Section 1:
Course preview , where are the resources , Github , how to utilize the course etc.
Section 2:
Theory of what a rest service is.
Section 3:
Lecture 10,11,12: How to setup spring boot project
Lecture 13 ? How REST is mapped , how URI is mapped

Lecture 14,15: Creating Hello world, add bean to it.
Lecture 16: Theory of springs
What is dispatcher servlet? ? Servlet that handles http requests ,it uses FRONT controller design pattern.It knows all the mapping present .

Who is configuring dispatcher servlet ? ? SpringBootAutoConfiguration

What does dispatcher servlet do ? ? handles all the requests as per the mapping

How does the HelloWorldBean gets converted to Object ? - Jackson2Mapper bean converter.

Who is configuring the Error mapping ? ? SpringBootAutoConfiguration

Lecture 17: Path Variable example
Lecture 18: Create User Bean/Service ,@component
Lecture 19: Implementing GET method for User Resource ,@RestController
Lecture 20: Implementing POST method for User Resource

@RequestBody ? maps the values from an http request body to the object

Invoke POST request? POSTMAN(or any REST Client)

Lecture 21: *Enhancing POST method for to return correct status /return type
*http best Practices ? return correct status for correct operation/infformation.

```
URI location=ServletUriComponentBuilder.fromCurrentRequest()
        .path(/{id})
        .buildAndExpand(savedUser.getId())
        .toUri();
        return ResponseEntity.created(location).build();
```

Lecture 22: Exception handling.

*To send a more fitting response use: @ResponseStatus(HttpStatus.NOT_FOUND)

* Best practices is :to return 404 if resource not found rather than 500.

i.e to return suitable status rather than any error or non 200 status.

Lecture 23: Exception Handling: Generic style.

? define the response structure(generally a class containing exception timestamp,message,category,severity,etc)

? ResponseEntityExceptionHandler:An abstract class that provides basic exception handling methods. Extend this class, override methods as shown below.

@ControllerAdvice ? This class is now treated as exception handler for all the controllers defined in this app.
@RestControllers ? Tells Springs that this class is a Controller
extends ResponseEntityExceptionHandler? to inherit the default methods & override as needed.
@ExceptionHandler(XXX.class) ? fire this annotated method when exception of the class xxx.class is thrown.
ExceptionResponse ? Org based exception structure.Throw this when exception occurs.

@ExceptionHandler(UsernotFoundException.class) ? Custom exception class that is thrown when a user is not found from the UserService.

Lecture 24: Implement Exception Handling for POST method.
Lecture 25: Implement DELETE method.

Lecture 26: Validation Framework .

The response is 400 ? but it doesnt tells what went wrong. Thus we need to provide a customized message.i.e override the handleMethodArgumentNotValid() from ResponseEntityExceptionHandler.class in our CustomizedResponseEntityExceptionHandler.java

Exceute the request & see the message.

There is too much of details , so we can customize the message by :
? Providing message at user entity.
? remove ex.getBindingResult().toString() with Validation Failed

Execute the REST request again, see the message outputs & the @nnotated message.:

Various type of validation present are listed under: javax.constraints package of validation-api-1.1.0.Final.jar & is implemented in hibernate-validator-5.4.jar. They are defined as spring web started web dependency, thus we get these jars when we import the project.

Lecture 27: HATEOAS
Hypermedia As The Engine Of Application State (HATEOAS) is a component of the REST application architecture that distinguishes it from other network application architectures. It mandates a service should provide information on how to access the resource.
Ref: https://spring.io/understanding/HATEOAS

Response = Response + Link to other useful resource.

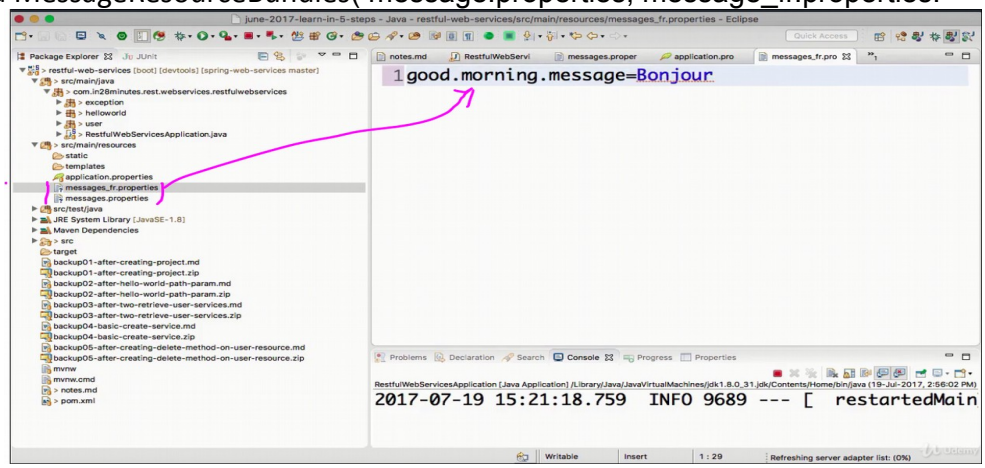
Lecture 28: Advanced REST using Spring (Internationalization, Content Negotiation,Documentation,Springs Actuator,
Lecture 29,30: INTERNATIONALIZATION (i18n)

how to achieve this :

