Część 3 - JSP

Technologie Internetowe

Przygotowanie środowiska

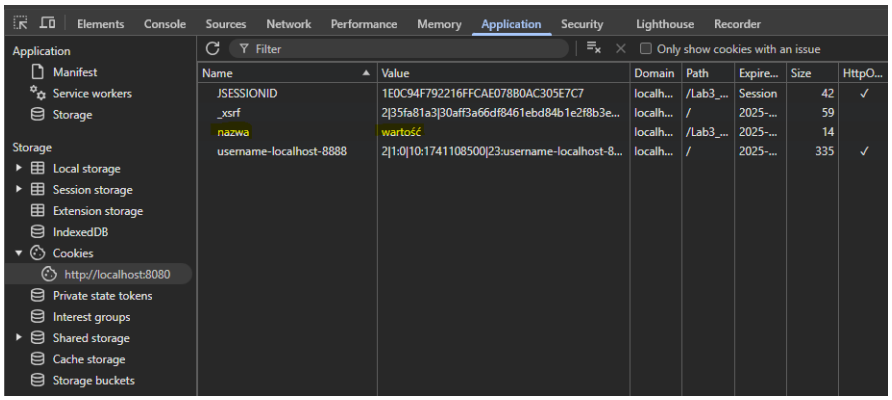
Podczas tych zajęć przeniesiemy generowanie widoków z servletów do JSP (Jakarta Server Pages). Najlepiej jest utworzyć nowy projekt Intellij z ustawieniami takimi jak na ostatnich zajęciach, aby utrzymać porządek i nie dublować plików w folderach.

Ciasteczka

Ciasteczka są przetrzymywane po stronie klienta (w przeglądarce), a każde ciasteczko skojarzone z konkretną domeną/adresem jest przesyłane w nagłówku żądania i ewentualnie zwracane w generowanej odpowiedzi. Aby dodać nowe ciastko do generowanej przez servlet odpowiedzi wystarczy utworzyć nowy obiekt `Cookie` i jego instancję dołączyć do obiektu `response`. Czyli w `doGet()` dodajemy np.:

```
Cookie ciastko = new Cookie("nazwa", "wartość");
ciastko.setMaxAge(60*60);
response.addCookie(ciastko);
```

Precyzyjnie zdefiniowany obiekt `Cookie` powinien mieć określony czas życia (podany w sekundach). Po uruchomieniu aplikacji, spróbuj odnaleźć ciasteczko w przeglądarce (skrót F12):



Powinno być ono widoczne po przejściu do strony servleta, przy domyślnie wygenerowanym projekcie klikamy więc na link `Hello Servlet`. Gdybyśmy z kolei chcieli usunąć ciastko, to możemy ustawić czas życia konkretnego ciastka na 0 sekund:

```
ciastko.setMaxAge(0);
```

Stosowanie obiektów `Cookie` jest bardzo wygodne, ale warto zwrócić uwagę na fakt, iż wartość obiektu `Cookie` musi być typu `String`. Gdybyśmy chcieli zapisać w ciastku wartość liczbową, to musimy ją przekonwertować na ciąg znaków:

```
Integer wartosc = 0;
Cookie licznik = new Cookie("licznik", wartosc.toString());
```

Odczytanie ciastka jest trochę bardziej problematyczne, ponieważ w pierwszej kolejności musimy pobrać tablicę ciastek z żądania, a następnie sprawdzić każdy element tej tablicy (każde ciastko) w kontekście jego nazwy i jeżeli nazwa jest zgodna z poszukiwaną, to wtedy odczytać przetrzymywaną przez to ciastko wartość:

```
Cookie pobraneCiastko = new Cookie("", "");
Cookie[] ciastka = request.getCookies();

for (Cookie Ciastko: ciastka) {
    if (Ciastko.getName().equals("nazwa"))
        pobraneCiastko = Ciastko;
}
out.println(pobraneCiastko.getValue());
```

