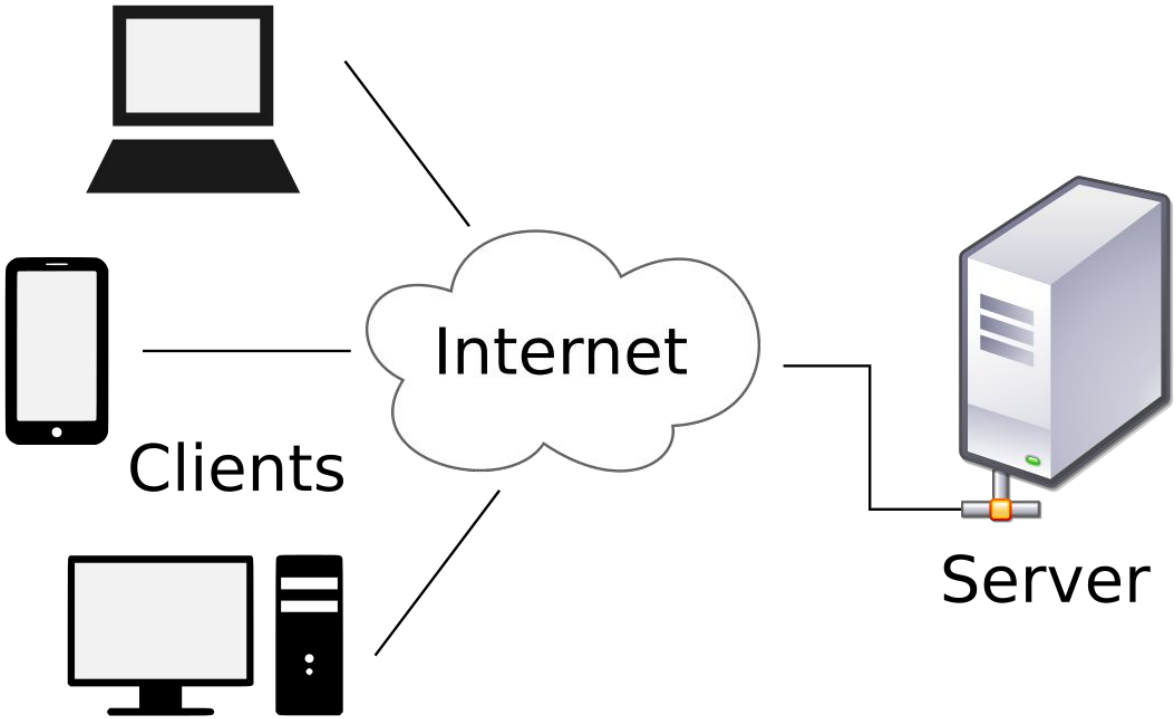
Spring Framework

What is Server/client Architecture

- Client-server architecture is a distributed application structure that many clients request and receive date from a centralized server
- Client is a application that send request to and receive response from server
- Server is a application that get request from and send response to client

What is Server/client Architecture



What is Framework

- SW Framework is like a half-finished product for SW
- Inversion of control: Framework has the overall program's flow of control
- Extensibility: A user can add user code to the framework for extending functionalities
- Non-modifiable framework code: The framework code, in general, is not supposed to be modified.

Differences between Framework and Library

- For Library, your codes call Library. But for Framework, Framework calls your code

- For Library, your codes control entire application. But for Framework, Framework controls

- For Library, you don't need to know about inside of Library. But for Framework, you need to know how the framework works and all the concepts inside it

Why should we use Framework?

- We can make everything without frameworks
- But by using framework, we can get MANY benefits from frameworks
- We can use very well tested and peer reviewed , validated code from thousands of SW engineers

What is Web Framework

- Framework for web development
- has MANY functionalities that is necessary when we building web application
  - security, database access, caching, web templating, URL mapping etc...

Varieties of web framework

- There are many types of web framework
- Spring for Java
- Django for Python
- Ruby on Rails for Ruby
- Laravel for PHP
- Functionalities of modern web frameworks are similar these days

Introduction of Spring Framework

- Spring Framework is an application framework for Java
- Spring Framework was first released in June 2003.
- Core functionalities are
  - Dependency Injection
  - Aspect Oriented Programming
- Now, Spring Projects provides almost everything for developing Java Enterprise Software

Advantage of Spring Framework

- High level of maturity, stability
- Unified, integrated framework environment(like Gift sets for Java developers)
- Large scale engineers communities
- Plenty of job opportunities(Korea, Japan, US etc.)

Disadvantage of Spring Framework

- Learning Curve
- It needs time to be familiar with

Projects of Spring (https://spring.io/projects)

- There are different types of Spring Project
- Spring Core for dependency injection and aspect oriented programming
- Spring Boot for quick and easy building of spring application
- Spring Data for accessing different types of databases
- Spring Security for authentication and authorization
- Spring Batch for batch processing
- etc...

Introduction of Spring Boot

- Spring Boot is a project built on the top of the Spring framework
- It provides a simpler and faster way to set up, configure, and run
  - Auto-configuration: providing pre-configured framework for specific purpose
  - Standalone: Way of running app all at once - packaging, downloading, install and configure WAS, deploy app on WAS
  - Opinionated: providing(enforcing) best practice pre-packaged structure by limiting the variety of choice

Topics we will study

- Spring Boot, Spring Web, Spring Data MySQL, Spring Data JPA, Spring Data Hibernate
- JSON, Maven, REST API, ORM
- Purpose is getting applicable level for simple Spring application

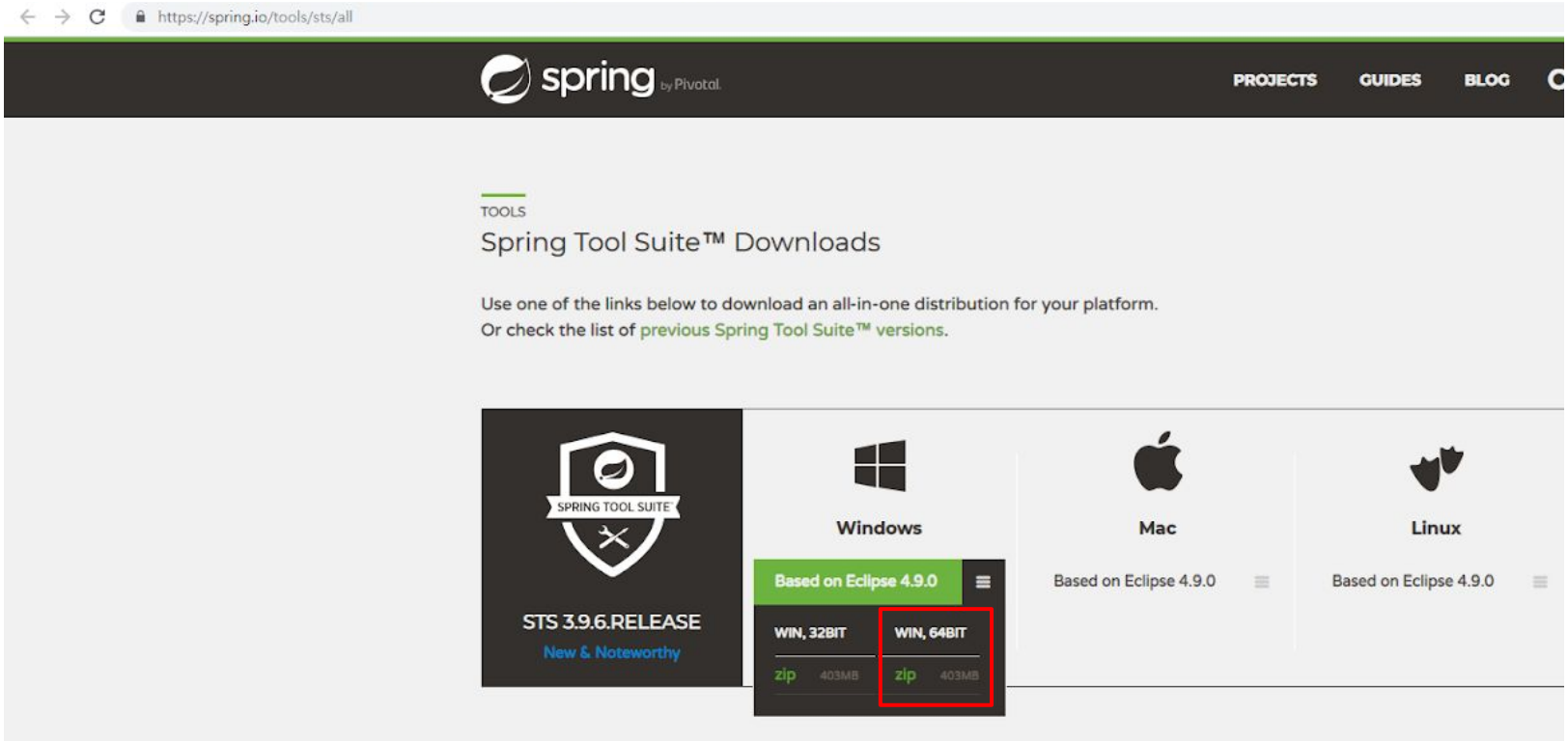
Topics we will skip(But important)

- Spring Security, Spring Batch, Spring Integration...
- Spring Mybatis, Spring Data for NoSQL, Cache ...
- Deep understanding of Spring

Download STS

- https://spring.io/tools/sts/all
- Download WIN 64 BIT
- Unzip the file

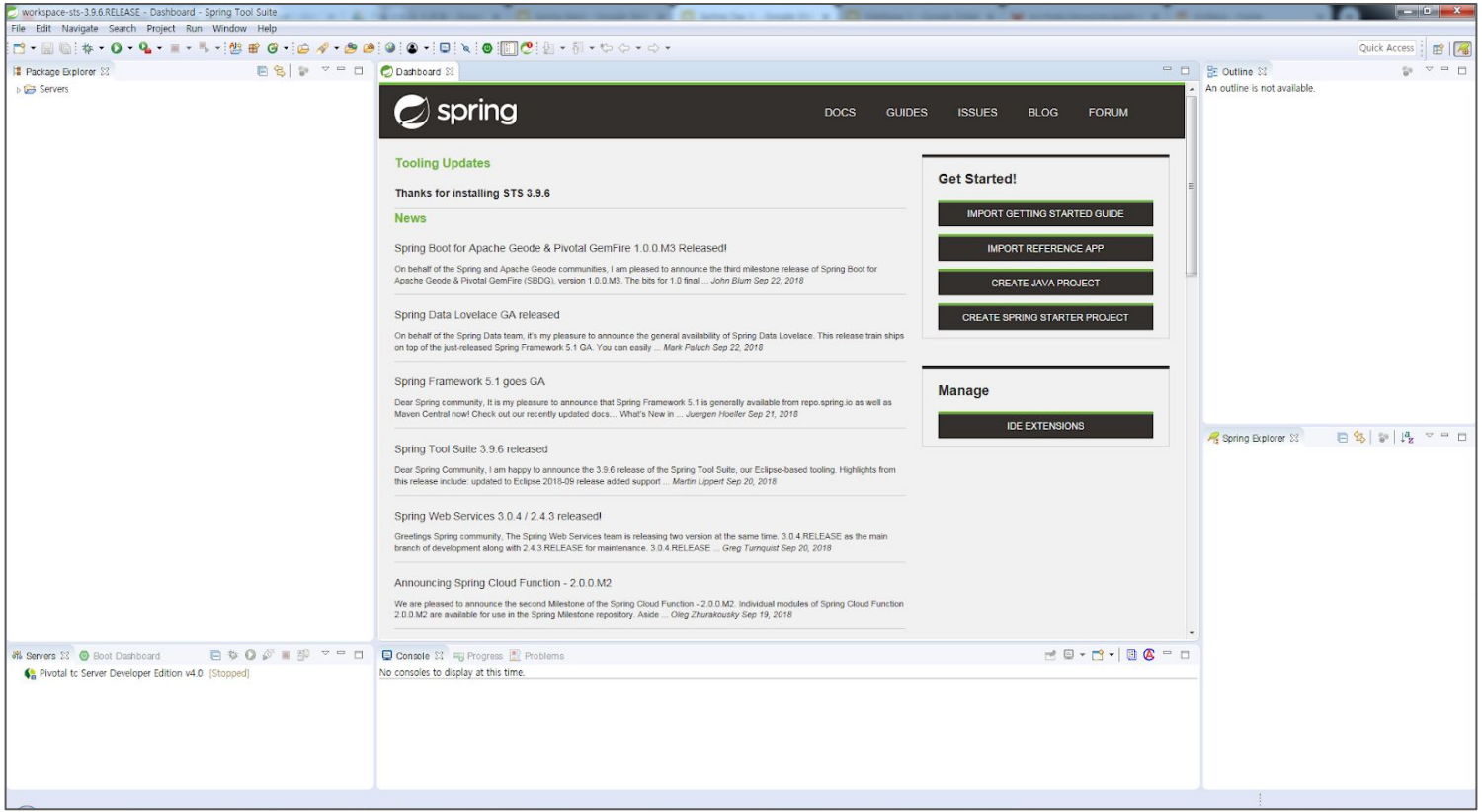
Download STS



Run the STS

- Execute STS_HOME\STS
  - some_directory\sts-bundle\sts-3.9.5.RELEASE\STS

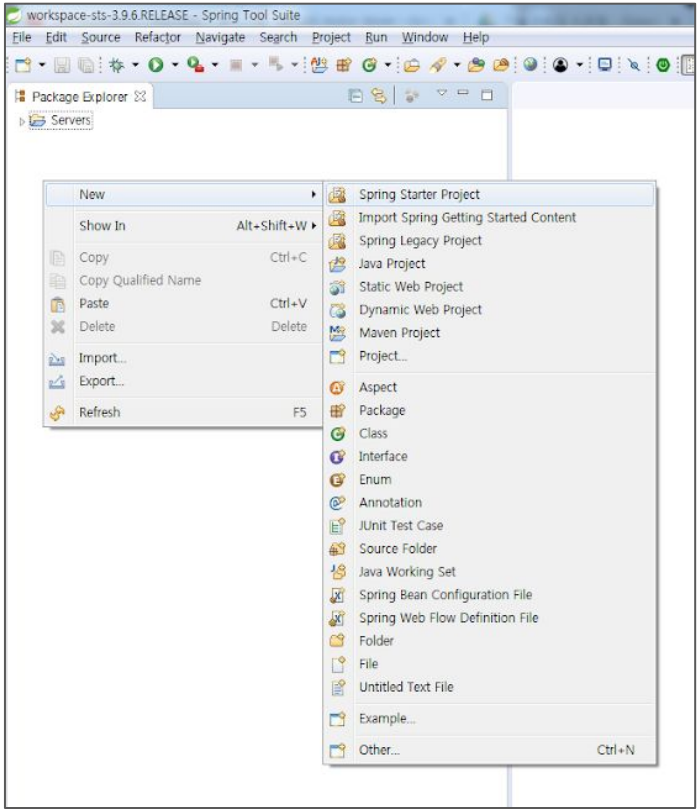
Run the STS

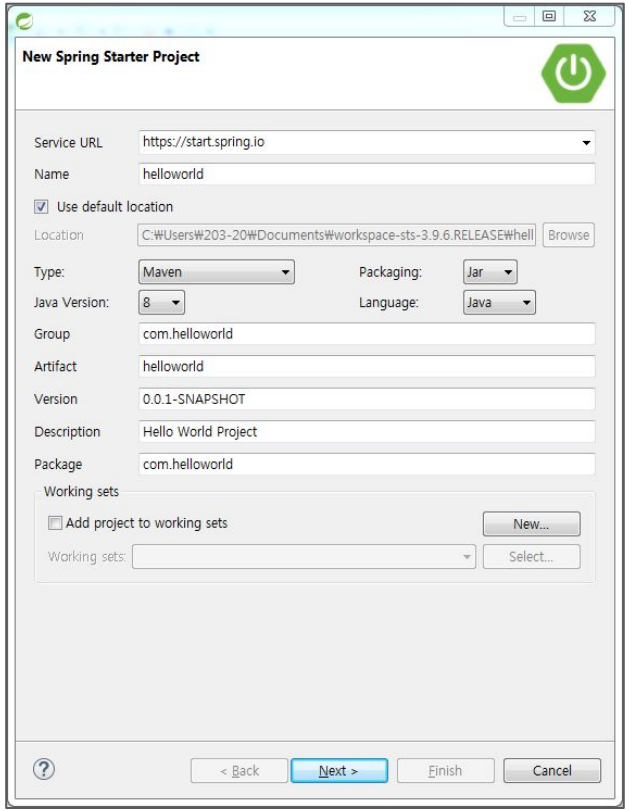

workspace-sts-3.9.6.RELEASE - Dashboard - Spring Tool Suite
File Edit Navigate Search Project Run Window Help
Quick Access
Package Explorer
Servers
Dashboard
spring
DOCS GUIDES ISSUES BLOG FORUM
Tooling Updates
Thanks for installing STS 3.9.6
News
Spring Boot for Apache Geode & Pivotal GemFire 1.0.0.M3 Released!
On behalf of the Spring and Apache Geode communities, I am pleased to announce the third milestone release of Spring Boot for Apache Geode & Pivotal GemFire (SBDG), version 1.0.0.M3. The bits for 1.0 final ... John Blum Sep 22, 2018
Spring Data Lovelace GA released
On behalf of the Spring Data team, it's my pleasure to announce the general availability of Spring Data Lovelace. This release train ships on top of the just-released Spring Framework 5.1 GA. You can easily ... Mark Paluch Sep 22, 2018
Spring Framework 5.1 goes GA
Dear Spring community, It is my pleasure to announce that Spring Framework 5.1 is generally available from repo.spring.io as well as Maven Central now! Check out our recently updated docs... What's New in ... Juergen Hoeller Sep 21, 2018
Spring Tool Suite 3.9.6 released
Dear Spring Community, I am happy to announce the 3.9.6 release of the Spring Tool Suite, our Eclipse-based tooling. Highlights from this release include: updated to Eclipse 2018-09 release added support ... Martin Lippert Sep 20, 2018
Spring Web Services 3.0.4 / 2.4.3 released!
Greetings Spring community, The Spring Web Services team is releasing two version at the same time. 3.0.4.RELEASE as the main branch of development along with 2.4.3.RELEASE for maintenance. 3.0.4.RELEASE ... Greg Turnquist Sep 20, 2018
Announcing Spring Cloud Function - 2.0.0.M2
We are pleased to announce the second Milestone of the Spring Cloud Function - 2.0.0.M2. Individual modules of Spring Cloud Function 2.0.0.M2 are available for use in the Spring Milestone repository. Aside ... Oleg Zhurakousky Sep 19, 2018
Get Started!
IMPORT GETTING STARTED GUIDE
IMPORT REFERENCE APP
CREATE JAVA PROJECT
CREATE SPRING STARTER PROJECT
Manage
IDE EXTENSIONS
Outline
An outline is not available.
Spring Explorer
Servers
Boot Dashboard
Pivotal tc Server Developer Edition v4.0 [Stopped]
Console
Progress
Problems
No consoles to display at this time.

