SOA – laboratorium nr 2

Temat: Servlety w JavaEE

Tematem lab są zagadnienia tworzenia prostych aplikacji web opartych o koncepcje servletu.

Część teoretyczna.

Servlety to klasy, których celem jest przetwarzanie żądań HTTP i generowanie zawartości która będzie odesłana w odpowiedzi na te żądania. Innymi słowy, servlety służą do implementacji dynamicznych aplikacji WWW.

Z założenia servlety mogą obsługiwać także protokoły inne niż HTTP, jednak w praktyce ich użycie ogranicza się niemal wyłącznie do implementacji aplikacji WWW opartych na HTTP. Możemy zatem myśleć o servletach jak o narzędziu służącym tylko i wyłącznie do tego właśnie celu.

Przechodząc do konkretów - servlety to klasy, które implementują interfejs javax.servlet.Servlet. Możemy zaimplementować servlet jako klasę implementującą ten interfejs, jednak łatwiej będzie nam zaimplementować klasę dziedziczącą z klasy javax.servlet.http.HttpServlet (która to klasa implementuje interfejs Servlet). Tak właśnie w praktyce robi się najczęściej. Przykład poniżej:

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

public class HelloWorldServlet extends HttpServlet {
  private static final long serialVersionUID = 1L;

  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
                                                          ServletException, IOException
 {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println("<html><head><title>Pierwszy Servlet</title></head><body>");
   out.println("Witaj na zajeciach z SOA!");
   out.println("<body></html>");

   out.close();
  }
}
```

Każde żądanie HTTP ma swój typ, definiujący akcję którą serwer powinien wykonać w odpowiedzi. Podstawowe typy żądań to żądania GET i POST. Żądanie GET oznacza, że klient wysyłający żądanie do serwera chce pobrać zasób wskazywany przez URI przekazany wraz z tym żądaniem. Żądanie POST oznacza, że klient wysyła do serwera, a konkretnie do zasobu wskazanego przez URI dołączony do żądania, pewną porcję danych i zleca przetworzenie tych danych. W każdym żądaniu przesyłane są dodatkowo (opcjonalnie) parametry, nagłówki, oraz ciasteczka. Każdy taki element żądania możemy pobrać w kodzie Servletu i wykorzystać zgodnie ze swoją niczym nie skrępowaną wolą.

Implementując klasę servletu implementujemy de facto metody obsługujące żądania HTTP. Aby zaimplementować obsługę metody HTTP GET musimy zaimplementować metodę doGet(…). Aby zaimplementować obsługę metody HTTP POST musimy zaimplementować metodę doPost(…). W powyższym przykładzie zaimplementowaliśmy jedynie metodę doGet(…), co oznacza że servlet ten obsługuje tylko i wyłącznie żądania HTTP typu GET.

Metody doGet(…) i doPost(…) przyjmują dwa parametry wywołania – obiekty reprezentujące żądanie i odpowiedź HTTP. Obiekty te są tworzone i przekazywane do implementowanego przez nas servletu przez serwer aplikacji.

Parametr reprezentujący żądanie HTTP to obiekt typu javax.servlet.http.HttpServletRequest. Obiekt reprezentujący odpowiedź to obiekt typu javax.servlet.http.HttpServletResponse. Z instancji reprezentującej żądanie odczytujemy szczegóły żądania (parametry, ciasteczka, nagłówki itd.) a do bufora związanego z instancją reprezentującą odpowiedź zapisujemy to, co ma być odesłane do klienta, czyli elementy strony WWW (np. HTML). W jaki sposób zapisujemy tę odpowiedź widzimy na powyższym przykładzie.
Żądanie HTTP, zarówno typu GET jak i POST, określa przede wszystkim URL zasobu do którego wysyłamy żądanie, ale może dodatkowo zawierać pewną liczbę parametrów. Wysyłając żądanie GET klient ma możliwość przekazania parametrów poprzez doklejenie ich w odpowiedni sposób do URL. W przypadku żądania POST parametry przekazywane są w treści właściwej żądania (której nie widzimy – tworzy ją za nas przeglądarka WWW).

Ogólna postać URL dla żądania GET jest następująca: http://{domena}:{port}/{zasób}?{parametry}
Zarówno {port} jak i {zasób} są opcjonalne. Opcjonalne są także {parametry}, ale jeśli chcemy je przekazać wraz z żądaniem, to doklejamy je do końca URL po znaku zapytania.
Składnia definicji pojedynczego parametru jest następująca: {nazwa}={wartość}

Element {nazwa} to nazwa parametru a {wartość} to jego wartość. Kolejne definicje parametrów oddzielamy od siebie znakiem &. Przykładowo, aby przekazać 3 parametry napisalibyśmy:
{nazwa}={wartość}&{nazwa}={wartość}&{nazwa}={wartość}

Żądania typu POST służą do wysyłania formularzy, a parametry takich żądań to nic innego jak wartości wpisane w pola formularza. Przykładowy formularz poniżej:

```
<form method="POST" action="http://test.pl/kalkulator">
   x: <input type="text" name="paramX">
   y: <input type="text" name="paramY">
   <input type="submit" value="Dodaj">
</form>
```

Wyświetlając powyższy formularz w ramach strony HTML przeglądarka wyświetli dwa pola tekstowe oraz przycisk którego kliknięcie spowoduje wysłanie żądania POST. Żądanie to będzie zawierało dwa parametry: parametr o nazwie paramX oraz paramY. Wartości tych parametrów będą takie, jakie wpiszemy w pola formularza.

Niezależnie od tego czy parametry przekazano w jeden czy drugi sposób, wraz z żądaniem GET czy POST, odczytujemy je w kodzie servletu w ten sam sposób, tzn. za pomocą metody String getParameter(String name) obiektu reprezentującego żądanie.
Parametry są zawsze typu String, tak więc aby np. zaimplementować servlet sumujący dwie liczby przekazane za pomocą formularza jako parametry żądania musielibyśmy sparsować je, tak aby otrzymać zmienne typu int. W tym celu możemy posłużyć się metodą statyczną int parseInt(String) z klasy Integer. Poniżej przykładowa implementacja takiego servletu:

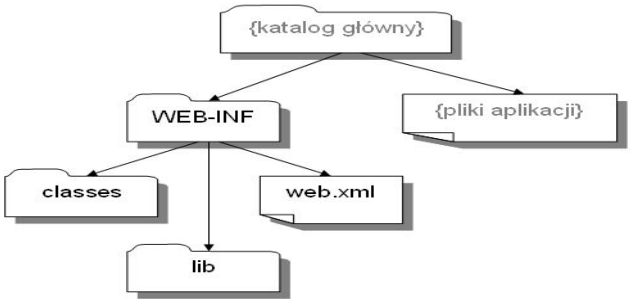
```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

   response.setContentType("text/html");