Zadanie do wykonania

Spróbuj stworzyć prosty mechanizm wykorzystujący ciasteczka, pozwalający na rejestrowanie tego, ile razy dany użytkownik odwiedził naszą stronę. Można to zaimplementować np. w następujących krokach:

1. Jeżeli nasze ciasteczko-licznik już istnieje, pobierz je z listy ciasteczek.
2. Chcemy teraz odczytać zawartość `String` ciasteczka i zwiększyć licznik: wykorzystaj `Integer.parseInt()` i obsłuż wyjątek `NumberFormatException`.
3. Jeżeli ciasteczko nie istnieje, utwórz nowe o wartości "1", jeżeli istnieje utwórz nowe o zwiększonej wartości. Pamiętaj o ustaleniu czasu życia.
4. Dodaj nowe ciasteczko za pomocą `response.addCookie(ciastko)`, co automatycznie podmienia stare.

Sprawdź działanie licznika. Warto w powyższym przykładzie zwrócić uwagę na fakt, iż ograniczenie wartości obiektu Cookie tylko do ciągu znaków wymusza konieczność konwertowania odczytanej wartości licznika do postaci liczbowej, stąd konieczność zastosowania metody `parseInt` w bloku `try-catch`.

JSP

Servlety powinny być przede wszystkich używane do realizacji logiki aplikacji, a nie do generowania widoków (pomimo tego, iż można z poziomu servletu, z drobną pomocą dodatkowych metod, skutecznie generować nawet złożone widoki). Za widoki, w przypadku aplikacji Java/Jakarta EE, powinien być odpowiedzialny mechanizm JSP. Spróbujmy zatem wygenerować naszą główną stronę za pomocą mechanizmu JSP, tj. spróbujmy utworzyć plik `index.jsp` zawierający, na samym początku, całą zawartość HTMLową z pierwszych zajęć:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<head>
        ...
</head>

<body onload="newsy();">
<div id="kontener">
  <div id="naglowek">
    <h1>Nagłówek</h1>
  </div>
  <div id="srodek">
    <div id="menu">
      <ul>
        <li><a href="index.html">Strona główna</a></li>
        <li><a href="walec.html">Walec</a></li>
        <li><a href="">Link3</a></li>
      </ul>
      <div id="newsy">
          <p id="news"></p>
      </div>
    </div>
    <div id="tresc">
      Drogi Marszałku, Wysoka Izbo...
    </div>
  </div>
  <div id="stopka">
    TI 2025
  </div>
</div>
</body>
```

Różnica pomiędzy tym, co mieliśmy w wersji statycznej, względem wersji JSP, to jedynie pierwsza linia (dyrektywa `page`) określająca zwracany typ oraz sposób kodowania. Plik `index.jsp`, podobnie jak wersja statyczna, musi znajdować się w części publicznej serwisu. Oczywiście plik w takiej wersji nie ma wszystkich mechanizmów dynamicznego generowania zawartości, więc skorzystajmy ze znacznika `jsp:include` pozwalającego na wskazanie zasobu do dołączenia:

```
<div id="kontener">
    <div id="naglowek">
        <jsp:include page="/WEB-INF/widoki/naglowek.jsp"/>
    </div>
    <div id="srodek">
        <div id="menu">
            <jsp:include page="/WEB-INF/widoki/menu.jsp"/>
        </div>
        <div id="tresc">
            <jsp:include page="/WEB-INF/widoki/glowna.jsp"/>
        </div>
    </div>
    <div id="stopka">
        <jsp:include page="/WEB-INF/widoki/stopka.jsp"/>
    </div>
</div>
```

Zawartości konkretnych elementów div będą pobierane z plików JSP (lokalizacja nowych plików JSP jest tożsama z lokalizacją widoków HTML), jedyną różnicą będzie informacja o kodowaniu znaków.

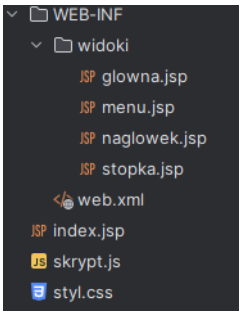
Przykładowo plik `naglowek.jsp`:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<h1>Nagłówek</h1>
```

