**Spring Boot**

Spring Boot is a tool creating Spring based application.



Spring Boot is Opinionated i.e. it says it’s a good starting point.

It features convention over configuration – if you have to do 100 different things in order to configure something then looks at the 80 percent use case and have that be the default and 20 percent you need to configure it.

We can generate the spring application from Spring Boot .Moreover , Spring Boot is a stand alone application i.e. it does not require any web server/ container to deploy the application .What we get its something that we run directly without any deployment or configuration required.Just run the command and run the stand alone application.

Spring Boot application are production ready. We don’t need to do anything extra to make it ready for production.

**Setting Up Development Environment**

>> Set up JDK 1.8 in the machine.

>> Install Spring Tool Suite 4.7.2 based on Eclipse from spring.io – This IDE has some Spring specific features.

**Maven**

Maven is a dependency management tool.Maven lets us declare all the dependencies that we want in a single file .Using Maven we are going to add dependencies to the POM.xml file and run maven command which the dependencies get downloaded to the class path of the application.

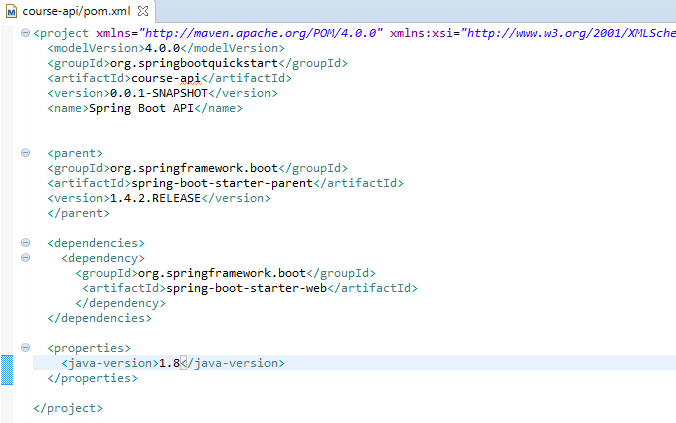
**Creating a Spring Boot project**

In POM.xml, parent element reflects , the project is child of parent project i.e. spring-boot-starter-parent project. The parent project and child project is a Maven concept.

Note :- 80 percent of spring project comes with default configuration and 20 percent will be our own configuration that we need to implement.

The spring-boot-starter-parent contains that opinionated set of maven configuration.

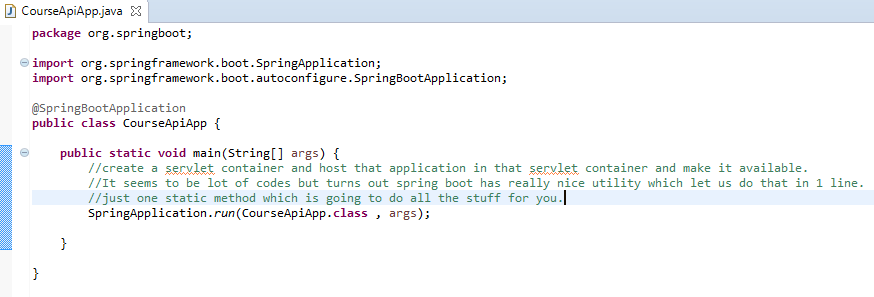
Now , the dependency element reflects , the set of web related jars , so that we wont need to add jars one by one.



Note : - Update project every time , if it anything gives error in project set up.

**Creating a Spring Boot application**

Create a class CourseApiApp.java under the package spring boot.



@SpringBootApplication tells the servlet container that this is the starting point of Spring Boot application.

Just run as we run simple java application.

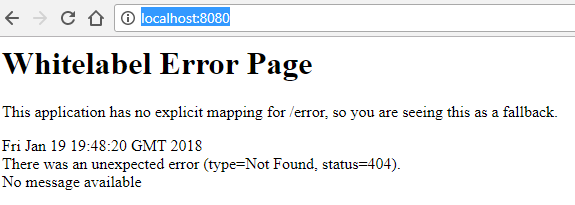
It shows the information in the console.

Tomcat started on port(s): 8080 (http)

2018-01-19 19:47:44.564 INFO 7544 --- [ main] org.springboot.CourseApiApp : Started CourseApiApp in 4.796 seconds (JVM running for 5.447)

To run the application in the browser type the below URL in the browser.

<http://localhost:8080/>



**Spring Boot Start-up Steps**

>> Set up the default configuration: - It covers the 80 percent of the configuration by default.

>> Starts the spring application context:- Spring acts as a container for all different services i.e. business services , controllers , data services etc. And this container is called application context. Every spring application has this application context which runs when spring application runs.

>> Performs class path scan :- Each class contains the marker with annotation .Depending on the marker it treats differently i.e. Service will treated as a service and controller will be treated as controller. This is done to identify different classes. In order to do all this , Spring has to scan the class path.

>> Starts the tomcat server: - Output of Spring Boot is a standalone application. That means, we need not create server container , it implicitly comes with Spring Boot framework.

**Adding Rest Controller**

We have a running servlet container, but it doesn’t do anything. If any request is given, then it throws any error page. What we want to do is to handle certain request. Given this request, I want this piece of code to execute on the servlet container. We can do that , by adding a controller in spring .Controller are basically java class having certain annotations. This annotations let spring know what is the URL you are mapping it to and what should happen when the request comes to this URL. So , there are two pieces of information’s that spring needs to know. We are proving both of them in the class itself and the annotations of that class.

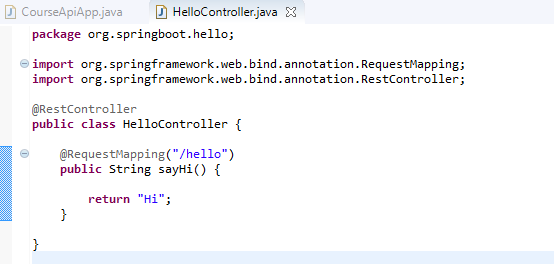
What we are going to do

>> We are going to create class which access the controller by marking with annotations. It also provides information about what URL access needs to triggered that controller execute .

>> Secondly, we are going to write the method needs to execute when that is triggered i.e. When that URL is called that the method to execute. So, write the method we are going to annotate it so that it gets mapped to the URL.

Secondly, we are going to write the method needs to execute when that is triggered i.e. When that URL is called that the method to execute. So, write the method we are going to annotate it so that it gets mapped to the URL.

Now , the web layer in the spring boot application reaches a spring framework called spring MVC.Spring MVC is child project of whole spring compiler and what is does it lets you build server side code which maps to URLs which provides responses. For example , responses such as REST API responses or JSON responses or full HTML page. We can have request maps to jsp or FTL response .In our case , we are going to build REST API , so we want to get simple string response.



In the above screenshot, HelloController class is created where there are two annotations.

>> One is the rest controller [i.e.@RestController](mailto:i.e.@RestController) which is marking class itself as the rest controller. Hence , this class deals with providing http requests and http responses

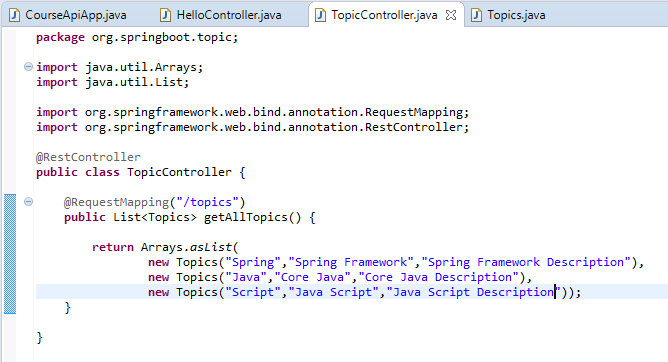
>> Second annotation [i.e.@RequestMapping](mailto:i.e.@RequestMapping) tells controller hey spring whenever there is a request with /hello just execute this method whatever the method returns just return it back.

