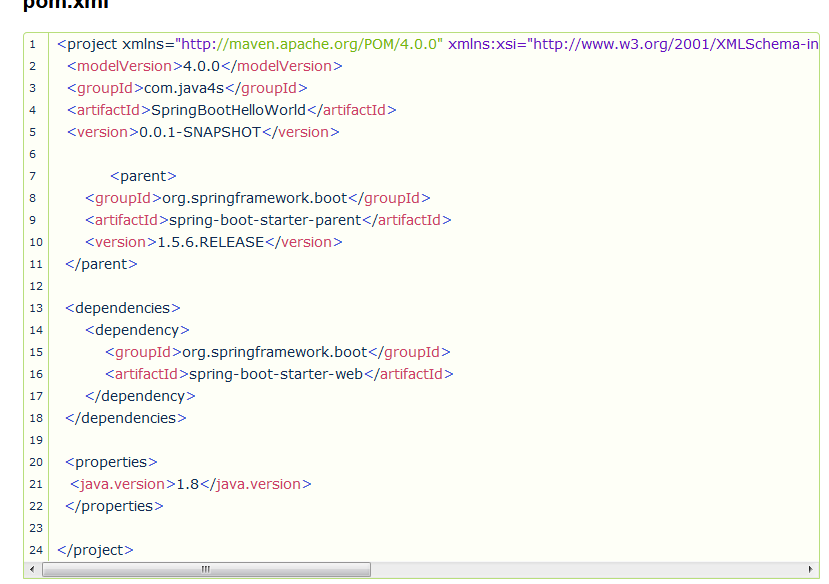
To create real-time spring applications? It includes writing many XML configurations, server setting, adding dependencies…etc. But with spring Boot we can avoid all these boilerplate code, writing XML configurations and annotations. We can create a real-time production ready applications with in minutes.

main advantage of Spring Boot is, we can create spring based applications easily in very less time, without need of any XML configurations. The main disadvantage is, it will be little tough to migrate existing spring enterprise applications to Spring Boot.

[Spring Boot + Maven – Hello World Example Step by Step](https://www.java4s.com/spring-boot-tutorials/spring-boot-maven-hello-world-example-step-by-step/)

Lets add Spring Boot related stuff in it :



I have added spring-boot-starter-parent, spring-boot-starter-web

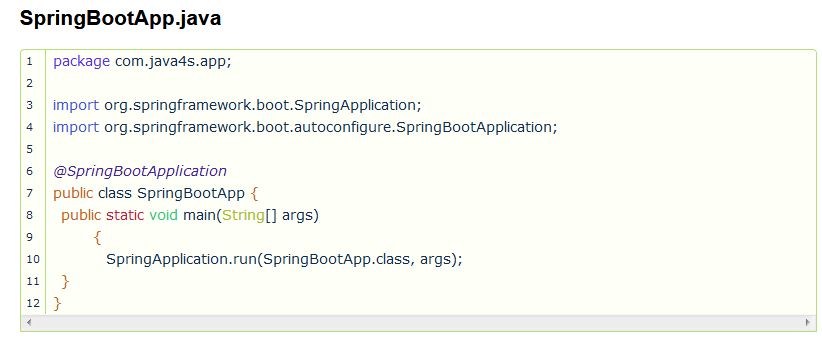
**What is spring-boot-starter-parent?**

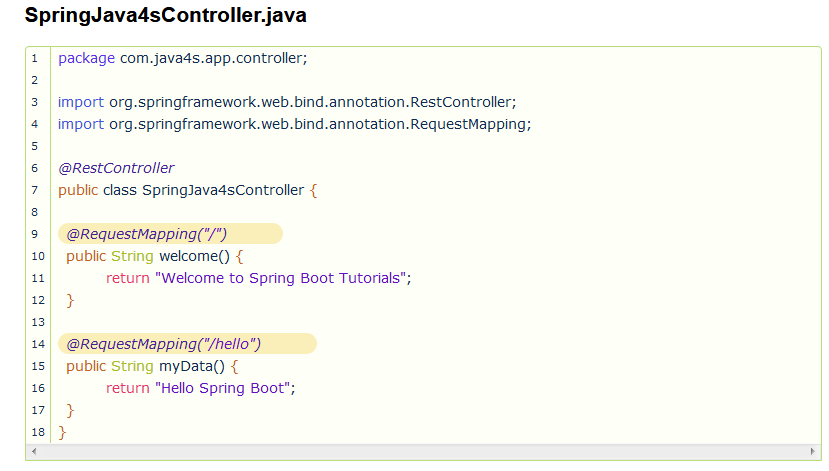
Actually this is an existing project given by spring team which contains Spring Boot supporting configuration data (remember just configuration data, it wont download any jars), we have added this in a <parent> tag means, we are instructing Maven to consider our SpringBootHelloWorld project as a child to it,

**Note**: Did you observe lines 13-18 of pom.xml? I haven’t included version number for spring-boot-starter-web 🙂 but maven downloaded some jar files with some version(s) related to spring-boot-starter-web, how its possible? that’s because of Maven’s parent child relation. While adding spring boot parent project, I have included version as 1.5.6.RELEASE, so again we no need to add version numbers for the dependencies.  As I told you earlier, spring-boot-starter-parent contains configuration meta data, this means, it knows which version of dependency need to be downloaded.  So we have no need to worry about dependencies versions., which will save lot of our time 😉

[Spring Boot – Creating a RESTful Web Service Example](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/)

**Required files**

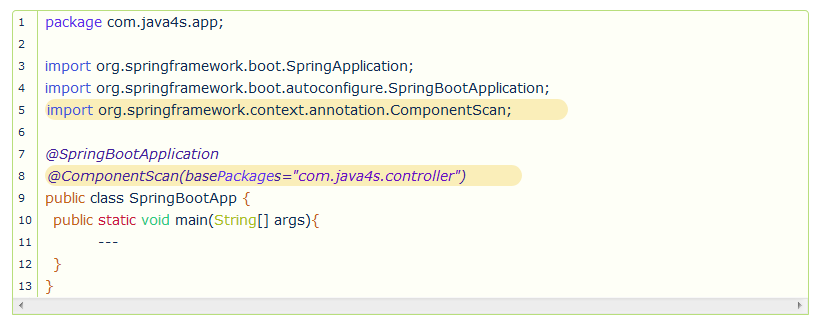
* SpringBootApp.java
* SpringJava4sController.java
* pom.xml
* 



***Note***:  Have you clearly observed the above directory structure? I have created Spring Boot main application class in com.java4s.app and controller class in com.java4s.app.controller, and in my controller class I have written my RESTful service logic and was able to execute the application successfully. How spring boot knows to scan our controller? As we have created our main class in com.java4s.app package, while starting our application, it will scan all the components under that package.

