ASP.NET 3.5 z wykorzystaniem C# i VB. Zaawansowane programowanie

Autor: Bill Evjen, Scott Hanselman, Devin Rader
Tłumaczenie: Paweł Dyl, Marek Pałczyński
ISBN: 978-83-246-1852-1
Tytuł oryginału: Professional ASP.NET 3.5: in C# and VB
Format: 172x245 , stron: 1792
Oprawa: twarda

Kompletne źródło informacji na temat ASP.NET!

- Jak rozpocząć pracę z ASP.NET?
- Jakie kontrolki dostarcza ASP.NET?
- Jak wykorzystać technologię AJAX?

ASP.NET powstał jako odpowiedź firmy Microsoft na rosnącą popularność aplikacji internetowych. Jest on częścią technologii .NET oraz bezpośrednim następcą rozwiązania ASP, dostępnym na rynku od stycznia 2002 roku. Od tego czasu ASP.NET zdobywa sobie coraz większą popularność. Teraz dzięki wykorzystaniu technologii .NET możesz projektować dynamiczne strony, korzystając z dowolnego języka obsługującego to rozwiązanie.

Dzięki tej książce dowiesz się, jak to robić przy użyciu języków C# i Visual Basic. Jednak zanim napiszesz pierwszą linię kodu, warto poznać nowe środowisko pracy,tak aby tworzenie stron przebiegało szybko, wygodnie i bezproblemowo. W kolejnych rozdziałach zdobędziesz wiedzę na temat dostępnych kontrolek, metod pracy ze stronami wzorcowymi oraz sposobów stosowania kompozycji i skórek. Ponadto nauczysz się wykorzystywać źródła danych oraz zarządzać informacjami przy użyciu ADO.NET. Książka ta sprawdzi się znakomicie zarówno w rękach początkującego, jak i zaawansowanego programisty.

- Środowisko pracy
- Dyrektywy strony ASP.NET 3.5
- Obsługa zdarzeń
- Wykorzystanie kontrolek serwerowych
- Wykonywanie skryptów po stronie klienta
- Walidacja danych
- Zastosowanie kompozycji i skórek
- Źródła danych
- Zarządzanie danymi przy użyciu ADO.NET
- Tworzenie zapytań z wykorzystaniem LINQ
- Konfiguracja i uruchomienie usługi IIS7
- Model dostawców
- Platformy portalowe
- Wsparcie dla HTML i CSS
- Wykorzystanie technologii AJAX
- Bezpieczeństwo aplikacji internetowych w ASP.NET
- Obsługa sytuacji wyjątkowych
- Debugowanie kodu
- Wykorzystanie plików i strumieni
- Obsługa żądań HTTP
- Możliwości Silverlight

Twórz dynamiczne rozwiązania, korzystając z nowoczesnych technologii!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

4

Walidacyjne kontrolki serwerowe

Patrząc na okno *Toolbox* w Visual Studio 2008 — zwłaszcza wtedy, gdy czyta się rozdziały 2. i 3., w których omówiono różne kontrolki serwerowe pozostające do dyspozycji — można być porażonym ilością kontrolek serwerowych udostępnianych przez ASP.NET 3.5. W niniejszym rozdziale omówiono specyficzny typ kontrolek serwerowych, które można znaleźć w oknie Toolbox: **walidacyjne kontrolki serwerowe**.

Walidacyjne kontrolki serwerowe są serią kontrolek, które pozwalają pracować z danymi wprowadzonymi przez użytkowników końcowych w elementach formularza tworzonej aplikacji. Kontrolki pozwalają zadbać o poprawność danych wpisywanych na formularzu.

Zanim przejdziemy do omówienia sposobów ich użycia, przyjrzyjmy się dokładnie procesowi walidacji.

Zrozumienie procesu walidacji

Ludzie tworzyli aplikacje internetowe przez wiele lat. Zwykle było to spowodowane potrzebą udostępnienia lub pobrania informacji. W tym rozdziale skupimy się na aspekcie pobierania informacji przez aplikacje internetowe. Podczas pobierania danych w aplikacji ważne jest to, aby były to dane **poprawne**. Jeżeli dane nie są poprawne, wtedy nie ma większego sensu w gromadzeniu ich.

Walidacja jest procesem wielostopniowym i stanowi zbiór reguł, które nakłada się na zbierane dane. Tych reguł może być dużo lub mało i mogą być ścisłe lub dość luźne. Zależy to jedynie od potrzeb twórcy aplikacji. Nie istnieje żaden perfekcyjny sposób walidacji, ponieważ niektórzy użytkownicy mogą znaleźć jakiś sposób oszukania tych procedur, bez względu na zastosowane reguły. Cały problem tkwi w znalezieniu złotego środka pomiędzy niewielką ilością zasad oraz ścisłą kontrolą, która nie będzie miała wpływu na użyteczność aplikacji.

Dane zbierane do procesu walidacji pochodzą z formularzy aplikacji. Formularze zbudowane są z różnych typów elementów HTML, które są tworzone za pomocą tradycyjnych elementów HTML, kontrolek serwerowych HTML ASP.NET oraz kontrolek serwerowych Web ASP.NET. Wszystko to na końcu i tak staje się zbiorem elementów HTML wchodzącym w skład formularzy. Są to na przykład pola tekstowe, przyciski opcji, przyciski wyboru, listy rozwijane i wiele innych.

Pracując z przykładami zaprezentowanymi w tym rozdziale, będzie można zauważyć różne typy reguł walidacyjnych, które można dodać do elementów formularza. Należy pamiętać, że nie ma możliwości sprawdzenia, czy dane są prawdziwe. Można jedynie wprowadzić zasady, które pomagają odpowiedzieć na pytania typu:

- Czy coś zostało wpisane w polu tekstowym?
- Czy dane wpisane w polu tekstowym posiadają format adresu e-mail?

Warto także zwrócić uwagę na to, że możliwe jest zastosowanie więcej niż jednej reguły walidacji do elementu formularza HTML (przykłady zostaną pokazane w dalszej części tego rozdziału). W rzeczywistości do każdego elementu można zastosować tyle reguł walidacyjnych, ile tylko potrzeba. Dodanie kolejnych reguł do elementów zwiększa poziom sprawdzania poprawności danych.

Należy pamiętać, że pobieranie danych na stronach internetowych jest jedną z najważniejszych funkcji internetu. Należy więc zadbać o to, aby zebrane dane posiadały pewną wartość i miały jakieś znaczenie. Można o to zadbać, eliminując przypadki, w których zbierane informacje nie spełniają nakreślonych reguł.

Walidacja po stronie klienta a walidacja po stronie serwera

Początkujący twórcy aplikacji internetowych mogą nie być świadomi różnicy pomiędzy walidacją po stronie klienta i walidacją po stronie serwera. Przypuśćmy, że użytkownik końcowy po uzupełnieniu kontrolek formularza naciska przycisk *Zatwierdź*. ASP.NET pakuje formularz do postaci **żądania** i wysyła je do serwera, na którym ta aplikacja jest umieszczona. W tym punkcie cyklu żądanie – odpowiedź można przeprowadzić proces sprawdzania poprawności wprowadzonych informacji. Takie podejście nazywamy **walidacją po stronie serwera**, ponieważ wszystko dzieje się na serwerze.

Z drugiej strony można umieścić skrypt (zwykle pod postacią kodu JavaScript). Wysyłany jest on razem ze stroną do użytkownika końcowego i umożliwia sprawdzenie poprawności danych wprowadzonych do formularza, **zanim** zostanie on przesłany do serwera aplikacji. W tym przypadku mamy do czynienia z **walidacją po stronie klienta**.

Oba typy walidacji mają swoje wady i zalety. Programiści Active Server Pages 2.0/3.0 (w czasach klasycznego ASP) mieli świadomość wad i zalet tych rozwiązań, ponieważ cały proces sprawdzania poprawności danych wykonywali własnoręcznie. Wielu programistów spędziło wiele dni z klasycznym ASP. W tym czasie wprowadzili różne techniki walidacji, które spełniają pewne wymagania związane z wydajnością i bezpieczeństwem.

Walidacja po stronie klienta jest szybka. Użytkownik natychmiast otrzymuje odpowiedź. To coś, czego spodziewają się użytkownicy końcowi na każdym formularzu. Jeżeli z formularzem coś jest nie tak, wtedy walidacja po stronie klienta powoduje, że użytkownik natychmiast jest o tym informowany. Walidacja po stronie klienta przerzuca obowiązek przetwarzania danych i sprawdzania ich poprawności na klienta. Oznacza to, że nie trzeba używać mocy obliczeniowej na serwerze do przetwarzania tych samych informacji, ponieważ klient wykonał już całą pracę.

Jak można się domyślić na podstawie powyższego, walidacja po stronie klienta jest formą sprawdzania poprawności bardziej narażoną na różne niebezpieczeństwa. Gdy strona generowana jest w przeglądarce użytkownika, wtedy można dość łatwo podejrzeć jej źródła (poprzez kliknięcie prawym przyciskiem myszy i wybranie opcji *Pokaż źródła*). Po wykonaniu takiej czynność można zobaczyć cały kod HTML strony. Oprócz tego można obejrzeć cały kod JavaScript, który na tej stronie został umieszczony. Jeżeli poprawność danych sprawdzana jest po stronie klienta, to dla sprawnego hakera nie stanowi żadnego problemu odesłanie spreparowanego formularza (zawierającego wartości, które są przez niego pożądane). Serwer może je wtedy odebrać jako prawidłowe. Istnieją także takie przypadki, gdy użytkownik zwyczajnie zablokuje w swojej przeglądarce obsługę skryptów — w ten sposób może uczynić walidację całkowicie bezużyteczną.

W związku z powyższym walidacja po stronie klienta powinna być rozważana w kategoriach wygody i ułatwienia życia użytkownikowi końcowemu. Nigdy nie powinien to być mechanizm zapewniający aplikacji bezpieczeństwo.

Bezpieczniejszą formą sprawdzania poprawności danych jest walidacja po stronie serwera. Walidacja po stronie serwera oznacza, że wszystkie procedury kontrolujące poprawność wykonywane są na serwerze, a nie na kliencie. Jest to bezpieczniejsze, ponieważ tego etapu nie da się w łatwy sposób ominąć. Dane na formularzu sprawdzane są przez kod serwera (C# lub VB) na serwerze. Jeżeli formularz nie jest prawidłowy, wtedy odsyłany jest do klienta jako nieprawidłowy. Jest to bezpieczniejsze, ale taka walidacja po stronie serwera może być wolna. Dzieje się tak, ponieważ strona musi być przesłana do zdalnego komputera i tam sprawdzona. Użytkownik końcowy na pewno nie będzie zadowolony, gdy po odczekaniu 20 sekund na odpowiedź dowie się, że wpisał swój adres e-mail w nieprawidłowym formacie.

Jaka jest zatem prawidłowa ścieżka? Generalnie obie są dobre! Najlepszym podejściem jest wykonanie walidacji po stronie klienta, a potem, po przejściu przez ten pierwszy etap i przesłaniu formularza na serwer, wykonywana jest walidacja po stronie serwera. Takie podejście jest najlepsze spośród wszystkich. Jest bezpieczne, ponieważ hakerzy nie mogą tak zwyczajnie ominąć procesu walidacji. Można oszukać walidację po stronie klienta, ale dane i tak zostaną jeszcze raz sprawdzone po przesłaniu ich na serwer. Taki sposób walidacji jest także dość efektywny — pozwala uzyskać szybkość i elegancję walidacji po stronie klienta.

Kontrolki walidacyjne ASP.NET

W czasach klasycznego ASP.NET programiści poświęcali dużo czasu na obsługę różnych schematów walidacji. Z tego powodu wraz z pierwszą wersją ASP.NET wprowadzono serię walidacyjnych kontrolek serwerowych, które pozwalają w łatwy sposób przeprowadzić sprawdzenie poprawności danych formularza.