STS

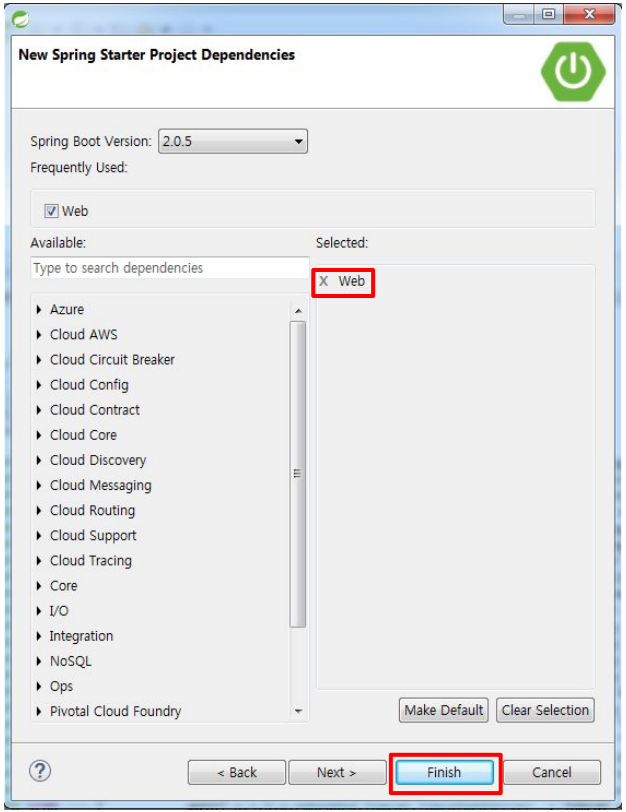
- STS stands for Spring Tool Suite
- Eclipse based IDE with fool and optimized plugins for Spring Framework
- You can use pure Eclipse with Spring plugins
- STS and Eclipse are both heavy and slow
- Recommendation is using IntelliJ
  - faster than eclipse
  - plugin environment is way more powerful

Creating HelloWorld Project



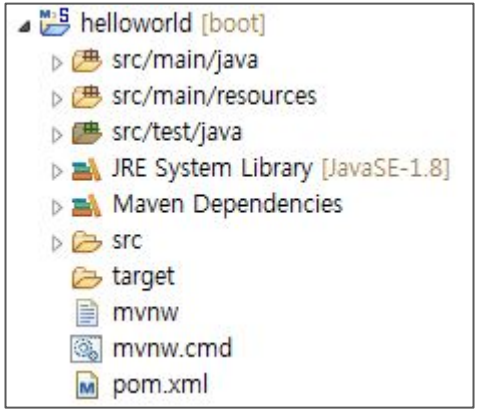


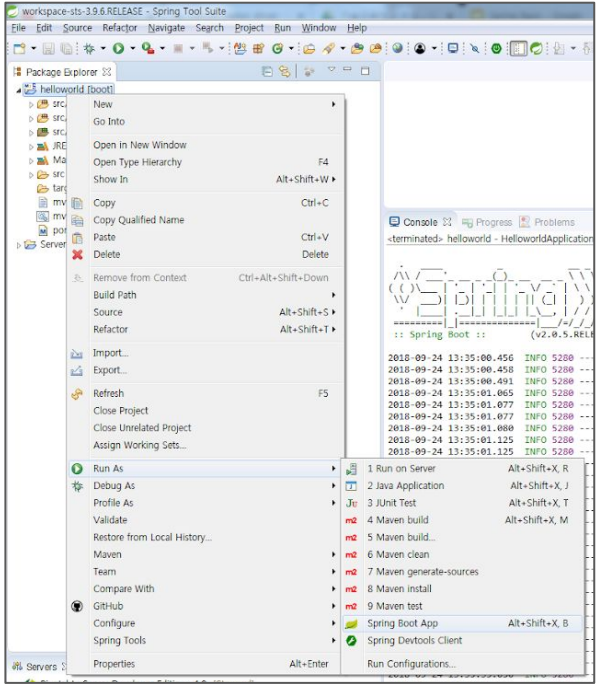
Creating HelloWorld Project



Running HelloWorld Project

- It takes few minutes
- After Creating, run the application

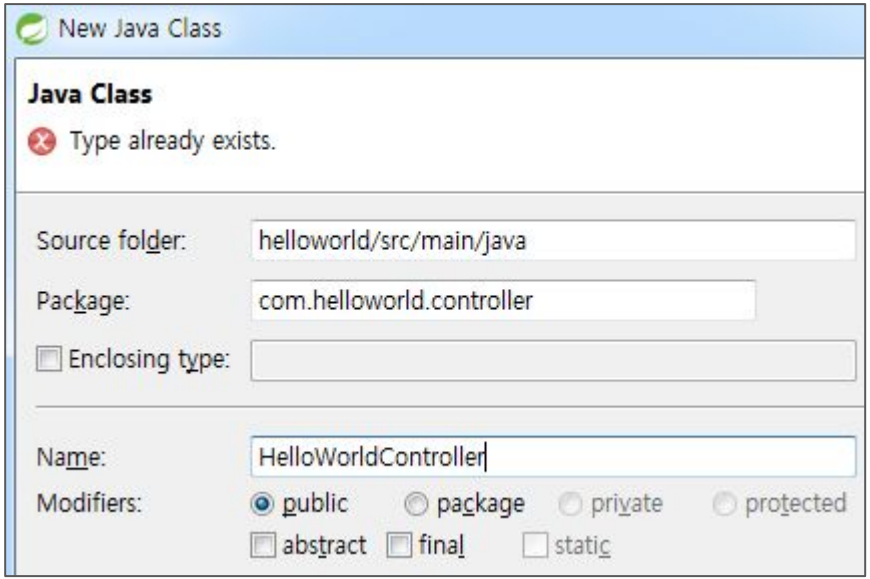




Running HelloWorld Project

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.5.RELEASE)

2018-09-24 13:39:02.171  INFO 3200 --- [           main] com.helloworld.HelloworldApplication     : Starting HelloworldApplication on 203-20-PC with PID 3200 (C:\Users\203-20\Documents\workspace-sts-3.9.6.RELEASE\hell
2018-09-24 13:39:02.173  INFO 3200 --- [           main] com.helloworld.HelloworldApplication     : No active profile set, falling back to default profiles: default
2018-09-24 13:39:02.206  INFO 3200 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicationContext@f2f2cc1: s
2018-09-24 13:39:02.777  INFO 3200 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-09-24 13:39:02.790  INFO 3200 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-09-24 13:39:02.790  INFO 3200 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.34
2018-09-24 13:39:02.793  INFO 3200 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found
2018-09-24 13:39:02.837  INFO 3200 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-09-24 13:39:02.837  INFO 3200 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 633 ms
2018-09-24 13:39:02.872  INFO 3200 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-09-24 13:39:02.874  INFO 3200 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-09-24 13:39:02.874  INFO 3200 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-09-24 13:39:02.874  INFO 3200 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-09-24 13:39:02.874  INFO 3200 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-09-24 13:39:02.938  INFO 3200 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRe
2018-09-24 13:39:03.058  INFO 3200 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.context.AnnotationConfigServletWebServerApplicati
2018-09-24 13:39:03.091  INFO 3200 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Obj
2018-09-24 13:39:03.092  INFO 3200 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework
2018-09-24 13:39:03.105  INFO 3200 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpReques
2018-09-24 13:39:03.105  INFO 3200 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler
2018-09-24 13:39:03.180  INFO 3200 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-09-24 13:39:03.203  INFO 3200 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-09-24 13:39:03.206  INFO 3200 --- [           main] com.helloworld.HelloworldApplication     : Started HelloworldApplication in 1.203 seconds (JVM running for 1.534)
```

Creating Controller Class




```java
package com.helloworld.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloWorldController {

    @RequestMapping("/")
    @ResponseBody
    public String getIndex() {
        return "<h1>Hello World!!</h1>";
    }
}
```

Calling Web Page of HelloWorld Project

- Open your browser

- Go to http://localhost:8080

URI

- Stands for Uniform Resource Identifier
- Like address or name for identifying a resource in World Wide Web
- URI contains URL(Uniform Resource Locator) and URN(Uniform Resource Name)

The Components of a URI

- Scheme: http://
- Host: someaddress.com:8080
- Path: /member
- Query String: ?id=1
- http://someaddress.com:8080/member?id=1

Controller Class

```java
package com.helloworld.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloWorldController {

    @RequestMapping("/")
    @ResponseBody
    public String getIndex() {
        return "<h1>Hello World!!</h1>";
    }
}
```

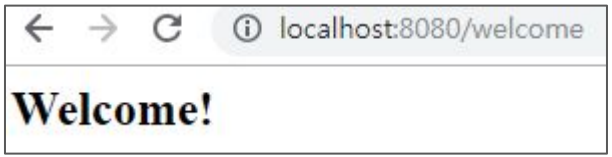
Specify that this class is controller class

Specify URI Path of Controller Method

Reponse Value of Controller Method

Exercise - Add Controller Method

- Add Controller method named ‘getWelComePage’ in the PostController class
- URI path is ‘/welcome’
- Response Value is ‘<h2>Welcome!</h2>’



What is Controller

- Controller is responsible for accepting client's request
- Controller is also responsible for returning the response to client
- Controller calls service method to process business logics
- Controller has request parameters, @Requestmapping, return type

REST API

- REST stands for REpresentational State Transfer
- REST API is not a standard but a architectural style or design pattern
- REST API uses HTTP Protocol to request and response
- Client can easily communicate with server by using REST API

REST API

- Resource
  - URI
- Method
  - GET for getting data
  - POST for saving data
  - PUT for updating data
  - DELETE for deleting data
- Representation of Resource
  - Form/Type of data used for communication(json, xml etc.)

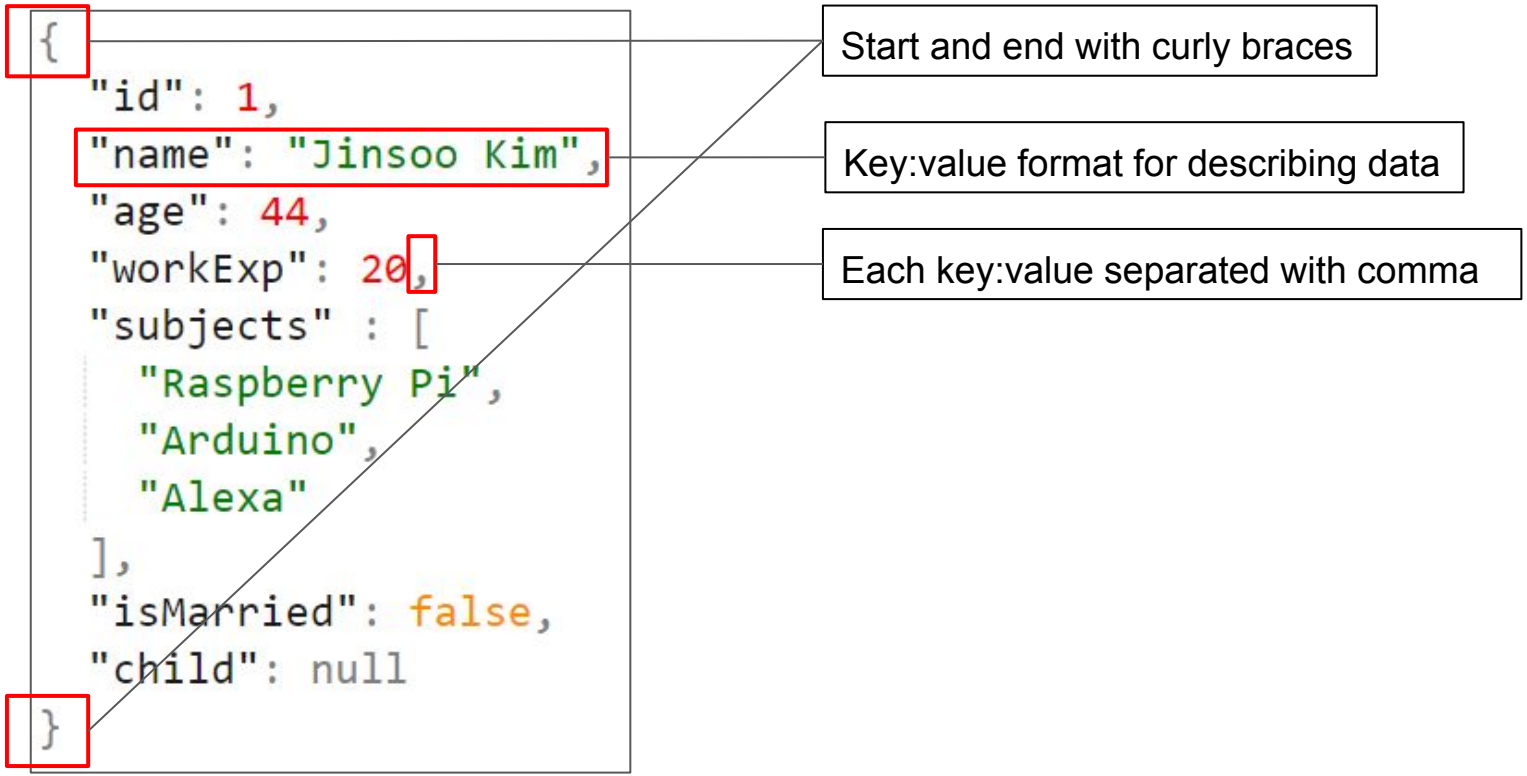
JSON

- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format(lighter than xml)
- It is easy for humans to read and write

JSON Data Types

- String
- Number
- Object (JSON object)
- Array
- Boolean
- Null

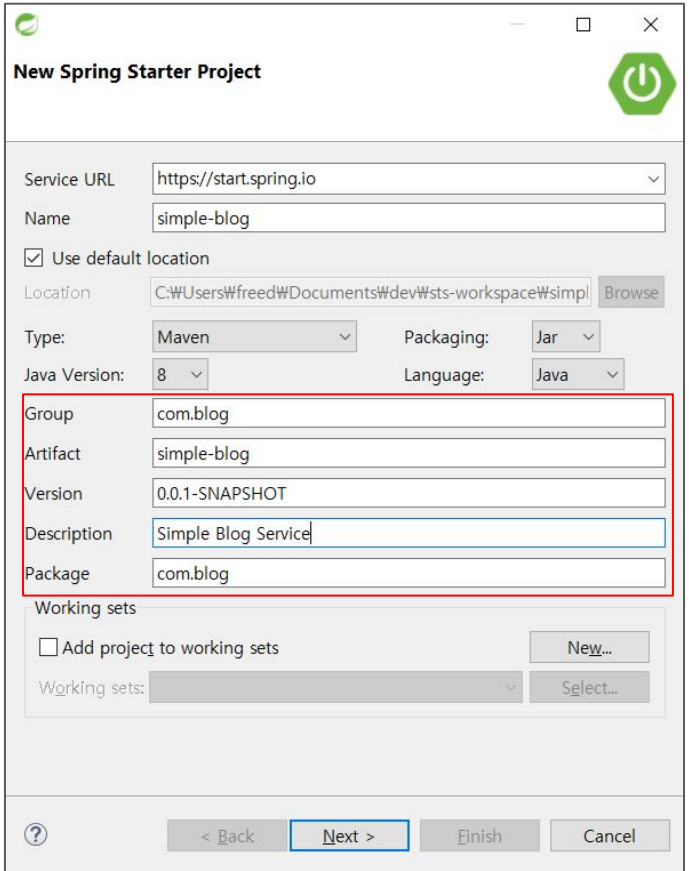
JSON Format


{
"id": 1,
"name": "Jinsoo Kim",
"age": 44,
"workExp": 20,
"subjects" : [
"Raspberry Pi",
"Arduino",
"Alexa"
],
"isMarried": false,
"child": null
}
Start and end with curly braces
Key:value format for describing data
Each key:value separated with comma

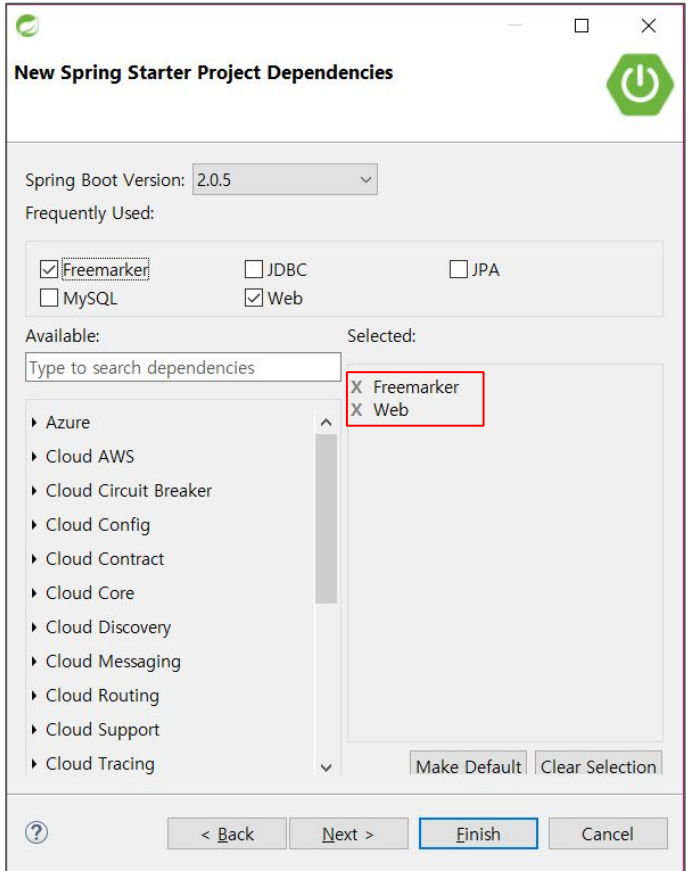
JSON Object

- JSON Object is basic unit
- JSON Object can contain
  - key/value
  - another JSON Object
  - JSON Array

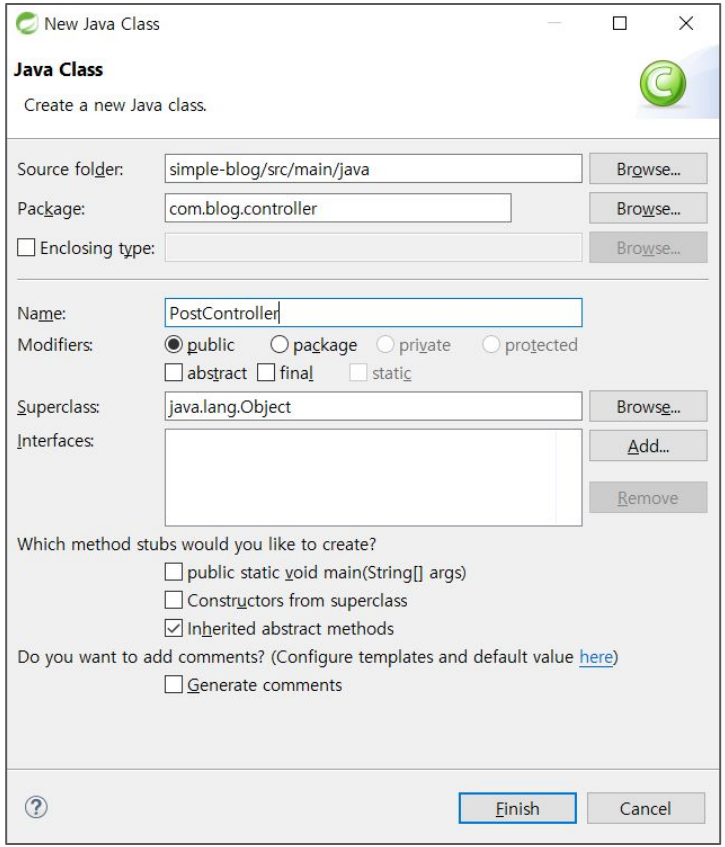
Create New Spring Starter Project



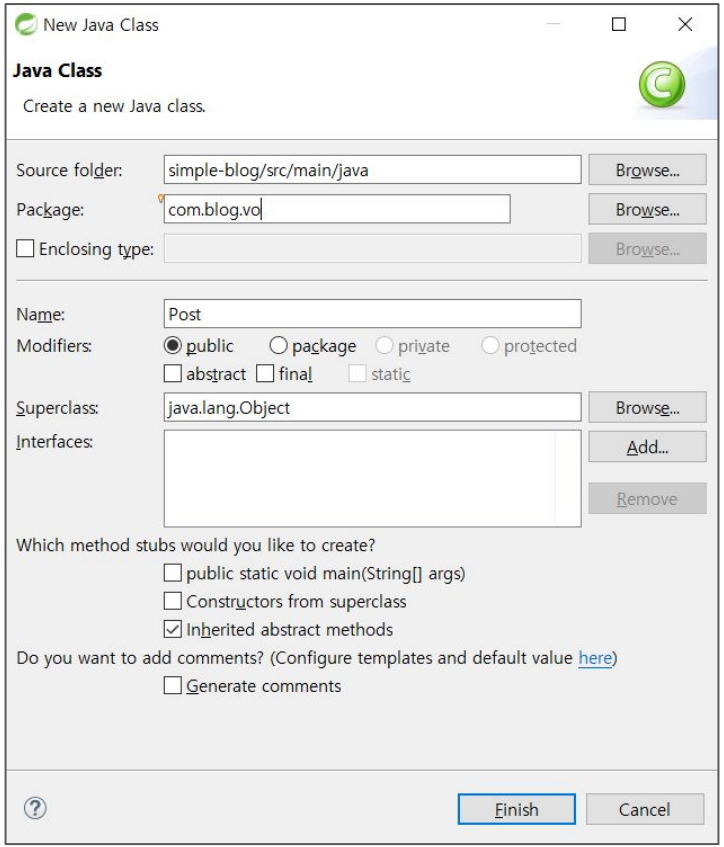
Add Dependencies for Spring Project



Create New Controller Class - PostController



Create New VO Class - Post



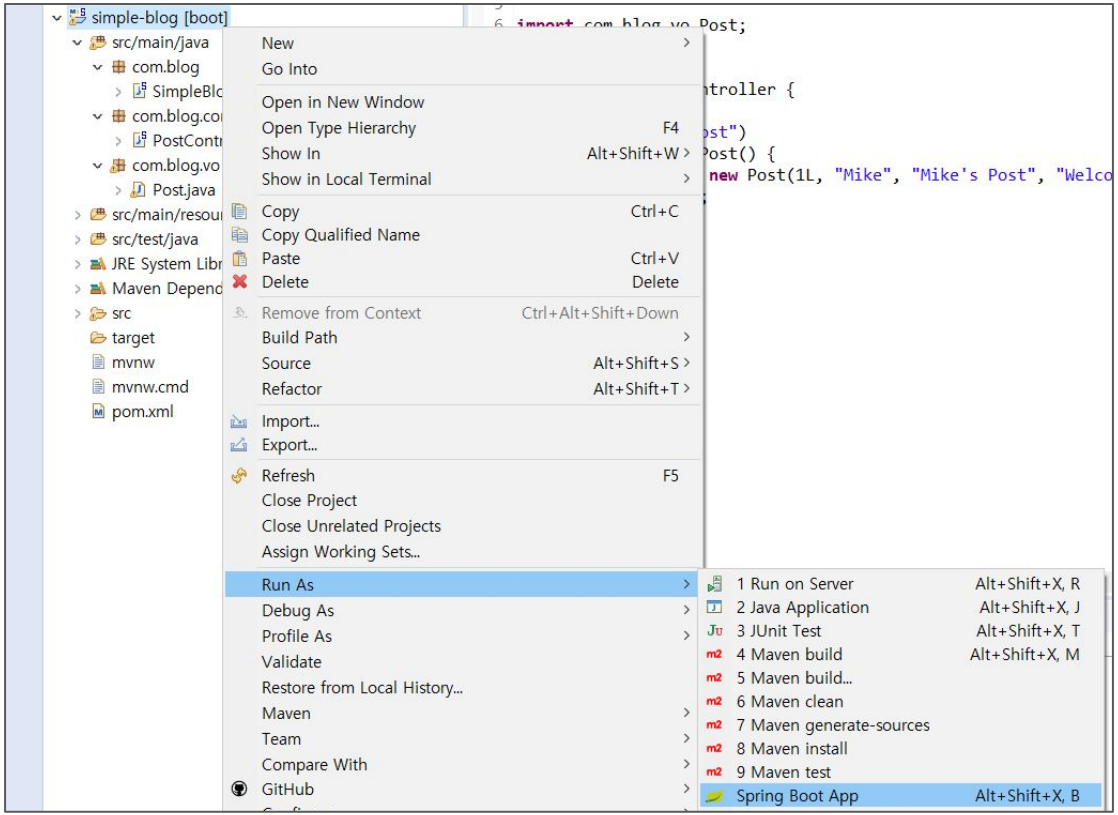
Post Class