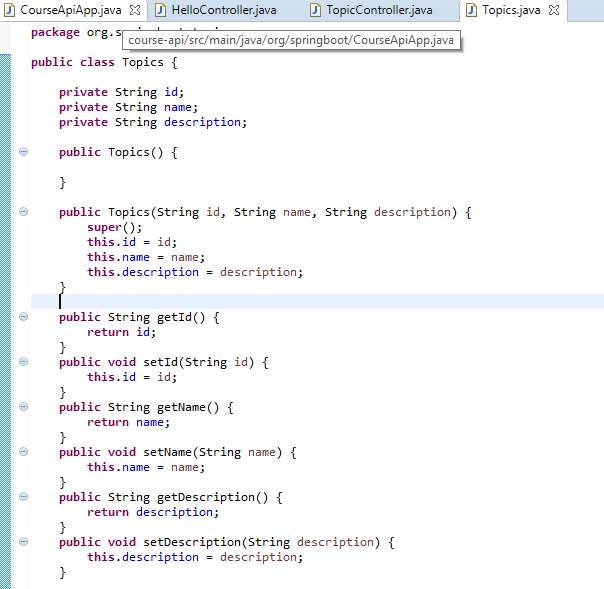
Note :- When ever you put this class in the class path , spring framework scans it and did know what needs to happen .And whenever the spring framework starts up it basically does the class path scan and when it scan it basically looking into all these classes and annotations like this as per the URL followed by forward slash with text.

**Returning Objects From Controller**

More controller mapping - Added one TopicController class which maps to the slash topics URL i.e. When request comes to /topics through get Request then we will respond the list of topic objects.

Note :- Here , the spring MVC framework will maps the topics request which will execute the mapped method and takes that return object and that return object will converted to JSON automatically and sent back as HTTP response. This is because the controller class is annotated to Rest Controller.

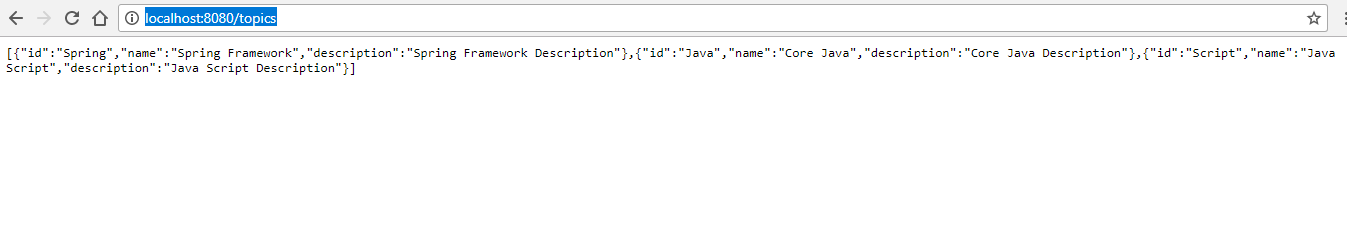




While we run the application by giving the below mentioned URL

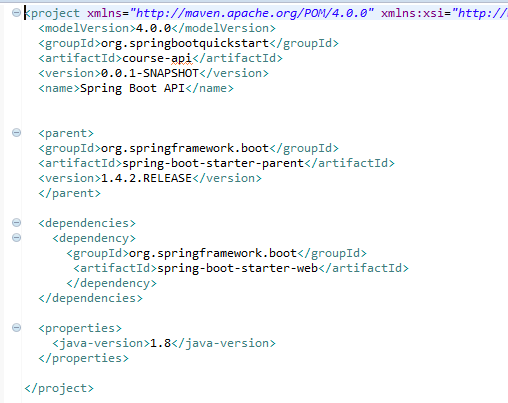
<http://localhost:8080/topics>

we will get the below mentioned JSON response as output



**Bill Of Materials**

In the POM.xml, there are two set of dependencies registered as shown in the below screenshot



In the above screenshot , the dependencies section tells Maven what jars to download and parent section configure what versions of those jars to download.

In addition to this , Spring Boot enables what JARs and what version without having to do all that work ourselves.

**Embedded Servlet Container**

>> Convenience :- While running application you don’t have to download the tomcat server , install it and then deploy it .However, in Spring Boot it is embedded implicitly which makes easy to run application during development time.

>> Servlet container config is now application config :- There are some servlet configuration steps that needs to happen i.e. Servlet container configuration steps. You may have to configure something on tomcat and most of the time the default works but there might be some configuration that we have to do in tomcat to have it run your application. However , now the servlet container configuration whatever that is will now become apart of application configuration. Just like you configure other things of your application in your source code you also have tomcat related configuration i.e. Everything is in one package.

>> Standalone application: - It’s easy to run and also easy to develop. If you remember this is one of the goal of spring boot project of standalone thing that can be executed.

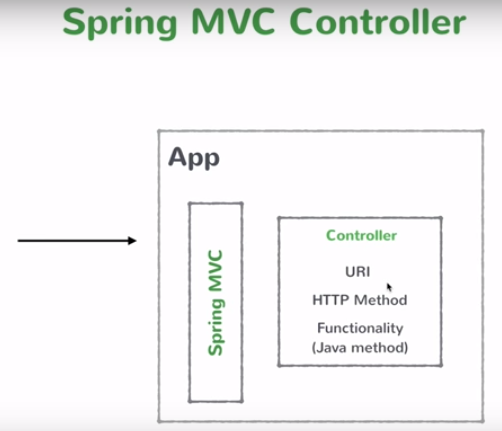
>> Useful of microservice architecture :- Just start one command and all microservices start executing being a part of standalone section.

These are some of the many reasons why spring boot deployable is something that includes the tomcat server with it.Moreover , we can choose any other servlet container , just specify the dependency in the POM.xml file.

**Unit 2: Spring MVC: The View Tier**

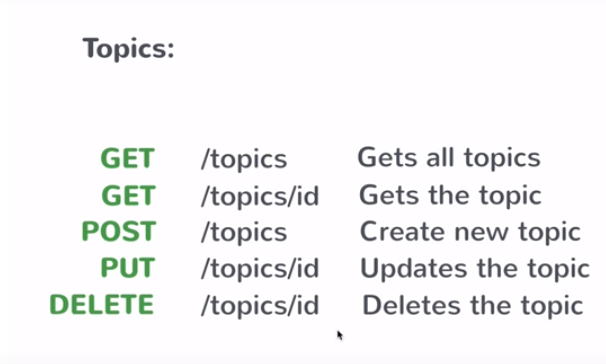
**How spring MVC works**

Spring MVC Controller



In the above snapshot , we will write the bunch of these controller classes and deploy them in the spring boot application. When the request comes in Spring MVC framework is going to look at the URI of that request and is also going to look at the http method .It can be POST request to URL A , PUT request to URL B , GET request to URL C .It looks at those parameters and then it examines the list of controllers has in the class path and see any of these methods having that URI and HTTP method maps the request that comes in .If there is a match then it executes that method and whatever its return type its convert into proper response and sends it back.

**The Rest API we will build**



In the above screenshot, id is the individual element.

**Creating a business service**

Instead of creating a new Topic list for every request, just move this to a new business service. This is a concept of business service in spring.

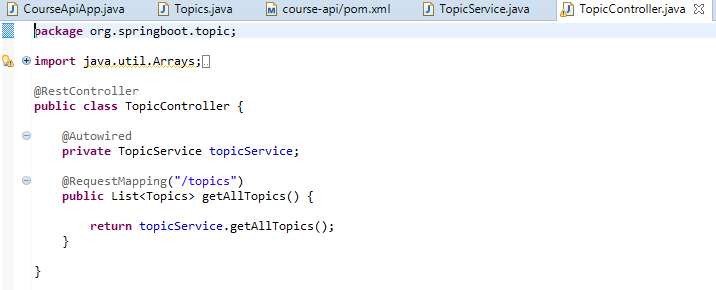
In Spring business service are typically singleton , when the application starts up spring creates a instance of this service and keeps that in its memory and register that instance. Moreover , other controller classes who do have dependency on this service injected to those different classes.

Once the class is annoted with @service then that particular class is marked as spring business service. In the application class path those classes detected with annotation @service it’s going to create an instance of those classes and register it.Now other classes which are going to call or use that instance of business service class its going to create a private member variable of business service class and need to tell spring to inject it by annoting @Autowired annotation to its private member .