Samo wprowadzenie walidacyjnych kontrolek serwerowych przez ASP.NET to nie wszystko. Kontrolki są wyjątkowo sprytne. Jak już zostało wcześniej napisane, jednym z zadań programistów klasycznego ASP było zadecydowanie, gdzie przeprowadzać walidację — czy na kliencie, czy na serwerze. Walidacyjne kontrolki serwerowe eliminują ten problem, ponieważ ASP.NET wykrywa przeglądarkę i na tej podstawie podejmuje właściwą decyzję.

Oznacza to, że jeżeli aplikacja obsługuje JavaScript, wtedy ASP.NET przeprowadza walidację po stronie klienta. Jeżeli przeglądarka klienta nie obsługuje JavaScript i walidacji po stronie klienta, wtedy kod JavaScript jest pomijany, a cała walidacja przeprowadzana jest po stronie serwera.

Najlepsze z tego wszystkiego jest to, że pomimo pomyślnego umieszczenia na stronie walidacji po stronie klienta ASP.NET w dalszym ciągu przeprowadza walidację po stronie serwera w momencie otrzymania danej strony. W ten sposób nie ma żadnych kompromisów związanych z bezpieczeństwem. Decyzyjna natura walidacyjnych kontrolek serwerowych oznacza, że można tworzyć strony ASP.NET tak dobre, jak tylko one mogą być. Nie trzeba szukać żadnego wspólnego mianownika bezpieczeństwa i szybkości.

Obecnie w ASP.NET 3.5 dostępnych jest 6 kontrolek walidacyjnych. Od czasu wprowadzenia pierwszej wersji technologii ASP.NET nie pojawiły się żadne nowe kontrolki. W ASP.NET 2.0 wprowadzono jednak kilka nowych możliwości, takich jak grupy walidacji oraz nowe możliwości zastosowania JavaScript. Obie te techniki omówione są w tym rozdziale. Do dyspozycji mamy następujące walidacyjne kontrolki serwerowe:

- `RequiredFieldValidator`,
- `CompareValidator`,
- `RangeValidator`,
- `RegularExpressionValidator`,
- `CustomValidator`,
- `ValidationSummary`.

Praca z walidacyjnymi kontrolkami serwerowymi ASP.NET nie różni się niczym od pracy z innymi kontrolkami serwerowymi ASP.NET. Każda z tych kontrolek może być przeciągnięta i upuszczona na powierzchnię projektową, ale może być także wprowadzona bezpośrednio do kodu strony ASP.NET. Kontrolki mogą być modyfikowane w taki sposób, aby odpowiadały one potrzebom aplikacji. W ten sposób aplikacja może uzyskać unikalny wygląd. Wiele przykładów pracy z tymi kontrolkami pojawi się w dalszej części rozdziału.

> Jeżeli walidacyjne kontrolki serwerowe nie spełniają wszystkich oczekiwań, wtedy zajdzie potrzeba napisania własnych kontrolek walidacyjnych. Istnieją jednak kontrolki napisane przez osoby trzecie, na przykład takie jak Validation and More Petera Bluma (VAM) z *www.peterblum.com/VAM*. Na wspomnianej stronie znajduje się ponad 40 kontrolek walidacyjnych ASP.NET.

W poniższej tabeli opisano funkcjonalność każdej z dostępnych walidacyjnych kontrolek serwerowych.

Walidacyjna kontrolka serwerowa	Opis
`RequiredFieldValidator`	Dba o to, aby użytkownik nie opuścił danego pola formularza.
`CompareValidator`	Pozwala porównać dane wprowadzone przez użytkownika z innym elementem za pomocą operatora porównania (równe, większe niż, mniejsze niż i tak dalej).
`RangeValidator`	Sprawdza, czy wartość wprowadzona przez użytkownika mieści się w podanym zakresie liczb lub znaków.
`RegularExpressionValidator`	Sprawdza, czy wpis użytkownika jest zgodny ze wzorcem zdefiniowanym przez wyrażenie regularne. To dobra kontrolka do sprawdzenia adresu e-mail oraz numeru telefonu.
`CustomValidator`	Sprawdza wpis użytkownika za pomocą własnej logiki walidacyjnej.
`ValidationSummary`	Wyświetla wszystkie komunikaty o błędach wszystkich kontrolek walidacyjnych w jednym miejscu na stronie.

Przyczyny walidacji

Walidacja nie jest przeprowadzana nagle. Pojawia się w wyniku odpowiedzi na zdarzenie. W większości przypadków jest to zdarzenie naciśnięcia przycisku. Kontrolki serwerowe `Button`, `LinkButton` oraz `ImageButton` posiadają możliwość uruchomienia procesu walidacji na formularzu. Jest to zachowanie domyślne. Po przeciągnięciu i upuszczeniu kontrolki `Button` na formularz otrzymujemy następujący rezultat:

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Przeglądając właściwości kontrolki `Button`, można zauważyć, że właściwość `CausesValidation` ustawiona jest na `True`. Jak już wspomniano, jest to ustawienie domyślne — wszystkie przyciski na stronie, bez względu na to, gdzie są, powodują uruchomienie procesu walidacji.

Jeżeli na stronie ASP.NET znajduje się wiele przycisków i nie ma potrzeby, aby każdy z nich wywoływał proces walidacji, wtedy można ustawić właściwość `CausesValidation` na `False` dla tych przycisków, które powinny zignorować proces walidacji (na przykład przycisk *Anuluj*):

```
<asp:Button ID="Button1" runat="server" Text="Anuluj" CausesValidation="false" />
```

Kontrolka serwerowa RequiredFieldValidator

Kontrolka `RequiredFieldValidator` zwyczajnie sprawdza, czy do elementu HTML formularza zostało **coś** wprowadzone. To prosta kontrolka walidacyjna, ale jest ona używana najczęściej. Kontrolka `RequiredFieldValidator` musi być wstawiona dla wszystkich elementów, które muszą spełniać regułę postaci **wartość wymagana**.

Na listingu 4.1 pokazano przykładowy sposób użycia kontrolki serwerowej `Required ↪FieldValidator`.

Listing 4.1. Przykładowy sposób użycia kontrolki serwerowej RequiredFieldValidator

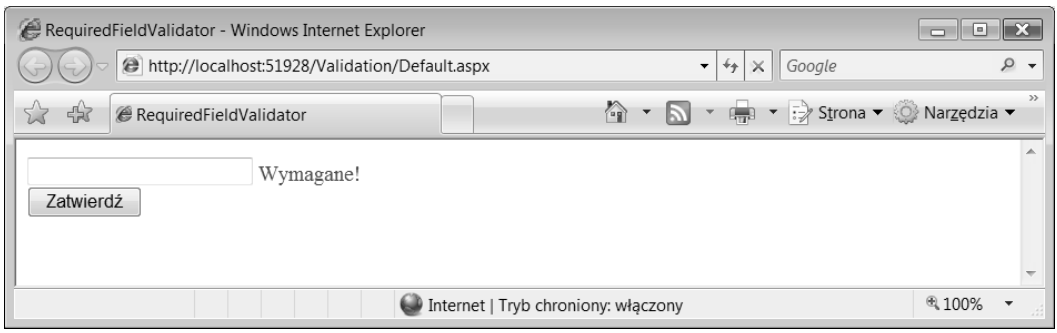
VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
      If Page.IsValid Then
            Label1.Text = "Strona jest prawidłowa!"
      End If
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server" id="Head1">
    <title>RequiredFieldValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
         Runat="server" Text="Wymagane!" ControlToValidate="TextBox1">
        </asp:RequiredFieldValidator>
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Zatwierdź"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" Runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
      if (Page.IsValid) {
       Label1.Text = " Strona jest prawidłowa!";
      }
    }
</script>
```

Zbudujmy stronę i spróbujmy ją uruchomić. Na stronie pokaże się proste pole tekstowe oraz przycisk. Spróbujmy teraz pozostawić pole tekstowe puste i nacisnąć przycisk *Zatwierdź*. Wyniki pokazane są na rysunku 4.1.



Rysunek 4.1

Popatrzmy teraz na kod z przykładu. Na początku można zauważyć, że przy kontrolkach `TextBox`, `Button` i `Label` nic się nie zmieniło. Wstawione są one tak, jakby nie była stosowana żadna technika walidacji. Na stronie umieszczono jednak prostą kontrolkę `RequiredFieldValidator`. Na kilka właściwości tej kontrolki należy zwrócić szczególną uwagę, ponieważ będą stosowane w każdej tworzonej walidacyjnej kontrolce serwerowej.

Pierwszą z tych ważnych właściwości jest `Text`. Wartość tej właściwości pokazywana jest użytkownikowi końcowemu na stronie wtedy, gdy walidacja nie powiedzie się. W pokazanym przykładzie jest to zwykły łańcuch znaków `Wymagane!`. Drugą właściwością, o której warto wspomnieć, jest `ControlToValidate`. Właściwość wykorzystywana jest do połączenia walidacyjnej kontrolki serwerowej z tym elementem formularza ASP.NET, który ma być sprawdzony. W tym przypadku właściwość wskazuje na jedyny element formularza — pole edycji.

W przykładzie można zauważyć, że komunikat o błędzie tworzony jest na podstawie atrybutu kontrolki `<asp:RequiredFieldValidator>`. Zadanie realizowane jest za pomocą atrybutu `Text`. Pokazano to na listingu 4.2.

Listing 4.2. Wykorzystanie atrybutu Text

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text="Wymagane!" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
```

Ten sam komunikat o błędzie można umieścić pomiędzy otwierającym i zamykającym węzłem `<asp:RequiredFieldValidator>`. Pokazano to na listingu 4.3.

Listing 4.3. Umieszczenie wartości pomiędzy węzłami

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ControlToValidate="TextBox1">
 Wymagane!
</asp:RequiredFieldValidator>
```

Sprawdzanie wygenerowanych wyników

Kontrolka `RequiredFieldValidator` używa walidacji po stronie klienta, jeżeli tylko przeglądarka na to pozwala. Można samemu przekonać się o obecności walidacji po stronie klienta (jeżeli przeglądarka pozwala), klikając na stronie prawym przyciskiem myszy i wybierając z menu podręcznego podgląd źródła. W kodzie strony można dostrzec funkcje JavaScript pokazane na listingu 4.4.

Listing 4.4. Wygenerowany kod JavaScript

```
… znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
function WebForm_OnSubmit() {
if (ValidatorOnSubmit() == false) return false;
return true;
}
// -->

… znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
var Page_Validators =  new Array(document.getElementById("RequiredFieldValidator1"));
// -->
</script>

<script type="text/javascript">
<!--
var RequiredFieldValidator1 = document.all ? document.all["RequiredFieldValidator1"] :
document.getElementById("RequiredFieldValidator1");
RequiredFieldValidator1.controltovalidate = "TextBox1";
RequiredFieldValidator1.text = "Wymagane!";
RequiredFieldValidator1.evaluationfunction = "RequiredFieldValidatorEvaluateIsValid";
RequiredFieldValidator1.initialvalue = "";
// -->
</script>

… znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
var Page_ValidationActive = false;
if (typeof(ValidatorOnLoad) == "function") {
    ValidatorOnLoad();
}

function ValidatorOnSubmit() {
    if (Page_ValidationActive) {
        return ValidatorCommonOnSubmit();
    }
    else {
        return true;
    }
}
// -->
</script>
```

W kodzie strony można zauważyć kilka zmian w elementach formularza (zwykłych kontrolkach formularza). Obsługują one teraz zatwierdzanie formularza oraz skojarzoną z tym zdarzeniem walidację.

Korzystanie z właściwości InitialValue

Kolejną ważną właściwością podczas pracy z kontrolką `RequiredFieldValidator` jest właściwość `InitialValue`. Czasami istnieją elementy, które zawierają pewną domyślną wartość początkową (na przykład pobraną z jakiegoś źródła danych). Takie elementy formularza mogą przedstawiać pewną wartość, która musi być zmieniona, zanim formularz zostanie przesłany na serwer.

Korzystając z właściwości `InitialValue`, określa się w kontrolce `RequiredFieldValidator` początkową wartość elementu. Użytkownik końcowy musi zmienić wartość tekstową tej kontrolki przed zatwierdzeniem formularza. Na listingu 4.5 pokazano przykład użycia tej właściwości.

Listing 4.5. Praca z właściwością InitialValue

