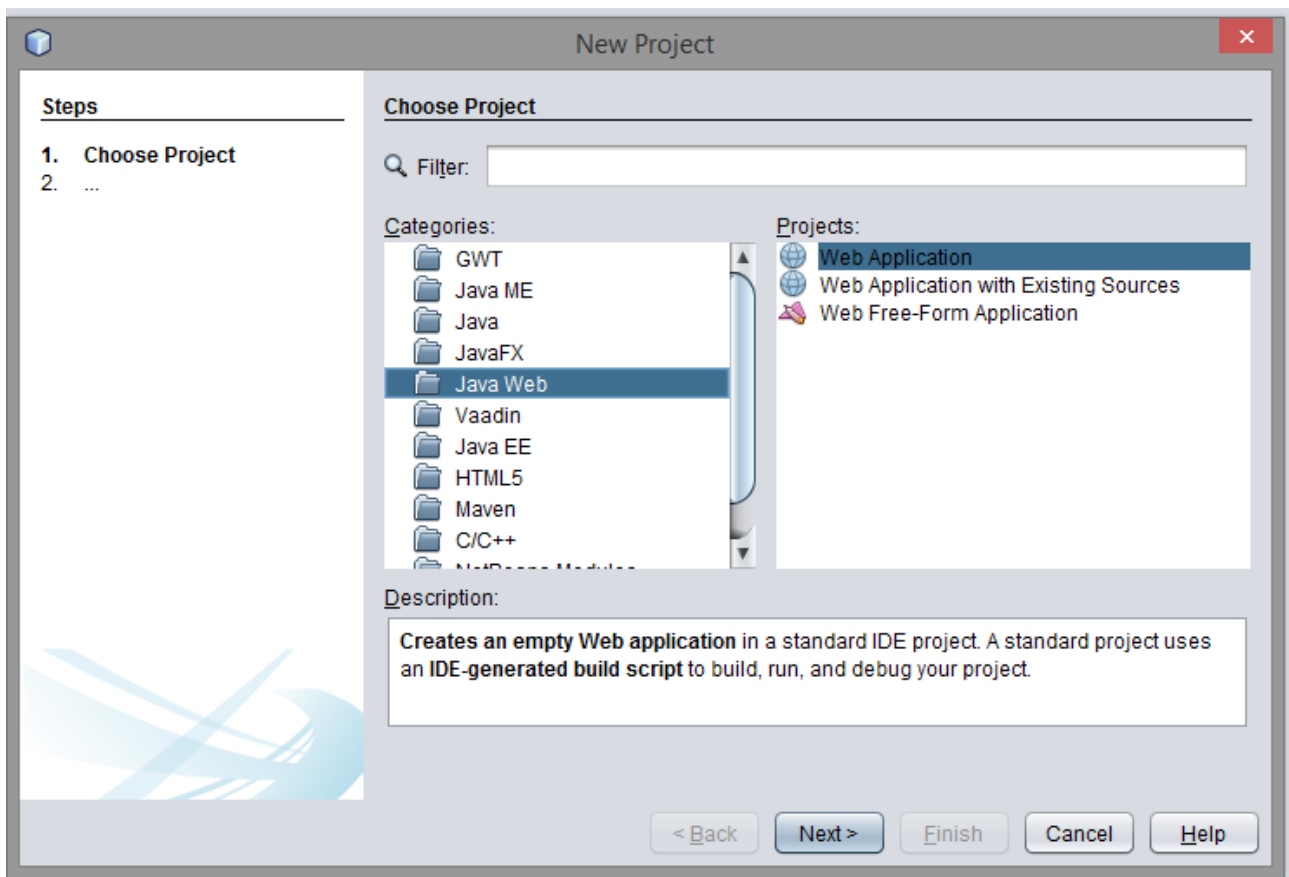


Spring MVC – pierwsza aplikacja

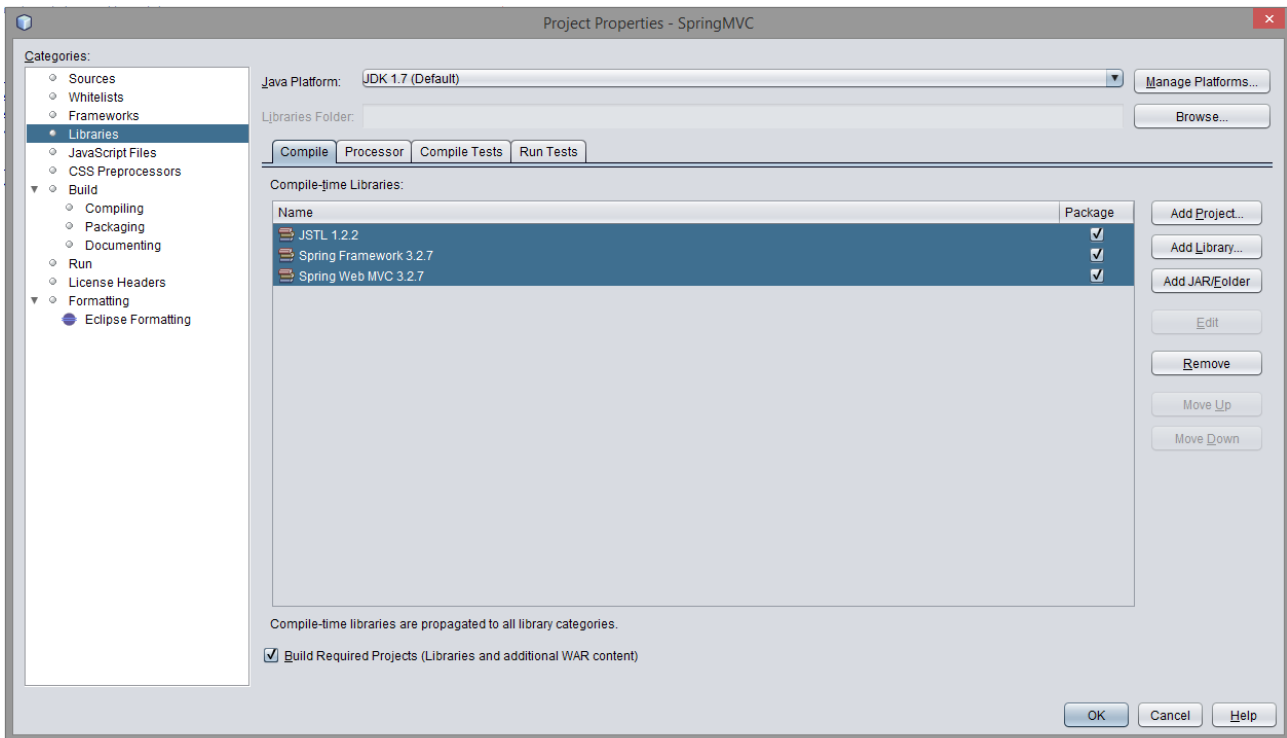
Kod źródłowy aplikacji którą tworzę w niniejszym kursie jest do pobrania z adresu:
<http://www.jsystems.pl/storage/spring/springmvc1.zip>

Aplikacja jest tworzona w NetBeans, a uruchamiana na serwerze Glassfish który to jest dołączany do w.w. IDE.