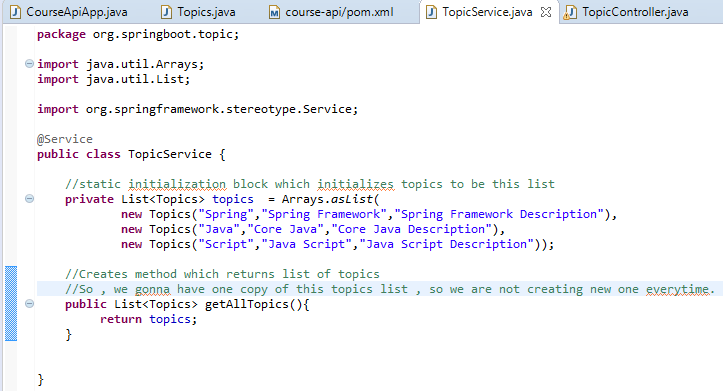
By doing this , when the request comes in with /topics i.e.getRequest() its going to call the service i.e. Call the method on that service and the method returns list of topics which is created which is initialized just once i.e.one copy of that list.

It is important to do this because proceeding further we are going to update list , adding topics , removing topics and thus its handy to have a separate list.

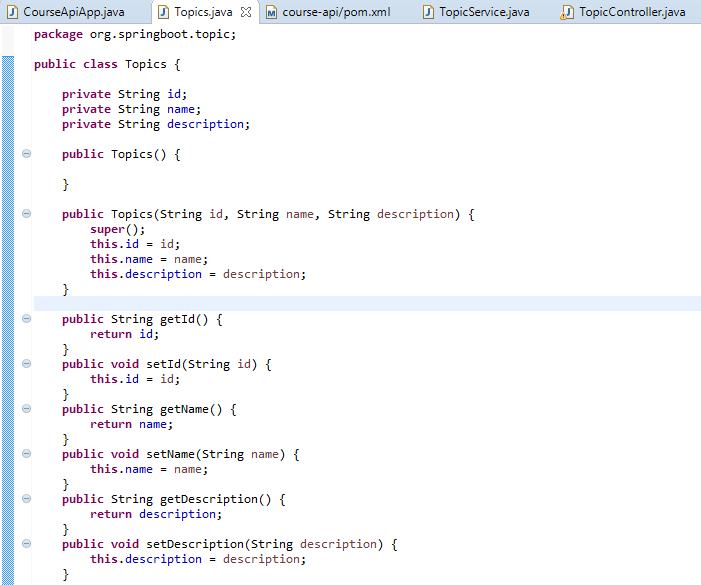
**TopicController.java**



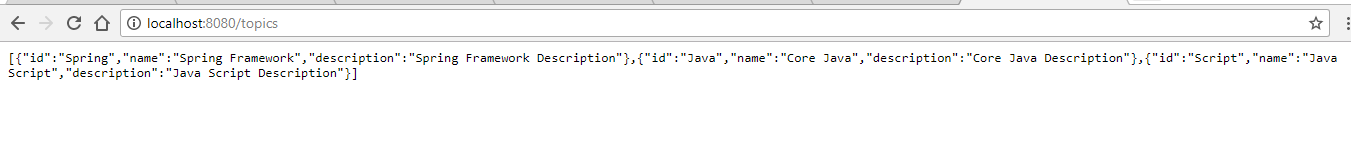
**TopicService.java**



**Topics.java**



While we run the application , gives the following response in the browser as shown in the below screenshot



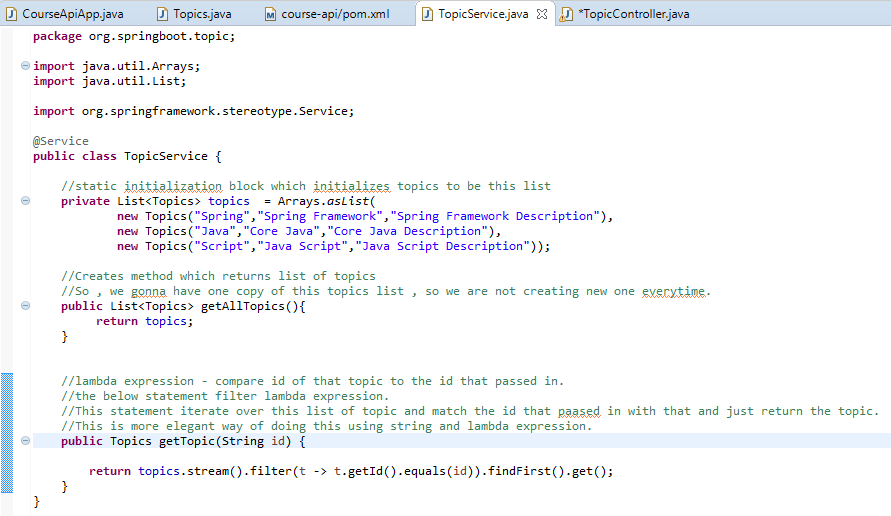
**Getting a single resource**

We will implement an API /topics/some\_id and the API will return only that topic which is requested.

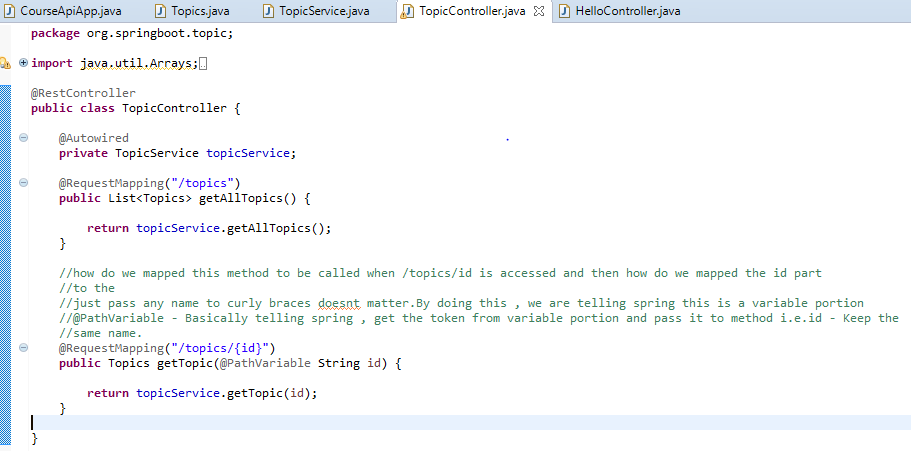
First, we need to write a method which need to execute when the URL is accessed. This is something which returns one topic and for that it accepts one id.When the request comes in to get the topic spring then it will get just one topic.

Just call a method on business service and pull off that particular topic.

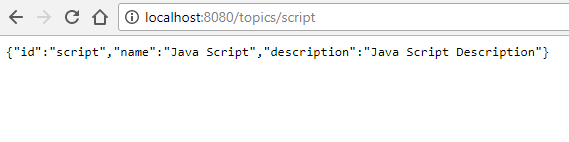
**TopicService.java**



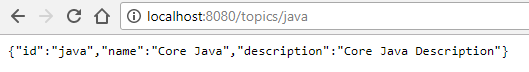
Secondly, we need to put @RequestMapping annotation on that method.



Finally, we will run the application and going to access /topics/spring and we will get



If we want to access /topics/java the we will get



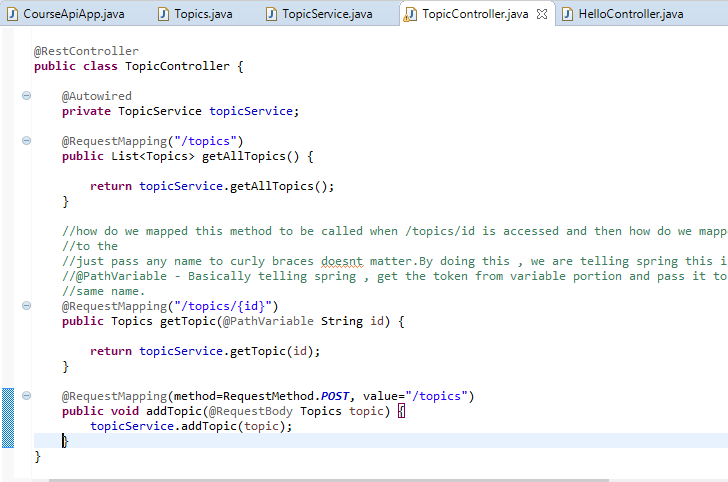
**Creating a new resource using POST**

We want to create new Topics and we will do this by making a POST request to the /topics URL i.e. When the request comes as /topics that will comes as POST request. In order to do this , we need to specify the POST .Now , we need to get that POST body convert into topic instant and add that to the list in the topic service. Spring MVC is smart enough to get the topic object from the request and convert into a topic instance. All we need to do is , pick the instance from the request body and there is a annotation for that @RequestBody.

