

Session 5

Form validation, Bootstrap and CSS

Validation happens twice: in the browser for fast feedback and on the server because the browser cannot be trusted. Bootstrap then gives the forms a consistent look without hand-written CSS for every element.

All four questions change the Session 4 admission form inside the `student-admission` project. The listings below are the end-of-session files; each question's explanation points at the lines that answer it.

Objectives

✗ Do not copy. Read for understanding and the viva

- Complete questions 21 to 24 of the manual: form validation, bootstrap and css
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✗ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q21	Apply the client validation in the form created in the above exercise 4 of session 4,...	Complete
Q22	Write a programme to bind form objects with entity bean in Spring MVC	Complete
Q23	Configure Bootstrap in Spring MVC and use default styling classes in the form and view...	Complete
Q24	Apply custom Styling to your pages in Spring MVC	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- Server-side validation uses Bean Validation annotations (`@NotBlank` , `@Size` , `@Past` , `@Email`) on the form object and `@Valid` plus `BindingResult` in the controller.
- Client-side validation is HTML5 attributes (`required` , `pattern` , `min`) or a few lines of JavaScript.
- Include Bootstrap from a CDN link in the page head or add the WebJar dependency.

Question 21

Problem Statement

■ Write in lab record

Apply the client validation in the form created in the above exercise 4 of session 4, along with server-side validation.

Solution

■ Write in lab record

Steps

1. Add the `hibernate-validator` dependency from the fragment to `pom.xml` ; Maven → Update Project.
2. Replace `AdmissionForm.java` with the annotated version. Every rule is an annotation on the field.
3. Replace `admission-form.jsp` . Client rules are HTML attributes on the inputs (`required` , `minlength` , `pattern` , `max`); server messages appear through `form:errors` .
4. Client test: rebuild, open `/admission` , leave the name empty and press Submit. The browser refuses to send the form and points at the field.
5. Server test: in the browser's developer tools add `novalidate` to the `form` element (or run `document.querySelector('form').noValidate = true` in the console), submit the empty form. The page comes back with red messages under each field.

Program

- Lab record: every tab is one file of the answer. Write all of them.

pom.xml

pom.xml (Q21 addition)

```
1 <!-- Q21: add inside <dependencies> of pom.xml. Hibernate Validator is the B
2     Validation 3.0 provider; Spring MVC picks it up automatically for @Vali
3     Tomcat 10.1 supplies the Jakarta EL implementation it needs; add
4     org.glassfish.expressly:expressly:5.0.0 only if you validate outside To
5 <dependency>
6     <groupId>org.hibernate.validator</groupId>
7     <artifactId>hibernate-validator</artifactId>
8     <version>8.0.1.Final</version>
9 </dependency>
```

Output

Expected, checked by reading, not executed. Client side: Chrome's bubble "Please fill in this field" on the name box, and "Please match the requested format" with the title text on a mobile number like `12345`. Server side (with `novalidate`), the form returns with these messages under the fields:

```
1 Full name      must not be blank
2 Email         must not be blank
3 Mobile        must not be blank
4 Date of birth  must not be null
5 Programme     must not be blank
6 Address       must not be blank
7 Hostel required? choose Yes or No
8 Courses       pick at least one course
```

A name of two letters with the browser check bypassed returns `size must be between 3 and 80`; a date of birth in the future returns `must be a past date`.

Explanation

- Client validation is the HTML5 attributes: `required` on every field, `minlength="3"` `maxlength="80"` on the name, `type="email"`, `pattern="[6-9][0-9]{9}"` on the mobile,

`type="date" max="${today}"` so the picker cannot choose a future date, `required` on the select and on the first radio (it applies to the whole radio group). Checkboxes have no "at least one" attribute; only the server enforces that rule.

- Server validation is Bean Validation: `@NotBlank`, `@Size`, `@Email`, `@Pattern`, `@Past`, `@NotNull`, `@NotEmpty` on `AdmissionForm`. Hibernate Validator is the provider; Spring MVC finds it on the classpath and runs it whenever a handler parameter carries `@Valid` (see the controller in Q22).
- The rules match on both sides (same regex, same length limits) so a user with JavaScript on and a user posting with `curl` get the same answer.
- `cssErrorClass` swaps the input's class to `form-control is-invalid` when that field has an error; `form:errors` prints the message in a `span` with class `invalid-feedback`, which Bootstrap shows only next to an invalid control.

