Spring Boot REST web service – Part 2 – CRUD operations, Service Layers, Assemblers and Utility classes

March 6, 2017 (https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes/) Damith Ganegoda (https://mydevgeek.com/author/fffpt/) Java (https://mydevgeek.com/category/java/), REST (https://mydevgeek.com/category/rest/), Spring (https://mydevgeek.com/category/spring/)

We already implemented simple REST web service in the previous tutorial.

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

Spring Boot REST web service – Part 1 – Spring Data and MySQL (http://mydevgeek.com/spring-boot-rest-web-service-part-1-spring-data-mysql/)

In this tutorial, I am going to implement new functionalities on top of the previous tutorial. Mainly, focusing on these following areas.

- CRUD Operations
- Service Layers
- Assemblers & Value Objects (VO)
- Utility classes

CRUD Operations

CRUD stands for Create, Read, Update and Delete. Actually, these are basic functions in most of the applications. As you remember, we created the **User** table in the previous tutorial. Create a new user, update existing user, find a user by user-id and delete user are CRUD operations in the **User** table.

Service Layers

Do you remember the **UserRepository** interface in the previous tutorial? Basically, it handles all of the operations that related to the **User** table. When you building an application, you need to call no of database request to fulfill a task. Let say, you want to create an invoice. You need to get customer details, order details, item details, calculate a total, calculate a taxes, etc. According to the example, it has to call the CustomerRepository, OrderRepository, ItemRepository. Where do we place these business logics? in **Controller**. Yes, we can place in Controllers. But it might be increased the code complexity and difficult to maintain when growing requirements. So that we can introduce a **Service Layer**. It can be contained business logics.

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.cor
boot-rest-web-service-p
crud-operations-service
assemblers-utility-class

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class InvoicingService {

    @Autowired
    private CustomerRepository customerRepository;

    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private ItemRepository itemRepository;

    @Autowired
    private OrderService orderService;

    @Autowired
    private ItemAssembler itemAssembler;

    public Invoice createInvoice(Long customerId, Long orderId, Lis
        return Invoice;
    }
}
```

This is an example for an Invoicing Service. You can see, it is connected with few of repositories. It might be connected with another service. Let say, an application has 30 repositories. But it might have 4 or 5 service layers. Another example is, an UserService might be connected with UserRepository, OrganizationRepository and UserRoleRepository.

Now, We are going to implement the **UserService** for our application. Create a new package **com.mydevgeek.service** and create an **UserService.java** interface and **UserServiceImpl.java** class.

```java
package com.mydevgeek.service;

import com.mydevgeek.domain.User;

public interface UserService {

    User getUserById(Long id);

    User createUser(User user);

    User updateUser(User user);

    void deleteUser(Long id);

}
```

```java
package com.mydevgeek.service;

import com.mydevgeek.domain.User;
import com.mydevgeek.repo.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserRepository userRepository;

    public User getUserById(Long id) {
        return userRepository.findOne(id);
    }

    public User createUser(User user) {
        return userRepository.save(user);
    }

    public User updateUser(User user) {
        return userRepository.save(user);
    }

    public void deleteUser(Long id) {
        User user = getUserById(id);
        if (user != null) {
            userRepository.delete(id);
        }
    }
}
```

Assemblers and Value Objects (VO)

The Assemblers are used to convert one model to another model. For an example, get the **User** model. It has no of properties such as first name, last name, created date & time, updated date and time, etc. Give all information

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
ak.com/

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)

about the user to outside the world, is not a good practice. So that, we have to modify the **User** model to another model. Basically, the idea behind the concept is outside world should not be known about internal models.

Assemblers – we can call it "**Transformers**" too. Anyway, there is no any naming convention. You can use **"Assemblers"**, **"Transformers"** or any suitable name. Their responsibility is some model is converted to another.

Value Objects – Actually, it is simple POJO class (properties and their getters and setters). According to this tutorial, we already have a **User** class. But we can not pass the same **User** class to the outside. So that we will have to create another class. We can name it like this **UserVO**, **CreateUserVO**, **UpdateVO**. Keep remember, there is no any naming convention. Or you can name it such as **CreateUserRequest**, **ViewUserResponse**, etc.

Now, we'll create User VOs and the User Assembler class.

We need to 3 User VOs

- **CreateUserVO –** use for getting user info when creating it.
- **UpdateUserVO –** use for getting user info when updating it.
- **UserVO –** use for returning user info.

Create a package **com.mydevgeek.vo**.

CreateUserVO.java Java

