4. Zabezpieczanie aplikacji przez logowanie

Tworzona od modułu 4 aplikacja bankowa posiada menu nawigacyjne, dynamicznie reagujące na rolę zalogowanego użytkownika. Niestety, mimo ukrycia niedostępnych poleceń, wystarczy znać pełny adres strony związanej z poleceniem menu, żeby wyświetlić ją w oknie przeglądarki. Najwyższa już pora na zastosowanie ograniczeń w dostępnie do składników aplikacji.

W celu zabezpieczenia aplikacji zostaną wprowadzone dwa mechanizmy. Pierwszym z nich będzie wykorzystanie zintegrowanego uwierzytelniania ASP.NET, opartego na logowaniu do aplikacji przez tzw. Web Forms (formularz sieci WWW). Drugim mechanizmem będzie wykorzystanie ról użytkownika do wprowadzenia — przez plik *web.config* — ograniczeń dostępu do wybranych plików i katalogów aplikacji.

4.1. Rozbudowa bazy danych

Rozpoczniesz pracę od stworzenia w bazie danych *bank.mdb* dwóch nowych tabel: `loginy` i `role`. Pierwsza z nich będzie przechowywać login oraz skrót hasła użytkownika wraz z kluczem obcym tabeli `klienci`. Druga tabela będzie przechowywać role dla loginów użytkowników, których one dotyczą.

W tym celu wykonaj poniższe czynności:

1. Korzystając z zakładki *Data* w oknie projektu połącz się z bazą *bank.mdb*.
2. Stwórz nową tabelę o nazwie `loginy`. Dodaj do niej kolumny zgodnie z poniższą tabelą:

Tabela 2. Kolumny tabeli *loginy*

Nazwa kolumny (Name)	Nazwa właściwości	Wartość właściwości
login	DataType	Text
	Size	20
	AllowNulls	False
	InPrimaryKey	True
	IsUniqueKey	False
password	DataType	Text
	Size	40
	AllowNulls	False
userid	DataType	Integer
	AllowNulls	False

3. Zamknij okno edycji schematu tabeli. Zatwierdź dokonane zmiany.
4. Stwórz drugą nową tabelę o nazwie `role`. Dodaj do niej kolumny zgodnie z poniższą tabelą:

Tabela 3. Kolumny tabeli *role*

Nazwa kolumny (Name)	Nazwa właściwości	Wartość właściwości
login	DataType	Text
	Size	20
	AllowNulls	False
	InPrimaryKey	True
	IsUniqueKey	False
rola	DataType	Text
	Size	20
	AllowNulls	False
	InPrimaryKey	True
	IsUniqueKey	False

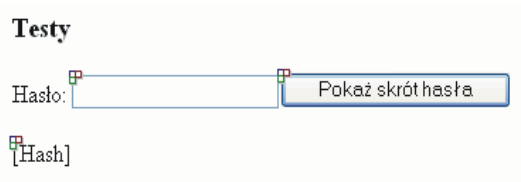
5. Zamknij okno edycji schematu tabeli. Zatwierdź dokonane zmiany.
6. Nie wprowadzaj jeszcze danych do stworzonych przed chwilą tabel.

4.2. Tworzenie skrótów haseł

Aby móc założyć loginy, potrzebujesz najpierw wiedzieć, jakie skróty odpowiadają wprowadzanym hasłom. W tym celu zmienisz teraz zawartość strony *Testy.aspx*. Zyskasz dzięki temu możliwość generowania skrótów haseł. Przyda się to w sam raz przy ręcznym wpisywaniu skrótów haseł do tabeli loginy bazy danych.

W tym celu wykonaj poniższe czynności:

1. Otwórz stronę *Testy.aspx*.
2. Usuń z niej kontrolki służące dotąd do wyboru roli i zapamiętaniu jej w sesji. Są to: `LabRola`, `DdListRole`, `BtnUstaw` i `BtnWyloguj`.
7. Usuń z kodu (zakładka *Code*) obsługę zdarzeń usuniętych przycisków: *BtnUstaw_Click* i *BtnWyloguj_Click*.
8. Zmień teraz (zakładka *Design*) wygląd strony, wstawiając tam pole tekstowe, przycisk i jeszcze jedno pole tekstowe, zgodnie z poniższym rysunkiem.



Rysunek 5. Strona umożliwiająca sprawdzanie skrótów haseł

9. Ustaw właściwości kontrolek zgodnie z poniższą tabelą:

Tabela 4. Kontrolki strony *Testy.aspx*

Typ kontrolki	Nazwa właściwości	Wartość właściwości
TextBox	(ID)	Password
Button	(ID)	BtnHash
	Text	Pokaż skrót hasła
TextBox	(ID)	Hash
	MaxLength	40
	ReadOnly	True
	Width	350px

10. Dodaj dla przycisku *BtnHash* obsługę zdarzenia `Click`. Wpisz kod generujący skrót dla podanego hasła:

Przykład 6. Generowanie skrótu hasła metodą szyfrującą SHA-1

```
void BtnHash_Click(object sender, EventArgs e) {
    Hash.Text = FormsAuthentication.HashPasswordForStoringInConfigFile(
        Password.Text, "SHA1");
}
```

11. Zapisz zmiany i uruchom stronę. Zauważ, że po wprowadzeniu hasła i wciśnięciu przycisku w polu tekstowym poniżej zostaje wyświetlony 40-znakowy skrót hasła. Taki skrót możesz skopiować i użyć przy wstawianiu danych do tabeli `loginy` w bazie danych *bank.mdb*.
12. Dla każdego klienta zapisanego w twojej bazie w tabeli `klienci` stwórz wpis w tabeli `loginy`, podając nazwę użytkownika (w kolumnie `login`), skrót hasła (w kolumnie `password` — odczytany ze strony *Testy.aspx*) oraz identyfikator użytkownika (w kolumnie `userid` — wartość kolumny `id` z tabeli `klienci`).
13. Dodaj również wpisy w tabeli `role`, podając w sposób unikatowy pary: login oraz nazwę roli. Jako nazwę roli stosuj `klient`, `pracownik` lub `administrator`. Zastosowanie takiej dodatkowej tabeli pozwala uzyskać możliwość przypisania wielu ról do jednego loginu.

4.3. Zmiana kontrolki menu

Wkrótce stary mechanizm zapamiętujący rolę użytkownika w sesji zostanie zastąpiony zintegrowanym uwierzytelnianiem ASP.NET, dlatego warto już teraz zadbać o zmianę

zachowania kontrolki menu nawigacyjnego. Zamiast odczytywać rolę użytkownika z sesji, kontrolka menu będzie korzystać z metody `Context.User.IsInRole`. Użycie tej metody daje nawet lepsze efekty, ponieważ pozwala na to, żeby użytkownik miał naraz przypisanych wiele ról.

Ponieważ zaplanowane są trzy role użytkowników aplikacji (*klient*, *pracownik* oraz *administrator*), należy dodatkowo umożliwić dostęp do polecenia *Testy* tylko administratorowi. W tym celu wykonaj poniższe czynności:

1. Stwórz najpierw nową kontrolkę poleceniem *File → New File → ASP.NET User Control*, upewnij się czy w polu *Language* jest wybrana opcja *C#*, a następnie zapisz tę kontrolkę w katalogu *ascx* pod nazwą `MenuAdministrator.ascx`.
2. Otwórz z podkatalogu *ascx* kontrolkę `Menu.ascx` i przejdź na zakładkę *HTML*.
3. Wytnij wskazaną część kodu HTML i przenieś do kontrolki `MenuAdministrator.ascx` (znacznik `<br/>` i link *Testy*):

Przykład 7. Kod kontrolki menu przenoszony do pliku *MenuAdministrator.ascx*

```
<p>
    <asp:Panel id="PanelMenu" runat="server" Width="200px">
        <p>
            ...
            <Bank:MenuPracownik id="MenuPracownik" runat="server">
            </Bank:MenuPracownik>
            <br />
            <br />
            <asp:HyperLink id="LnkTesty" runat="server"
                NavigateUrl="/Testy.aspx">Testy</asp:HyperLink>
        </p>
    </asp:Panel>
</p>
```