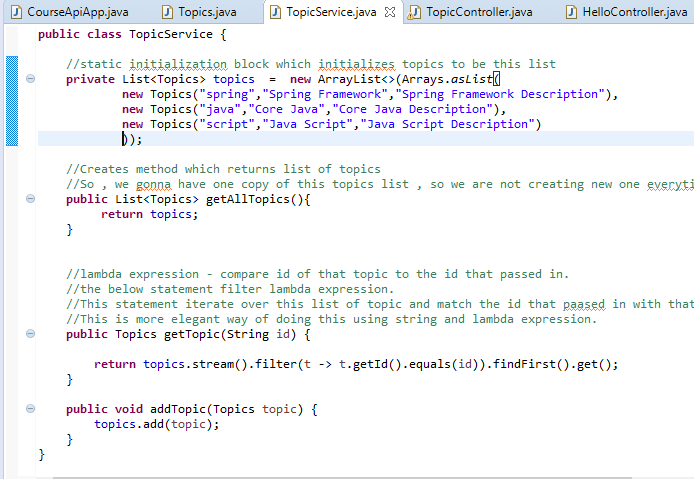
@RequestBody annotation :- It tells spring MVC that your request payload is going to contain JSON representation of this topic instance and we are asking to take this request body and convert into a topic instance and pass it to the addTopic method when the URL is mapped then the instance of the topic is ready to add in the list.

Postman tool is the chrome plugin which will let you make REST API calls. There are similar plugin for other browsers and this is what we are going to use for REST API calls.

**TopicController.java**



**TopicService.java**



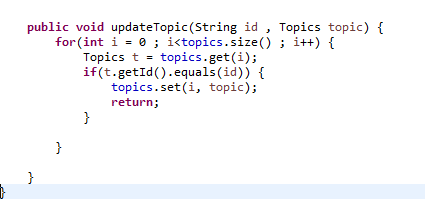
**Implementing update and delete operations**

Update Topic

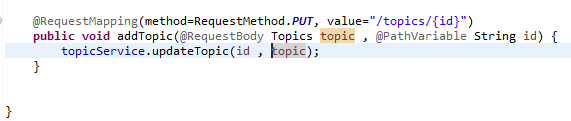
Each topic in the list will compare the input id and matches which we going to send the updated topic which is set to list and return. This is the service.

All we are going to have the updated service

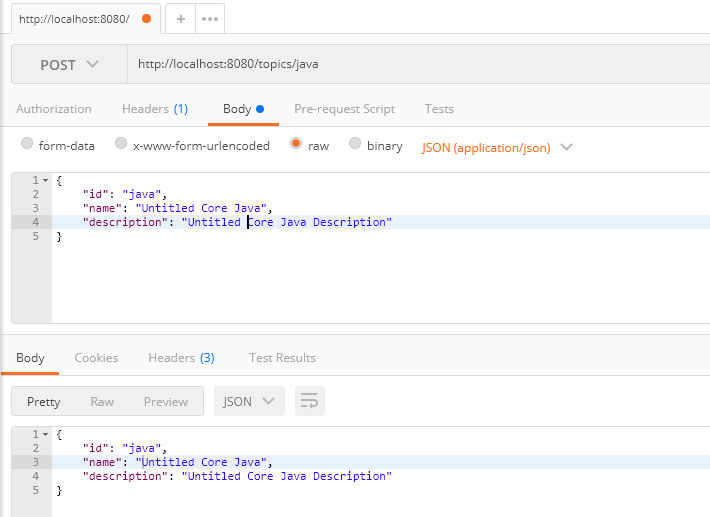
**TopicService.java**



**TopicController.java**



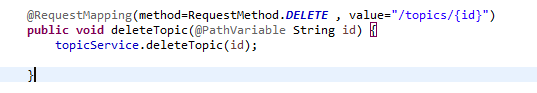
While we run the postman client ,



Delete Topic

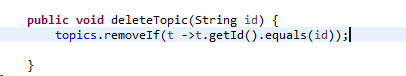
We have delete to the single resource URL which deletes the topic.

**TopicController.java**

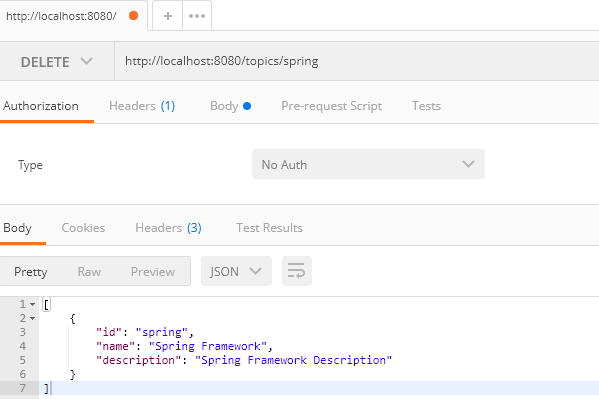


The below service , delete all topics where the brigade is true.

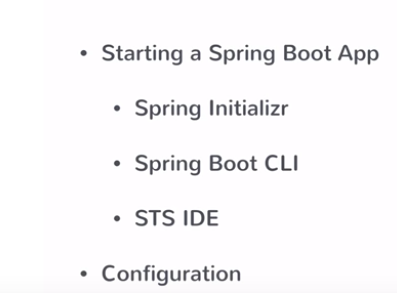
**TopicService.java**



While we run the postman browser on delete operation

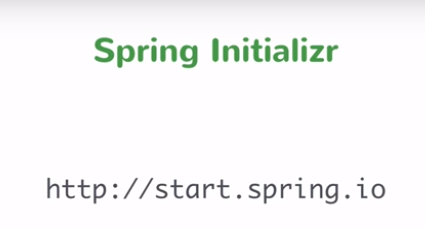


**Unit 3: Booting Spring Boot**

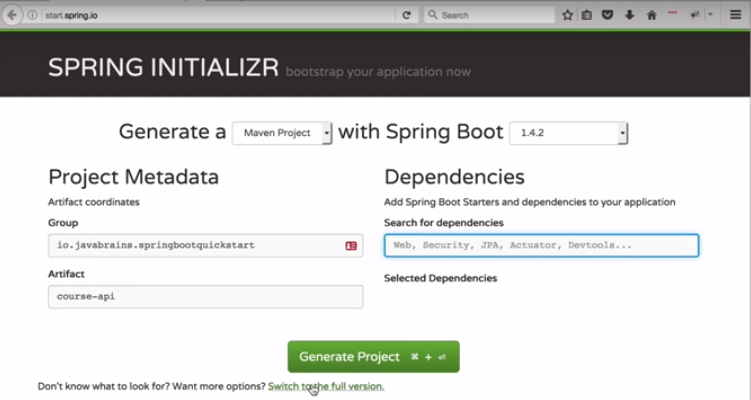


**Using Spring Initializr**

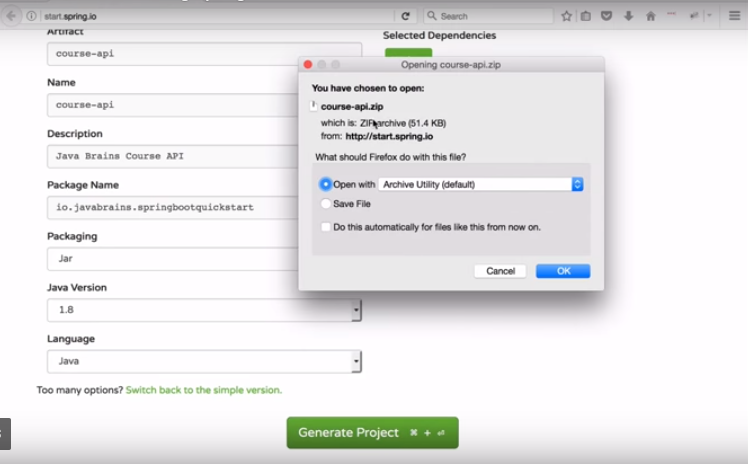
The below screenshot will let you start the Spring Boot application.