If you create the controller class outside of the main package, lets say com.java4s.controller, If you run the application it gives 404 error,

What’s the solution for this?  we have to add @ComponentScan annotation in our Spring Boot main class, something like this..

application.properties:

Generally we will create property files for writing static values related to our application.

# How to Change Default Tomcat Server Port :

In Spring Boot, we can change tomcat’s port number in 2 ways…

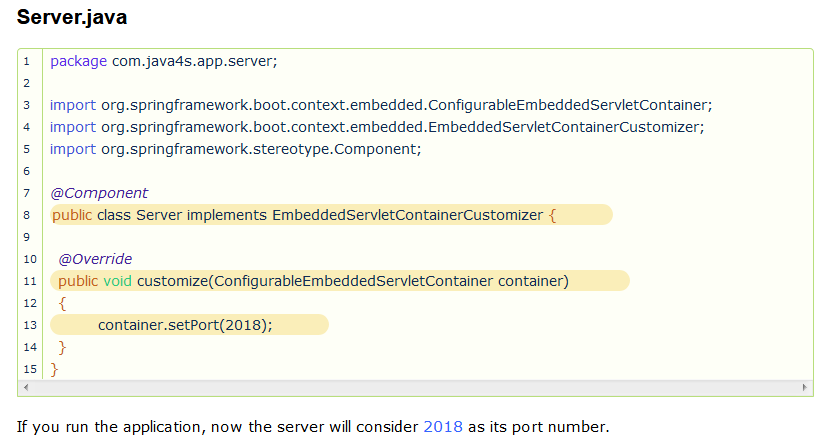
* Using application.properties
* Using Java code change

## application.properties

server.port = 2017

## Using Java code change

In this approach, we will create a simple java class which implements EmbeddedServletContainerCustomizer interface of Spring Boot, this is a strategy interface for customizing auto-configured embedded servlet containers, and we need to override customize() method of that interface that’s it, let me show you an example.



# How to Change Default Context Path:

what is this context path? -🡪 simply its our application name.

Generally while we are hitting any application in the browser, we will write the URL with the application name(context) right?

I mean…

*http://localhost:<port>/****<****application\_name or context\_path****>****/operation\_name*

But if you check [*Spring Boot RESTful Web Service Example*](https://www.java4s.com/spring-boot-tutorials/spring-boot-creating-a-restful-web-service-example/) we haven’t included any context path, we directly ran the application with the path we have given in @RequestMapping

Spring Boot by default consider the context path as ‘**/**‘ so we no need to give our application name or context path, but in real-time we should use some context path for the applications

In Spring Boot, we can change application default context path in two ways…

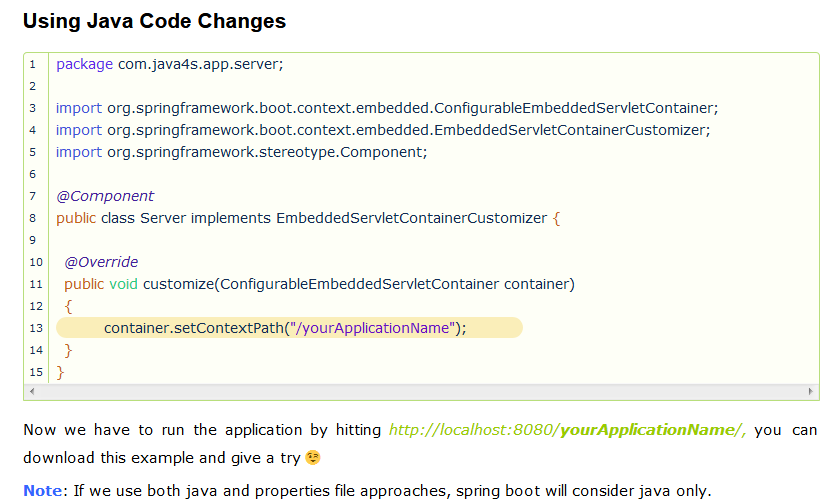
* Using applications.properties
* Using Java code changes

## Using application.properties

Create application.properties in your application src/main/resources and write this line..

server.contextPath=/yourApplicationName

## Using Java Code Changes



# How to Reload Changes Without Restarting the Server

In Spring Boot this can be achieved by adding a DevTools module, just add the following dependency in your Spring Boots pom.xml and build it.

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

Spring Boot DevTools module does exactly what developers needed, this eliminates the process of manually deploying the changes. DevTools will auto restart the server when we have changes. Spring team they haven’t included this feature in Spring Boot’s initial version, upon several request they added this feature later.

# Spring Boot JDBC + MySQL – How to Create/Configure a DataSource

We are all aware that the process of creating a traditional Spring JDBC application is little tedious because of its XML configurations and we need to follow few steps to configure any datasource. But believe me with Spring Boot creating a JDBC application is as easy as counting 1,2,3.. 🙂

**Steps to Create DataSource in Spring Boot Application**

* Add Spring Boot JDBC dependency in POM.xml
* Add datasource information in application.properties
* Get JDBCTemplate object in your DAO with @Autowired annotation and use it

…there is no point 4, that’t it

## 1. Add Spring Boot JDBC Dependency

In order to work with Spring Boot JDBC, first we need to add the following dependency in your applications POM.xml

<!-- Spring boot jdbc dependency -->

<dependency>

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

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

</dependency>

<!-- MySql dependency -->

<dependency>

     <groupId>mysql</groupId>

     <artifactId>mysql-connector-java</artifactId>

</dependency>

## 2. Add Datasource Information in application.properties

In real time applications, we will write the datasources information generally in the XML’s, I hope you all aware of that. In Spring Boot, rather than writing in XML’s just open your application.properties and add your datasource information to the Spring Boot’s predefined keys.