4. Zapisz zmiany dokonane w kontrolce `MenuAdministrator.ascx`.
5. Powróć do dokumentu *Menu.ascx* i w miejscu, gdzie były wycięte linie wstaw nowy znacznik (wywołanie utworzonej kontrolki):

Przykład 8. Użycie stworzonej kontrolki wewnątrz głównej kontrolki menu

```
<p>
    <asp:Panel id="PanelMenu" runat="server" Width="200px">
        <p>
```

```
            ...
            <Bank:MenuPracownik id="MenuPracownik" runat="server">
            </Bank:MenuPracownik>
            <br />
            <Bank:MenuAdministrator id="MenuAdministrator" runat="server">
            </Bank:MenuAdministrator>
        </p>
    </asp:Panel>
</p>
```

6. Przejdź na zakładkę *All* i dodaj deklarację znacznika kontrolki (bez niego nie będzie działać):

Przykład 9. Deklaracje znaczników dla stworzonych kontrolek

```
<%@ Control Language="C#" %>
<%@ Register TagPrefix="Bank" TagName="MenuKlient" Src="MenuKlient.ascx" %>
<%@ Register TagPrefix="Bank" TagName="MenuPracownik" Src="MenuPracownik.ascx" %>
<%@ Register TagPrefix="Bank" TagName="MenuAdministrator"
    Src="MenuAdministrator.ascx" %>
<script runat="server">
```

7. Przejdź na zakładkę *Code* i zmień kod odpowiedzialny za wyświetlanie i ukrywanie elementów menu:

Przykład 10. Zmiana zasady działania dynamicznego wyświetlania elementów menu

```
void PanelMenu_PreRender(object sender, EventArgs e) {
    MenuKlient.Visible = Context.User.IsInRole("klient") ||
        Context.User.IsInRole("administrator");
    MenuPracownik.Visible = Context.User.IsInRole("pracownik") ||
        Context.User.IsInRole("administrator");
    MenuAdministrator.Visible = Context.User.IsInRole("administrator");
}
```

8. Zapisz wszystkie dokonane zmiany i sprawdź, czy strony nadal poprawnie działają z nową kontrolką menu.

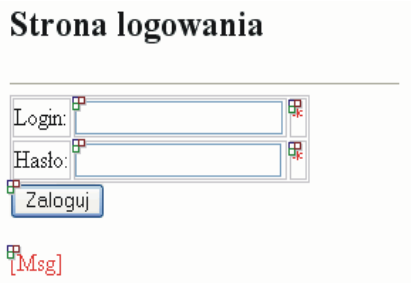
4.4. Tworzenie stron logowania i wylogowania

Posiadając gotową do wykorzystania strukturę danych w bazie, stworzysz teraz strony logowania oraz zakończenia pracy z aplikacją. Przy logowaniu sprawdzany będzie w bazie

login użytkownika oraz skrót jego hasła, następnie zaś — jeśli dane będą się zgadzać — w sesji ASP.NET będzie zapamiętywany identyfikator użytkownika. Dzięki temu aplikacja będzie mogła rozpoznawać każdą zalogowaną osobę i wykorzystywać to przy operacjach wymagających takiej informacji. Będzie to wykorzystane także później, do powiązania użytkownika z rolami, w jakich występuje względem aplikacji. Wylogowanie z aplikacji będzie polegać na usunięciu z sesji identyfikatora użytkownika oraz usunięciu informacji o użytkowniku przez mechanizm uwierzytelniania ASP.NET.

Web Matrix posiada możliwość stworzenia bazowych stron logowania i wylogowania. Wykorzystasz więc tę możliwość, aby stworzyć potrzebne strony, a następnie dostosujesz ich wygląd i działanie. W tym celu wykonaj poniższe czynności:

1. Stwórz nową stronę logowania, poleceniem *File → New File → kategoria Security → Login Page*. Nazwij tą stronę *Login.aspx*, upewnij się czy w polu *Language* wybrany jest język C# i zapisz ją w katalogu głównym projektu.
2. Zmień napisy na stronie i w kontrolkach, zgodnie z poniższym rysunkiem.



Rysunek 6. Wygląd strony logowania

3. Przełącz się na zakładkę *All* i dodaj w drugiej linii deklarację znacznika kontrolki menu:

Przykład 11. Deklaracja znacznika kontrolki menu

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="Bank" TagName="Menu" Src="/ascx/Menu.ascx" %>
<script runat="server">
. . .
```

4. Przełącz się na zakładkę *HTML* i zmień kod: dodaj znacznik `<link>` włączający styl CSS, usuń styl wbudowany ze znacznika `<body>` oraz dodaj dyrektywy włączania plików *Header.txt* oraz *Footer.txt*:

Przykład 12. Modyfikacja kodu HTML strony logowania, w celu uzyskania układu zgodnego z pozostałymi stronami witryny bankowej

```
<html>
<head>
    <link href="/style/bank.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <form runat="server">
        <!-- #Include file="Header.txt" -->
        <h2>Strona logowania
        </h2>

        . . .

        <p>
            <asp:Label id="Msg" runat="server" forecolor="red"></asp:Label>
        </p>
        <!-- #Include file="Footer.txt" -->
    </form>
</body>
</html>
```

5. Dodaj teraz metodę odczytu skrótu hasła użytkownika z bazy danych. Będzie ona użyta przy obsłudze próby logowania do aplikacji. Przełącz się w tym celu na zakładkę *Code* i przeciągnij poniżej metody `LoginBtn_Click` generator kodu *SELECT Data Method*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
6. W polu wyboru *Select a database* wybierz połączenie z bazą danych *bank.mdb* (jeśli jest otwarta) lub wskaż pozycję *<New Database Connection>* i wciśnij przycisk *Create...*, aby stworzyć nowe połączenie do tej bazy.
7. Kliknij w przycisk *Next*.
8. W liście tabel wskaż tabelę `loginy`. Zaznacz w liście kolumn: `login` i `userid`.
9. Kliknij w przycisk *WHERE*.
10. Kliknij w przycisk *OK* — spowoduje to dodanie kolumny `login` jako parametru zapytania.
11. Kliknij w przycisk *AND Clause*.

12. W liście kolumn wskaż pozycję *password*. Kliknij w przycisk *OK*.
13. Kliknij w przycisk *Next*.
14. Jeśli chcesz, możesz sprawdzić budowę zapytania klikając w przycisk *Test Query*.
15. Kliknij w przycisk *Next*.
16. W polu nazwy metody wpisz `WczytajLogin`.
17. Kliknij w przycisk radiowy *DataReader*.
18. Kliknij w przycisk *Finish*.
19. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `WczytajLogin`. Jako parametr wejściowy przyjmuje ona login użytkownika, a wyniku działania zwraca obiekt `DataReader`.
20. Wygenerowany w ten sposób kod ma wpisaną pełną ścieżkę dostępu do pliku *bank.mdb*. Popraw początek metody `WczytajLogin`, aby ścieżka do bazy była odczytywana z pliku *web.config*:

Przykład 13. Odczyt z bazy danych informacji potrzebnych do zalogowania klienta

```
System.Data.IDataReader WczytajLogin(string login, string password) {
    string connectionString =
        ConfigurationSettings.AppSettings["connectionstring"];
    System.Data.IDbConnection dbConnection = new
        System.Data.OleDb.OleDbConnection(connectionString);

    string queryString = "SELECT [loginy].[login], " +
        "[loginy].[userid] FROM [loginy] " +
        "WHERE (([loginy].[login] = @login) " +
        "AND ([loginy].[password] = @password))";
    ...
```

21. Zmodyfikuj kod obsługi zdarzenia *Click* przycisku *LoginBtn*:

Przykład 14. Logowanie klienta do aplikacji