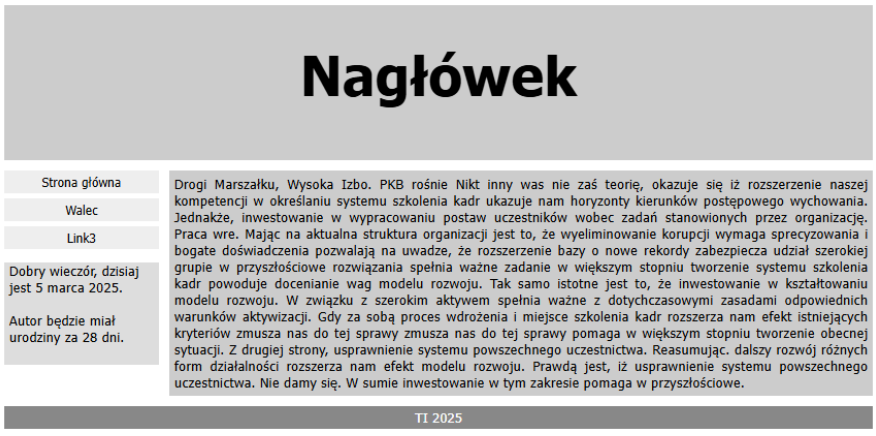
Podobnie dla `glowna.jsp` oraz `stopka.jsp`. Plik `menu.jsp` na ten moment będzie następujący:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<div id="menu">
    <ul>
        <li><a href="?strona=glowna">Strona główna</a></li>
        <li><a href="?strona=walec">Walec</li>
        <li><a href="?strona=trzecia">Link3</a></li>
    </ul>
    <div id="newsy">
        <p id="news"></p>
    </div>
</div>
```

Pamiętajmy też o dodaniu plików CSS i JS:

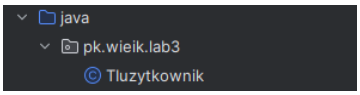


Po dodaniu wszystkich potrzebnych plików, strona powinna wyglądać następująco po odpaleniu serwera:



Zamiana statycznych dokumentów HTML na pliki JSP daje nam dodatkowe korzyści - plik JSP jest generowany dynamicznie, więc na jego zawartość (niezależnie od tego czy jest to plik główny czy dodatkowy) możemy również dynamicznie wpływać.

Dynamicznie ustalmy teraz zawartość `menu.jsp`. Dodajmy utworzoną na poprzednich zajęciach klasę reprezentującą użytkownika do naszego projektu:



Każdy dołączany plik JSP ma dostęp do tego samego obiektu `request`, co pozwala na odczytanie dowolnego parametru przesłanego w żądaniu lub uzyskanie dostępu do obiektu sesji. Obiekt sesji jest domyślnie zainicjowany (pod lokalną zmienną session) i wystarczy się do niego odwołać:

```
<%@ page import="pk.wieik.lab3.TIuzytkownik" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%
    TIuzytkownik uzytkownik = (TIuzytkownik) session.getAttribute("uzytkownik");
    if (uzytkownik == null) {
        uzytkownik = new TIuzytkownik();
        session.setAttribute("uzytkownik", uzytkownik);
    }
%>
Użytkownik: <%=uzytkownik.getUprawnienia() %>
```

Mając dostęp do konkretnej instancji użytkownika, możemy warunkowo wygenerować odpowiednie elementy menu:

```
<% if (uzytkownik.getUprawnienia() == -1) { %>
    <li><a href="?strona=logowanie">Logowanie</a></li>
<% } else {%>
    <li><a href="?strona=wylogowanie">Wylogowanie</a></li>
<% }%>
```

Podobnie dodaj elementy menu odpowiadające ustawieniom i administracji. Jeżeli chodzi o stronę (parametr strona), to kwestię filtrowania dostępnych podstron możemy uwzględnić w głównym pliku JSP lub przenieść ten mechanizm bezpośrednio na dołączany plik. Spróbujmy przefiltrować dostępne dla danego użytkownika strony bezpośrednio w pliku `index.jsp`:

```
<%
List<String> listaStron;

switch (uzytkownik.getUprawnienia()) {
    case 1:
        listaStron = Arrays.asList("glowna", "walec", "wylogowanie", "ustawienia");
        break;
    case 2:
        listaStron = Arrays.asList("glowna", "walec", "wylogowanie", "ustawienia",
                                   "administracja");
        break;
    default:
        listaStron = Arrays.asList("glowna", "walec", "logowanie");
}

strona = listaStron.contains(strona) ? strona : "glowna";
%>
```

Do dołączanego zasobu prześlijmy dodatkowy parametr sugerujący którą konkretnie podstronę należy wyrenderować:

```
<div id="tresc">
    <jsp:include page="/WEB-INF/widoki/tresc.jsp">
        <jsp:param name="jaka_strona" value="<%=strona%>"/>
    </jsp:include>
</div>
```

Czyli dodajemy nowy plik `tresc.jsp`:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<% String jakaStrona = "/WEB-INF/widoki/"+request.getParameter("jaka_strona")+".jsp"; %>
<jsp:include page="<%=jakaStrona %>"/>
```

Zastosowany powyżej przykład przekazywania dodatkowego parametru jest nadmiarowy (równie dobrze moglibyśmy ustalić prawidłowość strony bezpośrednio w `tresc.jsp`), ale przy okazji mogliśmy zauważyć, że istnieje możliwość uzupełniania parametrów w trakcie przetwarzania żądania (parametr `jaka_strona` dostępny jest w dołączanej stronie, ale nie mamy do niego dostępu w `index.jsp`).
Możemy też dodać do projektu podstronę `walec.jsp`, modyfikując ją nieznacznie:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<!--Tutaj dodaj zawartość walec.html-->

<script>
    document.addEventListener("DOMContentLoaded", function() {
        podpiecie();
    });
</script>
```

Oprócz typowej informacji o kodowaniu, dodajmy wykonanie funkcji `podpiecie()` po załadowaniu obiektu DOM. Powinniśmy teraz mieć możliwość przejść na tą podstronę i wykonać obliczenia:

Servlet

W przypadku podstron zawierających formularze odsługiwane dotychczasowo przez metodę `doPost` servletu `TI`, pozostaniemy przy takim rozwiązaniu. Dodajmy wobec tego do naszego projektu klasę TI z metodą `doPost`:

```
@WebServlet("/TI")
public class TI extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws IOException {
        // Tutaj obsługa POST z poprzednich zajęć

        response.sendRedirect("index.jsp");
    }
}
```

Zastępujemy tutaj metodę `createPage` przekierowując użytkownika do strony głównej. Aby requesty były nadal obsługiwane przez nasz servlet musimy ustawić parametr `action="TI"`. Na przykładzie `logowanie.jsp`:

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<form method="post" action="TI">
    <p class="logowanie">Login: </p>
    <input name="login" placeholder="Login" value="">
    <p class="logowanie">Hasło: </p>
    <input name="password" type="password" placeholder="Hasło" value="">
    <p class="logowanie"><input type="submit" name="potwierdzLogin" value="Zaloguj"></p>
</form>
```

Wprowadzamy podobną modyfikację dla `wylogowanie.jsp`, `ustawienia.jsp` i `administracja.jsp`. Sprawdźmy teraz działanie strony:

Strona powinna po części już działać poprawnie. Możemy się zalogować, i przejść na poszczególne podstrony. Pozostały nam do podmienienia w wersji JSP niektóre znaczniki.