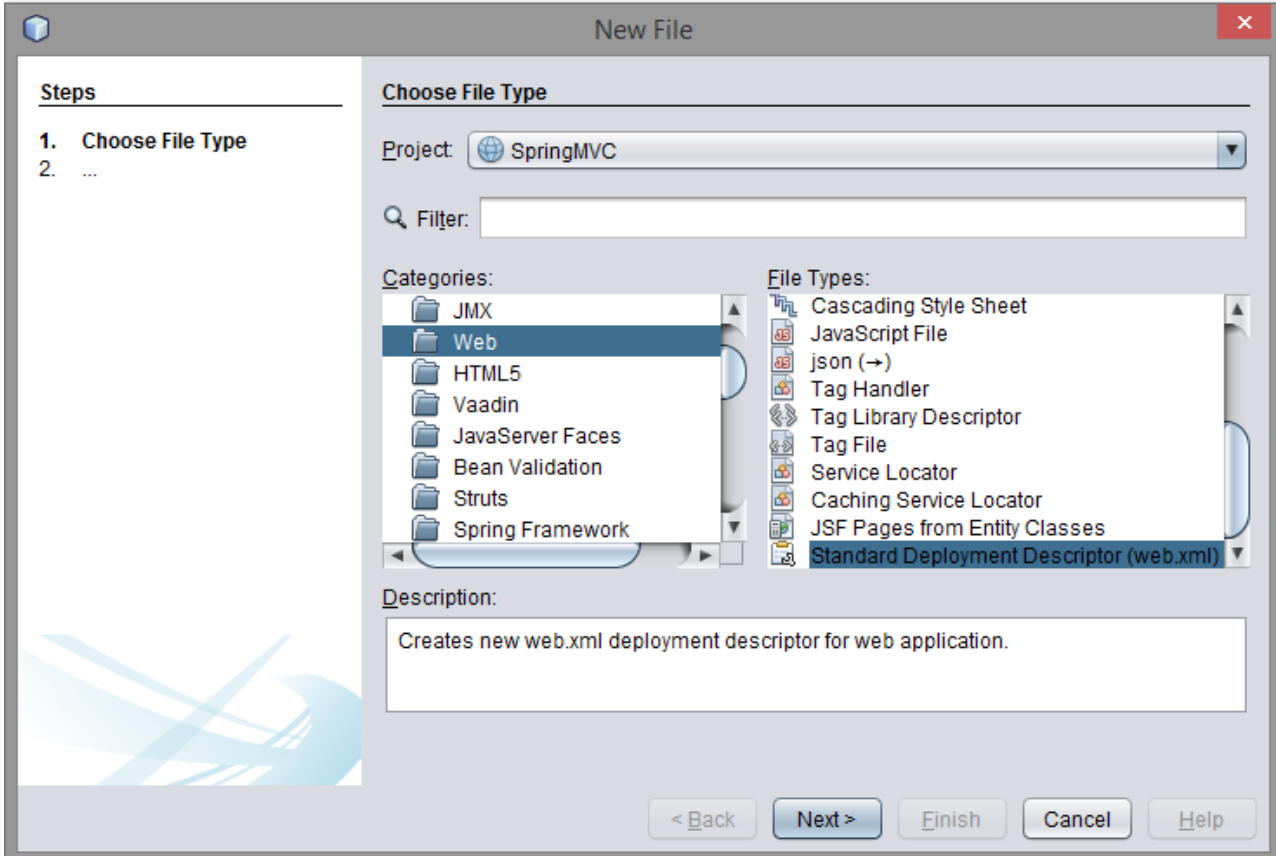
Zacznijmy od stworzenia zwykłej aplikacji WEBowej:



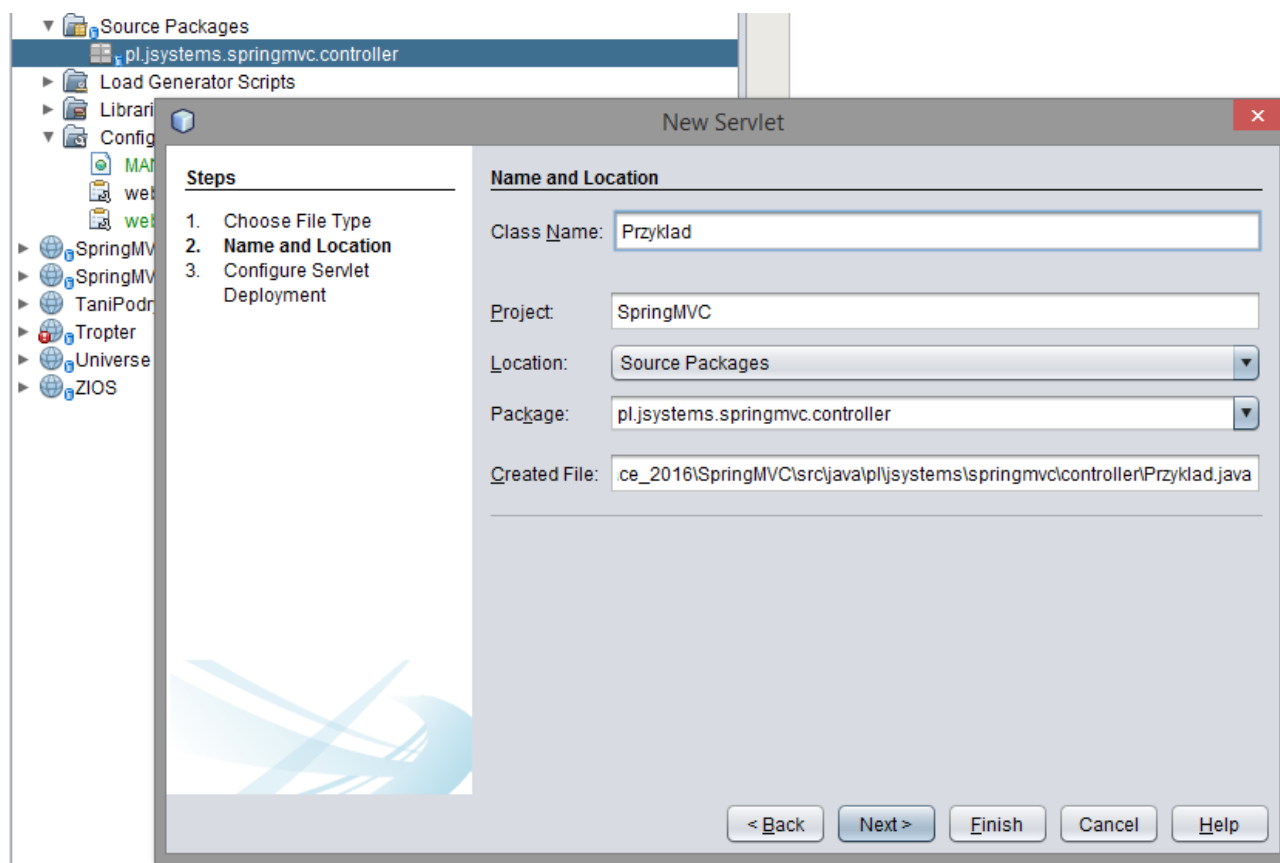
Po jej utworzeniu musimy dodać niezbędne biblioteki. W Netbeans należy wybrać właściwości projektu, przejść do sekcji „Libraries” a następnie kliknąć AddLibrary i wybrać potrzebne:



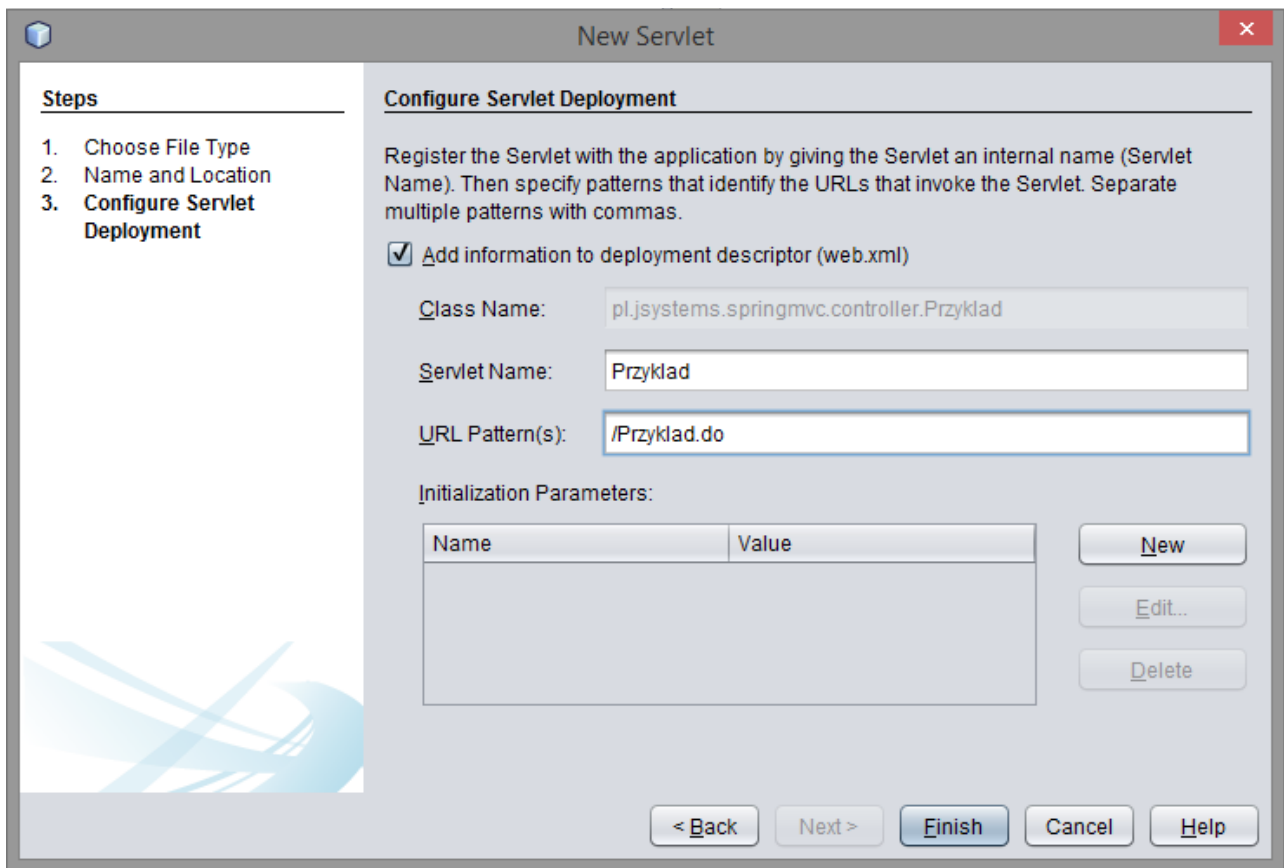
Będzie nam też potrzebny plik konfiguracyjny web.xml, dlatego dodajemy do katalogu WEB-INF:



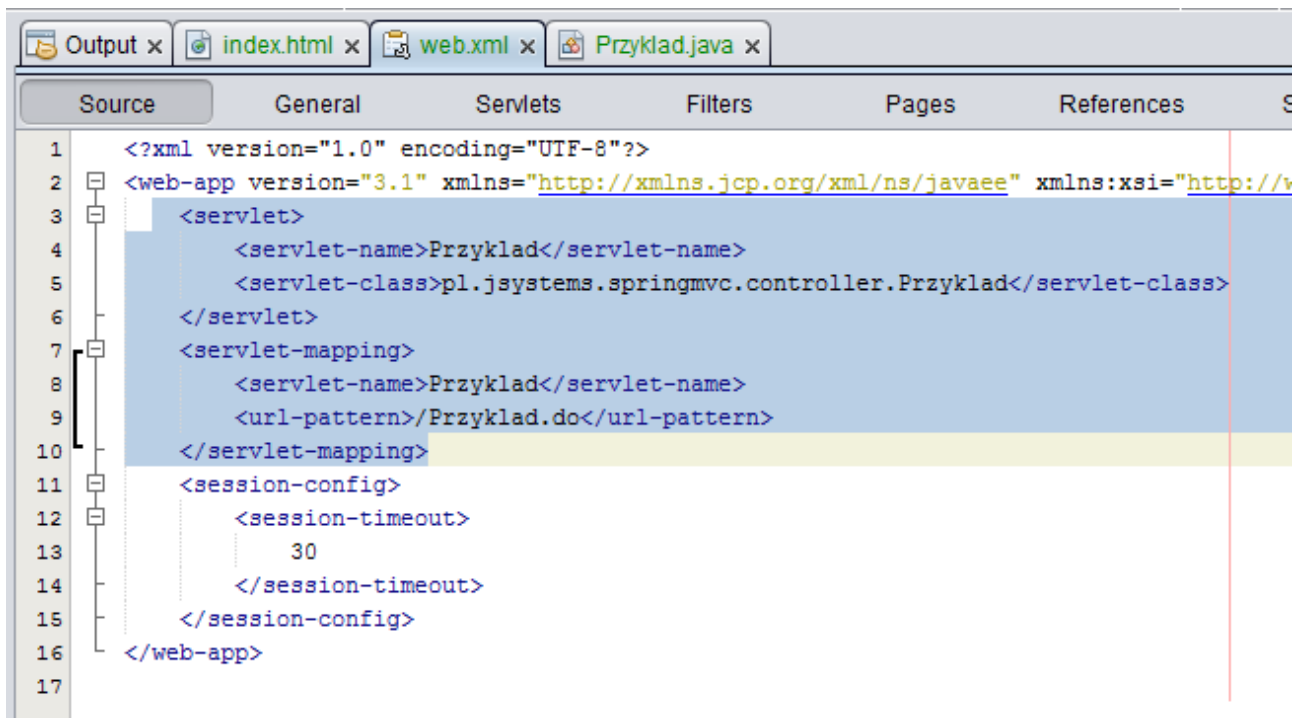
Będzie nam też potrzebny jakiś pakiet w którym umieścimy nasze klasy:



Na początek aby przyjrzeć się sposobowi przekazywania kontroli nad wywołaniami Springowi, stworzymy zwyczajny serwet. Pamiętaj by zaznaczyć dodanie informacji o nim do web.xml!

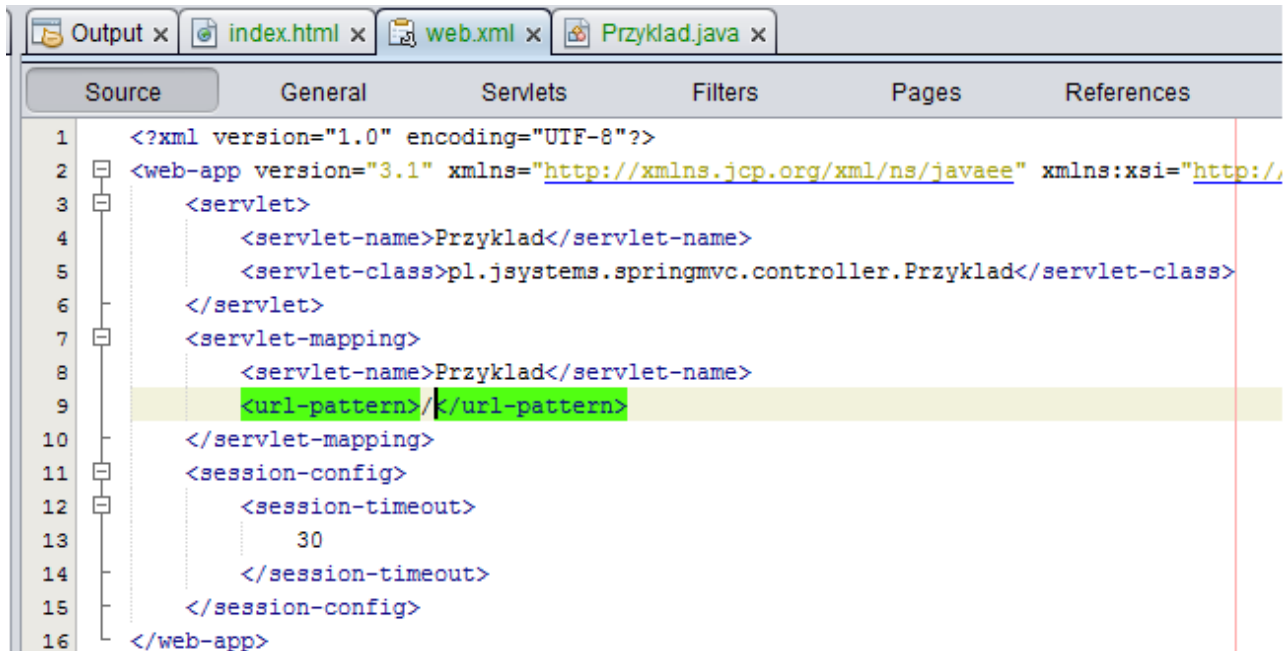


Gdy zajrzemy do web.xml po dodaniu serwletu, zobaczymy że pojawił się w nim taki oto wpis:



W liniach 7-9 mamy zapisane, że wywołanie podstrony „Przyklad.do” będzie obsługiwane przez

nasz nowy serwlet. Nieco ten wpis przerobimy. Przyjrzyj się linii 9. Wpis „/” oznacza, że strona początkowa naszej aplikacji będzie obsługiwana przez nasz serwlet. Gdybyśmy wprowadzili tam wpis „/*” oznaczałoby to, że każde wywołanie adresu w naszej aplikacji będzie przez ten serwlet obsługiwane- tj. każdy podadres np. <http://localhost:8080/SpringMVC/niematakiejstrony.do> również byłoby obsługane. Różnica w gwiazdce :)



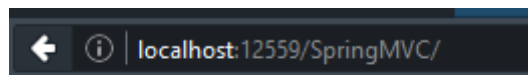
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://
3     <servlet>
4         <servlet-name>Przyklad</servlet-name>
5         <servlet-class>pl.jsystems.springmvc.controller.Przyklad</servlet-class>
6     </servlet>
7     <servlet-mapping>
8         <servlet-name>Przyklad</servlet-name>
9         <url-pattern>/</url-pattern>
10    </servlet-mapping>
11    <session-config>
12        <session-timeout>
13            30
14        </session-timeout>
15    </session-config>
16 </web-app>
```

Dokonyamy teraz małej zmiany w naszym serwlecie. W momencie wywołania naszej aplikacji na ekranie w przeglądarce powinna się wyświetlić treść „Halo, tutaj serwlet!”.



```
6 package pl.jsystems.springmvc.controller;
7
8 import java.io.IOException;
9 import java.io.PrintWriter;
10 import javax.servlet.ServletException;
11 import javax.servlet.http.HttpServlet;
12 import javax.servlet.http.HttpServletRequest;
13 import javax.servlet.http.HttpServletResponse;
14
15 /**
16  *
17  * @author andrzej
18  */
19 public class Przyklad extends HttpServlet {
20
21     @Override
22     protected void doGet(HttpServletRequest request, HttpServletResponse response)
23         throws ServletException, IOException {
24         response.setContentType("text/html;charset=UTF-8");
25         PrintWriter out = response.getWriter();
26         out.println("Halo, tutaj serwlet!");
27     }
28
29 }
30
```

Uruchommy więc naszą aplikację:

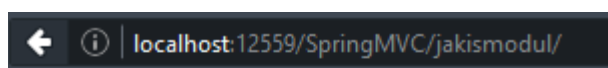


Halo, tutaj servlet!

Moglibyśmy podzielić naszą aplikację na moduły i obsługiwać je przez różne serwlety... albo np. tylko jeden moduł obsługiwać z użyciem Spring MVC. Nieco przerabiam mój plik web.xml:

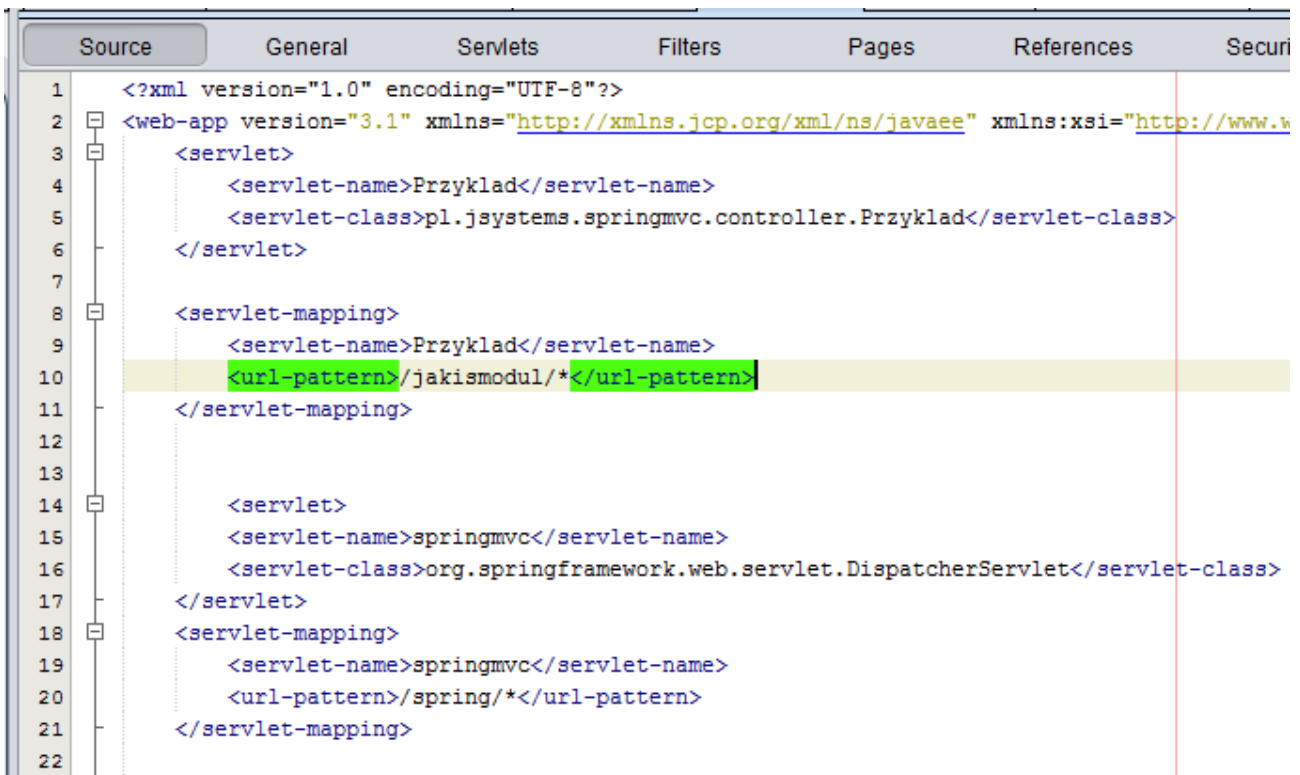
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http
3   <servlet>
4     <servlet-name>Przyklad</servlet-name>
5     <servlet-class>pl.jsystems.springmvc.controller.Przyklad</servlet-class>
6   </servlet>
7   <servlet-mapping>
8     <servlet-name>Przyklad</servlet-name>
9     <url-pattern>/jakismodul/</url-pattern>
10  </servlet-mapping>
11  <session-config>
12    <session-timeout>
13      30
14    </session-timeout>
15  </session-config>
16 </web-app>
```

Porównajmy adres wywołania naszego serwletu:



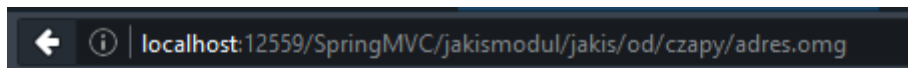
Halo, tutaj servlet!

