**Annotation Based Web Service**

Here we are using REST API, as SOAP is outdated. Some of the annotation we need to familiarize before going further with this topic:

**@RestController** : First of all, we are using Spring 4’s new @RestController annotation. This annotation eliminates the need of annotating each method with @ResponseBody. Under the hood, @RestController is itself annotated with @ResponseBody, and can be considered as combination of @Controller and @ResponseBody.

**@RequestBody** : If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter. While doing that, Spring will [behind the scenes] use [HTTP Message converters](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#rest-message-conversion) to convert the HTTP request body into domain object [deserialize request body to domain object], based on ACCEPT or Content-Type header present in request.

**@ResponseBody** : If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will [behind the scenes] use [HTTP Message converters](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#rest-message-conversion) to convert the return value to HTTP response body [serialize the object to response body], based on Content-Type present in request HTTP header. As already mentioned, in Spring 4, you may stop using this annotation.

**ResponseEntity** is a real deal. It represents the entire HTTP response. Good thing about it is that you can control anything that goes into it. You can specify status code, headers, and body. It comes with several constructors to carry the information you want to sent in HTTP Response.

**@PathVariable** This annotation indicates that a method parameter should be bound to a URI template variable [the one in ‘{}’].

Basically, @RestController , @RequestBody, ResponseEntity & @PathVariable are all you need to know to implement a REST API in Spring 4. Additionally, spring provides several support classes to help you implement something customized.

**MediaType :** With @RequestMapping annotation, you can additionally, specify the MediaType to be produced or consumed (using **produces** or **consumes** attributes) by that particular controller method, to further narrow down the mapping.

**@GetMapping** is specialized version of [@RequestMapping](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html) annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). [@GetMapping](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/GetMapping.html) annotated methods handle the HTTP GET requests matched with given URI expression. E.g.

|  |
| --- |
| @GetMapping("/home")  public String homeInit(Model model) {  return "home";  }    @GetMapping("/members/{id}")  public String getMembers(Model model) {  return "member";  } |

**@PostMapping** is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST). [@PostMapping](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html) annotated methods handle the HTTP POST requests matched with given URI expression. e.g.

|  |
| --- |
| @PostMapping(path = "/members", consumes = "application/json", produces ="application/json")  public void addMember(@RequestBody Member member) {  //code  } |

## **@PostMapping vs @RequestMapping**

As noted above @PostMapping annotation is one specialized version of @RequestMappingannotation which handle HTTP POST requests.

@PostMapping acts as a shortcut for @RequestMapping(method = RequestMethod.POST).

|  |
| --- |
| Ex:  @Target({ java.lang.annotation.ElementType.METHOD })  @Retention(RetentionPolicy.RUNTIME)  @Documented  @RequestMapping(method = { RequestMethod.POST })  public @interface PostMapping  {  //code  } |

Passing URL information is same in both annotations.

Let’s see the difference between PostMapping and @RequestMapping annotations with simple code.

|  |
| --- |
| @RequestMapping(value = "/employees", method = RequestMethod.POST) //1    @PostMapping("/employees") //2 |

**@PutMapping** is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.PUT)

Ex:

@PutMapping("/{userId}")

public String updateUser(@PathVariable String userId, @RequestBody UserDetailsRequestModel requestUserDetails)

{

return "HTTP PUT was called";

}

**@PatchMapping** is a composed annotation that acts as a shortcut for RequestMapping(method = RequestMethod.PATCH).

Ex:

@PatchMapping(value="/{id}", headers="Accept=application/json")

public ResponseEntity<User> updateUserPartial(@PathVariable("id") int id, @RequestBody User currentUser)

{

User user = userService.findById(id);

if(user ==null)

{

return new ResponseEntity<User>(HttpStatus.NOT\_FOUND);

}

userService.updatePartially(currentUser, id);

return new ResponseEntity<User>(user, HttpStatus.OK);

}

**@DeleteMapping** is a composed annotation that acts as a shortcut for RequestMapping(method = RequestMethod.DELETE).

Ex:

@DeleteMapping("/{userId}")

public String deleteUser(@PathVariable String userId)

{

return "HTTP DELETE was called";

}

**@RestController:** It is used at class level to make RESTful web service endpoints. **@RestController** is combination of @Controller and @ResponseBody.

**@CrossOrigin:** It is used for CORS support that permits cross-origin requests on class level as well as method level.

**@RequestMapping:** It maps web requests onto methods in REST web service endpoints to provide flexible method signature.

**@GetMapping:** It is @RequestMapping with HTTP GET method.

**@PostMapping:** It is @RequestMapping with HTTP POST method.

**@PutMapping:** It is @RequestMapping with HTTP PUT method.

**@DeleteMapping:** It is @RequestMapping with HTTP DELETE method.

**@PatchMapping:** It is @RequestMapping with HTTP PATCH method.

**@PathVariable:** It indicates that a method parameter should be bound to a URI template variable.

**@RequestBody:** It is used with the method parameter to bind the body of the web request.

**@RequestParam:** It is used with method parameter to bind the web request parameter.

**ResponseEntity:** It is the extension of HttpEntity that represents HTTP request or response entity, consisting of headers and body.

**UriComponentsBuilder:** It is the builder for UriComponents that represents an immutable collection of URI components.

The annotations @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping and @PatchMapping are having optional elements as following.

**consumes:** It defines an array of consumable media types of mapped request.

**produces:** It defines an array of producible media types of mapped request.

**headers:** It defines the acceptable headers of mapped request.

**params:** It defines the parameters of the mapped request, narrowing the primary mapping.

**path:** It defines path mapping URIs in servlet environment.

**name:** It assigns a name to this mapping.

**value:** It defines primary mapping expressed by this annotation.