**SPRING BOOT:**

* Spring boot is a tool that lets us to create and run standalone, production-run spring based applications easily.
* Spring is a framework that allows us to create standard enterprise applications which has its own features like Dependency Injection, JDBC, AOP, Security and MVC. Spring Boot takes care of lot of broiler plate code that we write and provides us flexibility to concentrate more on the Business Problems. It provides infrastructure support as well. Writing queries for us and connecting to Data Base and providing security to the web application.
* Drawbacks in spring framework:
  + It is a huge framework and provides support for almost all the requirements of us.
  + There are multiple configuration steps we have to follow when we are developing spring application like getting the dependencies in the pom and if we need to connect to the database then we have to configure the steps where it makes bit difficult for us.
* Spring boot generates standalone spring application that we can simply run, we don’t need to configure any server and we don’t need to build and deploy the application in the server.
* When we create a spring boot application we need to tell that this is spring boot application by annotating the class with main() as @SpringBootApplication. The class annotated with this annotation is the starting point for the Spring Boot Application.
* SpringApplication.run() when we execute this method inside the main() of Spring Boot Application then below sequence of steps will execute:
  + Sets up the configuration, we don’t need to create a DispatcherServlet as we does in plain Spring MVC Application.
  + Creates and Starts the Spring ApplicationContext container.
  + Scans the class path for the @Service, @Repository, @component and @Controller or @RestController annotations and treats that classes as specified.
  + Creates the Servlet Container and Starts the Tomcat Server.

**What’s happening behind the scenes?**

**What actually below set of lines doing inside the pom.xml:**

**How actually Spring boot downloads all set of jars:**

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.0.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent> |

* The above set of lines provides the configuration information to the maven i.e. which version of jars has to be downloaded, and how many jars has to be downloaded for the specified maven dependency.
* For Example below is the dependency I have mentioned in my pom.xml.

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency> |

* Spring boot starter parent 2.0.0 downloads 36 jars for the spring web application to work and version of the spring jars is 5.0.4.RELEASE. This is how the spring boot starter parent works and downloads the jars for our requirements.
* When I changed the spring boot starter version to 1.3.3.RELEASE then it has downloaded 33 jars and spring jars version is 4.2.5.RELEASE.
* The unit of jars that spring boot downloads for the application to work is called Bill of Materials by Spring Framework.

