DURGA SOFTWARE SOLUTIONS®

spring

BY NAGOOR BABU

SPRING BOOT
BY
NAGOOR BABU

CONTACT US:

Mobile: +91- 8885 25 26 27

Mail ID: durgasoftonlinetraining@gmail.com

+91- 7207 21 24 27/28

WEBSITE: www.durgasoftonline.com

US NUM: 4433326786

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

SPRING BOOT:

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

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

--> Spring Boot provides production-ready features such as metrics, health checks and externalized configuration.

--> It simplifies the bootstrapping and development of Spring Applications.

--> Spring Boot was designed on the top of existed Spring Framework to simplify and speedup the spring based applications.

--> Spring Boot can be used to develop Spring based applications either by using JAVA or by using Groovy.

--> Spring Boot Simplifies application development by having all configurations as defaults.

--> Spring Boot reduces application development time.

--> Spring Boot will improve Productivity.

--> Spring Boot will provide very simple approach to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

--> Spring Boot follows “Opinionated Defaults Configuration” approach to reduce Developer effort, that is,Spring Boot avoids writing lots of boilerplate Code,Annotations and XML Configuration,...

--> Spring Boot provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

--> Spring Boot provides CLI (Command Line Interface) tool to develop and test Spring Boot Applications from command prompt very easily and quickly.

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

--> It provides lots of plugins to work with embedded and in-memory Databases very easily.

--> Where In-Memory databases are the databases, they will be created when application startup and it will be destroyed when application terminated.

EX: H2 Database.

--> In simple words, Spring Boot is the Existed Spring Framework with Embedded Web Servers like Tomcat and Jetty with No XMl configurations and less Annotations.

SPRING BOOT FEATURES

To describe Spring Boot nature, Spring Boot has provided the following features.

1.SpringApplication

2.Externalized Configuration

3. Profiles

4. Logging

5. Internationalization

6. JSON

7. Developing Web Applications:

8. RSocket:

9.Security:

10. Working with SQL Databases:

11. Working with NoSQL Technologies:

12. Cache:

13.Messaging:

14. Calling REST Services with RestTemplate and WebClient:

15. Validations:

16. Sending Email:

17.Distributed Transactions with JTA:

18.Application events and listeners:

19.Admin features:

20.Properties Files

21.YAML Support:

22.Type-safe Configuration:

1.SpringApplication:

SpringApplication is a class, it provides a convenient way to bootstrap a Spring application that is started from a main() method. SpringApplication has a static method run() to execute Spring application.

EX:

```
class MyApp{

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

  }

}
```

When we run the above application then we will get the following output .

```
     .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::   v2.2.0.BUILD-SNAPSHOT


019-04-31 13:09:54.117  INFO 56603 --- [           main] o.s.b.s.app.SampleApplication            :
Starting SampleApplication v0.1.0 on mycomputer with PID 56603 (/apps/myapp.jar
started by pwebb)

2019-04-31 13:09:54.166  INFO 56603 --- [           main]
ationConfigServletWebServerApplicationContext : Refreshing
org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicat
ionContext@6e5a8246: startup date [Wed Jul 31 00:08:16 PDT 2013]; root of context
hierarchy

2019-04-01 13:09:56.912  INFO 41370 --- [           main] .t.TomcatServletWebServerFactory :
Server initialized with port: 8080

2019-04-01 13:09:57.501  INFO 41370 --- [           main] o.s.b.s.app.SampleApplication
: Started SampleApplication in 2.992 seconds (JVM running for 3.658)
```

If your application fails to start, registered FailureAnalyzers get a chance to provide a dedicated error message and a concrete action to fix the problem. For instance, if you start a web application on port 8080 and that port is already in use, you should see something similar to the following message:

```
***************************
APPLICATION FAILED TO START
***************************
Description:Embedded servlet container failed to start. Port 8080 was already in use.
Action:Identify and stop the process that's listening on port 8080 or configure this
application to listen on another port.
```

2.Externalized Configuration:

Spring Boot is giving an option to use like to externalize our application configurations so that we can work with the same application code in different environments. To provide Externalizable configurations we can use properties files, YAML files, environment variables, and command-line arguments,..

In Spring Boot appliocations, we are able to provide externalized configurations like

1.ServletConfig init parameters.

2.ServletContext init parameters.

3.JNDI attributes from java:comp/env.

4.Java System properties (System.getProperties()).

5.OS environment variables.

6.Profile-specific application properties outside of your packaged jar (application-{profile}.properties and YAML variants).

3. PROFILES:

Spring Profiles provide a way to segregate parts of your application configuration and make it be available only in certain environments.

IN SPring Boot applications, we are able to provide profiles configuration either through @Profile() annotation or through properties file or through Command Line Arguments.

EX: Through Annotation

```
@Configuration(proxyBeanMethods = false)

@Profile("production")

public class ProductionConfiguration {
```

```
  // ...

}
```

EX: Through Properties file

```
spring.profiles.active=dev,hsqldb
```

EX: Through Command Line Arguments.

```
--spring.profiles.active=dev,hsqldb.
```

4. Logging

Spring Boot uses Commons Logging for all internal logging but leaves the underlying log implementation open. Default configurations are provided for Java Util Logging, Log4J2, and Logback. In each case, loggers are pre-configured to use console output with optional file output also available.

5. Internationalization:

Spring Boot supports localized messages so that your application can provide services to users of different languages . By default, Spring Boot looks for the presence of a messages resource bundle at the root of the classpath.

To provide Internationalization support we have to configure messages properties file base name in application configuration file like below.

```
spring.messages.basename=messages,config.i18n.messages

spring.messages.fallback-to-system-locale=false
```

6. JSON

Spring Boot provides integration with three JSON mapping libraries

1. Gson
2. Jackson
3. JSON-B

Spring Boot is using "Jackson" as the preferred and default library.

7. Developing Web Applications:

Spring Boot is well suited for web application development.We can create a self-contained HTTP server by using embedded Tomcat, Jetty, Undertow, or Netty.

Most web applications use the spring-boot-starter-web module to get up and running quickly.

8. RSocket:

RSocket is a binary protocol for use on byte stream transports. It enables symmetric interaction models via async message passing over a single connection.

The spring-messaging module of the Spring Framework provides support for RSocket requesters and responders, both on the client and on the server side.

9.Security:

If Spring Security is on the classpath, then web applications are secured by default. Spring Boot relies on Spring Security’s content-negotiation strategy to determine whether to use

httpBasic or formLogin. To add method-level security to a web application, you can also add @EnableGlobalMethodSecurity with your desired settings

10. Working with SQL Databases:

The Spring Framework provides extensive support for working with SQL databases, from direct JDBC access using JdbcTemplate to complete "object relational mapping" technologies such as Hibernate. Spring Data provides an additional level of functionality like creating Repository implementations directly from interfaces and using conventions to generate queries from your method names.

11. Working with NoSQL Technologies:

Spring Data provides additional projects that help us access a variety of NoSQL technologies, including:

1.MongoDB

2.Neo4J

3.Elasticsearch

4.Cassandra

5.Couchbase

6.LDAP

12. Cache:

--> The Spring Framework provides support for transparently adding caching to an application. At its core, the abstraction applies caching to methods, thus reducing the number of executions based on the information available in the cache.

--> The caching logic is applied transparently, without any interference to the invoker. Spring Boot auto-configures the cache infrastructure as long as caching support is enabled via the @EnableCaching annotation.

13.Messaging:

--> The Spring Framework provides extensive support for integrating with messaging systems, from simplified use of the JMS API using   JmsTemplate to a complete infrastructure to receive messages asynchronously.

--> Spring AMQP provides a similar feature set for the Advanced Message Queuing Protocol.

--> Spring Boot also provides auto-configuration options for RabbitTemplate and RabbitMQ.