## application.properties

# Applicationn context name

server.contextPath=/springbootds

# Here 'test' is the database name

spring.datasource.url=jdbc:mysql://localhost/test

spring.datasource.username=java4s

spring.datasource.password=java4s

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

With this we have configured the datasource in our spring boot application.

## 3. Get JDBCTemplate object in your DAO with @Autowired annotation

Go to your DAO class and get the object of JdbcTemplate by @Autowired annotation and use it. Spring Boot will automatically get the datasource details from application.propeties and injects to jdbcTemplate object while auto wiring.

@Autowired

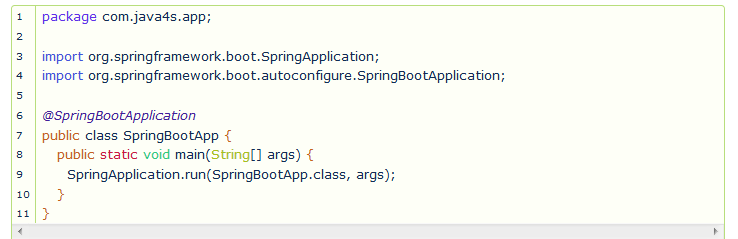
private JdbcTemplate jdbcTemplate;

**Required files**

* pom.xml



* **SpringBootApp.java**





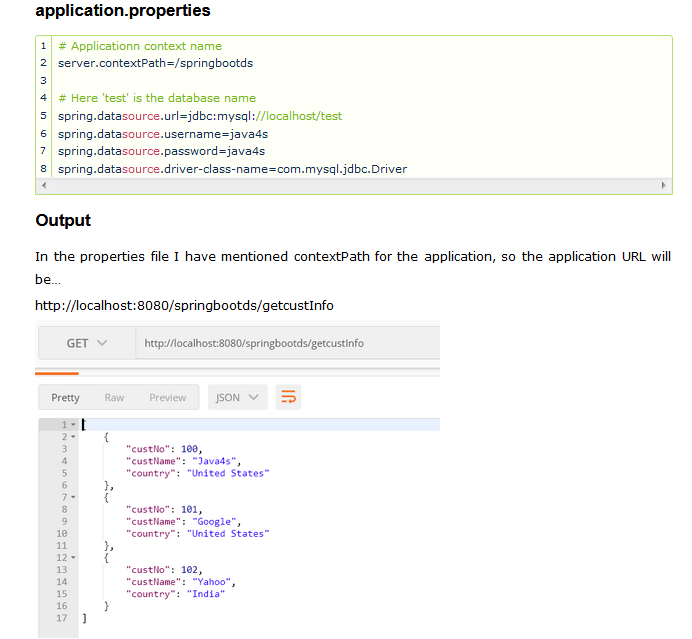
* **SpringJava4sDAO.java**



**Customer.java**



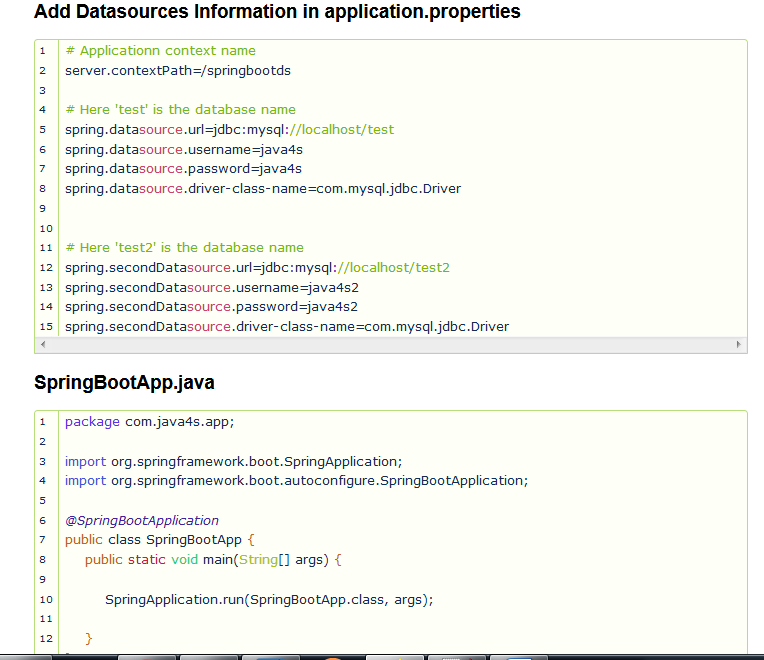
**application.propeties :**



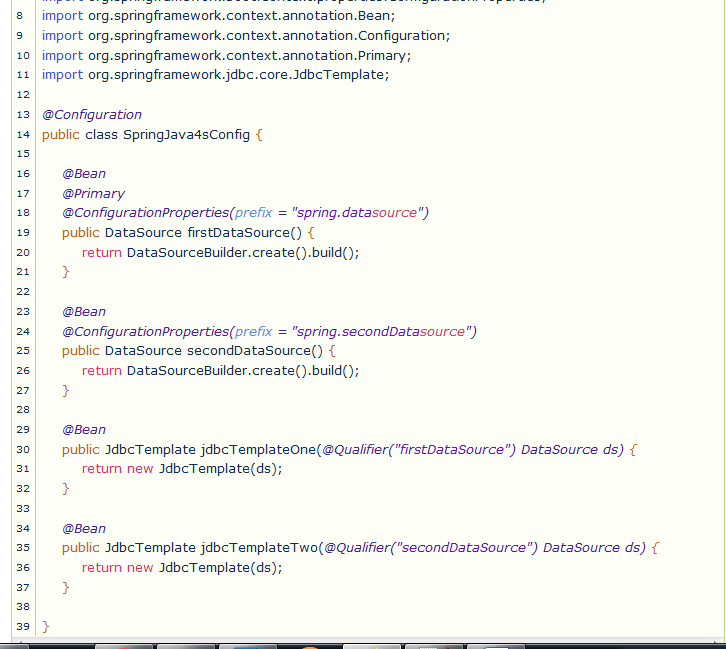
# Spring Boot JDBC + MySQL – How to Configure Multiple DataSource :

