ЗАТВЕРДЖУЮ
Начальник кафедри інформаційних технологій та систем електронних комунікацій, к.т.н., доцент
Назарій БУРАК
«____» _____ 20___ р.

МЕТОДИЧНА РОЗРОБКА

для проведення лабораторного заняття із здобувачами освіти першого (бакалаврського) рівня вищої освіти спеціальності F3 «Комп'ютерні науки»
з дисципліни «WEB-технології та WEB-програмування»

Лабораторна робота № 7: «Розроблення вебконтролерів, вебсторінок на Thymeleaf та RESTful API з валідацією даних і обробкою помилок» (тема 2.3)

Мета: навчитися створювати контролери вебсторінок і RESTful API на Spring MVC, формувати сторінки засобами Thymeleaf з фрагментами і формами, працювати з параметрами запиту і cookies, реалізовувати повний набір CRUD-операцій з правильними кодами стану, валідувати вхідні дані засобами Bean Validation, централізовано обробляти помилки у форматі Problem Details і документувати API за стандартом OpenAPI.

Час: 4 години.

Місце заняття: комп'ютерна лабораторія.

Матеріальне забезпечення: ПК з доступом до мережі Internet; JDK 21; IntelliJ IDEA; Maven; Postman або curl; браузер Chrome із засобами розробника; Git; демо-проєкт https://github.com/SidiromUA/shelter-registry (тег `lab-07`).

Після лабораторного заняття здобувачі освіти повинні:

вміти: створювати `@Controller` і `@RestController`; отримувати дані через `@PathVariable`, `@RequestParam`, `@CookieValue`, `@ModelAttribute`, `@RequestBody`; створювати шаблони Thymeleaf з фрагментами; обробляти форми з виведенням помилок біля полів і шаблоном PRG; повертати 201 із заголовком `Location`, 204, 400, 404, 409; описувати обмеження Bean Validation; створювати `@RestControllerAdvice`; документувати API в Swagger UI;

знати: відмінність контролерів сторінок і REST, вирази й атрибути Thymeleaf, роль `BindingResult`, принципи REST, семантику методів і кодів

стану HTTP, рівні валідації, формат Problem Details, призначення OpenAPI.

I. Підготовка до заняття

Викладач перевіряє на ПК лабораторії наявність JDK 21, Maven, Postman. Здобувачі освіти до заняття опрацьовують лекцію 7 і мають власний проєкт з ЛР 6 (профілі dev/prod, сервіси, репозиторії на `JdbcClient`).

II. Структура заняття

Короткі теоретичні відомості

`@Controller` повертає ім’я шаблону Thymeleaf з `templates/`, дані передає через `Model`; `@RestController` повертає об’єкт, який Jackson перетворює на JSON. У шаблоні `th:text` виводить дані з екрануванням, `th:each` – цикл, `th:if` – умова, `@{…}` – URL, `th:replace` – фрагмент. Форму зв’язують з об’єктом атрибутами `th:object` і `th:field`; обробник приймає `@Valid @ModelAttribute` і одразу після нього `BindingResult`. RESTful API моделює ресурси іменниками у множині, а дії – методами GET, POST, PUT, DELETE; створення повертає **201** і `Location`, видалення – **204**, некоректні дані – **400**, відсутній ресурс – **404**. Помилки обробляє клас `@RestControllerAdvice`, що повертає `ProblemDetail` (RFC 9457). Бібліотека springdoc-openapi генерує документацію `/swagger-ui.html`.

2.1. Огляд демо-проєкту

```
git checkout lab-07
git diff lab-06 lab-07 --stat
mvn spring-boot:run
# сторінки: http://localhost:8080/shelters      API:
http://localhost:8080/swagger-ui.html
```

Клас / файл	Призначення
`web.ShelterWebController`	сторінки списку (фільтр у cookie), картки, створення, редагування, видалення, реєстрації перевірки
`templates/fragments/layout.html`	фрагменти `head(title)`, `header`, `flash`
`templates/shelters/*.html`	`list`, `detail`, `form`; `error/404.html`
`api.ShelterApiController`	CRUD API укриттів і перевірок з анотаціями OpenAPI
`dto.ShelterForm`, `dto.InspectionRequest`	вхідні DTO з обмеженнями Bean Validation
`api.ApiExceptionHandler`, `web.WebExceptionHandler`	централізована обробка помилок API (Problem Details) і сторінок

Таблиця 1 – Нові компоненти демо-проєкту (тег lab-07)

2.2. Дослідження сторінок і API

1. Відкрийте список укриттів, оберіть район. У DevTools (*Application* → *Cookies*) знайдіть cookie `district` і її термін дії. Відкрийте `/shelters` без параметра – фільтр має відновитися. Оберіть «Усі райони» – cookie має зникнути.
2. Додайте укриття з порожньою назвою і місткістю 0: помилки мають з’явитися біля полів, введені значення – зберегтися. Додайте коректне укриття: на вкладці *Network* знайдіть ланцюжок 302 → 200 і повідомлення «Укриття … додано». Спробуйте додати укриття з наявною адресою.
3. Відкрийте `/shelters/999` – має з’явитися сторінка 404 з шаблону, а не «Whitelabel Error Page».
4. У Postman виконайте запити з табл. 2 і зафіксуйте коди стану, заголовки та тіла відповідей.
5. Відкрийте `/swagger-ui.html`, знайдіть опис схеми `ShelterForm` з обмеженнями і виконайте `POST /api/shelters` кнопкою «Try it out». Перегляньте опис `/v3/api-docs`.

Запит	Очікувана відповідь
`POST /api/shelters` з коректним JSON	201, заголовок `Location`, створене укриття
`POST /api/shelters` з `{"name":"","capacity":0}`	400, `application/problem+json`, `errors` за полями
`POST /api/shelters` з `"type":"CAVE"` або не JSON	400 «Тіло запиту не вдалося прочитати»
`PUT /api/shelters/{id}`	200 з оновленими даними; для неіснуючого id – 404
`DELETE /api/shelters/{id}` (двічі)	204, потім 404
`POST /api/shelters/3/inspections` з `"inspectorId":999`	409 Conflict
`PATCH /api/shelters/3`	405 Method Not Allowed

Таблиця 2 – Запити для перевірки API

2.3. Індивідуальне завдання

Доповніть власний проєкт з ЛР 6 для своєї предметної області:

1. Підключіть стартери `thymeleaf`, `validation` і бібліотеку `springdoc-openapi-starter-webmvc-ui`.
2. Створіть фрагменти макета (head з параметром-заголовком, шапка з навігацією і посиланням «Перейти до основного вмісту», блок повідомлень) і стилі в `static/css`.
3. Створіть контролер сторінок головної сутності: список з фільтром (параметр запиту) і підсумком у `caption`; картка об’єкта з пов’язаними даними (1:N); форма створення і редагування; видалення методом POST з підтвердженням.
4. Збережіть один параметр користувача в cookie (фільтр, сортування або кількість записів на сторінці) з атрибутом `HttpOnly` і терміном дії.
5. Створіть вхідний DTO з обмеженнями Bean Validation і власними повідомленнями українською; обробник форми з `BindingResult`, виведенням помилок біля полів (з `aria-describedby`) і шаблоном PRG з flash-повідомленням; бізнес-помилки сервісу показуйте так само.
6. Створіть REST-контролер з повним CRUD для головної сутності і щонайменше одним вкладеним ресурсом: 200, 201 + `Location`, 204, 400, 404 – відповідно до табл. 5 лекції.

