUnitName" value="emp"/>
       <property name="jpaVendorAdapter">
           <bean
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"/
>
       </property>
       <property name="mappingResources">
           <list>
               <value>Employee.xml</value>
           </list>
       </property>
       <property name="jpaProperties">
           <props>
               <prop
key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
               <prop key="hibernate.show_sql">true</prop>
           </props>
       </property>
   </bean>
   <bean name="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
       <property name="entityManagerFactory"
ref="entityManagerFactoryBean"/>
   </bean>
   <bean name="employeeDao"
class="com.durgasoft.dao.EmployeeDaoImpl"/>
```

```
</beans>
```

pom.xml

```
<?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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.durgasoft</groupId>
   <artifactId>springjpaapp02</artifactId>
   <version>1.0-SNAPSHOT</version>

   <properties>
       <maven.compiler.source>17</maven.compiler.source>
       <maven.compiler.target>17</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
   </properties>
   <dependencies>
       <!--
https://mvnrepository.com/artifact/org.springframework/spring-orm -->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-orm</artifactId>
           <version>6.0.11</version>
       </dependency>
       <!--
https://mvnrepository.com/artifact/org.springframework/spring-context
-->
       <dependency>
           <groupId>org.springframework</groupId>
           <artifactId>spring-context</artifactId>
           <version>6.0.11</version>
       </dependency>

       <!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
       <dependency>
           <groupId>mysql</groupId>
           <artifactId>mysql-connector-java</artifactId>
           <version>8.0.33</version>
       </dependency>
       <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
       <dependency>
           <groupId>org.hibernate</groupId>
           <artifactId>hibernate-core</artifactId>
           <version>6.2.7.Final</version>

       </dependency>

   </dependencies>
</project>
```

Spring Boot Data JPA:
Q)To interact with the databases we already have Spring ORM module with JPA implementation then what is the requirement to use Spring Data JPA?

Ans:
----
1. To perform database operations if we use a Spring ORM module with JPA we must provide implementation for all the dao methods like save() or persist(), find(), findAll(), update(), delete(),... in the Dao implementation classes like EmployeeDao, StudentDao, ProductDao, ….. , it will increase boilerplate code.

In the above context, to reduce boilerplate code we have to use Spring Data JPA, in Spring Data JPA we will not provide implementation for all the DAO methods, Spring Data JPA will use some Repository interfaces, which has the default methods implementation with the CRUD operations, no need to provide implementation for the DAO methods.

2. In Spring JPA we must provide a lot of configurations in the configuration files like EntityManagerFactoryBean, dataSource, Dao , ….

In the case of Spring Data JPA , it is not required to provide configurations explicitly, because Spring Data JPA comes along with the feature like Autoconfiguration.

3. Spring ORM module does not have default embedded databases, every data is explicit database only.

In the case of Spring Data JPA, Spring Data JPA has some embedded databases like H2, HSQLDB, Apache Derby.

Note: Spring Data JPA is supporting some explicit databases also like Oracle, MySQL,...

4. Spring ORM modules do not have the default inbuilt support for the both SQL and NOSQL databases, everything we must configure explicitly.

Spring Data JPA has some default support for the SQL[Oracle, MySQL,...] and NOSQL databases[MONGODB,...].

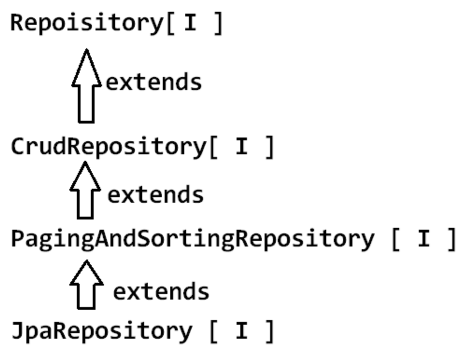
5. Spring ORM does not have predefined support for the Query methods.

Spring Data JPA is able to support the query methods as per the requirement.
6. The default connection pooling mechanism in Spring ORM is a low level DriverManagerDataSource.

Spring Data JPA has the strong default connection pooling mechanism in the form of “Hikary”.

Repository interfaces:
—-----------------------
In Spring Data JPA, Repository interfaces are able to reduce boilerplate code , they are able to provide DAO methods with default implementations that developers can reuse in order to perform database operations.

Spring Data JPA has provided the following Repository interfaces.



If we want to perform database operations with Spring Data JPA , first we have to declare an user defined interface as a child interface to JpaRepository interface, here in the user defined interface all methods of the JpaRepository interface are available that we can access in the main class.

```
@Reporitory
public interface EmployeeRepository extends JpaRepository<Employee, Integer>{

}
```

Steps to prepare Spring Data JPA Application:
----

1. Create a Database table with a primary key column having IDENTITY Capability.

```
sql>create table product(PID int(5) primary key auto_increment, PNAME
char(10), PCOST int(5));
```

2. Create a Spring Boot project with the following dependencies.
   a. Spring Data JPA
   b. Spring Web
   c. MySQL Driver

3. Provide the following configurations in the properties file.
   a. spring.datasourece.driver-class-name=com.mysql.cj.jdbc.Driver
   b. spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
   c. spring.datasource.username=root
   d. spring.datasource.password=root
   e. spring.jpa.show-sql=true
   f. spring.jpa.database-flatform=org.hibernate.dialect.MySQLDSialect

4. Prepare a bean class or model class.

```
      public class Product{
            private int pid;
            private String pname;
            private int pcost;
            setXXX()
            getXxx()
      }
```

5. Prepare a Repository interface:
   a. Declare an user defined interface.

b. Extend user defined interface from JpaRepository
c. If we want to perform select operations for an entity on the basis of a particular field then we have to declare the method with the following format, no need to provide implementation explicitly, for these methods JpaRepository will provide implementation internally.

```
public EntityName findEntityNameByFieldName(propertyName);
EX: public Product findProductByPname(String pname);
    public Product findProductByPcost(int pcost);
```

d. In the Repository interface , we can provide our own query methods as per the requirement with the help of @Query() annotation.

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Integer>{
      public Product findProductByPname(String pname);
      public Product findProductByPcost(int pcost)
}
```

6. Prepare Service interface , implementation class and access Repository methods.

a. Declare an interface with service methods.
b. Provide an implementation class for the service interface.
c. Inside the implementation class inject a Repository object by using @Autowired annotation.
d. Access the Repository methods like save(), findById(), deleteBy Id(),..

```
public interface ProductService{
      public String addProduct(Product product);
      public Product getProduct(int productId);
      public String updateProduct(Product product);
      public String deleteProduct(int productId);
}

@Service
public class ProductServiceImpl implements ProductService{
      @Autowired
      private ProductRepository productRepository;

      @Transactional
```

```
	public String addProduct(Product product){
		Product prd = productRepository.save(product);
		return “success”;
	}
	public Product getProduct(int productId){
		Product product = productRepository.findById(productId);
		return product;
	}
	@Transactional
	public String updateProduct(Product product){
		Product prd = productRepository.findById(productId);
		prd.setPname(product.getPname());
		prd.setPcost(product.getPcost());
	}

	@Transactional
	public String deleteProduct(int productId){
		productRepository.deleteById(productId);
		return “success”;
	}
}
```

7. Prepare Controller class with controller methods and access Service methods.
   a. Declare an user defined class with @Controller annotation.
   b. Provide controller methods and access Service methods.

```
@Controller
public class ProductController{
	@Autowired
	private ProductService productService;
	public String addProduct(Product product){
		String status = productService.addProduct(product);
		return status;
	}
	public Product getProduct(int productId){
		Product product = productService.getProduct(productId);
		return product;
	}
	public String updateProduct(Product product){
		String status = productService.updateProduct(product);
		return status;
	}
	public String deleteProduct(int productId){
		String status = productService.deleteProduct(productId);
		return status;
	}
}
```

```
8. Create a Runner class:
@Component
public class ProductRunner implements ApplicationRunner{
	@Autowired
	private ProductController productController;
	public void run(...){
		Product product = new Product();
		product.setPname("AAA");
		product.setPcost(100);
		String status = productController.addProduct(product);
		System.out.println(status);
		----
		----
	}
}
```

EX:
Application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

Product.java

```
package com.durgasoft.springbootapp22.bean;

import jakarta.persistence.*;

@Entity
@Table(name="PRODUCT")
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```java
    @Column(name = "PID")
    private int pid;
    @Column(name = "PNAME")
    private String pname;
    @Column(name = "PCOST")
    private int pcost;

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }

    public String getPname() {
        return pname;
    }

    public void setPname(String pname) {
        this.pname = pname;
    }

    public int getPcost() {
        return pcost;
    }

    public void setPcost(int pcost) {
        this.pcost = pcost;
    }

    @Override
    public String toString() {
        return "Product{" +
                "pid=" + pid +
                ", pname='" + pname + '\'' +
                ", pcost=" + pcost +
                '}';
    }
}
```

Springbootapp22Application.java

```java
package com.durgasoft.springbootapp22;
```

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp22Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp22Application.class, args);
   }

}
```

ProductRunner.java

```java
package com.durgasoft.springbootapp22;

import com.durgasoft.springbootapp22.bean.Product;
import com.durgasoft.springbootapp22.controller.ProductController;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class ProductRunner implements ApplicationRunner {
   @Autowired
   private ProductController productController;
   @Override
   public void run(ApplicationArguments args) throws Exception {
      /* Product product = new Product();
       product.setPname("Mobile");
       product.setPcost(25000);
       Product prd = productController.addProduct(product);
       System.out.println(prd);*/

       /*Product product = productController.getProductByPID(1);
       System.out.println(product);*/

      /* Product product =
productController.getProductByPNAME("Laptop");
       System.out.println(product);*/
       /*Product product =
productController.getProductByPCOST(20000);
       System.out.println(product);*/


       /*Product product = new Product();
       product.setPid(2);
       product.setPname("Laptop");
       product.setPcost(50000);
       Product prd = productController.update(product);
       System.out.println(prd);*/

       Product product = new Product();
       product.setPid(3);
       product.setPname("Mobile");
       product.setPcost(25000);
       int rowCount = productController.updateProduct(product);
       System.out.println(rowCount);

      /* String status = productController.delete(1);
       System.out.println(status);*/

   }
}
```

ProductRepository.java

```
package com.durgasoft.springbootapp22.repository;

import com.durgasoft.springbootapp22.bean.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

//@Repository
public interface ProductRepository extends JpaRepository<Product,
Integer> {
   public Product findProductByPname(String pname);
   public Product findProductByPcost(int pcost);

   @Modifying
   @Transactional
```

```
    @Query("update Product as p set p.pname=:pname, p.pcost = :pcost
where p.pid=:pid")
    public int updateProduct(int pid, String pname, int pcost);


}
```

ProductService.java
```
package com.durgasoft.springbootapp22.service;

import com.durgasoft.springbootapp22.bean.Product;

public interface ProductService {
    public Product addProduct(Product product);
    public Product getProductByPID(int pid);
    public Product getProductByPNAME(String pname);
    public Product getProductByPCOST(int pcost);
    public Product update(Product product);
    public int updateProduct(Product product);
    public String delete(int pid);
}
```

ProductServiceImpl.java
```
package com.durgasoft.springbootapp22.service;

import com.durgasoft.springbootapp22.bean.Product;
import com.durgasoft.springbootapp22.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class ProductServiceImpl implements ProductService{
    @Autowired
    private ProductRepository productRepository;
    //@Transactional
    @Override
    public Product addProduct(Product product) {
        Product prd = productRepository.save(product);
        return prd;
    }

    @Override
```

```java
    public Product getProductByPID(int pid) {
        Product product = productRepository.findById(pid).get();
        return product;
    }

    @Override
    public Product getProductByPNAME(String pname) {
        Product product = productRepository.findProductByPname(pname);
        return product;
    }

    @Override
    public Product getProductByPCOST(int pcost) {
        Product product = productRepository.findProductByPcost(pcost);
        return product;
    }

    @Transactional
    @Override
    public Product update(Product product) {
        Product prd =
productRepository.findById(product.getPid()).get();
        prd.setPname(product.getPname());
        prd.setPcost(product.getPcost());
        return prd;
    }

    @Override
    public int updateProduct(Product product) {
        int rowCount =
productRepository.updateProduct(product.getPid(), product.getPname(),
product.getPcost());
        return rowCount;
    }

    public String delete(int pid){
        productRepository.deleteById(pid);
        return "SUCCESS";
    }
}
```

ProductController.java

```java
package com.durgasoft.springbootapp22.controller;
```

```java
import com.durgasoft.springbootapp22.bean.Product;
import com.durgasoft.springbootapp22.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

@Controller
public class ProductController {
   @Autowired
   private ProductService productService;

   public Product addProduct(Product product){
       Product prd = productService.addProduct(product);
       return prd;
   }
   public Product getProductByPID(int pid){
       Product product = productService.getProductByPID(pid);
       return product;
   }
   public Product getProductByPNAME(String pname){
       Product product = productService.getProductByPNAME(pname);
       return product;
   }

   public Product getProductByPCOST(int pcost){
       Product product = productService.getProductByPCOST(pcost);
       return product;
   }
   public Product update(Product product){
       Product prd = productService.update(product);
       return prd;
   }
   public int updateProduct(Product product){
       int rowCount = productService.updateProduct(product);
       return rowCount;
   }
   public String delete(int pid){
       String status = productService.delete(pid);
       return status;
   }
}
```

pom.xml

```xml
<?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>3.1.2</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>springbootapp22</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp22</name>
   <description>springbootapp22</description>
   <properties>
      <java.version>17</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>

      <dependency>
         <groupId>com.mysql</groupId>
         <artifactId>mysql-connector-j</artifactId>
         <scope>runtime</scope>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
      </dependency>
   </dependencies>

   <build>
```

```
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>

</project>
```

```
server.port=1234
spring.jpa.show-sql=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
#spring.h2.console.settings.web-admin-password=sa

spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=sa
```

PagingAndSortingRepository in Spring Boot Data JPA:
—---

Its main purpose is to sort all the results and generate the results as per the pagination.

In Spring Data JPA applications, if we want to sort all the results then we have to use the “Sort” class like below.