```java
package com.mydevgeek.vo;

public class CreateUserVO {
    private String firstName;
    private String lastName;
    private String username;
    //getters and setters
}
```

```java
package com.mydevgeek.vo;

public class UpdateUserVO {
    private Long userId;
    private String firstName;
    private String lastName;
    private String username;
    //getters and setters
}
```

```java
package com.mydevgeek.vo;

public class UserVO {
    private Long userId;
    private String fullName;
    private String username;
    //getters and setters
}
```

Then create another package **com.mydevgeek.assemblers**.

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)

```java
package com.mydevgeek.assemblers;

import com.mydevgeek.domain.User;
import com.mydevgeek.vo.CreateUserVO;
import com.mydevgeek.vo.UpdateUserVO;
import com.mydevgeek.vo.UserVO;
import org.springframework.stereotype.Component;

@Component
public class UserAssembler {

    /**
     * CreateUserVO convert to User.
     *
     * @param createUserVO
     * @return
     */
    public User toUser(CreateUserVO createUserVO) {
        User user = new User();
        user.setFirstName(createUserVO.getFirstName());
        user.setLastName(createUserVO.getLastName());
        user.setUsername(createUserVO.getUsername());
        return user;
    }


    /**
     * User to UserVO.
     *
     * @param user
     * @return
     */
    public UserVO toUserVO(User user) {
        UserVO userVO = new UserVO();
        userVO.setUserId(user.getId());
        userVO.setFullName(UserUtil.convertToFullName(user.getFirst
        userVO.setUsername(user.getUsername());
        return userVO;

    }

    /**
     * UpdateUserVO to user.
     *
     * @param updateUserVO
     * @return
     */
    public User toUser(UpdateUserVO updateUserVO) {
        User user = new User();
        user.setId(updateUserVO.getUserId());
        user.setFirstName(updateUserVO.getFirstName());
        user.setLastName(updateUserVO.getLastName());
        user.setUsername(updateUserVO.getUsername());
        return user;
    }
}
```

`@Component` -(Spring Documentation (http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/beans.html#beans-stereotype-annotations)) This is a general-purpose stereotype annotation indicating that the class is spring component. This will be created a Singleton instance. That's suitable for this task as well.

If you want to create a new class instance each time one is needed, use this annotation with `@Component` .

```java
@Component
@Scope("prototype")
```

I think, now you have an idea about the responsibility of **Assembler** classes.

Another important thing is, in **Assembler,** we can call another **Services** or **Repositories** like what we are doing in **Controllers**.

Utility Classes

A utility class is a class that uses for define common and reusable methods.

Create a package **com.mydevgeek.util** and create a utility class that's name is **UserUtil.java**.

```java
package com.mydevgeek.util;

public class UserUtil {

    private UserUtil() {

    }

    public static String convertToFullName(String firstName, String
        return firstName + " " + lastName;
    }
}
```

- Create ***private*** constructor to avoid create multiple instance.
- All methods must be ***static***.
- Should not call other **Services**, **Assemblers**, **Controllers**, **Repositories**.

Controllers

Now we're going to create a controller class that uses to connect with outside. Create a package ***com.mydevgeek.controller***. Create a ***UserController.java*** class.

