Aplikacje webowe w Javie - serwlety, JSP i JSTL

Tworzenie projektu w Netbeans

- Stwórz nowy projekt typu **Java Web** / **Web Application**
- W trzecim kroku kreatora (**Server and Settings**) wybierz:
  - w polu **Server**: **GlassFish**
  - w polu **Context Path**: `/lab4` (będzie to część URL, ścieżka do naszej aplikacji - pełny adres będzie wówczas wyglądał np. tak `http://localhost:8080/lab4`)
- W kroku 4 nie zaznaczaj żadnych frameworków / bibliotek, kliknij **Finish**
- Po zakończeniu kreatora powinieneś zobaczyć stronę główną `index.html` - jest to zwykły, statyczny plik HTML. Zmień jego zawartość np. na taką:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Java Web - laboratorium 4</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h1>Witamy</h1>
    </body>
</html>
```

- Uruchom projekt. W tle powinien wystartować serwer GlassFish (za pierwszym razem może to chwilę potrwać), a po chwili w oknie przeglądarki powinieneś zobaczyć stronę główną. Zapamiętaj:
- na jakim porcie został uruchomiony serwer (domyślnie: 8080)
- jaki jest context path i pełny adres projektu (domyślnie: http://localhost:8080/lab4/)

Pierwszy serwlet

Dodajmy do projektu pierwszy serwlet. Jego zadaniem będzie wyświetlenie **Hello World**.

- Wybierz polecenie **File** -> **New File** -> kategoria **Web**, typ pliku **Servlet**
- W drugim kroku:
  - klasę serwletu nazwij `HelloServlet`
  - w polu **Package** wpisz `tpsi.web`
- W trzecim kroku:

  - **Servlet Name** - jest to nazwa, której używać będziemy w adnotacjach i plikach konfiguracyjnych; po tej nazwie serwer identyfikuje dany servlet; zostawmy wartość bez zmian, `HelloServlet`;
  - **URL Pattern** - jest to URL, który będzie obsługiwany przez dany serwlet; zmieńmy wartość na `/hello`

- Przejrzyj wygenerowany kod serwletu. Znajdziesz w nim metody:
  - `doGet()` i `doPost()` - wywoływane przez serwer, jeśli na adres `/hello` nadejdzie żądanie HTTP, odpowiednio metodą GET lub POST;
  - `processRequest()` - nie jest to część standardu, lecz wymysł Netbeans; metody `doGet()` i `doPost()` wywołują `processRequest()`, także możemy w jednym miejscu obsłużyć żądanie, bez względu na użytą metodę (GET lub POST).

  oraz adnotację:
  - `@WebServlet` - oznacza ona klasę jako serwlet i informuje serwer aplikacji, że należy go zainicjować przy starcie aplikacji; wskazuje też nazwę serwletu (`name = "HelloServlet"`) oraz ścieżki URL, które mają być przez ten serwlet obsługiwane (`urlPatterns = {"/hello"}`)

- Uprośćmy nieco kod, pozbywając się zbędnych metod. **Zastąp cały** kod serwletu następującym:

```
package tpsi.web;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");

        try (PrintWriter out = response.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Java Web - laboratorium 4</title>");
```

```
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet mówi Hello</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
}
```

- Dodajmy na stronie głównej link do nowo utworzonego serwletu - na stronie `index.html` dopisz:

  ```
  <p><a href="/lab4/hello">HelloServlet</a></p>
  ```

- Uruchom projekt, sprawdź działanie serwletu.

Obsługa formularzy

Przygotowanie formularza

- Na stronie głównej `index.html` zamiast linka do serwletu umieśćmy prosty formularz:

  ```
  <form action='/lab4/hello' method='post'>
      Przedstaw się ładnie:
      <input type='text' name='imie'>
      <input type='submit'>
  </form>
  ```

W polu `action` mamy URL serwletu, dane prześlemy metodą POST.

- Uruchom projekt, wpisz imię i wyślij formularz - co się stało?

  Dlaczego nie działa?

  ... *chwila na zastanowienie* ...

  ...

  ... *i jeszcze jedna* ...

  ...

  Mamy błąd `HTTP Status 405 - HTTP method POST is not supported by this URL`. W naszym serwlecie `HelloServlet` mamy tylko metodę `doGet()`, nie mamy `doPost()`. Zatem nasz serwlet nie obsługuje metody HTTP POST, tylko GET.

- Napraw błąd zmieniając w serwlecie `HelloServlet` nazwę metody `doGet()` na `doPost()`.

Odebranie danych w serwlecie

- W serwlecie odczytamy imię wpisane w formularzu i wypiszemy stosowne powitanie.
- W metodzie `doPost()` odczytaj parametr `imie` przesłany w żądaniu:

  ```
  String imie = request.getParameter("imie");
  ```

  i zmień wypisywany komunikat - zamiast:

  ```
  out.println("<h1>Servlet mówi Hello</h1>");
  ```

  wpisz:

  ```
  out.println("<h1>Cześć, " + imie + "!</h1>");
  ```

- Uruchom projekt, sprawdź działanie formularza.

Strony JSP (*Java Server Pages*)

- Dodamy do projektu stronę JSP:
  - **File** -> **New File** -> kategoria **Web**, typ pliku **JSP**
  - nazwij stronę `hello.jsp`
- Zastąp kod strony następującym:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Java Web - laboratorium 4</title>
    </head>
    <body>
        <h1>Witamy w JSP!</h1>
    </body>
</html>
```

- Na stronie głównej dodaj link do nowej strony `hello.jsp`:

  ```
  <a href="hello.jsp">hello.jsp</a>
  ```

- Przetestuj działanie.

Przekierowanie z serwletu do JSP

Powiążemy teraz serwlet `HelloServlet` ze stroną `hello.jsp`. Zamiast wypisywać w serwlecie zawartość HTML przez `out.println("...")`, przygotujemy dane do wyświetlenia i przekażemy żądanie do strony JSP.

- W kodzie serwletu `HelloServlet`:

  - usuń cały kod związany z generowaniem odpowiedzi (ustawienie *content type*, cały blok `try` z wypisywaniem kodu HTML)
  - zostaw kod pobierający imię z żądania (`request.getParameter("imie")`)
- Zadaniem serwletu będzie przygotowanie danych do wyświetlenia przez JSP. Dane przekazujemy do strony JSP jako **atrybuty** żądania:

  ```
  request.setAttribute("imie", imie);
  ```
- Wreszcie przekierujmy żądanie do obsługi przez stronę `hello.jsp`:

  ```
  request.getRequestDispatcher("hello.jsp").forward(request, response);
  ```
- Ze strony głównej usuń link do strony `hello.jsp`
- Po wysłaniu formularza z imieniem żądanie zostanie wysłane do serwletu, przetworzone, a następnie przekierowane do `hello.jsp` - powinniśmy zobaczyć zatem zawartość `hello.jsp`.

Wyświetlenie danych na stronie JSP

Proste wyrażenia EL

Pozostaje wyświetlić na stronie JSP dane przekazane z serwletu. Użyjemy w tym celu wyrażeń EL (wyrażeń zapisanych w języku o oryginalnej nazwie *Expression Language*...).

- Wyrażenia EL poprzedzamy znakiem dolara, samo wyrażenie może odnosić się do atrybutów przekazanych z serwletu. Przykłady:

  ```
  ${idProduktu}

  ${dniTygodnia[3]}

  ${produkt.suma}
  ```
- Zastąpmy komunikat na stronie `hello.jsp` następującym kodem:

  ```
  <h1>Cześć, ${imie}!</h1>
  ```

  W miejsce `${imie}` zostanie podstawiony atrybut o nazwie `imie` przekazany z serwletu.

Wyświetlanie obiektów

Jako atrybut żądania możemy przekazać nie tylko proste łańcuchy tekstowe, ale także złożone obiekty. Wyrażenia EL pozwalają na dostęp do pól takich obiektów, pod warunkiem, że spełniają one założenia **Java Beans** (odpowiednie gettery i settery).

Wyrażenie:

`${produkt.cena}`

wyświetli nam cenę produktu - pobraną za pomocą metody `getCena()`. Nie jest przy tym istotne, czy w obiekcie `produkt` istnieje pole o nazwie `cena` - ważny jest wyłącznie publiczny getter o nazwie `getCena()`.