--> Spring WebSocket natively includes support for STOMP messaging, and Spring Boot has support for that through starters and a small   amount of auto-configuration. Spring Boot also has support for Apache Kafka.

14. Calling REST Services with RestTemplate and WebClient:

--

Spring Framework has provided RestTemplate class to access remote Rest Services from our machine, but, Spring Boot has provided RestSupportBuilder to create RestTemplate instance as part of auto configurations.

If we want to use Web Client to call remote Rest Services then we have to use WebFlux module.

15. Validations:

Spring Boot is supporting Bean validations like Hibernate Bean Validations inorder to validate data in bean objects.

16. Sending Email:

The Spring Framework provides an easy abstraction for sending email by using the JavaMailSender interface, and Spring Boot provides auto-configuration for it as well as a starter module.

17.Distributed Transactions with JTA:

Spring Boot supports distributed JTA transactions across multiple XA resources by using either an Atomikos or Bitronix embedded transaction manager. JTA transactions are also supported when deploying to a suitable Java EE Application Server.

When a JTA environment is detected, Spring's JtaTransactionManager is used to manage transactions. Auto-configured JMS, DataSource, and JPA beans are upgraded to support XA

transactions. You can use standard Spring idioms, such as @Transactional, to participate in a distributed transaction. If you are within a JTA environment and still want to use local transactions, you can set the spring.jta.enabled property to false to disable the JTA auto-configuration.

18.Application events and listeners:

--

Spring Boot uses events to handle variety of tasks. It allows us to create factories file that are used to add listeners. we can refer it by using ApplicationListener key.

Always create factories file in META-INF folder like: META-INF/spring.factories

19.Admin features:

Spring Boot provides the facility to enable admin related features for the application. It is used to access and manage application remotely. We can enable it by simply using spring.application.admin.enabled property.

20.Properties Files:

Spring Boot provides rich set of Application Properties. So, we can use that in properties file of our project. Properties file is used to set properties like: server-port = 8082 and many others. It helps to organize application properties.

21.YAML Support:

It provides convenient way for specifying hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically support YAML. It is successful alternative of properties.

22.Type-safe Configuration:

Strong type-safe configuration is provided to govern and validate the configuration of application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

Key Components of Spring Boot Framework:

Spring Boot Framework has mainly four major Components.

1.Spring Boot Starters

2.Spring Boot AutoConfigurator

3.Spring Boot CLI

4.Spring Boot Actuator

1.Spring Boot Starters:

--> The main intention of Spring Boot Starter is to combine a group of common or related dependencies into single dependencies.

--> If we want to prepare any web application with tomcat server and with Database interaction then we have to add the following JArs minimum to the project library.

Spring beans jat file(sprinmg-beans-xx.jar)

Spring core Jar file(spring-core-xx.jar)

Spring context Jar file(spring-context-xx.jar)

Spring Web Jar file(spring-web-xx.jar)

Spring Web MVC Jar file(spring-webmvc-xx.jar)

Servlet Jar file(servlet-xx.jar)

Spring JDBC Jar file(spring-jdbc-xx.jar)

Spring ORM Jar file(spring-orm-xx.jar)

Spring Transaction Jar file(spring-transaction-xx.jar)

The above approach will increase no of dependencies in our build file, it will increase build file size and it is very much difficult task to manage all these dependencies.

--> To overcome all the above problems, we have to use Spring Boot Starter Component.

--> Spring Boot Starter component combines all related jars into single jar file so that we can add only jar file dependency to our build files instead of adding above jars files to our build file, we need to add one and only one jar file like

"spring-boot-starter-web" jar file.

--> When we add "spring-boot-starter-web" jar file dependency to our build file, then Spring Boot Framework will automatically download all required jars and add to our project classpath.

Spring-boot-starter-web dependency will provide the following jars to our project library.

1. spring-boot-starter

    a.spring-boot

    b.spring-boot-autoconfiogure

    c.spring-boot-starter-logging

2. spring-web

3. spring-webmvc

4. spring-boot-starter-tomcat

    a. tomcat-embeded-core

    b.tomcat-embeded-logging-juli

2.Spring Boot AutoConfigurator:

In general, in spring based applications we have to provide lot of configuration details either through XML files or through Annotations.

EX: Beans configurations and their dependencies

HandlerMapping Configuration

ViewResolver configurations

To avoid all the above configurations Spring Boot has provided an autoConfigurator. Spring Boot Autoconfigurator will come automatically when we add "spring-boot-starter-web" dependency in build file, it will resolve views and Viewresolvers in Spring web MVC applications with out having Spring configuration file and if we use @SpringBootApplication annotation then it will avoid the explicit utilization of @Configuration, @ComponentScan and @EnableAutoConfiguration and provide them all these annotations to the Java Class internally.

@SpringBootApplication = @Confioguration + @ComponentScan + @EnableAutoConfiguration

3.Spring Boot CLI:

--> Spring Boot CLI(Command Line Interface) is a Spring Boot software, it will provide very good environment to run and test Spring Boot applications from command prompt.

--> When we run Spring Boot applications using CLI, then it internally uses Spring Boot Starter and Spring Boot AutoConfigurator components to resolve all dependencies and execute the application.

--> Spring Boot Command Line Interface has provided "spring" command to execute "Groovy" scripts from Command Prompt, that is, we have to prepare Controller classes in ".groovy" file.

Syntax: spring run HelloWorld.groovy

Where HelloWorld.groovy is like Javas HelloWorld.java file name.

--> When we run the above "spring" command on Command Line Interface then Spring Boot will activate Groovyc[Groovy Compiler] and Groovy Grap to add all the dependencies to project library and to execute and test Groovy script.

4.Spring Boot Actuator:

--> Spring Boot Actuators are providing no of Production-Ready features like

1. Spring Boot Actuator is providing environment to manage and monitor your application by using HTTP endpoints or with JMX.

2. Spring Boot Actuators are providing Auditing, health, and metrics automatically to your application.

--> When we run our Spring Boot Web Application using CLI, Spring Boot Actuator automatically provides hostname as “localhost” and default port number as “8080”. We can access this application using “http://localhost:8080/” end point.

--> To activate Spring Boot Actuators we have to use the following dependecies in build file.

EX:

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

Steps to Prepare Spring Boot Applications:

Spring Boot Applications are prepared and executed in the following four ways.

1. Using Simple MAVEN project in Eclipse IDE.

2. STS IDE

3. Spring Boot Initializer

4. Spring Boot CLI[Command Line Interface]

1. Using Simple MAVEN project in Eclipse IDE:

----

1. Make Ready Setups for JDK1.8 and Eclipse IDE latest version.

2. Create MAVEN Project in Eclipse IDE.

3. Provide the required Dependencies in pom.xml file.

4. Prepare Java Class With main() method and the required Spring Boot Annotations.

5. Run Spring Boot Application.

1. Make Ready Setups for JDK1.8 and Eclipse IDE latest version:

Already we have JDK and Eclipse in our system, no need to reinstall them seperately.

2. Create MAVEN Project in Eclipse IDE:

----

1. Right Click on Project Explorer.

2. Select "New".

2. Select "Others".

3. Search for "Maven".

4. Select "Maven Project".

5. Click on "Next" button.

6. select "Create a simple project(Skip Archutype)" checkbox.