```
void LoginBtn_Click(object sender, EventArgs e) {
    if (Page.IsValid) {
        System.Data.OleDb.OleDbDataReader loginDR =
           (System.Data.OleDb.OleDbDataReader) WczytajLogin(
                UserName.Text,
                FormsAuthentication.HashPasswordForStoringInConfigFile(
                UserPass.Text, "SHA1"));
        if (loginDR.HasRows && loginDR.Read()) {
            Session["userid"] = Convert.ToInt32(loginDR["userid"]);
            FormsAuthentication.RedirectFromLoginPage(UserName.Text, true);
```

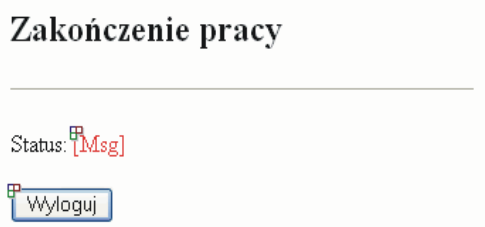
```
        }
        else {
            Msg.Text = "Błędny login lub hasło. Spróbuj ponownie.";
        }
    }
}
```

Powyższy kod wymaga komentarza. Przy żądaniu logowania użytkownik podaje login oraz hasło. Ponieważ w bazie danych przetrzymywany jest tylko skrót hasła, trzeba również stworzyć skrót z przysłanego hasła. Dopiero taka wartość może być przekazana do metody `WczytajLogin`.

Udane logowanie rozpoznawane jest po tym, że w uzyskanym obiekcie `DataReader` znajdują się dane (dokładnie jeden wiersz). Oznacza to, że hasło oraz skrót zgadzają się. W takim razie w sesji zapamiętywany jest identyfikator użytkownika (*userid*). Następnie wywoływana jest metoda `RedirectFromLoginPage`, która daje dostęp do zawartości zablokowanej hasłem strony. Przy jej wywołaniu podawany jest login użytkownika, który jest zapamiętywany przez mechanizm uwierzytelniania ASP.NET. Niezależnie od tego identyfikator użytkownika także będzie potrzebny aplikacji — na stronie wylogowania.

Wykonaj teraz poniższe czynności:

1. Stwórz nową stronę wylogowania, poleceniem *File → New File → kategoria Security → Logout Page*. Nazwij tą stronę *Wyloguj.aspx*, upewnij się, czy w polu *Language* wybrany jest język C# i zapisz ją w katalogu głównym projektu.
2. Zmień napisy na stronie i w kontrolkach, zgodnie z poniższym rysunkiem.



Rysunek 7. Wygląd strony wylogowania

3. Zmień nazwę *(ID)* etykietki ze `Status` na `Msg`. Przypomnij sobie, że w pliku *Footer.txt* jest już etykietka o takiej nazwie.
4. Przełącz się na zakładkę *All* i dodaj w drugiej linii deklarację znacznika kontrolki menu:

Przykład 15. Deklaracja znacznika kontrolki menu

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="Bank" TagName="Menu" Src="/ascx/Menu.ascx" %>
<script runat="server">
. . .
```

5. Przełącz się na zakładkę *HTML* i zmień kod: dodaj znacznik `<link>`, włączający styl CSS, usuń styl wbudowany ze znacznika `<body>` oraz dodaj dyrektywy włączania plików *Header.txt* oraz *Footer.txt*:

Przykład 16. Modyfikacja kodu HTML strony wylogowania, w celu uzyskania układu zgodnego z pozostałymi stronami witryny bankowej

```
<html>
<head>
    <link href="/style/bank.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <form runat="server">
        <!-- #Include file="Header.txt" -->
        <h2>Zakończenie pracy
        </h2>

        . . .

        <p>
            Status: <asp:Label id="Msg" runat="server"
            forecolor="red"></asp:Label>
        </p>
        <asp:Button id="LogOffBtn" onclick="LogOffBtn_Click" runat="server"
        text="Wyloguj"></asp:Button>
        <!-- #Include file="Footer.txt" -->
    </form>
</body>
</html>
```

6. Zmodyfikuj kod obsługi zdarzenia `Click` przycisku `LogOffBtn`. Dzięki dodaniu przekierowania na tę samą stronę, uzyskany zostanie efekt zmiany zawartości menu nawigacyjnego po wylogowaniu. Zostanie również wyświetlona informacja o braku zalogowania.

Przykład 17. Odświeżenie strony w przeglądarce po wylogowaniu użytkownika

```
void LogOffBtn_Click(Object sender, EventArgs e) {
    FormsAuthentication.SignOut();
    Response.Redirect("Wyloguj.aspx");
}
```

7. Dodaj teraz dwie metody: odczytującą imię i nazwisko użytkownika z bazy danych oraz odczytującą sformatowaną nazwę użytkownika. Będą one używane przy ładowaniu strony, w celu wyświetlenia informacji o użytkowniku.
8. Przełącz się w tym celu na zakładkę *Code* i przeciągnij generator kodu *SELECT Data Method* poniżej metody `LogOffBtn_Click`. Na ekranie pojawi się okno dialogowe *Select a database connection*.
9. Aktywne jest ostatnie, właściwe połączenie. Kliknij w przycisk *Next*.
10. W liście tabel wskaż tabelę `klienci`. Zaznacz w liście kolumn: `imie` i `nazwisko`.
11. Kliknij w przycisk *WHERE*.
12. Kliknij w przycisk *OK* — spowoduje to dodanie kolumny `id` jako parametru zapytania.
13. Kliknij w przycisk *Next*.
14. Jeśli chcesz, możesz sprawdzić budowę zapytania klikając w przycisk *Test Query*.
15. Kliknij w przycisk *Next*.
16. W polu nazwy metody wpisz `WczytajImieNazwisko`.
17. Kliknij w przycisk radiowy *DataReader*.
18. Kliknij w przycisk *Finish*.
19. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `WczytajImieNazwisko`. Jako parametr wejściowy przyjmuje ona identyfikator użytkownika, a w wyniku działania zwraca obiekt `DataReader`.
20. Wygenerowany w ten sposób kod ma wpisaną pełną ścieżkę dostępu do pliku *bank.mdb*. Popraw początek metody `WczytajImieNazwisko`, aby ścieżka do bazy była odczytywana z pliku *web.config*:

Przykład 18. Odczyt z bazy danych imienia i nazwiska wskazanego klienta

```
System.Data.IDataReader WczytajImieNazwisko(int id) {
    string connectionString =
        ConfigurationSettings.AppSettings["connectionstring"];
```

```
        System.Data.IDbConnection dbConnection = new
            System.Data.OleDb.OleDbConnection(connectionString);

        string queryString = "SELECT [klienci].[imie], " +
            "[klienci].[nazwisko] FROM [klienci] " +
            "WHERE ([klienci].[id] = @id)";

        ...
```

21. Poniżej metody `WczytajImieNazwisko` stwórz metodę odczytującą sformatowaną nazwę użytkownika:

Przykład 19. Odczyt sformatowanej nazwy użytkownika

```
string NazwaUzytkownika(object userid) {
    int id = Convert.ToInt32(userid);
    System.Data.OleDb.OleDbDataReader dr =
        (System.Data.OleDb.OleDbDataReader) WczytajImieNazwisko(id);
    if (dr.HasRows && dr.Read()) {
        return dr["imie"].ToString() + " " + dr["nazwisko"] + " ";
    }
    return "";
}
```

22. Zmodyfikuj kod obsługi zdarzenia `Page_Load`:

Przykład 20. Wyświetlanie informacji o zalogowanym użytkowniku

```
void Page_Load(object sender, EventArgs e) {
    if (Request.IsAuthenticated == true) {
        Msg.Text = "zalogowany(a) " + NazwaUzytkownika(Session["userid"]) +
            "[" + User.Identity.Name + "].";
        LogOffBtn.Visible = true;
    } else {
        Msg.Text = "nie zalogowany.";
        LogOffBtn.Visible = false;
    }
}
```

23. Zapisz dokonane zmiany. Sprawdzisz działanie logowania i wylogowania z aplikacji już niedługo — po włączeniu tego mechanizmu przez plik *web.config*.

4.5. Role zalogowanego użytkownika

Zalogowanie użytkownika w opisany powyżej sposób powoduje, że odtąd wszystkie żądania użytkownika będą już uwierzytelnione. Jednak mechanizm ten pozwala również na przypisanie użytkownika do zdefiniowanych samodzielnie ról i zbudowanie na tej podstawie dokładnie określonych zasad dostępu do zasobów aplikacji.