```
<asp:TextBox ID="TextBox1" Runat="server">Moja wartość początkowa</asp:TextBox>
 
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text="Wartość pola tekstowego musi być zmieniona!"
 ControlToValidate="TextBox1" InitialValue="Moja wartość początkowa">
</asp:RequiredFieldValidator>
```

W tym przypadku właściwość `InitialValue` zawiera wartość `Moja wartość początkowa`. Po zbudowaniu i uruchomieniu strony pole tekstowe również zawiera tę wartość. Kontrolka `RequiredFieldValidator` wymaga zmiany w tym polu. Tylko wtedy strona zostanie zakwalifikowana jako poprawna.

Uniemożliwienie pustych wpisów i równoczesne wymaganie zmian

W poprzednim przykładzie użycia właściwości `InitialValue` pojawił się interesujący problem. Po pierwsze, jeżeli aplikacja z takim kodem zostanie uruchomiona, wtedy użytkownik może przejść przez proces walidacji, przesyłając stronę bez żadnej wartości wpisanej w polu edycji. Puste pole edycji nie spowoduje wygenerowania błędu walidacji, ponieważ kontrolka `RequiredFieldValidator` jest teraz przebudowana tak, aby zmusić użytkownika wyłącznie do **zmiany** domyślnej wartości pola edycji (co użytkownik zrobił, usuwając starą wartość). Zmieniając kontrolkę `RequiredFieldValidator` w ten sposób, nie nakłada się żadnego ograniczenia, które wymaga wpisania **czegoś** w polu edycji — zmieniona musi być tylko wartość początkowa. Proces walidacji może być ominięty poprzez usunięcie wszystkiego, co wpisano w polu tekstowym.

Istnieje jednak sposób obejścia tego problemu. Dochodzimy tu do techniki, o której już wcześniej wspomniano — na formularzu można umieścić wiele reguł walidacyjnych. Niektóre z nich mogą być przypisane do tego samego elementu. Aby nałożyć wymóg zmiany

wartości początkowej pola edycji i równocześnie zabronić pustego wpisu (przez co element staje się elementem wymaganym), należy na stronie umieścić dodatkową kontrolkę `RequiredFieldValidator`. Pokazano to na przykładzie z listingu 4.6.

Listing 4.6. Użycie dwóch kontrolek RequiredFieldValidator dla jednego elementu formularza

```
<asp:TextBox ID="TextBox1" Runat="server">Moja wartość początkowa</asp:TextBox> 

<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="Zmień wartość pola" ControlToValidate="TextBox1"
 InitialValue="Moja wartość początkowa"></asp:RequiredFieldValidator>

<asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
 Text="Nie zostawiaj pustego pola" ControlToValidate="TextBox1"
</asp:RequiredFieldValidator>
```

W pokazanym przykładzie można zauważyć, że pole edycji rzeczywiście posiada dwie skojarzone ze sobą kontrolki `RequiredFieldValidator`. Pierwsza, `RequiredFieldValidator1`, wymaga zmiany domyślnej wartości pola edycji poprzez użycie właściwości `InitialValue`. Druga kontrolka `RequiredFieldValidator`, `RequiredFieldValidator2`, sprawia, że kontrolka `TextBox1` staje się elementem formularza, który potrzebuje pewnej wartości. Pożądane zachowanie można uzyskać, stosując dwie reguły walidacji do pojedynczego elementu formularza.

Walidacja list rozwijanych za pomocą kontrolki RequiredFieldValidator

Do tej pory pokazaliśmy dużo przykładów użycia kontrolki `RequiredFieldValidator` z serią pól tekstowych. Kontrolka `RequiredFieldValidator` może być jednak z powodzeniem stosowana wraz z innymi elementami.

Można na przykład zastosować kontrolkę `RequiredFieldValidator` z kontrolką serwerową `<asp:DropDownList>`. Przypuśćmy, że mamy listę rozwijaną. Użytkownik musi wybrać z tej listy swój zawód. Pierwszy element listy rozwijanej zawiera instrukcje dla użytkownika i informuje o tym, co lista zawiera. Ten element listy jest wymagany. Kod, który zrealizuje zaprezentowany scenariusz, pokazano na listingu 4.7.

Listing 4.7. Walidacja listy rozwijanej

```
<asp:DropDownList id="DropDownList1" runat="server">
   <asp:ListItem Selected="True">Wybierz zawód</asp:ListItem>
   <asp:ListItem>Programista</asp:ListItem>
   <asp:ListItem>Prawnik</asp:ListItem>
   <asp:ListItem>Lekarz</asp:ListItem>
   <asp:ListItem>Artysta</asp:ListItem>
</asp:DropDownList>
   
<asp:RequiredFieldValidator id="RequiredFieldValidator1"
  runat="server" Text="Wybierz element"
  ControlToValidate="DropDownList1"
  InitialValue="Wybierz zawód">
</asp:RequiredFieldValidator>
```

Tak samo jak podczas pracy z kontrolką edycji, tak i w tym przypadku kontrolka `RequiredFieldValidator` skojarzona jest z kontrolką `DropDownList` za pomocą właściwości `ControlToValidate`. Lista rozwijana, do której podpięta jest kontrolka walidacyjna, posiada wartość początkową — `Wybierz zawód`. Oczywiście taka wartość nie jest właściwa w momencie przesyłania formularza na serwer. Znów należy użyć właściwości `InitialValue` kontrolki `RequiredFieldValidator`. Wartością tej kontrolki jest wartość początkowa listy rozwijanej. Zmusza to oczywiście użytkownika do wyboru jednego z zawodów z listy rozwijanej. W przeciwnym razie nie będzie on w stanie przesłać strony na serwer.

Kontrolka serwerowa CompareValidator

Kontrolka `CompareValidator` pozwala wykonywać porównania pomiędzy dwoma elementami formularza, a także służy do porównania dwóch wartości umieszczonych we wskazanych elementach formularza. Można na przykład określić, że wartością elementu formularza musi być liczba całkowita większa niż inna zadana wartość. Można także wskazać, że wartościami muszą być łańcuchy znaków, daty lub inne typy, które są do dyspozycji.

Walidacja względem innych kontrolek

Jednym z dość często stosowanych sposobów wykorzystania kontrolki `CompareValidator` jest użycie jej do porównania dwóch elementów formularza. Przypuśćmy, że dana jest aplikacja, która w celu uzyskania dostępu do stron przez użytkowników wymaga podania hasła. Tworzymy zatem jedno pole edycji, które będzie służyło do pobrania hasła, i drugie pole tekstowe, które będzie służyło do potwierdzenia tego hasła. W związku z tym, że pole tekstowe działa w trybie wprowadzania hasła, użytkownik końcowy nie widzi tego, co wpisuje — zna tylko liczbę wprowadzonych znaków. Aby zmniejszyć prawdopodobieństwo pomyłki przy wpisywaniu hasła i zapobiec wprowadzeniu do systemu złego hasła, użytkownik proszony jest o potwierdzenie tego hasła. Po wprowadzeniu haseł do systemu należy porównać te dwa pola edycji i sprawdzić, czy wartości w nich wpisane są takie same. Jeżeli są takie same, wtedy jest duże prawdopodobieństwo, że użytkownik wpisał hasło poprawnie i może ono być wprowadzone do systemu. Jeżeli dwa pola tekstowe nie są zgodne, wtedy formularz należy traktować jako nieprawidłowy. W przykładzie z listingu 4.8 pokazano tę sytuację.

Listing 4.8. Wykorzystanie CompareValidator do porównania wartości z wartościami innych kontrolek

VB

```
<%@ Page Language="VB" %>
<script runat="server">

    Protected Sub Button1_Click(sender As Object, e As EventArgs)
        If Page.IsValid Then
            Label1.Text = "Hasła są zgodne"
        End If
    End Sub

</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>CompareFieldValidator</title>
```

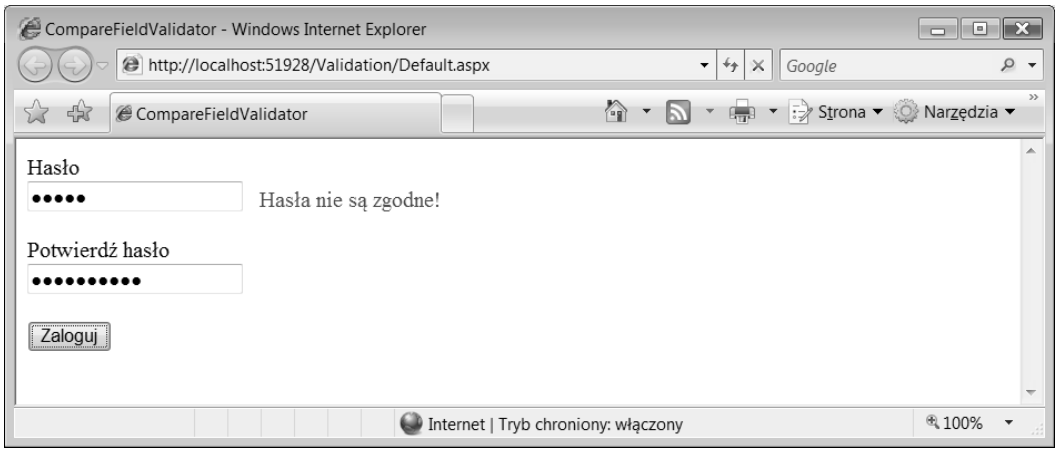
```
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            Hasło<br>
            <asp:TextBox ID="TextBox1" Runat="server"
             TextMode="Password"></asp:TextBox>
              
            <asp:CompareValidator ID="CompareValidator1"
             Runat="server" ErrorMessage="Hasła nie są zgodne!"
             ControlToValidate="TextBox2"
             ControlToCompare="TextBox1"></asp:CompareValidator>
        </p>
        <p>
            Potwierdź hasło<br>
            <asp:TextBox ID="TextBox2" Runat="server"
             TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Zaloguj"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
        if (Page.IsValid)
            Label1.Text = "Hasła są zgodne";
    }

</script>
```

Patrząc na kontrolkę `CompareValidator` na formularzu, można zauważyć, że jest ona podobna do kontrolki `FieldValidator`. Kontrolka `CompareValidator` posiada właściwość `ControlToValidate`, która łączy się z jednym z elementów formularza na stronie. W tym przypadku potrzebna jest tylko jedna kontrolka `CompareValidator`, ponieważ wymagane jest tylko jedno porównanie. W pokazanym przykładzie sprawdzana jest zgodność wartości pól edycji `TextBox2` oraz `TextBox1`. W związku z tym użyto właściwości `ControlToCompare`. Określa się w ten sposób wartość porównywaną z `TextBox2`. W tym przykładzie wartością jest `TextBox1`.

Jest to wyjątkowo proste. Jeżeli dwa pola tekstowe w momencie zatwierdzania strony posiadają różne wartości, wtedy w przeglądarce wyświetlana jest wartość właściwości `Text` kontrolki `CompareValidator`. Przykład pokazany jest na rysunku 4.2.



Rysunek 4.2

Walidacja względem stałej

Kontrolki `CompareValidator` można użyć nie tylko do walidacji wartości względem wartości w innych kontrolkach, ale też do porównania wartości względem stałych określonych typów danych. Przypuśćmy, że na formularzu rejestracyjnym umieszczono pole edycji, w którym użytkownik wpisuje swój wiek. W większości tego typu przypadków celem jest pobranie liczby całkowitej, a nie czegoś w postaci `aa` lub `bb`. Na listingu 4.9 pokazano, w jaki sposób można wymusić podanie liczby całkowitej.

Listing 4.9. Wykorzystanie CompareValidator do walidacji względem stałej

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server" MaxLength="3">
</asp:TextBox>
 
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  Text="Musisz wpisać liczbę"
  ControlToValidate="TextBox1" Type="Integer"
  Operator="DataTypeCheck"></asp:CompareValidator>
```

W pokazanym przykładzie użytkownik końcowy musi wpisać w polu tekstowym liczbę. Jeżeli spróbuje obejść proces walidacji, wpisując wartość nieprawidłową, która zawiera coś innego niż cyfry, wtedy strona będzie traktowana jako nieprawidłowa. Kontrolka `CompareValidator` wyświetla wartość właściwości `Text`.