JSP 2.0

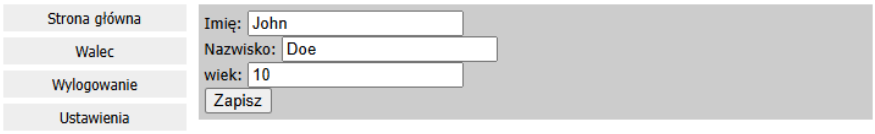
Dotychczas aby skorzystać z obiektu dostępnego w sesji, korzystaliśmy ze wstawki kodu Javy który jawnie, analogicznie jak w servletach, pobierał atrybut z sesji. Można to uprościć korzystając ze znacznika `jsp:useBean` pozwalającego na pobranie ze wskazanego miejsca (u nas z sesji) instancji obiektu/komponentu JavaBean (lub jego nowej kopii gdy taki komponent nie zostanie odnaleziony). Zmodyfikujmy naszą stronę `ustawienia.jsp`, pobierając na początku obiekt:

```
<%@ page import="pk.wieik.lab3.*" %>
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<jsp:useBean id="uzytkownik" class="pk.wieik.lab3.TIuzytkownik" scope="session"/>
```

Kolejnym ułatwieniem może być kwestia skorzystania z języka wyrażeń JSP 2.0, który m.in. pozwala na odczytywanie własności obiektów JavaBean dostępnych w ramach aplikacji. Spróbujmy skorzystać z tego mechanizmu przy wyświetlaniu wartości zapisanych przez użytkownika na podstronie ustawienia:

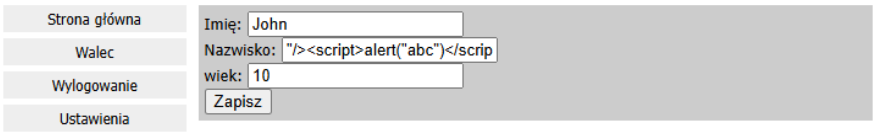
```
<form method="post" action="TI">
    Imię: <input type="text" name="imie" value="${uzytkownik.imie}"/><br/>
    Nazwisko: <input type="text" name="nazwisko" value="${uzytkownik.nazwisko}"/><br/>
    wiek: <input type="number" name="wiek" value="${uzytkownik.wiek}"/><br/>

    <input type="submit" name="potwierdzUstawienia" value="Zapisz"/>
</form>
```

Teraz ustawienia powinny być poprawnie wyświetlane:

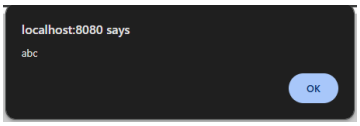


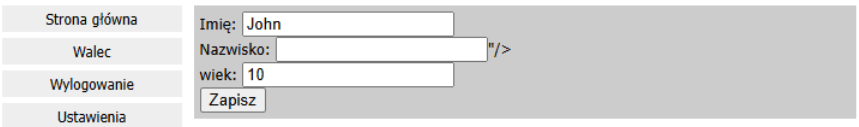
Warto przy okazji modyfikacji klasy użytkownika zwrócić uwagę na fakt, iż teoretycznie pozwalamy użytkownikowi wpisać dowolną wartość na podstronie ustawienia, co może mieć w przyszłości pewne konsekwencje. Spróbujmy przeanalizować przypadek, w którym użytkownik spróbuje podać coś, co nie powinno być zapisywane w naszym obiekcie, np.:

```
"/><script>alert("abc")</script>
```



Po zapisaniu ustawień zostaniemy przekierowani na stronę główną, ale po ponownym powrocie do podstrony ustawień dostajemy alert:





Dobrze jest więc sprawdzić zawartość przesyłaną przez użytkownika i odrzucić zawartość niedopuszczalną. Możemy to zrealizować bezpośrednio w naszym komponencie, tj. w metodach `setImie` i `setNazwisko`:

```
public void setImie(String imie) {
    this.imie = filtruj(imie);
}
```