Question 22

Problem Statement

■ Write in lab record

Write a programme to bind form objects with entity bean in Spring MVC.

Solution

■ Write in lab record

Steps

1. Replace `AdmissionController.java`: the POST handler takes `@Valid AdmissionForm` and a `BindingResult`, and on success converts the form to a `Student` entity and saves it.
2. Add `save()` to `StudentRepository.java`.
3. Replace `admission-result.jsp`; it now shows the saved entity, including the generated database id.
4. Rebuild, submit a valid application, then check MySQL: `SELECT id, name, programme, courses FROM ignou.student;`

Program

- Lab record: every tab is one file of the answer. Write all of them.

AdmissionController.java

web/AdmissionController.java

```
1 package com.ignou.lab.admission.web;
2
3 import java.time.LocalDate;
4 import java.util.List;
5 import jakarta.validation.Valid;
6 import org.springframework.stereotype.Controller;
7 import org.springframework.ui.Model;
8 import org.springframework.validation.BindingResult;
9 import org.springframework.web.bind.annotation.GetMapping;
10 import org.springframework.web.bind.annotation.ModelAttribute;
11 import org.springframework.web.bind.annotation.PostMapping;
12 import com.ignou.lab.admission.entity.Student;
13 import com.ignou.lab.admission.repo.StudentRepository;
14
15 @Controller
16 public class AdmissionController {
17
18     private static final List<String> PROGRAMMES = List.of("MCA", "BCA", "PG
19     private static final List<String> COURSES = List.of(
20         "MCS-218 Data Communication and Computer Networks",
21         "MCS-219 Object Oriented Analysis and Design",
22         "MCS-220 Web Technologies",
23         "MCS-221 Data Warehousing and Data Mining");
24
25     private final StudentRepository repo;
26
27     public AdmissionController(StudentRepository repo) {
28         this.repo = repo;
29     }
30
31     @ModelAttribute("programmes")
32     public List<String> programmes() {
33         return PROGRAMMES;
34     }
35
36     @ModelAttribute("courseList")
37     public List<String> courseList() {
38         return COURSES;
39     }
40
41     /** used by the date picker's max attribute (client-side @Past) */
42     @ModelAttribute("today")
43     public LocalDate today() {
```

```
44         return LocalDate.now();
45     }
46
47     @GetMapping("/admission")
48     public String showForm(Model model) {
49         model.addAttribute("admission", new AdmissionForm());
50         return "admission-form";
51     }
52
53     @PostMapping("/admission")
54     public String submit(@Valid @ModelAttribute("admission") AdmissionForm f,
55                         BindingResult result, Model model) {
56         if (result.hasErrors()) {
57             return "admission-form"; // re-render with form:errors filled in
58         }
59         Student saved = repo.save(form.toStudent()); // Q22: form object →
60         model.addAttribute("student", saved);
61         return "admission-result";
62     }
63 }
```

Output

Expected, checked by reading, not executed. Tomcat's console shows the INSERT
Hibernate issued:

```
1 Hibernate:
2     insert
3     into
4         student
5         (address, courses, dob, email, enrolment_no, hostel_required, mobile
6     values
7         (?, ?, ?, ?, ?, ?, ?, ?, ?)
```

The result page reads `Application saved. Database id: 3` followed by the table of values and
`Enrolment no: pending approval`. The MySQL query returns the new row with `id = 3`.

Explanation

- Two objects, two jobs. `AdmissionForm` is shaped like the screen (a `List` of ticked courses, a `Boolean` that may be null before validation). `Student` is shaped like the table (one comma-

separated `courses` column, an `id`). `toStudent()` is the binding between them: one setter per field, plus `String.join` for the list.

- Spring binds the request to the form object first (data binding), validates it (`@Valid`), and only then does the code bind the form to the entity. A bad request never reaches the entity or the database.
- `BindingResult` must be the parameter immediately after the `@Valid` one. If it is missing, Spring throws `MethodArgumentNotValidException` instead of letting the controller re-render the form.
- `repo.save()` runs inside a transaction (`@Transactional` on the class). `em.persist` schedules the INSERT; commit at method exit executes it and MySQL's auto-increment value is copied into `student.getId()`.
- The manual's Thymeleaf example binds the form straight to the entity class. That works when the two shapes coincide; the separate form object is the pattern that survives Session 6, where `Student` gains relations.