7. Click on "Next" button.

8. Provide the following details in Wizard.

   1. group Id: com.durgasoft

   2. Artifact Id: helloapp

   3. Name : Hello Application

9. Click on "Finish" button.

3. Provide the required Dependencies in pom.xml file.

---

open pom.xml and provide the following dependencies in pom.xml.

a)Provide Java-compiler-plugin in build path.

```
<build>
        <plugins>
```

```
                <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>3.6.0</version>
                        <configuration>
                                <source>1.8</source>
                                <target>1.8</target>
                        </configuration>
                </plugin>
        </plugins>
     </build>
```

Right CLick on Project, Select Maven and Select Update Project and Click on OK toget JAVA8 version in lib.

b)Provide Spring Boot Starter Parent

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

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

      <version>1.5.1.RELEASE</version>

    </parent>
```

c)Add Spring Boot Starter web in pom.xml

```
     <dependencies>

      <dependency>

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

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

      </dependency>

     </dependencies>
```

Observe all the dependent JARs under MAVEN Dependencies in MAVEN Project.

4. Prepare Java Class With main() method and the required Spring Boot Annotations:

```
package com.durgasoft;


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

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


@SpringBootApplication

public class HelloWorld {

    public static void main(String[] args) {
        SpringApplication.run(HelloWorld.class, args);
        System.out.println("Hello From First Spring boot Application");
    }
}
```

Where @SpringBootApplication annotation will provide the required @Component, @ComposentScan and @EnableAutoConfigurator annotations.

Where SpringApplications provided run() method will start spring boot application.

If 8080 is busy in our system then Change Server port number from 8080 to 1111 [any number] by providing properties file:

```
src/main/resources/application.properties

-------------------------------------

server.port = 1111
```

5. Run Spring Boot Application:

--

1. Right Click on project

2. Select "RunAs"

3. Select "Java Application".

4. Select "HelloWord"

Then the following output on Console.

```
          .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.1.RELEASE)


2019-10-06 22:02:48.309  INFO 14800 --- [           main] com.durgasoft.HelloWorld                 :
Starting HelloWorld on DESKTOP-LTICM7V with PID 14800
(D:\spring_boot_practice\helloapp\target\classes started by Nagoor in
D:\spring_boot_practice\helloapp)

2019-10-06 22:02:48.312  INFO 14800 --- [           main] com.durgasoft.HelloWorld                 :
No active profile set, falling back to default profiles: default
```

```
2019-10-06 22:02:48.396  INFO 14800 --- [           main]
ationConfigEmbeddedWebApplicationContext : Refreshing
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplication
Context@44a3ec6b: startup date [Sun Oct 06 22:02:48 IST 2019]; root of context hierarchy

2019-10-06 22:02:49.587  INFO 14800 --- [           main]
trationDelegate$BeanPostProcessorChecker : Bean
'org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration' of type
[class org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration] is
not eligible for getting processed by all BeanPostProcessors (for example: not eligible for
auto-proxying)

2019-10-06 22:02:49.686  INFO 14800 --- [           main]
trationDelegate$BeanPostProcessorChecker : Bean 'validator' of type [class
org.springframework.validation.beanvalidation.LocalValidatorFactoryBean] is not eligible
for getting processed by all BeanPostProcessors (for example: not eligible for auto-
proxying)

2019-10-06 22:02:50.277  INFO 14800 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 1111 (http)

2019-10-06 22:02:50.294  INFO 14800 --- [           main]
o.apache.catalina.core.StandardService   : Starting service Tomcat

2019-10-06 22:02:50.295  INFO 14800 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.11

2019-10-06 22:02:50.424  INFO 14800 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]
: Initializing Spring embedded WebApplicationContext
```

```
2019-10-06 22:02:50.424  INFO 14800 --- [ost-startStop-1] o.s.web.context.ContextLoader
: Root WebApplicationContext: initialization completed in 2033 ms

2019-10-06 22:02:50.635  INFO 14800 --- [ost-startStop-1]
o.s.b.w.servlet.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet' to [/]

2019-10-06 22:02:50.644  INFO 14800 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]

2019-10-06 22:02:50.645  INFO 14800 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]

2019-10-06 22:02:50.645  INFO 14800 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]

2019-10-06 22:02:50.645  INFO 14800 --- [ost-startStop-1]
o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]

2019-10-06 22:02:51.217  INFO 14800 --- [           main]
s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice:
org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplication
Context@44a3ec6b: startup date [Sun Oct 06 22:02:48 IST 2019]; root of context hierarchy

2019-10-06 22:02:51.387  INFO 14800 --- [           main]
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public
org.springframework.http.ResponseEntity<java.util.Map<java.lang.String,
java.lang.Object>>
org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.
HttpServletRequest)
```

```
2019-10-06 22:02:51.388  INFO 14800 --- [           main]
s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}"
onto public org.springframework.web.servlet.ModelAndView
org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.
http.HttpServletRequest,javax.servlet.http.HttpServletResponse)

2019-10-06 22:02:51.511  INFO 14800 --- [           main]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler
of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-10-06 22:02:51.512  INFO 14800 --- [           main]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type
[class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-10-06 22:02:51.618  INFO 14800 --- [           main]
o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto
handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

2019-10-06 22:02:51.810  INFO 14800 --- [           main] o.s.j.e.a.AnnotationMBeanExporter
: Registering beans for JMX exposure on startup

2019-10-06 22:02:51.937  INFO 14800 --- [           main]
s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 1111 (http)

2019-10-06 22:02:51.942  INFO 14800 --- [           main] com.durgasoft.HelloWorld                 :
Started HelloWorld in 4.047 seconds (JVM running for 4.509)

Hello From First Spring boot Application
```

Note: If we want to access the same application through Web Client then we have to prepare a Controller class , where we have to provide handler method with a particular return value.

EX:

HelloController.java:

```
package com.durgasoft.controller;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@EnableAutoConfiguration
```

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

Access the above controller class by using the following URL at client browser after executing HelloWorld standalone application.

http://localhost:1111/hello

Note: in URL, application name is not required to specify.

2.Steps to prepare Spring Boot Application by using STS[Spring Tool Suit] IDE:

---

a)Download and instal STS in our System

b)Create Spring Starter project

c)

a)Download and install STS in our System:

----

a)Open "https://spring.io/tools" in Browser.

b)Click on "Spring Tool Suite for Eclipse "Windows 64 bit".

c)Get "spring-tool-suite-4-4.4.0.RELEASE-e4.13.0-win32.win32.x86_64.zip" from downloads and unzip at C drive.

d)Find "sts-4.4.0.RELEASE" folder at C driver and open "sts-4.4.0.RELEASE" and double click on "SpringToolSuite4" Application.

e)Provide workspace "D:\springbootpractice" .

b)Create Spring Starter project:

----

a)Select "File"--> "New"--> "Spring Starter Project".

b)Provide the following details.

a)Name: helloapp

b)Group : com.durgasoft

c)Artifact: helloapp

d)Description: HelloApp for Spring Boot

e)package: com.durgasoft

c)Click on "Next" button.

d)Select the required modules like "web" --"Spring Web".

e)Click on "Next".

f)Click on "Finish" button.

If we do the above steps then we will get SpringBoot project at Project Explorer part.

c)Open HelloApplication.java file which is existed at "src\main\java\com.durgasoft.HelloApplication.java" and write sop(-).

EX: HelloApplication.java:

```
package com.durgasoft;
```

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

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class HelloappApplication {


        public static void main(String[] args) {

                SpringApplication.run(HelloappApplication.class, args);

                System.out.println("HelloApplication from Spring");

        }


}
```

d)Run "HelloApplication" by clicking on "Run" button then we will get Output.

```
   .   ____          _            __ _ _
  /\\/ ___'_ __ _ _(_)_ __  __ _ \ \ \ \
 ( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
```

```
        \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
         '  |____| .__|_| |_|_| |_\__, | / / / /
        =========|_|==============|___/=/_/_/_/
        :: Spring Boot ::        (v2.1.9.RELEASE)


2019-10-07 10:43:43.483  INFO 16632 --- [           main] com.durgasoft.HelloappApplication
: Starting HelloappApplication on DESKTOP-LTICM7V with PID 16632
(D:\spring_boot_practice\sts\helloapp\target\classes started by Nagoor in
D:\spring_boot_practice\sts\helloapp)

2019-10-07 10:43:43.486  INFO 16632 --- [           main] com.durgasoft.HelloappApplication
: No active profile set, falling back to default profiles: default

2019-10-07 10:43:44.444  INFO 16632 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 2222 (http)

2019-10-07 10:43:44.475  INFO 16632 --- [           main]
o.apache.catalina.core.StandardService   : Starting service [Tomcat]

2019-10-07 10:43:44.475  INFO 16632 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.26]

2019-10-07 10:43:44.603  INFO 16632 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       :
Initializing Spring embedded WebApplicationContext

2019-10-07 10:43:44.603  INFO 16632 --- [           main] o.s.web.context.ContextLoader
: Root WebApplicationContext: initialization completed in 1065 ms
```

```
2019-10-07 10:43:44.796  INFO 16632 --- [           main]
o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService
'applicationTaskExecutor'

2019-10-07 10:43:44.925  INFO 16632 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 2222 (http) with
context path ''

2019-10-07 10:43:44.928  INFO 16632 --- [           main] com.durgasoft.HelloappApplication
: Started HelloappApplication in 1.831 seconds (JVM running for 2.894)

HelloApplication from Spring
```

Note: If 8080 port is busy in our system then change server port number by using key-value pair "server.port=2222" in application.properties which is available under "src\main\resources\application.properties".

e)If we want to execute through Web Client then prepare controller class with handler method.

EX: HelloController.java

```
package com.durgasoft.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 "Hello User!";
        }
}
```

f)Execute HelloApplication appliocation and open browser and use the following URL at address bar.

http://localhost:2222/hello

3. Spring Boot Initializr:

-->Spring Boot initializr is a web tool , it will provide option to generate Spring boot projects directly as per the provided project details.

-->Spring Boot Initializr will give an option to download the project directly from web and to execute through Eclipse IDE or some other.

Steps:

----

1. Create Spring Boot Project and Download project:

2. Import Spring boot Application in STS:

1. Create Spring Boot Project and Download project:

----

a)Open "https://start.spring.io" URL at address bar in browser.

b)Provide the following details:

a)project: Maven Project

b)Language : JAVA

c)Spring Boot: 2.1.9

d)Project Metadata :

group: com.durgasoft

artifact: helloapp

Options:

Name: helloapp

Description: Helloapp for Spring Boot

Packaging : jar

JAVA: 8

Dependencies: web --> Spring Web

c)Click on "Generate" button.

If we do the above step then helloapp.zip file is generated and downloaded.

2. Import Spring boot Application in STS:

----

a)Copy helloapp.zip" file to "D:\mywebapps" location from downloads.

b)Unzip "helloapp.zip" file.

c)Open STS and import the project as existed Maven project:

1)Select "File" --> "import" --> "Maven" --> "Existed Maven Projects".

2)Click on "Next" button

3)provide Root Directory "D:\mywebapps".

4)Click on "Finish" button.

d)Open "com.durgasoft.HelloappApplication.java" from src\main\java and do required modifications.

EX: HelloappApplication.java

```
package com.durgasoft;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class HelloappApplication {


        public static void main(String[] args) {

                SpringApplication.run(HelloappApplication.class, args);
        }


}
```

Note: If 8080 port is busy in our system then change port number from 8080 to some other by providing "server.port" in application.properties file which is existed in "src\Main\resources\application.properties".

e)Execute HelloappApplication.java file by clicking on "Run" button.

Output:

```
.   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
\\/ ___)| |_)| | | | | || (_| |  ) ) ) )
'  |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
:: Spring Boot ::        (v2.1.9.RELEASE)
```

```
2019-10-07 11:37:04.769  INFO 3492 --- [           main] com.durgasoft.HelloappApplication
: Starting HelloappApplication on DESKTOP-LTICM7V with PID 3492
(D:\mywebapps\helloapp\target\classes started by Nagoor in D:\mywebapps\helloapp)

2019-10-07 11:37:04.771  INFO 3492 --- [           main] com.durgasoft.HelloappApplication
: No active profile set, falling back to default profiles: default

2019-10-07 11:37:05.744  INFO 3492 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 5555 (http)

2019-10-07 11:37:05.769  INFO 3492 --- [           main]
o.apache.catalina.core.StandardService   : Starting service [Tomcat]

2019-10-07 11:37:05.769  INFO 3492 --- [           main]
org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.26]

2019-10-07 11:37:05.876  INFO 3492 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       :
Initializing Spring embedded WebApplicationContext

2019-10-07 11:37:05.876  INFO 3492 --- [           main] o.s.web.context.ContextLoader
: Root WebApplicationContext: initialization completed in 1063 ms

2019-10-07 11:37:06.098  INFO 3492 --- [           main]
o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService
'applicationTaskExecutor'

2019-10-07 11:37:06.279  INFO 3492 --- [           main]
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 5555 (http) with
context path ''
```

```
2019-10-07 11:37:06.283  INFO 3492 --- [           main] com.durgasoft.HelloappApplication
: Started HelloappApplication in 1.89 seconds (JVM running for 2.8)

HelloApp from Spring Boot initializer
```

e)If we want to prepare Spring Boot application through Web Client then prepare Controller class with handler method and execute that:

EX: HelloController.java

```java
package com.durgasoft.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>";
          }
}
```

To run this , first we have to execute "HelloappApplication" then open browser and use the following URL.

http://localhost:5555/hello

4. Spring Boot CLI[Command Line Interface]:

--

--> Spring Boot CLI(Command Line Interface) is a Spring Boot software to run and test Spring Boot applications from command prompt. When we run Spring Boot applications using CLI, then it internally uses Spring Boot Starter and Spring Boot AutoConfigurate components to resolve all dependencies and execute the application

--> Spring Boot internally contains Groovy and Grape (JAR Dependency Manager) to add Spring Boot Defaults and resolve all dependencies automatically.

--> To use Spring Boot CLI we have to use the following steps.

1. Download and Install Spring Boot CLI.

2. Prepare Groovy Script Code.

3. Run Groovy Script code through Spring Boot CLI.

1. Download and Install Spring Boot CLI:

----

a)Open the link in Browser address bar "https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing- spring-boot.html#getting-started-installing-the-cli". or else search for it in Google.

b)click on "spring-boot-cli-2.1.9.RELEASE-bin.zip" link under Manual Installation.

c)Copy the downloaded ""spring-boot-cli-2.1.9.RELEASE-bin.zip"" in C drive and extract it then we will find "spring-2.1.9.RELEASE" folder in C Driver.

d)Set "path" environment variable to "C:\spring-2.1.9.RELEASE\bin" location.

```
  set path=%path%;C:\spring-2.1.9.RELEASE\bin;
```

e)Check whether Spring command is running or not in Command prompt.

Spring Boot CLI commands:

1. spring --version --> To get Spring CLI version.

2. spring --version --> To get help for Spring CLI.

3. spring run Groovy_Script_File_Name.groovy --> to run Groovy Script

2. Prepare Groovy Script Code:

It is almost all same as JAVA code only, but, the file extension should be .groovy.

EX: D:\spring_boot_practice\cli\HelloController.groovy

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

3. If 8080 is busy then change tomcat port number from 8080 to 1234 by providing application.properties file at the same location:

D:\spring_boot_practice\cli\application.properties

```
--------------------------------------------------

server.port=1234
```

4. Run Groovy Script code through Spring Boot CLI.

----

1)Run Groovy Script code by using "spring run" command.

```
  D:\spring_boot_practice\cli>spring run HelloController.groovy
```

2)Use the following URL at Browser:

```
 http://localhost:1234/hello
```

Postman:

Postman is a web tool, it can be used to send different types of request like GET, POST, PUT, OPTIONS,......, but, it required HttpServer environment.

