Wicket

Apache Wicket, czy też po prostu Wicket jest jednym z wielu frameworków przygotowanych do wytwarzania aplikacji w języku Java. Istnieje jednak kilka ciekawych aspektów, które wyróżniają go spośród pozostałych frameworków. Warto zwrócić także uwagę na fakt, że korzystanie z niego jest tak lekkie, jak on sam.

Wstęp

Wicket jest tworzony od wiosny 2005 roku przez Jonathana Locke. Wersja 1.0 została wydana w czerwcu tegoż roku. Dwa lata później Wicket stał się projektem ze stajni Apache – licencjonowany jest na Apache Licence 2.0. Framework ten jest ciągle rozwijany (aktualna wersja w czasie pisania tego tekstu, to 1.4.5). Jego oficjalne logo widoczne jest na rysunku 1.

Rysunek 1: Oficjalne logo projektu Apache Wicket

W porównaniu do innych tego typu projektów (Spring, JSP, JSF), Wicket jest bardzo prosty w konfiguracji. Ilość XMLa została bowiem ograniczona do niezbędnego minimum, który sprowadza się praktycznie bardziej do konfiguracji aplikacji, aniżeli samego frameworka – tutaj mamy do czynienia właściwie tylko z kodem HTML oraz Java. Oczywiście Wicket wprowadza kilka swoich znaczników, bądź uzupełnia znaczniki HTMLa o dodatkowe atrybuty, ale nie jest tego aż tak wiele, jak w przypadku np. JSF, przez co czas, jaki należy poświęcić na „nauczenie się" Wicketa będzie bardzo krótki. Powstało także kilka narzędzi, które integrując się ze środowiskiem programistycznym, ułatwią przygotowanie aplikacji (np. edytory typu WYSIWYG).

Zalety Wicketa

Wicket opiera się, podobnie jak Swing, na budowie komponentowej, w pełni zorientowanej obiektowo. Zarówno strony jak i komponenty przygotowane przy użyciu Wicketa są prawdziwymi obiektami rozumianymi w sensie Javy, wspierają one enkapsulację, dziedziczenie oraz zdarzenia. Ponadto, jako że Wicket to praktycznie HTML i Java, można w łatwy sposób tworzyć aplikacje korzystając ze swojego ulubionego edytora kodu.

Interesującym jest też fakt, że Wicket nie miesza znaczników HTMLa z kodem Javy. Obowiązuje prosta zasada – każda strona składa się z pliku .html i z pliku .java o tej samej nazwie. Powiązanie odbywa się dzięki atrybutowi *wicket:id*, który zamieszczamy w znaczniku HTML i który w odpowiedni sposób przypisujemy obiektom w Javie.

Jak przystało na poważny framework, Wicket zapewnia także bezpieczeństwo aplikacji na stosownym poziomie. Adresy URL nie zawierają wrażliwych informacji, mamy wsparcie dla sesji. W planach istnieje także wprowadzenie szyfrowania do adresów URL, by uzyskać jeszcze większe bezpieczeństwo.

Wicket dostarcza również wsparcie dla transparentnego skalowania aplikacji oraz dostarcza transparentną obsługę przycisku „Cofnij" przeglądarki, a także mnóstwo komponentów, które

można wielokrotnie wykorzystywać w swoim kodzie. Lokalizacja o standardowe pliki .properties nie sprawia problemów, także w bardziej wymagających przypadkach, jak teksty parametryzowane, etc. Oczywiście można także programowo modyfikować każdy parametr wykorzystywany w HTMLu, a na stronie domowej można odnaleźć wiele przykładów, których samo przestudiowanie dostarczy ogromnej wiedzy z zakresu tego frameworka.

Zobaczmy, jak Wicket sprawdza się w praktyce.