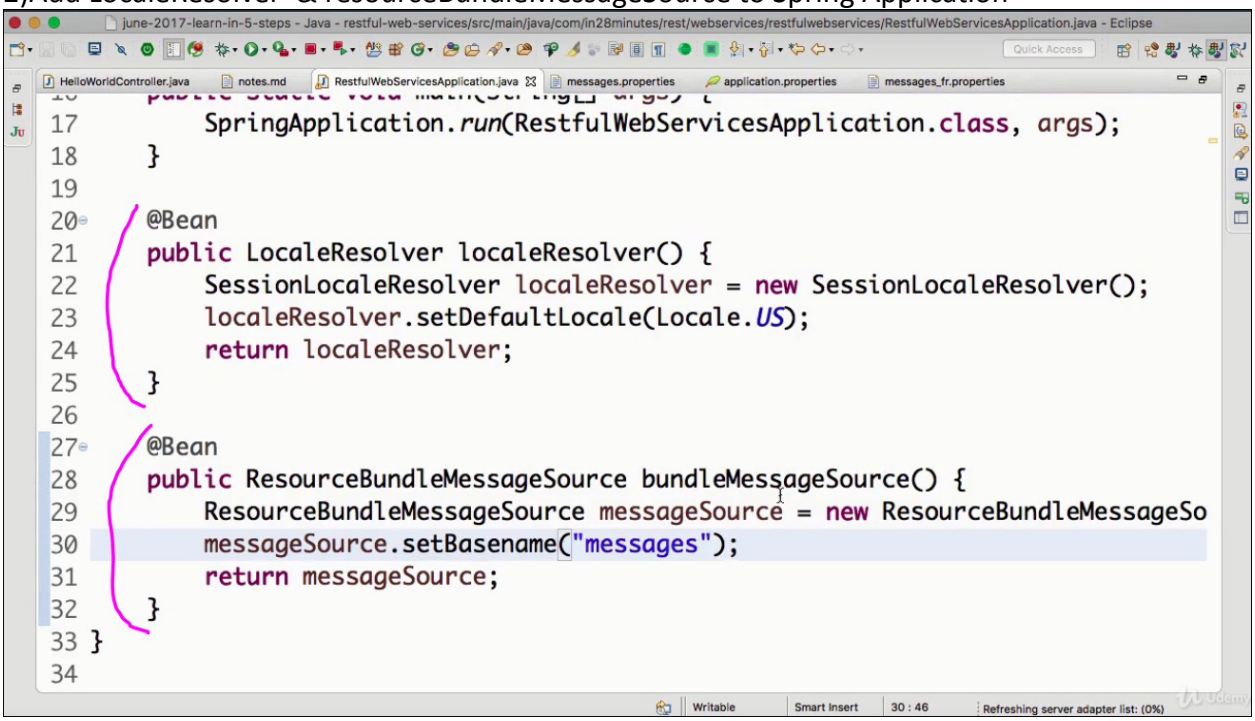
Configure LocaleResolver
set default Resolver
Set ResourceBundleMessageSource
Use Autowiring to get the MessageSource

1) Create message resources

Add MessageResourceBundles(message.properties, message_fr.properties.



2)Add LocaleResolver & resourceBundleMessageSource to Spring Application



. Note

messageSource.setBaseName("message") ==> our messages are stored in message* files.
I.e message.properties, message_fr.properties,message_in.properties etc .

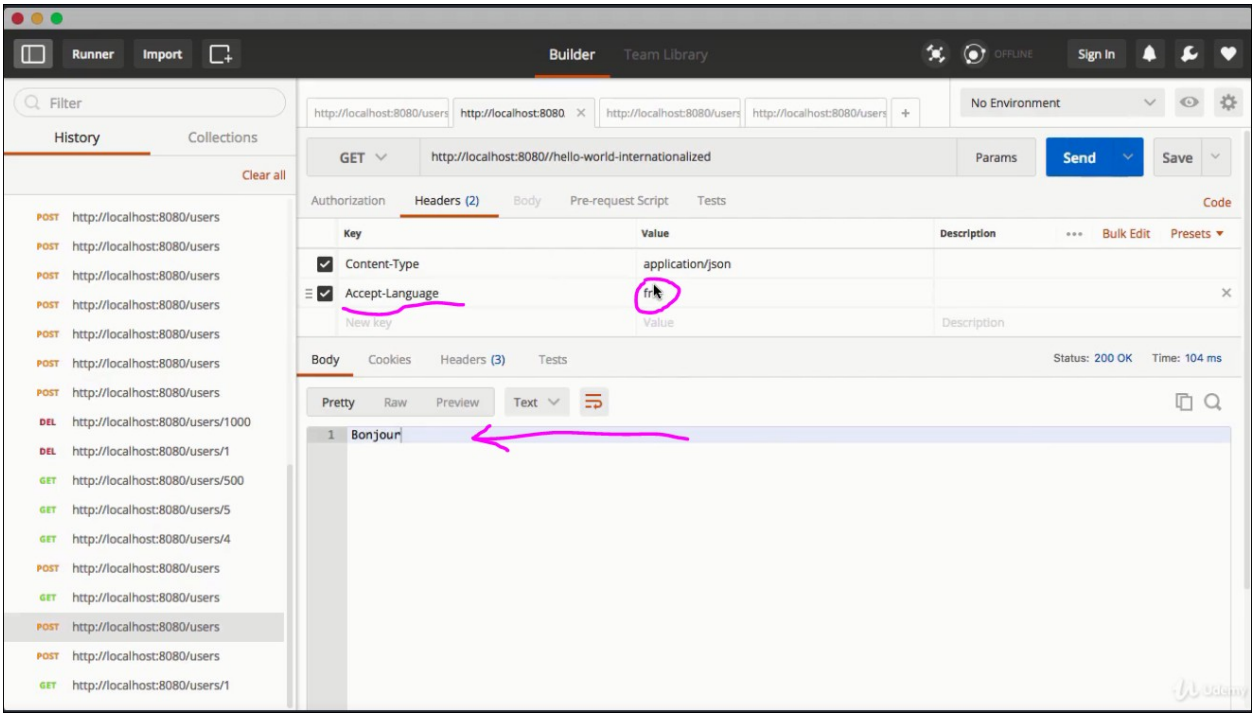
3) Update Controller to use the resource bundle

```
2
3 import org.springframework.beans.factory.annotation.Autowired;
8
9 //Controller
10 @RestController
11 public class HelloWorldController {
12
13     @Autowired
14     private MessageSource messageSource;
15
16     @GetMapping(path = "/hello-world")
17     public String helloWorld() {
18         return "Hello World";
19     }
20
21     @GetMapping(path = "/hello-world-bean")
22     public HelloWorldBean helloWorldBean() {
23         return new HelloWorldBean("Hello World");
```

```java
    @GetMapping(path = "/hello-world-bean")
    public HelloWorldBean helloWorldBean() {
        return new HelloWorldBean("Hello World");
    }

    ///hello-world/path-variable/in28minutes
    @GetMapping(path = "/hello-world/path-variable/{name}")
    public HelloWorldBean helloWorldPathVariable(@PathVariable String name) {
        return new HelloWorldBean(String.format("Hello World, %s", name));
    }

    @GetMapping(path = "/hello-world-internationalized")
    public String helloWorldInternationalized(
            @RequestHeader(name="Accept-Language", required=false) Locale local
        return messageSource.getMessage("good.morning.message", null, locale);
    }

}
```