   PrintWriter out = resp.getWriter();

   String paramX = req.getParameter("paramX");
   String paramY = req.getParameter("paramY");

   out.println("<html><body>");

   int sum = Integer.parseInt(paramX) + Integer.parseInt(paramY);

   out.println("Suma liczb x i y wynosi " + sum);
   out.println("</body><html>");

   out.close();
}
```

Jeśli chcemy aby nasz servlet obsługiwał także żądania wysyłane metodą GET to wystarczyłoby do powyższego dopisać:

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

   doPost(req, resp);
}
```

Tym samym przekazujemy obsługę żądań typu GET do metody doPost(…). Jest to możliwe, jako że parametry, niezależnie od tego jakiego typu żądanie HTTP otrzymaliśmy, obsługujemy w ten sam sposób. Także w ten sam sposób generujemy stronę HTML która odsyłana jest jako odpowiedź na żądanie.

Typowa aplikacja WWW składa się z wielu różnego rodzaju komponentów: plików konfiguracyjnych, plików zasobów statycznych (np. HTML, JPG, CSS), plików JSP, katalogów i oczywiście plików zawierających skompilowane klasy, w tym servlety.
Poszczególne komponenty aplikacji WWW muszą być umieszczone w odpowiedniej strukturze katalogowej. Strukturę tą pokazano na poniższej ilustracji:



Element oznaczony jako *{katalog główny}* to katalog grupujący wszystkie elementy naszej aplikacji. Jego nazwa może być dowolna, jednak domyślnie nazwa ta często używana jest przez serwer aplikacji do określenia adresu URL pod jakim aplikacja ta jest serwowana.

W takim wypadku adresem URL naszej aplikacji będzie: **http://**{adres serwera}**/**{katalog główny}
Jeśli np. katalog główny naszej aplikacji nazwiemy *myApp* i aplikację zainstalujemy na serwerze aplikacji dostępnym pod adresem *http://test.pl:8080*, to aplikacja będzie dostępna pod adresem *http://test.pl:8080/myApp*.

Bardzo ważnym katalogiem obecnym w każdej aplikacji jest katalog o nazwie *WEB-INF*, który musi nazywać się dokładnie tak i musi być umieszczony bezpośrednio w katalogu głównym.
Katalog *WEB-INF* ma tę własność, że wszystkie umieszczone w nim pliki i katalogi są chronione przed bezpośrednim dostępem, tzn. nie jest możliwe pobranie żadnego z umieszczonych w nim plików przy pomocy żądania HTTP. Dla odmiany, wszystkie pliki umieszczone poza tym katalogiem serwer aplikacji zwróci w odpowiedzi na żądanie typu GET, jeśli do adresu URL aplikacji dodamy poprawną ścieżkę dostępu do tego pliku (względem katalogu głównego aplikacji).
Np. jeśli w katalogu głównym utworzymy katalog o nazwie *html* i w nim plik o nazwie *strona.html*, to będziemy mogli pobrać ten plik (i wyświetlić w przeglądarce) wysyłając żądanie na adres *{adres aplikacji}/html/strona.html*, przy czym *{adres aplikacji}* to np. *http://test.pl:8080/myApp*.