```
Sort sort = Sort.by(Sort.Direction.fromString("ASC"), "saddr");
Iterable<Student> iterable = studentRepository.findAll(sort);
```

The “by()” method is able to define the Sorting direction like Ascending or Descending and the parameter on which we want to perform Sorting.

To get all the elements from the Database with a particular Sorting order we have to use the following method.

public Iterable findAll(Sort sort)

In Spring Data JPA if we want to read all the results page by page , that is as per Pagination then we have to use the predefined libraries “PageRequest” and “Page” like below.

```
PageRequest pageRequest = PageRequest.of(0,3);
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
```

Where PageRequest is able to manage the page number and number of results in the respective page by using the following static method.

public static PageRequest of(int pageNo, int numOfRecords)

Where Page object is able to manage the records which are retrieved as per the PageRequest object provided page number and number of records. To get a Page object we have to access the findAll() method from the PagingAndSortingRepository interface with the pageRequest parameter.

public Page findAll(PageRequest pageRequest)

If we want to get all Results from Page object to List we have to use the following code.
List<Student> stdList = page.stream().toList();
In Spring Data JDBC if we want to read all the results page by page , that is as per Pagination with a particular Sorting order then we have to use the following code.

```
PageRequest pageRequest = PageRequest.of(0,3,
Sort.Direction.fromString("DESC"),"sname");
Page<Student> page = studentRepository.findAll(pageRequest);
List<Student> stdList = page.stream().toList();
```

EX:
application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.show-sql=true
```

Student.java

```
package com.durgasoft.springbootapp23.beans;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "student")
public class Student {
   @Id
   private String sid;
   private String sname;
   private String saddr;

   public String getSid() {
       return sid;
   }

   public void setSid(String sid) {
       this.sid = sid;
   }

   public String getSname() {
       return sname;
   }

   public void setSname(String sname) {
       this.sname = sname;
   }

   public String getSaddr() {
       return saddr;
   }

   @Override
   public String toString() {
       return "Student{" +
               "sid='" + sid + '\'' +
               ", sname='" + sname + '\'' +
               ", saddr='" + saddr + '\'' +
               '}';
   }

   public void setSaddr(String saddr) {
       this.saddr = saddr;
   }
}
```

StudentRepository.java

```
package com.durgasoft.springbootapp23.repository;

import com.durgasoft.springbootapp23.beans.Student;

import
org.springframework.data.repository.PagingAndSortingRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentRepository extends
PagingAndSortingRepository<Student, String> {

}
```

StudentRunner.java

```
package com.durgasoft.springbootapp23.runner;

import com.durgasoft.springbootapp23.beans.Student;
import com.durgasoft.springbootapp23.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Component;

@Component
public class StudentRunner implements CommandLineRunner {
   @Autowired
```

```
    private StudentRepository studentRepository;
    @Override
    public void run(String... args) throws Exception {
        /*Sort sort = Sort.by(Sort.Direction.fromString("ASC"),
"saddr");
        Iterable iterable = studentRepository.findAll(sort);
        iterable.forEach(System.out::println);*/

        /*PageRequest pageRequest = PageRequest.of(1,3);
        Page<Student> page = studentRepository.findAll(pageRequest);
        page.forEach(System.out::println);*/

        PageRequest pageRequest = PageRequest.of(1,3,
Sort.Direction.fromString("DESC"), "saddr");
        Page<Student> page = studentRepository.findAll(pageRequest);
        page.forEach(System.out::println);

    }
}
```

Springbootapp23Application.java

```
package com.durgasoft.springbootapp23;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp23Application {

    public static void main(String[] args) {
        SpringApplication.run(Springbootapp23Application.class, args);
    }

}
```

pom.xml

```
<?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>
```

```xml
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.1.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>springbootapp23</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>springbootapp23</name>
    <description>springbootapp23</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

In Memory Databases In Spring Boot:
————————————————————————————————————
In any enterprise applications, if we want to use traditional databases like Oracle , MySQL,... then we have to use the following steps.

1. Install Database Software
2. Providing Database Setups
3. Create Schema
4. Create table
5. Populating data
6. Create data sources to connect from application to database.

All the above steps in our applications are difficult to test the applications , in this context, to perform test cases without having all the above configurations we will use “In Memory Databases”.

In memory Databases are able to provide the following advantages.
1. Zero project setup and infrastructure.
2. Zero configurations in our system.
3. Zero maintenance
4. Ease to use

Spring Boot Data JPA is supporting In memory databases like H2, Derby, …

If we configure In-Memory databases in Spring Boot application then that in-memory database will be created at the time of starting Spring Boot application and it will be terminated at the time of terminating spring boot application.

H2 is a Relational Database Management System written in JAVA, it can be embedded with Java applications like Spring Boot applications.

If we want to use In-Memory Databases in Spring Boot applications then we have to use the following steps.

1. Create a Spring boot application with the following dependencies.
      Spring web, h2

2. Provide h2 configurations in the properties file.

```
      spring.server.port=1234
      spring.datasource.driver-class-name=org.h2.Driver
      spring.datasource.url=jdbc:h2:mem:durgadb
      spring.datasource.username=sa
      spring.datasource.password=sa
      spring.h2.console.enabled=true
```

3. Create a Bean class:

```
      @Entity
      public class Employee{
            @Id
            @GeneratedValue
            private int eno;
            private String ename;
            ---
      }
```

4. Create Repository:

```
      public interface EmployeeRepository extends JpaRepository{
      }
```

5. Create Runner class:

```
      @Component
      public class EmployeeRunner implements CommandLineRunner{
            @Autowired
            private EmployeeRepository employeeRepository;
            public void run(String[] args){
                  -----
            }
      }
```

6. Execute the Spring Boot application and open H2 console:
To open H2 console use the following url

http://localhost:1234/h2-console

EX:
application.properties
```
server.port=1234
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.url=jdbc:h2:mem:durgadb
spring.datasource.username=durga
spring.datasource.password=durga
spring.h2.console.path=/h2-console
spring.h2.console.enabled=true
spring.jpa.show-sql=true
```

Employee.java
```
package com.durgasoft.springbootapp24.beans;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;

@Entity
public class Employee {
   @Id
   @GeneratedValue
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public Employee(int eno, String ename, float esal, String eaddr) {
       this.eno = eno;
       this.ename = ename;
       this.esal = esal;
       this.eaddr = eaddr;
```

```
    }

    public Employee() {

    }

    public int getEno() {
        return eno;
    }

    public void setEno(int eno) {
        this.eno = eno;
    }

    public String getEname() {
        return ename;
    }

    public void setEname(String ename) {
        this.ename = ename;
    }

    public float getEsal() {
        return esal;
    }

    public void setEsal(float esal) {
        this.esal = esal;
    }

    public String getEaddr() {
        return eaddr;
    }

    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "eno=" + eno +
                ", ename='" + ename + '\'' +
                ", esal=" + esal +
                ", eaddr='" + eaddr + '\'' +
                '}';
    }
 }
```

EmployeeRepository.java

```
package com.durgasoft.springbootapp24.repository;

import com.durgasoft.springbootapp24.beans.Employee;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee,
Integer> {
}
```

EmployeeRunner.java

```
package com.durgasoft.springbootapp24.runner;

import com.durgasoft.springbootapp24.beans.Employee;
import com.durgasoft.springbootapp24.repository.EmployeeRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
public class EmployeeRunner implements CommandLineRunner {
   @Autowired
   private EmployeeRepository employeeRepository;
   @Transactional
   @Override
   public void run(String... args) throws Exception {
       Employee employee = new Employee();
       employee.setEname("Nag");
       employee.setEsal(6000);
       employee.setEaddr("Chennai");
       Employee emp = employeeRepository.save(employee);
       //System.out.println(emp);

       Employee emp1 = employeeRepository.findById(1).get();
       System.out.println(emp1);
```

```
        emp1.setEname("Ramesh");
        emp1.setEsal(9999);
        emp1.setEaddr("Pune");

        Employee emp2 = employeeRepository.findById(1).get();
        System.out.println(emp2);

        employeeRepository.deleteById(1);

    }
}
```

Springbootapp24Application.java

```
package com.durgasoft.springbootapp24;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp24Application {

    public static void main(String[] args) {
        SpringApplication.run(Springbootapp24Application.class, args);
    }

}
```

pom.xml

```
<?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>3.1.2</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>springbootapp24</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

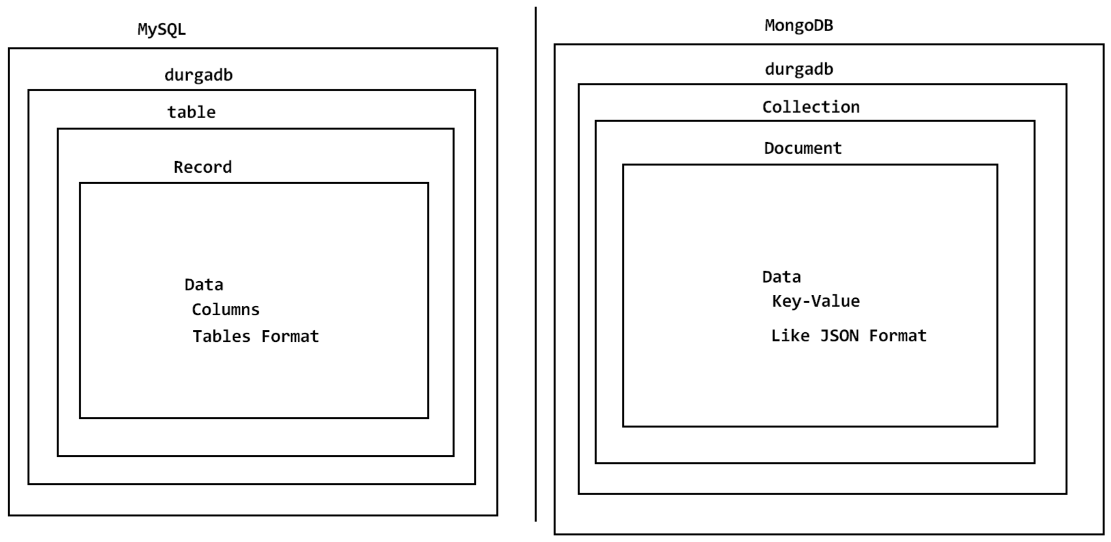
```xml
   <name>springbootapp24</name>
   <description>springbootapp24</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-web</artifactId>
       </dependency>

       <dependency>
           <groupId>com.h2database</groupId>
           <artifactId>h2</artifactId>
           <scope>runtime</scope>
       </dependency>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-data-jpa</artifactId>
       </dependency>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-test</artifactId>
           <scope>test</scope>
       </dependency>
   </dependencies>

   <build>
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
           </plugin>
       </plugins>
   </build>

</project>
```

MongoDB:

MongoDB is No SQL Database, where it is not required to write SQL queries.

MongoDB is a Document Based Database, it is very easy to manage and develop.

MongoDB contains a number of Databases, Where the Databases Contains number of Collections , Where each and every Collection contains Documents.



MongoDB is a Document Database, Where Document is referred to a record, Where Document is a data structure containing data in the form of Key-Value pairs, Where Values may be a single value or another Document or Array or Array of other documents.

Document Structure:

```
{
      Key: Value
      ---
```

```
    ----
}
```

EX:

```
{
    ENO: 111
    ENAME: AAA
    ESAL: 5000
    EADDR: Hyd
    SKILLS: [“JAVA”,”.NET”,”PYTHON”]
}
```

Installation Process:
----

1. Download MongoDB:
https://www.mongodb.com/try/download/community

2. Install MongoDB:
   a. Double Click on “mongodb-windows-x86_64-7.0.0-signed.msi” file.
   b. Click on the “Next” button.
   c. Select the Checkbox for “Accept the License Agreement”.
   d. Click on the “Next” button.
   e. Click on the “Complete” button.
   f. Click on the “Next” button.
   g. Click on the “Next” button.
   h. Click on the “Install” button.
   i. Click on the “Finish” button.

