Spring tutorial document

What is the Spring frame work?

The **Spring Framework** is an application **framework** and inversion of control container for the **Java** platform. The **framework's** core features can be used by any **Java** application, but there are extensions for building web applications on top of the **Java** EE (Enterprise Edition) platform. ... The **Spring Framework** is open source.

Note : tomcat port kill netstat –ano | findstr :8080

Taskkill /PID *** /F**

Application context : Application context will load the context.xml file , its load the every thing

Bean factory : bean factory will little more then application context file

Bean scope : bean scope will declare in the context.xml file it will give the scope attribute it will be produce the new instance each and every time.

Prototype :this is the stereo type of object this scope will do instantiate the each and every time each time it will be instantiate the new bean object

For example : when your clicking the button multiple times each and every time it will be create the new instantiated object

Singleton : this is the default scope of the IOC container object its exactly single instantiate the object

Lazy loading and Pre loading : in spring contain two type methods in

Init() : it will be get execute immediately after bean is instantiated

Destroy() it will be destroy the just before

```
  <bean id ="welcome" class="org.purinis.Model.Restaurant"  init-
method="init" destroy-method="destry"></bean>
```

```
public void init () {
     System.     out.println("Bean is going to through
init");
     }

     public void destry () {
     System.     out.println("Bean is going to through
destry");
     }
     }
```

Note : in main method after the Application context file we will write the

This s Statement

```
((AbstractApplicationContext)context).registerShutdownHoo
k();
```

Bean Factory : bean factory is the container which is helps to change the property values we can write the files to pass the bean value in context.xml file

Note : we can implement the BeanFactoryPostProcessor

```
@Override
     public void
postProcessBeanFactory(ConfigurableListableBeanFactory
beanFactory) throws BeansException {

     BeanDefinition      beanDefinition =
beanFactory.getBeanDefinition("restaruantBean");
     MutablePropertyValues      mutablePropertyValues =
beanDefinition.getPropertyValues();



mutablePropertyValues.addPropertyValue("welcomeText",
"Welcome to Bean Factory");

     mutablePropertyValues.addPropertyValue("location",
"Banglore");
```

Spring MVC

M – model -> represent the data

V – view **->** represent the ui part

C – controller -> manage the application flow makes a call to some sort of service produce the model and then model to the view

Note : spring frame work provide two way to create the controller

One is extended abstract class

Second is annotation based

what is the multiaction controller ?

if controller having more then one controller method , its called as multi action controller

@PathVariable : is a Spring annotation which indicates that a method parameter should be bound to a URI template variable.

@RequestParam : Spring MVC **RequestParam** Annotation. In **Spring** MVC, the @**RequestParam** annotation is used to read the form data and bind it automatically to the parameter .

@ModelAttribute : annotation is used as part of a Spring MVC web app and can be used in two scenarios.

1. Firstly, it can be used to inject data objects in the model before a JSP loads. ...
2. Secondly, it can be used to read data from an existing model, assigning it to handler method parameters.

Form Validation :- this form validation restack the field min and max value if your unfortunately cross the valid length it will never accept the value

Sprig has provide the some of the annotation like

@Size(min=2 ,max=5) at the same time you have to use @Valid annotation with

Note : if you want to give your won error message you can give like this statement in pojo class

@Size(min=2,max=20, message =”write here your won message ..!!”)

(OR)

We can put the same error message as a properties file

Steps : 1 write the message.properties file

2 we need to add some of the springcontext.xml file

3 <bean id =”messageSource”

Class =”org.springframework.context.support.RealoadableResourceBundleMessageSource”>

<property name=”beanName” value=”/WEB-INF/propertyFileName write here”/>

</bean>

Note : in properties you have to write Annotation Name object reference and variable name

Exe : Size.student.name = “ your won exception message write here ”

@ModelAttribute,

This will take the third party helper either JSR(Java specification request) JCP(Java communication Process)

@Pattern(regexp="[^0-9]*") this is the form binding whenever your going to perform any name validation including any num it will throw some of the err message

For exe : String name =”kiran”;

Unfortunately your adding some of the numb “kir2an” like this it will throw some of the err message

@Past : if you want to give the feature or future numb it wont allow to the past value you just mention in the pojo class on top up the valuable @Past annotation

Class Student{

@Past

Private Date date;

}

@Max(222) : this is also using to validation form only

@min(100) : this is also using to validation form only

Interceptor

Spring Interceptor are used to intercept client requests and process them. Sometimes we want to intercept the HTTP Request and do some processing before handing it over to the controller handler methods. That's where **Spring MVC Interceptor** come handy.

Steps to fallow the Interceptor

- Included the java class which extended the HandlerInterceptorAdaptor class and Override the One of the method with the name **preHandle**
- Write the code in preHandle method which you want to spring MVC to execute before handling the request
- Put an entry of this newly added java class in the spring 's conformation file

Note : - Class Extended

```
public class DayOfWeekBasedAccessInteraptor  extends
HandlerInterceptorAdapter{

     public boolean preHandle(HttpServletRequest request,
HttpServletResponse response) throws Exception{

     Calendar      cal = Calendar.getInstance();
```

```
          int dayOfWeek =
cal.get(cal.getWeeksInWeekYear());
          if(dayOfWeek==1) {

               response.getWriter().write("the website is
closed on sunday  please try after some week days ");

               return false;

     }

          return true;

     }
}
```

```
// this tow method will be print on console

public void postHandle(HttpServletRequest request,
HttpServletResponse response,
     ModelAndView           modelAndView) throws
Exception{

     System.     out.println("HandlerInterceptorAdapter"
+ request.getRequestURI().toString());


     }


     public void afterCompletion(HttpServletRequest
request, HttpServletResponse response,
     ModelAndView           modelAndView) throws
Exception{

     System.     out.println("HandlerInterceptorAdapter"
+ request.getRequestURI().toString());


     }
```

Note : Wirte the Configurtation.xml file

```
<mvc:interaptors>
     <bean

     class="com.purinis.controller.DayOfWeekBasedAcces
sInteraptor" />
     </mvc:interaptors>

</bean>
```

Note : if you want to stop the specification url only then you need to follow these steps to put inside the configuration.xml file writing the mapping

```
  <mvcinteraptors>

          <mvcinteraptor>
     <mvc:mapping path="/admitionForm.html" />
     <bean

     class="com.purinis.controller.DayOfWeekBasedAccessInt
eraptor" />
          </mvcinteraptor>
     </mvc:interaptors>

     </bean> -->
```

Internalization and Localization : spring will support the multiple language we can write the properties look into the google example code;

Spring Exception Handling

Exception handling is very essential feature of any Java application. Every good open source framework allows to write the exception handlers in such a way that we can separate then from our application code. Well, Spring framework also allows us to do so using annotation **Spring @ExceptionHandler**.

`@ExceptionHandler` annotation is used for handling exceptions in specific handler classes and/or handler methods.

To handle exceptions in String MVC, we can define a method in controller class and use the annotation *@ExceptionHandler* on it. Spring configuration will detect this annotation and register the method as exception handler for **argument exception class and its subclasses**.

```
@ExceptionHandler(value = NullPointerException.class)
     public String
handlerNullPointerException(Exception e) {
     System.     out.println("Null pointer exception "
+ e);
          return "NullPointerException ";
     }
```

JSP.page:

```
<html>


<body>

     <h1>Purinis College Of Engineering</h1>

     <H3>
     <p>Application has Enconter Nullpointer Exception
Please contact support team </p>

</H3>
</body>
</html>
```

Note : your controller class might be lot of exception class handle here

Either Io, null, arithmetic…etc , you need to write the separate jsp file for each and every exception and separate @ExceptionHandlerAnnotatlion method()

Note : Generic Exception can handle the all Exception single Exception handler can handle all the exception

Exe :

```
     @ExceptionHandler(value = Exception.class)
     public String handlerExceptionsGeneric(Exception e)
{
     System.     out.println("Exception Handling " + e);
          return "Exception"; // jsp page this exception
     }
```

JSP Page :

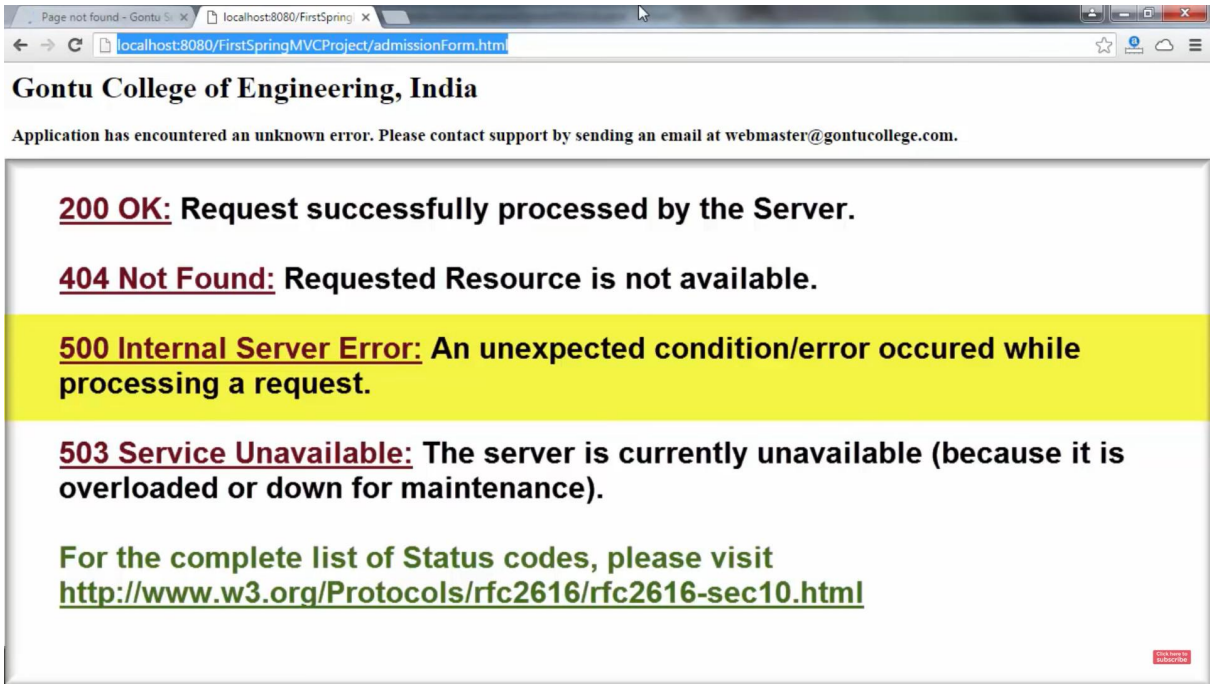
```
<html>


<body>

      <h1>Purinis College Of Engineering</h1>

      <H3>
      <p>Application has Enconter Exception  Please contact support team
</p>

</H3>
</body>
</html>
```