Aby określić typ danych wykorzystywany w porównaniach, korzysta się z właściwości `Type`. Właściwość `Type` może przyjmować następujące wartości:

- `Currency`,
- `Date`,
- `Double`,
- `Integer`,
- `String`.

Wprowadzenie ograniczenia ze względu na typ to nie wszystko, co można zrobić przy wykorzystaniu tej kontrolki. Można także zadbać o to, aby wartość była porównywana z określoną stałą. Istnieje sposób na sprawdzenie, czy wartość wpisana w elemencie formularza jest większa niż, mniejsza niż, równa, większa niż lub równa i mniejsza niż lub równa pewnej wartości. Przykład pokazano na listingu 4.10.

Listing 4.10. Wykonywanie porównań za pomocą kontrolki CompareValidator

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
   
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  Operator="GreaterThan" ValueToCompare="18"
  ControlToValidate="TextBox1"
  Text="Aby się zapisać musisz mieć więcej niż 18 lat" Type="Integer">
</asp:CompareValidator>
```

W tym przypadku kontrolka `CompareValidator` oprócz połączenia się z kontrolką oraz nałożenia wymogu wprowadzenia liczby całkowitej robi coś jeszcze. Wykorzystywane są również atrybuty `Operator` oraz `ValueToCompare`. Dzięki nim można się upewnić, że liczba jest większa niż 18. Jeżeli zatem użytkownik wprowadzi liczbę równą 18 lub mniejszą, wtedy walidacja nie powiedzie się i strona będzie traktowana jako nieprawidłowa.

Właściwość `Operator` może przyjmować następujące wartości:

- `Equal`,
- `NotEqual`,
- `GreaterThan`,
- `GreaterThanEqual`,
- `LessThan`,
- `LessThanEqual`,
- `DataTypeCheck`.

Właściwość `ValueToCompare` to miejsce, gdzie umieszcza się wartość wykorzystywaną w porównaniu. W pokazanym przykładzie jest to liczba `18`.

Kontrolka serwerowa RangeValidator

Kontrolka `RangeValidator` jest podobna do kontrolki `CompareValidator`. Pozwala ona sprawdzić, czy wartość wprowadzona przez użytkownika lub jego wybór znajduje się w podanym zakresie. Za pomocą kontrolki `CompareValidator` sprawdza się, czy wartość jest większa albo mniejsza niż wskazana stała. Jako przykład zastosowania kontrolki niech posłuży nieco zmodyfikowany przykład formularza pobierającego wiek. Nieco inny jest rodzaj walidacji wprowadzonej wartości. Pokazano go na listingu 4.11.

Listing 4.11. Wykorzystanie kontrolki RangeValidator do sprawdzenia wartości całkowitej

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  
<asp:RangeValidator ID="RangeValidator1" Runat="server"
```

```
ControlToValidate="TextBox1" Type="Integer"
Text="Wiek musi być pomiędzy 30 i 40"
MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

W pokazanym przykładzie umieszczono pole edycji, za pomocą którego pobierany jest wiek użytkownika. Kontrolka `RangeValidator` analizuje wprowadzoną wartość i sprawdza, czy mieści się ona w zakresie od 30 do 40. Wykonywane jest to za pomocą właściwości `MaximumValue` oraz `MinimumValue`. Kontrolka `RangeValidator` pozwala także zadbać o to, aby wprowadzona wartość była liczbą całkowitą. W tym celu wykorzystywana jest właściwość `Type`, która ustawiona jest na `Integer`. Kolekcja zrzutów ekranów na rysunku 4.3 pokazuje działanie omawianej kontrolki.

Rysunek 4.3

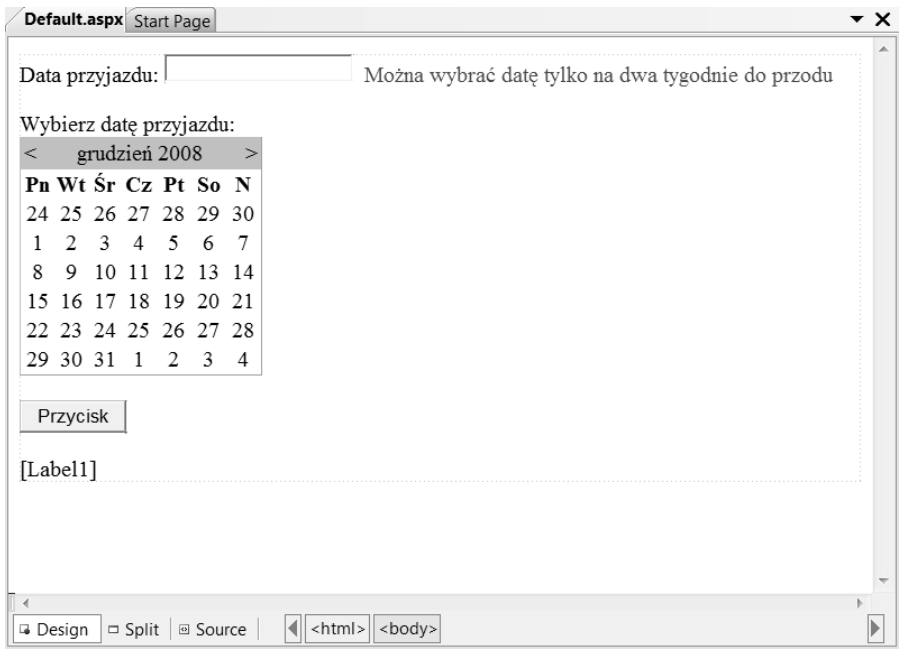
Jak można zauważyć na zrzutach ekranów z rysunku 4.3, wartość mniejsza niż 30 zostaje przez kontrolkę `RangeValidator` zasygnalizowana. To samo dzieje się w przypadku wartości większych niż 40. Wartość pomiędzy 30 i 40 (w tym przypadku 36) spełnia warunki nałożone przez kontrolkę walidacyjną.

Kontrolka `RangeValidator` nie jest stosowana wyłącznie do porównywania liczb (w takich sytuacjach jest jednak najczęściej wykorzystywana). Może ona posłużyć także do walidacji zakresu łańcuchów znaków oraz innych elementów, włączając w to daty pobrane z kalendarza. Domyślnie atrybut `Type` każdej z kontrolek walidacyjnych ustawiony jest na `String`. Kontrolka `RangeValidator` może posłużyć do sprawdzenia, czy wartość w innej kontrolce serwerowej (takiej jak na przykład `Calendar`) mieści się w podanym zakresie dat.

Przypuśćmy, że tworzony jest formularz internetowy, który pozwala na pobranie od użytkownika daty przyjazdu. Data przyjazdu musi mieścić się w zakresie dwóch tygodni od daty bieżącej. Kontrolka `RangeValidator` może być z łatwością dostosowana do takich potrzeb.

W związku z tym, że zakres dat musi być generowany dynamicznie, atrybuty `MaximumValue` oraz `MinimumValue` muszą być przypisane programowo w zdarzeniu `Page_Load`. W oknie projektanta przykładowa strona powinna wyglądać podobnie do tej z rysunku 4.4.

Rysunek 4.4



Cała idea polega na tym, że użytkownik będzie wybierał datę w kontrolce `Calendar`, a jego wybór będzie wyświetlany w kontrolce `TextBox`. Potem, gdy użytkownik kliknie na formularzu przycisk, wtedy w przypadku wybrania nieprawidłowej daty zostanie o tym powiadomiony. Jeżeli wybrana data jest prawidłowa, wtedy pokazywana jest ona przez kontrolkę `Label`. Kod przykładu pokazany jest na listingu 4.12.

Listing 4.12. Wykorzystanie kontrolki RangeValidator do sprawdzenia wartości daty w postaci łańcucha znaków

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString()
        RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).ToShortDateString()
    End Sub

    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString()
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs)
        If Page.IsValid Then
            Label1.Text = "Data przybycia ustawiona jest na: " & TextBox1.Text
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Sprawdzenie poprawności daty</title>
</head>
<body>
    <form id="form1" runat="server">
        Data przyjazdu:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox> 
        <asp:RangeValidator ID="RangeValidator1" runat="server"
         Text="Można wybrać datę tylko na dwa tygodnie do przodu"
         ControlToValidate="TextBox1" Type="Date"></asp:RangeValidator><br />
        <br />
        Wybierz datę przyjazdu:<br />
        <asp:Calendar ID="Calendar1" runat="server"
         OnSelectionChanged="Calendar1_SelectionChanged"></asp:Calendar>
         
        <br />
        <asp:Button ID="Button1" runat="server" Text="Przycisk"
         OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>
```

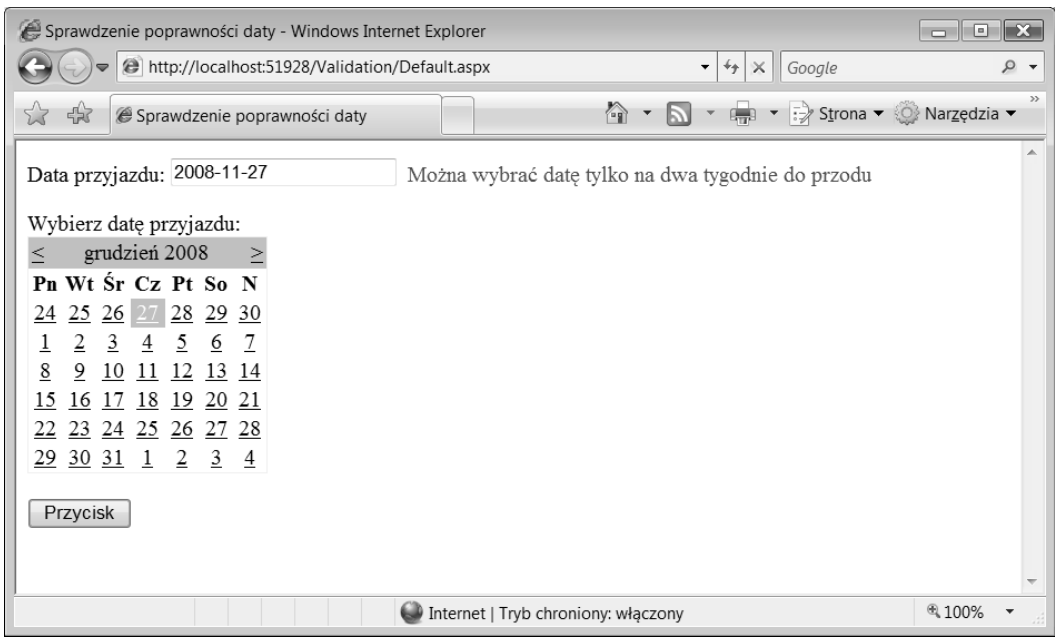
C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString();
```

```
        RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).ToShortDateString();
    }