3. Open MongoDB And Perform Operations:
   1. Search for “MongoDBCompass” in System Search, Select it.
   2. Click on the “Connect” button.
   3. Open “_MONGOSH” in MongoDBCompass Which is located at the bottom.
   4. Use the following commands.
      a. To get all databases :

```
show dbs
admin    40.00 KiB
config  108.00 KiB
local    40.00 KiB
```

      b. To create and use our own DB:

```
use durgadb
switched to db durgadb
```

c. To know the current Database:

```
db
durgadb
```

d. To create Collection:

```
db.createCollection('emp1')
```

e. To insert a document in the collection:

```
db.emp1.insert({ENO:111,ENAME:'AAA',ESAL:5000,EADDR:'Hyd'})
```

Note:insert() function is deprecated in Mongo 7.0 version, alternatively we have to use insertOne() or insertMany(),...

```
db.emp1.insertOne({ENO:222,ENAME:'BBB',ESAL:6000,EADDR:'Hyd'})
{
  acknowledged: true,
  insertedId: ObjectId("64eaa371d0ebec21099a58a4")
}
```

f. To insert Multiple Documents:

```
db.emp1.insertMany([{ENO:333,ENAME:'CCC',ESAL:7000,EADDR:'Hyd'},{ENO:44
4,ENAME:'DDD',ESAL:8000,EADDR:'Hyd'},{ENO:555,ENAME:'EEE',ESAL:9000,EAD
DR:'Hyd'}])

{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64eaa456d0ebec21099a58a5"),
    '1': ObjectId("64eaa456d0ebec21099a58a6"),
    '2': ObjectId("64eaa456d0ebec21099a58a7")
  }
}
```

g. Get all Documents which are available in emp1 collection:

```
db.emp1.find()
{
  _id: ObjectId("64eaa2d8d0ebec21099a58a3"),
  ENO: 111,
  ENAME: 'AAA',
  ESAL: 5000,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa371d0ebec21099a58a4"),
  ENO: 222,
  ENAME: 'BBB',
  ESAL: 6000,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa456d0ebec21099a58a5"),
  ENO: 333,
  ENAME: 'CCC',
  ESAL: 7000,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa456d0ebec21099a58a6"),
  ENO: 444,
  ENAME: 'DDD',
  ESAL: 8000,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa456d0ebec21099a58a7"),
  ENO: 555,
  ENAME: 'EEE',
  ESAL: 9000,
  EADDR: 'Hyd'
}
```

h. To get a single document:

```
db.emp1.findOne({ENO:111})
{
  _id: ObjectId("64eaa2d8d0ebec21099a58a3"),
  ENO: 111,
  ENAME: 'AAA',
  ESAL: 5000,
  EADDR: 'Hyd'
}
```

i. To perform Updations:

```
db.emp1.updateOne({ENO:111},{$inc:{ESAL:500}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
db.emp1.findOne({ENO:111})
 {
  _id: ObjectId("64eaa2d8d0ebec21099a58a3"),
  ENO: 111,
  ENAME: 'AAA',
  ESAL: 5500,
  EADDR: 'Hyd'
}
```

j. To perform Updations on All Documents:

```
db.emp1.updateMany({ESAL:{$lt:10000}},{$inc:{ESAL:500}})

{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}

db.emp1.find()
{
  _id: ObjectId("64eaa2d8d0ebec21099a58a3"),
  ENO: 111,
  ENAME: 'AAA',
  ESAL: 6000,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa371d0ebec21099a58a4"),
  ENO: 222,
  ENAME: 'BBB',
  ESAL: 6500,
  EADDR: 'Hyd'
}
{
  _id: ObjectId("64eaa456d0ebec21099a58a5"),
  ENO: 333,
  ENAME: 'CCC',
  ESAL: 7500,
  EADDR: 'Hyd'
}
—---
—----
k. To Delete a single document from Collection:
db.emp1.deleteOne({ENO:555})
{
  acknowledged: true,
  deletedCount: 1
}

l. To delete multiple documents from the Collection:
db.emp1.deleteMany({ESAL:{$lt:10000}})
{
  acknowledged: true,
  deletedCount: 4
}
```

In Spring Boot Applications, if we want to use MongoDB then we have to use the following steps.

1. Create a Spring Boot Project with the following Dependencies:
   Spring Web
   Mongo JPA

2. Create Entity class:
   ```
   @Document
   public class Product{
         private int id;
         —----
         —----
   }
   ```

3. Create Repository interface:
   ```
   public interface ProductRepository extends MongoRepository<Product,
   Integer>{
   }
   ```

4. Create Runner class and access Repository methods:
   ```
   @Component
   public class ProductRunner implements CommandLineRunner{
         @Autowired
   ```

```
          private ProductRepository productRepository;
          public void run(String … args)throws Exception{
                Product prd = new Product();
                prd.setPno(1);
                prd.setPname("Book");
                prd.setPcost(500);
                productRepository.save(prd);
          }
    }
```

EX:
application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=durgadb
```

Product.java

```
package com.durgasoft.springbootapp25.beans;

import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

@Document("products")
public class Product {
   @Id
   private int id;
   private int pid;
   private String pname;
   private int pcost;

   public Product(int id, int pid, String pname, int pcost) {
       this.id = id;
       this.pid = pid;
       this.pname = pname;
       this.pcost = pcost;
   }

   public Product() {
   }

   public int getId() {
       return id;
```

```
    }

   public void setId(int id) {
        this.id = id;
    }



   public int getPid() {
        return pid;
    }

   public void setPid(int pid) {
        this.pid = pid;
    }

   public String getPname() {
        return pname;
    }

   public void setPname(String pname) {
        this.pname = pname;
    }

   public int getPcost() {
        return pcost;
    }

   public void setPcost(int pcost) {
        this.pcost = pcost;
    }

   @Override
   public String toString() {
        return "Product{" +
                "pid=" + pid +
                ", pname='" + pname + '\'' +
                ", pcost=" + pcost +
                '}';
    }
}
```

ProductRepository.java

```java
package com.durgasoft.springbootapp25.repository;

import com.durgasoft.springbootapp25.beans.Product;
import org.springframework.data.mongodb.repository.MongoRepository;

public interface ProductRepository extends MongoRepository<Product,
Integer> {
   public Product findProductByPname(String pname);
   public Product findProductByPcost(int pcost);
}
```

ProductRunner.java

```java
package com.durgasoft.springbootapp25;

import com.durgasoft.springbootapp25.beans.Product;
import com.durgasoft.springbootapp25.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import java.util.Arrays;
import java.util.List;

@Component
public class ProductRunner implements CommandLineRunner {

   @Autowired
   private ProductRepository productRepository;
   @Transactional
   @Override
   public void run(String... args) throws Exception {
       /*Product product = new Product();
       product.setId(1);
       product.setPid(111);
       product.setPname("Mobile");
       product.setPcost(50000);
       Product prd = productRepository.save(product);
       System.out.println(prd);*/

       /*List<Product> productList = Arrays.asList(new
Product(2,222,"Laptop",50000),
                                                 new Product(3, 333, 
"Book", 500),
                                                 new Product(4, 444, 
"Pen", 100));
       List<Product> prdList = 
productRepository.saveAll(productList);
       System.out.println(prdList);*/

       /*Product product = productRepository.findById(1).get();
       System.out.println(product);*/

       /*List<Product> productList = productRepository.findAll();
       productList.stream().forEach(System.out::println);*/

       /*Product product = 
productRepository.findProductByPname("Pen");
       System.out.println(product);*/

       /*Product product = productRepository.findProductByPcost(500);
       System.out.println(product);*/

       /*productRepository.deleteById(4);
       System.out.println("Product Deleted Successfully");*/

       productRepository.deleteAll();
       System.out.println("Deleted All Documents....");
   }
}
```

Springbootapp25Application.java

```
package com.durgasoft.springbootapp25;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp25Application {

   public static void main(String[] args) {
       SpringApplication.run(Springbootapp25Application.class, args);
   }


}
```

pom.xml

```xml
<?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>3.1.3</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>springbootapp25</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp25</name>
   <description>springbootapp25</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-data-mongodb</artifactId>
       </dependency>
       <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>
       </dependency>
   </dependencies>

   <build>
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
```

```
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Spring Boot Web:
—----------------
To prepare web applications, if we use the Spring web module then we have to use the following steps.

1. Create a Project with web Archetype in MAVEN.
2. Provide the required dependencies in pom.xml.
3. Provide the bean components
4. Provide the layers like Controller, Service and Dao.
5. Provide all the configuration files.
6. Provide the deployment descriptor with the DispatcherServlet configuration.
   —-------
   —-------

For every project performing all the above steps is very difficult

In the above approach we have to provide a lot of configurations in the Configuration file like Handler mapping , View Resolver, MVC layers related components like Service, Controller, Daos,....

In the above approach, we have to use the server explicitly , it will not provide any default integration of the new technologies and tools, everything we have to configure explicitly.

To overcome all the above problems and to simplify the web application development we have to use Spring Boot Web.

Spring Boot has provided a starter for web application, it is similar to the WEB MVC execution but it reduces work done by the programmer like

1. Creating Maven project and its setup.
2. Pom.xml and its dependencies and plugins.
3. Writing common code like Configuration classes, Configuration files,...
4. Handle Runtime Environments and creating WAR, JAR,..

All the above steps are taken care of by Spring Boot with the "Autoconfiguration".

In Spring Boot, Handler mappings are not required to be configured , which are managed by the SPring Boot internally.

In Spring Boot, View Resolvers are not required to be configured explicitly, just we have to provide prefix, suffix configurations in the properties file.

```
application.properties:
server.port=1234
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

Spring boot has three embedded servers internally they are Tomcat, Jetty, Undertow. No need to download and install servers explicitly.

Spring Boot application is a standalone application, it will start server, it will deploy the application in server and it will provide all the required configurations internally, here just we have to run spring boot application as like J2SE Application.

Steps to Prepare Spring Boot Web Application:

1. Create a Spring Boot starter project with the dependencies.
2. To work with JSP pages effectively, use the following dependency in pom.xml explicitly.

   ```
   tomcat-embdded-jasper
   ```

3. Provide properties file with the following configurations:

   ```
   server.port = 1234
   server.servlet.context-path=/empApp
   spring.mvc.views.prefix=/WEB-INF/views/
   spring.mvc.views.suffix=.jsp
   ```

4. Create Folders as per the requirement:
   a. webapp under main.
   b. WEB-INF under webapp.
   c. Views under WEB-INF

```
5. Prepare Controller components:

@Controller / @Component
@RequestMapping(“emp”)
public class EmployeeController{
      @RequestMapping(“addform”)
      public String addForm(Model model){
            model.addAttribute(“message”, “Welcome To Boot Appl”);
            return “addform”;
      }
}

http://localhost:1234/empApp/emp/addform

6. Prepare View Pages under “views” folder.
    home.jsp, addform.jsp,...

EX:
application.properties
```

```
server.port=1234
server.servlet.context-path=/empapp
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

EmployeeCOntroller.java

```
package com.durgasoft.springbootapp26.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("emp")
public class EmployeeController {
   @RequestMapping("homepage")
   public String showHomePage(Model model){
        model.addAttribute("message", "Welcome To Spring Boot WEB MVC
Application");
        return "home";
   }
}
```

home.jsp
```
<html>
<body>
<h2 style="color: red;" align="center">
   ${message}
</h2>
</body>
</html>
```

Springbootapp26Application.java
```
package com.durgasoft.springbootapp26;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp26Application {

   public static void main(String[] args) {
       SpringApplication.run(Springbootapp26Application.class, args);
   }

}
```

pom.xml
```
<?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>3.1.3</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>springbootapp26</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>springbootapp26</name>
   <description>springbootapp26</description>
```

```
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <!--
https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-emb
ed-jasper -->
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <version>10.1.11</version>
        </dependency>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

EX:
welcome.jsp

```
<html>
<body>
<h2 style="color: green" align="center">
   Welcome To Product Inventory System
</h2>
<h3 align="center">
   <a href="addform">ADD Product</a> <br><br>
   <a href="searchform">SEARCH Product</a><br><br>
   <a href="updateform">UPDATE Product</a><br><br>
   <a href="deleteform">Delete Product</a>
</h3>
</body>
</html>
```

addform.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Add Form</h3>
<form:form method="post" modelAttribute="product" action="save">
   <table align="center">
       <tr>
           <td>Product Id</td>
           <td><form:input path="pid"/></td>
       </tr>
       <tr>
           <td>Product Name</td>
           <td><form:input path="pname"/></td>
       </tr>
       <tr>
           <td>Product Cost</td>
           <td><form:input path="pcost"/></td>
       </tr>
       <tr>
           <td><input type="submit" value="ADD"/></td>
       </tr>
   </table>
</form:form>
</body>
</html>
```

searchform.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Search Form</h3>
<form:form method="post" modelAttribute="product" action="search">
   <table align="center">
       <tr>
           <td>Product Id</td>
           <td><form:input path="pid"/></td>
       </tr>
       <tr>
           <td><input type="submit" value="SEARCH"></td>
       </tr>
   </table>
</form:form>
</body>
</html>
```

updateform.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Search Form</h3>
<form:form method="post" modelAttribute="product" action="editform">
   <table align="center">
       <tr>
           <td>Product Id</td>
           <td><form:input path="pid"/></td>
       </tr>
       <tr>
           <td><input type="submit" value="EDIT FORM"></td>
       </tr>
   </table>
</form:form>
</body>
</html>
```

editform.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Edit Form</h3>
<form:form method="post" modelAttribute="product" action="update">
   <table align="center">
       <tr>
           <td>Product Id</td>
           <td><form:input path="pid" readonly="true"/></td>
       </tr>
       <tr>
           <td>Product Name</td>
           <td><form:input path="pname"/></td>
       </tr>
       <tr>
           <td>Product Cost</td>
           <td><form:input path="pcost"/></td>
       </tr>
       <tr>
           <td><input type="submit" value="UPDATE"/></td>
       </tr>
   </table>
</form:form>
</body>
</html>
```

deleteform.jsp

```
<%@taglib uri="http://www.springframework.org/tags/form"
prefix="form"%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Search Form</h3>
<form:form method="post" modelAttribute="product" action="delete">
   <table align="center">
       <tr>
           <td>Product Id</td>
           <td><form:input path="pid"/></td>
       </tr>
       <tr>
           <td><input type="submit" value="DELETE"></td>
```

```
        </tr>
    </table>
</form:form>
</body>
</html>
```

status.jsp

```
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Status Page</h3>
<h1 style="color: deeppink" align="center">
    ${status}
</h1>
<h3 align="center">
    <a href="/productapp/product/">Welcome Page</a>
</h3>
</body>
</html>
```

productdetails.jsp

```
<%@ page isELIgnored="false" %>
<%--
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
--%>
<html>
<body>
<h2 style="color: red;" align="center">Durga Software Solutions</h2>
<h3 style="color: blue;" align="center">Product Details</h3>
<table align="center" border="1">
    <tr>
        <td>Product Id</td>
        <td>${product.pid}</td>
    </tr>
    <tr>
        <td>Product Name</td>
        <td>${product.pname}</td>
    </tr>
    <tr>
        <td>Product Cost</td>
        <td>${product.pcost}</td>
    </tr>
</table>
```

```
<h3 align="center">
   <a href="/productapp/product/">Welcome Page</a>
</h3>
</body>
</html>
```

application.properties

```
server.port=1234
server.servlet.context-path=/productapp

spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

Product.java

```
package com.durgasoft.springbootapp26.model;

import jakarta.persistence.*;

@Entity
@Table(name = "PRODUCT")
public class Product {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private int pid;
   private String pname;
   private int pcost;

   public int getId() {
       return id;
   }

   public void setId(int id) {
       this.id = id;
   }

   public int getPid() {
       return pid;
   }

   public void setPid(int pid) {
       this.pid = pid;
   }

   public String getPname() {
       return pname;
   }

   public void setPname(String pname) {
       this.pname = pname;
   }

   public int getPcost() {
       return pcost;
   }

   public void setPcost(int pcost) {
       this.pcost = pcost;
   }

   @Override
   public String toString() {
       return "Product{" +
               "id=" + id +
               ", pid=" + pid +
               ", pname='" + pname + '\'' +
               ", pcost=" + pcost +
               '}';
   }
}
```

ProductController.java

```
package com.durgasoft.springbootapp26.controller;

import com.durgasoft.springbootapp26.model.Product;
import com.durgasoft.springbootapp26.service.ProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@RequestMapping("/product")
public class ProductController {
   @Autowired
   private ProductService productService;

   @RequestMapping("/")
   public String showWelcomePage(){
       return "welcome";
   }

   @RequestMapping("/addform")
   public String showAddForm(Model model){
       model.addAttribute("product", new Product());
       return "addform";
   }
   @RequestMapping("/searchform")
   public String showSearchForm(Model model){
       model.addAttribute("product", new Product());
       return "searchform";
   }
   @RequestMapping("/updateform")
   public String showUpdateForm(Model model){
       model.addAttribute("product", new Product());
       return "updateform";
   }
   @RequestMapping("/editform")
   public String showEditForm(@ModelAttribute Product product, Model
model){
       //model.addAttribute("product", new Product());
       Product product1 =
productService.searchProduct(product.getPid());
       if(product1 == null){
           model.addAttribute("status", "Product Does Not Exist");
           return "status";
       }else {
           model.addAttribute("product",product1);
```

```
            return "editform";
        }
    }
    @RequestMapping("/deleteform")
    public String showDeleteForm(Model model){
        model.addAttribute("product", new Product());
        return "deleteform";
    }
    @RequestMapping("/save")
    public String saveProduct(@ModelAttribute Product product, Model
model){
        Product prd = productService.saveProduct(product);
        if(prd.getPid() == product.getPid()){
            model.addAttribute("status","Product Inserted
Successfully");
        }else{
            model.addAttribute("status", "Product Insertion Failure");
        }
        return "status";
    }

    @RequestMapping("/search")
    public String searchProduct(@ModelAttribute Product product, Model
model){
        //System.out.println("In Conroller pid : "+product.getPid());
        Product product1 =
productService.searchProduct(product.getPid());
        model.addAttribute("product", product1);
        if(product1 == null){
            model.addAttribute("status","Product Does Not Exist");
            return "status";
        }else {
            return "productdetails";
        }
    }

    @RequestMapping("/update")
    public String updateProduct(@ModelAttribute Product product, Model
model){
        int rowCount = productService.updateProduct(product);
        if(rowCount == 1){
            model.addAttribute("status", "Product Updated
Successfully");
```

```java
        }else{
            model.addAttribute("status", "Product Update Failure");
        }
        return "status";
    }

    @RequestMapping("/delete")
    public String deleteProduct(@ModelAttribute Product product, Model
model){
        Product product1 =
productService.searchProduct(product.getPid());
        if(product1 == null){
            model.addAttribute("status", "Product Does Not Exist");
        }else {
            int rowCount = productService.deleteProduct(product);
            if (rowCount == 1) {
                model.addAttribute("status", "Product Deleted
Successfully");
            } else {
                model.addAttribute("status", "Product Deletion
Failure");
            }
        }
        return "status";
    }
}
```

ProductService.java

```java
package com.durgasoft.springbootapp26.service;

import com.durgasoft.springbootapp26.model.Product;

public interface ProductService {
    public Product saveProduct(Product product);
    public Product searchProduct(int pid);
    public int updateProduct(Product product);
    public int deleteProduct(Product product);
}
```

ProductServiceImpl.java

```java
package com.durgasoft.springbootapp26.service;

import com.durgasoft.springbootapp26.model.Product;
import com.durgasoft.springbootapp26.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class ProductServiceImpl implements ProductService{
   @Autowired
   private ProductRepository productRepository;
   @Transactional
   @Override
   public Product saveProduct(Product product) {
       Product product1 = productRepository.save(product);
       return product1;
   }

   @Override
   public Product searchProduct(int pid) {
       Product product = productRepository.findProductByPid(pid);
       return product;
   }


   @Override
   public int updateProduct(Product product) {
       int rowCount =
productRepository.updateProductByPid(product.getPid(),
product.getPname(), product.getPcost());

       return rowCount;
   }

   @Override
   public int deleteProduct(Product product) {
       int rowCount =
productRepository.deleteProductByPid(product.getPid());
       return rowCount;
   }

}
```

ProductRepository.java

```java
package com.durgasoft.springbootapp26.repository;
```

```
import com.durgasoft.springbootapp26.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public interface ProductRepository extends JpaRepository<Product,
Integer> {
   public Product findProductByPid(int pid);

   @Transactional
   @Modifying
   @Query("update Product set pname=:pname, pcost=:pcost where
pid=:pid")
   public int updateProductByPid(int pid, String pname, int pcost);
   @Transactional
   public int deleteProductByPid(int pid);
}
```

Springbootapp26Application.java

```
package com.durgasoft.springbootapp26;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp26Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp26Application.class, args);
   }

}
```

```
<?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>
```

```xml
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.1.3</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>Springbootapp26</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>Springbootapp26</name>
   <description>Springbootapp26</description>
   <properties>
      <java.version>17</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <!--
https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-emb
ed-jasper -->
      <dependency>
         <groupId>org.apache.tomcat.embed</groupId>
         <artifactId>tomcat-embed-jasper</artifactId>
         <version>10.1.13</version>
      </dependency>
      <!--
https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.
jsp.jstl -->
      <dependency>
         <groupId>org.glassfish.web</groupId>
         <artifactId>jakarta.servlet.jsp.jstl</artifactId>
         <version>3.0.1</version>
      </dependency>


      <dependency>
         <groupId>com.mysql</groupId>
```

```xml
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Thymeleaf:

Thymeleaf is a server side Java based Template engine , it will be used for both Web and non web applications.

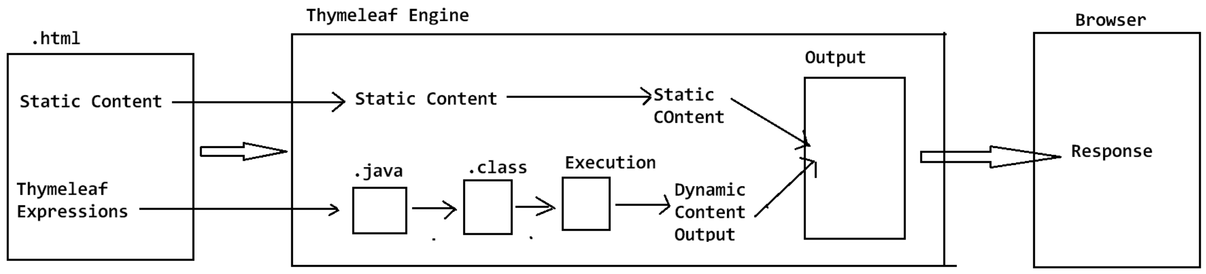
The main goal of Thymeleaf is to provide stylish and well formed web pages.

Thymeleaf is a UI Technology, it can be used to implement dynamic web pages with lightweight UI Engine.

Thymeleaf is capable of processing HTML, XML, XHTML, CSS, BOOTSTRAP, JAVA SCRIPT ,....

When compared with JSP, it will provide an environment for less code, less memory and less execution time.

Spring Boot supports working with Thymeleaf Engine in the applications, it converts the dynamic content into the equivalent java code, Compiles it, executes it, and mixes the generated result with the static content.



In Web pages, Thymeleaf is able to provide the following types of Expressions.

1. @{url} :
   It can be used to represent URLs, path values,...
   EX: <form action="#" th:action="@{/emp/save}" ....>

   It can be used to include the CSS files and Java Script files.
   EX: <link rel="stylesheet" th:ref="@{css/myfile.css}"/>
   EX: <script type="text/javascript" th:src="@{js/myfile.js}"/>

2. ${Expression} :
   It can be used to provide an expression with the bean components.
   EX: <form action="#" th:action="@{url}" th:object="${employee}">
   EX: <h3 th:text="${ename}/>

3. *{varName} :
   It can be used to link the present UI component with a particular bean property in order to set or get data.
   EX: <input type="text" th:field="*{ename}"/>
   EX: <select th:field="*{eskillSet}">

4. #{PropName} :
   It can be used to get data from the properties file on the basis of the name.
   EX: <label th:text="#{label.uname}"/>

If we want to use all the above Thymeleaf expressions in Html pages then we have to use the following namespace.

```
<html xmlns:th=”http://www.thymeleaf.org”>
      -------
</html>
```

If we want to use Thymeleaf in Spring boot applications then we have to use the following dependency in pom.xml

thymeleaf.

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

EX:
application.properties

```
server.port=1234
```

empregform.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Employee Registration Form</title>
</head>
<body>
<form method="post" action="#" th:action="@{/emp/reg}"
th:object="${employee}">
   <table align="center">
       <tr>
           <td>Employee Number</td>
           <td><input type="text" th:field="*{eno}"></td>
       </tr>
       <tr>
           <td>Employee Name</td>
           <td><input type="text" th:field="*{ename}"></td>
       </tr>
       <tr>
           <td>Employee Salary</td>
```

```
            <td><input type="text" th:field="*{esal}"></td>
        </tr>
        <tr>
            <td>Employee Address</td>
            <td><input type="text" th:field="*{eaddr}"></td>
        </tr>
        <tr>
            <td><input type="submit" value="Register"></td>
        </tr>
    </table>
</form>
</body>
</html>
```

empdata.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Employee Registration Details</title>
</head>
<body>
<table align="center">
   <tr>
       <td>Employee Number</td>
       <td th:text="${employee.eno}">
   </tr>
   <tr>
       <td>Employee Name</td>
       <td th:text="${employee.ename}">
   </tr>
   <tr>
       <td>Employee Salary</td>
       <td th:text="${employee.esal}">
   </tr>
   <tr>
       <td>Employee Address</td>
       <td th:text="${employee.eaddr}">
   </tr>
</table>
</body>
</html>
```

Employee.java

```
package com.durgasoft.springbootapp27.model;

public class Employee {
   private int eno;
   private String ename;
   private float esal;
   private String eaddr;

   public int getEno() {
       return eno;
   }

   public void setEno(int eno) {
       this.eno = eno;
   }

   public String getEname() {
       return ename;
   }

   public void setEname(String ename) {
       this.ename = ename;
   }

   public float getEsal() {
       return esal;
   }

   public void setEsal(float esal) {
       this.esal = esal;
   }

   public String getEaddr() {
       return eaddr;
   }

   public void setEaddr(String eaddr) {
       this.eaddr = eaddr;
   }
}
```

EmployeeController.java

```java
package com.durgasoft.springbootapp27.controller;

import com.durgasoft.springbootapp27.model.Employee;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
@RequestMapping("/emp")
public class EmployeeController {
   @RequestMapping(value = "/reg", method = RequestMethod.GET)
   public String showRegForm(Model model){
       model.addAttribute("employee", new Employee());
       return "empregform";
   }

   @RequestMapping(value = "/reg", method = RequestMethod.POST)
   public String register(@ModelAttribute Employee employee, Model
model){
       model.addAttribute("employee", employee);
       return "empdata";
   }
}
```

Springbootapp27Application.java

```java
package com.durgasoft.springbootapp27;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp27Application {

   public static void main(String[] args) {
      SpringApplication.run(Springbootapp27Application.class, args);
   }

}
```

pom.xml

```xml
<?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>3.1.3</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>Springbootapp27</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>Springbootapp27</name>
   <description>Springbootapp27</description>
   <properties>
      <java.version>17</java.version>
   </properties>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-thymeleaf</artifactId>
      </dependency>
      <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>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
         </plugin>
      </plugins>
   </build>

</project>
```

In Spring boot applications, by using thymeleaf template engine we are able to add CSS files , JS files … .

For CSS files:
1. Prepare CSS file under src/main/resources/static
    styles.css
2. Link the css file to html file through thymeleaf expressions:
    <link rel="stylesheet" th:href="@{/styles.css}">

For Java Script files:
1. Prepare Java Script file under src/main/resources/static
    script.js
2. Link the js file to Html through thymeleaf expressions:
    <script type="text/javascript" th:src="@{srcipt.js}"></script>

For Bootstrap Integration:
1. Get Bootstrap link into the html file from the bootstrap website
www.getbootstrap.com
2. Prepare Html components and use Bootstrap provided CSS classes.

EX:
application.properties
```
server.port=1234
spring.thymeleaf.suffix=.html
```

styles.css
```
h2{
   color: red;
   font-family: sans-serif;
   font-weight: bold;
   font-size: large;
   text-align: center;
}

h3{
   color: blue;
   font-family: sans-serif;
   font-weight: bold;
   font-size: medium;
   text-align: center;
}
table{
   margin-left: auto;
   margin-right: auto;
}
```

script.js

```
function validateForm(){
   let sid = document.forms["regForm"]["sid"].value;
   let sname = document.forms["regForm"]["sname"].value;
   let semail = document.forms["regForm"]["semail"].value;
   let smobile = document.forms["regForm"]["smobile"].value;
   let saddr = document.forms["regForm"]["saddr"].value;

   if(sid == ""){
       alert("Student Id is required.");
       return false;
   }
   if(sname == ""){
       alert("Student Name is required.");
       return false;
   }
   if(semail == ""){
       alert("Student Email Id is required.");
       return false;
   }
   if (smobile == ""){
       alert("Student Mobile Number is required.");
       return false;
   }
   if(saddr == ""){
       alert("Student Address is required.");
       return false;
   }

}
```

stdregform.html

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Student Registration Form</title>
   <link rel="stylesheet" th:href="@{/styles.css}" >
   <script type="text/javascript" th:src="@{script.js}"></script>
</head>
<body>
<h2>Durga Software Solutions</h2>
<h3>Student Registration Form</h3>
<form name="regForm" method="post" action="#" th:action="@{/reg}"
th:object="${student}" onsubmit="return validateForm()">
   <table>
       <tr>
           <td>Student Id</td>
           <td><input type="text" th:field="*{sid}"></td>
       </tr>
       <tr>
           <td>Student Name</td>
           <td><input type="text" th:field="*{sname}"></td>
       </tr>
       <tr>
           <td>Student Email Id</td>
           <td><input type="text" th:field="*{semail}"></td>
       </tr>
       <tr>
           <td>Student Mobile No</td>
           <td><input type="text" th:field="*{smobile}"></td>
       </tr>
       <tr>
           <td>Student Address</td>
           <td><input type="text" th:field="*{saddr}"></td>
       </tr>
       <tr>
           <td><input type="submit" value="Register"></td>
       </tr>
   </table>
</form>
</body>
</html>
```

stddetails.html

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Student Registration Details</title>
   <link rel="stylesheet" th:href="@{/styles.css}" >
</head>
<body>
<h2>Durga Software Solutions</h2>
<h3>Student Registration Details</h3>
<table th:border="1">
   <tr>
       <td>Student Id</td>
       <td th:text="${student.sid}"/>
   </tr>
   <tr>
       <td>Student Name</td>
       <td th:text="${student.sname}"/>
   </tr>
   <tr>
       <td>Student Email Id</td>
       <td th:text="${student.semail}"/>
   </tr>
   <tr>
       <td>Student Mobile No</td>
       <td th:text="${student.smobile}"/>
   </tr>
   <tr>
       <td>Student Address</td>
       <td th:text="${student.saddr}"/>
   </tr>
</table>
</body>
</html>
```

allstudents.html

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
   <meta charset="UTF-8">
   <title>Registered Students Details</title>
   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap
```

```
.min.css" rel="stylesheet"
integrity="sha384-4bw+/aepP/YC94hEpVNVgiZdgIC5+VKNBQNGCHeKRQN+PtmoHDE
XuppvnDJzQIu9" crossorigin="anonymous">
</head>
<body>
<h2 th:text="${message}"/>
<table class="table  table-striped">
   <thead>
   <th>ID</th>
   <th>NAME</th>
   <th>EMAIL ID</th>
   <th>MOBILE NO</th>
   <th>ADDRESS</th>
   </thead>
   <tbody>
   <tr th:each="student:${students}">
       <td th:text="${student.sid}"/>
       <td th:text="${student.sname}"/>
       <td th:text="${student.semail}"/>
       <td th:text="${student.smobile}"/>
       <td th:text="${student.saddr}"/>
   </tr>
   </tbody>
</table>
</body>
</html>
```

Student.java

```
package com.durgasoft.springbootapp28.model;

public class Student {
   private String sid;
   private String sname;
   private String semail;
   private String smobile;
   private String saddr;

   public Student(String sid, String sname, String semail, String
smobile, String saddr) {
       this.sid = sid;
       this.sname = sname;
       this.semail = semail;
```

```
        this.smobile = smobile;
        this.saddr = saddr;
    }

    public Student() {
    }

    public String getSid() {
        return sid;
    }

    public void setSid(String sid) {
        this.sid = sid;
    }

    public String getSname() {
        return sname;
    }

    public void setSname(String sname) {
        this.sname = sname;
    }

    public String getSemail() {
        return semail;
    }

    public void setSemail(String semail) {
        this.semail = semail;
    }

    public String getSmobile() {
        return smobile;
    }

    public void setSmobile(String smobile) {
        this.smobile = smobile;
    }

    public String getSaddr() {
        return saddr;
    }
```

```java
    public void setSaddr(String saddr) {
        this.saddr = saddr;
    }
}
```

StudentController.java

```java
package com.durgasoft.springbootapp28.controller;

import com.durgasoft.springbootapp28.model.Student;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import java.util.Arrays;

@Controller
public class StudentController {
    @RequestMapping(value="/reg", method = RequestMethod.GET)
    public String showStudentRegForm(Model model){
        model.addAttribute("student", new Student());
        return "stdregform";
    }

    @RequestMapping(value = "/reg", method = RequestMethod.POST)
    public String registration(@ModelAttribute Student student, Model
model){
        model.addAttribute("student", student);
        return "stddetails";
    }

    @RequestMapping("/allstds")
    public String getAllStudents(Model model){
        model.addAttribute("message","All Students Details");
        model.addAttribute("students", Arrays.asList(
                new Student("S-111", "Durga",
"durga@durgasoft.com","91-9988776655", "Hyderabad"),
                new Student("S-222", "Ramesh",
"ramesh@durgasoft.com","91-99887722", "Chennai"),
                new Student("S-333", "Rahul",
"rahul@durgasoft.com","91-9988776633", "Pune"),
                new Student("S-444", "Anil",
"anil@durgasoft.com","91-9988776611", "Mumbai"),
                new Student("S-555", "Krishna",
"krishna@durgasoft.com","91-9988776600", "Delhi")

        ));
        return "allstudents";
    }
}
```

Springbootapp28Application.java

```
package com.durgasoft.springbootapp28;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp28Application {

    public static void main(String[] args) {
        SpringApplication.run(Springbootapp28Application.class, args);
    }

}
```

pom.xml

```
<?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>3.1.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>Springbootapp28</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Springbootapp28</name>
    <description>Springbootapp28</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <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>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Spring Boot Email:

IN general, in enterprise applications, it is a frequent requirement to send emails to the customers or to the clients,....as per the applications requirements.

To send Emails Spring Boot has provided Mail API on the basis of Java Mail API.

In Spring Boot Applications, if we want to send mails by using Spring Boot mail API we have to use the following steps.

1. Create Spring Boot Project with “Java Mail Sender” dependency.

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

2. Provide Email Configurations in the properties file.

```
application.properties
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=User Name
spring.mail.password=[zbvthtnramltngvl]Generated AppPassword
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

3. Create a Service class to prepare MailMessage and to send Mail:

To prepare a Mail Message we have to use the SimpleMailMessage class.

SimpleMailMessage class has the following methods to prepare Mail Message which includes from Mail Id, To mail Id, Subject, body,...

```
public void setFrom(String fromMailId)
public void setTo(String toMailId)
public void setSubject(String subject)
public void setText(String body)
```

EX:

```
SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
simpleMailMessage.setFrom(“abc@gmail.com”);
simpleMailMesage.setTo(“xyz@gmail.com”);
simpleMailMessage.setSubject(“Sample Mail”)
simpleMailMessage.setText(“Hello , How are you”);
```

To send a MailMessage we have to use the following method from JavaMailSender.

```
public void send(SimpleMailMessage simpleMailMessage)
EX: javaMailSender.send(simpleMailMessage)
```

To generate an AppPassword we have to use the following steps.

1. Open Gmail Account
2. Click on Profile and click on “Manage Google Account”.
3. Select “Security”.
4. Search for “App Passwords” and select “App Passwords”.
5. Enter password for checking whether you are generating password or not.
6. Click on “Select App”
7. Selecty “Other[Custom Name]” and provide “SpringBootApp” or any name.
8. Click on “Generate”.
9. Copy the password and Use that password in the properties file.

Note: Your gmail account must be enabled with 2-step Authentication.

EX:
application.properties
```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=nagoornurbhash@gmail.com
spring.mail.password=dbfmgquoakosdaup
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

EmailService.java
```
package com.durgasoft.springbootapp29.service;

public interface EmailService {
   public String sendEmail(String from, String to, String subject,
String body);

}
```

EmailServiceImpl.java
```
package com.durgasoft.springbootapp29.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class EmailServiceImpl implements EmailService{
   @Autowired
```

```java
   private JavaMailSender javaMailSender;

   @Override
   public String sendEmail(String from, String to, String subject,
String body) {
       SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
       simpleMailMessage.setFrom(from);
       simpleMailMessage.setTo(to);
       simpleMailMessage.setSubject("Simple Mail Testing");
       simpleMailMessage.setText("Hello Nagoor , How are you...");
       javaMailSender.send(simpleMailMessage);
       return "Email Sent Successfully";
   }
}
```

EmailRunner.java

```java
package com.durgasoft.springbootapp29.runner;

import com.durgasoft.springbootapp29.service.EmailService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class EmailRunner implements CommandLineRunner {
   @Autowired
   private EmailService emailService;
   @Override
   public void run(String... args) throws Exception {
       String status = emailService.sendEmail(
               "nagoornurbhash@gmail.com",
               "nagoordurgasoft@gmail.com",
               "Mail Test from SpringBootApplication",
               "Hello Nagoor, This is from SpringBoot Application,
just it is test mail, dont replay");
       System.out.println(status);
   }
}
```

Springbootapp29Application.java

```java
package com.durgasoft.springbootapp29;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp29Application {

   public static void main(String[] args) {
       SpringApplication.run(Springbootapp29Application.class, args);
   }

}
```

pom.xml

```xml
<?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>3.1.3</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>Springbootapp29</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>Springbootapp29</name>
   <description>Springbootapp29</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-mail</artifactId>
       </dependency>

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

   <build>
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
           </plugin>
       </plugins>
   </build>

</project>
```

If we want to send the same mail to multiple users then we have to provide multiple users' mail Ids as an array in the setTo() method in SimpleMailMessage class.

public void setTo(String[] to)

EX:
application.properties

```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=nagoornurbhash@gmail.com
spring.mail.password=clkwlihbjtmbczyi
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

```

EmailService.java

```
package com.durgasoft.springbootapp29.service;

public interface EmailService {
   public String sendEmail(String from, String[] to, String subject,
String body);

}
```

EmailServiceImpl.java

```
package com.durgasoft.springbootapp29.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
```

```java
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;

@Service
public class EmailServiceImpl implements EmailService{
   @Autowired
   private JavaMailSender javaMailSender;

   @Override
   public String sendEmail(String from, String[] to, String subject,
String body) {
       SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
       simpleMailMessage.setFrom(from);
       simpleMailMessage.setTo(to);
       simpleMailMessage.setSubject(subject);
       simpleMailMessage.setText(body);
       javaMailSender.send(simpleMailMessage);
       return "Email Sent Successfully";
   }
}
```

EmailRunner.java

```java
package com.durgasoft.springbootapp29.runner;

import com.durgasoft.springbootapp29.service.EmailService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class EmailRunner implements CommandLineRunner {
   @Autowired
   private EmailService emailService;
   @Override
   public void run(String... args) throws Exception {
       String status = emailService.sendEmail(
               "nagoornurbhash@gmail.com",
               new
String[]{"nagoordurgasoft@gmail.com","nagoorbabuclasses@gmail.com"},
               "Mail Test from SpringBootApplication",
               "Hello Nagoor, This is test mail from Spring Boot
Application, please dont reply");
       System.out.println(status);
```

```
    }
}
```

Springbootapplication29Application.java

```
package com.durgasoft.springbootapp29;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp29Application {

    public static void main(String[] args) {
        SpringApplication.run(Springbootapp29Application.class, args);
    }

}
```

pom.xml

```
<?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>3.1.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>Springbootapp29</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Springbootapp29</name>
    <description>Springbootapp29</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-mail</artifactId>
        </dependency>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

IN Spring boot mail , if we want to send a mail with an attachment then we have to use the following steps.

1. Create MimeMessage class Object:
MimeMessage mimeMessage = javaMailSender.createMimeMessage();

2. Create MimeMessageHelper object with the MimeMessage in order to keep mail messages in MimeMessage object:
MimeMessageHelper mmh = new MimeMessageHelper(mimeMessage, true);

3. Provide mail data to the MimeMesageHelper:

```
mmh.setFrom(“from id”);
mmh.setTo(“to id”);
mmh.setSubject(“subjectMessage”);
mmh.setText(“mail body”);
```

If we want to provide an attachment then we have to use the following method.

public void addAttachment(String fileName, File file)

```
File file = new File(“E:/images/myimage.jpg”);
mmh.addAttachment(file.getName(), file);
```

4. Send mail with MimeMessage:

javaMailSender.send(mimeMessage);

EX:
application.properties
```
spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=nagoornurbhash@gmail.com
spring.mail.password=clkwlihbjtmbczyi
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true
```

EmailService.java
```
package com.durgasoft.springbootapp29.service;

import java.io.File;

public interface EmailService {
   public String sendEmail(String from, String[] to, String subject,
String body);
   public String sendEmailWithAttachment(String from, String[] to,
String subject, String body, String attachmentName, File resource);
}
```

EmailServiceImpl.java
```
package com.durgasoft.springbootapp29.service;

import jakarta.mail.MessagingException;
import jakarta.mail.internet.MimeMessage;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Service;

import java.io.File;

@Service
public class EmailServiceImpl implements EmailService{
   @Autowired
   private JavaMailSender javaMailSender;

   @Override
   public String sendEmail(String from, String[] to, String subject,
String body) {
       SimpleMailMessage simpleMailMessage = new SimpleMailMessage();
       simpleMailMessage.setFrom(from);
       simpleMailMessage.setTo(to);
       simpleMailMessage.setSubject(subject);
       simpleMailMessage.setText(body);
       javaMailSender.send(simpleMailMessage);
       return "Email Sent Successfully";
   }

   @Override
   public String sendEmailWithAttachment(String from, String[] to,
String subject, String body, String attachmentName, File resource) {
       String status = "";
       try {
           MimeMessage mimeMessage =
javaMailSender.createMimeMessage();
           MimeMessageHelper mimeMessageHelper = new
MimeMessageHelper(mimeMessage, true);
           mimeMessageHelper.setFrom(from);
           mimeMessageHelper.setTo(to);
           mimeMessageHelper.setSubject(subject);
           mimeMessageHelper.setText(body);
           mimeMessageHelper.addAttachment(attachmentName,resource);

           javaMailSender.send(mimeMessage);
           status = "Mail Sent Successfully";
       } catch (MessagingException e) {
           status = "Error Report In Mail Sent..";
           e.printStackTrace();
       }
       return status;
   }
}
```

EmailRunner.java

```
package com.durgasoft.springbootapp29.runner;

import com.durgasoft.springbootapp29.service.EmailService;
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.io.File;

@Component
public class EmailRunner implements CommandLineRunner {
   @Autowired
   private EmailService emailService;
   @Override
   public void run(String... args) throws Exception {
       /*String status = emailService.sendEmail(
               "nagoornurbhash@gmail.com",
               new
String[]{"nagoordurgasoft@gmail.com","nagoorbabuclasses@gmail.com"},
               "Mail Test from SpringBootApplication",
               "Hello Nagoor, This is test mail from Spring Boot
Application, please dont reply");
       System.out.println(status);*/

       File file = new File("E:\\images\\salman.jpg");
       String status = emailService.sendEmailWithAttachment(
               "nagoornurbhash@gmail.com",
               new
String[]{"nagoordurgasoft@gmail.com","nagoorbabuclasses@gmail.com",
"naziak7989@gmail.com","deepaksharmasdm@gmail.com"},
               "Test Mail With Attachment",
               "Hello User, This is Nagoor With an attachment",
               file.getName(),
               file
       );
       System.out.println(status);
   }
}
```

Springbootapp29Application.java

```java
package com.durgasoft.springbootapp29;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
```

```java
public class Springbootapp29Application {

   public static void main(String[] args) {
       SpringApplication.run(Springbootapp29Application.class, args);
   }

}
```

pom.xml

```xml
<?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>3.1.3</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>Springbootapp29</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>Springbootapp29</name>
   <description>Springbootapp29</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-mail</artifactId>
       </dependency>

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

   <build>
```

```xml
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
           </plugin>
       </plugins>
   </build>

</project>
```

Spring Boot Batch Processing:

The main purpose of the Spring Boot Batch processing is to send large amounts of data from Source to Destination with a particular data transformation.

