Studium przypadku: serwis *Newsletter*

6.1 Biznesowa wizja systemu

6.2 Projektowa wizja systemu

6.3 Przygotowanie projektu

6.4 Przygotowanie bazy danych

6.5 Dodanie klasy zarządzającej operacjami na bazie danych

6.6 Utworzenie strony domowej

Strona domowa serwisu *Newsletter* zawiera formularz, który umożliwia subskrypcję i desubskrypcję newslettera. Tworzenie tej strony rozpoczniemy od wpisania kodu z listingu 6-3 do klasy **Default.aspx**. Kod jest niezwykle prosty – używając podstawowych kontrolek ASP.NET utworzony został formularz z dwiema akcjami: **Subskrybuj** oraz **Anuluj subskrypcję**. Na uwagę zasługuje obecność standardowych kontrolek walidujących wprowadzane przez użytkownika dane oraz zabezpieczenie Captcha.

Listing 6-3
Modyfikacje pliku Default.aspx

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="SerwisNewsletter.Default" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent"
runat="server">
    <div class="jumbotron">
        <h1>Usługi Microsoft Azure</h1>
        <h2>Serwis Newsletter - automatyczne powiadomienia e-mail</h2>
        <p class="lead">
            Dopisz swój e-mail, aby na bieżąco otrzymywać nowe wiadomości i
powiadomienia
                lub anuluj subskrypcję jeśli nie chcesz otrzymywać wiadomości.
        </p>
```

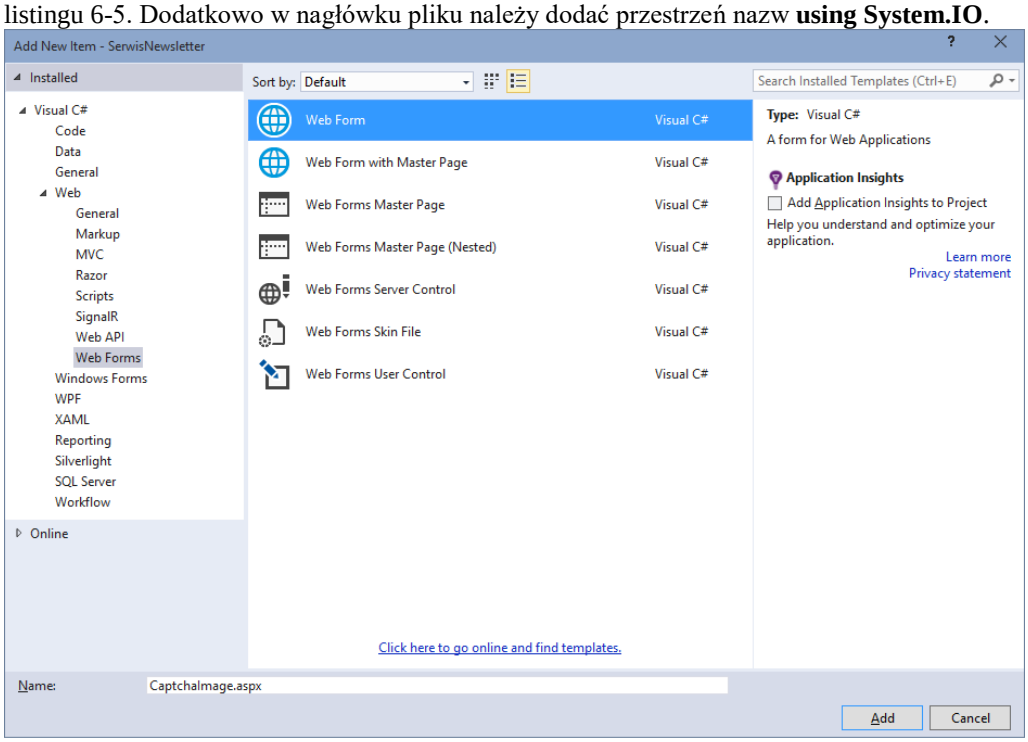
```
    </div>
    <div class="panel panel-default">
        <div class="panel-heading">Formularz zgłoszeniowy</div>
        <div class="panel-body">
            <asp:Label runat="server" ID="infoLabel" CssClass="label label-
success"></asp:Label><br />
            <asp:Label runat="server" ID="errorLabel" CssClass="label label-
danger"></asp:Label><br />
            <asp:ValidationSummary runat="server" ValidationGroup="Validation"
CssClass="alert alert-danger" />
            <asp:RequiredFieldValidator runat="server" Display="None"
ErrorMessage="Email jest wymagany"
                ControlToValidate="textBoxEmail" ValidationGroup="Validation"
/>
            <asp:RegularExpressionValidator runat="server"
                ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-
.]\w+)*"
                ControlToValidate="textBoxEmail" ErrorMessage="Niepoprawny
adres email"
                Display="None" ValidationGroup="Validation" />
            <asp:RequiredFieldValidator runat="server" Display="None"
ErrorMessage="Kod jest wymagany"
                ControlToValidate="captchaTextBox" ValidationGroup="Validation"
/>
            <div class="form-group">
                <asp:Label runat="server" CssClass="control-label col-md-3"
AssociatedControlID="textBoxEmail" Text="Podaj email:" />
                <div class="col-md-9">