I will show you how to configure multiple datasources in spring boot application. Unlike single datasource, in order to create multiple datasources we may need to write little configuration,





**SpringJava4sConfig.java**



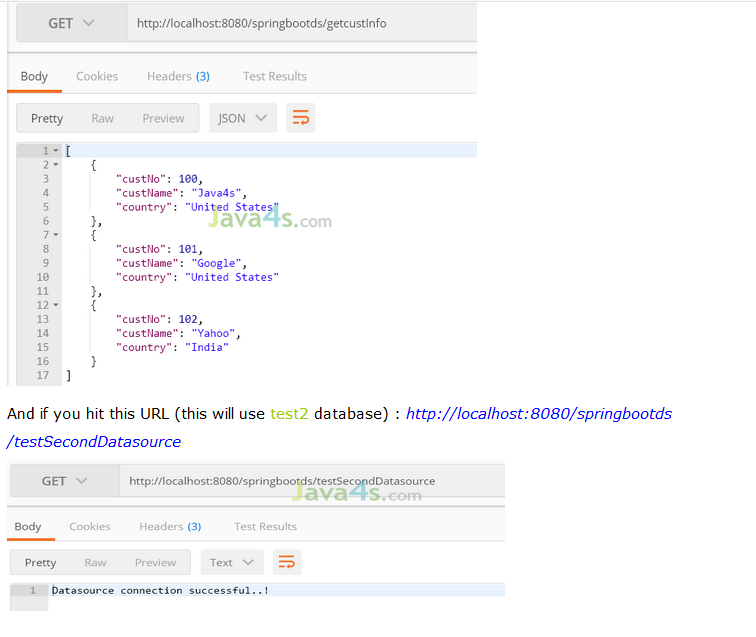






## Output:

If you run the app and hit the URL (this will use test database) : http://localhost:8080/springbootds/getcustInfo



# How to Deploy Spring Boot Applications on External Tomcat Server

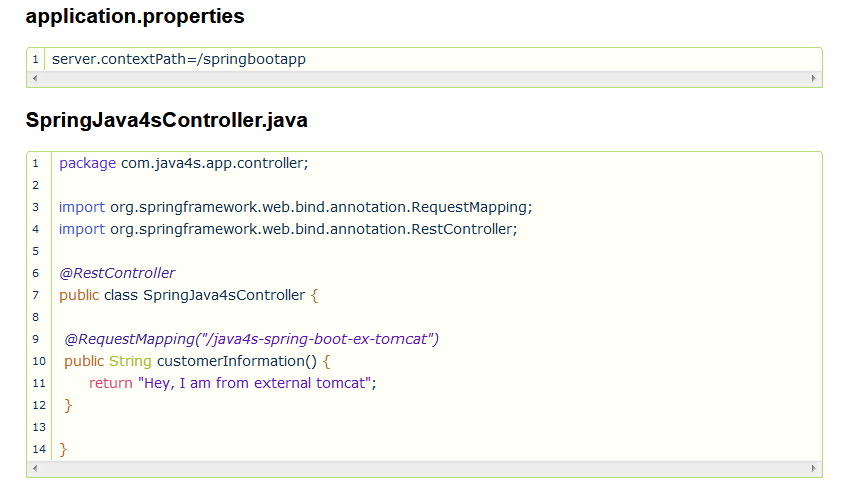
Generally in the real-time projects we wont use inbuilt servers provided by the frameworks because of many reasons like security, maintenance and control. So in this article I will show you how can we deploy the spring boot applications on external servers (in this tutorial I am going to consider the external server as Tomcat).

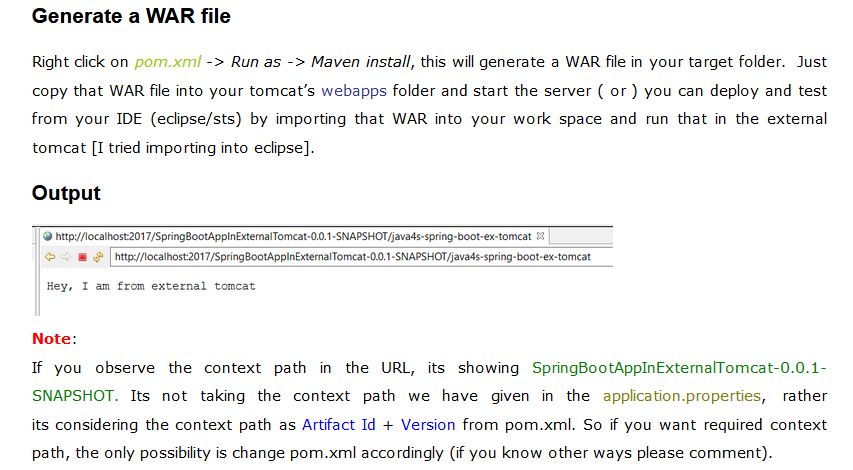
Just do these changes to your spring boot application which you want to deploy it on to external tomcat server.