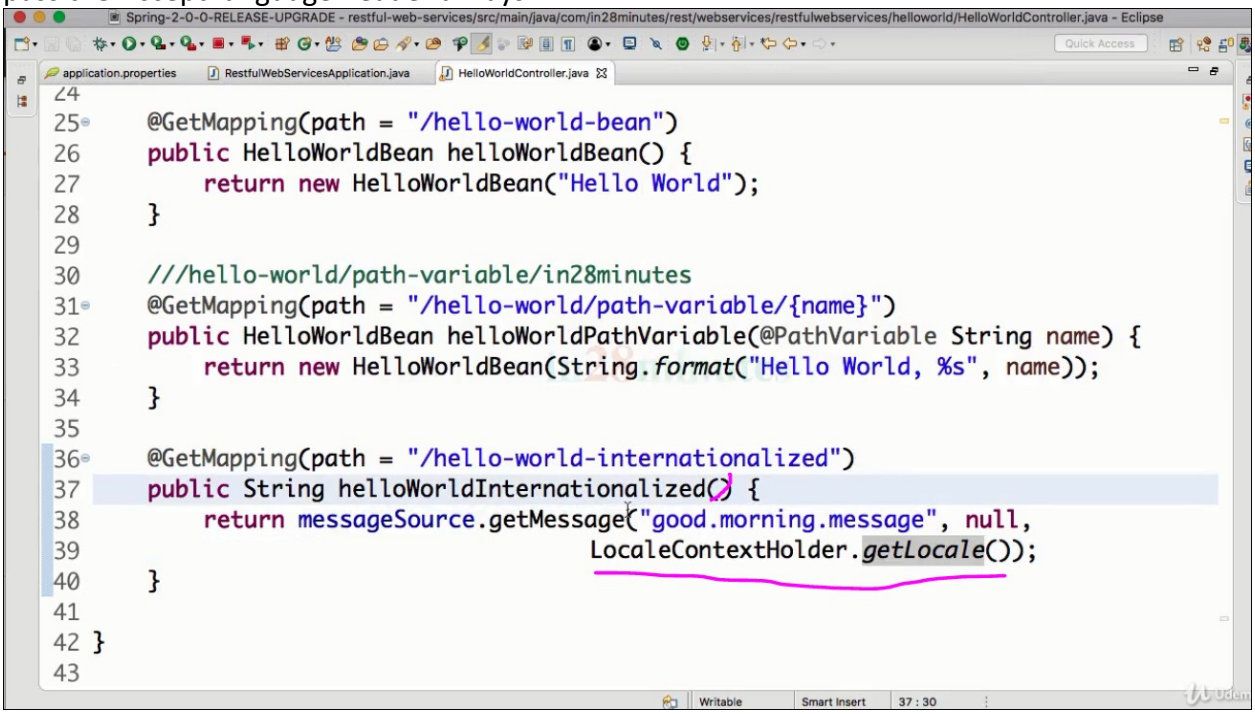
@ResourceHeader(name=Accept-Language,required=false) Locale locale
If the header attribute Accept-Language is present take the locale preference from here ,also required=false=> if not present take default locale.

Test from postman:



FIX-1 (needs FIX-2 as wel)

Now we have added to the method , its a pain to add it to every method. So instead of picking it from the Accept-Language header lets pick it from LocaleContextHolder. That way we need not pass the Accept-language header always



FIX-2

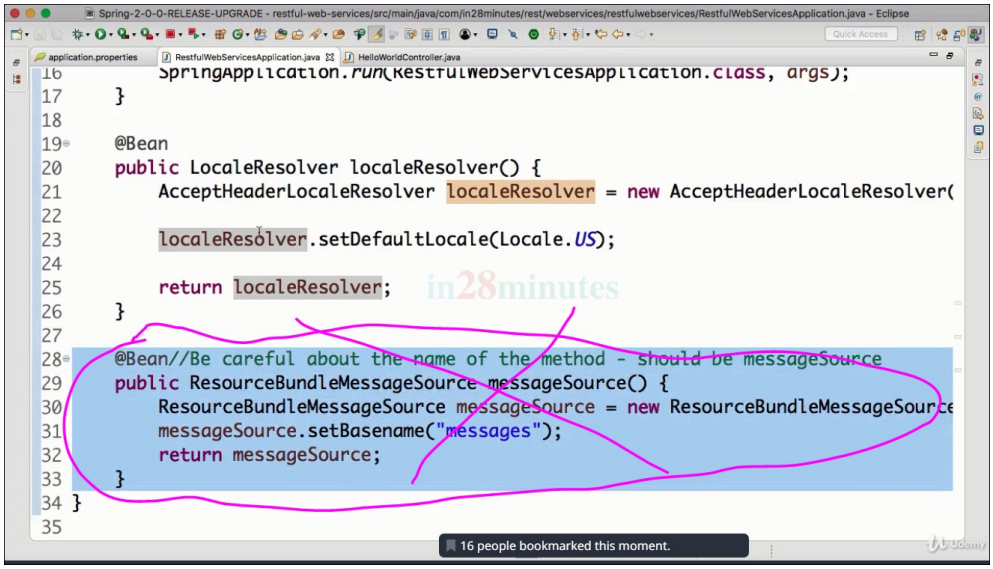
Instead of SessionLocaleResolver we can use AcceptLocaleResolver , this enables SPRINGS to lead locale preference from Accept-Language header. If it is not here , it picks the default.

```
Spring-2-0-0-RELEASE-UPGRADE - restful-web-services/src/main/java/com/in28minutes/rest/webservices/restfulwebservices/RestfulWebServicesApplication.java - Eclipse
application.properties | RestfulWebServicesApplication.java | HelloWorldController.java
16      SpringApplication.run(RestfulWebServicesApplication.class, args);
17  }
18
19  @Bean
20  public LocaleResolver localeResolver() {
21      AcceptHeaderLocaleResolver localeResolver = new AcceptHeaderLocaleResolver();
22
23      localeResolver.setDefaultLocale(Locale.US);
24
25      return localeResolver;
26  }
27
28  @Bean//Be careful about the name of the method - should be messageSource
29  public ResourceBundleMessageSource messageSource() {
30      ResourceBundleMessageSource messageSource = new ResourceBundleMessageSource();
31      messageSource.setBasename("messages");
32      return messageSource;
33  }
34
35
16 people bookmarked this moment.
```

FIX-3: The messageSource() can be removed from RestfulWebServiceApplication.java & done in the message.properties directly as follows:

```
Spring-2-0-0-RELEASE-UPGRADE - restful-web-services/src/main/resources/application.properties - Eclipse
*application.properties | RestfulWebServicesApplication.java | HelloWorldController.java
1 logging.level.org.springframework = info
2 spring.jackson.serialization.write-dates-as-timestamps=false
3 spring.messages.basename=messages
```

restart the



application & test in POSTMAN client.