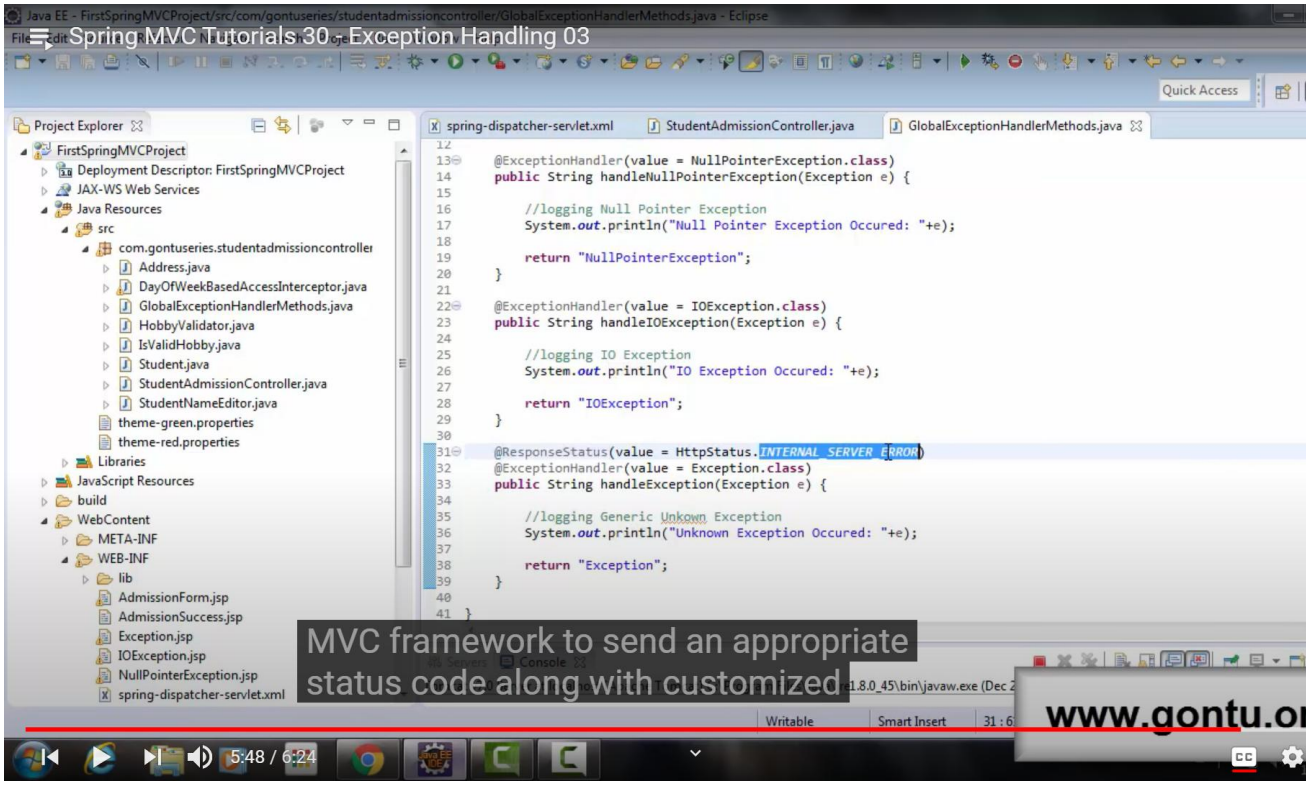
Globally Exception in spring

ControllerAdvice class :

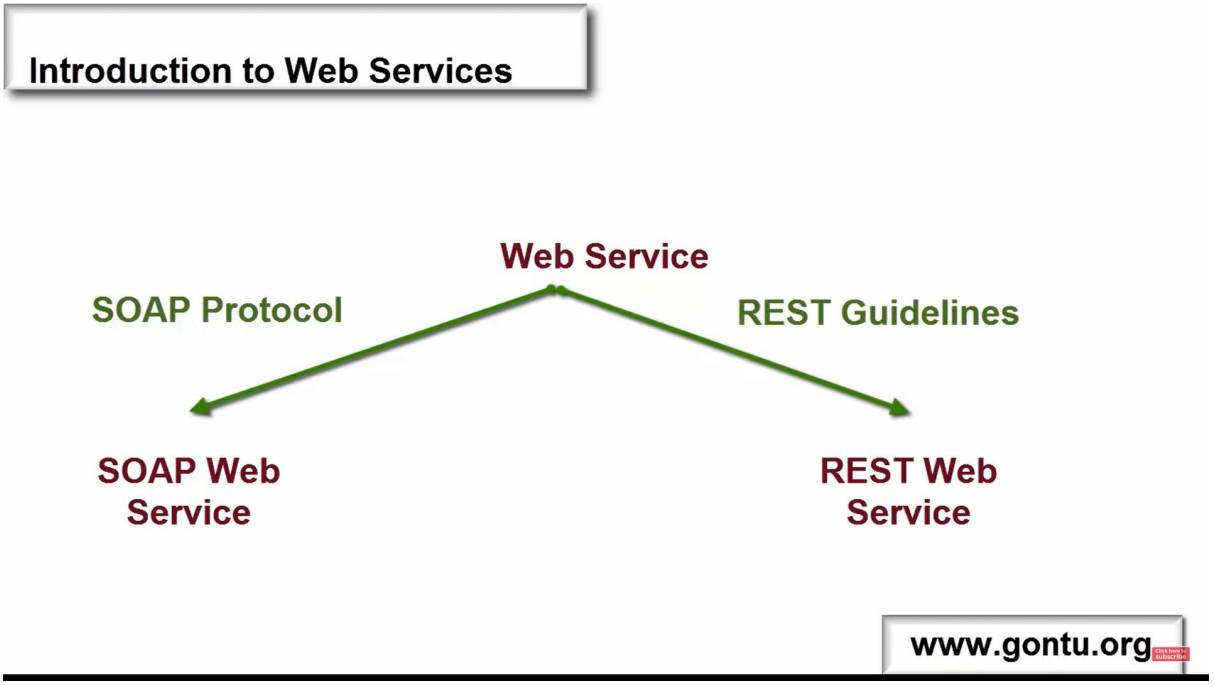
This calss can be handle the all the exceptions we can write all the exceptions at one place,

Write the spepare clsass and add the **@ControllerAdvice** annotation this annotation can handle the all the exception in spring