* pom.xml, add dependency and packaging to war
* Extend your main class with SpringBootServletInitializer and override its *configure* method
* Generate WAR and deploy into the external server









# Spring Boot + Spring Security – RESTful Web Service with basic Authentication

I am going to explain you how to implement basic authentication for RESTful web services using Spring Boot and Spring Security. We will need to create a java file with spring security

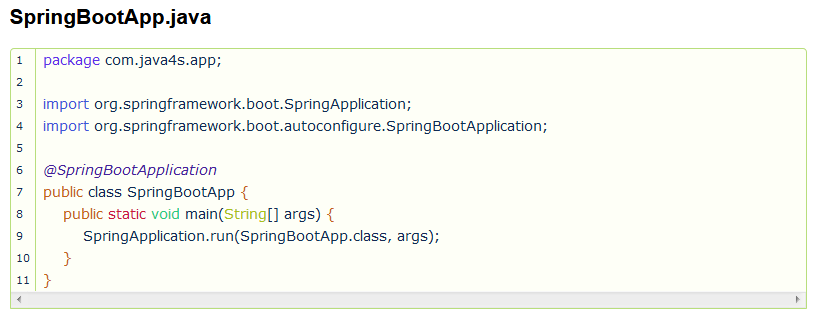
## Required Dependency

<dependency>

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

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

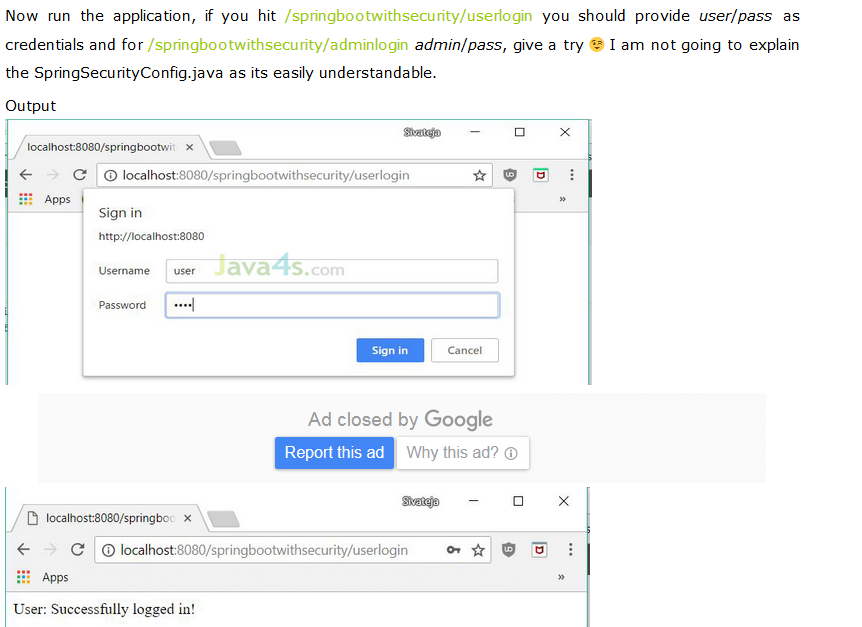
</dependency>





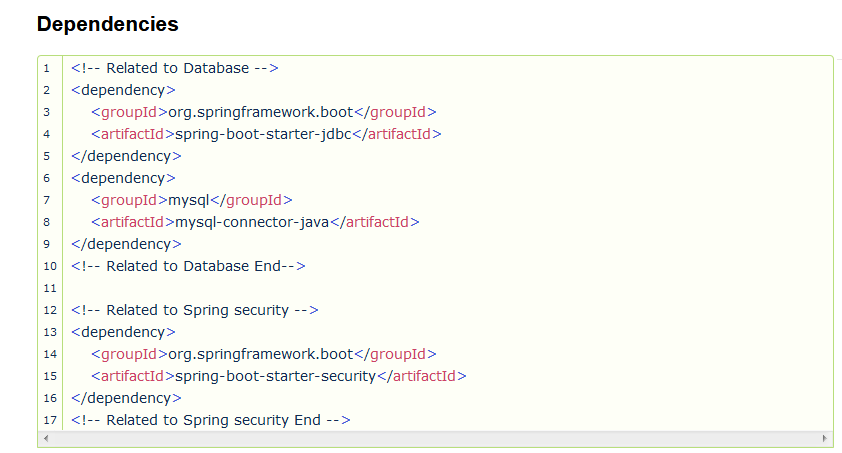
*// Authentication : set user/password details and mention the role***protected void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  
 auth.inMemoryAuthentication().passwordEncoder(org.springframework.security.crypto.password.NoOpPasswordEncoder.getInstance())  
 .withUser(**"user"**).password(**"pass"**).roles(**"USER"**)  
 .and()  
 .withUser(**"admin"**).password(**"pass"**).roles(**"USER"**, **"ADMIN"**);  
}



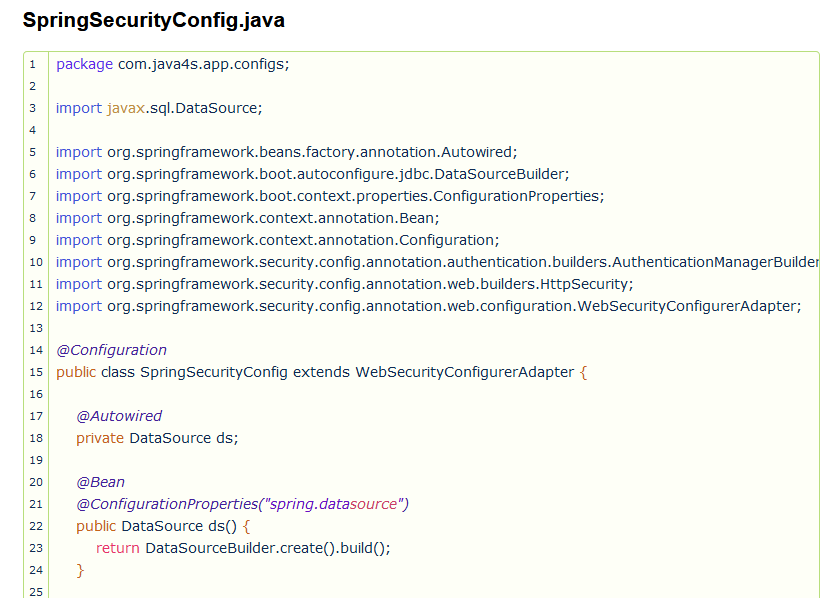


# Spring Boot + Spring Security – RESTful Web Service with Database Authentication

describes how to implement database authentication for your RESTful web services using Spring Boot and Spring Security. Let me start with the required dependencies..









Note :

FYI.

CREATE TABLE Users (

id int,

userName varchar(255),

email varchar(255),

userPassword varchar(255),

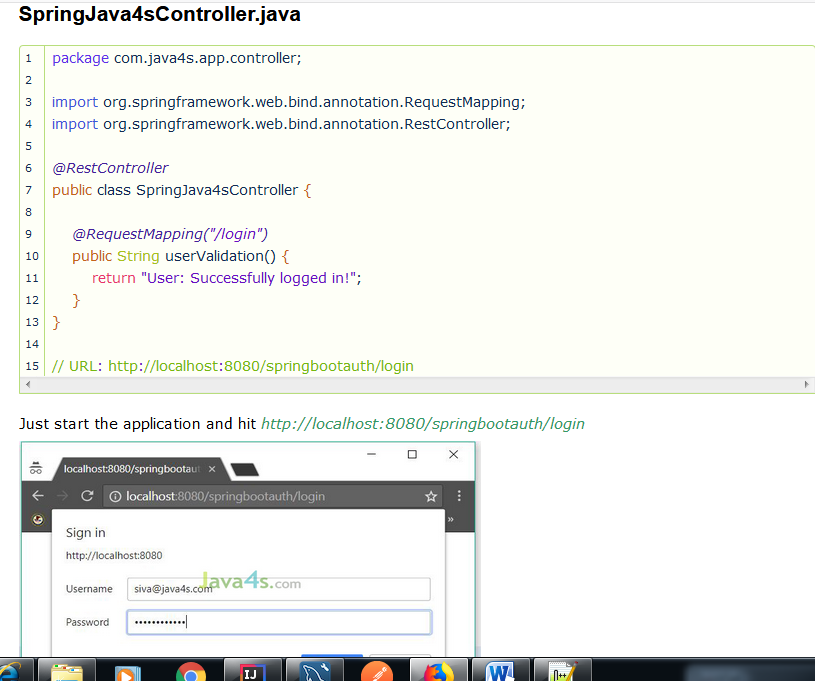
role varchar(10),

created timestamp

);

insert into users (id, userName, email, userPassword, role, created) values(1,'java4s', 'java4s@java4s.com', 'java4spassword','ADMIN', CURRENT\_TIMESTAMP)

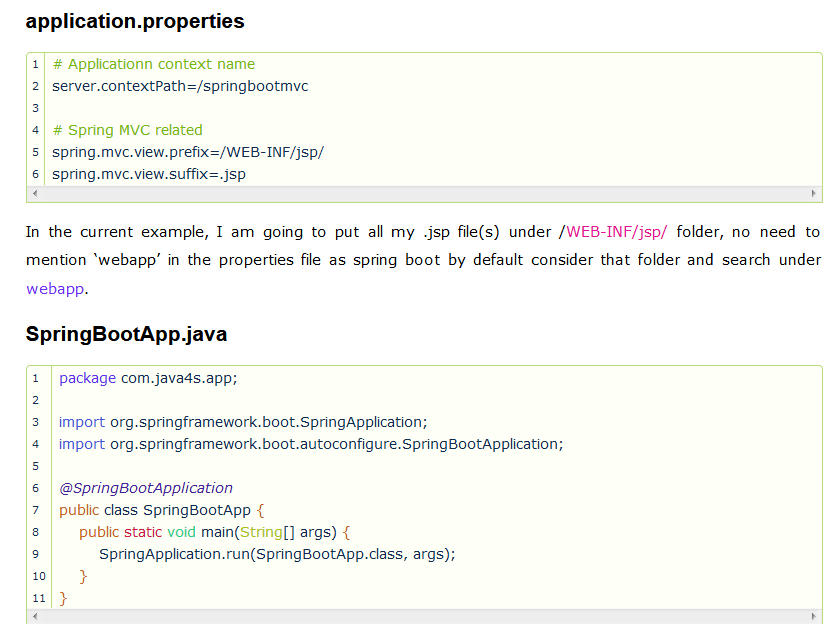
insert into users (id, userName, email, userPassword, role, created) values(1,'siva', 'siva@java4s.com', 'sivapassword','USER', CURRENT\_TIMESTAMP)

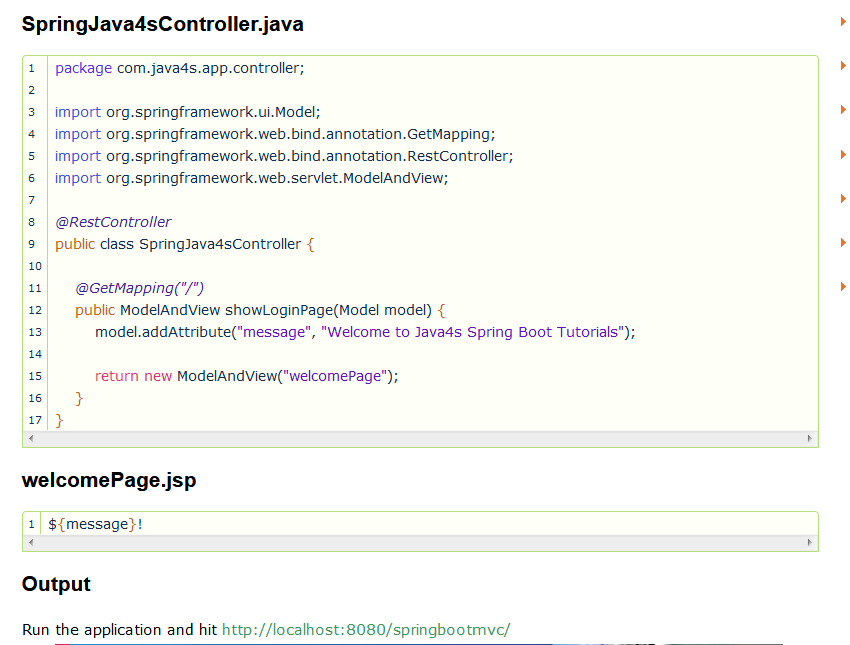


# Spring Boot + Spring MVC + JSP Hello World Example

we have to create a webapp folder under /src/main (src > main > webapp) where we will place all our .jsp files.







# Spring Boot – Example of RESTful Web Service with XML Response

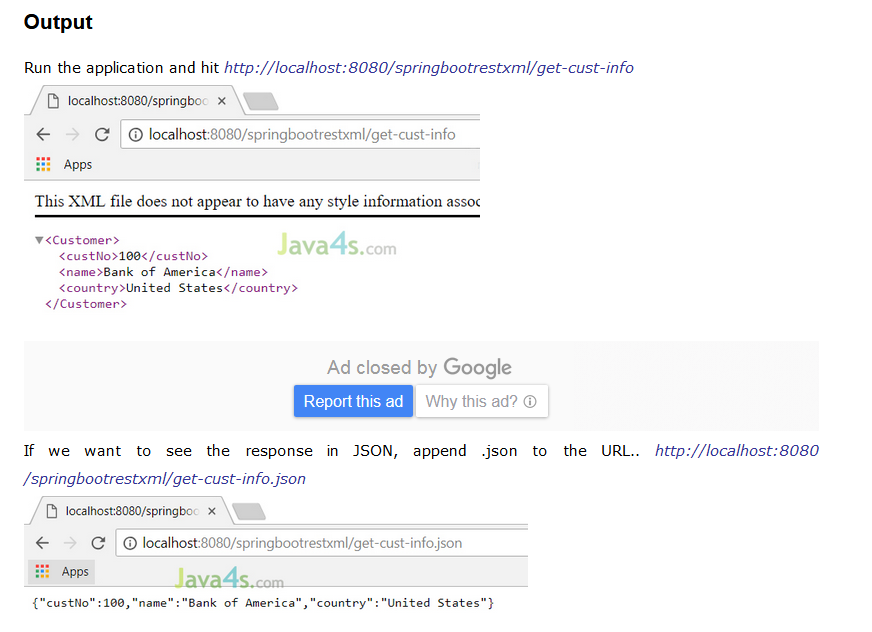
Spring boot services by default gives the response in JSON format, but we can reverse this functionality in such a way that the default response will be in XML.  In order to do that we have to add a new dependency called jackson-dataformat-xml.  With this dependency services by default gives the response in XML format and if you want to see the response in JSON, just append .json to the URL that’s it 🙂 I will show you with an example.





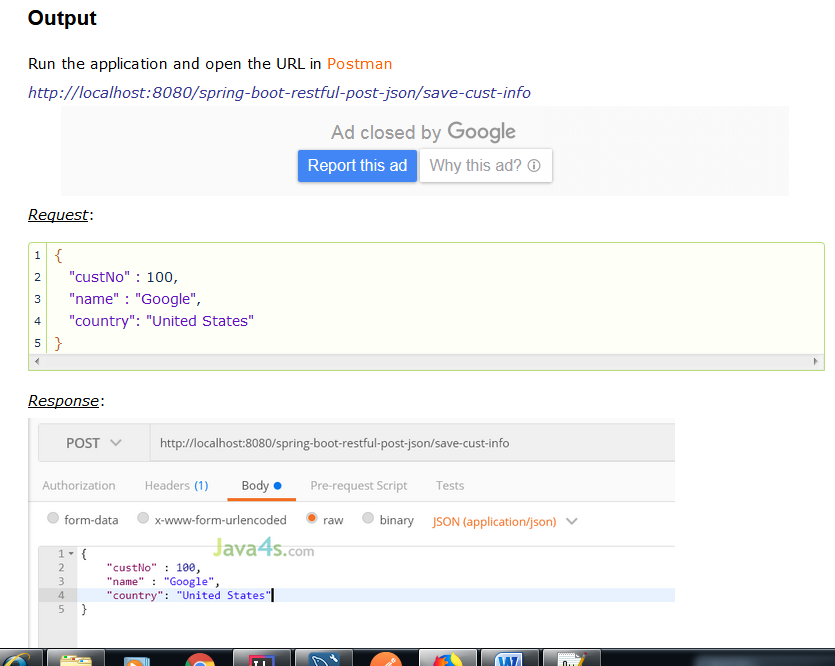


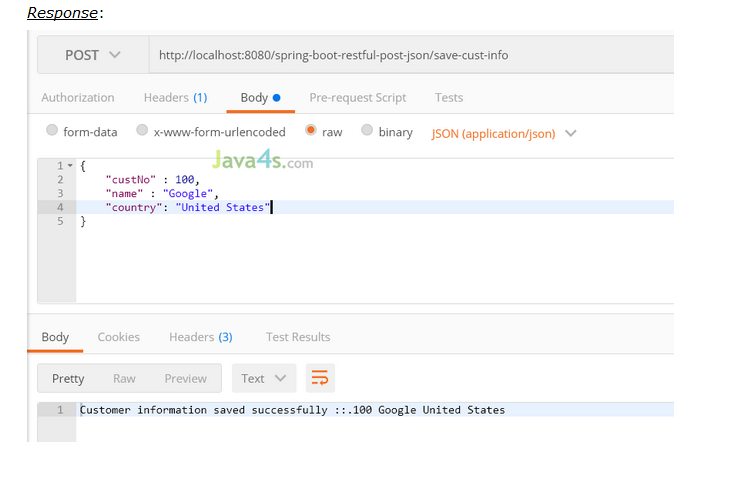




# Spring Boot – RESTful Web Service with POST Request in JSON Example







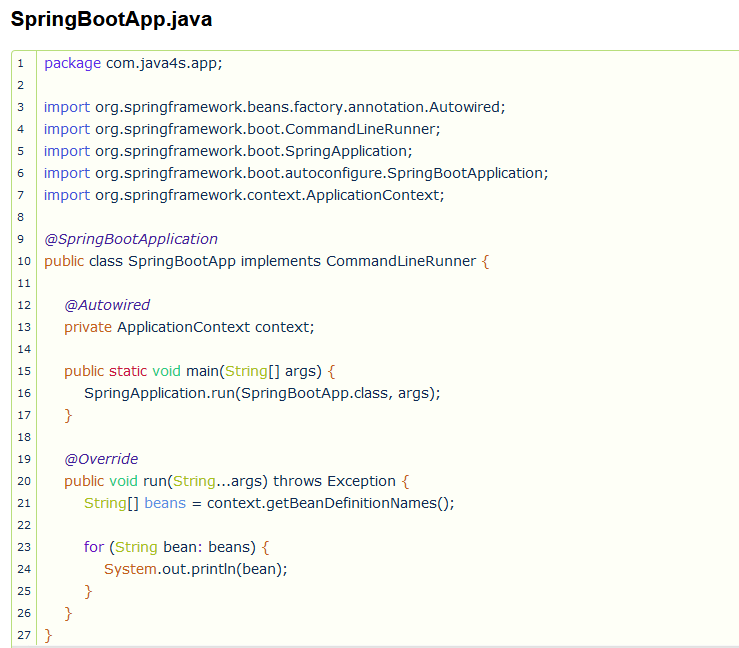
# Spring Boot – Display All Beans Available in ApplicationContext

how to see the beans that are loaded by the Spring Boot from the ApplicationContext. we have to do is implement main class with CommandLineRunner/ApplicationRunner interface and override its run method.

CommandLineRunner/ApplicationRunner’s run() method will get execute right after ApplicationContext is created and before Spring Boot application initialized. This run() method will execute only once in an application’s life cycle.  So what’s the difference between CommandLineRunner/ApplicationRunner? Basically, both will do the same trick, the only difference is CommandLineRunner’s run() method will accept String array and ApplicationRunner’s run() will accept ApplicationArguments as arguments.

Steps to configure cache in spring boot applications..

* In pom.xml add spring cache dependency *spring-boot-starter-cache module*
* Enable cache in spring boot application by writing the *@EnableCaching* annotation to the main class
* Add @Cacheable annotation to the method which you would like to cache the result



# How to Configure Cache in Spring Boot Applications

Caching helps to increase the performance of the application by reducing number of round trips between the database or any expensive resources. In real time we will face the scenarios like we have to execute heavy database query and lets say the data in the database will change very rarely, for this kind of scenarios its not a good idea to hit the database for every call, rather just cache the result at the first time when it hits the database and return the same data again for the other calls.

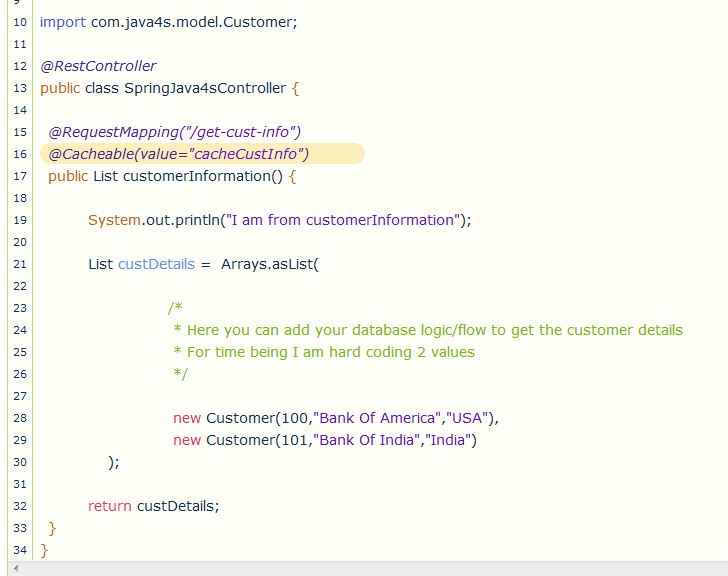
Steps to configure cache in spring boot applications..

* In pom.xml add spring cache dependency *spring-boot-starter-cache module*
* Enable cache in spring boot application by writing the *@EnableCaching* annotation to the main class
* Add @Cacheable annotation to the method which you would like to cache the result



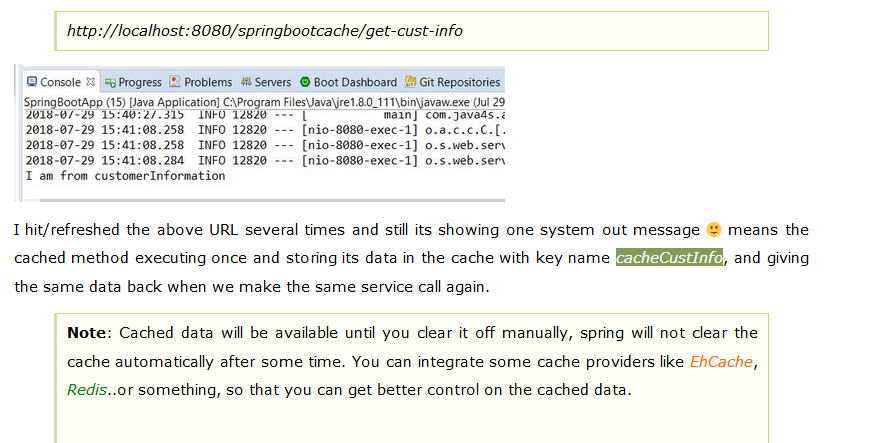


## SpringJava4sController.java



## Output

Run the application at.. <http://localhost:8080/springbootcache/get-cust-info>



## Clear the Cache

Generally from our side we will clean or flush the cache after any update or delete operations, for that we use @CacheEvict annotation on the required method.

@CacheEvict(value = "cacheCustInfo", allEntries=true) // @CacheEvict will clear the cache when delete any customer info from the database.

public void removeEmployee(Id customerId) {

//Database logic will go here to remove the particular customer from the DB.

}

## How to Disable cache

If we would like to disable the cache, no need to remove all the annotations 🙂 just add the below line in the application.properties file, it takes care everything for you.

spring.cache.type=none

# Spring Boot Configure DataSource Using JNDI with Example

We already saw the default approach to configure datasource, in this article I am going to explain you how to configure datasources using JNDI lookup in spring boot applications.