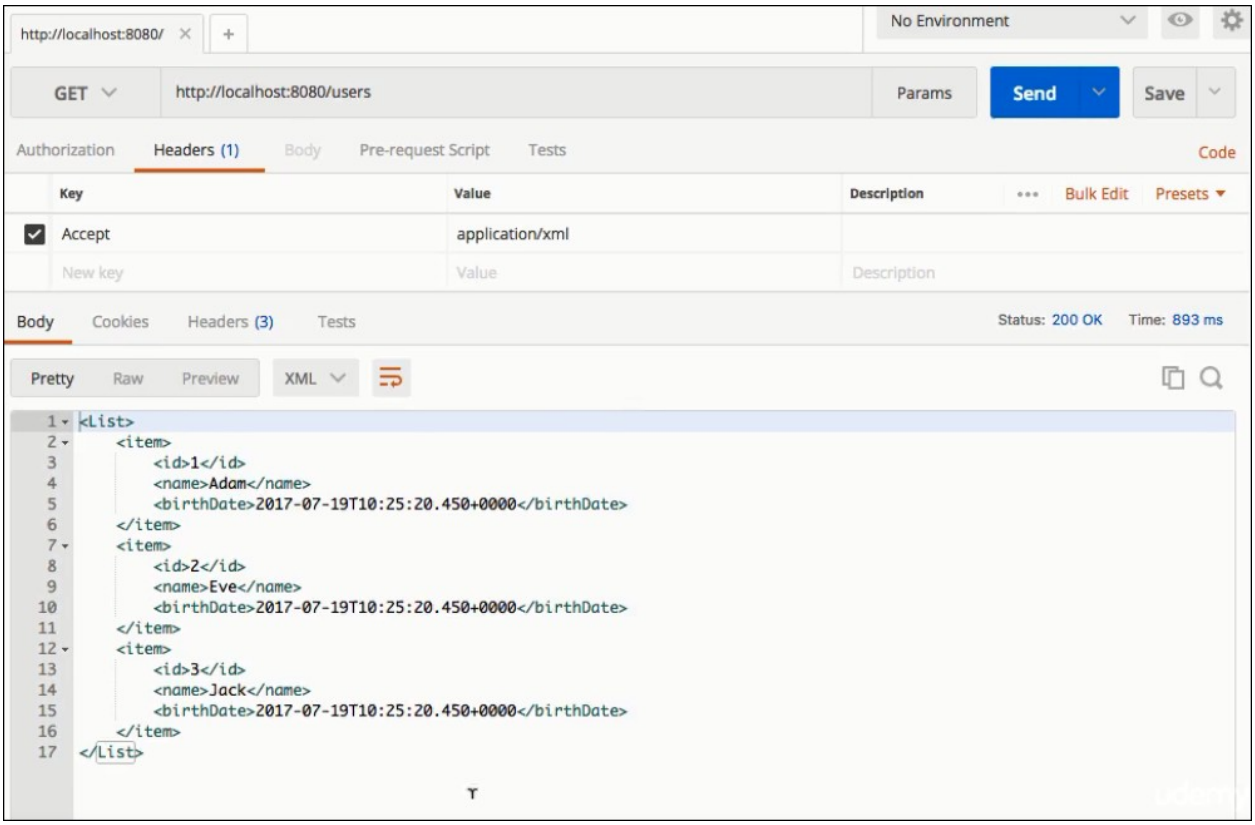
Lecture 31: Content Negotiation

For GET Request

1. **Add the below dependency to the pom**

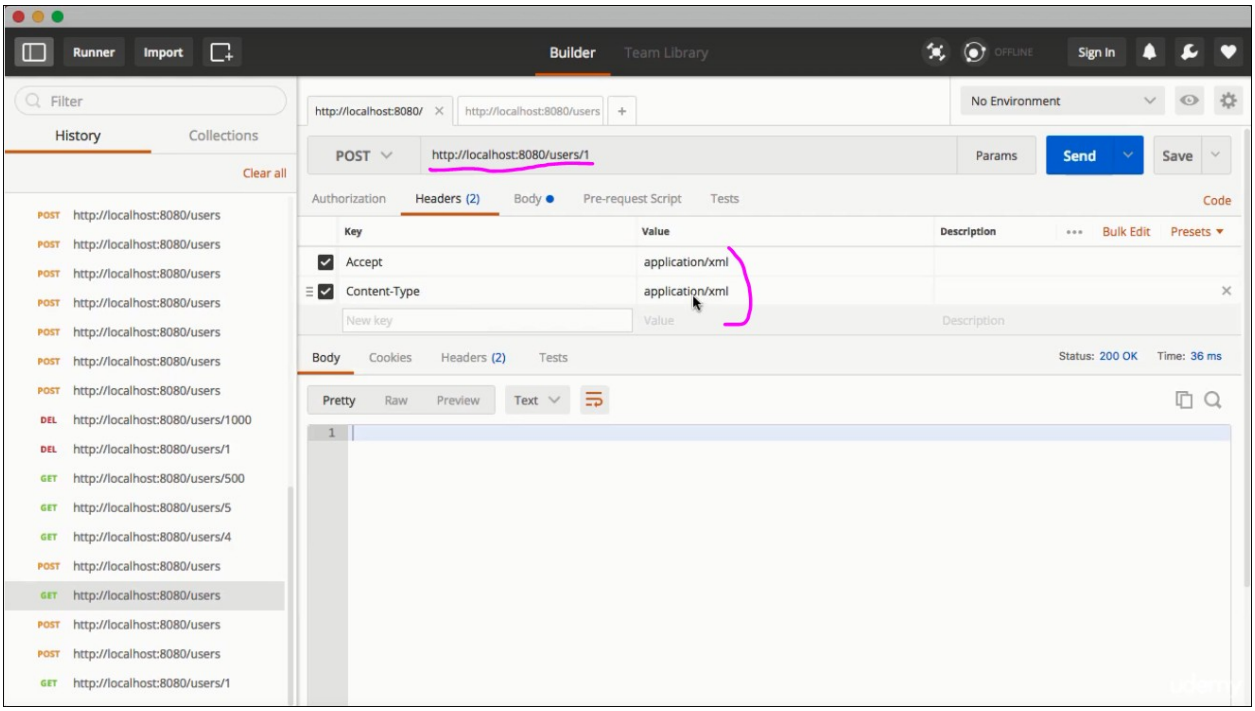
```
<dependency>
   <groupId>com.fasterxml.jackson.dataformat</groupId>
   <artifactId>jackson-dataformat-xml</artifactId>
   <version>2.8.5</version>
</dependency>
```

