

Spring Boot REST web service – Part 1 – Spring Data and MySQL

📅 February 28, 2017 (<https://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/>)
👤 Damith Ganegoda (<https://mydevgeek.com/author/ffft/>)
📁 IntelliJ (<https://mydevgeek.com/category/intellij/>), Java (<https://mydevgeek.com/category/java/>), REST (<https://mydevgeek.com/category/rest/>), Spring (<https://mydevgeek.com/category/spring/>)



In this tutorial series, we are going to discuss developing a REST web service using spring-boot. There is a tutorial for Simple REST Service using spring-boot (<http://mydevgeek.com/building-a-simple-rest-service-with-spring-boot/>). We are going to continue that tutorial.

What we will need

- JDK 1.8
- Maven 3
- spring-boot 1.5.1
- MySQL database – (You can use your favorite one)

Create a database

Create a table that name is “user” using following scripts and insert test data.

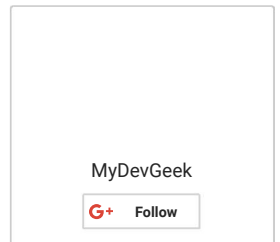
```
user.sql MySQL
1 DROP TABLE IF EXISTS `user`;
2
3 CREATE TABLE `user` (
4   `id` int(11) unsigned NOT NULL AUTO_INCREMENT,
5   `first_name` varchar(11) DEFAULT NULL,
6   `last_name` varchar(11) DEFAULT NULL,
7   `username` varchar(11) DEFAULT NULL,
8   `created` datetime DEFAULT NULL,
9   `updated` datetime DEFAULT NULL,
10  PRIMARY KEY (`id`)
11  ) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;
12
13 LOCK TABLES `user` WRITE;
14
15 INSERT INTO `user` (`id`, `first_name`, `last_name`, `username`, `created`, `updated`)
16 VALUES
17 (1, 'DAMITH', 'GANEGODA', 'test', '2017-02-22 00:00:00', '2017-02-22 00:00:00');
```

Add Spring-boot and MySQL connector dependency

```
1 <dependency>
2   <groupId>mysql</groupId>
3   <artifactId>mysql-connector-java</artifactId>
4   <version>6.0.5</version>
5 </dependency>
```

Search... 🔍

MYDEVGEEK
([HTTPS://WWW.FACEBOOK.COM/MYDEVGEEK-391403891211607/](https://www.facebook.com/MYDEVGEEK-391403891211607/))



RECENT POSTS

Linked List – Insert a node at a specific position
(<https://mydevgeek.com/linked-list-insert-a-node-at-a-specific-position/>)

Linked List – Insert a node at the head
(<https://mydevgeek.com/linked-list-insert-a-node-at-the-head/>)

Angular 4 CRUD application with Spring Boot REST service – Part 3
(<https://mydevgeek.com/angular-4-crud-application-with-spring-boot-rest-service-part-3/>)

Angular 4 CRUD application with Spring Boot REST service – Part 2
(<https://mydevgeek.com/angular-4-crud-application-with-spring-boot-rest-service-part-2/>)

Java List of Objects to JSON – Jackson 2
(<https://mydevgeek.com/java-list-json-jackson-2/>)

POPULAR POSTS



Spring Boot REST web s
Part 2 – CRUD operatio
Service Layers, Assemb
Utility classes
(<https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-utility-classes/>)

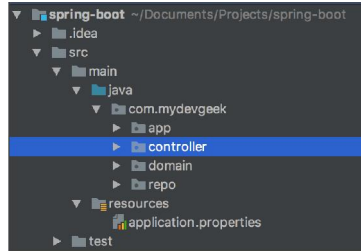
```

1 <dependency>
2 <groupId>org.springframework.boot</groupId>
3 <artifactId>spring-boot-starter-data-jpa</artifactId>
4 <version>1.5.1.RELEASE</version>
5 </dependency>

```

Database and Spring Configuration

Create a package structure and add the **application.properties** into the **resources** directory.



```

application.properties
1 spring.datasource.url=jdbc:mysql://localhost:3306/mydevgeek
2 spring.datasource.username=root
3 spring.datasource.password=
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
6 spring.jpa.hibernate.ddl-auto=update

```

Note :- use same property keys. The spring boot automatically configures it.