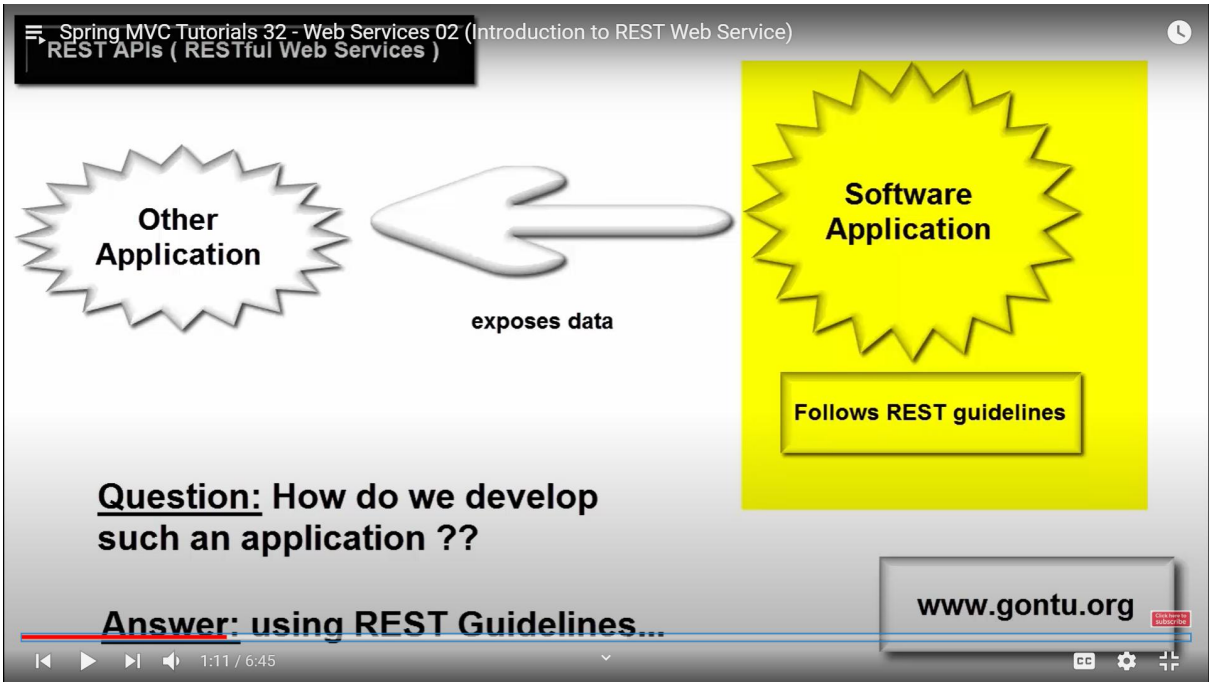


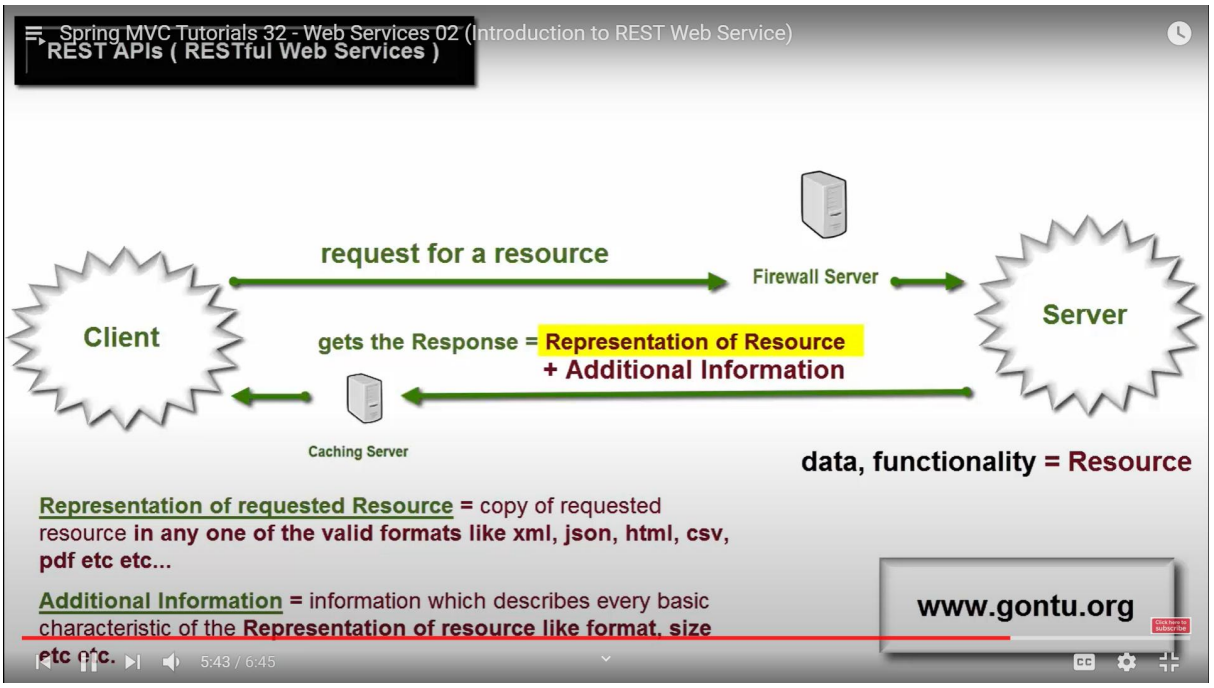


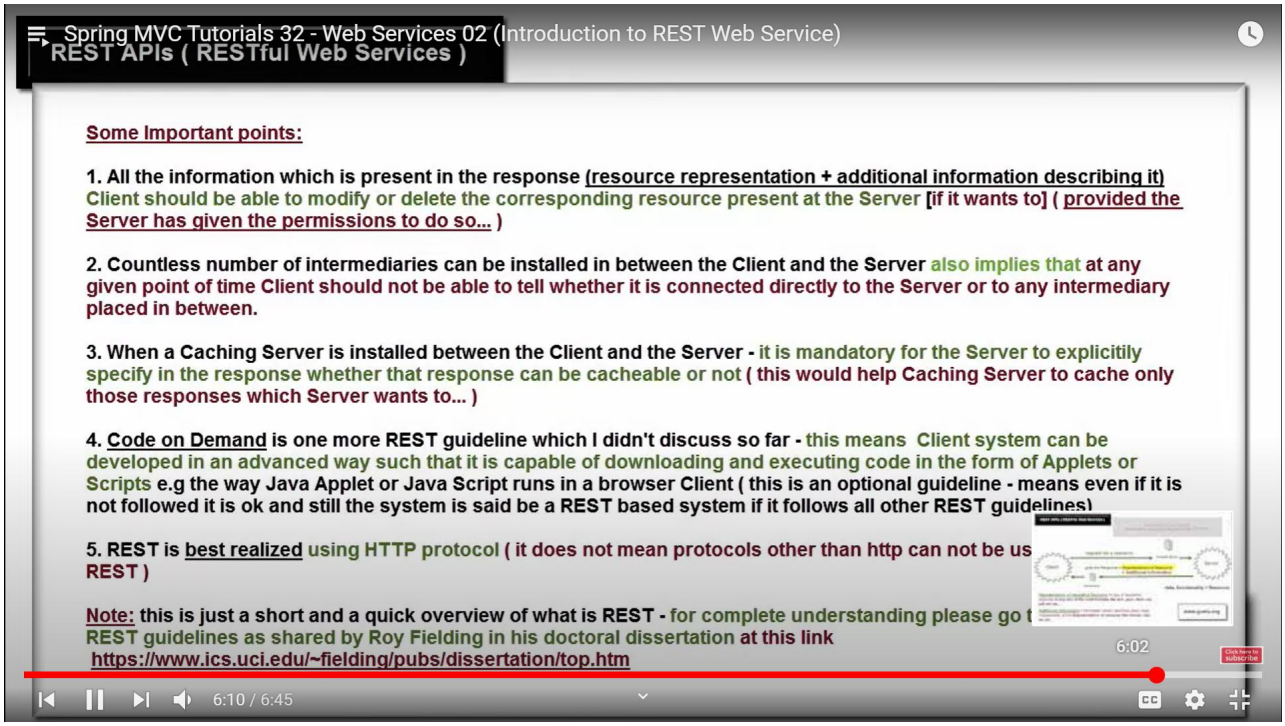
Web Service_Introduction

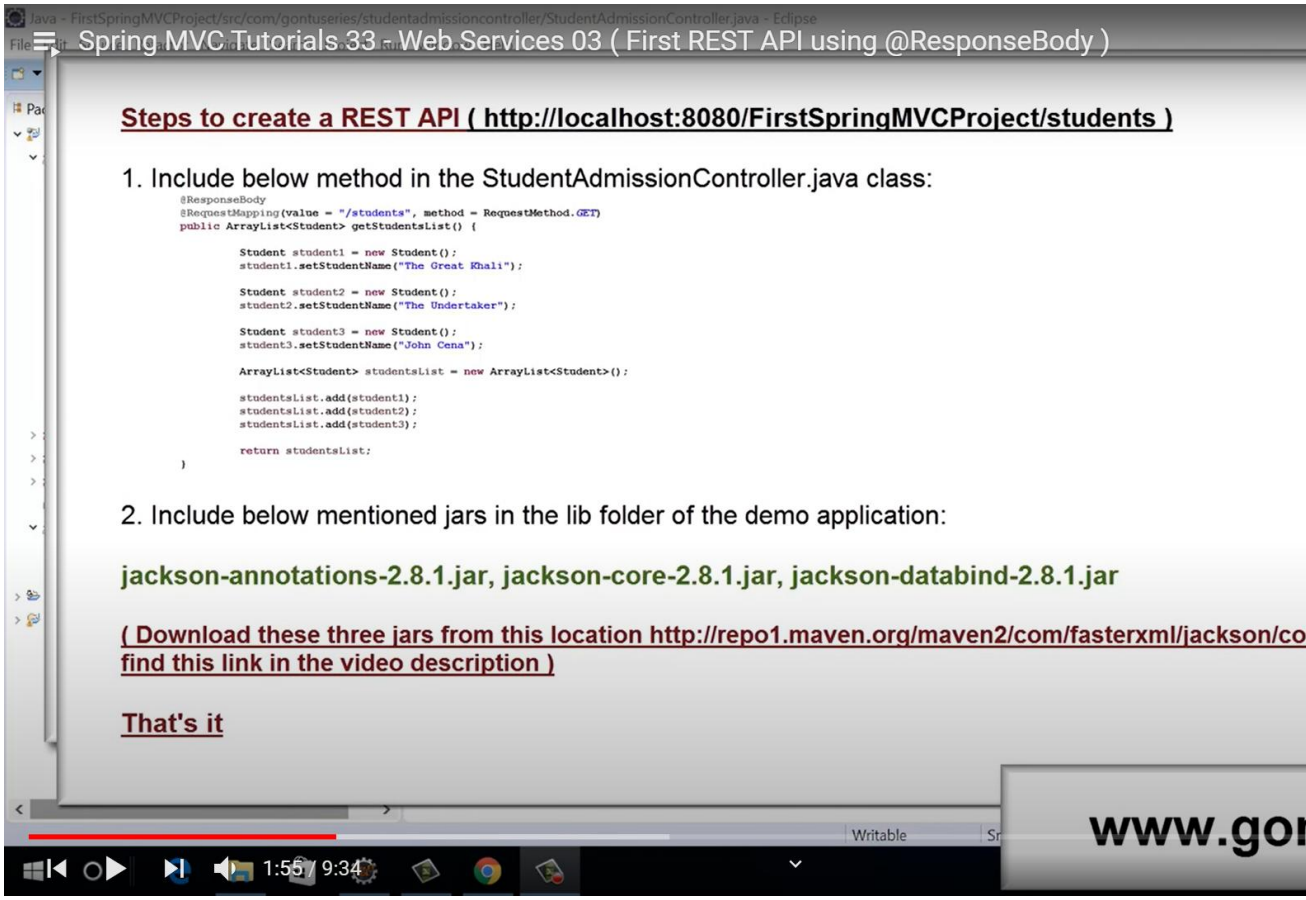


Restful web service


Spring MVC Tutorials 32 - Web Services 02 (Introduction to REST Web Service)
REST APIs (RESTful Web Services)
Other Application
exposes data
Software Application
Follows REST guidelines
Question: How do we develop such an application ??
Answer: using REST Guidelines...
www.gontu.org
1:11 / 6:45

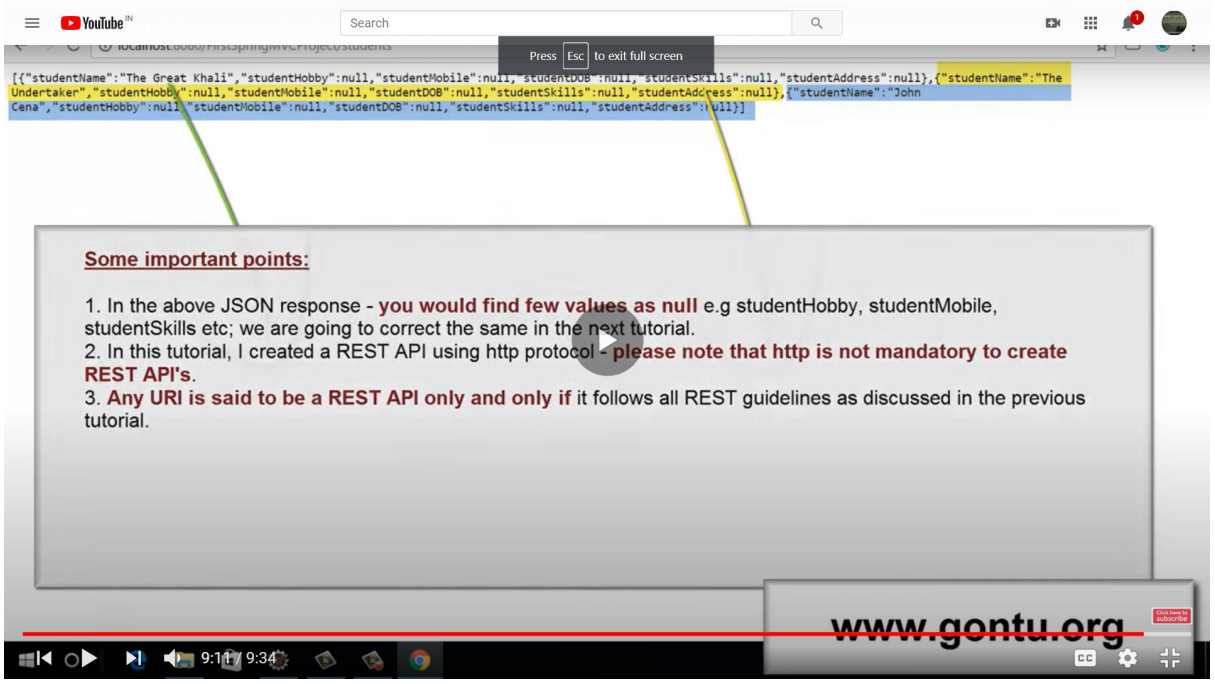

Spring MVC Tutorials 32 - Web Services 02 (Introduction to REST Web Service)
REST APIs (RESTful Web Services)
request for a resource
Firewall Server
Client
Server
gets the Response = Representation of Resource
+ Additional Information
Caching Server
data, functionality = Resource
Representation of requested Resource = copy of requested resource in any one of the valid formats like xml, json, html, csv, pdf etc etc...
Additional Information = information which describes every basic characteristic of the Representation of resource like format, size etc etc.
www.gontu.org
5:43 / 6:45


Spring MVC Tutorials 32 - Web Services 02 (Introduction to REST Web Service)
REST APIs (RESTful Web Services)
Some Important points:
1. All the information which is present in the response (resource representation + additional information describing it) Client should be able to modify or delete the corresponding resource present at the Server [if it wants to] (provided the Server has given the permissions to do so...)
2. Countless number of intermediaries can be installed in between the Client and the Server also implies that at any given point of time Client should not be able to tell whether it is connected directly to the Server or to any intermediary placed in between.
3. When a Caching Server is installed between the Client and the Server - it is mandatory for the Server to explicitily specify in the response whether that response can be cacheable or not (this would help Caching Server to cache only those responses which Server wants to...)
4. Code on Demand is one more REST guideline which I didn't discuss so far - this means Client system can be developed in an advanced way such that it is capable of downloading and executing code in the form of Applets or Scripts e.g the way Java Applet or Java Script runs in a browser Client (this is an optional guideline - means even if it is not followed it is ok and still the system is said be a REST based system if it follows all other REST guidelines)
5. REST is best realized using HTTP protocol (it does not mean protocols other than http can not be us
REST)
Note: this is just a short and a quick overview of what is REST - for complete understanding please go t
REST guidelines as shared by Roy Fielding in his doctoral dissertation at this link
https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
6:02
6:10 / 6:45


Spring MVC Tutorials 33 - Web Services 03 (First REST API using @ResponseBody)
Steps to create a REST API (http://localhost:8080/FirstSpringMVCProject/students)
1. Include below method in the StudentAdmissionController.java class:
@ResponseBody
@RequestMapping(value = "/students", method = RequestMethod.GET)
public ArrayList<Student> getStudentsList() {
Student student1 = new Student();
student1.setStudentName("The Great Khali");
Student student2 = new Student();
student2.setStudentName("The Undertaker");
Student student3 = new Student();
student3.setStudentName("John Cena");
ArrayList<Student> studentsList = new ArrayList<Student>();
studentsList.add(student1);
studentsList.add(student2);
studentsList.add(student3);
return studentsList;
}
2. Include below mentioned jars in the lib folder of the demo application:
jackson-annotations-2.8.1.jar, jackson-core-2.8.1.jar, jackson-databind-2.8.1.jar
(Download these three jars from this location http://repo1.maven.org/maven2/com/fasterxml/jackson/co
find this link in the video description)
That's it
Writable
1:55 / 9:34
www.gon

Note :

@ResponseBody : The @ResponseBody annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the HttpResponse object.