W katalogu *WEB-INF* umieszczamy główny plik konfiguracyjny aplikacji WWW, tzw. deskryptor wdrożenia. Jest to pokazany na powyższej ilustracji plik *web.xml*.
Deskryptor wdrożenia to główny plik konfiguracyjny aplikacji WWW. Jest to plik XML o nazwie web.xml, który umieszczamy bezpośrednio w katalogu WEB-INF.
W deskryptorze wdrożenia konfigurujemy między innymi adresy URL servletów. Wiemy już, z wcześniejszego opisu jak skonstruować URL dla zapytań, dla których w odpowiedzi odsyłane są pliki umieszczone bezpośrednio w katalogu głównym aplikacji, ale w jaki sposób wysłać żądanie HTTP do servletu? Otóż to, które żądania zostaną obsłużone przez servlet konfigurujemy właśnie w deskryptorze wdrożenia, tzn. w pliku web.xml. Poniżej przykład:

```
<web-app version="2.5"
 xmlns="http://java.sun.com/xml/ns/javaee"

 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee

                http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <servlet>

   <servlet-name>HelloServlet</servlet-name>
```

```
    <servlet-class>pl.test.HelloWorldServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

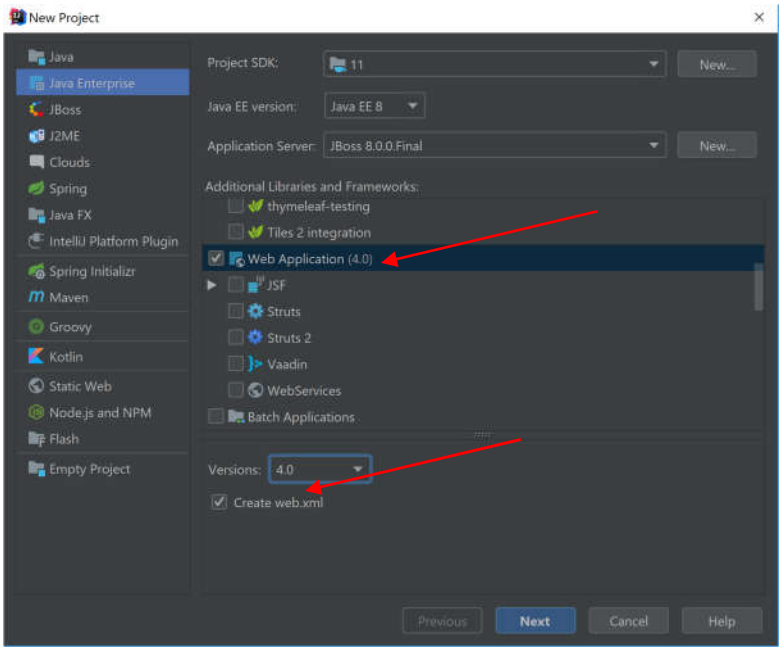
Element o nazwie web-app to element nadrzędny, grupujący wszystkie elementy konfiguracji. Zawiera on niezbędne definicje w postaci atrybutów, w których znaczenie nie potrzebujemy póki co wnikać – z naszego punktu widzenia jest to element stały każdego pliku web.xml i zawsze wygląda on tak samo (dla specyfikacji servletów w wersji 2.5).

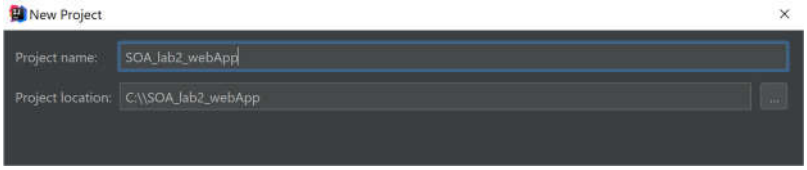
To co nas szczególnie interesuje, to element servlet oraz servlet-mapping.
Element servlet to definicja servletu; w pod-elemencie servlet-class podajemy pełną (poprzedzoną nazwą pakietu) nazwę klasy która implementuje servlet, zaś servlet-name to dowolna, wymyślona przez nas nazwa. Nazwa servletu którą wpisujemy jako wartość pod-elementu servlet-name służy w zasadzie tylko do tego, aby powiązać definicję servletu z definicją mapowania na URL. Element servlet-mapping to właśnie definicja wzorca adresów URL, dla których żądania będą obsługiwane przez powiązany servlet. Uwaga! Element servlet-name wiąże definicję servletu z definicją mapowania i jego wartość musi być w obydwu tych definicjach taka sama.

W powyższym przykładzie skonfigurowaliśmy aplikację WWW w ten sposób, że wszystkie żądania wysłane na adres URL postaci {adres aplikacji}/hello zostaną przekazane do servletu pl.test.HelloWorldServlet i w odpowiedzi na takie żądanie zostanie do klienta odesłana treść wygenerowana w tym właśnie servlecie.
W deskryptorze wdrożenia konfigurujemy także szereg innych parametrów aplikacji. W szczególności, możemy umieszczając w nim odpowiednią konfigurację zabezpieczyć aplikację przed nieautoryzowanym dostępem.

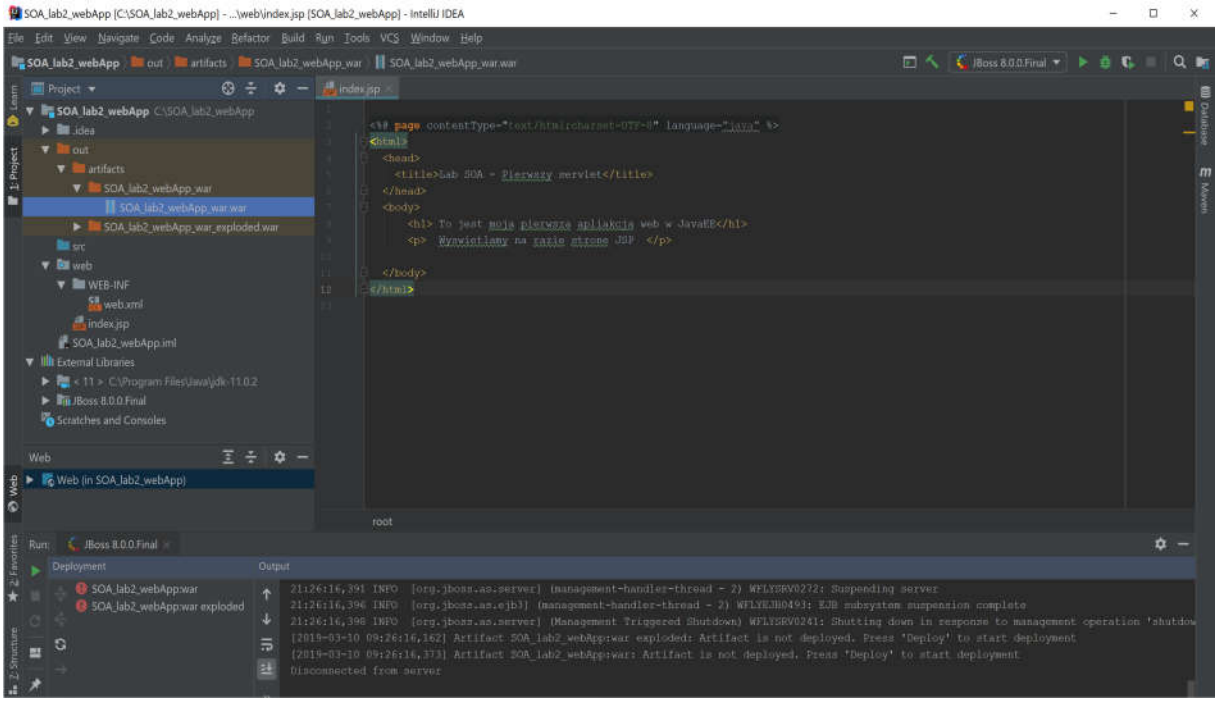
Jak stworzyć pierwsza aplikacje w Intelij

1. Tworzymy nowy projekt typu Java Enterprise – wybierając biblioteki do tworzenia Web Aplication

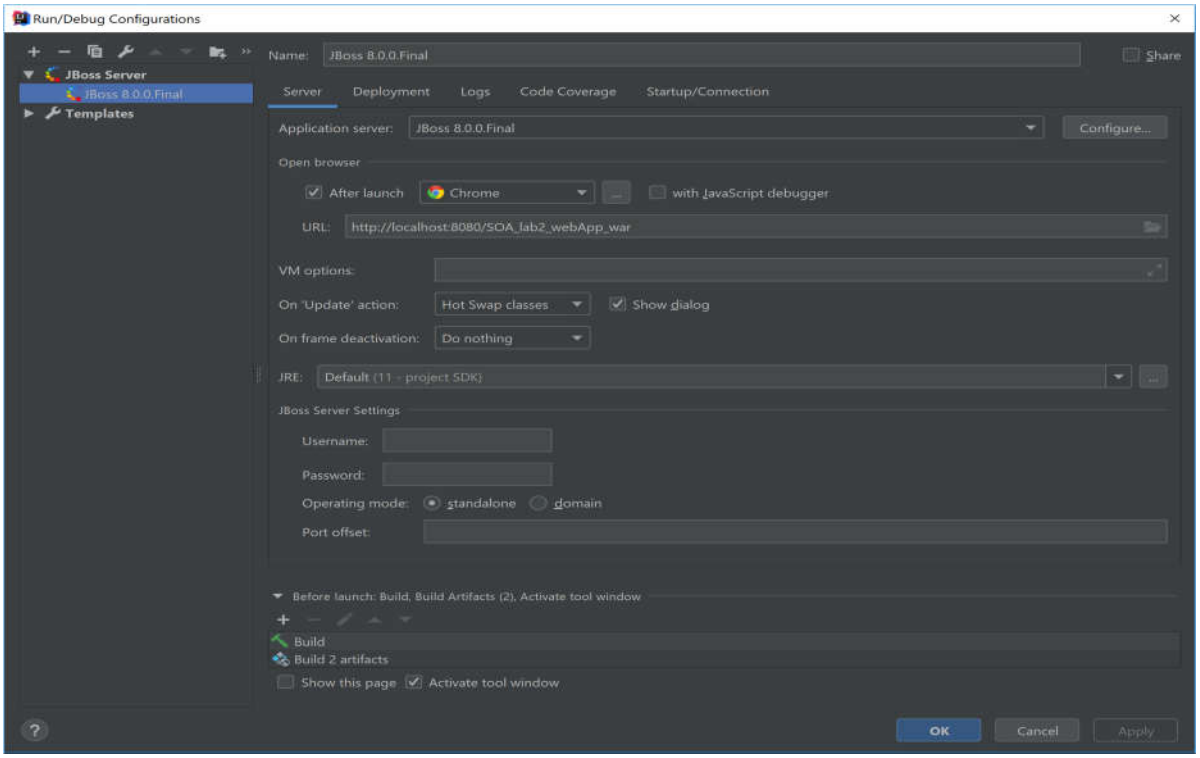




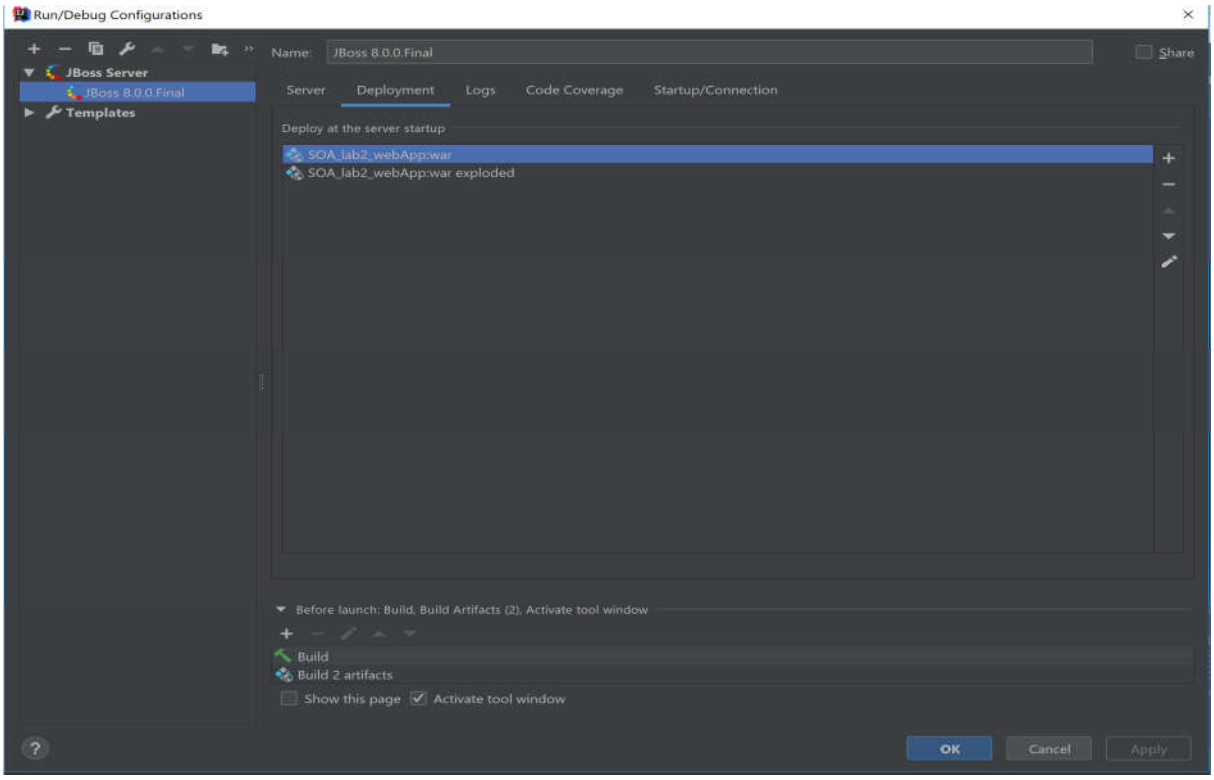
2. Po stworzeniu projektu jego struktura wygląda jak na rysunku poniżej. Modyfikujemy zawartość pliku index.jsp – wpisują do niego dowolną zawartość statyczną.



3. Następnie konfigurujemy serwer Aplikacyjny tak by można było szybko zrobić deploy apliakacji i przetestować współprace pomiędzy Intelij a WildFly.

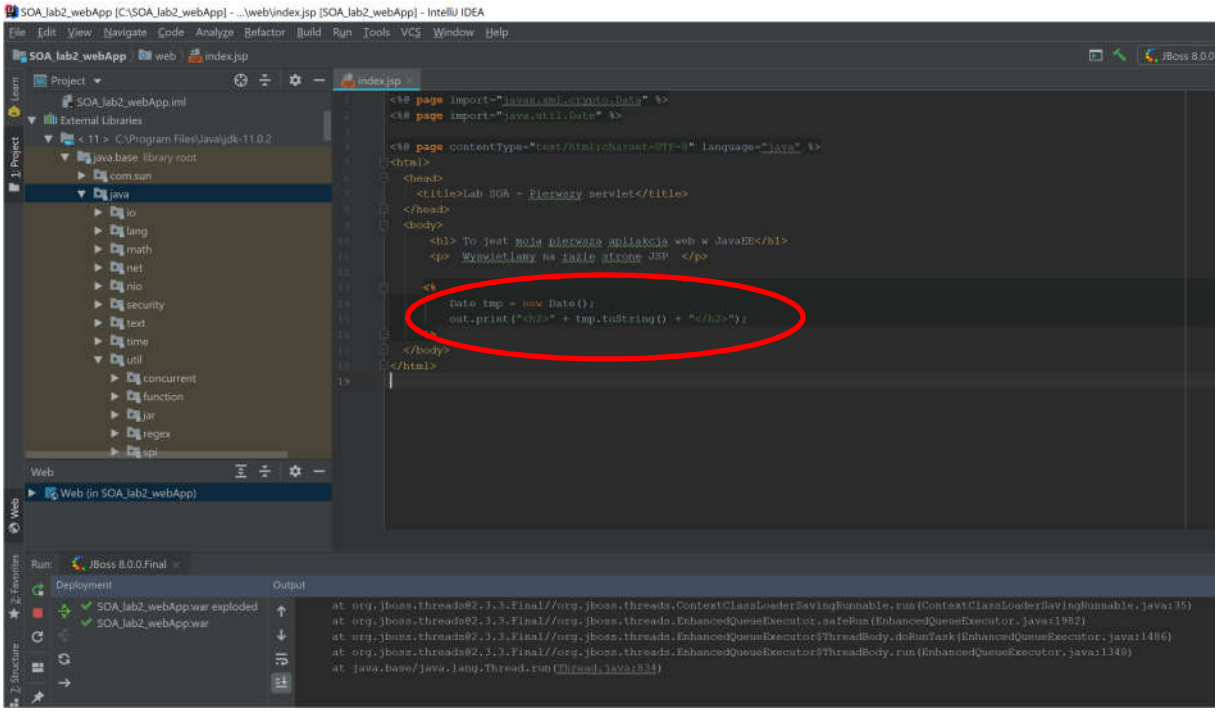


W tego typu aplikacji możliwe jest wygenerowanie dwóch typów artefaktów: wersji spakowanej .war lub wersji rozszerzonej z dostępną struktura wewnętrzną (exploded)



4. Uruchamiamy apliakcje na serwerze i otrzymujemy następujący efekt

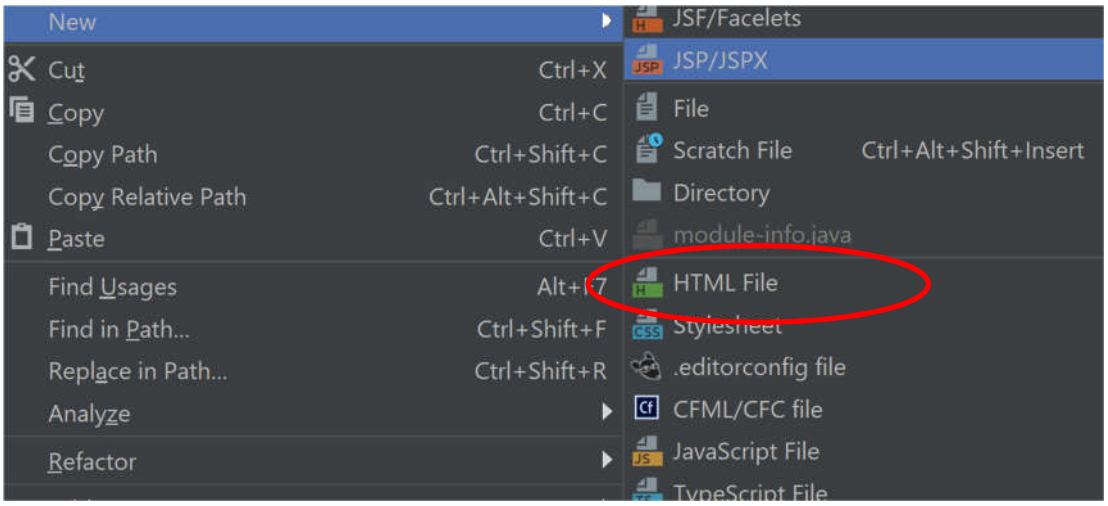
Nasza aplikacje można rozszerzyć o elementy skryptowe (wykonywane po stronie serwera) umieszczone bezpośrednio w kodzie strony jsp.



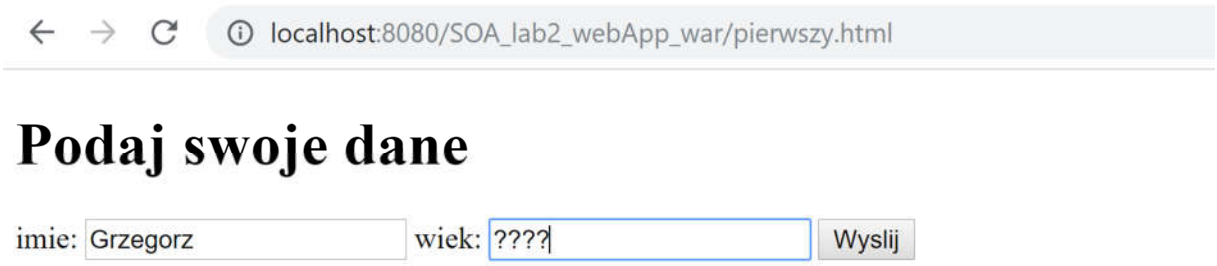
Po ponownym uruchomieniu nasza aplikacja wygląda teraz tak.

5. Czas na rozszerzenie naszej aplikacji o obsługę servletów. W tym celu przygotujemy nową stronę zawierającą formularz do podawania swojego imienia i wieku.

Dodajemy do projektu nowy plik typu html.



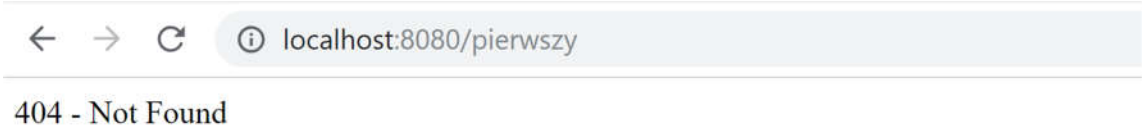
Implementujemy jego zawartość tworząc formularz, który po uruchomieniu wyglądać powinien tak jak na rysunku poniżej.



Pliku pierwszy.html zawierać będzie następujący fragment kodu:

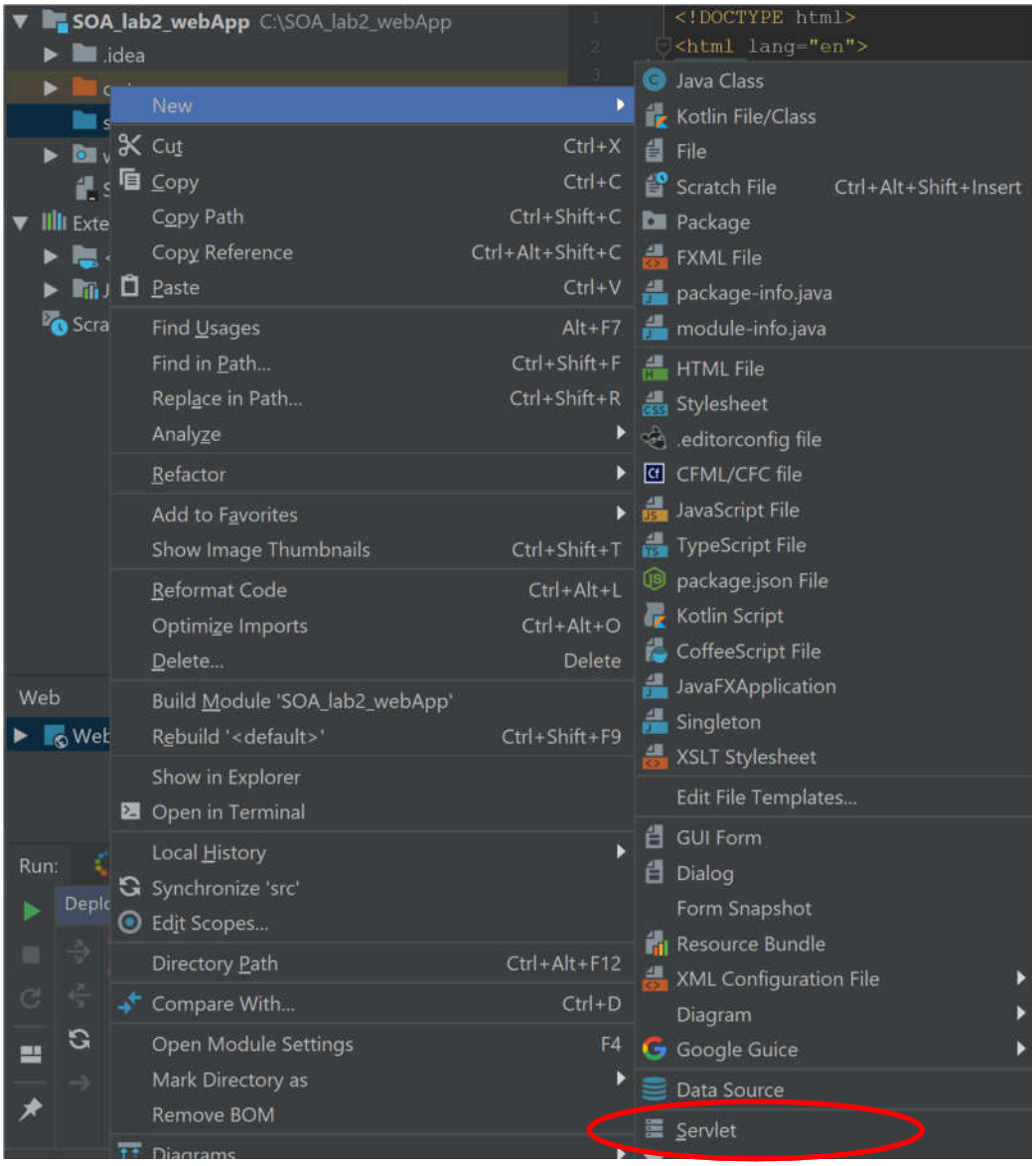
```
<form action="pierwszy" method="get">

  Imie: <input type="text" name="imie"/>
  Wiek:  <input type="text" name="wiek" size="2"/>
  <input type="submit" value="Wyslij"/>
</form>
```

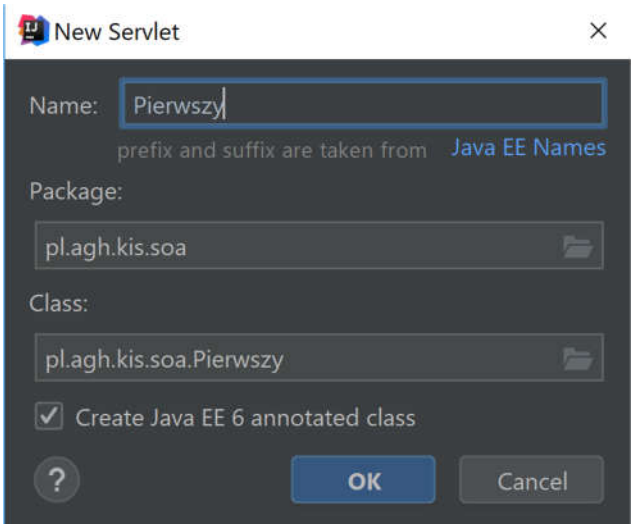
Po naciśnięciu przycisku na ekranie pojawia się następujący komunikat:



Jest tak dlatego, że brak nam servleta obsługującego przesłany formularz. Napiszmy go wiec.

5. Dodajmy do projektu plik typu Servlet.



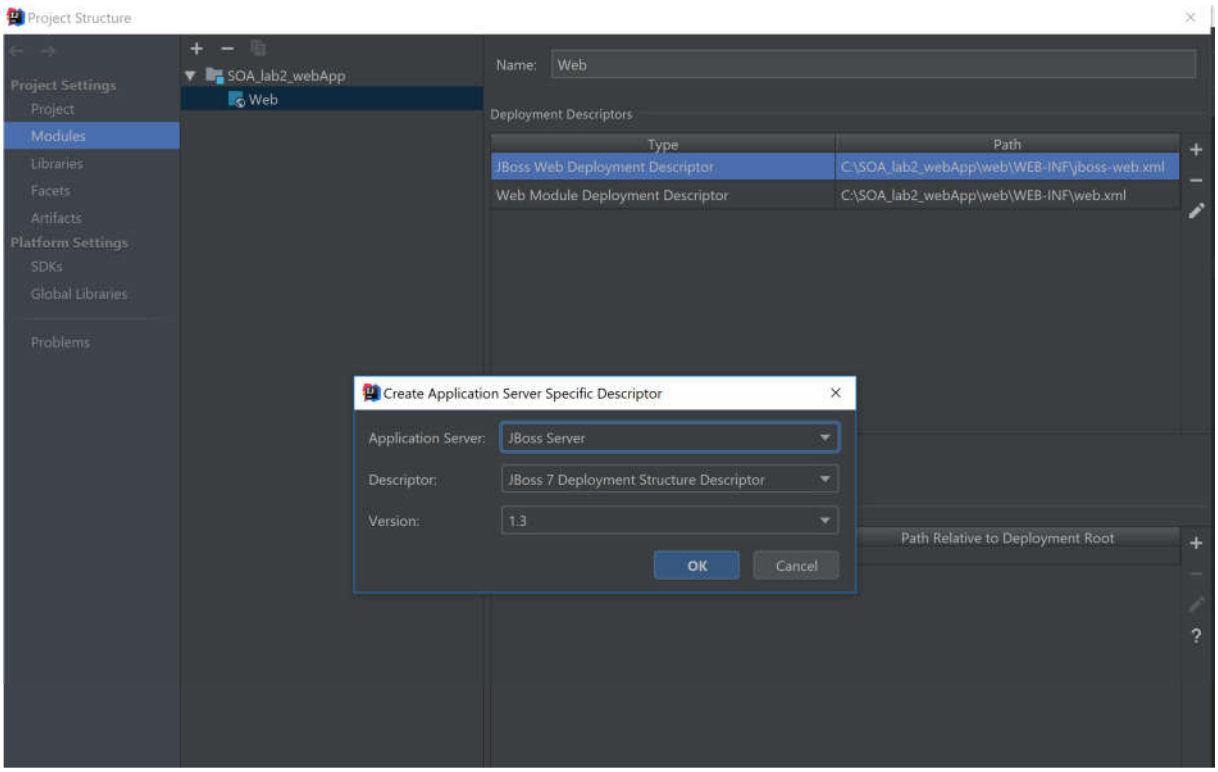


W edytorze kodu otwórz zawartość pliku `pierwszy.java`.
Zmodyfikuj kod metody `doGet()` w poniższy sposób. Zwróć uwagę na konieczność jawnego rzutowania parametrów innych niż łańcuchy znaków.

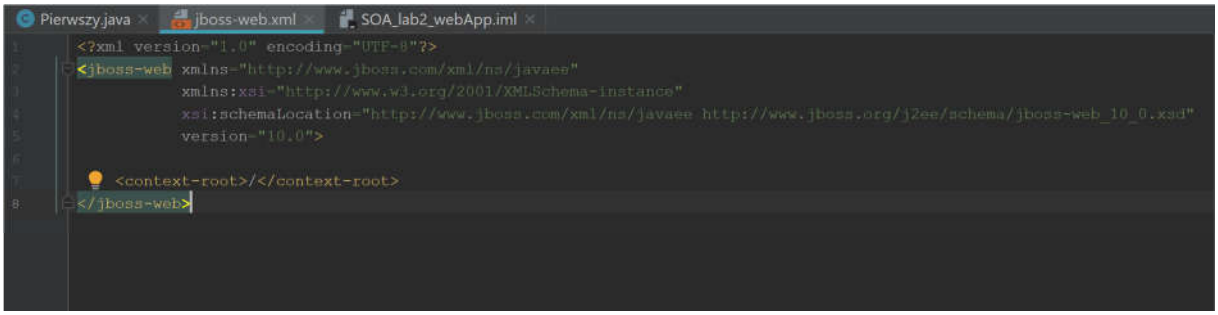
```
response.setContentType("text/html");
response.setCharacterEncoding("windows-1250");
PrintWriter out = response.getWriter();
String name = request.getParameter("imie");
int age = Integer.parseInt(request.getParameter("wiek"));
out.println("<html>");
out.println("<head><title>Pierwszy Servlet</title></head>");
out.println("<body>");
out.println("<p>Witaj, " + imie + ", masz " + wiek + " lat</p>");
out.println("</body>");
out.println("</html>");
out.close();
```

6. Pozostaje nam sprawdzic czy wszystko działa OK. Zanim to jedna zrobimy chcemy zmienic jeszcz jedna rzecz. Nie podoba nam się ściezka potrzebna do uruchamiania naszej apliakcji. Chcemy aby apliakcja uruchamiała się bezpośrednio z pod adresu localhost:8080/pierwszy.html a nie tak jak dotychczas.

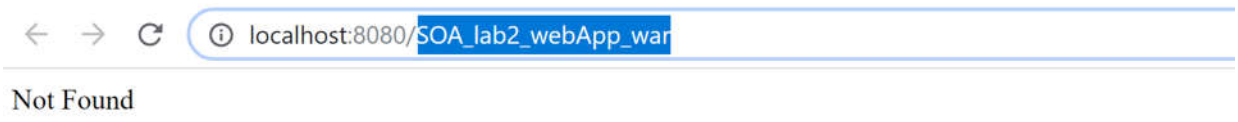
W tym celu należy zmienic context root. Potrzebujmy do tego plik jboss-web.xml. Można go utworzyc dodajac do projektu description serwera WildFly – tak jak poniżej.



W projekcie pojawia się plik jboss-web.xml, do którego należy dodać wpis context-root tak jak poniżej.



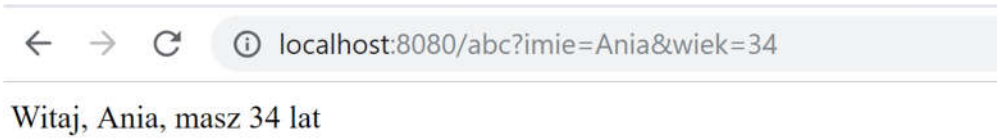
7. Sprawdzamy teraz nasza aplikację. Pod starym adresem już nie działa.



Wpisujemy nowy adres – strona startowa aplikacji się pojawia

Wpisujemy wiec adres formularza i podajemy jakieś przykładowe dane:

Jak widać servlet teraz pracuje już poprawnie.



Proszę zwrócić uwagę na pojawiający się adres - abc. Dlaczego taki adres skoro nasz servlet nazywał się pierwotnie pierwszy. Jak się łatwo domyślić dokonałem podmiany nazwy serwletu i jego adresu.

Proszę zrobić coś podobnego samodzielnie.

Zadanie do realizacji

Zadanie 1.

- Napisz servlet, który będzie przyjmował 5 liczb całkowitych metodą GET, wyliczy ich średnią i zwróci wynik.
- Napisz servlet, który będzie przyjmował dowolną ilość parametrów metodą POST i sprawdzał, czy parametry te są liczbami. Jeśli są, niech je wyświetli w kolejności od najmniejszej do największej. Jeśli parametry nie są liczbami, niech servlet zwróci informacje o błędnych danych.

  podpowiedź– sprawdź co robi funkcja „getParameterNames()" i czy można ją wykorzystać ją do pobierania dowolnej ilości parametrów z request'a.

Zadanie 2. Doradca piwny

Celem zadania jest napisanie prostej aplikacji internetowej o nazwie *Doradca Piwny*. Aplikacja powinna zostać zbudowana zgodnie z prostą architekturą MVC (z wyróżnionym widokiem/ami kontrolerem oraz modelem)
Aplikacja będzie posiadać dwie strony będące widokiem: *form.html* z formularzem wyboru koloru piwa, oraz stronę *wynik.jsp*, która wyświetla sugerowaną markę piwa na podstawie wyboru koloru. Kontroler aplikacji jest skonstruowany w postaci serwletu *WyborPiwa*. Modelem będzie komponent będącym klasą *EkspertPiwny*.
Dodatkowo aplikacja powinna wykorzystując koncepcje filtra na samym początku sprawdzić czy osoba zainteresowana wyborem marki piwa jest pełnoletnia. Jeśli nie to wyświetlamy jej odpowiedni komunikat. Tylko osoby pełnoletnie mogą przejść do ekranu wyboru koloru piwa.

Zadanie 3. Obsługa księgi gości

(wykorzystanie koncepcji sesji i cookies)

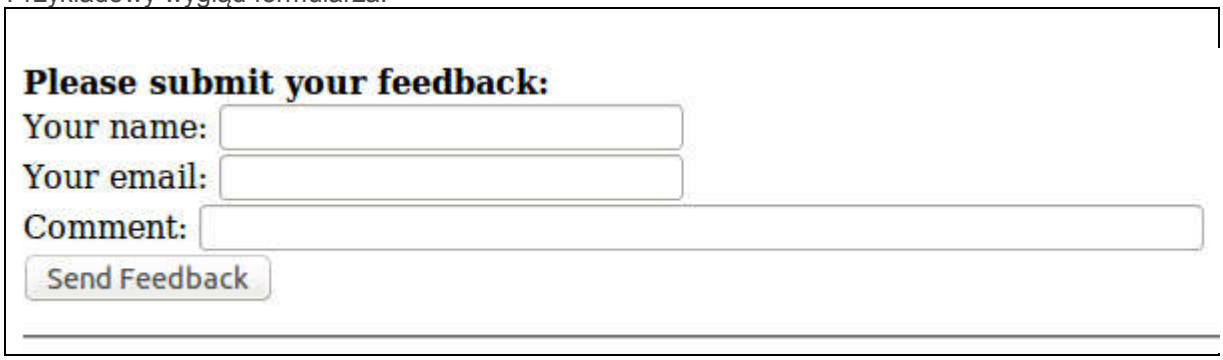
Przygotuj stronę logowania. Założenia:

- Formularz wyświetla 2 pola (login i hasło) oraz przycisk "Zaloguj".
- Wewnątrz skryptu w sposób jawny jest zadeklarowany wektor przechowujący elementy typu *DaneOsobowe* (składowe: login, hasło, imie, nazwisko). Wektor ten powinien zostać wypełniony przykładowymi danymi.
- Scenariusze użycia:
    - nie podano loginu - komunikat "Podaj login" i ponowne wyświetlenie formularza,
    - nie podano hasła - komunikat "Podaj hasło" i ponowne wyświetlenie formularza,
    - błędne dane logowania - komunikat o błędzie (tekst dowolny) i ponowne wyświetlenie formularza,
    - poprawne dane logowania – przejście do strony obsługującej księgę gości. (patrz poniżej).

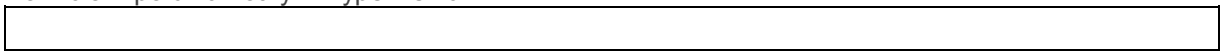
Przygotuj **sevlet** obsługujący prostą księgę gości. Założenia:

- Dane podane przez użytkownika powinny być utrwalane w pamięci serwera - zalecana struktura to *Vector*.
    - Dane nie muszą być utrwalane na stałe. Po restarcie serwera mogą być zerowane.
    - Dane wpisane w formularzu powinny być widoczne również z innej sesji przeglądarki (proszę przeprowadzić testy w przeglądarce trybie "Prywatnym").\

Przykładowy wygląd formularza:



Formularz po dwukrotnym wypełnieniu:



Informacje pomocnicze:

Parametry inicjalizacyjne servletu

Dodanie parametrów inicjalizacyjnych servletu polega na modyfikacji pliku web.xml i dodaniu wpisów określających parametry:

```
<servlet>
<servlet-name>StateSaverServlet</servlet-name>
<servlet-class>Servlets.StateSaverServlet</servlet-class>
<init-param>
<param-name>login</param-name>
<param-value>koper</param-value>
</init-param>
</servlet>
```

Dostęp do parametrów inicjalizacyjnych

Odczytanie parametrów konfiguracyjnych servletu wymaga pobrania obiektu klasy ServletConfig poprzez wywołanie metody getServletConfig() w obiekcie servletu, a następnie wywołaniu metody getInitParameter():

```
getServletConfig().getInitParameter()
```

Dostęp do sesji

Obiekt klasy HttpSession reprezentujący sesję jest dostępny jako poprzez wywołanie request.getSession() w przypadku servletu oraz zmiennej lokalnej session w przypadku JSP.

Przydatne metody:

- session.setAttribute() - dodanie obiektu do sesji,
- session.getAttribute() - pobranie obiektu z sesji,
- session.removeAttribute() - usuniecie obiektu z sesji,
- session.invalidate() - zniszczenie sesji.

Dostęp do kontekstu servletu (aplikacji!)

W przypadku servletu należy pobrać referencję do obiektu klasy ServletContext poprzez metodę getServletContext(), zaś w przypadku strony JSP poprzez zmienną lokalną application.

Przydatne metody:

- servletContext.setAttribute() - dodanie obiektu do kontekstu servletu,

• servletContext.getAttribute() - pobranie obiektu z kontekstu servletu.
Obsługa ciasteczek
Dostęp do ciasteczek realizowany jest poprzez obiekty request oraz response:
• response.addCookie() - metoda dodająca ciasteczko do przeglądarki klienta,
• request.getCookies() - metoda zwracając tablicę wszystkich ciasteczek pobranych od klienta.
Ustawianie czasu życia ciasteczka poprzez metodę cookie.setMaxAge(). Natychmiastowe usunięcie ciasteczka: cookie.setMaxAge(0) – jako wysłanie ciasteczka o tej samej nazwie do klienta poprzez obiekt response!!!

Przekierowanie obsługi żądania (w obrębie kontenera!)
Najpierw należy pobrać obiekt klasy RequestDispatcher za pomocą metody:
request.getRequestDispatcher()
Następnie trzeba wywołać metodę forward() zwróconego obiektu klasy RequestDispatcher

Przekierowanie obsługi żądania (poprzez przeglądarkę klienta!)
Przekierowanie przeglądarki do innego adresu URL: response.sendRedirect()