Note :- If you use `com.mysql.jdbc.Driver`, got this error

```

1 Loading class com.mysql.jdbc.Driver. This is deprecated. The new dri
2 is com.mysql.cj.jdbc.Driver. The driver is automatically registered
3 and manual loading of the driver class is generally unnecessary.

```

Use latest driver name (`com.mysql.cj.jdbc.Driver`) to fix this issue.

```

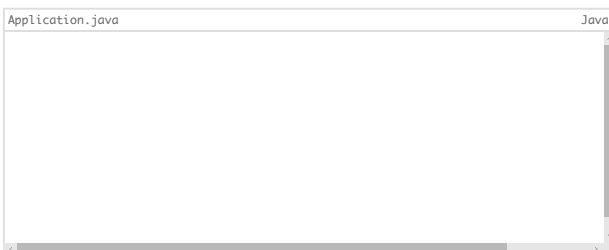
Application.java
1 package com.mydevgeek.app;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class Application {
8
9     public static void main(String[] args) {
10         SpringApplication.run(Application.class, args);
11     }
12 }

```

Now we are going to add the domain, repository and controller classes. Here's the tricky configuration.

@SpringBootApplication = This annotation automatically detects all the **@Entity**, **@Repository**, **@Controller**, **@RestController**, **@Service** and **@Component**. But all components must be in same or child package level. In this example, the application.java in separate the package so that we have to explicitly define the domain, repository and controllers.

here the example with explicitly defines packages.



06 Mar , 2017



Spring Boot REST web s
Part 1 – Spring Data ani
(<https://mydevgeek.com/boot-rest-web-service-fspring-data-mysql/>)
28 Feb , 2017

(<https://mydevgeek.com/4-crud-application-with-spring-boot-rest-service-part-1/>)
18 Sep , 2017



Building a simple REST
with Spring Boot
(<https://mydevgeek.com/a-simple-rest-service-wspring-boot/>)
14 Feb , 2017



Deploying a Spring Boo
Application on AWS usi
RDS
(<https://mydevgeek.com/spring-boot-application-on-aws-ec2-rds/>)
12 Apr , 2017

CATEGORIES

Angular 4
(<https://mydevgeek.com/category/angular-4/>) (3)

Apache Spark
(<https://mydevgeek.com/category/apache-spark/>) (1)

AWS
(<https://mydevgeek.com/category/aws/>) (1)

Data Structures
(<https://mydevgeek.com/category/data-structures/>) (6)

DesignPattern
(<https://mydevgeek.com/category/designpattern/>) (2)

Docker
(<https://mydevgeek.com/category/docker/>) (1)

Dropwizard
(<https://mydevgeek.com/category/dropwizard/>) (1)

IntelliJ
(<https://mydevgeek.com/category/intellij/>) (4)

Java
(<https://mydevgeek.com/category/java/>) (26)

```

1 package com.mydevgeek.app;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.boot.autoconfigure.domain.EntityScan;
6 import org.springframework.context.annotation.ComponentScan;
7 import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
8
9 @SpringBootApplication
10 @ComponentScan(basePackages = "com.mydevgeek")
11 @EnableJpaRepositories(basePackages = "com.mydevgeek.repo")
12 @EntityScan(basePackages = "com.mydevgeek.domain")
13 public class Application {
14
15     public static void main(String[] args) {
16         SpringApplication.run(Application.class, args);
17     }
18 }

```

`@ComponentScan` – use for scanning the components. In this example, controllers and services can not be identified, if we don't define the component scan path.

`@EnableJpaRepositories` – use for identified the repositories.

`@EntityScan` – use for identified entities.

Entity & Repository

Now, we are ready to create an entity class and repository.

