

-------------------------------Day1-----------------------------

Demo1:standalone app with 4 layers

1.create a model :

com.atossyntel.cms.model.Cutomer

2.create data source:

com.atossyntel.cms.dataAccess.CustomerMap

3.create data access layer :

com.atossyntel.cms.CutomerDao<interface>

com.atossyntel.cms.CutomerDaoImple<Class which implements the interface>

4.create service layer: com.atossyntel.cms.CustomerService<interface>

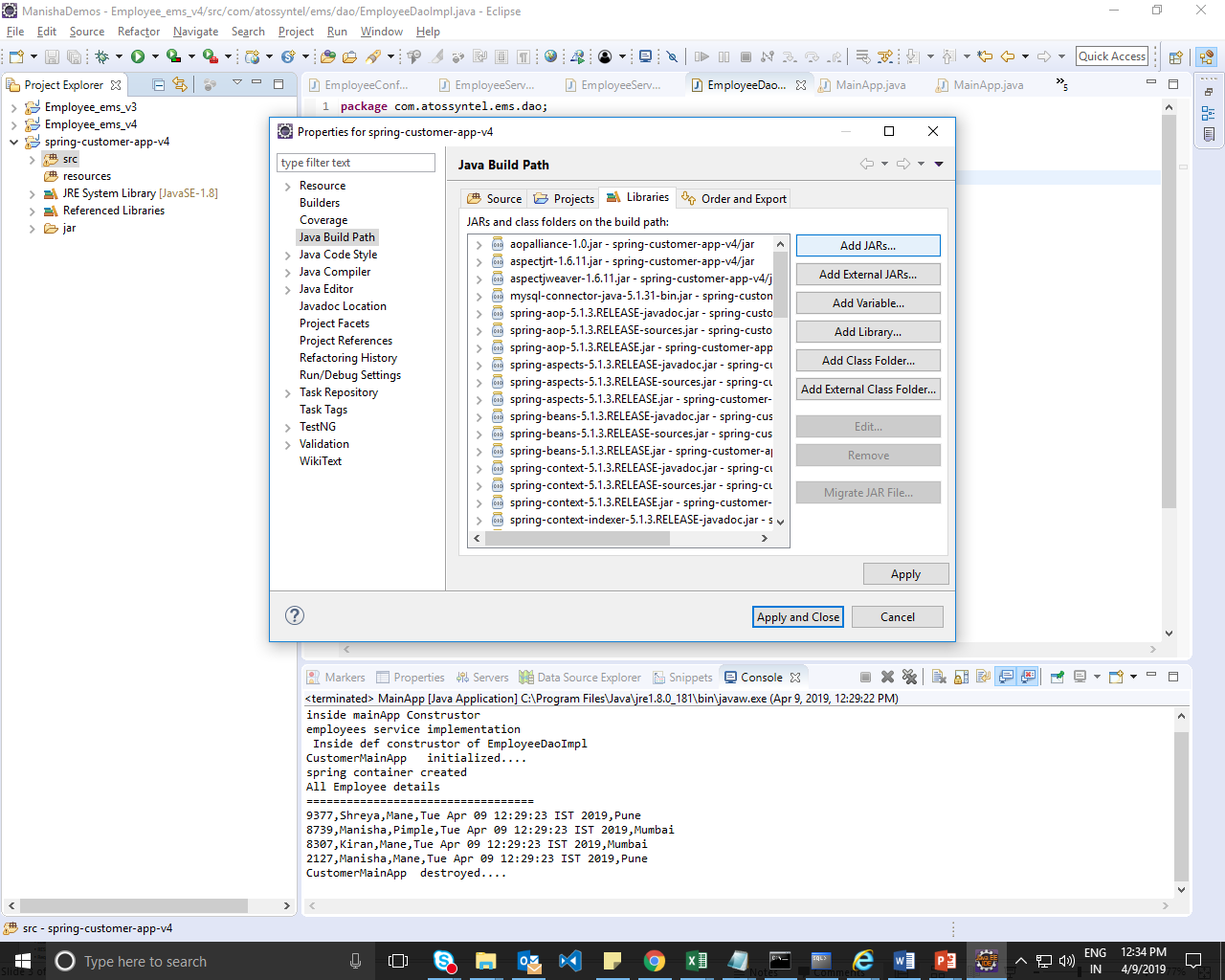
com.atossyntel.cms.CustomerServiceImplClass which implements the interface>

5:create the presentaion Layer:

com.atossyntel.cms.presentaion.MainApp

Demo2:same demo with spring dependency injection

Note : create a new source folder as jar copy all jar files in this folder so that while sharing along with folder jar’s will also be shared





Select Add Jar 🡪select jar files from jar

explain lazy-init

explain scope

explain generic class

init and destroy method

refer mainapp.java

autowiring byname bytype(setter injection)

refer mainapp2.java

byType:if we create 2 bean object then we get an error

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'mainapp' defined in class path resource [beans-anno.xml]: Initialization of bean failed; nested exception is org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'employeeService' defined in class path resource [beans-anno.xml]: Unsatisfied dependency expressed through bean property 'employeeDao': : No qualifying bean of type [com.atossyntel.ems.dao.EmployeeDao] is defined:

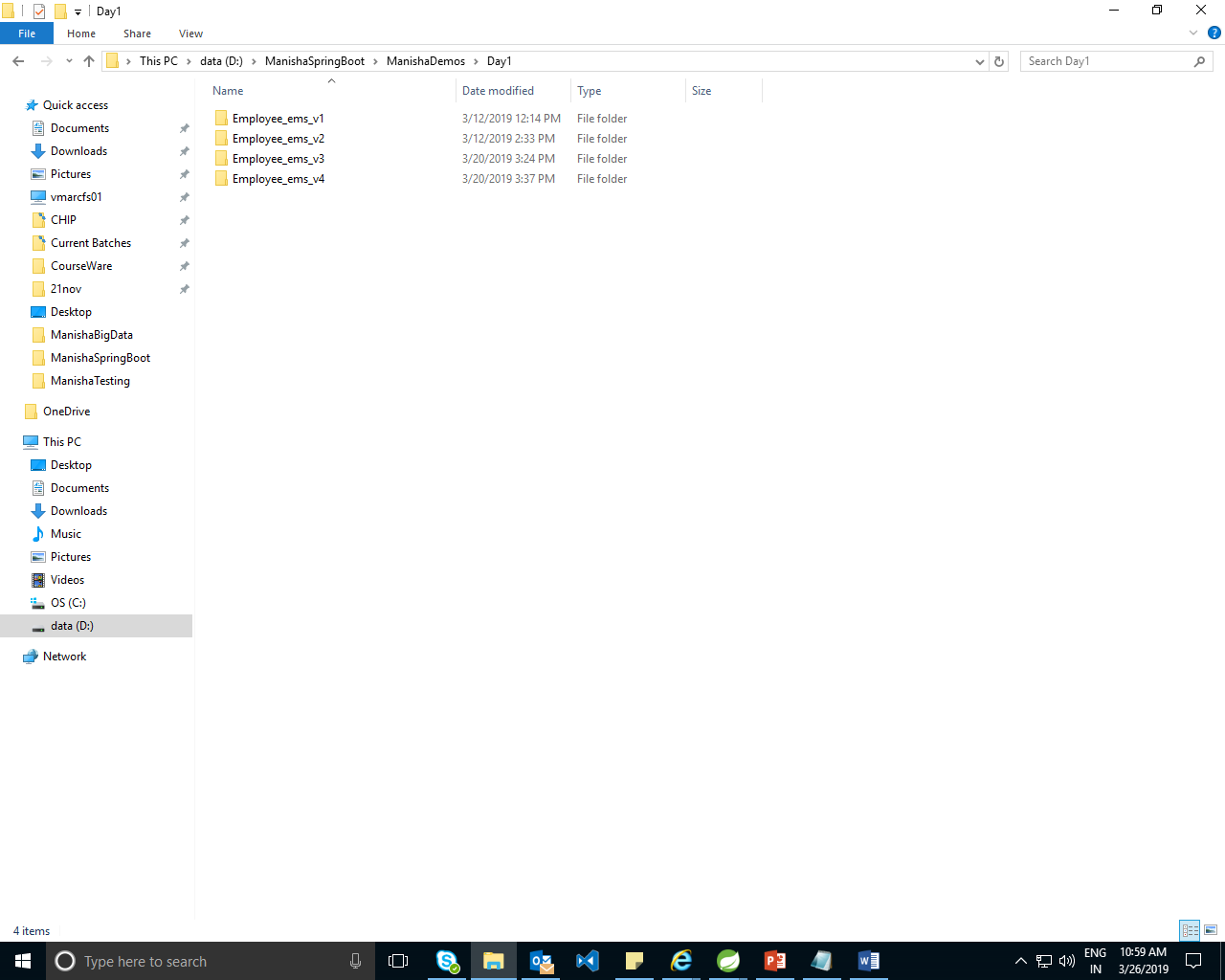
Both will use setter injection

If we use autowiring with constructor then constructor injection will exec.

Demo3: with annotation

Demo 4: without xml(D:\ManishaSpringBoot\ManishaDemos\Day2\Employee\_ems\_v4)

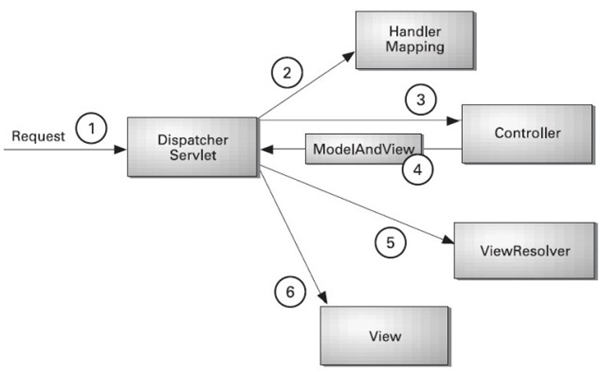
D:\ManishaSpringBoot\ManishaDemos\Day1



Examples:

--------------------------------Day2----------------------------

MVC Demo1:



Request mapping can be done at class level or at method level.

Load-on-startup:

* When we configured the servlet with Load-on-startup the servlet will get initialized and loaded at the startup of web application.
* Generally the instance is getting created when we send a request but since we are using DiaptcherServlet as frontcontroller ,it need to be eagerly loaded ie need to be initialized before any request comes .
* We can have any no of dispatcher servlet

Demo1: Add 3 controllers

helloController:

**package** com.controller;

**import** org.springframework.stereotype.Controller;

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

@Controller

**public** **class** helloController {

helloController()

{

System.***out***.println("=======HelloConstroller Constructor=======");

}

@RequestMapping("/hello")

**public** String hello()

{

System.***out***.println("#####inside hello Method");

**return** "hello";

}

}

welcomeController:

@Controller

**public** **class** WelcomeController {

WelcomeController()

{

System.***out***.println("----inside welcome ---------");

}

@RequestMapping(value="/welcome",method=RequestMethod.***GET***)

**public** ModelAndView welcome()

{

System.***out***.println("In welcome controller method");

**return** **new** ModelAndView("welcome","message","Welcome to first demo on spring web mvc");

}

}

greetController:

**package** com.controller;

**import** java.util.Date;

**import** org.springframework.stereotype.Controller;

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

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

**import** org.springframework.web.servlet.ModelAndView;

@RequestMapping(value="/greet")

@Controller

**public** **class** GreetController {

**public** GreetController() {

// **TODO** Auto-generated constructor stub

System.***out***.println("-----Greet Controller created");

}

@RequestMapping

**public** ModelAndView greet()

{

System.***out***.println("In GreetContrller greet method");

**return** **new** ModelAndView("greet","message","Today Is "+**new** Date());

}

@RequestMapping("/today")

**public** @ResponseBody String today()

{ System.***out***.println("In GreetContrller today method");

**return** "Today is"+**new** Date();

}

}

Step 2:

Configure the dispatcher servlet in web.xml file

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/spring/\*</url-pattern>

</servlet-mapping>

</web-app>

Step 3: add xml with name spring-servlet.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd"*>

<context:component-scan base-package=*"com"*/>

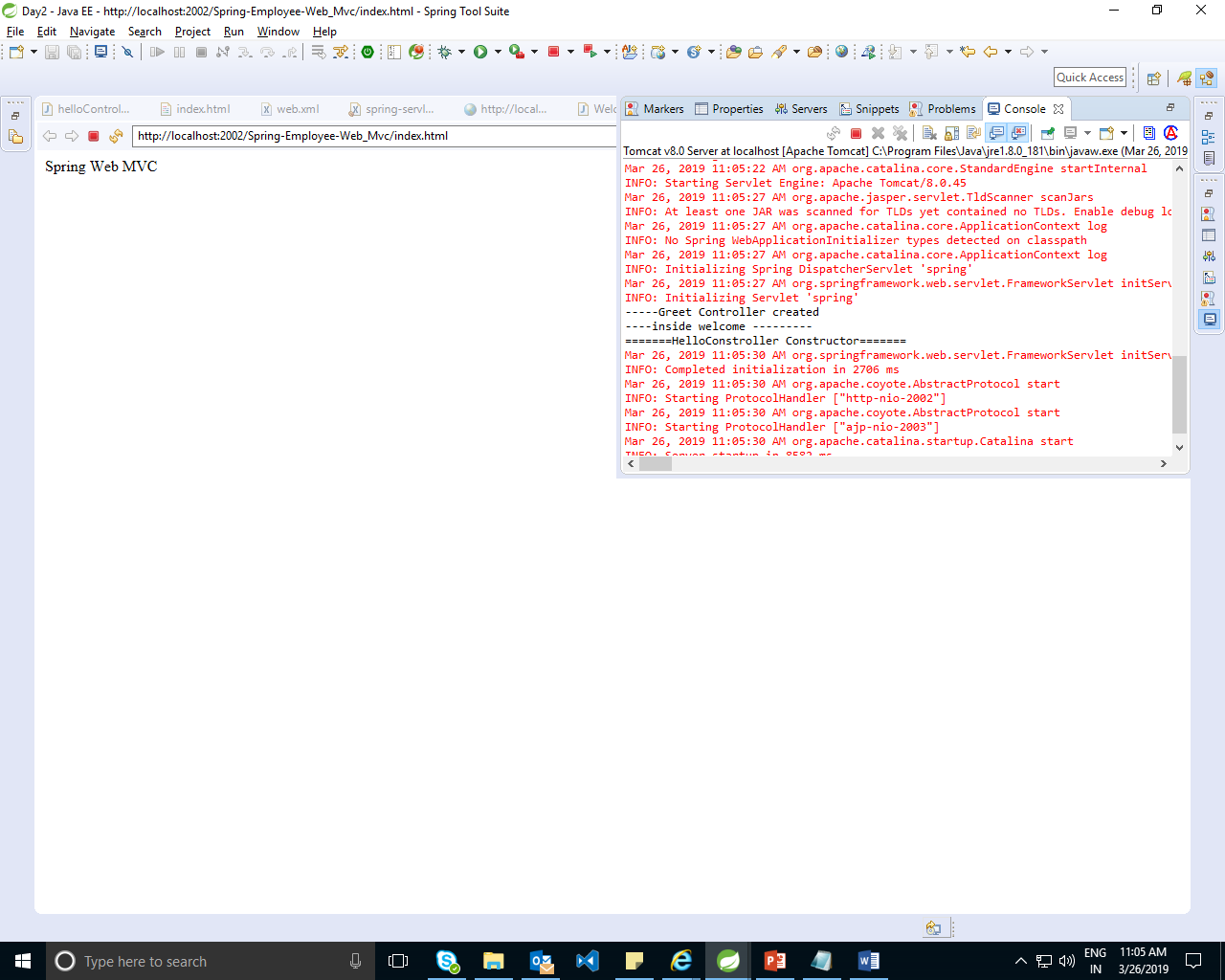
<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/views/"*></property>

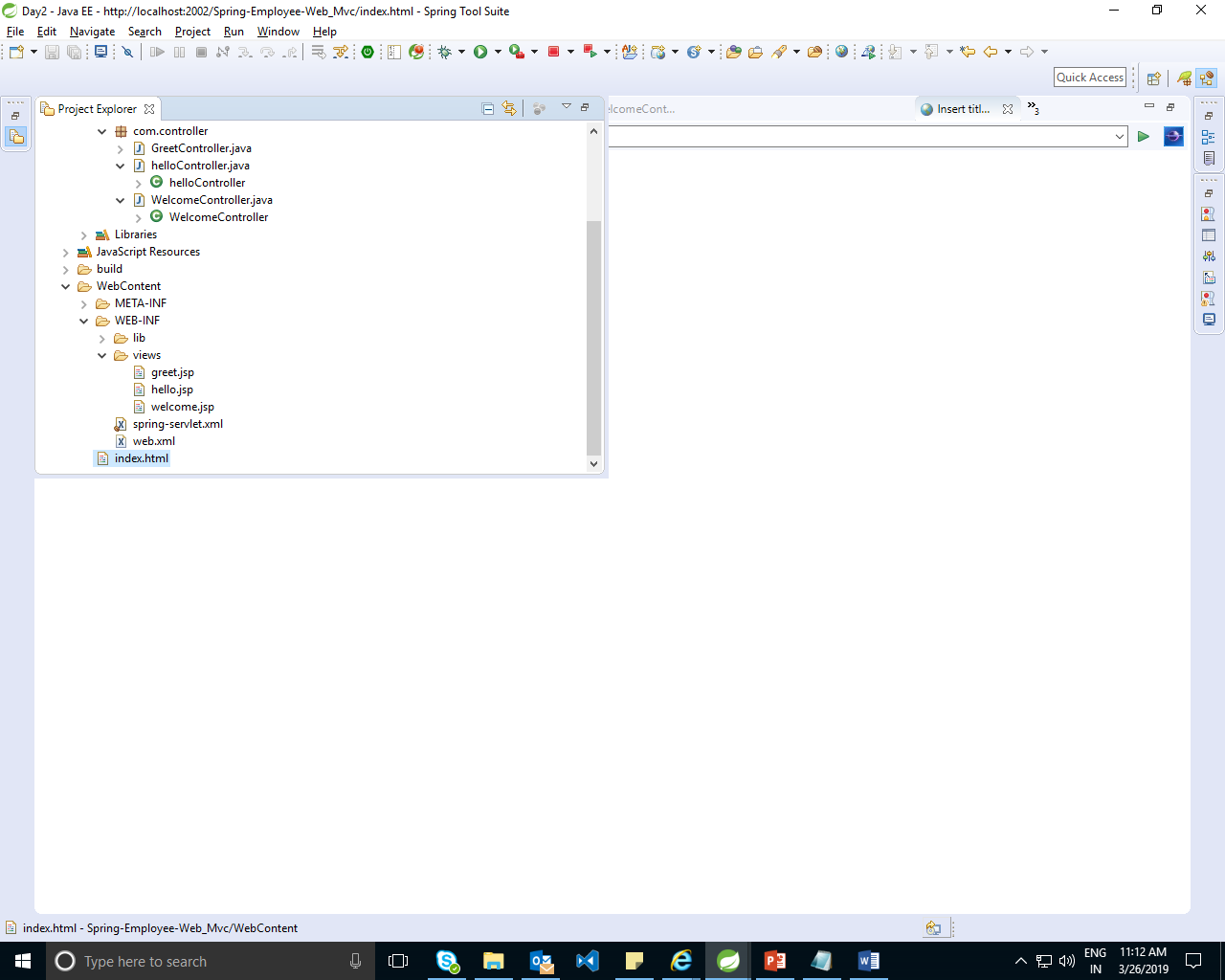
<property name=*"suffix"* value=*".jsp"*></property></bean>

</beans>

step 4: add index.jsp



Step 5: create new folder views



Hello.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>

Welcome ....

