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/fffpt/) 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

```
DROP TABLE IF EXISTS `user`;

CREATE TABLE `user` (
`id` int(11) unsigned NOT NULL AUTO_INCREMENT,
`first_name` varchar(11) DEFAULT NULL,
`last_name` varchar(11) DEFAULT NULL,
`username` varchar(11) DEFAULT NULL,
`created` datetime DEFAULT NULL,
`updated` datetime DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=latin1;

LOCK TABLES `user` WRITE;

INSERT INTO `user` (`id`, `first_name`, `last_name`, `username`, `c
VALUES
(1,'DAMITH','GANEGODA','test','2017-02-22 00:00:00','2017-02-22 00:
```

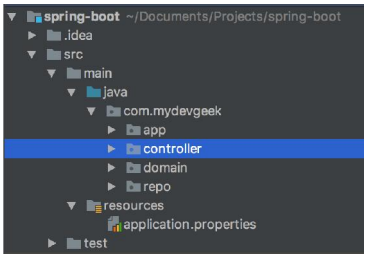

Add Spring-boot and MySQL connector dependency

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>6.0.5</version>
</dependency>
```

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>1.5.1.RELEASE</version>
</dependency>
```

Database and Spring Configuration

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



application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/mydevgeek
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL5Dialect
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

```
Loading class com.mysql.jdbc.Driver. This is deprecated. The new dri
is com.mysql.cj.jdbc.Driver. The driver is automatically registered
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 Java

```java
package com.mydevgeek.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

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.

Application.java Java

```java
```

06 Mar , 2017

Spring Boot REST web s
Part 1 – Spring Data an
(https://mydevgeek.cor
boot-rest-web-service-p
spring-data-mysql/)
28 Feb , 2017

(https://mydevge
Angular 4 CRUD applica
Spring Boot REST servic
(https://mydevgeek.cor
4-crud-application-with
boot-rest-service-part-1
18 Sep , 2017

Building a simple REST
with Spring Boot
(https://mydevgeek.cor
a-simple-rest-service-w
boot/)
14 Feb , 2017

Deploying a Spring Boo
Application on AWS usi
RDS
(https://mydevgeek.cor
spring-boot-application
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)

```java
package com.mydevgeek.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepo

@SpringBootApplication
@ComponentScan(basePackages = "com.mydevgeek")
@EnableJpaRepositories(basePackages = "com.mydevgeek.repo")
@EntityScan(basePackages = "com.mydevgeek.domain")
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

`@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

```java
package com.mydevgeek.domain;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import javax.persistence.*;
import java.io.Serializable;
import java.sql.Date;

@Entity
@Table(name = "user")
public class User implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id")
    private Long id;

    @Column(name = "first_name")
    private String firstName;

    @Column(name = "last_name")
    private String lastName;

    @Column(name = "username")
    private String username;

    @Column(name = "created")
    @CreationTimestamp
    private Date created;

    @Column(name = "updated")
    @UpdateTimestamp
    private Date updated;

    //getters and setters

}
```

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

UserRepository.java Java

```java
package com.mydevgeek.repo;

import com.mydevgeek.domain.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

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 — Java

```java
package com.mydevgeek.controller;

import com.mydevgeek.domain.User;
import com.mydevgeek.repo.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    @ResponseStatus(HttpStatus.OK)
    public User getUser(@PathVariable("id") Long id) {
        return userRepository.findOne(id);
    }
}
```

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

How to run it

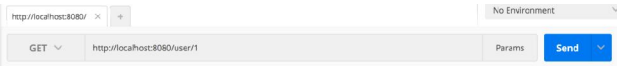
First of all, make sure database is up and run. Then we can run our service via command line or IntelliJ.

- **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**".

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/fhbjgbiflinjbdggehcddcbncdddomop?hl=en).



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.

Related Post

Angular 4 CRUD application with Spring Boot REST s...	How to use Spring RestTemplate (https://mydevgeek.com/use-	Building a simple REST Service with Spring Boot (https://mydevge	Spring 4.3 – Event Listener (https://mydevgeek.com/spring-4-

IntelliJ (https://mydevgeek.com/tag/intellij/) java8 (https://mydevgeek.com/tag/java8/) rest (https://mydevgeek.com/tag/rest/) spring (https://mydevgeek.com/tag/spring/) spring-boot (https://mydevgeek.com/tag/spring-boot/)

‹ Google Guava Cache for 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/)

PAGES

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

Email Address

SUBSCRIBE