Zadanie 1

- Dodaj do projektu klasę `Person` (osoba opisana przez `firstName`, `lastName`, `email`).
- Rozbuduj formularz na stronie głównej, tak by zawierał powyższe trzy pola.
- W serwlecie `HelloServlet` na podstawie danych z formularza stwórz obiekt klasy `Person`. Obiekt ten przekaż jako atrybut do wyświetlenia na stronie JSP.
- Na stronie `hello.jsp` wyświetl powitanie w formie "Witaj, Jan Kowalski!" oraz link do emaila (postaci `mailto:jan.kowalski@poczta.pl`).

Biblioteka JSTL

Biblioteka JSTL rozszerza JSP o zestaw pomocnych tagów. Znajdziemy w niej między innymi:

- wyświetlanie danych kodowaniem encji HTML (*HTML entity escaping*; zabezpieczenie przed atakami XSS),
- warunkowe wyświetlanie bloków HTML (odpowiednik `if`),
- pętle,
- formatowanie liczb i dat.

Dodanie biblioteki do projektu i do strony JSP

- Musimy dodać bibliotekę JSTL do projektu:
  - w drzewku projektu znajdź pozycję **Libraries**, kliknij na niej prawym klawiszem, wybierz **Add Library**;
  - z listy bibliotek wybierz **JSTL 1.2.1**
- Na każdej stronie JSP, na której chcemy korzystać z JSTL, musimy zadeklarować odpowiednie biblioteki tagów za pomocą dyrektywy `taglib`:
  - na stronie `hello.jsp` na górze, zaraz po dyrektywie `@page`, dodaj:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

- Zadeklarowaliśmy użycie biblioteki JSTL Core, zwyczajowo używa się dla niej prefiksu `c` (wszystkie tagi z tej biblioteki poprzedzane będą `c:`, np. `c:out`)

Wyświetlanie danych z użyciem c:out

- Sprawdźmy działanie JSTL - użyjmy tagu `c:out` zamiast zwykłego wyrażenia EL. W kodzie strony `hello.jsp` zamiast:

  ```
  <h1>Cześć, ${imie}!</h1>
  ```

  wpisz:

  ```
  <h1>Cześć, <c:out value="${imie}"/>!</h1>
  ```

  Jeśli strona działa tak samo, jak wcześniej - JSTL działa prawidłowo.

Zadanie 2

- Porównaj działanie obu wersji strony: pierwszej, z wyrażeniem `${imie}` i drugiej, z tagiem `c:out`.
- Co się stanie w obu wersjach, jeśli jako imie wpiszemy w formularzu tekst ze znacznikami HTML, np.:
  - `<span style='color: pink'>Piotr</span>`
  - `<script>alert('Jestem zlym hackerem i zaraz cie zjem!');</script>`

Wyświetlanie danych w pętli

Z serwletu do strony JSP możemy przekazać tablicę lub listę obiektów. Przetestujmy wypisywanie zawartości takiej listy z użyciem pętli JSTL.

- W serwlecie `HelloServlet` stwórzmy listę dni tygodnia:

  ```
  List<String> dni = new ArrayList<>();
  dni.add("Poniedziałek");
  dni.add("Wtorek");
  dni.add("Środa");
  dni.add("Czwartek");
  dni.add("Piątek");
  ```

- Przekażmy listę do strony JSP jako atrybut żądania:

  ```
  request.setAttribute("dniTygodnia", dni);
  ```

- Na stronie `hello.jsp` wypiszmy dni tygodnia używając pętli `forEach`:

  ```
  <c:forEach items="${dniTygodnia}" var="dzien">
      <p>
          ${dzien}
      </p>
  </c:forEach>
  ```

Zadanie 3

- Dodaj do projektu nowy serwlet `PersonListServlet`, skojarz go z adresem URL `/personList`.
- Na stronie głównej dodaj link do serwletu.
- W serwlecie stwórz listę i dodaj do niej kilka obiektów `Person` (dowolnych, z danymi wpisanymi na sztywno).
- Do wyświetlenia listy osób stwórz stronę JSP `personList.jsp`.
- Używając JSTL wyświetl tabelkę HTML z listą osób.