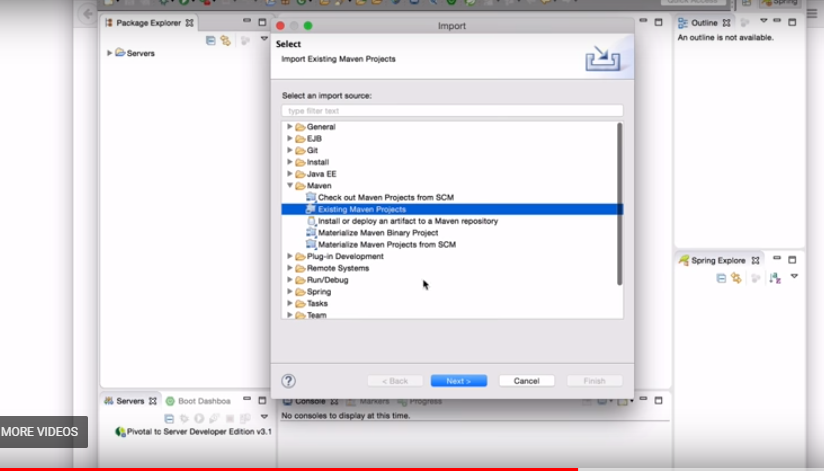
You can set up your application by opening the above URL and fill up the required fields as shown in the below screenshot



You can also switch to the full version where it gives you a lot of dependencies. Select and choose dependencies that you required to be downloaded in the POM.xml of your application .Finally , check all the things that you need and then generate project .



Once you click generate project then it download the zip file that you can extract and import that project from your IDE as shown in the below screenshot

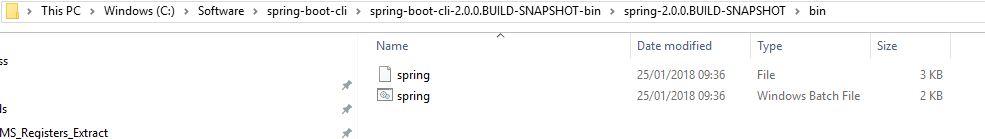


**Using Spring Boot CLI**

It is a tool which you can download from the official site of Spring Framework. Download CLI from the official tool as shown in the below screenshot.



After downloading, extract the zip file. It contains a bin folder, in which spring setup is stored. We can use it to execute Spring Boot application.



CLI executes groovy files. So, first, we need to create a groovy file for Spring Boot application.

Open terminal and cd into the bin location of cli folder.

**Using STS IDE**

There are four ways of creating Spring Boot project.

>> First , simple maven project – addning the spring boot dependencies manually.

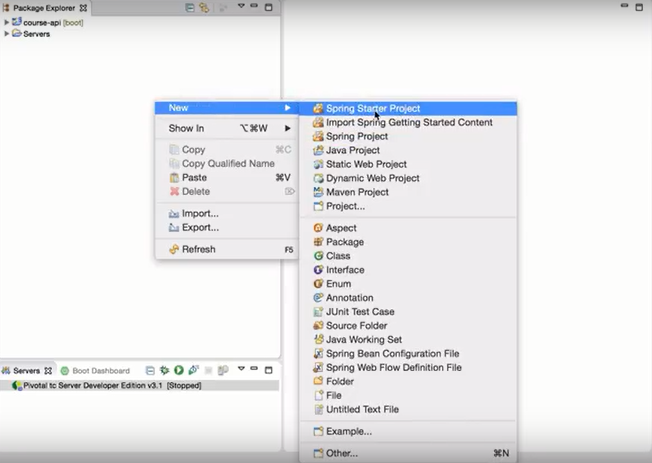
>> Second , online tool using the spring initializer – Choosing the project in the web UI and clicking download to download the project and then extract and open it in your IDE.

>> Third , using spring CLI i.e. command line interface

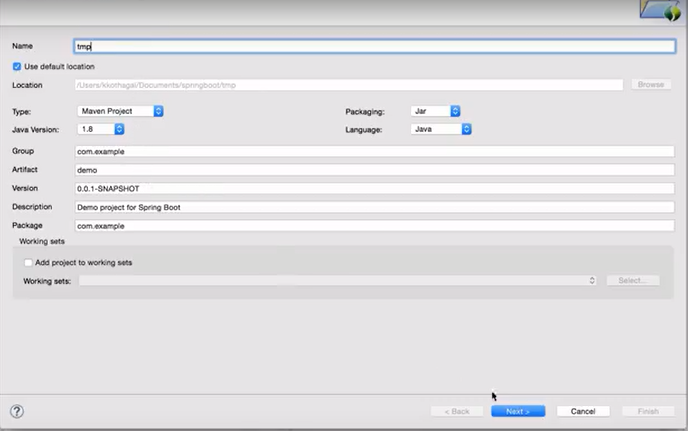
>> Fourth , using the spring tool suite – using the UI of the IDE i.e. to create a new spring boot project.

**Steps to create project using Spring – spring tool suite IDE**

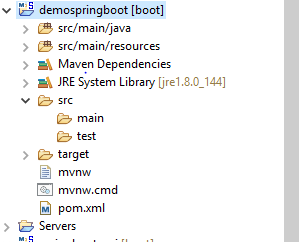
Step 1:- Click on New 🡪 Spring Starter Project as shown in the below screenshot



Step 2 :- Fill up all the details which will help you to create a maven project as shown in the below screenshot



Step 3:- Choose the spring boot version and dependencies that you need to download and add in your project , then click finish to downloaded , extracted and opened up a project in IDE as shown in the below screenshot.

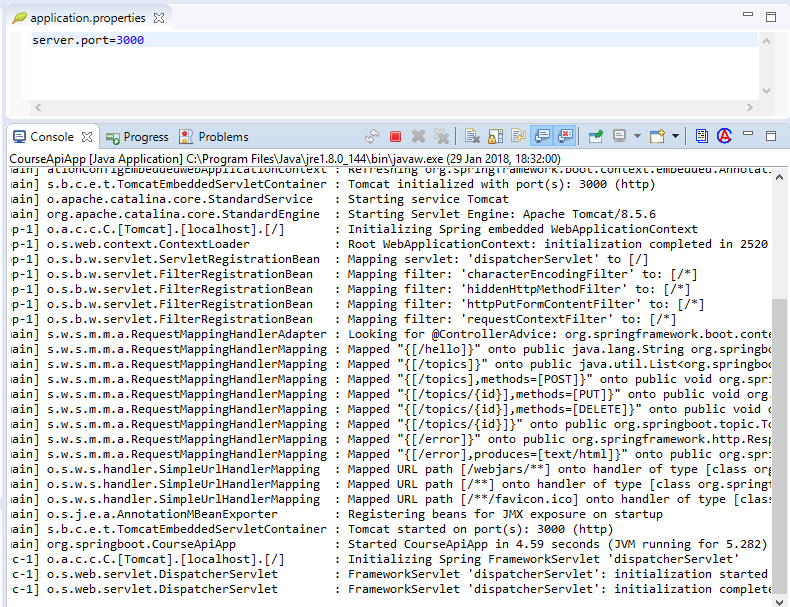


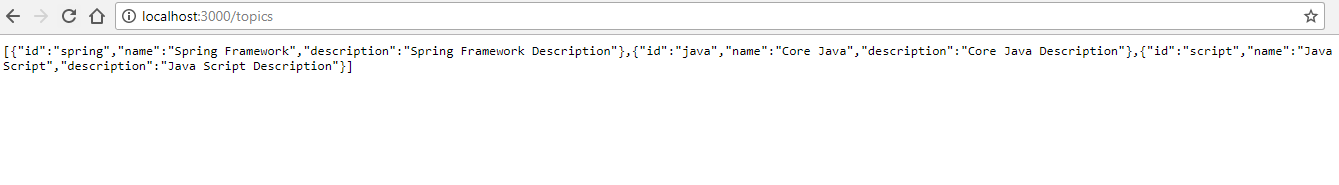
**Using application properties**

The way to customize spring boot application by using the property file i.e.application.property.This is actual way to customize spring application itself.