                    <asp:TextBox runat="server" ID="textBoxEmail"
CssClass="form-control" TextMode="Email" />
                </div>
            </div>
            <div class="form-group">
                <asp:Label runat="server" CssClass="control-label col-md-3"
AssociatedControlID="captchaTextBox" Text="Przepisz kod z obrazka:" />
                <div class="col-md-9">
                    <asp:TextBox ID="captchaTextBox" runat="server"
CssClass="form-control"></asp:TextBox>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-offset-3 col-md-9">
                    <asp:Image ID="captchaImage" runat="server"
ImageUrl="~/CaptchaImage.aspx" />
                    <asp:Label ID="captchaLabel" runat="server" CssClass="label
label-danger"></asp:Label>
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-offset-3 col-md-9">
                    <asp:Button runat="server" ID="buttonAdd"
ValidationGroup="Validation" CssClass="btn btn-default" Text="Subskrybuj"
OnClick="ButtonAdd_OnClick" />
                    <asp:Button runat="server" ID="buttonRemove"
ValidationGroup="Validation" CssClass="btn btn-default" Text="Anuluj
subskrypcję" OnClick="ButtonRemove_OnClick" />
                </div>
            </div>
        </div>
    </div>
</asp:Content>
```

Jedynym niestandardowym elementem jest tutaj obrazek captcha. Dynamicznie generowany obrazek będzie pobierany ze strony **CaptchaImage.aspx**, którą dodamy teraz do projektu:

1. Kliknij prawym przyciskiem myszy na projekcie **SerwisNewsletter** i z menu kontekstowego wybierz **Add/New Item…**.
2. Pojawi się nowe okno dialogowe. W jego lewej części wybierz **Installed/Visual**

C#/Web/Web Forms, a następnie wskaż pozycję **Web Form**. W polu **Name** wpisz nazwę **CaptchaImage.aspx** i zatwierdź ustawienia klikając przycisk **Add** – rysunek 6-8.

Ciekawsza jest zawartość pliku **CaptchaImage.aspx.cs** zawierająca logikę generowania obrazka captcha. Sam proces generowania kodu rozpoczyna się od wygenerowania losowego tekstu składającego się z 5 liter lub cyfr - metoda **GenerateRandomCode**, listing 6-4. Następnie kody znaków są zapamiętane w sesji – metoda **Page_Load** – tak, aby możliwe było porównanie ciągu wprowadzonego przez użytkownika z wygenerowanym wcześniej napisem. W ostatnim kroku jest generowany graficzny odpowiednik ciągu znaków. Jest on zwracany w odpowiedzi na żądanie do strony. Wszystkie te kroki przedstawia metoda **Page_Load** z listingu 6-5. Dodatkowo w nagłówku pliku należy dodać przestrzeń nazw **using System.IO**.



Rysunek 6-8
Dodanie klasy generującej captcha

Listing 6-4
Metoda GenerateRandomCode z pliku CaptchaImage.aspx.cs

```
/// <summary>
/// Generuje losowy kod składający się z 5 cyfr lub liter
/// </summary>
/// <returns></returns>
private string GenerateRandomCode()
{
    var random = new Random();
    var code = string.Empty;
    for (var i = 0; i < 5; i++)
    {
        var j = random.Next(3);
        int ch;
        switch (j)
        {
            case 0:
                ch = random.Next(0, 9);
                code = code + ch;
                break;
            case 1:
                ch = random.Next(65, 90);
                code = code + Convert.ToChar(ch);
                break;
            case 2:
                ch = random.Next(97, 122);
```

```
                code = code + Convert.ToChar(ch);
                break;
            }
        }
        return code;
    }
```

Listing 6-5
Metoda Page_Load z pliku CaptchaImage.aspx.cs

```
protected void Page_Load(object sender, EventArgs e)
{
    //Dodanie do sesji losowo wygenerowanego tekstu
    Session["CaptchaImageText"] = GenerateRandomCode();
    //Stworzenie obrazka przedstawiającego wygenerowany tekst
    var captchaImage = new RandomImage(Session["CaptchaImageText"].ToString(),
280, 100);
    Response.Clear();
    Response.ContentType = "image/jpeg";
    using (var memoryStream = new MemoryStream())
    {
        captchaImage.Image.Save(memoryStream, ImageFormat.Jpeg);
        //Zapis obrazka jako odpowiedź na żądanie do strony
        memoryStream.WriteTo(Response.OutputStream);
    }
}
```

Jak można zauważyć, aby wygenerować obrazek z kodu znaków korzystamy tutaj z własnej klasy **RandomImage**, która rozszerza standardową klasę Image. Aby dodać ją do projektu musimy:

1. Kliknij prawym przyciskiem myszy na projekcie **SerwisNewsletter** i z menu kontekstowego wybierz **Add/New Item…**.
2. Pojawi się nowe okno dialogowe. W jego lewej części wybierz **Installed/Visual C#/Code**, a następnie wskaż pozycję **Class**. W polu **Name** wpisz nazwę **RandomImage** i zatwierdź ustawienia klikając przycisk **Add**.
3. Do pliku **RandomImage.cs** wpisz kod z listingu 6-6. Uzupełnij brakujące przestrzenie nazw.

Główna funkcjonalność klasy **RandomImage** zawarta jest w metodzie **GenerateImage**. Zadaniem tej metody jest przygotowanie obrazka o zadanym rozmiarze, a następnie naniesienie na niego tekstu, tak aby cały zmieścił się na obrazku. Ostatnim etapem jest dodanie szumu w postaci losowo rozmieszczonych niebieskich elips, których zadaniem jest utrudnienie automatycznego rozpoznawania tekstu z obrazka, co pozwoliło by obejść antyspamowe zabezpieczenie.

Listing 6-6
Klasa RandomImage

```
internal sealed class RandomImage
{
    #region Private Definitions

    private readonly int width;
    private readonly int height;
    private readonly Random random = new Random();
    private string text;
    private Bitmap image;

    #endregion

    public RandomImage(string text, int width, int height)
    {
        this.text = text;
        this.width = width;
```

```
        this.height = height;
        GenerateImage();
    }

    public Bitmap Image
    {
        get { return image; }
    }

    /// <summary>
    /// Tworzy obrazek o zadanym rozmiarze, na podstawie
                dostarczonego tekstu
    /// </summary>
    private void GenerateImage()
    {
        //stworzenie bitmapy o zadanym rozmiarze
        var bitmap = new Bitmap
          (width, height, PixelFormat.Format32bppArgb);
        var graphics = Graphics.FromImage(bitmap);
        graphics.SmoothingMode = SmoothingMode.AntiAlias;
        var rect = new Rectangle(0, 0, width, height);
        var hatchBrush = new HatchBrush(HatchStyle.SmallConfetti,
            Color.LightGray, Color.White);
        graphics.FillRectangle(hatchBrush, rect);
        SizeF size;
        float fontSize = rect.Height + 1;
        Font font;

        do
        {
            //dopasowanie wielkości czcionki do rozmiarów obrazka
            fontSize--;
            font = new Font(FontFamily.GenericSansSerif,
                fontSize, FontStyle.Bold);
            size = graphics.MeasureString(text, font);
        } while (size.Width > rect.Width);

        var format = new StringFormat
        {
            Alignment = StringAlignment.Center,
            LineAlignment = StringAlignment.Center
        };

        var path = new GraphicsPath();
        //dodanie tekstu do obrazka
        path.AddString(text, font.FontFamily,
            (int)font.Style, font.Size, rect, format);
        PointF[] points =
        {
            new PointF(random.Next(rect.Width)/4f, random.Next(
                rect.Height)/4f),
            new PointF(rect.Width - random.Next(rect.Width)/4f,
                random.Next(rect.Height)/4f),
            new PointF(random.Next(rect.Width)/4f,
                rect.Height - random.Next(rect.Height)/4f),
            new PointF(rect.Width - random.Next(rect.Width)/4f,
                rect.Height - random.Next(rect.Height)/4f)
        };
        var matrix = new Matrix();
        matrix.Translate(0F, 0F);
        path.Warp(points, rect, matrix, WarpMode.Perspective, 0F);
        hatchBrush = new HatchBrush(HatchStyle.Percent10,
            Color.Black, Color.SkyBlue);
        graphics.FillPath(hatchBrush, path);
        var max = Math.Max(rect.Width, rect.Height);
        for (var i = 0; i < (int)(rect.Width * rect.Height / 30F); i++)
        {
            var x = random.Next(rect.Width);
            var y = random.Next(rect.Height);
            var w = random.Next(max / 50);
```

```
            var h = random.Next(max / 50);
            //dodanie szumu do obrazka aby utrudnić
            //próby automatycznego rozpoznawania tekstu
            graphics.FillEllipse(hatchBrush, x, y, w, h);
        }
        font.Dispose();
        hatchBrush.Dispose();
        graphics.Dispose();
        image = bitmap;
    }
}
```

W ostatnim kroku uzupełnimy kod **Default.aspx.cs** o metody odpowiedzialne za dodawanie i anulowanie subskrypcji newslettera. W typ celu musimy najpierw utworzyć obiekt klasy kontekstowej wygenerowanej dzięki mechanizmowi **LINQ to SQL**. Wyznaczymy także adres strony domowej budowanego serwisu, tak aby poinformować użytkownika mailowo, gdzie może dokonać subskrypcji/desubskrypcji - listing 6-7. Listing zawiera także metodę **Page_Load**, która oprócz czyszczenia systemowych komunikatów sprawdza czy zalogowany użytkownik jest administratorem. Jeśli jest, to odsyła go na stronę z listą subskrypcji. Dodatkowo w nagłówku pliku należy dodać przestrzeń nazw **using System.Net.Mail**.

Listing 6-7
Modyfikacje pliku Default.aspx.cs

```
public partial class Default : Page
{
#region Private Definitions

    /// <summary>
    /// Obiekt odpowiedzialny za zarządzanie tabelą subskrybentów
    /// </summary>
    private readonly SubscriptionsDataContext dataContext =
        new SubscriptionsDataContext();
    /// <summary>
    /// Wyznacza adres URL głównej strony serwisu
    /// </summary>
    /// <value>
    /// Adres URL głównej strony serwisu
    /// </value>
    private string RootAbsolutePath
    {
        get
        {
            return HttpContext.Current.Request.Url.Scheme
                   + "://"
                   + HttpContext.Current.Request.Url.Authority
                   + HttpContext.Current.Request.ApplicationPath;
        }
    }