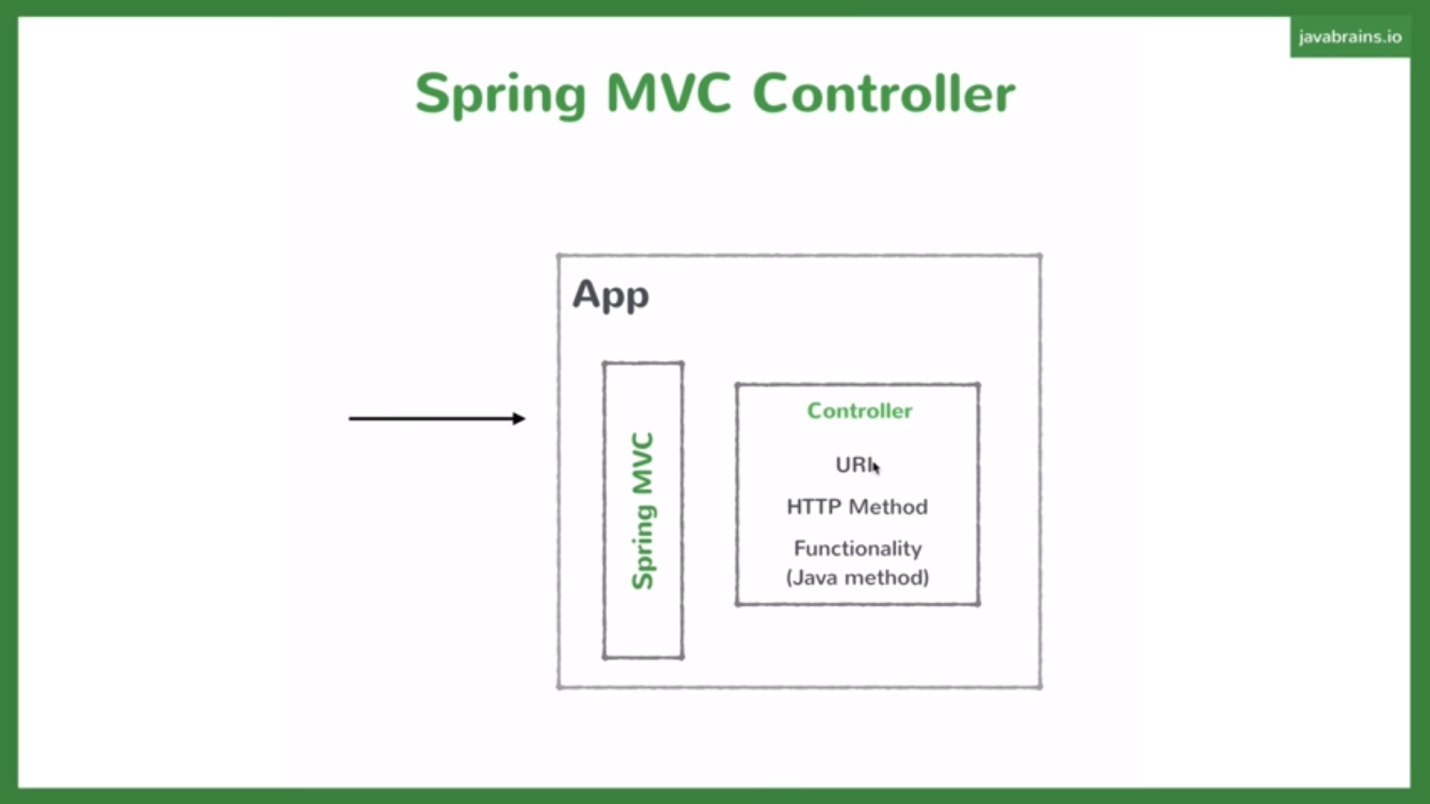
**EMBEDDED TOMCAT SERVER WITH THE SPRING BOOT APPLICATION:**

* Spring boot provides an embedded tomcat server along with standalone spring mvc application.

**Advantages with the Embedded Tomcat Server:**

* We don’t need to download, install and deploy our application explicitly. By running a simple command we can actually execute and deploy our application and hence can make our application as the standalone application.
* Whatever the explicit Servlet Container configuration we do now it will be part of the Application.

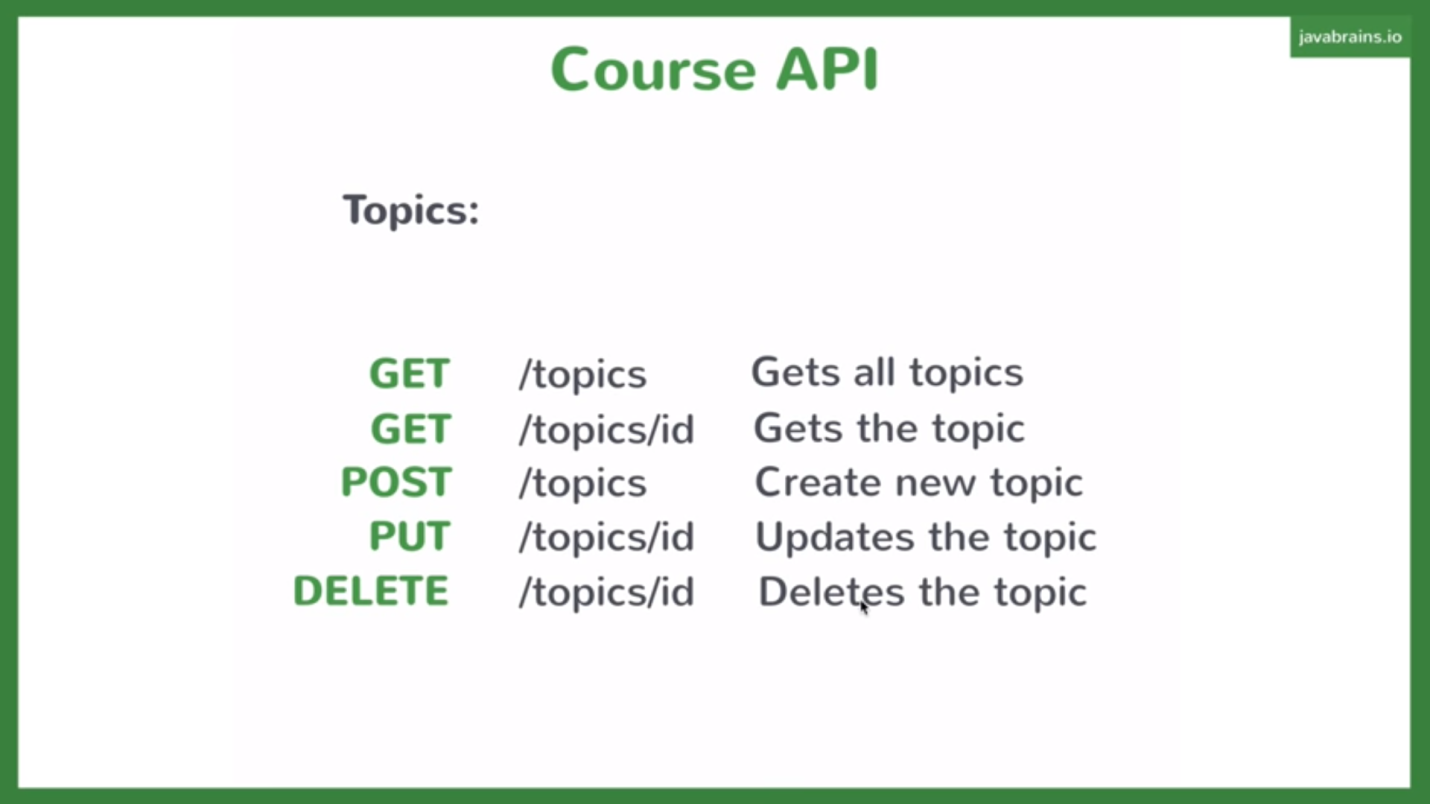
**Functionality of the Spring MVC and Controller is as below:**



