Advanced/Enterprise Java

Agenda

- Spring Web MVC
  - Detailed Flow
  - Spring Boot JSP limitations
  - Using static resources
  - Spring form tags
  - @ModelAttribute
  - Spring validations
- REST web services

Spring Web MVC

Spring Boot - JSP limitations

- Spring boot runs with "embedded" web server like tomcat/jetty. With executable jar, JSP files are not accessible directly.
- JBoss Undertow web server does not support JSPs.
- Creating a custom error.jsp page does not override the default view for error handling. Custom error pages needs to be created and configured.

Spring MVC Navigation

- Control can be transferred from one request handler method to another request handler method by returning specialized urls as view names.
  - redirect:url
  - forward:url

redirect:url

- Sends a temporary response to the browser (status=302) that force browser to make new request to the given url.
- It internally does response.sendRedirect("url").

- Can be used to redirect to another url in application or outside the application.

forward:url

- Transfers current request to the given url (request handler method).
- It internally does requestDispatcher.forward(request, response).
- Can be used to redirect to another url in application only.

Using static resources

- Spring boot configures ResourceHttpRequestHandler to serve static resources like HTML pages, JS, CSS, images, etc.
- By default it search resources under
  - /static
  - /public
  - /resources
  - /META-INF/resources
- Default locations can be modified in application.properties.

```
spring.web.resources.static-locations=classpath:/files,file:/opt/files
```

- Default static url is "/" and it can be altered in application.properties.

```
spring.mvc.static-path-pattern=/content/**
# access resources as http://localhost:8080/content/resource
```

- Reference: https://www.baeldung.com/spring-mvc-static-resources
- External CSS can be added directly (giving link).
  - `<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.2.1/css/bootstrap.min.css"/>`

@ModelAttribute

- View --> Request params --> Controller
    - FrontController accept req params and fill it in a object and send as argument to request handler method.

```
@RequestMapping("/login")
String validate(Credentials cred) {
    // ...
}
```

        - index.jsp --> Req params (email and password) --> FrontController --> create object of Credentials and set req params in it --> call req handler validate() with Credentials as its argument.

- Controller --> Model attributes (Request attributes) --> View

```
@RequestMapping("/login")
String validate(Credentials cred, Model model) {
    // ...
    model.addAttribute("message", "login successful.");
}
```

```
@RequestMapping("/login")
String validate(@ModelAttribute("cr") Credentials cred, Model model) {
    // ...
    model.addAttribute("message", "login successful.");
}
```

    - @ModelAttribute("cr") Credentials cred --> internally equivalent to model.addAttribute("cr", cred); + add BindingResult (errors) in model.

```

```

```
@RequestMapping("/url")
ModelAndView method() {
    Pojo obj = new Pojo();
    return new ModelAndView("viewName", "command", obj);
}
```

- return new ModelAndView("viewName", "command", obj); --> equivalent to --

```
model.addAttribute("command", obj);
return "viewName";
```

Spring form tag

- Must have a form backing bean/pojo to initialize/auto-fill the form -- given as attribute `<sf:form modelAttribute="beanName" action="...">`.
  - If `modelAttribute="beanName"` not given, default `modelAttribute="command"`.
  - Controller must send a pojo with name "beanName" in request/model scope.
  - Controller can send this using one of the following ways.
    - model.addAttribute("command", obj);
    - @ModelAttribute Pojo obj -- in argument or return value
    - return ModelAndView from req handler method
- https://docs.spring.io/spring-framework/docs/4.2.x/spring-framework-reference/html/spring-form-tld.html#spring-form.tld.select

Spring MVC/Boot Validations

- End user may enter wrong data while registration/sign in/data entry.
- In "general" there are two ways of Validations
  - Client side Validations
  - Server side Validations

Client side Validations

- HTML5 : `<input type="text" required>`
- JS or jQuery framework for validation.
- Client side validations are faster and gives better user experience.
- Client side validations can be compromized by the end user (disable JS, ...).

Server side Validations

- Spring MVC supports server side validations.
- It use standard annotations for the data validations (JSR 303).
    - @NotBlank -- should not be empty (primitive types and string)
    - @NotEmpty -- should not be empty (collections)
    - @NotNull -- should not be null
    - @Min and @Max -- numeric values like age.
    - @Size -- string length
    - @DateTimeFormat -- desired date format (`<input type="date">` -- format=yyyy-MM-dd)
    - @Email -- Email pattern check (internally use regex)
    - @Pattern -- Regular expression
- To process validations use @Valid on command/pojo object -- req handler method arg.
- In Spring MVC use @ModelAttribute to add the object into model.
- To collect the validation errors use BindingResult or Errors object as next argument (after command object).

Spring Boot + JSP Validation Steps

- In pom.xml add validation starter.

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

- In Customer class, use appropriate validation annotations.

```
public class Customer {
    private int id;
    @NotEmpty
    private String name;
    @NotEmpty
    @Email
    private String email;
    @NotEmpty
    @Pattern(regexp = "[0-9]{10}")
    private String mobile;
    @NotEmpty
    private String address;
    @NotEmpty
    @Size(min = 4)
    private String password;
    @DateTimeFormat(pattern="yyyy-MM-dd")
    @NotNull
    private Date birth;
    // ...
}
```

- In request handler method -- LoginControllerImpl.saveUser()
  - The command/model object should be marked as @Valid.
  - In Spring MVC project, also use @ModelAttribute("...").
  - Immediate next argument must be BindingResult to collect validation errors.
  - Check BindingResult object to see errors -- hasErrors(). Send user to current view if error is found.

```
@RequestMapping("/register")
public String saveUser(@ModelAttribute("u") @Valid Customer user, BindingResult res, Model model) {
    try {
        if(res.hasErrors())
            return "newuser";
```

```
        // ...
        userService.saveUser(user);
        model.addAttribute("message", "Customer added successfully " + user.getId());
    } catch (Exception e) {
        e.printStackTrace();
        model.addAttribute("message", "Customer add failed " + e.getMessage());
    }
    return "newuser";
}
```

- In newuser.jsp use sf:errors to render the errors.

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form" %>
<sf:form action="/register">
    Name: <sf:input path="name"/> <sf:errors path="name" cssClass="error"/> <br/><br/>
    Email: <sf:input path="email"/> <sf:errors path="email" cssClass="error"/> <br/><br/>
    Password: <sf:password path="password"/> <sf:errors path="password" cssClass="error"/> <br/><br/>
    <input type="submit" value="Sign Up"/>
    <a href="/">Sign In</a> <br/><br/>
    ${message} <br/><br/>
    All validation errors <br/>
    <sf:errors path="*" cssClass="error"/>
</sf:form>
```

- Run application and check if validations are working.
- To provide custom error messages, in messages.properties add messages in a format like Annotation.command.field=message.

```
NotEmpty.u.name = name is mandatory
NotEmpty.u.mobile = mobile is mandatory
Pattern.u.mobile = mobile must be 10 digits
NotEmpty.u.email = email is mandatory
Email.u.email = email is invalid
```

```
NotEmpty.u.password = password is mandatory
Size.u.password = password must be atleast {0} chars
NotEmpty.u.address = address is mandatory
NotNull.u.birth = birth date is mandatory
```

- Re-run application and check custom error messages.

Assignments

1. Complete Bookshop application using Spring Boot.
    - Add external css for showing error messages in red color.
    - Complete delete book and add book functionality.
    - Show "Hello, username" on each page.