Steps:

1.Download and install Postman tool.

2.Open Postman tool and send request to Spring boot Application which is running at present.

1.Download and install Postman tool:

--

a)Open "https://www.getpostman.com/downloads/" link in brower Addressbar.

b)Click on Download button and select Windows 64 bit

With the above Postman tool will download with the setup file "Postman-win64-7.8.0-Setup.exe.

c)Execute Postman setup file by double clicking on "Postman-win64-7.8.0-Setup.exe".

d)Provide signin details or click on Open with google Account.

With the above Postman tool will open.

2.Send request to Spring boot Application which is running at present:

----

1. Enter Application URL

2. Select the request type like GET or POST,...

3. Click on Send button and check the response in Body part.

EX:

---

application.properties

```
server.port=1234
```

HelloappApplication.java

```
package com.durgasoft;

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

```
@SpringBootApplication
public class HelloappApplication {

        public static void main(String[] args) {
                SpringApplication.run(HelloappApplication.class, args);
        }
}
HelloController.java
---------------------
package com.durgasoft.controller;

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

@RestController
public class HelloController {
```

```
        @RequestMapping(value="/hello",method = RequestMethod.GET )
        public String sayHello() {
                return "Hello User!";
        }

        @RequestMapping(value="/hello",method = RequestMethod.POST )
        public String sayWelcome() {
                return "Welcome User!";
        }
}
```

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>2.1.9.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
```

```
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>helloapp</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>helloapp</name>
    <description>helloapp for Spring Boot</description>

    <properties>
        <java.version>1.8</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 Postman.

If we send GET request with the url "http://localhost:1234/hello" then we will get "Hello User!" response.

If we send POST request with the url "http://localhost:1234/hello" then we will get "Welcome User!" response.

How to use User forms in Spring boot Application:

---

1. Prepare Spring starter project in STS.

2. Prepare "webapp/WEB-INF" folders under "src/main" folders to provide JSP and html pages.

3. Add "tomcat-jasper" dependency in pom.xml inorder to support for JSP.

4. Prepare HTML, JSP pages under "WEB-INF" folder.

5. Prepare Controller classes as per Spring WEB MVC and as per the requirement.

6. Provide server.port=1234, spring.mvc.view.prefix=/WEB-INF/, spring.mvc.view.suffix=.jsp key value pairs in application.properties file.

7. Access SpringBoot application as standalone application.

8. Open browser and send request.

EX:

```
helloapp
   |---/src/main/java
   |         |---com.durgasoft
   |                  |-----HelloappApplication.java
   |---com.durgasoft.controller
   |                   |HelloController.java
   |---/src/main/resources
   |            |---static
   |            |---templates
         |            |---application.properties
         |---/src/test/java
         |---src
         |      |---main
         |          |----webapp
         |                 |----WEB-INF
         |                        |---hellopage.jsp
         |                        |---wishpage.jsp
         |---/src/test
         |---target
         |---HELP.md
         |---mvnw
         |---mvnw.cmd
         |---pom.xml
```

hellopage.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
```

```
<body>
        <h2 style="color: red;" align="center">Durga Software Solutions</h2>
        <h3 style="color: blue;" align="center">User Hello Form</h3>
        <form method="post" action="wish">
        <table align="center">
        <tr>
            <td>User Name</td>
            <td><input type="text" name="username"></td>
        </tr>
        <tr>
            <td><input type="submit" value="SayHello"/></td>
        </tr>
        </table>
        </form>
        </body>
        </html>
```

wishpage.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1 style="color: red;" align="center">Hello ${username}!</h1>
<h2 align="center">
<a href="hellopage">|User Hello Page|</a>
</h2>
</body>
</html>
```

HelloController.java:

----

package com.durgasoft.controller;

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class HelloController {

    @RequestMapping(value="/hellopage",method = RequestMethod.GET )
    public String sayHello() {
        return "hellopage";
    }

    @RequestMapping(value="/wish",method = RequestMethod.POST )
    public String wish(@RequestParam String username, ModelMap map) {
        map.addAttribute("username", username);
        return "wishpage";
    }
}
```

HelloappApplication.java:

----

package com.durgasoft;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

Application.properties:

```
server.port=1234
spring.mvc.view.prefix = /WEB-INF/
spring.mvc.view.suffix = .jsp
```

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>2.1.9.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>helloapp</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>helloapp</name>
    <description>helloapp for Spring Boot</description>

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

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


        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jasper</artifactId>
            <version>9.0.26</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:

```
loginapp
  |---/src/main/java
  |       |---com.durgasoft
  |             |-----LoginappApplication.java
  |---com.durgasoft.controller
  |              |LoginController.java
  |---/src/main/resources
  |          |---static
  |          |---templates
  |          |---application.properties
  |---/src/test/java
  |---src
```

```
|     |---main
|         |----webapp
|               |----WEB-INF
|                     |---failure.jsp
|                     |---loginform.jsp
|                     |---success.jsp
|---/src/test
|---target
|---HELP.md
|---mvnw
|---mvnw.cmd
|---pom.xml
```

loginform.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
```

```
<title>Insert title here</title>
</head>
<body>
    <h2 style="color: red;" align="center">Durga Software Solutions</h2>
    <h3 style="color: blue;" align="center">User Login Form</h3>
    <form method="post" action="/login">
    <table align="center">
    <tr>
        <td>User Name</td>
        <td><input type="text" name="uname"></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password" name="upwd"></td>
    </tr>
    <tr>
        <td><input type="submit" value="Login"/></td>
    </tr>
    </table>
    </form>
    </body>
    </html>
```

success.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1 style="color: red;" align="center">Login Success</h1>
<h2 align="center">
<a href="loginpage">|Login Page|</a>
```

```
</h2>
</body>
</html>
```

failure.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1 style="color: red;" align="center">Login Success</h1>
<h2 align="center">
<a href="loginpage">|Login Page|</a>
</h2>
</body>
</html>
```

LoginController.java

```
package com.durgasoft.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class LoginController {

    @RequestMapping(value="/loginpage", method = RequestMethod.GET)
    public String showLoginPage() {
        return "loginform";
```

```java
    }

    @RequestMapping(value="/login", method = RequestMethod.POST)
    public String login(@RequestParam String uname, @RequestParam String upwd) {
        String status = "";
        if(uname.equals("durga") && upwd.equals("durga")) {
            status = "success";
        }else {
            status = "failure";
        }
        return status;
    }
}
```

LoginappApplication.java

----

package com.durgasoft;

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

@SpringBootApplication
public class LoginappApplication {

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

}
```

Application.properties

----

server.port = 1234

```
spring.mvc.view.prefix=/WEB-INF/

spring.mvc.view.suffix=.jsp
```

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>2.1.9.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>loginapp</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>loginapp</name>
    <description>Spring boot Login app</description>

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

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


```

```
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jasper</artifactId>
            <version>9.0.26</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>
```

First Execute LoginappApplication as standalone application then open browser and access the application by using the following URL.

http://localhost:1234/loginpage.

Spring Boot Application with Spring JDBC Application:

To prepare Spring Boot Application with Spring-JDBC , we have to use the following Steps.

1. Install Maven in our System:

a)Download Maven software from "http://maven.apache.org/download.cgi" url.

b)Unzip "apache-maven-3.6.2-bin.zip" file.

c)Set "path" environment variable to "C:\apache-maven-3.6.2\bin" either temporarily or permanently.

```
set path=C:\apache-maven-3.6.2\bin;
```

2. Prepare ojdbc6 dependency manually:

Oracle Driver is a driver provided Oracle , it is licenced, it is not possible to get dependency details manually, so we must prepare ojdbc6.jar dependency.

To prepare ojdbc6 dependency manually we have to use the following command on Command prompt.