Create an **User.java** in **com.mydevgeek.domain**.

```

User.java Java
1 package com.mydevgeek.domain;
2
3 import org.hibernate.annotations.CreationTimestamp;
4 import org.hibernate.annotations.UpdateTimestamp;
5
6 import javax.persistence.*;
7 import java.io.Serializable;
8 import java.sql.Date;
9
10 @Entity
11 @Table(name = "user")
12 public class User implements Serializable {
13
14     @Id
15     @GeneratedValue(strategy = GenerationType.AUTO)
16     @Column(name = "id")
17     private Long id;
18
19     @Column(name = "first_name")
20     private String firstName;
21
22     @Column(name = "last_name")
23     private String lastName;
24
25     @Column(name = "username")
26     private String username;
27
28     @Column(name = "created")
29     @CreationTimestamp
30     private Date created;
31
32     @Column(name = "updated")
33     @UpdateTimestamp
34     private Date updated;
35
36     //getters and setters
37
38 }

```

Then create a **UserRepository.java** in **com.mydevgeek.repo**

```

UserRepository.java Java
1 package com.mydevgeek.repo;
2
3 import com.mydevgeek.domain.User;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 @Repository
8 public interface UserRepository extends JpaRepository<User, Long> {
9 }

```

Spring data supports following interfaces.

Java 8
(<https://mydevgeek.com/category/java-8/>) (4)

Java Core
(<https://mydevgeek.com/category/java-core/>) (2)

Maven
(<https://mydevgeek.com/category/maven/>) (3)

REST
(<https://mydevgeek.com/category/rest/>) (10)

Spring
(<https://mydevgeek.com/category/spring/>) (11)

SQL
(<https://mydevgeek.com/category/sql/>) (1)

ARCHIVES

October 2017
(<https://mydevgeek.com/2017/10/>) (2)

September 2017
(<https://mydevgeek.com/2017/09/>) (4)

June 2017
(<https://mydevgeek.com/2017/06/>) (1)

May 2017
(<https://mydevgeek.com/2017/05/>) (2)

April 2017
(<https://mydevgeek.com/2017/04/>) (3)

March 2017
(<https://mydevgeek.com/2017/03/>) (8)

February 2017
(<https://mydevgeek.com/2017/02/>) (12)

`CrudRepository` (<http://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>) - provides basic crud functionalities.

JpaRepository (<http://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>) – provides JPA related functionalities such as persist, flush, batch delete, etc.

PagingAndSortingRepository (<http://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/PagingAndSortingRepository.html>) – provides additional functionalities such as paging and sorting.

You can use `CrudRepository` (<http://docs.spring.io/spring-data/data-commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>) Or `JpaRepository` (<http://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html>) for this example.

Controller

Create a new java class. It's called **UserController.java** in **com.mydevgeek.controller**.

```
UserController.java
```

```
1 package com.mydevgeek.controller;
2
3 import com.mydevgeek.domain.User;
4 import com.mydevgeek.repo.UserRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.HttpStatus;
7 import org.springframework.web.bind.annotation.*;
8
9 @RestController
10 @RequestMapping("/user")
11 public class UserController {
12
13     @Autowired
14     private UserRepository userRepository;
15
16     @RequestMapping(value = "/{id}", method = RequestMethod.GET)
17     @ResponseStatus(HttpStatus.OK)
18     public User getUser(@PathVariable("id") Long id) {
19         return userRepository.findOne(id);
20     }
21 }
```

At this moment our web service supports only get user details that we already inserted.

How to run it

First of all, make sure database is up and run. Then we can run our service via command line or IntelliJ.

- **Command Line** – go to the project directory. Run “**mvn spring-boot:run**”
- **IntelliJ** – Build the project. Then, right click on the **Application.java** class and select the “**Run**”.

[illegible]

If you clearly look at the console. We can see, there is an API for `"/user/{id}"`. This is a good way to identify whether rest API running or not.

How to test it

Send **GET** request through the POSTMAN

(<https://chrome.google.com/webstore/detail/postman/fhbjgibflinjbdbggehccddcbncdddomop?hl=en>).

http://localhost:8080/

X

+

No Environment

GET

http://localhost:8080/user/1

Params

Send

Response

```
1 | {"id":1,"firstName":"DAMITH","lastName":"GANEGODA","username":"test"}
```

Project code :- **GitHub** (<https://github.com/damithme/spring-boot-REST/tree/master/spring-boot-part1>)

Next tutorial (<http://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes/>) will be how to implement CRUD operations and some best practices.



