**Spring Boot**

Spring Boot is built on the top of spring and contains all the features of spring.

**It is a** Spring framework module which provides RAD (**Rapid Application Development**) feature to the Spring framework.

Advantage

It allows to avoid heavy configuration of XML which is present in spring:

It provides easy maintenance and creation of REST end points:

It includes embedded Tomcat-server, Jetty server, Undertow server

Deployment is very easy, war and jar file can be easily deployed in the tomcat server:

It is Microservice Based Architecture:

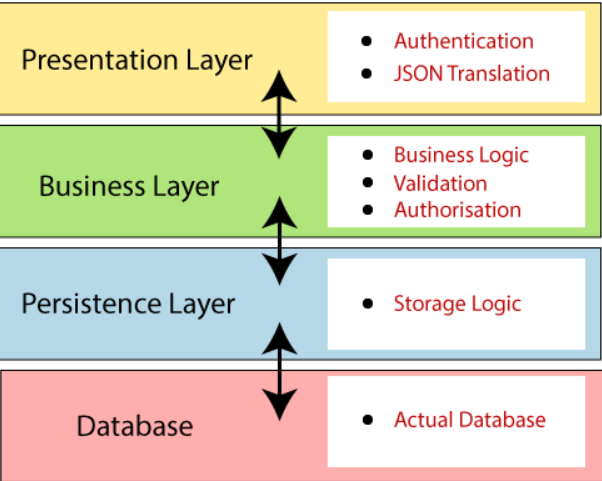
Micro Service is an architecture that allows the developers to develop and deploy services independently.

Let us consider an example of a hospital management system.

* In case of monolithic systems, there will be a single code containing all the features which are very tough to maintain on a huge scale.
* But in the microservice-based system, each feature can be divided into smaller subsystems like service to handle patient registration, service to handle database management, service to handle billing etc.

Spring Boot Architecture

Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.

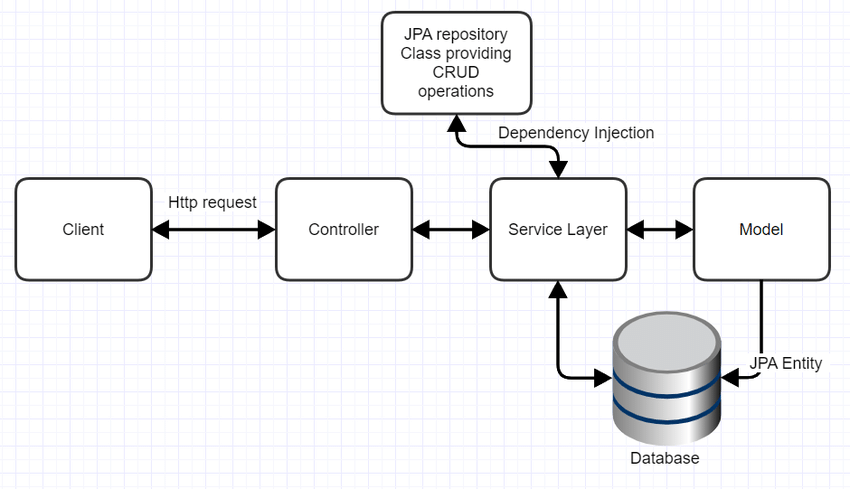


**Presentation Layer:** The presentation layer handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer

**Business Layer:** The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

**Persistence Layer:** The persistence layer contains all the **storage logic** and translates business objects from and to database rows.

**Database Layer:** In the database layer, **CRUD** (create, retrieve, update, delete) operations are performed.



Spring Boot Application – IDE – Eclipse, Intelij, SpringTool Suite

**What is spring-boot-starter-parent dependency?**

The spring-boot-starter-parent dependency is the parent POM providing dependency and plugin management for Spring Boot-based applications. It contains the default versions of Java to use, the default versions of dependencies that Spring Boot uses, and the default configuration of the Maven plugins.

Example

<?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>

    <parent>

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

        <artifactId>spring-boot-dependencies</artifactId>

        <version>${revision}</version>

        <relativePath>../../spring-boot-dependencies</relativePath>

    </parent>

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

    <packaging>pom</packaging>

    <name>Spring Boot Starter Parent</name>

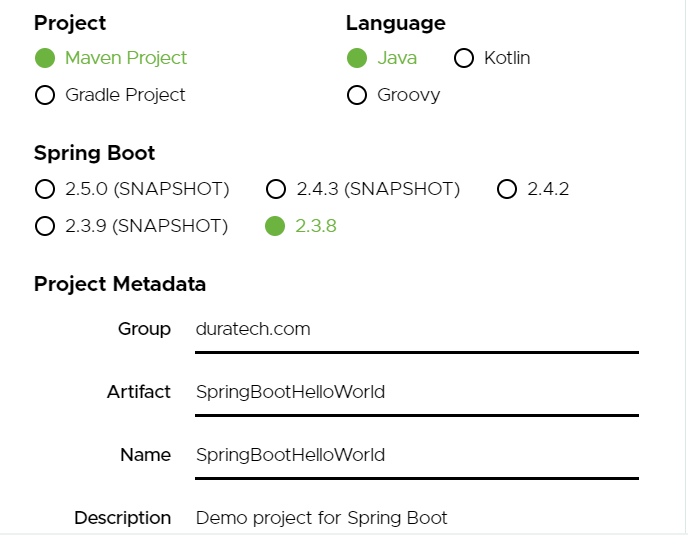
    <description>Parent pom providing dependency and plugin management for applications