```java
package com.blog.vo;

import java.util.Date;

public class Post {
    private Long id;
    private String user;
    private String title;
    private String content;
    private Date regDate;
    private Date updtDate;

    public Post() {
    }

    public Post(Long id, String user, String title, String content) {
        super();
        this.id = id;
        this.user = user;
        this.title = title;
        this.content = content;
        this.regDate = new Date();
        this.updtDate = new Date();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Date getRegDate() {
        return regDate;
    }

    public void setRegDate(Date regDate) {
        this.regDate = regDate;
    }

    public Date getUpdtDate() {
        return updtDate;
    }

    public void setUpdtDate(Date updtDate) {
        this.updtDate = updtDate;
    }

}
```

PostController Class

```java
package com.blog.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import com.blog.vo.Post;

@RestController
public class PostController {

    @GetMapping("/post")
    public Post getPost() {
        Post post = new Post(1L, "Mike", "Mike's Post", "Welcome to My blog");
        return post;
    }
}
```

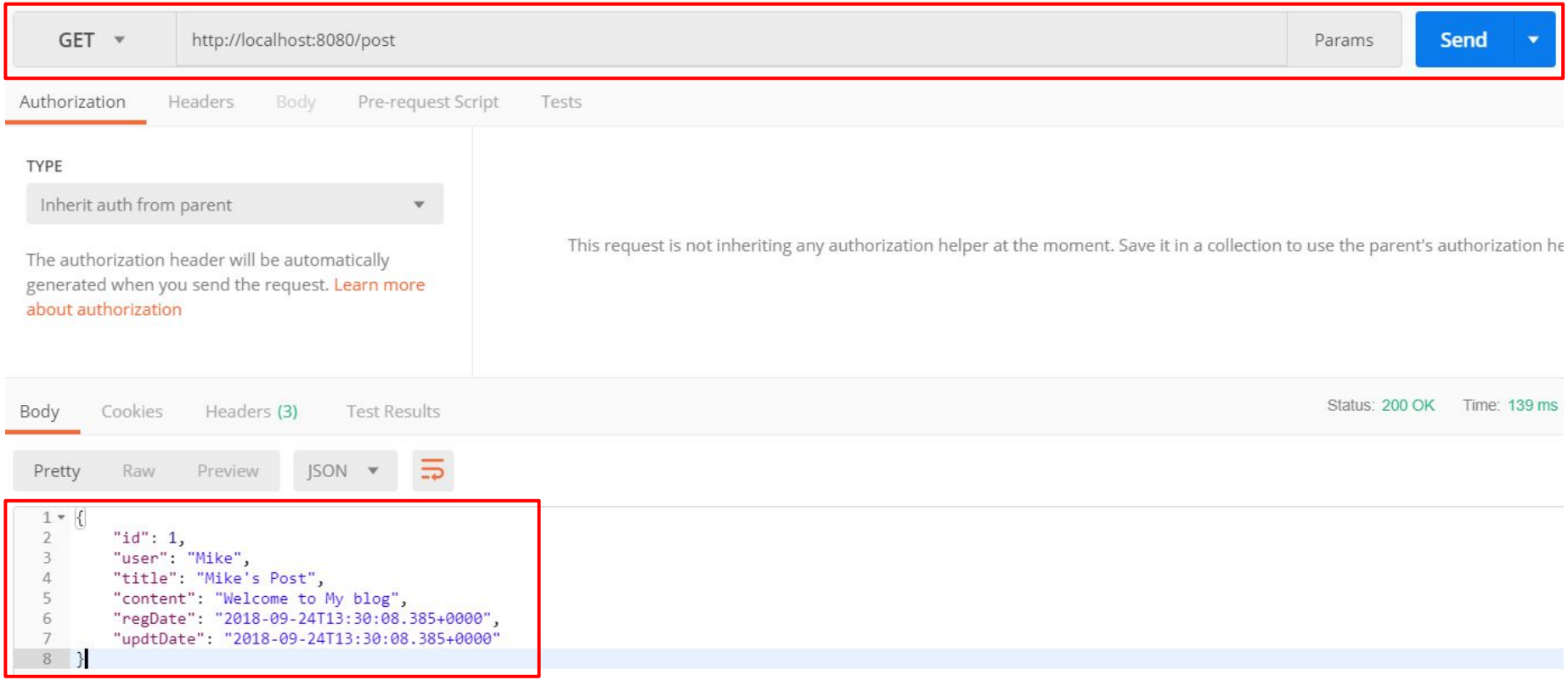
Run the application



Test Rest API - Postman

- https://www.getpostman.com/apps
- Download and install
- Signup with Google account or create new account

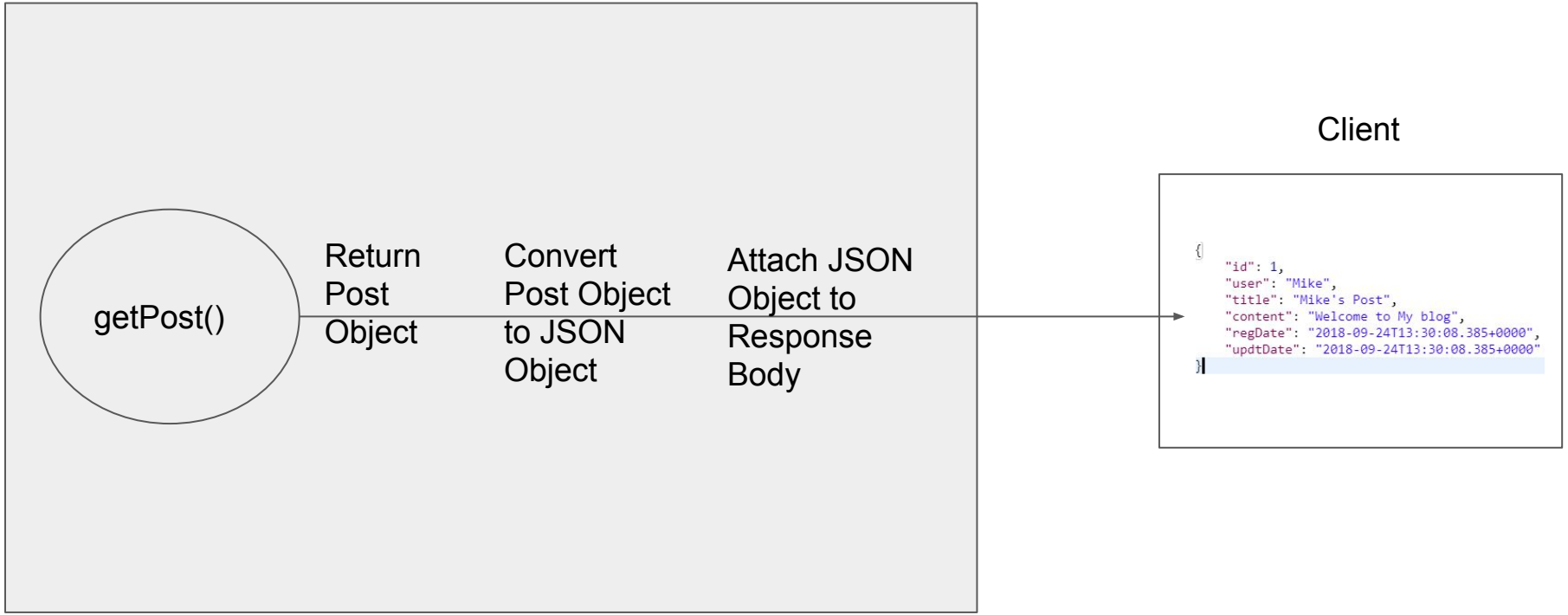
Test Rest API - Postman



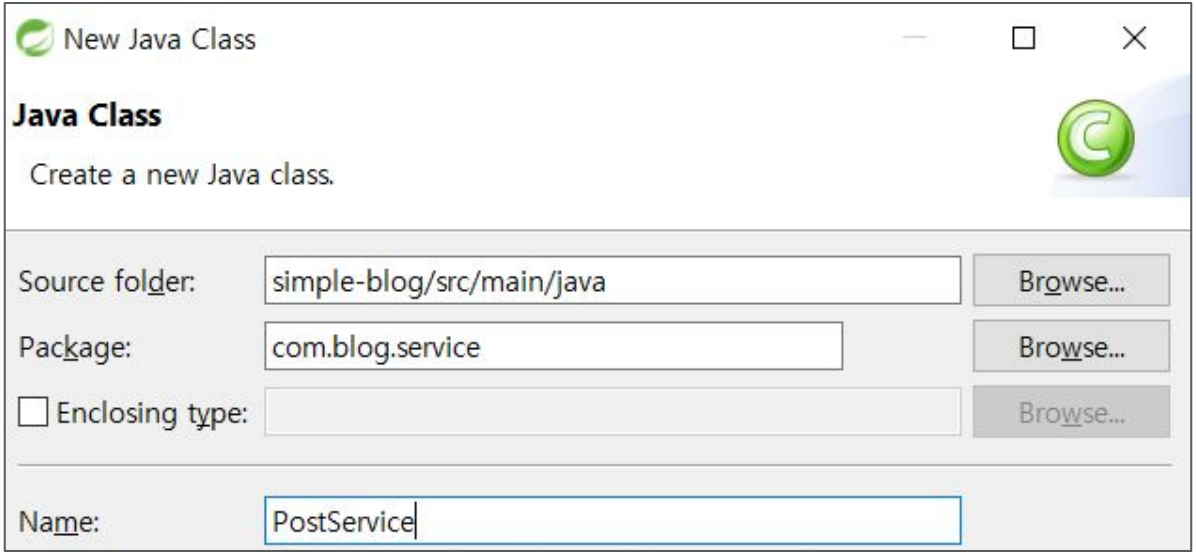
Result JSON Object


```
{
    "id": 1,
    "user": "Mike",
    "title": "Mike's Post",
    "content": "Welcome to My blog",
    "regDate": "2018-09-24T13:30:08.385+0000",
    "updtDate": "2018-09-24T13:30:08.385+0000"
}
```

What Happened in Spring Framework



Create New PostService Class



PostService

```java
package com.blog.service;

import org.springframework.stereotype.Service;

import com.blog.vo.Post;

@Service
public class PostService {

    public Post getPost() {
        Post post = new Post(1L, "Mike", "Mike's Post", "Welcome to My blog");

        return post;
    }
}
```

Modify PostController Class

```java
package com.blog.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import com.blog.service.PostService;
import com.blog.vo.Post;

@RestController
public class PostController {

    @Autowired
    PostService postService;

    @GetMapping("/post")
    public Post getPost() {
        Post post = postService.getPost();
        return post;
    }
}
```

Check the Result

- Rerun the application
- Call http://localhost:8080/post in PostMan
- Check the Result

@Autowired Annotation

- To call method in another class, we should create object by using new keyword
- ‘PostService postService = new PostService()’ is needed
- PostController class uses postService without the code
- This is magic of @Autowired Annotation

Spring Bean

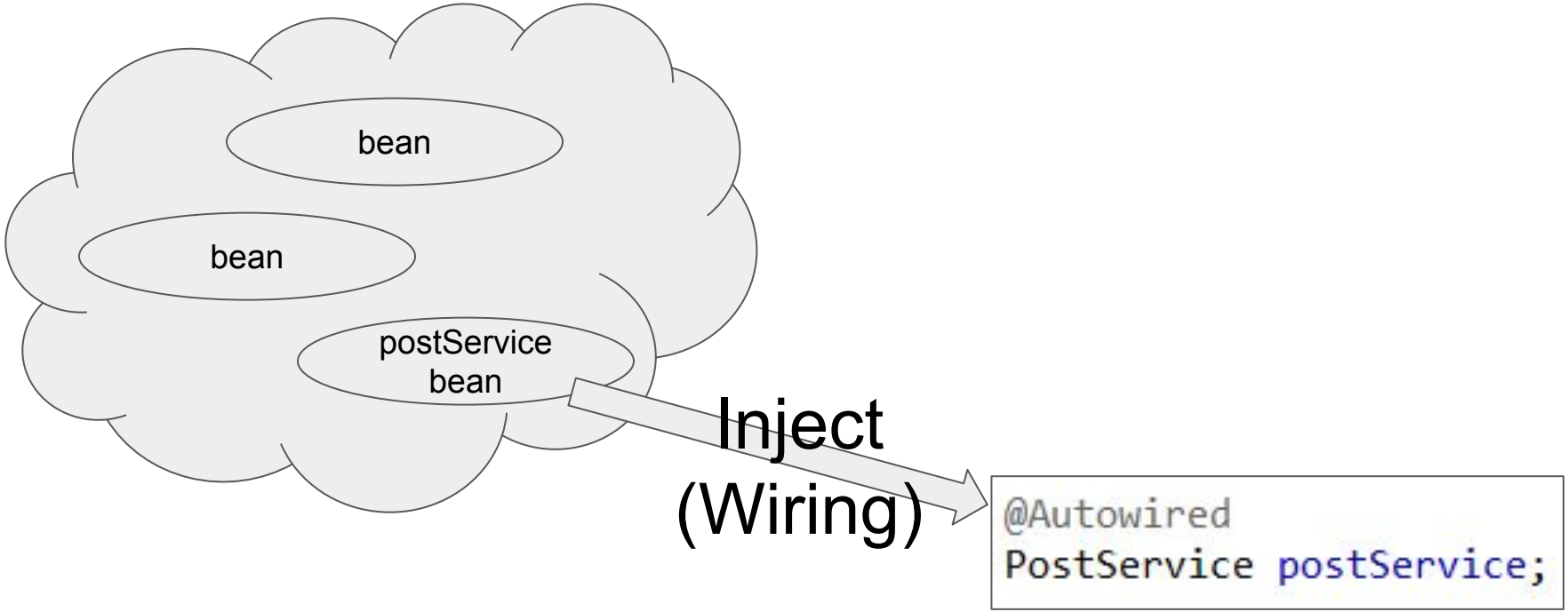
- Spring Bean is simply a Java object
- Java object is created by application code
- Spring Bean is created and managed by Spring Framework
- To make Spring Bean
  - @Controller, @RestController, @Service, @Repository annotations are used
- Spring Beans can be used in any place of Spring app by using @Autowired annotation

How To Create Spring Beans

- When application starts, Spring Framework scans all user generated classes
- If a user generated class has annotation such as @Controller, @Service etc, Spring Framework creates bean from the class
- This process is called Component Scanning
- The SW component that scans and creates beas is called Bean Factory or Application Context

Dependency Injection

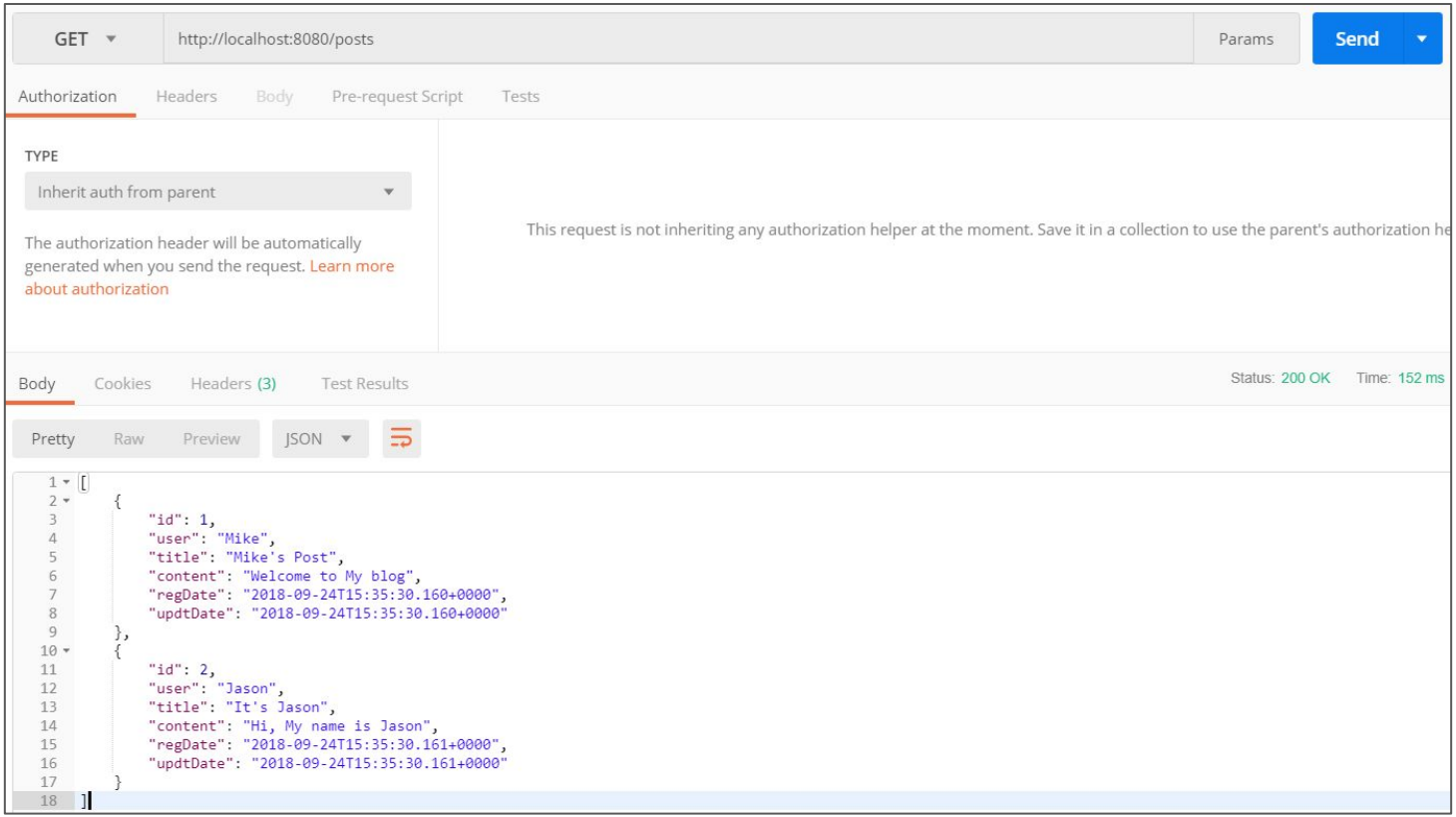
- Dependency Injection is a design pattern that delegating the role of creating and managing dependencies to the other SW such as framework



Add New Method to PostController Class

```
@GetMapping("/posts")
public Post[] getPosts() {
    Post[] posts = new Post[] {
        new Post(1L, "Mike", "Mike's Post", "Welcome to My blog"),
        new Post(2L, "Jason", "It's Jason", "Hi, My name is Jason")
    };

    return posts;
}
```

Test posts REST API



Result of posts REST API


```
[
  {
      "id": 1,
      "user": "Mike",
      "title": "Mike's Post",
      "content": "Welcome to My blog",
      "regDate": "2018-09-24T15:35:30.160+0000",
      "updtDate": "2018-09-24T15:35:30.160+0000"
  },
  {
      "id": 2,
      "user": "Jason",
      "title": "It's Jason",
      "content": "Hi, My name is Jason",
      "regDate": "2018-09-24T15:35:30.161+0000",
      "updtDate": "2018-09-24T15:35:30.161+0000"
  }
]
```

JSON Array

- JSON array are ordered list of values
- JSON array can store multiple value types
- JSON array can store string, number, boolean, object or other array inside JSON array
- The square brackets [ ] are used to declare JSON array

How To Get Json Array Response

- Make return type of controller method to Java Array
- Make return type of controller method to List

Modify getPosts() Method

```
@GetMapping("/posts")
public List<Post> getPosts() {
    List<Post> posts = new ArrayList<>();

    posts.add(new Post(1L, "Mike", "Mike's Post", "Welcome to My blog"));
    posts.add(new Post(2L, "Jason", "It's Jason", "Hi, My name is Jason"));

    return posts;
}
```

Test Modified Controller Method

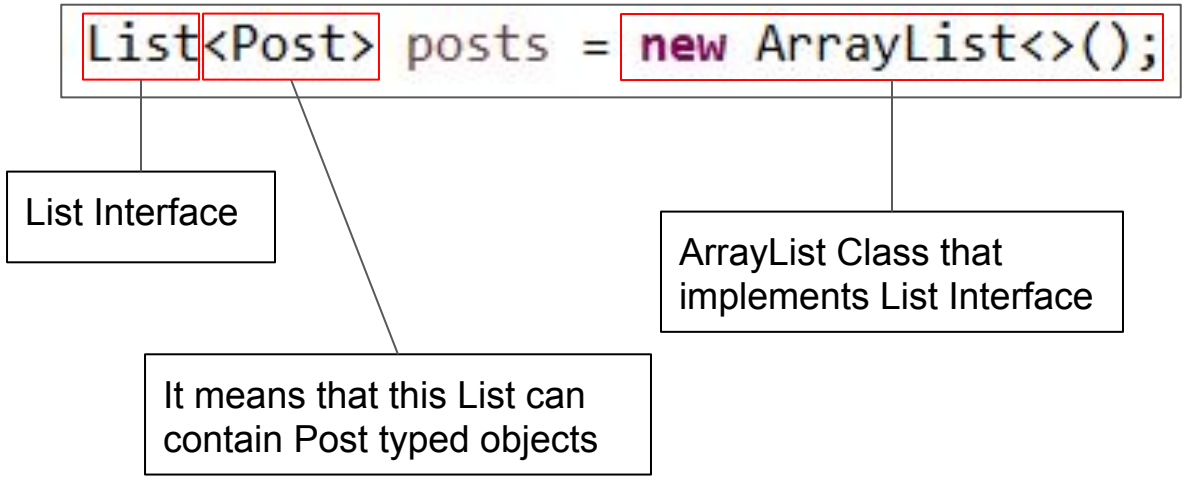
- Rerun the application
- Call http://localhost:8080/posts API
- Check the result

Java Collection Framework

- The Java Collections Framework is a collection of interfaces and classes which helps in storing and processing the data efficiently
- Many well-known data structures are implemented in JCF such as Linked List, Hash Set, Sorted Set, Hash Table etc.
- We can use various data structure easily and in similar way with JCF

List and ArrayList

- List/ArrayList are most frequently used component in JCF
- ArrayList can have flexible size in contrast with array



Useful Method for List

- add(Object o): Adding new object to List
- get(int index): Returning a object in specified index from List
- remove(Object o): Deleting specified object in List
- contains(Object o): Returning whether specified object exists or not

JSON Response of Real World Application

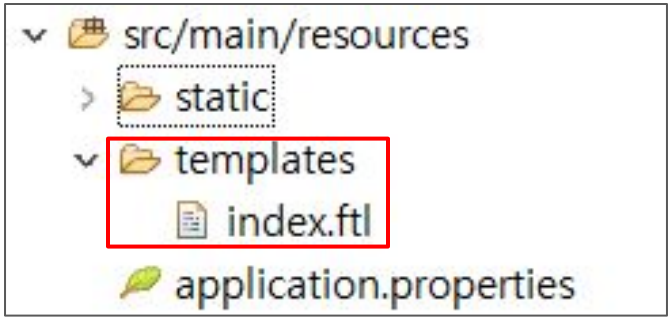
- http://www.onstove.com
- Social Network Platform for Gamer made by SmileGate
- 1,000,000,000+ articles

Work with Web Page

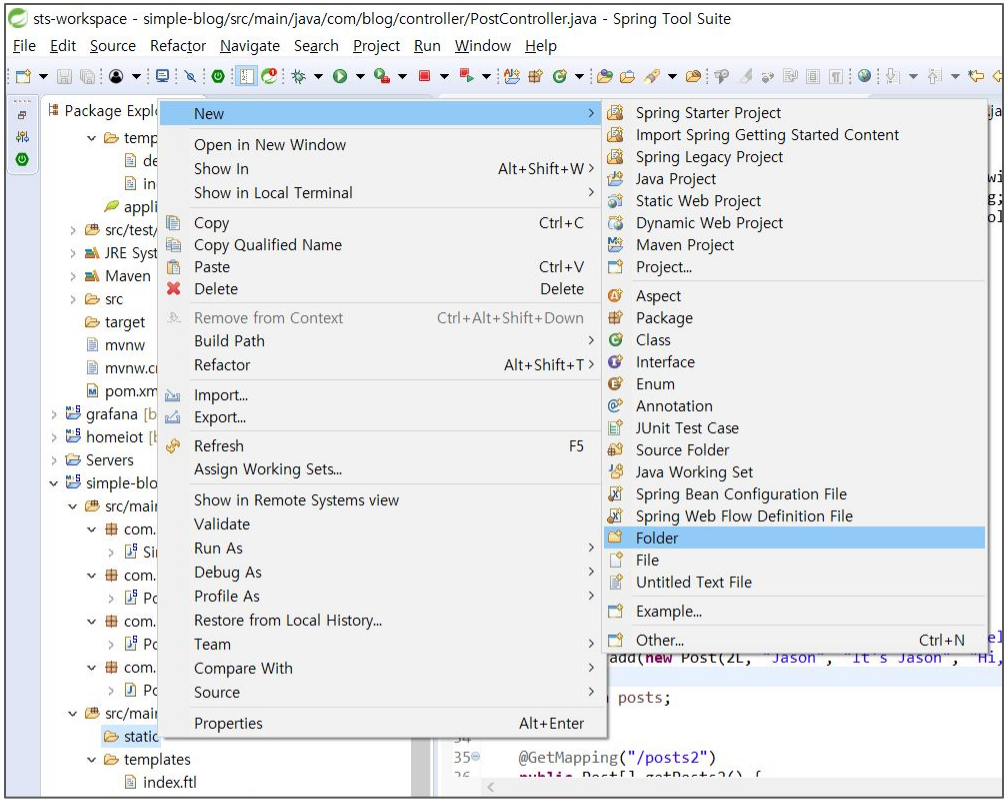
- Download freemarker and javascript files
- Copy to proper folders

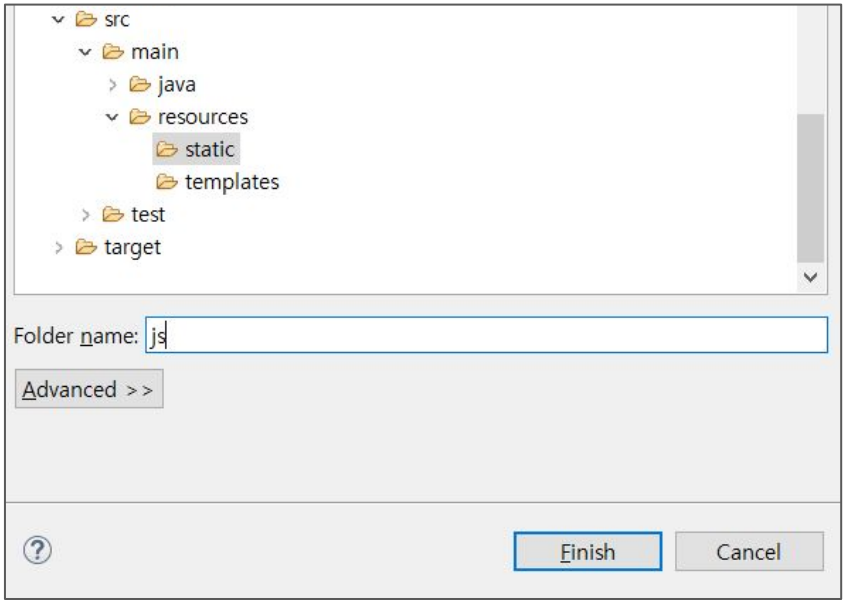
Copy index.ftl File

- Copy index.ftl file to
  - src/main/resource/templates folder



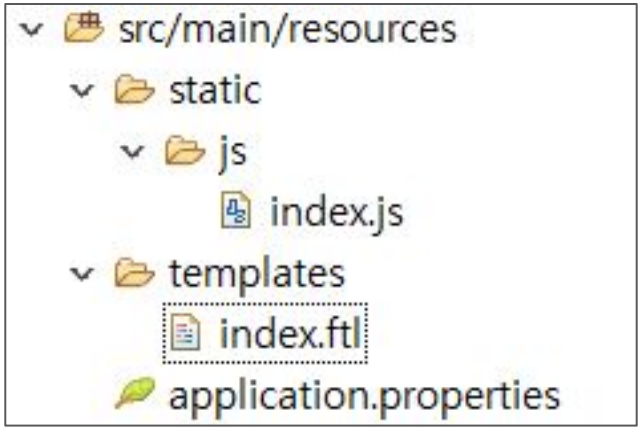
Create ‘js’ Folder in src/main/resource/static





Copy index.js File

- Copy index.js file to
  - src/main/resource/static/js folder



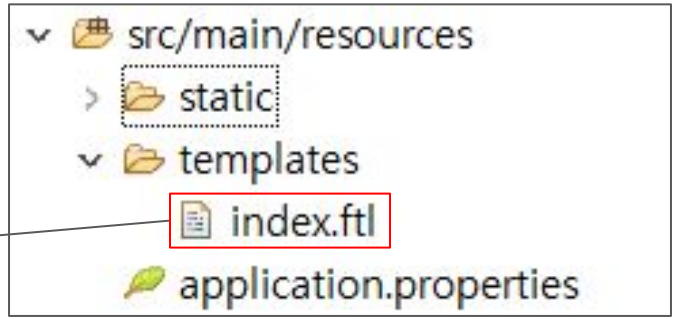
Make New PostPageController Class

```java
package com.blog.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class PostPageController {

    @RequestMapping("/page/index")
    public String getIndexPage() {
        return "index";
    }
}
```

Controller for Getting HTML Page

- To get html pages, template and corresponding controller are needed
- Template files would be located in src/main/resource/templates
- In controller, to return template controller method return the name of template without extension

```java
package com.blog.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class PostPageController {

    @RequestMapping("/page/index")
    public String getIndexPage() {
        return "index";
    }
}
```



Two Types of Controller

- Controller for REST API
  - use @RestController for Class
  - controller method can return any types
- Controller for Page
  - use @Controller for class
  - controller method should return String typed value;

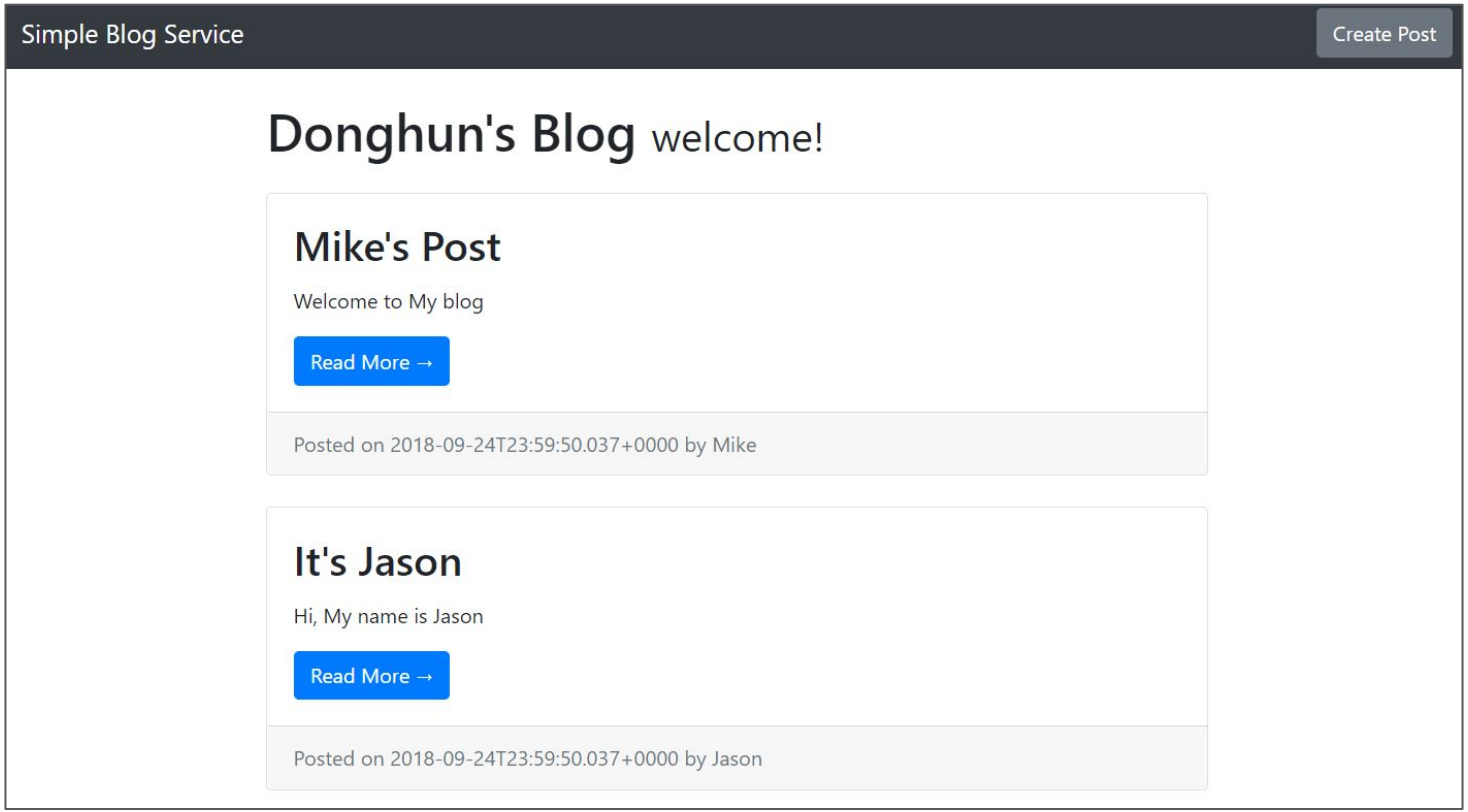
```
@RestController
public class PostController {
```

```
@GetMapping("/post")
public Post getPost() {
    Post post = postService.getPost();
    return post;
}
```

```
@Controller
public class PostPageController {
```

```
@RequestMapping("/page/index")
public String getIndexPage() {
    return "index";
}
```

Check Page - http://localhost:8080/page/index



Frontend Development

- Frontend technology is evolving, growing crazily fast now days
- ReactJs, AngularJs, VueJs is 3 major framework right now
- For big application, using frontend framework is good choice
- But for small application. using template engine is more suitable
- Freemarker is one of the template engine

Freemarker

- Freemarker is template engine for Java application
- Freemarker make HTML code from Java Object
- The extension of Freemarker file is ftl

Add New Method to PostService Class

```java
package com.blog.service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Service;

import com.blog.vo.Post;

@Service
public class PostService {
    private static List<Post> posts;

    public Post getPost() {
        Post post = new Post(1L, "Mike", "Mike's Post", "Welcome to My blog");

        return post;
    }

    public List<Post> getPosts() {
        posts = new ArrayList<>();
        posts.add(new Post(1L, "Mike", "Mike's Post", "Welcome to My blog"));
        posts.add(new Post(2L, "Jason", "It's Jason", "Hi, My name is Jason"));

        return posts;
    }

}
```

Modify getPosts() method

```
@GetMapping("/posts")
public List<Post> getPosts() {
    List<Post> posts = postService.getPosts();
    return posts;
}
```

Rerun and Test

- Rerun the application
- Call http://localhost:8080/posts
- Check the result
- Check the web page http://localhost:8080/page/index

Modify getPost method in PostService Class

```
public Post getPost(int id) {
    Post post = posts.get(id-1);

    return post;
}
```

Modify getPost method in PostController Class

```
@GetMapping("/post")
public Post getPost(@RequestParam("id") int id) {
    Post post = postService.getPost(id);
    return post;
}
```

Test Modified API

- Rerun the application
- Call http://localhost:8080/posts API first
- And then call http://localhost:8080/post?id=1
- Call http://localhost:8080/post?id=2

Request Parameter

- There are 3 types of request parameter
- Query String
  - http://localhost:8080/post?id=1
- Path Parameter
  - http://localhost:8080/post/1
- Body Parameter
  - key/value or json type and attached to request body

Add a New Method to PostController

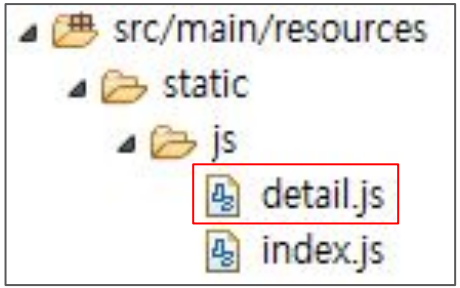
```
@GetMapping("/post/{id}")
public Post getPostPathParam(@PathVariable("id") int id) {
    Post post = postService.getPost(id);
    return post;
}
```

Processing Request Parameter in Spring

- @RequestParam for Query String
- @PathVariable for Path Parameter
- @RequestBody for Body Parameter

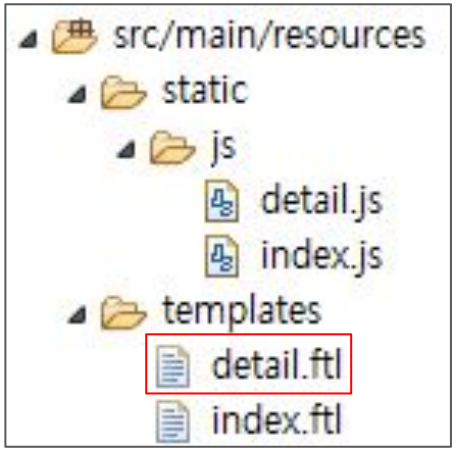
Copy detail.js File

- Copy detail.js file to
  - src/main/resource/static/js folder



Copy detail.ftl File

- Copy detail.ftl file to
  - src/main/resource/templates folder



Add New Method to PostPageController

```
@RequestMapping("/page/detail/{id}")
public String getDetailPage(@PathVariable("id") int id, Model model) {
    model.addAttribute("id", id);
    return "detail";
}
```

Model in Spring Framework

- Model is used to deliver some data from controller to web page
- We can use the data from controller in freemarker template by using ${} operator