</body>

</html>

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>

${message}

</body>

</html>

Greet.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>

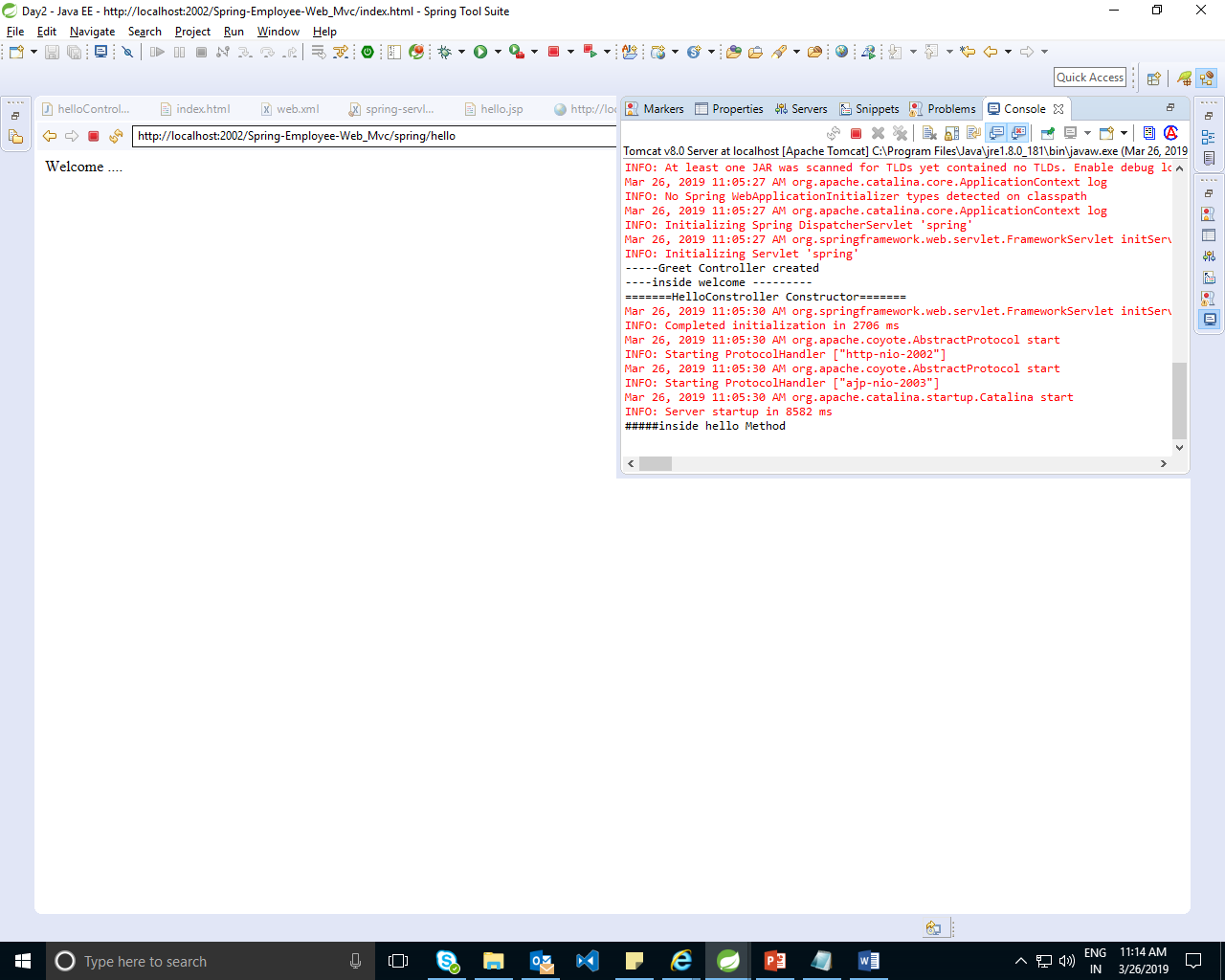
<body>

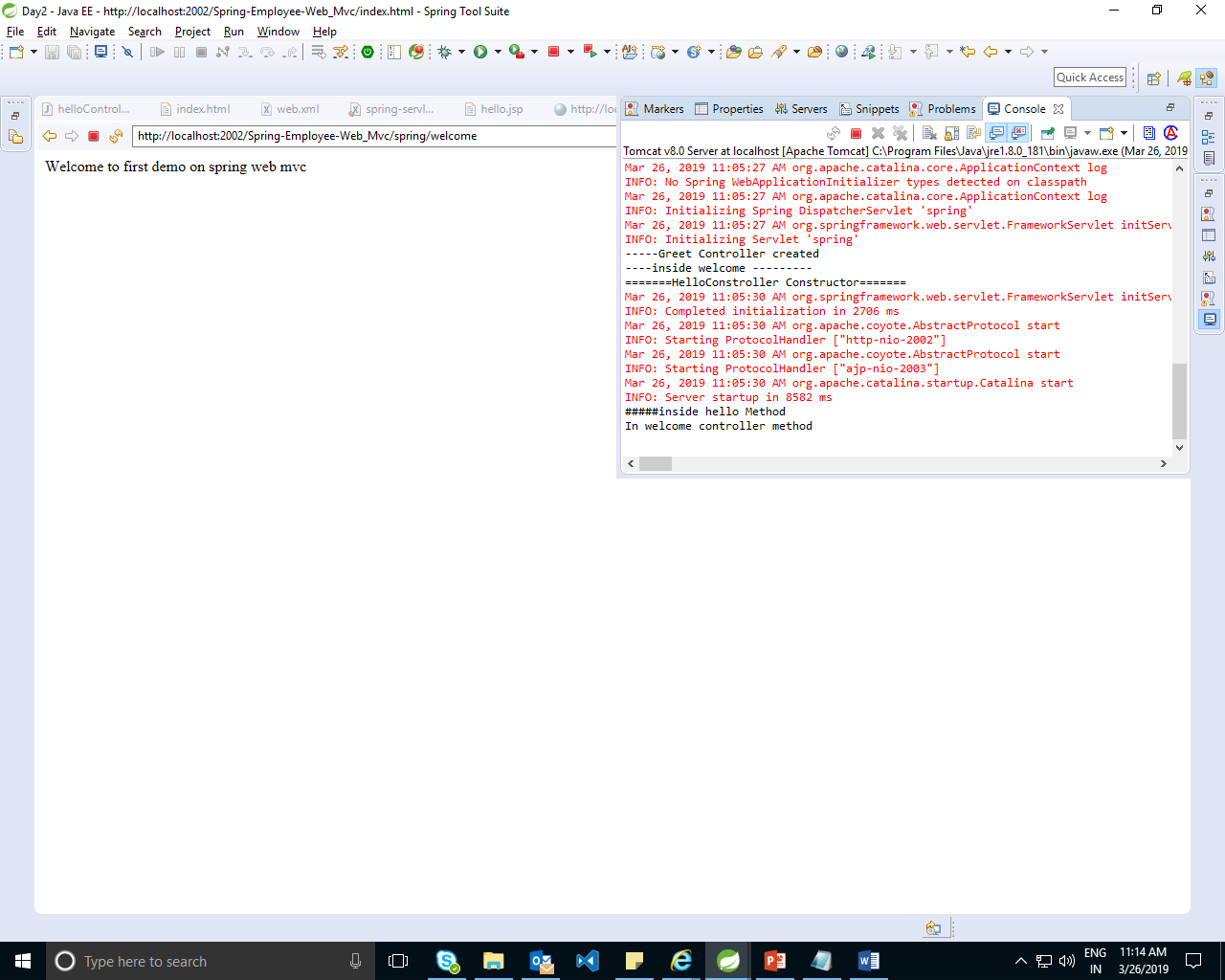
${message}

</body>

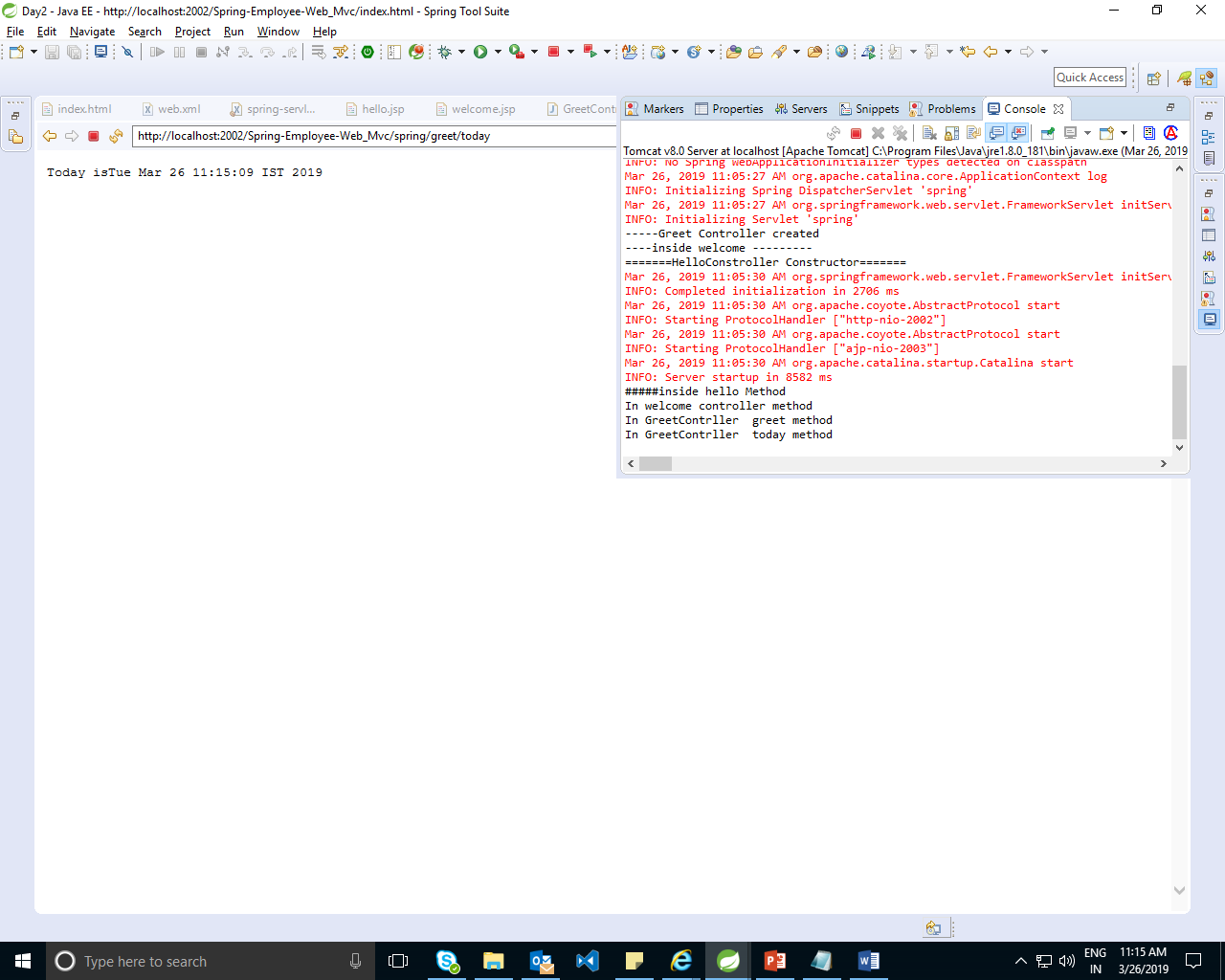
</html>

Execution:









D:\ManishaSpringBoot\ManishaDemos\Day2\Employee-ems-mvc-demo1

Demo2: MVC with 2 controller spring and rest

D:\ManishaSpringBoot\ManishaDemos\Day2\Employee-ems-mvc-demo2

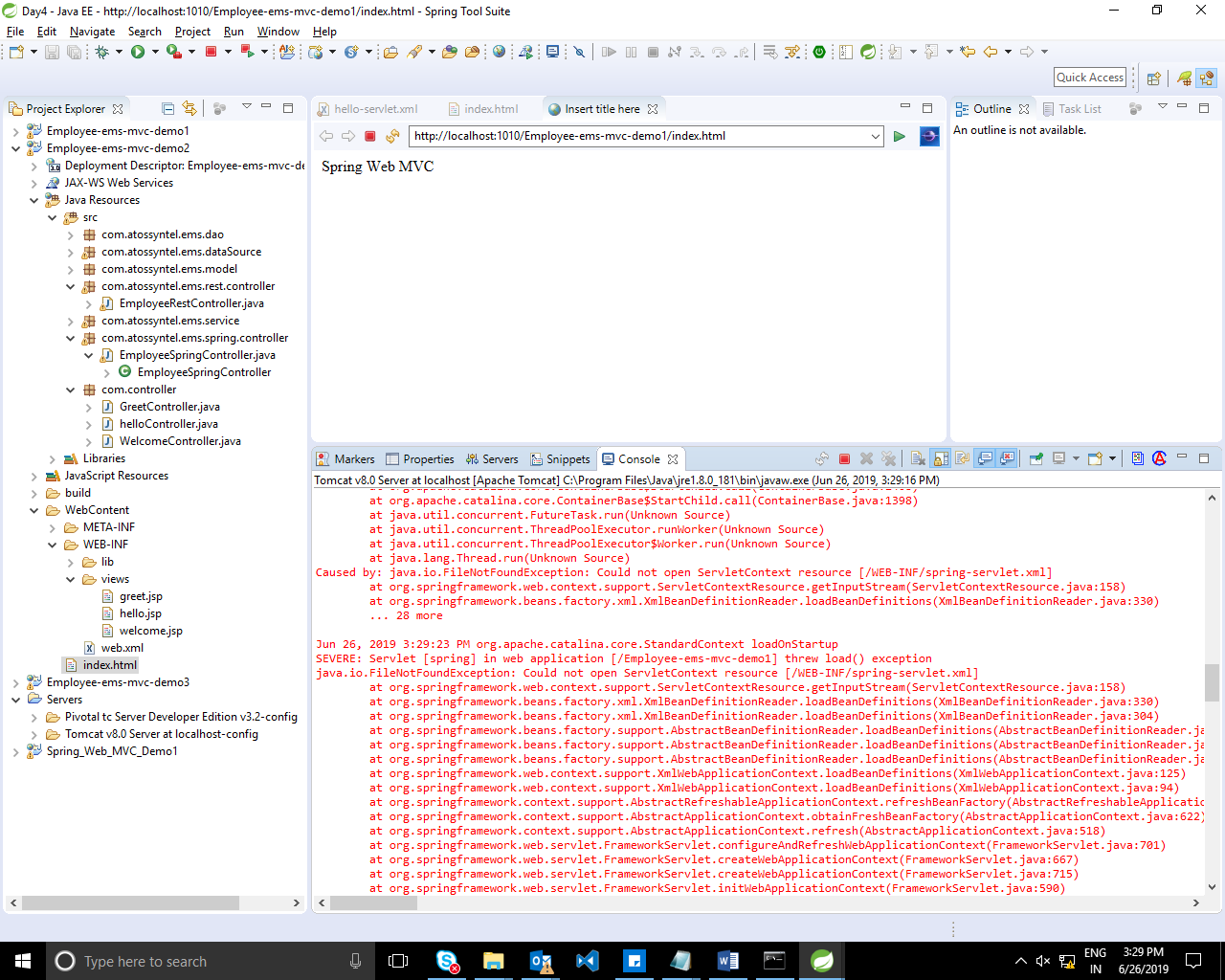
We are creating two controller one for spring and one for restfull api

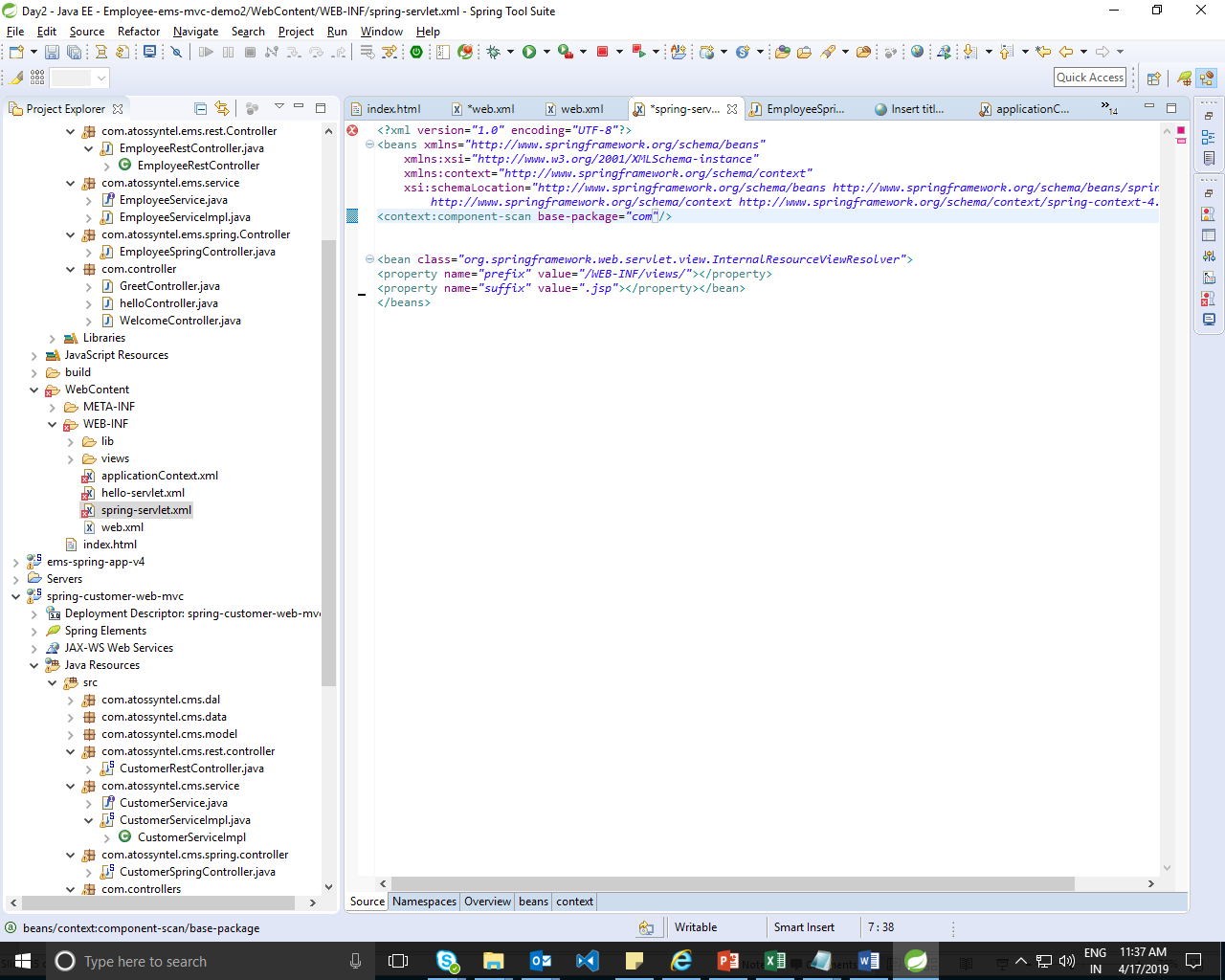
Spring always returns JSP

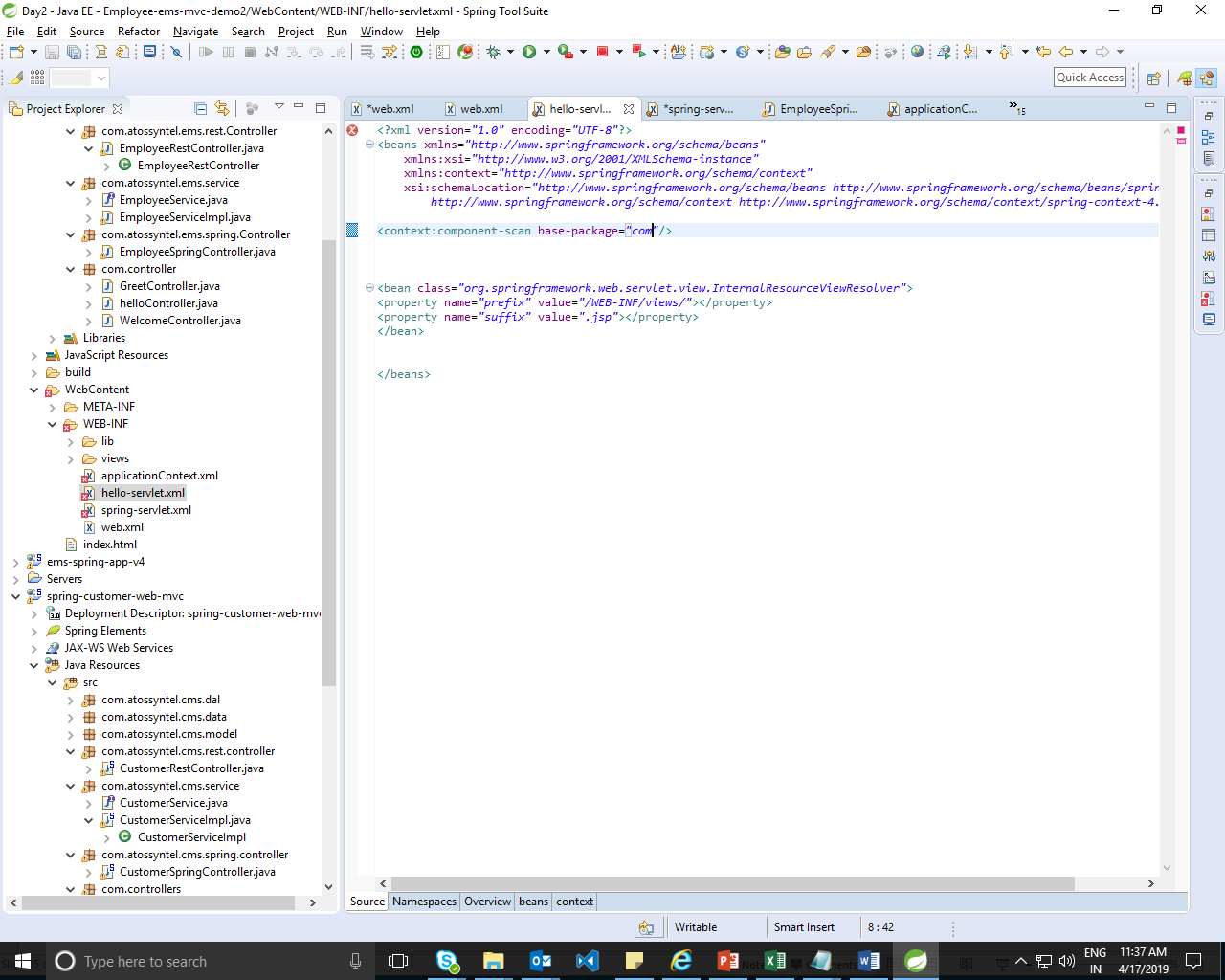
Rest always returns Content.

Copy src from day1 demo3 to current project. Rename presentation pkg to empSpringController and copy same as empRestController . Rename MainApp to EmpSpringController and EmprestController.

Show demo without the spring-servlet.xml and hello-servlet.xml it will throw an IOException.







Container will search rest-servlet.xml butwe are using different file name as hello-servlet.xml so to configure the change add init-param tag in hello-servlet.xml configuration as shown above.and this configaurtion will help us to load multiple files.

In both servlet the component scan pkg is com.