@JsonProperty("student_Name") :-

this annotation will change the current pojo class property will change after the adding

```
@JsonProperty("student_Name")
```

Now it will display property name is **(student_Name)**

@JsonProperty("student_Name") :-

This order property will display the pojo class properties as custom order format

@JsonIgnoreProperties({"studentSkills"}) :-

This property is not print the object in console these **@JsonIgnoreProperties** avoid the to print the object in console

@JsonInclude(JsonInclude.Include.*NON_NULL*) :-

These annotation will do avoid the Null value to print the console

Get Method() :-

```
@ResponseBody // The @ResponseBody annotation tells a
controller that the object returned is
                          // automatically serialized into
JSON and passed back into the HttpResponse
                          // object.
     @RequestMapping(value = "/students", method =
RequestMethod.GET)
     public ArrayList<Student> getStudentList() {

     Student       student = new Student();
          student.setStudentName("Manoja");

     Address       address = new Address();
          address.setCity("Nellore");
          address.setCountry("US");
          address.setStreet("Blab");
          address.setPincode(45632);

     Student       student2 = new Student();
          student2.setStudentName("Kiran Kumar");

     Student       student3 = new Student();
          student3.setStudentName("Heemaja");

     ArrayList<Student>       list = new
ArrayList<Student>();
          list.add(student3);
          list.add(student2);
          list.add(student);
          return list;
     }
```

Get Single User returned() :-

```
// return single user details
     @ResponseBody
     @RequestMapping(value = "/students/{name}",method
=RequestMethod.GET)
     public Student
getSingleOperation(@PathVariable("name") String
studenName) {

     Student       student = new Student();
          student.setStudentName(studenName);
          student.setStudentHobby("WWW");

          return student;
```

```
}
```

@RestController :-

If you make it @RestController on top-up java class, whatever method you have indicated inside @RestController class, all methods related to basically related to rest Api, for any of the methods which you are going to include such a Rest controller class, you're not required to explicitly mention @ResponseBody annotation

Note : - these jars required to work on Rest API

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.8.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.8.1</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.8.1</version>
</dependency>


<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.dataformat/jackson-dataformat-xml -->
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
    <version>2.8.1</version>
</dependency>


<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-validator -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.3.Final</version>
```

```
</dependency>

<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.module/jackson-
module-jaxb-annotations -->
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jaxb-annotations</artifactId>
    <version>2.8.1</version>
</dependency>
```

Note :- if your application restrict any one type of response you can you use the

```
produces = MediaType.APPLICATION_ATOM_XML_VALUE
```

exe :-

```
     @RequestMapping(value = "/students", method =
RequestMethod.GET,produces =
MediaType.APPLICATION_ATOM_XML_VALUE)
```

PUT() :- this method will do whenever client can update certain information , that time its using put method()

GET() :-this method will do whenever client can read certain information , from the server those kind of situation we can use the Get method()

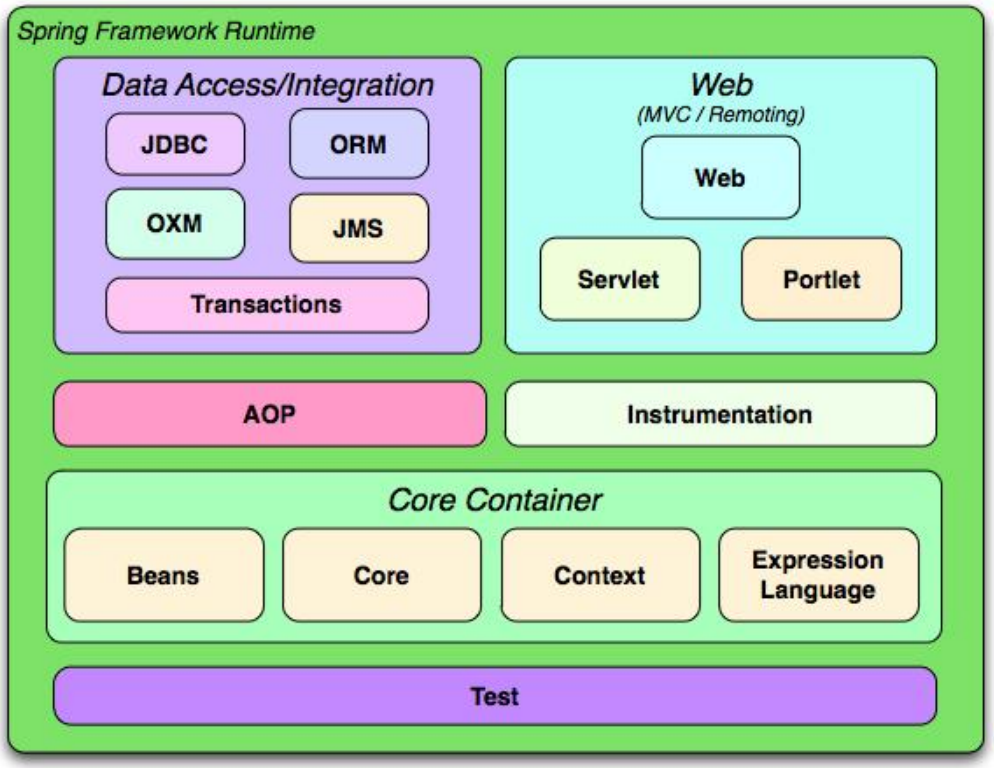
@POST() :- if you want to create the new client request, then can go for it

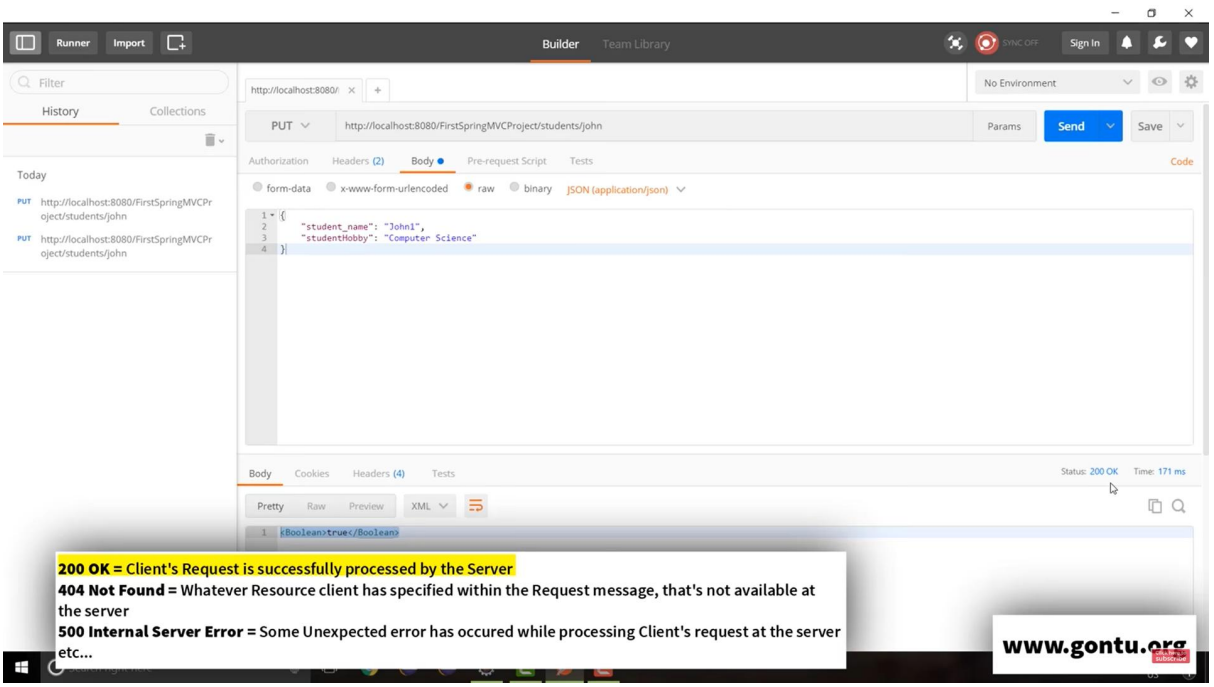
@DELETE() : The HTTP DELETE method is used to delete a resource from the server. Unlike GET and HEAD requests, the DELETE requests may change the server state.

@RequestBody :- this annotation would be convert the specific java object, if request come Json object it will convert the Json to Java Object,
If its come xml format, it will convert to xml to Java Object in the method

@Consumes :- These annotation is used to specify which MIME media types of representations a resource can accept, or consume, from the client. If @Consumes is applied at the class level, all the response methods accept the specified MIME types by default.

Spring Modules :



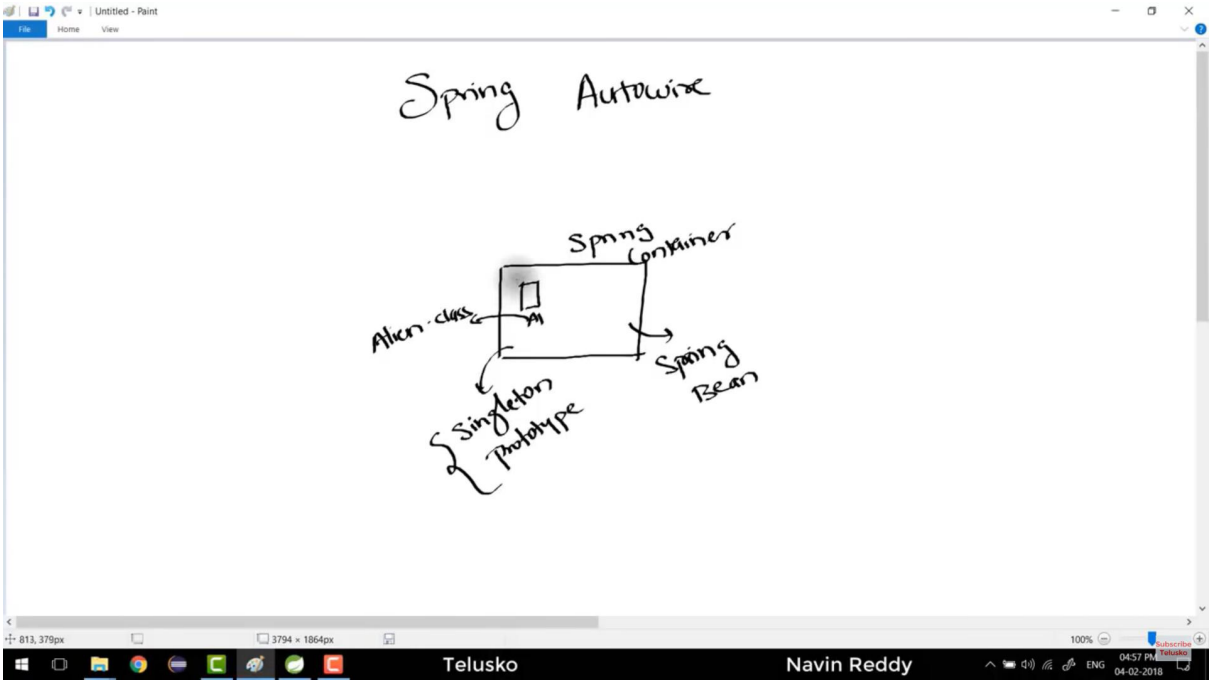