    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            Label1.Text = "Data przybycia ustawiona jest na: " +
TextBox1.Text.ToString();
        }
    }
</script>
```

W pokazanym kodzie widać, że podczas wczytywania strony atrybuty `MinimumValue` oraz `MaximumValue` ustawiane są dynamicznie. W tym przypadku wartością `MinimumValue` staje się wynik wywołania `DateTime.Now.ToShortDateString()`, a wartością `MaximumValue` data późniejsza o 14 dni.

Po wybraniu daty przez użytkownika jej wartość wyświetlana jest w kontrolce `TextBox1`. Wykonywane jest to w procedurze obsługi zdarzenia `Calendar1_SelectionChanged`. Po wybraniu daty i kliknięciu przycisku na stronie wykonywana jest procedura obsługi zdarzenia `Button1_Click`, a poprawność walidacji całej strony sprawdzana jest za pomocą właściwości `Page.IsValid`. Nieprawidłowa strona pokaże wyniki zaprezentowane na rysunku 4.5.



Rysunek 4.5

Kontrolka serwerowa RegularExpressionValidator

Jedną ze wspaniałych kontrolek, którą programiści najczęściej wykorzystują, jest `RegularExpressionValidator`. Kontrolka oferuje wysoki poziom elastyczności podczas wprowadzania do formularza różnych reguł. Wykorzystując kontrolkę `RegularExpressionValidator`, można sprawdzić zgodność wprowadzonych danych z pewnym wzorcem, wyrażonym za pomocą wyrażenia regularnego.

Oznacza to, że można zdefiniować pewną strukturę, która będzie potem wykorzystywana do sprawdzania zgodności z danymi wprowadzonymi przez użytkownika. Można na przykład zdefiniować strukturę, która wymusza wprowadzenie danych w postaci adresu e-mail lub adresu internetowego URL. Jeżeli wartość nie jest zgodna z definicją, wtedy strona traktowana jest jako nieprawidłowa. Na listingu 4.13 pokazano, w jaki sposób sprawdzić poprawność danych wprowadzonych do pola edycji pod kątem ich zgodności z formatem adresu e-mail.

Listing 4.13. Sprawdzenie, czy pole edycji zawiera właściwy adres e-mail

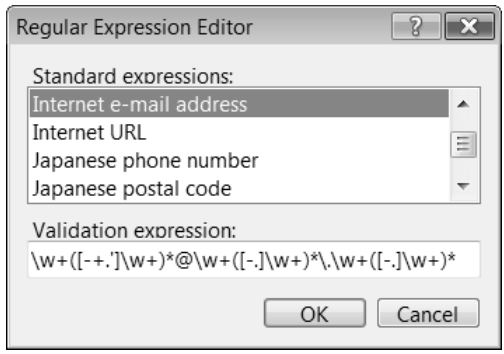
```
E-mail:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
   
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
  Runat="server" ControlToValidate="TextBox1"
  Text="Musisz wpisać właściwy adres e-mail"
  ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
```

Tak jak to jest w przypadku innych walidacyjnych kontrolek serwerowych, kontrolka `RegularExpressionValidator` używa właściwości `ContolToValidate` do połączenia się z inną kontrolką. Zawiera także właściwość `Text`, która zawiera tekst wyświetlany na ekranie po niepomyślnym zakończeniu się procesu walidacji. Unikatową właściwością tej kontrolki jest `ValidationExpression`. Właściwość przechowuje wartość tekstową, która jest wyrażeniem regularnym stosowanym przez kontrolkę do sprawdzania poprawności danych wejściowych.

Dzięki Visual Studio 2008 praca z wyrażeniami regularnymi jest nieco łatwiejsza. Wszystko to dzięki modułowi *Regular Expression Editor*. W edytorze umieszczono kilka najczęściej używanych wyrażeń regularnych, które można zastosować w kontrolce `RegularExpressionValidator`. Aby przejść do okna edytora, należy przełączyć się na widok *Design*. Należy się jednocześnie upewnić, że kontrolka serwerowa `RegularExpressionValidator1` w widoku *Design* jest podświetlona i pokazane są jej właściwości. Teraz, aby uruchomić *Regular Expression Editor*, należy w oknie *Properties Visual Studio* kliknąć obok właściwości `ValidationExpression`. Edytor pokazany jest na rysunku 4.6.

Korzystając z tego edytora, można odnaleźć wyrażenia regularne dla takich wartości, jak na przykład adresy e-mail, URL, kody pocztowe, numery telefonów, numery ubezpieczeniowe. Narzędzie Regular Expression Editor pomaga w korzystaniu z tych dość często skomplikowanych wyrażeń regularnych. Pokaźną kolekcję takich wyrażeń można znaleźć w internecie. Na przykład na witrynie RegExLib pod adresem *www.regexlib.com*.

Rysunek 4.6



Kontrolka serwerowa CustomValidator

Do tej pory pokazaliśmy zestaw kontrolek walidacyjnych, które można natychmiast wykorzystać. W większości przypadków pokazane kontrolki walidacyjne doskonale sprawdzają się podczas wprowadzania reguł walidacyjnych na formularzach. Czasami jednak żadna z tych kontrolek nie będzie odpowiednia i trzeba będzie polegać wyłącznie na tym, co się zrobi samemu. To właśnie do takich zastosowań stworzono kontrolkę `CustomValidator`.

Kontrolka `CustomValidator` pozwala utworzyć własne algorytmy walidacji, które będą stosowane zarówno po stronie klienta, jak i po stronie serwera. Skorzystanie z takiej możliwości pozwala sprawdzić poprawność wartości lub obliczeń wykonanych w warstwie danych (na przykład w bazie danych) lub upewnić się, czy na wejściu pojawia się liczba spełniająca pewne matematyczne założenia (na przykład czy jest parzysta, czy nieparzysta). Wszystko to można zrobić dzięki kontrolce `CustomValidator`.

Korzystanie z walidacji po stronie klienta

Jedną z zalet kontrolki `CustomValidator` wartych uwagi jest możliwość łatwego wprowadzenia mechanizmu walidacji po stronie klienta. Większość programistów ma swoje własne kolekcje funkcji JavaScript, które znajdują zastosowanie w ich aplikacjach. Dzięki kontrolce `CustomValidator` te funkcje mogą być szybko i łatwo zaimplementowane na stronie.

Przyjrzyjmy się prostemu formularzowi, który pobiera od użytkownika liczbę. Wykorzystano w nim kontrolkę `CustomValidator` do wprowadzenia walidacji po stronie klienta. Jej zadaniem jest sprawdzenie, czy liczba jest podzielna przez 5. Pokazano to na listingu 4.14.

Listing 4.14. Użycie kontrolki CustomValidator do przeprowadzenia walidacji po stronie klienta

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "PRAWIDŁOWA LICZBA!"
   End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>CustomValidator</title>

    <script type="text/javascript">
        function validateNumber(oSrc, args) {
           args.IsValid = (args.Value % 5 == 0);
        }
    </script>

</head>
<body>
    <form id="form1" runat="server">
    <div>
        <p>
            Liczba:
            <asp:TextBox ID="TextBox1"
             Runat="server"></asp:TextBox>
              
            <asp:CustomValidator ID="CustomValidator1"
             Runat="server" ControlToValidate="TextBox1"
             Text="Liczba musi być podzielna przez 5"
             ClientValidationFunction="validateNumber">
            </asp:CustomValidator>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Button"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
       Label1.Text = "PRAWIDŁOWA LICZBA!";
    }

</script>
```

W pokazanym formularzu można zauważyć kilka rzeczy. Po pierwsze, jest to prosty formularz; zawiera tylko jedno pole tekstowe, w którym użytkownik może wprowadzić dane. Użytkownik naciska przycisk i wywoływana jest procedura obsługi zdarzenia `Button1_Click`, która uzupełnia na stronie zawartość kontrolki `Label1`. Cała ta operacja wykonywana jest tylko wtedy, gdy na stronie wykonany zostanie proces walidacji, a dane wejściowe przejdą wszystkie testy.

Jedyną rzeczą, która wyróżnia tę stronę spośród innych tego typu, jest obecność drugiego bloku `<script>` w sekcji `<head>`. To własny kod JavaScript. Należy zauważyć, że Visual Studio 2008 umożliwia stosowanie tego typu konstrukcji. Działa to nawet wtedy, gdy zachodzi potrzeba przełączania się pomiędzy widokami *Design* oraz *Source* — to coś, z czym kiepsko radziły sobie poprzednie wersje Visual Studio. Funkcja `validateNumber` napisana w JavaScript pokazana jest poniżej:

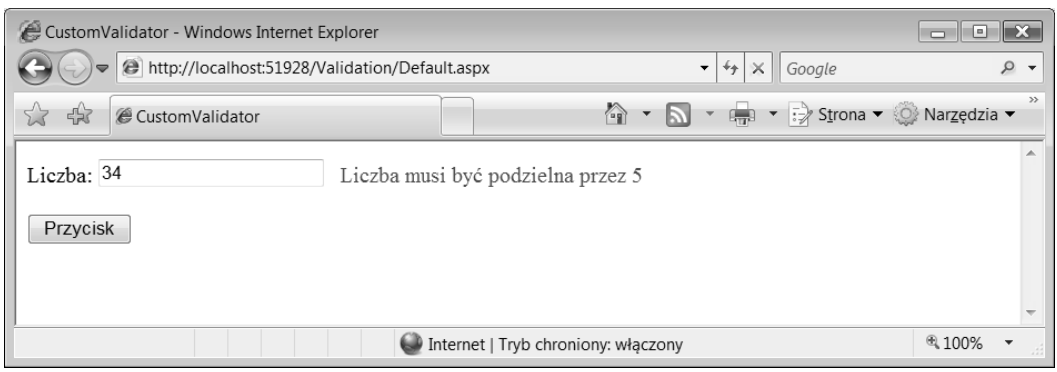
```
<script type="text/javascript">
    function validateNumber(oSrc, args) {
       args.IsValid = (args.Value % 5 == 0);
    }
</script>
```

Druga sekcja `<script>` jest kodem JavaScript używanym w kontrolce `CustomValidator` i jest wywoływana podczas przeprowadzania procesu sprawdzania poprawności danych wpisanych w polu edycji. Wstawiane funkcje JavaScript używają właściwości `args.IsValid` i ustawiają ją na `True` lub `False`, w zależności od wyniku przeprowadzonych testów sprawdzających. W tym przypadku wartość wprowadzona przez użytkownika (`args.Value`) jest sprawdzana pod kątem podzielności przez `5`. Zwrócona wartość logiczna przypisywana jest do właściwości `args.IsValid`, która na ostatnim etapie tego procesu używana jest przez kontrolkę `CustomValidator`.

Kontrolka `CustomValidator`, tak jak inne kontrolki przedstawione wcześniej, korzysta z właściwości `ControlToValidate`. Za pomocą tej właściwości kontrolka może być powiązana z określonym elementem strony. Właściwością, którą należy się w tym przypadku zainteresować, jest `ClientValidationFunction`. Wartość w postaci łańcucha znaków przypisywana do tej funkcji określa funkcję po stronie klienta, która ma być wywołana w momencie sprawdzania formularza przez kontrolkę `CustomValidator`. W tym przypadku jest to `validateNumber`:

```
ClientValidationFunction="validateNumber"
```

Po uruchomieniu strony i wprowadzeniu nieprawidłowego wpisu można otrzymać wynik pokazany na rysunku 4.7.



Rysunek 4.7

Tworząc własne procedury walidacji po stronie serwera, można stworzyć tak zaawansowane algorytmy, jakie są tylko aplikacji potrzebne. Użycie walidacji po stronie serwera może się przydać wtedy, gdy zachodzi potrzeba sprawdzenia danych wprowadzonych przez użytkownika

i porównania ich z dynamicznymi wartościami pochodzącymi z plików XML, baz danych lub innego źródła.

Jako przykład własnej walidacji po stronie serwera za pomocą kontrolki `CustomValidator` może posłużyć jej odpowiednik po stronie klienta. Zróbmy to samo. Stwórzmy procedurę walidującą po stronie serwera, która sprawdza podzielność liczby przez 5. Pokazano to na listingu 4.15.

Listing 4.15. Wykorzystanie kontrolki CustomValidator do przeprowadzenia walidacji po stronie serwera

VB

```
<%@ Page Language="VB" %>

<script runat="server">
   Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
       If Page.IsValid Then
          Label1.Text = "PRAWIDŁOWY WPIS!"
       End If
   End Sub

   Sub ValidateNumber(sender As Object, args As ServerValidateEventArgs)
       Try
          Dim num As Integer = Integer.Parse(args.Value)
          args.IsValid = ((num mod 5) = 0)
       Catch ex As Exception
          args.IsValid = False
       End Try
   End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>CustomValidator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <p>
            Liczba:
            <asp:TextBox ID="TextBox1"
             Runat="server"></asp:TextBox>
              
            <asp:CustomValidator ID="CustomValidator1"
             Runat="server" ControlToValidate="TextBox1"
             Text="Liczba musi być podzielna przez 5"
             OnServerValidate="ValidateNumber"></asp:CustomValidator>
        </p>
        <p>
            <asp:Button ID="Button1" OnClick="Button1_Click"
             Runat="server" Text="Przycisk"></asp:Button>
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </div>
    </form>
```

```
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
         if (Page.IsValid) {
            Label1.Text = "PRAWIDŁOWY WPIS!";
         }
    }

    void ValidateNumber(object source, ServerValidateEventArgs args)
    {
       try
       {
          int num = int.Parse(args.Value);
          args.IsValid = ((num%5) == 0);
       }
       catch(Exception ex)
       {
          args.IsValid = false;
       }
    }

</script>
```

Zamiast umieszczać w kodzie funkcję JavaScript, w tym przykładzie wykorzystano funkcję po stronie serwera — `ValidateNumber`. Funkcja `ValidateNumber`, tak jak wszystkie funkcje przystosowane do pracy z kontrolką `CustomValidator`, musi jako parametr przyjmować obiekt `ServerValidateEventArgs`. Dzięki temu do funkcji sprawdzającej zostaną przekazane właściwe dane. Funkcja `ValidateNumber` nie robi niczego ciekawego. Po prostu sprawdza, czy liczba jest podzielna przez `5`.

Wewnątrz własnej funkcji, która została zaprojektowana do pracy z kontrolką `CustomValidator`, otrzymujemy wartość pochodzącą z elementu formularza poprzez obiekt `args.Value`. Ustawiamy potem właściwość `args.IsValid` na `True` lub `False`, w zależności od wyników procedury sprawdzającej. W przykładzie można zauważyć, że `args.IsValid` ustawiane jest na `False` wtedy, gdy liczba nie jest podzielna przez 5, i wtedy, gdy wyrzucony zostanie wyjątek (może on zostać wygenerowany wtedy, gdy zamiast liczby z elementu formularza otrzymamy jakiś tekst). Po utworzeniu własnej funkcji walidującej należy przejść do kolejnego kroku. Wstawmy zatem kontrolkę `CustomValidator` na stronę, jak pokazano na poniższym listingu:

```
<asp:CustomValidator ID="CustomValidator1"
Runat="server" ControlToValidate="TextBox1"
Text="Liczba musi być podzielna przez 5"
OnServerValidate="ValidateNumber"></asp:CustomValidator>
```

Aby skojarzyć kontrolkę `CustomValidator` z funkcją umieszczoną w kodzie po stronie serwera, korzysta się z atrybutu `OnServerValidate`. Wartością przypisywaną do tej właściwości jest nazwa funkcji — w tym przypadku jest to `ValidateNumber`.

Wykonanie kodu z tego przykładu spowoduje wysłanie strony z powrotem na serwer i tam zostanie wykonana walidacja (dzięki funkcji `ValidateNumber`). Potem strona jest przeładowywana i wywoływane jest zdarzenie `Page_Load`. Na listingu 4.15 można zauważyć, że sprawdzana jest wartość określająca, czy strona jest prawidłowa. Wykonywane jest to za pomocą właściwości `Page.IsValid`:

```
If Page.IsValid Then
   Label1.Text = "PRAWIDŁOWY WPIS!"
End If
```

Jednoczesne użycie walidacji po stronie serwera i po stronie klienta

Jak już zostało wcześniej wspomniane, należy pomyśleć o bezpieczeństwie formularzy i zadbać o to, aby zbierane przez formularz dane były prawidłowe. Z tego powodu, decydując się na walidację po stronie klienta (pokazano to na listingu 4.14), należy jednocześnie przerobić funkcje działające po stronie klienta na ich odpowiedniki po stronie serwera. Po wykonaniu tego kroku powinno się skojarzyć funkcje po stronie klienta i po stronie serwera z kontrolką `CustomValidator`. W przypadku stosowania algorytmu walidującego z listingu 4.14 i listingu 4.15 można użyć na stronie dwóch funkcji walidacyjnych i tak zmodyfikować kontrolkę `CustomValidator`, aby wskazywała na obie te funkcje. Pokazano to na listingu 4.16.

Listing 4.16. Kontrolka CustomValidator z walidacją po stronie klienta i po stronie serwera

```
<asp:CustomValidator ID="CustomValidator1"
    Runat="server" ControlToValidate="TextBox1"
    Text="Liczba musi być podzielna przez 5"
    ClientValidationFunction="validateNumber"
    OnServerValidate="ValidateNumber"></asp:CustomValidator>
```

Jak widać na zaprezentowanym przykładzie, nie ma żadnego problemu związanego z jednoczesnym użyciem atrybutów `ClientValidationFunction` oraz `OnServerValidate`.

Kontrolka serwerowa ValidationSummary

Kontrolka `ValidationSummary` nie jest kontrolką, która przeprowadza jakiegoś rodzaju testy sprawdzające poprawność danych wpisanych na formularzu. Jest to kontrolka typowo raportująca, która jest wykorzystywana przez inne kontrolki walidacyjne na stronie. Kontrolka walidacyjna `ValidationSummary` może być użyta do zebrania wszystkich zgłoszonych błędów walidacyjnych na stronie i może stanowić alternatywę wyświetlania błędów przez każdą kontrolkę walidacyjną indywidualnie.

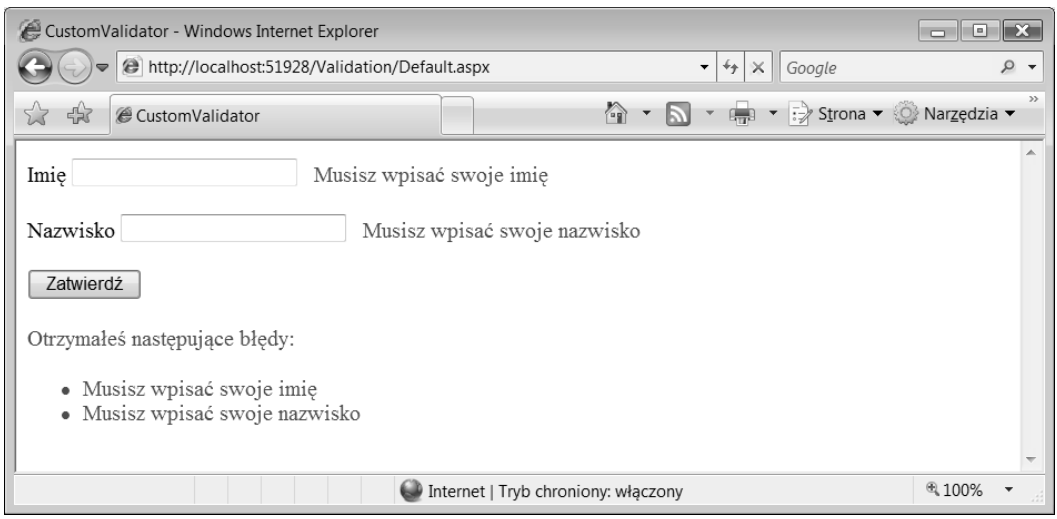
Taka możliwość może być wykorzystana na większych formularzach, które wykorzystują pełny proces walidacji. W takim przypadku rozwiązaniem bardziej przyjaznym dla użytkownika może być umieszczenie wszystkich zgłoszonych błędów w sposób jednolity i łatwy do ogarnięcia. Takie komunikaty o błędach mogą być wyświetlone za pomocą listy, listy wypunktowanej lub paragrafu.

Domyślnie kontrolka `ValidationSummary` pokazuje listę zgłoszonych błędów walidacji w postaci listy wypunktowanej. Pokazano to na listingu 4.17.

Listing 4.17. Przykład fragmentu strony korzystającej z kontrolki ValidationSummary

```
<p>Imię
   <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
     
   <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    Runat="server" ErrorMessage="Musisz wpisać swoje imię"
    ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
</p>
<p>Nazwisko
   <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
     
   <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
    Runat="server" ErrorMessage="Musisz wpisać swoje nazwisko"
    ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
</p>
<p>
   <asp:Button ID="Button1" OnClick="Button1_Click" Runat="server"
    Text="Zatwierdź"></asp:Button>
</p>
<p>
   <asp:ValidationSummary ID="ValidationSummary1" Runat="server"
    HeaderText="Otrzymałeś następujące błędy:">
   </asp:ValidationSummary>
</p>
<p>
   <asp:Label ID="Label1" Runat="server"></asp:Label>
</p>
```

Zadaniem przykładowego kodu jest pobranie imienia i nazwiska użytkownika. Każde pole tekstowe na formularzu ma powiązaną ze sobą kontrolkę `RequiredFieldValidator`. Po zbudowaniu i uruchomieniu strony oraz naciśnięciu przez użytkownika przycisku *Zatwierdź* bez wpisywania w pola edycji żadnych wartości powstaną dwa błędy. Wynik działania aplikacji pokazany jest na rysunku 4.8.



Rysunek 4.8

We wcześniejszych przykładach walidacji kontrolek na formularzu błędy pojawiają się obok każdego z pól tekstowych. Można jednak zauważyć, że kontrolka `ValidationSummary` pokazuje błędy walidacyjne w postaci czerwonej listy wypunktowanej w miejscu, w którym ta kontrolka się znajduje. W większości przypadków nie ma potrzeby wyświetlania tych samych błędów na stronie dwukrotnie. Można zmienić to zachowanie, korzystając nie tylko z właściwości `ErrorMessage`, ale także z właściwości `Text` kontrolek walidacyjnych, tak jak było to wykonywane w tym rozdziale do tej pory. Zaprezentowane podejście pokazano na listingu 4.18.

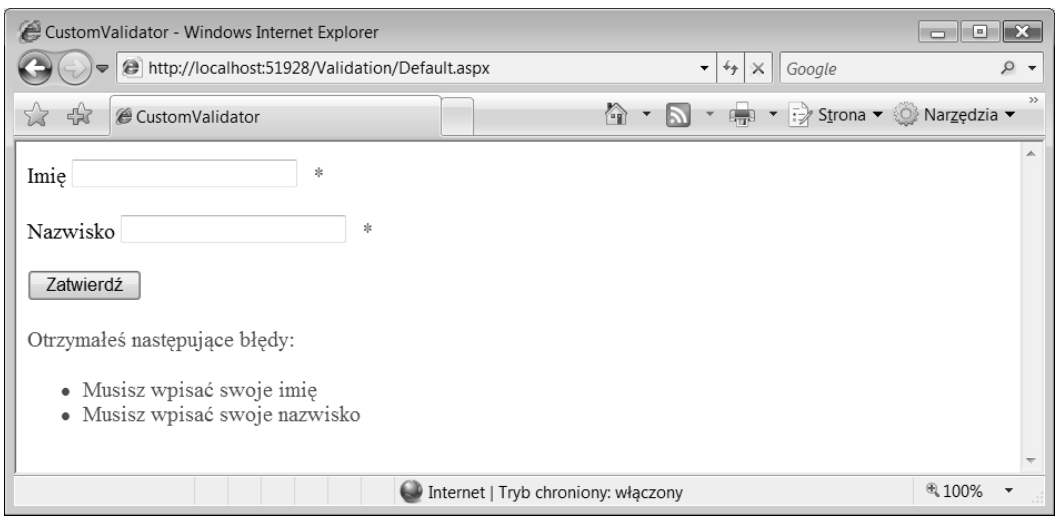
Listing 4.18. Wykorzystanie właściwości Text kontrolki walidacyjnej