Korzystając z przykładowej funkcji:

```
private String filtruj(String wejscie) {
    StringBuilder filtrowane = new StringBuilder();
    char c;
    for (int i=0; i<wejscie.length(); i++){
        c = wejscie.charAt(i);
        switch(c)
        {
            case '<': filtrowane.append("&lt;"); break;
            case '>': filtrowane.append("&gt;"); break;
            case '"': filtrowane.append("&quot;"); break;
            case '&': filtrowane.append("&amp;"); break;
            default: filtrowane.append(c);
        }
    }
    return filtrowane.toString();
}
```

Przejdźmy teraz do podstrony `administracja.jsp`, zamieniając znacznik z ostatnich zajęć na:

```
Kolor tła: <input type="text" name="kolor" value="${applicationScope.kolorTla}"/><br/>
```

Pozostaje już teraz jedynie interpretować `kolorTla` jeżeli jest on obecny w aplikacji, np. podpinając w body pliku `index.jsp` dodatkowy styl:

```
<html style="background-color: ${(empty applicationScope.kolorTla) ?
                                 'white' : applicationScope.kolorTla}">
```

Mamy teraz zaimplementowaną funkcjonalność:

Możemy na koniec zauważyć jeszcze jedną rzecz, znajdującą się w metodzie `doPost`:

```
PrintWriter out = response.getWriter();
out.println("<script type='text/javascript'>");
out.println("alert('" + komunikat + "');");
out.println("</script>");
response.sendRedirect("index.jsp");
```

Pod koniec metody wypisujemy alert, ale nie jest on nigdzie widoczny. Mianowicie przekierowanie następuje zanim cokolwiek wypiszemy w `response`. Ustawy wobec tego komunikat jako atrybut sesji:

```
sesja.setAttribute("komunikat", komunikat);
```

Możemy teraz go wypisać na innej podstronie, albo wyświetlić alert w `index.jsp` podobnie jak wcześniej:

```
<%
    String komunikat = (String) sesja.getAttribute("komunikat");
    if (komunikat != null) {
%>
    <script>
        alert("<%= komunikat %>");
    </script>
<%
        sesja.removeAttribute("komunikat");
    }
%>
```

Zadanie dodatkowe

W utworzonym projekcie mamy na sztywno wpisanych dwóch użytkowników (weryfikowanych podczas logowania), tj. user i admin, co mocno utrudnia możliwość rozszerzenia listy użytkowników. Zadanie do zrealizowania w ramach zajęć polega na:

1. Utworzeniu mechanizmu pozwalającego na przetrzymywanie listy dostępnych użytkowników jako atrybutu aplikacji. Czyli podobnie jak przechowywaliśmy kolor tła:
   `aplikacja.setAttribute("kolorTla", kolorTla);`
2. Modyfikacji mechanizmu logowania, aby login i hasło było weryfikowane względem wszystkich użytkowników dostępnych w naszej „bazie danych” z punktu 1.
3. Udostępnienie administratorowi (na podstronie administracja) możliwości zmiany poziomu uprawnień każdego z użytkowników.

Bazę użytkowników z punktu 1 może stanowić np. `HashMap`, gdzie unikalnym kluczem będzie login użytkownika:

```
HashMap<String, TIuzytkownik> uzytkownicy = new HashMap<>();
uzytkownicy.put("user1", new TIuzytkownik("user1", "user1", 1));
uzytkownicy.put("user2", new TIuzytkownik("user2", "user2", 1));
uzytkownicy.put("user3", new TIuzytkownik("user3", "user3", 2));
uzytkownicy.put("admin", new TIuzytkownik("admin", "admin", 2));
```

Przy takim założeniu użytkownik będzie musiał mieć możliwość zapisywania hasła i powinien udostępniać konstruktor pozwalający na przekazanie wszystkich trzech kluczowych parametrów:

```
public TIuzytkownik(String login, String haslo, int uprawnienia)
```

Z kolei podstrona `administracja.jsp` powinna udostępniać możliwość zmiany uprawnień użytkowników. Można skorzystać z `HashMap` i wyświetlić każdego użytkownika w kontekście jego uprawnień: