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

МЕТОДИЧНА РОЗРОБКА

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

Лекція № 4. Тема 1.4: «Вебсервери та контейнери сервлетів»

Мета: сформувати у здобувачів освіти розуміння ролі вебсервера і контейнера сервлетів у роботі Java-вебзастосунку, знання Servlet API, життєвого циклу сервлета, механізмів сесій, фільтрів і слухачів, а також навички побудови та розгортання вебзастосунку на Apache Tomcat.

Час: 2 години.

Місце заняття: лекційна аудиторія (або онлайн у MS Teams).

Матеріальне забезпечення: мультимедійний проєктор, ПК з доступом до мережі Internet, IntelliJ IDEA, JDK 21, Maven, браузер із засобами розробника, презентація до лекції, електронний курс у «Віртуальному університеті».

Навчальні питання:

1. Вебсервер, сервер застосунків і контейнер сервлетів. Apache Tomcat.
2. Servlet API. Життєвий цикл сервлета.
3. Обробка запитів і форм. Переадресація, шаблон Post/Redirect/Get.
4. Представлення на JSP: мова виразів EL і бібліотека JSTL.
5. Стан у протоколі HTTP: cookies і сесії.
6. Фільтри і слухачі. Автентифікація та авторизація.
7. Структура вебзастосунку та його розгортання (WAR, web.xml).

Після лекційного заняття здобувачі освіти повинні:

знати: призначення вебсервера і контейнера сервлетів, архітектуру Apache Tomcat, основні інтерфейси Servlet API, етапи життєвого циклу сервлета, відмінність forward від redirect, механізм сесій на основі cookies, призначення фільтрів і слухачів, структуру WAR-архіву;

вміти: створювати сервлети й зіставляти їх з URL, обробляти параметри запиту і форми, формувати сторінки за допомогою JSP та JSTL, зберігати дані в сесії, реалізовувати фільтр авторизації, запускати застосунок на вбудованому Tomcat і розгортати WAR-архів на окремому сервері.

Література:

1. Віртуальне навчальне середовище ЛДУ БЖД «Віртуальний університет» [Електронний ресурс]. – Доступний з: https://virt.ldubgd.edu.ua/course/view.php?id=2111
2. Jakarta Servlet Specification, Version 6.0 [Електронний ресурс]. – Eclipse Foundation, 2022. – Доступний з: https://jakarta.ee/specifications/servlet/6.0/
3. Apache Tomcat 10.1 Documentation [Електронний ресурс]. – Apache Software Foundation. – Доступний з: https://tomcat.apache.org/tomcat-10.1-doc/
4. Murach J., Urban M. Murach’s Java Servlets and JSP. 3rd ed. – Mike Murach & Associates, 2014. – 758 p.
5. OWASP Session Management Cheat Sheet [Електронний ресурс]. – Доступний з: https://cheatsheetseries.owasp.org/

I. Підготовка до заняття

Перед початком заняття викладач перевіряє роботу проєктора, доступ до демо-проєкту (тег `lab-04`), наявність запущеної БД PostgreSQL з ЛР 2 і можливість запуску застосунку командою `mvn compile exec:exec`.

ІІ. Зміст лекції

Вступ

У лабораторній роботі № 3 застосунок уже мав шари контролера, сервісу і репозиторію, але сторінки генерувалися у файли методом `main`. Щоб браузер міг надсилати запити і отримувати відповіді в реальному часі, Java-код потрібно розмістити на **сервері**, який слухає мережевий порт, розбирає HTTP-запити і передає їх нашим класам. Для Java цю роль виконує **контейнер сервлетів**. На цій темі ґрунтуються всі сучасні Java-фреймворки: Spring MVC, який ми вивчатимемо в модулі 2, теж працює всередині контейнера сервлетів.

1. Вебсервер, сервер застосунків і контейнер сервлетів

Вид сервера	Призначення	Приклади
Вебсервер	віддає статичні файли (HTML, CSS, зображення), виконує роль зворотного проксі, шифрування TLS	Nginx, Apache HTTP Server, jwebserver (ЛР 1)
Контейнер сервлетів	виконує Java-код вебзастосунку: сервлети, фільтри, JSP; керує їх життєвим циклом і сесіями	Apache Tomcat, Jetty, Undertow
Сервер застосунків	повна платформа Jakarta EE: контейнер сервлетів, транзакції, черги повідомлень, EJB	WildFly, GlassFish, Payara, Open Liberty

Таблиця 1 – Види серверів для вебзастосунків

Apache Tomcat – найпоширеніший контейнер сервлетів з відкритим кодом. Версія 10.1 реалізує специфікацію Jakarta Servlet 6.0 (пакети `jakarta.servlet.*`; до версії 10 пакети називалися `javax.servlet.*`). Основні компоненти Tomcat: **Connector** (Coyote) приймає TCP-з’єднання на порту 8080 і розбирає HTTP; **Engine** (Catalina) обробляє запити; **Host** – віртуальний хост (наприклад, localhost); **Context** – окремий вебзастосунок; **Wrapper** – окремий сервлет. Компілятор **Jasper** перетворює JSP-сторінки на сервлети. Spring Boot містить Tomcat у вигляді бібліотеки (*embedded*), тому окремо встановлювати сервер не потрібно.

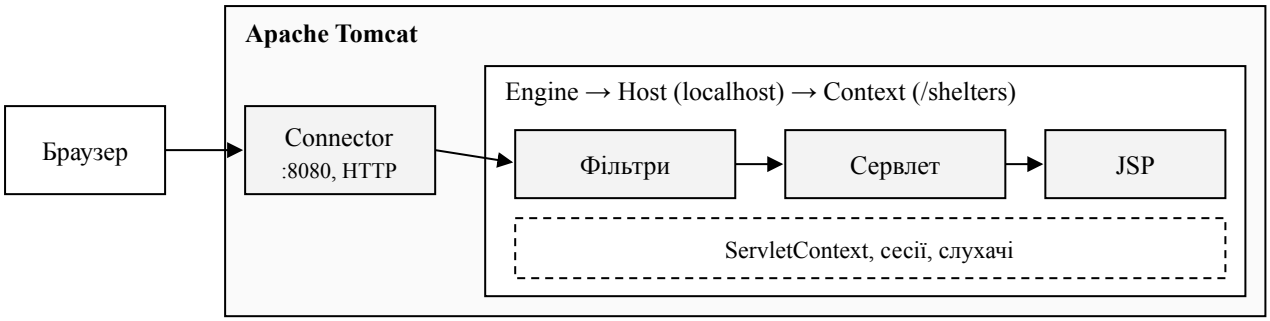

Рисунок 1 – Шлях HTTP-запиту всередині Apache Tomcat

2. Servlet API. Життєвий цикл сервлета

> **Сервлет** – Java-клас, що виконується в контейнері, приймає HTTP-запит (`HttpServletRequest`) і формує HTTP-відповідь (`HttpServletResponse`). Зазвичай сервлет успадковує клас `HttpServlet` і перевизначає методи `doGet`, `doPost`, `doPut`, `doDelete`.

Контейнер керує **життєвим циклом** сервлета: (1) завантажує клас і створює **один** екземпляр; (2) один раз викликає `init()` – тут готують ресурси; (3) для кожного запиту в окремому потоці з пулу викликає `service()`, який за HTTP-методом передає керування `doGet()`, `doPost()` тощо; (4) перед зупинкою застосунку викликає `destroy()`. Оскільки один екземпляр сервлета обслуговує багато запитів одночасно, **у полях сервлета не можна зберігати дані окремого запиту чи користувача** – лише спільні незмінні об’єкти (наприклад, посилання на сервіс). Дані запиту зберігають у локальних змінних, дані користувача – у сесії.

```
@WebServlet("/shelters")            // зіставлення з URL
public class ShelterServlet extends HttpServlet {
    private ShelterService service;  // спільний для всіх

    @Override
    public void init() {             // один раз
        service = (ShelterService) getServletContext()
            .getAttribute(AppContextListener.SHELTER_SERVICE);
    }

    @Override
    protected void doGet(HttpServletRequest req,
            HttpServletResponse resp)
            throws ServletException, IOException {
        req.setAttribute("shelters", service.findAll());
        req.getRequestDispatcher(
            "/WEB-INF/views/shelters.jsp").forward(req, resp);
    }
}
```

Лістинг 1 – Сервлет списку укриттів (демо-проєкт, тег lab-04)

Основні методи `HttpServletRequest`: `getParameter(name)` – параметр рядка запиту або форми; `getMethod()`, `getRequestURI()`, `getHeader(name)`; `getSession()` – сесія; `setAttribute()` – передати об'єкт представленню. Методи `HttpServletResponse`: `setStatus(code)`, `setContentType()`, `getWriter()`, `sendRedirect(url)`, `sendError(code, message)`.

3. Обробка запитів і форм. Шаблон Post/Redirect/Get

Дані HTML-форми з методом POST надходять у тілі запиту у форматі `application/x-www-form-urlencoded` (`name=…&capacity=250`). Усі параметри – рядки, тому сервлет перетворює їх на потрібні типи і передає сервісу для валідації. Кодування UTF-8 слід встановити **до** першого читання параметра, інакше кирилиця буде спотворена.

	forward	**redirect**
Виклик	`req.getRequestDispatcher(path).forward(req, resp)`	`resp.sendRedirect(url)`
Де відбувається	усередині сервера, браузер не знає	сервер повертає код 302 і заголовок `Location`, браузер надсилає новий GET
URL у браузері	не змінюється	змінюється
Атрибути запиту	зберігаються	втрачаються (потрібна сесія)
Застосування	показ JSP-представлення, помилки валідації	після успішного POST (шаблон PRG)

Таблиця 2 – Порівняння forward і redirect

Шаблон **Post/Redirect/Get** (PRG): після успішної обробки POST сервер не повертає сторінку, а перенаправляє браузер на GET-адресу списку. Тоді оновлення сторінки (F5) не надішле форму повторно і не створить дублікат запису. Повідомлення «Укриття додано» передають через сесію і показують один раз (*flash-повідомлення*). У разі помилок валідації сервлет встановлює код 400 і виконує forward на ту саму форму, зберігаючи введені значення.

```
try {
    var created = service.create(form);
    req.getSession().setAttribute("flash",
        "Укриття «" + created.name() + "» додано");
    resp.sendRedirect(req.getContextPath() + "/shelters");
} catch (ValidationException e) {
    resp.setStatus(HttpServletResponse.SC_BAD_REQUEST);
    req.setAttribute("form", form);    // введені дані
    render(req, resp, e.getErrors());  // forward на JSP
}
```

Лістинг 2 – Обробка форми за шаблоном PRG

4. Представлення на JSP: EL і JSTL

JSP (Jakarta Server Pages) – HTML-шаблон із вставками, який Tomcat (компілятор Jasper) перетворює на сервлет під час першого звернення. Сучасний стиль JSP не містить Java-коду: дані виводять мовою виразів **EL** (`${shelters.size()}`, `${sessionScope.user}`), а умови й цикли – тегами **JSTL** (`<c:if>`, `<c:forEach>`, `<c:choose>`). Тег `<c:out>` екранує символи `< > & " '` і захищає від XSS. Представлення розміщують у каталозі `WEB-INF/views`: файли з `WEB-INF` недоступні напряму з браузера, їх можна відкрити лише через forward із сервлета, тобто з підготовленими даними.

```
<%@ taglib prefix="c" uri="jakarta.tags.core" %>
<c:forEach var="s" items="${shelters}">
  <tr>
    <td><c:out value="${s.name()}"/></td>
    <td>${s.capacity()}</td>
  </tr>
</c:forEach>
```

Лістинг 3 – Виведення таблиці укриттів засобами JSTL

5. Стан у протоколі HTTP: cookies і сесії

HTTP не зберігає стан: кожен запит незалежний. Щоб «пам’ятати» користувача, сервер використовує **cookies** – пари «ім’я=значення», які сервер надсилає заголовком `Set-Cookie`, а браузер повертає в кожному наступному запиті заголовком `Cookie`. **Сесія** (`HttpSession`) – сховище атрибутів на сервері, прив’язане до користувача через cookie `JSESSIONID`. Метод `req.getSession()` повертає наявну сесію або створює нову; `req.getSession(false)` – лише наявну або `null`.

Правила безпеки сесій (OWASP): cookie сесії з атрибутами `HttpOnly` (недоступна для JavaScript) і `Secure` (лише HTTPS); обмежений час бездіяльності (у демо-проєкті – 30 хв); **новий ідентифікатор сесії після входу** –

захист від атаки фіксації сесії; знищення сесії під час виходу (`session.invalidate()`); у сесії зберігають мінімум даних (об'єкт користувача без пароля).

```
HttpSession old = req.getSession(false);
if (old != null) old.invalidate();    // захист від фіксації
HttpSession session = req.getSession(true);
session.setAttribute("user", user.get());
session.setMaxInactiveInterval(30 * 60);
resp.sendRedirect(req.getContextPath() + "/shelters");
```

Лістинг 4 – Створення сесії після успішного входу (LoginServlet)