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ErrorMessage="Musisz wpisać swoje imię" Text="*"
 ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

lub

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" ErrorMessage="Musisz wpisać swoje imię"
 ControlToValidate="TextBox1">*</asp:RequiredFieldValidator>
```

Na listingu 4.18 pokazano dwa sposoby wykonania tego samego zadania. Pierwszy z nich polega na użyciu właściwości `Text`. Drugi polega na umieszczeniu wartości pomiędzy znacznikami elementu `<asp:RequiredFieldValidator>`. Wprowadzenie do kontrolek walidacyjnych takich zmian spowoduje powstanie wyników pokazanych na rysunku 4.9.



Rysunek 4.9

Aby otrzymać taki wynik, należy pamiętać, że kontrolka `ValidationSummary` w celu wyświetlenia błędów walidacji korzysta z właściwości `ErrorMessage`. Właściwość `Text` jest używana przez kontrolki walidacyjne i nie jest w żaden sposób przetwarzana przez kontrolkę `ValidationSummary`.

Oprócz list wypunktowanych można stosować inne formaty wyświetlania wyników. Zmian można dokonać za pomocą właściwości `DisplayMode`. Kontrolka może przyjmować następujące wartości:

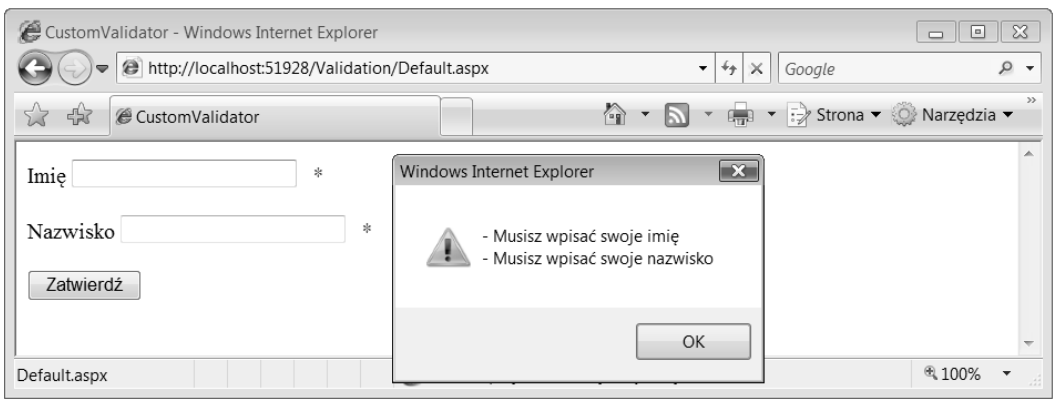
- `BulletList`,
- `List`,
- `SingleParagraph`.

Zamiast wyświetlać wyniki na stronie, można także wykorzystać okno dialogowe. Na listingu 4.19 pokazano przykład takiego podejścia.

Listing 4.19. Wykorzystanie okna dialogowego do raportowania błędów walidacji

```
<asp:ValidationSummary ID="ValidationSummary1" Runat="server"
 ShowMessageBox="True" ShowSummary="False"></asp:ValidationSummary>
```

W pokazanym przykładzie właściwość `ShowSummary` ustawiona jest na `False`, co oznacza, że błędy walidacyjne nie będą wyświetlane na stronie. Jednak w związku z tym, że właściwość `ShowMessageBox` ustawiona jest na `True`, wszystkie błędy walidacji będą wyświetlone w oknie dialogowym. Pokazano to na rysunku 4.10.



Rysunek 4.10

Wyłączanie walidacji po stronie klienta

Walidacyjne kontrolki serwerowe automatycznie przeprowadzają walidację po stronie klienta (jeżeli aplikacja wysyłająca żądanie potrafi poprawnie przetworzyć utworzony kod JavaScript). Czasami jednak zachodzi konieczność przejęcia kontroli nad tego typu zachowaniem.

Można wyłączyć walidację tych kontrolek po stronie klienta w taki sposób, aby nie wysyłały one do aplikacji żądających żadnych funkcji walidujących. Takie podejście umożliwia przeprowadzanie wyłącznie walidacji po stronie serwera, bez względu na rzeczywiste możliwości kontrolek walidacyjnych. Istnieje kilka możliwości wyłączenia tego typu funkcjonalności.

Pierwszy sposób może być zrealizowany na poziomie kontrolki. Każda z walidacyjnych kontrolek serwerowych posiada właściwość o nazwie `EnableClientScript`. Właściwość jest domyślnie ustawiona na `True`. Ustawienie właściwości na `False` blokuje wysyłanie funkcji JavaScript na stronę klienta. Zamiast tego walidacja przeprowadzana jest na serwerze. Przykład użycia tej właściwości pokazany jest na listingu 4.20.

Listing 4.20. Zablokowanie w kontrolce walidacyjnej walidacji po stronie klienta

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="*" ControlToValidate="TextBox1" EnableClientScript="false">
```

Można także zablokować walidację po stronie klienta programowo. Pokazano to na listingu 4.21.

Listing 4.21. Programowe blokowanie walidacji po stronie klienta

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    RequiredFieldValidator1.EnableClientScript = False
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
    RequiredFieldValidator1.EnableClientScript = false;
End Sub
```

Kolejną opcją jest wyłączenie walidacji na kliencie dla wszystkich kontrolek na stronie w zdarzeniu `Page_Load`. Może to być przydatne wtedy, gdy trzeba dynamicznie zadecydować o zezwoleniu na walidację po stronie klienta lub o jej zablokowaniu. Pokazano to na listingu 4.22.

Listing 4.22. Blokowanie walidacji po stronie klienta dla wszystkich kontrolek w zdarzeniu Page_Load

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    For Each bv As BaseValidator In Page.Validators
        bv.EnableClientScript = False
    Next
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e) {
    foreach(BaseValidator  bv in Page.Validators)
    {
        bv.EnableClientScript = false;
    }
}
```

Instrukcja `For Each` przeglądająca każdą instancję klasy `BaseValidator` w składowej `Validators` strony ASP.NET może wyłączyć walidację po stronie klienta wszystkich kontrolek, które zostały umieszczone na tej stronie.

Korzystanie z obrazków i dźwięków w powiadomieniach o błędach

Do tej pory wyświetlaliśmy proste wiadomości tekstowe, które informowały o pojawieniu się błędów w walidacyjnych kontrolkach serwerowych. W większości przypadków takie rozwiązanie jest wystarczające — prosty komunikat tekstowy informuje użytkownika końcowego o tym, że wprowadził do formularza jakąś nieprawidłową wartość, która nie spełnia nałożonych wymagań.

Interesującą wskazówką jest to, że nie trzeba wcale używać tekstu — w celu powiadomienia użytkownika można także używać dźwięków i obrazków.

W tym celu korzysta się z właściwości `Text`, która jest dostępna w każdej kontrolce walidacyjnej. Aby do zasygnalizowania błędu użyć obrazka, można po prostu przypisać do właściwości właściwy kod HTML. Pokazano to na listingu 4.23.

Listing 4.23. Używanie obrazków do powiadomienia o błędach

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text='<img src="error.gif">'
 ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

Jak można zauważyć na przykładzie, zamiast wyświetlać tekst na stronie, do właściwości przypisywany jest tekst HTML. Ten fragment tekstu HTML używany jest do wyświetlenia obrazka. Należy zwrócić szczególną uwagę na obecność cudzysłowów pojedynczych i podwójnych. W ten sposób po wygenerowaniu strony w przeglądarce nie pojawią się żadne błędy. Kod pokazany na listingu 4.23 spowoduje wygenerowanie strony podobnej do tej z rysunku 4.11.

Rysunek 4.11

Kolejnym interesującym rozwiązaniem jest możliwość dodania do strony powiadomienia dźwiękowego, które będzie używane przy błędach. Można to zrobić w taki sam sposób, jaki został wykorzystany przy powiadomieniu obrazkowym. Na listingu 4.24 pokazano przykład.

Listing 4.24. Wykorzystanie dźwięku do powiadomienia o błędach

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
 Runat="server" Text='<bgsound src="C:\Windows\Media\tada.wav">'
 ControlToValidate="TextBox1" EnableClientScript="False">
</asp:RequiredFieldValidator>
```

Dużo różnych dźwięków systemu Windows można znaleźć w katalogu `C:\Windows\Media`. W tym przykładzie właściwość `Text` przyjmuje wartość elementu `<bgsound>`. Zadaniem tego elementu jest umieszczenie na formularzu dźwięku (działa tylko z Internet Explorer). Dźwięk odgrywany jest tylko w momencie wywołania przez użytkownika kontrolki walidacyjnej.

Pracując z powiadomieniami dźwiękowymi, trzeba zablokować skrypty po stronie klienta. Jeżeli się tego nie zrobi, wtedy dźwięk zostanie odegrany już w momencie wczytania strony w przeglądarce, bez względu na to, czy pojawi się jakiś błąd.

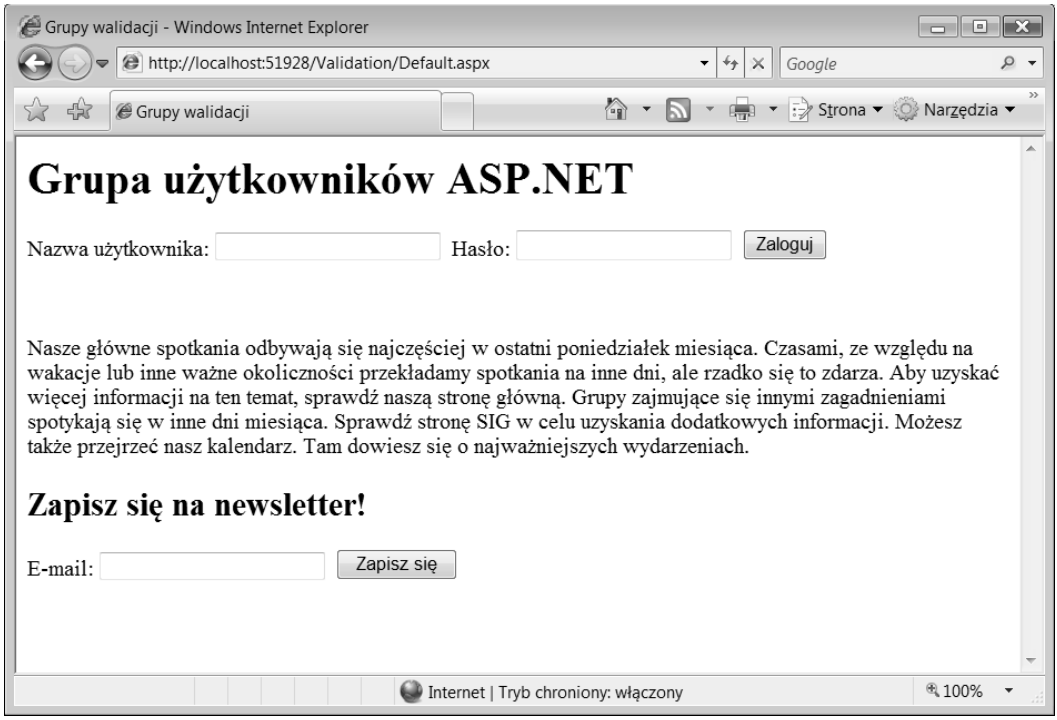
Praca z grupami walidacyjnymi

W większości przypadków programiści umieszczali na jednej stronie większą ilość formularzy. W ASP.NET 1.0/1.1 było to możliwe, ponieważ kliknięcia różnych przycisków mogły wywoływać różne zdarzenia. Korzystając z takiego rozwiązania, można się było narazić na pewne problemy.

Jeden z tych problemów polegał na tym, że pojawiały się pewne trudności z posiadaniem walidacji dla wszystkich kontrolek na każdym formularzu strony. Różne kontrolki walidacyjne były często przypisywane do dwóch różnych formularzy na stronie. Gdy użytkownik zatwierdzał jeden z formularzy, wtedy uruchamiane były procedury walidacyjne na drugim formularzu (ponieważ użytkownik nie pracował z tym formularzem). Błędy na drugiej stronie mogą więc zablokować zatwierdzenie pierwszego formularza.

Na rysunku 4.12 pokazano przykładową stronę grupy użytkowników ASP.NET, która zawiera dwa formularze.

Jeden formularz jest przeznaczony dla członków grupy. Za jego pomocą podaje się nazwę użytkownika i hasło, które są potrzebne do wejścia do strefy tylko dla członków. Drugi formularz na stronie jest przeznaczony dla wszystkich użytkowników i pozwala zapisać się do newslettera. Każdy z formularzy posiada swój przycisk i związane z nim kontrolki walidacyjne. Problem pojawia się wtedy, gdy ktoś przesyła informacje za pomocą jednego formularza. Przypuśćmy, że mamy do czynienia z członkiem grupy, który wpisuje nazwę użytkownika, hasło i naciska przycisk *Zaloguj*. Kontrolki walidacyjne dla newslettera mogą zasygnalizować błąd, ponieważ w tym formularzu nie uzupełniono adresu e-mail. Jeżeli ktoś jest zainteresowany korzystaniem z newslettera, wtedy umieszcza tylko adres e-mail w ostatnim polu i naciska przycisk *Zapisz się*. Kontrolka walidacyjna z pierwszego formularza sygnalizuje błąd, ponieważ nie wprowadzono nazwy użytkownika i hasła do pierwszego formularza.



Rysunek 4.12

W ASP.NET 3.5 pojawiła się właściwość `ValidationGroup`, która pozwala podzielić kontrolki walidacyjne na kilka niezależnych grup. W momencie naciśnięcia przycisku na stronie pozwala ona aktywować tylko określone kontrolki walidacyjne. Na listingu 4.25 pokazano przykład podzielenia kontrolek walidacyjnych na stronie grupy użytkowników na dwie różne grupy.

Listing 4.25. Korzystanie z właściwości ValidationGroup

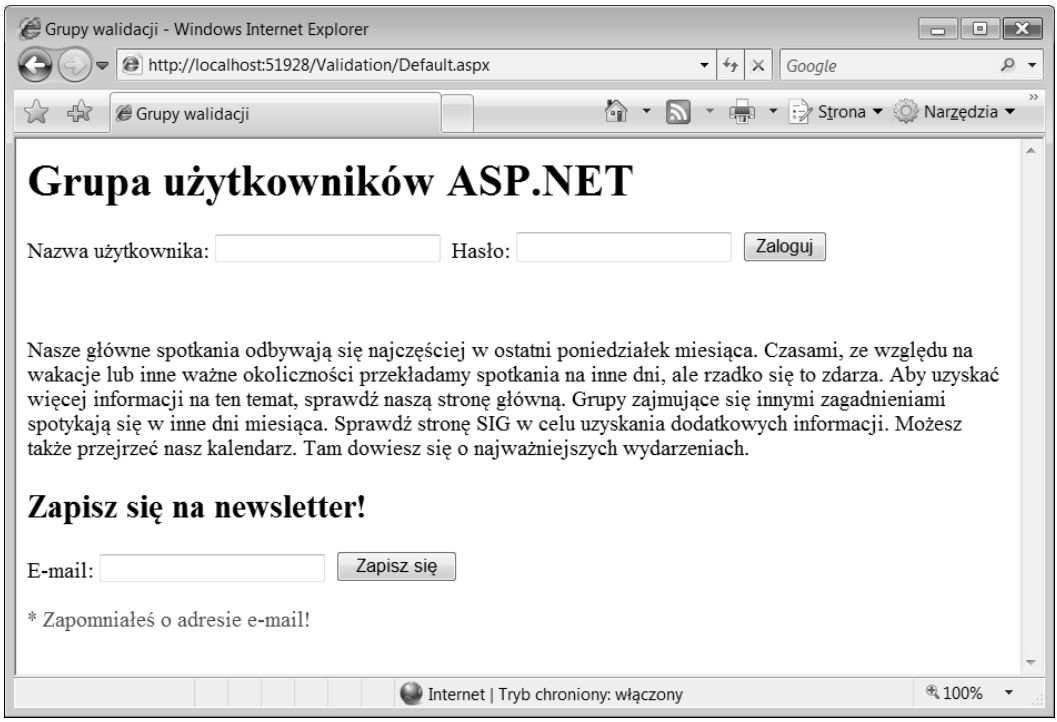
```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Grupy walidacji</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1>Grupa użytkowników ASP.NET</h1>
        <p>Nazwa użytkownika:
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>  Hasło:
        <asp:TextBox ID="TextBox2" Runat="server"
         TextMode="Password"></asp:TextBox> 
        <asp:Button ID="Button1" Runat="server" Text="Zaloguj"
         ValidationGroup="Login" />
            <br />
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
             Text="* Musisz wpisać nazwę użytkownika!"
```