* Spring MVC is actually a MVC framework and that scans the class path controllers with @Controller or @RestController for JSON response and searches for the corresponding URI and the HTTP method (GET or POST or PUT) and maps the same to the Request Handler Methods which actually processes the requests and sends the response. This response will be converted by Spring MVC framework to JSON if we add @RestController at the top of our class.
* Controller class is actually a simple Java class. Above diagram shows diagrammatic representation of the Spring MVC and the Controller functionality.

**BUILDING THE REST API:**

* Before we build an API in our case its COURSE API, first we need to identify the resources that are required for our API. The resources in our API is Topic, Course and Lesson.
* And we have to follow the standard REST API standards as below:
* Let’s take Topic Resource first:



* Suppose if we need to get all the topics of a particular course then we will use GET Http Method and /topics Request URI.
* The same way for we will use POST if we want to create a new topic and DELETE if we want to delete any topic based on id and finally PUT method if we are updating the contents of the particular topic.

**IMPORTANT POINTS ABOUT @SpringBootApplication Annotation:**

* We need to specify a certain way to spring to identify the spring components which we have used in our project so that it will create the same and place in the Container.
* @SpringBootApplication annotation is a combination of @SpringBootConfiguration+@EnableAutoConfiguration+@ComponentScan annotations and this @SpringBootApplication will auto scan for the Stereotype annotations like @Component, @Service, @Controller, @Repository annotations in the same package and its subpackages.
* If we specify the stereotype annotations other than current package and its subpackages then we need to explicitly scan those packages with the @ComponentScan otherwise dependency injections will fail.
* To switch-off the banner generated by the spring boot we have to set the below property value to off.
* spring.main.banner-mode=off. It can take 3 values off, console and log.
* Console value writes into console.
* log value writes into log file.
* If we want to create custom banner then we should create banner.txt inside the resources folder whatever we want.
* To compile jsps when we are using spring boot below tomcat-embed-jasper dependency has to be included.

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency> |

* We should create webapp folder and place all the jsps inside that folder.
* Below properties has to be defined inside the application.properties.
* spring.mvc.view.prefix=/views/
* spring.mvc.view.suffix=.jsp

**@PathVariable:**

* The above annotation maps the variable value in the URI to the Request Mapping Method as (@PathVariable String id).
* In the Request URI we have to accept the variable values as @GetMapping{“/topics/{id}”}.
* Check your Spring Boot Application for More Information.
* The above annotation is commonly used for GET requests which can be handled by the @GetMapping annotation.