```
D:\>mvn install:install-file -
Dfile=C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar -
DgroupId=com.oracle -   DartifactId=ojdbc6 -Dversion=11.2.0 -Dpackaging=jar
```

For ojdbc6.jar file, we have to use the following dependency.

groupid: com.oracle

artifactid: ojdbc6

version:11.2.0

3. Prepare Spring boot Starter project in STS with Spring Web and JDBC API in SQL part.

4. Open pom.xml and add "ojdbc6" and tomcat-jasper:

Where ojdbc6.jar file is to interact with Oracle Database.

Where tomcat jasper is to provide JSPs support.

```
<project>
   ----
   <dependencies>
      <dependency>
     <groupId>com.oracle</groupId>
     <artifactId>ojdbc6</artifactId>
     <version>11.2.0</version>
     </dependency>
     <dependency>
   <groupId>org.apache.tomcat</groupId>
   <artifactId>tomcat-jasper</artifactId>
   <version>9.0.26</version>
     </dependency>
      -----
  </dependencies>
  ----
</project>
```

5. Provide the following details in application.properties file:

```
  server.port=1234

  spring.mvc.view.prefix=/WEB-INF/

  spring.mvc.view.suffix=.jsp

  spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

  spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

  spring.datasource.username=system

  spring.datasource.password=durga
```

Note: datasource name is not required, because, Spring boot is using Apache's "Hikary Connection pooling Mechanism" bydefault.

6. Prepare "webapp/WEB-INF" folder under "src/main" folder.

7. Prepare all jsp pages under "src/main/webapp/WEB-INF" folder.

8. Prepare all Java Resources as per the structer.

EX:

----

application.properties:

```
server.port=1234
spring.mvc.view.prefix=/WEB-INF/
spring.mvc.view.suffix=.jsp
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=durga
```

welcome.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1 style="color: red;" align="center">Durga Software Solutions</h1>
<hr>
<table align="center" width="100%" height="100%">
<tr>
    <td align="center">
```

```
        <a href="addform">ADD Student</a>
        <a href="searchform">SEARCH Student</a>
        <a href="updateform">UPDATE Student</a>
        <a href="deleteform">DELETE Student</a>
    </td>
</tr>
</table>
</body>
</html>
```

Addform.jsp

```
<%@ page <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>

<br><br><br><br><br>
<form method="post" action="add">
<table align="center">
<tr>
    <td>Student Id</td>
    <td><input type="text" name="sid"></td>
</tr>
<tr>
    <td>Student Name</td>
    <td><input type="text" name="sname"></td>
</tr>
<tr>
    <td>Student Address</td>
    <td><input type="text" name="saddr"></td>
```

```
</tr>
<tr>
    <td><input type="submit" value="ADD"/></td>
</tr>
</table>
</form>
</body>
</html>
```

searchform.jsp:

```
<%<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-
//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>

<br><br><br><br><br>
<form method="post" action="search">
<table align="center">
<tr>
    <td>Student Id</td>
    <td><input type="text" name="sid"></td>
</tr>

<tr>
    <td><input type="submit" value="SEARCH"/></td>
</tr>
</table>
</form>
```

```
</body>
</html>
```

updateform.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-
//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br>
<form method="post" action="editform">
<table align="center">
<tr>
    <td>Student Id</td>
    <td><input type="text" name="sid"></td>
</tr>

<tr>
    <td><input type="submit" value="GetEditForm"/></td>
</tr>
</table>
</form>
</body>
</html>
```

Editform.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br>
<form method="post" action="update">
<table align="center">
<tr>
    <td>Student Id</td>
    <td>${std.sid}</td>
</tr>
<tr>
    <td>Student Name</td>
    <td><input type="text" name="sname" value="${std.sname}"></td>
</tr>
<tr>
    <td>Student Address</td>
    <td><input type="text" name="saddr" value="${std.saddr}"></td>
</tr>
<tr>
    <td><input type="submit" value="Update"></td>
</tr>

</table>
<input type="hidden" name="sid" value="${std.sid}">
</form>
</body>
</html>
```

Deleteform.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>

<br><br><br><br><br>
<form method="post" action="delete">
<table align="center">
<tr>
    <td>Student Id</td>
    <td><input type="text" name="sid"></td>
</tr>

<tr>
    <td><input type="submit" value="DELETE"/></td>
</tr>
</table>
</form>
</body>
</html>
```

studentdetails.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
```

```
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br>
<table align="center" border="1">
<tr>
    <td>Student Id</td>
    <td>${std.sid}</td>
</tr>
<tr>
    <td>Student Name</td>
    <td>${std.sname}</td>
</tr>
<tr>
    <td>Student Address</td>
    <td>${std.saddr}</td>
</tr>
</table>
</body>
</html>
```

success.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br><br>
<h1 style="color: red;" align="center">${operation} Success</h1>
</body>
</html>
```

failure.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br><br>
<h1 style="color: red;" align="center">${operation} Failure</h1>
</body>
</html>
```

existed.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br><br>
<h1 style="color: red;" align="center">Student Existed Already</h1>
</body>
</html>
```

Notexisted.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="/WEB-INF/welcome.jsp"/>
<br><br><br><br><br><br>
<h1 style="color: red;" align="center">Student Not Existed</h1>
</body>
</html>
```

StudentController.java:

```
package com.durgasoft.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import com.durgasoft.dto.Student;
import com.durgasoft.service.StudentService;

@Controller
public class StudentController {

```

```java
    @Autowired
    StudentService studentService;

    @RequestMapping("/welcomepage")
    public String showWelcomePage() {
        return "welcome";
    }
    @RequestMapping("/addform")
    public String showAddPage() {
        return "addform";
    }
    @RequestMapping("/searchform")
    public String showSearchPage() {
        return "searchform";
    }
    @RequestMapping("/updateform")
    public String showUpdatePage() {
        return "updateform";
    }
    @RequestMapping("/editformform")
    public String showEditPage() {
        return "editform";
    }
    @RequestMapping("/deleteform")
    public String showDeletePage() {
        return "deleteform";
    }

    @RequestMapping(value="/add", method = RequestMethod.POST)
    public String add(ModelMap map, @RequestParam String sid, @RequestParam String sname,@RequestParam String saddr) {
        map.addAttribute("operation", "Student Insertion");
        Student std = new Student();
        std.setSid(sid);
        std.setSname(sname);
        std.setSaddr(saddr);

        String status = studentService.addStudent(std);

        return status;
    }
```

```
    @RequestMapping(value="/search", method=RequestMethod.POST)
    public String search(ModelMap map, @RequestParam String sid) {
        String status = "";
        Student std = studentService.searchStudent(sid);
        if(std == null) {
            status = "notexisted";
        }else {
            map.addAttribute("std", std);
            status = "studentdetails";
        }
        return status;
    }

    @RequestMapping(value="editform", method = RequestMethod.POST)
    public String getEditForm(ModelMap map, @RequestParam String sid) {
        String status = "";
        Student std = studentService.searchStudent(sid);
        if(std == null) {
            status = "notexisted";
        }else {
            map.addAttribute("std", std);
            status="editform";
        }
        return status;
    }

    @RequestMapping(value="update", method = RequestMethod.POST)
    public String update(ModelMap map,@RequestParam String sid, @RequestParam String sname, @RequestParam String saddr) {
        String status = "";
        map.addAttribute("operation", "Student Updation");
        Student std = new Student();
        std.setSid(sid);
        std.setSname(sname);
        std.setSaddr(saddr);

        status = studentService.updateStudent(std);
        return status;
    }

```

```
    @RequestMapping(value = "delete", method = RequestMethod.POST)
    public String delete(ModelMap map, @RequestParam String sid) {
        map.addAttribute("operation", "Student Deletion");
        String status = studentService.deleteStudent(sid);
                return status;
           }
       }
```