Question 23

Problem Statement

■ Write in lab record

Configure Bootstrap in Spring MVC and use default styling classes in the form and view created in the above exercises.

Solution

■ Write in lab record

Steps

1. Create `WEB-INF/views/head.jspf` with the Bootstrap 5.3 CDN link (copy the `link` tag from getbootstrap.com; the `integrity` attribute is optional and can be pasted from there).
2. In every JSP replace the `meta` lines inside `head` with `<%@ include file="head.jspf" %>`.
`admission-form.jsp` and `admission-result.jsp` from Q21 and Q22 already do this; replace `home.jsp` with the version below.
3. Rebuild and reload. The form is centred in a card, inputs have rounded borders, the submit button is blue. A missing link (typo in the URL) shows the unstyled Session 4 look, which is the

quickest way to confirm the CDN line is loading.

4. The WebJar alternative from the manual: add `org.webjars:bootstrap:5.3.3` and `org.webjars:webjars-locator-core` to the pom and link `/webjars/bootstrap/css/bootstrap.min.css`; the CDN needs no Maven change so it is used here.

Program

- Lab record: every tab is one file of the answer. Write all of them.

head.jspf

WEB-INF/views/head.jspf

```
1 <!-- /WEB-INF/views/head.jspf : shared <head> contents, pulled in with <%@ i
2 <meta charset="UTF-8">
3 <meta name="viewport" content="width=device-width, initial-scale=1">
4 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.
5 <link href="${pageContext.request.contextPath}/css/admission.css" rel="style
6 <!-- ponytail: no Bootstrap JS bundle; add the script tag when a dropdown or
```

Output

Expected, checked by reading, not executed: a dark navigation bar with "IGNOU Student Admission", below it a white card 760 px wide holding the form in two columns (email beside mobile, date beside programme), inline Yes/No radios, stacked course checkboxes, and a right-aligned Cancel / Submit pair. Invalid fields get a red border and red text underneath. The result page shows a green "Application saved" alert and a striped table.

Explanation

- Bootstrap is only a stylesheet. Configuring it in Spring MVC means one `link` tag reaches every page; the `jspf` include keeps that tag in one file, so a version bump is one edit.
- Classes used, all Bootstrap defaults: layout `container`, `row g-3`, `col-12`, `col-md-6`; form controls `form-label`, `form-control`, `form-select`, `form-check`, `form-check-inline`, `form-check-input`; validation `is-invalid`, `invalid-feedback`; components `card`, `card-body`, `navbar`, `alert alert-success`, `table table-striped`, `btn btn-primary`, `btn-outline-secondary`.
- Spring form tags take `cssClass` instead of `class` (and `cssErrorClass` for the error state); plain HTML elements in the same page use `class` as usual.

- No Bootstrap JavaScript is included because nothing on these pages needs it; add the bundle script tag when a dropdown menu or modal appears.

Question 24

Problem Statement

■ Write in lab record

Apply custom Styling to your pages in Spring MVC.

Solution

■ Write in lab record

Steps

1. Create `src/main/webapp/css/admission.css` (outside `WEB-INF`, so the browser can fetch it).
2. Replace `WebConfig.java` with the version that adds a resource handler for `/css/**`. Without it the `DispatcherServlet`, mapped on `/`, tries to find a controller for `/css/admission.css` and returns 404.
3. `head.jspf` already links `/css/admission.css` after Bootstrap, so the custom rules win over Bootstrap's on equal specificity.
4. Rebuild, hard-reload (Ctrl+Shift+R) and open `http://localhost:8080/student-admission/css/admission.css` directly to confirm it is served.

Program

- Lab record: every tab is one file of the answer. Write all of them.