```
@RequestMapping("/page/detail/{id}")
public String getDetailPage(@PathVariable("id") int id, Model model) {
    model.addAttribute("id", id);
    return "detail";
}
```

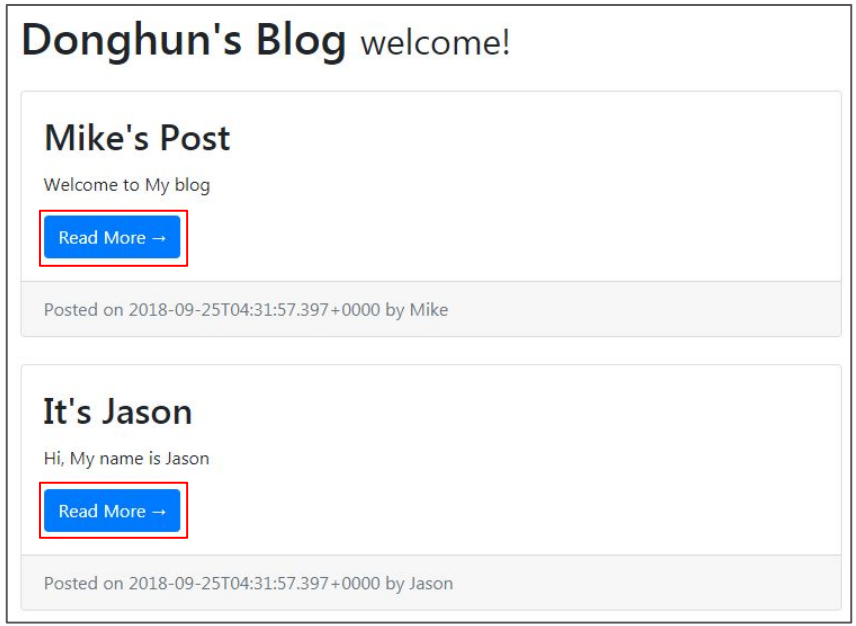
```
<input type="hidden" id="detail_post_id" value="${id}">
```

Test New Controller Method

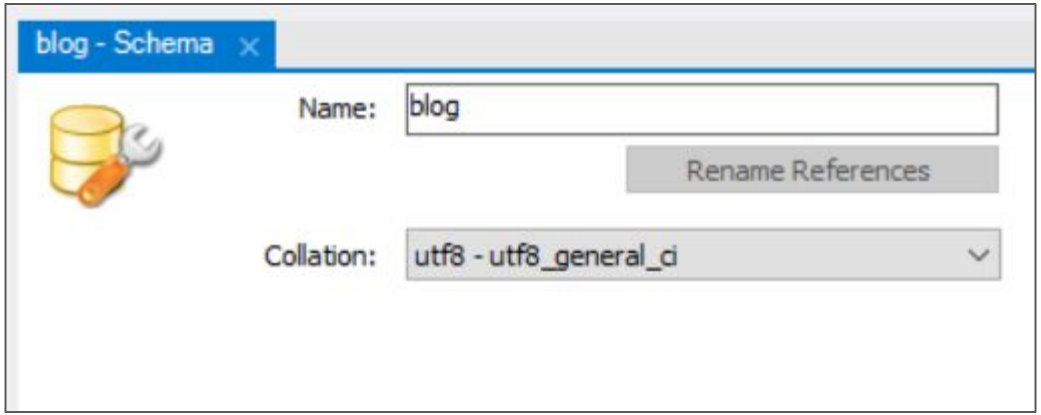
- Rerun the application
- Call http://localhost:8080/posts API first
- Open the browser
- Go to http://localhost:8080/page/detail/1 page
- Go to http://localhost:8080/page/detail/2 page

Test index Page

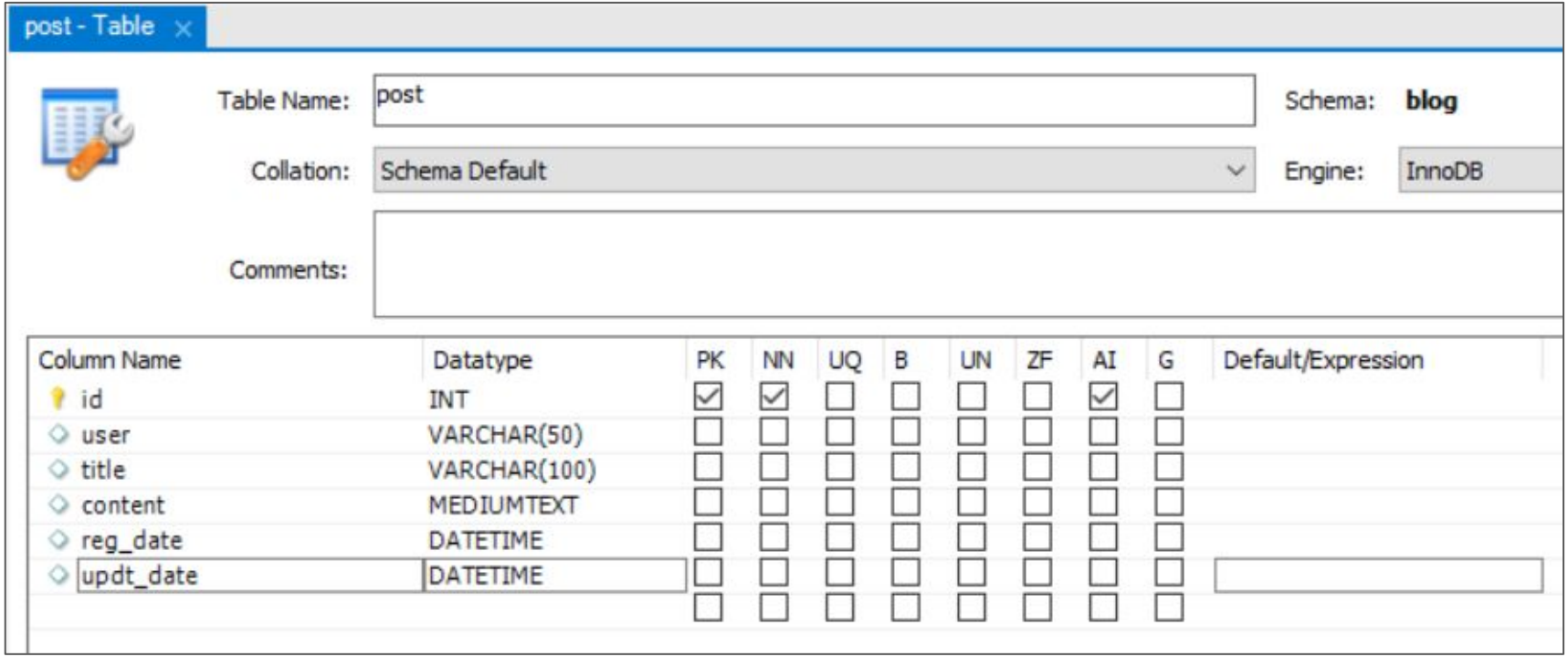
- Go to http://localhost:8080/page/index
- Click ‘Read More’ button for each post



Create blog Schema

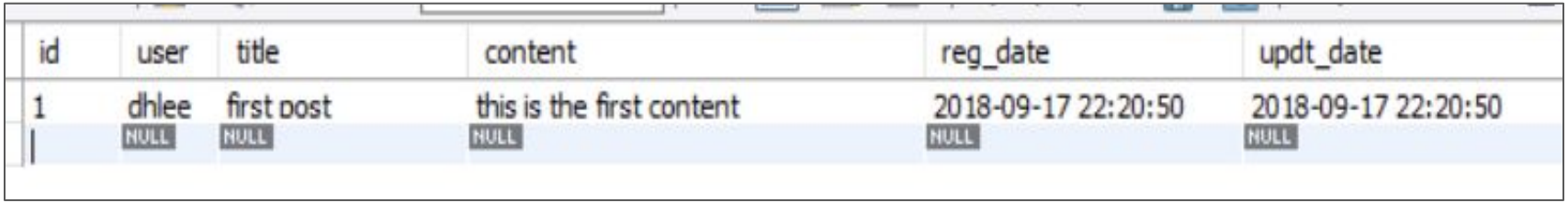


Create ‘post’ Table in the blog Schema



Insert Sample Data to post Table

- INSERT INTO `blog`.`post` (`user`, `title`, `content`, `reg_date`, `updt_date`) VALUES ('dhlee', 'first post', 'this is the first content', now(), now());



id	user	title	content	reg_date	updt_date
1	dhlee	first post	this is the first content	2018-09-17 22:20:50	2018-09-17 22:20:50
	NULL	NULL	NULL	NULL	NULL

Add dependencies

- Add dependencies to the pom.xml file
- Following dependencies should be located in <dependencies> tag

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

Configure Database

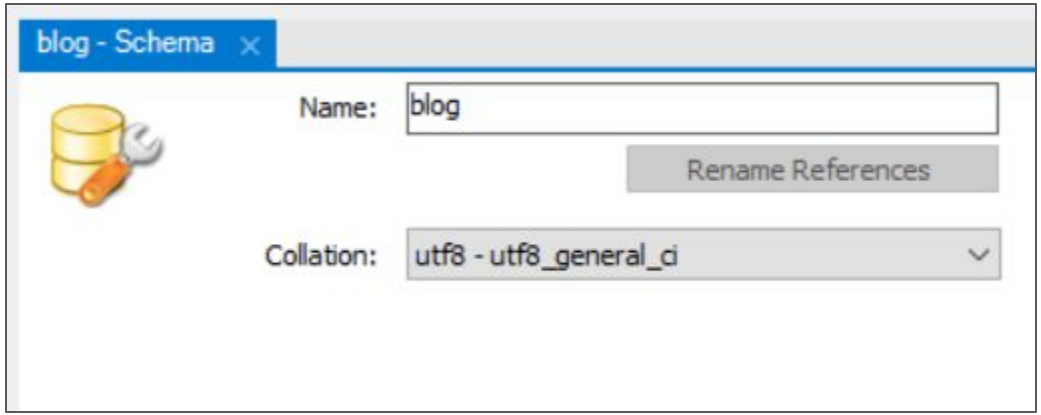
- Configuration should be located in ‘application.properties’ file
- Add following configuration to ‘application.properties’ file

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/blog
spring.datasource.username=root
spring.datasource.password=
```

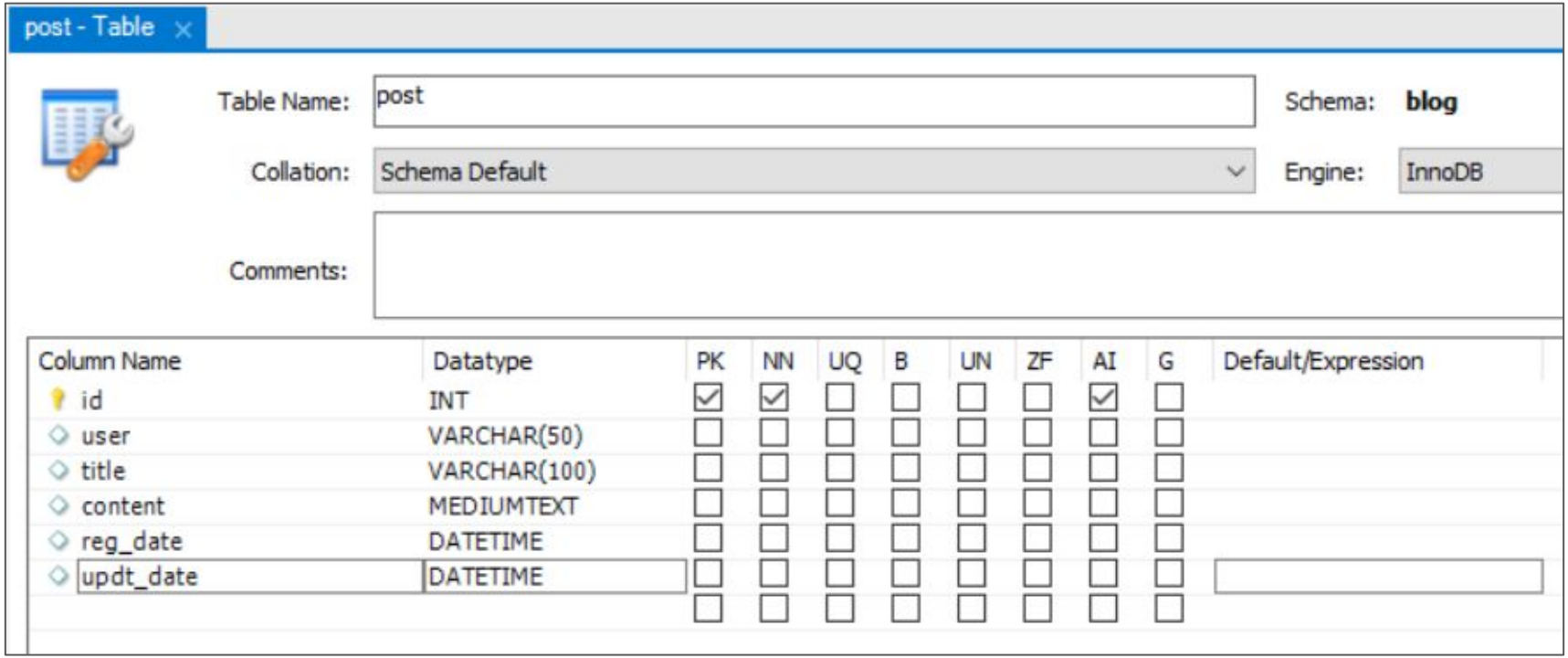
Test the Application

- Rerun the application
- If you don’t see any error message, Database configuration is completed

Create blog Schema

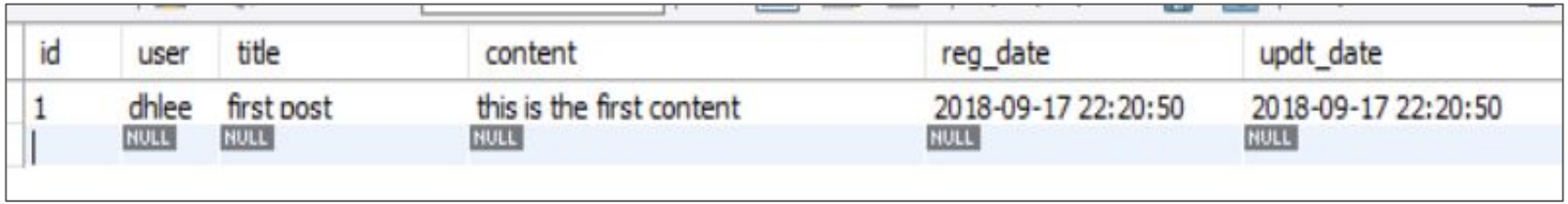


Create ‘post’ Table in the blog Schema



Insert Sample Data to post Table

- INSERT INTO `blog`.`post` (`user`, `title`, `content`, `reg_date`, `updt_date`) VALUES ('dhlee', 'first post', 'this is the first content', now(), now());



id	user	title	content	reg_date	updt_date
1	dhlee	first post	this is the first content	2018-09-17 22:20:50	2018-09-17 22:20:50
	NULL	NULL	NULL	NULL	NULL

Add dependencies

- Add dependencies to the pom.xml file
- Following dependencies should be located in <dependencies> tag

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

Configure Database

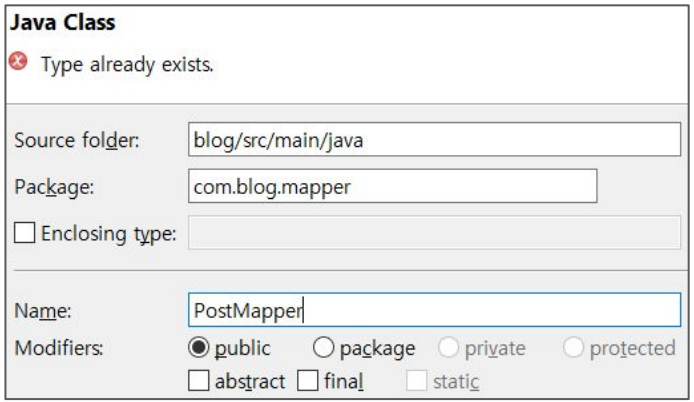
- Configuration should be located in ‘application.properties’ file
- Add following configuration to ‘application.properties’ file

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/blog
spring.datasource.username=root
spring.datasource.password=
```

Test the Application

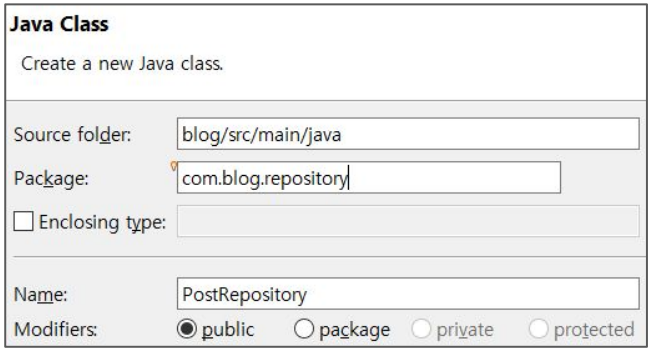
- Rerun the application
- If you don’t see any error message, Database configuration is completed

Create PostMapper Class




```java
import java.sql.ResultSet;
import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.blog.vo.Post;

public class PostMapper implements RowMapper<Post> {

    @Override
    public Post mapRow(ResultSet rs, int rowNum) throws SQLException {
        Post post = new Post();

        post.setId(rs.getLong("id"));
        post.setUser(rs.getString("user"));
        post.setTitle(rs.getString("title"));
        post.setContent(rs.getString("content"));
        post.setRegDate(rs.getDate("reg_date"));
        post.setUpdtDate(rs.getDate("updt_date"));

        return post;
    }

}
```

Create PostRepository Class




```java
package com.blog.repository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import com.blog.mapper.PostMapper;
import com.blog.vo.Post;

@Repository
public class PostRepository {

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public Post findOne(Long id) {
        String sql = "SELECT * FROM post WHERE id = ?";

        RowMapper<Post> rowMapper = new PostMapper();

        return this.jdbcTemplate.queryForObject(sql, rowMapper, id);
    }
}
```

Modify getPost Method in PostService

- Add postRepository field
- Modify getPost method

```
@Autowired
PostRepository postRepository;

public Post getPost(Long id) {
    Post post = postRepository.findOne(id);

    return post;
}
```

Modify getPost Method in PostController

```
@GetMapping("/post")
public Post getPost(@RequestParam("id") Long id) {
    Post post = postService.getPost(id);
    return post;
}
```

Test the Get Post API

- Rerun the application
- Call http://localhost:8080/post?id=1
- Check the result

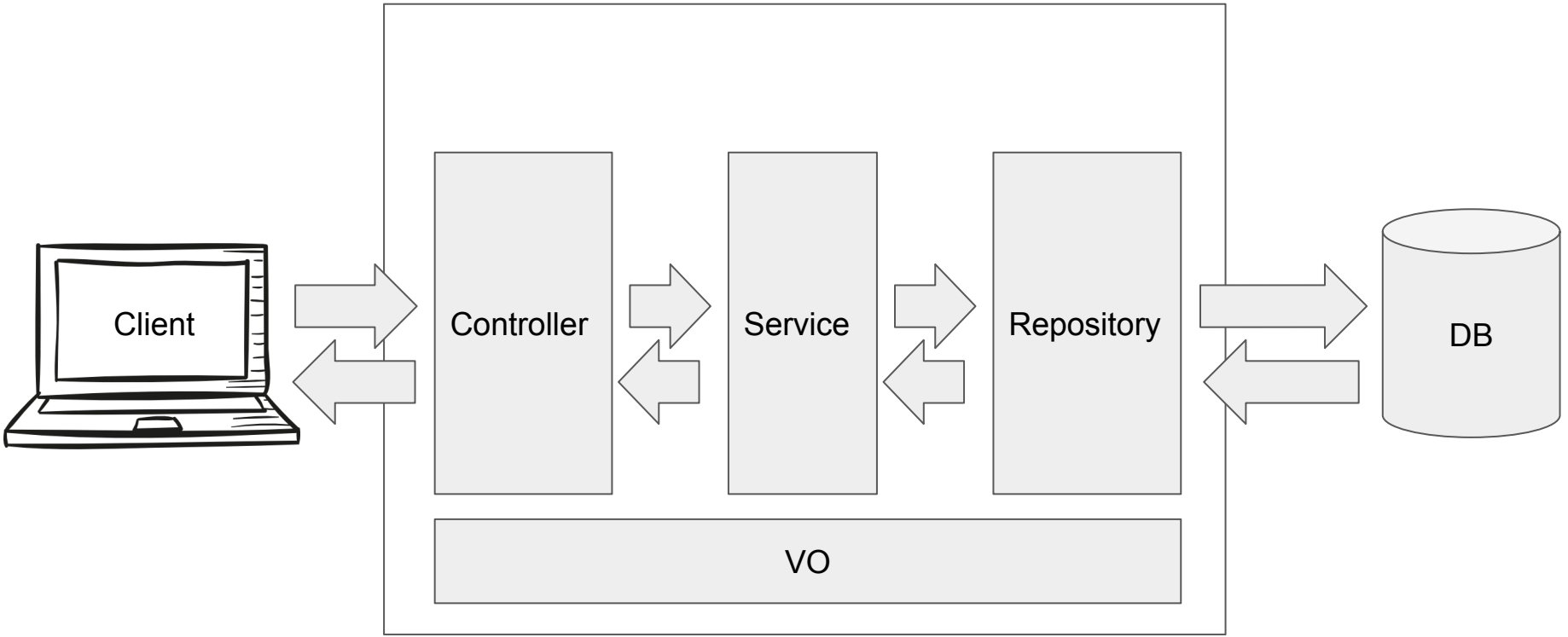
Mapper Class

- Mapper class is for mapping each table column to fields of java object
- JdbcTamplate can decide which column is mapped to proper field by using mapper class

Repository Class

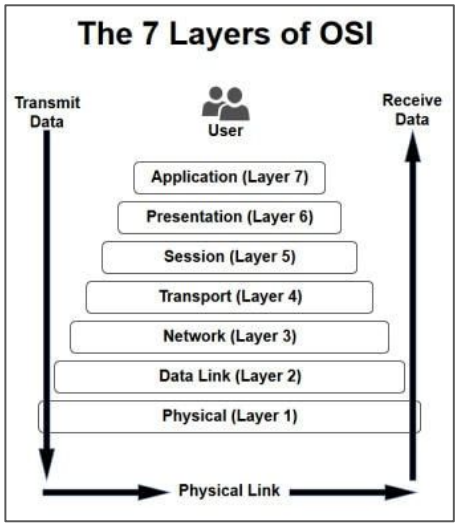
- Repository class is responsible for data access
- In other words, we call repository to dao class
- DAO stands for Data Accessing Object
- Repository doesn’t have any logic

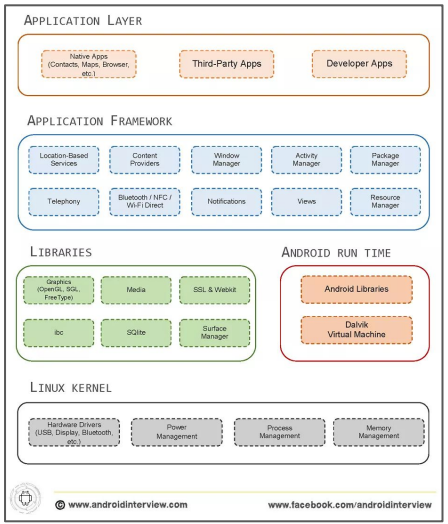
Entire Architecture



Layered Architecture

- Layered architecture/design is has SW layer for clear responsibility
- It’s The most common architecture pattern
- It’s De Facto standard for Java Spring Application
- It’s also called Multitier architecture





Role of Each Layer in Spring Application

- Controller layer is responsible for getting request and responding data
- Repository layer is responsible for accessing databases
- Service layer is responsible for business logic and error handling etc
- Another layer can be added to application
- But using 3 layer is really basic form of Spring application

Add a Method in PostRepository Class

```
public List<Post> findPost() {
    String sql = "SELECT * FROM post ORDER BY updt_date DESC";
    RowMapper<Post> rowMapper = new PostMapper();
    return this.jdbcTemplate.query(sql, rowMapper);
}
```

Modify a Method in PostService Class

```
public List<Post> getPosts() {
    List<Post> postList = postRepository.findPost();

    return postList;
}
```

Test the Get Posts API

- Rerun the application
- Call http://localhost:8080/posts
- Check the result

Insert Data to the post Table

```
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('jason', 'hi there', 'nice to meet you', '2018-09-22 22:50:57', '2018-09-22 22:50:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('mark', 'how are you', 'I matk nice to see you', '2018-09-22 22:51:57', '2018-09-22 22:51:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('mason', 'how you doing guys', 'I am doing great', '2018-09-22 22:52:57', '2018-09-22 22:52:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('jacob', 'very cold', 'It is really windy outside', '2018-09-22 22:53:57', '2018-09-22 22:53:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('elice', 'Hello~', 'have good day', '2018-09-22 22:54:57', '2018-09-22 22:54:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('elsa', 'nice to see you', 'Lets do funny things today', '2018-09-22 22:55:57', '2018-09-22 22:55:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('dustin', 'I am Dustin', 'I like pizza', '2018-09-22 22:56:57', '2018-09-22 22:56:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('justin', 'good to see you', 'have nice dinner', '2018-09-22 22:57:57', '2018-09-22 22:57:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('bill', 'Hi!!!', 'nice to meet you', '2018-09-22 22:58:57', '2018-09-22 22:58:57');
INSERT INTO blog.post (user, title, content, reg_date, updt_date) VALUES ('kathy', 'really hungry', 'I need to eat something', '2018-09-22 22:59:57', '2018-09-22 22:59:57');
```

Exercise - Make REST API

- Make REST API to get post list ordered by update date in ascending order
- Path is ‘/posts/updtdate/asc’
- SQL should be ‘SELECT * FROM post ORDER BY updt_date Asc’
- The name of Controller method will be ‘getPostsOrderByUpdtAsc’
- The name of Service method will be ‘getPostsOrderByUpdtAsc’
- The name of Repository method will be ‘findPostOrderByUpdtDateAsc’

Exercise - Make REST API

- Make REST API to get post list ordered by update date in ascending order
- Path is ‘/posts/regdate/desc’
- The name of Controller method will be ‘getPostsOrderByRegDesc’
- The name of Service method will be ‘getPostsOrderByRegDesc’
- The name of Repository method will be ‘findPostOrderByRegDateDesc’

Add new Method in PostRepository Class

```
public List<Post> findPostLikeTitle(String query) {
    String sql = "SELECT * FROM post WHERE title LIKE ?";
    RowMapper<Post> rowMapper = new PostMapper();
    return this.jdbcTemplate.query(sql, rowMapper, '%'+query+'%');
}
```

Add new Method in PostService Class

```
public List<Post> searchPostByTitle(String query) {
    List<Post> posts = postRepository.findPostLikeTitle(query);
    return posts;
}
```

Add new Method in PostController Class

```
@GetMapping("/posts/search/title")
public List<Post> searchByTitle(@RequestParam("query") String query) {
    List<Post> posts = postService.searchPostByTitle(query);
    return posts;
}
```

Test the Get Posts API

- Rerun the application
- Call http://localhost:8080/posts/search/title?query=how
- Check the result

Exercise - Make REST API for Searching Content

- Make REST API to search the post on the content
- Path is ‘/posts/search/content?query=nice
- The name of Controller method will be ‘searchByContent’
- The name of Service method will be ‘searchPostByContent’
- The name of Repository method will be ‘findPostLikeContent’

Add New Method in PostRepository Class

```
public int savePost(Post post) {
    String sql = "INSERT INTO post(user, title, content, reg_date, updt_date) VALUES(?,?,?,?,?)";
    return jdbcTemplate.update(sql, post.getUser(), post.getTitle(), post.getContent(), post.getRegDate(), post.getUpdtDate());
}
```

Add New Constructor in Post Class

```
public Post(String user, String title, String content) {
    this.user = user;
    this.title = title;
    this.content = content;
    this.regDate = new Date();
    this.updtDate = new Date();
}
```

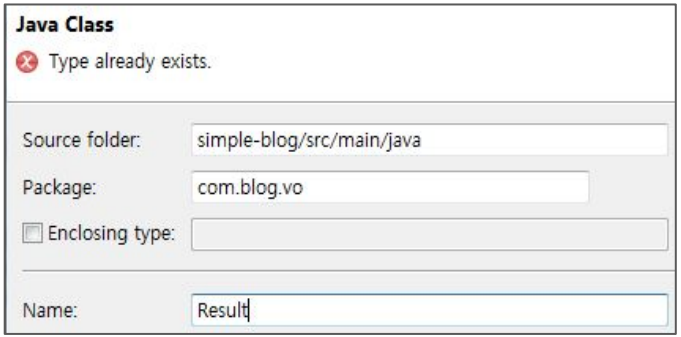
Add New Method in PostService Class

```
public boolean  savePost(Post post) {
    int result = postRepository.savePost(post);
    boolean isSuccess = true;

    if(result == 0) {
        isSuccess = false;
    }

    return isSuccess;
}
```

Add New Method in PostController Class

```
@PostMapping("/post")
public Object savePost(HttpServletResponse response, @RequestBody Post postParam)  {
    Post post = new Post(postParam.getUser(), postParam.getTitle(), postParam.getContent());
    boolean isSuccess = postService.savePost(post);

    if(isSuccess) {
        return new Result(200, "Success");
    } else {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        return new Result(500, "Fail");
    }
}
```

Create New Class Named ‘Result’

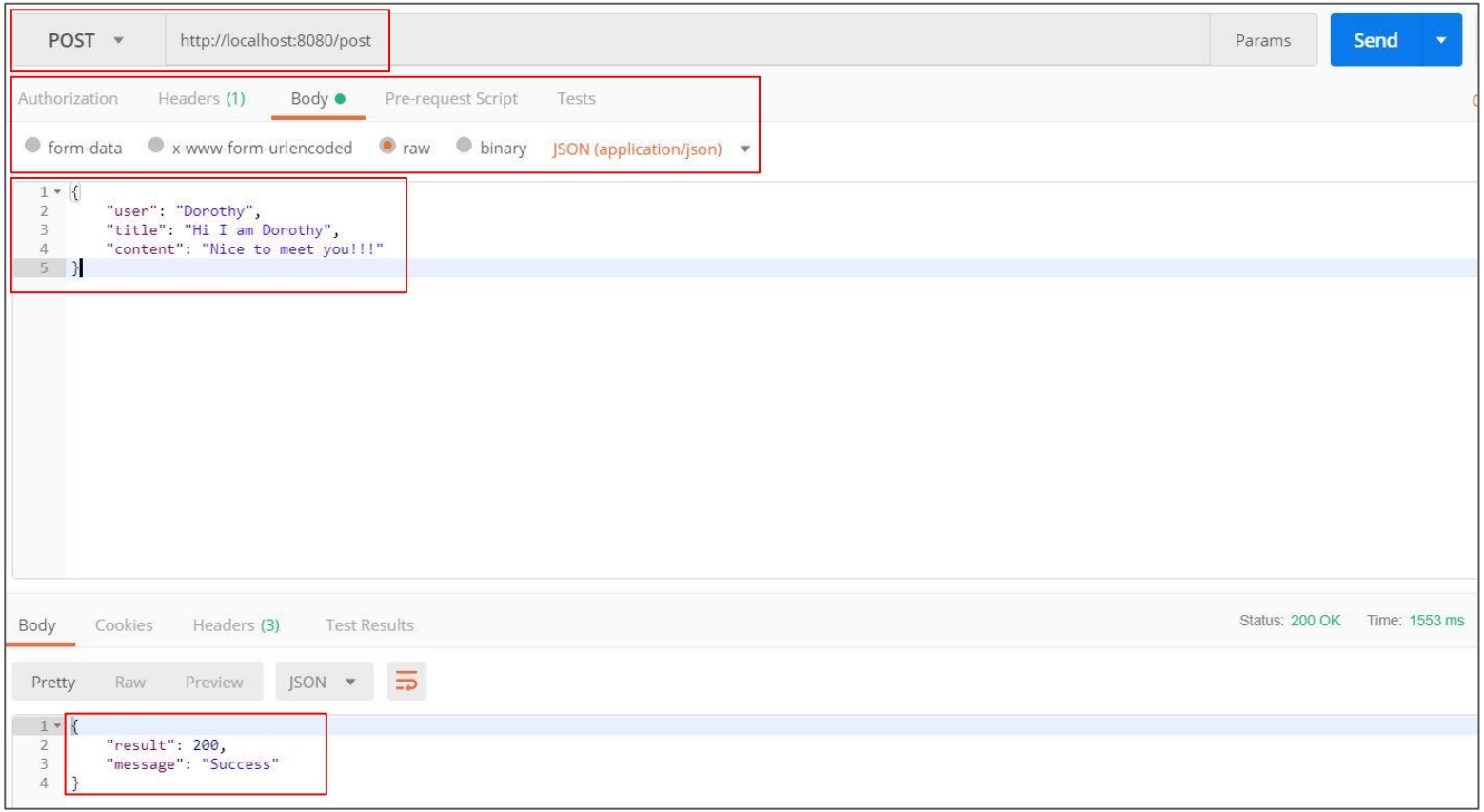



```java
package com.blog.vo;

public class Result {
    int result;
    String message;

    public Result() {
    }

    public Result(int result, String message) {
        this.result = result;
        this.message = message;
    }

    public int getResult() {
        return result;
    }

    public void setResult(int result) {
        this.result = result;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

}
```

Test new REST API

- Rerun the application
- Open the Postman

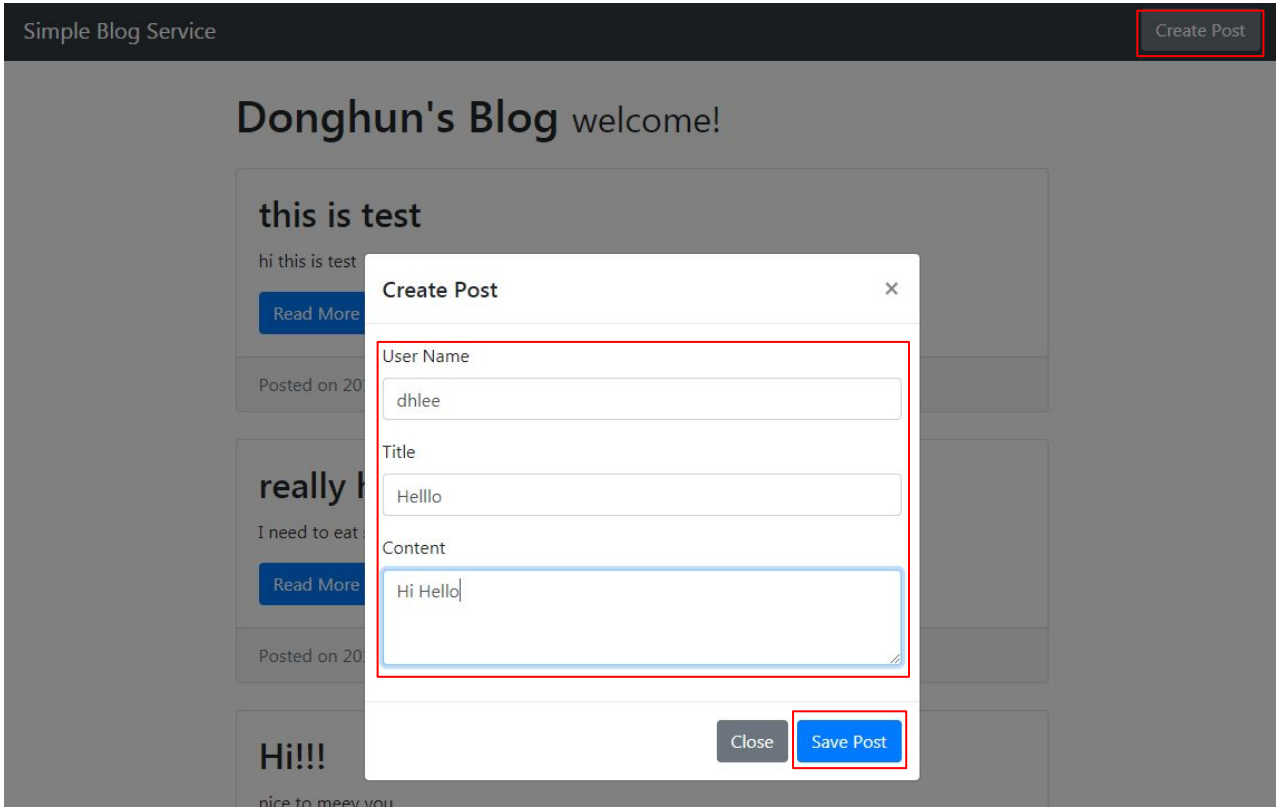
Test new REST API



Test new REST API

12	Dorothy	Hi I am Dorothy	Nice to meet you!!!	2018-09-26 13:19:03	2018-09-26 13:19:03

Test new REST API



Test new REST API

Add dependency for JPA

- Add dependency to the pom.xml file
- Following dependencies should be located in <dependencies> tag

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

Configure JPA & Hibernate

- Configuration should be located in ‘application.properties’ file
- Add following configuration to ‘application.properties’ file

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql = true
```

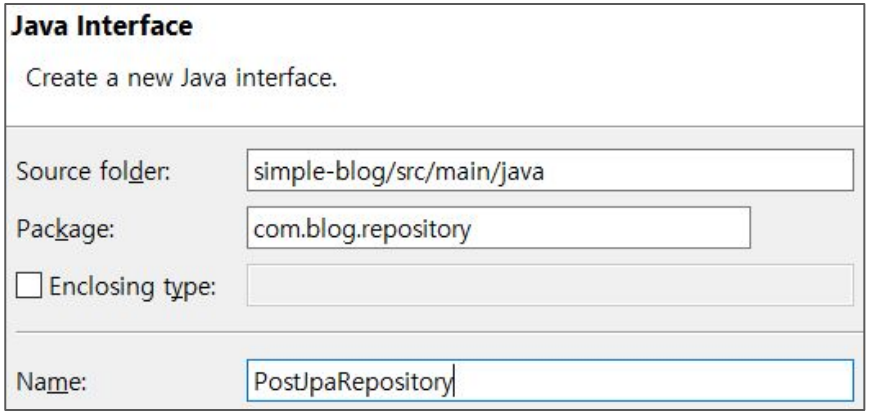
Modify Post Class

This is just part of Post class.

Don’t remove any Constructors and Getter/Setters

```java
@Entity
@Table(name = "post")
public class Post {

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

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

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

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

    @Column(name="regDate")
    private Date regDate;

    @Column(name="updtDate")
    private Date updtDate;
```

Create New PostJpaRepository Interface



Create New PostJpaRepository Interface

```java
package com.blog.repository;

import java.io.Serializable;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.blog.vo.Post;

@Repository("PostJpaRepository")
public interface PostJpaRepository extends JpaRepository<Post, Serializable> {
    Post findOneById(Long id);
}
```

Add a Field to PostService Class

```
@Autowired
PostJpaRepository jpaRepository;
```

Modify getPost Method in PostService Class

```
public Post getPost(Long id) {
    Post post = postJpaRepository.findOneById(id);

    return post;
}
```

Modify getPost Method in PostController Class

```
@GetMapping("/post")
public Post getPost(@RequestParam("id") Long id) {
    Post post = postService.getPost(id);
    return post;
}
```

Test the Get Post API

- Rerun the application
- Call http://localhost:8080/post?id=1
- Check the result
- Check Log in STS

```
Hibernate:
    select
        post0_.id as id1_0_,
        post0_.content as content2_0_,
        post0_.reg_date as reg_date3_0_,
        post0_.title as title4_0_,
        post0_.updt_date as updt_dat5_0_,
        post0_.user as user6_0_
    from
        post post0_
    where
        post0_.id=?
```

JPA and Hibernate

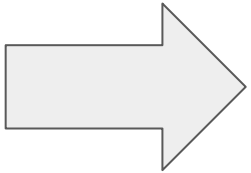
- JPA stands for Java Persistence API
- JPA is Abstraction or Specification for ORM(Object Relational Mapping) in Java
- Hibernate is one of the implementation of JPA
- There are another implementation for JPA
- In other words, Hibernate is an ORM framework in Java

ORM

- ORM stands for Object Relational Mapping
- We can think Relational is a Database Table
- ORM is technology for mapping the data of table to Java object

ORM in Post Table/Object

Column Name	Datatype
id	INT(11)
user	VARCHAR(50)
title	VARCHAR(100)
content	MEDIUMTEXT
reg_date	DATETIME
updt_date	DATETIME



```java
@Entity
@Table(name = "post")
public class Post {

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

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

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

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

    @Column(name="regDate")
    private Date regDate;

    @Column(name="updtDate")
    private Date updtDate;
```

Just Remember

- ORM is general technology for mapping database table to java object
- JPA specification for ORM for Java
- Hibernate is SW framework for ORM in Java
- Understanding JPA deeply is difficult, but using JPA is pretty easy

Magic of Hibernate

- Convert method name to real SQL automatically
- Map database table to java object automatically

Syntax of JPA

- Repository Class for using jpa
  - should be interface not class
  - extends JpaRepository
- Method name should be properly used
  - find - SELECT
  - One - LIMIT 1
  - WHERE id =

```
public interface PostJpaRepository extends JpaRepository<Post, Serializable> {
    Post findOneById(Long id);
}
```

Syntax of JPA

- findById : SELECT * FROM post WHERE id =
- deleteById : DELETE FROM post WHERE id =
- save : INSERT INTO post(...) VALUES(...)
- There is no method for updating data
    - Use save for updating

Add New Method in PostJpaRepository Interface

```
List<Post> findAllByOrderByUpdtDateDesc();
```

Modify getPosts Method in PostService Class

```
public List<Post> getPosts() {
    List<Post> posts = postJpaRepository.findAllByOrderByUpdtDateDesc();
    return posts;
}
```

Test the Get Posts API

- Rerun the application
- Call http://localhost:8080/posts
- Check the result
- Check Log in STS

```
Hibernate:
    select
        post0_.id as id1_0_,
        post0_.content as content2_0_,
        post0_.reg_date as reg_date3_0_,
        post0_.title as title4_0_,
        post0_.updt_date as updt_dat5_0_,
        post0_.user as user6_0_
    from
        post post0_
    order by
        post0_.updt_date desc
```

JPA Syntax for SQL Operators

And	findByUserAndTitle	… where user = ?1 and title = ?2
Or	findByUserOrTitle	… where user = ?1 or title = ?2
Between	findByRegDateBetween	… where regDate between ?1 and ?2
Like	findByTitleLike	… where title like ?1
Containing	findByTitleContaining	… where Title like ?1
OrderBy	findByUserOrderByRegDateDesc	… where user = ?1 order by regDate desc

Exercise

- Change the ‘/posts/updtdate/asc’ API to use JPA
- Create new JPA method corresponding following SQL in PostJpaRepository corresponding
  - ‘SELECT * FROM post ORDER BY updt_date ASC’
- Modify getPostsOrderByUpdtAsc() method to call the method you made in PostJpaRepository

```
Hibernate:
    select
        post0_.id as id1_0_,
        post0_.content as content2_0_,
        post0_.reg_date as reg_date3_0_,
        post0_.title as title4_0_,
        post0_.updt_date as updt_dat5_0_,
        post0_.user as user6_0_
    from
        post post0_
    order by
        post0_.updt_date asc
```

Modify savePost in PostService Class

```
public boolean  savePost(Post post) {
    Post result = jpaRepository.save(post);
    boolean isSuccess = true;

    if(result == null) {
        isSuccess = false;
    }

    return isSuccess;
}
```

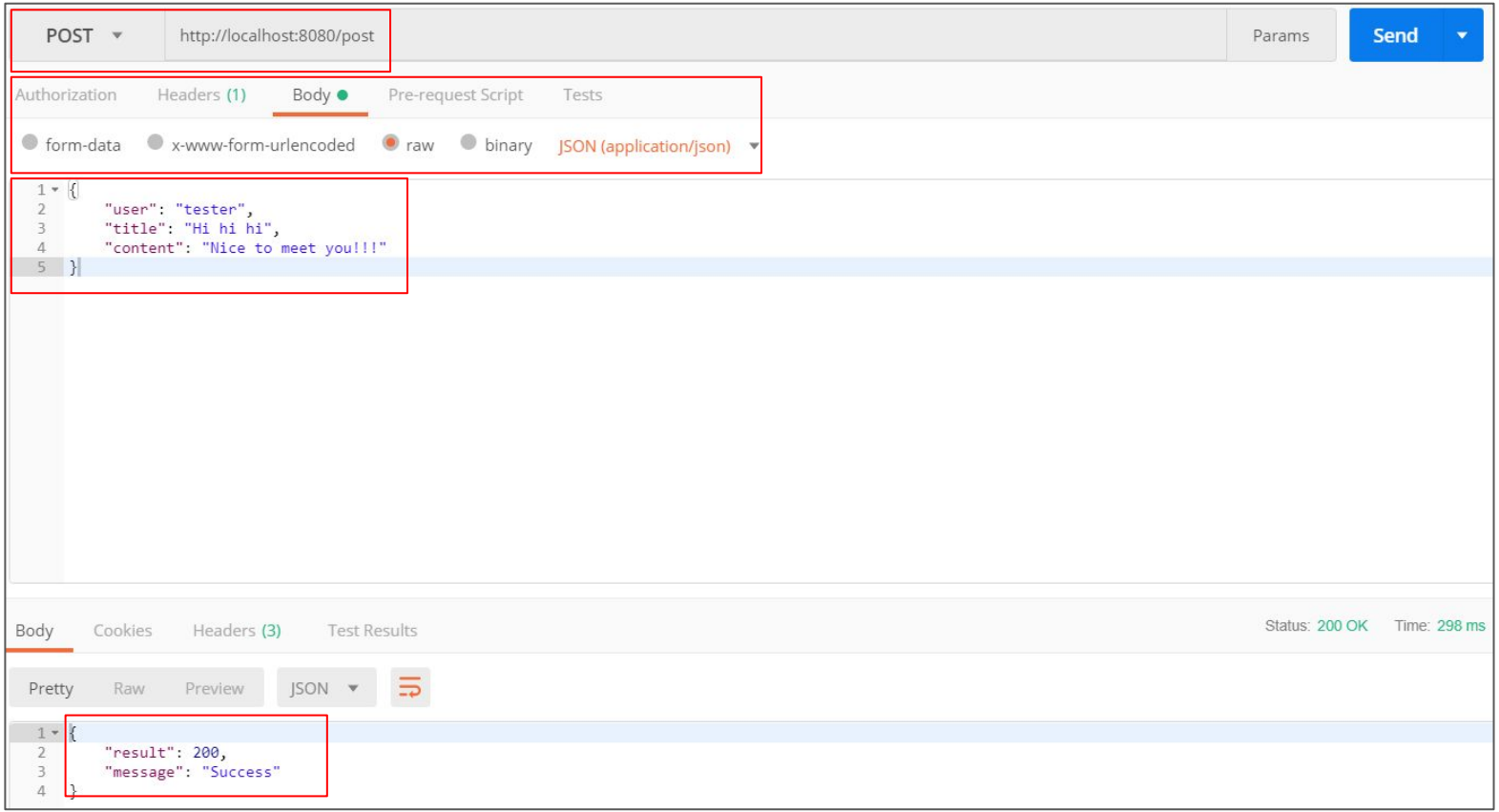
Add a Configuration to application.properties File

```
spring.jpa.properties.hibernate.id.new_generator_mappings=false
```

Test the API

- Rerun the application
- Open the Postman

Test the API

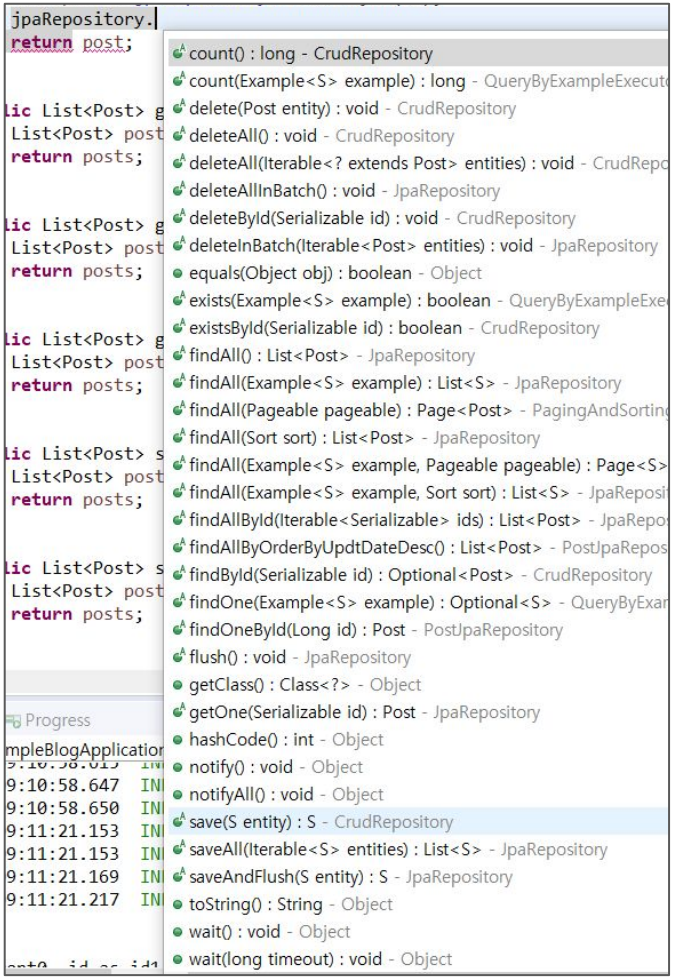

POST
http://localhost:8080/post
Params
Send
Authorization
Headers (1)
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"user": "tester",
"title": "Hi hi hi",
"content": "Nice to meet you!!!"
}
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 298 ms
Pretty
Raw
Preview
JSON
{
"result": 200,
"message": "Success"
}

Check the Log Message for Insert

```
Hibernate:
    insert
    into
        post
        (content, reg_date, title, updt_date, user)
    values
        (?, ?, ?, ?, ?)
```

JPA Magic

```java
@Repository("PostJpaRepository")
public interface PostJpaRepository extends JpaRepository<Post, Serializable> {
    Post findOneById(Long id);

    List<Post> findAllByOrderByUpdtDateDesc();
}
```



JPA Magic

- JPA Provide some useful methods
- It’s for
  - find*
  - save
  - delete*
  - count
- We can use these default method without adding new method

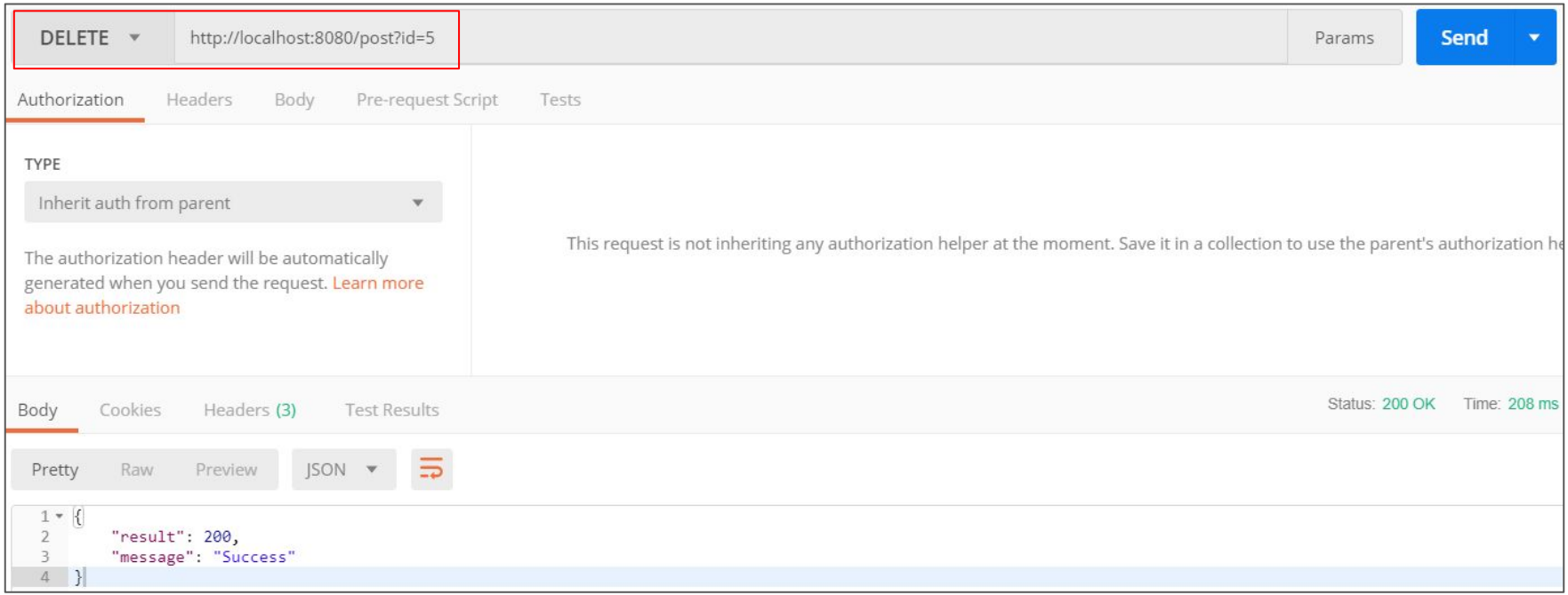
Add New Method in PostService Class

```
public boolean deletePost(Long id) {
    Post result = jpaRepository.findOneById(id);

    if(result == null)
        return false;

    jpaRepository.deleteById(id);
    return true;
}
```

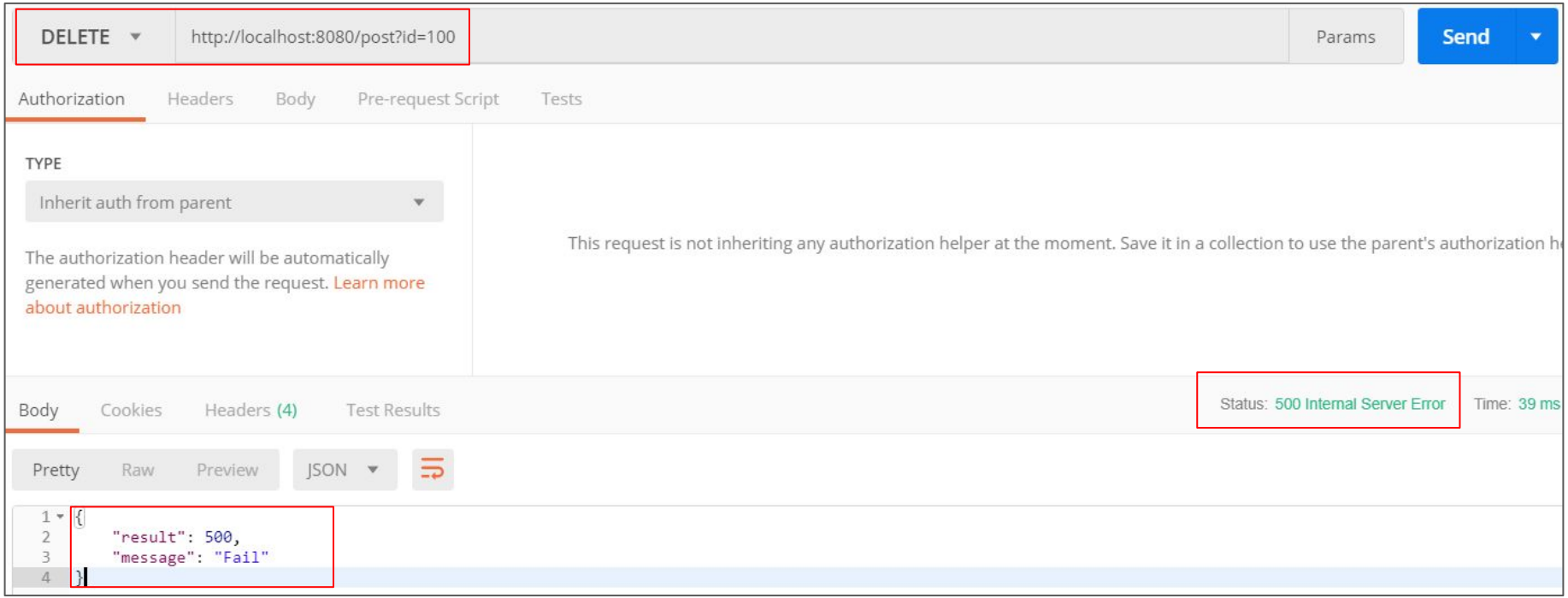
Add New Method in PostController Class

```
@DeleteMapping("/post")
public Object deletePost(HttpServletResponse response, @RequestParam("id") Long id)  {
    boolean isSuccess = postService.deletePost(id);

    log.info("id ::: " + id);

    if(isSuccess) {
        return new Result(200, "Success");
    } else {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        return new Result(500, "Fail");
    }
}
```

Test new REST API



Test new REST API



Check the Log Message for Delete

```
Hibernate:
    delete
    from
        post
    where
        id=?
```

Add New Constructor in Post Class

```
public Post(Long id, String title, String content) {
    super();
    this.id = id;
    this.title = title;
    this.content = content;
}
```

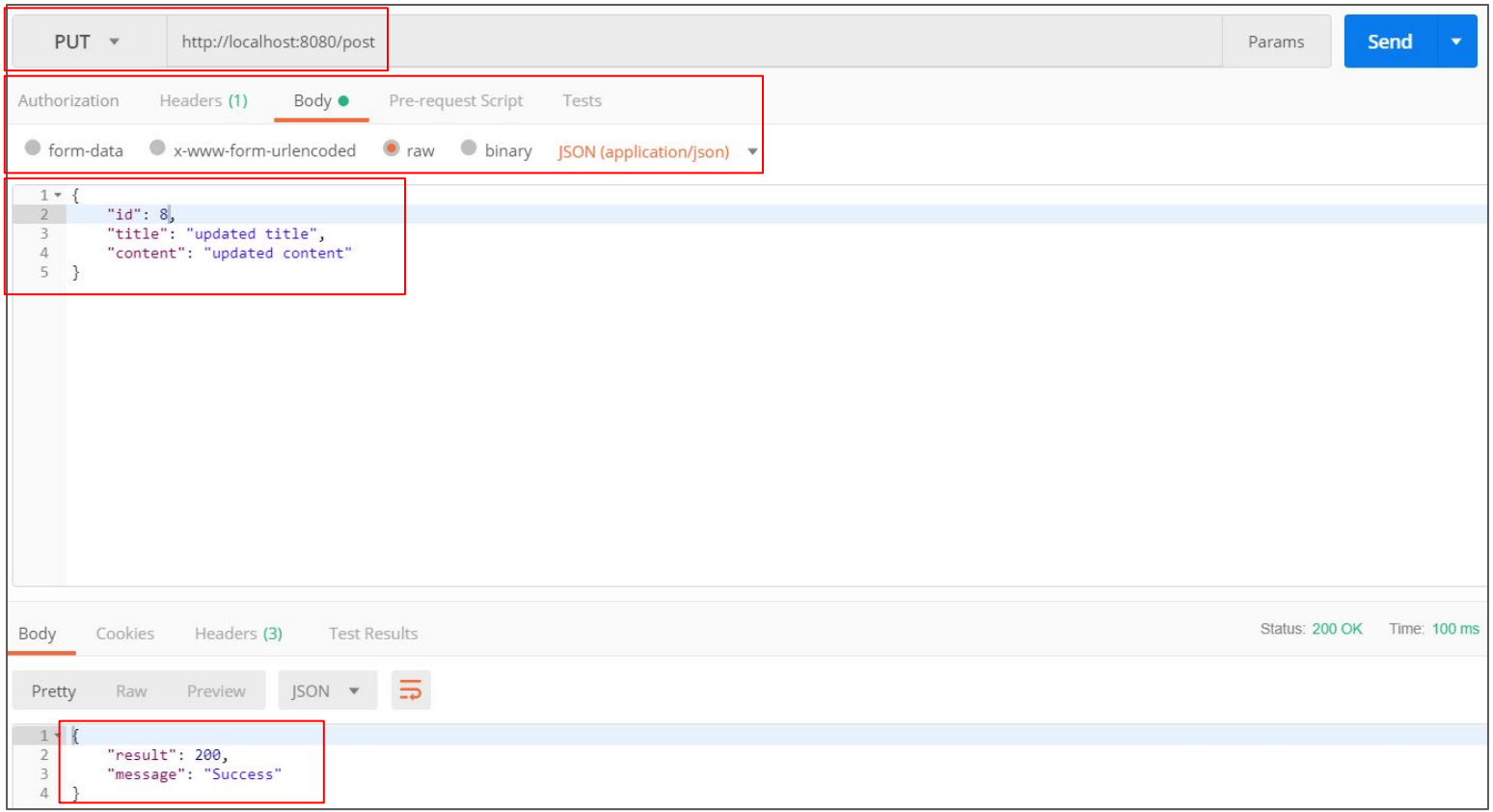
Add New Method in PostService Class

```
public boolean updatePost(Post post) {
    Post result = jpaRepository.findOneById(post.getId());

    if(result == null)
        return false;

    if(!StringUtils.isEmpty(post.getTitle())) {
        result.setTitle(post.getTitle());
    }

    if(!StringUtils.isEmpty(post.getContent())) {
        result.setContent(post.getContent());
    }

    jpaRepository.save(result);

    return true;
}
```

Add New Method in PostController

```
@PutMapping("/post")
public Object modifyPost(HttpServletResponse response, @RequestBody Post postParam)  {
    Post post = new Post(postParam.getId(), postParam.getTitle(), postParam.getContent());
    boolean isSuccess = postService.updatePost(post);

    if(isSuccess) {
        return new Result(200, "Success");
    } else {
        response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
        return new Result(500, "Fail");
    }
}
```

Test new REST API



Test new REST API

8	iustin	good to see you	have nice dinner	2018-09-22 22:57:57	2018-09-22 22:57:57

8	iustin	updated title	updated content	2018-09-22 22:57:57	2018-09-22 22:57:57

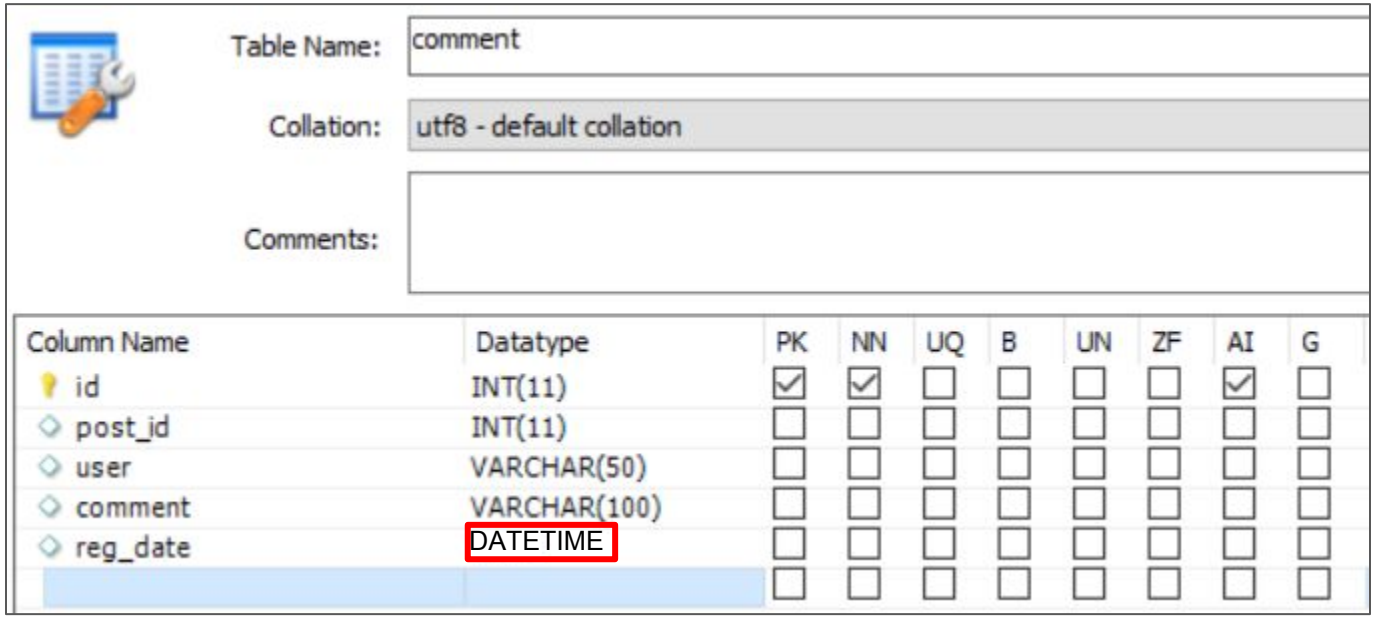
Check the Log Message for Update

```
Hibernate:
    update
        post
    set
        content=?,
        reg_date=?,
        title=?,
        updt_date=?,
        user=?
    where
        id=?
```

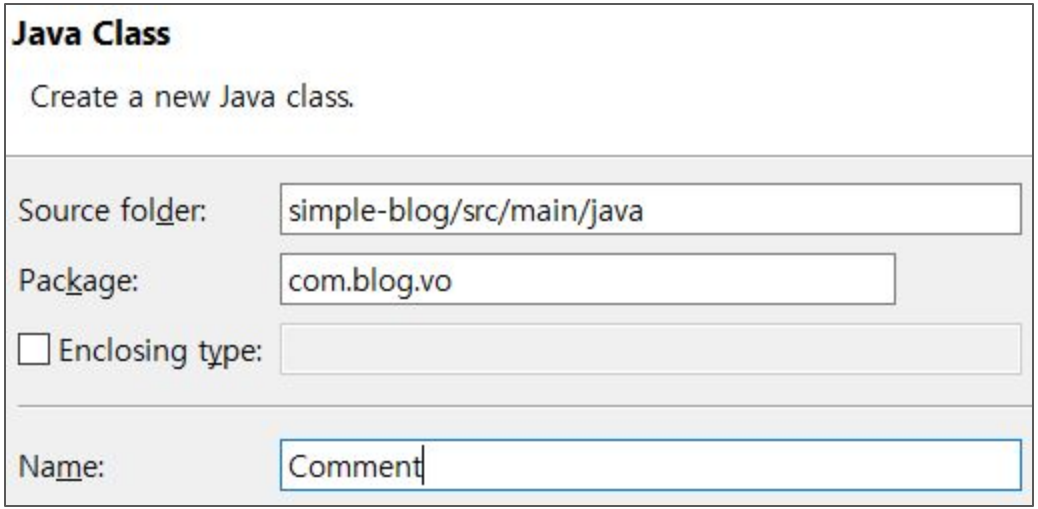
Error Case in REST API

- You should really care about the error case for REST API
- Many types of error case can be possible for REST API
- Client can handle error case if and only if you define error code for API and you document all error code in the documentation
- example
  - https://developers.facebook.com/docs/graph-api/using-graph-api/error-handling
  - https://docs.microsoft.com/en-us/rest/api/storageservices/common-rest-api-error-codes

Create New Table



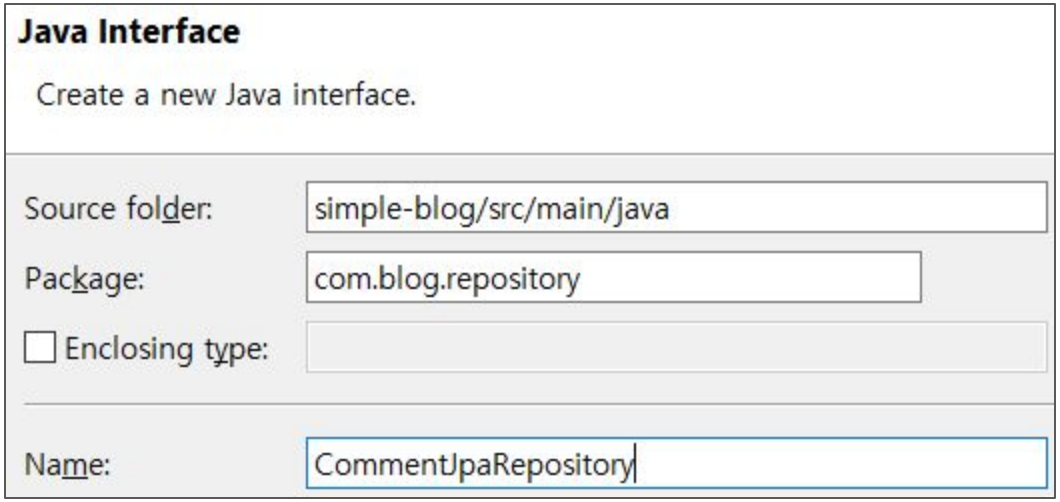
Create New Class



Comment Class

```java
package com.blog.vo;

import java.util.Date;

@Entity
@Table(name = "comment")
public class Comment {

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

    @Column(name="postId")
    private Long postId;

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

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

    @Column(name="regDate")
    private Date regDate;

    public Comment() {
    }

    public Comment(Long postId, String user, String comment) {
        this.postId = postId;
        this.user = user;
        this.comment = comment;
        this.regDate = new Date();
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Long getPostId() {
        return postId;
    }

    public void setPostId(Long postId) {
        this.postId = postId;
    }

    public String getUser() {
        return user;
    }

    public void setUser(String user) {
        this.user = user;
    }

    public String getComment() {
        return comment;
    }

    public void setComment(String comment) {
        this.comment = comment;
    }

    public Date getRegDate() {
        return regDate;
    }

    public void setRegDate(Date regDate) {
        this.regDate = regDate;
    }
}
```

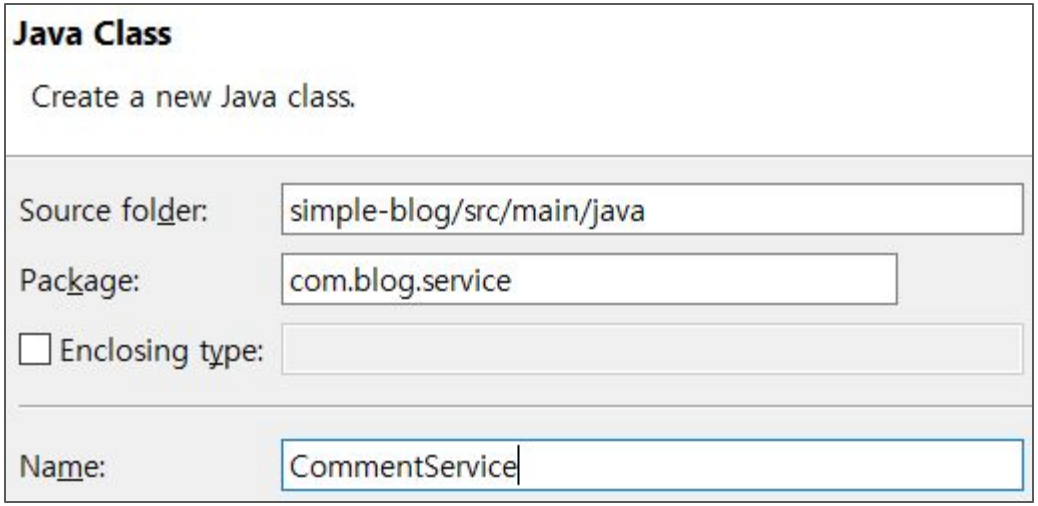
Create New Interface



CommentJpaRepository Interface

```java
package com.blog.repository;

import java.io.Serializable;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.blog.vo.Comment;

@Repository
public interface CommentJpaRepository  extends JpaRepository<Comment, Serializable> {

}
```

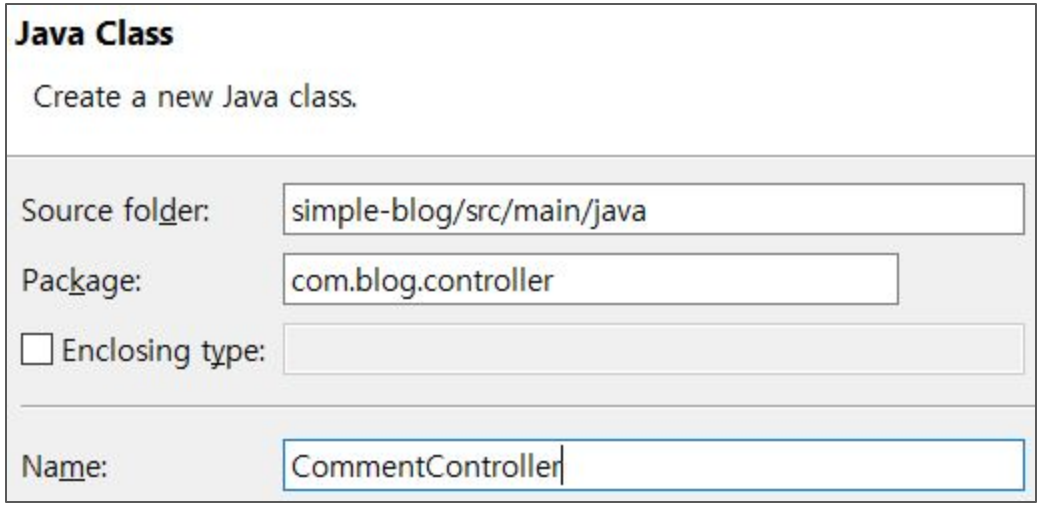
Create New Class



CommentService Class

```java
package com.blog.service;

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

import com.blog.repository.CommentJpaRepository;
import com.blog.vo.Comment;

@Service
public class CommentService {

    @Autowired
    CommentJpaRepository commentJpaRepository;

    public boolean  saveComment(Comment comment) {
        Comment result = commentJpaRepository.save(comment);
        boolean isSuccess = true;

        if(result == null) {
            isSuccess = false;
        }

        return isSuccess;
    }
}
```

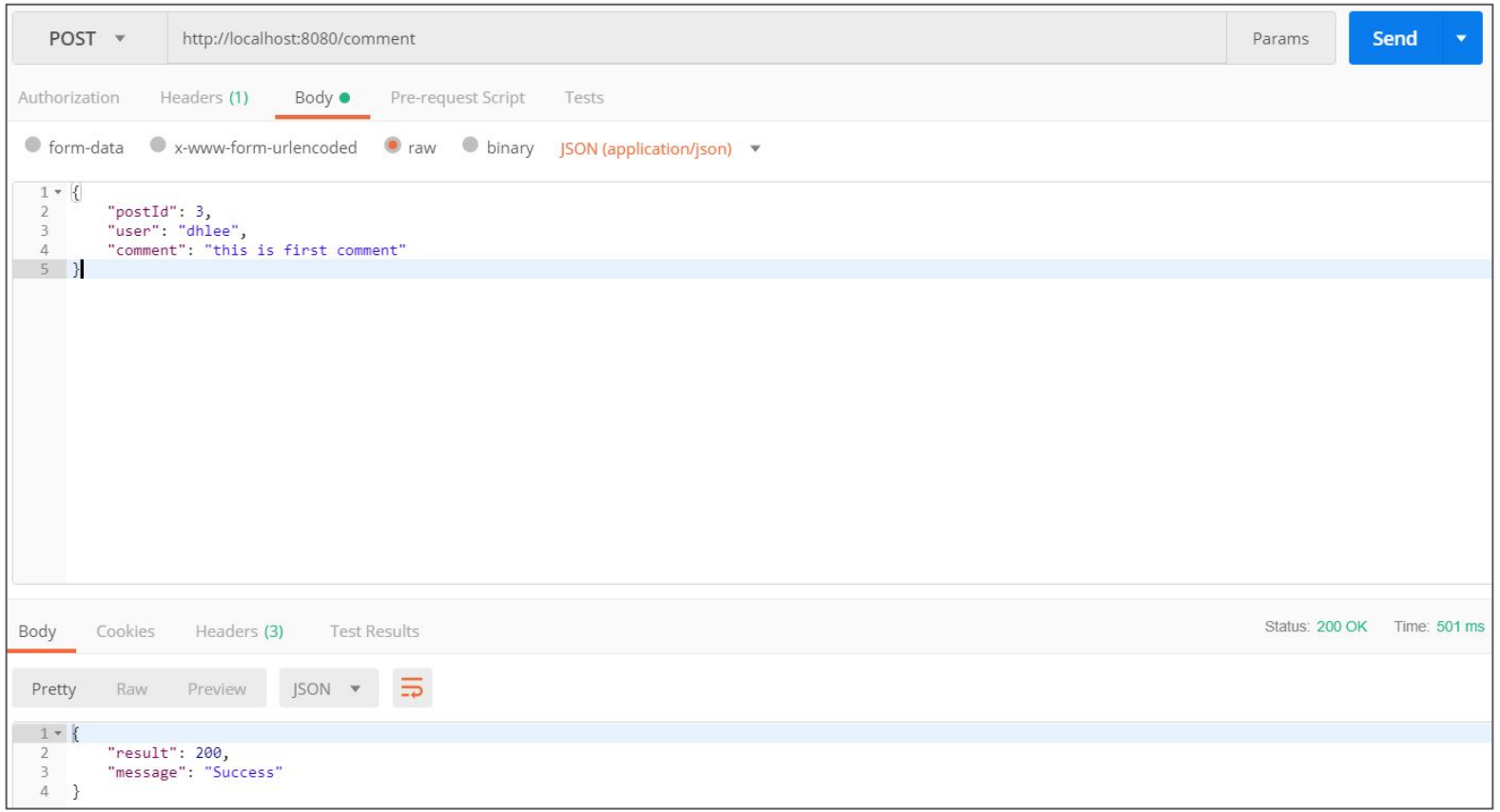
Create New Class



CommentController Class

```java
package com.blog.controller;

import javax.servlet.http.HttpServletResponse;

@RestController
public class CommentController {

    @Autowired
    CommentService commentService;

    @PostMapping("/comment")
    public Object savePost(HttpServletResponse response, @RequestBody Comment commentParam)  {
        Comment comment = new Comment(commentParam.getPostId(), commentParam.getUser(), commentParam.getComment());
        boolean isSuccess = commentService.saveComment(comment);

        if(isSuccess) {
            return new Result(200, "Success");
        } else {
            response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
            return new Result(500, "Fail");
        }
    }

}
```

Test New API


POST
http://localhost:8080/comment
Params
Send
Authorization
Headers (1)
Body
Pre-request Script
Tests
form-data
x-www-form-urlencoded
raw
binary
JSON (application/json)
{
"postId": 3,
"user": "dhlee",
"comment": "this is first comment"
}
Body
Cookies
Headers (3)
Test Results
Status: 200 OK
Time: 501 ms
Pretty
Raw
Preview
JSON
{
"result": 200,
"message": "Success"
}

Check Database and Log

id	post_id	user	comment	reg_date
1	3	dhlee	this is first comment	2018-09-27

```
Hibernate:
    insert
    into
        comment
        (comment, post_id, reg_date, user)
    values
        (?, ?, ?, ?)
```

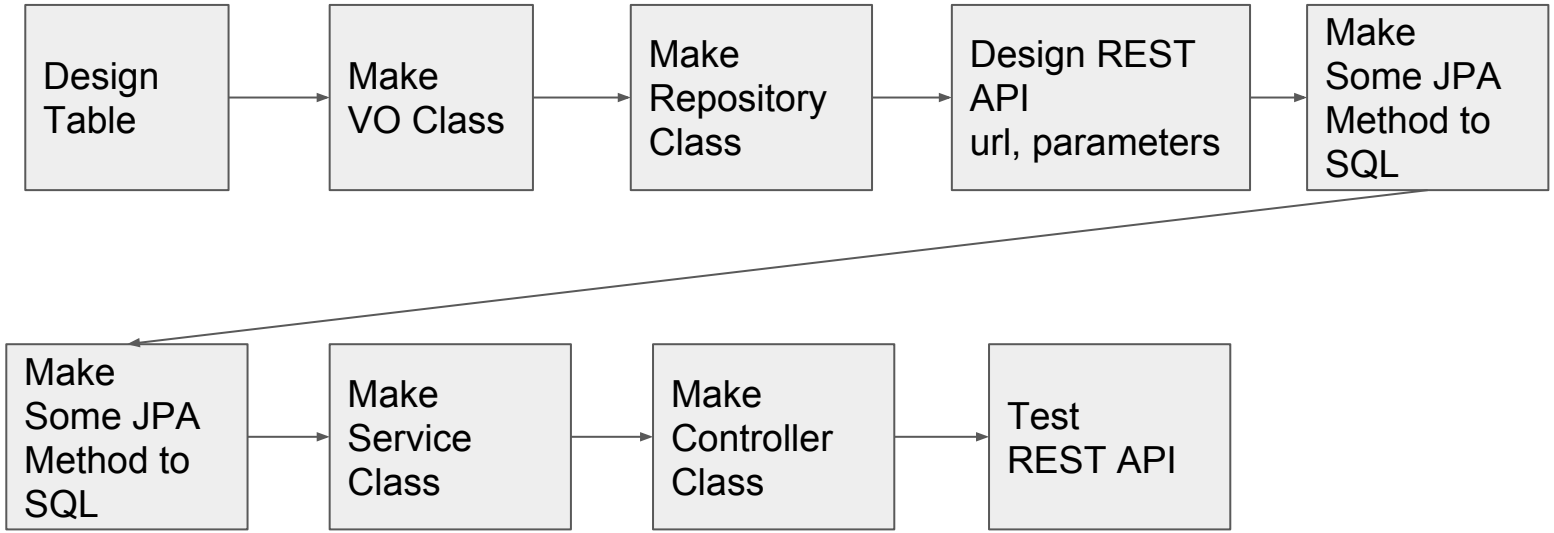
Database Table and Java Classes

- Commonly, there is one VO class per one database table
  - Post class for post table
  - Comment class for comment table
- Commonly, there is one Repository class per one VO class
  - PostJpaRepository for Post class
  - CommentJpaRepository for Comment Class
- When you add a new table in db, simply add a new VO class and new Repository class

Database Table and Java Classes

- For Controller class and Service Class,
- Basically we make one Controller class and Service Class for each VO class
- But using two or more Controller and Service for one VO class is also comment
- It’s because Controller and Service could be very long and big
  - long: class has too many line count
  - big: class has too many methods
- Long or Big Class is one of the bad code smell that we need to avoid

Process of Making REST API



Exercise

- Make REST API for
  - HTTP Method: GET
  - Path: ‘/comments?post_id=3’
- Add new method in CommentController
  - public List<Comment> getComments(@RequestParam("post_id") Long postId)
- Add new method in CommentService
  - public List<Comment> getCommentList(Long postId)
- Add new method corresponding following SQL in CommentJpaRepository
  - SELECT * FROM comment WHERE post_id = ? ORDER BY reg_date DESC

Exercise

- Make REST API for
  - HTTP Method: GET
  - Path: ‘/comment?id=1’
- Add new method in CommentController
  - public Object getComment(@RequestParam("id") Long id)
- Add new method in CommentService
  - public Comment getComment(Long id)
- Add new method corresponding following SQL in CommentJpaRepository
  - SELECT * FROM comment WHERE id=1
- Refer to ‘GET /post’ API, It’s very similar

Exercise

- Make REST API for
  - HTTP Method: DELETE
  - Path: ‘/comment?id=1’
- Add new method in CommentController
  - public Object deleteComment(HttpServletResponse response, @RequestParam("id") Long id)
- Add new method in CommentService
  - public boolean deleteComment(Long id)
  - Check whether the comment data that has id exists or not
- Add new method corresponding following SQL
  - DELETE FROM comment WHERE id=1
- Refer to ‘DELETE /post’ API, It’s very similar

Add New Method in PostJpaRepository Interface

```
List<Post> findByTitleContainingOrderByUpdtDateDesc(String query);
```

Modify New Method in PostService Class

```
public List<Post> searchPostByTitle(String query) {
    List<Post> posts = jpaRepository.findByTitleContainingOrderByUpdtDateDesc(query);
    return posts;
}
```

Test the Modified API

- Call http://localhost:8080/posts/search/title?query=how
- Check the result and sql log in STS

```
Hibernate:
    select
        post0_.id as id1_1_,
        post0_.content as content2_1_,
        post0_.reg_date as reg_date3_1_,
        post0_.title as title4_1_,
        post0_.updt_date as updt_dat5_1_,
        post0_.user as user6_1_
    from
        post post0_
    where
        post0_.title like ?
    order by
        post0_.updt_date desc
```

Exercise

- Modify existing ‘/posts/search/content’ API to use JPA method
- Imitate new ‘/posts/search/title’ API

Exercise(Advanced)

- Create new REST API for searching comment
  - HTTP Method: GET
  - URI Path: /comments/search?post_id=3&query=hi
- Create methods in Controller, Service, Repository for yourself
- SQL should be following thing

```
Hibernate:
    select
        comment0_.id as id1_0_,
        comment0_.comment as comment2_0_,
        comment0_.post_id as post_id3_0_,
        comment0_.reg_date as reg_date4_0_,
        comment0_.user as user5_0_
    from
        comment comment0_
    where
        comment0_.post_id=?
        and (
            comment0_.comment like ?
        )
    order by
        comment0_.reg_date desc
```