```java
package com.mydevgeek.controller;

import com.mydevgeek.assemblers.UserAssembler;
import com.mydevgeek.domain.User;
import com.mydevgeek.repo.UserRepository;
import com.mydevgeek.service.UserService;
import com.mydevgeek.vo.CreateUserVO;
import com.mydevgeek.vo.UpdateUserVO;
import com.mydevgeek.vo.UserVO;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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

    @Autowired
    private UserAssembler userAssembler;

    @Autowired
    private UserService userService;

    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public UserVO getUser(@PathVariable("id") Long id) {
        return userAssembler.toUserVO(userService.getUserById(id));
    }

    @RequestMapping(method = RequestMethod.POST)
    public UserVO createUser(@RequestBody CreateUserVO userVO) {
        //convert to User
        User user = userAssembler.toUser(userVO);
        //save User
        User savedUser = userService.createUser(user);
        //convert to UserVO
        return userAssembler.toUserVO(savedUser);
    }

    @RequestMapping(method = RequestMethod.PUT)
    public UserVO updateUser(@RequestBody UpdateUserVO updateUserVO
        //convert to User
        User user = userAssembler.toUser(updateUserVO);
        //update User
        User updatedUser = userService.updateUser(user);
        //convert to UserVO
        return userAssembler.toUserVO(user);
    }

    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    public void delete(@PathVariable("id") Long id) {
        userService.deleteUser(id);
    }
}
```

Best practices

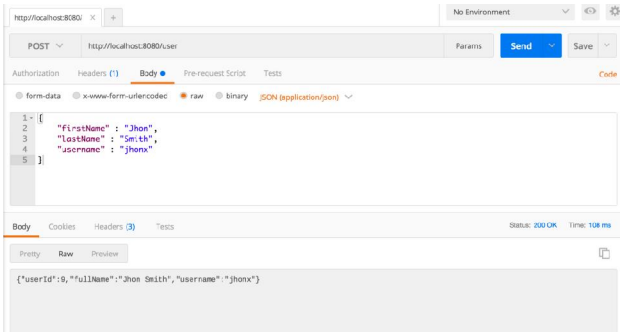
- Use ***RequestMethod.GET*** to get some data.
- Use ***RequestMethod.POST*** to create new data.
- Use ***RequestMethod.PUT*** to modify data.
- Use ***RequestMethod.DELETE*** to remove data.
- Finally, as a practice, don't directly call with the **Repository**. Every time call through the Service Layer.

Run and Test it

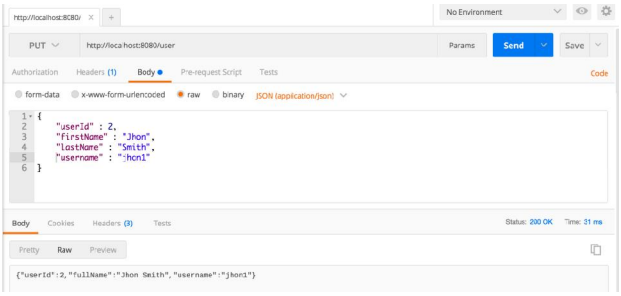
As mentioned in Part-1 (http://mydevgeek.com/spring-boot-rest-web-service-

part-1-spring-data-mysql/), you can run using command line or intellij.

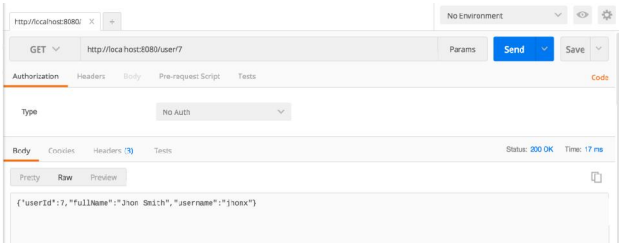
Create a User using POSTMAN.



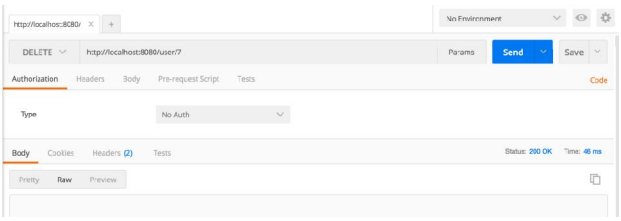
Modify a User



Get a User by ID



Delete a User



Project Code :- GitHub (https://github.com/damithme/spring-boot-REST/tree/master/spring-boot-part2)

Next tutorial will be Validation and Exception Handling.

(http://www.facebook.com/sharer.php?u=https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-asse
classes/&t=Spring+Boot+REST+web+service+%26%238211%3B+Part+2+%26%238211%3B+CRUD+operations%2C+Service+Layers%2C+Assembl

(http://twitter.com/share?text=Spring+Boot+REST+web+service+%26%238211%3B+Part+2+%26%238211%3B+CRUD+operations%2C+Serv
operations-service-layers-assemblers-utility-classes/&via=MyDevGeek)

(https://plusone.google.com/_/+1/confirm?hl=fr-FR&url=https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-servi

(http://www.tumblr.com/share/photo?source=https%3A%2F%2Fi0.wp.com%2Fmydevgeek.com%2Fwp-content%2Fuploads%2F2017%2F02
logo.png%3Ffit%3D300%252C300%26ssl%3D1&caption=Spring+Boot+REST+web+service+%26%238211%3B+Part+2+%26%238211%3B+CRUD+c
boot-rest-web-service-part-2-crud-operations-service-layers-assemblers-utility-classes%2F)

(http://www.linkedin.com/shareArticle?mini=true&url=https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service
classes/&title=Spring+Boot+REST+web+service+%26%238211%3B+Part+2+%26%238211%3B+CRUD+operations%2C+Service+Layers%2C+Assem

(https://www.blogger.com/blog-this.g?u=https://mydevgeek.com/spring-boot-rest-web-service-part-2-crud-operations-service-layers-assem
classes/&n=Spring+Boot+REST+web+service+%26%238211%3B+Part+2+%26%238211%3B+CRUD+operations%2C+Service+Layers%2C+Assembl

9 Comments mydevgeek.com Login

Recommend 1 Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Jorge Reyes-Spindola • 10 months ago
Thank you very much for a very enlightening tutorial. It helped me quite a bit.
1 • Reply • Share ›

mydevgeek Mod → Jorge Reyes-Spindola • 10 months ago
Thanks!!
• Reply • Share ›

Jayakumar Jayaraman • a month ago
Nice tutorial. Can we have the Data / Service / Controller in a separate Java projects? If so, can they all be a spring boot project? Will there be any issues in dependencies, if we have all these in separate projects? Or is it fine to have all these 3 layers in one java spring boot project, in separate packages? I think all in one project may be simple, but is it good option? Thanks again.
• Reply • Share ›

Francisco Amaro • a month ago
Thank you very much, you help me a lot but i have a question:
How can use the assembler with a List<user> from a findAll method?
• Reply • Share ›

Oleksandr Mandryk • 5 months ago
Hi Damith, thank you for sharing great content :)

Few questions:

1. Are there any benefits of making assembler as Spring component over simple utility class with static methods?

2. Shouldn't assembler be injected and used in service? For example if controller needs to return more complex object that consists from several entities...

Thanks!
• Reply • Share ›

akhmadGuntar → Oleksandr Mandryk • 3 months ago
Yes, this is a great content. The series are great, short yet profound. and this concept of assembler and value object is very interesting!
• Reply • Share ›

mydevgeek Mod → Oleksandr Mandryk • 4 months ago
Hi, Thanks your comment.
1. Main advantage is, you can easily mock it. Normally, we're using utility methods as a part of complex method. If we use it as a Spring component, we will be able to mock and test the complex method.
2. Yes, definitely you can use it. There is no defined way to do that. you can use it in service, controller or both.
• Reply • Share ›

eli • 6 months ago
WOW. Among the best tutorials i have ever read. It really shed the much needed light i needed.
I had a question. How can i send data from a form in a web application(made with spring boot), to an api, then the database.
Thanks alot.
Keep them comming.
Eli
• Reply • Share ›

mydevgeek Mod → eli • 4 months ago

ArrayList Sorting :- Part 1 – using Comparable and Comparator (https://mydevgeek.com/arraylist-sorting-part-1-using-comparable-comparator/)

Spring Boot REST web service – Part 3 – Exception Handling and Validation using @ControllerAdvice, @Valid and Custom Annotations (https://mydevgeek.com/spring-boot-rest-web-service-part-3-exception-handling-validation-using-controlleradvice-valid-custom-annotations/)

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