Git hub project : SpringRest ()- We will be building a REST API for movie review posts. It allows one to search for a movie review, fetch all movies reviews, get a single movie review, create a new movie review, update, and delete an existing movie review.

Resource :

<https://medium.com/better-programming/building-a-spring-boot-rest-api-part-ii-7ff1e4384b0b>

**Notes :**

* Maven is a dependency manager.
* RestController is similar to servlet class- controls all the http requests from the user and returns an appropriate response.
* a singleton is a design pattern used in object-oriented programming to permit no more than one instance of a class.
* A singleton is implemented in Java by creating a static method that returns an instance of that class.
* Get carries request parameter in url whereas post carried in message body which makes it more secure.

**Project** :

**Day 1: (6th March 2020)**

**Java Classes**

**MainApplication**: needs annotation @SpringBootApplication which tells it’s a spring boot framework application which will run the run method to start the application. But we need controller to send and receive http request.

**BlogController**: RestController receives and send request to http. Using different annotations and url added

**BlogMockedData** : backend to add all the features of the application and is a single instance class.

**Step 1**: In the [previous tutorial](https://medium.com/@salisuwy/building-a-spring-boot-rest-api-a-php-developers-view-part-i-6add2e794646), we set up a [Spring Boot](https://spring.io/projects/spring-boot) application with a single dummy controller

**Step 2**: In the [previous tutorial](https://medium.com/@salisuwy/building-a-spring-boot-rest-api-part-ii-7ff1e4384b0b)**,** we’ve seen how we can send a request and get a response from the controller using hard-coded mocked data.

**Step 3:**

* Adding database: my SQL and java persistence API : https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html
* Install mySQL

<https://medium.com/employbl/how-to-install-mysql-on-mac-osx-5b266cfab3b6>

**MoviesRepository**

* While JPA provides us with the default CRUD operations (i.e. findAll, findOne, save, delete) out of the box, it is left to us to define our custom queries.
* We can add custom queries to jpa by using following

<returnType>**findBy<**Title>(<supply paramters>)

Eg.:

JPA: findByIdGreatherThan(20)   
SQL: WHERE id > 20  
-----------------------------------------------------------------JPA: findByIdGreatherThanAndTitleContaining(20, "google")  
SQL: WHERE id > 20 AND title LIKE "%google%"  
-----------------------------------------------------------------  
JPA: findByIdGreaterThanEqualOrTitle(5, "google api")  
SQL: WHERE id >= 5 OR title = "google api"

**Updated** **BlogController class**

The first change you may notice is that we are now using the BlogRepository and @Autowired annotation.

The method’s definition remains the same, however, the implementation changes.

You may notice that blogRepository has methods, such as findAll, findOne, save, delete, which we did not define in the BlogRepository.java.

These methods are provided by the JpaRepository we extended.

findAll() returns all the rows in the table. This is equivalent to:

SELECT \* FROM moviesList

findOne(**param**) returns one item that matches the primary key field with param.

SELECT \* FROM moviesList WHERE id=**param** LIMIT 1

save(**blog**) saves the entry into the database. This function will create a new record if a new blog item is supplied and updates an existing one if an existing blog item is supplied.

INSERT INTO blog(title, content) VALUES (**blog.title**, **blog.content**)

Or:

UPDATE blog SET title=**blog.title**, content=**blog.content** WHERE id=**blog.id**

delete(**param**) deletes an entry in the table given the provided id.

DELETE FROM blog WHERE id=**param**