Przygotowanie projektu

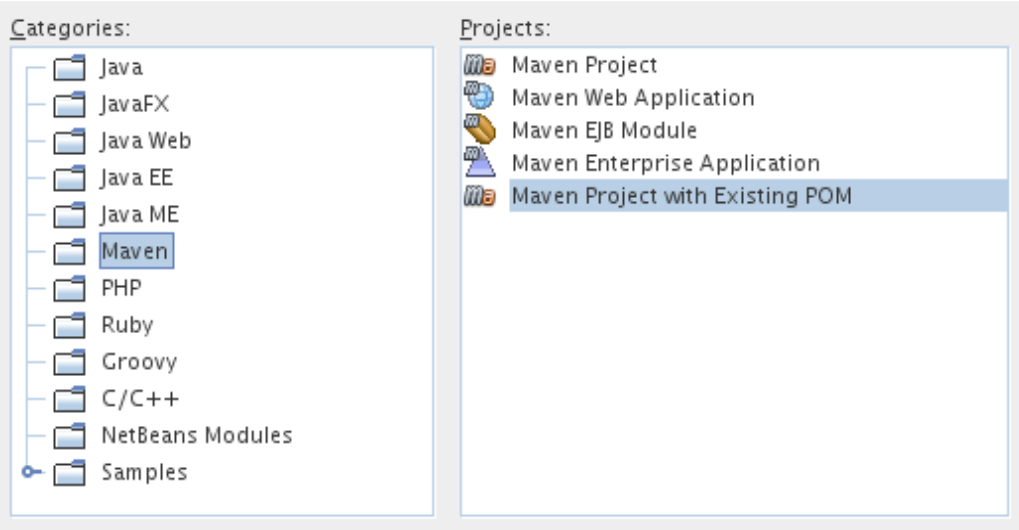
Najprościej jest utworzyć projekt przy pomocy narzędzia Maven. Wystarczy bowiem jedno polecenie: *mvn archetype:create -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-quickstart -DarchetypeVersion=1.4.5 -DgroupId=com.mycompany -DartifactId=myproject*, gdzie oczywiście dobieramy odpowiednie dla projektu parametry. Jeżeli chcemy tworzyć aplikację przy użyciu takich środowisk jak NetBeans, czy Eclipse, wystarczy utworzyć wtedy projekt na bazie istniejącego pliku POM narzędzia Maven i już można zaczynać pracę. Oczywiście istnieją także wtyczki, które ułatwiają obsługę Wicketa w środowisku programistycznym, np. Wicket Bench dla Eclipse. Wtyczka ta pozwala na utworzenie projektu z poziomu środowiska, a także dodawanie kolejnych stron, czy też paneli (o których nieco później). Można także utworzyć projekt na bazie aplikacji Wicket Quickstart, która dostępna jest na stronie domowej projektu i zawiera szkielet, o który możemy oprzeć własne produkty. Przygotowany jest on w taki sposób, że nie wymaga zewnętrznego serwera (kontenera servletów), bowiem pobierając archiwum, otrzymujemy także bardzo lekki serwer Jetty, który waży mniej megabajtów, niż można by się spodziewać – zachęcam do przekonania się samodzielnie. Zatem pobierając szkielet, otrzymujemy także środowisko uruchomieniowe dla naszej aplikacji. Przygotowany jest w tym także szkielet pod testy jednostkowe.

W niniejszym artykule przedstawię sposób tworzenia prostej aplikacji przy użyciu Wicketa w wersji 1.4.5 oraz środowiska programistycznego NetBeans w wersji 6.8.

