Tworzenie stron WWW z wykorzystaniem Java.

Java to nie tylko język wykorzystywany w klasycznych aplikacjach. Java pozwala także na wykonywanie operacji po stronie serwerów stron WWW. W przeciwieństwie do innych języków, takich jak PHP, JavaScript czy Python, strona pisana w Java jest kompilowana do aplikacji binarnej, co pozwala na bardzo szybkie wykonywanie napisanej aplikacji. To z kolei powoduje, że strony wykorzystujące ten język działają znacznie szybciej od pozostałych.

Wykorzystanie języka Java do tworzenia stron WWW ma sporo zalet, z której największą jest szybkość działania. Inny aspektem jest lepsza współpraca z JavaScript oraz możliwość osadzania aplikacji pisanych w Java.

Poniższy plik ma charakter informacyjny jak rozpocząć tworzenie stron WWW z wykorzystaniem JSP, wskazuje na szczegóły rozwiązania. Zakłada przy tym, że czytający miał już do czynienia z HTML, JavaScript oraz dowolnym, innym językiem serwerowym. Jednak jeżeli nie – przykłady kodu powinny nieco rozjaśniać mechanizmy zachodzące w projektach.

1. Konfiguracja środowiska.

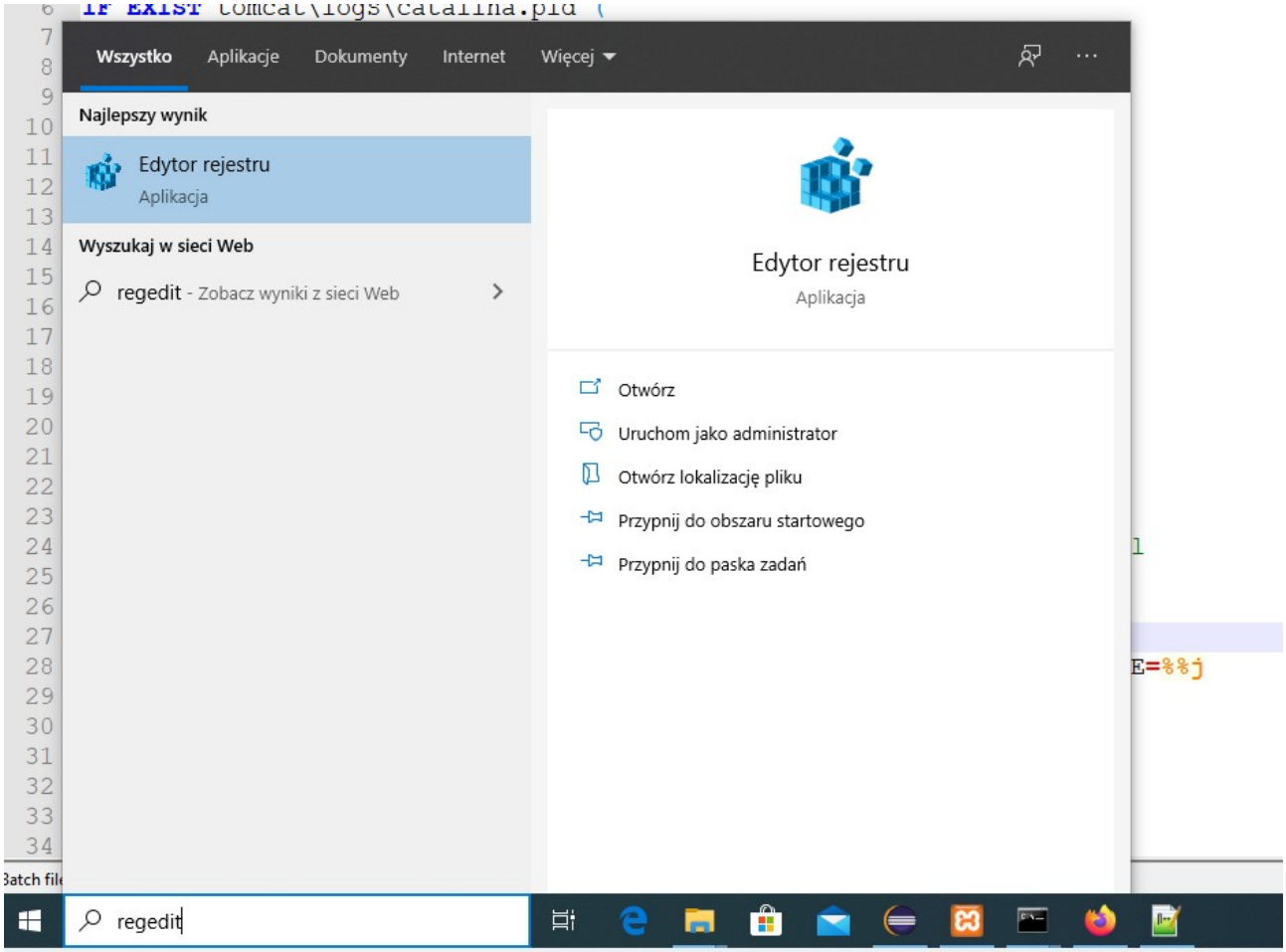
W przeciwieństwie do dotychczas używanych aplikacji Java, nasza technologia wymaga dodatkowo serwera stron WWW. Nie ma bowiem możliwości uruchomienia przykładów tak jak zwykłych programów wykonywalnych.

Celem demonstracji działania wykorzystany zostanie dobrze znany z baz danych XAMPP. Podczas instalacji posiada on opcję Apache Tomcat. Nie jest to nic innego jak serwer stron WWW, który interpretuje język Java umieszczany na stronach WWW.

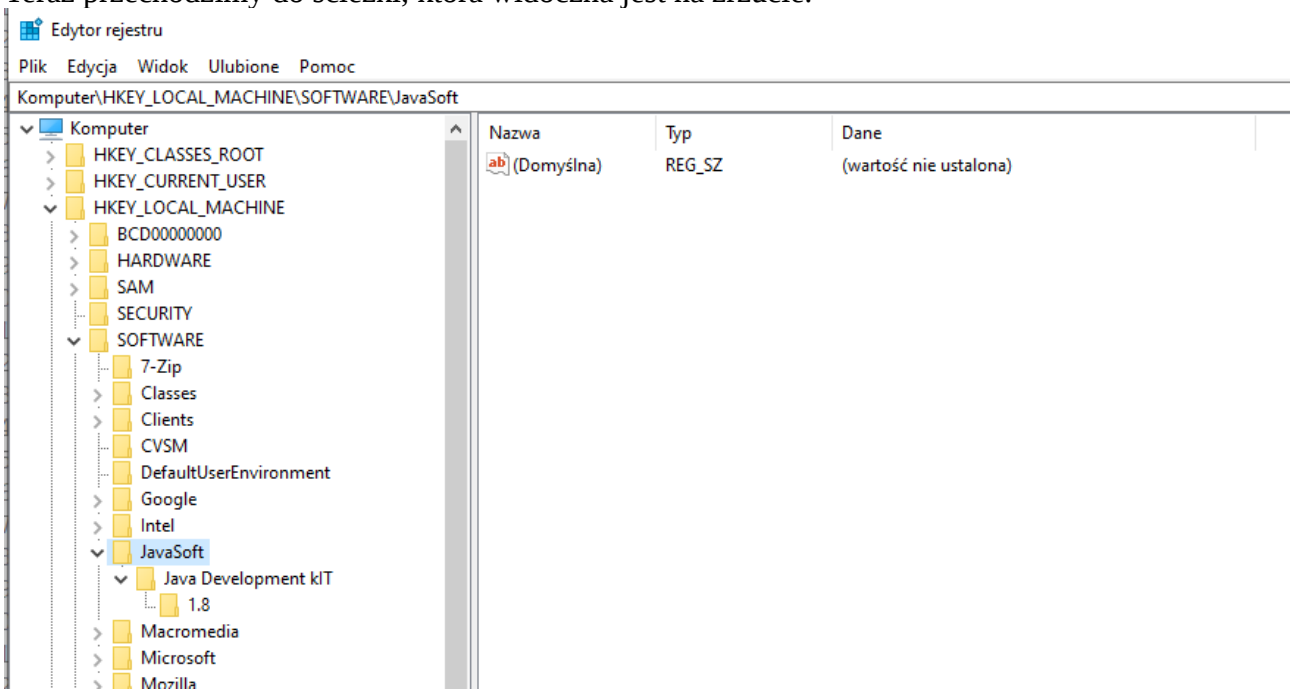
Ponieważ na zajęciach korzystamy z OpenJDK (nie zaś z JRE dostępnego od firmy Oracle) musimy brać pod uwagę, że będziemy zmuszeni do ręcznej modyfikacji jednego z parametrów:
- serwera
LUB
- systemu operacyjnego.

W drugim wypadku będziemy zmuszeni do dodania w ścieżce systemowej (PATH) dostępu do naszej maszyny wirtualnej Java. Jeżeli nie korzystamy z niej na co dzień – dodanie tej zmiennej może nieznacznie opóźniać nam start całego systemu operacyjnego. Ponadto może zdarzyć się, że cała operacja nie przyniesie zamierzonego rezultatu (ponieważ realnie Tomcat z XAMPP przegląda klucze rejestru).

W związku z tym lepszym rozwiązaniem będzie zmiana rejestru systemowego. W tym celu otwieramy regedit:

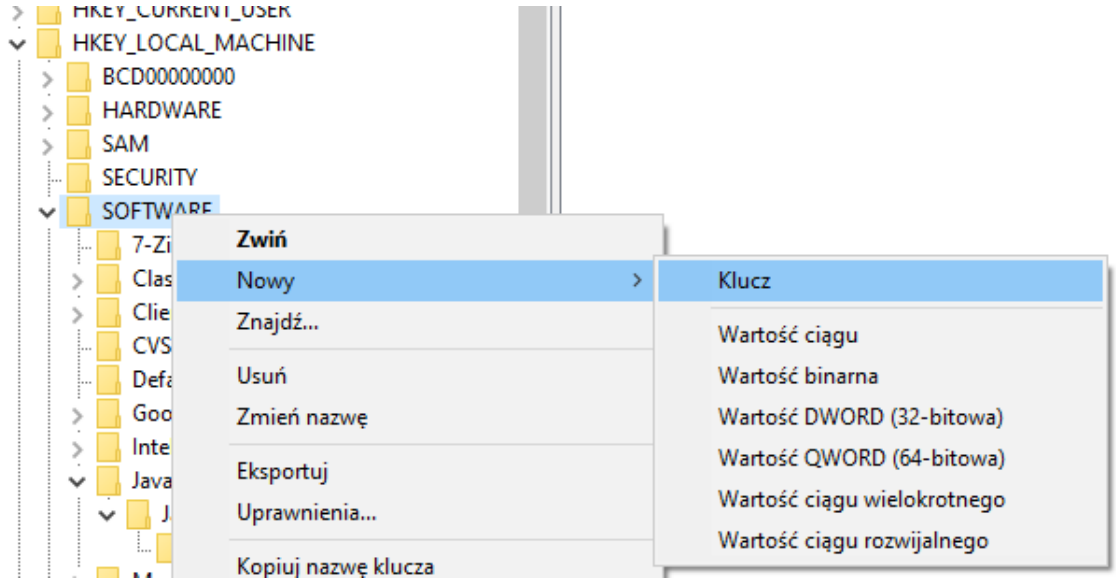


Najlepiej jest wybrać opcję Uruchom jako administrator (chociaż to narzędzie w inny sposób nie powinno się uruchomić).
Teraz przechodzimy do ścieżki, która widoczna jest na zrzucie.

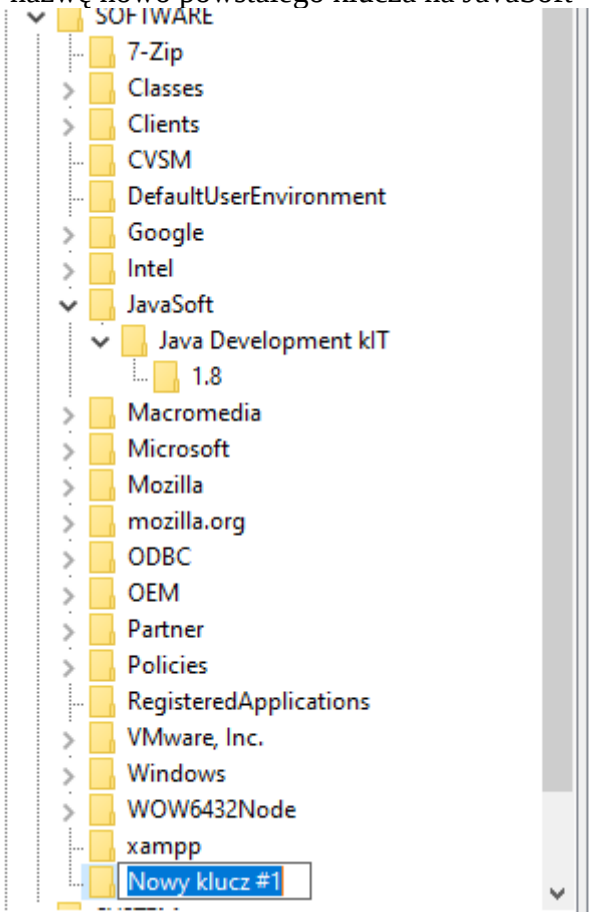


Jeżeli jej nie posiadamy – tworzymy ją w następujący sposób:

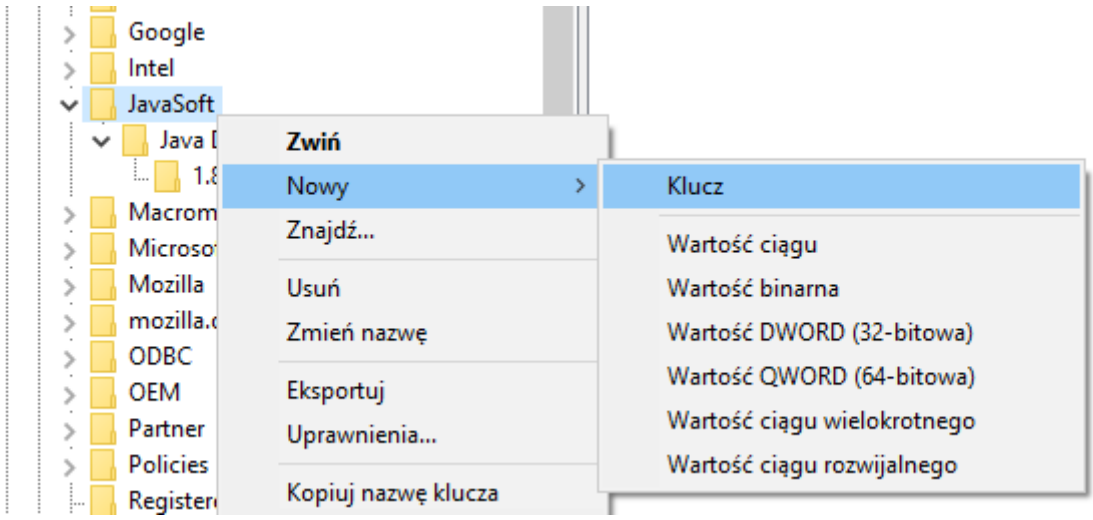
- przechodzimy do ścieżki: HKEY_LOCAL_MACHINE\SOFTWARE (lub HKLM\SOFTWARE)
- klikamy na SOFTWARE prawym przyciskiem myszy i wybieramy Nowy→Klucz:



Teraz zmieniamy nazwę nowo powstałego klucza na JavaSoft

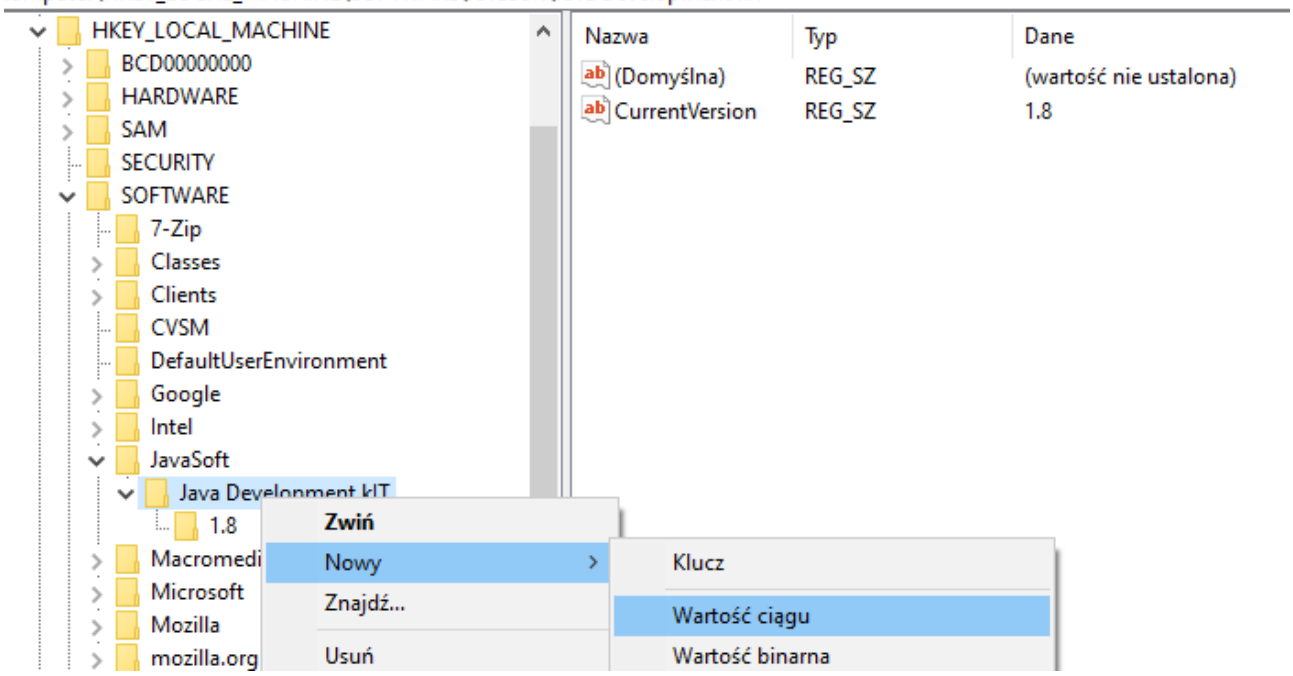


Mając nowy klucz klikamy na niego prawym przyciskiem myszy i powtarzamy operację:

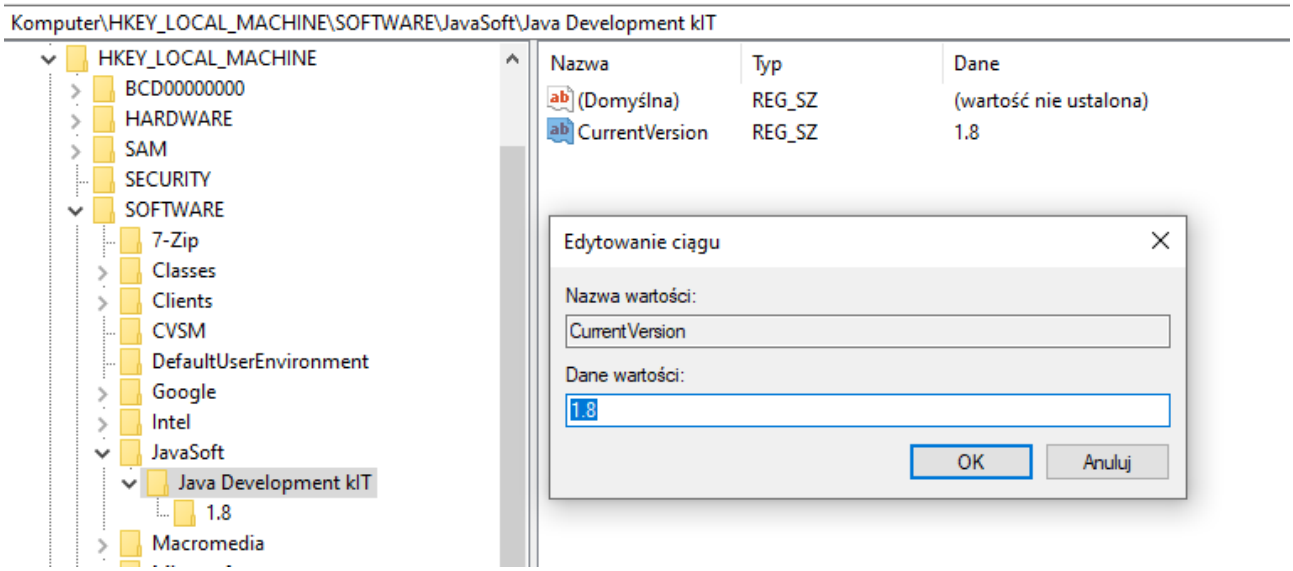


Tym razem klucz nazywamy Java Development KIT. Na koniec klikamy na nowym kluczu prawym przyciskiem myszy i tworzymy nowy klucz o nazwie 1.8