7. Створіть `@RestControllerAdvice`, що повертає `ProblemDetail` з полем `errors` для помилок валідації, бізнес-правил, нечитабельного тіла, відсутнього ресурсу і порушення цілісності даних (409); для сторінок – шаблон помилки 404.
8. Додайте анотації `@Tag`, `@Operation`, `@ApiResponse` і переконайтеся, що всі маршрути з'явилися в Swagger UI.
9. Створіть колекцію Postman (або файл `requests.http` для IntelliJ IDEA) з усіма запитами табл. 2 для вашого API і збережіть її в репозиторії.
10. Оновіть `README.md` (таблиці маршрутів сторінок і API з кодами відповіді), зафіксуйте зміни, створіть тег `lab-07` і надішліть на GitHub.

Додаткові завдання (за бажанням): реалізуйте `PATCH` для часткової зміни одного поля; додайте пагінацію списку (`?page=0&size=10`) із заголовком `X-Total-Count`; локалізуйте повідомлення валідації файлом `messages.properties`; створіть власну анотацію валідації (наприклад, перевірка формату номера телефону).

2.4. Вимоги до звіту

Звіт (PDF або DOC, до 6 сторінок) містить: мету; таблиці маршрутів сторінок і API; фрагменти коду контролера сторінок з обробкою форми, шаблону форми, REST-контролера, DTO з обмеженнями і обробника помилок; знімки сторінок (список з фільтром, форма з помилками, картка з повідомленням, сторінка 404) і cookie в DevTools; знімки Postman для відповідей 201 (з `Location`), 204, 400 (Problem Details), 404, 409; знімок Swagger UI; посилання на репозиторій з тегом `lab-07`; відповіді на 3 контрольні запитання; висновки. У ВУ в завданні «ЛР 7» у поле «Текстова відповідь» вставте посилання на репозиторій, файлом додайте звіт.

2.5. Контрольні запитання

1. Чим `@Controller` відрізняється від `@RestController`?
2. Як отримати змінну шляху, параметр запиту, cookie і тіло запиту?
3. Як Thymeleaf захищає від XSS? Для чого фрагменти?
4. Яку роль відіграє `BindingResult` і чому його місце в сигнатурі важливе?
5. Назвіть принципи REST. Як правильно називати ресурси в URI?

6. Які коди стану повертаються під час створення, видалення, помилки валідації, відсутності ресурсу?
7. Навіщо розділяти вхідні і вихідні DTO?
8. Як працює `@RestControllerAdvice`? Які поля має Problem Details?
9. Що таке OpenAPI і як springdoc створює документацію?

2.6. Критерії оцінювання (максимально 5 балів)

Бали	Критерій
5	сторінки з фрагментами, фільтром і cookie; форма з валідацією, помилками біля полів і PRG; повний CRUD API з правильними кодами і `Location`; централізована обробка помилок у форматі Problem Details; Swagger UI; колекція запитів; тег `lab-07` на GitHub; здобувач вільно пояснює принципи REST і роботу валідації
4	робота виконана повністю, є незначні недоліки (напр. неточні коди стану, немає cookie або колекції запитів)
3	виконано не менше 60 % (напр. лише сторінки або лише API, немає обробника помилок) або є суттєві помилки
0–2	виконано менше 60 % або здобувач не може пояснити поданий код

Робота, подана після встановленого терміну без поважних причин, оцінюється з пониженням на 1 бал.

III. Підведення підсумків заняття

Викладач перевіряє роботу застосунків, звертає увагу на типові помилки (дієслова в URI, 200 замість 201/204, `th:utext` для даних користувачів, `BindingResult` не на своєму місці, `int` замість `Integer` у DTO, сутності моделі в API, стек викликів у відповіді з помилкою, лише браузерна валідація), повідомляє, що в ЛР 8 доступ до даних буде переведено на Spring Data JPA, додано міграції Flyway, Spring Security і автоматичні тести.

Методичну розробку розробив:

Доцент кафедри ІТтаСЕК, к.т.н. Роман ГОЛОВАТИЙ

Методична розробка обговорена на засіданні кафедри ІТтаСЕК
Протокол № ____ від «____» _____ 20___ р.