2. **send in header Content-type: application/xml in the POSTMAN client**

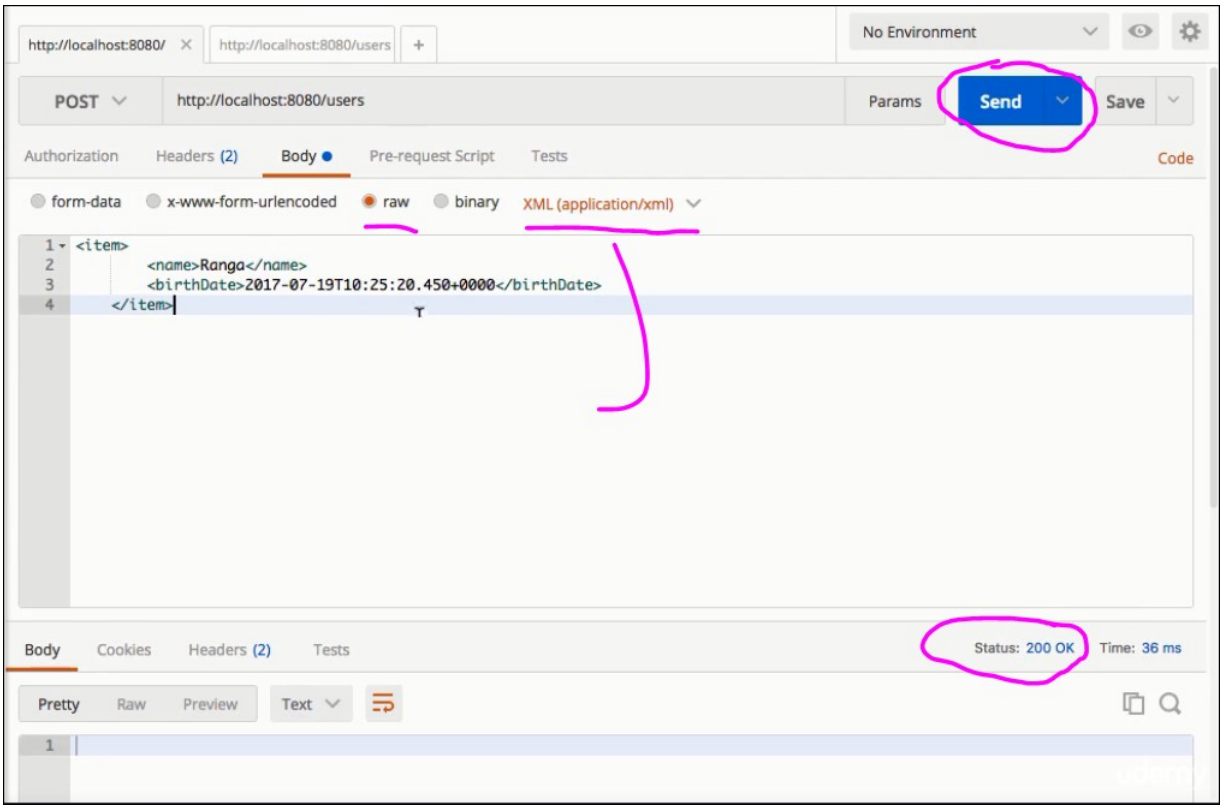


For POST request:

1. In Header put the below :

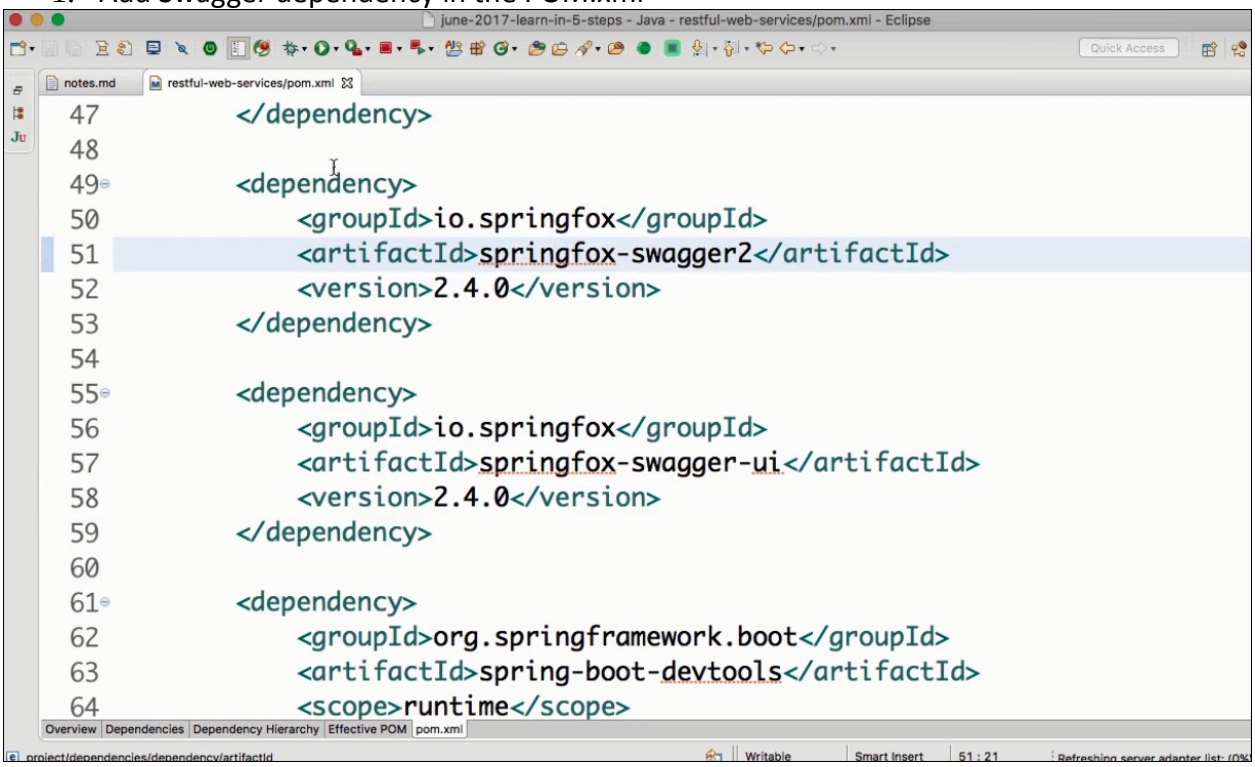


2. In Body put the below & test:



Lecture 32,33,34: Swaggar Documentation

1. Add Swagger dependency in the POM.xml



2. Add the Swagger Configuration

```
package com.in28minutes.rest.webservices.restfulwet

//Configuration
//Enable Swagger
public class SwaggerConfig {
    //Bean - Docket
      // Swagger 2
      // All the paths
      // All the apis
}
```

```
    // Bean - Docket
    @Bean
    public Docket api() {
        HashSet<String> consumesAndProduces = new HashSet<String>(Arrays.asList("application/json", "application/xml"));
        // Swagger 2- doc type
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(metadata())
                .consumes(consumesAndProduces)
                .produces(consumesAndProduces)
                .pathMapping("/");
    }

    private ApiInfo metadata() {
        return new ApiInfoBuilder()
                .title("Spring Boot Proj API")
                .description("Sprng Boot API Description")
                .version("1.0")
                .contact(new Contact("Ranga", "http://www.in28minutes.com",
                        "in28minutes@gmail.com"))
                .license("Apache 2.0")
                .licenseUrl("http://www.apache.org/licenses/LICENSE-2.0")
                .build();
    }


    // All the paths
    // All the apis
}
```

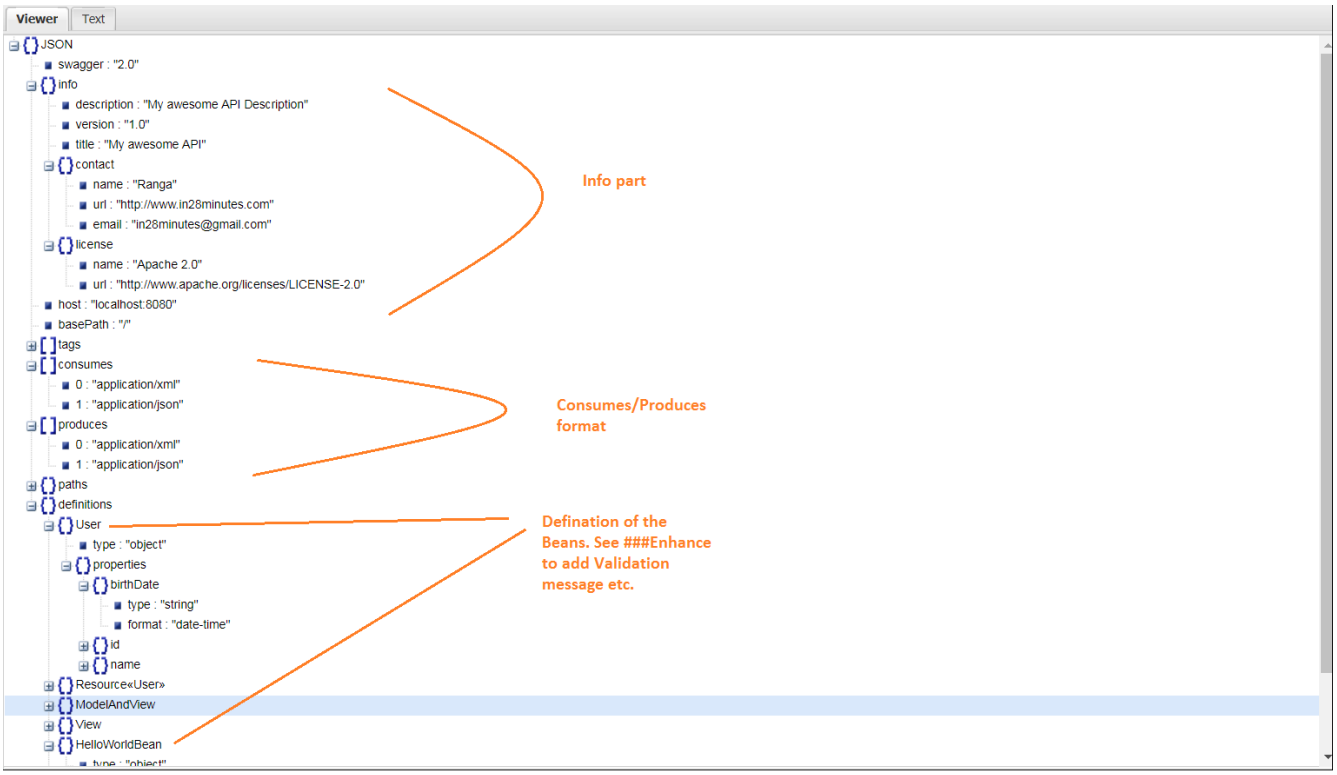
Info

3. Check the output at the below URL:

http://localhost:8080/v2/api-docs → JSON view (analyze using a JSON viewer)