Add rest servlet detail in web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>Employee-ems-mvc-demo1</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

<welcome-file>index.jsp</welcome-file>

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

<welcome-file>default.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/spring/\*</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>rest</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/hello-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

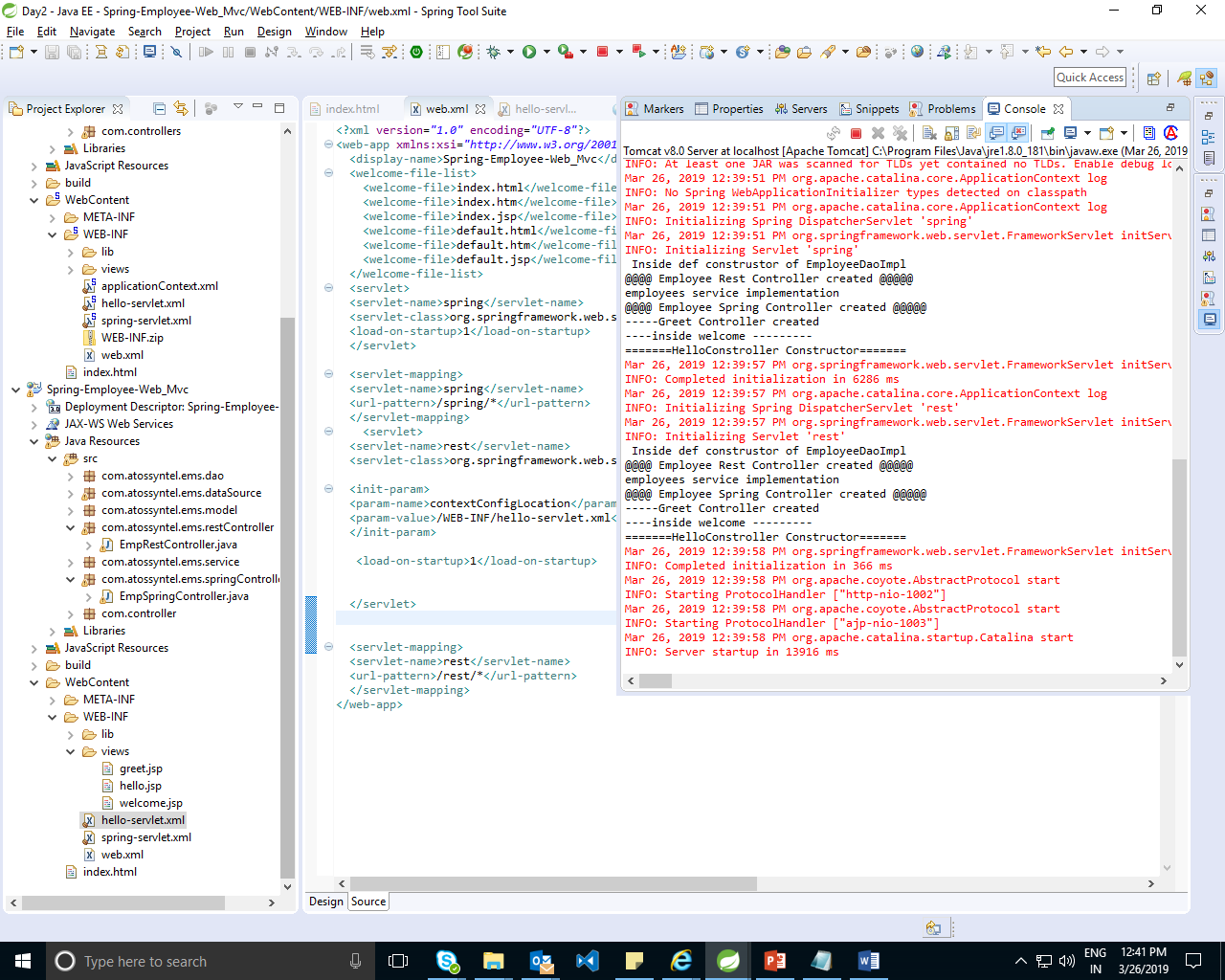
<servlet-name>rest</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

</web-app>

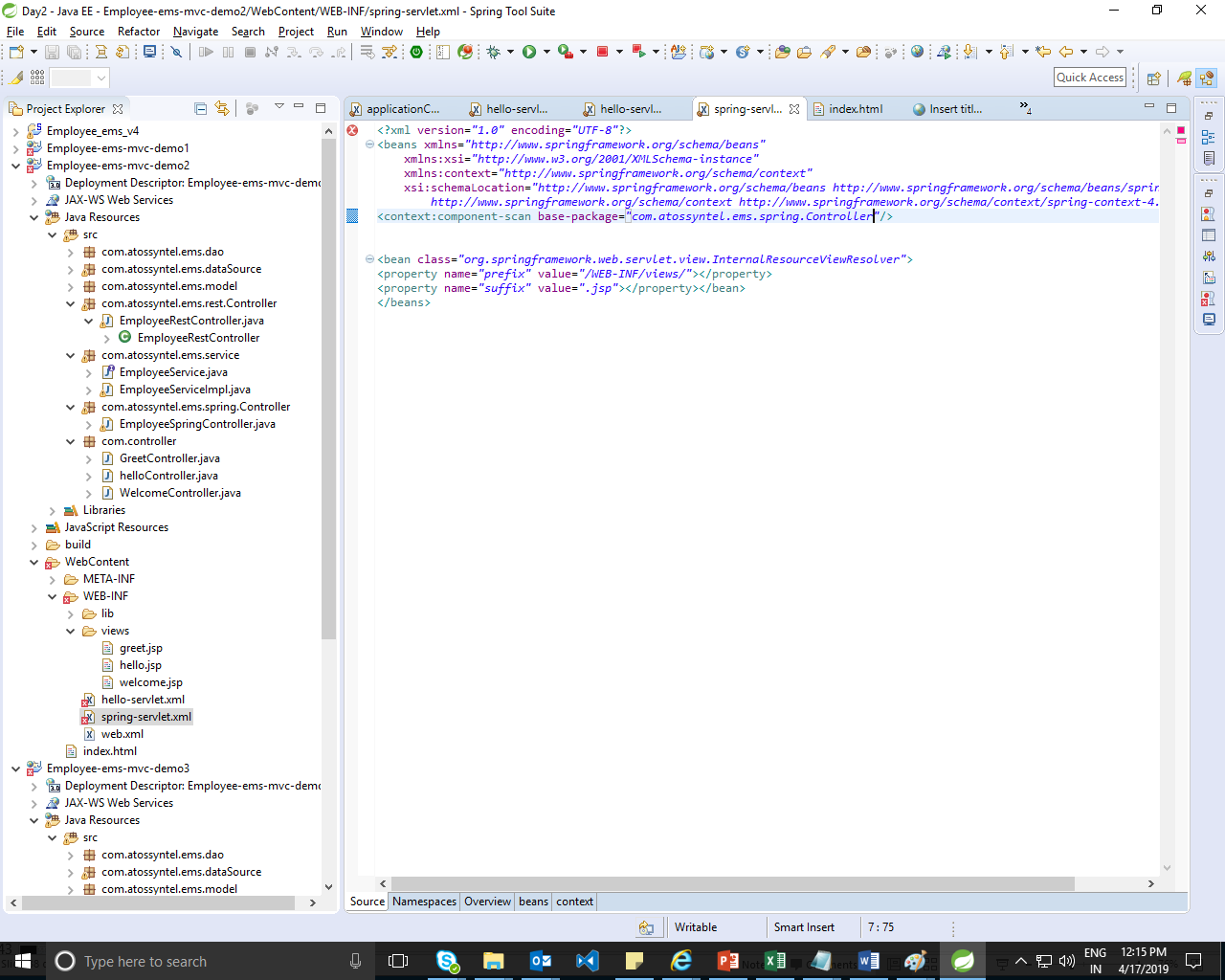
Output:

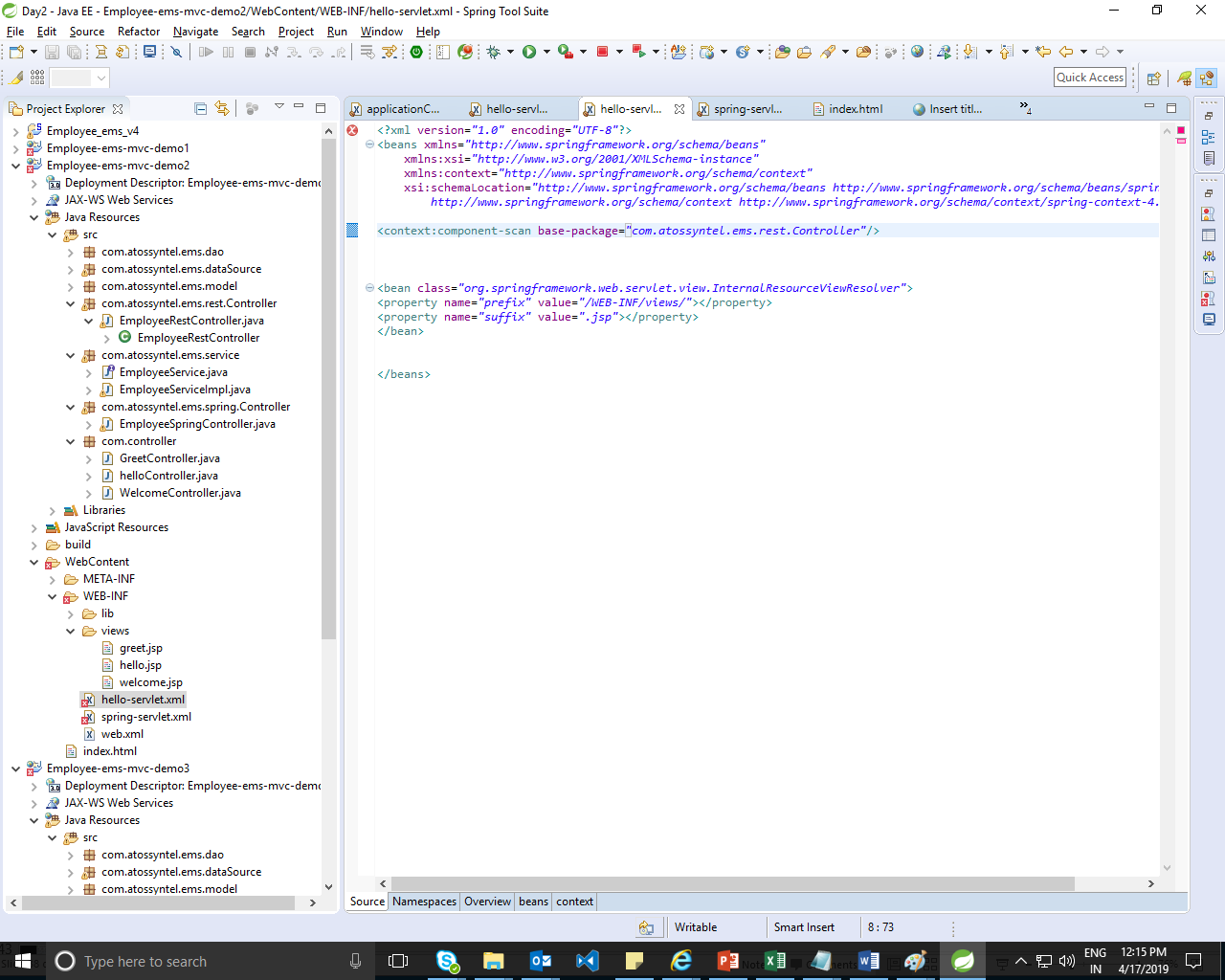


Issue :

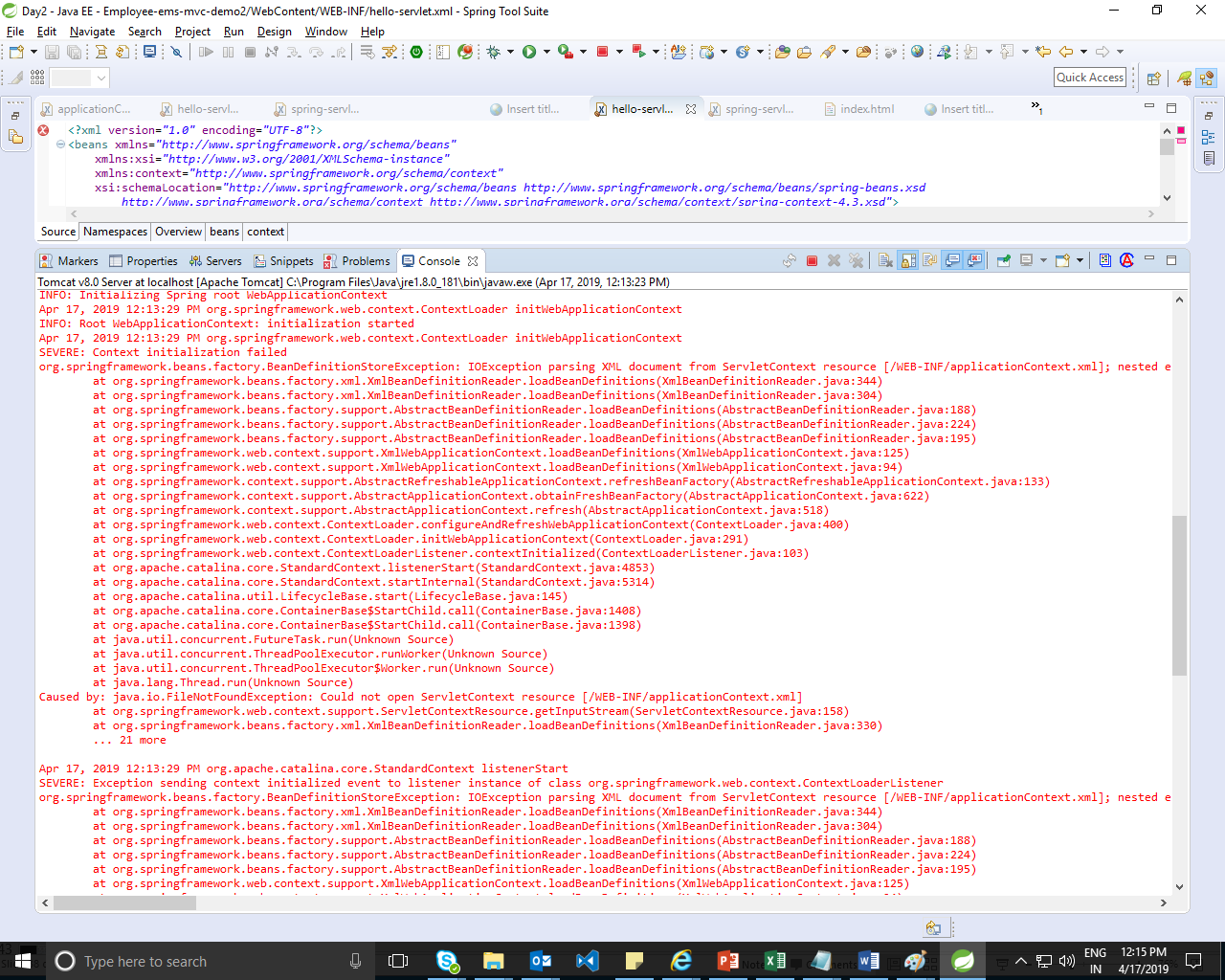
All controls are called for spring servlet and rest servlet. Both are reading data from same map here both repository and service is initialized twice which is not recommended as data is same so it shd be instantiated only once

ṢIn th same demo change context scan pkg to





Will through IOException.



Issue :

All controls are called for spring servlet and rest servlet. Both are reading data from same map data source here both repository and service is initialized twice which is not recommended as data is same so it shd be instantiated only once.

DispatcherServlet internally creates ServletWebApplicationContext .

3 containers

BeanFactory

ApplicatContextBased

ServletWebApplicationContext. Here since we have configured 2 dispatcherservlet so ServletWebApplicationContext will created twice .both are inistan the service & reposit twice. So we ar using another RootWebAppliactionContext

so we are using RootWebApplicationContext to do this use listener .ContextLoaderListener provide by spring framework in web.xml.

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

when we use this listener and start with the application then container will first search for applicationContext.xml file.if applicationContext.xml file name is different then we have to configure

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/applicationContext.xml</param-value>

</context-param>

In web.xml file

Bean ,AppliactionContext,ServletWebApplicationContext

To resolve this add applicationContext.xml

Refer: D:\ManishaSpringBoot\ManishaDemos\Day2\Employee-ems-mvc-demo3

Output after adding applicationContext.xml

3 times the instance for beans created

Rootwebapplicationcontext

Spring

Rest



In the Web MVC framework, each DispatcherServlet has its own **WebApplicationContext** (i.e own \*-**servlet**.xml), which inherits all the beans already defined in the **root WebApplicationContext**. ... **WebApplicationContext** has javax.**servlet**.ServletContext that means it's able to communicate with the container.

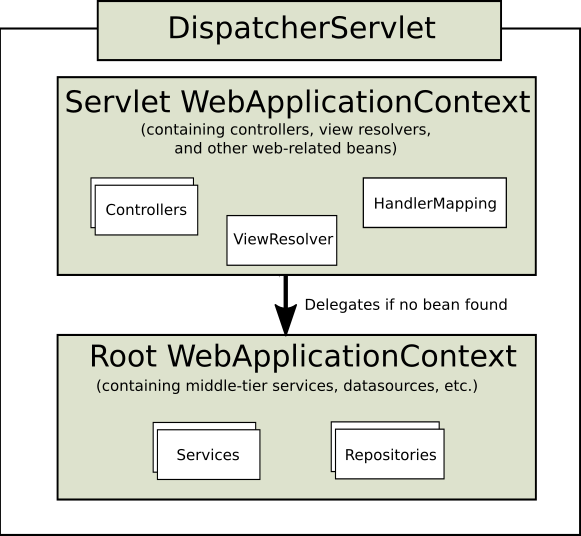
**Every Spring webapp has an associated application context that is tied to its lifecycle: the root web application context.**

This is an old feature that predates Spring Web MVC, so it’s not tied specifically to any web framework technology.

The context is started when the application starts, and it’s destroyed when it stops, thanks to a servlet context listener. The most common types of contexts can also be refreshed at runtime, although not all ApplicationContext implementations have this capability.

The context in a web application is always an instance of WebApplicationContext. That’s an interface extending ApplicationContext with a contract for accessing the ServletContext.

Anyway, applications usually should not be concerned about those implementation details: **the root web application context is simply a centralized place to define shared beans.**



Put reusable code in rootwebapplication context : such as services ,repositories

When we deploy the application first contextloaderlistener will

First it will search the bean in servletwebapplicationcontext if it is not present then search in rootwebapplicationcontext.

As per diagram rootwebapplication shd not scan the constrollers so we are excluding it from applicationContext.xml

<context:component-scan base-package=*"com"*>

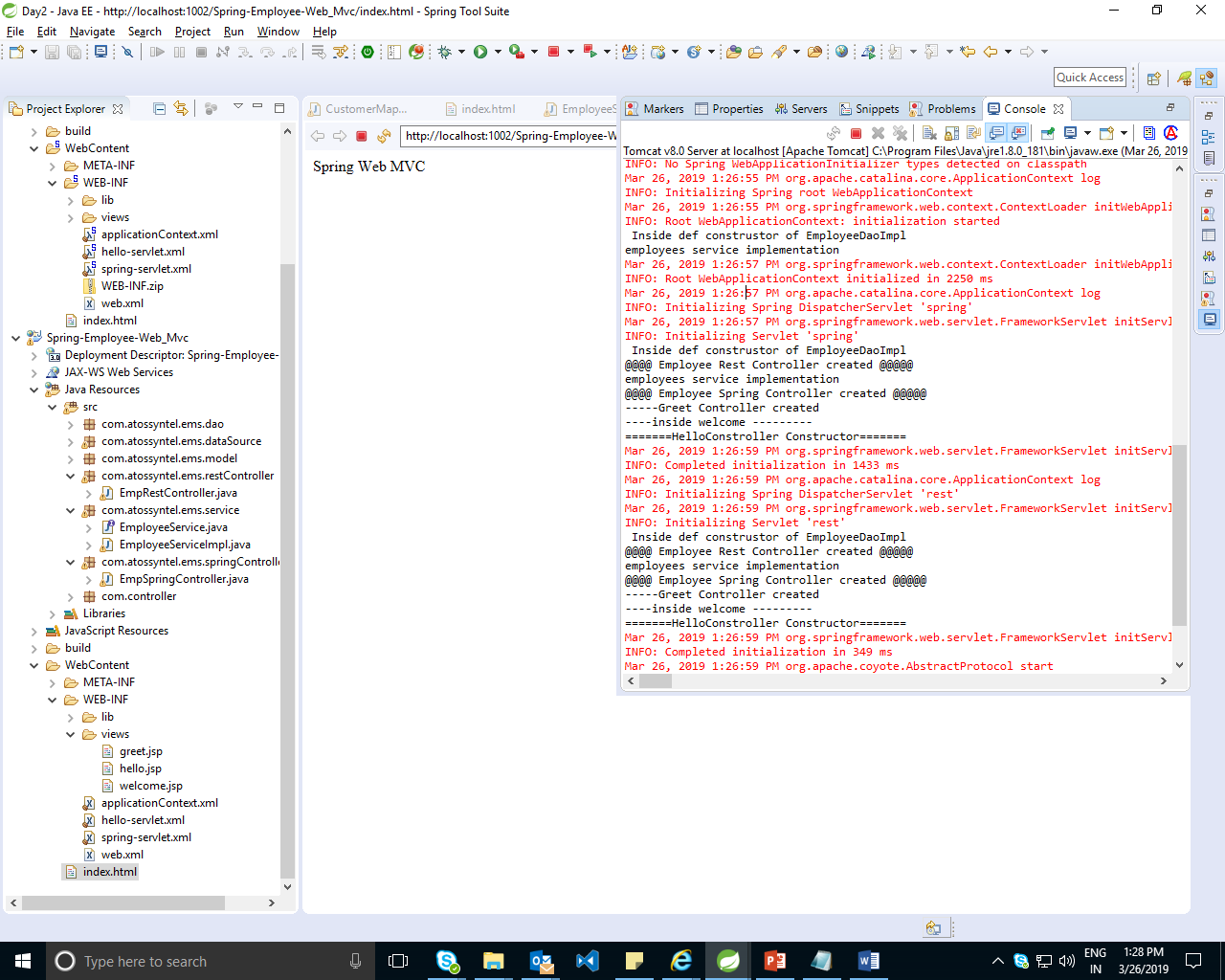
<context:exclude-filter type=*"annotation"* expression=*"org.springframework.stereotype.Controller"*/>

</context:component-scan>

Here when we run the application it will first execute the applicationContext.xml

It will scan all except controller.

And then it will load the servlet and rest dispatcher servlet and scan the controller from the respective pkgs.



Demo4:

Spring servlet and rest servlet for CRUD opration:

Spring

Rest servlet for crud application

Restful always communicate data as JSON DATA

1. @RestController->@Controller and @ResponseBody
2. @RequestBody->Client ->Server JSON

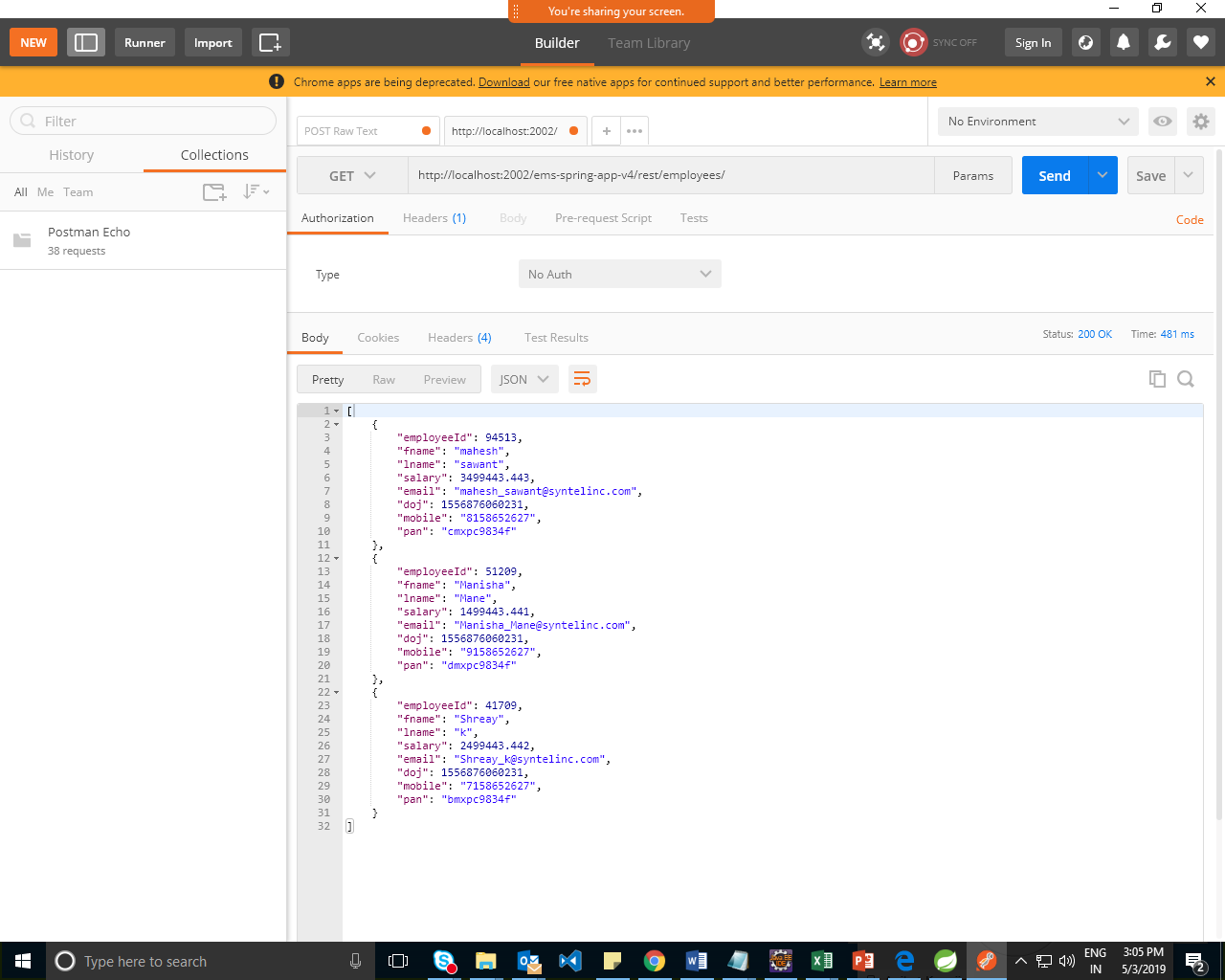
Data need to be converted to JAVA->Unmarshaling

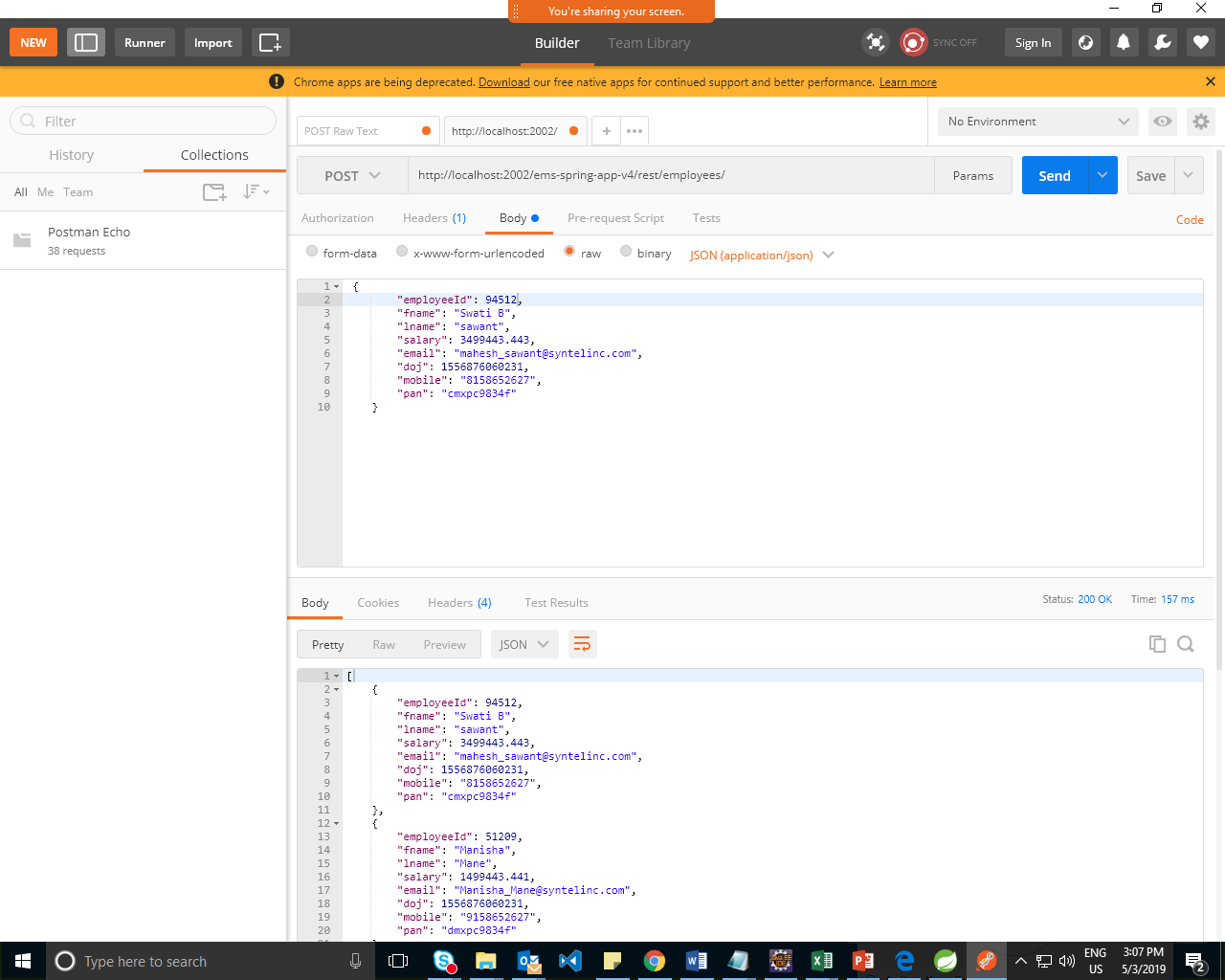
1. @ResponseBody->Server->Client

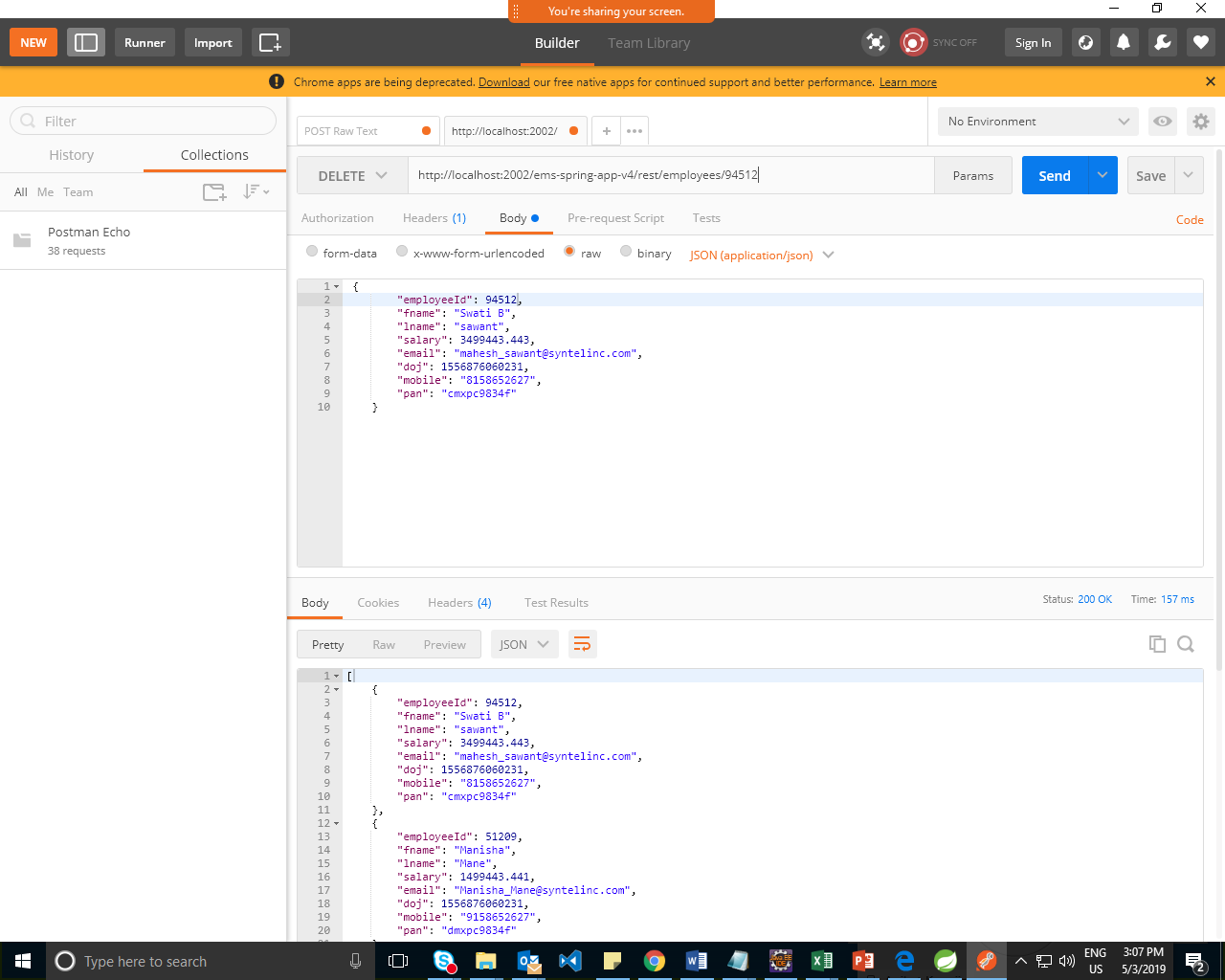
Java->JSON (marshaling)

Use postman for execution.

Get :







----------------------------Day3:----------------------------

Demo1: REST demo completed

Demo2:Spring Boot demo

Set maven configuration

import a maven project

Always run the class with main method

What is Spring Boot?

======================

Spring Boot is a brand new framework from the team at Pivotal, designed to simplify the bootstrapping and development of a new Spring application. The framework takes an opinionated approach to configuration, freeing developers from the need to define boilerplate configuration.

Step 1 :Install Apche Maven (i.e Extract Apche maven zip)

Stpe 2: do the below chanes in settings.xml (D:\apache-maven-3.5.0\conf\settings.xml)

<localRepository>d://pradeep//repository</localRepository>

<proxies>

<proxy>

<id>HTTP</id>

<active>true</active>

<protocol>http</protocol>

<host>10.119.32.71</host>

<port>8080</port>

</proxy>

<proxy>

<id>HTTPS</id>

<active>true</active>

<protocol>https</protocol>

<host>10.119.32.71</host>

<port>8080</port>

</proxy>

<proxy>

<id>FTP</id>

<active>true</active>

<protocol>ftp</protocol>

<host>10.119.32.71</host>

<port>8080</port>

</proxy>

</proxies>

Step 3: set environment variable

M2\_HOME = D:\apache-maven-3.5.0

path = D:\apache-maven-3.5.0\bin

Step 4: Configure Maven in STS

Windows->Preferences->Maven->Installations-> Add

=> Instalation Type=>External

Installation Home =>D:\apache-maven-3.5.0

Users setting =>

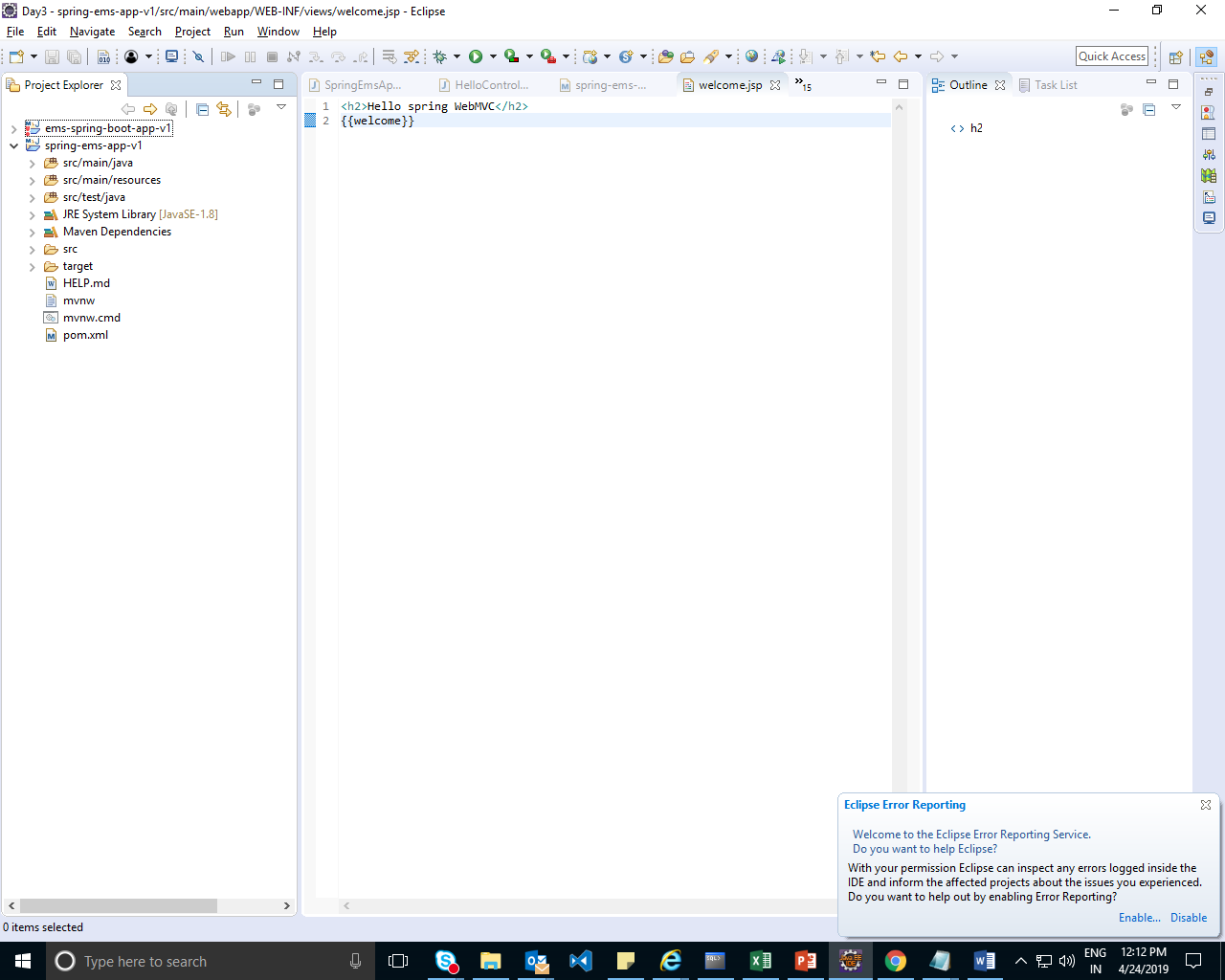
change location of settings.xml to

(D:\apache-maven-3.5.0\conf\settings.xml)

Step 5 :

Open => https://start.spring.io/

Generate Maven Project with necessary deepndencies (Web,Devtools...etc)



Step: Add changes in applicationContext.xml

server.port=1212

server.port=1212

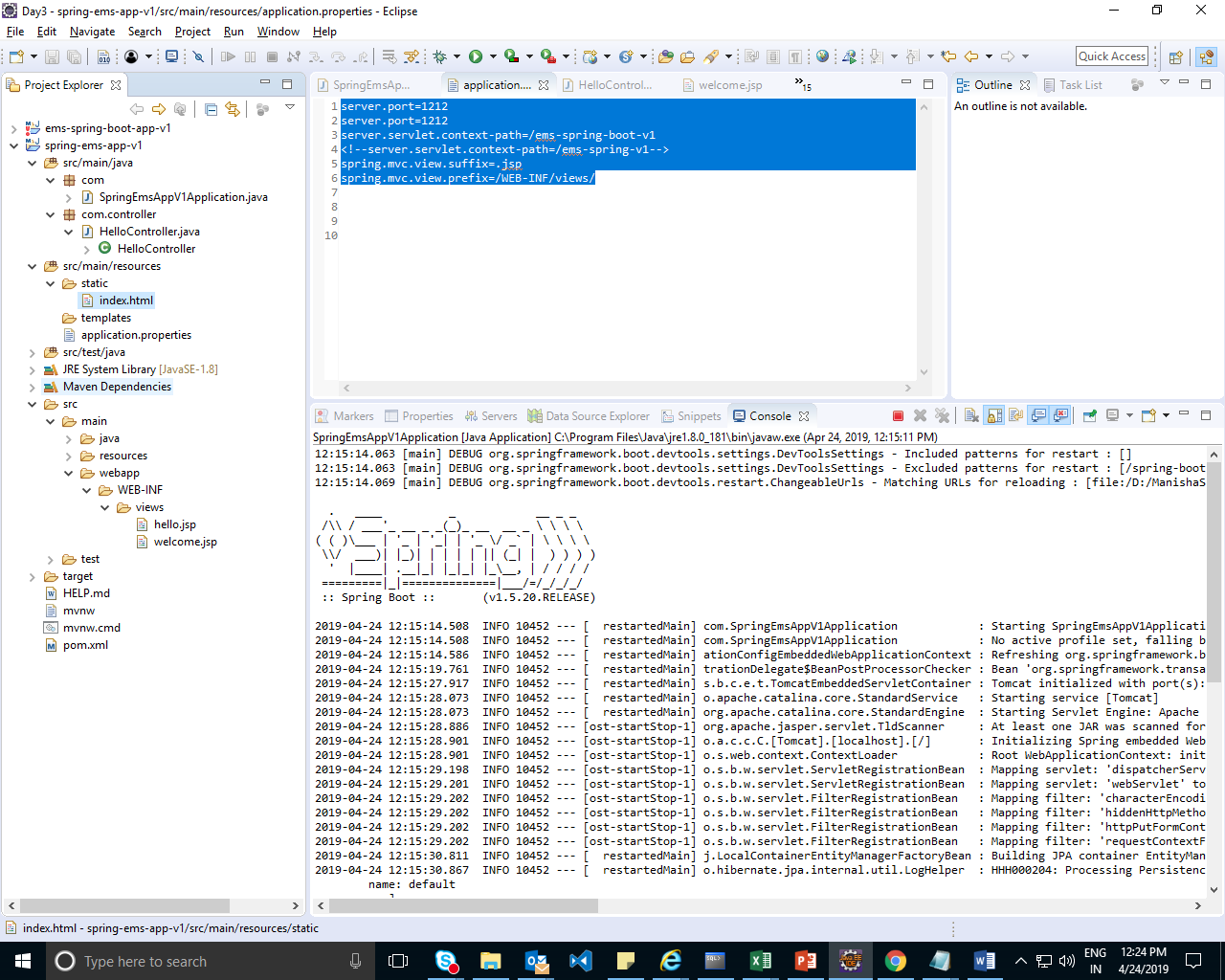
server.servlet.context-path=/ems-spring-boot-v1

<!--server.servlet.context-path=/ems-spring-v1-->

spring.mvc.view.suffix=.jsp

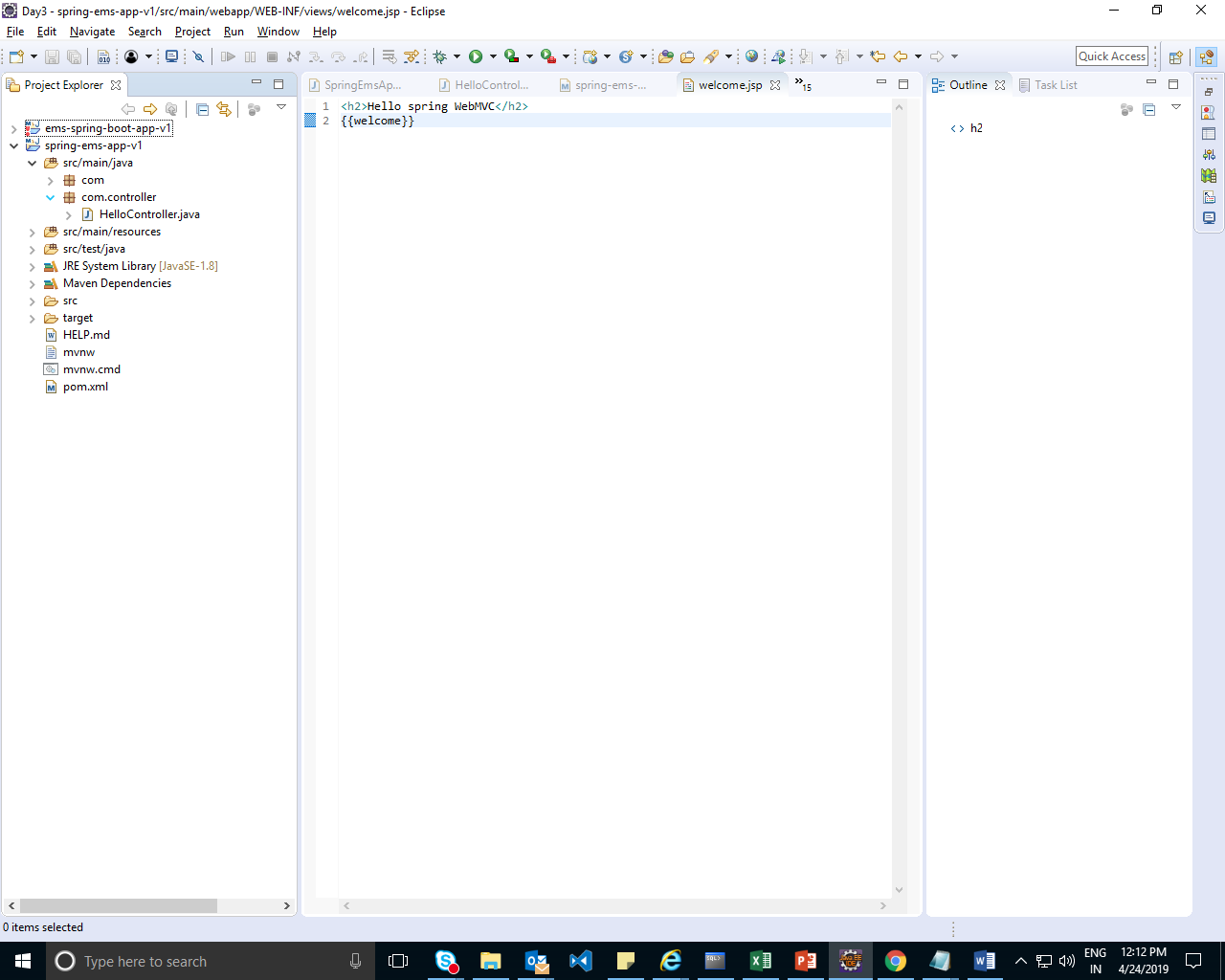
spring.mvc.view.prefix=/WEB-INF/views/

add index.jsp page in static folder



Run The class with main method.

Demo2: Add a Hellocontroller



**package** com.controller;

**import** java.util.Date;

**import** org.springframework.stereotype.Controller;

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

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

**import** org.springframework.web.servlet.ModelAndView;

@Controller

**public** **class** HelloController {

**public** HelloController() {

System.***out***.println("Hello controller created...");

}

@RequestMapping("/welcome")

**public** ModelAndView welcome(){

**return** **new** ModelAndView("welcome","message","Welcome To Spring Web MVC");

}

@RequestMapping("/today")

**public** @ResponseBody String today(){

**return** "Today is :"+**new** Date();

}

@RequestMapping("/hello")

**public** @ResponseBody String hello1(){

**return** "HEllo World!!";

}

}

Note Change package name where main method is used to com as all cotrollers are in package com.controller……

Add dependencies in POM.XML for displaying the JSP pages

<!-- To Send JSP Response -->

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

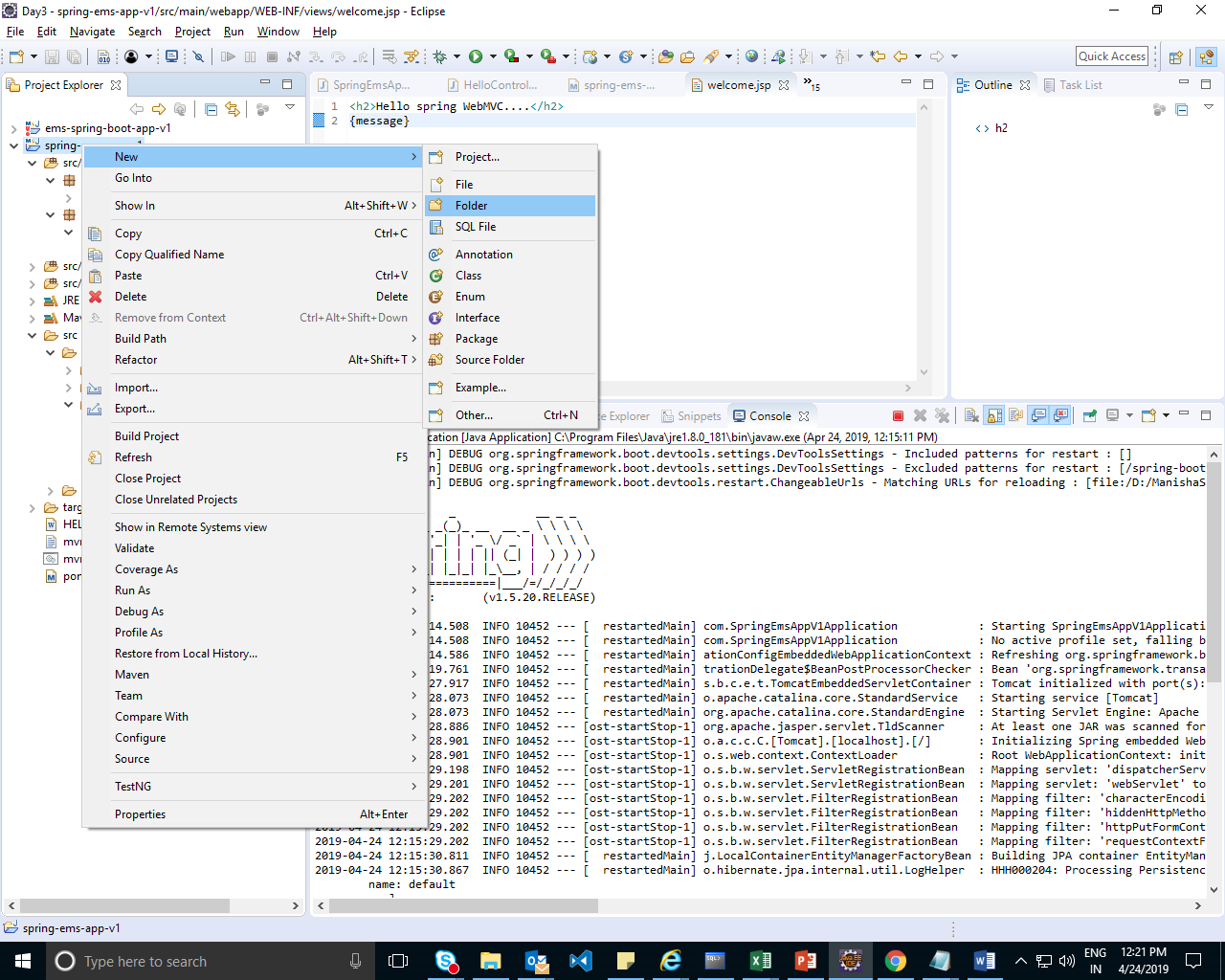
<dependency>

<groupId>javax.servlet</groupId>

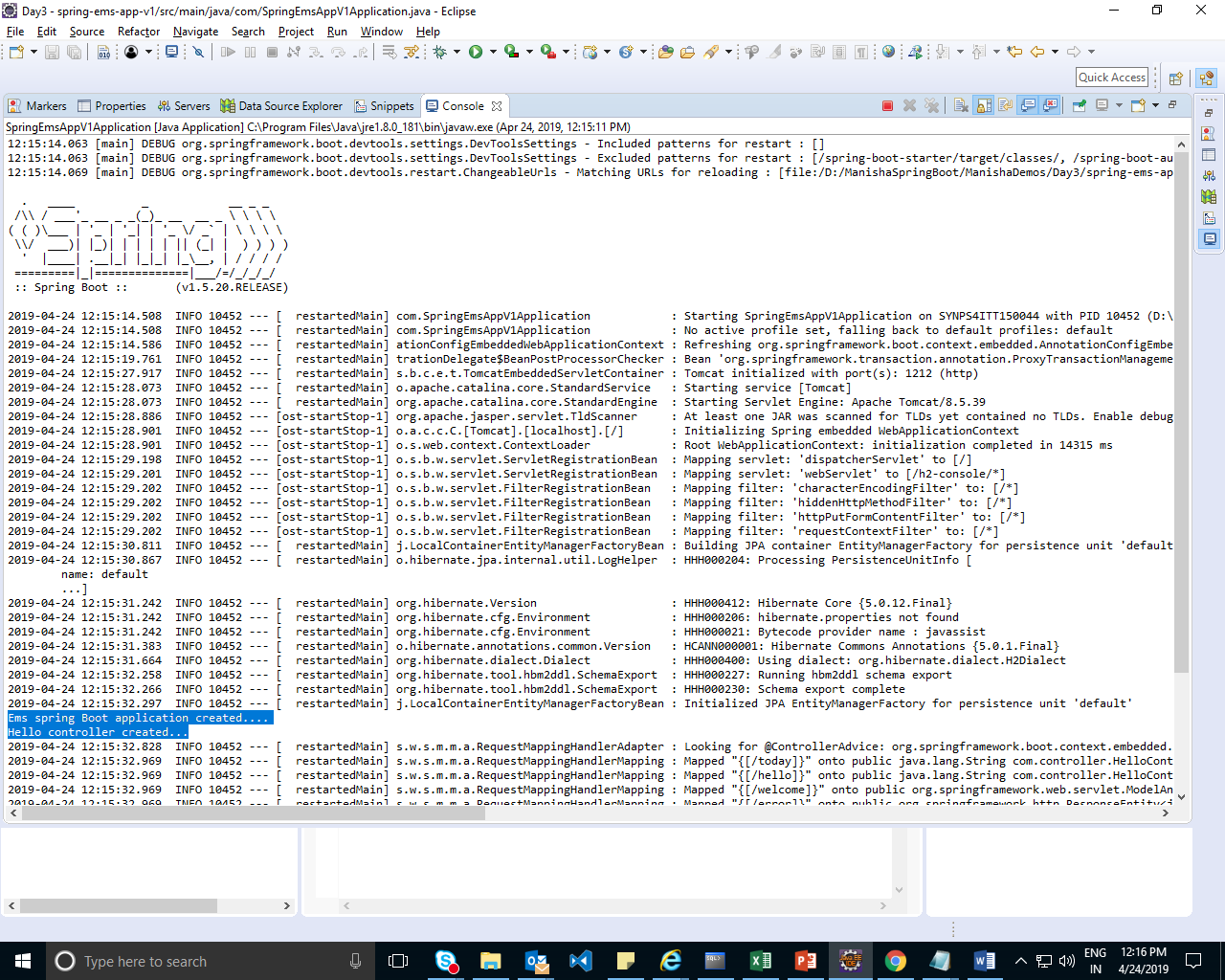
<artifactId>jstl</artifactId>

</dependency>

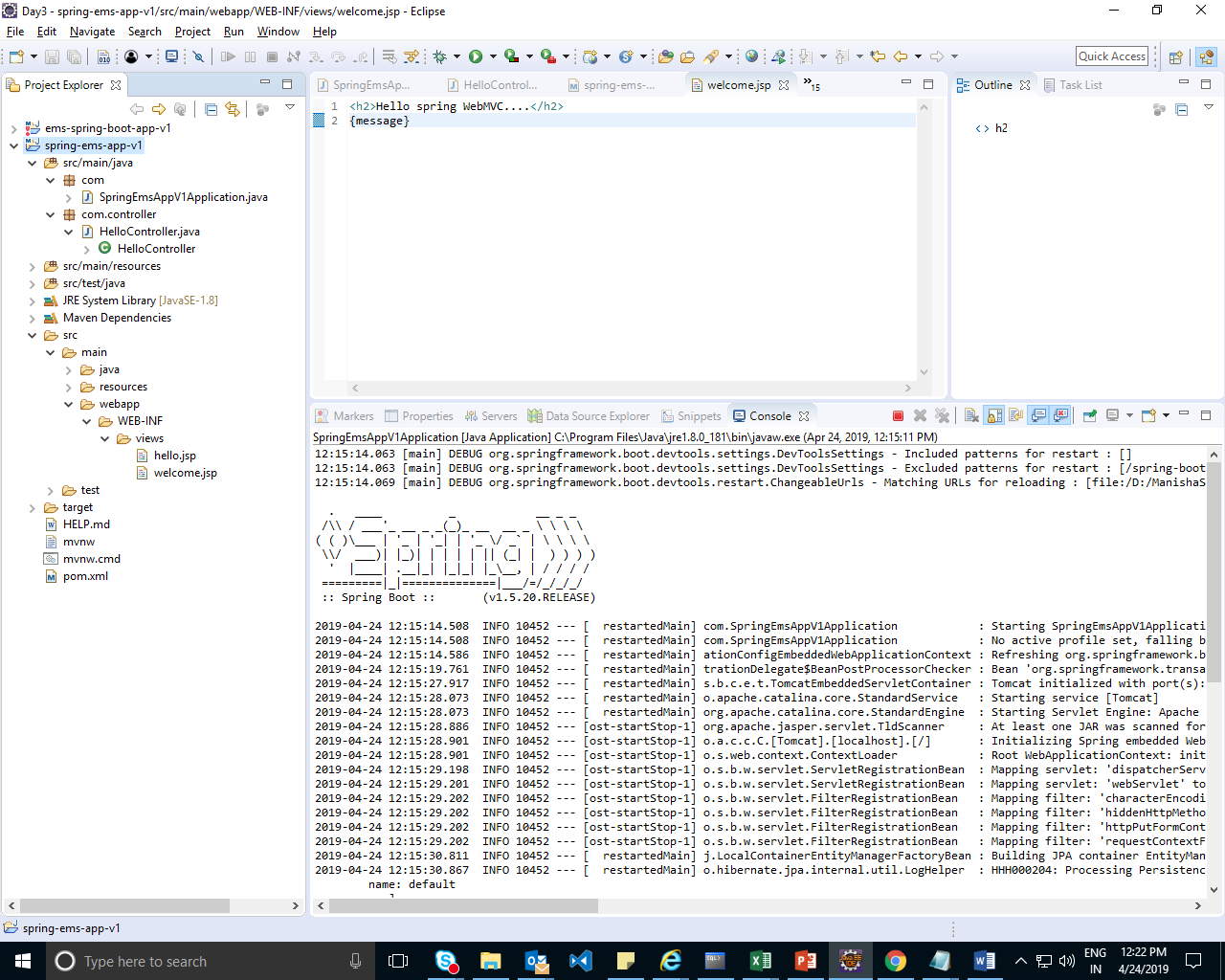
Add a new Folder at with name webapp



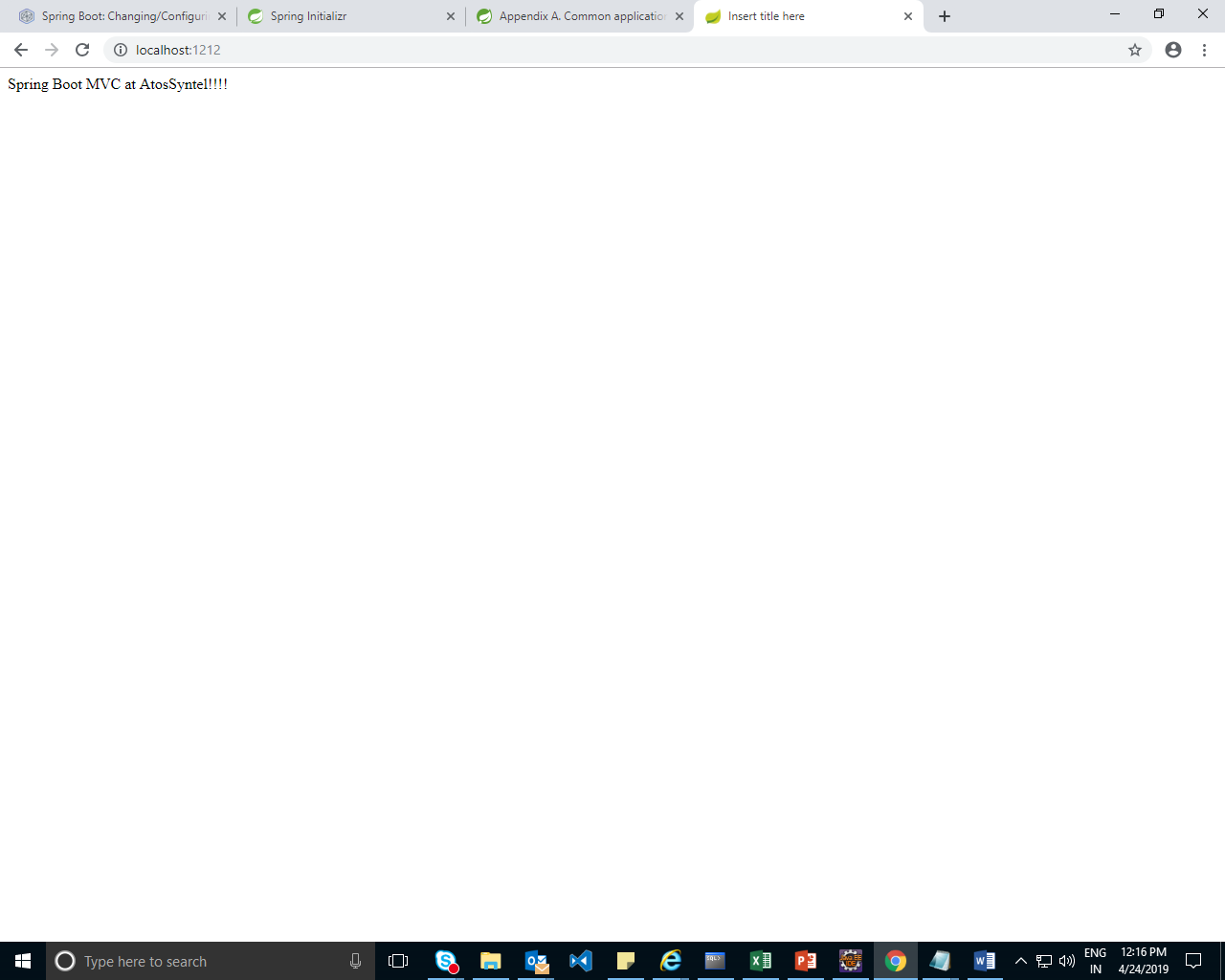
Run th MainApp

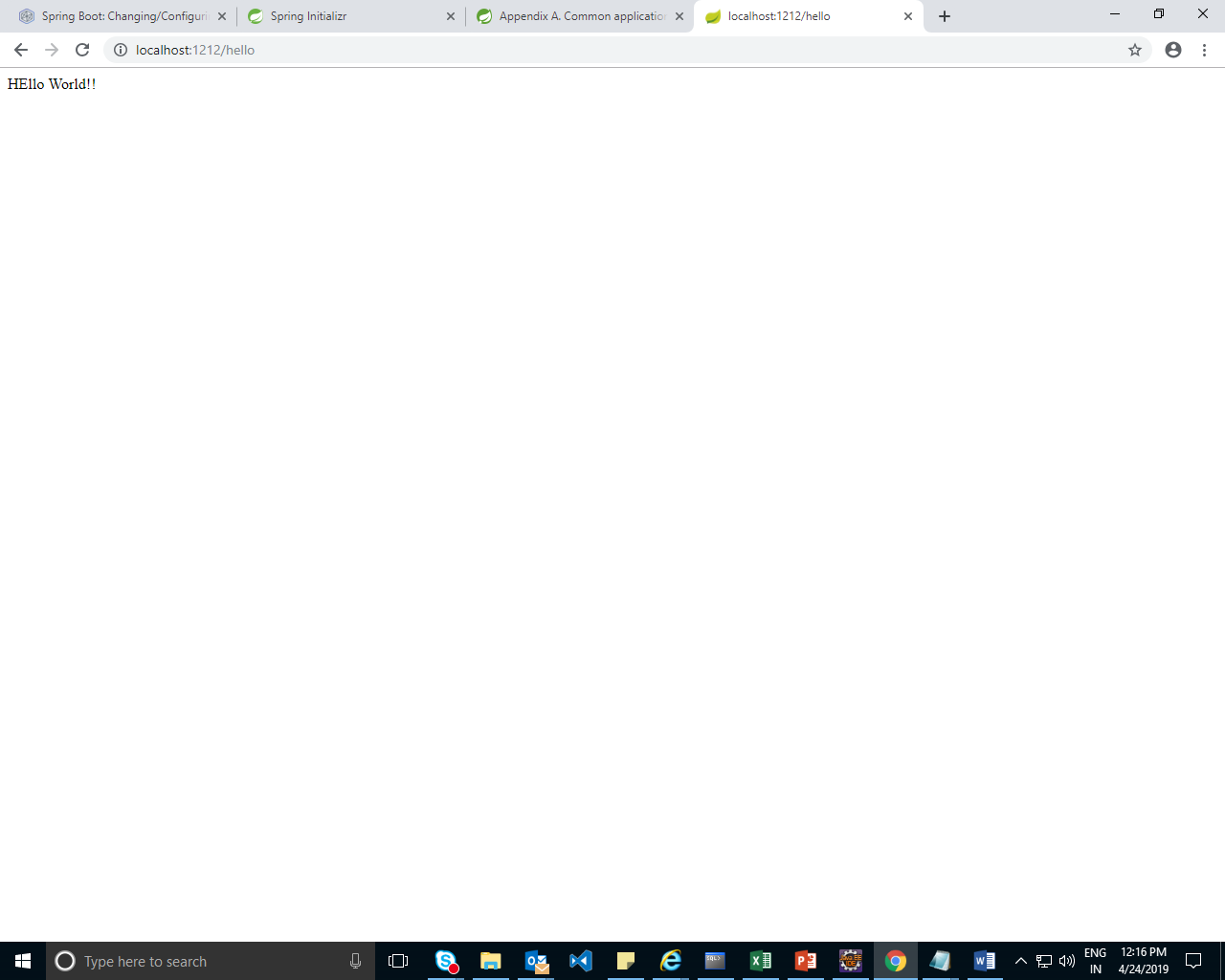


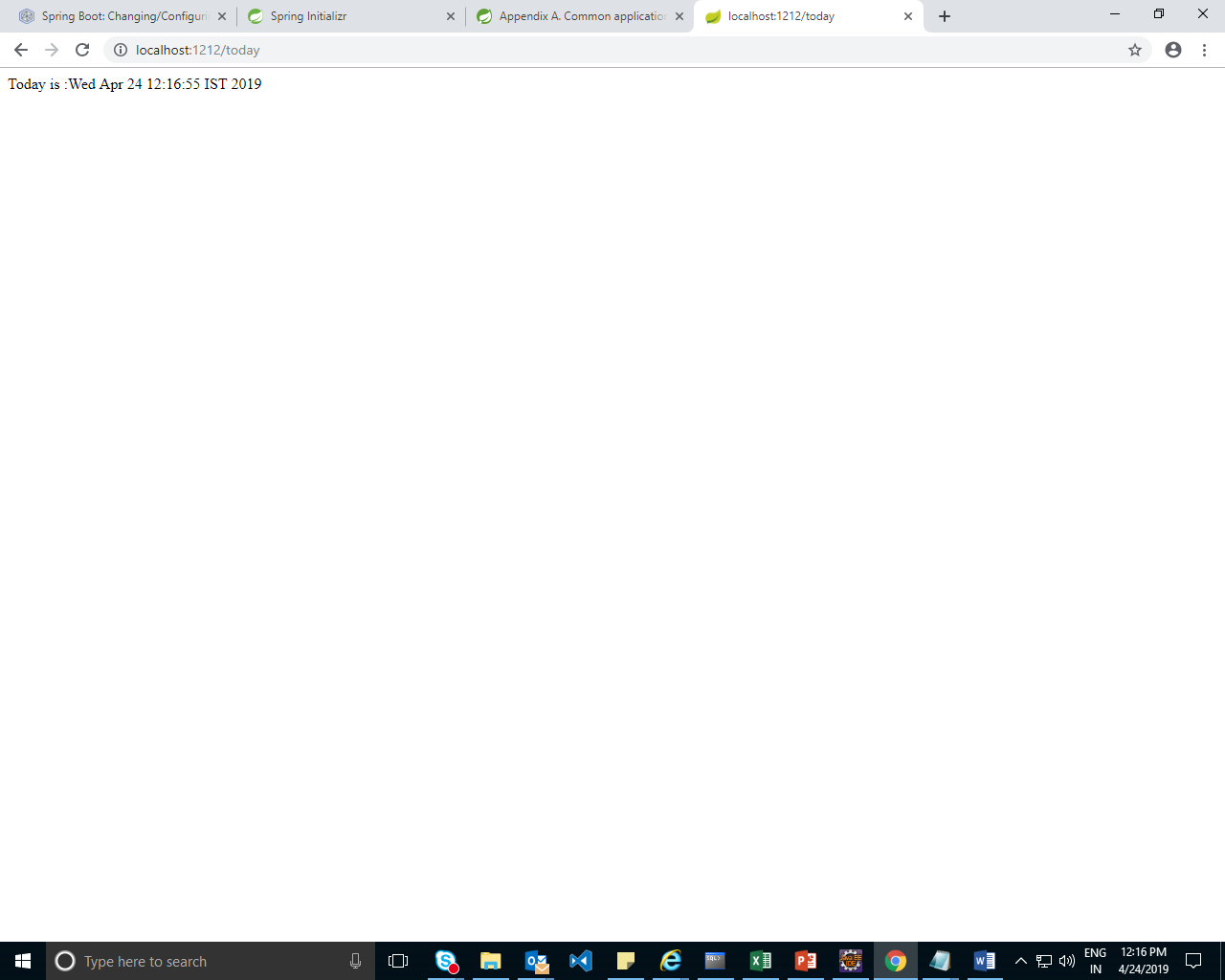
Add folder webapp/WEB-IBF/Views and add jsp pages hello.jsp and welcome.jsp



Open Chrome and start the application:







D:\ManishaSpringBoot\ManishaDemos\Day3\Spring-ems-app-v1

Note in applicationConrtext.xml

server.context-path=/ems-spring-boot-v1 is used for version 1.5

server.servlet.context-path=/ems-spring-v1 is used for version 2.

Demo2 : Demo1: add the src from ems application with spring and rest

D:\ManishaSpringBoot\ManishaDemos\Day3\springboot-ems-app-v2

* To send jsp reponse add dependencies
* <!-- To Send JSP Response -->
* <dependency>
* <groupId>org.springframework.boot</groupId>
* <artifactId>spring-boot-starter-tomcat</artifactId>
* </dependency>
* <dependency>
* <groupId>org.apache.tomcat.embed</groupId>
* <artifactId>tomcat-embed-jasper</artifactId>
* </dependency>
* <dependency>
* <groupId>javax.servlet</groupId>
* <artifactId>jstl</artifactId>
* </dependency>

Demo3 with h2 DB:

D:\ManishaSpringBoot\ManishaDemos\Day3\ems-spring-boot-app-v1

Hibernate

Demos:

D:\ManishaSpringBoot\Demos\day3\spring-boot-custmomer-app-v1

Spring Boot with h2 DB

Add dependencies

<!-- JPA Reposiotry -->

<dependency>

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

<artifactId>spring-–––––</artifactId>

</dependency>

<!-- h2 database -->

<dependency>

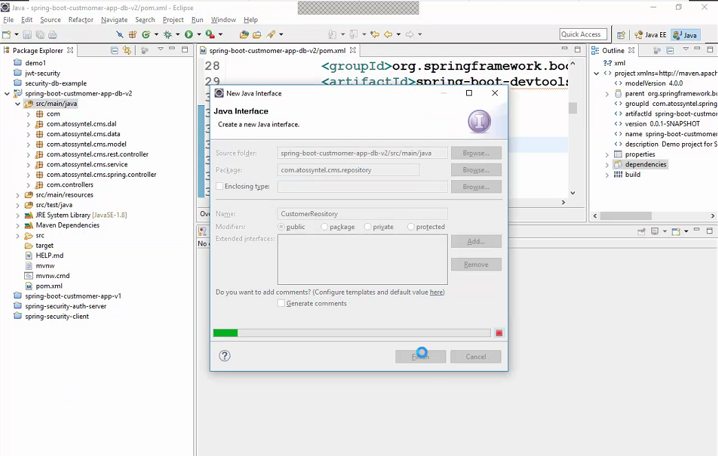
<groupId>com.h2database</groupId>

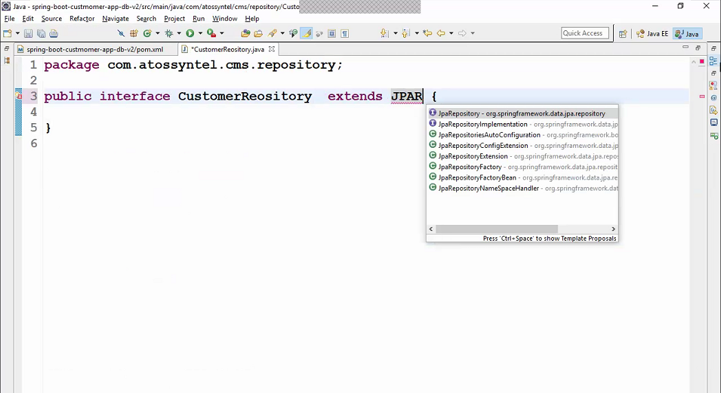
<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

Add new interface :





package com.atossyntel.cms.repository;

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

import org.springframework.stereotype.Repository;

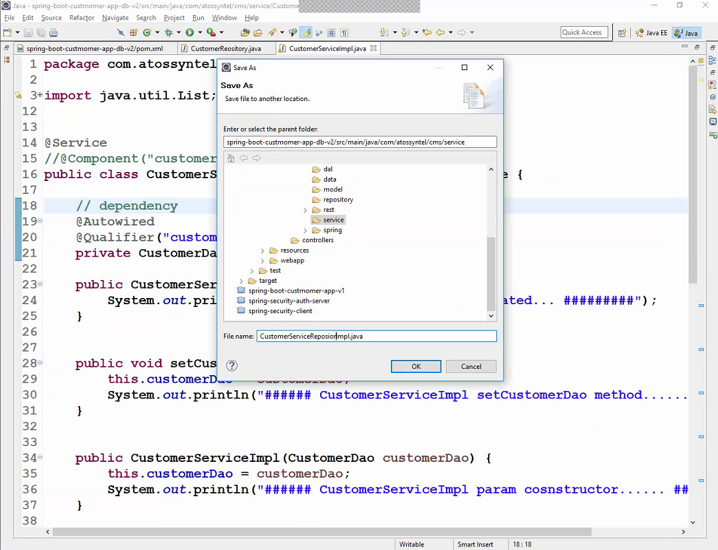
import com.atossyntel.cms.model.Customer;

@Repository

public interface CustomerReository extends JpaRepository<Customer, Integer> {

}

In Service Layer Add 1 service



package com.atossyntel.cms.service;

import java.util.List;

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

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

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Service;

import com.atossyntel.cms.dal.CustomerDao;

import com.atossyntel.cms.model.Customer;

import com.atossyntel.cms.repository.CustomerReository;

@Service

// @Component("customerServiceImpl")

public class CustomerServiceReposioryImpl implements CustomerService {

// dependency

@Autowired

private CustomerReository repository;

public CustomerServiceReposioryImpl() {

System.out.println("###### CustomerServiceReposioryImpl created... #########");

}

@Override

public boolean addCustomer(Customer customer) {

// TODO Auto-generated method stub

return repository.save(customer) == customer;

}

@Override

public boolean updateCustomer(Customer customer) {

Customer c = repository.findById(customer.getCustomerId()).get();

if (c != null) {

return repository.save(customer) == customer;

}

return false;

}

@Override

public boolean deleteCustomer(int customerId) {

Customer customer = repository.findById(customerId).get();

if (customer != null) {

repository.delete(customer);

return true;

}

return false;

}

@Override

public Customer findCustomerById(int customerId) {

// TODO Auto-generated method stub

return repository.findById(customerId).get();

}

@Override

public List<Customer> findAllCustomers() {

// TODO Auto-generated method stub

return repository.findAll();

}

}

Change the model Cutomer add annotations

package com.atossyntel.cms.model;

import java.util.Random;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name="pc\_customers")

public class Customer {

@Id

private int customerId;

private String firstName;

private String lastName;

private String gender;

private String email;

private String address;

private String city;

private String state;

public Customer() {

this.customerId=new Random().nextInt(100000);

}

public Customer(String firstName, String lastName, String gender, String email, String address, String city,

String state) {

this.customerId=new Random().nextInt(100000);

this.firstName = firstName;

this.lastName = lastName;

this.gender = gender;

this.email = email;

this.address = address;

this.city = city;

this.state = state;

}

public int getCustomerId() {

return customerId;

}

public void setCustomerId(int customerId) {

this.customerId = customerId;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getCity() {

return city;

}

public void setCity(String city) {

this.city = city;

}

public String getState() {

return state;

}

public void setState(String state) {

this.state = state;

}

@Override

public String toString() {

return "Customer [customerId=" + customerId + ", firstName=" + firstName + ", lastName=" + lastName + ", gender="

+ gender + ", email=" + email + ", address=" + address + ", city=" + city + ", state=" + state + "]";

}

}

Enable Hibernate in application.properties

# Server HTTP port.

server.port=8000

# Context path of the application.

server.servlet.context-path=/spring-customer-web-mvc

#register view resolver

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

spring.mvc.view.suffix=.jsp

spring.jpa.hibernate.ddl-auto=update

#logging.level.org.hibernate.SQL=debug

# Enabling H2 Console

spring.h2.console.enabled=true

#Turn Statistics on

#spring.jpa.properties.hibernate.generate\_statistics=true

#logging.level.org.hibernate.stat=debug

# Show all queries

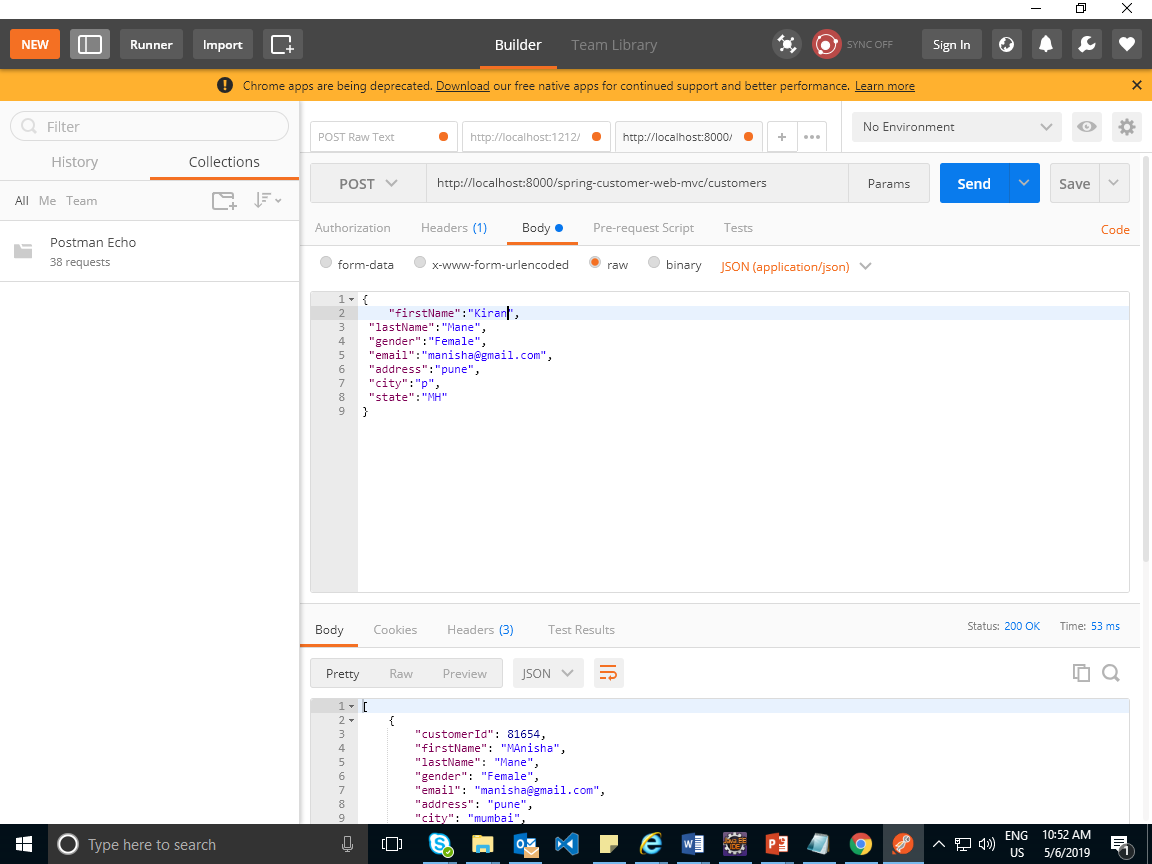
spring.jpa.show-sql=true

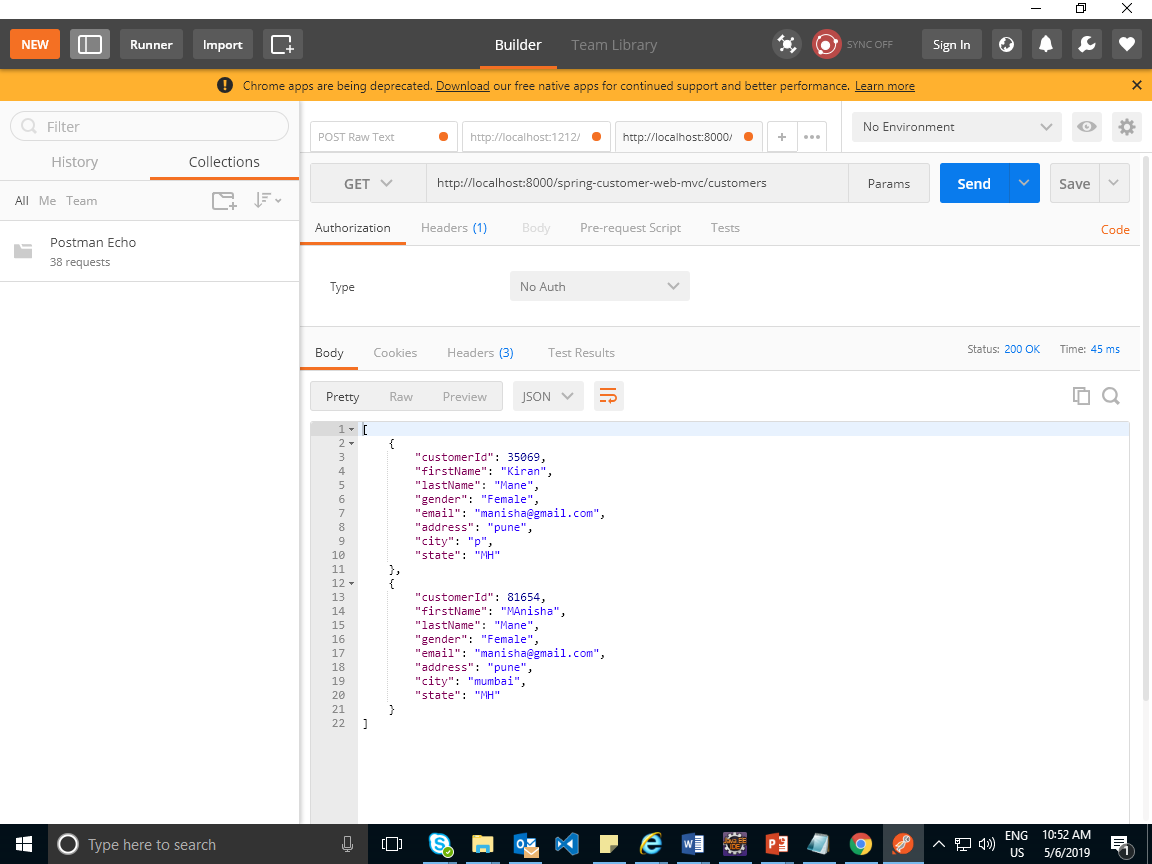
spring.jpa.properties.hibernate.format\_sql=true

#logging.level.org.hibernate.type=trace

Demo

D:\ManishaSpringBoot\ManishaDemos\Day4\spring-boot-custmomer-app-v2\_db





Demo:same with configuration files:

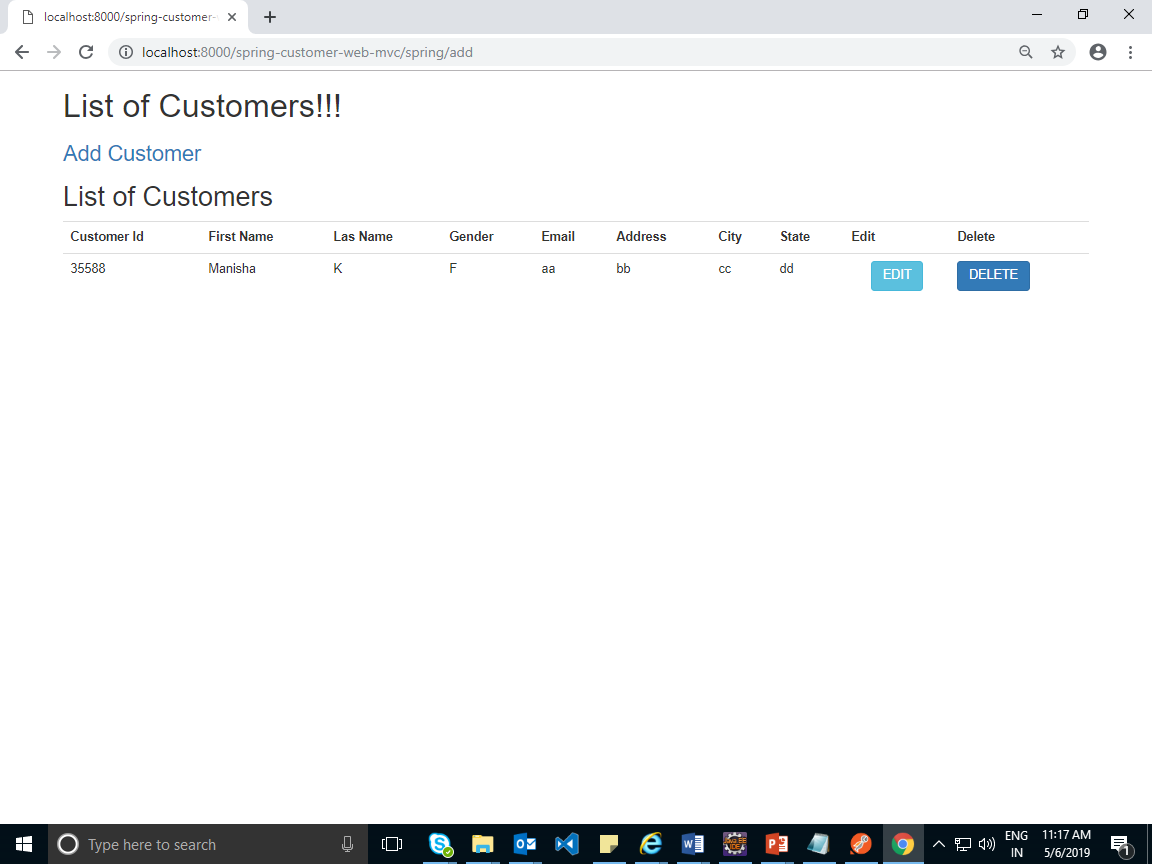
Add 3 configuration Files

CustomerDSConfig

RestConfig

SpringConfig

Add spring word in all views



--------------------------------Day4------------------------

MongoDB

RDBMS NOSQL

Tables Collection

Rows/tuples documents

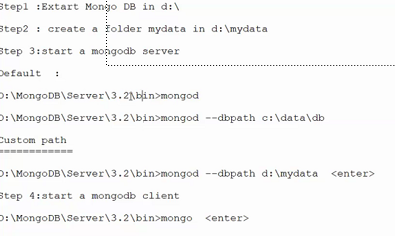
Columns fields

Primary key \_id(auto generated by MongoDB)

Mogodb is available in 2 versions

MSI and zip version

For database configuration update the application.properties



spring boot with mongodb

Mongodb Quesries:

Command to start server

mongod.exe --dbpath "d:\data"

Command to start client:

mongo.exe

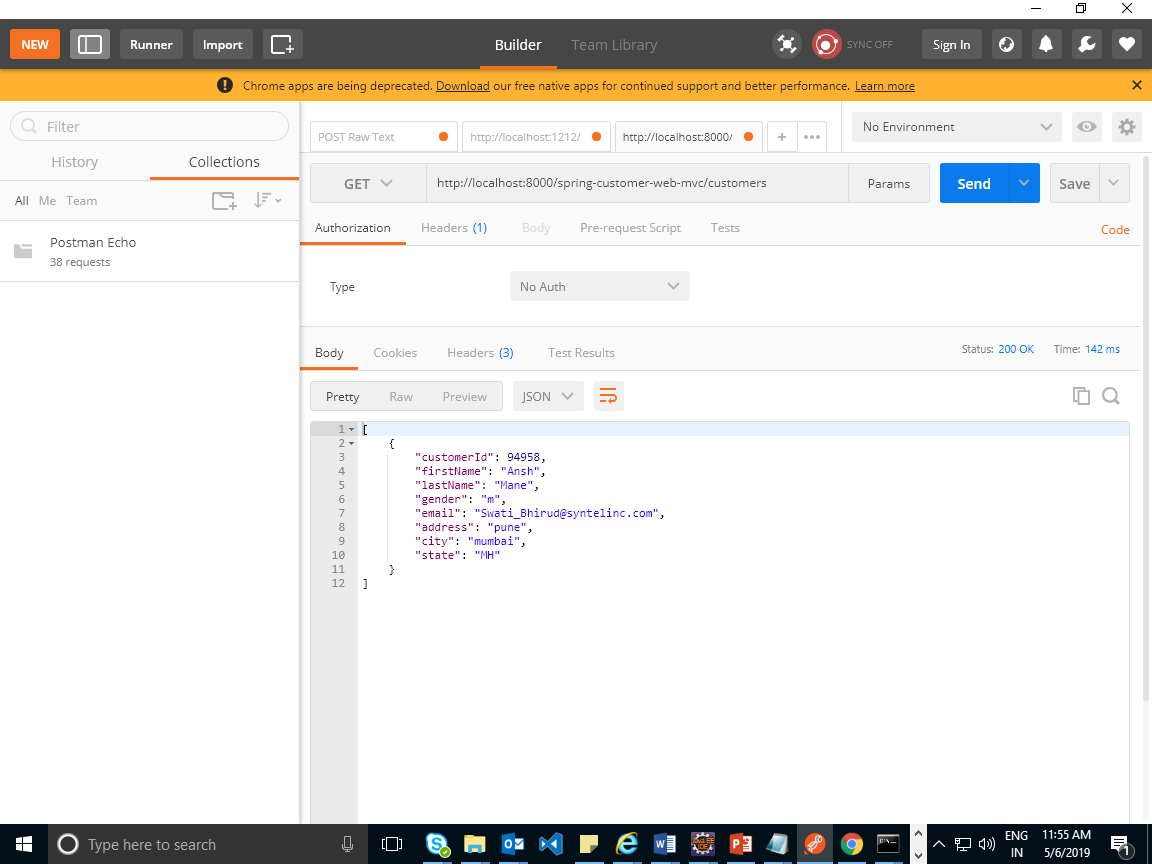
MongoDB Demo:

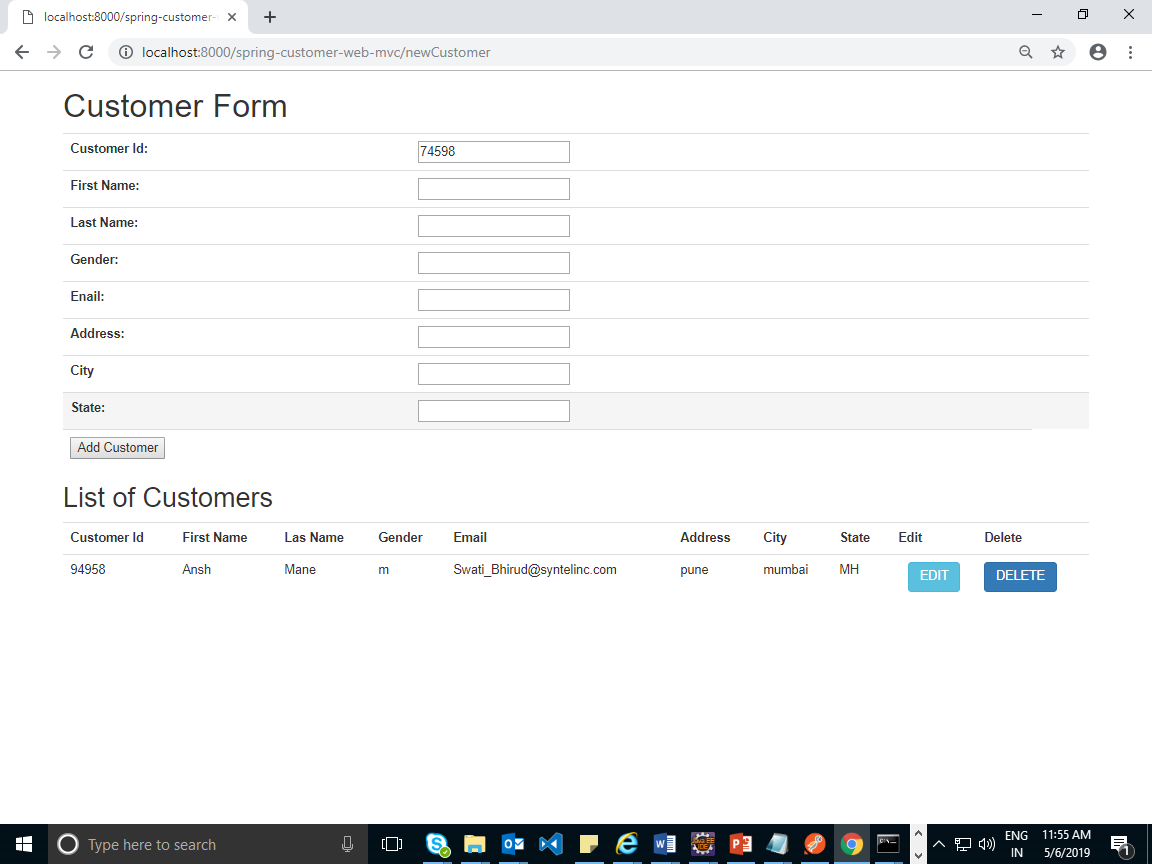
Use db—to create or change db

Show databases;

Show collections;

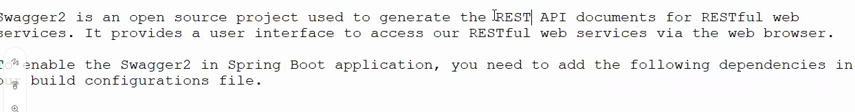
Test The Demo with spring and rest





Swagger:is used for documentation for REST Ful API.

Same as java documentation



RESTFul—services invoked via http url

Example use already available REST ful api

Use url <https://jsonplaceholder.typicode.com/> to get few FAKE Restful api’s

It provides User interface to access the restful api istead of using POST man

Demo

Add swagger dependencies in POM.xml

<!-- swagger dependencies -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.7.0</version>

</dependency>

Enable Swagger in Main App: to generate the documentation enable swagger

**package** com;

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.context.annotation.Bean;

**import** springfox.documentation.builders.RequestHandlerSelectors;

**import** springfox.documentation.spi.DocumentationType;

**import** springfox.documentation.spring.web.plugins.Docket;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2

@SpringBootApplication

**public** **class** SpringBootCustmomerAppV1Application {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBootCustmomerAppV1Application.**class**, args);

}

@Bean

**public** Docket csutomerApi() {

System.***out***.println("####### Generating swagger documentaation for customer restful api #########");

**return** **new** Docket(DocumentationType.***SWAGGER\_2***).select()

.apis(RequestHandlerSelectors.*basePackage*("com.atossyntel.cms.rest.controller")).build();

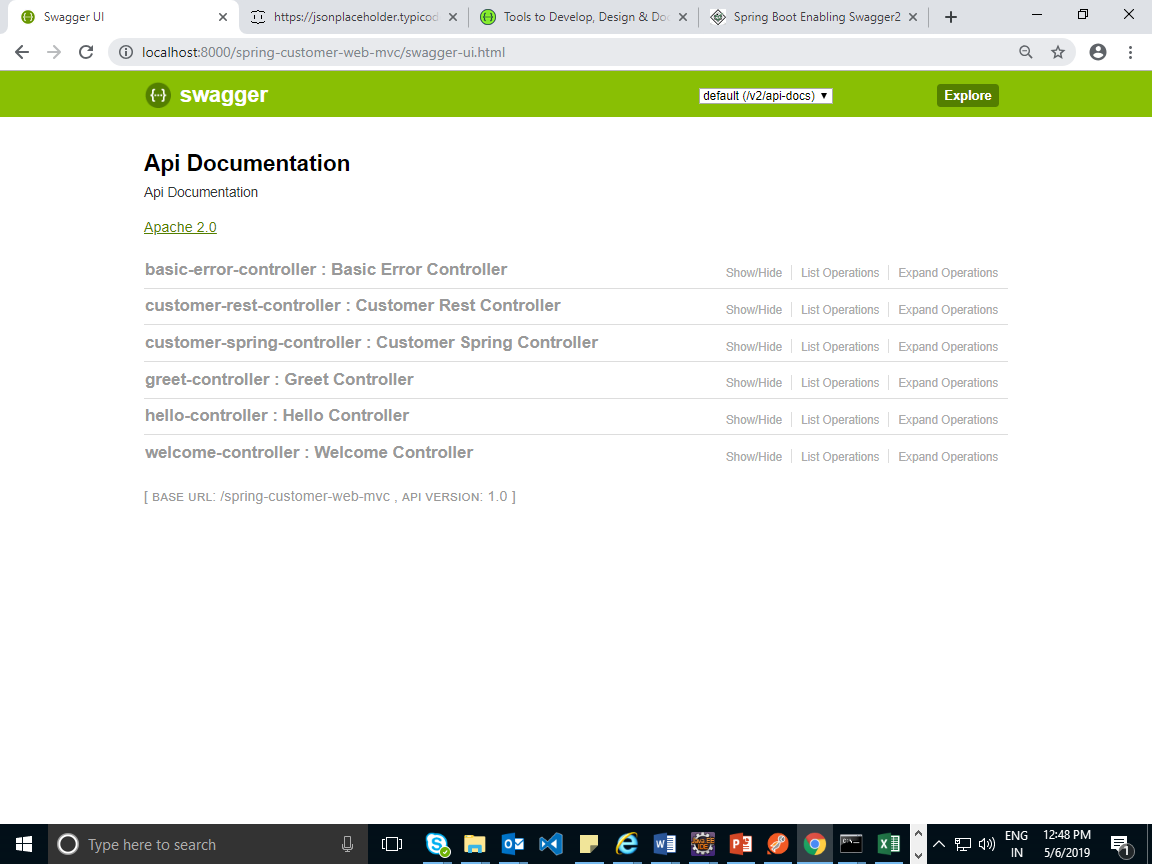
}

}

Run the main App

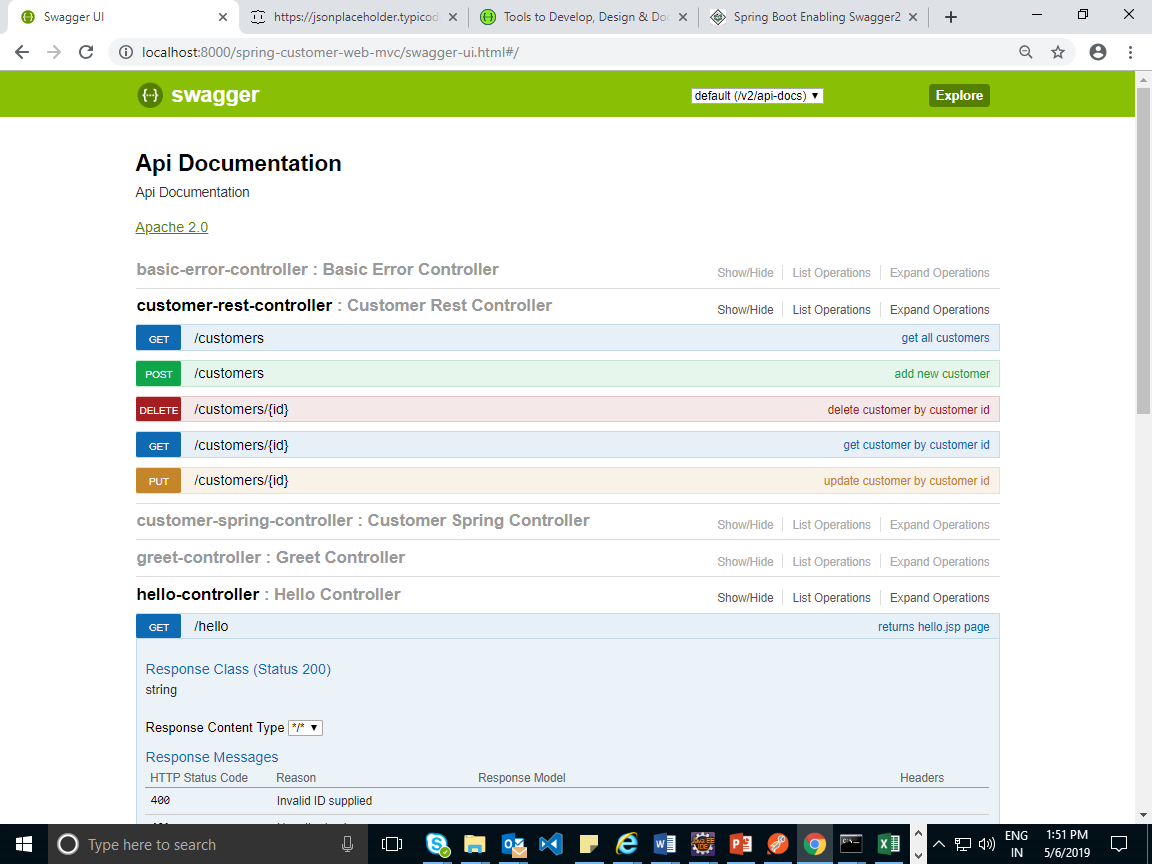
Use below url for swagger:

<http://localhost:8000/spring-customer-web-mvc/swagger-ui.html>

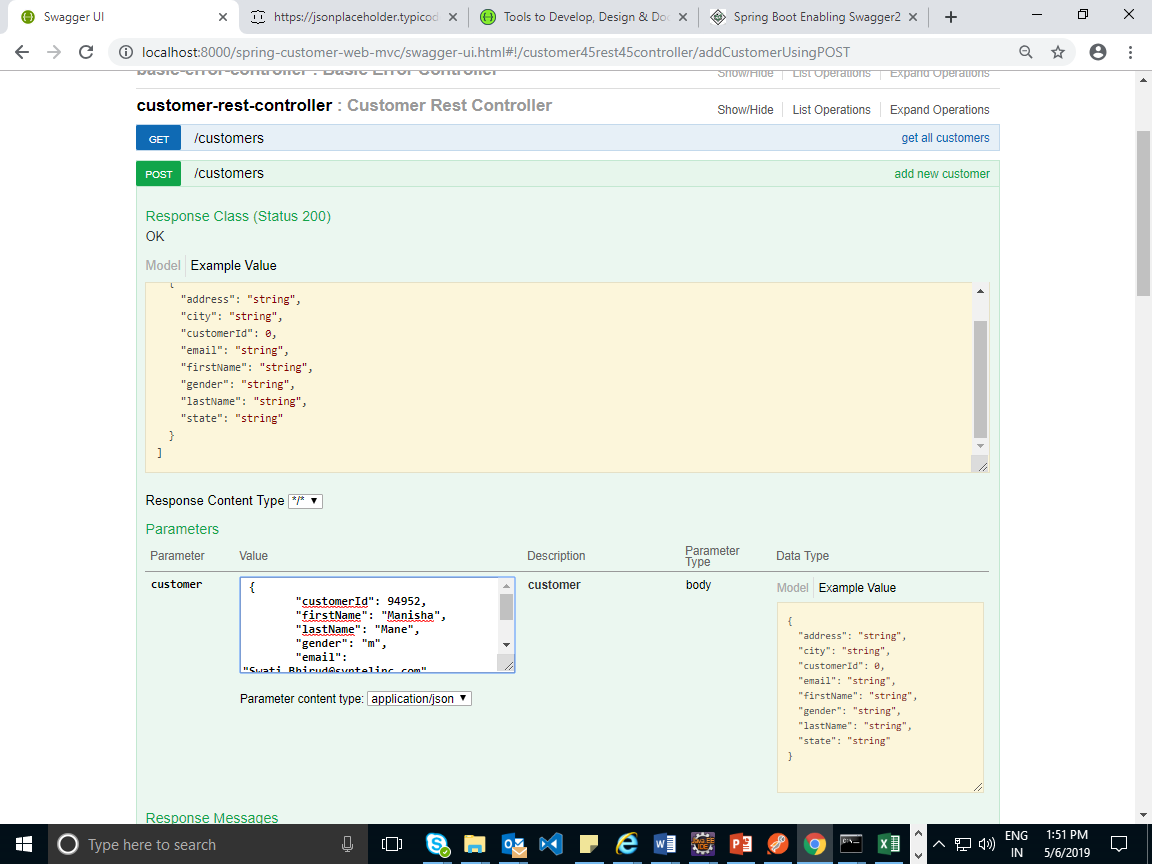


@ApiOperation: will change the opration details

How to test Rest API in swagger:



Add customer details as show below

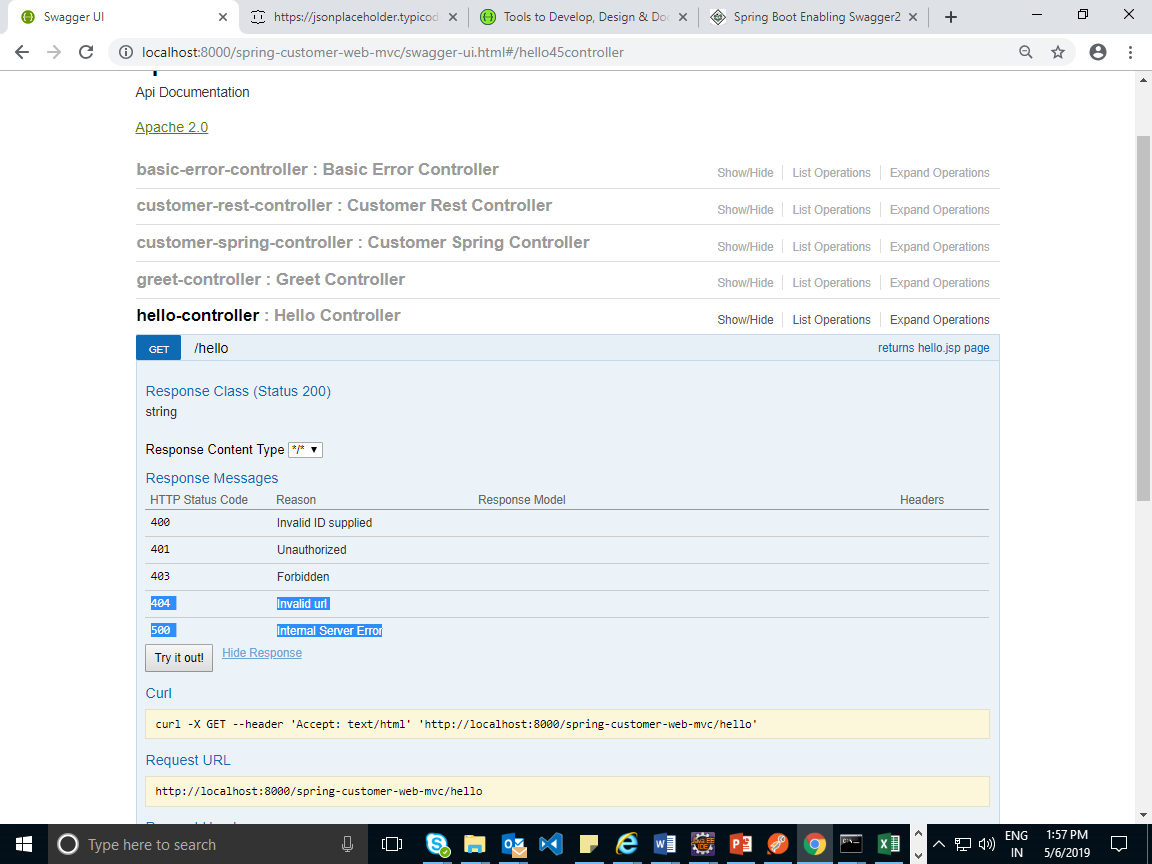


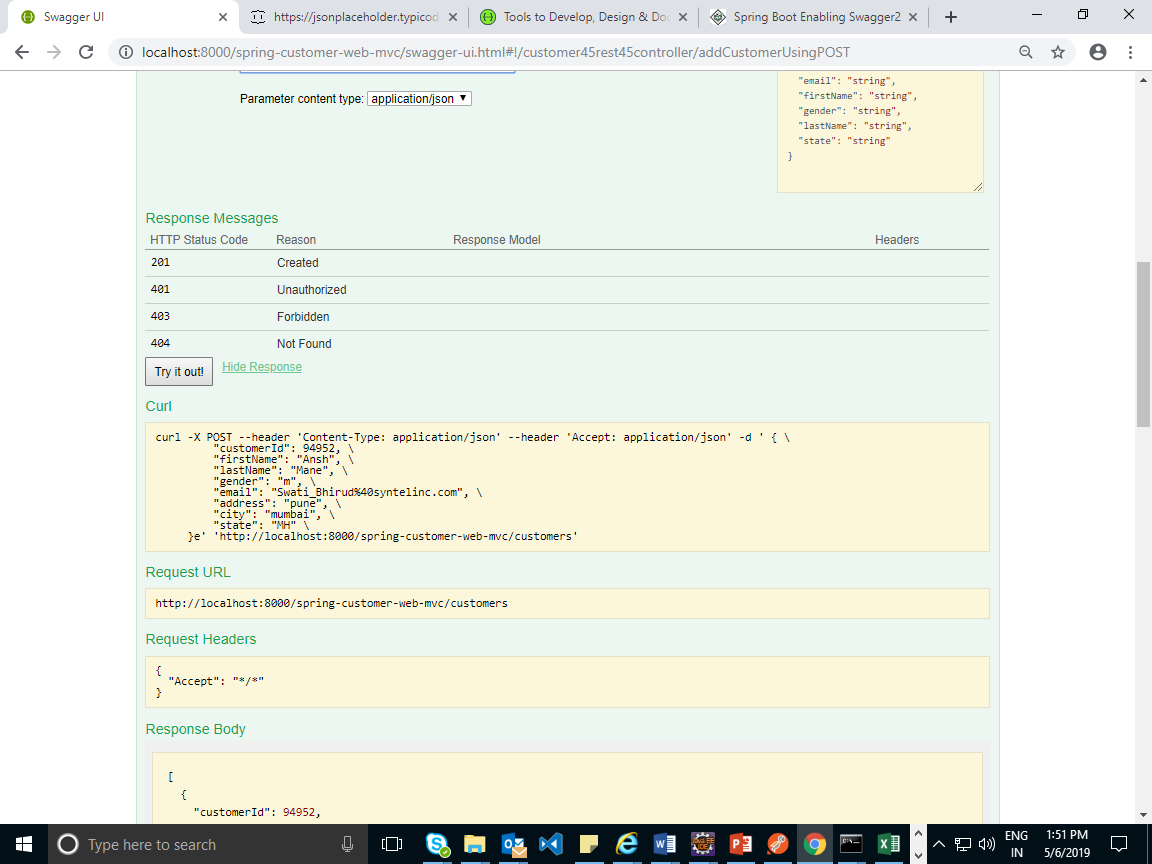
Click on try out button

Ouput



@ApiResponse: To add customized response





To display selected end points:

package com;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2// it will generated documnetion for all endpoints

@SpringBootApplication

public class SpringBootCustmomerAppV1Application {

public static void main(String[] args) {

SpringApplication.run(SpringBootCustmomerAppV1Application.class, args);

}

//specic annotation

@Bean

public Docket csutomerApi() {

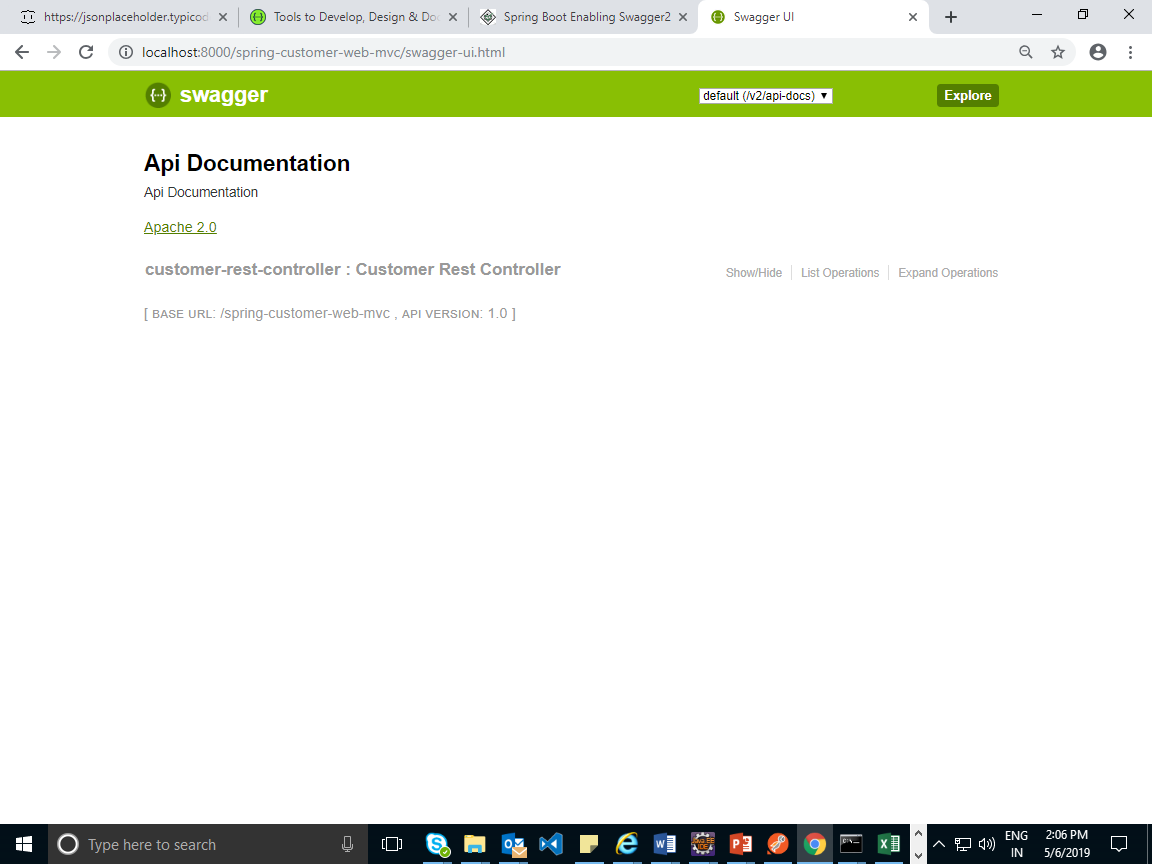
System.out.println("####### Generating swagger documentaation for customer restful api #########");

return new Docket(DocumentationType.SWAGGER\_2).select()

.apis(RequestHandlerSelectors.basePackage("com.atossyntel.cms.rest.controller")).build();

}

}



Actuator:

**Monitoring our app, gathering metrics, understanding traffic or the state of our database becomes trivial with this dependency.**

Actuator is mainly used to **expose operational information about the running application** – health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.

It is basically used to check the health of application for example suppose we have created 3 services order ,cancel ,payment services to check the traffic, database connection we use actuator in spring boot.

### ****Getting started****

To enable Spring Boot Actuator we’ll just need to add the spring-boot-actuator dependency to our package manager. In Maven:

[?](https://www.baeldung.com/spring-boot-actuators)

|  |  |
| --- | --- |
| 1  2  3  4 | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-actuator</artifactId>  </dependency>  When we add this dependencies in POM.xml  By default health ,info END POINTS are enabled |

Here are some of the most common endpoints Boot provides out of the box:

* */auditevents –* lists security audit-related events such as user login/logout. Also, we can filter by principal or type among others fields
* */beans – r*eturns all available beans in our *BeanFactory*. Unlike */auditevents*, it doesn’t support filtering
* */conditions –* formerly known as /*autoconfig*, builds a report of conditions around auto-configuration
* */configprops –* allows us to fetch all *@ConfigurationProperties* beans
* */env –* returns the current environment properties. Additionally, we can retrieve single properties
* */flyway –* provides details about our Flyway database migrations
* */health –* summarises the health status of our application
* */heapdump –* builds and returns a heap dump from the JVM used by our application
* */info –* returns general information. It might be custom data, build information or details about the latest commit
* */liquibase – b*ehaves like */flyway* but for Liquibase
* */logfile –* returns ordinary application logs
* */loggers –* enables us to query and modify the logging level of our application
* */metrics –* details metrics of our application. This might include generic metrics as well as custom ones
* */prometheus –* returns metrics like the previous one, but formatted to work with a Prometheus server
* */scheduledtasks –* provides details about every scheduled task within our application
* */sessions –* lists HTTP sessions given we are using Spring Session
* */shutdown –* performs a graceful shutdown of the application
* */threaddump –* dumps the thread information of the underlying JVM

<https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html>

the health endpoint provides basic application health information.Each individual endpoint can be [enabled or disabled](https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html#production-ready-endpoints-enabling-endpoints). To be remotely accessible an endpoint also has to be [exposed via JMX or HTTP](https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html#production-ready-endpoints-exposing-endpoints). Most applications choose HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

D:\ManishaSpringBoot\ManishaDemos\Day5\spring-boot-custmomer-app-mongodb-swagger-actuators

Add actuator configuration in application.properties file.

Server HTTP port.

server.port=8000

# Context path of the application.

server.servlet.context-path=/spring-customer-web-mvc

#register view resolver

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

spring.mvc.view.suffix=.jsp

# Mongo DB details

spring.data.mongodb.database=company

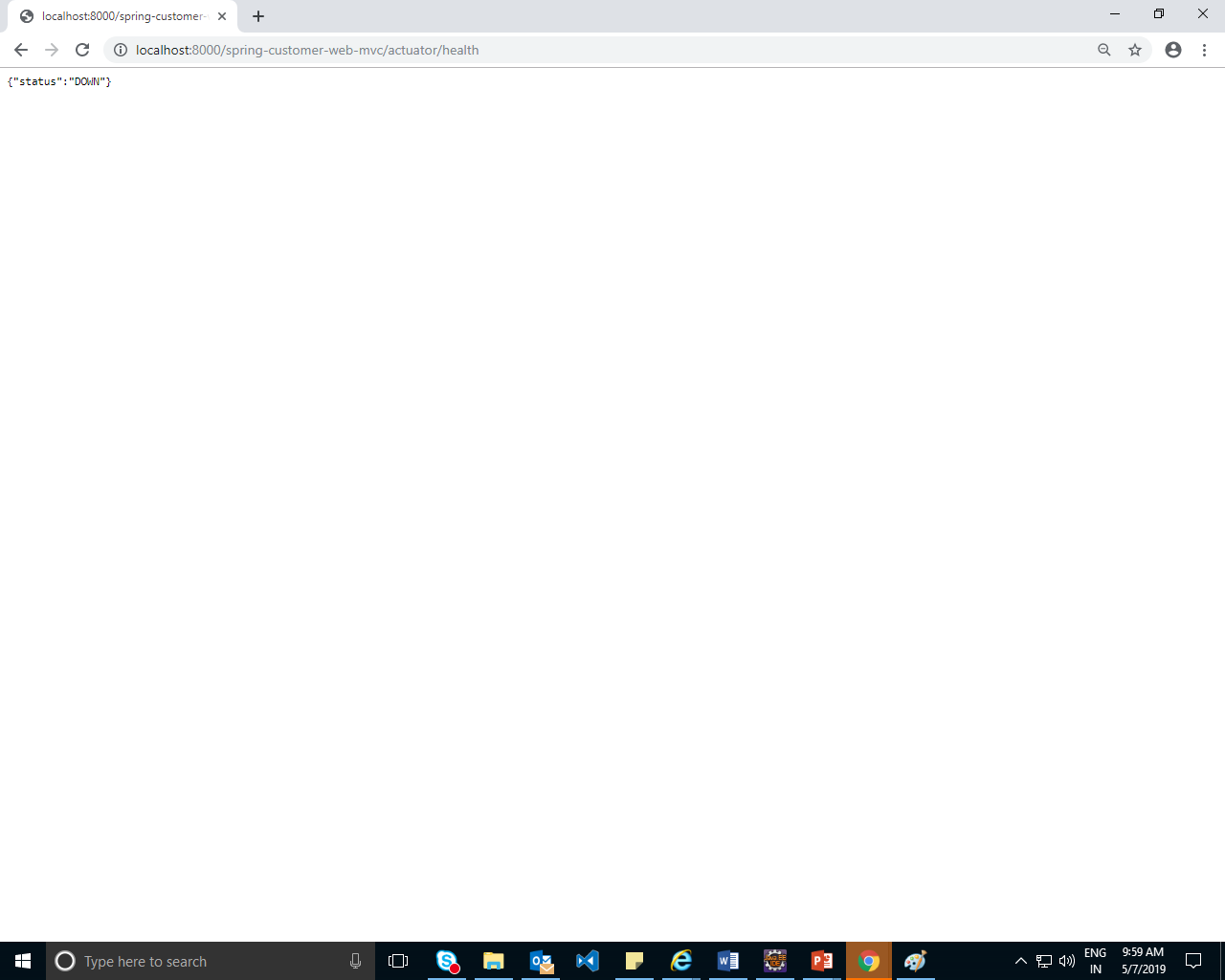
spring.data.mongodb.host=localhost

spring.data.mongodb.port=27017

#enable health and info

management.endpoints.web.exposure.include=\*

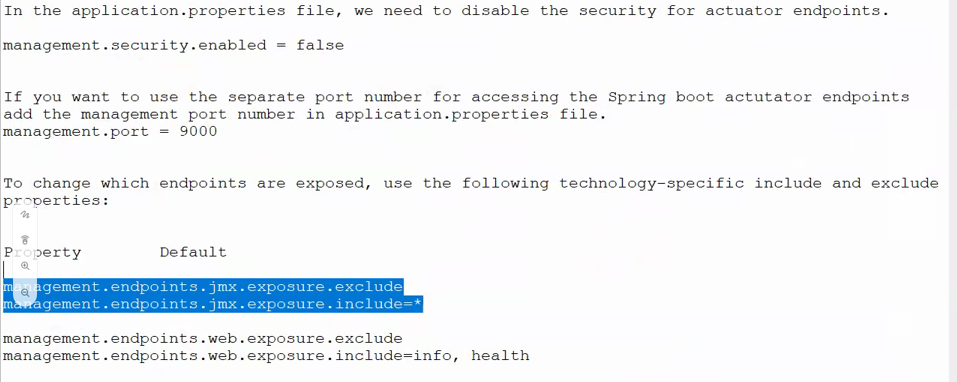
output:don’t start with MONGODB and execute application check helth of your application

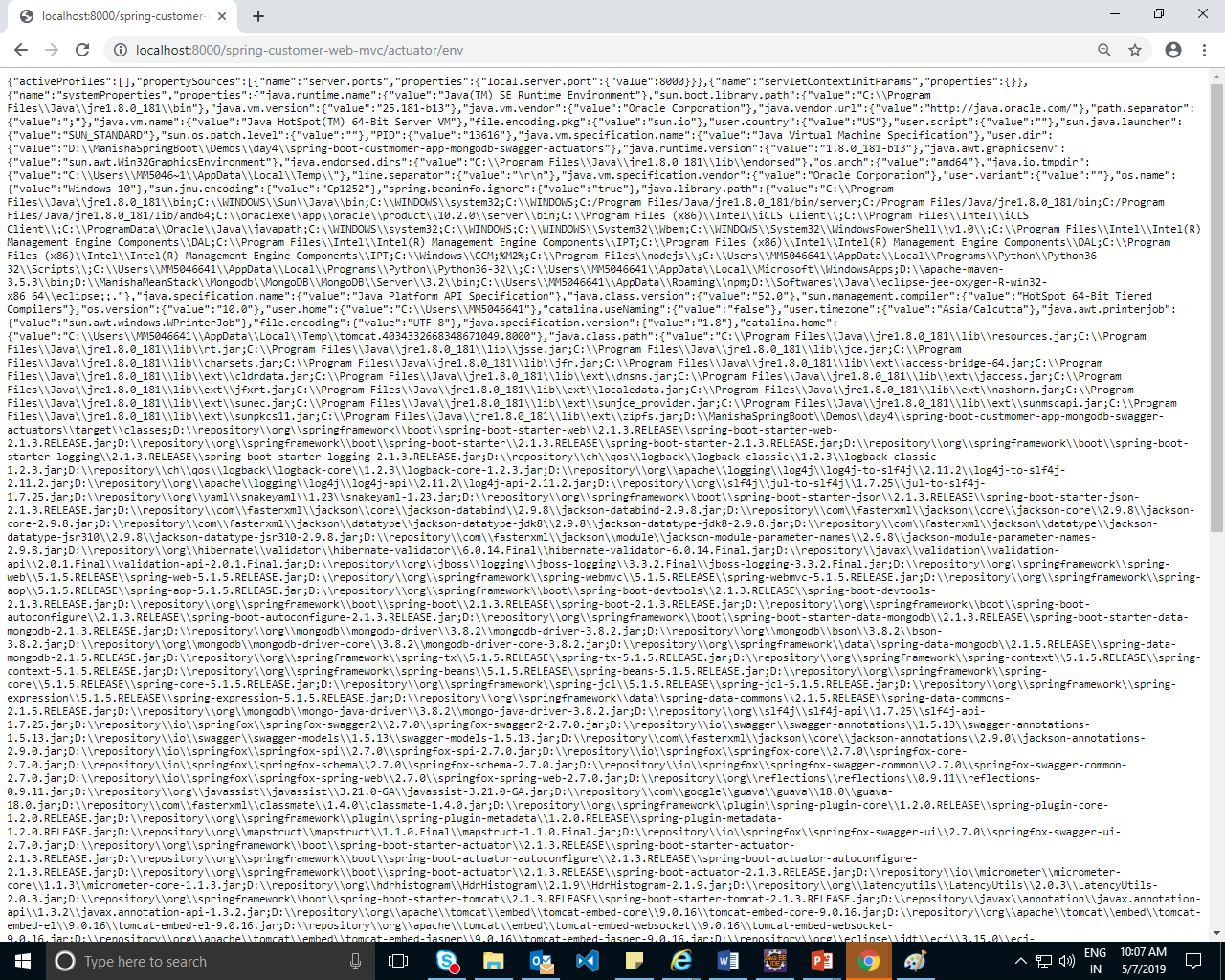


Example to enable all & disable health & info



You can set the management.server.port property to change the HTTP port





Jmx is for java management

Actuatos is used for administration purpose

Security:all endpoints are used directly we want to secure it so we are using securiry.

Add dependencies in POM.XML

File

<dependency>

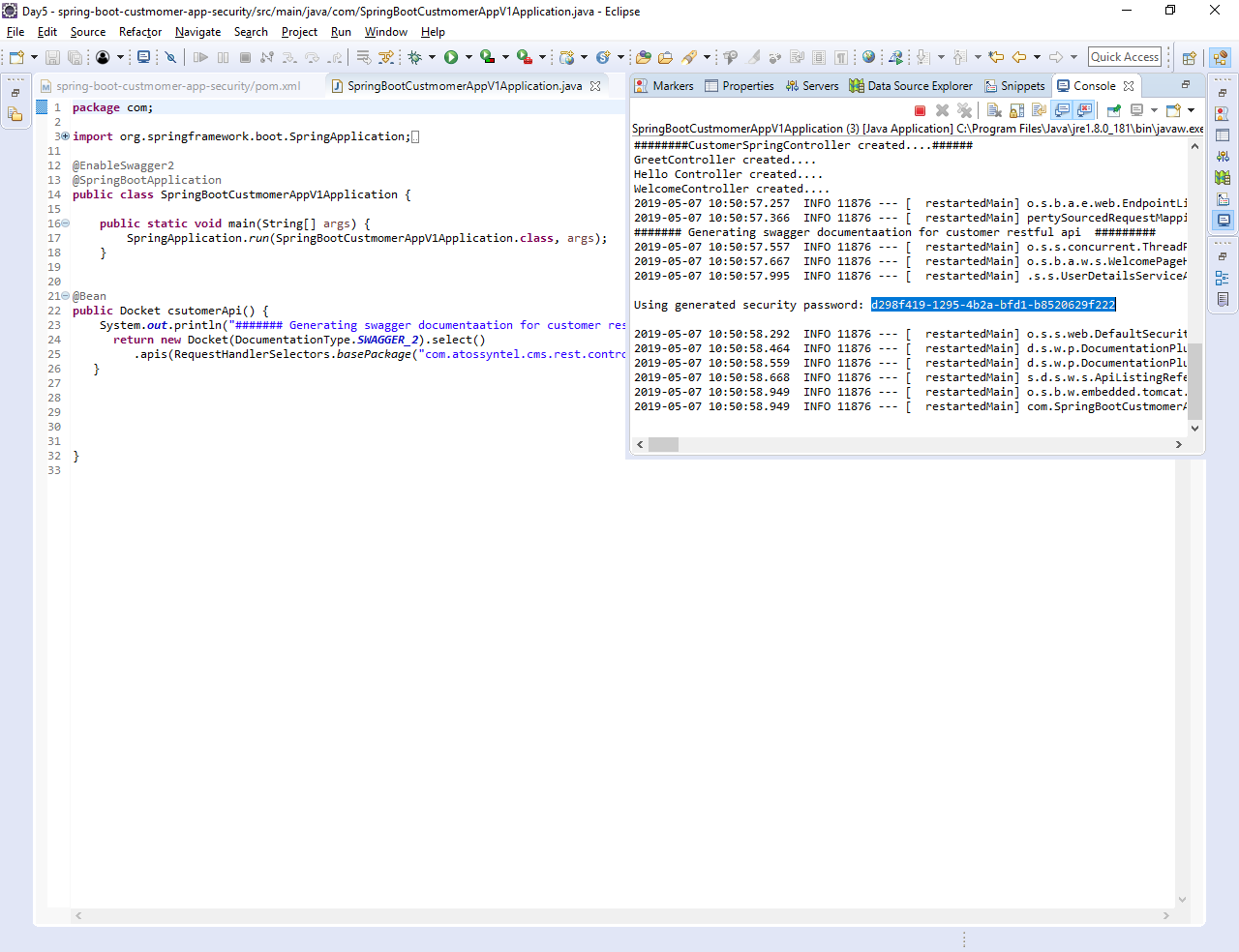
<groupId>org.springframework.boot</groupId>

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

</dependency>

Generates password

default user name is :user and dynamic password is generated by spring boot

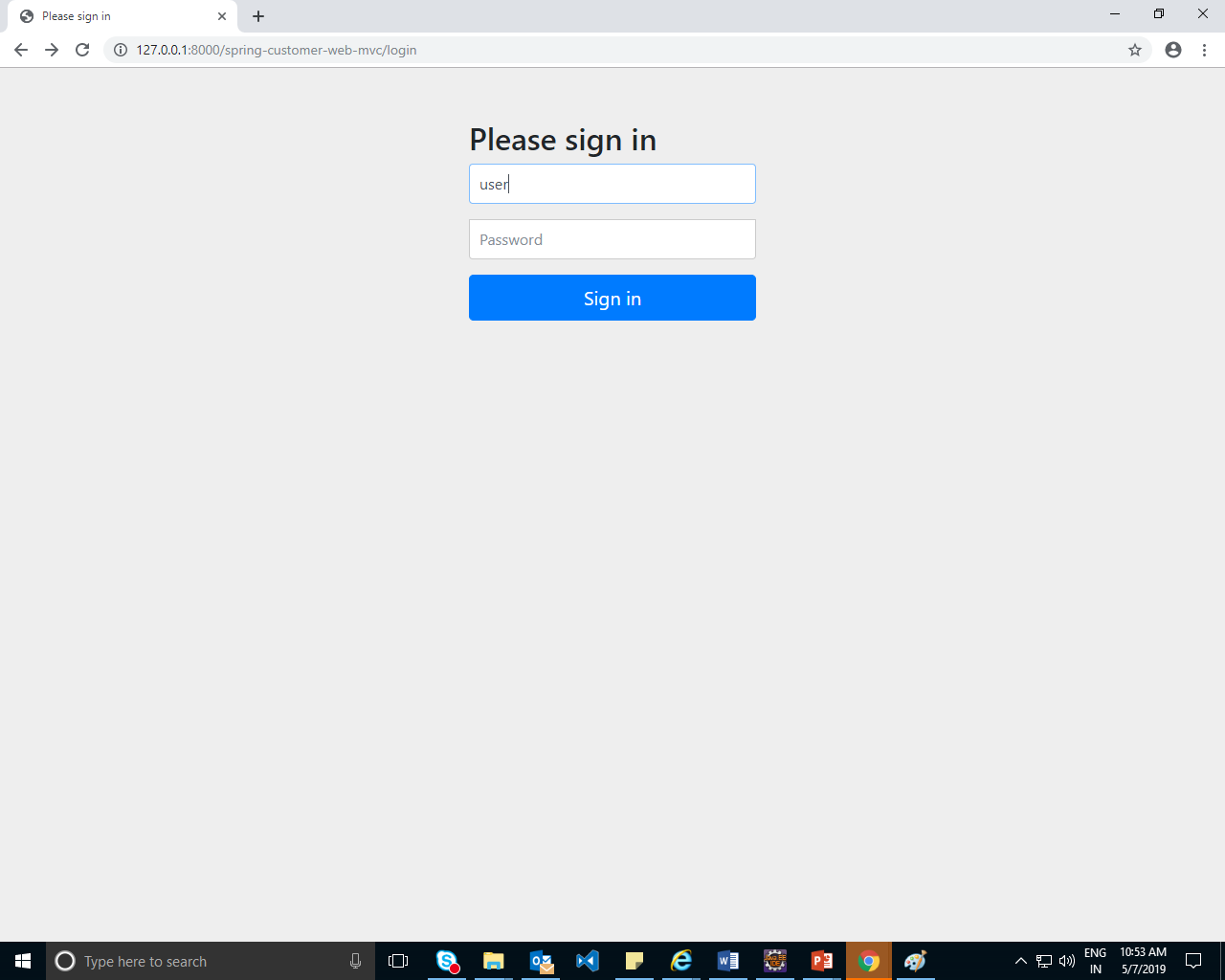


Customized the user credentials:

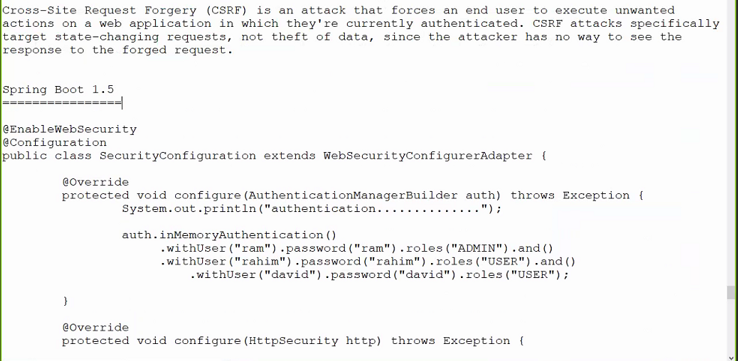
Set 2 properties in application.properties file

spring.security.user.name=manisha

spring.security.user.password=manisha@123



Authentication and Authorization configuration programitically:



* Csrf-cross site request forgery

Error

Spring Security – There is no PasswordEncoder mapped for the id “null” to do the below changes

In spring-security-core:5.0.0.RC1, the default PasswordEncoder is built as a DelegatingPasswordEncoder. When you store the users in memory, you are providing the passwords in plain text and when trying to retrieve the encoder from the DelegatingPasswordEncoder to validate the password it can't find one that matches the way in which these passwords were stored.

Use this way to create users instead.

User.withDefaultPasswordEncoder().username("user").password("user").roles("USER").build();

You can also simply prefix {noop} to your passwords in order for the DelegatingPasswordEncoder use the NoOpPasswordEncoder to validate these passwords. Notice that NoOpPasswordEncoder is deprecated though, as it is not a good practice to store passwords in plain text.

User.withUsername("user").password("{noop}user").roles("USER").build();

**package** com.security.config;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter

{

SecurityConfiguration()

{

System.***out***.println("SecurityConfiguration created");

}

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

System.***out***.println("authontication");

auth

.inMemoryAuthentication()

.withUser("sam").password("{noop}sam").roles("ADMIN").and()

.withUser("ram").password("{noop}ram").roles("USER").and()

.withUser("ham").password("{noop}ham").roles("USER");

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

System.***out***.println("authorization");

http.authorizeRequests()

.antMatchers("/hello").hasRole("ADMIN")

.anyRequest()//get put post

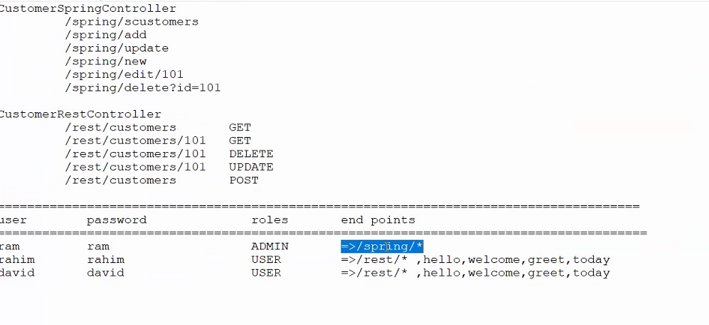
.fullyAuthenticated()

.and().httpBasic();//basic security

http.csrf().disable();//cross site request forgery unnecessary popups

}

}



FormBasedAuthontication

package com.security.config;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class SecurityConfiguration extends WebSecurityConfigurerAdapter

{

SecurityConfiguration()

{

System.out.println("SecurityConfiguration created");

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

System.out.println("authontication");

auth

.inMemoryAuthentication()

.withUser("sam").password("{noop}sam").roles("ADMIN").and()

.withUser("ram").password("{noop}ram").roles("USER").and()

.withUser("ham").password("{noop}ham").roles("USER");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

System.out.println("authorization");

http.authorizeRequests()

.antMatchers("/rest/\*").hasRole("ADMIN")

.antMatchers("/spring/\*").hasRole("USER")

.antMatchers("/hello","welcome","/greet/today").hasRole("USER")

.anyRequest()//get put post

//.fullyAuthenticated()

.permitAll()

//.and().httpBasic();//basic security

//http.csrf().disable();//cross site request forgery unnecessary popups

.and().formLogin();

}

}

REST ALL IS SAME!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

Find how to customize the login page

eureka server know as registry server

Zuul proxy api getway

–domain object such as insurance,banking

Example online shopping domain objects are customer,product,order etc.

Customer-mongodb

Product-mongodb