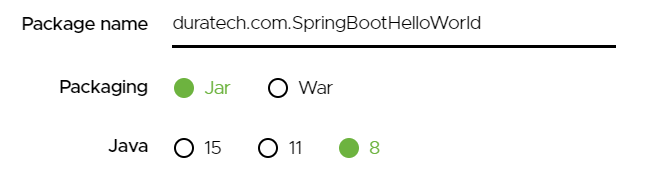
        built with Maven</description>

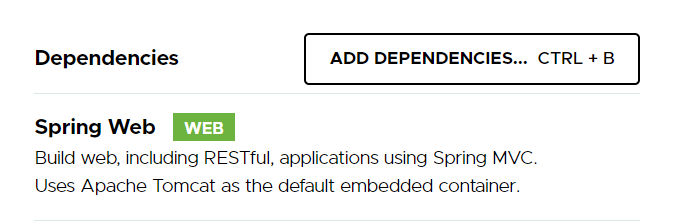
It is highly dependent on the **starter templates** feature which is very powerful.

**Example Simple Spring Boot Hello World Application**

1. Start SpringToolSuite(STS)
2. Create—New 🡪 Spring starter project

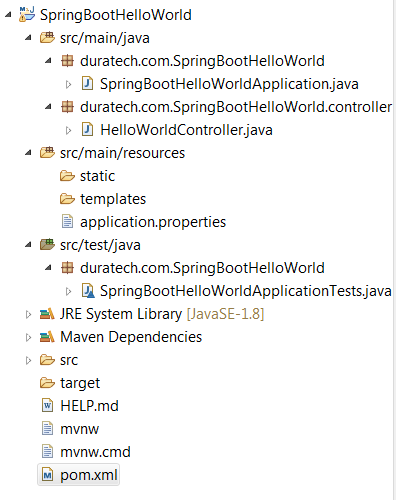






**SpringBootHelloWorld** project will be imported. Dependencies mentioned in the **pom.xml** will be automatically downloaded and added into the classpath.

Project structure



**pom.xml**

<parent>

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

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

<version>2.3.8.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>duratech.com</groupId>

<artifactId>SpringBootHelloWorld</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBootHelloWorld</name>

<description>Demo project 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>

Spring Boot provides various starters for creating Spring Boot based java application. The **spring-boot-starter-parent** is a special type of starter. It is used as a parent in the **pom.xml** file of any kind of Spring Boot application.  
  
The **spring-boot-starter-parent** provides the common configurations such as plugin configuration, default java compiler level, dependency management, UTF-8 source encoding, etc.  
  
There are other starters as well like **spring-boot-starter-web**for creating Spring web applications,  **spring-boot-starter-web-services** is used for creating Spring web-services application.

**Spring Boot Application Class**

**SpringBootHelloWorldApplication.java**

package duratech.com.SpringBootHelloWorld;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootHelloWorldApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootHelloWorldApplication.class, args);

}

}

**@SpringBootApplication** is equivalent to **@Configuration**, **@ComponentScan** and **@EnableAutoConfiguration** combined.

**@EnableAutoConfiguration :**It tells Spring Boot to "guess" how you want to configure Spring, based on the jar dependencies that you have added.  
For example, **spring-boot-starter-web** added Spring MVC and Tomcat, the auto-configuration assumes that you are developing a web application and sets up Spring accordingly.  
  
**@Configuration :**It indicates that the class has @Bean definition methods by defining methods with the @Bean annotation. As a result,  Spring container can process the class and produce Spring Beans to be used in the application  
  
  
**@ComponentScan : @ComponentScan**without arguments indicates Spring to scan the current package and all of its sub-packages.

**Create the Controller**

Create a new Controller java class named **HelloWorldController.java** under package **com.javahungry.controller**. Add the following code into it.  
  