6. Фільтри і слухачі. Автентифікація та авторизація

Фільтр (`Filter`, `HttpFilter`) перехоплює запити до сервлетів за шаблоном URL і виконує спільну для багатьох сторінок дію: встановлення кодування, журналювання, перевірку доступу, стиснення. Фільтри утворюють **ланцюжок**: виклик `chain.doFilter(req, resp)` передає запит далі, а якщо його не викликати – запит зупиняється у фільтрі. **Автентифікація** – перевірка, хто користувач (логін і пароль); **авторизація** – перевірка, що йому дозволено (роль). У демо-проєкті перегляд списку відкритий для всіх, зміни дозволено після входу, а видаляти укриття може лише роль ADMIN (інакше – код 403).

```
@WebFilter(urlPatterns = {"/shelters", "/shelters/*"})
public class AuthFilter extends HttpFilter {
  @Override
  protected void doFilter(HttpServletRequest req,
      HttpServletResponse resp, FilterChain chain)
      throws IOException, ServletException {
    boolean modifying = !"GET".equals(req.getMethod());
    HttpSession s = req.getSession(false);
    boolean loggedIn = s != null
        && s.getAttribute("user") != null;
    if (modifying && !loggedIn) {
      resp.sendRedirect(req.getContextPath() + "/login");
      return;                          // далі не передаємо
    }
    chain.doFilter(req, resp);
  }
}
```

Лістинг 5 – Фільтр авторизації

Слухачі (`@WebListener`) реагують на події контейнера: `ServletContextListener` – запуск і зупинка застосунку, `HttpSessionListener` – створення і знищення сесії. У демо-проєкті `AppContextListener` під час запуску створює сервіси й репозиторії та зберігає їх в атрибутах

`ServletContext` – спільному для всього застосунку сховищі. Це «ручне» впровадження залежностей; у Spring цю роботу виконує IoC-контейнер (тема 2.1).

7. Структура вебзастосунку та розгортання

Вебзастосунок пакують у **WAR** (Web Application Archive) – ZIP-архів зі стандартною структурою: кореневий каталог із загальнодоступними файлами (CSS, зображення, `index.jsp`), каталог `WEB-INF` з дескриптором `web.xml`, скомпільованими класами (`WEB-INF/classes`) і бібліотеками (`WEB-INF/lib`). У Maven для цього вказують `<packaging>war</packaging>`, а залежність Servlet API позначають `scope=provided`, бо її надає сам Tomcat. Дескриптор `web.xml` містить налаштування, які не задаються анотаціями: сторінку привітання, параметри сесії, сторінки помилок для кодів 403, 404, 500.

```
mvn compile exec:exec       # вбудований Tomcat (DevServer)
                            # http://localhost:8080/shelters
mvn package                 # target/shelters.war
cp target/shelters.war $CATALINA_HOME/webapps/
$CATALINA_HOME/bin/startup.sh   # /shelters/shelters
```

Лістинг 6 – Два способи запуску демо-проєкту

Під час розгортання Tomcat розпаковує WAR і створює **контекст** з шляхом, що збігається з назвою архіву (`/shelters`); тому в посиланнях використовують `req.getContextPath()`, а не жорстко задані адреси. Журнали сервера містяться в каталозі `logs` (`catalina.out`), а для налагодження зручно запускати вбудований Tomcat безпосередньо з IDE.

Висновки

Контейнер сервлетів приймає HTTP-запити і передає їх Java-коду вебзастосунку. Сервлет існує в одному екземплярі, тому зберігає лише спільні дані; дані користувача зберігаються в сесії, яку пов’язує з браузером cookie `JSESSIONID`. Шаблон PRG запобігає повторному надсиланню форм, фільтри реалізують наскрізні дії, зокрема авторизацію, а слухачі – ініціалізацію застосунку. Розібрані механізми лежать в основі Spring MVC: `DispatcherServlet` є звичайним сервлетом, а Spring Security – ланцюжком фільтрів (модуль 2).

Контрольні запитання

1. Чим контейнер сервлетів відрізняється від вебсервера і сервера застосунків?
2. Назвіть основні компоненти Apache Tomcat і їх призначення.
3. Опишіть етапи життєвого циклу сервлета.
4. Чому в полях сервлета не можна зберігати дані окремого користувача?
5. Чим forward відрізняється від redirect? Коли застосовують кожен з них?
6. У чому полягає шаблон Post/Redirect/Get і яку проблему він розв’язує?
7. Як працює сесія в HTTP? Які правила безпеки сесій ви знаєте?
8. Яке призначення фільтрів і ланцюжка фільтрів?
9. Яка структура WAR-архіву? Навіщо Servlet API позначають scope=provided?

Завдання для самостійної роботи

Опрацювати матеріал лекції. Встановити Apache Tomcat 10.1 і розгорнути на ньому WAR-архів демо-проєкту. Ознайомитися з рекомендаціями OWASP щодо керування сесіями. Пройти тест № 4 у «Віртуальному університеті».

III. Підведення підсумків заняття

Узагальнити основні положення лекції, відповісти на запитання здобувачів освіти. Наголосити на зв’язку матеріалу з лабораторною роботою № 4 і на тому, що наступний модуль (Spring Boot) спирається на розглянуті механізми. Видати завдання для самостійної роботи.

Методичну розробку розробив:

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

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