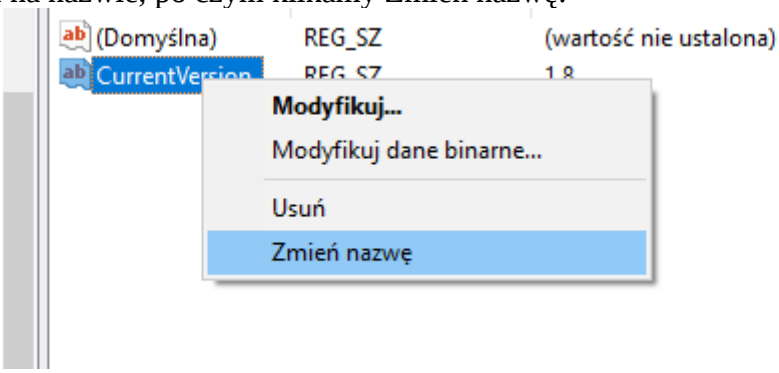
Mając już powyższe klucze w wybieramy lewym przyciskiem myszy Java Development KIT. Klikamy na nim prawym przyciskiem myszy i tworzymy Wartość ciągu:



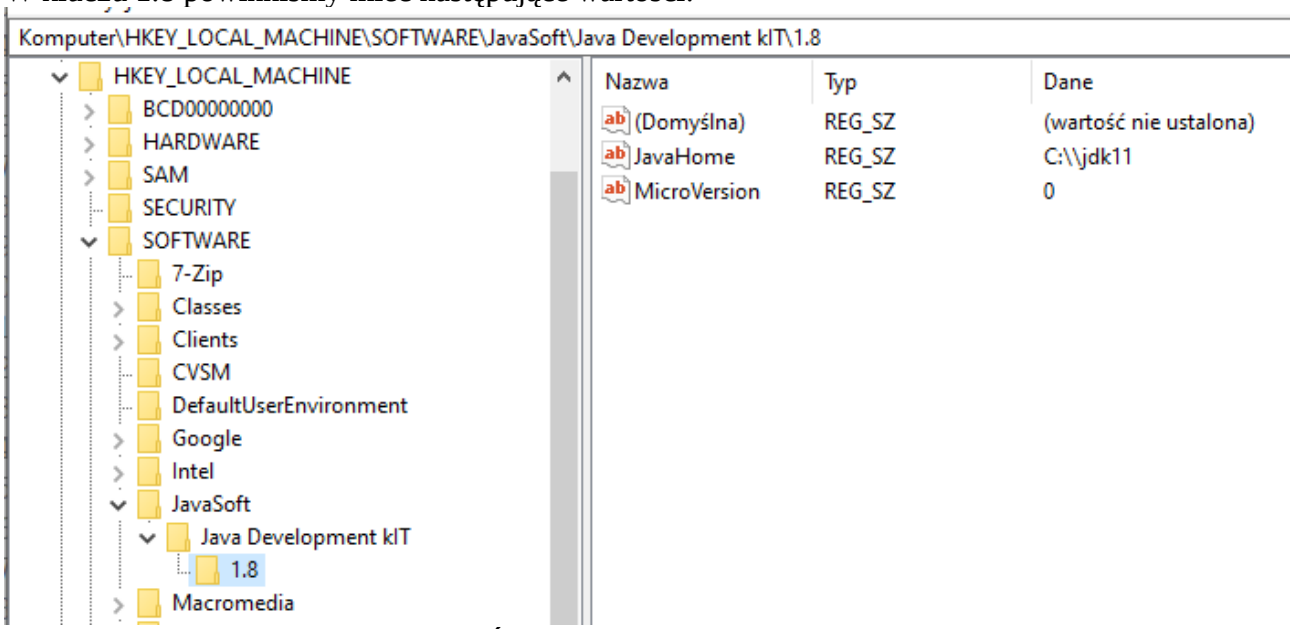
Nazwę zmieniamy na CurrentVersion. Klikamy dwa razy w nową wartość i nadajemy jej Dane 1.8:



Jeżeli ustawiliśmy niepoprawną nazwę dla ciągu to możemy ją zmienić, klikając prawym przyciskiem na nazwie, po czym klikamy Zmień nazwę:



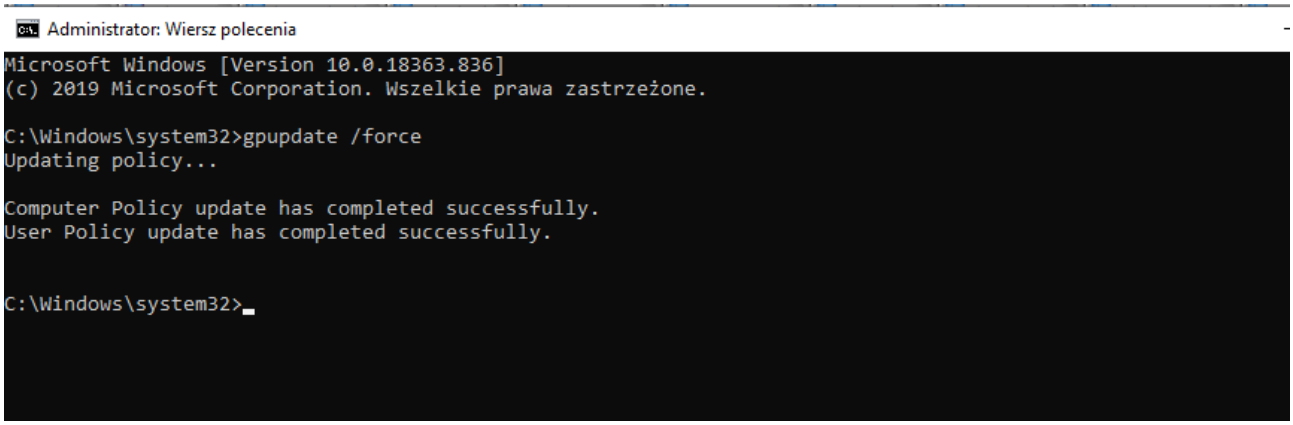
W kluczu 1.8 powinniśmy mieć następujące wartości:



Z czego JavaHome MUSI POSIADAĆ ścieżkę do naszej JVM (tutaj to C:\jdk11; proszę pamiętać, że w rejestrze należy za każdym razem podawać dla znaki backslash!).

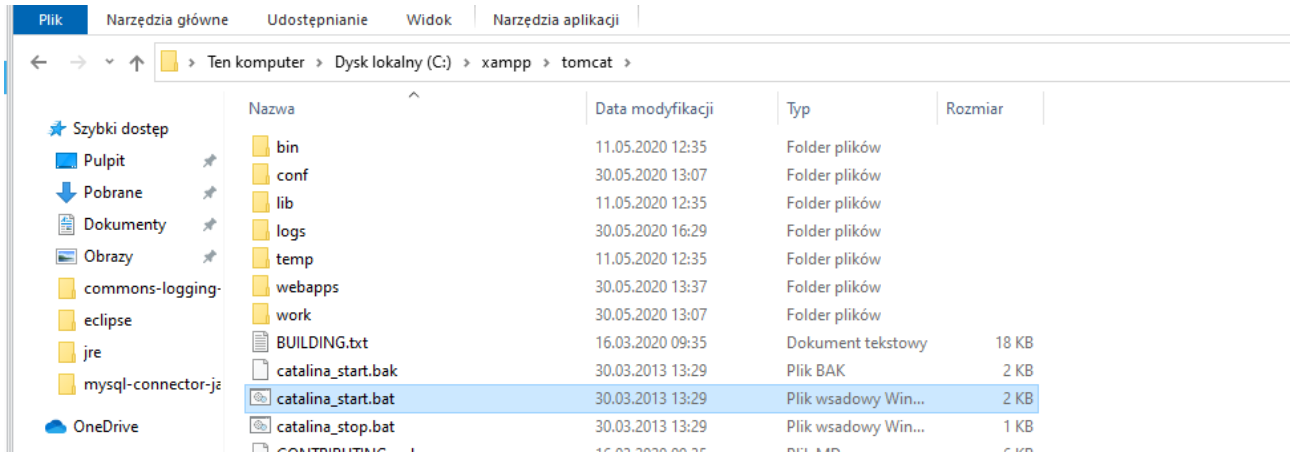
Na koniec otwieramy linię poleceń (najlepiej z prawami administracyjnymi) i wykonujemy następujące polecenie:

gpupdate /force



Spowoduje to przeładowanie rejestru i uwzględnienie ścieżki do naszej JVM.

Istnieje też możliwość zmiany pliku startowego samego serwera. W tym celu należy udać się do folderu serwera Tomcat:



Należy pamiętać, że serwer ten może znajdować się w innej ścieżce instalacji (np. na innym dysku). Plik catalina_start.bat możemy na wszelki wypadek skopiować i przemieścić tak, by ewentualnie móc go później przywrócić (na powyższym zrzucie catalina_start.bak jest taką kopią).

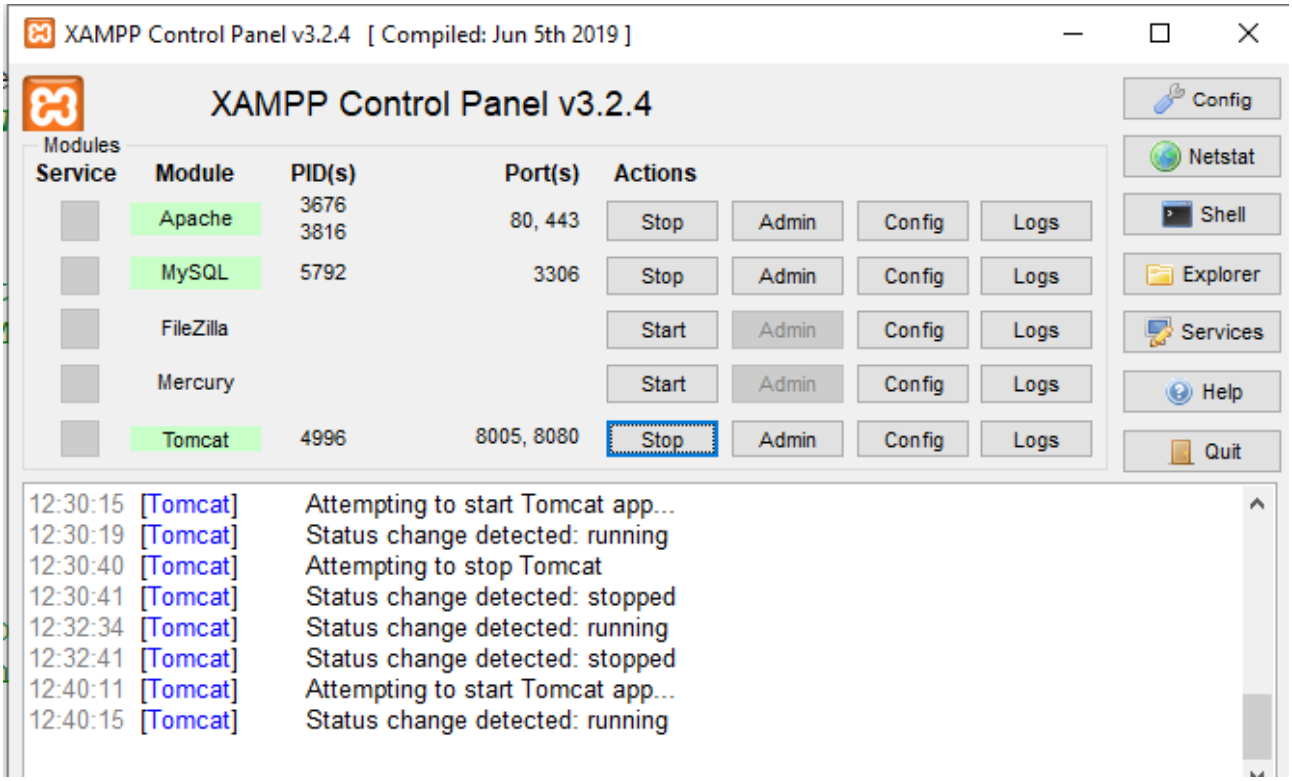
Plik zmieniamy jak na zrzucie:

```
@echo off
::::::::::::::::::::::::::::::::::::
::  Set JAVA_HOME and   ::
::::::::::::::::::::::::::::::::::::

IF EXIST tomcat\logs\catalina.pid (
  del /F/Q tomcat\logs\catalina.pid
)

echo.
echo [XAMPP]: Searching JDK HOME with reg query ...
rem set KeyName=HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Development Kit

rem reg query "%KeyName%" /s
rem if %ERRORLEVEL% == 1 (
rem  echo . [XAMPP]: Cannot find current JDK installation!
rem  echo . [XAMPP]: Cannot set JAVA_HOME. Aborting ...
rem   goto :END
rem )

set "CURRENT_DIR=%cd%"
set "CATALINA_HOME=%CURRENT_DIR%"

:: only for windows 32 bit if you have problems with the tcnative-1.dll
:: set CATALINA_OPTS=-Djava.library.path="%CATALINA_HOME%\bin"

rem set Cmd=reg query "%KeyName%" /s
rem for /f "tokens=2*" %%i in ('%Cmd% ^| find "JavaHome"') do set JAVA_HOME=%%j

set JAVA_HOME="C:\jdk11"

echo.
echo [XAMPP]: Seems fine!
echo [XAMPP]: Set JAVA_HOME : %JAVA_HOME%
```

To co nie jest nam potrzebne zostało obłożone komentarzem (rem). Zaś ręcznie dołączona została zmienna JAVA_HOME, wskazująca na folder z JVM. Oczywiście należy wskazać katalog z pobranym OpenJDK.

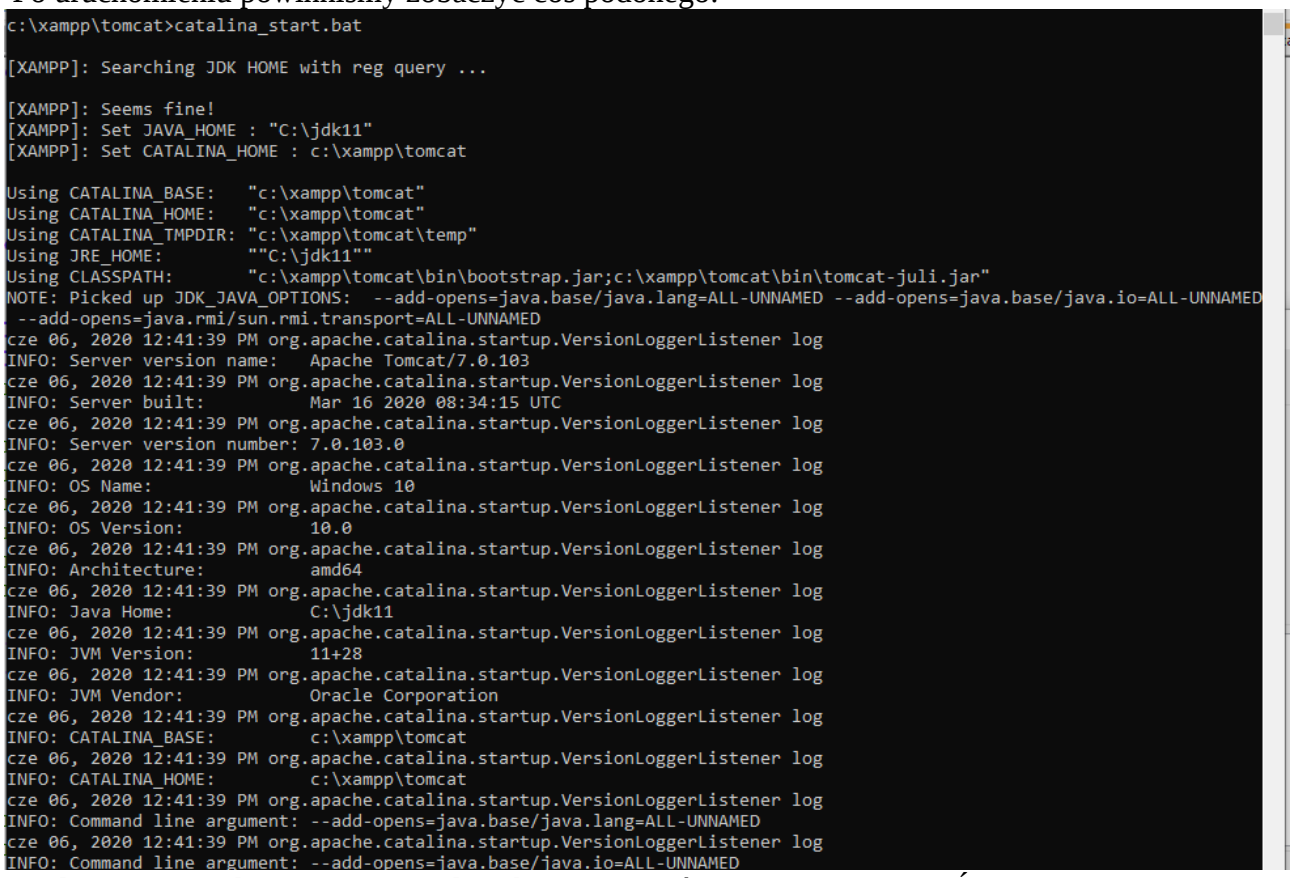
Mając już taką konfigurację powinniśmy mieć możliwość uruchomienia naszego serwera. W tym celu:

- jeżeli zmieniliśmy wartości w rejestrze odpalamy serwer przez panel XAMPP:



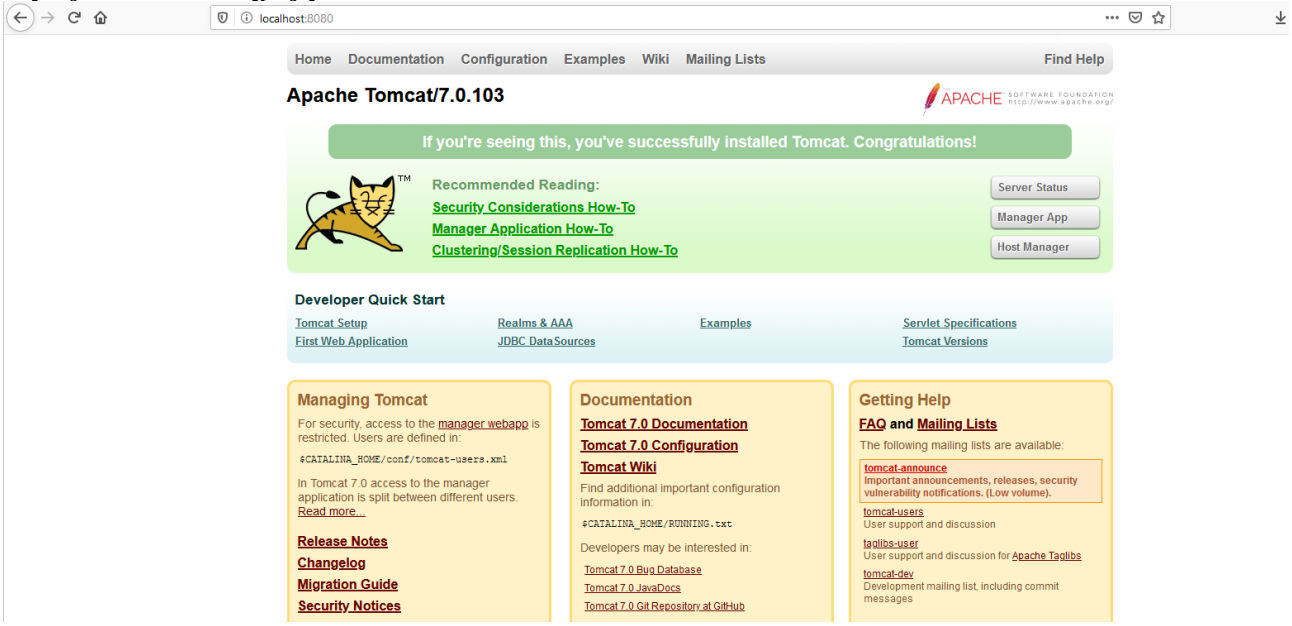
- jeżeli z jakichś powodów konfiguracja nie działa, otwieramy linię poleceń i odpalamy plik catalina_start.bat:

Po uruchomieniu powinniśmy zobaczyć coś podonego:



Jeżeli serwer uruchomimy tą metodą należy pamiętać BY NIE ZAMYKAĆ OKNA CMD!