```
            ControlToValidate="TextBox1" ValidationGroup="Login">
        </asp:RequiredFieldValidator>
        <br />
        <asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
         Text="* Musisz wpisać hasło!"
         ControlToValidate="TextBox2" ValidationGroup="Login">
        </asp:RequiredFieldValidator>
    <p>
        Nasze główne spotkania odbywają się najczęściej w ostatni poniedziałek
        miesiąca. Czasami, ze względu na wakacje lub inne ważne okoliczności
        przekładamy spotkania na inne dni, ale rzadko się to zdarza. Aby uzyskać
        więcej informacji na ten temat, sprawdź naszą stronę główną. Grupy zajmujące
        się innymi zagadnieniami spotykają się w inne dni miesiąca. Sprawdź stronę
        SIG w celu uzyskania dodatkowych informacji. Możesz także przejrzeć nasz
        kalendarz. Tam dowiesz się o najważniejszych wydarzeniach.<br />
    </p>
    <h2>Zapisz się na newsletter!</h2>
    <p>E-mail:
    <asp:TextBox ID="TextBox3" Runat="server"></asp:TextBox> 
    <asp:Button ID="Button2" Runat="server" Text="Zapisz się"
     ValidationGroup="Newsletter" /> 
        <br />
        <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
         Runat="server"
         Text="* Musisz wpisać właściwy adres e-mail!"
         ControlToValidate="TextBox3" ValidationGroup="Newsletter"
         ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
        </asp:RegularExpressionValidator>
        <br />
        <asp:RequiredFieldValidator ID="RequiredFieldValidator3" Runat="server"
         Text="* Zapomniałeś o adresie e-mail!"
         ControlToValidate="TextBox3" ValidationGroup="Newsletter">
        </asp:RequiredFieldValidator>
    </p>
  </div>
  </form>
</body>
</html>
```

Właściwość `ValidationGroup` na przykładzie została pogrubiona. Można zauważyć, że wartością właściwości jest łańcuch znaków. Warto także zwrócić uwagą na to, że nie tylko kontrolki walidacyjne posiadają tę właściwość. Kluczowe kontrolki serwerowe także mają właściwość `ValidationGroup`, ponieważ zdarzenia takie jak kliknięcia myszą także muszą być powiązane z określoną grupą walidacyjną.

W pokazanym przykładzie każdy z przycisków przypisany jest do innej grupy walidacyjnej. Pierwszy przycisk jako wartości używa `Login`, drugi z nich używa wartości `Newsletter`. Każda z kontrolek walidacyjnych także powiązana jest z określoną grupą walidacyjną. Dzięki takiemu rozróżnieniu, jeżeli użytkownik kliknie na stronie przycisk *Zaloguj*, ASP.NET jest w stanie rozpoznać grupę. Konsekwencją jest to, że przycisk pracuje wyłącznie z walidacyjnymi kontrolkami serwerowymi należącymi do tej samej grupy. ASP.NET ignoruje walidacyjne kontrolki przypisane do innych grup walidacyjnych.

Korzystając z tego udogodnienia, można zastosować wiele grup z regułami walidacyjnymi, które będą wywoływane tylko wtedy, gdy chcemy. Pokazano to na rysunku 4.13.



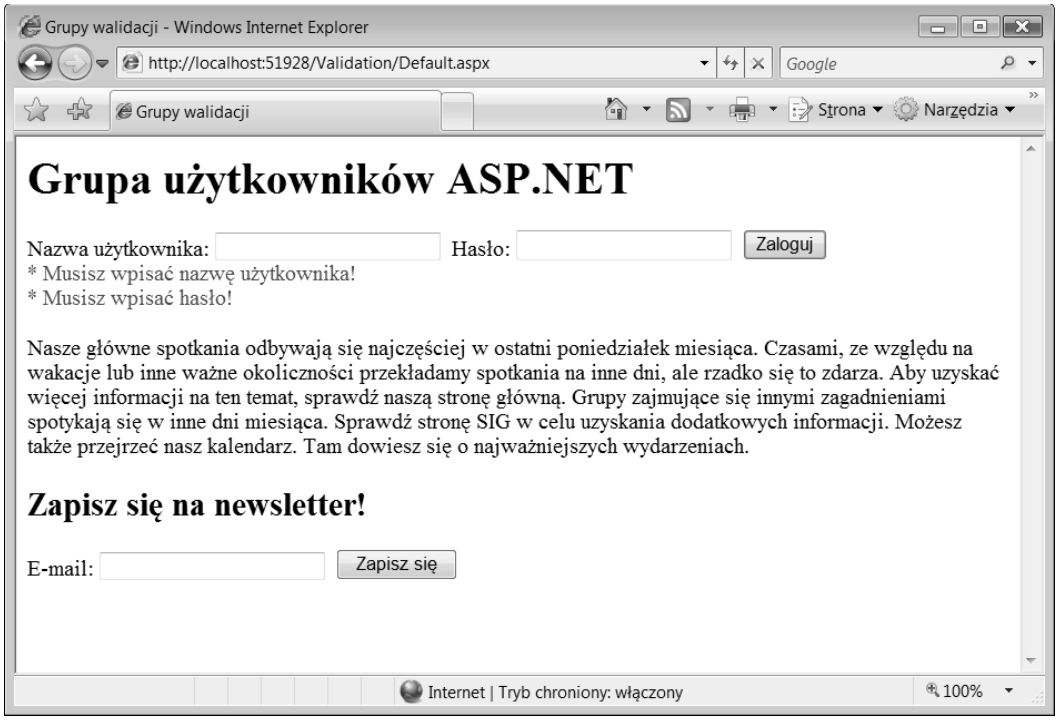
Rysunek 4.13

Kolejnym udogodnieniem dodanym do kontrolek walidacyjnych jest właściwość `SetFocusOnError`. Właściwość pobiera wartość typu logicznego. Jeżeli podczas zatwierdzania formularza sygnalizowany jest błąd, wtedy właściwość umieszcza fokus na elemencie, który ten błąd zawiera. Właściwość `SetFocusOnError` pokazana jest na poniższym listingu:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
 Text="* Musisz wpisać nazwę użytkownika!"
 ControlToValidate="TextBox1" ValidationGroup="Login" SetFocusOnError="true">
</asp:RequiredFieldValidator>
```

Jeżeli `RequiredFieldValidator1` zasygnalizuje błąd, ponieważ użytkownik nie wstawił żadnej wartości w polu `TextBox1`, wtedy strona jest odrysowywana z fokusem ustawionym na kontrolce `TextBox1`. Pokazano to na rysunku 4.14.

Należy także zwrócić uwagę na to, że wiele kontrolek walidacyjnych może zawierać ustawioną na `True` właściwość `SetFocusOnError`. Jeżeli w takim przypadku wystąpi więcej błędów walidacji, wtedy fokus otrzyma pierwszy element formularza. Jako przykład weźmy pokazany formularz. Jeżeli zarówno pole tekstowe z nazwą użytkownika (`TextBox1`), jak i pole tekstowe z hasłem (`TextBox2`) zawierają błędy walidacji, wtedy fokus zostanie ustawiony na polu tekstowym z nazwą użytkownika. Jest to bowiem pierwsza kontrolka tego formularza z błędem.



Rysunek 4.14

Podsumowanie

Kontrolki walidacyjne są miłym dodatkiem dla tych programistów, którzy przenoszą się z Active Server Pages na ASP.NET. Skrywają one wielkie możliwości pod postacią łatwej w użyciu paczki i tak jak wiele rzeczy w świecie .NET mogą być w prosty sposób dostosowywane. Dzięki temu będą działały dokładnie tak, jak sobie tego zażyczymy.

Należy pamiętać, że celem formularzy jest zbieranie danych. Takie zbieranie danych nie ma jednak większego sensu, gdy dane są nieprawidłowe. Oznacza to, że trzeba zastosować pewne reguły sprawdzania poprawności danych, które będą implementowane na formularzach za pomocą zestawu różnych kontrolek — walidacyjnych kontrolek serwerowych.

W niniejszym rozdziale omówiono różne walidacyjne kontrolki, włączając w to:

- `RequiredFieldValidator`,
- `CompareValidator`,
- `RangeValidator`,
- `RegularExpressionValidator`,

- `RegularExpressionValidator`,
- `CustomValidator`,
- `ValidationSummary`.

Oprócz zaprezentowania podstawowych kontrolek walidacyjnych w rozdziale poruszono także zagadnienia związane ze stosowaniem walidacji po stronie klienta i walidacji po stronie serwera.