`admission.css`

webapp/css/admission.css

```
1  /* src/main/webapp/css/admission.css : custom styling on top of Bootstrap */
2  :root {
3      --ignou-maroon: #7b1e3c;
4      --ignou-sand: #f6f1ea;
5  }
6
7  body {
8      background: var(--ignou-sand);
9  }
10
11  .navbar {
12      background: var(--ignou-maroon);
13  }
14
15  .card-form {
16      max-width: 760px;
17      margin: 2rem auto;
18      border: 0;
19      box-shadow: 0 0.5rem 1rem rgba(0, 0, 0, 0.08);
20  }
21
22  .card-title {
23      color: var(--ignou-maroon);
24      border-bottom: 2px solid var(--ignou-maroon);
25      padding-bottom: 0.5rem;
26      margin-bottom: 1.25rem;
27  }
28
29  .btn-primary {
30      background: var(--ignou-maroon);
31      border-color: var(--ignou-maroon);
32  }
33
34  .btn-primary:hover {
35      background: #5d1630;
36      border-color: #5d1630;
37  }
38
39  /* red asterisk after every mandatory label */
40  label.required::after {
41      content: " *";
42      color: #dc3545;
43  }
```

```
44
45 /* the lab record is printed: hide navigation, keep the data */
46 @media print {
47     .navbar, .btn { display: none; }
48     .card-form { box-shadow: none; max-width: none; margin: 0; }
49 }
```

Output

Expected, checked by reading, not executed: sand-coloured page background, maroon navigation bar and maroon primary buttons instead of Bootstrap blue, card title underlined in maroon, a red asterisk after each mandatory label, and in print preview (Ctrl+P) the navigation bar and buttons disappear while the form data stays.

Explanation

- `addResourceHandlers` maps a URL pattern to a location inside the WAR; Spring serves the file with caching headers and never involves a controller.
- Custom CSS overrides by cascade order: same selector specificity, loaded later, wins. `.btn-primary` and `.navbar` redefine Bootstrap's colours; the `:root` variables keep the brand colour in one place.
- `label.required::after` adds the asterisk from CSS, so the JSP only sets a class; the mandatory marker cannot drift from the `required` attribute if both come from the same template line.
- The `@media print` block exists because the lab record is printed; hiding navigation is the difference between a page and a form.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** Why validate on both client and server? **A:** Client checks give instant feedback; the server check is the only one that cannot be bypassed (disable JavaScript, use curl, edit the DOM).
- **Q:** What does `@Valid` do? **A:** Tells Spring to run Bean Validation on the bound object before calling the handler and to record violations in the following `BindingResult`.
- **Q:** What happens if `BindingResult` is not the next parameter? **A:** Spring throws the validation exception and the user sees an error page instead of the form with messages.

- **Q:** Difference between `@NotNull`, `@NotEmpty` and `@NotBlank`? **A:** `@NotNull` rejects null; `@NotEmpty` also rejects empty strings or collections; `@NotBlank` also rejects whitespace-only strings.
- **Q:** Why a separate form object instead of binding to the entity? **A:** The screen and the table have different shapes; the form object carries validation rules and screen-only fields without polluting the entity.
- **Q:** Why does the CSS file live outside `WEB-INF`? **A:** Anything under `WEB-INF` is never served directly; a stylesheet must be fetched by the browser.
- **Q:** CDN or WebJar for Bootstrap? **A:** CDN: no build change, browser may already have it cached, needs internet. WebJar: version pinned in the pom, works offline.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Adding validation annotations and forgetting `@Valid` on the parameter; every submission passes.
- Testing only in the browser and concluding the server rules work. Bypass the browser once (`novalidate`) to see them.
- Writing `class="form-control"` on a `form:input` tag; the attribute is `cssClass`.
- Mapping `/css/**` but putting the folder under `WEB-INF`; the resource handler cannot serve it.
- Loading the custom stylesheet before Bootstrap, so Bootstrap overrides the custom colours.

Session Summary

■ Write in lab record

- Question 21: HTML5 attributes on the form plus Bean Validation on `AdmissionForm`, messages rendered with `form:errors`
- Question 22: `@Valid` and `BindingResult` in `AdmissionController`, `toStudent()` binding the form to the `Student` entity, `StudentRepository.save()` inserting the row
- Question 23: Bootstrap 5.3 from the CDN through `head.jspf`, default classes on the form, result and home pages
- Question 24: `admission.css` served through a resource handler, brand colours, required-field marker and print rules

[✎ Edit this page](#)[< Previous](#)
Session 4**Session 6** [Next >](#)