Niezależnie od sposobu uruchomienia, po wpisaniu adresu localhost:8080 w przeglądarkę powinna się wyświetlić następująca strona WWW:



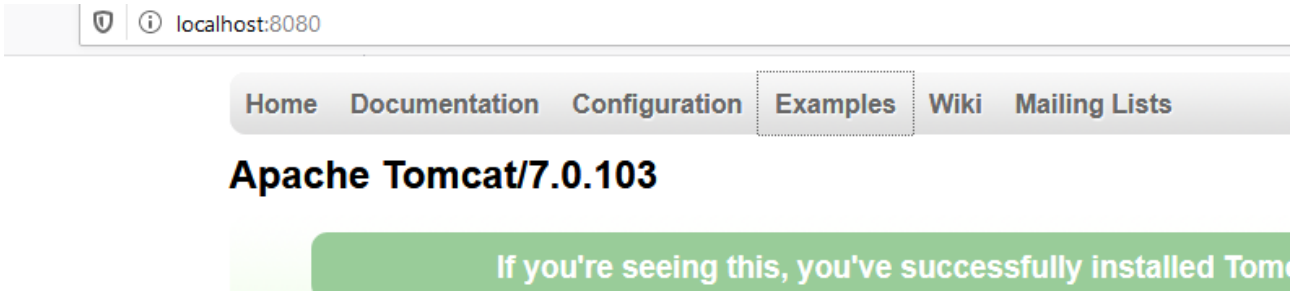
Będzie to znaczyło, że konfiguracja serwera jest poprawna i wszystko powinno działać.

INFORMACJA: W przeciwieństwie do innych witryn, po nazwie naszego serwera (w tym wypadku to nasza lokalna maszyna - localhost) pojawia się dwukropek i seria cyfr. To numer portu, czyli jednoznaczny identyfikator usługi, do której chcemy się połączyć. Ponieważ najczęściej „podstawowy" serwer Apache działa na porcie 80 (którego nie musimy podawać przeglądarce – sama się domyśla, że skoro go nie podajemy to chodzi nam właśnie o ten), Tomcat ma zmienioną konfigurację by działał (nasłuchiwał zapytań) właśnie na porcie 8080. Wartość tę można zmienić w pliku server.xml (dostępnym w katalogu tomcat/conf):

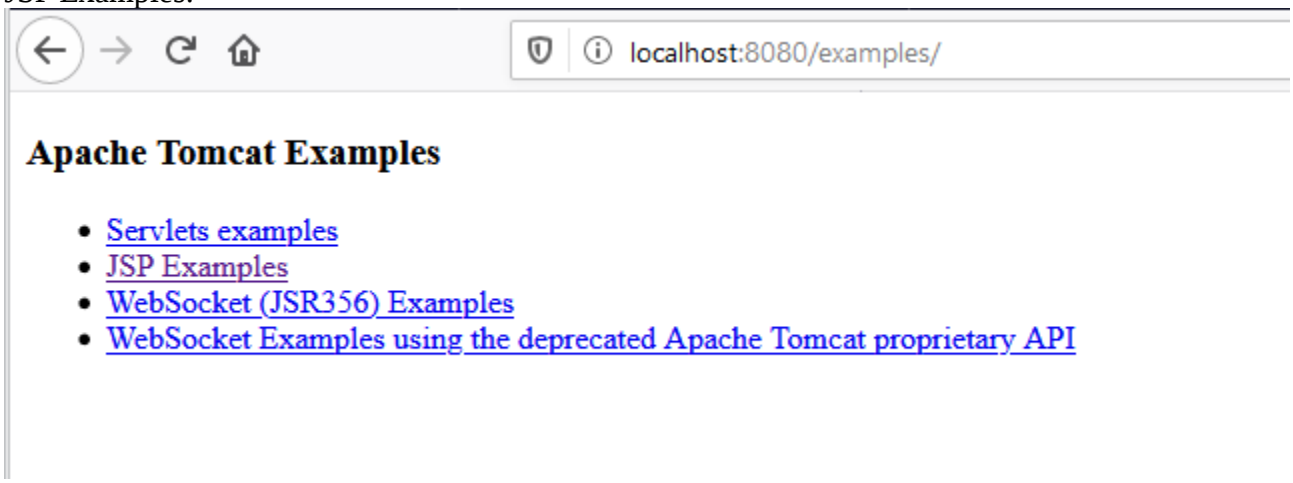
```
62
63
64      <!-- A "Connector" represents an endpoint by which requests are received
65           and responses are returned. Documentation at :
66           Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
67           Java AJP  Connector: /docs/config/ajp.html
68           APR (HTTP/AJP) Connector: /docs/apr.html
69           Define a non-SSL HTTP/1.1 Connector on port 8080
70      -->
71      <Connector port="8080" protocol="HTTP/1.1"
72                 connectionTimeout="20000"
73                 redirectPort="8443" />
74      <!-- A "Connector" using the shared thread pool-->
75      <!--
76      <Connector executor="tomcatThreadPool"
77                 port="8080" protocol="HTTP/1.1"
78                 connectionTimeout="20000"
79                 redirectPort="8443" />
80      -->
```

Po zmianie należy wyłączyć i ponownie uruchomić serwer.

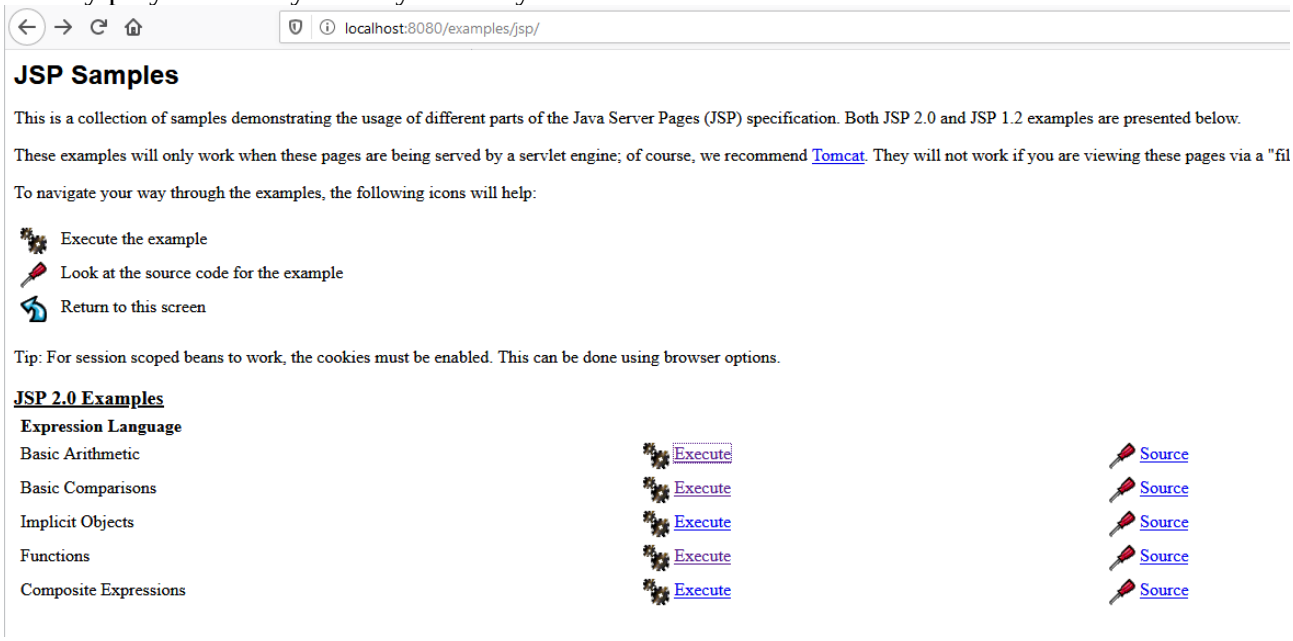
Oczywiście warto sprawdzić, czy nasz serwer pozwala na poprawne działanie JSP. W tym celu wybieramy przycisk Examples:



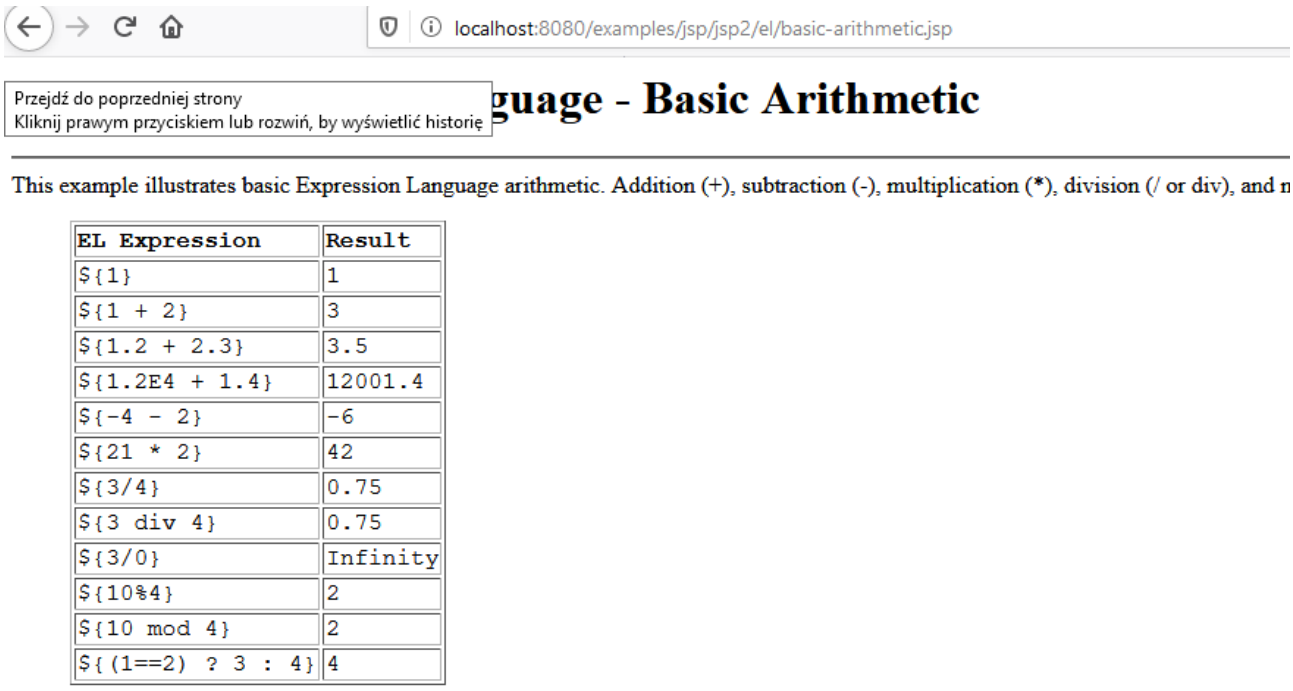
JSP Examples:



I z listy przykładów wybieramy dowolny z nich:



Przykład działania Basic Arithmetic (pierwszy przykład ze strony):



UWAGA! W przypadku systemu Linux instalacja sprowadza się przeważnie do zainstalowania pakietu OpenJDK oraz tomcat (np. zypper in tomcat w przypadku openSUSE, w innych wypadkach można pobrać spakowaną wersję i rozpakować ją najlepiej do katalogu /opt/tomcat). Szczegóły konfiguracji można znaleźć tutaj: https://www.tecmint.com/install-apache-tomcat-on-debian-10/. Ważne by w domyślnym katalogu ze stronami tomcat (przeważnie to /opt/tomcat/webapps) nie wrzucać swoich plików do głównego katalogu lecz utworzyć dowolny podkatalog (i jego wywoływać w przeglądarce – np. localhost/folder).

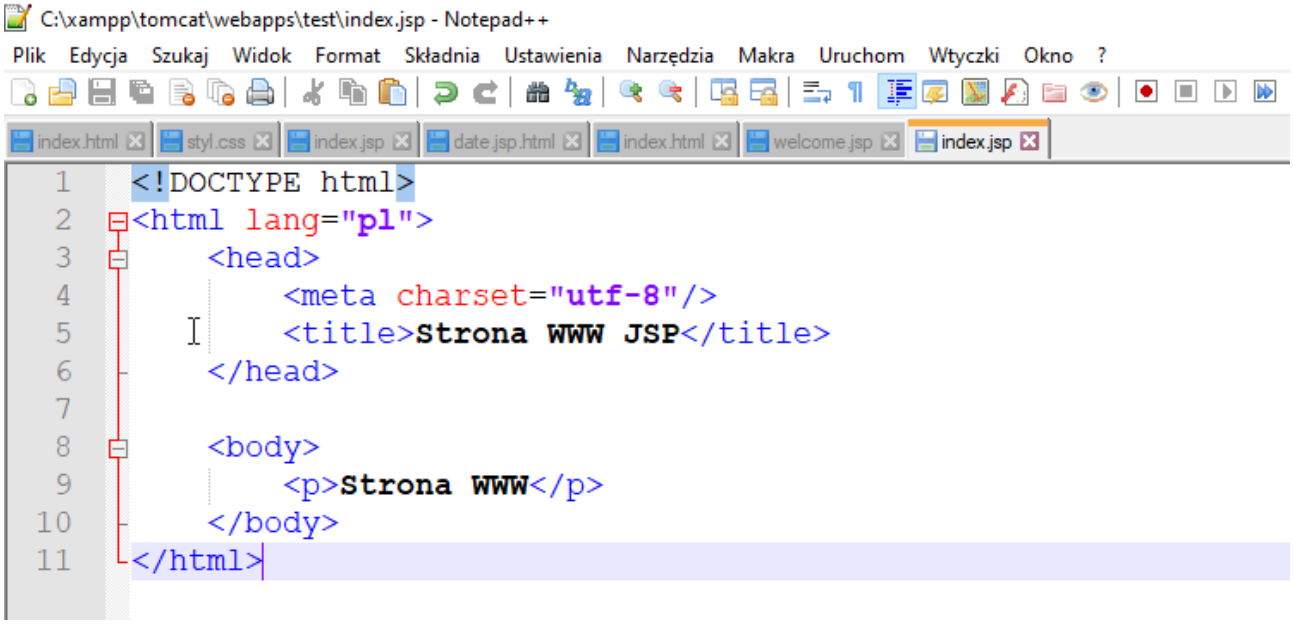
2. Tworzenie własnej, podstawowej strony WWW z JSP.

Wszystkie nasze strony będziemy umieszczać w katalogu tomcat\webapps. To tam znajduje się między innymi katalog examples (ten, z którego uruchomiony został poprzedni przykład).

Na początek utwórzmy prostą stronę WWW, do której włączymy jakąkolwiek wstawkę JSP. Mamy dwie możliwości tworzenia stron WWW: poprzez proste narzędzie (np. Notatnik, Notepad++) bądź poprzez poznanego już Eclipse.

Pierwszą strona zostanie utworzona w notatniku (w przykładzie to Notepad++). Najpierw należy utworzyć prosty szablon strony HTML:

```
<!DOCTYPE html>
<html lang="pl">
	<head>
		<meta charset="utf-8"/>
		<title>Strona WWW JSP</title>
	</head>

	<body>
		<p>Strona WWW</p>
	</body>
</html>
```

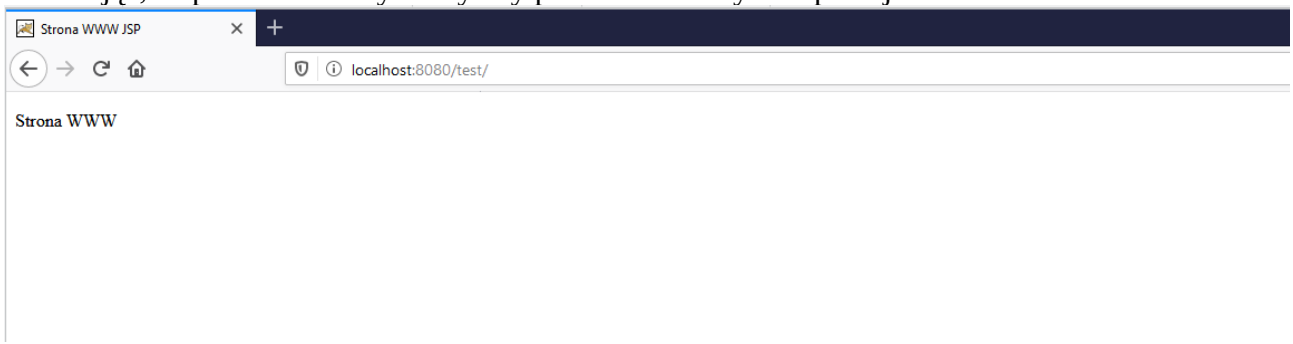


Należy zwrócić uwagę, że plik został zapisany nie z końcówką html a jsp. W ten sposób poinformujemy Tomcat, że plik będzie zawierał wstawki kodu Java. Dodatkowo należy mieć na uwadze, że plik MUSI zostać zapisany w folderze webapps (chyba, że zmienialiśmy opcje serwera dotyczące katalogu bazowego).

Żeby sprawdzić naszą stronę odpalamy przeglądarkę i wpisujemy :

localhost:8080/test

zakładając, że pozostawiliśmy domyślny port Tomcat. Wynik operacji:



Strona niekoniecznie imponująca, jednak potwierdzająca działanie serwera oraz naszego kodu HTML.

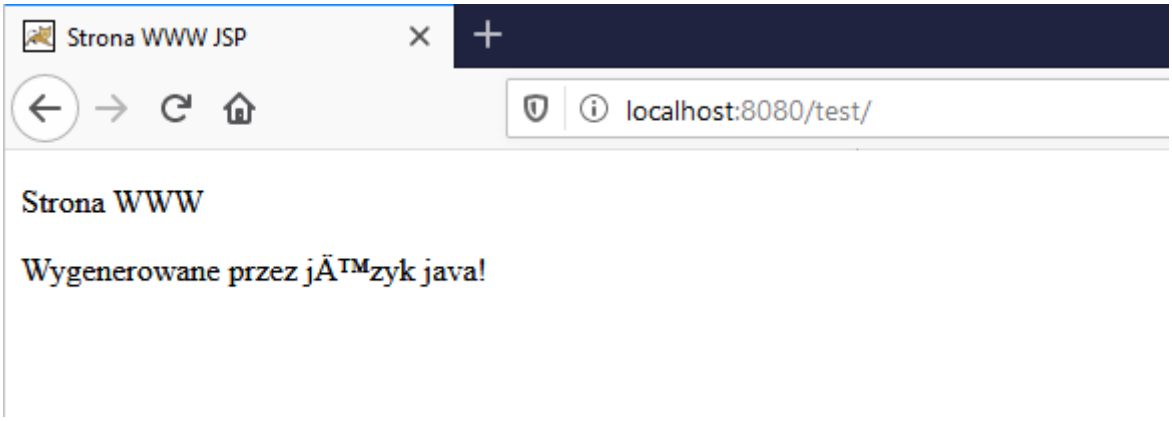
Teraz wstawimy fragment kodu JSP. Kod JSP wstawia się do pliku pomiędzy znaczniki <% %>. Chcąc wyświetlić poprzez kod Java napis „Wygenerowane przez język Java” musimy zrobić coś takiego:

```
<!DOCTYPE html>
<html lang="pl">
	<head>
		<meta charset="utf-8"/>
		<title>Strona WWW JSP</title>
	</head>

	<body>
		<p>Strona WWW</p>
```

```
            <%
                out.println("Wygenerowane przez język java!");
            %>
        </body>
</html>
```

Fragment z nowym kodem został zaznaczony na czerwono. Jak widać – wstawienie go nie jest trudne. Cechą charakterystyczną jest brak odwołania do klasy System; na stronach internetowych nie ma systemu operacyjnego (w sensie w oprogramowaniu serwera stron WWW) stąd brak odwołania do systemu.