Teraz będzie drobna zmiana. W linii 10 do adresu /jakismodul/ dodałem *. To oznacza że każde wywołanie z początkiem /jakismodul/ będzie obsługiwane przez nasz przykładowy serwlet:



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w
3 <!--
4 <servlet>
5 <!--
6 <!--
7 <!--
8 <servlet-mapping>
9 <!--
10 <url-pattern>/jakismodul/*</url-pattern>
11 </servlet-mapping>
12 <!--
13 <!--
14 <servlet>
15 <!--
16 <!--
17 </servlet>
18 <servlet-mapping>
19 <!--
20 <url-pattern>/spring/*</url-pattern>
21 </servlet-mapping>
22 </web-app>
```

I sprawdzamy :



Halo, tutaj serwlet!

Mam nadzieję, że po tych przykładach to co się dzieje w nowych liniach 14-21 w pliku web.xml będzie dla Ciebie oczywiste :) Wszystkie wywołania których adres będzie się zaczynał od /SpringMVC/spring/ będą obsługiwane przez serwlet o nazwie „springmvc”. A ten serwlet to klasa DispatcherServlet której... nie definiowaliśmy :) To właśnie tutaj następuje przekazanie kontroli do Springa. To jest klasa dostarczana z biblioteką Springa, która przejmie kontrolę nad wywołaniami w naszej aplikacji (przynajmniej we wskazanym zakresie adresowym). Co by się stało gdybyśmy w linii 20 zadeklarowali „/*” ? Wszystkie wywołania w naszej aplikacji byłyby obsługiwane przez Springa..

```
Source    General    Servlets    Filters    Pages    References    Security
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org
3  <
4  <servlet>
5  <servlet-name>Przyklad</servlet-name>
6  <servlet-class>pl.jsystems.springmvc.controller.Przyklad</servlet-class>
7  </servlet>
8  <
9  <servlet-mapping>
10 <servlet-name>Przyklad</servlet-name>
11 <url-pattern>/jakismodul/*</url-pattern>
12 </servlet-mapping>
13
14 <
15 <servlet>
16 <servlet-name>springmvc</servlet-name>
17 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
18 </servlet>
19 <servlet-mapping>
20 <servlet-name>springmvc</servlet-name>
21 <url-pattern>/spring/*</url-pattern>
22 </servlet-mapping>
23
24
25 <session-config>
26 <session-timeout>
27 30
28 </session-timeout>
29 </session-config>
30 </web-app>
31
```

OK, sprawdźmy teraz co się stanie kiedy wywołamy w przeglądarce adres nowego modułu:

localhost:12559/SpringMVC/spring/

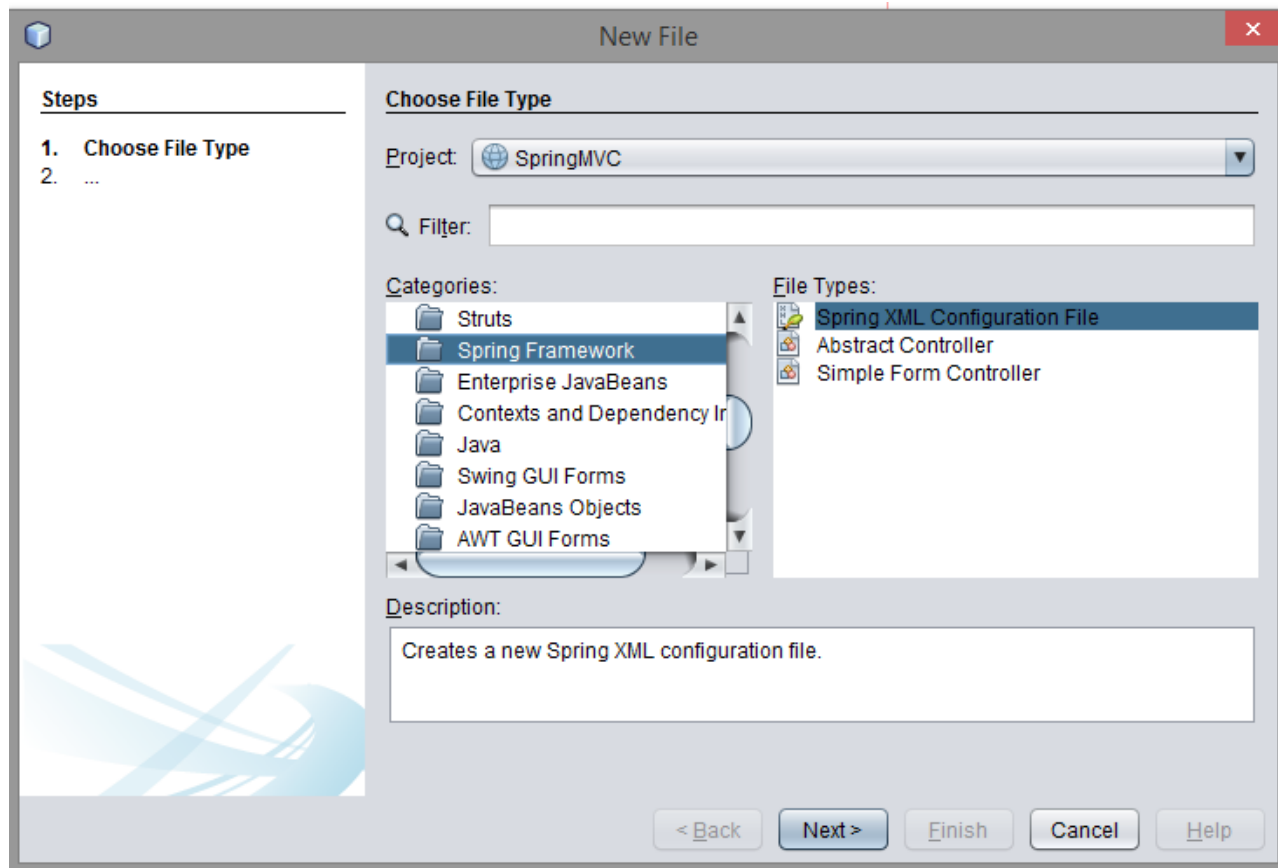
HTTP Status 500 - Internal Server Error

type Exception report
message Internal Server Error
description The server encountered an internal error that prevented it from fulfilling this request.
exception
javax.servlet.ServletException: Servlet.init() for servlet springmvc threw exception
root cause
org.springframework.beans.factory.BeanDefinitionStoreException: IOException parsing XML document from S
root cause
java.io.FileNotFoundException: Could not open ServletContext resource [/WEB-INF/springmvc-servlet.xml]
note The full stack traces of the exception and its root causes are available in the GlassFish Server Open Source Edition 4.1 logs.

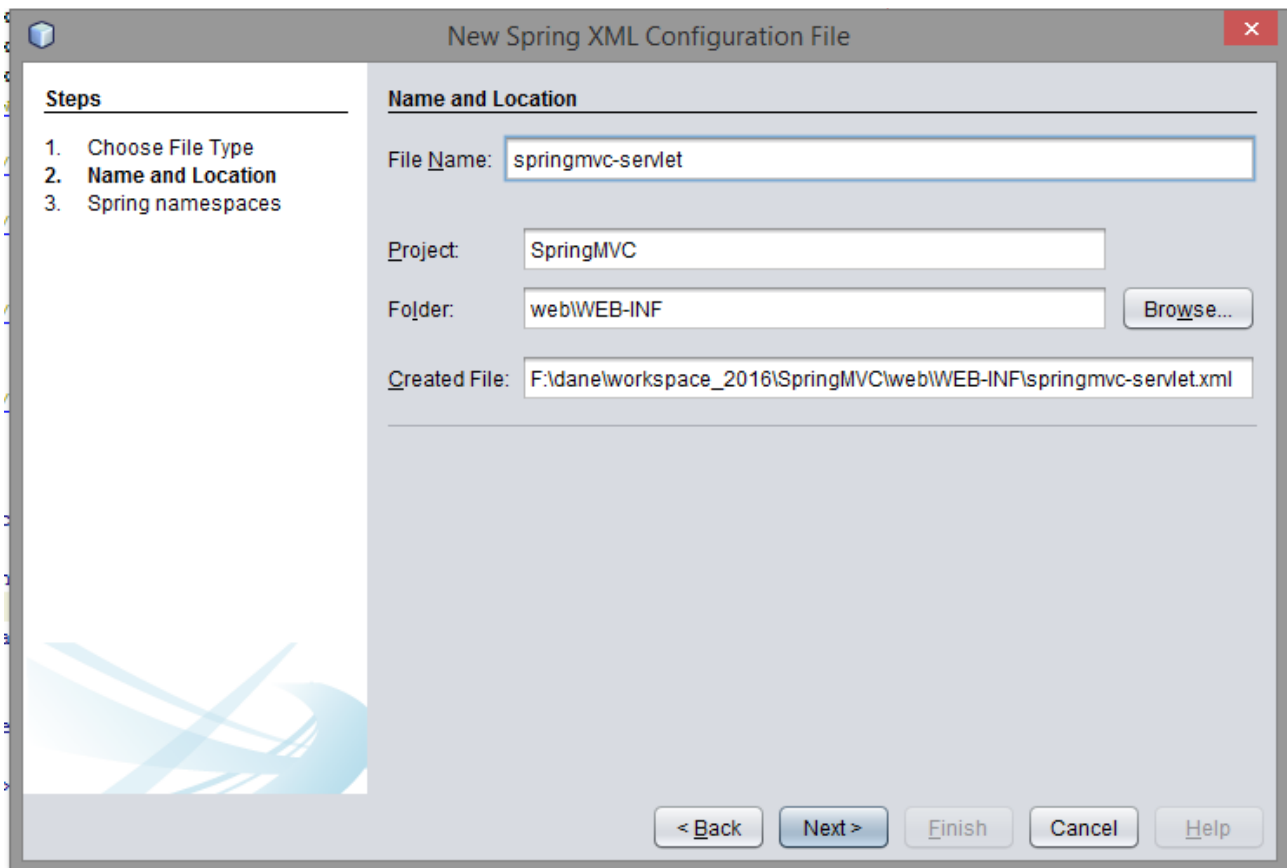
GlassFish Server Open Source Edition 4.1

Pojawił się błąd, a gdy mu się przyjrzymy zobaczymy że problem polega na braku pliku springmvc-servlet.xml To jest plik konfiguracyjny Spring MVC w którym będziemy deklarowali m.in. kontrolery naszej aplikacji. Nazwa pliku springmvc-servlet.xml nie jest przypadkowa. Wynika ona z tego jak nazwaliśmy servlet dla klasy DispatcherServlet w linii 15 pliku web.xml. Gdybyś w tym miejscu wpisał zamiast springmvc np. gdzieJestNemo, Spring szukałby pliku gdzieJestNemo-servlet.xml. Wielkość liter ma znaczenie.

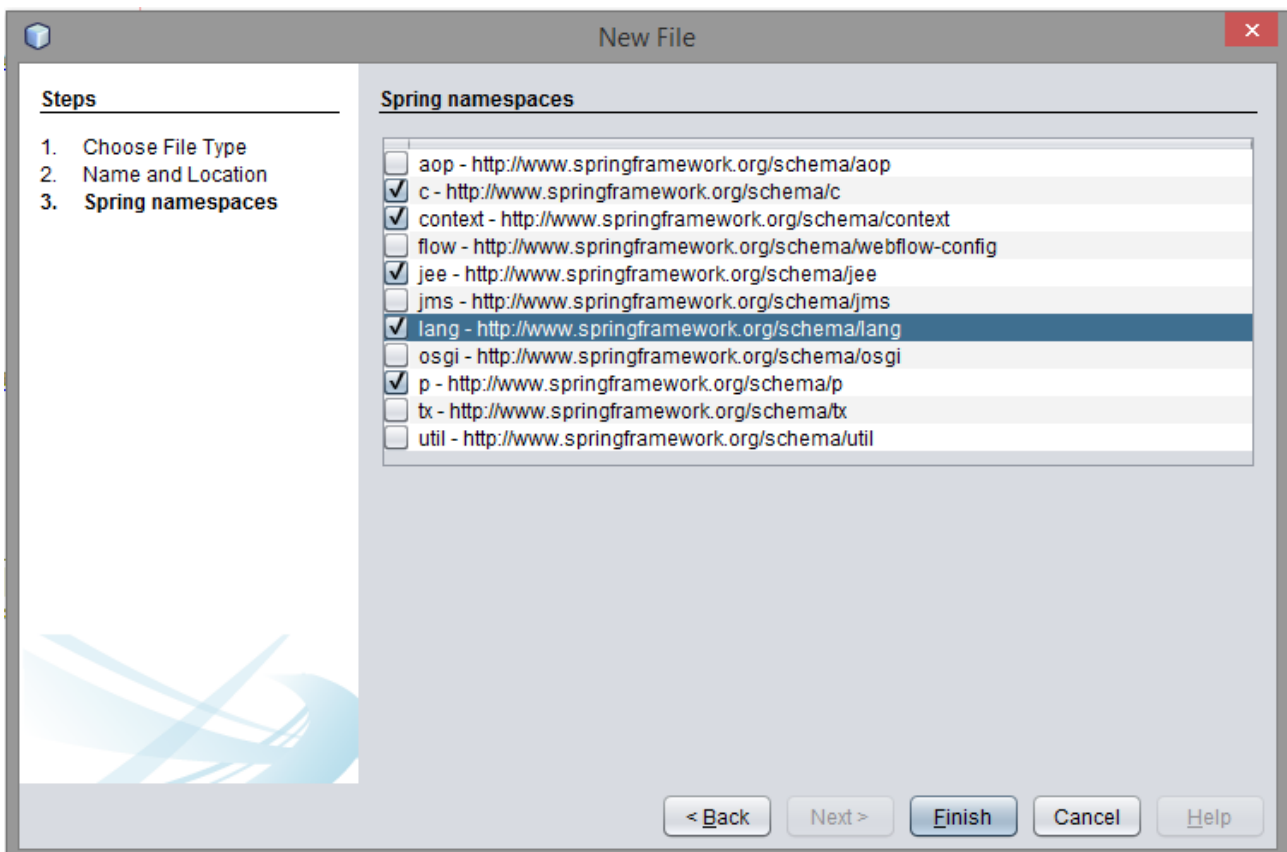
Skoro Spring tak bardzo potrzebuje tego pliku, to mu go dajmy :



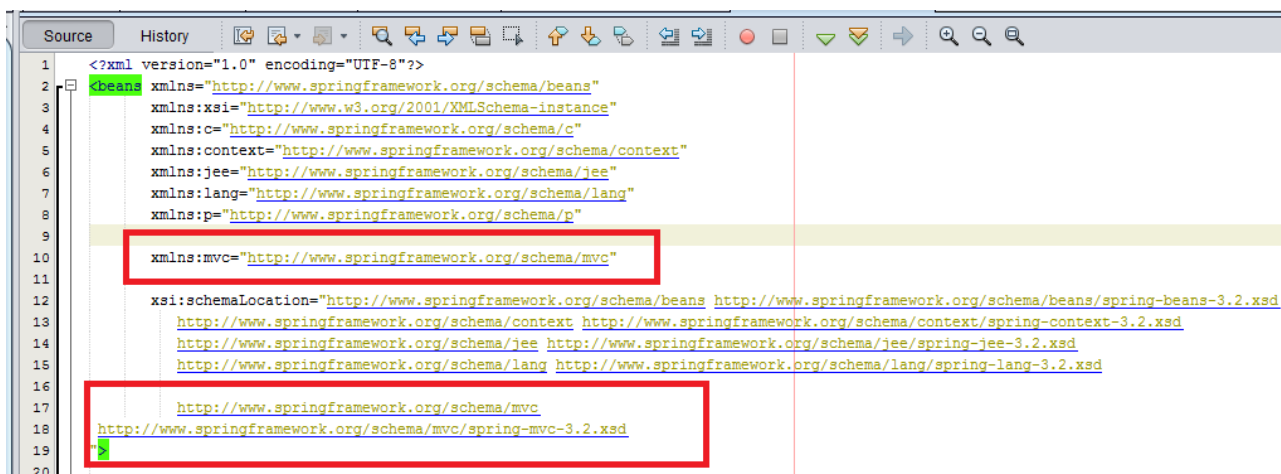
Nadamy mu nazwę springmvc-servlet.xml:



Będą nam też potrzebne pewne przestrzenie nazw XML które będziemy wykorzystywać, dlatego je zaznaczamy:

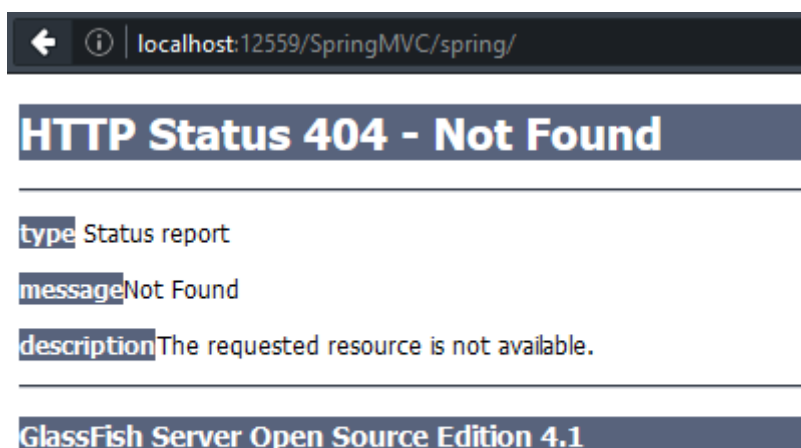


Nie ma możliwości wyboru przestrzeni MVC dlatego musimy dodać ją ręcznie do nowego pliku. Możesz też gotowy plik pobrać z przykładowego kodu którego adres znajdziesz na początku tego rozdziału i umieścić go w katalogu WEB-INF.

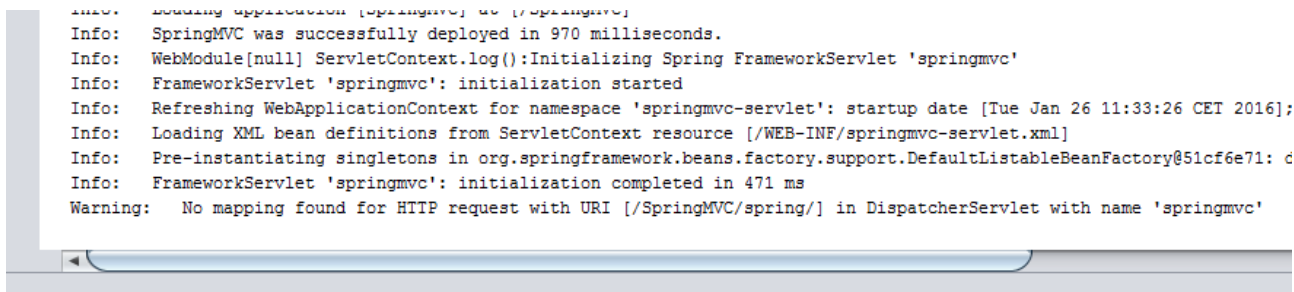


```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xmlns:c="http://www.springframework.org/schema/c"
5       xmlns:context="http://www.springframework.org/schema/context"
6       xmlns:jee="http://www.springframework.org/schema/jee"
7       xmlns:lang="http://www.springframework.org/schema/lang"
8       xmlns:p="http://www.springframework.org/schema/p"
9
10      xmlns:mvc="http://www.springframework.org/schema/mvc"
11
12      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd
13                        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd
14                        http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd
15                        http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang-3.2.xsd
16
17                        http://www.springframework.org/schema/mvc
18                        http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
19
20
```

Ponówmy próbę dostępu do modułu:



Zaglądamy do konsoli serwera:



```
Info: Loading application [springmvc] at [/springmvc]
Info: SpringMVC was successfully deployed in 970 milliseconds.
Info: WebModule[null] ServletContext.log():Initializing Spring FrameworkServlet 'springmvc'
Info: FrameworkServlet 'springmvc': initialization started
Info: Refreshing WebApplicationContext for namespace 'springmvc-servlet': startup date [Tue Jan 26 11:33:26 CET 2016];
Info: Loading XML bean definitions from ServletContext resource [/WEB-INF/springmvc-servlet.xml]
Info: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@51cf6e71: d
Info: FrameworkServlet 'springmvc': initialization completed in 471 ms
Warning: No mapping found for HTTP request with URI [/SpringMVC/spring/] in DispatcherServlet with name 'springmvc'
```

Widzimy że Spring odnalazł nasz nowy plik (linia z Loading XML bean definitions...). Przyjrzyjmy się teraz ostatniej linii z ostrzeżeniem. Problem polega na tym, że nie określiliśmy w pliku springmvc-servlet.xml przez jaką klasę ma być obsługiwany adres /SpringMVC/spring/.

Dodajemy więc nowy wpis do springmvc-servlet.xml:

```
16
17     http://www.springframework.org/schema/mvc
18     http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd
19     ">
20
21     <mvc:annotation-driven/>
22
23     <context:component-scan base-package="pl.jsystems.springmvc.controller"/>
24
25
26 </beans>
27
```

Linia 21 oznacza, że Spring ma poszukać deklaracji mapowań obsługiwanych adresów w adnotacjach znajdujących się w klasach pakietu (i jego podpakietów) który wskazaliśmy w linii 23 :)

Do pakietu pl.jsystems.springmvc.controller dodajemy teraz zwyczajną klasę Hello. Wprowadzamy metodę „sayHello” która wypisuje na konsoli serwera tekst „HELLO MUPPET”. Koniecznie dodaj wpis @Controller nad deklaracją klasy, oraz @RequestMapping nad metodą SayHello.

```
6     package pl.jsystems.springmvc.controller;
7
8     import org.springframework.stereotype.Controller;
9     import org.springframework.ui.Model;
10    import org.springframework.web.bind.annotation.RequestMapping;
11
12    /**
13     *
14     * @author andrzej
15     */
16    @Controller
17    public class Hello {
18
19        @RequestMapping("/hello.do")
20        public String sayHello(Model model) {
21            System.out.println("HELLO MUPPET!");
22            return "hello";
23        }
24    }
25
```

Wpis @Controller z linii 16 deklaruje, że klasa ta obsługuje żądania HTTP, a @RequestMapping z parametrem wskazują która klasa do robi i dla jakiego konkretnie żądania. Adres /hello.do jest względny i oznacza wywołanie /SpringMVC/spring/hello.do.

Wywołajmy więc ten adres:



Nasze ulubione 404. Czy to znaczy że coś nie zadziałało? To zależy :) „This is not a bug, this is a feature” :) A tak poważnie – wszystko zgodnie z planem. Zajrzyjmy do konsoli serwera:

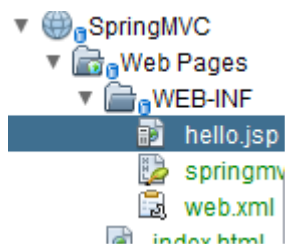


Wyświetlił nam się komunikat „Hello Muppet”, a to oznacza że nasza metoda sayHello została zgodnie z założeniem wywołana. Błąd 404 pojawia się dlatego, że nie mam strony JSP którą powinienem w odpowiedzi wyświetlić. Konkretnie to plik powinien się nazywać hello.jsp – ponieważ metoda sayHello zwraca ciąg tekstowy „hello” i powinien znajdować się bezpośrednio w katalogu WEB-INF co zdeklarujemy sobie w pliku springmvc-servlet:



W linii 27 deklaruje gdzie Spring ma szukać plików JSP, a w linii 28 jakie mają mieć rozszerzenie. Jeśli chcesz, w linii 27 możesz dodać jakiś podkatalog np. /WEB-INF/jsp/, albo wydzielić w ogóle osobny katalog na pliki jsp związane z tym modulem np. /WEB-INF/jsp/spring/

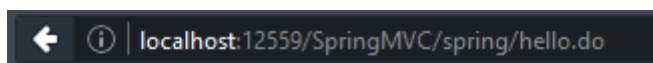
Stworzymy jeszcze w zadeklarowanym katalogu plik hello.jsp:



i umieścimy w nim taki oto kod:

```
7 <%@page contentType="text/html" pageEncoding="UTF-8"%>
8 <!DOCTYPE html>
9 <html>
10 <head>
11 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12 <title>JSP Page</title>
13 </head>
14 <body>
15 <h1>Hello Muppet!</h1>
16 </body>
17 </html>
18
```

Teraz przy wywołaniu adresu /SpringMVC/spring/hello.do powinniśmy zobaczyć taki komunikat:



Hello Muppet!

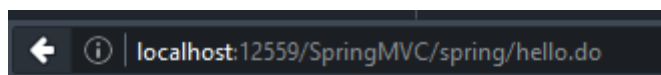
Przydałoby się jednak zrobić coś więcej niż wyświetlanie statycznej wartości. Przekażmy więc jakieś dane z kontrolera do widoku (patrz linia 23):

```
12  /**
13  *
14  * @author andrzej
15  */
16  @Controller
17  public class Hello {
18
19      @RequestMapping("/hello.do")
20      public String sayHello(Model model) {
21          System.out.println("HELLO MUPPET!");
22          String info="witaj w świecie Spring MVC!";
23          model.addAttribute("wiadomosc", info);
24          return "hello";
25      }
26  }
27
```

a następnie wyświetlmy ją na stronie JSP (patrz linia 16):

```
5  -->
6  <@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
7  <@page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE html>
9  <html>
10 <head>
11     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12     <title>JSP Page</title>
13 </head>
14 <body>
15     <h1>Hello Muppet!</h1>
16     <h3>${wiadomosc}</h3>
17 </body>
18 </html>
19
```

Zauważ że w JSP odwołuję się do przekazanego elementu po nazwie która ustaliłem w pierwszym parametrze metody `addAttribute` tj. „wiadomosc”. Efekt:



Hello Muppet!

witaj w świecie Spring MVC!

Przekazywać oczywiście możemy też obiekty, listy, a także możemy obsługiwać formularze, ale tym zajmiemy się w kolejnych częściach tego kursu. W tej chwili zrobiliśmy chyba najprostszą możliwą implementację Spring MVC. Mamy oczywiście wiele możliwych wariantów – jak choćby w miejsce adnotacji użycie deklaracji beanów w pliku XML.

Jeszcze pozwolę sobie na małe rozwinięcie tematu zarządzania wywołaniami. Przypuśćmy że tworzymy dużą aplikację złożoną z kilku modułów i chcielibyśmy mieć osobne pliki konfiguracyjne. Kod źródłowy do następnych przykładów znajdziesz pod adresem :

<http://www.jsystems.pl/storage/spring/springmvc2.zip>

Dodamy sobie kolejną paczkę do `web.xml`:

```
23
24 <servlet>
25     <servlet-name>drugimodul</servlet-name>
26     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
27 </servlet>
28 <servlet-mapping>
29     <servlet-name>drugimodul</servlet-name>
30     <url-pattern>/drugimodul/*</url-pattern>
31 </servlet-mapping>
32
33
```


W związku z tym że nasz serwlet nazywa się drugimodul dodajemy też plik „drugimodul-servlet.xml” . W nim dodajemy takie wpisy jak poprzednio, z tą różnicą że pliki JSP znajdują się w osobnym podkatalogu w WEB-INFie (patrz linia 27), a dodatkowo wydzielimy sobie osobny pakiet na kontrolery tego modułu. Dzięki temu będziemy mogli mieć klasy kontrolerów o takiej samej nazwie.

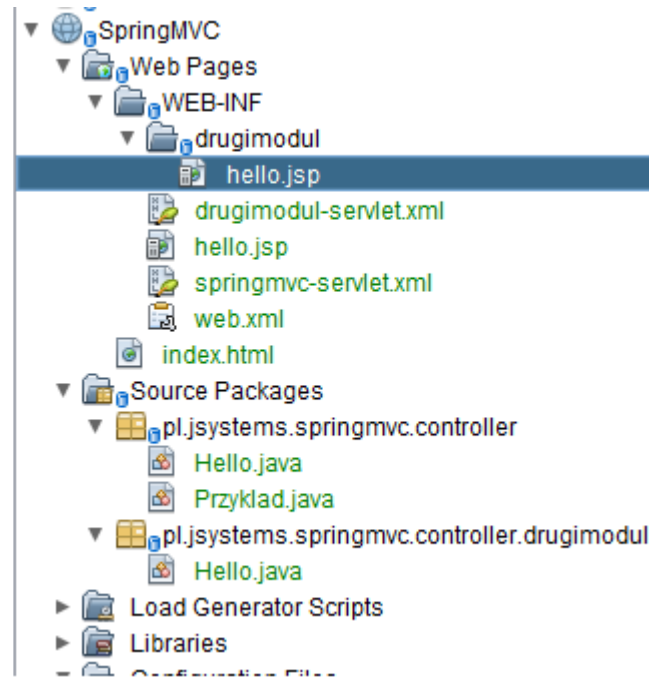
```
20
21     <mvc:annotation-driven/>
22
23     <context:component-scan base-package="pl.jsystems.springmvc.controller.drugimodul"/>
24
25
26     <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
27         <property name="prefix" value="/WEB-INF/drugimodul/"/>
28         <property name="suffix" value=".jsp"/>
29     </bean>
30
31 </beans>
32
```

Oczywiście taki pakiet tworzymy, a w nim umieszczamy odrobinę różniącą się klasę kontrolera:

```
5  L  */
6  package pl.jsystems.springmvc.controller.drugimodul;
7
8  import pl.jsystems.springmvc.controller.*;
9  import org.springframework.stereotype.Controller;
10 import org.springframework.ui.Model;
11 import org.springframework.web.bind.annotation.RequestMapping;
12
13 /**
14  *
15  * @author andrzej
16  */
17 @Controller
18 public class Hello {
19
20     @RequestMapping("/hello.do")
21     public String sayHello(Model model) {
22         System.out.println("HELLO MUPPET w drugim module!");
23         String info="witaj w świecie Spring MVC!";
24         model.addAttribute("wiadomosc", info);
25         return "hello";
26     }
27 }
28
```

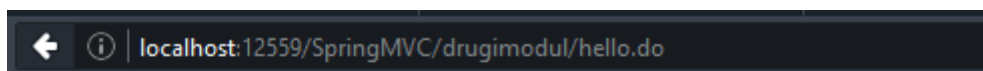
Zauważ że tutaj również określone jest mapowanie dla „/hello.do”, ale jak pamiętamy jest to adres względny i w tym przypadku oznacza wywołanie „/SpringMVC/drugimodul/hello.do”, a nie „/SpringMVC/spring/hello.do”. Metoda nadal zwraca tekst hello, co oznacza że Spring poszuka pliku hello.jsp by go wyświetlić, tym razem jednak będzie go szukał w katalogu „WEB-INF/drugimodul”.

W katalogu WEB-INF tworzymy podkatalog (taki jaki wskazaliśmy we wpisie w pliku drugimodul-servlet.xml) i umieszczamy w nim plik jsp analogiczny do poprzedniego, z tym że dla odróżnienia zmienimy troszkę jego zawartość.



```
5  L  -->
6  <@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
7  <@page contentType="text/html" pageEncoding="UTF-8"%>
8  <!DOCTYPE html>
9  <html>
10     <head>
11         <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12         <title>JSP Page</title>
13     </head>
14     <body>
15         <h1>Hello Muppet w drugim module!</h1>
16         <h3>${wiadomosc}</h3>
17     </body>
18 </html>
```

Sprawdźmy:



Hello Muppet w drugim module!

witaj w świecie Spring MVC!