Jacek Matulewski
http://www.phys.uni.torun.pl/~jacek/

Tworzenie aplikacji Windows
Rozszerzenia serwerów WWW

Ćwiczenia

Toruń, 6 marca 2003

Najnowsza wersja tego dokumentu znajduje się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad4_www.pdf

Źródła opisanych w tym dokumencie programów znajdują się pod adresem
http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad4_www.zip

I. Spis treści

II. Aplikacje usług stron WWW

Uwaga!
Aby móc testować stworzone w tym skrypcie aplikację niezbędny jest serwer WWW. Idealnie do tego nadaje się serwer Microsoft dołączany do niektórych wersji Windows IIS (*Internet Information Services* – Internetowe usługi informacyjne) lub uznany serwer Apache (http://www.apache.org). Ponadto niezbędne jest Delphi/C++ Builder w wersji co najmniej 4 Professional.

Jest kilka standardów tworzenia aplikacji – rozszerzeń serwerów WWW. Historycznie pierwszym był CGI (*Common Gateway Interface*), który umożliwia zastosowania każdego języka programowania, w tym języków skryptowych, a jedynym warunkiem jest to, żeby system serwera potrafił je wykonać lub zinterpretować. Mają one jednak tę słabość, że każde wejście internauty na stronę wiąże z uruchomieniem osobnej instancji aplikacji. Powstały więc standardy ISAPI (dla serwera IIS Microsoftu) i NSAPI (dla serwera Netscape'a Enterprise Server), w których aplikacja w postaci biblioteki DLL uruchamiana jest wyłącznie raz i obsługuje wszystkie wejścia użytkowników sieci.
Z dydaktycznego punktu widzenia lepiej jest zacząć od CGI – pozwala on na podmienianie kolejnych wersji programu bez zatrzymywania i ponownego uruchamiania serwisu IIS lub Apache, co jest konieczne w przypadku ISAPI/NSAPI. Ponadto po napisaniu w Delphi aplikacji CGI niezwykle łatwo ją przekształcić w ISAPI/NSAPI.

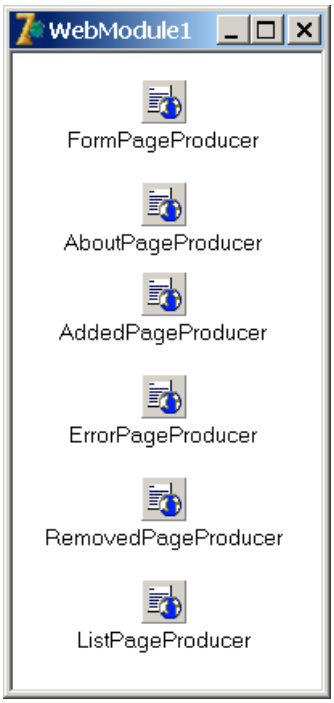
1. Lista mailowa (CGI)

Rozszerzenia serwerów WWW w standardzie CGI to aplikacje, które pobierając dane ze standardowego strumienia wejścia, na standardowy strumień wyjścia wysyłają kod HTML, który jest interpretowany przez przeglądarkę.

TPageProducer
Kluczowym komponentem wspomagającym tworzenie serwisów Webowych w Delphi i C++ Builderze jest `TPageProducer`, który z podanego szablonu dokumentu HTML przygotowuje gotowe kody stron WWW. W szablonie można umieszczać znaczniki typu <#NAZWA>, które za pomocą metody związanej ze zdarzeniem `OnHTMLTag` tego komponentu może być podmieniany na dowolny tekst lub kod HTML.
`TPageProducer` ma kilka pochodnych komponentów służących do prezentacji danych z baz danych (np. `TDataSetTableProducer`)

WebModule.Actions
Zasadniczą pracę wykonuje za nas klasa `TCGIApplication` obsługująca strumienie wejścia i wyjścia, odbiera zapytania HTTP, a do przeglądarki zwraca kod HTML. Jedyny obiekt tej klasy nazywa się tak jak w typowych projektach po prostu `Application`. W trakcie inicjacji aplikacji wywoływana jest metoda `Application.CreateForm`, która tworzony moduł typu `TWebModule` dziedziczący ze znanego skądinąd `TDataModule`. Zasadniczym dla projektów CGI są zdefiniowane w jego obrębie akcje. Związane z nimi względne adresy dostępu do aplikacji (własność TWebActionItem.PathInfo) zastępują przekazywanie argumentów do aplikacji CGI (tzn. zamiast pisać http://localhost/Project1.exe?hello napiszemy http://localhost/Project1.exe/hello).

Z tych dwóch elementów można już zbudować aplikację CGI. W menu New w zakładce New wybierzmy pozycję Web Server Application. W okienku dialogowym mamy do wyboru przedstawione wyżej typy rozszerzeń serwerów, a ponadto możemy wybrać projekt aplikacji współpracującej z serwerami Apache. My wybieramy „CGI Stand-alone executable". Nasz pierwszy projekt będzie składał się z kilku komponentów `TPageProducer`:



FormPageProducer

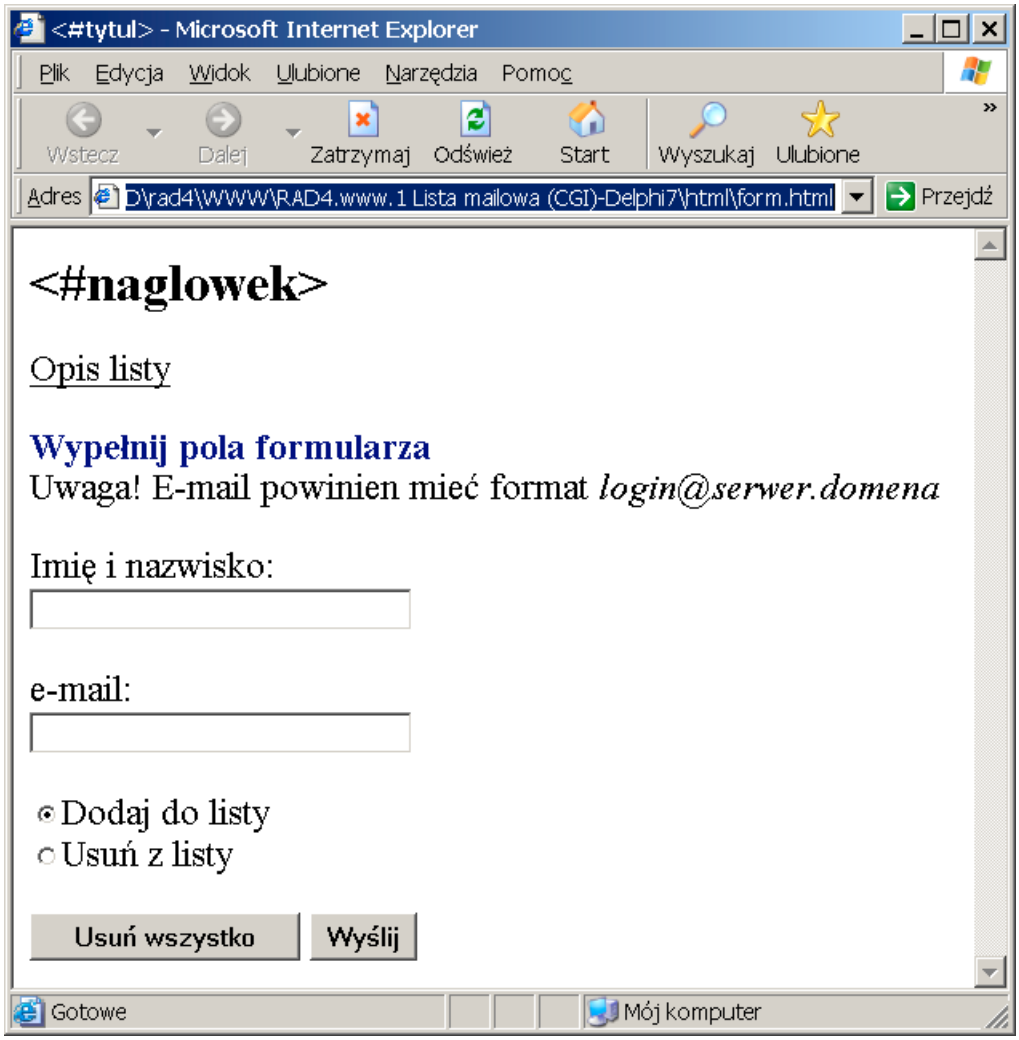
Do własności HTMLDoc wpisujemy szablon strony[1]:

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<A HREF="./about"><FONT COLOR="black">Opis listy</FONT></A><P>

<FONT COLOR="navy"><B>Wypełnij pola formularza</B></FONT><BR>
Uwaga! E-mail powinien mieć format <I>login@serwer.domena</I><P>

<FORM ACTION="./post" METHOD="post">

Imię i nazwisko:<BR>
<INPUT TYPE=TEXT NAME="nazwisko" SIZE=30><P>

e-mail:<br>
<INPUT TYPE=TEXT NAME="email" SIZE=30><P>

<INPUT TYPE=RADIO NAME="akcja" VALUE="dodaj" CHECKED>Dodaj do
listy</INPUT><BR>
<INPUT TYPE=RADIO NAME="akcja" VALUE="usun">Usuń z listy</INPUT><P>

<INPUT TYPE=RESET VALUE="Usuń wszystko">
<INPUT TYPE=SUBMIT VALUE="Wyślij">
</FORM>
```

[1] Bardzo skrótowy elementarz HTML można znaleźć na stronie http://www.phys.uni.torun.pl/~jacek/dydaktyka/puk/html.htm.

```
</BODY>
</HTML>
```

Alternatywnie można wczytać kod z pliku wskazując na niego we własności HTMLFile.
Kod szablonu zawiera znaczniki `<#tytul>` i `<#naglowek>`, które mogą być zamienione przez aplikację na dowolny łańcuch zawierający formatowania HTML. Interpretacja szablonu przez IE pokazana jest na rysunku:



Strona zawiera formularz pozwalający na wpisanie Imienia i nazwiska oraz adresu e-mail, a następnie na wybór akcji w liście mailowej (dodaj lub usuń) oraz przycisk służący do wysłania danych metodą POST. W zasadzie cała pozostała część aplikacji będzie reakcją na naciśnięcie tego przycisku.

Z tym komponentem `TPageProducer` związemy akcję o nazwie `hello`. Ponieważ ma to być akcja pojawiająca się na powitanie musimy ustawić jej własność `Default = True`. Należy podać względną ścieżkę `PathInfo = /hello`. oraz ustawić własność `Producer = FormPageProducer`.

Aby szablon zaczął wyglądać odpowiednio napiszmy metodę zdarzeniową związaną z `FormPageProducer.OnHTMLTag`:

```
procedure TWebModule1.AddedPageProducerHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
if TagString='tytul' then ReplaceText:='Jacek Matulewski - Lista mailowa';
if TagString='naglowek' then ReplaceText:='Lista mailowa';
end;
```

Metoda wywoływana jest za każdym razem, gdy przy przesyłaniu zawartości `HTMLDoc` lub `HTMLFile` do standardowego strumienia wyjścia program napotka strukturę *<#nazwa>*. Nazwa *nazwa* zostanie umieszczona w argumencie `TagString`. Zwracany przez nas zamiennik powinien zostać przypisany do referencji `ReplaceText`. Warto zauważyć, że znacznik może posiadać ogólną formę *<#nazwa parametr1=wartosc1 parametr2=wartosc2>* – lista parametrów przesyłana jest w liście łańcuchów `TagParams`.

Możemy uruchomić aplikację. Korzystanie z klawisza F9 na niewiele się zda, bo aplikacja tylko mignie nam przed oczami. Najlepiej otworzyć sobie intepreter poleceń MSDOS (command.com lub cmd.com w NT) i napisać `Project1.exe > test.html`, aby zapisać strumień wyjścia do pliku. Plik ten zawiera pełnoprawny dokument HTML, który można obejrzeć w przeglądarce.

Reakcja aplikacji na wysłanie (akcja post)

Ponieważ w formularzu adresem wysłania jest `./post` to przeglądarka postara się wczytać naszą aplikację z adresem względnym `post` np. `http://localhost/Scripts/Project.exe/post`. To będzie kluczowy moment dla jej działania. Należy dodać akcję o nazwie post (`MethodType = mtPost`, `PathInfo = /post`[2], własność `Producer` pozostaje puste). Kod decydujący co pokaże się w okienku przeglądarki umieszczamy w metodzie związanej ze zdarzeniem `OnAction` akcji:

```
procedure TWebModule1.WebModule1PostAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  email,nazwisko :String;
  email_correct,akcja :Boolean;
begin
email:=Request.ContentFields.Values['email'];
email_correct:=True;
nazwisko:=Request.ContentFields.Values['nazwisko'];
if Request.ContentFields.Values['akcja']='dodaj' then akcja:=True
                                                 else akcja:=False;

if akcja
  then //dodawanie do listy
  begin
    if (Pos('@',email)=0) or (Pos('.',email)=0) then email_correct:=False;
    if (Request.ContentFields.Values['nazwisko']<>'') and email_correct
      then
        begin
        Response.Content:=AddedPageProducer.Content;
        ZapiszDoPliku(email,nazwisko);
        end
      else Response.Content:=ErrorPageProducer.Content;
  end
  else //usuwanie z listy pierwszego napotkanego
  begin
    UsunZPliku(email,nazwisko);
    Response.Content:=RemovedPageProducer.Content;
  end;
end;
```

Jej struktura jest dość prosta. Na początku sprawdzane są poprawność podawanych danych i jeżeli wszystko jest w porządku w zależności od wyboru dodaj/usuń wywoływana jest metoda `ZapiszDoPliku()` lub `UsunZPliku()`. Równocześnie ustalana jest wartość `Response.Content` na komponenty `TPageProducer.Content`, który wysyłany jest do standardowego strumienia wyjścia. Przygotować musimy trzy szablony – reakcja na dodanie, usunięcie i błędnie wpisane dane. Jak zwykle w tym skrypcie pominięta została obsługa błędów działania aplikacji.

[2] Nie nazwa akcji, ale jej własność `Path` decyduje o tym która akcja będzie wywołana na skutek wywołania `http://localhost/Scripts/Project1.exe/post`

AddedPageProducer:

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<FONT COLOR="green"><B>Operacja zakończona pomyślnie.</B></FONT><BR>
Adres e-mail został dodany do listy.<P>

E-mail: <#email> (<#nazwisko>)

</BODY>
</HTML>
```

RemovedPageProducer:

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>


<FONT COLOR="green"><B>Operacja zakończona pomyślnie.</B></FONT><BR>
Adres e-mail został usunięty z listy.<P>

E-mail: <#email> (<#nazwisko>)

</BODY>
</HTML>
```

oraz

ErrorPageProducer:

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<FONT COLOR="red"><B>Błąd!</B></FONT></BR>
Niewypełnione pole "Imię i Nazwisko" lub adres e-mailowy nie posiada
właściwego formatu (<I>login@serwer.domena</I>)<P>

</BODY>
</HTML>
```

We wszystkich tych szablonach wykorzystywane są znaczniki `<#tytul>` i `<#naglowek>`. Dla prostoty można podłączyć do ich zdarzenia `OnHTMLTag` przygotowaną wcześniej, ale uzupełnioną metodę:

```
procedure TWebModule1.AddedPageProducerHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
if TagString='tytul' then ReplaceText:='Jacek Matulewski - Lista mailowa';
if TagString='naglowek' then ReplaceText:='Lista mailowa';
if TagString='email' then ReplaceText:=Request.ContentFields.Values['email'];
if TagString='nazwisko' then ReplaceText:=Request.ContentFields.Values['nazwisko'];
end;
```

Łańcuchy „Imię i nazwisko” oraz „e-mail” pobierane są wprost ze strony zawierającej formularz, a więc strony zgłaszającej zapytanie HTTP – umieszczonej w argumencie `Request`.

Pozostaje napisać metody operujące na pliku (nazwa tego pliku umieszczone są w stałej `const datafilename='Project1.txt';`). Implementujemy dodanie zapisu do pliku

```
procedure TWebModule1.ZapiszDoPliku(email,nazwisko :String);
var tf: TextFile;
begin
AssignFile(tf,datafilename); //W TurboPascalu bylo Assign, ale jest taka metoda w VCL
if FileExists(datafilename)
  then Append(tf)
  else Rewrite(tf);
WriteLn(tf,email+' ('+nazwisko+')');
CloseFile(tf); //W TurboPascalu bylo Open, ale np. forma ma taka metode
end;
```

i jego usunięcie

```
procedure TWebModule1.UsunZPliku(email,nazwisko :String);
var
  tf: TextFile;
  linia,szukany :String;
  bufor :TStringList;
begin
szukany:=email+' ('+nazwisko+')';
linia:='';
bufor:=TStringList.Create;
bufor.Clear;

AssignFile(tf,datafilename);
if not FileExists(datafilename)
  then Exit
  else
    begin
    Reset(tf);
    while not Eof(tf) do
      begin
      ReadLn(tf,linia);
      //przy okazju wyrzucamy puste
      if (linia<>szukany) and (linia<>'') then bufor.Add(linia);
      end;
    CloseFile(tf);
    bufor.SaveToFile(datafilename);
    end;
end;
```

Procedury te należy jeszcze zadeklarować w sekcji `interface`. To kończy zasadniczą część aplikacji. Można jeszcze dodać akcję `list` (`PathInfo = /list`, `Producer = ListPageProducer`) i związany z nią komponent `TPageProducer` pokazujący aktualną listę zapisanych. Jej własność `HTMLDoc` zawiera jedynie początek kodu.

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<B>Lista adresów e-mail:</B><BR>
```

(również tu wykorzystamy tą samą metodę w zdarzeniu `OnHTMLTag`). Reszta strony zostanie dodana w metodzie związanej ze zdarzeniem `OnAction` akcji `list`:

```
procedure TWebModule1.WebModule1ListAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var
  tf: TextFile;
  linia :String;
begin
AssignFile(tf,datafilename);
if FileExists(datafilename) then
  begin
  Reset(tf);
  while not Eof(tf) do
    begin
    ReadLn(tf,linia);
    ListPageProducer.HTMLDoc.Add(linia+'<BR>')
    end;
  CloseFile(tf);
  end
  else ListPageProducer.HTMLDoc.Add('<I>Lista jest pusta</I>');
ListPageProducer.HTMLDoc.Add('</BODY></HTML>');

Response.Content:=ListPageProducer.Content;
end;
```

Jak widać do kodu dodawana jest zawartość pliku z uwzględnieniem znaczników końca linii `<BR>` oraz zakończenie dokumentu `</BODY></HTML>`. W razie braku pliku w kodzie umieszczany jest odpowiedni napis.

Umieszczanie na serwerze
Działającą aplikację umieszczamy w odpowiednim katalogu serwera, w której mogą być wykonywane programy (w IIS jest to katalog Inetpub\Scripts, a Apache katalog CGI). Wywołanie naszego serwisu może być następujące (dla IIS):

```
<A HREF="http://localhost/scripts/Project1.exe/">Lista mailowa</A>
```

a listy zapisanych adresów:

```
<A HREF="http://localhost/scripts/Project1.exe/list">Lista</A><BR>
```

W pierwszym przypadku dodanie „/” na końcu adresu jest zabiegiem oszukującym programy wspomagające ściąganie plików z Internetu (GetRight, FlashGet, Download Accelerator itp.), które reagują na pliki z rozszerzeniem exe próbując je ściągać. Równoznacznym do pierwszego wywołania jest jawne wskazanie na akcję `hello` przez dodanie „/hello” na końcu adresu.

Zadanie
W formularzu głównym znajduje się link do opisu strony. Zdefiniować akcję ze względną ścieżką `/about`, który będzie związany z komponentem `TPageProducer` zawierającym odpowiedni opis.

2. Projekty zaliczeniowe (CGI + bazy danych, ADO)

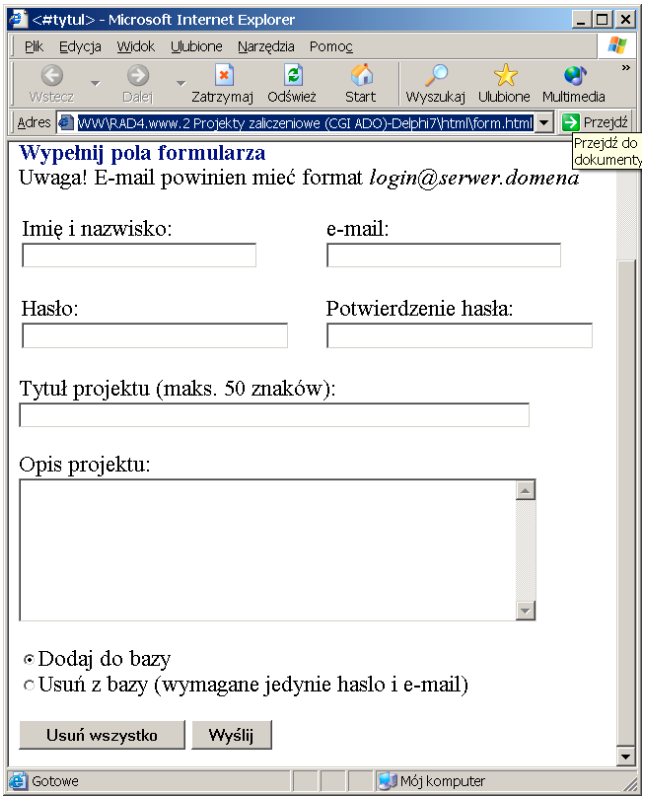
Rozbudujemy teraz nasz projekt w taki sposób, żeby poza imieniem i nazwiskiem oraz adresem e-mail przechowywał dodatkowe informacje np. temat i opis pracy zaliczeniowej. Ponadto dane, których będzie teraz znacznie więcej niż poprzednio przechowywać będziemy w znacznie wygodniejszy sposób – umieścimy je w bazie danych Microsoft Access, z którą będziemy się łączyć za pośrednictwem mechanizmu systemu Windows o nazwie ADO (więcej np. w RAD2).

W nowym projekcie rozbudujemy formularz dodając okienka edycyjne pozwalające na wpisanie hasła i jego potwierdzenia, tytułu projektu i jego opisu. Pełen kod strony zawierającej formularz może być następujący:

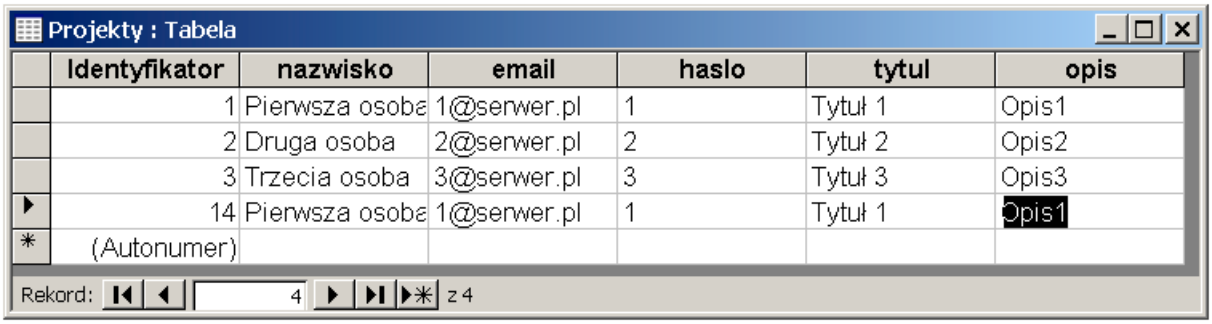
```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<A HREF="./about"><FONT COLOR="black">Instrukcja</FONT></A>  
<A HREF="./data"><FONT COLOR="black">Zawartość bazy danych</FONT></A><P>

<FONT COLOR="navy"><B>Wypełnij pola formularza</B></FONT><BR>
Uwaga! E-mail powinien mieć format <I>login@serwer.domena</I><P>

<FORM ACTION="./post" METHOD="post">

<TABLE>
<TR>
<TD>
Imię i nazwisko:<BR>
<INPUT TYPE=TEXT NAME="nazwisko" SIZE=30><P>
</TD>

<TD WIDTH=30></TD>

<TD>
e-mail:<br>
<INPUT TYPE=TEXT NAME="email" SIZE=30><P>
</TD>
</TR>

<TR>
<TD>
Hasło:<br>
<INPUT TYPE=PASSWORD NAME="haslo" SIZE=30><P>
</TD>

<TD></TD>

<TD>
Potwierdzenie hasła:<br>
<INPUT TYPE=PASSWORD NAME="haslo2" SIZE=30><P>
</TD>
</TR>
</TABLE>
```

```
Tytuł projektu (maks. 50 znaków):<br>
<INPUT TYPE=TEXT NAME="tytul" SIZE=70><P>

Opis projektu:<br>
<TEXTAREA NAME="opis" ROWS=7 COLS=50></TEXTAREA><P>

<INPUT TYPE=RADIO NAME="akcja" VALUE="dodaj" CHECKED>Dodaj do
bazy</INPUT><BR>
<INPUT TYPE=RADIO NAME="akcja" VALUE="usun">Usuń z bazy</INPUT> (wymagane
jedynie haslo i e-mail)<P>

<INPUT TYPE=RESET VALUE="Usuń wszystko">
<INPUT TYPE=SUBMIT VALUE=" Wyślij ">
</FORM>

</BODY>
</HTML>
```

Jego interpretacja jest przedstawiona na rysunku.



Konieczne są zmiany związane z obsługą bazy danych: dodawanie i usuwanie rekordu zastąpi operacje na pliku, a komponent `TDataSetTableProducer` jest doskonałym sposobem udostępnienia zawartości tabeli z danymi. Podobnie jak w standardowym komponencie bazodanowym `TDBGrid` można wybrać prezentowane kolumny. Jego połączenie z bazą danych jest również bardzo podobne do `TDBGrid`, nie musi być jednak zapośredniczone komponentem `TDataSource`. Do projektu dodajemy komponenty `ADODataSet1` i `DataSetTableProducer1`. Konfiguracja ich własności przedstawiona jest w tabelach.



Należy także przygotować bazę danych zawierającą tabelę „Projekty” z sześcioma kolumnami: automatycznie utworzony Identyfikator, nazwisko, email, haslo, tytul i opis. Plik bazy danych powinien nazywać się Projekt2.mdb.

ADODataSet1

Własność	Ustalona wartość	Uwagi
`ConnectionString`	`Provider=Microsoft.Jet.OLEDB.4.0;Data Source=Project2.mdb;Persist Security Info=False`	Łańcuch można z łatwością zbudować edytorem własności
`CommandType`	`cmdText`	Podamy polecenie SQL
`CommandText`	`select * from Projekty`	W bazie danych Projekt2.mdb korzystamy z tabeli Projekty. Program wymaga istniejącej tabeli.

DataSetTableProducer1

Własność	Ustalona wartość
`DataSet`	`ADODataSet1`
`Columns`	Należy uwzględnić kolumny `nazwisko`, `email`, `tytul`

`Header`	`<HTML>`
	`<HEAD>`
	`<TITLE>Jacek Matulewski - Projekty zaliczniowe</TITLE>`
	`</HEAD>`
	
	`<BODY>`
	
	`<H2>Projekty zaliczeniowe:</H2>`

`Footer`	`<P><A HREF="./"><FONT COLOR="black">Powrót</FONT></A>`
	
	`</BODY>`
	`</HTML>`

`MaxRows`	`-1` (dzięku temu ilość wierszy jest nieograniczona)

W łańcuchu ConnectionString komponentu `ADODataSet1` podano nazwę danych bez ścieżki dostępu, co oznacza, że powinna znajdować się w katalogu roboczym aplikacji. Analogicznie do poprzedniego projektu przygotowujemy metodę zdarzeniową akcji `Post`. Tym razem nieco więcej uwagi trzeba było poświęcić obsłudze błędów.

```
procedure TWebModule1.WebModule1PostAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
```

```
var
  nazwisko,email,haslo,haslo2,tytul,opis :String;
  email_correct,haslo_zgodne,akcja :Boolean;
begin
email:=Request.ContentFields.Values['email'];
email_correct:=True;
nazwisko:=Request.ContentFields.Values['nazwisko'];
haslo:=Request.ContentFields.Values['haslo'];
haslo2:=Request.ContentFields.Values['haslo2'];
tytul:=Request.ContentFields.Values['tytul'];
opis:=Request.ContentFields.Values['opis'];

if Request.ContentFields.Values['akcja']='dodaj' then akcja:=True
                                                 else akcja:=False;

//Bledy
if Request.ContentFields.Values['nazwisko']='' then
                               komunikat:='Wypełnij pole "Imię i nazwisko"';
if Request.ContentFields.Values['tytul']='' then
                               komunikat:='Wypełnij pole "Tytuł projektu"';
if Request.ContentFields.Values['opis']='' then
                               komunikat:='Wypełnij pole "Opis projektu"';

if (Pos('@',email)=0) or (Pos('.',email)=0) then
  begin
  email_correct:=False;
  komunikat:='Zly format e-maila.';
  end;

haslo_zgodne:=((haslo=haslo2) and (haslo<>''));
if not haslo_zgodne then
            komunikat:='Brak hasla lub potwierdzenie nie jest identyczne';

//Akcja
if akcja then

  begin //dodawanie do listy
  if (Request.ContentFields.Values['nazwisko']<>'')
     and email_correct
     and haslo_zgodne
     and (Request.ContentFields.Values['tytul']<>'')
     and (Request.ContentFields.Values['opis']<>'')
     then //dodawanie do listy, jeżeli nie ma błędów
       begin
       if ZapiszDoBazy(nazwisko,email,haslo,tytul,opis)
         then Response.Content:=AddedPageProducer.Content
         else begin
              komunikat:='Identyczny rekord istnieje.';
              Response.Content:=ErrorPageProducer.Content;
              end;
       end
     else //dodawanie do listy, blad danych
       Response.Content:=ErrorPageProducer.Content
  end

   else

   begin //usuwanie z listy
   if email_correct and haslo_zgodne then //usuwanie z listy, email i haslo
     begin
       if UsunZBazy(email,haslo)
```

```
          then Response.Content:=RemovedPageProducer.Content
          else begin
               komunikat:='W bazie danych nie ma rekordu
                           zawierającego podany e-mail i haslo';
               Response.Content:=ErrorPageProducer.Content
               end;
     end
     else
     begin
     komunikat:='Niewypełnione pole "e-mail" lub brak zgodności hasła.';
     Response.Content:=ErrorPageProducer.Content;
     end;
   end;

end;
```

Należy zastąpić obsługę plików wykorzystaniem bazy danych. Oto metody `ZapiszDoBazy()` i `UsunZBazy()`:

```
function TWebModule1.ZapiszDoBazy(nazwisko,email,haslo,tytul,opis :String)
:Boolean;
begin
Result:=True;
ADODataSet1.Open;
if ADODataSet1.Locate('email;tytul', VarArrayOf([email,tytul]),
                      [loCaseInsensitive]) then
  begin
  Result:=False;
  Exit;
  end;
//nil dla automatycznego identyfikatora
ADODataSet1.AppendRecord([nil,nazwisko,email,haslo,tytul,opis]);
ADODataSet1.Close;
end;


function TWebModule1.UsunZBazy(email,haslo :String) :Boolean;
begin
Result:=False;
ADODataSet1.Open;
ADODataSet1.Filter:='email = '+QuotedStr(email)+
                                          ' AND haslo='+QuotedStr(haslo);
ADODataSet1.Filtered:=True;
while not ADODataSet1.Eof do //jezeli sie powtarzaja
  begin
  ADODataSet1.Delete;
  Result:=True;
  end;
ADODataSet1.Filtered:=False;
ADODataSet1.Close;
end;
```

W pierwszej metodzie metodą `TADODataSet.Locate` sprawdzam, czy w tabeli jest rekord posiadający ten sam e-mail[3] i tytuł projektu zaliczeniowego. Jeżeli istnieje funkcja natychmiast zwraca wartość `False`. Podobnie w drugiej metodzie poszukuję rekordów do skasowani. Tym razem korzystam z prostej sztuczki pozwalającej sprawdzić nie tylko czy interesujący nas zapis już istnieje w bazie, ale także sprawdza czy się powtarzają, co jest możliwe pomimo weryfikowania danych przy zapisie do bazy i kasuje wszystkie. Wystarczy ustalić i uaktywnić odpowiedni filtr sprawdzający e-mail i hasło.

[3] Praktyka mówi, że lepiej sprawdzać e-mail niż Imię i Nazwisko, bo format zapisu jest dokładnie określony, nie ma możliwości zamiany miejscami wyrazów lub dopisania niepotrzebnej spacji.

Aby dodać i usunąć rekord z bazy danych można posłużyć się odpowiednimi metodami komponentu `ADODataSet1` a mianowicie `AppendRecord` i `Delete`.

Aby dowiedzieć się dalszych szczegółów polecam analizę źródła projektu umieszczonego w pliku http://www.phys.uni.torun.pl/~jacek/dydaktyka/rad/rad4_www.zip.

3. Przekształcenie projektu do standardu ISAPI/NSAPI

Przekształcenie gotowego projektu CGI do standardu ISAPI/NSAPI okazuje się dziecinnie proste. Tworzymy nowy projekt Web Service Application wybierając tym razem ISAPI/NSAPI. Zapisujemy na dysk, a następnie podmieniamy wszystkie pliki związane z `WebModule1` (pliki z rdzeniem `Unit1.*`). I gotowe.

Działanie takiego rozszerzenia serwera WWW jest inne niż w przypadku CGI. Aplikacja dla każdego klienta strony WWW uruchamia osobny wątek, zamiast oddzielnej instancji aplikacji. To pozwala zaoszczędzić zasoby systemowe serwera.

Ponieważ serwis typu ISAPI kompilowany jest do postaci biblioteki DLL katalogiem roboczym będzie katalog wywołującego ją serwisu IIS lub innego serwera, a więc najprawdopodobniej katalog systemowy (prawdopodobnie WINDOWS\SYSTEM32). Skoro nie podaliśmy ścieżki dostępu do bazy danych, to w tym katalogu należy umieścić plik bazy danych (nazwa pliku Project2.mdb – co wynika z tego, że skopiowaliśmy pliki z poprzedniego paragrafu).

III. Rejestrowanie uruchomień, wykorzystanie „ciasteczek”, informacje o zdalnym komputerze

W tej części znajduje się kilka usprawnień do projektu CGI z paragrafu II.2[4].

1. Pobieranie podstawowych informacji o zdalnym komputerze (kliencie)

Dodajmy do strony z formularzem (`FormPageProducer.HTMLDoc`) informacje o komputerze kliencie:

```
<FONT COLOR="navy"><B>Informacje o Twoim komputerze</B></FONT><BR>
Komputer: <#remotehost><BR>
Przeglądarka: <#browser><P>
```

W konsekwencji do metody `TWebModule1.AddedPageProducerHTMLTag`, której adres przechowuje zdarzenie `FormPageProducer.OnHTMLTag` należy dopisać wytłuszczoną część poniższego kodu:

```
procedure TWebModule1.AddedPageProducerHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
var remotehost :String;
begin
if TagString='tytul' then ReplaceText:='Jacek Matulewski - Projekty
zaliczeniowe';
if TagString='naglowek' then ReplaceText:='Projekty zaliczeniowe';

if TagString='email' then
ReplaceText:=Request.ContentFields.Values['email'];
if TagString='nazwisko' then
ReplaceText:=Request.ContentFields.Values['nazwisko'];
if TagString='temat' then
ReplaceText:=Request.ContentFields.Values['tytul'];

if TagString='komunikat' then ReplaceText:=komunikat;

//Informacje o zdalnym komputerze
if TagString='remotehost' then
    begin
    remotehost:=Request.RemoteHost;
    if remotehost<>Request.RemoteAddr then
            remotehost:=remotehost+' ('+Request.RemoteAddr+')';
    ReplaceText:=remotehost;
    end;
if TagString='browser' then ReplaceText:=Request.UserAgent;
end;
```

2. Rejestrowanie uruchomień programu

Uzupełnijmy nasz program CGI o możliwość rejestrowania uruchomień w pliku tekstowym „Projekt2b.log”. Nie ma to w zasadzie nic wspólnego z programowaniem sieciowym pod Windows, ale z dwóch względów jest pożyteczne. Po pierwsze ułatwia pracę administratorowi serwisu, a po drugie pozwoli się nam przekonać jak w rzeczywistości działa nasz program.

[4] Na wykorzystanie projektu CGI zamiast ISAPI zdecydowałem się ze względu na łatwość jego debugowania – do podmienienia pliku exe na nowszą wersję nie potrzeba w tym przypadku restartować usługi IIS.

Dodajmy do obiektu metodę o nazwie `SaveToLogFile` przyjmującą przez głowę łańcuch z opisem zdarzenia. Metoda zapisuje ten łańcuch do pliku o nazwie „Project2b.log" dodając aktualną datę i czas:

```
procedure TWebModule1.SaveToLogFile(eventstr :AnsiString);
const
  LogFileName='Project2b.log';
var
  LogFile: TextFile;
begin
AssignFile(LogFile,LogFileName);
if FileExists(LogFileName) then Append(LogFile) else Rewrite(LogFile);
WriteLn(LogFile,eventstr+': '+DateToStr(Date)+', '+TimeToStr(Time));
CloseFile(LogFile);
end;
```

Do otwarcia i operacji na plikach korzysta się ze standardowych poleceń Object Pascala. Warto zauważyć, że ze względu na konflikt z metodami `TForm.Close` i `TForm.Assign` zmieniona została nazwa odpowiednich poleceń dotyczących plików na `CloseFile` i `AssignFile`.

Pozostaje jedynie „powtykać" wywołanie tej metody do kluczowych miejsc w programie. Rozpoczęcie i zakończenie programu można wychwycić za pomocom zdarzeń `TWebModule1.OnCreate` i `OnDestroy`:

```
procedure TWebModule1.WebModuleCreate(Sender: TObject);
begin
SaveToLogFile('Start');
end;

procedure TWebModule1.WebModuleDestroy(Sender: TObject);
begin
SaveToLogFile('Stop');
end;
```

Wywołania `SaveToLogFile` umieszczamy także w innych metodach. Przy operacjach na bazie zapisywaną informację uzupełniamy e-mailem, co powinno ułatwić identyfikację wielu zapisów przy równocześnie uruchomionych sesjach.

Metody `TWebModule1`	**Polecenie**

`AddedPageProducerHTMLTag`	`SaveToLogFile('RemoteHost='+remotehost);`
	
	polecenie należy dodać w obrębie warunku
	`if TagString='remotehost' then`
	żeby uniknąć wielokrotnego zapisu

`ZapiszDoBazy`	`SaveToLogFile('ZapiszDoBazy ('+email+')');`
`UsunZBazy`	`SaveToLogFile('UsunZBazy ('+email+')');`

Jeżeli rejestracja zdarzeń ma być pożyteczna dla administratorów niezbędne jest również dodanie rejestracji błędów. Najwygodniej jest dodawanie wpisów umieścić w metodzie `TWebModule1.OnException`, która wywoływana jest w przypadku wystąpieniu błędu i związanego z nim wyjątku:

```
procedure TWebModule1.WebModuleException(Sender: TObject; E: Exception;
  var Handled: Boolean);
begin
SaveToLogFile('Blad! ('+E.Message+')');
end;
```

Obiekt `E` typu `Exception` jest typem przekazującym informacje o wyjątkach. Z naszego punktu widzenia najważniejszą własnością tego obiektu jest łańcuch `Message` zawierający opis błędu. Rejestrowane są oczywiście tylko błędy programu tj. wówczas gdy pojawi się wyjątek. Błędy typu „Identyczny rekord w bazie już istnieje" nie są w ogóle rejestrowane, bo nie nastąpiła modyfikacja bazy danych, ani nic nie zagraża stabilności aplikacji.

3. Cookies – przechowywanie danych na komputerze kliencie

Przykładowy plik zawierający przygotowane w poprzednim paragrafie wpisy znajduje się poniżej:

```
Start: 2002-12-18, 13:32:24
RemoteHost=158.75.5.7: 2002-12-18, 13:32:24
Stop: 2002-12-18, 13:32:24

Start: 2002-12-18, 13:33:32
ZapiszDoBazy (jacek@phys.uni.torun.pl): 2002-12-18, 13:33:34
Stop: 2002-12-18, 13:33:34
```

Z jego analizy wynika, że program został uruchomiony dwukrotnie. Pierwszy raz w celu wyprodukowania kodu formularza (przy tej okazji zapisywany jest adres komputera klienta) i kolejny raz przy wysłaniu formularza (w tym przypadku efektem jest dopisanie nowego rekordu do bazy danych). Oba uruchomienia programu są całkowicie niezależne, o czym świadczy obecność zapisów „Start" i „Stop", a więc aplikacja nie ma możliwości przechowania zmiennych ustalonych przy pierwszym uruchomieniu inaczej niż przez zapisanie ich do pliku. W przypadku naszej aplikacji w dotychczasowym kształcie przechowywanie danych nie jest jeszcze konieczne. Pierwsze jej uruchomienie owocuje powstaniem formularza. W tym etapie użytkownik nie podaje programowi żadnych danych. Dopiero po wpisaniu danych do formularza i ich wysłaniu program reaguje tworząc kod HTML odpowiedniej strony informującej o dodaniu/usunięciu danych z bazy danych lub odpowiedni komunikat błędu.

Z naszego punktu widzenia interesujące jest drugie uruchomienie aplikacji – reakcja na wysłanie danych wpisanych do formularza. Ponieważ w formularzu wybraliśmy metodę przesłania zapytania w następujący sposób

```
<FORM ACTION="./post" METHOD="post">
```

Po jego wysłaniu wywołany zostanie nasz program `Project2c.exe/post`, co oznacza, że wywołana zostanie akcja `Post` (dzięki zadeklarowanej własnością `Post.PathInfo='/post'`) i związana z nią (dokładniej z jej zdarzeniem `OnAction`) metoda `WebModule1PostAction`. W obrębie tej metody widoczna jest strona formularza jako `Request` i tworzona przez nas strona jako `Response`. Zawartość tworzonej strony może być wybierana np. w zależności od informacji przeczytanych w `Request`:.

```
if ZapiszDoBazy(nazwisko,email,haslo,tytul,opis)
          then Response.Content:=AddedPageProducer.Content
          else begin
               komunikat:='Identyczny rekord istnieje.';
               Response.Content:=ErrorPageProducer.Content;
               end;
```

W zależności od wartości zwróconej przez metodę `ZapiszDoBazy` wyświetlona zostanie informacja o dodaniu do bazy lub o błędzie. Nie istnieje więc problem przechowywania danych, bo cała obróbka danych jest wykonywana wyłącznie podczas drugiego uruchomienia.

Problem pojawia się wówczas, gdy między formularzem, a działaniem na bazie danych zechcemy dodać stronę np. żądającą od internauty sprawdzenia dodawanych informacji. Wówczas po wysłaniu formularza zostanie wywołana akcja Post, która widzi informacje z formularza i wyświetli wpisane informacje oraz zawierać będzie formularz pozwalający na powrót do pierwszego formularza lub zapis informacji do bazy. W drugim przypadku wywołana zostanie kolejna akcja, ale jej metoda w argumencie `Request` zawierać będzie nie pierwotny formularz, a stronę z pytaniem o potwierdzenie. Jednym słowem nie będzie widziała wpisanych, przez użytkownika danych.

Problemu nie można rozwiązać wykorzystując globalne zmienne w aplikacji ponieważ przejście od formularza do strony z prośbą o potwierdzenie i przejście od tej strony do strony z informacją o zapisaniu do bazy lub o błędzie związane jest z dwoma zupełnie niezależnymi uruchomieniami programu. Wszelkie zmienne są ponownie inicjowane.

Jedynym sposobem jest zapisanie danych w pliku. Nie może to być plik serwera, bo kilka jednocześnie uruchomionych sesji mogłoby pomieszać dane. Najlepiej, i najbezpieczniej z punktu widzenia ochrony danych, byłoby zapisać potrzebne nam informacje w pliku na komputerze terminala. Służą do tego pliki cookies (ciasteczka – prezenty od serwera)[5].

Do projektu dodajemy dodatkowy komponent klasy `TPageProducer` związany ze stroną potwierdzenia, którą umieścimy pomiędzy formularzem, a stroną informującą o dodaniu/usunięciu rekordu do bazy lub błędzie. Dokładnie rzecz biorąc należy zastąpić wywołanie metod `DodajDoBazy`/`UsunZBazy` w metodzie `WebModule1PostAction` (związanej z `OnAction` akcji `Post`) zapisem danych do cookie i przypisaniem do `Response.Content` zawierającej potwierdzenie. Zmodyfikowana część metody jest następująca:

```
//Akcja
if akcja then

  begin //dodawanie do listy
  if (Request.ContentFields.Values['nazwisko']<>'')
     and email_correct
     and haslo_zgodne
     and (Request.ContentFields.Values['tytul']<>'')
     and (Request.ContentFields.Values['opis']<>'')
     then
       begin
       ZapiszCookies(nazwisko,email,haslo,tytul,opis,'dodaj');
       Response.Content:=ConfirmPageProducer.Content;
       {
       if ZapiszDoBazy(nazwisko,email,haslo,tytul,opis)
         then Response.Content:=AddedPageProducer.Content
         else begin
              komunikat:='Identyczny rekord istnieje.';
              Response.Content:=ErrorPageProducer.Content;
              end;
       }
       end
     else //dodawanie do listy, blad danych
       Response.Content:=ErrorPageProducer.Content;
  end

   else

   begin //usuwanie z listy
   if email_correct and haslo_zgodne then
     begin
       ZapiszCookies(nazwisko,email,haslo,tytul,opis,'usun');
       Response.Content:=ConfirmPageProducer.Content;
       {
       if UsunZBazy(email,haslo)
          then Response.Content:=RemovedPageProducer.Content
          else begin
               komunikat:='W bazie danych nie ma
                              rekordu zawierającego podany e-mail i haslo';
               Response.Content:=ErrorPageProducer.Content
               end;
       }
     end
```

[5] Co ciekawe, dokładnie z tego samego mechanizmu można skorzystać nie tylko do tworzenia „stanu wewnętrznego" w trakcie sesji, ale także, jeżeli chcemy zapamiętać informacje o logowaniu pomiędzy dwoma, nawet odległymi czasowo sesjami. Jeżeli do pliku cookies zostaną zapisane login i hasło, następnego dnia użytkownik, który korzysta z tego samego komputera i profilu w Windows może oczekiwać, że po wejściu na stronę w odpowiednik okienkach formularza pojawi się jego e-mail i hasło.

```
    else
    begin
    komunikat:='Niewypełnione pole "e-mail" lub brak zgodności hasła.';
    Response.Content:=ErrorPageProducer.Content;
    end;
  end;
```

Obiekt, niech nazywa się na przykład ConfirmPageProducer, we właściwości HTMLDoc zawiera szablon HTML:

```
<HTML>
<HEAD>
<TITLE><#tytul></TITLE>
</HEAD>

<BODY>
<H2><#naglowek></H2>

<FONT COLOR="navy"><B>Sprawdź dane:</B></FONT></BR>

<TABLE BORDER=0>
<TR><TD VALIGN="top" WIDTH=200>Imię i nazwisko: <TD><#nazwisko></TR>
<TR><TD VALIGN="top">E-mail: <TD><#email></TR>
<TR><TD VALIGN="top">Tytuł projektu: <TD><#temat></TR>
<TR><TD VALIGN="top">Opis projektu: <TD><#opis></TR>
</TABLE>
<P>

<FORM ACTION="./update" METHOD="post">
<INPUT TYPE=SUBMIT VALUE=" <#akcja> ">
</FORM>
<P>

<A HREF="./"><FONT COLOR="black">Powrót</FONT></A>

</BODY>
</HTML>
```

Wyświetla podane przez użytkownika informacje (bez hasła) i prosi o wciśnięcie klawisza. Jego wciśnięcie spowoduje wywołanie akcji `/update`. Ze względu na zmiany, jakie należy wprowadzić do metod związanymi ze zdarzeniami `OnHTMLTag` tymczasem powstrzymamy się od tworzenia odpowiedniej metody.

Tworzymy akcję o nazwie Update i własnościach `MethodType=mtPost`, `PathInfo=/update`. Z jej jedynym zdarzeniem `OnAction` wiążemy metodę:

```
procedure TWebModule1.WebModule1UpdateAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
var nazwisko,email,haslo,tytul,opis :String;
    akcja :Boolean;
begin
nazwisko:=Request.CookieFields.Values['nazwisko'];
email:=Request.CookieFields.Values['email'];
haslo:=Request.CookieFields.Values['haslo'];
tytul:=Request.CookieFields.Values['tytul'];
opis:=Request.CookieFields.Values['opis'];
akcja:=(Request.CookieFields.Values['akcja']='dodaj');

if akcja
  then
```

```
  if ZapiszDoBazy(nazwisko,email,haslo,tytul,opis)
    then Response.Content:=AddedPageProducer.Content
    else begin
         komunikat:='Identyczny rekord istnieje.';
         Response.Content:=ErrorPageProducer.Content;
         end

  else

  if UsunZBazy(email,haslo)
    then Response.Content:=RemovedPageProducer.Content
    else begin
         komunikat:='W bazie danych nie ma rekordu
                                    zawierającego podany e-mail i haslo';
         Response.Content:=ErrorPageProducer.Content
         end;
end;
```

Jej kod rozpoczyna się od odczytania wartości nazwiska, e-maila, tytułu i opisu pracy z pliku cookie. Sposób ich odczytu jest niezwykle podobny jak w przypadku elementów przesłanego formularza, z tym, że zamiast `Request.ContentFields` odwołujemy się do własności `Request.CookieFields`. Odpowiednie zawartości zapisujemy dla wygody do zmiennych lokalnych. W zależności od wartości pozycji akcja zapisanej w pliku cookie wywołujemy metody `ZapiszDoBazy` lub `UsunZBazy` oraz do `Response.Content` przypisujemy zawartość strony informującej o dodaniu lub usunięciu rekordu z bazy. W razie niepowodzenia przypisywana jest strona informująca o błędzie.

Pozostaje nam uporządkowanie metod `OnHTMLTag`. Dotychczas wszystkie komponenty typu `TPageProducer` korzystały ze wspólnej metody tego typu. Teraz nie jest to już wskazane; stwórzmy więc odpowiednie metody zgodnie z podaną tabelą:

Obiekt klast `TPageProducer`	Metoda
`FormPageProducer`	`FormPageProducerHTMLTag`
`ConfirmPageProducer`	`ConfirmPageProducerHTMLTag`
`AddedPageProducer`	`AddedPageProducerHTMLTag`
`RemovedPageProducer`	
`ErrorPageProducer`	

`FormPageProducerHTMLTag` jest prostą metodą – podaje stały łańcuch tytułu strony i nagłówka każdej ze stron. Pozostałe metody wywołują go, żeby ustalić te elementy szablonu. Ponadto, i to jest wykorzystywane tylko na stronie formularza, pobiera informacje o zdalnym komputerze:

```
procedure TWebModule1.FormPageProducerHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
var remotehost :AnsiString;
begin
//Tytul i naglowek
if TagString='tytul' then ReplaceText:='Jacek Matulewski - Projekty
zaliczeniowe';
if TagString='naglowek' then ReplaceText:='Projekty zaliczeniowe';

//Informacje o zdalnym komputerze
if TagString='remotehost' then
    begin
    remotehost:=Request.RemoteHost;
    if remotehost<>Request.RemoteAddr then
                  remotehost:=remotehost+' ('+Request.RemoteAddr+')';
    ReplaceText:=remotehost;
    SaveToLogFile('RemoteHost='+remotehost);
    end;
```

```
if TagString='browser' then ReplaceText:=Request.UserAgent;
end;
```

Metoda `ConfirmPageProducerHTMLTag` widzi dane przesłane z formularza i pobiera z nich wartości bezpośrednio:

```
procedure TWebModule1.ConfirmPageProducerHTMLTag(Sender: TObject;
  Tag: TTag; const TagString: String; TagParams: TStrings;
  var ReplaceText: String);
var akcja :Boolean;
begin
//Tytul i naglowek
FormPageProducerHTMLTag(Sender,Tag,TagString,TagParams,ReplaceText);

//Dane pobrane bezposrednio z formularza
if TagString='email' then ReplaceText:=Request.ContentFields.Values['email'];
if TagString='nazwisko' then ReplaceText:=Request.ContentFields.Values['nazwisko'];
if TagString='temat' then ReplaceText:=Request.ContentFields.Values['tytul'];
if TagString='opis' then ReplaceText:=Request.ContentFields.Values['opis'];

if Request.ContentFields.Values['akcja']='dodaj' then akcja:=True
                                                 else akcja:=False;
if TagString='akcja' then
  begin
  if akcja then ReplaceText:='Dodaj do bazy'
           else ReplaceText:='Usuń z bazy';
  SaveToLogFile('Potwierdzenie ('+ReplaceText+')');
  end;
end;
```

Ostatnia metoda nie widzi formularza – korzysta więc z danych zapisanych w pliku cookie:

```
procedure TWebModule1.AddedPageProducerHTMLTag(Sender: TObject; Tag: TTag;
  const TagString: String; TagParams: TStrings; var ReplaceText: String);
begin
//Tytul i naglowek
FormPageProducerHTMLTag(Sender,Tag,TagString,TagParams,ReplaceText);

//Dane pobierane z pliku cookie
if TagString='email' then ReplaceText:=Request.CookieFields.Values['email'];
if TagString='nazwisko' then ReplaceText:=Request.CookieFields.Values['nazwisko'];
if TagString='temat' then ReplaceText:=Request.CookieFields.Values['tytul'];

if TagString='komunikat' then ReplaceText:=komunikat;
end;
```

Ponadto w tej metodzie, na użytek ErrorPageProducer podawana jest w zamian za odpowiedni znacznik szablonu informacja o błędzie pobierana z własności globalnej w TWebModule1 komunikat.

Zadanie
Stworzyć akcję i związany z nim komponent TPageProducer pokazujący pełną zawartość pliku cookie. Rozwiązanie zob. w rad4_www.zip.