In the above context, the source and destination may be a file like CSV file, XML file, ... and Databases like MySQL, MongoDB, H2 DB,....

EX: In any supermarket getting the list of items which are sold out in the last year.

EX: Getting transaction history in the last 6 months of time in a bank application.

Spring Boot Batch Processing API and Terminology:

JOB: It is a task indicating to transfer a large amount of data from Source to destination with a particular data transformation[Operation on Data].

In Spring Boot Batch Processing a single JOB contains multiple Steps[Minimum one Step].

Step: It is a sub task , it contains the following elements

1. Item Reader
2. Item Processor
3. Item Writer

Item Reader: It will read data from the Source and it will submit that data to the Item Processor.

Item Processor: It will take data from Item Reader , it will process the data and it will submit the processed data to the Item Writer.

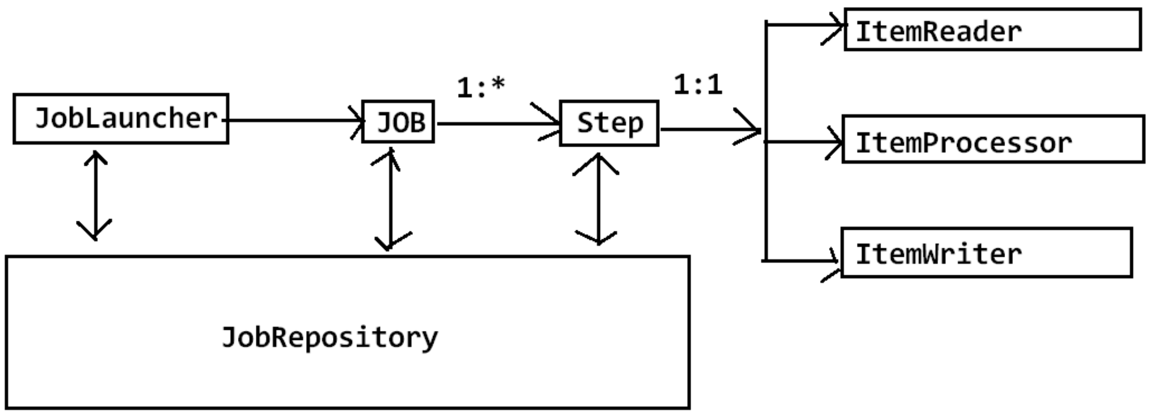
Item Writer: It will get data from the Item Processor , it will write the data into the destination on the basis of the chunk size.

Where the Chunk size is the number of items that ItemWriter will write at a time to the Destination.

In Spring Boot Batch Processing, For every time ItemWriter will work, ItemProcessor will work but ItemWriter will work on the basis of the Chunk size, that is over the 100 items ItemWriter and ItemProcessor will work on each and every item but if the Chunk size is 10 then ItemWriter will work only 10 times over the 100 items that is every time 10 items that are in 10 attempts.

JobRepository: It is a memory, it is able to store all details of the Job and Steps like their startup time, ending time,....

Job Launcher: It will launch or start a job.



The Item may be a String, an array, an Object of a class, a Collection like List, Set,....

ItemReader will read one by one item from the Source , for example a source has 10 items then ItemReader will be executed 10 times.

ItemReader generic Type must be matched with ItemProcessor Input Generic Type.

The ItemProcessor will read item by item from the ItemReader and perform a particular operation that is some calculations over the item , Converting item from one form to another form, checking conditions over the items,,...

ItemProcessor will provide the item as an output that is called Transformed Item.

ItemWriter will collect items from the ItemProcessor at a time as per the chunk size.

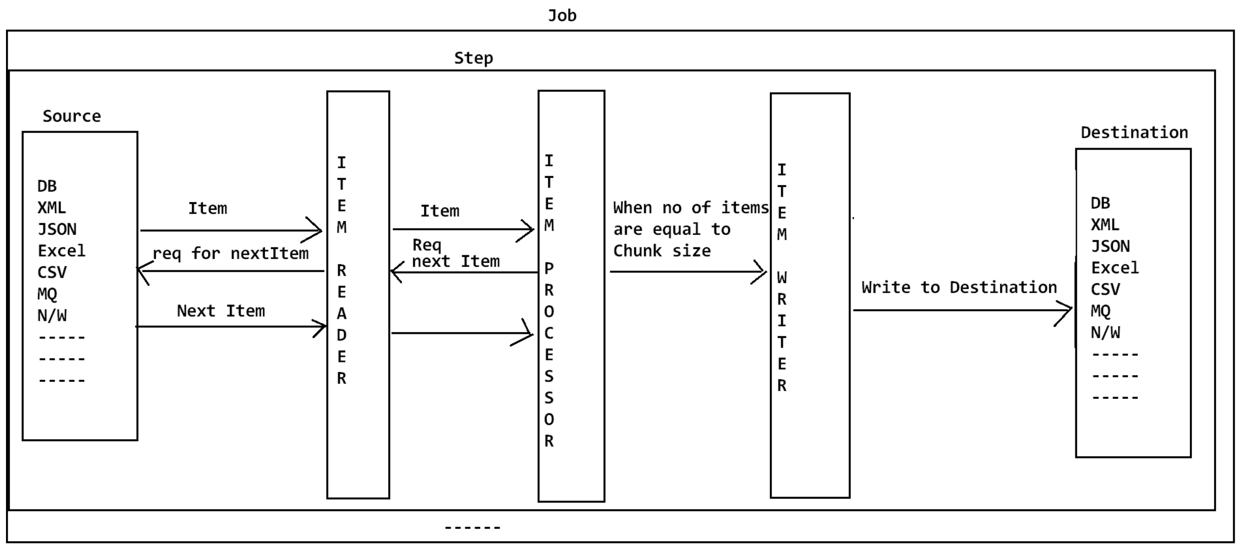
ItemWriter will write all the items at a time to the destination.

ItemWriter generic Type must be the same as the ItemProcessor output generic type.

```
ItemReader<T>
ItemProcessor<T,R>
ItemWriter<R>
```

The Source / Destination may be a text file, Database , a Network, Message Queue, Excel sheet, CSV file, JSON file, XML file,...

Step Flow of Execution:
———————————————————————



If we want to prepare Batch processing applications, we have to provide the following steps mainly.

1. Step Creation
2. Job Creation
3. Job Execution

Step Creation
—-------------
This process will create one or more steps like step1, step2,....

In Spring Boot Batch Processing Step is an interface and it is provided by StepBuilderFactory[2.x] /StepBuilder[3.x] with the following Inputs.

ItemReader<T>, ItemProcessor<T,R>, ItemWriter<R>
The above three components are provided by the Spring Boot in the form of interface, for them we have to provide implementation classes with the respective methods.

Note: All the above interfaces are functional interfaces we can provide lambda expressions for them as implementations.

Job Creation:
—-------------
Job is the collection of Steps which are executed in an order one by one.
Job must have at least one Step.

Job is created by a Factory class that is JobBuilderFactory [2.x], JobBuilder[3.x]

Note: We can provide a listener in the form of “JobExecutionListener” to the Job inorder to execute some logic before the Job and After the Job.

```
Public void beforeJob(---)
Public void afterJob(---)
```

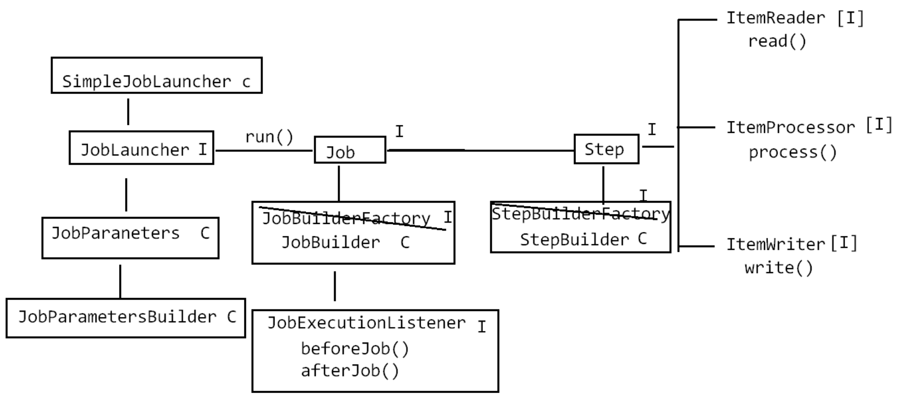
JobExecution:
—-------------
Once Steps and Job are created and configured then we have to start them by using JobLauncher.

JobLauncher is an interface, it has the following method.

```
public void run(Job job, JobParameters jobParameters)
```

Where JobParameters can be used to provide input data to the Job like Server date and time , Task Name,...

To implement Batch processing in Spring Boot applications, we have to use the following libraries.



StepBuilderFactory / StepBuilder:

It can be used to create a Step object by using chunk, reader, processor and Writer.

SpringBoot2.x:
```
stepBuilderFactory.get("step1")
.<Customer,Customer>chunk(5)
.reader(reader)
.processor(processor)
.writer(writer)
.build();
```

      Or

SpringBoot3.x:
```
new StepBuilder("StepA",jobRepository)
.<Customer,Customer>chunk(5, transactionManager)
.reader(reader)
.processor(processor)
.writer(writer)
.build();
```

JobBuilderFactory[2.x] / JobBuilder[3.x]:

It can be used to Create a Job by using steps, Listeners,....

To prepare Job we have to use the following instruction.
SpringBoot2.x version:

```
jobBuilderFactory.get("jobName")
.listener(jobExecutionListener)
.start(stepA)
.next(stepB)
.next(stepC)
.build();
```

Or

SpringBoot3.x Version:

```
new JobBuilder("jobName", jobRepository)
.listener(jobExecutionListener)
.flow(stepA).
.next(stepB)
.next(stepC)
.end()
.build();
```

JobExecutionListener:

The main purpose of the JobExecutionListener is to execute a particular logic before job execution and after job execution.

In Spring Batch applications, we have to provide a separate Listener for each and every job.

JobExecutionListener is an interface, it has the following two methods.

```
public void beforeJob(---)
public void afterJob(---)
```

Spring Boot has provided an adapter class for the JobExecutionListener in the form of JobListenerAdapter, it has the default implementation for the above two methods.

If we want to use JobListenerAdapter then we have to declare an user defined class, it must be extended from the JobAdapter class and override the required methods.

```
public class MyJobListener extends JobListenerAdapter{
    public void beforeJob(---){
      ------
    }
    public void afterJob(){
      -----
    }
}
```

In The JobExecutionListener, we are able to represent the status of the Job by using JobStatus enum, it has the following constants to represent the job status.

