Aplikacja webowa z wykorzystaniem Spring Framework w 5 etapach

Tutorial

Paweł Ociepa, Paweł Pierzchała

ociepa.pawel@gmail.com

paw.pierzchala@gmail.com

26.01.2010

Spis treści:

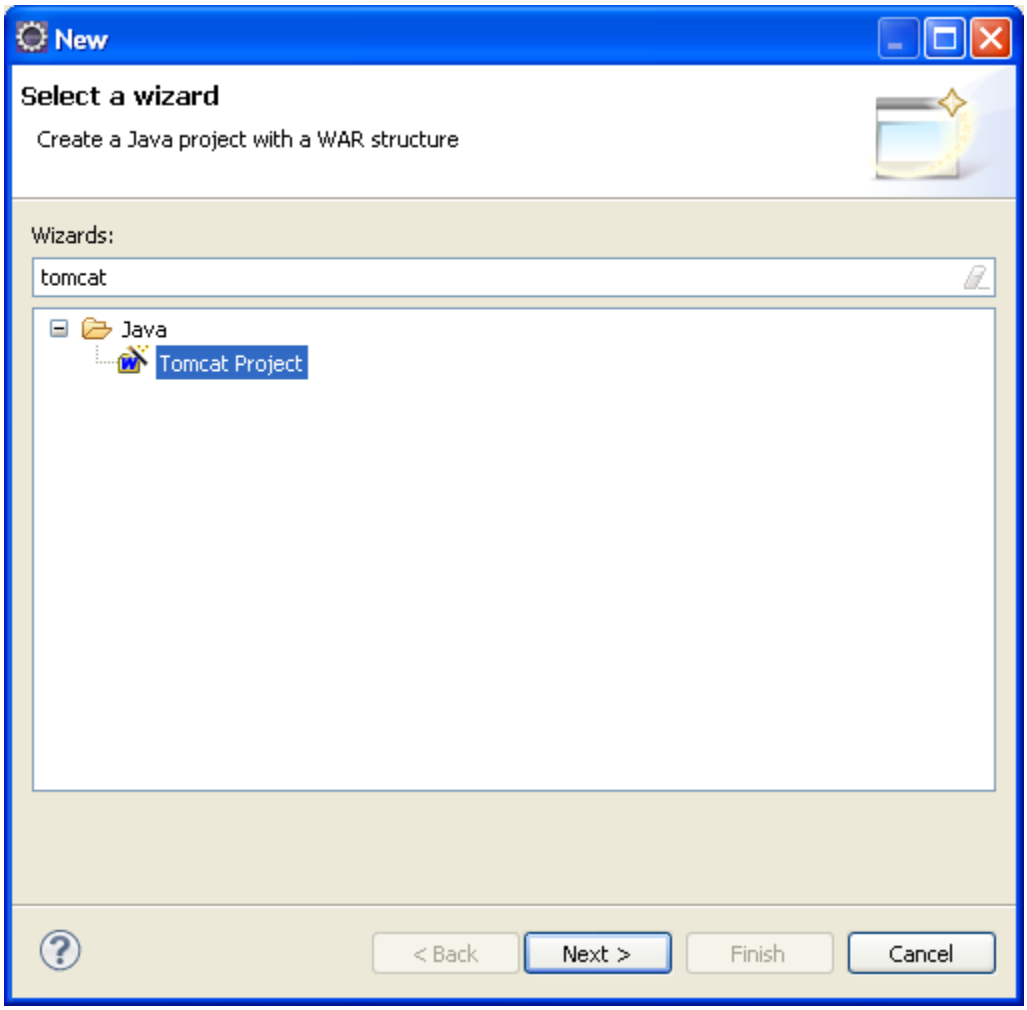
Wprowadzenie:

Przedstawiony poniżej tutorial wykonany został przy użyciu środowiska Eclipse Galileo Java EE w wersji 3.5.1 . Przy tworzeniu nowego projektu korzystaliśmy ponadto z wtyczek SysDeo, która umożliwia uruchamianie projektu na skonfigurowanym wcześniej serwerze Tomcat bezpośrednio z folderu projektu w workspace: Auto-deployment oraz Spring Ide. Natura Springa pozwala między innymi na wykrywanie błędów w wykorzystywanych plikach XML.

Wymagania wstępne:

- Kontener aplikacji Apache Tomcat
- Dystrybucja Spring Framework with dependencies 2.5 (http://www.springsource.org/)
- Hibernate 3.5 (https://www.hibernate.org)
- Spring Security (http://static.springsource.org/spring-security/site/)
- Baza danych (w przykładzie PostgreSQL)

1. Szkielet aplikacji

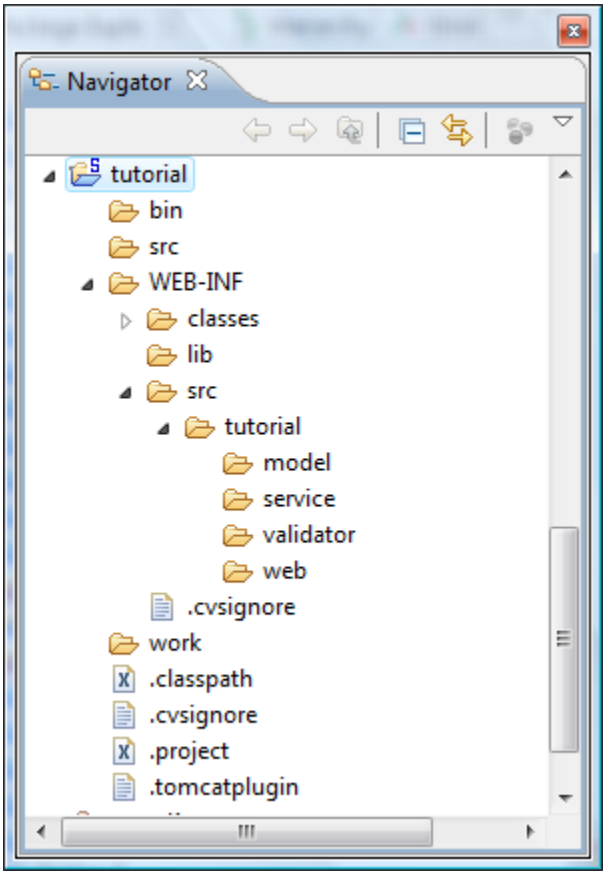


Przy wyborze nowego projektu zdecydowaliśmy się na Tomcat Project. Dzięki temu szkielet aplikacji automatycznie uzupełniony został o foldery: classes,WEB- INF.