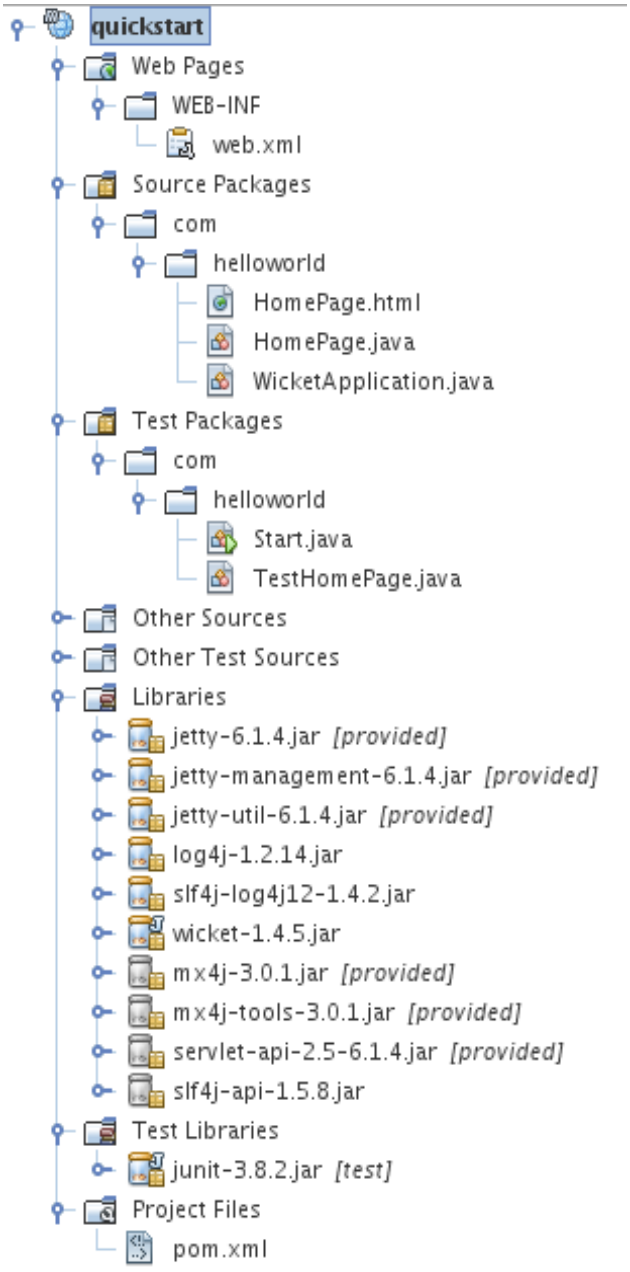
Zacznijmy więc od utworzenia projektu z wykorzystaniem Mavena przy pomocy wspomnianego już wyżej polecenia, które przybierze na nasze potrzeby następującą postać: *mvn archetype:create -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-quickstart -DarchetypeVersion=1.4.5 -DgroupId=com.helloworld -DartifactId=HelloWorld*. Maven utworzy przede wszystkim katalog o nazwie podanej w parametrze *DartifactId*, a bezpośrednio w nim znajdzie się plik *pom.xml*. Jest to plik konfiguracyjny Mavena – aby poznać szczegóły, odsyłam do dokumentacji tego narzędzia.

Uruchamiamy teraz NetBeansa. Klikamy kolejno menu *File*, *New Project*. Pojawia się okno, w którym tworzymy nowy projekt. Jako kategorię projektu, wybieramy oczywiście *Maven*, a projektem będzie *Maven Project with Existing POM*. Przechodząc przez to okno dialogowe (rys.2) przyciskiem *Next* i *Finish*, NetBeans poprosi nas o wskazanie pliku *pom.xml*, co też czynimy.

Po przygotowaniu projektu, w NetBeansie pojawia się drzewo projektu, które odzwierciedla rysunek 3. U góry, w katalogu WEB-INF znajduje się standardowy plik *web.xml*, który konfiguruje aplikację webową. Nieco niżej mamy już pliki źródłowe, a te interesują nas najbardziej. Widzimy, że strona domowa projektu podzielona jest na dwa pliki – *HomePage.java* oraz *HomePage.html*. Logika oddzielona jest od warstwy widoku. Plik *WicketApplication.java* będzie zawierał kod odpowiedzialny za działanie aplikacji (obsługa sesji, mapowanie adresów URL, etc.). Następnie uwagę przykuwają testy jednostkowe. Warto zaglądnąć do pliku *Start.java*, który przygotowuje serwer Jetty (to właśnie on jest wykorzystany do przeprowadzania testów) oraz do pliku *TestHomePage.java*, gdzie zauważyć można asercje specyficzne dla Wicketa – pozwolą one w prosty sposób sprawdzić, czy poszczególne strony zachowują się w żądany sposób.



Rysunek 2: Tworzenie projektu

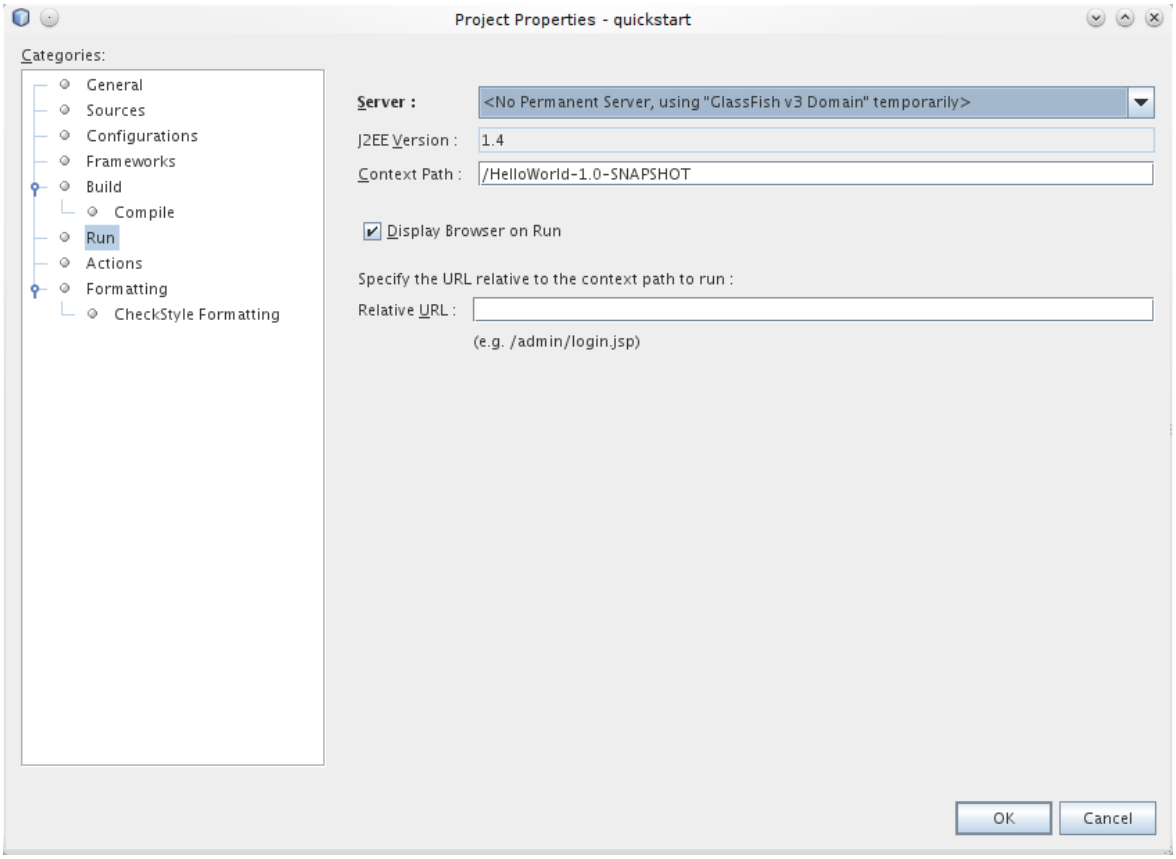


Rysunek 3: Drzewo projektu

Uruchamianie aplikacji

Tak przygotowaną aplikację, a właściwie jej szkielet, można już uruchomić. Wystarczy kliknąć prawym przyciskiem na nazwę projektu i z menu kontekstowego wybrać polecenie *Run.* Za pierwszym razem NetBeans poprosi nas o wybór serwera. Uruchamiania aplikacji, w przeciwieństwie do testów, nie odbywa się automatycznie i transparentnie w serwerze Jetty – tutaj musimy sami skonfigurować środowisko uruchomieniowe. Szczęśliwie jednak się składa, że NetBeans dostarcza wsparcie dla Glassfisha (którego można pobrać wraz z pełną wersją NetBeansa). Glassfish jest serwerem rozwijamy przez firmę Sun, która to z kolei zajmuję się rozwijaniem Javy, także mimo faktu, iż serwer ten jest dość ciężki, jest on na pewno bardzo dobry jakościowo.

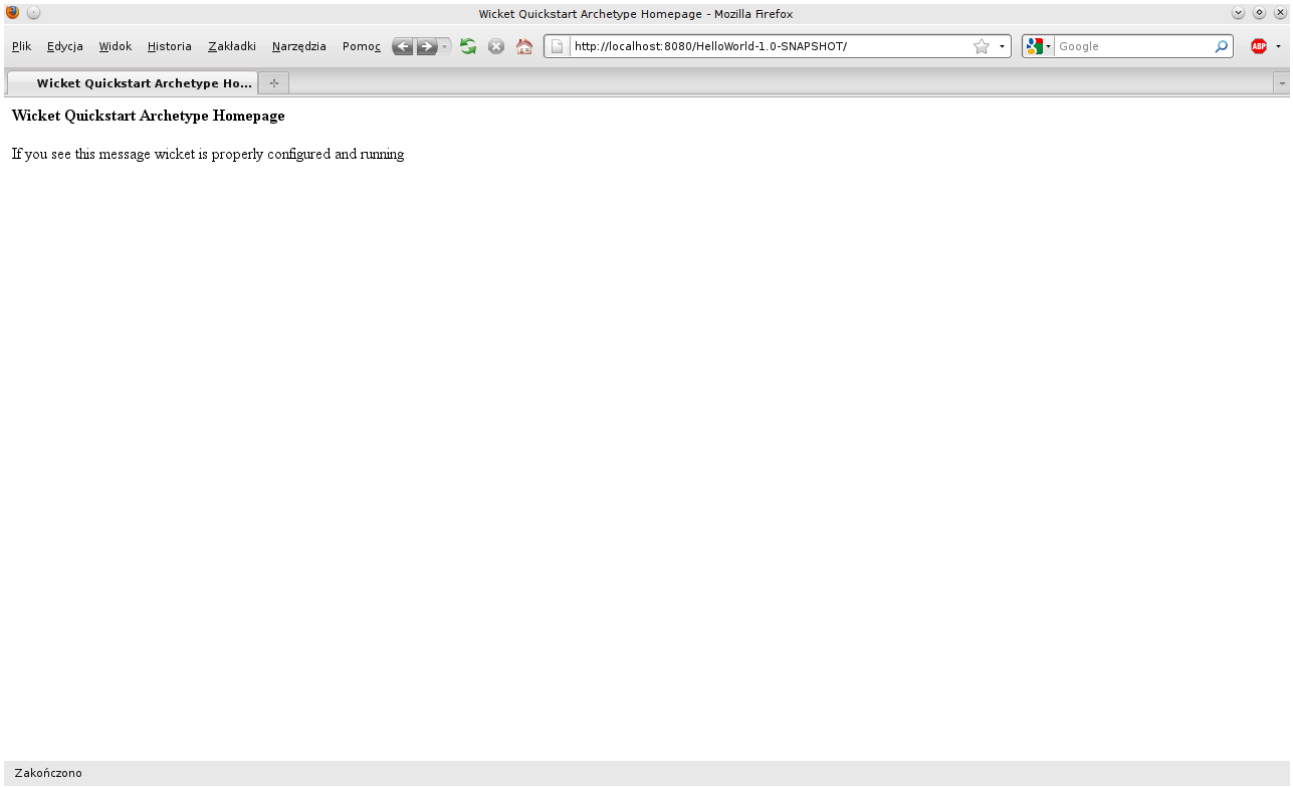
Serwer można wybrać także klikając prawym przyciskiem myszki na projekt i wybierając z menu kontekstowego pozycję *Properties*. W oknie, które zostanie wyświetlone, wybieramy zakładkę *Run*. Następnie można wybrać odpowiedni serwer (rys. 4).



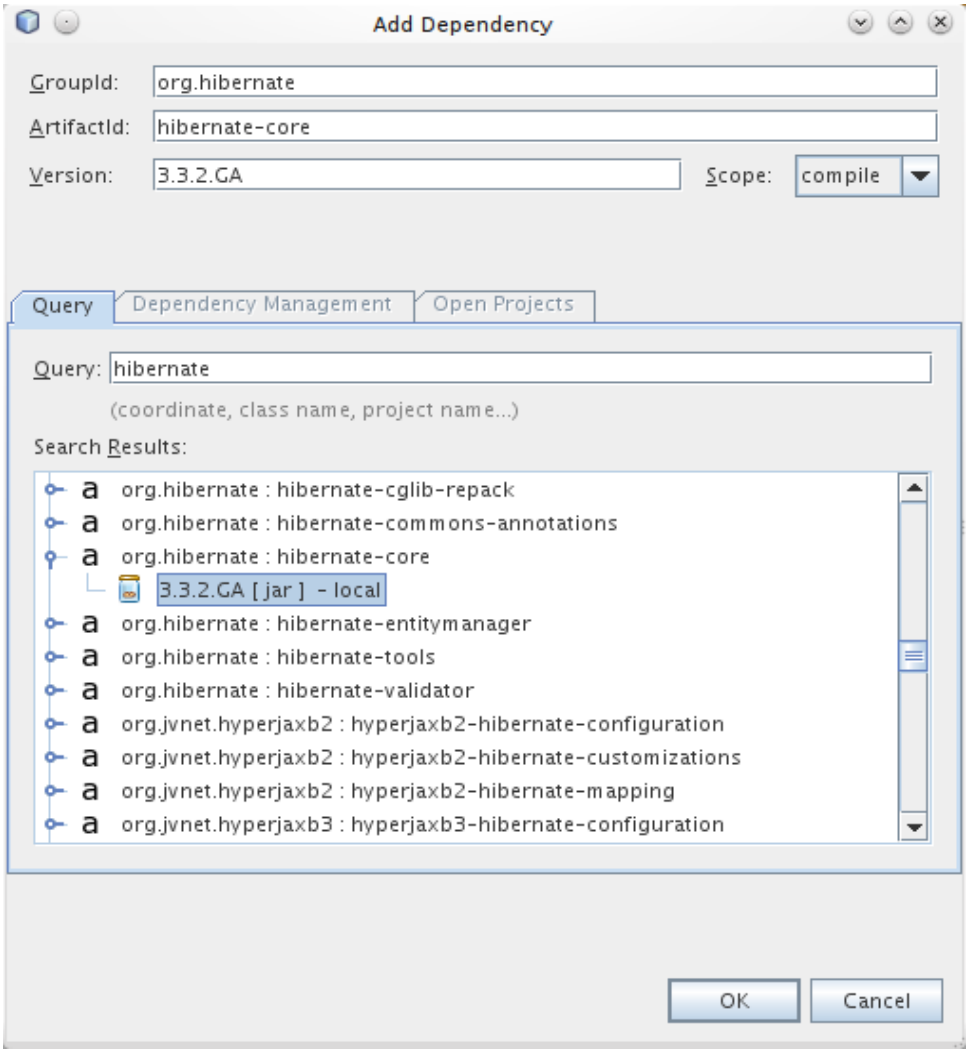
Rysunek 4: Wybór serwera

Wybierając więc Glassfisha, uruchamiamy aplikację. Efekt widoczny jest na rysunku 5. W tym momencie wiemy już, że projekt został przygotowany i można go rozwijać. Oczywiście, w razie potrzeby, można dodać także inne frameworki – sam Wicket tego nie zabrania. Można go połączyć z takimi frameworkami, jak Spring, Hibernate i wiele innych. Jeżeli planujemy więc stworzyć jakąś bazę danych, z której będzie korzystać nasza aplikacja, dorzućmy ten framework do projektu. Netbeans ma bardzo dobre wsparcie dla Mavena, więc wystarczy kliknąć prawym przyciskiem na pozycję *Libraries* w drzewie projektu i wybrać pozycję *Add Dependency*. Pojawi się okno dialogowe, przy pomocy którego łatwo dodamy różnego rodzaju zależności do naszego projektu – por. rysunek 6. Prawda, że proste?

Aplikacja domyślnie dostępna jest pod adresem *http://localhost:8080/HelloWorld-1.0-SNAPSHOT/*.



Rysunek 5: Pierwsze uruchomienie aplikacji



Rysunek 6: Dodajemy framework Hibernate do projektu

Sesje

Chcąc dodać obsługę sesji, musimy odpowiednio przygotować plik *WicketApplication.java*. Należy przeciążyć w nim metodą *newSession* oraz utworzyć plik *WicketApplicationSession.java*, który będzie odpowiadał za naszą sesję. Metodę *newSession* znaleźć można na listingu 1, a treść pliku *WicketApplicationSession.java* na listingu 2.

Listing 1. Przeciążona metoda *newSession*

```
@Override
public Session newSession(Request request, Response response) {
   return new WicketApplicationSession(WicketApplication.this,
request);
}
```

Listing 2. Treść pliku *WicketApplicationSession.java*

```
package com.helloworld;

import org.apache.wicket.Request;
import org.apache.wicket.protocol.http.WebApplication;
import org.apache.wicket.protocol.http.WebSession;

@SuppressWarnings("serial")
public final class WicketApplicationSession extends WebSession {
   /* Tutaj deklarujemy pola, które będą przechowywać wartości w
      czasie trwania sesji, np.:       */
   private boolean zalogowany;
   protected WicketApplicationSession(WebApplication application,
Request request) {
      super(request);
      zalogowany = false;
    }
   public void setZalogowany(boolean zalogowany) {
      this.zalogowany = zalogowany;
   }
   public boolean getZalogowany() {
      return zalogowany;
   }
}
```

Widzimy więc, że nie jest to skomplikowane. Przystąpimy teraz do utworzenia bardzo prostego formularza, który umożliwi użytkownikowi logowanie. Dodatkowo, utworzymy stronę powitalną dla użytkownika zalogowanego oraz plik lokalizacyjny.

Plik lokalizacyjny dla danej strony powinien mieć taką samą nazwę jak jej pliki, czyli np.

HomePage.properties – taki plik też utwórzmy. Jego zawartość widoczna jest na listingu 3.

Listing 3. HomePage.properties

```
login=Login:
password=Password:
```

Tworząc stronę Zalogowany, należy pamiętać o utworzeniu pary plików, czyli *Zalogowany.html* oraz *Zalogowany.java*. Klasa Zalogowany powinna oczywiście dziedziczyć po klasie *WebPage*. W pliku *Zalogowany.html* umieśćmy fragment z listingu 4. Pozwoli on na dynamiczną zmianę wartości, zależnie od tego, czy użytkownik jest zalogowany, czy też nie.

Listing 4. Strona *Zalogowany*

```
<body>
   <p wicket:id=”welcome”></p>
</body>
```

W pliku Zalogowany.java umieszczamy natomiast zawartość z listingu 5.

Listing 5. Klasa *Zalogowany*

```
public class Zalogowany extends WebPage {
   private Label welcome;
   public Zalogowany(final PageParameters parameters) {
      String welcomeText = "";
      WicketApplicationSession session =(WicketApplicationSession)
getSession();
      if (session.getZalogowany() == true)
         welcomeText = "Witaj, Tajniak!";
      else
         welcomeText = "Nie masz uprawnień do przeglądania tej
strony! Znajdź sobie inną.";
      welcome = new Label("welcome", new Model(welcomeText));
      add(welcome);
    }
}
```

Ponadto, uzupełnijmy klasę *WicketApplication* o przeciążoną metodę *init* tak, jak na listingu 6. Mapuje ona klasę adres do klasy *Zalogowany* na bardziej przyjazny, ponieważ możemy użyć adresu w postaci *http://localhost/HelloWorld/zalogowany*.

Listing 6. Metoda *init*

```
public void init() {
   super.init();
   mountBookmarkablePage("/zalogowany",
      com.helloworld.Zalogowany.class);
}
```

Przechodzimy teraz do tworzenia formularza. W pliku *HomePage.html* nie ma większych

problemów – zaglądamy w listing 7. Trudniejszy w realizacji natomiast może być plik *HomePage.java* – listing 8.

Listing 7. Plik *HomePage.html*

```
<form wicket:id=”loginForm”>
   <wicket:message key=”login”/><input wicket:id=”login”
        type=”text”/><br/>
   <wicket:message key=”password”/><input wicket:id=”password”
        type=”password”/><br/>
   <input wicket:id=”loginButton” type=”submit”/>
</form>
```

Warto jedynie zauważyć, że stosujemy znacznik *<wicket:message/>*, który przyjmuje parametr *key*, który to z kolei odwzorowuje klucz podany w pliku *HomePage.properties* – ten prosty zabieg zapewnia nam lokalizację komunikatów. Tworząc więc kolejny plik, np. *HomePage_pl_PL.properties* i podając w nim te same klucze, ale polskie odpowiedniki tekstów, uzyskamy zlokalizowaną aplikację.

Listing 7. Plik *HomePage.java*

```
public class HomePage extends WebPage {
   private Form form;
   public HomePage(final PageParameters parameters) {
      form = new LoginForm("loginForm");
      add(form);
   }
   class LoginForm extends Form {
      private TextField login;
      private PasswordTextField pass;
      private Button submit;
      private LoginForm(String id) {
         super(id);
         login = new TextField("login", new Model());
         pass = new PasswordTextField("password", new Model());
         submit = new Button("loginButton", new Model());
         add(login); add(pass); add(submit);
     }
     @Override
     public void onSubmit() {
        if (login.getInput().equals("tajniak") &&
                        pass.getInput().equals("tajniackie")) {
          WicketApplicationSession session =
                     (WicketApplicationSession) getSession();
          session.setZalogowany(true);
          setResponsePage(Zalogowany.class);
```

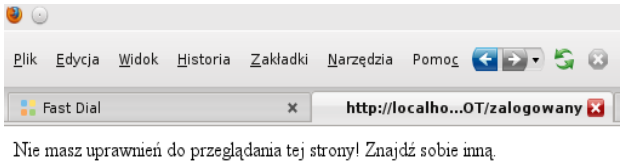
```
            }
            else {
                setResponsePage(getPage().getClass());
            }
        }
    }
}
```

W przypadku pliku *HomePage.java* istotny jest fakt, że formularz tworzymy dziedzicząc po klasie *Form*. Musimy implementować metodę *onSumbit*, jeśli chcemy podjąć jakieś akcje przy jego wysyłaniu – w tym przypadku sprawdzamy, czy użytkownik podał odpowiedni login i hasło. Jeśli tak, zapisujemy to odpowiednio w sesji o przekierowujemy użytkownika na stronę *Zalogowany*. W przeciwnym wypadku nie robimy nic – a raczej wracamy do aktualnej strony.

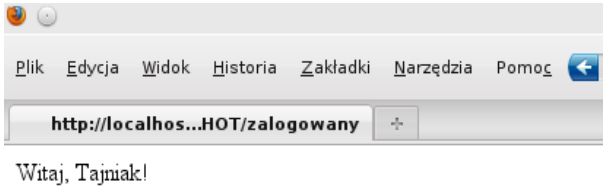
Przedstawiona metoda sprawdzania loginu i hasła oczywiście ma się nijak do realnego świata, bowiem wymagałoby to zastosowania co najmniej bazy danych, w której mamy zapisane takie dane, zamiast bezpośrednio w kodzie.

Po przejściu na stronę *Zalogowany* przed zalogowaniem się, ujrzymy informację, że użytkownik nie uzyskał do niej dostępu (rys. 7). Jeżeli natomiast wcześniej się zalogujemy, zawartość tej strony będzie inna (rys. 8). Wszystko to dzieje się za sprawą warunku, który zamieściliśmy w konstruktorze tej strony na listingu 5 – odwołujemy się tam do danych zapisanych w sesji.

Warto zwrócić też uwagę na postać adresu URL – po przekierowaniu na stronę dla użytkownika zalogowanego, nie zawiera on żadnych parametrów, które mogłyby by stanowić niebezpieczeństwo. Nie ma w nim np. identyfikatora sesji, co niestety często zdarza się w wielu aplikacjach webowych.



Rysunek 7: Użytkownik niezalogowany



Rysunek 8: Użytkownik zalogowany

Podsumowanie

Jak widzimy, Wicket jest bardzo lekkim i prostym w użyciu frameworkiem, który nadaje się do wielu zastosowań. Niniejszy artykuł z pewnością nie wyczerpuje tematu. Zachęcam więc do sięgnięcia po dodatkową literaturą i bliższe zapoznanie się z opisywanym projektem, gdyż jego możliwości są znacznie większe, a w połączeniu z innymi frameworkami – wręcz nieograniczone.

Literatura

Istnieje wiele pozycji książkowych, które pozwolą czytelnikowi zgłębić wiedzę dotyczącą Wicketa Obecnie jednak pozycje te dotyczą wersji od 1.3 w dół. Niestety książki nie powstają równolegle do wydawanych wersji, więc warto przeglądać także dokumentację dostarczaną przez twórców. Polecić mogę poniższe książki:

- Wicket in Action - Martijn Dashorst, Eelco Hillenius, wydawnictwo Manning Publications, rok 2008
- Enjoying Web Development with Wicket - Kent Tong, rok 2007
- Pro Wicket - Karthik Gurumurthy, wydawnictwo Apress, rok 2006