```
public ResponseEntity<Void> :- this method doesn't return
any response body, its return only Http Status code,
```

for more Http status code.

Please visit this url :
https://www.restapitutorial.com/httpstatuscodes.html

Spring Boot Tutorial



. *@Component annotation*

`@Component` annotation marks a java class as a bean so the component-scanning mechanism of spring can pick it up and pull it into the application context. To use this annotation, apply it over class as below:

@**Scope() :** here we can define the scope of the Bean object

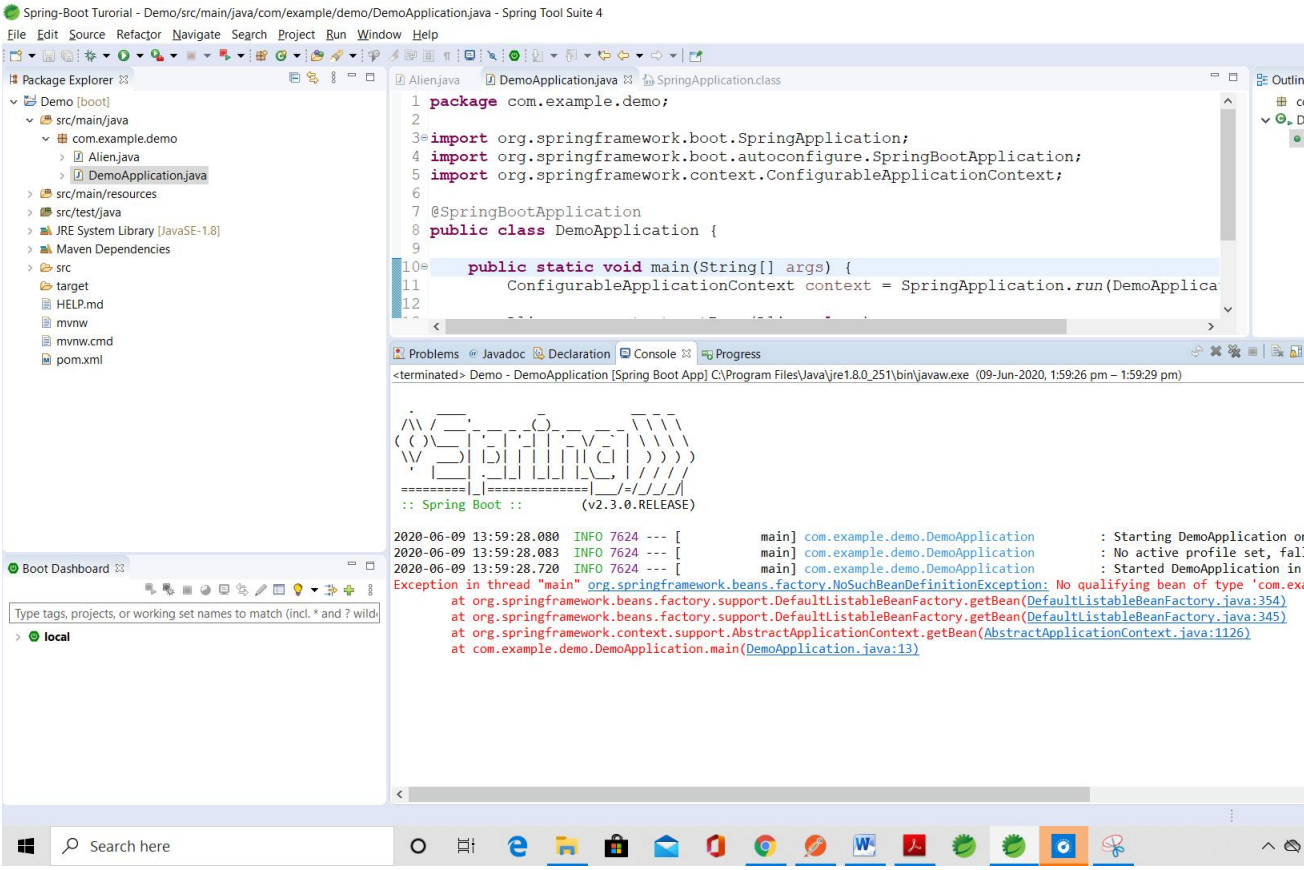
Exe : @Scope(value="prototype") this can be create the each and every time new instantiation

@Singletone : by default spring provide the single tone it can initiate the once how many time your going to be performance or create object

@**Autowire** :- Marks a constructor, field, setter method or config methods, this will try to search for the Object in the spring container

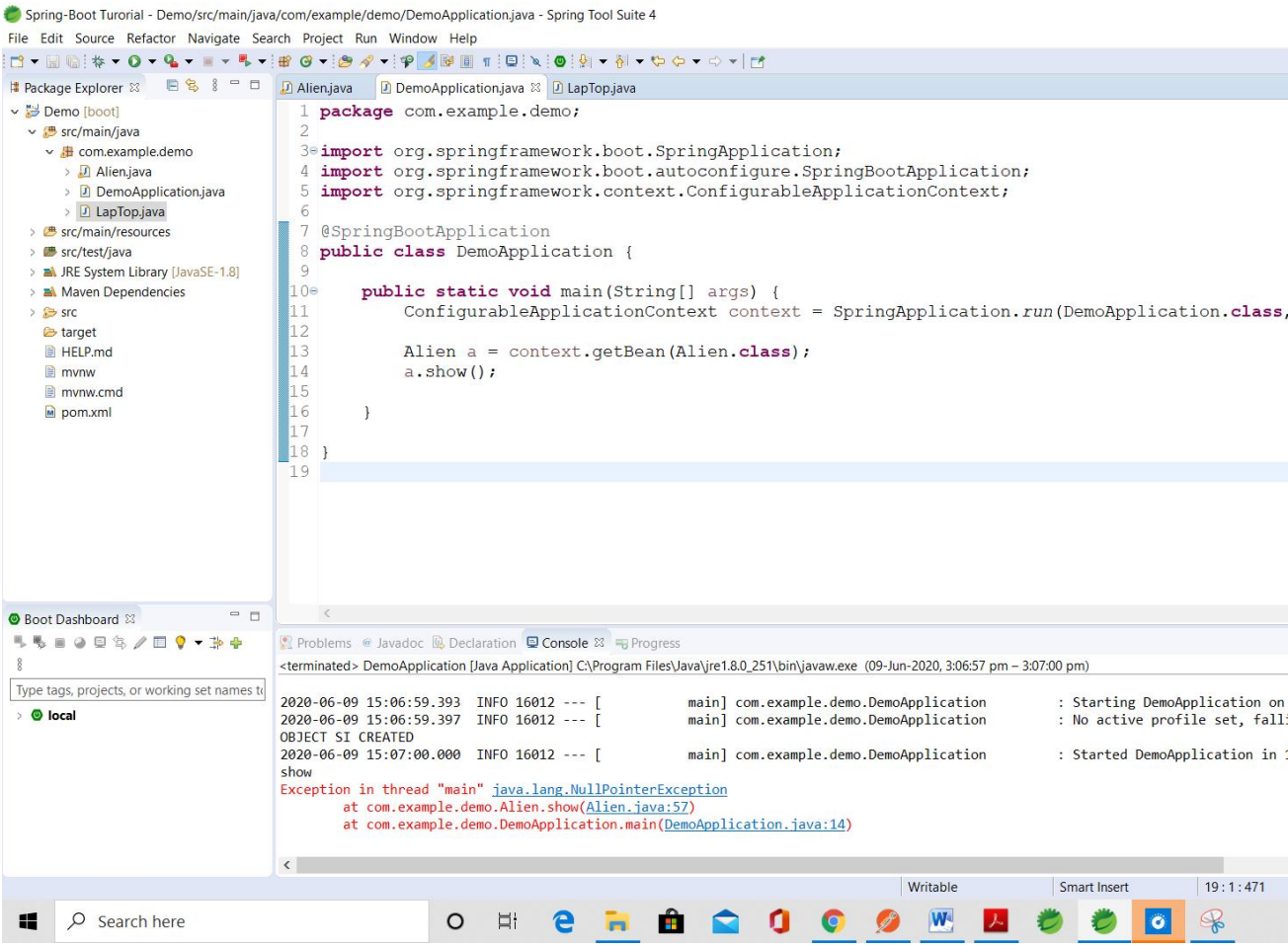
Error :

If you don't use the component annotation bean cant be find the object then you may get the error like this



Note : if your calling one calls to another call with out using @Autowire and component annotation you should get this exception

Error :-



@Qualifier :- There may be a situation when you create more than one bean of the same type and want to wire only one of them with a property. In such cases, you can use the **@Qualifier** annotation, search by the name of bean

Note : By default @Autowire search by a type and @Qualifier will search by a name

Note :- By default Spring boot never support the jsp.page , you need to add the external jar file (**Tomcat jsper**) dependence you can include, then it will support the jsp page in spring boot application

```
<!-- https://mvnrepository.com/artifact/org.apache.tomcat.embed/tomcat-embed-jasper -->
            <dependency>
                  <groupId>org.apache.tomcat.embed</groupId>
                  <artifactId>tomcat-embed-jasper</artifactId>
                  <version>9.0.35</version>

            </dependency>
```

Note :- if your are going to be mention Html page or jsp page, you have to put the specification path in the **application. Properties**

Exe :-

```
spring.mvc.view.prefix=/pages      // pages  here folder name
spring.mvc.view.suffix=/jsp        // jsp type of language
```

@RequestParam is a **Spring** annotation used to bind a web request parameter to a method parameter.

Spring Boot DB configuration :-

Write in the application.properties file , this is for H2 datanase

```
spring.h2.console.enabled=true
spring.datasource.platform=h2
spring.datasource.url=jdbc:h2:mem:kiran
```

H2 Databse :- configuration in online use this url :
http://localhost:8080/h2-console/

NOTE :-

@PathVariable annotation is used for data passed in the URI (e.g. RESTful web services)

@RequestParam is used to extract the data found in query parameters.

Spring Boot Annotations

@SpringBootApplication: -Spring Boot Annotations**.** Spring Boot Annotations **is a form of metadata that provides data about a programEncapsulates** *@Configuration*, *@EnableAutoConfiguration*, **and** *@ComponentScan* **annotations with their default attributes.**

`@ResponseStatus` :- this annotation will tells to perfect, exception will to user,

For example :-

If user send unregister number it will throws the 500 internal server error BUT, actually its 404 exception

Exe :
```
@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotfoundException extends
RuntimeException {

     public UserNotfoundException(String message) {
          super(message);
     }
```

```
package com.purinis.exceptions;

import java.util.Date;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import
org.springframework.web.bind.annotation.ControllerAdvice;
import
org.springframework.web.bind.annotation.RestController;
import
org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.Res
ponseEntityExceptionHandler;

import com.purinis.SpringBoot28Min.UserNotfoundException;

@RestController
@ControllerAdvice
public class CustomizeResponseEntityExceptionHandler
extends ResponseEntityExceptionHandler{

     public final ResponseEntity<Object>
handleExceptions(Exception ex, WebRequest request) {

          new ExceptionResource(new Date(),
ex.getMessage(), request.getDescription(false));

 return new
ResponseEntity(ex,HttpStatus.INTERNAL_SERVER_ERROR);
     }


     public final ResponseEntity<Object>
handleNotfoundExceptions(UserNotfoundException
userNotfoundException, WebRequest request) {

          new ExceptionResource(new Date(),
userNotfoundException.getMessage(),
request.getDescription(false));
```

```
 return new
ResponseEntity(userNotfoundException,HttpStatus.NOT_FOUND
);
      }