http://localhost:8080/swagger-ui.html → web UI View

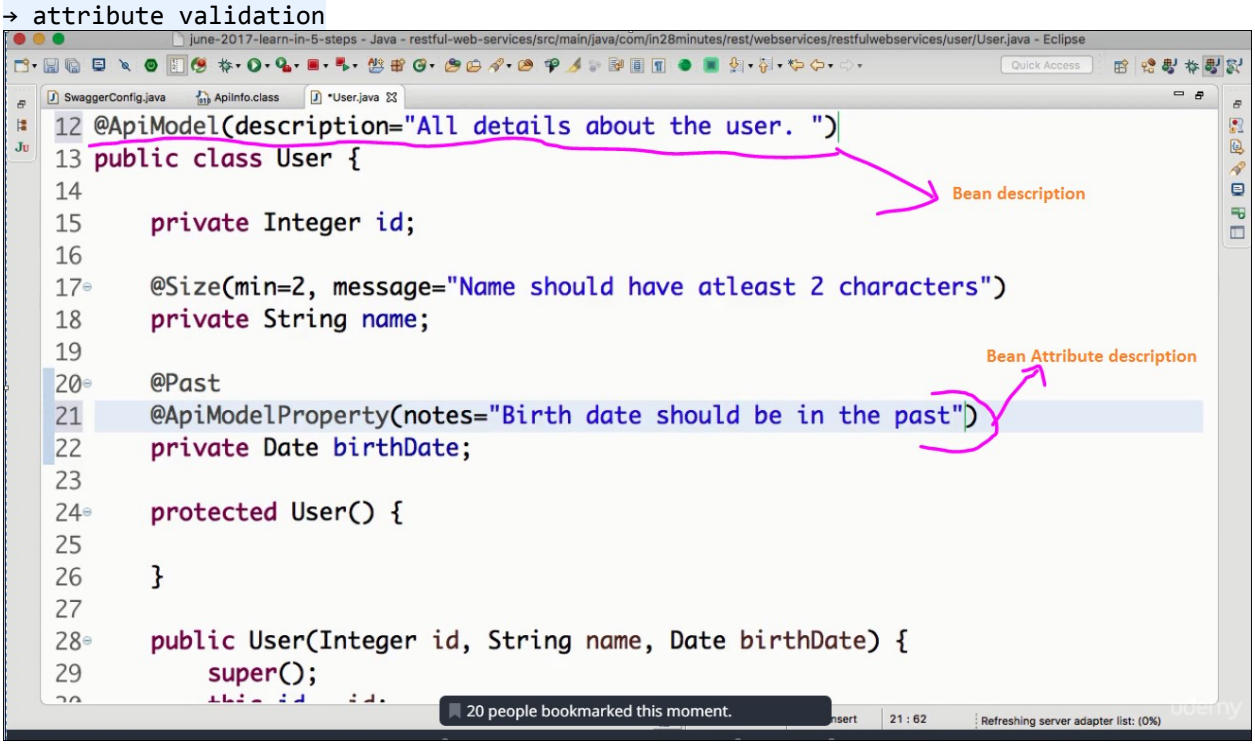
4. Now See the section of definitions. Bean Definitions are provided .



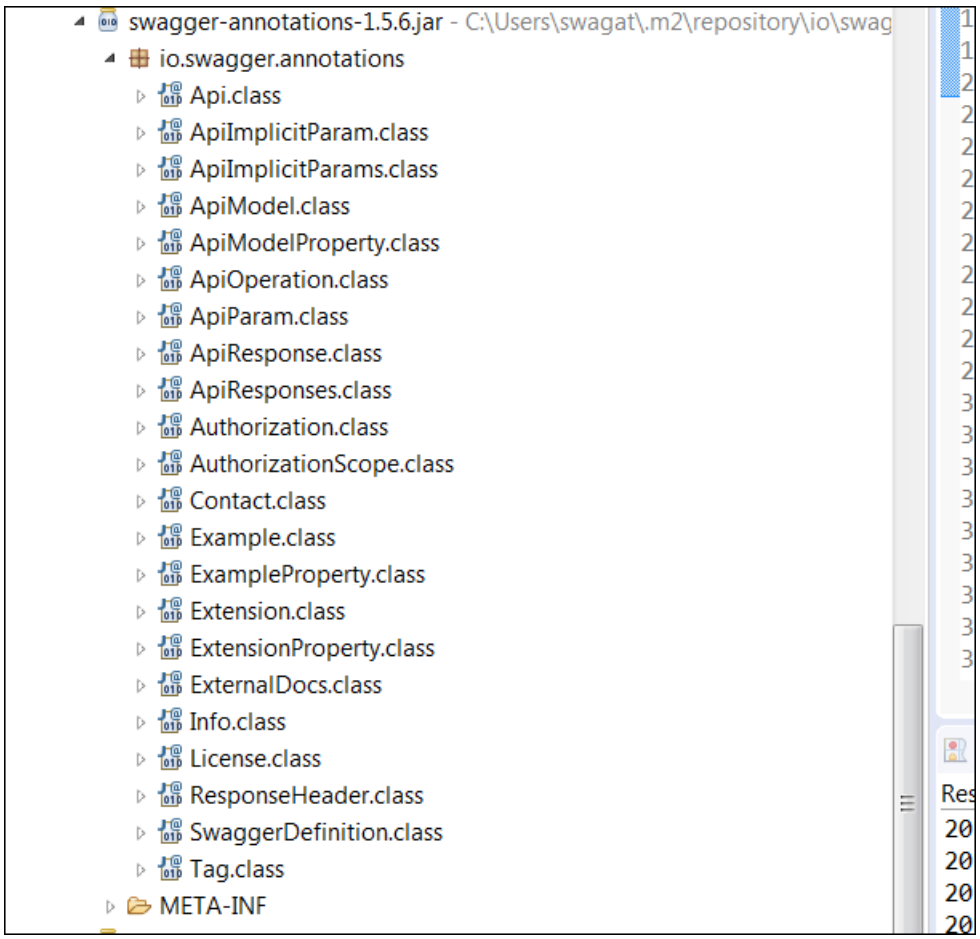
###Enhance:

Use the below annotations on the bean:

```
@ApiModel(description="All Details about the user") → Bean description
@ApiModelProperty(notes="Name should at least be 2 character & max of 50 charater")
→ attribute validation
```



Check out the swagger annotations below:



Lecture 35: Monitoring API with --- Spring Boot Actuator

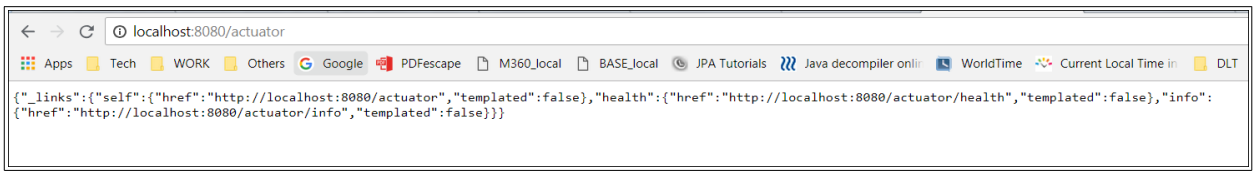
1. Add dependency in pom.xml

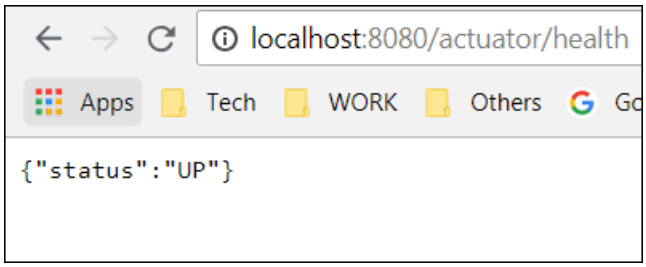
```
june-2017-learn-in-5-steps - Java - restful-web-services/pom.xml - Eclipse
*restful-web-services/pom.xml
32          </dependency>
33
34          <dependency>
35              <groupId>org.springframework.boot</groupId>
36              <artifactId>spring-boot-starter-web</artifactId>
37          </dependency>
38
39          <dependency>
40              <groupId>org.springframework.boot</groupId>
41              <artifactId>spring-boot-starter-actuator</artifactId>
42          </dependency>
43
44          <dependency>
45              <groupId>org.springframework.data</groupId>
46              <artifactId>spring-data-rest-hal-browser</artifactId>
47          </dependency>
48
49
Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml
```

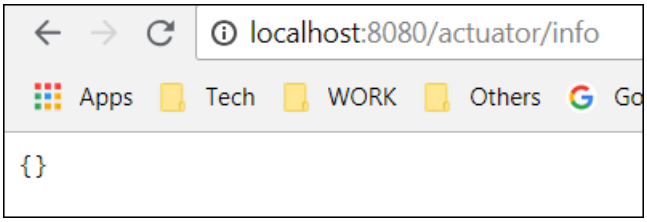
starter-actuator is for → monitoring health/matrices
hal-browser-→ for interpreting the actuator info. HAL => Hypertext Application Language
Ref: http://stateless.co/hal_specification.html
2. Restart the application
3. Check the URL :http://localhost:8080/actuator

```
localhost:8080/actuator
        href: "http://localhost:8080/actuator",
        templated: false,
    },
  - health: {
        href: "http://localhost:8080/actuator/health",
        templated: false,
    },
  - info: {
        href: "http://localhost:8080/actuator/info",
        templated: false,
    },
  }
}
```

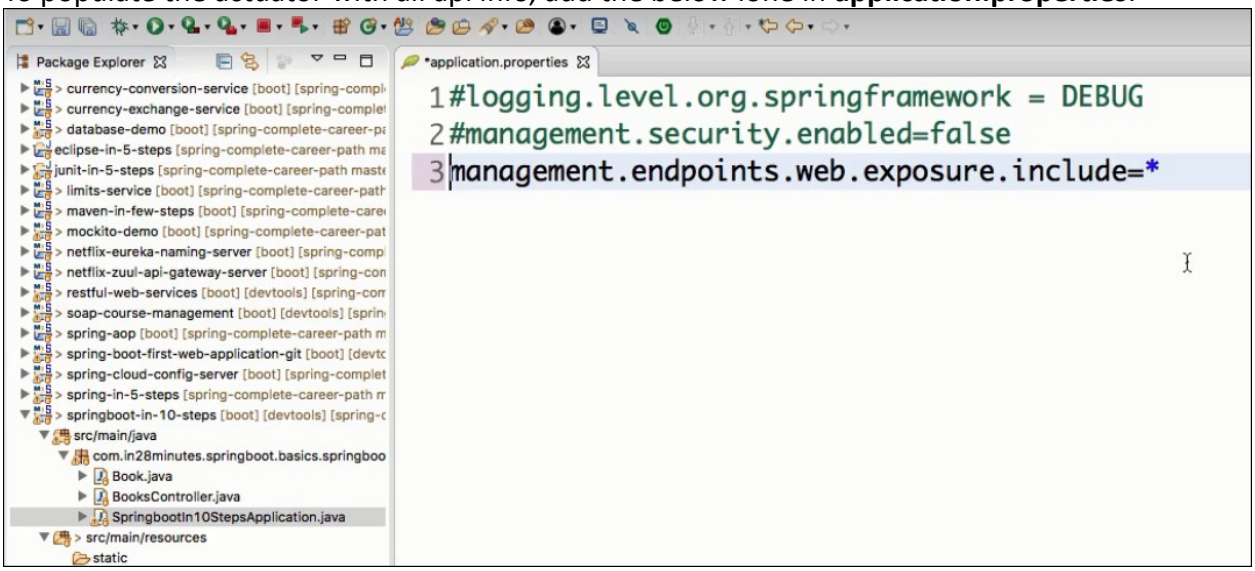
When visited these 3 links:

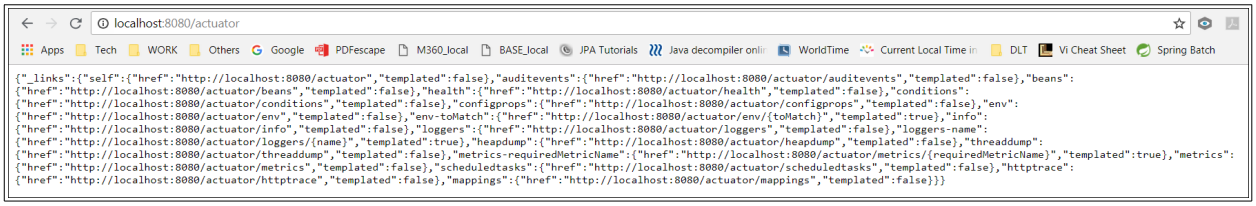






To populate the actuator with all api info, add the below lone in **application.properties**.





Now lets use the HAL-Browser to see the output: