EXCLUSIVE

SPRING WEB MVC CHEAT SHEET

QUICK REFERENCE TO SOLVE RE-OCCURRING TASKS FOR YOUR ENDPOINTS LIKE VALIDATION, CONTENT-NEGOTIATION, FILE-HANDLING, ETC.

CONTENTS

SPRING WEB MVC CHEAT SHEET

How can I return JSON payload from a Spring MVC endpoint?

```
@RestController
@RequestMapping("/json")
public class JsonPayloadController {

  @GetMapping(path = "/orders", produces = MediaType.APPLICATION_JSON_VALUE)
  public List<Order> getOrders() {
    List<Order> orders = List.of(
      new Order("42L", Set.of("foreign", "books"), 1L, LocalDateTime.now()),
      new Order("58B", Set.of("fractile", "computer"), 48L, LocalDateTime.now()));
    return orders;
  }
}
```

How to return a different HTTP status for a Spring MVC controller?

```
@GetMapping(path = "/orders", produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<List<Order>> getOrders() {
  List<Order> orders = List.of(
    new Order("42L", Set.of("foreign", "books"), 1L, LocalDateTime.now()),
    new Order("58B", Set.of("fractile", "computer"), 48L, LocalDateTime.now())
  );
  return ResponseEntity.status(HttpStatus.I_AM_A_TEAPOT).body(orders);
}
```

How can I validate the incoming payload to my controller?

```
@Validated
@RestController
@RequestMapping("/validation")
public class ValidationController {

  @GetMapping("/{message}")
  public String validateParameters(
   @PathVariable("message") @Size(min = 5, max = 10) String message,
   @RequestParam("size") @Positive Long size) {
     return "Query parameters where valid";
  }
}
```

The example above will validate that the path variable has between 5 and 10 characters and that the query parameter size is a positive number. Sending an invalid request to this endpoint, like:

```
curl -v http://localhost:8080/validation/foo?size=-42
```

results by default in an HTTP status 500. If you want to change this behavior, you can add an *ExceptionHandler* to catch the *ConstraintViolationException* and return HTTP status 400 (Bad Request) instead:

```
@ExceptionHandler(ConstraintViolationException.class)
@ResponseStatus(HttpStatus.BAD_REQUEST)
ResponseEntity<String> handleException(ConstraintViolationException e){
  return new ResponseEntity("Error: " + e.getMessage(),HttpStatus.BAD_REQUEST);
}
```

Either add this to your controller class or to a class annotated with `@ControllerAdvice`

Validating an incoming HTTP request body payload works a little bit differently. First, ensure you use the Bean Validation annotations on the POJO that maps to the incoming request body:

```
public class Payload {

  @NotBlank
  private String message;

  @Email
  private String email;

  @Future
  private LocalDateTime memberSince;

  // getters & setters
}
```

Next, add @Valid to the @RequestBody annotation of your controller method:

```
@PostMappingpublic
String validatePayload(@Valid @RequestBody Payload payload) {
  return "Payload is valid";
}
```

Sending an payload will result in an HTTP 400 response including information about validation errors:

```
curl -X POST -H "Content-Type: application/json" \
-d '{"message":"Hello","email":"notAnEmail","memberSince": "2025-04-20T12:00"}' \
http://localhost:8080/validation
```

How can I return a Thymeleaf view from a controller?

To start using Thymeleaf, add the following Spring Boot Starter to your project:

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

Next, you need an endpoint to return your Thyemleaf view. For this use @Controller to annotate your controller class (hint: you can't use @RestController here, as this returns the payload as part of the HTTP body). There are multiple ways to tell Spring which view name to render. The simplest is to return the name of the view as a String:

```
@Controller
@RequestMapping("/welcome")
public class ViewController {
  @GetMapping
  public String getWelcomePage(Model model) {
    model.addAttribute("message", "Hello World!");    return "welcome";
  }
}

<!DOCTYPE HTML>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
</head>
<body>
  <div>
    <span th:text="${message}"></span>
  </div>
</body>
</html>
```

How can I return XML from my Spring MVC controller?

To return XML from your controller, you need an additional dependency from Jackson:

```
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

If you are using the Spring Boot Starter Web, Spring Boot ensure to auto-configure everything to make use of the JacksonXmlModule. What's left is to annotate your POJO with @XmlRootElement

```
@XmlRootElement
public class Order {
  private String id;
  private Set<String> tags;
  private Long customerId;
  private LocalDateTime orderAt;
  // getters & setters
}
```

Now you can return XML from your Spring MVC controller methods:

```
@GetMapping(path = "/orders", produces = MediaType.APPLICATION_XML_VALUE)
public List<Order> getOrders() {
  List<Order> orders = List.of(
    new Order("42L", Set.of("foreign", "books"), 1L, LocalDateTime.now()),
    new Order("58B", Set.of("fractile", "computer"), 48L, LocalDateTime.now())
  );
  return orders;
}
```

How can I provide different content types for a controller?

To be able to return different content types and let the client decide which type he can process (content negotiation), pass multiple MediaTypedefinitions to the produces field of your mapping annotation. This might be @GetMapping,@RequestMapping, @PostMapping, etc:

```
@GetMapping(path="/orders",
      produces={MediaType.APPLICATION_XML_VALUE,MediaType.APPLICATION_JSON_VALUE})
public List<Order> getOrders() {
  List<Order> orders = List.of(
    new Order("42L", Set.of("foreign", "books"), 1L, LocalDateTime.now()),
    new Order("58B", Set.of("fractile", "computer"), 48L, LocalDateTime.now())
  );
  return orders;
}
```

A client can now decide which content type he wants to process using the HTTP Accept header:

```
curl -H "Accept: application/json" http://localhost:8080/mix/orders
[{"id":"42L","tags":["books","foreign"],"customerId":1,"orderAt":"2020-04
06T05:45:39.40289"},
{"id":"58B","tags":["computer","fractile"],"customerId":48,"orderAt":"2020-04
06T05:45:39.402908"}]
```

```
curl -H "Accept: application/xml" http://localhost:8080/mix/orders
<List><item><id>42L</id><tags><tags>books</tags><tags>foreign</tags></tags>
<customerId>1</customerId><orderAt>2020-04-06T05:48:08.077563</orderAt></item>
<item><id>58B</id><tags><tags>computer</tags><tags>fractile</tags></tags>
<customerId>48</customerId><orderAt>2020-04-06T05:48:08.077587</orderAt></item>
</List>
```

How can I up-/download a file with Spring Web MVC?

To upload a file to a Spring Web MVC controller endpoint, you can make use of Spring's *MultipartFile* class. A common approach to upload files is using an HTML form and the content type *multipart/form-data*

```
@PostMapping
public void upload(@RequestParam("file") MultipartFile file) throws IOException {

  System.out.println("Filename: " + multipartFile.getOriginalFilename());
  System.out.println("Size: " + multipartFile.getSize());

  byte[] content = multipartFile.getBytes();

}
```

For an example on how to download a file, let's use a file available on the classpath (part of *src/main/resources*). To instruct a client e.g. a browser that the response is a file, you have to set some HTTP headers.

Using Content-Type you help a client to understand which kind of file is downloaded to e.g. suggest opening the file with a PDF viewer. Furthermore, with *Content-Length* and *Content-Disposition* you can add metadata like the filename and size to the response:

```
@GetMapping
public ResponseEntity<byte[]> download() throws IOException {


  byte[] pdfContent = = this.getClass()
     .getResourceAsStream("/document.pdf").readAllBytes();


  HttpHeaders header = new HttpHeaders();
  header.setContentType(MediaType.APPLICATION_PDF);
  header.setContentLength(pdfContent.length);
  header.set("Content-Disposition", "attachment; filename=document.pdf");


  return new ResponseEntity(pdfContent, header, HttpStatus.OK);
}
```

Full-stack example https://rieckpil.de/howto-up-and-download-files-with-react-and-spring-boot/