Wynik działania kodu:



Niestety – polskie znaki zostały źle zinterpretowane. To dlatego, że domyślnie Tomcat działa w standardzie ASCII. By zmienić kodowanie znaków należy dodać następującą linię kodu:

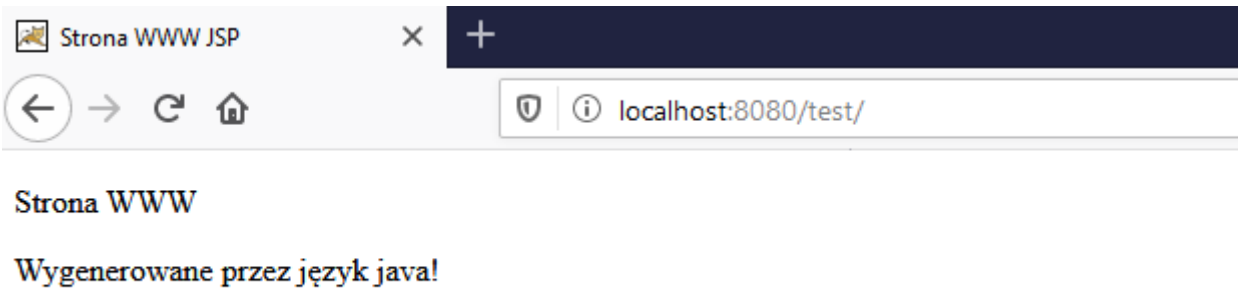
<%@ page contentType="text/html; charset=UTF-8" %>

Możliwie w najwcześniejszej (najwyższej) części strony:

index.html | styl.css | index.jsp | date.jsp.html | index.html | welcome.jsp | index.jsp

```
1  <%@ page contentType="text/html; charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html lang="pl">
4      <head>
5          <meta charset="utf-8"/>
6          <title>Strona WWW JSP</title>
7      </head>
8
9      <body>
10         <p>Strona WWW</p>
11         <%
12             out.println("Wygenerowane przez język java!");
13         %>
14     </body>
15 </html>
```

Efekt będzie taki jak pożądamy:



Zamiast znaczników <% %> można używać również notacji XML:

```
<jsp:scriptlet>
        out.println("Wygenerowane przez język java!");
</jsp:scriptlet>
```

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>

    <body>
        <p>Strona WWW</p>
        <jsp:scriptlet>
            out.println("Wygenerowane przez język java!");
        </jsp:scriptlet>
    </body>
</html>
```

Na tej podstawie możemy pisać dowolny kod Java bezpośrednio w pliku JSP. Jeżeli potrzebowalibyśmy zaimportować jakikolwiek element (np. List), można zrobić to poprzez:

<%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>

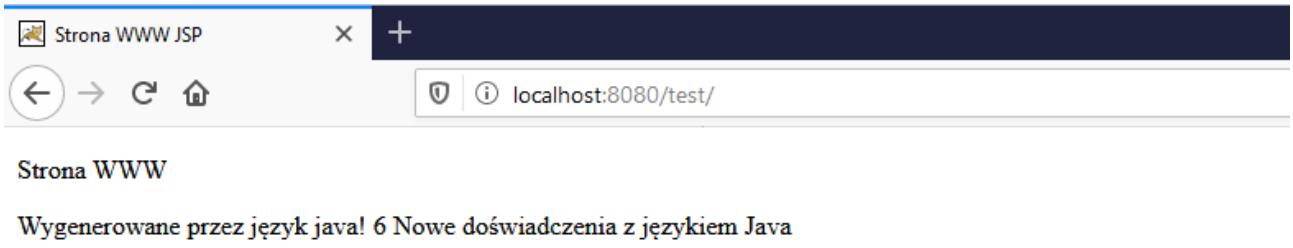
Powyższy kod zaimportuje nam wszystko z paczki java.io, interfejs List oraz klasę ArrayList z java.util. Odpowiednikiem kodu w Java byłoby:

```
import java.io.*;
import java.util.List;
import java.util.ArrayList;
```

Kod powinien znajdować się przed faktycznym wywołaniem naszego kodu JSP, np. jak na zrzucie ekranu:

```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <% int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <%
            List<String> lista = new ArrayList<String>();
            lista.add("Nowe");
            lista.add("doświadczenia");
            lista.add("z");
            lista.add("językiem");
            lista.add("Java");
            out.println("Wygenerowane przez język java! " + i);
            for (String s: lista)
                out.print(s + " ");
        %>
    </body>
</html>
```

Wynik tak napisanej strony będzie następujący:



Dlaczego napis z foreach nie jest poniżej poprzedniej linii? To już kwestia działania samego HTML, który domyślnie nie czyta spacji ani znaków nowej linii. Wygenerowany kod jest bowiem poprawny:

Jak łatwo zauważyć, sam kod HTML generuje się prawidłowo (napisy są odpowiednio przeniesione). Przy okazji jak widać nasz kod JSP NIE JEST przesyłany do klienta (do przeglądarki). Ma to znaczenie w późniejszym pisaniu kodu np. do łączenia się z bazą danych.

JSP pozwala także na używanie deklaracji. Zamiast deklarowania zmiennej w standardowym fragmencie kodu (tak jak na powyższym przykładzie zmienna i) można dokonać poprzez następujący znacznik:

<%! int i = 6; %>

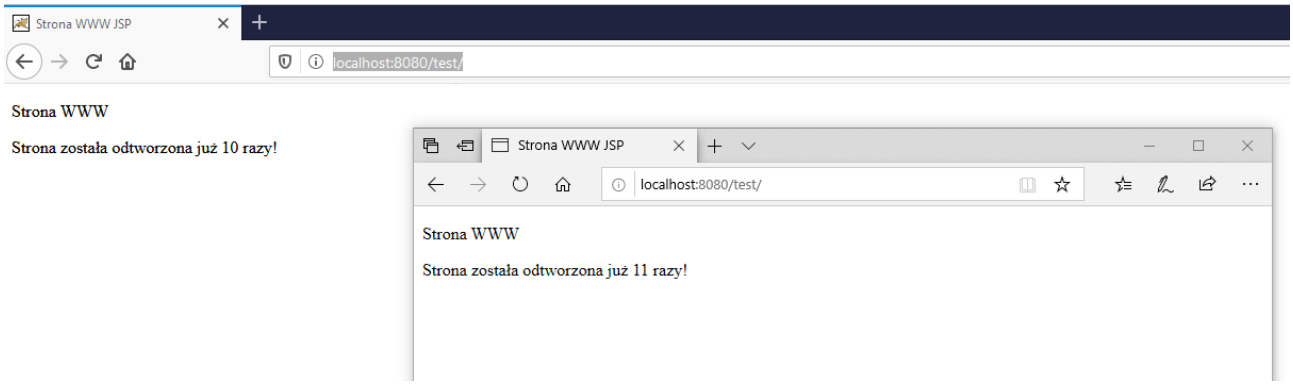
Deklarowanie charakteryzuje się dodatkowym wykrzyknikiem po otworzeniu kodu JSP. Nie zmienia to oczywiście samego działania kodu:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <%! int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <%
            List<String> lista = new ArrayList<String>();
            lista.add("Nowe");
            lista.add("doświadczenia");
            lista.add("z");
            lista.add("językiem");
            lista.add("Java");
            out.println("Wygenerowane przez język java! " + i);
            for (String s: lista)
                out.print(s + " ");

        %>
    </body>
</html>
```

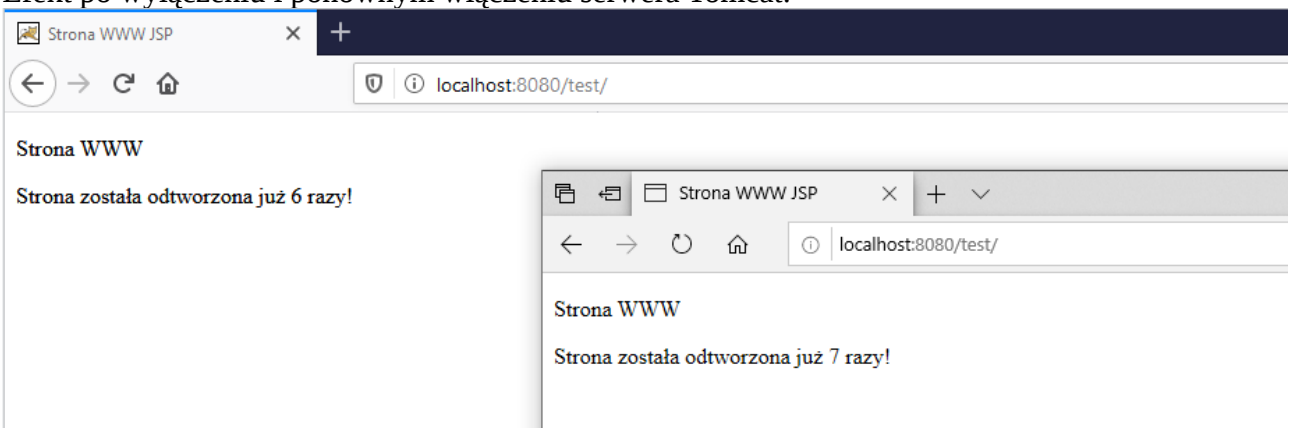
Bardziej charakterystyczne są przypisania wartości. Możemy je wykonywać w dowolnym fragmencie strony, w którym chcemy wstawić określoną wartość jednak nie chcemy tworzyć zmiennej w języku Java. Przykład:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <%! static int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <p>Strona została odtworzona już <%= i++ %> razy!</p>
    </body>
</html>
```

Proszę zauważyć, że zmienna i dostała dodatkowy modyfikator static. Oznacza to, że póki serwer będzie działał nasz licznik będzie liczył kolejne odsłonięcie strony WWW (niezależnie od otwartej przeglądarki!). Oczywiście nie jest to poprawne rozwiązanie licznika otwarcia jednak obrazuje pewne mechanizmy działania serwera stron Java:



Efekt po wyłączeniu i ponownym włączeniu serwera Tomcat:



Jak widać „licznik” został zrestartowany (i ponownie liczy od 6).

JSP posiada także system komentarzy:

<%-- TO jest komentarz w JSP. On nie pojawi się na stronie --%>

Tak wygląda w kodzie:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <%! static int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <%-- TO jest komentarz w JSP. On nie pojawi się na stronie --%>
        <p>Strona została odtworzona już <%= i++ %> razy!</p>
    </body>
</html>
```

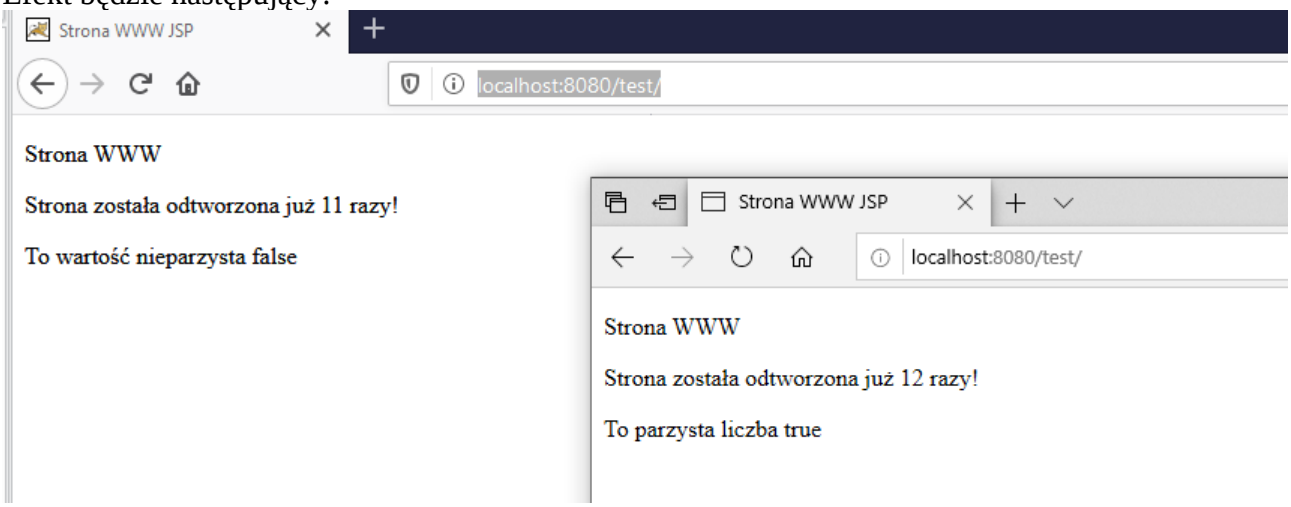
Dodatkowo można rozbić wstawki kodowe JSP naprzemiennie z HTML. Jeżeli np. Chcemy wyświetlać tekst czy obecnie mamy parzystą/nieparzystą wizytę na stronie możemy zrobić to tak:

```
<% boolean parity = ((i % 2 == 0) ? true : false); %>
                <p>Strona została odtworzona już <%= i++ %> razy!</p>
                <% if (parity==true) { %>
                <p>To parzysta liczba <%= parity %></p>
                <% } else { %>
                <p>To wartość nieparzysta <%= parity %></p>
                <% } %>
```

W kodzie ten fragment będzie wyglądało następująco:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <%! static int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <%-- TO jest komentarz w JSP. On nie pojawi się na stronie --%>
        <% boolean parity = ((i % 2 == 0) ? true : false); %>
        <p>Strona została odtworzona już <%= i++ %> razy!</p>
        <% if (parity==true) { %>
        <p>To parzysta liczba <%= parity %></p>
        <% } else { %>
        <p>To wartość nieparzysta <%= parity %></p>
        <% } %>
    </body>
</html>
```

Efekt będzie następujący:



Tak jak inne języki, tak i tutaj możliwe jest dołączanie kodu innych plików JSP. Robi się to przez dyrektywę include:

<%@include file="metoda.jsp"%>

Dzięki takiemu podłączeniu możemy np. ładować metody definiowane w dołączanym pliku:

Plik index.jsp:

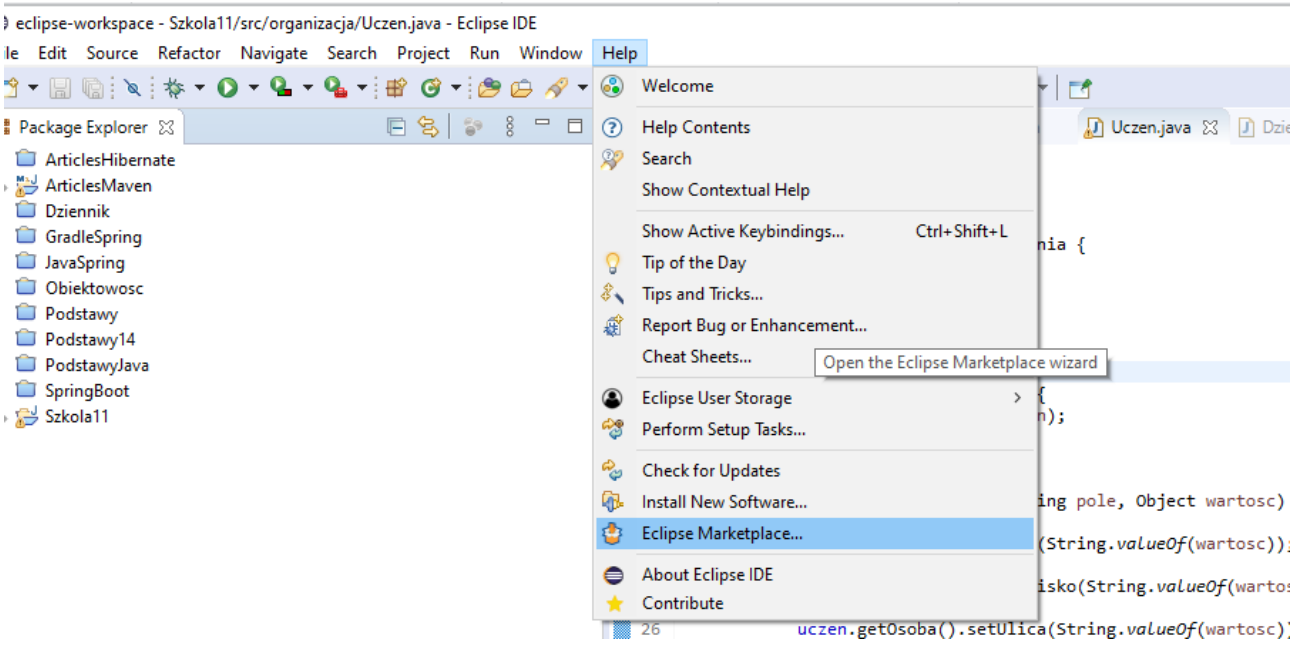
```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<!DOCTYPE html>
<html lang="pl">
    <%@ page import = "java.io.*,java.util.List,java.util.ArrayList" %>
    <head>
        <meta charset="utf-8"/>
        <title>Strona WWW JSP</title>
    </head>
    <%! static int i = 6; %>
    <body>
        <p>Strona WWW</p>
        <%-- TO jest komentarz w JSP. On nie pojawi się na stronie --%>
        <% boolean parity = ((i % 2 == 0) ? true : false); %>
        <p>Strona została odtworzona już <%= i++ %> razy!</p>
        <%@include file="metoda.jsp"%>
        <%= showParity(parity) %>
    </body>
</html>
```

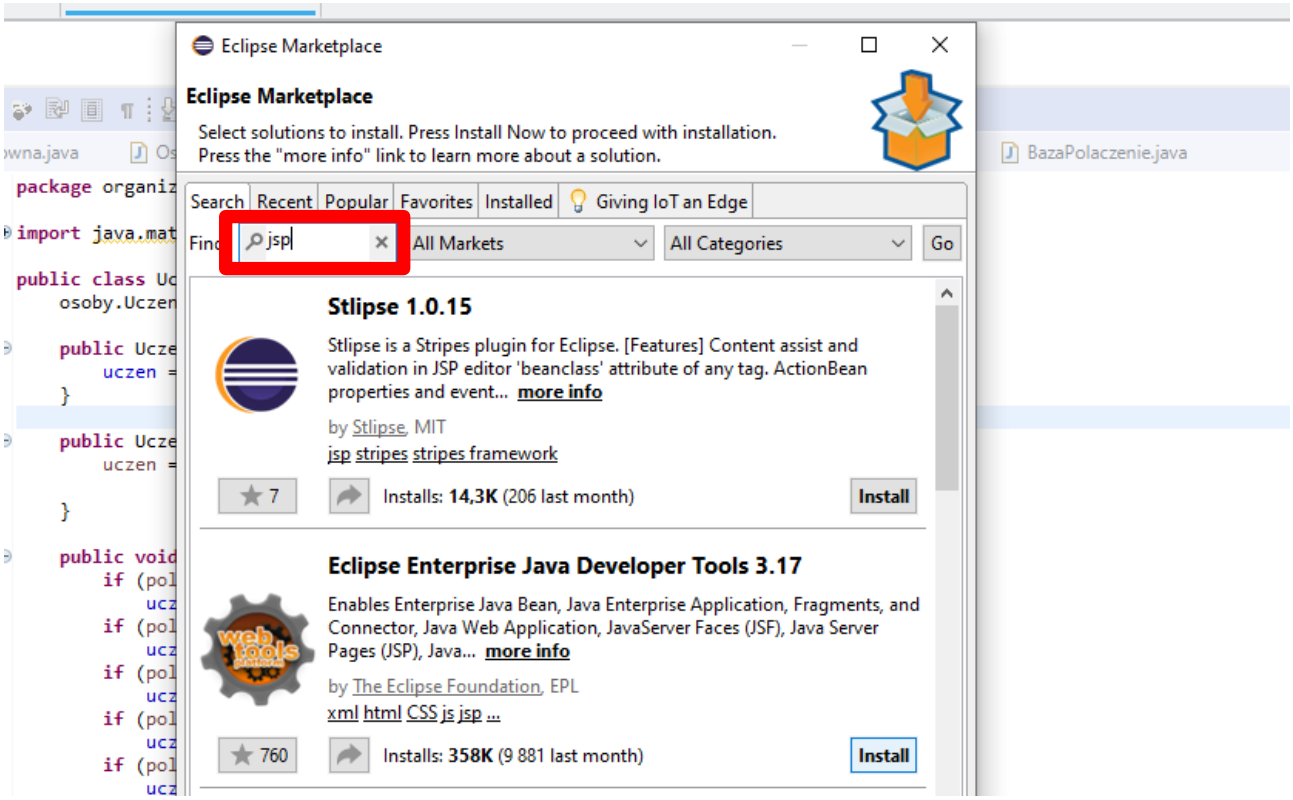
```jsp
<%@ page contentType="text/html; charset=UTF-8" %>
<%!
    String showParity(boolean state) {
        return "<p>Funkcja: Wartość jest " + ((state) ? "parzysta" : " nieparzysta") + "</p>";
    }
    %>
```

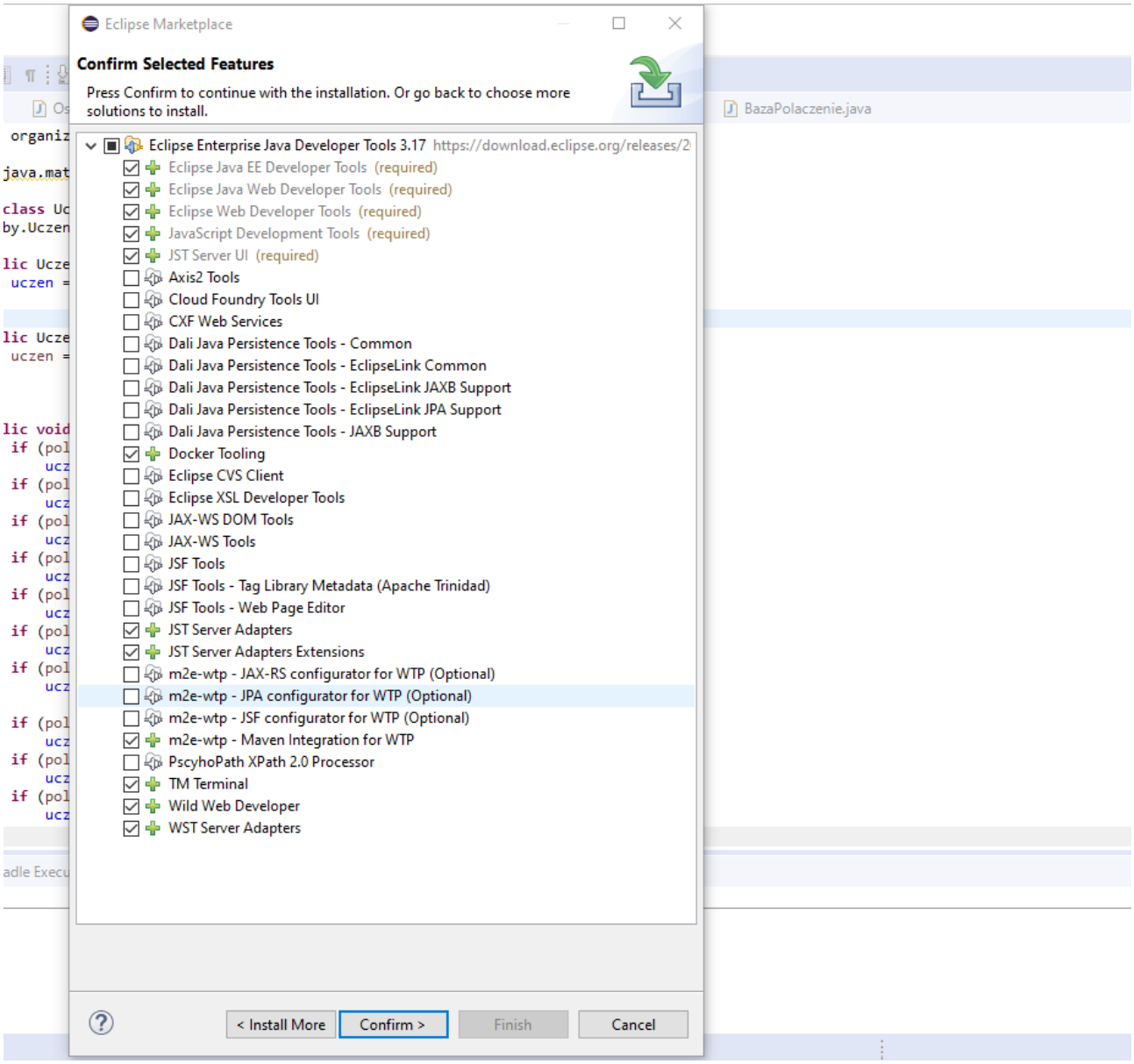
3. Tworzenie projektu przez Eclipse.

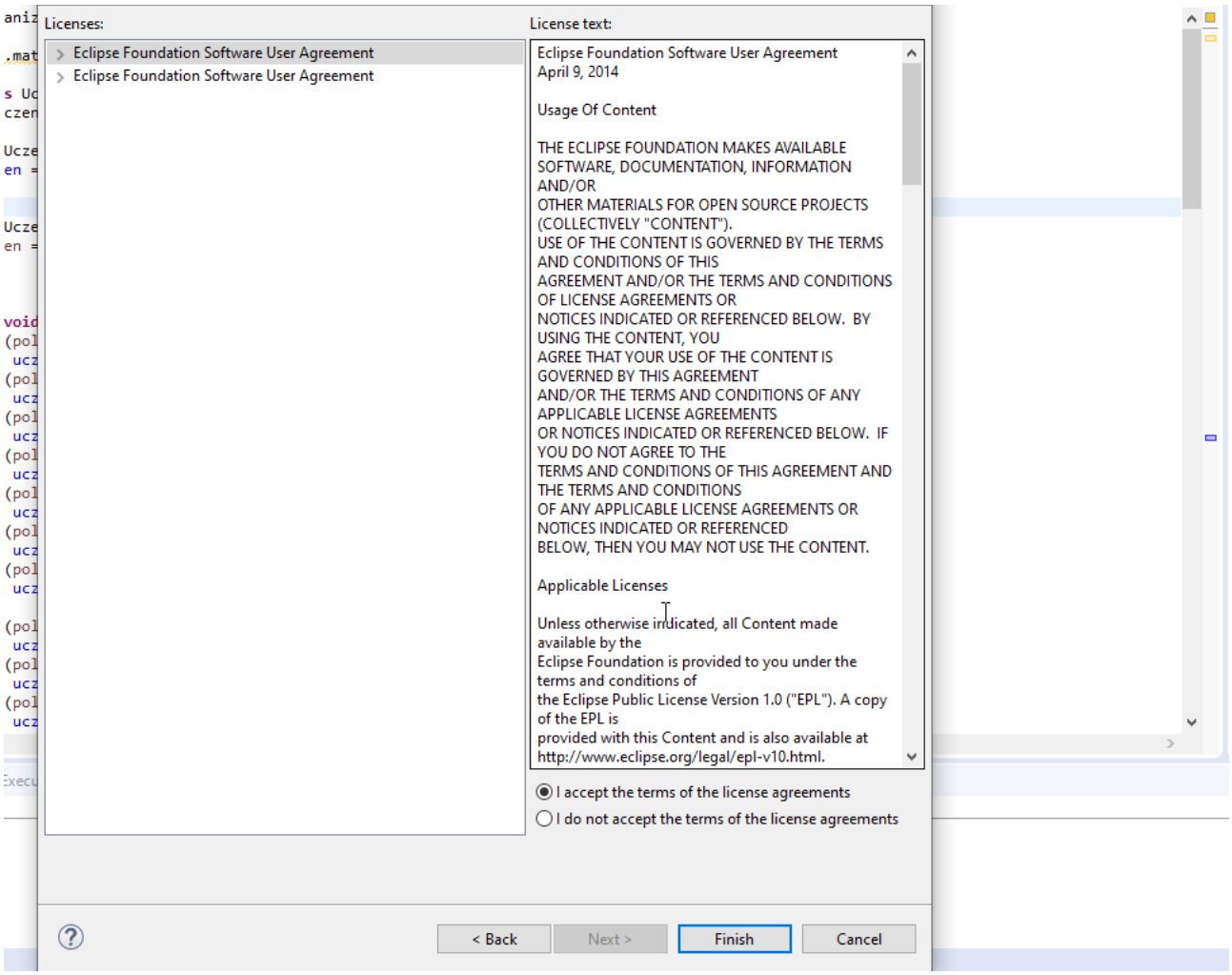
Jak zostało wspomniane, naszą stronę możemy tworzyć także poprzez dobrze znane środowisko Eclipse. W tym celu należy:

- pobrać odpowiednią wersję Eclipse lub doinstalować odpowiednią wtyczkę poprzez Marketplace:



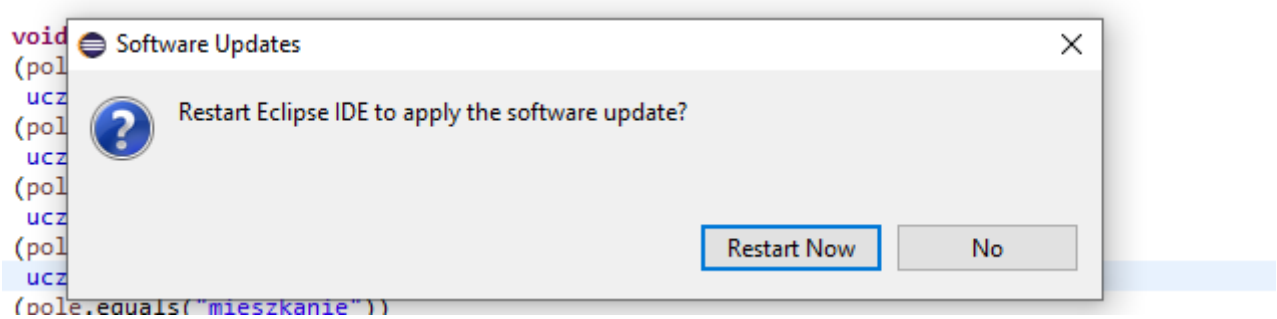

Eclipse Marketplace
Eclipse Marketplace
Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.
Search
Recent
Popular
Favorites
Installed
Giving IoT an Edge
Find
jsp
All Markets
All Categories
Go
Stlipse 1.0.15
Stlipse is a Stripes plugin for Eclipse. [Features] Content assist and validation in JSP editor 'beanclass' attribute of any tag. ActionBean properties and event... more info
by Stlipse, MIT
jsp stripes stripes framework
7
Installs: 14,3K (206 last month)
Install
Eclipse Enterprise Java Developer Tools 3.17
Enables Enterprise Java Bean, Java Enterprise Application, Fragments, and Connector, Java Web Application, JavaServer Faces (JSF), Java Server Pages (JSP), Java... more info
by The Eclipse Foundation, EPL
xml html CSS js jsp ...
760
Installs: 358K (9 881 last month)
Install
BazaPolaczenie.java


Eclipse Marketplace
Confirm Selected Features
Press Confirm to continue with the installation. Or go back to choose more solutions to install.
Eclipse Enterprise Java Developer Tools 3.17 https://download.eclipse.org/releases/2
Eclipse Java EE Developer Tools (required)
Eclipse Java Web Developer Tools (required)
Eclipse Web Developer Tools (required)
JavaScript Development Tools (required)
JST Server UI (required)
Axis2 Tools
Cloud Foundry Tools UI
CXF Web Services
Dali Java Persistence Tools - Common
Dali Java Persistence Tools - EclipseLink Common
Dali Java Persistence Tools - EclipseLink JAXB Support
Dali Java Persistence Tools - EclipseLink JPA Support
Dali Java Persistence Tools - JAXB Support
Docker Tooling
Eclipse CVS Client
Eclipse XSL Developer Tools
JAX-WS DOM Tools
JAX-WS Tools
JSF Tools
JSF Tools - Tag Library Metadata (Apache Trinidad)
JSF Tools - Web Page Editor
JST Server Adapters
JST Server Adapters Extensions
m2e-wtp - JAX-RS configurator for WTP (Optional)
m2e-wtp - JPA configurator for WTP (Optional)
m2e-wtp - JSF configurator for WTP (Optional)
m2e-wtp - Maven Integration for WTP
PscyhoPath XPath 2.0 Processor
TM Terminal
Wild Web Developer
WST Server Adapters
< Install More
Confirm >
Finish
Cancel

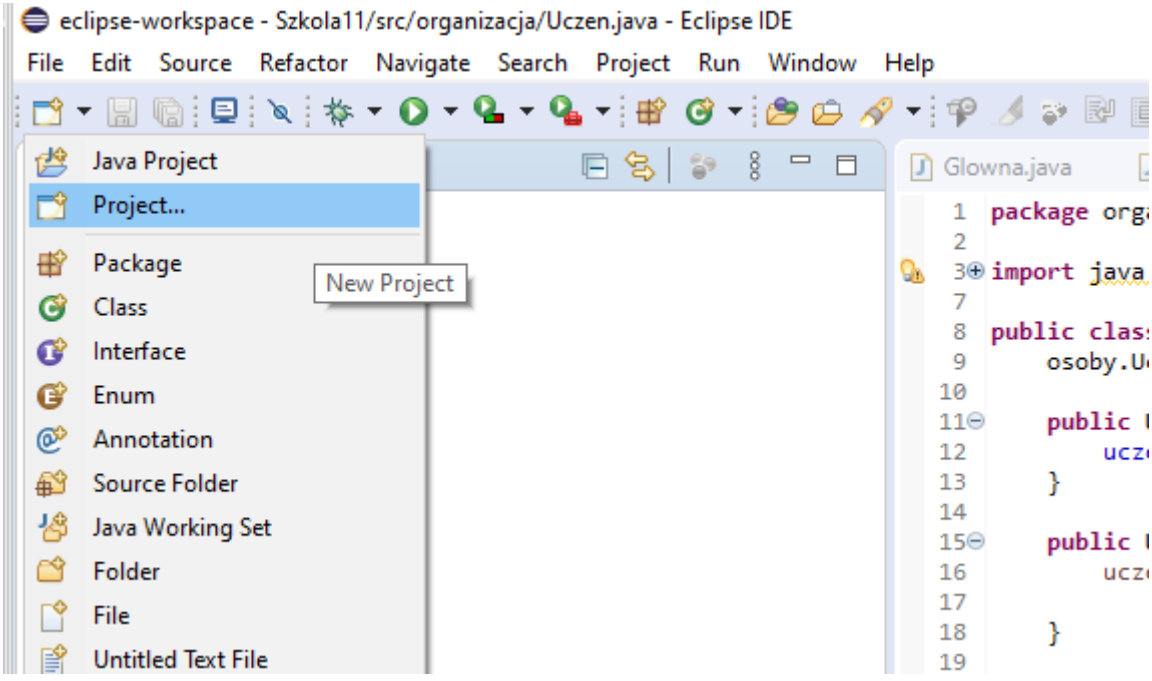

Licenses:
Eclipse Foundation Software User Agreement
Eclipse Foundation Software User Agreement
License text:
Eclipse Foundation Software User Agreement
April 9, 2014
Usage Of Content
THE ECLIPSE FOUNDATION MAKES AVAILABLE SOFTWARE, DOCUMENTATION, INFORMATION AND/OR
OTHER MATERIALS FOR OPEN SOURCE PROJECTS (COLLECTIVELY "CONTENT").
USE OF THE CONTENT IS GOVERNED BY THE TERMS AND CONDITIONS OF THIS
AGREEMENT AND/OR THE TERMS AND CONDITIONS OF LICENSE AGREEMENTS OR
NOTICES INDICATED OR REFERENCED BELOW. BY USING THE CONTENT, YOU
AGREE THAT YOUR USE OF THE CONTENT IS GOVERNED BY THIS AGREEMENT
AND/OR THE TERMS AND CONDITIONS OF ANY APPLICABLE LICENSE AGREEMENTS
OR NOTICES INDICATED OR REFERENCED BELOW. IF YOU DO NOT AGREE TO THE
TERMS AND CONDITIONS OF THIS AGREEMENT AND THE TERMS AND CONDITIONS
OF ANY APPLICABLE LICENSE AGREEMENTS OR NOTICES INDICATED OR REFERENCED
BELOW, THEN YOU MAY NOT USE THE CONTENT.
Applicable Licenses
Unless otherwise indicated, all Content made available by the
Eclipse Foundation is provided to you under the terms and conditions of
the Eclipse Public License Version 1.0 ("EPL"). A copy of the EPL is
provided with this Content and is also available at http://www.eclipse.org/legal/epl-v10.html.
I accept the terms of the license agreements
I do not accept the terms of the license agreements
< Back
Next >
Finish
Cancel

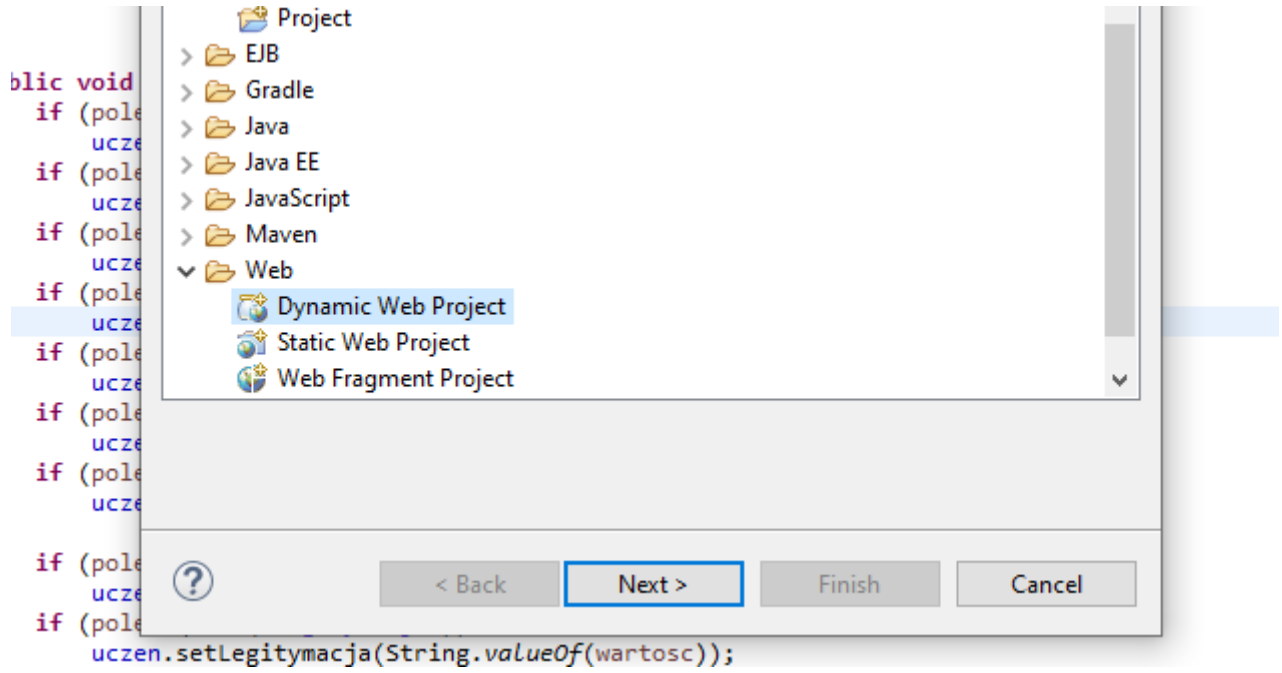
Czekamy aż proces się zakończy:

Kończymy restartując Eclipse:

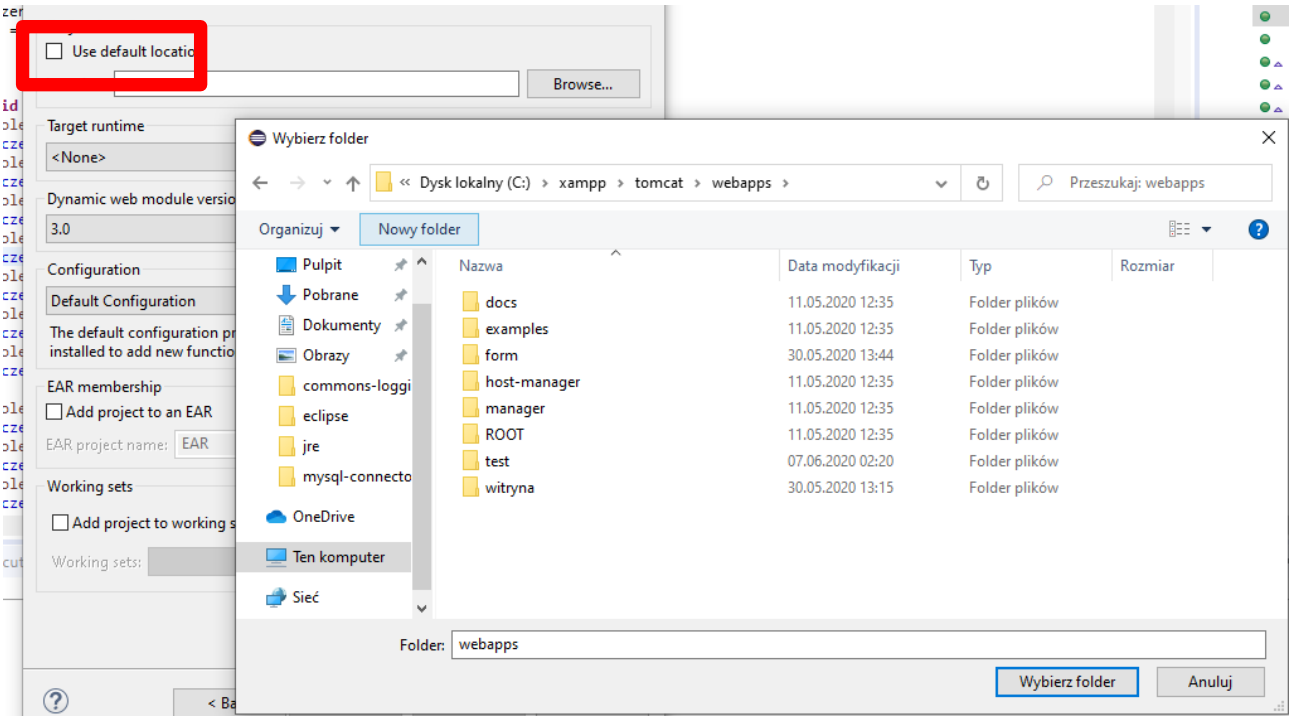


Mając już zainstalowane rozszerzenia możemy utworzyć projekt Web:

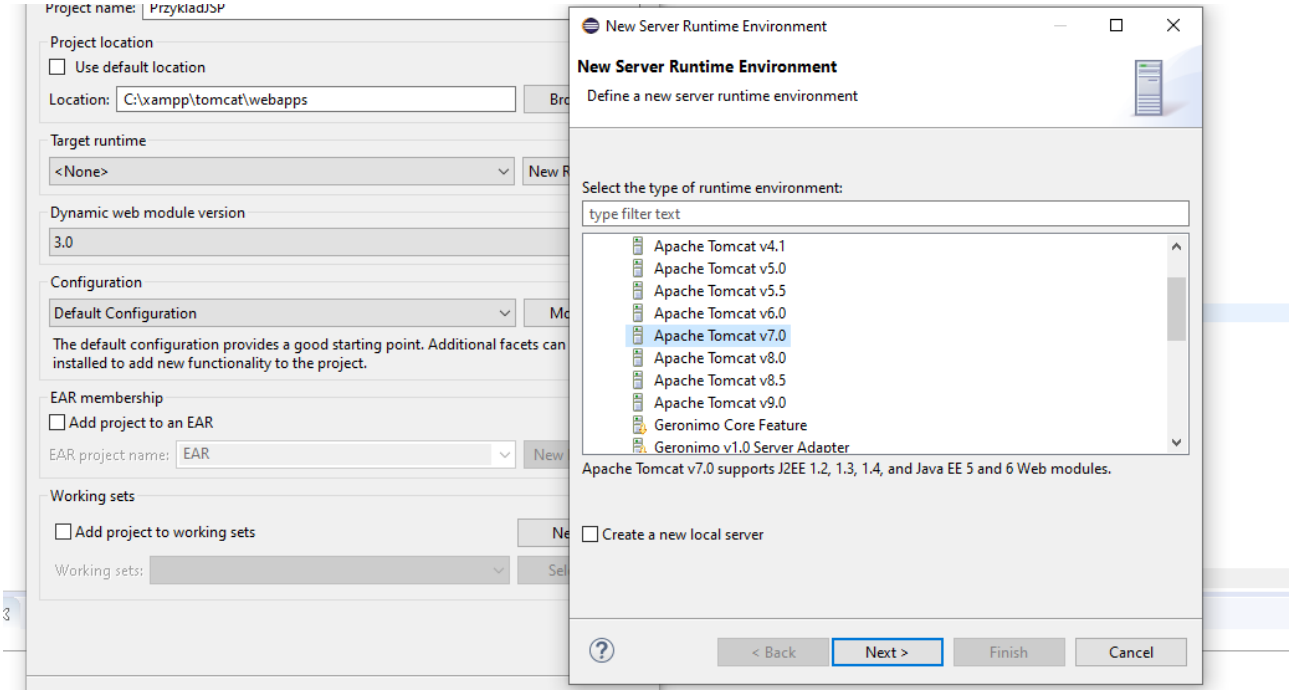




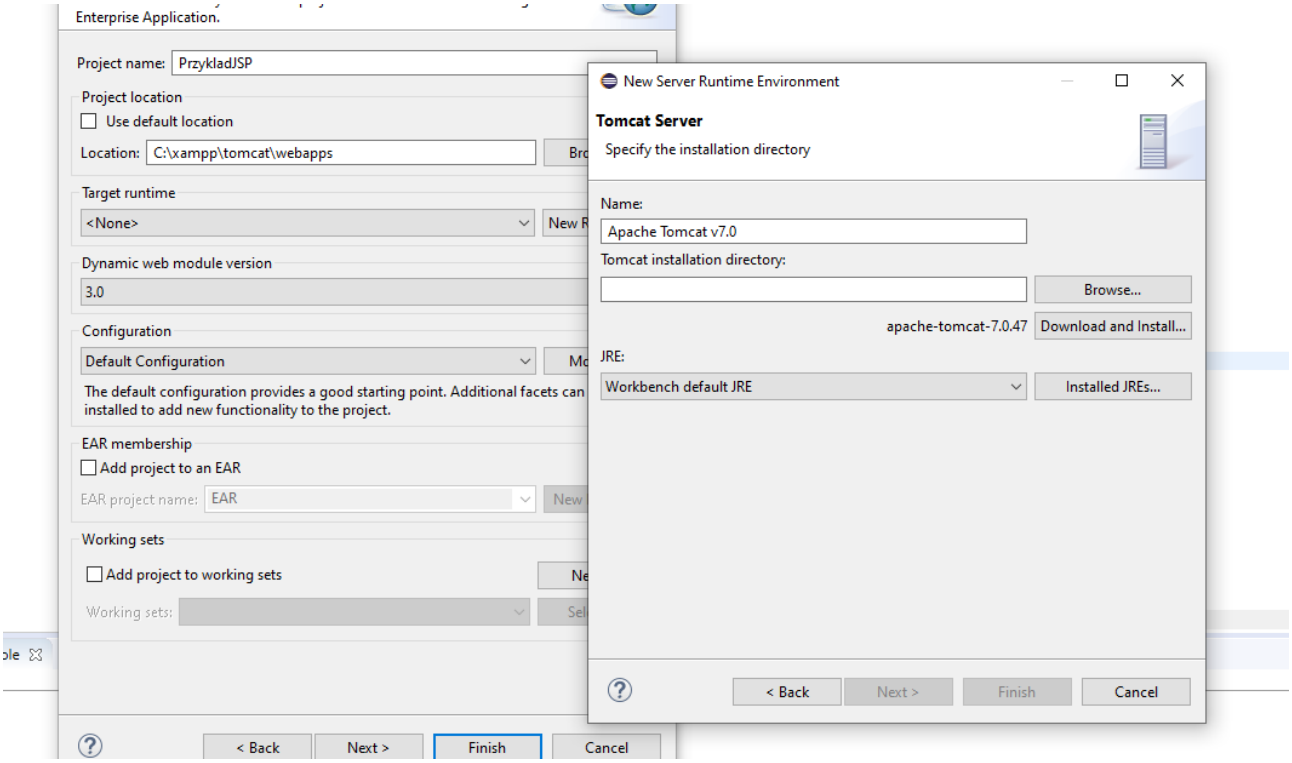
Tworząc projekt możemy zmienić lokalizację na nasz serwer:

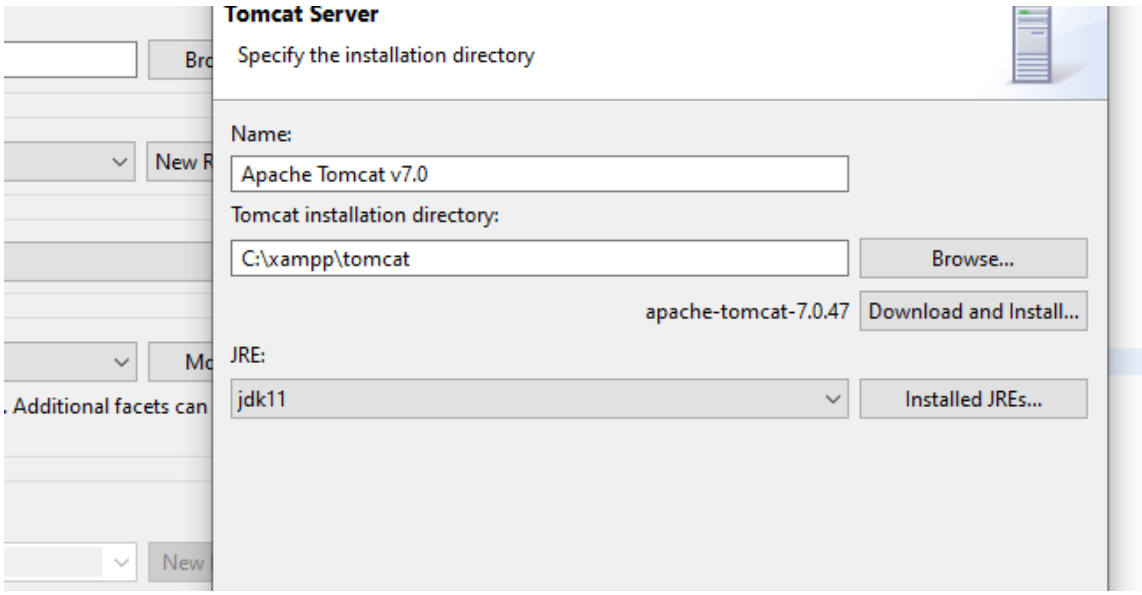


Możemy również wskazać wersję Tomcat, na której będziemy pracować. XAMPP domyślnie posiada wersję 7 (w Linux będzie to najnowsza, dziewiąta, aczkolwiek można wskazać wcześniejsze wersje).

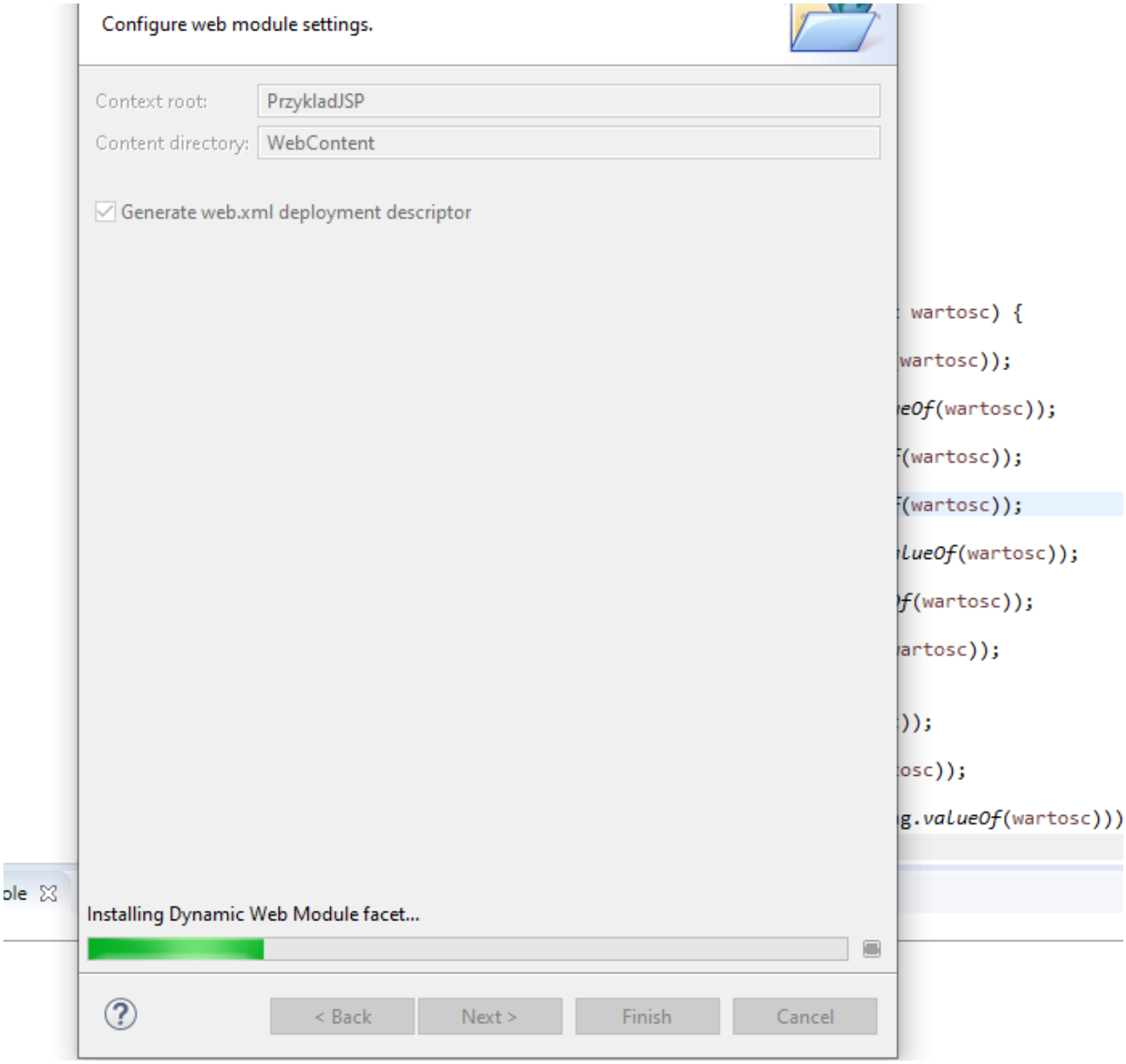


Gdybyśmy nie mieli serwera – możemy go sobie pobrać i skonfigurować specjalnie pod nasz projekt:





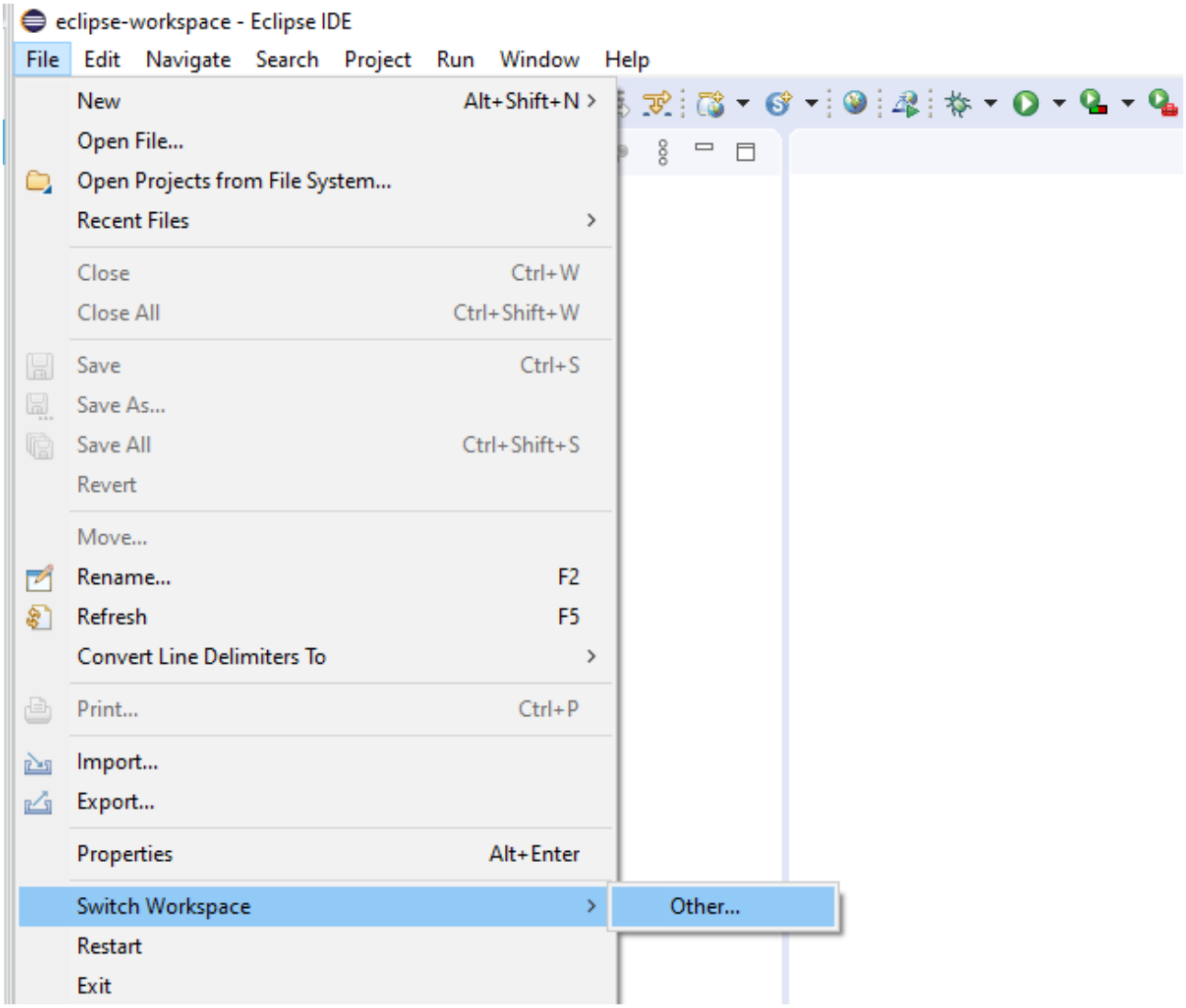
Kończymy tworzenie projektu:

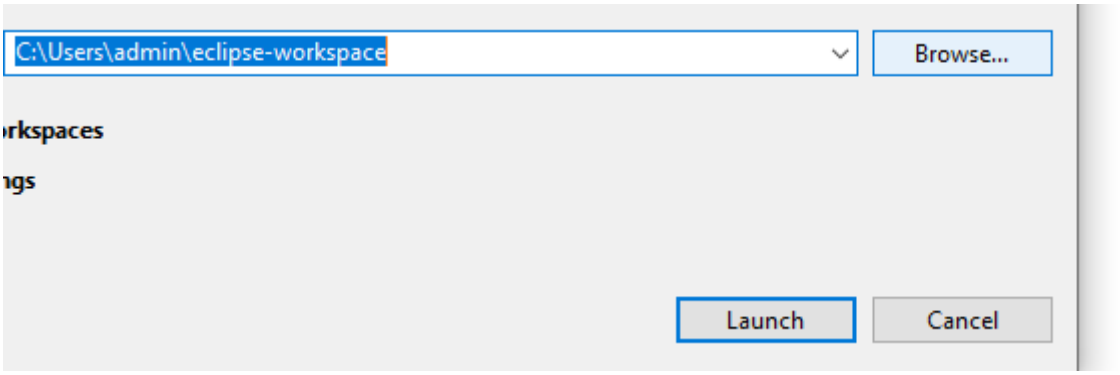


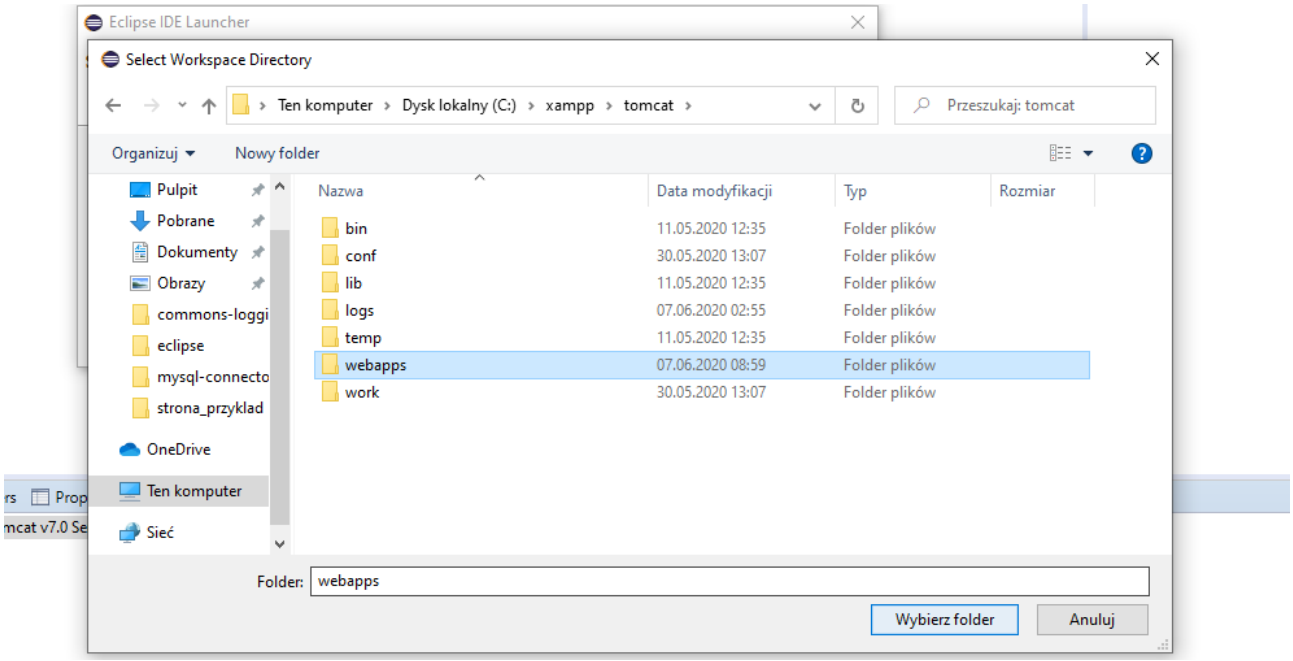
Od teraz możemy tworzyć stronę i odpalać ją na naszym serwerze Tomcat.

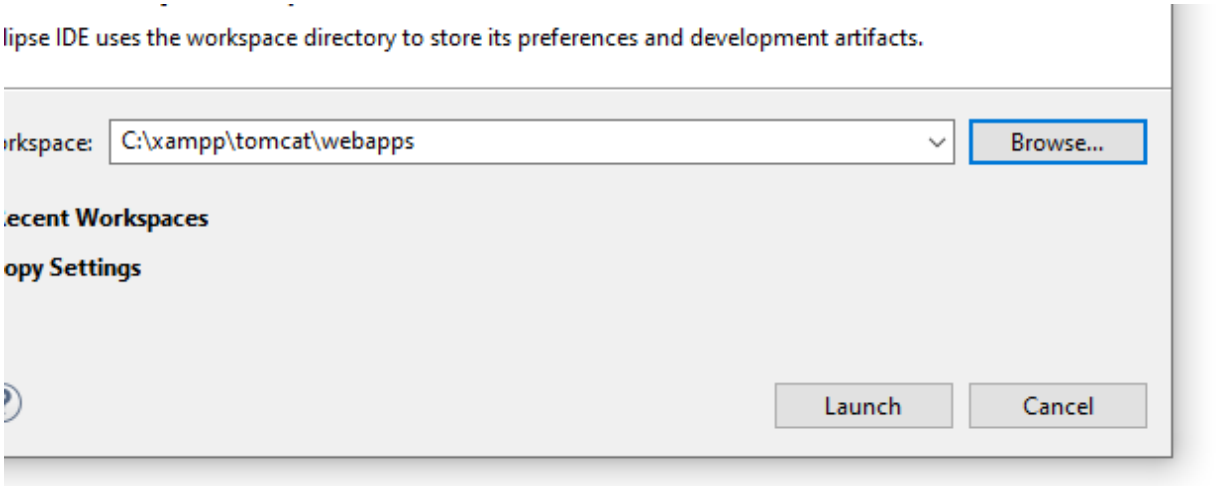
UWAGA! Jak widać z powyższych zrzutów Eclipse oferuje teraz możliwość tworzenia dedykowanego serwera dla JSP. Jednak to rozwiązanie może niekoniecznie działać „stale” gdy np. wyłączymy Eclipse. Dlatego, jeżeli chcemy naszą stronę usprawniać także po zamknięciu Eclipse (albo po prostu ją otworzyć w przeglądarce) Tomcat może być nadal najlepszym sposobem na serwer WWW.

INFORMACJA: Jeżeli zdecydujemy się na własny serwer Tomcat dobrym rozwiązaniem jest także zmienienie naszej przestrzeni roboczej na główny katalog aplikacji webapps. Dzięki temu nasza strona zawsze będzie umieszczana niejako automatycznie na serwerze. W tym celu możemy wykonać następującą operację (zakładając, że mamy już włączony Eclipse lub podczas startu wybraliśmy opcję nie wybierania przestrzeni roboczej):




C:\Users\admin\eclipse-workspace
Browse...
rkspaces
ngs
Launch
Cancel

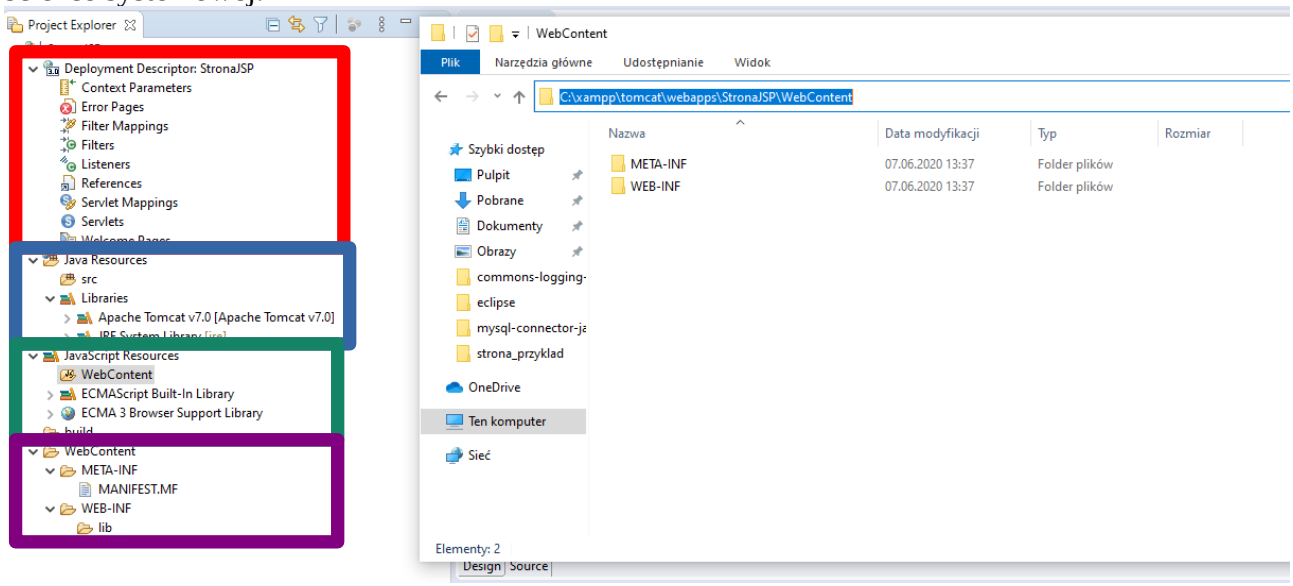

Eclipse IDE Launcher
Select Workspace Directory
Ten komputer › Dysk lokalny (C:) › xampp › tomcat ›
Przeszukaj: tomcat
Organizuj
Nowy folder
Pulpit
Pobrane
Dokumenty
Obrazy
commons-loggi
eclipse
mysql-connecto
strona_przyklad
OneDrive
Ten komputer
Sieć
Nazwa
Data modyfikacji
Typ
Rozmiar
bin 11.05.2020 12:35 Folder plików
conf 30.05.2020 13:07 Folder plików
lib 11.05.2020 12:35 Folder plików
logs 07.06.2020 02:55 Folder plików
temp 11.05.2020 12:35 Folder plików
webapps 07.06.2020 08:59 Folder plików
work 30.05.2020 13:07 Folder plików
Folder: webapps
Wybierz folder
Anuluj



Oczywiście po kliknięciu Launch Eclipse przywita nas w nowej przestrzeni roboczej. Od teraz będziemy mogli przełączać się pomiędzy tymi przestrzeniami w ten sam sposób, który został opisany powyżej.

4. Praca nad projektem strony w Eclipse.

Mając już utworzony projekt JSP w Eclipse możemy przystąpić do tworzenia strony WWW. Najpierw jednak warto zapoznać się z układem projektu oraz odwzorowaniem jego katalogów w ścieżce systemowej:



Jak widać, wynika z tego, że by uruchomić stronę poprzez dowolną przeglądarkę internetową trzeba będzie wprowadzić adres:

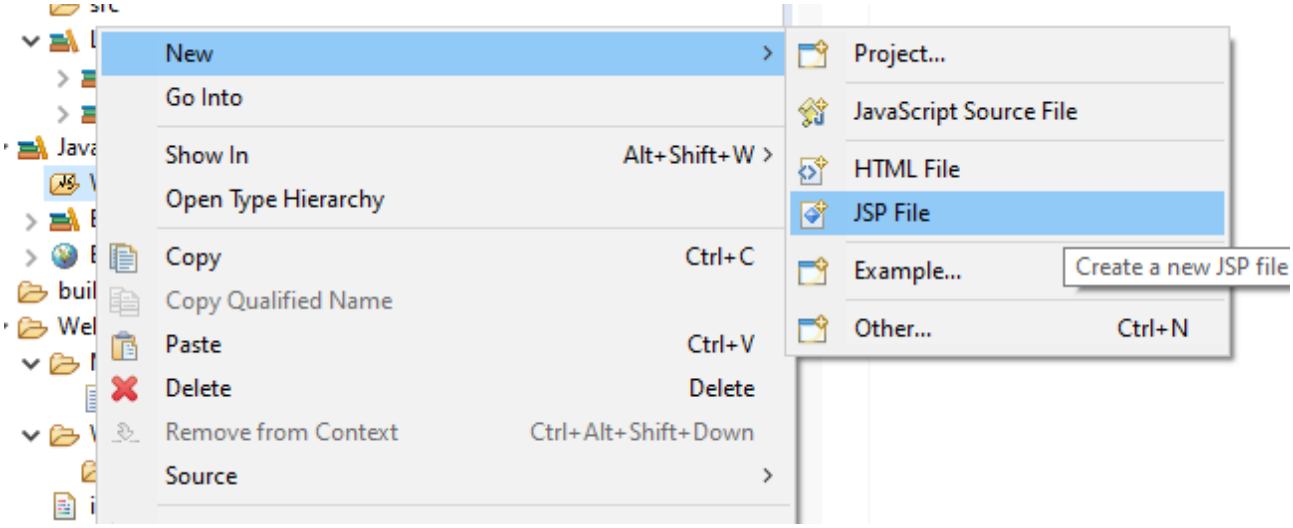
localhost:8080/StronaJSP/WebContent

Serwer Tomcat jest wrażliwy na małe i duże litery, przez co adres serwisu musi być podany tak jak zostało to opisane powyżej.

Kolejno w ramkach zostały oznaczone następujące elementy projektu:

- czerwona (pierwsza) – zawiera informacje o ustawieniach naszego projektu, jak chociażby domyślne strony błędów, strony startowe, dodatkowe moduły itp. Dane te są zbierane na podstawie pliku web.xml (projekt ze zrzutu powyżej go nie posiada)
- niebieska (druga) – może zawierać kod klas w języku Java, które będą miały być załączone do strony JSP. Możemy je załączać np. jako pluginy (wtyczki) lub jako aplety (wstawiane na s tronę, aczkolwiek to rozwiązanie od dłuższego czasu jest stygmatyzowane jako niebezpieczne dla użytkownika docelowego).
- zielona (trzecia) – pozwala zobaczyć kod naszej strony WWW (tej tworzonej przez nas). W WebContent będziemy mieli wszystkie pliki html, jsp, css czy js. Będą też widoczne katalogi strony WWW. W zasadzie folder WebContent jest tutaj aliasem do prawdziwego folderu WebContent…
- filoletowa (ostatnia) - ...oznaczonego właśnie tą ramką. Prócz tego mamy tutaj dostęp do folderów zawierających pliki konfiguracyjne naszego projektu (WEB-INF pozwalającego na konfigurację JSP poprzez pliki xml).

Spróbujmy stworzyć nową stronę – dodajmy plik index.jsp:

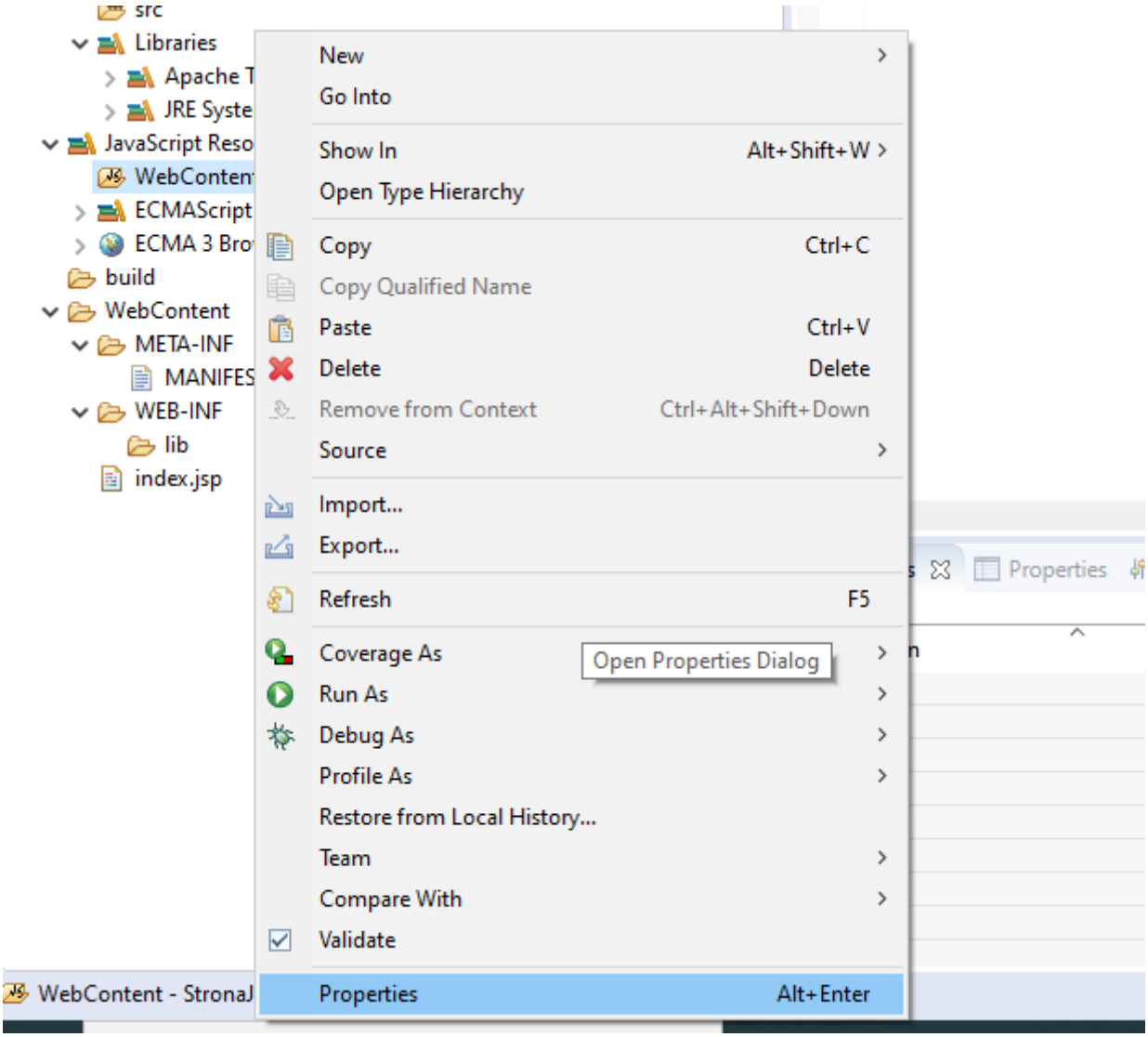


Jak widać zasada dodawania nowych plików zasadniczo się nie zmienia. Po utworzeniu pliku otrzymamy już wstępnie wygenerowany kod strony WWW:

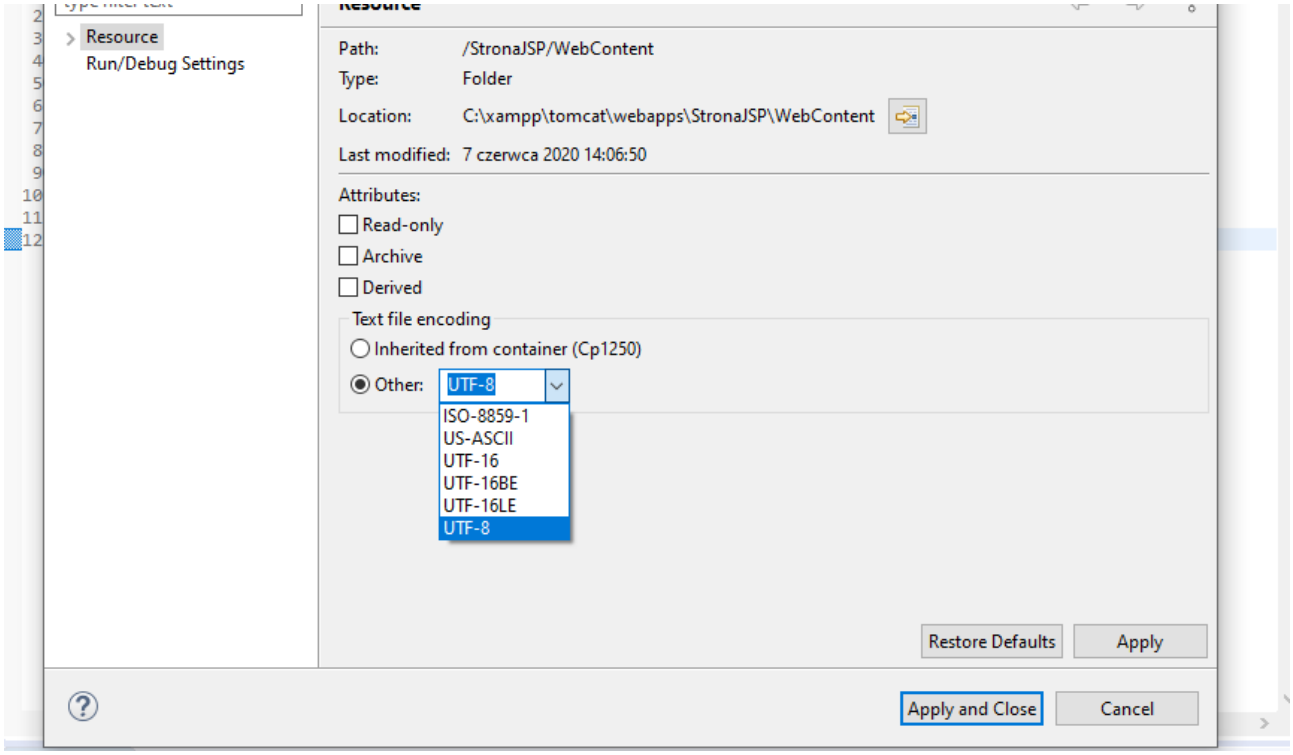
web.xml index.jsp

```
1  <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2      pageEncoding="ISO-8859-1"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="ISO-8859-1">
7  <title>Insert title here</title>
8  </head>
9  <body>
10
11 </body>
12 </html>
```

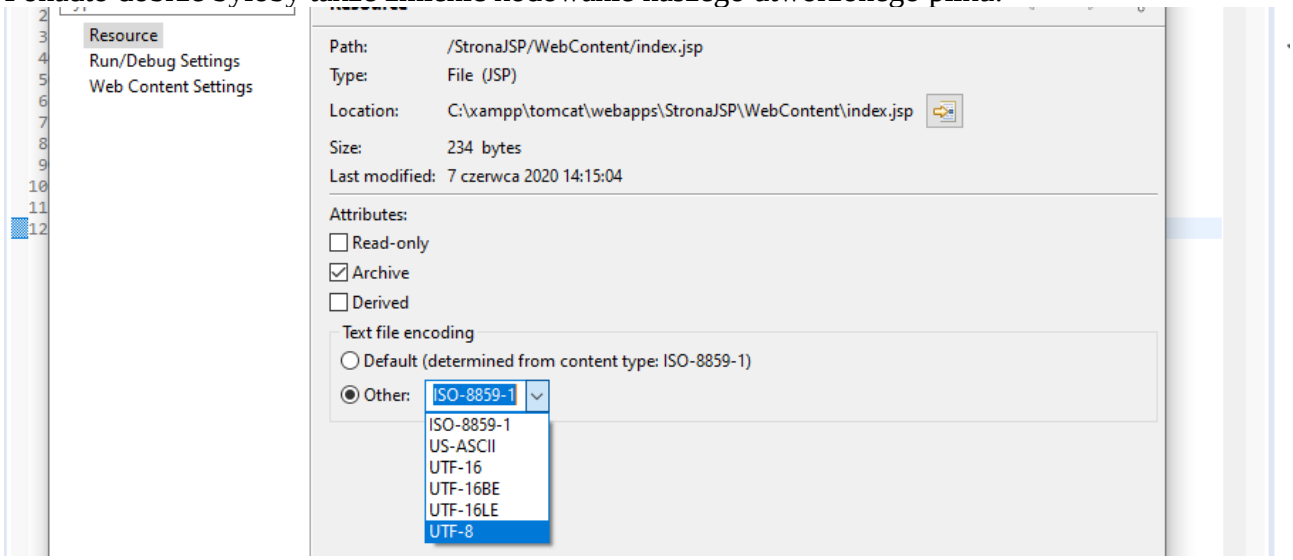
Należy pamiętać, że w zależności od używanego systemu otrzymamy inne kodowanie znaków narodowych. Dlatego, jeżeli chcemy posiadać kodowanie UTF-8 dobrym rozwiązaniem będzie zmienienie kodowania plików projektu:



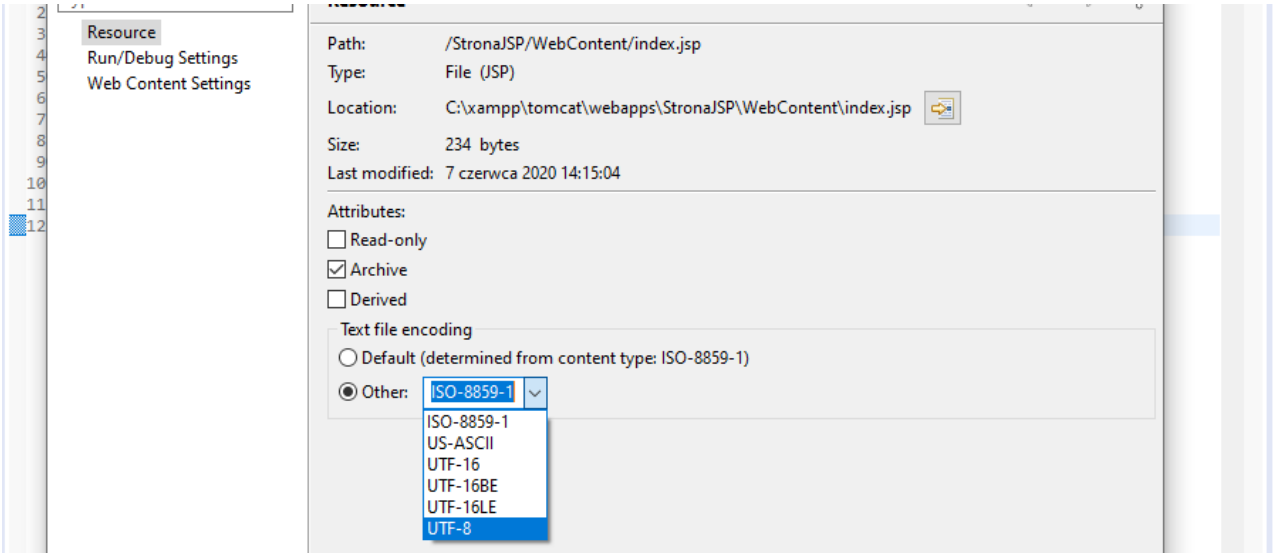
Będąc we właściwościach projektu zmieniamy tę opcję:



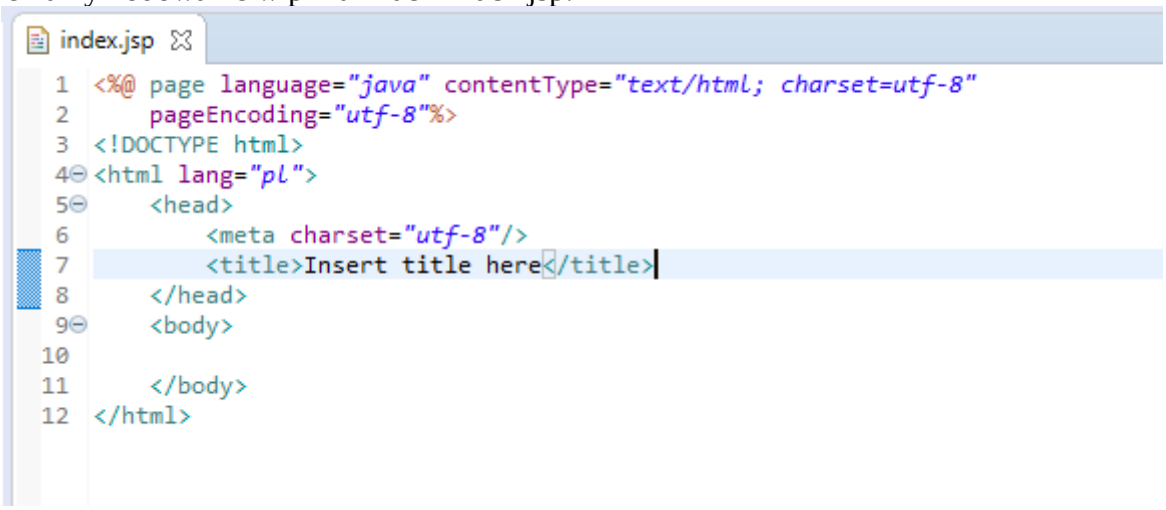
Ponadto dobrze byłoby także zmienić kodowanie naszego utworzonego pliku:



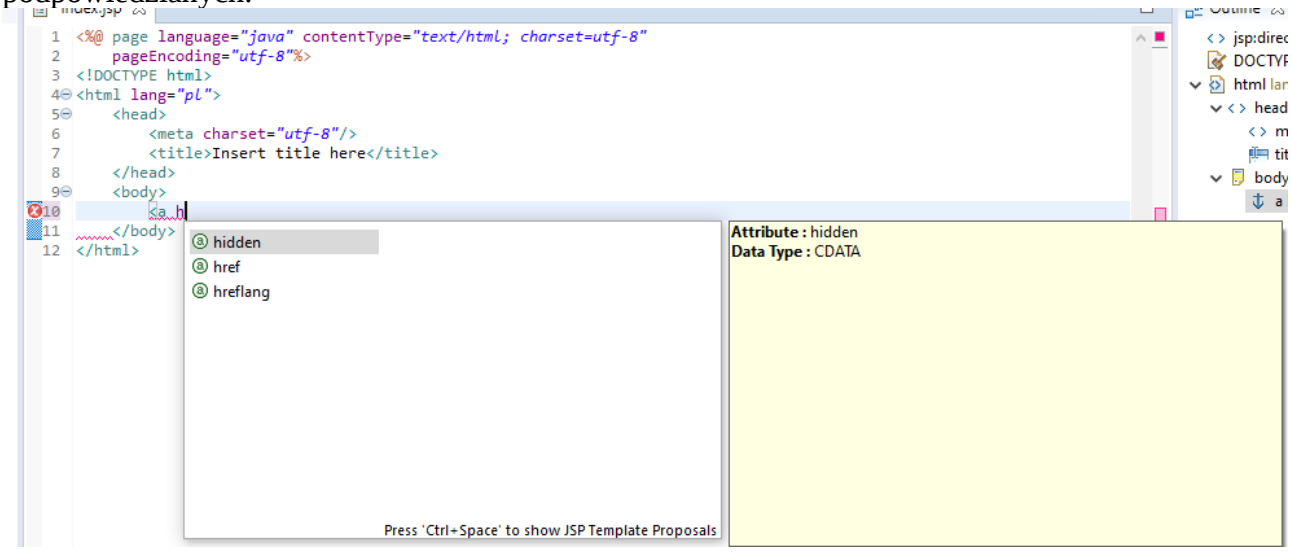
Na koniec potwierdzamy zmiany:



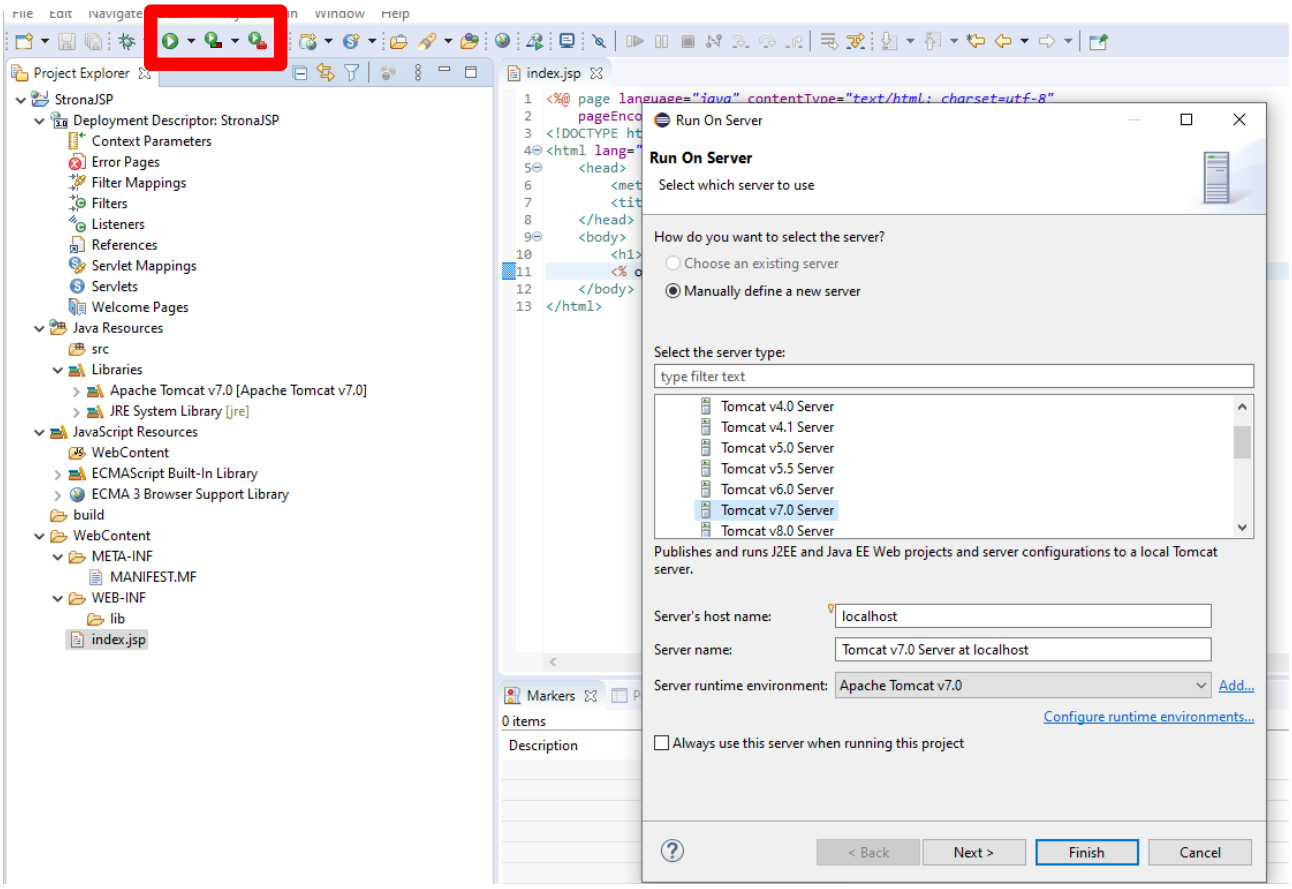
I zmieniamy kodowanie w pliku znacznikach jsp:



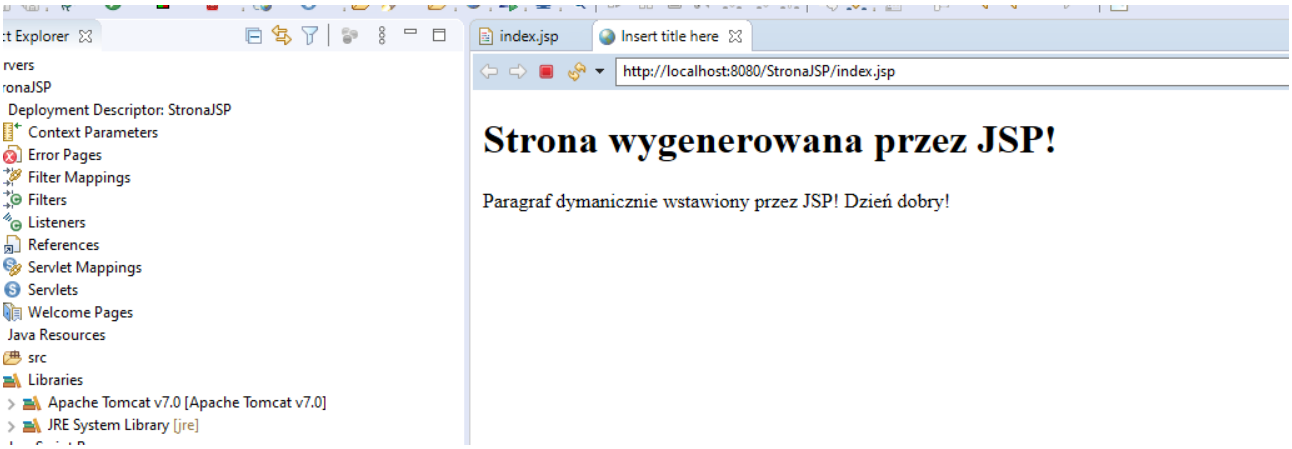
Miłym akcentem będzie też uporządkowanie kodu, celem lepszego wglądu w zawartość HTML. Tworząc strony JSP w Eclipse mamy ten komfort, że większość poleceń zostanie nam podpowiedzianych:



Na zrzucie powyżej widać podpowiedzi do kodu HTML.
Jeżeli chcemy uruchomić nasz projekt:



Po kliknięciu przycisku Finish, jeżeli ustawiliśmy posiadany serwer Tomcat jako uruchomieniowy, zobaczymy stronę odpaloną w oknie naszego Eclipse:



Należy mieć świadomość, że adres został zmieniony jedynie na potrzeby Eclipse. Do chwili gdy serwer na Eclipse działa, adres będzie taki jak podany nawet w przeglądarkach poza Eclipse (np. Firefox czy Chrome/Chromium). Jeżeli jednak wyłączymy serwer Tomcat Eclipse i zechcemy używać serwera dostarczanego przez XAMPP/własnego, wtedy musimy podać adres taki jak poprzednio napisany (z WebContent).

INFORMACJA: By wyłączyć serwer JSP w Eclipse wystarczy kliknąć na przycisk zatrzymania (zaznaczony na zrzucie poniżej):

Na koniec warto pamiętać, że strony WWW dobrze jest dzielić na następujące katalogi:

- css – powinien zawierać wszystkie pliki css, których będziemy używać z naszymi plikami html.
- js – tutaj najlepiej byłoby umieszczać pliki z kodem JavaScript
- image/figures/photos/multimedia – w ten sposób powinien być nazwany folder przechowujący wszelkie pliki multimedialne. Dodatkowo może być także podzielony na podfoldery, dla każdego rodzaju plików i/lub podstron
- content – tutaj powinny trafiać pliki html/jsp, które będą wykorzystywane na stronie WWW.
- base – folder pozwalający na przechowywanie baz tekstowych (2D).

W katalogu głównym projektu powinien znajdować się jedynie plik index.jsp!

5. Plik web.xml

Plik web.xml pozwala na zdefiniowanie wielu elementów naszej strony WWW. Od stron powitalnych, porzez strony błędów, na filtrach i servletach kończąc. Trzonem naszego pliku będzie taka oto składnia:

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
        version="3.1">
</web-app>
```

Jak można zauważyć, plik jest typowym przedstawicielem XML. Posiada odnośniki do słowników definiujących poszczególne elementy.

Elementami, które można zdefiniować w web.xml:

```
<welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
```

Powyższy fragment wskazuje, które pliki będą czytane jako strony startowe.

```
<error-page>
        <error-code>404</error-code>
        <location>/errors/brak.jsp</location>
    </error-page>
```

To zaś pozwoli obsłużyć wyjątek braku zasobu na serwerze.

DODATKOWE MATERIAŁY:
https://www.tutorialspoint.com/jsp/jsp_quick_guide.htm
https://www.tutorialspoint.com/jsp/jsp_actions.htm
https://www.tutorialspoint.com/jsp/include_directive.htm
https://www.tutorialspoint.com/what-are-jsp-literals
https://www.javatpoint.com/creating-jsp-in-eclipse-ide
https://mkyong.com/web-development/the-web-xml-deployment-descriptor-examples/
https://cloud.google.com/appengine/docs/standard/java/config/webxml