For example: - Configure the port of servlet container in the property file of spring boot application. The way to configure anything like this in property file in spring boot is to know what is the key is .Put the key and change the value then spring boot is automatically going to pick it up.

Start the application in the localhost 3000 i.e. **http://localhost:3000/topics**

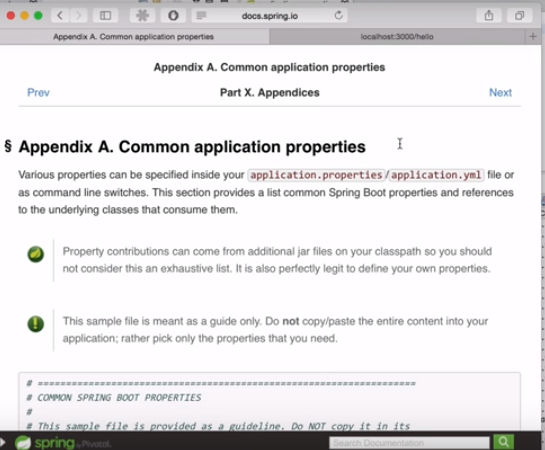




Our application is running on port 3000 that we changed in the property file.

How do we know what to add in the application.properties.

We can search this in the spring boot website i.e. common application properties as shown in the below screenshot.



**Spring Data JPA: The Data Tier**

**JPA**

We can have our spring boot application connect to a database , run queries on the database , return results and also make updates to the database.This is the whole CRUD operation.

We will learn what are the configuration steps required that we need to connect to a database.

JPA is basically a specification that lets we do ORM when we are connecting to a relational database which is object relational mapping.

Most of the enterprise applications relay on sql databases which is relational database.

So , when we need to connect from our java application to a relational database the we need to use something like JDBC and run SQL queries which could be named queries which run on relational database. After we get result which convert into object instances. In java application we have classes and objects.

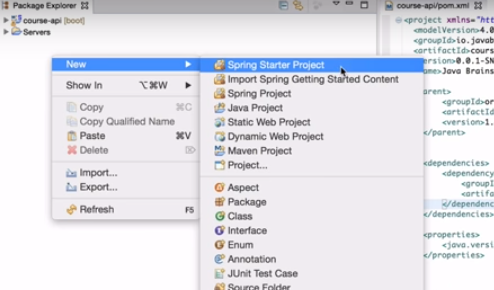
ORM: - It let us map entity into sql tables, so that we can connect to a database. We provide some kind of metadata on our entity classes so that we don’t have to do the query and then mapping yourself. It’s the framework that handles it for you.JPA is the API that let us configure our entity classes and give it to a framework , so that the framework does the rest. It handles the conversion from sql queries to object instances. Spring data JPA is the separate project which lets us make working of all these ORM tools even easier.

Spring data JPA let us connect to a relational database and let provide JPA related configuration on our entity classes then the framework handles the lot of work for you.

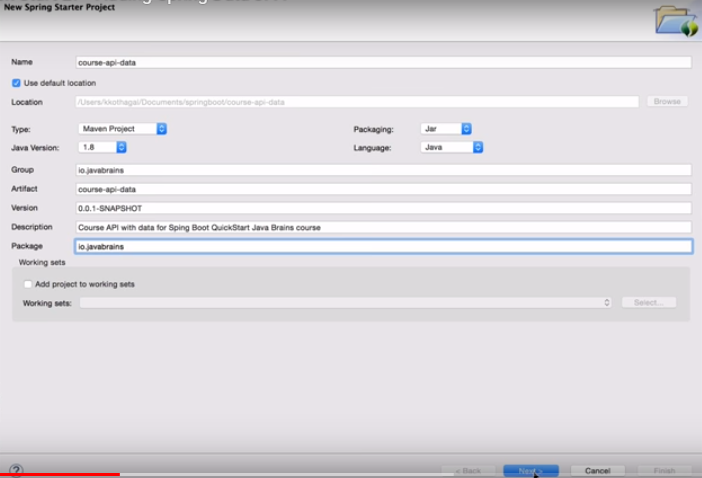
**Adding Spring Data JPA**

Add few more dependencies in order to connect to database.

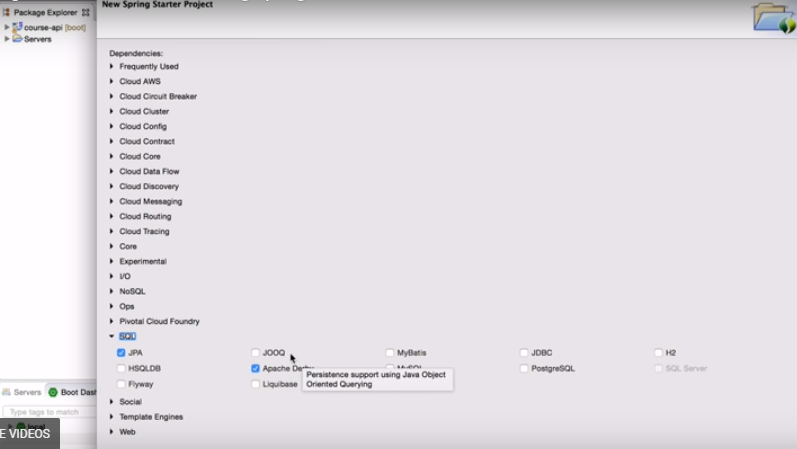
Create a new starter project



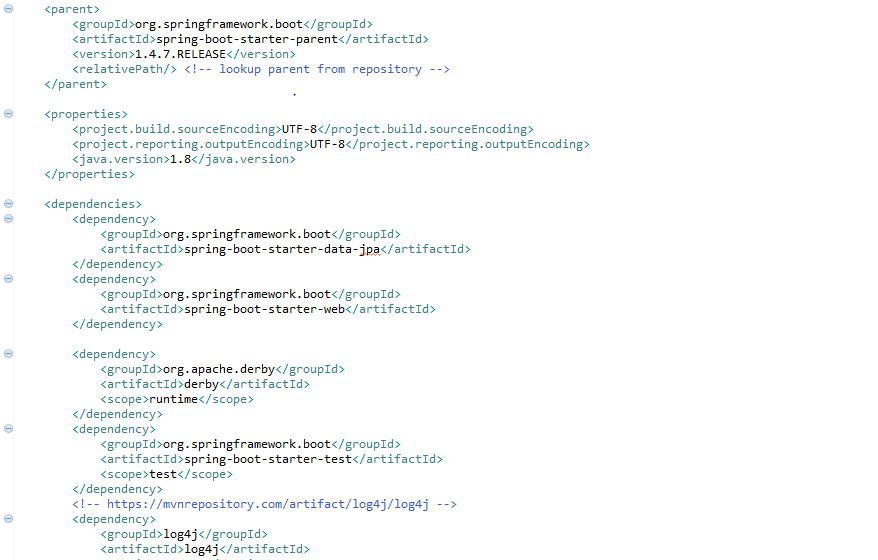
Add values to the new new window and click next



Now , in the next window , we can choose the dependencies such as web , JPA – brings in spring data JPA , Apache Derby – embeded database , download the JAR into your class path which is the Apache Derby database and actually starts your database when you start your application. This is typically not used in production scenario but when your are building application for development i.e. for test database .Finally , after all selection of required dependencies click next and then finish which downloads it and open project in our Eclipse STS IDE.



Below screenshot shows the list of dependencies in POM.xml file



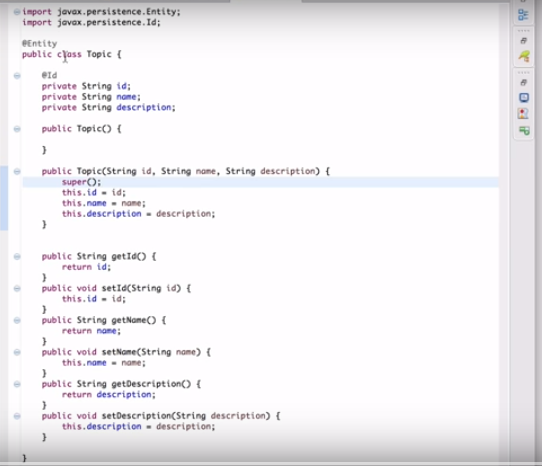
Here , the list of dependencies shows the starter-web dependency for web needs and starter-data-jpa for jpa needs .This altogether , all we need the web application connects to our database. Moreover , derby dependency is the database which leaves in our class path of our project.