StudentService.java:

```
package com.durgasoft.service;

import com.durgasoft.dto.Student;

public interface StudentService {
    public String addStudent(Student std);
    public Student searchStudent(String sid);
    public String updateStudent(Student std);
    public String deleteStudent(String sid);
}
```

StudentServiceImpl.java:

```
package com.durgasoft.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.durgasoft.dao.StudentDao;
import com.durgasoft.dto.Student;

@Service
public class StudentServiceImpl implements StudentService {

    @Autowired
```

```
    StudentDao studentDao;
    @Override
    public String addStudent(Student std) {
        String status = studentDao.add(std);
        return status;
    }

    @Override
    public Student searchStudent(String sid) {
        Student std = studentDao.search(sid);
        return std;
    }

    @Override
    public String updateStudent(Student std) {
        String status = studentDao.update(std);
        return status;
    }

    @Override
    public String deleteStudent(String sid) {
        String status = studentDao.delete(sid);
        return status;
    }

}
```

StudentDao.java

----

package com.durgasoft.dao;

import com.durgasoft.dto.Student;

```
public interface StudentDao {

        public String add(Student std);

        public Student search(String sid);

        public String update(Student std);

        public String delete(String sid);

}
```

StudentDaoImpl.java:

----

package com.durgasoft.dao;

```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.durgasoft.dto.Student;

@Repository
public class StudentDaoImpl implements StudentDao {

    @Autowired
    JdbcTemplate jdbcTemplate;
```

```
    @Override
    public String add(Student std) {
        String status = "";
        Student student = search(std.getSid());
        if(student == null) {
            int rowCount = jdbcTemplate.update("insert into student values('"+std.getSid()+"','"+std.getSname()+"','"+std.getSaddr()+"')");
            if(rowCount ==1) {
                status = "success";
            }else {
                status = "failure";
            }
        }else {
            status = "existed";
        }
        return status;
    }

    @Override
    public Student search(String sid) {
        Student std = null;
        List<Student> studentsList = jdbcTemplate.query("select * from student where SID = '"+sid+"'", (rs, rowNum) ->{
            Student s = new Student();
            s.setSid(rs.getString("SID"));
            s.setSname(rs.getString("SNAME"));
            s.setSaddr(rs.getString("SADDR"));
            return s;
        });
        if(studentsList.isEmpty() == true) {
            std = null;
        }else {
            std = studentsList.get(0);
        }
        return std;
    }

    @Override
    public String update(Student std) {
        String status = "";
```

```java
        int rowCount = jdbcTemplate.update("update student set SNAME = '"+std.getSname()+"', SADDR = '"+std.getSaddr()+"' where SID = '"+std.getSid()+"'");
        if(rowCount == 1) {
            status = "success";
        }else {
            status = "failure";
        }
        return status;
    }

    @Override
    public String delete(String sid) {
        String status = "";
        Student std = search(sid);
        if(std == null) {
            status = "notexisted";
        }else {
            int rowCount = jdbcTemplate.update("delete from student where SID = '"+sid+"'");
            if(rowCount == 1) {
                status = "success";
            }else {
                status = "failure";
            }
        }
        return status;
    }
}
```

StudentApplication.java

```java
package com.durgasoft;

import org.springframework.boot.SpringApplication;

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

```
@SpringBootApplication

public class StudentappApplication {

        public static void main(String[] args) {

                SpringApplication.run(StudentappApplication.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>2.1.9.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>employee</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>employeeapp</name>
    <description>Spring boot employee app</description>

    <properties>
        <java.version>1.8</java.version>
```

```
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
        </dependency>
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.2.0</version>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat</groupId>
            <artifactId>tomcat-jasper</artifactId>
            <version>9.0.26</version>
        </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>
```

Connection Pooling In Spring Boot:

Spring Boot will follow an algorithm to find Connection pooling mechanism internally.

1. Spring boot willsearch for Hikary Connection Pooling if it is existed.

2. If Hikary Connection pooling is not existed then Spring boot will search for Tomcat Pooling DataSource.

3. If both Hikary and Tomcat Pooling DataSource are not existed then Spring boot will search for Commons DBCP2 mechanism if it is existed.

To check the above algorithm then we have to use the following Steps:

1. Prepare SpringBootApplication By extending CommandLineRunner and by providing run(String... args) method:

```
package com.durgasoft;

import javax.sql.DataSource;


import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

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

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

import org.springframework.context.annotation.ComponentScan;


@SpringBootApplication

@ComponentScan

public class StudentappApplication implements CommandLineRunner{

        @Autowired

        private DataSource dataSource;

        public static void main(String[] args) {

                SpringApplication.run(StudentappApplication.class, args);

        }


        @Override

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

                System.out.println("Connection Pooling  : "+dataSource);


        }

}
```

2. Apply a particular Connection pooling Mechanism:

a)case-1: only Hikary Connection Pooling, No Tomcat JDBC and No Apache DBCP2

In pom.xml

No changes , normal pom.xml.

Bydefault, Spring boot will search for Hikary Connection pooling mechanism.

If we run the application then we will get the following output.

```
---Spring Boot logo----
 ----
Connection Pooling  : HikariDataSource (HikariPool-1)
```

2)case-2:Only Tomcat Jdbc Connection pooling, No Hikary Connection Pooling and No Apache DBCP2 Connection pooling

In pom.xml: Exclude Hikary Connection Pooling in spring-boot-starter-jdbc and provide tomcat-jdbc dependency.

```
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-jdbc</artifactId>
                <exclusions>
                        <exclusion>
                                <groupId>com.zaxxer</groupId>
                                <artifactId>HikariCP</artifactId>
                        </exclusion>
                </exclusions>
        </dependency>
        <dependency>
                <groupId>org.apache.tomcat</groupId>
                <artifactId>tomcat-jdbc</artifactId>
                <version>9.0.10</version>
        </dependency>
```

```
            ------
</dependencies>
```

If we run the application then we will get the following output.

```
---Spring Boot logo----
  ----
Connection Pooling  : org.apache.tomcat.jdbc.pool.DataSource@6438a7fe{----}
```

3)Case-3: Only Apache DBCP2, No Hikary Connection Pooling and No Tomcat-Jdbc Connection pooling.

In pom.xml: Exclude "Hikary" and "Tomcat Jdbc" Connection Pooling mechanisms in "Spring-Boot-Starter-jdbc". and include Apache DBCP2 dependency.

EX:

```
---
<dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.apache.tomcat</groupId>
                <artifactId>tomcat-jdbc</artifactId>
            </exclusion>
             <exclusion>
                <groupId>com.zaxxer</groupId>
                <artifactId>HikariCP</artifactId>
            </exclusion>

        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-dbcp2</artifactId>
        <version>2.1</version>
    </dependency>
```

```
    -------
</dependencies>
```

If we run the application then we will get the following output.

```
---Spring Boot logo----
  ----
Connection Pooling  : org.apache.commons.dbcp2.BasicDataSource@7c8f9c2e
```

For all the above cases we have to use the following content in application.properties file.

```
application.properties
------------------------
server.port=1234
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=durga
```

If we want to use any other Connection Pooling Mechanisms like C3PO or proxool in Spring boot applications then we have to use the following steps.

1. Provide Connection pooling mechanisms dependency in pom.xml file.

2. Provide "spring.datasource.type" property in application.properties file.

EX: C3P0 Connection Pooling Mechnaims:

----

1. Prepare c3p0.properties file under "src/main/resources"

```
  c3p0.properties
  ----------------
  c3p0.initialPoolSize = 10
  c3p0.maxPoolSize = 10
  c3p0.maxStatements = 20
  c3p0.maxIdleTime = 20000
```

2. Provide "spring.datasource.type" property in application.properties file:

  application.properties

```
-----------------------

  spring.datasource.type = com.mchange.v2.c3p0.ComboPooledDataSource

  ----

  ----
```

3. Add c3p0 dependency in pom.xml file and exclude Hikary in pom.xml:

```
 pom.xml

 ------

<dependencies>

     <dependency>

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

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


     <exclusions>

            <exclusion>

                    <groupId>com.zaxxer</groupId>

                    <artifactId>HikariCP</artifactId>

            </exclusion>

     </exclusions>
```

```
        <dependency>
                <groupId>c3p0</groupId>
                <artifactId>c3p0</artifactId>
                <version>0.9.1.2</version>
        </dependency>
    -----
</dependecies>


EX:
----
c3p0.properties
-----------------
c3p0.initialPoolSize = 10
c3p0.maxPoolSize = 10
c3p0.maxStatements = 20
c3p0.maxIdleTime = 20000


application.properties
```

```
------------------------

server.port=1234

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.url==jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=durga

spring.datasource.type = com.mchange.v2.c3p0.ComboPooledDataSource
```

CustomConnectionPoolingApplication.java

```
---------------------------------------

package com.durgasoft;


import javax.sql.DataSource;


import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

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

```
@SpringBootApplication
public class CustomconnectionpoolingApplication implements CommandLineRunner {

	@Autowired
	private DataSource dataSource;

	public static void main(String[] args) {
		SpringApplication.run(CustomconnectionpoolingApplication.class, args);
	}
	@Override
	public void run(String... args) throws Exception {
		System.out.println("Connection Pooling Data Source : "+dataSource);
	}
}
```

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>2.1.9.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>customconnectionpooling</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>customconnectionpooling</name>
    <description>Spring boot application on Customized Connection Pooling</description>

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

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jdbc</artifactId>
            <exclusions>

                <!-- <exclusion>
                    <groupId>org.apache.tomcat</groupId>
                    <artifactId>tomcat-jdbc</artifactId>
                </exclusion> -->
                <exclusion>
                    <groupId>com.zaxxer</groupId>
                    <artifactId>HikariCP</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>11.2.0</version>
        </dependency>

        <dependency>
            <groupId>c3p0</groupId>
            <artifactId>c3p0</artifactId>
            <version>0.9.1.2</version>
        </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 Memory Databases in Spring Boot:

IN enterprise applications, if we want to use traditional Databases then we have to provide lot of configurations like

1. Install Database

2. Providing Database setups

3. Create Schema

4. Create table

5. Populating Data.

6. Creating dataSource to connect with database

To overcome the above problems, we will use In Memory databases.

In Memory Databases are providing the following Advantages.

Zero project setup or infrastructure

Zero Configuration

Zero Maintainance

Easy to use for Learning, POCs and Unit Tests

Spring Boot provides Simple Configuration to switch between a real database and an in memory database like H2

In general, In Memory databases are started when application is started and destroyed when application is stopped.

EX: H2

H2 is a relational database management system written in Java. It can be embedded in Java applications or run in the client-server mode.

If we want to use H2 Im Memory Database for our applications then we have to use the following steps.

1. Prepare Spring boot starter project with "web, JDBC" "h2" Dependencies in STS.
2. Provide the following properties in application.properties file:

```
spring.h2.console.enabled=true

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa
```

```
  spring.datasource.password=
```

3. Prepare SQL script files under "src/main/resources".

```
  schema.sql
  -------------
  drop table if exists emp1;
  create table emp1(ENO number(5) primary key, ENAME varchar2(10), ESAL float(5),
EADDR varchar2(10));


  data.sql
  --------
  insert into emp1 values(111, 'AAA', 5000, 'Hyd');
  insert into emp1 values(222, 'BBB', 6000, 'Hyd');
  insert into emp1 values(333, 'CCC', 7000, 'Hyd');
  insert into emp1 values(444, 'DDD', 8000, 'Hyd');
  insert into emp1 values(555, 'EEE', 9000, 'Hyd');
```

Spring boot application will load and execute these sql scripting files.

EX:

----

schema.sql

```
drop table if exists emp1;
create table emp1(ENO number(5) primary key, ENAME varchar2(10), ESAL float(5), EADDR varchar2(10));
```

data.sql

```
insert into emp1 values(111, 'AAA', 5000, 'Hyd');
insert into emp1 values(222, 'BBB', 6000, 'Hyd');
insert into emp1 values(333, 'CCC', 7000, 'Hyd');
insert into emp1 values(444, 'DDD', 8000, 'Hyd');
insert into emp1 values(555, 'EEE', 9000, 'Hyd');
```

```
application.properties
-----------------------
server.port=1234
spring.mvc.view.prefix=/WEB-INF/
spring.mvc.view.suffix=.jsp

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

H2DatabaseApplication.java
---------------------------
package com.durgasoft;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
@SpringBootApplication

public class H2databaseApplication //implements CommandLineRunner

{


	public static void main(String[] args) {

		SpringApplication.run(H2databaseApplication.class, args);

	}


}


EmployeeController.java

------------------------

package com.durgasoft.controller;


import java.util.List;


import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;


import com.durgasoft.dao.EmployeeDao;
import com.durgasoft.dto.Employee;


@Controller
public class EmployeeController {


    @Autowired
    private EmployeeDao employeeDao;


    @RequestMapping("getallemployees")
    public String getAllEmployees(ModelMap map) {
        List<Employee> allEmployees = employeeDao.getAllEmployees();
        map.addAttribute("allEmployees", allEmployees);
        return "employeesdetails";
    }
```

```
}


EmployeeDao.java

-----------------

package com.durgasoft.dao;


import java.util.List;


import com.durgasoft.dto.Employee;


public interface EmployeeDao {

	public List<Employee> getAllEmployees();

}
```

Employee.java:

```
package com.durgasoft.dto;
```

```java
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;
      }
}
```

EmployeeDaoImpl.java

----

package com.durgasoft.dao;

```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.stereotype.Repository;


import com.durgasoft.dto.Employee;


@Repository

public class EmployeeDaoImpl implements EmployeeDao {


        @Autowired

        private JdbcTemplate jdbcTemplate;

        @Override

        public List<Employee> getAllEmployees() {

                List<Employee> list = jdbcTemplate.query("select * from emp1", (rs, rowNum)
-> {

                        Employee emp = new Employee();

                        emp.setEno(rs.getInt("ENO"));
```

```
            emp.setEname(rs.getString("ENAME"));

            emp.setEsal(rs.getFloat("ESAL"));

            emp.setEaddr(rs.getString("EADDR"));

            return emp;

        });

        return list;

    }


}
```

employeedetails.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

   pageEncoding="ISO-8859-1"%>

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
```

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h2 style="color: red;" align="center">Employees Details</h2>
<table align="center" border="1">
<tr><th>ENO</th><th>ENAME</th><th>ESAL</th><th>EADDR</th></tr>
<c:forEach var="emp" items="${allEmployees}">
<tr>
        <td>${emp.eno}</td>
        <td>${emp.ename}</td>
        <td>${emp.esal}</td>
        <td>${emp.eaddr}</td>
</tr>
</c:forEach>
</table>
</body>
```

</html>

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>2.1.9.RELEASE</version>
        <relativePath /> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.durgasoft</groupId>
    <artifactId>h2database</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>h2database</name>
    <description>Spring boot application on H2 In Memory Database</description>

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

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


        <dependency>
            <groupId>org.apache.tomcat</groupId>
```

```
            <artifactId>tomcat-jasper</artifactId>
            <version>9.0.26</version>
        </dependency>

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>


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

        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</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>
```

To get H2 In memory Database use the following End Point.

http://localhost:1234/h2-console and click on "Connect" button.

To get all Employees list use the following End Point.

http://localhost:1234/getallemployees