**@RequestBody:**

* The above annotation will be used in the Request Handler Method for converting the JSON data to the instance of a class. This is used popularly for the POST requests which can be handled with the @PostMapping annotation.

**BELOW ARE NEW ANNOTATIONS IN SPRING 5:**

@**PostMapping:**

* Used to Map POST requests.

**@GetMapping:**

* Used to Map GET requests.

**@PutMapping:**

* Used to Map PUT requests.

**@DeleteMapping:**

* Used to Map Delete requests.

**BELOW ARE MULTIPLE WAYS TO CREATE A SPRING BOOT APPLICATION:**

* We can choose any one of the ways to actually create the Spring Boot Application and configure the same.
* **Below are different ways:**



**1)** First Way to create Spring Boot Application:

* We can create a simple Maven Project and we can add the Spring Boot Starter Parent and dependencies which we need in the pom.xml and we can mention the java version which our application can use.
* Here we need to create the Spring Boot Starter class by ourselves and has to annotate with the @SpringBootApplication. And we have to call SpringApplication.run() inside the main() of this class.

2) Second Way to create Spring Boot Application:

* Go to <http://start.spring.io/> and we can mention our requirements there and we can generate the readymade Spring Boot Application with all the above requirements met automatically without making any manual changes.
* We can add all the dependencies at one shot for whatever the requirements we need like web related, AOP, Security, JPA etc…

3) Third Way is to create with CLI(Command Line Interface):

* We can create Spring Boot Application with the Groovy Script. Please refer the below urls for more information about installation and execution.
* <https://www.youtube.com/watch?v=bNqD5P-huJc&index=23&list=PLmbC-xnvykcghOSOJ1ZF6ja3aOgZAgaMO>
* <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started-installing-spring-boot.html>

4) Using Spring Tool Suite: New -> Spring Starter Project.

* This is very handy way to create the spring boot application. Download the Spring Tool Suite software from spring.io or install the plugins for the eclipse.
* New -> Spring Starter Project ->Select the dependencies you need and generate the project.

**APPLICATION.PROPERTIES:**

* Application.properties file is very helpful if we want to change the default configuration of what Spring Provides.
* Spring boot automatically scans the application.properties file which is there in the class path.

**JAVA PERSISTANCE API (JPA):**

* JPA is actually a standard or specialization where we provide the Entity Classes i.e. what are the fields we need to persist and what is Primary Key and what is Foreign Key in the Java Classes and give it one ORM framework which does rest of the activities like creating tables, writing queries and mapping the data from classes-> tables and vice versa.

**SPRING DATA JPA:**

* It provides lot of simpler ways to do CRUD operations.
* Spring boot provides the in-built Database or embedded database for development purpose if we choose some dependency like apache derby database when we are actually generating the spring boot application.
* To implement the spring data JPA we need to spring boot starter jpa in our pom.xml which in-turn brings the dependencies of the spring jpa, hibernate and orm dependencies.
* Spring data JPA provides an inbuilt interface named CrudRepository where we can get the default crud operations implemented by the Spring Data JPA.
* Our Repository interface has to just extend the CrudRepository inorder get the implementations for default CRUD operations.
* **We don’t need to provide any database information, if we add derby database to our class path then when we start our spring boot application, derby database also gets started and we don’t need to provide any connection information or any other stuff in our application. If we connect to the external database then we need to provide that database information.**
* We have lot of predefined methods in the CrudRepository interface with which we make our CRUD operations very easy like save(), findAll() etc..

**CRUDREPOSITORY INTERFACE:**

* CrudRepository interface provides all the methods by default for doing the crud operations like insert, update, delete, select.
* All we have to do is Create our own interface and extend the CrudRepository interface and Autowired wherever you need this implementation.
* Below are the important methods and its implementation:

|  |  |
| --- | --- |
| **Method Name** | **Description** |
| findAll() | Retrieves all the records from the DataBase |
| findById() | Retrieves the record based on the id |
| save() | Inserts a new record into the database if it does not exist. If present will update the record. |
| deleteById() | Deletes the records based on the Id from the DataBase. |

* We can write our custom methods that gives hint to the Spring Data JPA inside the interface where we extend the CrudRepository interface.
* The syntax of the new methods is as follows findBy{PropertyName}() or findBy{Object}{Object Property}().
* We can have the ManyToOne, OneToMany relationships also established in the Spring Data JPA.
* **Please refer 30 and 31 videos for more information in the Java Brains Videos.**

**Running and Packaging Spring boot App in Command Prompt:**

* We can execute the jar file after building in the Command prompt with command mvn clean install.
* And the jar can be run as java –jar target/jarname.
* The above command will deploy our application as jar in the tomcat container.
* The same way we can deploy the war as well in the tomcat container.