Konfiguracja ograniczeń z użyciem ról jest deklaratywna i umieszcza się ją w pliku *web.config*. Otwórz więc ten plik i zmodyfikuj go, dopisując poniższe sekcje `<allow roles>` i `<deny users>`. Zauważ, że użyte tutaj klauzule `roles` zezwalają na dostęp użytkownikom o odpowiednich rolach, zaś klauzule `users` blokują dostęp wszystkich pozostałych użytkowników. W tym celu wykonaj poniższe czynności:

1. Otwórz plik *web.config* aplikacji. Odnajdź w nim umieszczoną w komentarzu sekcję `<authentication>`. Odblokuj ją, przenosząc odpowiednio koniec komentarza. Zmień także sposób uwierzytelniania z `Windows` na `Forms`:

Przykład 21. Włączenie mechanizmu uwierzytelniania w pliku *web.config*

```
<configuration>
    <system.web>

        ...

        <!--

            The <authentication> section enables configuration of the
            security authentication mode used by ASP.NET to identify
            an incoming user.  It supports a "mode" attribute with four
            valid values: "Windows", "Forms", "Passport" and "None":

            The <forms> section is a sub-section of the <authentication>
            section, and supports configuring the authentication values
            used when Forms authentication is enabled above:

        -->

            <authentication mode="Forms">

                    <forms name=".ASPXAUTH"
                           loginUrl="Login.aspx"
```

```
                        protection="Validation"
                        timeout="999999" />

        </authentication>
```

2. Podobnie odblokuj znajdującą się poniżej sekcję `<authorization>` i wskaż, że dostęp do elementów aplikacji ma być ustalony domyślnie tylko dla zalogowanych użytkowników:

Przykład 22. Włączenie mechanizmu autoryzacji w pliku *web.config*

```
        <!--

            The <authorization> section enables developers/administrators
            to configure whether a user or role has access to a particular
            page or resource.  This is accomplished by adding "<allow>"
            and "<deny>" sub-tags beneath the <authorization> section -
            specifically detailing the users/roles allowed or denied access

            Note: The "?" character indicates "anonymous" users
           (ie: non authenticated users).
            The "*" character indicates "all" users.

        -->

            <authorization>
               <deny users="?" />
            </authorization>

    </system.web>
```

3. Dodaj za końcem głównej sekcji `<system.web>` dodatkowe sekcje `<location>`, podobnie jak dla plików *Header.txt*, *Footer.txt* i katalogu *temp*. Bez tego dostęp do elementów publicznych aplikacji także będzie wyłączony, a chcemy przecież umożliwić klientom czytanie regulaminu i zakładanie kont. Posłuż się w tym celu poniższą tabelą.

Przykład 23. Przypomnienie sekcji ograniczającej dostęp do pliku *Header.txt* aplikacji

```
    <!-- Elementy niedostępne dla nikogo z zewnątrz -->

    <location path="Header.txt">
```

```
        <system.web>
            <authorization>
                <deny users="*" />
            </authorization>
        </system.web>
    </location>
```

Tabela 5. Sekcje w pliku *web.config* konfigurujące dostęp do aplikacji

Elementy niedostępne dla nikogo z zewnątrz	
<location path=	Podsekcje do umieszczenia w sekcji **<authorization>**
Header.txt	<deny users="*" />
Footer.txt	<deny users="*" />
temp	<deny users="*" />
dane	<deny users="*" />
ascx	<deny users="*" />
Elementy dostępne dla wszystkich	
<location path=	Podsekcje do umieszczenia w sekcji **<authorization>**
Default.aspx	<allow users="*" />
style	<allow users="*" />
Regulamin.aspx	<allow users="*" />
Cennik.aspx	<allow users="*" />
Wyloguj.aspx	<allow users="*" />
Elementy dostępne dla klientów	
<location path=	Podsekcje do umieszczenia w sekcji **<authorization>**
klient	<allow roles="klient, administrator" /> <deny users="*" />
Elementy dostępne dla pracowników	
<location path=	Podsekcje do umieszczenia w sekcji **<authorization>**
bank	<allow roles="pracownik, administrator" /> <deny users="*" />
Elementy administracyjne	
<location path=	Podsekcje do umieszczenia w sekcji **<authorization>**
Testy.aspx	<allow roles="administrator" /> <deny users="*" />
bank/DaneKlientow.aspx	<allow roles="administrator" /> <deny users="*" />

Na tym etapie dostęp do wskazanych elementów aplikacji jest ograniczony i będzie wymagał zalogowania. Jednak role użytkowników, jakkolwiek uwzględnione w kontrolce menu oraz w pliku *web.config*, nie są jeszcze aktywne po zalogowaniu. Należy teraz jeszcze dodać kod, który po udanym zalogowaniu użytkownika odczyta z bazy danych jego role względem aplikacji i powiąże go z tymi rolami. W tym celu konieczne będzie stworzenie i rozbudowa pliku *Global.asax*. Przechowuje on bowiem kod obsługujący globalne zdarzenia aplikacji. Nas będzie interesować zdarzenie wywoływane tuż po zakończonym sukcesem uwierzytelnieniu (autentykacji) użytkownika.

Ponieważ do pliku *Global.asax* nie można dodawać kodu odczytującego informacje z bazy danych, pierwszy krok polegać będzie na stworzeniu metody odczytu ról użytkownika

w pliku *Login.aspx*. Następnie kod ten trzeba będzie wyciąć i przenieść do pliku *Global.asax*.

Oto ostatnie czynności:

4. Otwórz plik *Login.aspx* i przełącz się na zakładkę *Code*.
5. Przeciągnij poniżej metody `WczytajLogin` generator kodu *SELECT Data Method*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
6. Powinno być aktywne ostatnie, właściwe połączenie. Kliknij w przycisk *Next*.
7. W liście tabel wskaż tabelę `role`. Zaznacz w liście kolumn: `rola`.
8. Kliknij w przycisk *WHERE*.
9. Kliknij w przycisk *OK* — spowoduje to dodanie kolumny `login` jako parametru zapytania.
10. Kliknij w przycisk *Next*.
11. Jeśli chcesz, możesz sprawdzić budowę zapytania klikając przycisk *Test Query* i wpisując wybrany login użytkownika.
12. Kliknij w przycisk *Next*.
13. W polu nazwy metody wpisz `WczytajRole`.
14. Kliknij w przycisk radiowy *DataSet* (zaznaczony domyślnie).
15. Kliknij w przycisk *Finish*.
16. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `WczytajRole`. Jako parametr wejściowy przyjmuje ona identyfikator użytkownika, a w wyniku działania zwraca obiekt `DataSet`.
17. Wygenerowany w ten sposób kod ma wpisaną pełną ścieżkę dostępu do pliku *bank.mdb*. Popraw początek metody `WczytajRole`, aby ścieżka do bazy była odczytywana z pliku *web.config*:

Przykład 23. Odczyt z bazy danych imienia i nazwiska wskazanego klienta

```
System.Data.DataSet WczytajRole(string login) {
    string connectionString =
        ConfigurationSettings.AppSettings["connectionstring"];
    System.Data.IDbConnection dbConnection = new
        System.Data.OleDb.OleDbConnection(connectionString);

    string queryString = "SELECT [role].[rola] FROM [role] " +
        "WHERE ([role].[login] = @login)";

    ...
```

18. Stwórz nowy plik *Global.asax* poleceniem *File → New File → Global.asax*. Zapisz ten plik w katalogu głównym aplikacji bankowej.
19. Przejdź do otwartego pliku *Login.aspx*. Zaznacz całą treść wygenerowanej metody `WczytajRole` i wytnij ją.
20. Wklej treść metody `WczytajRole` do pliku *Global.asax*, tuż za początkiem sekcji `<script runat="server">`.

Pamiętasz zapewne, że każdy plik aplikacji ASP.NET jest przy pierwszym wywołaniu kompilowany, w celu zwiększenia wydajności. Dla każdego pliku *.aspx* tworzona jest odpowiadająca mu klasa, która dziedziczy z `System.Web.UI.Page`. Definiuje to równoznacznie podstawową funkcjonalność zapewnianą przez każdą stronę.

Dla pliku *.asax* jest podobnie, z tym, że odpowiadająca mu klasa dziedziczy z `System.Web.HttpApplication`. Możesz sprawdzić w dokumentacji, że obiekt klasy `HttpApplication` posiada nie tylko zdarzenia, których obsługę zaproponował WebMatrix — interesującym nas zdarzeniem jest tak naprawdę zdarzenie `AuthenticateRequest`. Jest to zdarzenie wywoływane po rozpoznaniu (zalogowaniu) użytkownika, czyli po wywołaniu metody `FormsAuthentication.RedirectFromLoginPage` z pliku *Login.aspx*. Jest ono idealnym miejscem, żeby po zainicjalizowaniu zintegrowanego mechanizmu uwierzytelniania ASP.NET i wysłaniu przekierowania do przeglądarki powiązać użytkownika z rolami.

21. Dodaj między metodami `Application_Error` i `Session_Start` metodę obsługi zdarzenia `AutheticateRequest` aplikacji, jak poniżej:

Przykład 24. Powiązanie użytkownika z rolami po zalogowaniu do aplikacji

```
public void Application_Error(object sender, EventArgs e) {
    // Code that runs when an unhandled error occurs
}

public void Application_AuthenticateRequest(object sender, EventArgs e) {
    // Kod wykonywany po zalogowaniu użytkownika
    if (Request.IsAuthenticated) {
        // Odczytanie ról użytkownika
        System.Data.DataSet roleDS =
            WczytajRole(HttpContext.Current.User.Identity.Name);
        System.Data.DataRowCollection wiersze = roleDS.Tables[0].Rows;
```

```
            // Skopiowanie ról do tablicy napisów
            string[] role = new string[wiersze.Count];
            for (int i = 0; i < wiersze.Count; i++) {
                role[i] = wiersze[i]["rola"].ToString();
            }
            // Stworzenie nowego opisu zalogowanego użytkownika,
            // uwzględniającego przypisane mu role
            System.Security.Principal.GenericPrincipal uzytkownik =
                new System.Security.Principal.GenericPrincipal(
                    HttpContext.Current.User.Identity, role);
            // Wymiana opisu użytkownika
            HttpContext.Current.User = uzytkownik;
        }
    }

    public void Session_Start(object sender, EventArgs e) {
        // Code that runs when a new session is started
    }
```

Powyższy kod wymaga komentarza. Jeśli użytkownik został poprawnie zalogowany (`Request.IsAuthenticated` zwraca `true`), to odczytywane są role użytkownika i zapamiętywane w obiekcie `DataSet`. Obiekt ten posiada jedną potrzebną właściwość — pozwala dowiedzieć się, ile wierszy zostało odczytanych z tabeli, której dotyczyło zapytanie (tu z tabeli `role`). Następnie tworzona jest jednowymiarowa tablica napisów (`string[]`) takiego samego rozmiaru i kopiowane są do niej nazwy ról. Kluczem do powiązania użytkownika z rolami jest utworzenie nowego obiektu opisującego użytkownika `System.Security.Principal.GenericPrincipal`, wraz z podaniem loginu (`HttpContext.Current.User.Identity`) oraz tablicy ról. Taki obiekt opisu użytkownika zastępuje stary opis użytkownika, który nie ma przypisanych ról (`HttpContext.Current.User`).

22. Zapisz wszystkie wprowadzone zmiany i sprawdź, czy można się zalogować i wylogować z aplikacji. Sprawdź, czy ASP.NET ogranicza dostęp do zabezpieczonych stron aplikacji oraz czy zmienia się prawidłowo zawartość menu nawigacyjnego.

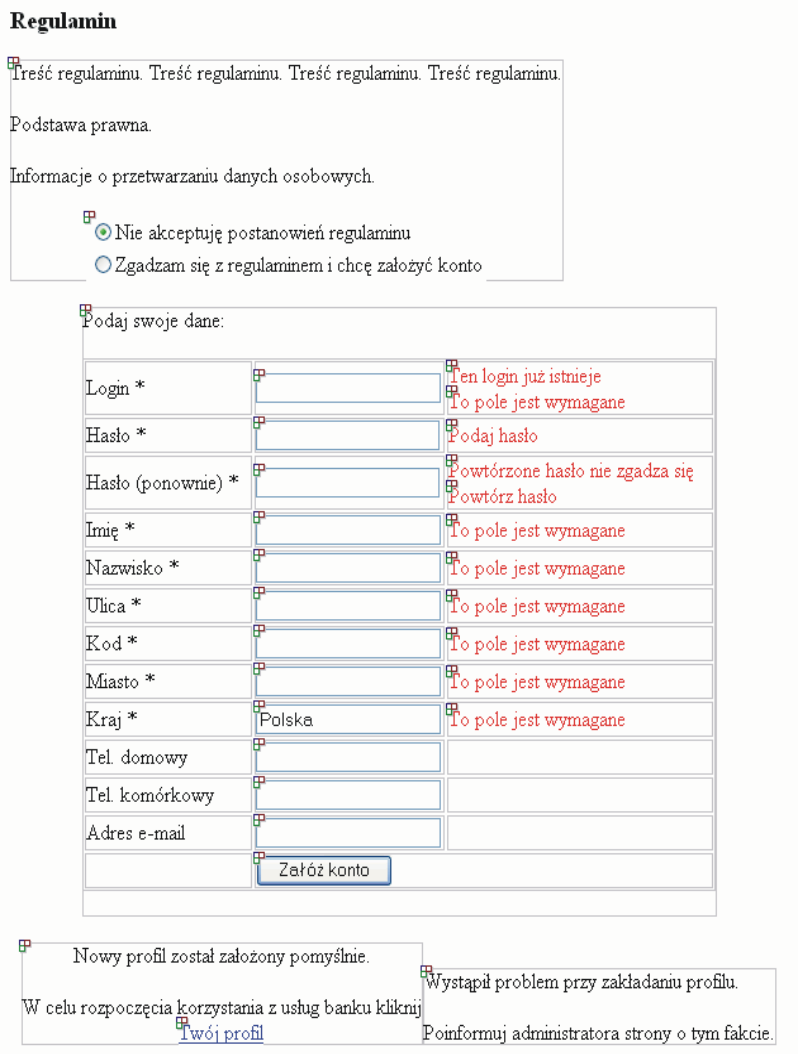
5. Tworzenie nowych danych

Zaimplementowanie dokładnego mechanizmu kontroli dostępu do pojedynczych stron aplikacji, opartego o wpisy w pliku *web.config*, ma uzasadnienie w przypadku aplikacji bankowej. Przyjrzymy się jeszcze, jak klienci banku mogą dodawać nowe konta (loginy) do bazy danych. Jest to o tyle ciekawe, że wymaga modyfikacji aż trzech tabel: `klienci` (do wpisania danych osobowych), `loginy` (do zapamiętania loginu i skrótu hasła) oraz `role` (aby połączyć nowy login z rolą `klient`). Wszystkie te operacje muszą się powieść, aby konto zostało poprawnie utworzone. Jeśli na którymś z etapów wystąpi błąd, wprowadzone dotąd modyfikacje muszą zostać wycofane. Jest to dokładnie takie zachowanie, jakiego oczekuje się po transakcji. Zobaczysz więc teraz jak dodać obsługę transakcji w aplikacji ASP.NET.

Strona do zakładania konta

Jeśli klient banku będzie chciał założyć konto, musi najpierw zaakceptować regulamin. Wykorzystasz ten fakt, aby do strony *Regulamin.aspx* dodać bogaty zbiór kontrolek, służący do podania danych osobowych. Będą one wyświetlane w momencie, gdy klient zdecyduje się zaakceptować treść regulaminu. Następnie po wciśnięciu przycisku zostanie wykonana próba założenia konta i pojawi się komunikat o sukcesie lub komunikat o błędzie. Rozbuduj więc stronę, wykonując poniższe czynności:

1. Rozpocznij od otwarcia istniejącej strony *Regulamin.aspx*. Stwórz na niej interfejs użytkownika zgodnie z opisem z następnych punktów, jak na rysunku poniżej:



Rysunek 8. Opis rysunku

2. Pod tytułem strony (*Regulamin*) wstaw panel i nazwij go *PanelRegulamin*. Wpisz tam treść regulaminu, który użytkownik będzie musiał zaakceptować, jeśli będzie chciał założyć konto.

3. Pod tekstem, na tym samym panelu wstaw grupę przycisków radiowych *RadioButtonList* i nazwij tę kontrolkę `RblAkceptacja`.
4. Otwórz okno edycji elementów wstawionej kontrolki (kolekcja *Items*) i dodaj dwa elementy, które staną się przyciskami radiowymi. Pierwszemu z nich ustaw właściwości: *Selected* na `True` oraz *Value* na *Odmowa*, zaś *Value* na *Zgoda*.
5. Wstaw teraz kolejny panel i nazwij go *PanelNoweKonto*. Dodaj na nim tabelę o 13 wierszach i 3 kolumnach.
6. Wstaw nazwy pól w pierwszej kolumnie, kontrolki `TextBox` w drugiej kolumnie oraz kontrolki walidatorów w trzeciej kolumnie — ale tylko dla pól, które w bazie danych mają ustawioną właściwość *Nullable* równą `False`.
7. Zauważ, że w pierwszym wierszu (login) potrzebne są walidatory: `CustomValidator` oraz `RequiredFieldValidator`, zaś w trzecim (hasło ponownie) potrzebne są walidatory: `CompareValidator` i `RequiredFieldValidator`. W pozostałych wierszach umieść tylko waidatory `RequiredFieldValidator`.
8. Połącz w ostatnim wierszu komórki kolumny drugiej i trzeciej i wstaw w tak powstałą komórkę przycisk zakładający konto. Nazwij go *BtnZaloz*.
9. Dodaj pod spodem dwa panele i nazwij je *PanelSukces* i *PanelProblem*.
10. Na pierwszym z nich wpisz komunikat o powodzeniu oraz wstaw kontrolkę `HyperLink` prowadzącą do strony *klient/Default.aspx*.
11. Na drugim panelu wpisz komunikat o błędzie przy zakładaniu konta.
12. Ustaw właściwości kontrolek zgodnie z poniższą tabelą:

Tabela 6. Kontrolki strony *Regulamin.aspx*

Typ kontrolki	Nazwa właściwości	Wartość właściwości
Panel	(ID)	PanelRegulamin
RadioButtonList	(ID)	RblAkceptacja
	AutoPostBack	True
Items.ListItem[0]	Selected	True
	Text	Nie akceptuję postanowień regulaminu
	Value	odmowa
Items.ListItem[1]	Text	Zgadzam się z regulaminem i chcę założyć konto
	Value	zgoda
Panel	(ID)	PanelNoweKonto
	BorderWidth	1
	Width	500px
TextBox	(ID)	TxtLogin
	MaxLength	20
TextBox	(ID)	TxtPassword
	EnableViewState	False
	MaxLength	30
	TextMode	Password
TextBox	(ID)	TxtPassTest
	EnableViewState	False
	MaxLength	30

	TextMode	Password
TextBox	(ID)	TxtImie
	MaxLength	50
TextBox	(ID)	TxtNazwisko
	MaxLength	80
TextBox	(ID)	TxtAdres
	MaxLength	60
TextBox	(ID)	TxtKod
	MaxLength	10
TextBox	(ID)	TxtMiasto
	MaxLength	40
TextBox	(ID)	TxtKraj
	MaxLength	30
TextBox	(ID)	TxtDomowy
	MaxLength	25
TextBox	(ID)	TxtKomorkowy
	MaxLength	25
TextBox	(ID)	TxtEmail
	MaxLength	50
CustomValidator	(ID)	CustomValidatorLogin
	ControlToValidate	TxtLogin
	Display	Dynamic
	ErrorMessage	Ten login już istnieje
RequiredFieldValidator	(ID)	RequiredFieldValidatorLogin
	ControlToValidate	TxtLogin
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorLogin
	ControlToValidate	TxtLogin
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorPass
	ControlToValidate	TxtPassword
	Display	Dynamic
	ErrorMessage	Podaj hasło
CompareFieldValidator	(ID)	CompareFieldValidator
	ControlToCompare	TxtPassword
	ControlToValidate	TxtPassTest
	Display	Dynamic
	ErrorMessage	Powtórzone hasło nie zgadza się
RequiredFieldValidator	(ID)	RequiredFieldValidatorPassTest
	ControlToValidate	TxtPassTest
	Display	Dynamic
	ErrorMessage	Powtórz hasło
RequiredFieldValidator	(ID)	RequiredFieldValidatorImie
	ControlToValidate	TxtImie
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorNazwisko
	ControlToValidate	TxtNazwisko
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorAdres
	ControlToValidate	TxtAdres
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorKod
	ControlToValidate	TxtKod
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorMiasto
	ControlToValidate	TxtMiasto
	Display	Dynamic
	ErrorMessage	To pole jest wymagane
RequiredFieldValidator	(ID)	RequiredFieldValidatorKraj
	ControlToValidate	TxtKraj
	Display	Dynamic
	ErrorMessage	To pole jest wymagane

Button	(ID)	BtnZaloz
	Text	Załóż konto
Panel	(ID)	PanelSukces
	Visible	False
Panel	(ID)	PanelProblem
	Visible	False

13. Po wstawieniu wszystkich kontrolek ukryj *PanelNoweKonto*, ustawiając jego właściwość *Visible* na `False`. W ten sposób ukryty zostanie sam panel, a nie kontrolki na nim umieszczone.
14. Dla kontrolki `RblAkceptacja` stwórz obsługę zdarzenia `SelectedIndexChanged`. Wpisz kod wyświetlający *PanelNoweKonto*, zależnie od tego, która opcja została wybrana (*Nie akceptuję*/*Zgadzam się*):

Przykład 25. Wyświetlanie panelu do zakładania nowego konta

```
void RblAkceptacja_SelectedIndexChanged(object sender, EventArgs e) {
    PanelNoweKonto.Visible = (RblAkceptacja.SelectedIndex == 1);
}
```

15. Stwórz metodę odczytu danych, która będzie sprawdzać, czy login podany przez użytkownika jest zajęty. Przeciągnij więc poniżej metody `RblAkceptacja_SelectedIndexChanged` generator kodu *SELECT Data Method*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
16. W polu wyboru *Select a database* wybierz połączenie z bazą danych *bank.mdb* (jeśli jest otwarta) lub wskaż pozycję <*New Database Connection*> i wciśnij przycisk *Create...*, aby stworzyć nowe połączenie do tej bazy.
17. Kliknij w przycisk *Next*.
18. W liście tabel wskaż tabelę `loginy`. Zaznacz w liście kolumn: `login`.
19. Kliknij w przycisk *WHERE*.
20. Kliknij w przycisk *OK* — spowoduje to dodanie kolumny `login` jako parametru zapytania.
21. Kliknij w przycisk *Next*.
22. Jeśli chcesz, możesz sprawdzić budowę zapytania klikając przycisk *Test Query*.
23. Kliknij w przycisk *Next*.
24. W polu nazwy metody wpisz `ZnajdzLogin`.
25. Kliknij w przycisk radiowy *DataReader*.
26. Kliknij w przycisk *Finish*.
27. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `ZnajdzLogin`. Jako parametr wejściowy przyjmuje ona login użytkownika, a w wyniku działania zwraca obiekt `DataReader`.

28. Wygenerowany w ten sposób kod ma wpisaną pełną ścieżkę dostępu do pliku *bank.mdb*. Popraw początek metody `WczytajLogin`, aby ścieżka do bazy była odczytywana z pliku *web.config*:

Przykład 26. Wyszukanie w bazie danych loginu proponowanego przez nowego klienta

```
System.Data.IDataReader ZnajdzLogin(string login) {
    string connectionString =
        ConfigurationSettings.AppSettings["connectionstring"];
    System.Data.IDbConnection dbConnection = new
        System.Data.OleDb.OleDbConnection(connectionString);

    string queryString = "SELECT [loginy].[login] FROM [loginy] " +
        "WHERE ([loginy].[login] = @login)";
    ...
```

29. Dla kontrolki `CustomValidatorLogin` stwórz obsługę zdarzenia `ServerValidate`. Wpisz kod sprawdzający, czy proponowany login jest unikatowy:

Przykład 27. Samodzielna obsługa walidacji (tu: sprawdzenie, czy login jest unikatowy

```
void CustomValidatorLogin_ServerValidate(object sender,
    ServerValidateEventArgs e)
{
    e.IsValid = true;
    try {
        System.Data.OleDb.OleDbDataReader dr =
            (System.Data.OleDb.OleDbDataReader) ZnajdzLogin(
                e.Value.Trim());
        if (dr.HasRows) {
            e.IsValid = false;
        }
    }
    catch (Exception ex) {
        e.IsValid = false;
    }
}
```

Zauważ, że w powyższym przykładzie odpowiedź na pytanie, czy walidacja przebiegła poprawnie polega na odpowiednim ustawieniu pola `IsValid` (na `true` — gdy dane są poprawne lub `false` — gdy dane są niepoprawne) w podanym argumencie typu

`ServerValidateEventArgs`. Jeśli odczyt z bazy powiódł się i dostajemy jeden wiersz odpowiedzi (login istnieje) lub jeśli pojawia się błąd — oznacza to, że walidacja nie powiodła się i ma zostać wyświetlony komunikat kontrolki walidującej.

Teraz pora na najciekawsze. Trzeba wstawić generatory kodu dodające nowe wiersze w trzech tabelach (*klienci*, *loginy* oraz *role*). Następnie należy tak zmienić kod wygenerowanych metod, żeby działały one na jednym połączeniu i na jednym obiekcie polecenia, co pozwoli objąć cały ten zbiór operacji w transakcję.

30. Przeciągnij poniżej metody `CustomValidatorLogin_ServerValidate` generator kodu *INSERT Query Builder*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
31. Aktywne jest ostatnie, właściwe połączenie. Kliknij w przycisk *Next*.
32. Na ekranie pojawi się okno dialogowe *Construct an INSERT Query*.
33. W liście tabel wskaż tabelę *klienci*.
34. Tu można podać samodzielnie domyślne wartości dla kolumn. Ponieważ jednak wszystkie parametry mają być wpisywane przez użytkownika, wystarczy kliknąć w przycisk *Next*.
35. W polu nazwy metody wpisz `DodajKlienta`. Kliknij w przycisk *Finish*.
36. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `DodajKlienta`. Jako parametry wejściowe przyjmuje ona dane potrzebne do założenia nowego konta dla klienta.
37. Wygenerowany w ten sposób kod ma wpisaną pełną ścieżkę dostępu do pliku *bank.mdb*. Popraw początek metody `DodajKlienta`, aby ścieżka do bazy była odczytywana z pliku *web.config*:

Przykład 28. Dodawanie do bazy nowego klienta

```
int DodajKlienta(string imie, string nazwisko, string adres,
    string kod, string miasto, string kraj, string domowy,
    string komorkowy, string email)
{
    string connectionString =
        ConfigurationSettings.AppSettings["connectionstring"];
    System.Data.IDbConnection dbConnection = new
        System.Data.OleDb.OleDbConnection(connectionString);

    string queryString = "INSERT INTO [klienci] ([imie], " +
        "[nazwisko], [adres], [kod], [miasto], [kraj], " +
        "[domowy], [komorkowy], [email]) " +
        "VALUES (@imie, @nazwisko, @adres, @kod, @miasto, " +
```

```
        "@kraj, @domowy, @komorkowy, @email)";

    ...
```

38. Ponieważ w tabeli klienci kolumny `domowy`, `komorkowy` i `email` mogą mieć wartości *NULL*, dodaj kod ustawiający takie wartości, gdy użytkownik nie poda żadnych danych:

Przykład 29. Wstawianie do tabeli wartości `NULL`

```
System.Data.IDataParameter dbParam_domowy =
    new System.Data.OleDb.OleDbParameter();
dbParam_domowy.ParameterName = "@domowy";
dbParam_domowy.Value = domowy;
dbParam_domowy.DbType = System.Data.DbType.String;
if (domowy == String.Empty) {
    dbParam_domowy.Value = DBNull.Value;
}
dbCommand.Parameters.Add(dbParam_domowy);
System.Data.IDataParameter dbParam_komorkowy =
    new System.Data.OleDb.OleDbParameter();
dbParam_komorkowy.ParameterName = "@komorkowy";
dbParam_komorkowy.Value = komorkowy;
dbParam_komorkowy.DbType = System.Data.DbType.String;
if (komorkowy == String.Empty) {
    dbParam_komorkowy.Value = DBNull.Value;
}
dbCommand.Parameters.Add(dbParam_komorkowy);
System.Data.IDataParameter dbParam_email =
    new System.Data.OleDb.OleDbParameter();
dbParam_email.ParameterName = "@email";
dbParam_email.Value = email;
dbParam_email.DbType = System.Data.DbType.String;
if (email == String.Empty) {
    dbParam_email.Value = DBNull.Value;
}
dbCommand.Parameters.Add(dbParam_email);
```

39. Przeciągnij poniżej metody `DodajKlienta` generator kodu *INSERT Query Builder*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
40. Aktywne jest ostatnie, właściwe połączenie. Kliknij w przycisk *Next*.

41. Na ekranie pojawi się okno dialogowe *Construct an INSERT Query*.
42. W liście tabel wskaż tabelę *loginy*.
43. Kliknij w przycisk *Next*.
44. W polu nazwy metody wpisz `DodajLogin`. Kliknij w przycisk *Finish*.
45. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `DodajLogin`. Jako parametry wejściowe przyjmuje ona login, skrót hasła oraz identyfikator użytkownika.
46. Zmień treść tej metody tak, aby nie tworzyła nowych, ale wykorzystywała podane obiekty `dbConnection` i `dbCommand`:

Przykład 30. Modyfikacja metody `DodajLogin`

```
int DodajLogin(System.Data.IDbConnection dbConnection,
    System.Data.IDbCommand dbCommand,
    string login, string password, int userid)
{
    string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;";
    System.Data.IDbConnection dbConnection =
        new System.Data.OleDb.OleDbConnection(connectionString);

    string queryString = "INSERT INTO [loginy] " +
        "([login], [password], [userid]) " +
        "VALUES (@login, @password, @userid)";
    System.Data.IDbCommand dbCommand =
        new System.Data.OleDb.OleDbCommand();
    dbCommand.CommandText = queryString;
    dbCommand.Connection = dbConnection;
    dbCommand.Parameters.Clear();

    System.Data.IDataParameter dbParam_login =
        new System.Data.OleDb.OleDbParameter();
    dbParam_login.ParameterName = "@login";
    dbParam_login.Value = login;
    dbParam_login.DbType = System.Data.DbType.String;
    dbCommand.Parameters.Add(dbParam_login);
    System.Data.IDataParameter dbParam_password =
        new System.Data.OleDb.OleDbParameter();
    dbParam_password.ParameterName = "@password";
    dbParam_password.Value = password;
    dbParam_password.DbType = System.Data.DbType.String;
    dbCommand.Parameters.Add(dbParam_password);
```

```
        System.Data.IDataParameter dbParam_userid =
            new System.Data.OleDb.OleDbParameter();
        dbParam_userid.ParameterName = "@userid";
        dbParam_userid.Value = userid;
        dbParam_userid.DbType = System.Data.DbType.Int32;
        dbCommand.Parameters.Add(dbParam_userid);

        int rowsAffected = 0;
        dbConnection.Open();
        try {
            rowsAffected = dbCommand.ExecuteNonQuery();
        }
        finally {
            dbConnection.Close();
        }

        return rowsAffected;
    }
```

47. Przeciągnij poniżej metody `DodajLogin` generator kodu *INSERT Query Builder*. Na ekranie pojawi się okno dialogowe *Select a database connection*.
48. Aktywne jest ostatnie, właściwe połączenie. Kliknij w przycisk *Next*.
49. Na ekranie pojawi się okno dialogowe *Construct an INSERT Query*.
50. W liście tabel wskaż tabelę *role*.
51. W liście kolumn kliknij w kolumnę `rola`. Pojawi się okno dialogowe *Set Value*, pozwalające na podanie wartości dla tej kolumny.
52. W polu wartości wpisz nazwę roli: `klient`. Kliknij w przycisk *OK*.
53. Kliknij w przycisk *Next*.
54. W polu nazwy metody wpisz `DodajRoleKlient`. Kliknij w przycisk *Finish*.
55. Okno kreatora zniknie, a w widoku *Code* pojawi się metoda `DodajRoleKlient`. Jako parametry wejściowe przyjmuje ona login oraz rolę, do której będzie przypisany użytkownik.
56. Zmień treść tej metody tak, aby nie tworzyła nowych, ale wykorzystywała podane obiekty `dbConnection` i `dbCommand`:

Przykład 31. Modyfikacja metody `DodajRoleKlient`

```
    int DodajRoleKlient(System.Data.IDbConnection dbConnection,
        System.Data.IDbCommand dbCommand,
        string login)
```

```
    {
        string connectionString = "Provider=Microsoft.Jet.OLEDB.4.0;";
        System.Data.IDbConnection dbConnection =
            new System.Data.OleDb.OleDbConnection(connectionString);

        string queryString = "INSERT INTO [role] ([login], [rola]) " +
            "VALUES (@login, \'klient\')";
        System.Data.IDbCommand dbCommand =
            new System.Data.OleDb.OleDbCommand();
        dbCommand.CommandText = queryString;
        dbCommand.Connection = dbConnection;
        dbCommand.Parameters.Clear();

        System.Data.IDataParameter dbParam_login =
            new System.Data.OleDb.OleDbParameter();
        dbParam_login.ParameterName = "@login";
        dbParam_login.Value = login;
        dbParam_login.DbType = System.Data.DbType.String;
        dbCommand.Parameters.Add(dbParam_login);

        int rowsAffected = 0;
        dbConnection.Open();
        try {
            rowsAffected = dbCommand.ExecuteNonQuery();
        }
        finally {
            dbConnection.Close();
        }

        return rowsAffected;
    }
```

57. Powróć teraz do metody `DodajKlienta` i zmodyfikuj jej kod tak, aby tworzył transakcję i używał pozostałych metod `DodajLogin` i `DodajRoleKlient`. Zauważ, że metoda musi teraz przyjmować dwa dodatkowe parametry (`login` i `password`), ponieważ są one potrzebne przy wywołaniu `DodajRoleKlient`.

Przykład 32. Obsługa transakcji zapewniająca atomowość zmian w trzech tabelach

```
int DodajKlienta(string login, string password,
    string imie, string nazwisko, string adres,
    string kod, string miasto, string kraj, string domowy,
```

```
        string komorkowy, string email)
    {
        string connectionString =
            ConfigurationSettings.AppSettings["connectionstring"];
        . . .
        if (email == String.Empty) {
            dbParam_email.Value = DBNull.Value;
        }
        dbCommand.Parameters.Add(dbParam_email);

        int rowsAffected = 0;
        // Flaga mówiąca o tym, czy zatwierdzić transakcję - domyślnie nie.
        bool commit = false;
        System.Data.OleDb.OleDbTransaction trans = null;
        dbConnection.Open();
        try {
            // Rozpoczęcie transakcji
            // (Uwaga: potrzebne jest rzutowanie, bo dbConnection zwraca
            // uchwyt do IdbTransaction, a potrzebujemy OleDbTransaction)
            trans = (System.Data.OleDb.OleDbTransaction)
                dbConnection.BeginTransaction();
            dbCommand.Transaction = trans;
            rowsAffected = dbCommand.ExecuteNonQuery();

            // Test czy udało się stworzyć dane w tabeli [klienci]
            if (rowsAffected == 1) {

                // Odczyt klucza głównego ostatnio dodanego wiersza
                dbCommand.CommandText = "SELECT @@IDENTITY";
                int userid = (int) dbCommand.ExecuteScalar();

                // Tworzenie danych w tabeli [loginy]
                rowsAffected = DodajLogin(dbConnection, dbCommand,
                    login, password, userid);
                if (rowsAffected == 1) {
                    // Tworzenie roli 'klient' w tabeli [role]
                    rowsAffected = DodajRoleKlient(dbConnection,
                        dbCommand, login);
                    if (rowsAffected == 1) {
                        // Dopiero teraz można zatwierdzić transakcję
                        commit = true;
                    }
```

```
                }
            }
        }
        catch (Exception ex) {
            // W reakcji na błąd zerujemy ilość zmodyfikowanych wierszy
            rowsAffected = 0;
        }
        finally {
            // Kończymy. Czy jest obiekt transakcji?
            if (trans != null) {
                // Czy zatwierdzić transakcję?
                if (commit)
                    trans.Commit();
                else
                    // Wycofanie tranakcji i wszystkich zmian w tabelach
                    trans.Rollback();
            }
            dbConnection.Close();
        }

        // Wynikiem jest ilość zmodyfikowanych wierszy w tabelach.
        return rowsAffected;
    }
```

Warto zauważyć jeszcze, w jaki sposób dowiadujemy się o wartości klucza głównego tabeli *klienci*. Nie jest to oczywiste, ponieważ kolumna ta ma typ `AutoNumber`, więc wartości są nadawane automatycznie przez silnik bazy danych. Otóż zaraz po wykonaniu wstawienia danych (`INSERT`) do tabeli `klienci` wykonywane jest polecenie SQL `SELECT @@IDENTITY` przez metodę `ExecuteScalar`. Wynik tego polecenia jest rzutowany do liczby całkowitej `int`, co w efekcie daje odczytanie wartości, która została wstawiona w kolumnie `id` tabeli. Należy podkreślić, że taki mechanizm (`@@IDENTITY`) działa tylko w bazach danych Access i MS SQL Server. Dla innych baz danych trzeba sprawdzić w instrukcji, jak odczytać podobną informację.

58. W takim razie jedyne, co jeszcze pozostaje, to dodać dla przycisku *BtnZaloz* obsługę zdarzenia `Click` oraz wpisanie poniższego kodu:

Przykład 33. Zakładanie nowego konta klienta i wyświetlenie informacji o wyniku

```
void BtnZaloz_Click(object sender, EventArgs e) {
    Page.Validate();
    if (Page.IsValid) {
```

```
        // Przygotowanie skrótu hasła
        string PasswordHash =
            FormsAuthentication.HashPasswordForStoringInConfigFile(
                TxtPassword.Text, "SHA1");
        // Tworzenie danych w trzech tabelach - w ramach transakcji
        int dodane = DodajKlienta(TxtLogin.Text, PasswordHash,
            TxtImie.Text, TxtNazwisko.Text, TxtAdres.Text,
            TxtKod.Text, TxtMiasto.Text, TxtKraj.Text,
            TxtDomowy.Text, TxtKomorkowy.Text, TxtEmail.Text);

        // Zmiany w wyglądzie strony
        PanelRegulamin.Visible = false;
        PanelNoweKonto.Visible = false;
        // Czy operacja powiodła się?
        if (dodane == 1) {
            // Konto założone poprawnie - w większości przypadków
            PanelSukces.Visible = true;
        }
        else {
            // Wyświetlenie informacji o problemie przy zakładaniu konta
            PanelProblem.Visible = true;
        }
    }
}
```

59. Jeśli wszystko poszło dobrze, dodawanie nowych klientów powinno udawać się za każdym razem. Pod warunkiem oczywiście, że podawane dane są prawidłowe.

Podsumowanie

W ten sposób aplikacja bankowa została zbudowana zgodnie z założeniami kursu. Posiada ona spójny interfejs użytkownika, dynamiczne menu nawigacyjne, ma możliwość logowania i wylogowania się, zaś informacje o użytkownikach i ich rolach są przechowywane w bazie danych. Zaprezentowane przy tym mechanizmy wystarczają do dalszej rozbudowy tej aplikacji o kolejną funkcjonalność. Może więc to być np. dalsza obsługa klientów banku (zakładanie kont, wykonywanie przelewów, lokat terminowych), czy samodzielna obsługa kont przez właścicieli (zgłaszanie prośby o założenie konta, wykonywanie przelewów, zarządzanie zleceniami stałymi). Przy operacjach tego typu często potrzebna będzie zaprezentowana w tym module obsługa transakcji.

Poznane w czasie kursu techniki związane z wykorzystaniem m.in. ASP.NET, ADO.NET i środowiska Web Matrix można wykorzystać także przy tworzeniu innych aplikacji działających na serwerze. Warto jednak nadmienić, że w wielu przypadkach aplikacje takie będą potrzebować podobnej funkcjonalności (jak np. logowania, dynamicznie zmieniających się elementów strony, dostępu do baz danych, czy wreszcie — opartego o spójny schemat — wyglądu). Wszystkim, którzy dotarli do końca kursu i uważają, że nie był to czas stracony, należy pogratulować wytrwałości i życzyć skutecznego wykorzystania zdobytej wiedzy.