Samodzielnie dodajemy następujące:

- `WEB-INF/src/nazwa_projektu/model`
- `WEB-INF/src/nazwa_projektu/service`
- `WEB-INF/src/nazwa_projektu/validator`
- `WEB-INF/src/nazwa_projektu/web`

Docelowy szkielet aplikacji powinien przedstawiać się następująco:



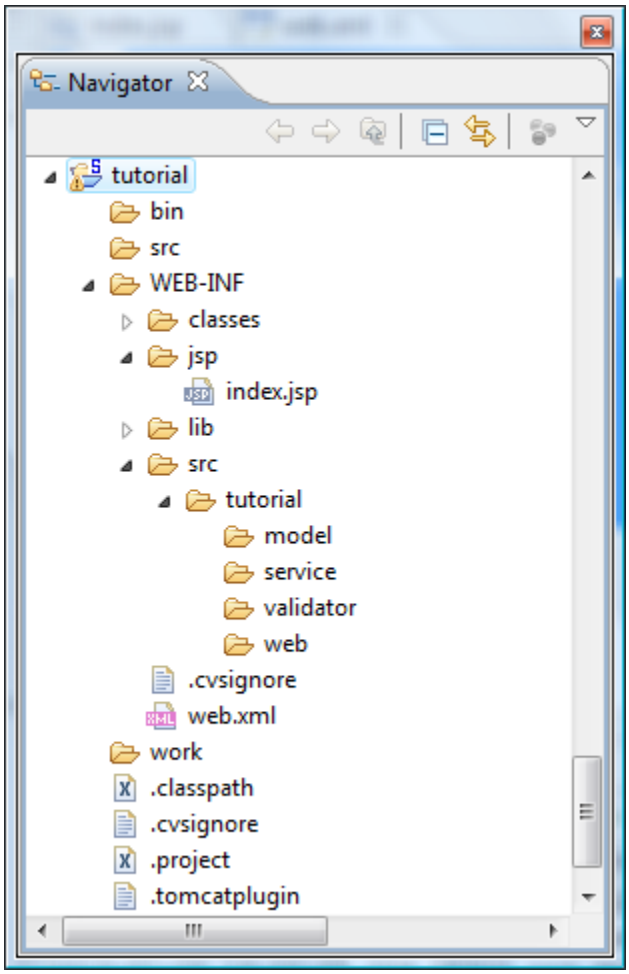
Kolejny etap stanowi dodanie pierwszej strony – index.jsp w nowo utworzonym folderze WEB-INF/jsp.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
              <meta http-equiv="content-type"  content="text/html;charset=utf-8" />

       </head>
<body>
       <p>It works!</p>

</body>
</html>
```

Następnie w folderze WEB-INF dodajemy plik konfiguracyjny web.xml o następującej zawartości.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <welcome-file-list>
    <welcome-file>
      /WEB-INF/jsp/index.jsp
    </welcome-file>
  </welcome-file-list>

</web-app>
```

Po obu operacjach tworzony projekt będzie miał następujący wygląd.

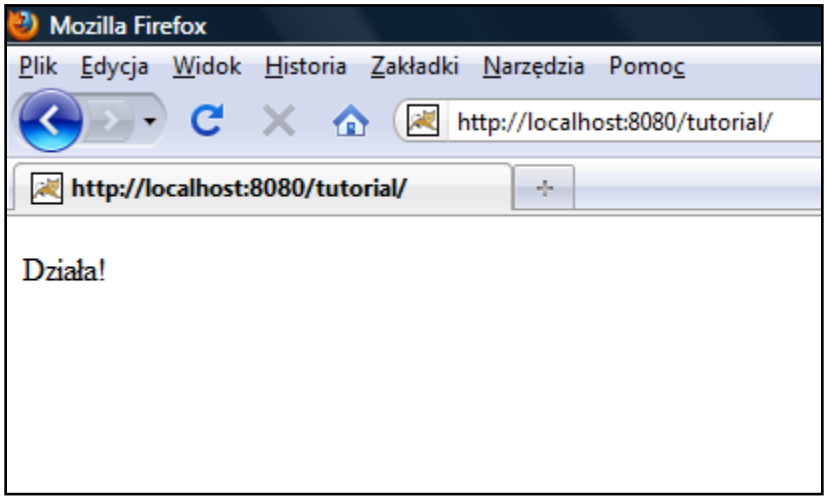


Dodajemy do folderu lib utworzonego w WEB-INF następujące biblioteki z dystrybucji Springa(wymieniona w wymaganiach).

- spring-framework/dist/spring.jar
- spring-framework/dist/modules/spring-webmvc.jar
- spring-framework/lib/jakarta-taglibs/standard.jar
- spring-framework/lib/jakarta-commons/commons-logging.jar
- spring-framework/lib/j2ee/servlet-api.jar
- spring-framework/lib/j2ee/jstl.jar

Po dodaniu pierwszej strony, konfiguracyjnego pliku xml oraz koniecznych bibliotek można uruchomić SysDeo:

Wówczas wystartuje Apache Tomcat, a strona projektu będzie już dostępna lokalnie w przeglądarce.



W przypadku wystąpienia błędu 404 należy upewnić się, iż Apache Tomcat został zainstalowany i skonfigurowany właściwie.

3. Pierwsze kroki ze Spring Framework

W naszym projekcie wykorzystamy wzorzec architektoniczny MVC. We wzorcu wyodrębione zostały 3 podstawowe komponenty aplikacji: model, widok, kontroler.

Model opisuje dane występujące i zależności pomiędzy nimi występujące w projekcie. W naszym przykładzie pliki modeli przechowywane są w folderze:

`WEB-INF/src/tutorial/model`

Widok wyświetla dane przechowywane w modelu, w sposób zdefiniowany przez użytkownika. W naszym przykładzie pliki widoku przechowywane są w folderze:

`WEB-INF/jsp`

Kontroler konwertuje żądanie wysłane przez użytkownika na sposób zrozumiały dla modelu. W naszym przykładzie pliki kontrolerów przechowywane są w folderze:

`WEB-INF/src/tutorial/web`

- W następnym kroku dodamy do pliku konfiguracyjnego web.xml DispatcherServlet oraz mapowanie. (Dodany kod został pogrubiony oraz zaznaczony na czerwono.)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <servlet>
       <servlet-name>tutorial</servlet-name>
       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
       <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
       <servlet-name>tutorial</servlet-name>
       <url-pattern>*.html</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>
      /WEB-INF/jsp/index.jsp
    </welcome-file>
  </welcome-file-list>

</web-app>
```

- Bazując na nazwie servletu zdefiniowanej powyżej w web.xml:

```
<servlet-name>tutorial</servlet-name>
```

kolejny krok stanowi dodanie pliku konfiguracyjnego w folderze `WEB-INF/tutorial-servlet.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
       <bean name="/contact.html" class="tutorial.web.ContactController" />
       <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
       </bean>
</beans>
```

W pliku mapujemy contact.html z kontrolerem ContactController. Zadeklarowany również został ViewResolver. W momencie wywołania w kontrolerze „contact" ViewResolver wywoła plik: /WEB-ING/jsp/contact.jsp.

- Kontroler

W tym kroku stworzymy ContactController. Zgodnie z przyjętą konwencją dodajemy do folderu `/WEB-INF/src/tutorial/web` nową klasę java o nazwie ContactController, w której implementujemy interfejs Controller.

```java
package tutorial.web;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class ContactController implements Controller {
	@Override
	public ModelAndView handleRequest(HttpServletRequest request,
			HttpServletResponse response) throws Exception {

		ModelAndView modelAndView = new ModelAndView("contact");

		modelAndView.addObject("address","Example address");

		return modelAndView;
	}
}
```

- Widok: contact.jsp

W pliku tutorial-servlet.xml zmapowaliśmy ContactController z plikiem contact.jsp. W kolejnym kroku dodajemy zatem w `/WEB-INF/jsp/contact.jsp`

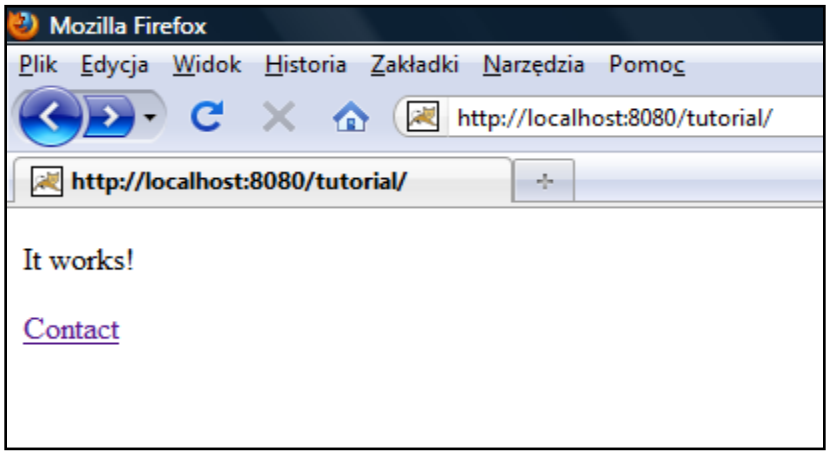
```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="content-type"  content="text/html;charset=utf-8" />

		<title>Contact page</title>
	</head>
<body>
	<h2>Address</h2>
	${address}
</body>
</html>
```