}
```

URI : User Resource Identifier

A resource can have different representations

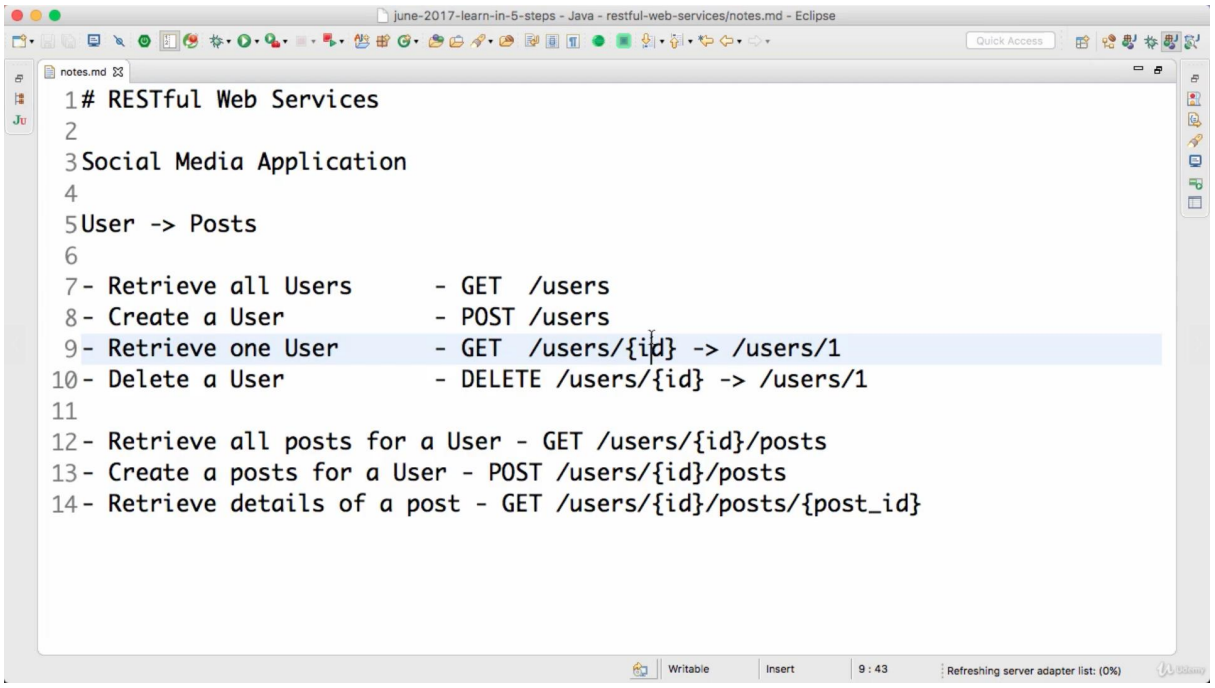
XML

JSON

HTML

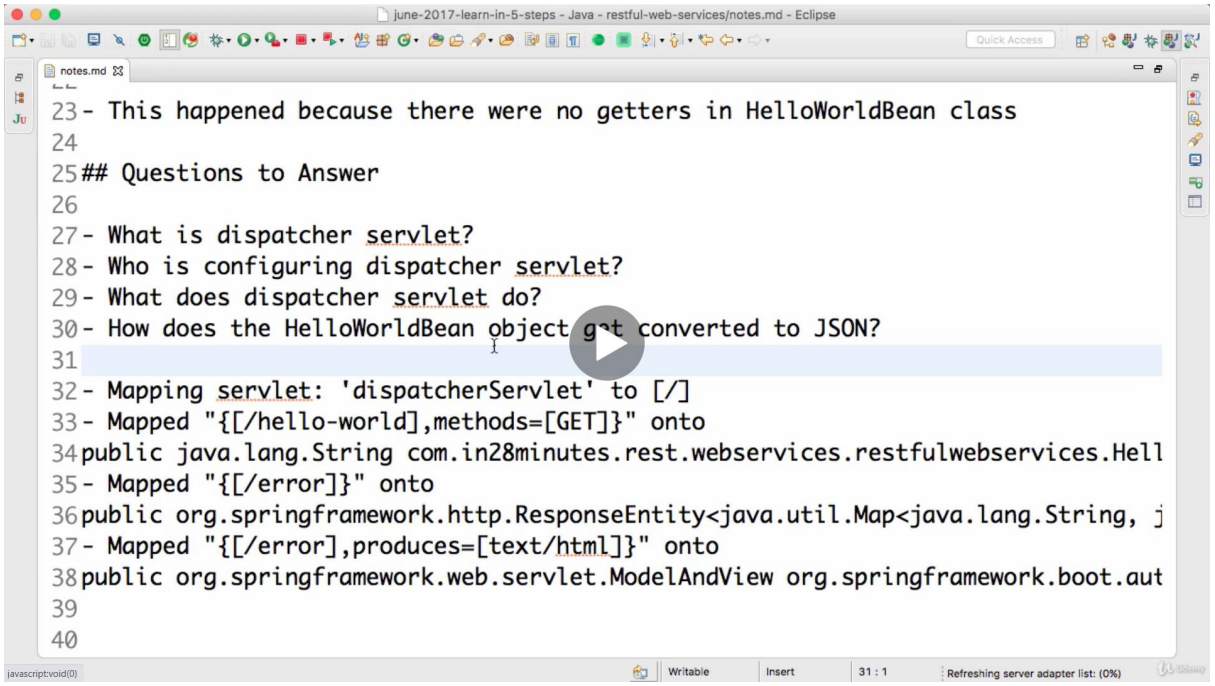
TEXT

etc.....



- **Rest method return the bean back**

```
package com.purinis.FirstApplicationMic;

public class HelloWorldBean {

	String message;

	public HelloWorldBean(String message) {

		this.message = message;
	}

	public String getMessage() {

	}

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

	@Override
	public String toString() {
		return "HelloWorldBean [message=" + message +
"]";
	}
}




// rest method return the bean back
	@RequestMapping(value = "/hello-world",method =
RequestMethod.GET)
	public HelloWorldBean helloworldBean() {

		return new HelloWorldBean( "Hello World");
	}
```



Note Setup the debug mode in spring boot :-

```
logging.level.org.springframework=debug
```

Dispatcher Servlet :-Dispatcher-Servlet contain auto_configuration_report, which is contain lot more in spring boot application,

Note :

If you are making use of Spring Boot Release (> 2.3.0) make sure to add the following dependency to your pom.xml:

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

If you are of the curious kind:

Here's the reason you need to add the dependency - https://github.com/spring-projects/spring-boot/issues/19550

Quick Tip : HATEOAS Recent Changes

(Hypermedia-Driven RESTful Web Service)

VERSION UPDATES FOR NEXT LECTURE

Hyper media as the engine application

HATEOAS provides some APIs to ease creating REST representations that follow the HATEOAS principle when working with Spring and especially Spring MVC. The core problem it tries to address is link creation and representation assembly.

There are a few modifications of HATEOAS in the latest release of Spring HATEOAS 1.0.0:

One of these should work

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

Option 1 : Spring Boot Release >= 2.2.0

```
import org.springframework.hateoas.EntityModel;

import org.springframework.hateoas.server.mvc.ControllerLinkBuilder;

import org.springframework.hateoas.server.mvc.WebMvcLinkBuilder;

---------

@GetMapping("/users/{id}")

public EntityModel<User> retrieveUser(@PathVariable int id) {

```

```
    User user = service.findOne(id);

    if (user== null)

            throw new UserNotFoundException("id-" + id);

    EntityModel<User> model = new EntityModel<>(user);

    WebMvcLinkBuilder linkTo = WebMvcLinkBuilder.linkTo(ControllerLinkBuilder.methodOn(this.getClass()).retrieveAllUsers());

    model.add(linkTo.withRel("all-users"));

    return model;

}
```

Option 2: Older versions

```
import static org.springframework.hateoas.mvc.ControllerLinkBuilder.linkTo;
import static org.springframework.hateoas.mvc.ControllerLinkBuilder.methodOn;
import org.springframework.hateoas.Resource;
import org.springframework.hateoas.mvc.ControllerLinkBuilder;

Resource<User> resource = new Resource<User>(user);
ControllerLinkBuilder linkTo = linkTo(methodOn(this.getClass()).retrieveAllUsers());
resource.add(linkTo.withRel("all-users"));
return resource;
```

Swagger2 configuration

Swagger2 is an open source project used to generate the REST API documents for RESTful web services. It provides a user interface to access our RESTful web services via the web browser.

Swagger url : http://localhost:8080/swagger-ui.html

Document share : http://localhost:8080/v2/api-docs

Course Update : Incompatibility in recent versions of Swagger and Hateoas

You can add these dependencies to your pom.xml for Swagger:

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

In the next lecture, if you face problems ensure that you configure a LinkDiscoverer:

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Bean
    public LinkDiscoverers discoverers() {
        List<LinkDiscoverer> plugins = new ArrayList<>();
        plugins.add(new CollectionJsonLinkDiscoverer());
        return new LinkDiscoverers(SimplePluginRegistry.create(plugins));
    }

    @Bean
```

```
13.     public Docket api(){
14.         return new Docket(DocumentationType.SWAGGER_2)
15.                 .apiInfo(DEFAULT_API_INFO)
16.          ...
17.     }
18.
19. }
```

```
package com.purinis.SpringBoot28Min;

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

import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import
springfox.documentation.swagger2.annotations.EnableSwagge
r2;
```

Swagger congiuration class

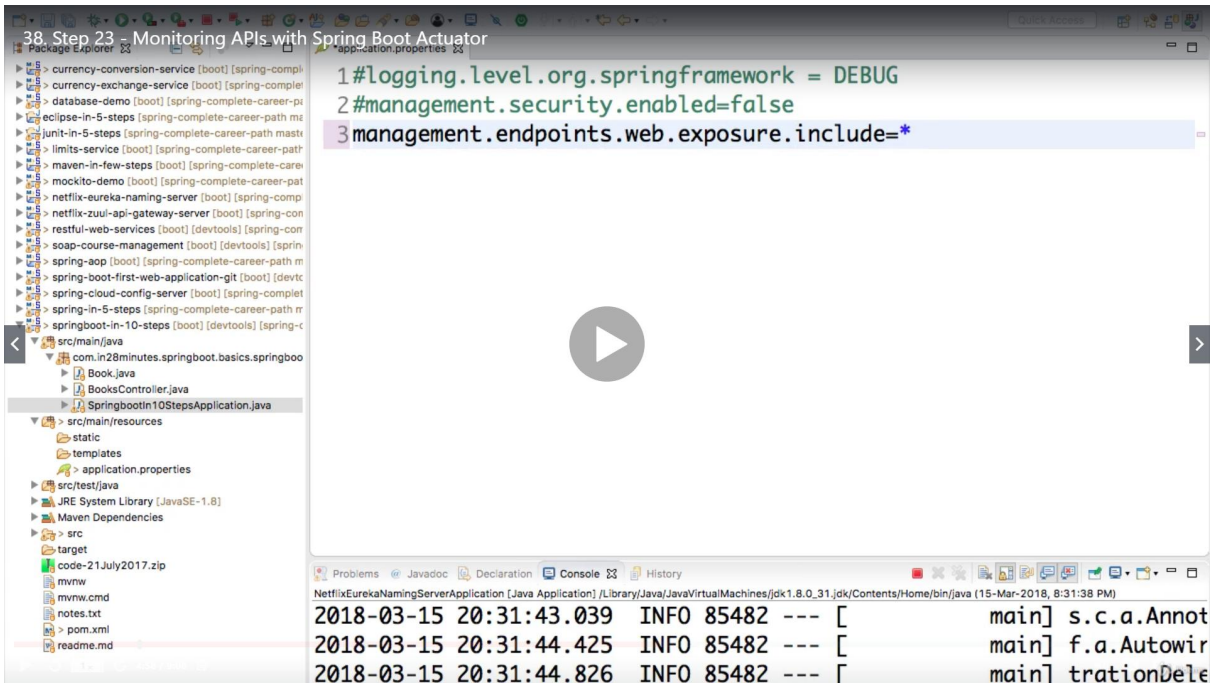
```
@Configuration
@EnableSwagger2
public class SwaggerConfiguration {

     public static final Contact DEFAULT_CONTACT = new
Contact("kiran kumar", "www.google.com",
"kpurini04@gmail.ocm");
     public static final ApiInfo DEFAULT_API_INFO = new
ApiInfo("Api Documentation", "Api Documentation", "1.0",
               "urn:tos", DEFAULT_CONTACT, "Apache 2.0",
"http://www.apache.org/licenses/LICENSE-2.0");

     @Bean
     public Docket api() {

          return new
Docket(DocumentationType.SWAGGER_2).apiInfo(DEFAULT_API_I
NFO);
     }
```

}

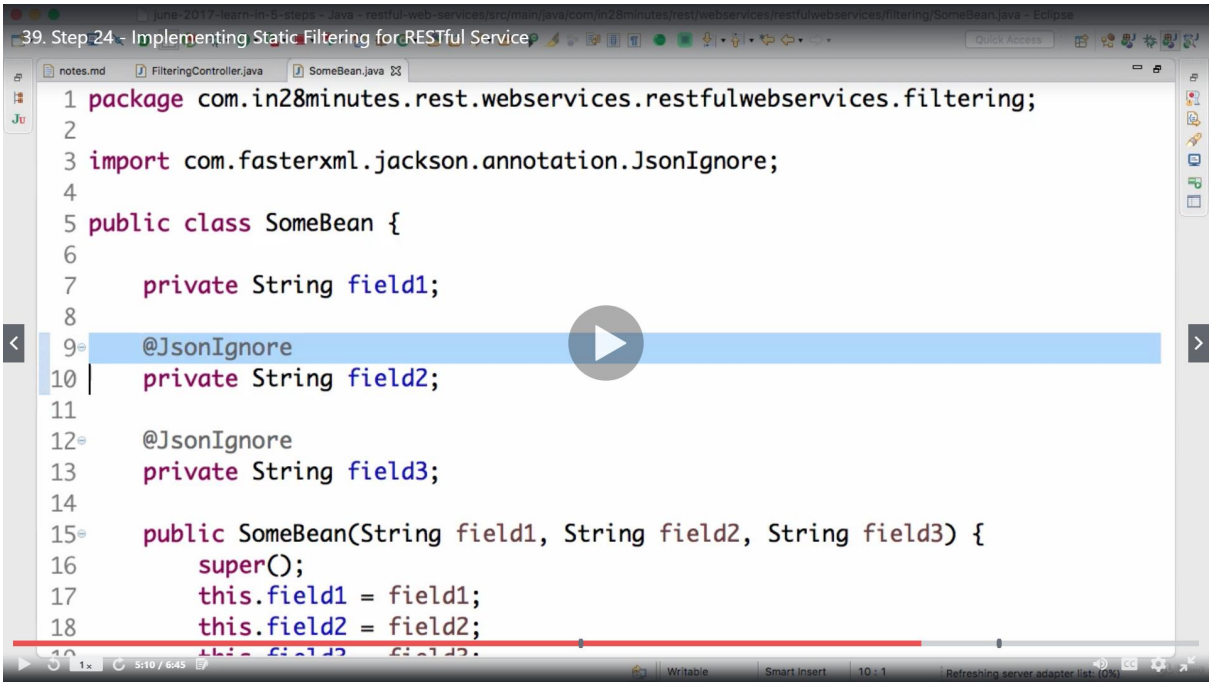
Actuator in spring boot

Spring Boot Actuator. **Spring Boot Actuator** is a sub-project of the **Spring Boot** Framework. It includes a number of additional features that help us to monitor and manage the **Spring Boot** application. It contains the **actuator** endpoints (the place where the resources live).

