Controller Layer Annotations in Spring Boot

The controller layer in Spring Boot is responsible for handling HTTP requests, processing business logic, and returning responses to clients. Spring provides various annotations to simplify controller development.

1. @Controller Annotation

- Marks a class as a Spring MVC controller.
- It is a specialized version of @Component, allowing Spring to detect it during component scanning.
- Typically used for applications that return views (e.g., JSP, Thymeleaf).
- Works with @RequestMapping to define request-handling methods.

When to Use?

Use @Controller when building a web application that returns views instead of raw data (e.g., HTML pages using Thymeleaf).

---

2. @RestController Annotation

- A combination of @Controller and @ResponseBody.
- Used in RESTful web services to return JSON or XML responses.
- Eliminates the need to explicitly annotate methods with @ResponseBody.

Example Use Case:

A REST API that returns JSON data instead of rendering a webpage.

Key Benefits:

✅ Reduces boilerplate code.
✅ Automatically converts Java objects into JSON/XML.
✅ Simplifies building REST APIs.

---

3. @ResponseBody Annotation

- Converts the return value of a method into the response body instead of resolving it as a view name.
- Used with @Controller to return JSON or XML responses.
- When @RestController is used, explicit @ResponseBody is unnecessary.

Example Use Case:

If a method returns a User object, @ResponseBody will serialize it to JSON/XML and send it as the HTTP response.

Without @ResponseBody

Spring interprets the return value as a view name and tries to render it.

4. @RequestMapping Annotation

- Maps HTTP requests to handler methods in a controller.
- Can be applied at the class or method level.
- Allows specifying:
  - value or path → The request URL pattern.
  - method → The HTTP method (GET, POST, PUT, DELETE).
  - consumes → Expected request content type (e.g., application/json).
  - produces → Response content type (e.g., application/json).

Key Features:

✅ Supports multiple HTTP methods.
✅ Can be used on both class and method levels.
✅ Allows defining request headers and parameters.

Example:

```
@RequestMapping(path = "/fetchUser", method = RequestMethod.GET)
public User fetchUserDetails() {
    return new User("John", "Doe", "john.doe@example.com");
}
```

Note: @GetMapping, @PostMapping, @PutMapping, and @DeleteMapping are specialized shortcuts for @RequestMapping.

---

5. @RequestParam Annotation

- Extracts query parameters from the request URL.
- Used to bind HTTP request parameters to method arguments.
- Can specify:
  - name → The request parameter name.
  - required → If false, the parameter is optional.
  - defaultValue → A fallback value if the parameter is missing.

Example:

Request URL:
http://localhost:8080/api/fetchUser?firstName=John&age=30

```
@GetMapping("/fetchUser")
public String getUserDetails(
    @RequestParam(name = "firstName") String firstName,
    @RequestParam(name = "age", required = false, defaultValue = "25") int age
) {
    return "Name: " + firstName + ", Age: " + age;
}
```

Key Benefits:

✅ Handles optional parameters without errors.
✅ Supports default values.
✅ Automatically converts types (e.g., String to int).

6. PropertyEditor in Spring Boot

- Used to customize property binding in Spring.
- Converts request parameter values before they are assigned to method parameters.
- Registered using @InitBinder.

Use Case:

If you want all firstName values to be converted to lowercase before processing.

```
@InitBinder
protected void initBinder(DataBinder binder) {
    binder.registerCustomEditor(String.class, "firstName", new FirstNamePropertyEditor());
}
public class FirstNamePropertyEditor extends PropertyEditorSupport {
    @Override
    public void setAsText(String text) throws IllegalArgumentException {
        setValue(text.trim().toLowerCase());
    }
}
```

---

7. @PathVariable Annotation

- Extracts values from the request URI path and binds them to method parameters.
- Commonly used for REST APIs where parameters are part of the URL.

Example:

Request URL:

http://localhost:8080/api/fetchUser/John

```
@GetMapping(path = "/fetchUser/{firstName}")
public String getUserDetails(@PathVariable(value = "firstName") String firstName) {
    return "Fetching user details for: " + firstName;
}
```

Key Benefits:

✅ Allows more readable and RESTful URLs.

✅ Automatically converts path variables to required types.

---

8. @RequestBody Annotation

- Binds the request body to a method parameter (usually a Java object).
- Used to receive JSON or XML data in POST and PUT requests.
- Spring automatically deserializes the request body into a Java object.

Example:

```
@PostMapping("/saveUser")
public String saveUser(@RequestBody User user) {
    return "User " + user.getUsername() + " saved successfully!";
}
```

Key Benefits:

✅ Automatically converts JSON into Java objects.

✅ Simplifies handling of request payloads.

---

9. ResponseEntity

- Represents the complete HTTP response (body, status code, headers).
- Gives fine-grained control over the response.

Example:

```
@GetMapping("/fetchUser")
public ResponseEntity<String> getUserDetails(@RequestParam(value = "firstName") String firstName) {
   String output = "Fetched user details of " + firstName;
   return ResponseEntity.status(HttpStatus.OK).body(output);
}
```

Key Benefits:

✅ Allows setting custom HTTP status codes.

✅ Supports adding custom headers.

✅ Can return different responses based on conditions.

Final Thoughts

These Spring Boot controller annotations make it easy to build RESTful APIs and web applications. Here's a quick summary:

Annotation	Purpose
@Controller	Defines a Spring MVC controller (for web applications with views).
@RestController	Defines a RESTful API controller (returns JSON/XML).
@ResponseBody	Serializes return values to JSON/XML instead of resolving views.
@RequestMapping	Maps HTTP requests to specific controller methods.
@RequestParam	Binds query parameters to method arguments.
@PathVariable	Extracts values from the URL path.
@RequestBody	Binds request JSON/XML payloads to method parameters.
ResponseEntity	Provides complete HTTP response control.
@InitBinder	Registers custom property editors for data binding.

Each of these annotations simplifies different aspects of handling requests and responses in Spring Boot. Let me know if you need further clarification!