We wcześniej utworzonym pliku index.jsp dodajemy następnie:

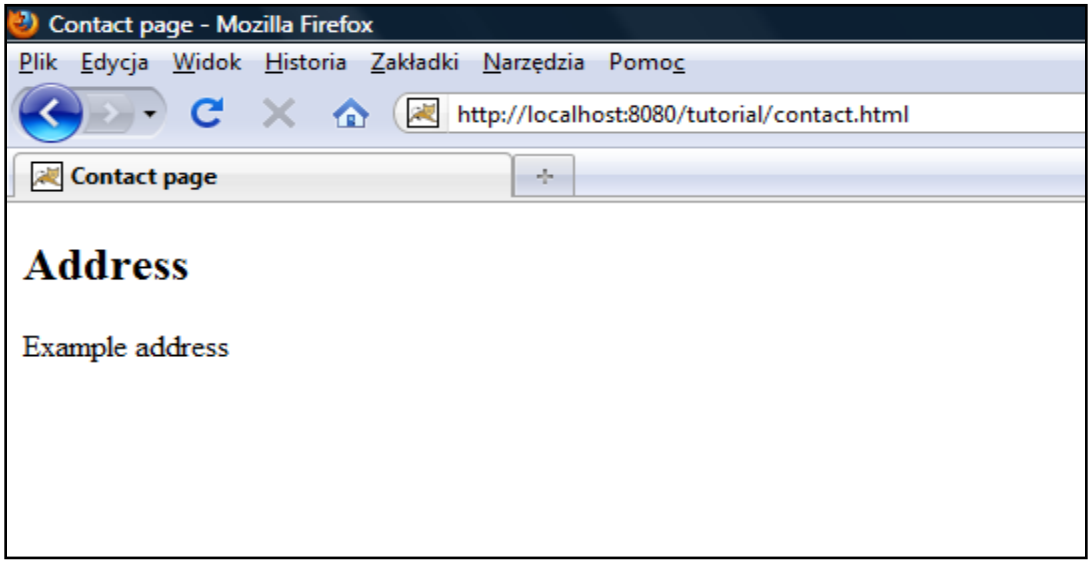
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
              <meta http-equiv="content-type"  content="text/html;charset=utf-8" />
       </head>
<body>
       <p>It works!</p>

       <a href="<c:url value="contact.html"/>">Contact</a>
</body>
</html>
```

- Uruchomienie aplikacji

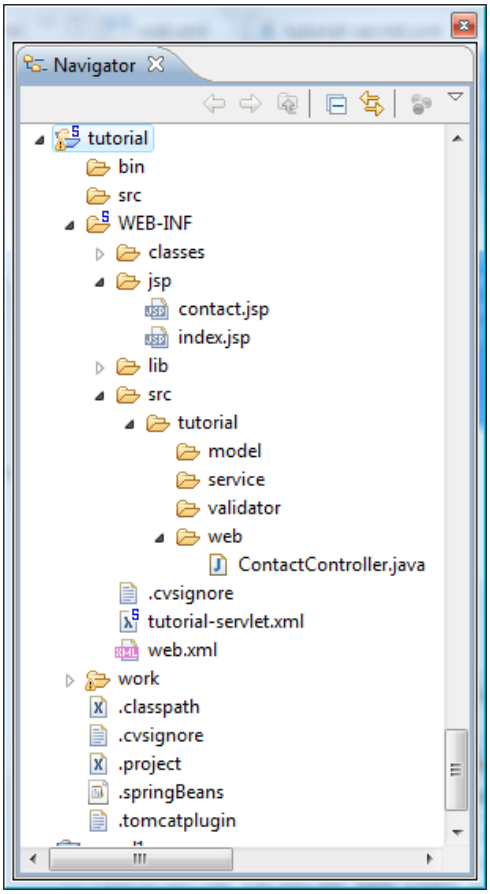
Po ponownym uruchomieniu aplikacji otrzymamy następujący widok w przeglądarce:



Po wybraniu Contact:



Struktura projektu wygląda natomiast następująco:



4. Spring Security

Spring Security to gotowy mechanizm uwierzytelniania i autoryzacji.

Aby skorzystać z tego rozwiązania niezbędne jest zmodyfikowanie plików konfiguracyjnych. (Dodane treści w poszczególnych plikach zostały pogrubione i oznaczone kolorem czerwonym.)

Dodane zostaną nowe pliki:

- Widoku: `login.jsp, logged.jsp`
- Kontrolery: `LoginController.java, LoggedController.java,`
- Konfiguracyjne: `applicationContext-security.xml`

W tym etapie dodajemy do folderu lib utworzonego w WEB-INF następujące biblioteki:

- ascpectjrt.jar
- postgresql-8.4-701.jdbc4.jar
- biblioteki Spring Security

- W pliku web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
         xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

    <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
    </listener>

       <context-param>
         <param-name>contextConfigLocation</param-name>
         <param-value>
             /WEB-INF/tutorial-servlet.xml,
             /WEB-INF/applicationContext-security.xml
         </param-value>
       </context-param>

       <filter>
        <filter-name>springSecurityFilterChain</filter-name>
       <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
    </filter>

    <filter-mapping>
      <filter-name>springSecurityFilterChain</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>

       <servlet>
         <servlet-name>tutorial</servlet-name>
         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
         <load-on-startup>1</load-on-startup>
       </servlet>

    <servlet-mapping>
       <servlet-name>tutorial</servlet-name>
       <url-pattern>*.html</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
      <welcome-file>
        /WEB-INF/jsp/index.jsp
      </welcome-file>
    </welcome-file-list>

</web-app>
```

- W pliku tutorial-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
       <bean name="/contact.html" class="tutorial.web.ContactController" />

       <bean name="/login.html" class="tutorial.web.LoginController" />

       <bean name="/logged.html" class="tutorial.web.LoggedController" />

       <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
       </bean>
    <bean id="myDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
           <property name="driverClassName" value="org.postgresql.Driver"/>
           <property name="url"  value="jdbc:postgresql://adres_bazy"/>
           <property name="username" value="garnuch"/>
           <property name="password" value="password"/>
       </bean>

</beans>
```

Zastosowane przez nas powyżej rozwiązanie z zastosowaniem DriverManagerDataSource jest jednym z 3 możliwych. Możliwe jest np. skorzystanie w tym miejscu z puli połączeń.

UWAGA!

Wykorzystanie mechanizmu Spring Security pociąga za sobą konieczność utworzenia dwóch tabel w bazie danych:

- users

Kolumna	Typ	Not Null
username	character varying(33)	NOT NULL
password	character varying(33)	
enabled	boolean	

- authorities

Kolumna	Typ	Not Null
username	character varying(44)	NOT NULL
authority	character varying(44)	

W pliku konfiguracyjnym dodaliśmy również 2 kolejne mapowania plików jsp z kontrolerami.

W przypadku przejścia na stronę logged.jsp bez zalogowania się użytkownik zostaje automatycznie przekierowany na stronę login.jsp. Należy dodać komentarz, iż w przypadku braku ręcznego ustawienia tej sekwencji Spring Security automatycznie wygeneruje następującą stronę do zalogowania.

Sposób zachowania zdefiniowany przez nas umieszczony jest w nowym niezbędnym pliku:

`WEB-INF/applicationContext-security.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
                        http://www.springframework.org/schema/security
                        http://www.springframework.org/schema/security/spring-
security-2.0.2.xsd">

       <global-method-security secured-annotations="enabled"></global-method-security>
    <http auto-config="true">
        <intercept-url pattern="/contact.html" filters="none" />
        <intercept-url pattern="/logged.html" access="ROLE_ADMIN" />
        <intercept-url pattern="/login.html" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <logout logout-success-url="/login.html" />
        <form-login authentication-failure-url="/login.html?login_error=1" login-
page="/login.html" default-target-url="/logged.html"  />
    </http>

   <authentication-provider>
        <jdbc-user-service data-source-ref="myDataSource"  users-by-username-
query="select username,password,enabled from users where username=?" />
   </authentication-provider>
</beans:beans>
```

Wymienione wcześniej nowe kontrolery odpowiadają za stronę logowania login.jsp (LoginController) oraz stronę wyświetlaną po zalogowaniu logged.jsp (LoggedController)

- LoginController

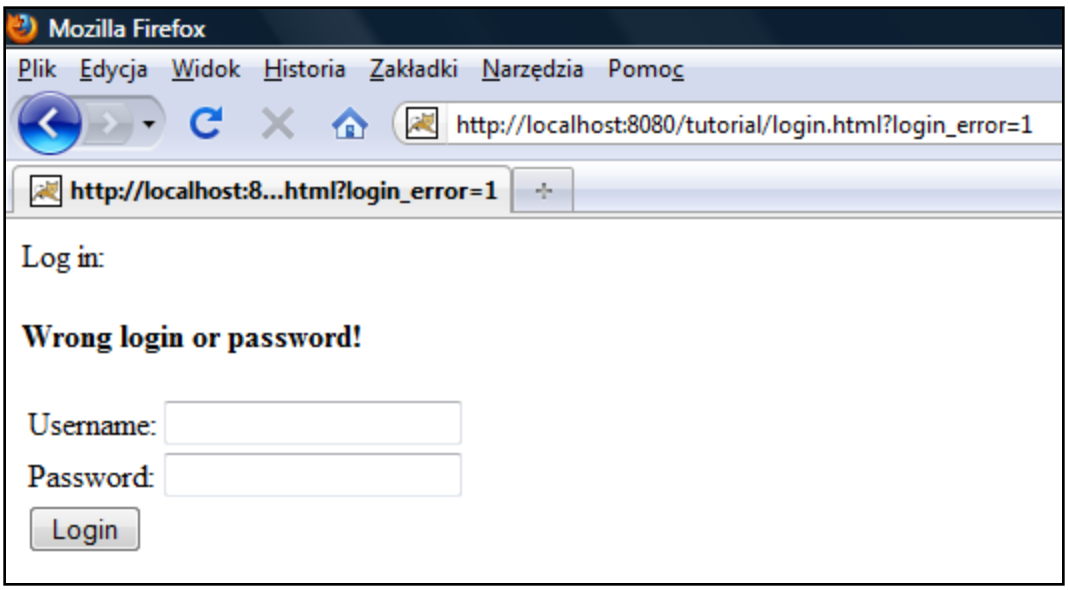
```
package tutorial.web;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class LoginController implements Controller {

	public ModelAndView handleRequest(HttpServletRequest request,
			HttpServletResponse response) throws ServletException {

		ModelAndView modelAndView = new ModelAndView("login");

		return modelAndView;
	}
}
```

- LoggedController.java

```
package tutorial.web;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;

public class LoggedController implements Controller {
	public ModelAndView handleRequest(HttpServletRequest request,
			HttpServletResponse response) throws Exception {

		ModelAndView modelAndView = new ModelAndView("logged");
		return modelAndView;
	}
}
```

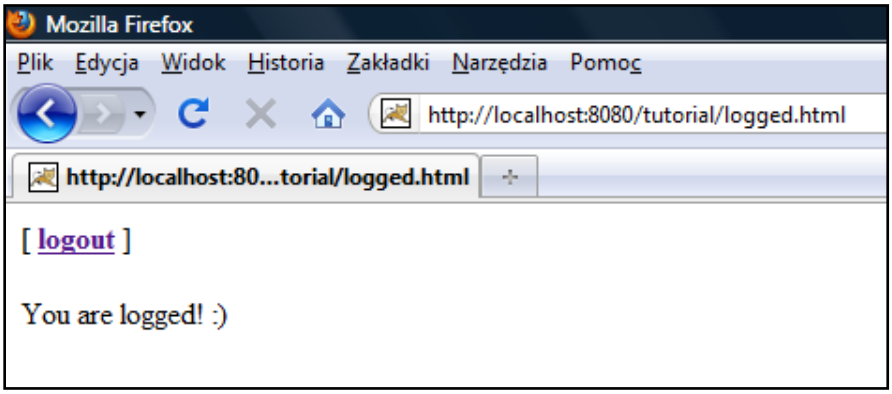
- Własny formularz logowania: login.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
              <meta http-equiv="content-type"  content="text/html;charset=utf-8" />

       </head>
<body>
       Log in:
       <form action="j_spring_security_check">
              <c:if test="${not empty param.login_error}"><br /><b>Wrong login or
password!</span></b><br /></c:if>
              <br />
              <table>
                     <tr>
                            <td>
                                   <label for="j_username">Username: </label>
                            </td>
                            <td>
                                   <input type="text" name="j_username" id="j_username"
/>
                             </td>
                     </tr>
                     <tr>
                            <td>
                                   <label for="j_password">Password: </label>
                            </td>
                            <td>
                                   <input type="password" name="j_password"
id="j_password"/>
                            </td>
                     </tr>
                     <tr>
                            <td>
                                   <input class="button" type="submit" value="Login"/>
                            </td>
                     </tr>
              </table>
       </form>
</body>
</html>
```



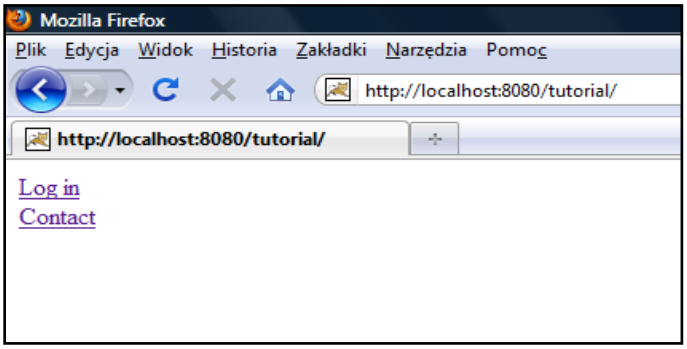
- Strona wyświetlana po pomyślnym zalogowaniu: logged.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
       <head>
              <meta http-equiv="content-type"  content="text/html;charset=utf-8" />
       </head>
<body>
       [ <a href="<c:url value="/j_spring_security_logout"/>"><b>logout</b></a> ]
       <br /><br />
       You are logged! :)
</body>
</html>
```

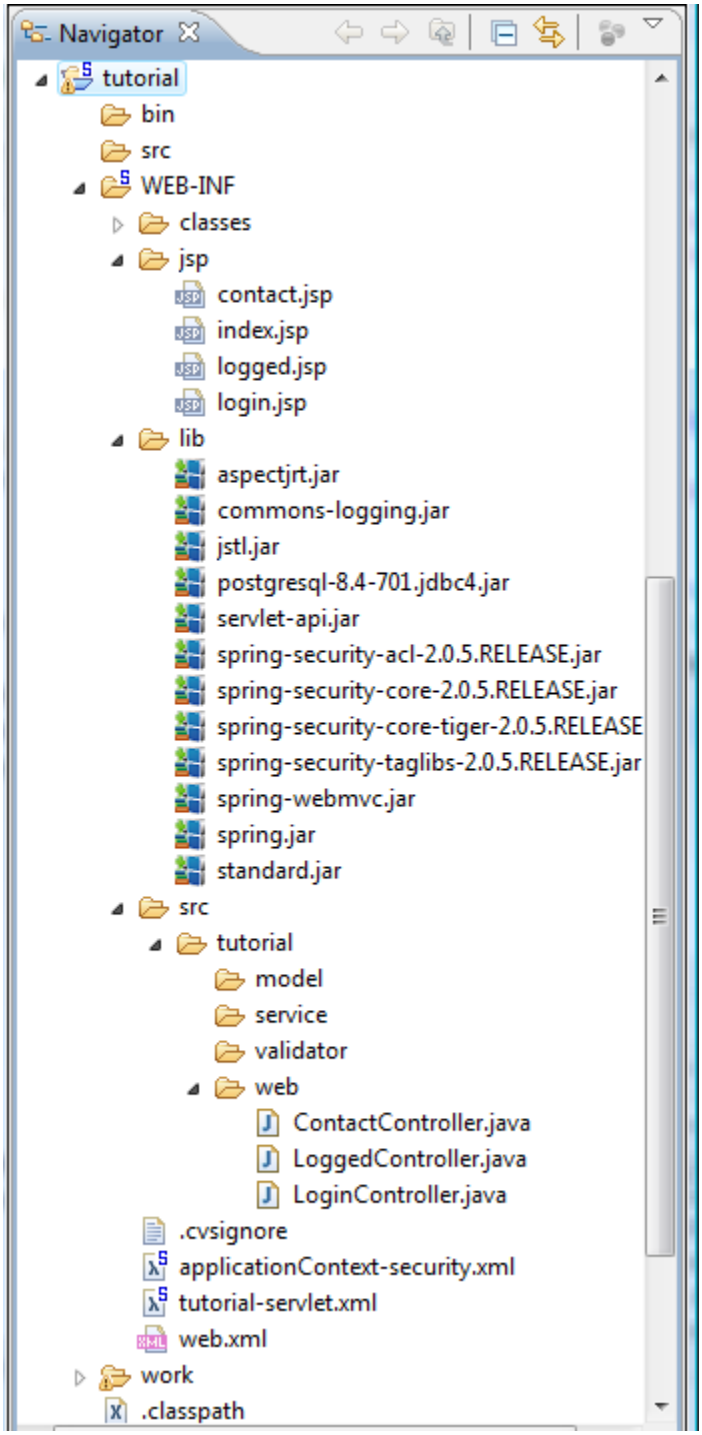


- Strona główna index.jsp
  Dodano odwołanie do strony logowania:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
      <head>
            <meta http-equiv="content-type"  content="text/html;charset=utf-8" />
      </head>
<body>
      <a href="<c:url value="login.html"/>">Log in</a>
      <br />
      <a href="<c:url value="contact.html"/>">Contact</a>
</body>
</html>
```



- Struktura projektu po zakończeniu 3 etapu przedstawia się następująco:



5. Hibernate

W tym kroku pokazana zostanie integracja framework'ów: Hibernate ze Springiem:

Wykorzystując Hibernate połączymy się z bazą danych, w której stworzona zostanie tabela Book odpowiadająca klasie z pakietu model w projekcie.

Dodane zostaną nowe pliki:

- Hibernate: `BookDaoInterface.java, BookDao.java, Book.hbm.xml`
- Model: `Book.java`

W tym etapie dodajemy do folderu lib utworzonego w WEB-INF następujące biblioteki:

- hibernate3.jar
- antlr-2.7.6.jar
- commons-collections-3.1.jar
- commons-logging.jar
- dom4j-1.6.1.jar
- jta-1.1.jar
- slf4j-api-1.5.8.jar
- slf4j-log4j12-1.5.0.jar

Plik konfiguracyjny `tutorial-servlet.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:aop="http://www.springframework.org/schema/aop"
     xmlns:tx="http://www.springframework.org/schema/tx"
     xsi:schemaLocation="
     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
     http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
     http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

       <bean name="/contact.html" class="tutorial.web.ContactController" />

       <bean name="/login.html" class="tutorial.web.LoginController" />

       <bean name="/logged.html" class="tutorial.web.LoggedController">
         <property name="bookDao" ref="BookDao" />
       </bean>
```

```
       <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
       </bean>

   <!-- HIBERNATE SECTION -->

    <bean id="myDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
          <property name="driverClassName" value="org.postgresql.Driver"/>
          <property name="url"  value="jdbc:postgresql://baza"/>
          <property name="username" value="garnuch"/>
          <property name="password" value="password"/>
      </bean>

      <bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean" >
          <property name="dataSource" ref="myDataSource"/>
          <property name="hibernateProperties">
                <props>
                      <prop
key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>
                      <prop
key="hibernate.connection.provider_class">org.hibernate.connection.C3P0ConnectionProvi
der</prop>
                      <prop
key="hibernate.current_session_context_class">thread</prop>
                </props>
          </property>
          <property name="mappingResources" >
              <list>
                  <value>Book.hbm.xml</value>
              </list>
          </property>
      </bean>

      <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
      <property name="sessionFactory" ref="mySessionFactory" />
      </bean>

      <tx:annotation-driven transaction-manager="transactionManager"/>

      <!-- BookDao -->
      <bean id="BookDao" class="tutorial.model.BookDao">
            <property name="sessionFactory" ref="mySessionFactory" />
      </bean>

</beans>
```

W pliku dodano beany Hibernate.

- Zmiany w pliku LoggedController

```
package tutorial.web;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
import tutorial.model.Book;
import tutorial.model.BookDao;
public class LoggedController implements Controller {

	private BookDao bookDao;

	public ModelAndView handleRequest(HttpServletRequest request,
			HttpServletResponse response) throws Exception {

		ModelAndView modelAndView = new ModelAndView("logged");

		//get list of books
		List<Book> books;
		try {

	bookDao.getSessionFactory().getCurrentSession().beginTransaction();

			books = bookDao.getListBook();


	bookDao.getSessionFactory().getCurrentSession().getTransaction().commit();
		}
		catch (Throwable e) {
			if
(bookDao.getSessionFactory().getCurrentSession().getTransaction().isActive()) {

	bookDao.getSessionFactory().getCurrentSession().getTransaction().rollback();
			}
			throw new ServletException(e);
		}

		modelAndView.addObject("books",books);
		return modelAndView;
	}
	public void setBookDao(BookDao bookDao) {
		this.bookDao = bookDao;
	}
}
```

- Dodajemy klasę reprezentującą obiekt modelu Book.java

```
package tutorial.model;

public class Book {
	private int id;
	private String title;
	private String description;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getTitle() {
		return title;
	}
	public void setTitle(String title) {
		this.title = title;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
}
```

- Dodajemy plik odpowiadający za zmapowanie klasy należącej do pakietu model z tablica w bazie danych classes/Book.hbm.xml (zalecana jest zmiana tego miejsca na inne)

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="tutorial.model">
  <class name="Book" table="book" lazy="false" >
		<id name="id"  type= "java.lang.Integer" column="id">
		    <generator class="increment"/>
		</id>
		<property name="title">
		      <column name="title"/>
		</property>
		<property name="description">
		      <column name="description"/>
		</property>
  </class>
</hibernate-mapping>
```

- Dodanie plików DAO: `BookDaoInterface.java, BookDao.java`

    - `BookDaoInterface.java`

```
import java.util.List;
import org.hibernate.SessionFactory;

public interface BookDaoInterface {

	public List<Book> getListBook();
	public int saveOrUpdate(final Book book);
	public void setSessionFactory(SessionFactory sessionFactory);
	public Book getBook(final int id);
	public List<Book> getListBook(String query);

}
```

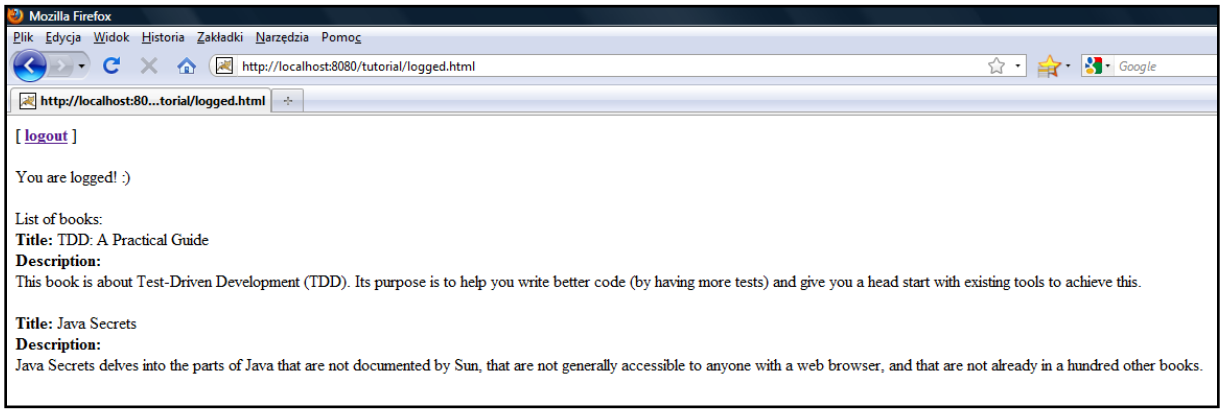
    - `BookDao.java`

```
package tutorial.model;
import tutorial.model.Book;
import java.util.List;
import org.hibernate.SessionFactory;
public class BookDao implements BookDaoInterface {
		public SessionFactory sessionFactory;

		public SessionFactory getSessionFactory() {
			return sessionFactory;
		}
		public void setSessionFactory(SessionFactory sessionFactory)
		{
		this.sessionFactory = sessionFactory;
		}
		public Book getBook(final int id){
		return (Book)sessionFactory.getCurrentSession().load(Book.class, id);
		}

		public List<Book> getListBook() {
		return sessionFactory.getCurrentSession().createQuery("from
Book").list();
		}

		public int saveOrUpdate(Book book) {
			sessionFactory.getCurrentSession().saveOrUpdate(book);

			return
Integer.parseInt(sessionFactory.getCurrentSession().getIdentifier(book).toString());
		}

		public List<Book> getListBook(String query) {
			 return
sessionFactory.getCurrentSession().createQuery(query).list();
		}

}
```

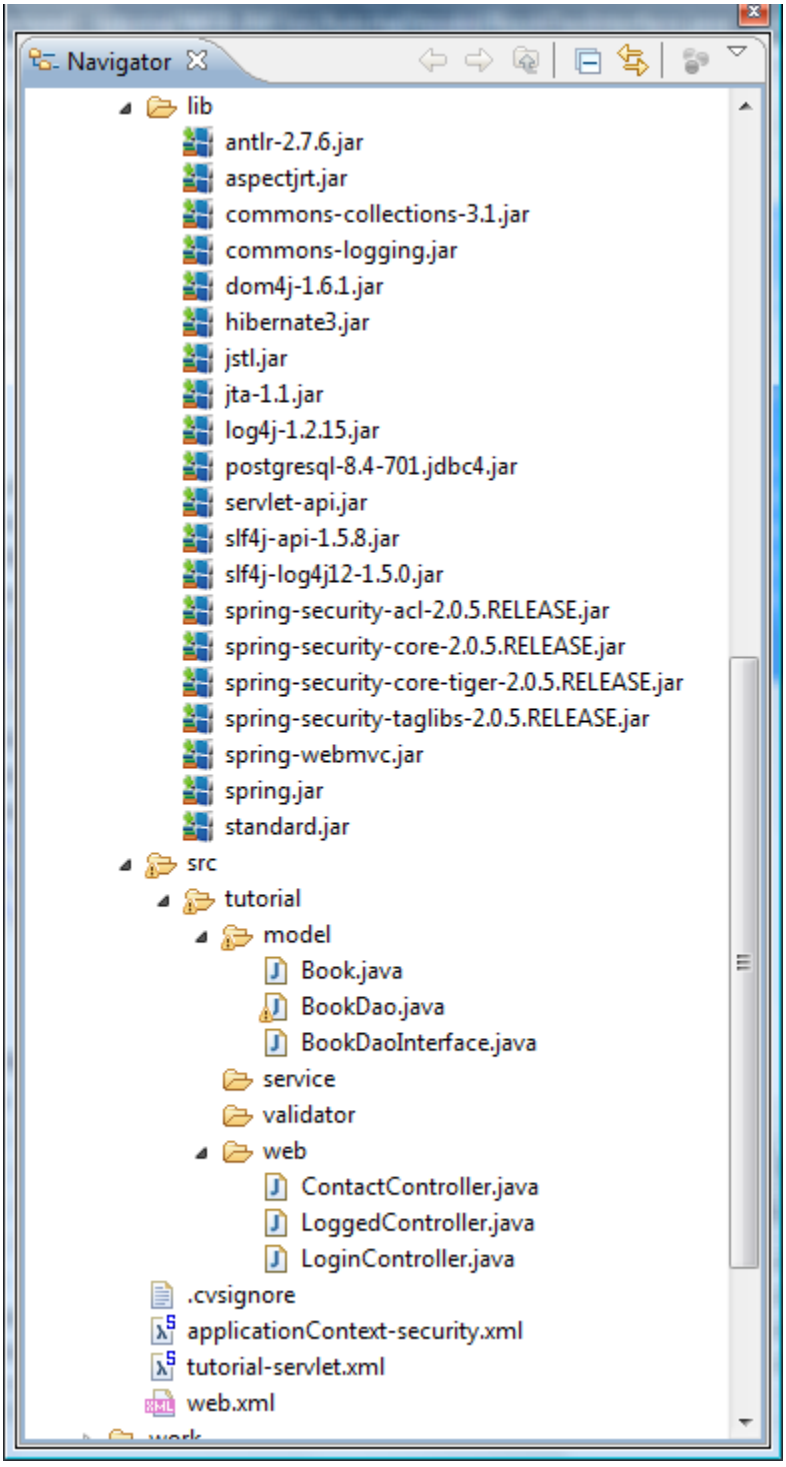
- Zmiany w pliku logged.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
      <head>
            <meta http-equiv="content-type"  content="text/html;charset=utf-8" />
      </head>
<body>
      [ <a href="<c:url value="/j_spring_security_logout"/>"><b>logout</b></a> ]
      <p>You are logged! :)</p>

      <p>List of books:</p>

      <c:forEach items="${books}" var="book">
            <p><b>Title:</b> ${book.title}</p>
            <p><b>Description:</b></p>
            <p>${book.description}</p>
      </c:forEach>

</body>
</html>
```

- Po wykonaniu wszystkich powyższych czynności, zrestartowaniu Tomcata- strona lokalnie powinna wyglądać następująco:



- Struktura projektu na koniec etapu:



6. Form processing

Dodane zostaną nowe pliki:

- Widoku: `add_book.jsp, add_book_success.jsp`
- Kontrolery: `AddBookController.java`
- Serwis: `BookManager.java`

W tym etapie nie dodajemy do folderu lib utworzonego w WEB-INF nowych bibliotek.

- Zmiany w `tutorial-servlet.xml`

  Zaznaczony kod odpowiada za dodany formularz, który widoczny będzie na stronie `add_book.jsp`
  Dodana została także walidacja formularza.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xmlns:aop="http://www.springframework.org/schema/aop"
     xmlns:tx="http://www.springframework.org/schema/tx"
     xsi:schemaLocation="
     http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
     http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd
     http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

       <bean name="/contact.html" class="tutorial.web.ContactController" />

       <bean name="/login.html" class="tutorial.web.LoginController" />

       <bean name="/logged.html" class="tutorial.web.LoggedController">
         <property name="bookDao" ref="BookDao" />
       </bean>

       <bean name="/add_book.html" class="tutorial.web.AddBookController">
              <property name="commandClass" value="tutorial.model.Book" />
              <property name="commandName" value="Book" />
              <property name="validator">
                     <bean class="tutorial.validator.AddBookValidator" />
              </property>
              <property name="bookManager" ref="bookManager" />
       </bean>

       <bean name="bookManager" class="tutorial.service.BookManager">
              <property name="bookDao" ref="BookDao" />
       </bean>

       <bean id="messageSource"
class="org.springframework.context.support.ResourceBundleMessageSource">
         <property name="basename" value="messages"/>
       </bean>
```

```
       <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="viewClass"
value="org.springframework.web.servlet.view.JstlView"/>
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
       </bean>

   <!-- HIBERNATE SECTION -->
    <bean id="myDataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource" >
          <property name="driverClassName" value="org.postgresql.Driver"/>
          <property name="url"
value="jdbc:postgresql://sql.garnuch.nazwa.pl:5433/garnuch_12"/>
          <property name="username" value="garnuch_12"/>
          <property name="password" value="!@12QWerty"/>
       </bean>

       <bean id="mySessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean" >
          <property name="dataSource" ref="myDataSource"/>
          <property name="hibernateProperties">
                <props>
                      <prop
key="hibernate.dialect">org.hibernate.dialect.HSQLDialect</prop>
                      <prop
key="hibernate.connection.provider_class">org.hibernate.connection.C3P0ConnectionProvi
der</prop>
                      <prop
key="hibernate.current_session_context_class">thread</prop>
                </props>
          </property>
          <property name="mappingResources" >
              <list>
                  <value>Book.hbm.xml</value>
              </list>
          </property>
       </bean>

       <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
       <property name="sessionFactory" ref="mySessionFactory" />
       </bean>

       <tx:annotation-driven transaction-manager="transactionManager"/>

       <!-- BookDao -->
       <bean id="BookDao" class="tutorial.model.BookDao">
             <property name="sessionFactory" ref="mySessionFactory" />
       </bean>

</beans>
```

- Zmiany w `applicationContext-security.xml`

  Do kontekstu dodajmy nową stronę: `add_book.jsp`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-
2.0.xsd
                        http://www.springframework.org/schema/security
                        http://www.springframework.org/schema/security/spring-
security-2.0.2.xsd">

      <global-method-security secured-annotations="enabled"></global-method-security>
    <http auto-config="true">
        <intercept-url pattern="/contact.html" filters="none" />
        <intercept-url pattern="/logged.html" access="ROLE_ADMIN" />
        <intercept-url pattern="/add_book.html" access="ROLE_ADMIN" />
        <intercept-url pattern="/login.html" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <logout logout-success-url="/login.html" />
        <form-login authentication-failure-url="/login.html?login_error=1" login-
page="/login.html" default-target-url="/logged.html"  />
    </http>

   <authentication-provider>
        <jdbc-user-service data-source-ref="myDataSource"  users-by-username-
query="select username,password,enabled from users where username=?" />
   </authentication-provider>
</beans:beans>
```

- Dodajemy nowy kontroler odpowiadający za formularz `AddBookController.java`

W kontrolerze występuje odwołanie do serwisu BookManager. Mechanizm polega na tym, iż po wypełnieniu formularza na stronie add_book.jsp przekazany zostanie obiekt do kontrolera AddBookController a ten przekaże go do serwisu. Po udanym dodaniu nowej książki żądanie zostanie przekazane spowrotem do kontrolera i ten przekieruje na stronę add_book_success.jsp.

```
package tutorial.web;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.springframework.validation.BindException;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.SimpleFormController;
import tutorial.model.Book;
import tutorial.service.BookManager;
public class AddBookController extends SimpleFormController {
	private BookManager bookManager;

	@Override
	protected ModelAndView onSubmit(HttpServletRequest request,
			HttpServletResponse response, Object command, BindException
errors)
			throws Exception {

		Book book = (Book) command;

		try {
			bookManager.addBook(book);
		}
		catch (ServletException e) {
			//something to handle this exception
		}
		ModelAndView modelAndView = new ModelAndView("add_book_success");

		return modelAndView;
	}

	public void setBookManager(BookManager bookManager) {
		this.bookManager = bookManager;
	}
}
```

- Dodany został serwis obsługujący formularz `BookManager.java`

Do serwisu przekazywany jest obiekt, 'wypełniany' w formularzu na stronie add_book.jsp, a następnie nowa książka zostaje dodana do bazy.

```
package tutorial.service;
import javax.servlet.ServletException;
import tutorial.model.Book;
import tutorial.model.BookDao;
public class BookManager {
	private BookDao bookDao;
	public void addBook(Book book) throws ServletException {

		try {
			bookDao.sessionFactory.getCurrentSession().beginTransaction();

			bookDao.saveOrUpdate(book);


	bookDao.getSessionFactory().getCurrentSession().getTransaction().commit();
		}
		catch (Throwable e) {
			if
(bookDao.getSessionFactory().getCurrentSession().getTransaction().isActive()) {

	bookDao.getSessionFactory().getCurrentSession().getTransaction().rollback();
			}
			throw new ServletException(e);
		}
	}

	public void setBookDao(BookDao bookDao) {
		this.bookDao = bookDao;
	}
}
```

- Dodany został walidator pól formularza : `AddBookValidator.java`

```
package tutorial.validator;
import org.springframework.validation.Errors;
import org.springframework.validation.ValidationUtils;
import org.springframework.validation.Validator;
import tutorial.model.Book;
public class AddBookValidator implements Validator {

	public boolean supports(Class aClass) {
		//classes suported by this validator
		return Book.class.equals(aClass);
	}

	public void validate(Object obj, Errors errors) {
		Book user = (Book) obj;

		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "title",
"field.required");
		ValidationUtils.rejectIfEmptyOrWhitespace(errors, "description",
"field.required");
	}
}
```

- Dodana strona z formularzem: `add_book.jsp`

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="content-type"  content="text/html;charset=utf-8" />

	</head>
<body>
	<h2>Add a book</h2>
	<form:form method="post" commandName="Book">
		<p>Title: <form:input path="title"/><form:errors path="title" /></p>
		<p>Description: <form:input path="description" /><form:errors
path="description" /></p>
		<input type="submit" value="Add" />
	</form:form>
</body>
</html>
```

- Dodana została strona informująca o prawidłowym dodaniu nowej książki: `add_book_success.jsp`

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="content-type"  content="text/html;charset=utf-8" />
	</head>
<body>
	<p>Your book was added successfully!</p>
	<p><a href="<c:url value="logged.html"/>">Back to main page</a></p>
</body>
</html>
```

- Dodany został plik `messages.properties` (wykorzystywany w walidatorze)

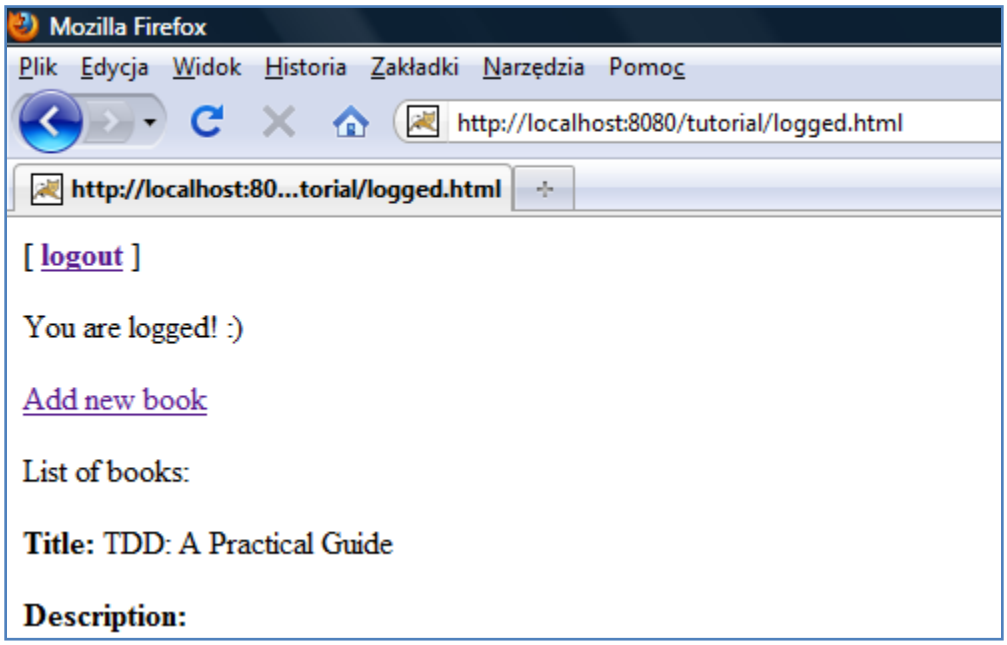
```
field.required= *field required
```

- Zmiany w pliku `logged.jsp` polegające na dodaniu odwołania do nowej strony `add_book.jsp`

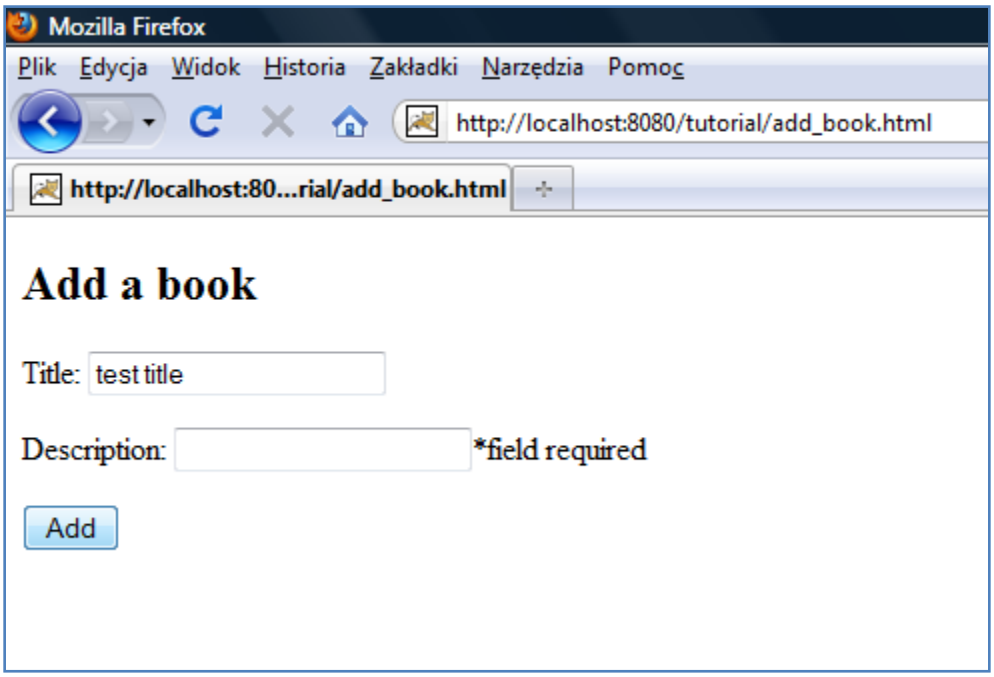
```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
	<head>
		<meta http-equiv="content-type"  content="text/html;charset=utf-8" />

	</head>
<body>
	[ <a href="<c:url value="/j_spring_security_logout"/>"><b>logout</b></a> ]
	<p>You are logged! :)</p>
	<p><a href="<c:url value="add_book.html"/>">Add new book</a></p>
	<p>List of books:</p>

	<c:forEach items="${books}" var="book">
		<p><b>Title:</b> ${book.title}</p>
		<p><b>Description:</b></p>
		<p>${book.description}</p>
	</c:forEach>

</body>
</html>
```

- Po wykonaniu wszystkich czynności zawartych w etapie, restartując Tomcata i uruchamiając projekt lokalnie otrzymujemy następujące wyniki:

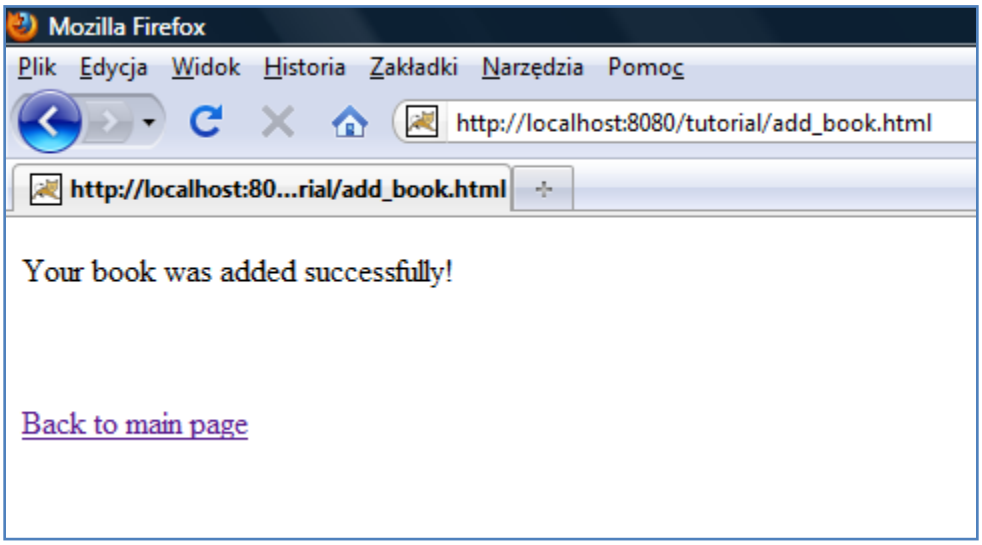
    - Strona `logged.jsp`



    - Nowa strona add_book.jsp



- Strona `add_book_success.jsp` informująca o dodaniu nowej książki.



- Strona logged.jsp z wypisanymi wszystkimi dotychczas dodanymi książkami.