**Spring Boot Actuator:**

* Spring boot actuator helps us to manage and monitor the production application and can do lot of whole things. We can monitor the health of the application with /health end point.
* Please check it’s not working in Spring boot 2.0.0 version in your project.

**SPRING DATA JPA:**

* It provides lot of inbuilt repositories to us which we can use and reduce the lot of broiler plate code every time we write when we access the data layer.
* The abstract repositories provided by the spring are implemented at the run time by the spring container and does the CRUD operations.
* There are three base interfaces defined in the Spring data commons Project.

**1) REPOSITORY:**

* This is a marker interface and does not have any methods. It is the central repository interface in the spring data.
* If we are implementing this interface then we have to provide our own methods.
* For this interface also we need to pass to parameters one is Entity class name and other is Entity class id name.
* It is the super interface for the CrudRepository Interface.

**2) CRUDREPOSITORY:**

* This repository provides the methods for the basic CRUD operations. This interface also needs the two parameters as input one is Entity class name and Entity class id. Below are all the methods available in the CrudRepository Interface.
* It is sub interface of the Central Repository interface.
* Below are the methods in the CrudRepository interface:
  + count() –returns the number of entities available in the database.
  + delete(T entity)- deletes the given entity.
  + deleteAll() – deletes all the entities managed by this repository.
  + deleteAll(Iterable<? extends T> entities) – deletes all the given entities.
  + deleteById(ID id) –deletes the record with the specified entity.
  + existsById(ID id)—returns boolean if the specified id exists in database.
  + findAll() – returns all the entities of particular type.
  + findAllById(Iterable<ID> ids) – returns all the instances with the particular ids.
  + findById(ID id) – returns the instance of the particular id.
  + save(Entity t) – saves the specified instance into the database.
  + saveAll(Iterable<? extends T> entities) –Saves all the specified entities into the database.

**JPAREPOSITORY:**

* If our applications need any JPA specific implementations then only our repository has to extend the JpaRepository. Otherwise it is highly recommended to go with the Crud Repository.
* This repository also needs the two parameters one is Entity class name and Entity class id.
* JpaRepository Extends CrudRepositoy, PagingAndSortingRepository, Repository, QueryByExampleExecutor interfaces.
* Below are the methods specific in the JpaRepository:
  + flush() –flushes all the pending changes to the database.
  + saveAndFlush(S entity) – saves the entity and flushes the changes immediately.
  + deleteInBatch(Iterable <T> entities)—deletes all the specified entities in a batch call i.e. by creating a single query.
  + deleteAllInBatch()--- deletes all entities in a single batch call.
  + getOne(ID id) --- returns the reference to the specified id.

**PAGINGANDSORTINGREPOSITORY:**

* This repository is the extension of the CrudRepository and extends CrudRepository interface.
* PagingAndSortingRepository is mainly for the purpose of the pagination and sorting operations.
* Below are the only two methods in the PagingAndSortingRepository:
  + findAll(Sort sort) ---returns all the entities based on the specified sorting mechanism.
  + findAll(Pageable pageable) ---returns the list of entities with the restricted number of entities per page based on the pageable object.
* Extending the CrudRepository and the JpaRepository by our repository enables spring to create some proxy object for our repository where we can use that proxy object anywhere in our application.
* **Depending on the requirement we need to go for extending the above three repositories.**
* **CrudRepository should be extended if you have only basic crud operations.**
* **JpaRepository has to be extended only if we have jpa specific operations in our application.**
* **PagingAndSortingRepository has to be extended only when we need the pagination and the data retrieval in the sorted order.**
* Repository, CrudRepository, PagingAndSortingRepository is implemented in the spring data commons.
* JpaRepository is implemented in the Spring Data JPA.
* **Three important steps in implementing the Spring Data JPA in our Project:**
  + Create a custom repository that extends one of the spring data jpa repository.
  + Add custom methods to the custom repository if required.
  + Use the proxy created by the spring container in the application for the spring to create the queries using the reflection.