**Creating a Spring Data JPA Repository**

The JPA and derby database dependencies added to the class path of the project.

Model class which is the entity class i.e.Topic.java class .We need to tell JPA that the Topic class need to save to database. We need to save Topic instances. We need to map Topic object to the relational database table. Ideally , Topic table to be created and each one these members variables should go as columns variables and each Topic instance of this class should go as rows in that table. All we can do this by using annotations. Once we provide @Entity in the class then JPA knows to create a table called Topic and the table has three columns i.e. id , name and description and all are will be text column. In the relational database every table should have the primary key .Whatever member variable that corresponds to the column that we want to be the primary key then that member variable marked in the class as @Id annotation. So , with this , JPA exactly know what to do to create a table. Now , it can convert Topic instance as row in the database table and it can convert the row and database table as Topic instance. We can do both. JPA can run queries for us and create Topic instances and vice-versa. This is the first step.

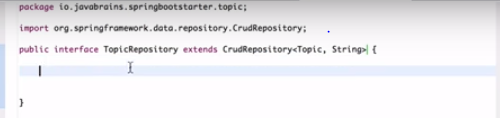
**Topic.java**



The second step is to use the Topic Service to connect to the database and run those commands. We need to tell JPA, hey we want to save this Topic instance /read this Topic instance. Let’s create a TopicRepository interface which extends CrudRepository interface which we will get all the common methods without implementation i.e.getting all the entities, selecting a particular entity , updating a particular entity etc and we need to implement only those special methods as per our requirement i.e. for custom need – selecting a particular text in the description.

CrudRepository is actually a generic type and need to provide generic type information necessary because all the common methods have type information i.e.getting a list of topics then the method returns list of topic classes , getting list of courses and returns list of topic courses. The interface CrudRepository requires couple of generic type information i.e. the entity class that we are working with which is Topic class , second generic type is what is the id do this entity class have which is String.

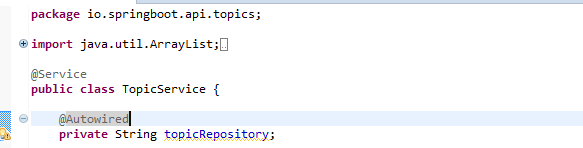
**TopicRepository.java**



**Making Crud Operation With Repository**

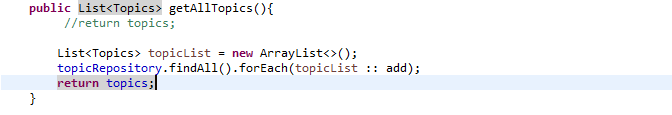
First of all, get the TopicRepository instance in TopicService class. The best way of doing this by using @Autowired annotation to member variable. Now, with this when Spring creates an instance of TopicService its going to inject an instance of TopicRepository. So, when we implement our method its going to use TopicRepository field knowing that its there.

**TopicService.java**



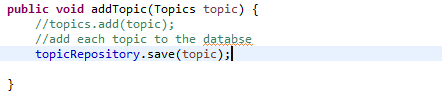
Now , get list of all topics/get all the instances from the table – findAll() method .Its a iterable , so it can be converted into a list and sends it back as shown in the below screenshot.

Note :- Just connect it to database , run a query to get all the topics , convert each of those rows into topic instances and get it back .Here , findAll() method is doing all that for us automatically.

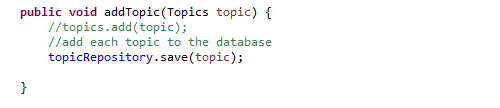


Here, each of the element that its find put that element in the list. So, each of the element in the iterable calling the add method on the topics and passing that element.

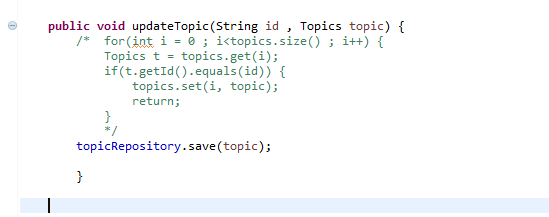
If we want to add each topic to the database then check the below code screenshot



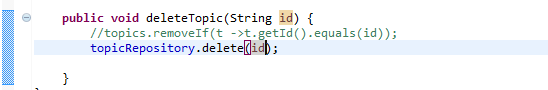
If we want to return a single record from the database which takes a id as String please check the below screenshot



If we want to update a single record then use the save() method from repository. Moreover, save() method can do both update and add. Well it can because we are sending it a topic instance and the topic instance has the information what the id is and what is instance itself is.So , the repository knows enough there is already a row in the table for that particular id .If that row does not exist then its going to do an insert and if the row does exist it knows that it need to do an update. We don’t need two separate methods in the repository. As long as the instance has the id set on it we don’t even need to pass in the id.If All we need to pass in the instance.



If we want to delete a record then check the below code as shown in the screenshot



**Unit 5: Deployment and monitoring**

**Packaging and running a Spring Boot app**

We can do things in the production deployed application.

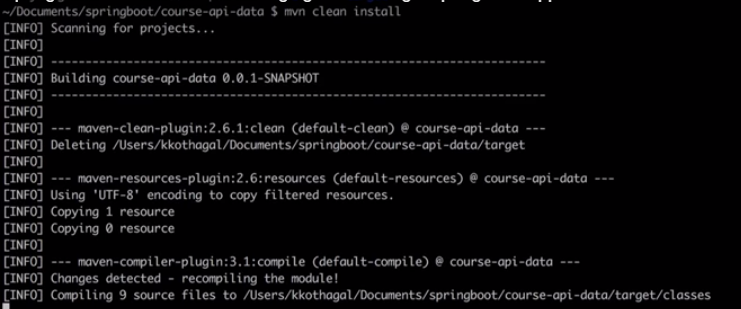
Getting the JAR file of the spring boot application, which can be deployed in any machine.

Here, we don’t need servlet container because you take just JAR file of the application and run it.It gives flexibility to run application. Just we need a java run time environment.

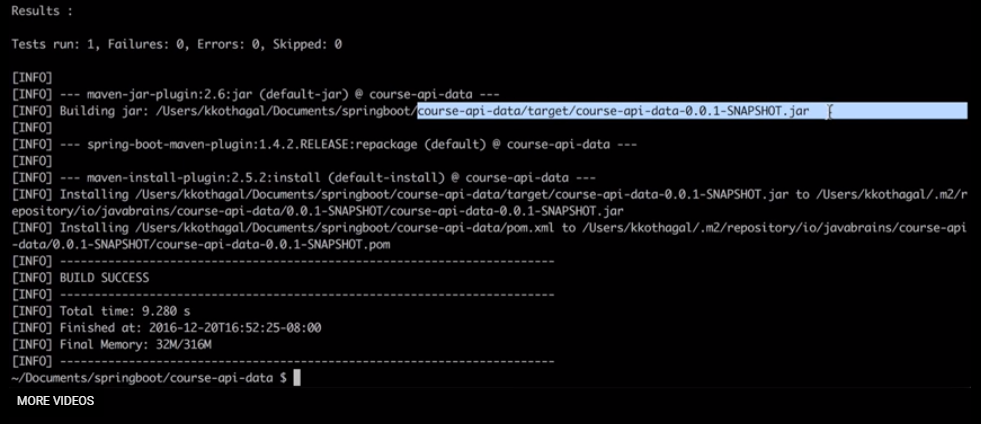
Note :- The spring boot application also bundled in servlet container and related configuration.

Generate JAR file in Spring CLI as shown in the below screenshot.

