Uwierzytelnianie i autoryzacja w ASP.NET 2.0

Jacek Matulewski, Sławomir Orłowski

stopień trudności

Jak wiadomo, urzędom nie są potrzebni obywatele, szpitale najlepiej funkcjonują bez pacjentów, a profesorom studenci tak naprawdę tylko przeszkadzają. Zgodnie z tą zasadą witryny internetowe najbezpieczniejsze są wówczas, gdy pracują off-line. Niestety smutna rzeczywistość zmusza administratorów do pogodzenia się z obecnością użytkowników.

To jednak wcale nie znaczy, że nie można ich kontrolować. Tak, jak petent musi stać w kolejkach, chory łykać gorzkie leki, a student zdawać egzaminy, tak internauta musi zrezygnować ze swojej anonimowości (choćby wyimaginowanej) i aby uzyskać dostęp do zasobów naszej witryny musi się zarejestrować, podając przynajmniej swój prawdziwy adres e-mail.

Trivia

Jako że celem artykułu jest wprowadzenie czytelnika do zagadnień związanych z uwierzytelnianiem w witrynach ASP.NET 2.0, powinniśmy zacząć od najbardziej podstawowych terminów związanych z tą problematyką. W informatyce używa się dwóch osobnych terminów na określenie weryfikacji tożsamości użytkownika i weryfikacji jego uprawnień. W pierwszym przypadku mówi się o uwierzytelnianiu (*ang. authentication*), w drugim o autoryzacji (*ang. authorization*). Czasem w polskiej literaturze przedmiotu terminu *uwierzytelnienie* używa się na określenie całego procesu sprawdzania tożsamości i uprawnień. Wówczas samo weryfikowanie tożsamości określane jest jako *autentykacja*. Ale słowo *autentykacja* jest tak brzydkie, że razi nawet tak niewrażliwe ucho, jak moje. Stąd raczej staram się go unikać.

Należy też wyraźnie odróżniać dwa procesy uwierzytelniania, z jakimi mamy do czynienia w aplikacjach sieciowych. Pierwszy z nich ma

Z artykułu dowiesz się

- o najpopularniejszym sposobie uwierzytelniania (autentykacji) i autoryzacji stosowanego w witrynach ASP.NET, a wiec mechanizmu Forms. Zaczniemy od prostego rozwiązania *ręcznej roboty*, w którym sami będziemy weryfikować login i hasło użytkownika na podstawie danych przechowywanych w zwykłej desktopowej bazie danych, aby kolejno wprowadzić mechanizmy udostępniane przez ASP.NET 2.0. Skończymy na w pełni zautomatyzowanym uwierzytelnianiu, korzystającym z bazy danych SQL Server, w którym ilość kodu, jaki musimy wprowadzić ręcznie będzie naprawdę znikoma,
- artykuł może być traktowany także jako poradnik migracji z ręcznego do automatycznego sposobu uwierzytelniania i autoryzacji.

Co powinieneś wiedzieć

- ogólne pojęcie o programowaniu dla platformy .NET, najlepiej w języku C#.

miejsce na styku aplikacji sieciowej i systemu operacyjnego serwera, na którym jest ona uruchomiona oraz zainstalowanego na nim oprogramowania. Dobrym przykładem może być próba uzyskania dostępu do bazy danych wymagającej podania hasła lub próba zapisu do pliku, który wymaga odpowiednich uprawnień. Natomiast drugą sprawą, choć nie pozostającej bez związku z poprzednią, jest autoryzowanie użytkownika – internauty, który stara się uzyskać dostęp do witryny (logowanie do aplikacji sieciowej). Uwierzytelnienie oznacza w tym przypadku, że to aplikacja sieciowa sprawdza tożsamość użytkownika i ewentualne udostępnia mu swoje zasoby i pośrednio zasoby systemu, do jakich ona sama ma dostęp. Zatem w pierwszym przypadku weryfikowana jest aplikacja i jej uprawnienia przez system, a w drugim aplikacja weryfikuje użytkownika. To proste rozróżnienie zaciera się jednak nieraz w praktyce, w przypadkach gdy aplikacja pełni jedynie rolę medium między użytkownikiem łączącym się z zewnątrz, a zasobami serwera lub gdy do uwierzytelnienia w witrynie stosuje się system kont zdefiniowanych w systemie operacyjnym serwera.

Aby zwiększyć bezpieczeństwo uwierzytelnienia wykorzystamy mechanizm Forms oferowany przez ASP.NET. Zakłada on, że informacje o kontach znajdują się w bazie danych, a tożsamość logowanego użytkownika sprawdzana jest w obrębie samej aplikacji sieciowej. Oczywiście ASP.NET daje nam do wyboru więcej mechanizmów wspierających uwierzytelnianie użytkowników przez aplikacje sieciowe. Obok Forms mamy tryb Windows, w którym witryna korzysta z systemu logowania do systemu operacyjnego Windows serwera. Wymaga on zatem założenia dla każdego użytkownika aplikacji sieciowej osobnego konta w systemie Windows, kontrolującym serwer WWW i to w zasadzie właśnie Windows jest odpowiedzialny za cały proces uwierzytelniania i autoryzacji. Dzięki temu możemy wykorzystać cały system poziomów uprawnień w stosunku do zalogowanego użytkownika, jaki oferuje Windows, a zatem ograniczenia w dostępie do katalogów i plików oraz kontrolę dostępu do usług systemowych (m.in. dostępu do bazy danych). Ten sposób weryfikacji sprawdza się w aplikacjach sieciowych dla niewielkiej grupy użytkowników, którym udostępniamy ważne zasoby serwera, ale nie nadaje się do witryn, w których oczekujemy rejestracji sporej ilości internautów. Jest również rozwiązanie oparte o Microsoft Password, ale nie stało się ono zbyt popularne. Należy również wspomnieć, że nowe mechanizmy uwierzytelniania wprowadzone zostały w bibliotekach dodanych do platformy .NET 3.0.

Naszym celem jest zaprojektowanie witryny ASP.NET 2.0, która korzystając z bazy danych przeprowadzać będzie uwierzytelnianie użytkowników – a mówiąc językiem bliższym codzienności – pozwalać będzie na zalogowanie użytkowników w witrynie. A priori dopuszczamy trzy grupy użytkowników: posiadający dostęp do wszystkich sfer witryny administratorzy, zwykli użytkownicy, którzy mają dostęp do wydzielonej sfery chronionej i osoby niezarejestrowane (anonimowe), których uprawnienia ograniczają się do oglądania niezabezpieczonych stron. Rejestrując się w witrynie można awansować z osoby anonimowej na użytkownika.

Listing 1. *Surowa metoda przeprowadzająca proces logowania*

```
protected void Button1_Click
      (object sender, EventArgs e)
{
   Uzytkownik uzytkownik=
         CzytajDaneUzytkownikaZBazy(TextBox1.Text);
   if (uzytkownik==null)
   {
      Label1.Text =
         "Podany e-mail nie został jeszcze użyty do rejestracji";
      Label1.ForeColor = System.Drawing.Color.Red;
      return;
   }
   //Sprawdzanie hasła
   if (TextBox2.Text == uzytkownik.Haslo)
   {
      Label1.Text = "Dane poprawnie odczytane z bazy";
      Label1.ForeColor = System.Drawing.Color.Green;
      Label2.Text =
            uzytkownik.Imie + " " + uzytkownik.Nazwisko;
      Label3.Text = uzytkownik.Grupa;
      TextBox1.Enabled = false;
      TextBox2.Enabled = false;
      Button1.Enabled = false;
      Button2.Enabled = false;
   }
   else
   {
      Label1.Text = "Niepoprawne hasło";
      Label1.ForeColor = System.Drawing.Color.Red;
   }
}
```

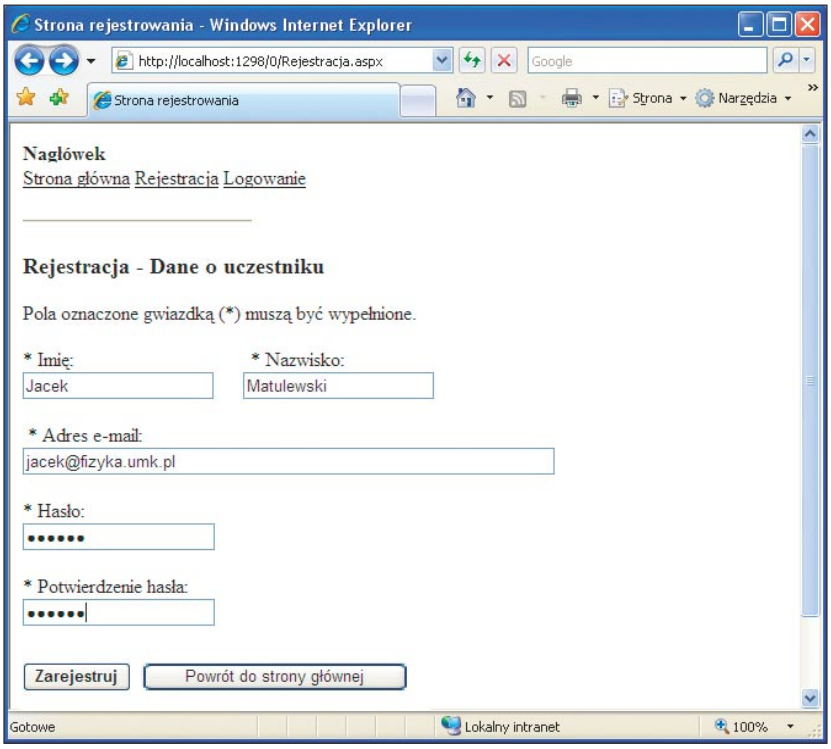
Listing 2. *Element określający sposób działania mechanizmu uwierzytelniania Forms*

```
<authentication mode="Forms">
   <forms name="Logowanie"
      loginUrl="Logowanie.aspx"
      protection="All"
      timeout ="10"
      path="/" />
</authentication>
```

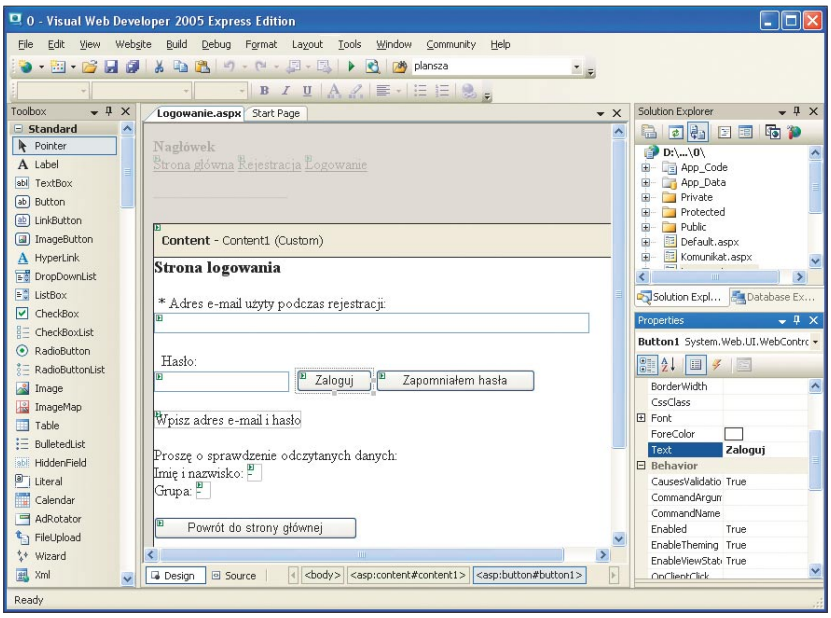
Administratorzy mogą być zaś dopisywani do bazy danych jedynie ręcznie. Zwiększa to bezpieczeństwo witryny, a przy okazji upraszcza moduł służący do rejestracji. Podziałowi na grupy odpowiadać będzie struktura katalogów witryny. Najbardziej chronionym miejscem będzie katalog *Private* – do niego zdalny dostęp będą mieli tylko administratorzy. Drugim katalogiem będzie *Protected,* który będą mogli przeglądać zarejestrowani użytkownicy witryny. I wreszcie katalog *Public* to miejsce, którego zawartość będą mogą przeglądać wszyscy, także osoby niezarejestrowane.

Dobry początek

Na CD w katalogu o nazwie *Uwierzytelnianie/0* znajduje się projekt witryny, od której zaczniemy nasze manewry. Witryna wyposażona jest między innymi w strony *Rejestracja.aspx* i *Logowanie.aspx* (Rysunek 1). Pierwsza z nich, korzystając z prostego formularza zbiera dane o użytkowniku (imię, nazwisko, adres e-mail, hasło, itd.) i zapisuje je do bazy danych ulokowanej w podkatalogu *App_Data*. Dane uzupełniane są o datę rejestracji i adres IP komputera, z którego korzystał internauta podczas rejestracji. Adres e-mail używany jest w witrynie jako identyfikator użytkownika (login), co generalnie jest dobrym pomysłem, bo ułatwia jego zapamiętanie. Strona *Logowanie.aspx* przeprowadza natomiast namiastkę uwierzytelniania użytkowników tzn. sprawdza, czy login (adres e-mail) i hasło wpisane w polach edycyjnych formularza pasują do wpisów znalezionych w bazie danych. W zasadzie cała logika tej strony skupiona jest w metodzie zdarzeniowej `Button1_Click` związanej z kliknięciem przycisku *Zaloguj*. W niej tworzone jest zapytanie *SQL* przesyłane do bazy danych i to ona sprawdza hasło odczytane z bazy, z tym podanym przez użytkownika (Listing 1). Przy przesyłaniu danych z formularzy do serwera zastosowany został mechanizm walidacji zapobiegający wysyłaniu danych, które nie spełniają określonych przez nas kryteriów (np. adres e-mail powinien mieć znak @ i przynajmniej jedną kropkę). Dla uproszczenia, pomijamy możliwość usunięcia użytkownika z bazy lub edycji jego danych. W zamian za to zadbaliśmy o system przypominania haseł – hasło może być wysłane na adres e-mail wykorzystany podczas rejestracji zapominalskiego użytkownika (odpowiada za to metoda `Button2_Click` w pliku *Logowanie.aspx.cs*).



Rysunek 1. *Początkowa postać strony rejestracji użytkowników*



Rysunek 2. *Środowisko Visual Web Developer*

Dane o użytkownikach przechowywane są w zwykłej desktopowej bazie danych Access. Równie dobrze mogłyby być umieszczone w SQL Server 2005 lub nawet w pliku XML. Jednak to, gdzie przechowujemy dane o użytkownikach nie ma większego znaczenia. Co prawda zwykle mówi się, że Access posłużyć może do co najwyżej średniej wielkości witryn, a SQL Server nawet

do największych, ale trzeba być wielkim optymistą, żeby marzyć o tym, że witryna szybko zdobędzie taką popularność, że ilość użytkowników przekroczy pojemność bazy Access. Poza tym migracja z Access do SQL Server to pestka, a obsługa plików *.mdb* przez wbudowane w system sterowniki i bibliotekę ADO.NET wydaje się chyba nawet szybsza.

Witryna wyposażona jest w trzy katalogi: *Private, Protected* i *Public*. Każdy z nich zawiera stronę Default.aspx z prostym komunikatem typu *Strefa chroniona* czy *Strefa publiczna*. W katalogu głównym znajdują się natomiast strony *Private.aspx* i *Protect.aspx* o podobnej zawartości. Wszystkie strony witryny korzystają ze wspólnego wzorca o niewyszukanej nazwie *Wzorzec.master*. Strona domyślna witryny (plik *Default.aspx* w katalogu głównym) zawiera jedynie linki do poszczególnych katalogów oraz do stron rejestracji i logowania. Jak widać, witryna ograniczona jest do niezbędnego minimum – nie ma w niej także żadnych upiększaczy w stylu kaskadowych arkuszy stylów czy AJAX. W całej stronie konsekwentnie stosowane jest oddzielenie kodu C# (pliki *.aspx.cs*) od szablonów stron HTML (pliki *.aspx*).

Dane, jakie pobierane są z bazy danych, przechowywane są w obiektach klasy `Uzytkownik`, która poza polami `Imie`, `Nazwisko`, `Email`, `Haslo` i `Grupa` posiada również metodę usuwającą z tych pól apostrofy, które nie powinny swobodnie pałętać się po budowanym z pomocą tych danych zapytaniu SQL. Baza danych zawiera jedną tabelę Użytkownicy, której pola odpowiadają polom klasy Uzytkownik (plus adres IP i data rejestracji). Wszystkie pola bazy danych są wymagane. Kluczem głównym tabeli jest adres e-mail, co zwalnia nas z obowiązku pilnowania, aby używane do rejestracji adresy e-mail były unikalne – baza danych dopilnuje tego za nas. Użytkownicy są domyślnie zapisywani

Listing 3. *Tworzenie biletu, czyli pliku cookie przechowywanego przez przeglądarkę zalogowanej osoby*

```
protected void Button1_Click(object sender, EventArgs e)
    {
        FormsAuthentication.Initialize(); //Inicjacja mechanizmy uwierzytelniania Forms
        Uzytkownik uzytkownik=CzytajDaneUzytkownikaZBazy(TextBox1.Text);

        if (uzytkownik==null)
        {
            Label1.Text = "Podany e-mail nie został jeszcze użyty do rejestracji";
            Label1.ForeColor = System.Drawing.Color.Red;
            return;
        }
        //Sprawdzanie hasła
        if (TextBox2.Text == uzytkownik.Haslo)
        {
            Label1.Text = "Dane poprawnie odczytane z bazy";
            Label1.ForeColor = System.Drawing.Color.Green;
            Label2.Text = uzytkownik.Imie + " " + uzytkownik.Nazwisko;
            Label3.Text = uzytkownik.Grupa;
            TextBox1.Enabled = false;
            TextBox2.Enabled = false;
            Button1.Enabled = false;
            Button2.Enabled = false;
            //Tworzymy bilet
            FormsAuthenticationTicket bilet = new FormsAuthenticationTicket(
                1, //wersja biletu
                uzytkownik.Email, //e-mail użytkownika
                DateTime.Now, //data i godzina założenia pliku cookie
                DateTime.Now.AddMinutes(30), //czas wygaśnięcia
                false, //czy cookie ma być trwałe
                FormsAuthentication.FormsCookiePath); //adres strony
            //Umieszczenie biletu na zdalnym komputerze użytkownika
            HttpCookie cookie = new HttpCookie(
                FormsAuthentication.FormsCookieName,
                FormsAuthentication.Encrypt(bilet));
            Response.Cookies.Add(cookie);
        }
        else
        {
            Label1.Text = "Niepoprawne hasło";
            Label1.ForeColor = System.Drawing.Color.Red;
        }
    }
```

do grupy Użytkownicy (zob. metoda `DodajUzytkownikaDoBazy` z pliku *Rejestracja.aspx.cs*).

Jednak prawda jest taka, że w tej postaci witryny cały proces uwierzytelniania przeprowadzony na stronie *Logowanie.aspx* można o kant stołu potłuc, bo wszystkie katalogi i strony witryny są i tak dostępne dla każdej osoby, która na nią zabłądzi. Naszym najpilniejszym zadaniem jest zatem zmiana tego stanu rzeczy i wprowadzenie uwierzytelniania z prawdziwego zdarzenia. Można to przeprowadzić w dwojaki sposób. Pierwszy, kontynuujący filozofię ręcznego budowania systemu uwierzytelniania, to po udanym logowaniu zapisać w zmiennych sesji dane potrzebne do identyfikacji użytkownika (choćby tylko jego login) i sprawdzać ich obecność na innych stronach witryny. Można również stworzyć plik cookie na komputerze internauty, którego obecność będzie świadectwem pomyślnej weryfikacji jego tożsamości. My skorzystamy z tego drugiego sposobu. Co więcej, wykorzystamy do tego mechanizmy wbudowane w ASP.NET 2.0 i w ten sposób wejdziemy na ścieżkę, która zaprowadzi nas na końcu artykułu do całkowitego zautomatyzowania procesu uwierzytelniania, w którym cały wpisany wcześniej kod okaże się zbędny. Jednak dzięki niemu zrozumiemy, na czym polega uwierzytelnianie i jak jest realizowane w gotowych do kontrolkach ASP.NET 2.0.

Projekt witryny został przygotowany w zintegrowanym środowisku programistycznym *Visual Web Developer 2005 Express Edition* (Rysunek 2). To darmowe narzędzie, oferowane przez Microsoft, pozwala na sprawne projektowanie witryn sieciowych, korzystających z platformy .NET w wersjach 2.0 i 3.0 (to drugie po zainstalowaniu platformy .NET 3.0 w systemie). Jednak w obu wersjach platformy .NET wykorzystywane są i tak te same biblioteki ASP.NET 2.0 i ADO.NET 2.0. Proponuję, aby z tego samego środowiska skorzystał także Czytelnik przy dalszej rozbudowie witryny. Można je ściągnąć ze strony *http://msdn.microsoft.com/vstudio/express/downloads/*.

Uwierzytelnianie z użyciem mechanizmu Forms

Uwierzytelnienie użytkownika, jakie zastosowaliśmy na stronie *Logowanie.aspx*, polega na weryfikacji hasła podanego przez internautę przez porównanie go z hasłem przechowywanym w bazie danych. Jest to całkiem niezły sposób w przypadku takiej witryny, jak nasza, na której nie przechowujemy żadnych poufnych danych, nie mamy do czynienia z pieniędzmi, przesyłaniem numerów kart kredytowych i możemy liczyć na to, że potencjalni użytkownicy pozbawieni są bezinteresownej złośliwości (jeżeli tacy w ogóle istnieją). W projekcie znajdującym się na CD zaimplementowana jest ta część mechanizmu, która bezpośrednio współpracuje z bazą danych – dopisuje nowych użytkowników i weryfikuje hasła zarejestrowanych osób. Drugą część tego mechanizmu, a więc kontrolę dostępu do stron i katalogów witryny zrealizujemy korzystając z mechanizmu Forms zaimplementowanego w ASP.NET. Dzięki niemu można nie tylko rozciągnąć uwierzytelnienie (lub jak ktoś woli autentykację) na inne strony witryny, ale również w prosty sposób autoryzować dostęp do poszczególnych jej części.

Mechanizm uwierzytelniania Forms korzysta z pliku cookie na komputerze, z którego loguje się internauta. Pozwala to, choć tego nie wykorzystaliśmy w tym przykładzie, na pozostawienie cookie na dłużej niż czas sesji. Dzięki temu możemy pozwolić użytkownikowi uniknąć logowania przy każdym odwiedzaniu naszej witryny – informacje o użytkowniku mogą być odczytane wprost z owego pliku cookie, czyli w języku uwierzytelniania Forms – z biletu. Tu postąpimy jednak inaczej – plik cookie nie będzie trwały i wygaśnie wraz z sesją użytkownika w aplikacji ASP.NET.

W podoknie *Solution Explorer Visual Web Developera* klikamy dwukrotnie plik Web.config i odnajdujemy w nim element `authentication`, który

Listing 4. *Element wstawiany do pliku Web.config zapisanym w katalogu Protected*

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
      <authorization>
        <deny users="?" />
        <allow users="*" />
      </authorization>
    </system.web>
</configuration>
```

Listing 5. *Klasa chronionej strony (plik Protected.aspx.cs)*

```
public partial class Protected : System.Web.UI.Page
{
    //kontrola biletu
    if (!User.Identity.IsAuthenticated)
        Response.Redirect("Logowanie.aspx?ReturnUrl=Private.aspx");
}
```

Listing 6. *Czy jest bezpiecznie?*

```
protected void Page_Load(object sender, EventArgs e)
{
    //kontrola biletu i grupy
    if (!User.Identity.IsAuthenticated || !User.IsInRole("Administratorzy"))
        Response.Redirect("Logowanie.aspx?ReturnUrl=Private.aspx");
}
```

w domyślnych ustawieniach wskazuje na uwierzytelnianie typu Windows i ma następującą prostą postać:

```
<authentication mode="Windows"/>
```

Modyfikujemy ten element, zmieniając tryb uwierzytelnienia na Forms (Listing 2). Użyty podelement `forms` zawiera ustawienia dotyczące tego typu uwierzytelnienia. Jego atrybuty mają następujące znaczenie: `loginUrl` – definiuje adres URL, do którego następuje przekierowanie w przypadku braku ważnego biletu uwierzytelnienia, `protection` – określa sposób szyfrowania pliku cookie i sprawdzania, czy jego treść nie została przechwycona (wartość *All* oznacza, że cookie jest zarówno szyfrowany, jak i weryfikowana jest jego treść), `timeout` – czas w minutach, po którym ważność biletu wygasa, i wreszcie `path` to ścieżka do pliku cookie, jaki zapisany zostanie na komputerze logującego się użytkownika. Jest jeszcze atrybut `slidingExpiration`, który jeżeli ustawiony jest na `false`, dodatkowo ogranicza żywotność biletu.

Przechodzimy teraz do edycji pliku *Logowanie.aspx.cs*. Sprawdzamy, czy w nagłówku pliku znajduje się deklaracja przestrzeni nazw zawierającej klasy związane z uwierzytelnianiem tzn. `using System.Web.Security;`.

W metodzie `Button1_Click`, związanej z przyciskiem *Zaloguj*, umieszczamy dodatkowe polecenia tworzące bilet, jaki wręczany jest logowanej osobie (Listing 3). Bezpieczeństwo aplikacji zapewnia fakt, że jej połączenie z bazą danych jest całkowicie niezależne od uwierzytelnienia użytkownika. Dzięki temu może ona samodzielnie zalogować się do serwera *Access* i sprawdzić listę zarejestrowanych użytkowników i ich haseł. W ten sposób to aplikacja w pełni kontroluje, jakie dane będą użytkownikowi udostępnione.

Metoda `Button1_Click` zawiera teraz całą procedurę uwierzytelniania. Na początku odczytywane są te rekordy tabel bazy danych, w których znajduje się podany przez internautę adres e-mail (realizuje to pomocnicza metoda `CzytajDaneUzytkownikaZBazy`). Jeżeli hasło znajdujące się w bazie danych pasuje do tego podanego w formularzu, metoda przystępuje do przygotowania pliku cookie – biletu. Obecność biletu będzie sprawdzana przy próbie dostępu do stron witryny.

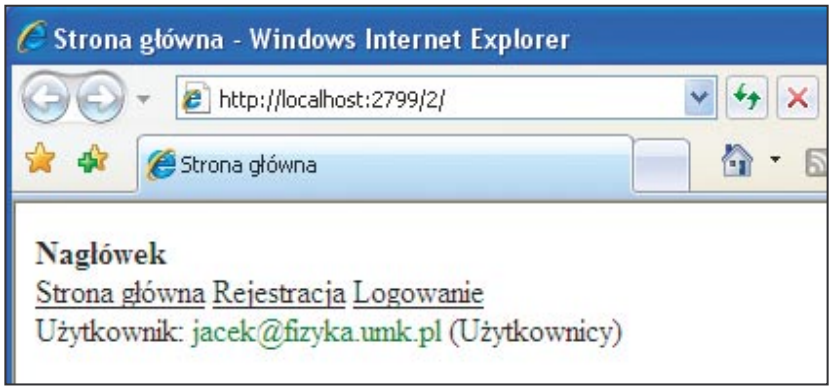
Autoryzacja – strefa chroniona

Teraz przystąpmy do tworzenia chronionej strefy witryny, czyli tej przeznaczonej dla wszystkich zalogowanych użytkowników. W tym celu w podoknie *Solution Explorer* zaznaczamy podkatalog Protected i z jego menu kontekstowego wybieramy pozycję *Add New Item*. W oknie o tym samym tytule zaznaczamy pozycję *Web configuration file*. W efekcie powstanie w tym katalogu plik *Web.config*. Definiujemy w nim element `authorization`, który zablokuje dostęp do tej części witryny użytkowników anonimowych (Listing 4). Jest to równoznaczne z wymuszeniem zalogowania na stronie wskazanej przez element `authentication` w głównym pliku konfiguracyjnym (zob. Listing 2), a więc na stronie *Logowanie.aspx*. Dostęp do katalogu Protected mają mieć wszyscy użytkownicy (element `allow` zawiera atrybut `users` równy * oznaczający wszystkich użytkowników), którzy nie są anonimowi (element `deny` zawiera atrybut `users` równy ?, co oznacza użytkowników anonimowych tj. niezalogowanych). Ważne jest ustawienie atrybutów – pierwszy powinien być `deny`, a dopiero po nim `allow` – gwiazdka oznacza więc w zasadzie nie tyle wszystkich użytkowników, co pozostałych.

Można bardziej wybiórczo potraktować autoryzację, umieszczając zamiast gwiazdki w elemencie `allow` listę użytkowników z kolejnymi loginami wymienionymi po przecinku. Taką samą listę można również umieścić w elemencie `deny`. Jednak w przypadku rzeczywistych witryn z otwartym dostępem takie rozwiązanie stosuje się rzadko, bo jest zwyczajnie niepraktyczne przy większej ilości użytkowników.

Dzięki obecności znacznika `authorization` w pliku *Web.config* z katalogu *Protected,* dostęp do stron w tym katalogu jest możliwy tylko w przypadku posiadania ważnego biletu (pliku cookie). Jeżeli go nie mamy, zostaniemy skierowani do kasy tj. na stronę *Logowanie.aspx*. Sprawdźmy to uruchamiając aplikację (F5) i bez logowania przechodząc do katalogu *Protected*. Zamiast jego zawartości zobaczymy stronę logowania (oczywiście jeżeli wszystko działa jak należy). Jeżeli jednak teraz zalogujemy się i dopiero wtedy przejdziemy do chronionego katalogu, to zobaczymy stronę *Protected/Default.aspx*.

Zwróćmy także uwagę, że przy wymuszonym przejściu do strony logowania do adresu strony dodawany jest parametr `ReturnUrl`, w którym zapisany jest względny adres strony, do której użytkownik chciał się dostać np. *http://localhost:1263/Uwierzytelnianie/Logowanie.aspx?ReturnUrl=%2fUwierzytelnianie%2fProtected%2fDefault.aspx*. Dzięki temu, po prawidłowym zalogowaniu można



Rysunek 3. *Tożsamość użytkownika widoczna w nagłówku każdej strony*

wymusić automatyczny powrót na tę stronę. Wystarczy do metody `Button1_Click` w pliku *Logowanie.aspx.cs* po utworzeniu biletu dodać polecenie:

```
if (Request.Params["ReturnUrl"] !=
null) Response.Redirect(Request.Params[
"ReturnUrl"]);
```

Wykorzystując mechanizm uwierzytelnienia Forms, można również chronić dostęp do pojedynczych stron, nie znajdujących się w chronionym katalogu. Należy wówczas samodzielnie sprawdzić obecność biletu, dodając do metody `Page_Load` polecenie z Listingu 5. Zróbmy to na stronie *Protected.aspx,* która powinna znajdować się w katalogu głównym projektu dostępnego na CD.

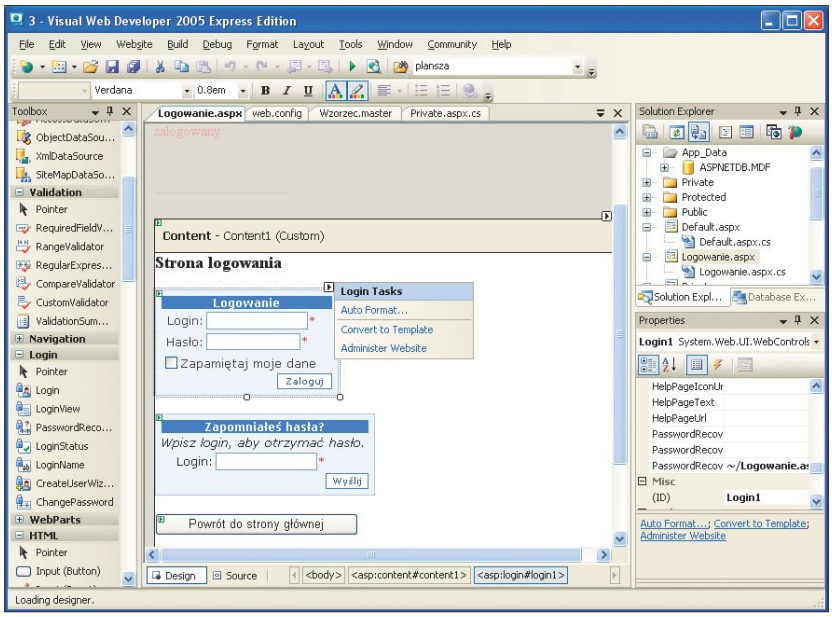
W metodzie `Page_Load` sprawdzamy wartość własności `User.Identity.IsAuthenticated`. W istocie oznacza to sprawdzenie obecności cookie/biletu na zdalnym komputerze osoby łączącej się z naszą aplikacją. Jeżeli go nie ma, uznajemy, że użytkownik nie jest zalogowany i zostaje natychmiast przeniesiony na stronę logowania (proszę zwrócić uwagę, że dodajemy parametr pozwalający na automatyczny powrót po zalogowaniu). Ten sposób sprawdzenia będzie działał zarówno w przypadku trwałych, jak i czasowych plików cookie. Działanie tej metody można sprawdzić ustawiając stronę Protected.aspx jako stronę startową w oknie projektu, lub wpisując jej adres do przeglądarki po uruchomieniu aplikacji.

Korzystając z tego samego sposobu odczytywania biletu, możemy w szablonie wzorca Wzorzec.master umieścić element wyświetlający adres e-mail zalogowanej osoby jak na Rysunku 3 (zob. projekt w katalogu Uwierzytelnianie/1 na CD).

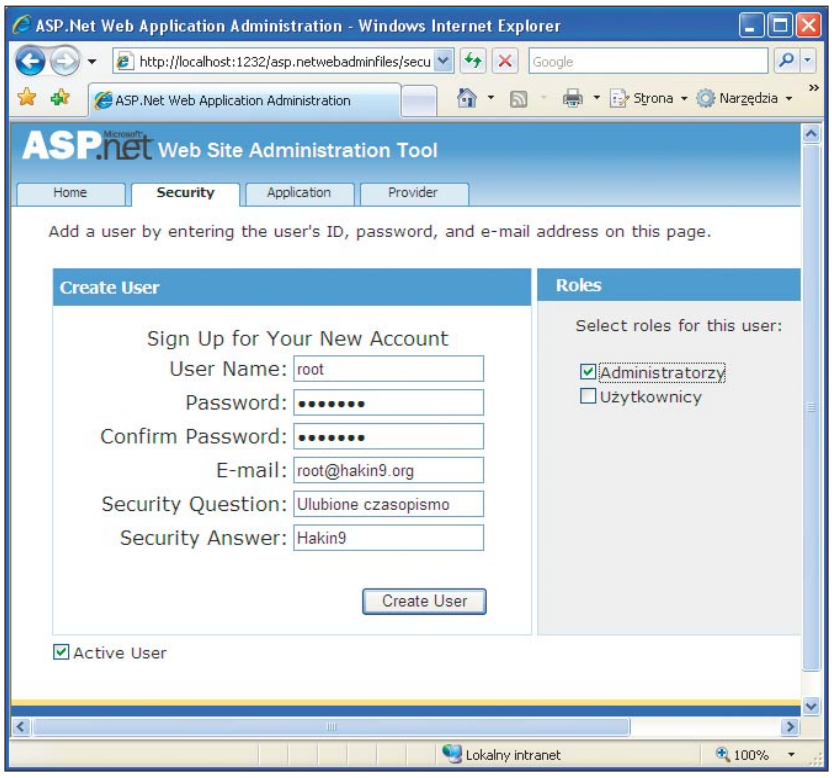
Zanim przejdziemy dalej, chciałbym zwrócić uwagę Czytelnika na fakt, że w pliku Global.asax można utworzyć metody do następujących zdarzeń obiektu `Application`: `AuthenticateRequest`, `AuthorizeRequest`, `PostAuthenticateRequest` i `PostAuthorizeRequest`. Mogą być one bardzo pomocne, jeżeli chcemy np. centralnie monitorować proces uwierzytelniania.

Autoryzacja – wykorzystanie mechanizmu ról

Teraz zajmijmy się strefą prywatną (katalog Private), do której dostęp mają tylko administratorzy witryny. Przede wszystkim musimy stworzyć konto administratora w bazie danych. W bazie projektów dostępnych na CD jest już konto z adresem *root@hakin9.org* i hasłem *hasło*, ale użytkownik może dodać własne lub po prostu zmienić nazwę grupy jednego z zarejestrowanych użytkowników (z Użytkownicy



Rysunek 4. *Kontrolka sieciowa Login i jego podręczna lista zadań*



Rysunek 5. *Interfejs pozwalający na zarządzanie kontami w bazie SQL Server*

na Administratorzy). Dysponując kontem administratora, które, jak i wszystkie inne konta w bazie, ma unikalny login (adres e-mail) moglibyśmy przeprowadzić autoryzację na jego podstawie. Wystarczyłoby utworzyć w katalogu *Private* plik konfiguracyjny *Web.config* z elementem:

```
<authorization>
    <allow users="root@hakin9.org" />
    <deny users="*" />
</authorization>
```

Takie rozwiązanie sprawdzi się, gdy witryna posiada jednego, względnie kilku administratorów. My jednak pójdziemy o krok dalej i skorzystamy z wbudowanego w ASP.NET i ADO.NET mechanizmu ról. Tak naprawdę będzie to nasz drugi krok w kierunku pełnej automatyzacji uwierzytelniania. Mechanizm ról, z którego korzystać będziemy w dalszej części artykułu, wymaga zainstalowanego i dostępnego dla ASP.NET serwera SQL Server 2005.

Zacznijmy od włączenia mechanizmu zarządzania rolami. Do pliku konfiguracyjnego witryny (plik Web.config w katalogu głównym) dodajemy następujący element:

```
<roleManager
    enabled="true"
    cacheRolesInCookie="true" >
</roleManager>
```

Mechanizm ról korzysta z automatycznie tworzonej bazy danych ulokowanej w pliku *App_Data/ASPNETDB.MDF*. Powinniśmy umieścić w niej definicje ról i przypisać do nich poszczególnych użytkowników. Każdy użytkownik może należeć do kilku grup, z których każda może decydować o innego typu uprawnieniach. Jednak w naszym przypadku, w którym Administratorzy mają wszystkie uprawnienia zwykłych Użytkowników, sprawdzi się system, w którym każdy użytkownik może należeć tylko do jednej grupy. Jej nazwa była zapisywana w naszej podstawowej bazie danych. Oznacza to, że informacje te powinny teraz zostać skopiowane do bazy danych wykorzystywanej przez mechanizm ról. Proponuję zrobić to półautomatycznie. Niech aplikacja sama rozbudowuje drzewo ról i przypisanych do nich użytkowników, w miarę ich logowania się do witryny. W tym celu musimy jeszcze raz wrócić do strony logowania i do metody `Button1_Click` dwa polecenia tworzące nową rolę (o ile ta już nie istnieje) i przypisujące do niej logowanego właśnie użytkownika (o ile ten już tam nie jest „zapisany"). Nazwy ról będą identyczne z nazwami grup, a więc Użytkownicy i Administratorzy. Poniższe polecenia należy umieścić tuż po poleceniach zapisujących bilet:

```
if (!Roles.RoleExists(uzytkownik.Grupa
)) Roles.CreateRole(uzytkownik.Grupa);
if (!Roles.IsUserInRole(uzytkownik.Ema
il,uzytkownik.Grupa))
Roles.AddUserToRole(uzytkownik.Email,uz
ytkownik.Grupa);
```

Zwróćmy uwagę, że rola, w przeciwieństwie do naszego biletu, nie jest czymś tymczasowym. Role i przypisani do nich użytkownicy pozostają zapisani w bazie do czasu, aż zostaną z niej usunięci. Poza tym informacje o rolach przechowywane są po stronie serwera, a bilet po stronie klienta.

Teraz zajmijmy się ograniczeniem dostępu do katalogu Private. Tworzymy w nim plik Web.config i, podobnie, jak zrobiliśmy wcześniej w katalogu Protected, dodajemy do niego element `authorization`. Jednak tym razem uprawnienia będziemy kontrolować nie na poziomie loginów, ale na poziomie grup, do których należą użytkownicy:

```
<authorization>
    <deny users="?" />
    <allow roles="Administratorzy" />
    <deny users="*" />
</authorization>
```

Jak czytać powyższy element? Przede wszystkim nie wpuszczamy anonimowych (niezalogowanych) internautów. Wpuszczamy za to administratorów, ale nikogo poza nimi.

Podobnie jak wcześniej, możemy przeprowadzić autoryzację samodzielnie. W katalogu projektu znajdzie Czytelnik plik *Private.aspx.cs* związany ze stroną *Private.aspx,* do którego należy dodać polecenia podobne do widocznego w Listingu 5, ale uzupełnione o weryfikację roli posiadacza biletu (Listing 6).

Warto także zajrzeć do pliku wzorca z projektu w katalogu *Uwierzytel-nianie/2* na CD, żeby zobaczyć, jak korzystając z tych samych poleceń wyświetlić nazwę grupy użytkownika w nagłówku wszystkich stron witryny.

Listing 7. *Poniższy element należy umieścić wewnątrz elementu configuration głównego pliku konfiguracyjnego witryny*

```
<system.net>
  <mailSettings>
    <smtp from="uwierzytelnianie@hakin9.org">
      <network host="smtp.hakin9.org" password="hasło" userName="uwierzyteln
                    ianie" />
    </smtp>
  </mailSettings>
</system.net>
```

Listing 8. *Nadal zachowujemy zasadę, że użytkownicy rejestrujący się na stronie mogą należeć jedynie do grupy Użytkownicy*

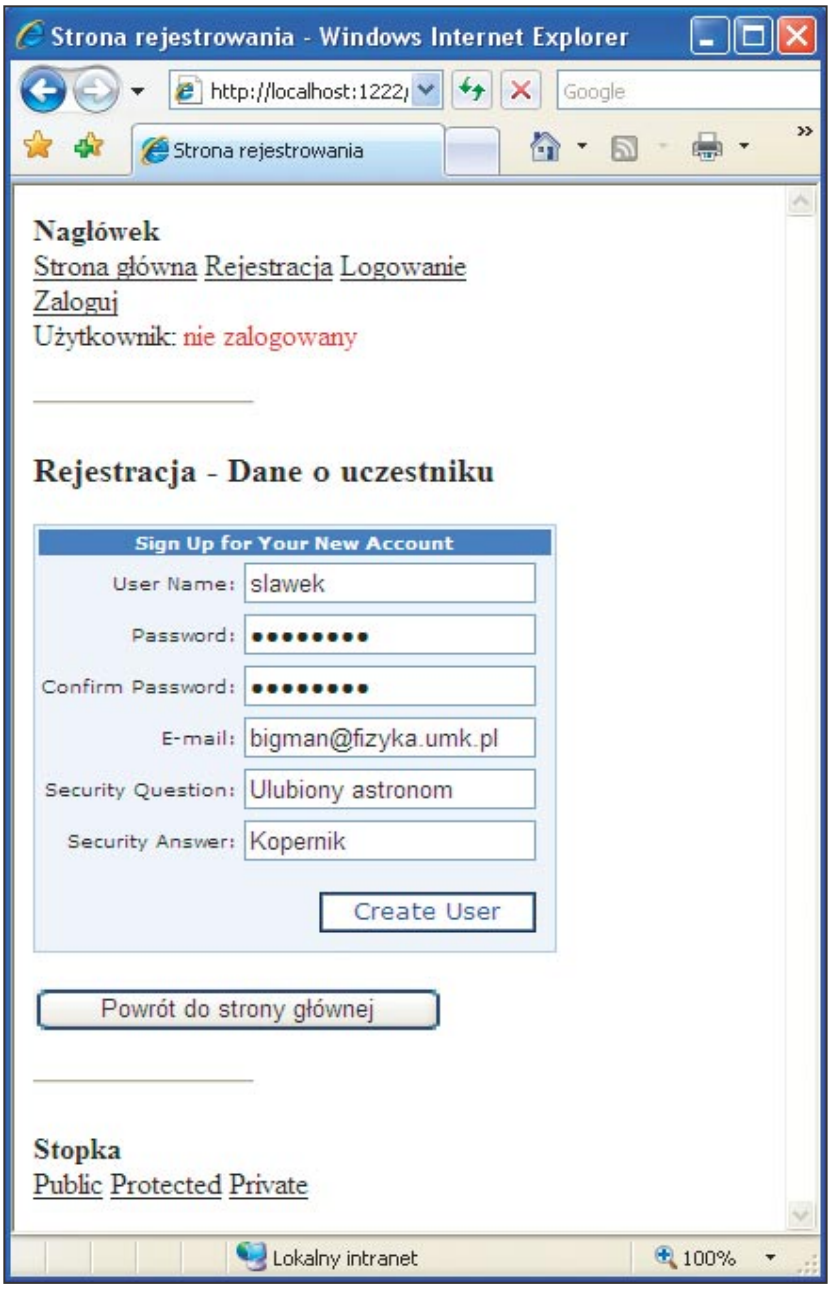
```
protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
{
    Roles.AddUserToRole(CreateUserWizard1.UserName, "Użytkownicy");
}
```

W stronę szablonu

Uwierzytelnianie i autoryzacja pozwalają zabezpieczyć witrynę przez dostępem osób niepowołanych. Przyzwoitość wymaga jednak chociażby wspomnieć o drugiej stronie medalu. Żądając od użytkowników podania loginu i hasła narażamy także ich dane na niebezpieczeństwo. O ile plik cookie biletu jest wysyłany w postaci zaszyfrowanej i weryfikowana jest jego zawartość, to dane wpisywane w formularzu rejestracji i logowania nie są w żaden sposób zabezpieczane. Warto zatem rozważyć zastosowanie zabezpieczenia Secure Socket Layer (SSL) do przesyłania informacji w postaci zaszyfrowanej z użyciem protokołu HTTPS. Szczególnie, gdy w grę wchodzi przesyłanie numeru karty kredytowej, numerów PIN i wszelkiego typu informacji, które nie powinny być dostępne innym. Aby korzystać z SSL witryna musi znajdować się na serwerze wyposażonym w odpowiednie certyfikaty, a użytkownik musi korzystać z przeglądarki, które umożliwiają korzystanie z czterdziestobitowego klucza (opcjonalnie może być wymagany nawet studwudziestoośmiobitowy).

W Sieci

- *http://msdn2.microsoft.com* – microsoftowe kompendium wiedzy o WinAPI, .NET i technologiach stowarzyszonych,
- *http://www.asp.net* – oficjalna witryna poświęcona bibliotece ASP.NET,
- *http://msdn.microsoft.com/vstudio/express/downloads/* – tu można pobrać darmowe środowisko projektowe dla twórców witryn ASP.NET – *Visual Web Developer Express Edition.*



Rysunek 6. *Nowa strona rejestrowania użytkowników*

Między innymi ze względów bezpieczeństwa przesyłanych danych, Microsoft zaleca, aby do przechowywania informacji o użytkownikach korzystać nie z własnych rozwiązań, jak to przedstawiono powyżej, ale z gotowego szablonu *Membership.* Ma to tym większy sens, że do przechowywania informacji o kontach mechanizm ten wykorzystuje tę samą bazę danych, co mechanizm ról. Unikniemy w ten sposób podwójnego źródła danych, a tym samym nasza migracja do aplikacji w pełni wykorzystującej wbudowane w ASP.NET 2.0 mechanizmy uwierzytelniania zostanie niemal zakończona. Ujednolicenie źródła danych nie jest jedyną zaletą wprowadzenia mechanizmu Membership do projektu naszej witryny. Do jego wykorzystania skłaniać mogą dodatkowo gotowe do użycia kontrolki z zakładki *Login* w *Visual Web Developerze* współpracujące z mechanizmem *Membership* i pulpit pozwalający na zarządzanie kontami w bazie SQL Servera. Tak jak zapowiedziałem, gdy wykonamy ten ostatni krok, okaże się, że niemal cały kod, który umieściliśmy w aplikacji (włączając w to obsługę bazy danych) jest zbędny. Ale w końcu jakoś trzeba się uczyć. Podejrzewam, że korzystanie z gotowych *czarnych skrzynek* bez zrozumienia ich działania nie jest tym, co Czytelnicy *hakin9u* lubią najbardziej. Wadą nowego rozwią-

zania będzie jego *szablonowość*, co oznacza, że trudniej niż własne rozwiązania będzie skłonić go do nietypowych pomysłów projektanta witryny. Z punktu widzenia bezpieczeństwa korzystanie z takiego szablonu może być korzystne, w końcu możemy założyć, że jest on porządnie przetestowany, ale jednocześnie groźne, każda wykryta i opublikowana dziura mechanizmu może być natychmiast wykorzystana przeciwko nam.

Zacznijmy od umieszczenia na stronie *Logowanie.aspx* komponentu `Login` z zakładki *Login* palety kontrolek *Toolbox.* Przejmie ona zadanie metody `Button1_Click` tzn. zajmie się weryfikowaniem loginu i hasła, a jeżeli się one będą zgadzać, utworzy bilet, którym na naszej witrynie będzie legitymował się zalogowany użytkownik. I wszystko to bez napisania choćby jednej linii kodu. Takie rozwiązanie bez wątpienia sprzyja efektywności projektowania stron w ASP.NET. Do tego komponent `Login` możemy szybko sformatować, korzystając z polecenia *Auto Format*, dostępnego z podręcznej listy zadań (zobacz Rysunek 4) lub w oknie własności. Ponadto każdy napis tego komponentu można przetłumaczyć, co oczywiście warto i należy zrobić (bo Polacy nie gęsi itd.).

Jak już wspomniałem, przy domyślnych ustawieniach (przy których zostaniemy) komponent `Login` i pozostałe komponenty z zakładki Login korzystają z bazy danych *App_Data/ASPNETDB.MDF.* Tej samej, z której korzysta mechanizm ról. Musimy dodać do niej informacje o użytkownikach. Nie będziemy, jak zrobiliśmy w przypadku grup, dodawać do strony poleceń przenoszących użytkowników z jednej bazy do drugiej w miarę logowania, bo nie ma sensu dłużej utrzymywać dwóch źródeł danych. Utworzymy nowe konta. Pierwszym sposobem tworzenia konta użytkownika jest panel administracyjny dostępny z menu Website, ASP.NET *Configuration Visual Web Developera.* Wybierając tę pozycję z menu, uruchomimy w przeglądarce aplikację ASP.NET (Rysunek 5), która na zakładce *Security* udostępnia narzędzia służące do zarządzania rolami (zobaczmy, że istnieją dwie role utworzone dzięki naszym poleceniom migracji) i użytkownikami (na razie nie ma żadnego). Jeżeli klikniemy link *Create User,* przejdziemy do strony zakładania konta. Załóżmy konto dla administratora witryny. Możemy zachować dotychczasową konwencję, w której rolę loginu pełni adres e-mail, ale proponuję od niej odstąpić i przystosować się do szablonu proponowanego przez Microsoft. Niech nowy użytkownik nazywa się root. Poza loginem musimy jeszcze podać hasło (przynajmniej siedem liter i znak nie będący literą lub cyfrą – hasło dla testowego konta root, które dodaliśmy do bazy to *hasło++*), adres e-mail oraz pytanie i odpowiedź wykorzystywane w razie zapomnienia hasła). Gdy już wypełnimy wszystkie pola, zaznaczamy z prawej strony rolę Administratorzy, do której przypiszemy nowe konto i klikamy przycisk *Create User.*

W analogiczny sposób stwórzmy także konto dla zwykłego użytkownika. Mając te dwa konta możemy przetestować nowy sposób logowania. Najpierw zalogujmy się jako zwykły użytkownik i sprawdźmy, czy mamy dostęp do stron wewnętrznych witryny. Powinniśmy mieć dostęp do katalogu *Protected,* ale nie do *Private.* Natomiast po wlogowaniu jako root, będziemy mieli nieograniczony dostęp do wszystkich stref witryny. Ponadto zauważmy, że działa ręczne sprawdzanie tożsamości na stronach *Private.aspx* i *Protected.aspx* w katalogu głównym oraz że nazwa użytkownika jest nadal wyświetlana w nagłówku wzorca (choć teraz widoczny jest tam login zamiast adresu e-mail). Wszystko to działa, ponieważ nie zmieniliśmy mechanizmu uwierzytelniania. Nadal jest nim mechanizm Forms korzystający z plików cookie. Jedynie zautomatyzowaliśmy jego wykorzystanie. W ramach tej automatyzacji możemy także zastąpić ręczne wyświetlanie nazwy zalogowanego użytkownika

w nagłówku wzorca przez użycie komponentu `UserName` z zakładki *Login* (zob. projekt w katalogu *Uwierzytelnianie/3* na CD).

Po tym, jak już przekonamy się, że nowy sposób działa, możemy ze strony *Logowanie.aspx* usunąć pola edycyjne i przycisk *Zaloguj,* a z pliku *Logowanie.aspx.cs* wysłużoną metodę `Button1_Click` wraz z jej metodami pomocniczymi. Należy też usunąć metodę `Button2_Click` wysyłającą adres e-mail do użytkowników, którzy zgłosili zapomnienie hasła. Co w takim razie z funkcją przypominania haseł? I na to jest automat. Umieśćmy na stronie kontrolkę `PasswordRecovery`. Podobnie, jak w przypadku kontrolki `Login` możemy ją automatycznie sformatować i przetłumaczyć wszystkie pojawiające się na niej komunikaty. Kontrolka najpierw spyta o login, a następnie zada pytanie ustalone w trakcie zakładania hasła. Jeżeli wszystko się będzie zgadzać, prześle na adres e-mail użytkownika jego nowe hasło. Do tego należy jednak skonfigurować wykorzystywane przez witrynę konto e-mail. Najłatwiej zrobić to za pomocą panelu konfiguracyjnego, który wykorzystaliśmy do tworzenia kont użytkowników. Potrzebne narzędzia znajdziemy na zakładce *Application, SMTP Settings.* Jeżeli jeszcze zupełnie nie wyzbyliśmy się potrzeby edycji kodu (w końcu daje to wrażenie sprawczości odbierane przez korzystanie z gotowych kontrolek i kreatorów), możemy do pliku konfiguracyjnego witryny dodać element z Listingu 7.

Oczywiście, tworzenie kont za pomocą panelu zarządzania bazą danych jest wskazane w przypadku kont dla administratorów, ale na większą skalę, gdy do rejestrowania w witrynie chcemy zachęcić jak najwięcej osób jest niewygodne. Musimy zatem przeorganizować stronę rejestrowania tak, aby pozwalała na dopisywanie użytkowników do nowej bazy. Przejdźmy zatem do jej widoku projektowania i podobnie jak w przypadku strony logowania, usuńmy z niej cały formularz. W konsekwencji z pliku z kodem C# należy usunąć metodę zdarzeniową byłego przycisku *Zarejestruj* i jej metody pomocnicze. W to puste miejsce wstawmy kontrolkę `CreateUserWizard`. Konfiguruje się ją analogicznie, jak kontrolki poznane wcześniej. Jedyny problem to fakt, że nowa kontrolka w żaden sposób nie odnosi się do ról. Musimy do nich przypisywać użytkowników samodzielnie. Na szczęście znamy już odpowiednie polecenia. Skorzystajmy zatem ze zdarzenia `CreatedUser` zgłaszanego w momencie utworzenia nowego konta i do związanej z nim metody zdarzeniowej dodajmy polecenia z Listingu 8. Warta zainteresowania jest jeszcze kontrolka `ChangePassword`, która służy do zmieniania hasła przez użytkownika. I ją konfiguruje się w sposób podobny do poznanych wcześniej.

Co ciekawe, mechanizm uwierzytelniania Forms pozwala nie tylko na kontrolę stron, do których wpuszczany jest zalogowany użytkownik, ale również na zmienianie ich zawartości w zależności od tego, jaki użytkownik ją ogląda. Służy do tego kontrolka `LoginView`, która jest jak zaczarowane pudełko, którego zawartość zależy od tego, w jaki sposób się je otworzy. Kontrolka `LoginView` ma dwa podstawowe widoki: pierwszy przeznaczony dla użytkowników anonimowych, drugi dla zalogowanych. Ponadto można też zdefiniować widoki przeznaczone dla użytkowników z grup (ról). Przetestujmy ten komponent w nagłówku wzorca. W jego podręcznej liście zadań znajdziemy rozwijane menu z dwoma pozycjami. Wybór pierwszej (o nazwie *AnonymousTemplate*) spowoduje, że będziemy mogli określić zawartość kontrolki w przypadku wykrycia braku biletu. Wstawmy tam po prostu napis *Użytkownik: niezalogowany* ze słowami *niezalogowany* wyróżnionymi czerwonym kolorem. Po przełączeniu na drugą pozycję (*LoggedIn Template*), umieśćmy wewnątrz pojemnika komponent `UserName`, który wyświetlać będzie nazwę użytkownika. Jego szablon (własność `FormatString`) możemy zmienić np. na Użytkownik: <FONT COLOR=green>{0}</FONT>. Korzystając z polecenia *Edit RoleGroups*, możemy również zdefiniować widok przeznaczony dla administratorów witryny (zobacz projekt na CD).

Podsumowanie

Wybór między samodzielnie przygotowanym i w pełni dostosowanym do własnych potrzeb formularzem, a kontrolkami wprowadzonymi do ASP.NET 2.0 zależy oczywiście od specyfiki projektowanej witryny. W powyższym artykule przedstawiliśmy oba podejścia, choć oczywiście tylko w zakresie, na jaki pozwala format artykułu. Po więcej informacji na temat ASP.NET 2.0 i ADO.NET 2.0 oraz w razie chęci zgłębiania problematyki projektowania witryn korzystających z tych technologii, odsyłamy do książki *Technologie ASP.NET i ADO.NET w Visual Web Developer* (Helion, 2007) przygotowanej przez autorów artykułu. ●

O autorach

Jacek Matulewski – fizyk zajmujący się na co dzień optyką kwantową i układami nieuporządkowanymi na Wydziale Fizyki, Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu. Jego specjalnością są symulacje ewolucji układów kwantowych, oddziaływujących z silnym światłem lasera. Od 1998 roku interesuje się programowaniem dla systemu Windows, w szczególności w środowisku Borland C++Builder. Ostatnio zainteresowany także platformą .NET i językiem C#. Wierny użytkownik kupionego w połowie lat osiemdziesiątych *komputera osobistego* ZX Spectrum 48k.
Kontakt z autorem:
jacek@fizyka.umk.pl

Sławomir Orłowski – z wykształcenia fizyk. Obecnie jest doktorantem na Wydziale Fizyki Astronomii i Informatyki Stosowanej Uniwersytetu Mikołaja Kopernika w Toruniu. Zajmuje się symulacjami komputerowymi układów biologicznych (dynamika molekularna) oraz bioinformatyką. Programowanie jest nieodzowną częścią jego pracy naukowej. Ma doświadczenie w programowaniu w językach C, C++, Delphi, Fortran, Java i Tcl. Z językiem C# i platformą .NET pracuje od 2002 roku.