1. COMPLETED: JOB Completed Successfully
2. STARTING : About to start a Job that is about to call the run() method.
3. STARTED : Accessed run() method and entered inside the run() method.
4. STOPPED : Abort run() method and come out of run() method.
5. STOPING : just run() method executing is completed.
6. FAILED   : Exception in run() method.
7. ABANDONED: Job Stopped due to some external services.
8. UNKNOWN : unable to provide the status

Steps to implement Spring Boot Batch Processing Application:
----
1. Create a Spring Boot project with the following dependencies.
   Spring Web
   Data JPA
   MySQL Driver
   Spring Batch

2. Provide configuration details in the Spring Boot Properties.
application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update

spring.batch.jdbc.initial-schema=always
spring.batch.job.enabled=false
```
Where “spring.batch.jdbc.initial-schema=always” allows the Spring Boot application to create tables to represent metadata of the Spring Boot batch processing.

Where “spring.batch.job.enabled=false” is able to Specify the Spring Boot Batch Processing that does not start the batch automatically at the time of Spring Boot application startup.

3. Prepare Bean Component with the properties names of the CSV file:

```
@Entity
@Table(name=”covidcases”)
public class CovidCases{
      @Id
      @Column(name=”COUNTRY”)
      private String country;
      @Column(name=”CONFIRMED”)
      private int confirmed;
      @Column(name=”DEATHS”)
      private int deaths;
      @Column(name=”RECOVERED”)
      private int recovered;
      @Column(name=”ACTIVE”)
      private int active;
      setXXX() and getXXX()
}
```

4. Create JPARepository:
```
public interface CovidCasesRepository extends JpARepository{

}
```

5. Create ItemProcessor class:
```
public class CovidCasesDataProcessor implements Itemprocessor<CovidCases,
CovidCases>{
	public CovidCases process(CovidCases covidCases){
		String country = covidCases.getCountry();
		country=country.toUpperCase();
		covidCases.setCountry(country);
		return covidCases;

	}
}
```

5. Prepare Spring Boot Batch Processing Configurations:
   a. Declare a Java class with @Configuration
   b. Declare the properties with @Autowired annotation.
      CovidCasesRepository
      JobRepository
      PlatfileTransactionManager

   c. Prepare a COnfiguration method for the Reader:

```
	@Bean
	public FlatFileItemReader<CovidCases> itemReader(){

		FlatFileItemReader<CovidCases> itemReader = new
		FlatfileItemReader<>();

		itemReader.setResource(new
		FileSystemResource("E:/documents/covidcases.csv");

		itemReader.setLinesToSkip(1);
		itemReader.setLineMapper(lineMapper());

	}

	private LineMapper<CovidCases> lineMapper(){
		DefaultLineMapper lineMapper = new DefaultLineMapper();

		DelimitedLineTokenizer tokenizer = new DelimitedLineTokenizer();
		tokenizer.setDelimiter(",");
		tokenizer.setStrict(false);
		tokenizer.setNames("Countries","Confirmed", "Deaths", "Active",
		"Recovered");

		BeanWrapperFieldSetMapper<CovidCases> beanWrapperFieldSetMapper =
		new  BeanWrapperFieldSetMapper<>();
		beanWrapperFieldSetMapper.setTargetType(CovidCases.class);
```

```
        lineMapper.setTokenizer(tokenizer);
        lineMapper.setFieldSetMapper(beanWrapperFieldSetMapper);
        return lineMapper;
  }
```

d. Prepare ItemProcessor Configuration:

```
  @Bean
  public CovidCasesDataProcessor itemProcessor(){
    return new CovidCasesDataProcessor();
  }
```

e. Prepare ItemWriter Configuration:

```
  public RepositoryItemWriter<CovidCases> itemWriter(){
        RepositoryItemWriter itemWriter = new RepositoryItemWriter();
        itemWriter.setRepository(covidCasesRepository);
        itemWriter.setMethodName(“save”);
        return repository;
  }
```

f. Create Step Configuration:

```
  @Bean
  public Step covidCasesStep(JobRepository jobRepository,
  PlatformTransactionManager transactionManager){
    return (Step)new StepBuilder(“covidCasesStep”,jobRepository).
        <CovidCases,CovidCases>chunk(10,, transactionManager).
        reader(itemReader()).
        processor(itemProcessor()).
        writer(itemWriter()).
        build();
  }
```

g. Prepare Job Creation COnfiguration:

```
  @Bean
  public Job runJob(JobRepository jobRepository){
        return new JobBuilder(“cvs-job”,jobRepository).
              flow(covidCasesStep(jobRepository, transactionManager).
              end().
              build();
```

```
      }
```

6. Prepare Runner class:

```
@Component
public class CovidCasesRunner implements CommandLineRunner{
   @Autowired
   private JobLauncher jobLauncher;
   @Autowired
   private Job job;

   public void run(String … args)throws Exception{
      JobParameters jobParameters = new JobparametersBuilder().
      addlong("startTime",System.currentTimeMillis()).
      toJobParameters();

      jobLauncher.run(job, jobParameters);

   }
}
```

EX:
----
application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3300/durgadb
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update

spring.batch.job.enabled=false
spring.batch.jdbc.initialize-schema=always
```

CovidCases.java

```java
package com.durgasoft.springbootapp30.entities;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name="CovidCases")
public class CovidCases {
   @Id
   @Column(name = "COUNTRY")
   private String country;
   @Column(name = "CONFIRMED")
   private int confirmed;
   @Column(name = "DEATHS")
   private int deaths;
   @Column(name = "RECOVERED")
   private int recovered;
   @Column(name = "ACTIVE")
   private int active;

   public String getCountry() {
      return country;
   }

   public void setCountry(String country) {
      this.country = country;
   }

   public int getConfirmed() {
      return confirmed;
   }

   public void setConfirmed(int confirmed) {
      this.confirmed = confirmed;
   }

   public int getDeaths() {
      return deaths;
   }
   public void setDeaths(int deaths) {
       this.deaths = deaths;
   }

   public int getRecovered() {
       return recovered;
   }

   public void setRecovered(int recovered) {
       this.recovered = recovered;
   }

   public int getActive() {
       return active;
   }

   public void setActive(int active) {
       this.active = active;
   }
}
```

CovidCasesRepository.java

```java
package com.durgasoft.springbootapp30.repository;

import com.durgasoft.springbootapp30.entities.CovidCases;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CovidCasesRepository extends
JpaRepository<CovidCases, String> {
}
```

CovidCasesDataProcessor.java

```java
package com.durgasoft.springbootapp30.config;

import com.durgasoft.springbootapp30.entities.CovidCases;
import org.springframework.batch.item.ItemProcessor;

public class CovidCasesDataProcessor implements
ItemProcessor<CovidCases, CovidCases> {

   @Override
```

```
    public CovidCases process(CovidCases covidCases) throws Exception
{
        covidCases.setCountry(covidCases.getCountry().toUpperCase());
        return covidCases;
    }
}
```

SpringBatchConfig.java

```
package com.durgasoft.springbootapp30.config;

import com.durgasoft.springbootapp30.entities.CovidCases;
import com.durgasoft.springbootapp30.repository.CovidCasesRepository;
import org.springframework.batch.core.Job;
import org.springframework.batch.core.Step;
import org.springframework.batch.core.job.builder.JobBuilder;
import org.springframework.batch.core.repository.JobRepository;
import org.springframework.batch.core.step.builder.StepBuilder;
import org.springframework.batch.item.data.RepositoryItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import org.springframework.batch.item.file.LineMapper;
import
org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper
;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import
org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;
import org.springframework.transaction.PlatformTransactionManager;

@Configuration
public class SpringBatchConfig {
    @Autowired
    private CovidCasesRepository covidCasesRepository;

    @Autowired
    private JobRepository jobRepository;

    @Autowired
    private PlatformTransactionManager platformTransactionManager;
```

```
   @Bean
   public FlatFileItemReader<CovidCases> itemReader(){
       FlatFileItemReader<CovidCases> flatFileItemReader = new
FlatFileItemReader<>();
       flatFileItemReader.setResource(new
FileSystemResource("E:\\documents\\covid_cases.csv"));
       flatFileItemReader.setLinesToSkip(1);
       flatFileItemReader.setLineMapper(lineMapper());
       return flatFileItemReader;
   }

   //@Bean
   public CovidCasesDataProcessor itemProcessor(){
     return new CovidCasesDataProcessor();
   }

   public RepositoryItemWriter<CovidCases> itemWriter(){
       RepositoryItemWriter<CovidCases> repositoryItemWriter = new
RepositoryItemWriter<>();
       repositoryItemWriter.setRepository(covidCasesRepository);
       repositoryItemWriter.setMethodName("save");
       return repositoryItemWriter;
   }
   private LineMapper<CovidCases> lineMapper() {
       DefaultLineMapper<CovidCases> defaultLineMapper = new
DefaultLineMapper<>();
       DelimitedLineTokenizer delimitedLineTokenizer = new
DelimitedLineTokenizer();
       delimitedLineTokenizer.setDelimiter(",");
       delimitedLineTokenizer.setStrict(false);
       delimitedLineTokenizer.setNames("Countries","Confirmed",
"Deaths", "Recovered", "Active");
       BeanWrapperFieldSetMapper<CovidCases>
beanWrapperFieldSetMapper = new BeanWrapperFieldSetMapper<>();
       beanWrapperFieldSetMapper.setTargetType(CovidCases.class);
       defaultLineMapper.setLineTokenizer(delimitedLineTokenizer);

defaultLineMapper.setFieldSetMapper(beanWrapperFieldSetMapper);
       return defaultLineMapper;
   }

   @Bean
```

```
    public Step covidCasesStep(JobRepository jobRepository,
PlatformTransactionManager platformTransactionManager){
        return new StepBuilder("covidCasesStep", jobRepository).
                <CovidCases,
CovidCases>chunk(10,platformTransactionManager).
                reader(itemReader()).
                processor(itemProcessor()).
                writer(itemWriter()).
                build();
    }

    @Bean
    public Job runJob(JobRepository jobRepository){
        return new JobBuilder("csv-job", jobRepository).
                flow(covidCasesStep(jobRepository,
platformTransactionManager)).
                end().build();
    }
}
```

CovidCasesRunner.java

```
package com.durgasoft.springbootapp30;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class CovidCasesRunner implements CommandLineRunner {

    @Autowired
    private JobLauncher jobLauncher;

    @Autowired
    private Job job;
    @Override
    public void run(String... args) throws Exception {
        JobParameters jobParameters = new JobParametersBuilder().
                addLong("startAt", System.currentTimeMillis()).
                toJobParameters();
        jobLauncher.run(job, jobParameters);
        System.out.println("CSV File Data Stored in MySQL Database");
    }
}
```

Springbootapp30Application.java

```
package com.durgasoft.springbootapp30;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Springbootapp30Application {

    public static void main(String[] args) {
        SpringApplication.run(Springbootapp30Application.class, args);
    }

}
```

pom.xml

```
<?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>3.1.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>Springbootapp30</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Springbootapp30</name>
    <description>Springbootapp30</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
```

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

        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.batch</groupId>
            <artifactId>spring-batch-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Spring Boot Messaging or Message Queues:
—---
In general, in enterprise applications , it is convention to exchange the messages from one application to another application and from one device to another device as per the application requirements.

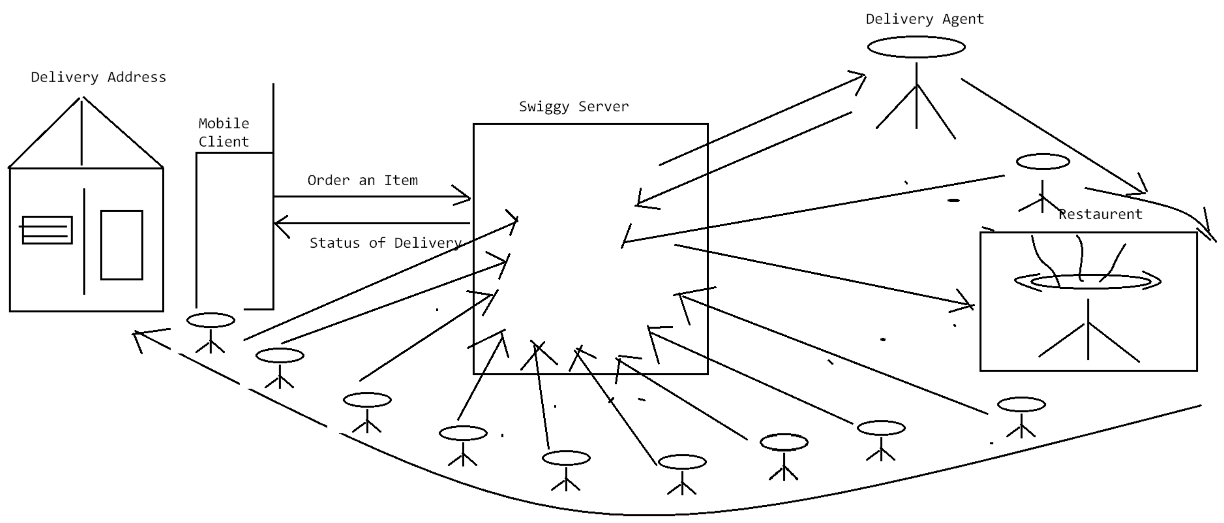
EX:
Swiggy Delivery Agent Tracking.
BUS Live Tracking
Train Live Tracking
Weather Updations
Stock Exchange Updations
Cricket Score board Updations
—--
—---

Swiggy Delivery Agent Tracking:
When we order an item in SWIGGY, SWIGGY Server will perform the following actions.

1. SWIGGY Server will receive the order details from the customer mobile application.
2. SWIGGY Server will search for the nearby Delivery Agent from the restaurant.
3. SWIGGY Server will assign a Delivery agent for the Order delivery.
4. When the Order is assigned to the Delivery Agent, Delivery Agent tracing will be started.
5. Every minute, SWIGGY Server will trace the order Delivery status from the Delivery agent and Every minute the Order Delivery status from the Delivery agent will be updated in the Mobile Client as a message continuously.

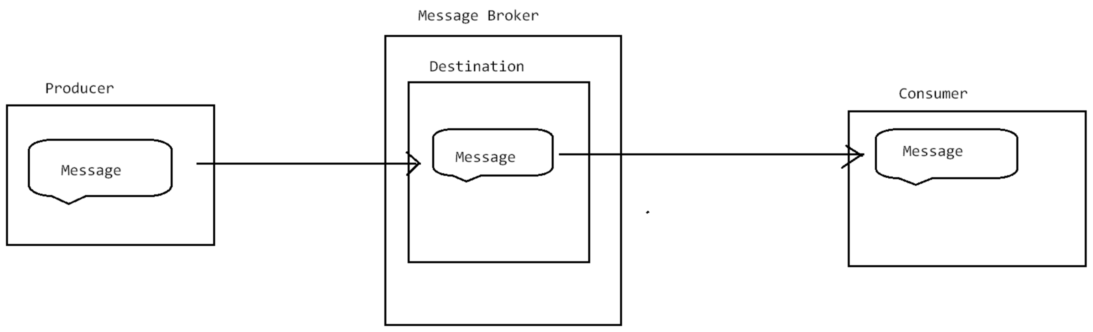
In the above use case, Messages are transferred from Delivery agent to the SWIGGY Server and SWIGGY Server to Mobile Client Application continuously.

In the above context, to transfer the messages from one application to another application and from one device to another device we have to use “Messaging System” or “Message Queue”.



Message Queue Architecture:
There are three main components in the Message Queue architecture.

1. Producer
2. Message Broker
3. Consumer



Where the Producer will create the messages and send those messages to the Message Broker.

Where Message Broker contains a buffer called destination to hold the messages and to send messages to the Consumer.

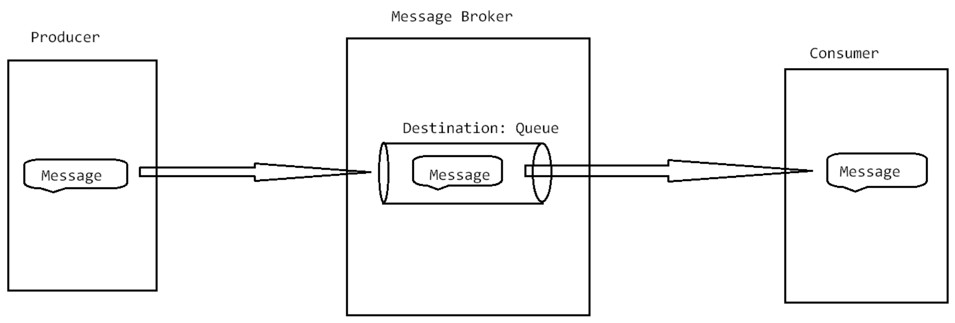
Where the consumer receives messages from the Message Broker and consumes the messages as per the requirement.

In the above Message Queue, there are two types of Communications.

1. Point-To-Point Communication or Peer-To-Peer communication.
2. Publisher-Subscriber communication or pub/sub communication.

Point-To-Point Communication:
In this communication, the Producer will send a message to the Message Broker, where the message broker will send a message to a single consumer.

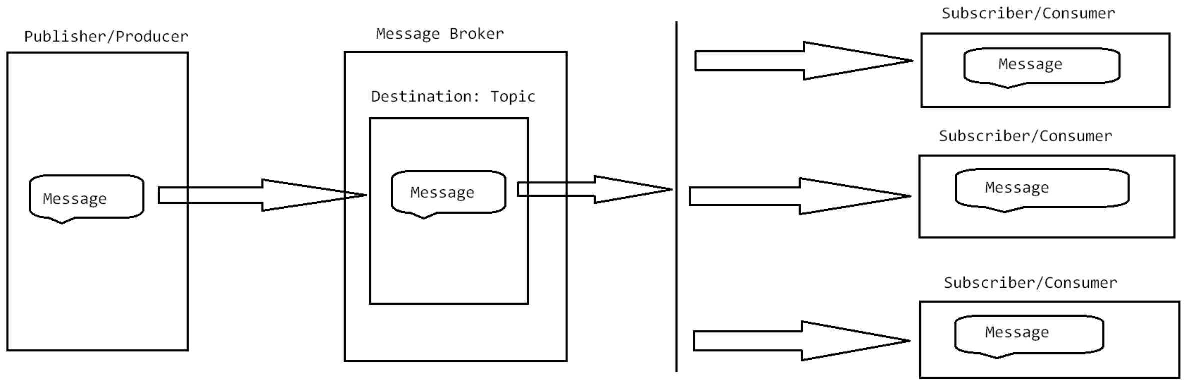
In Point-To-Point communication, the Message broker will use a Queue as Destination.



EX:
Swiggy Delivery Agent Tracking.
Mail Services

Publisher-Subscriber communication
In this communication, the Producer or Publisher will create a message and it will send that message to the Message Broker, Where the Message Broker will create the number of copies for the message on the basis of the number subscribers and Message Broker will send those copies to the subscribers.

In Pub/Sub communication, the message broker will use Topic as destination order to hold the message and in order to send messages to the consumers or Subscribers.



EX:
BUS Live Tracking
Train Live Tracking
Weather Updations
Stock Exchange Updations
Cricket Score board Updations

In Spring Boot:
1. Spring Boot Logging
2. Spring Boot Security
3. Spring Boot Rest

Microservices
Web Services

IN general, there are two approaches to implementing Message queues.

1. Basic Message Queue Protocol [BMQP]:
   EX: Apache Active MQ

2. Advanced Message Queue Protocol [AMQP]:
   EX: Robbit MQ, Apache Kafka

Apache Active MQ

Apache Active MQ is the most popular open source, multi protocol and java based message broker.

To use Activemq message broker in the Spring Boot applications we have to use the following steps.

1. Download and Install Activemq Software.
2. Prepare Producer Application.
3. Prepare Consumer application.
4. Start Activemq , open Activemq admin page.
5. Execute both producer and Consumer applications.
6. Check for the messages in the activemq admin page.

Download and Install Activemq Software
----
1. Open browser the following link.
   https://activemq.apache.org/activemq-6000000-release
2. Select “apache-activemq-6.0.0-bin.zip” file under the Windows operating system specification.
3. Copy the downloaded “apache-activemq-6.0.0-bin.zip” file to C driver , extract it and find the home directory for the activemq “apache-activemq-6.0.0”.
4. Start activemq by executing the activemq.bat available at the location “C:\apache-activemq-6.0.0\bin\win64”.
5. Open Activemq Web console by using the following url.
   http://localhost:8161/
6. Select “Manage ActiveMQ broker” link.
7. Provide the following credentials to open the admin console.
   User Name : admin [Default userName]
   Password : admin [Default Password]

   Note: We can change the User credentials by modifying the data in the users.properties file available at the location “C:\apache-activemq-6.0.0\conf\users.properties”.

   We must provide the user name and password in the following format.
   userName = password.

8. Check the queues and topics in the admin page by providing the user credentials.

Prepare Producer Application:
——————————————————————————————
The main purpose of the Producer application is to generate a message and to send that message to the Message Broker.

To prepare a Producer application we have to use the following steps.

1. Create a Spring boot project with the activemq dependency.
2. Provide activemq configuration details in the Spring boot configuration file that in application.properties file.

   ```
   application.properties
   spring.activemq.broker-url=tcp://localhost:61616
   spring.activemq.user=admin
   spring.activemq.password=nagoor
   spring.jms.pub-sub-domain=false [By default false][to disable pub/sub]
   queuename=MyQueue [User defined name to the queue].
   ```

3. Prepare a Runner class:
   a. Declare a runner class by implementing CommandLineRunner and by using @Component annotation.
   b. Declare JmsTemplate property with @Autowire annotation.
   c. Create a MessageCreator object by using Session functional interface and its lambda expression.

      ```
      MessageCreator mc = session -> session.createMessage("Hello..");
      ```

   d. Send a message to the Message Broker.

      ```
      jmsTemplate.send(destinationName, messageCreator);
      ```

Prepare Consumer application:
—————————————————————————————
The main purpose of the consumer application is to get a message from the Message Broker and utilize that message for the application logic.

Steps.
1. Create a Spring Boot project with the "activemq" dependency.
2. Provide activemq configuration details in the Spring boot configuration file in application.properties file.

```
application.properties
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
queuename=MyQueue [User defined name to the queue].
```

3. Prepare a Reader class.
   a. Declare an user defined class with the @Component annotation.
   b. Declare a method with the String parameter and with the @JmsListener

```
@Component
public class MessageReader{
      @JmsListener(destination=”${queuename}”)
      public void readMessage(String message){
            System.out.println(message);
      }
}
```

EX:
Producer Application:
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=false
queuename=MyQueue
```

ActivemqProducerRunner.java

```
package com.durgasoft.activemqproducer.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

@Component
public class ActivemqProducerRunner implements CommandLineRunner {
   @Autowired
```

```
    private JmsTemplate jmsTemplate;
    @Value("${queuename}")
    private String destinationName;
    @Override
    public void run(String... args) throws Exception {
        MessageCreator messageCreator = session ->
session.createTextMessage("From Producer : "+ LocalDateTime.now());
        jmsTemplate.send(destinationName, messageCreator);
    }
}
```

ActivemqProducerApplication.java

```
package com.durgasoft.activemqproducer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ActivemqproducerApplication {

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

}
```

pom.xml

```
<?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>3.2.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>activemqproducer</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```xml
    <name>activemqproducer</name>
    <description>activemqproducer</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-activemq</artifactId>
        </dependency>

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Consumer Application:
application.properties

```properties
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=false
queuename=MyQueue
```

MessageReader.java

```java
package com.durgasoft.activemqconsumer.reader;
```

```java
import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
   @JmsListener(destination = "${queuename}")
   public void readMessage(String message){
       System.out.println(message);
   }
}
```

Activemqconsumerappliocation.java

```java
package com.durgasoft.activemqconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ActivemqconsumerApplication {

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

}
```

If we want to provide pub-sub communications then we have to use the following steps in the Spring boot applications.

1. Prepare Producer Application.
2. Prepare multiple Consumer applications.

In the Producer applications and consumer applications, enable pub-sub domain by using the following property in the configuration file.

```
spring.jms.pub-sub-domain=true
```

Execution Process:

1. Start activemq and open web console

2. Execute all the consumer applications.
3. Execute Producer application
4. Check at the Consumer Applications , we will get messages.

EX:
Producer Application
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

ActivemqRunner.java

```
package com.durgasoft.activemqproducer.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

@Component
public class ActivemqProducerRunner implements CommandLineRunner {
   @Autowired
   private JmsTemplate jmsTemplate;
   @Value("${topicname}")
   private String destinationName;
   @Override
   public void run(String... args) throws Exception {
       MessageCreator messageCreator = session ->
session.createTextMessage("From Producer : "+ LocalDateTime.now());
       jmsTemplate.send(destinationName, messageCreator);
   }
}
```

activemqproducerApplication.java

```
package com.durgasoft.activemqproducer;
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;


@SpringBootApplication
public class ActivemqproducerApplication {

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

}
```

pom.xml

```
<?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>3.2.0</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>activemqproducer</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>activemqproducer</name>
   <description>activemqproducer</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-activemq</artifactId>
       </dependency>
```

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Consumer Application:
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

MessageReader.java

```java
package com.durgasoft.activemqconsumer.reader;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
    @JmsListener(destination = "${topicname}")
    public void readMessage(String message){
        System.out.println("Consumer : "+message);
    }
}
```

activemqconsumerApplication.java

```java
package com.durgasoft.activemqconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqconsumerApplication {

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

}
```

pom.xml

```xml
<?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>3.2.0</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>activemqconsumer</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>activemqconsumer</name>
   <description>activemqconsumer</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-activemq</artifactId>
       </dependency>
```

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Activemqmobileconsumeapplication
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

MessageReader.java

```java
package com.durgasoft.activemqmobileconsumer.reader;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
    @JmsListener(destination = "${topicname}")
    public void readMessage(String message){
        System.out.println("MobileConsumer : "+message);
    }
}
```

ActivemqMobileCOnsumerApplication.java

```
package com.durgasoft.activemqmobileconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqmobileconsumerApplication {

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

}
```

ActivemqWebConsumerApplication:
Application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

MessageReader.java

```
package com.durgasoft.activemqwebconsumer.reader;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
   @JmsListener(destination = "${topicname}")
   public void readMessage(String message){
       System.out.println("WebConsumer : "+message);
   }
}
```

ActivemqMobileConsumerApplication.java

```
package com.durgasoft.activemqwebconsumer;
```

```java
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqwebconsumerApplication {

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

}
```

If we want to provide scheduled messages from producer to the Consumer then we have to use the following annotations in the Producer application.

1. @Scheduled(fixedDelayed=timeValue)
2. @EnableScheduling

@Scheduled annotation must be provided just before the method which is submitting messages to the COnsumers.

@EnableScheduling annotation will be used for the Producer Starter class.

EX:
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

ActivemqProducerRunner.java

```java
package com.durgasoft.activemqproducer.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.CommandLineRunner;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.jms.core.MessageCreator;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

@Component
public class ActivemqProducerRunner{// implements CommandLineRunner {
   @Autowired
   private JmsTemplate jmsTemplate;
   @Value("${topicname}")
   private String destinationName;
   //@Override
   //public void run(String... args) throws Exception {
   @Scheduled(fixedDelay = 500)
   public void sendMessage()throws Exception{
       MessageCreator messageCreator = session ->
session.createTextMessage("From Producer : "+ LocalDateTime.now());
       jmsTemplate.send(destinationName, messageCreator);
   }
}
```

ActivemqproducerApplication.java

```java
package com.durgasoft.activemqproducer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;
import org.springframework.scheduling.annotation.EnableScheduling;

@EnableScheduling
@SpringBootApplication
public class ActivemqproducerApplication {

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


}
```

pom.xml

```xml
<?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>3.2.0</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>activemqproducer</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>activemqproducer</name>
   <description>activemqproducer</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
           <groupId>org.springframework.boot</groupId>
           <artifactId>spring-boot-starter-activemq</artifactId>
       </dependency>

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

   <build>
       <plugins>
           <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
           </plugin>
       </plugins>
   </build>
```

```
</project>
```

activemqconsumerApplication.java

```java
package com.durgasoft.activemqconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqconsumerApplication {

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

}
```

pom.xml

```xml
<?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>3.2.0</version>
       <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.durgasoft</groupId>
   <artifactId>activemqconsumer</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>activemqconsumer</name>
   <description>activemqconsumer</description>
   <properties>
       <java.version>17</java.version>
   </properties>
   <dependencies>
       <dependency>
```

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

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

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Activemqmobileconsumeapplication
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

MessageReader.java

```
package com.durgasoft.activemqmobileconsumer.reader;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
   @JmsListener(destination = "${topicname}")
   public void readMessage(String message){
       System.out.println("MobileConsumer : "+message);
```

```
    }
}
```

ActivemqMobileCOnsumerApplication.java

```
package com.durgasoft.activemqmobileconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqmobileconsumerApplication {

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

}
```

ActivemqWebConsumerApplication:
Application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=nagoor
spring.jms.pub-sub-domain=true
topicname=Mytopic
```

MessageReader.java

```
package com.durgasoft.activemqwebconsumer.reader;

import org.springframework.jms.annotation.JmsListener;
import org.springframework.stereotype.Component;

@Component
public class MessageReader {
   @JmsListener(destination = "${topicname}")
   public void readMessage(String message){
       System.out.println("WebConsumer : "+message);
   }
}
```

ActivemqMobileConsumerApplication.java

```java
package com.durgasoft.activemqwebconsumer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jms.annotation.EnableJms;

//@EnableJms
@SpringBootApplication
public class ActivemqwebconsumerApplication {

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

}
```

RobbitMQ:

Next class will be on Tuesday