(<http://www.facebook.com/sharer.php?u=https://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/&t=Spring+Boot+REST+web+service+%26%238211%3B+Part+1+%26%238211%3B+Spring+Data+and+MySQL>)



(<http://twitter.com/share?text=Spring+Boot+REST+web+service+%26%238211%3B+Part+1+%26%238211%3B+Spring+Data+and+MySQL-&data-mysql/&via=MyDevGeek>)



(https://plusone.google.com/_/+1/confirm?hl=fr-FR&url=https://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/)



(<http://www.tumblr.com/share/photo?source=https%3A%2F%2Fi0.wp.com%2Fmydevgeek.com%2Fwp-content%2Fuploads%2F2017%2F02%2Flogo.png%3Ffit%3D300%252C300%26ssl%3D1&caption=Spring+Boot+REST+web+service+%26%238211%3B+Part+1+%26%238211%3B+Spring+boot-rest-web-service-part-1-spring-data-mysql%2F>)



(<http://www.linkedin.com/shareArticle?mini=true&url=https://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/&title=Spring+Boot+REST+web+service+%26%238211%3B+Part+1+%26%238211%3B+Spring+Data+and+MySQL&source=MyDevGeek>)



(<https://www.blogger.com/blog-this.g?u=https://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/&n=Spring+Boot+REST+web+service+%26%238211%3B+Part+1+%26%238211%3B+Spring+Data+and+MySQL&t=MyDevGeek>)

Related Post

Angular 4 CRUD application with Spring Boot REST S...

How to use Spring RestTemplate (<https://mydevgeek.com/use-spring-rest-template/>)

Building a simple REST Service with Spring Boot (<https://mydevgeek.com/building-a-simple-rest-service-with-spring-boot/>)

Spring 4.3 – Event Listener (<https://mydevgeek.com/spring-4-3-event-listener/>)



10363 total visits, 43 visits today

[Intellij \(https://mydevgeek.com/tag/intellij/\)](https://mydevgeek.com/tag/intellij/) [java8 \(https://mydevgeek.com/tag/java8/\)](https://mydevgeek.com/tag/java8/)

[rest \(https://mydevgeek.com/tag/rest/\)](https://mydevgeek.com/tag/rest/) [spring \(https://mydevgeek.com/tag/spring/\)](https://mydevgeek.com/tag/spring/)

[spring-boot \(https://mydevgeek.com/tag/spring-boot/\)](https://mydevgeek.com/tag/spring-boot/)



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

**Sarvar Nishonboyev** • 22 days ago

As Java Doc says: "Servlet container will typically write an HTML error page therefore making the use of a reason unsuitable for REST APIs. For such cases it is preferable to use a ResponseEntity as a return type and avoid the use of @ResponseStatus altogether"

<https://docs.spring.io/spring...>

Why don't you just use a ResponseEntity instead of returning User type and not to use @ResponseStatus?

^ | v • Reply • Share

**pravesh shrestha** • 22 days ago

you have to add hibernate core dependency in pom.xml too else intellij will show error thanks for the tutorial

It is easy to understand and implement.

^ | v • Reply • Share

**Fidel Lascano** • 2 months ago

amigo esto no me funciona lo seguí

paso a paso y no me reconoce la ruta

puedes decirme si hay alguna configuración adicional

^ | v • Reply • Share

**Fher Andrade**  Fidel Lascano • 9 days ago

Hola amigo puede ser que @ComponentScan no encuentre el componente Controller.

^ | v • Reply • Share

**Albin Hasani** • a year ago

content is great. congrats! but you could fix the social media sharing links as they have formatting issues. cheers

^ | v • Reply • Share

**mydevgeek** Mod  Albin Hasani • a year ago

Thanks, I'll do. cheers!!

^ | v • Reply • Share

[◀ Google Guava Cache for caching \(https://mydevgeek.com/google-guava-cache-caching/\)](https://mydevgeek.com/google-guava-cache-caching/)
[ArrayList Sorting :- Part 1 – using Comparable and Comparator ▶ \(https://mydevgeek.com/arraylist-sorting-part-1-using-comparable-comparator/\)](https://mydevgeek.com/arraylist-sorting-part-1-using-comparable-comparator/)

PAGES

[Contact \(https://mydevgeek.com/contact/\)](https://mydevgeek.com/contact/)

SUBSCRIBE TO BLOG VIA EMAIL

Enter your email address to subscribe to MyDevGeek blog and receive notifications of new posts by email.

Join 58 other subscribers

MyDevGeek.com (<http://mydevgeek.com>) All rights reserved.

Email Address

SUBSCRIBE