The best way of creating JAR file is **mvn clean install**



It downloads the dependencies if any which is not complied and create the JAR file in the project directory as shown in the below screenshot

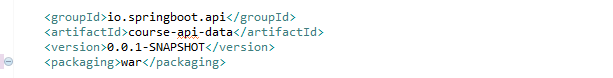


If we outside the context of IDE , we can still run application by specifying one command i.e.**java -jar target/course-api-data-0.0.1-SNAPSHOT.jar** as shown in the below screenshot

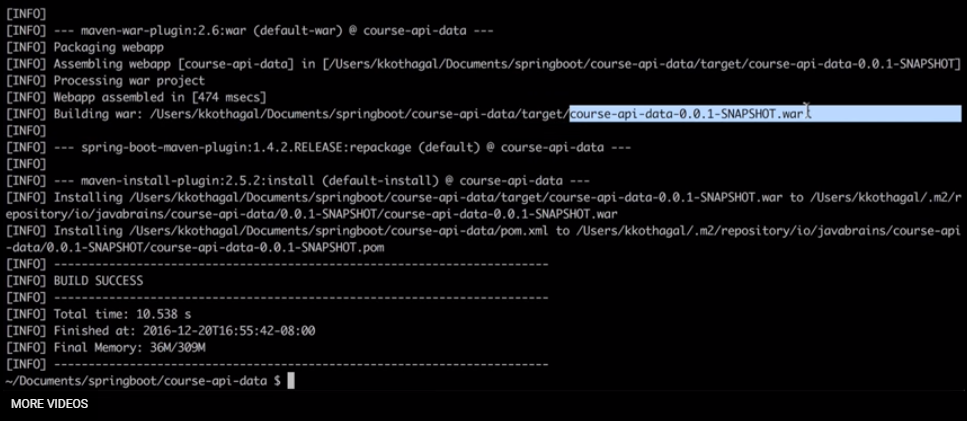


This is easy for us to start our application. This is very handy when we have microservice based environement.We have a bunch of stateless servers which start up really quick.

If we want to create WAR file of spring boot application then modify in the POM.xml from jar to war as shown the below screenshot and then run the above steps as we created JAR i.e. **mvn clean install**



Now , once we run mvn clean install then we will get a WAR file in the same project path as shown in the below screenshot.

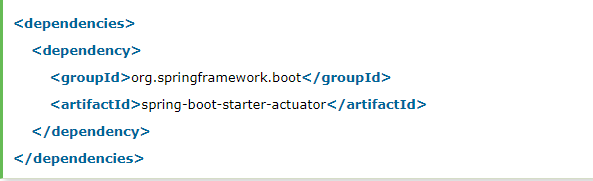


These are some of the process where we can get our project in executable format and deploy to run it any environment.

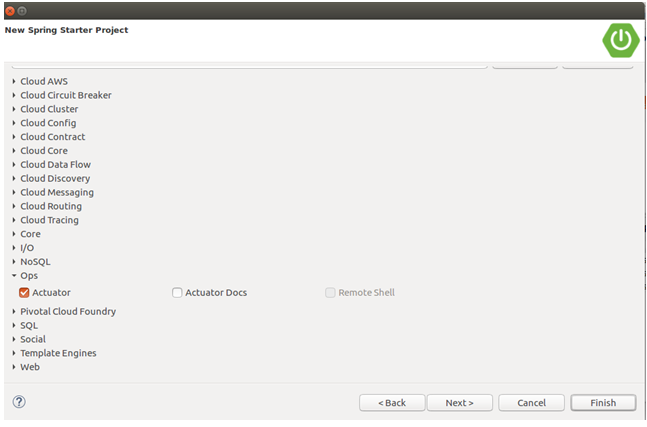
**Spring Boot Actuator**

Spring boot provides actuator to monitor and manage our application. Actuator is a tool which has http endpoints. When application pushed to production, we can choose to manage and monitor our application using HTTP endpoints.

To get production ready feature we should use spring-boot-actuator module. We can enable this feature by adding it to the POM.xml file.



In addition to this , we can opt it while creating project by using the starter wizard.



It will do the same as we explicitly add in the POM.xml file.

**Endpoints**

Actuator endpoints allow us to monitor and interact with our Spring Boot application. Spring Boot includes number of built-in endpoints and we can also add custom.

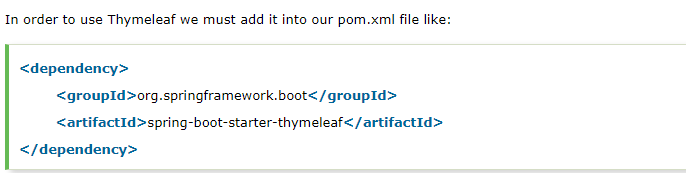
Just , follow the spring official website which provides us bunch of endpoints which makes very easy to look at what is happening.

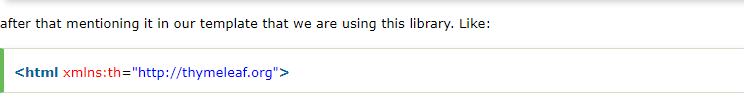
**docs.spring.io** – Official website

**Spring Boot Thymeleaf View**

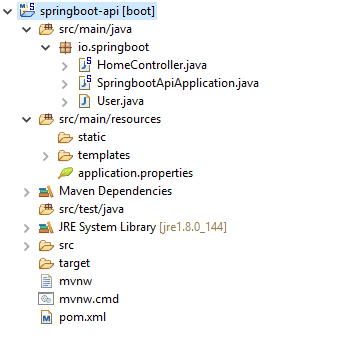
It is a server-side java template engine for web application. Its main goal is to bring elegant natural templates to our web application. It can integrate with spring framework and ideal for HTML5 java web application.

For example, we can use thyme leaf as out html template and rendering it from controller.





The project structure shows as shown in the below screenshot



**Index.html**

<html lang=*"en"*>

<head>

<title>Index Page</title>

</head>

<body>

<form action=*"save"* method=*"post"*>

<table>

<tr>

<td><label for=*"user-name"*>User Name</label></td>

<td><input type=*"text"* name=*"name"*></input></td>

</tr>

<tr>

<td><label for=*"email"*>Email</label></td>

<td><input type=*"text"* name=*"email"*></input></td>

</tr>

<tr>

<td></td>

<td><input type=*"submit"* value=*"Submit"*></input></td>

</tr>

</table>

</form>

</body>

</html>

**User-data.html**

<html xmlns:th=*"http://thymeleaf.org"*>

<table>

<tr>

<td><h4>User Name: </h4></td>

<td><h4 th:text=*"${user.name}"*></h4></td>

</tr>

<tr>

<td><h4>Email ID: </h4></td>

<td><h4 th:text=*"${user.email}"*></h4></td>

</tr>

</table>

</html>

**HomeController.java**

**package** io.springboot;

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

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

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

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

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

@Controller

**public** **class** HomeController {

@RequestMapping("/")

**public** String index(){

**return**"index";

}

@RequestMapping(value="/save", method=RequestMethod.***POST***)

**public** ModelAndView save(@ModelAttribute User user){

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.setViewName("user-data");

modelAndView.addObject("user", user);

**return** modelAndView;

}

}

**User.java**

**package** io.springboot;

**public** **class** User {

String name;

String email;

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** String getEmail() {

**return** email;

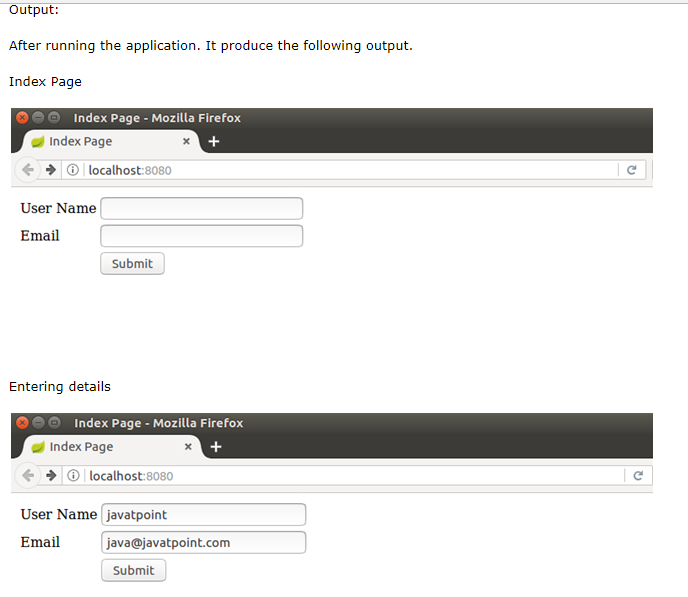
}

**public** **void** setEmail(String email) {

**this**.email = email;

}

}



On submitting the form