Properties configuration in actuator :-



@JsonIgnore : - this is annotation does ignore the property in the response body object



Other way : - we can directly using @JsonIngnoreProperties(value={"property1","property2"})

Spring boot Base Security :-

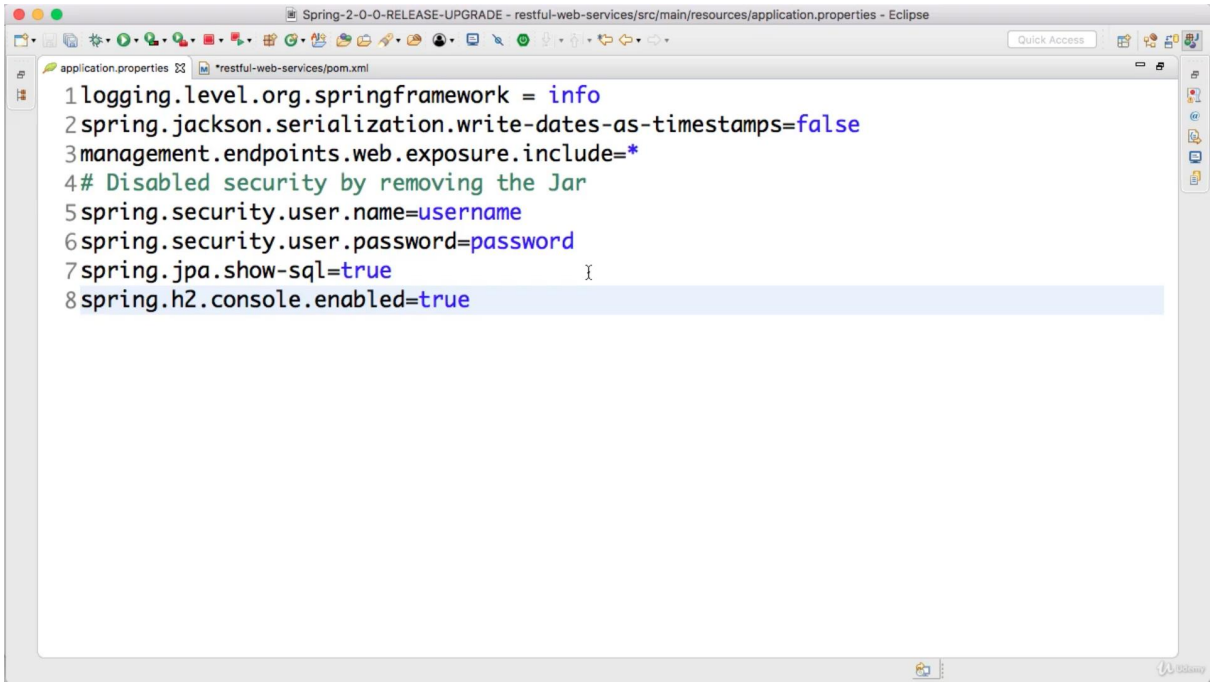
In this tutorial we just add the security dependency in pom.xml file this dependency will do create the default password each and every time run the application

NOTE :- if u don't want to use the default password , you can write your won user name and password like this,

Application.properties

Security.user.name =username

Securiry.user.password= p assword



(id integer not null, birth_date timestamp, name varchar(20), primary key (id)

Note :-

"default constructor" refers to a nullary constructor that is automatically generated by the compiler if no constructors have been defined for the class. The default constructor implicitly calls the superclass's nullary constructor, then executes an empty body. All fields are left at their initial value of 0 (integer types), 0.0 (floating-point types), `false` (`boolean` type), or `null` (reference types). A programmer-defined constructor that takes no parameters is also called a default constructor in C#, but not in Java.

Hibernate Mapping two Pojos

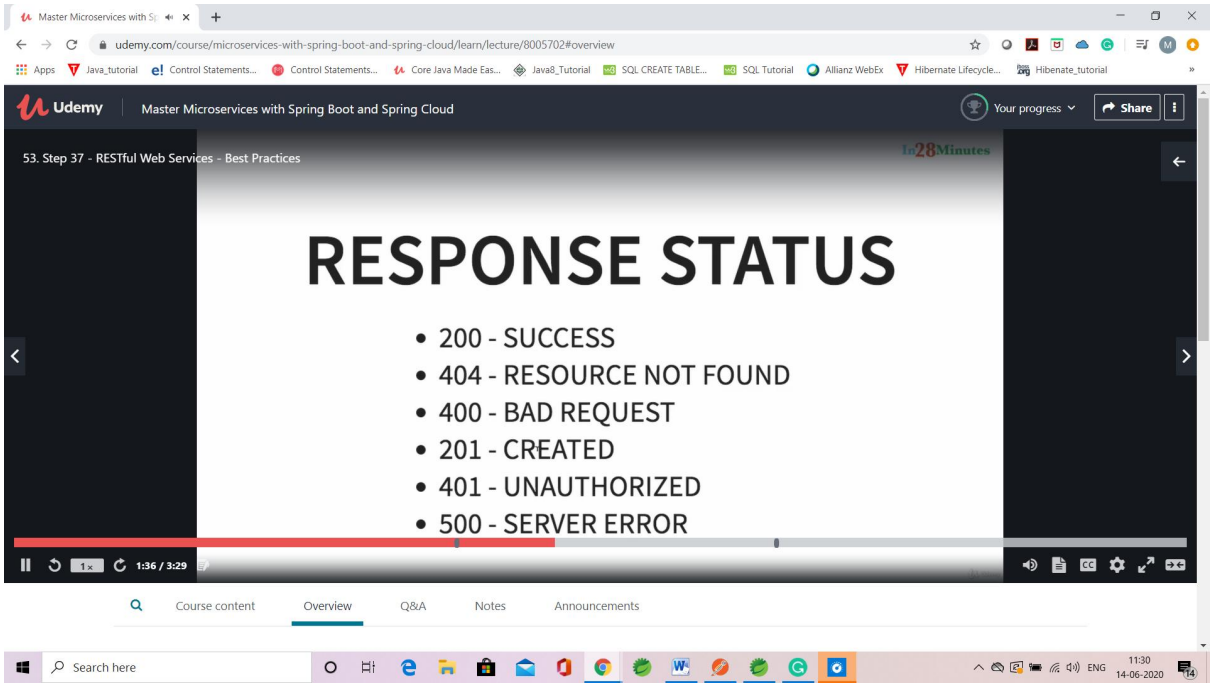
- **This is the many post**
-
- `@Entity`
- `public class Post {`
-
- `@Id`
- `@GeneratedValue`
-
- `private Integer id;`
- `private String discription;`

```
@ManyToOne(fetch = FetchType.LAZY)
private User user;
```

Note: - here Post class don't select toString method for User, class

User class

```
@OneToMany(mappedBy = "user")
private List<Post> posts;

public List<Post> getPosts() {
    return posts;
}

public void setPosts(List<Post> posts) {
    this.posts = posts;
}
```



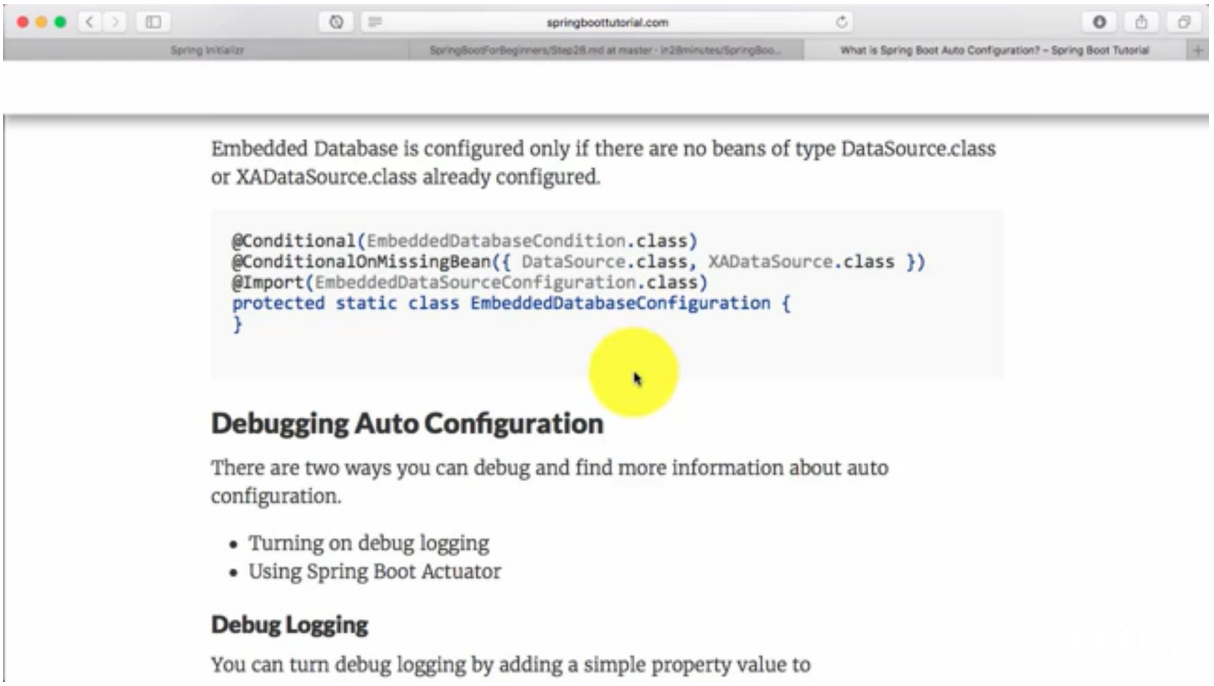
Question :- what is the @SpringBootApplication ?

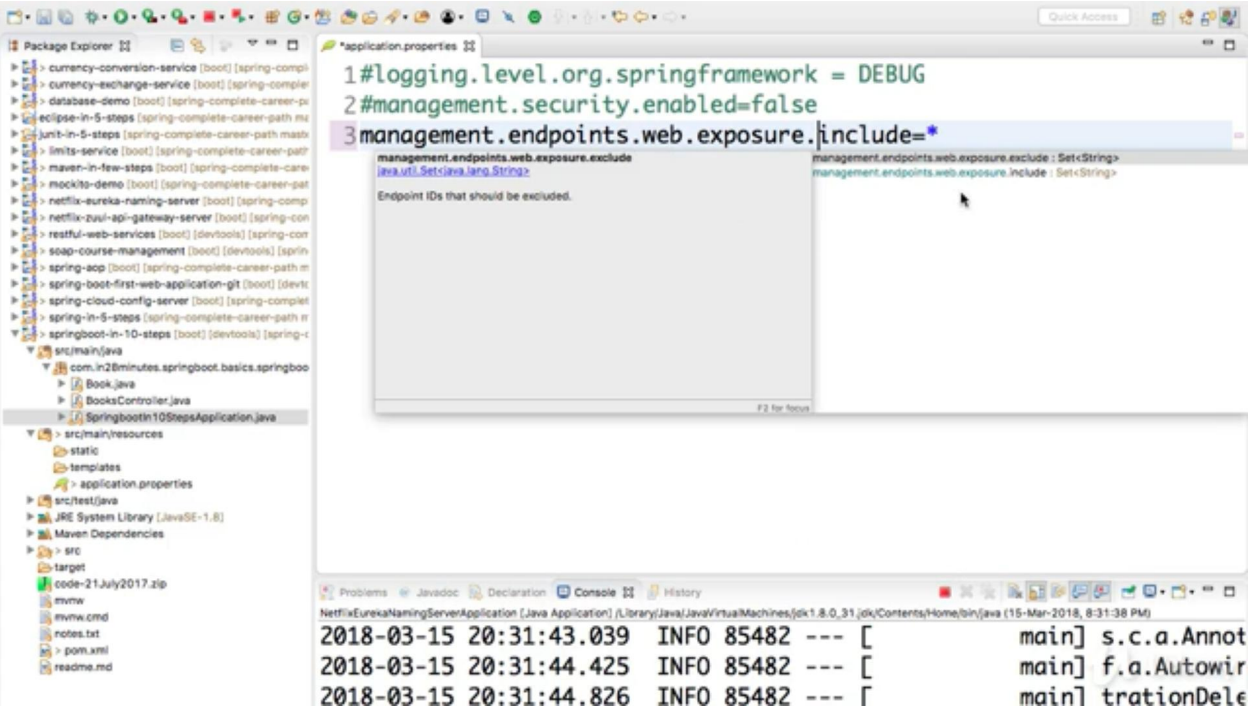
- Its indicate this is the spring boot application

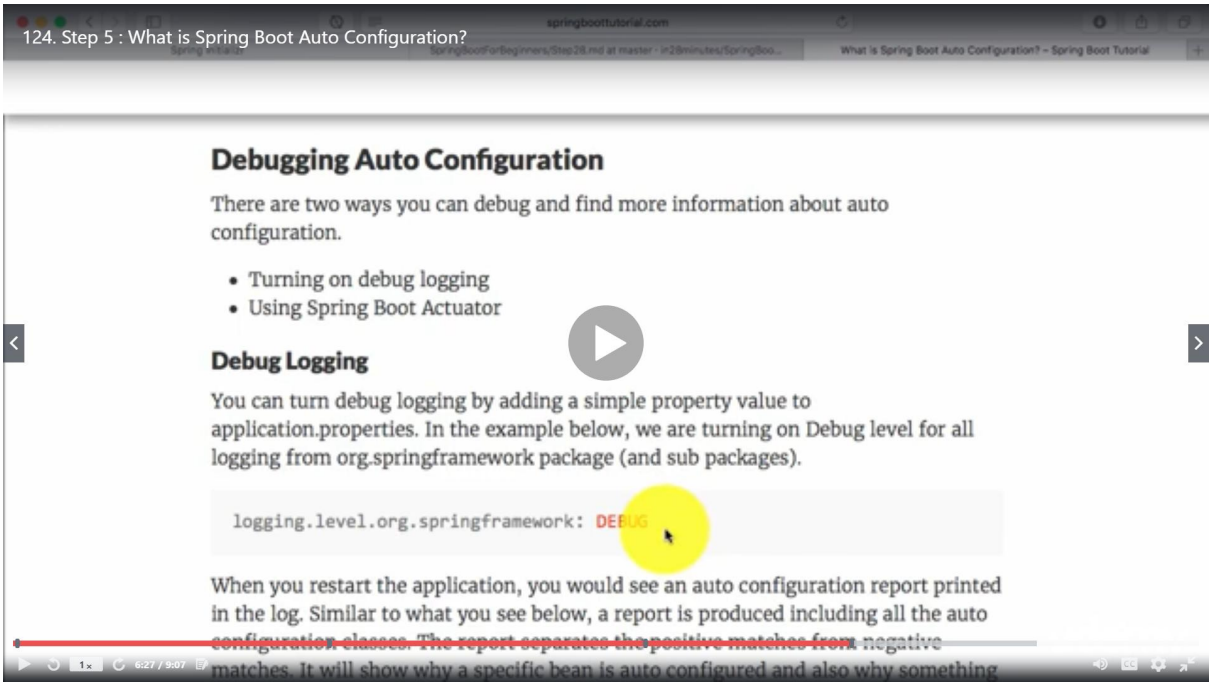
- Its enable something called AutoConfiguration

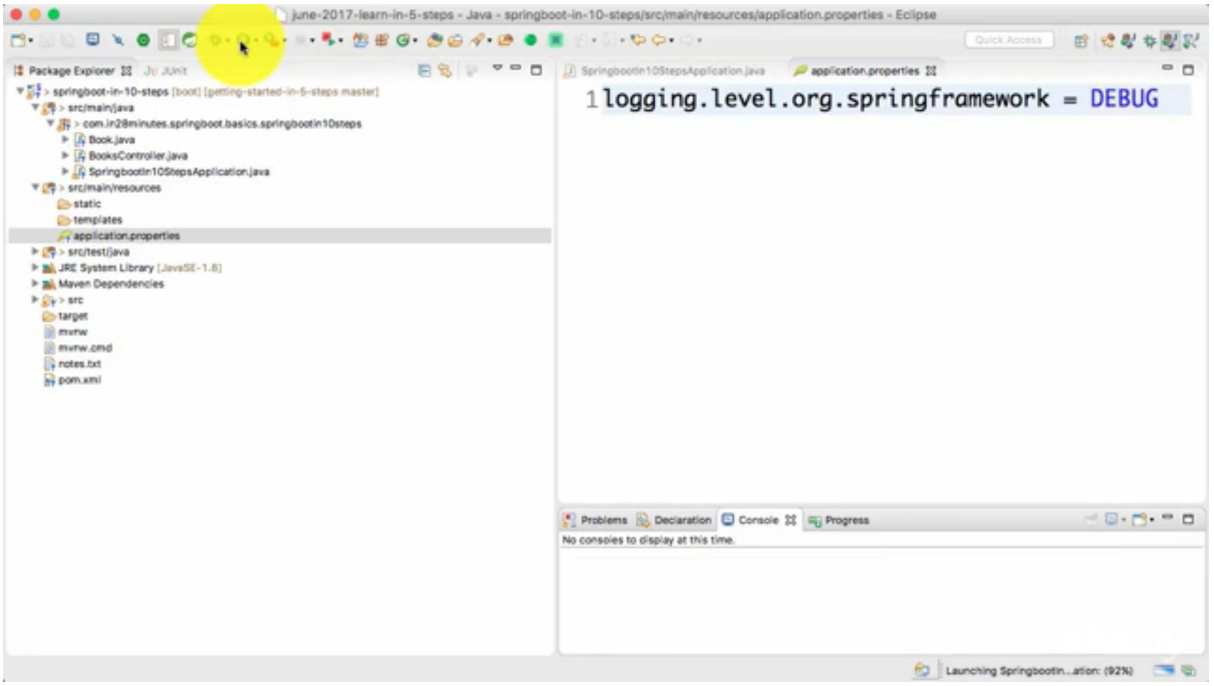
- Its enable something called ComponentScan

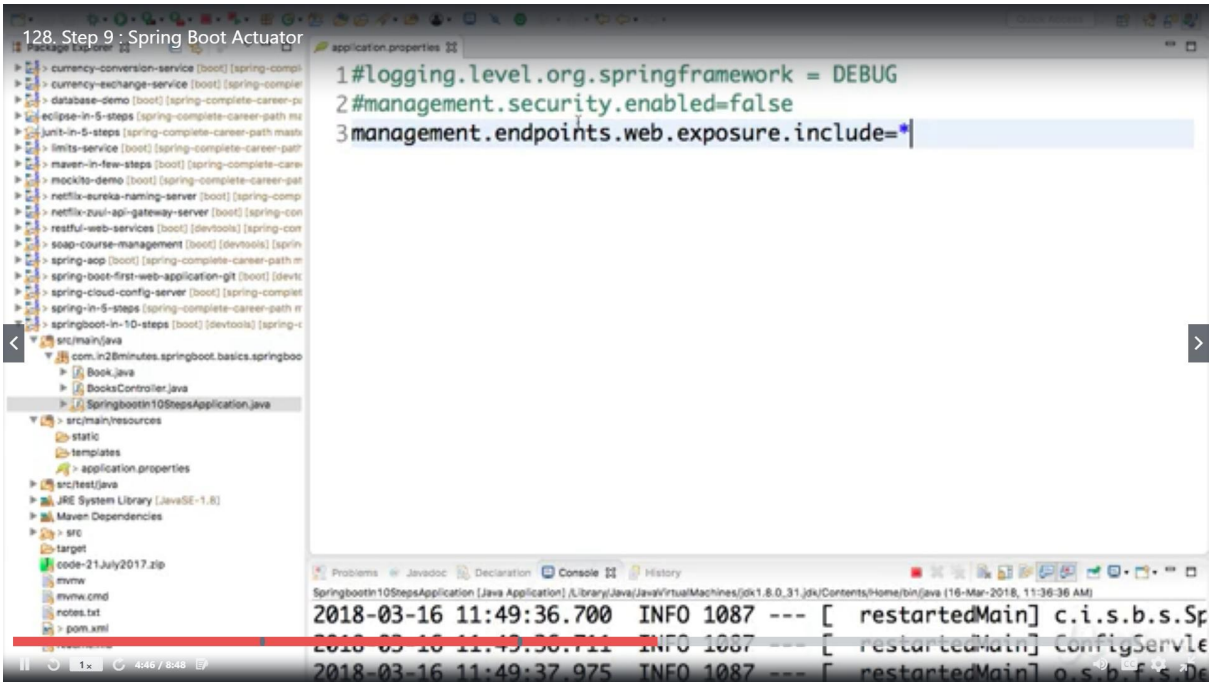
NOTE:- component Scan is one of the important feature in spring boot application it will scan automatically spring boot application




#logging.level.org.springframework = DEBUG
#management.security.enabled=false
management.endpoints.web.exposure.include=*
management.endpoints.web.exposure.exclude
Endpoint IDs that should be excluded.
2018-03-15 20:31:43.039 INFO 85482 --- [main] s.c.a.Annot
2018-03-15 20:31:44.425 INFO 85482 --- [main] f.a.Autowir
2018-03-15 20:31:44.826 INFO 85482 --- [main] trationDele


124. Step 5 : What is Spring Boot Auto Configuration?
Debugging Auto Configuration
There are two ways you can debug and find more information about auto configuration.
Turning on debug logging
Using Spring Boot Actuator
Debug Logging
You can turn debug logging by adding a simple property value to application.properties. In the example below, we are turning on Debug level for all logging from org.springframework package (and sub packages).
logging.level.org.springframework: DEBUG
When you restart the application, you would see an auto configuration report printed in the log. Similar to what you see below, a report is produced including all the auto configuration classes. The report separates the positive matches from negative matches. It will show why a specific bean is auto configured and also why something


june-2017-learn-in-5-steps - Java - springboot-in-10-steps/src/main/resources/application.properties - Eclipse
Package Explorer
springboot-in-10-steps [boot] [getting-started-in-5-steps master]
src/main/java
com.in28minutes.springboot.basics.springbootin10steps
Book.java
BooksController.java
SpringbootIn10StepsApplication.java
src/main/resources
static
templates
application.properties
src/test/java
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
mvnw
mvnw.cmd
notes.txt
pom.xml
1 logging.level.org.springframework = DEBUG
Problems
Declaration
Console
Progress
No consoles to display at this time.


128. Step 9 : Spring Boot Actuator
application.properties
1 #logging.level.org.springframework = DEBUG
2 #management.security.enabled=false
3 management.endpoints.web.exposure.include=*
2018-03-16 11:49:36.700 INFO 1087 --- [restartedMain] c.i.s.b.s.Sp