**HelloWorldController.java**

**package** duratech.com.SpringBootHelloWorld.controller;

**import** org.springframework.web.bind.annotation.GetMapping;

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

@RestController

**public** **class** HelloWorldController {

@GetMapping(path="/") /

**public** String sayHello() {

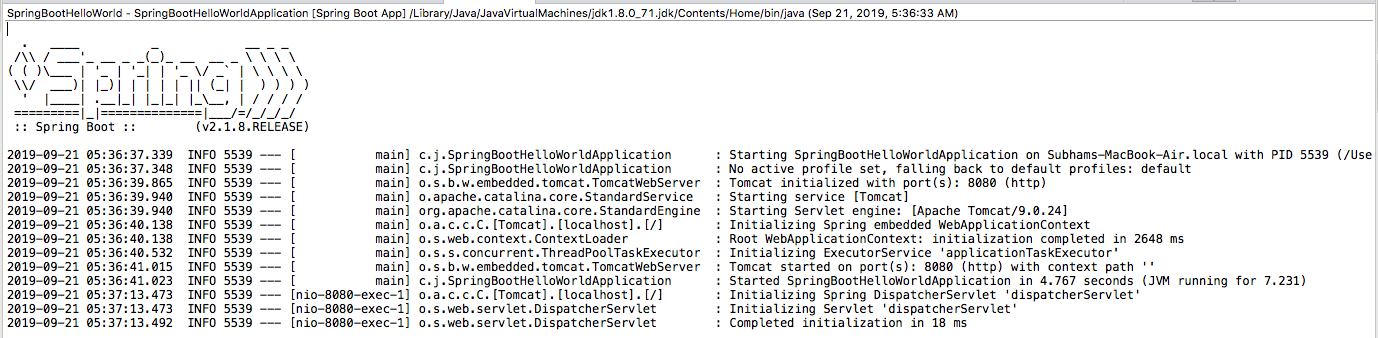
**return** "Hello!! from CGI";

}

}

**Run Application**

Run the **SpringBootHelloWorldApplication.java** by using the main() method



Now enter **http://localhost:8080** in your browser and see the output.

Example-2 Spring Data

Spring Tool Suite - https://spring.io/tools#suite-three

Example REST API

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>dura.com.sb</groupId>

<artifactId>SampleRest</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<parent>

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

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

<version>2.0.5.RELEASE</version>

<relativePath />

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

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

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter</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>

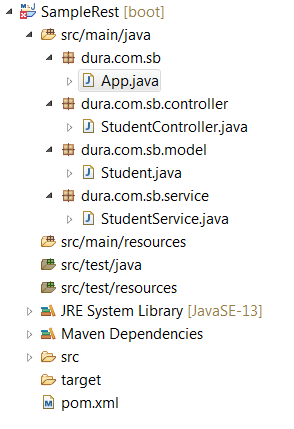
<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>



App.java

package dura.com.sb;

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);

}

}

Model Class

Create a Student model class along with its attributes and a parameterized constructor will all attributes.

**package** dura.com.sb.model;

**public** **class** Student {

// Generate Getters and Setters...

**private** Integer rollNo;

**private** String name;

**private** String course;

**private** String college;

**public** Student(Integer rollNo, String name, String course, String college) {

**this**.rollNo = rollNo;

**this**.name = name;

**this**.course = course;

**this**.college = college;

}

**public** Integer getRollNo() {

**return** rollNo;

}

**public** **void** setRollNo(Integer rollNo) {

**this**.rollNo = rollNo;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getCourse() {

**return** course;

}

**public** **void** setCourse(String course) {

**this**.course = course;

}

**public** String getCollege() {

**return** college;

}

**public** **void** setCollege(String college) {

**this**.college = college;

}

}

Service Class

StudentService class provides the services to the controller based on the request either you want all student data or one and @Service annotation indicates that an annotated class is a **“Service”**.

package dura.com.sb.service;

import java.util.Hashtable;

import org.springframework.stereotype.Service;

import dura.com.sb.model.Student;

@Service

public class StudentService {

Hashtable<Integer, Student> ht = new Hashtable<>();

public StudentService() {

Student student = new Student(1, "Sashi", "PhD", "NIT");

ht.put(1, student);

student = new Student(2, "Sai", "BE", "IIT");

ht.put(2, student);

student = new Student(3, "Chandru", "BE", "GCT");

ht.put(3, student);

student = new Student(4, "Kaviya", "MS", "IIT");

ht.put(4, student);

}

public Student findByRollNo(Integer rollNo) {

if (ht.containsKey(rollNo)) {

return ht.get(rollNo);

} else {

return null;

}

}

public Hashtable<Integer, Student> findAll() {

return ht;

}

}

Controller Class

In Spring’s approach to building RESTful web services, **HTTP** requests are handled by a controller. These components are easily identified by the @RestController annotation, and the StudentController below handles GET requests.

package dura.com.sb.controller;

import java.util.Hashtable;

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

import org.springframework.web.bind.annotation.PathVariable;

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

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

import dura.com.sb.model.Student;

import dura.com.sb.service.StudentService;

@RestController

@RequestMapping("/student")

public class StudentController {

@Autowired

StudentService studentService;

@RequestMapping("/{rollNo}")

public Student getOne(@PathVariable("rollNo") Integer rollNo) {

return studentService.findByRollNo(rollNo);

}

@RequestMapping("/all")

public Hashtable<Integer, Student> getAll() {

return studentService.findAll();

}

}