    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        //Administrator nie może dodać się do subskrypcji
        if (User.IsInRole("administrator"))
        {
            Server.Transfer("Subscriptions.aspx");
        }

        if (!IsPostBack)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Empty;
        }
    }
}
```

Metoda odpowiedzialna za subskrypcję newslettera jest przedstawiona na listingu 6-8. Schemat jej działania jest stosunkowo prosty. Najpierw pobieramy z sesji wygenerowany kod Captcha i porównujemy go z kodem podanym przez użytkownika. Jeśli kody się nie zgadzają, to informujemy o tym użytkownika i przerywamy operację subskrypcji. Jeśli kody są zgodne, to pobieramy dane o zalogowanym użytkowniku oraz email podany na formularzu i sprawdzamy czy w bazie danych nie istnieje już subskrypcja z podanym adresem. Jeśli istnieje, to informujemy użytkownika o niepowodzeniu subskrypcji. Jeśli jednak wszystko przebiegnie pomyślnie, to dodajemy nowy rekord do tabeli **Subscription** i powiadamiamy użytkownika o dokonanej właśnie subskrypcji. W emailu z powiadomieniem zawarty jest odsyłacz do strony głównej serwisu oraz informacja jak można anulować subskrypcję.

Listing 6-8
Metoda ButtonAdd_OnClick

```
protected void ButtonAdd_OnClick(object sender, EventArgs e)
{
    //Porównanie wygenerowanego kody z tekstem wpisanym
    //przez użytownika
    if (captchaTextBox.Text == Session["CaptchaImageText"].ToString())
    {
        captchaLabel.Text = string.Empty;
        captchaTextBox.Text = string.Empty;
    }
    else
    {
        captchaLabel.Text = "Niepoprawny kod!!!";
        captchaTextBox.Text = string.Empty;
        return;
    }

    string username = User.Identity.Name;
    try
    {
        //Sprawdzenie czy podany przez użytkownika adres
        //e-mail nie istnieje już w bazie danych
        var subscription =
            dataContext.Subscriptions.FirstOrDefault(
                n => n.username == username && n.email == textBoxEmail.Text);

        if (subscription != null)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Adres e-mail {0} istnieje już w
                bazie subskrybentów",
                textBoxEmail.Text);
            return;
        }
        else
        {
            //Stworzenie nowego rekordu tabeli Subscription
            subscription = new Subscription()
            {
                email = textBoxEmail.Text,
                username = username
            };

            dataContext.Subscriptions.InsertOnSubmit(subscription);
            errorLabel.Text = string.Empty;
            infoLabel.Text = string.Format("Adres e-mail {0} dodano do
                newslettera", textBoxEmail.Text);
        }
    }
    catch (Exception)
    {
        infoLabel.Text = string.Empty;
        errorLabel.Text =
            string.Format(
                "Wystąpił błąd podczas próby dodania adresu e-mail {0}.
```

```
                Spróbuj ponownie później.",
                textBoxEmail.Text);
        return;
    }
    try
    {
        //Dodanie rekordu do tabeli
        dataContext.SubmitChanges();
        //Stworzenie wiadomości e-mail z informacją
        //o dokonanej subskrypcji newslettera
        var message = new MailMessage
        {
            Subject = "Newsletter",
            Body = string.Format(
                "Twój adres e-mail został dodany do subskrypcji newslettera.")
+
                " Jeśli chcesz zrezygnować z otrzymywania wiadomości wejdź
                na stronę: {0}, podaj swój e-mail i wybierz opcję
                'Anuluj subskrypcję'.", RootAbsolutePath)
        };

        var client = new SmtpClient();
        message.To.Add(textBoxEmail.Text);
        //Wysłanie wiadomości na podany adres e-mail
        client.Send(message);
    }
    catch (Exception)
    {
        infoLabel.Text = string.Empty;
        errorLabel.Text =
            string.Format(
                "Adres e-mail {0} dodano do newslettera, ale wystąpił błąd
                podczas wysyłania potwierdzenia.",
                textBoxEmail.Text);
    }
}
```

Wskazówka
Usługa poczty Gmail wymaga zmiany ustawień poziomu bezpieczeństwa. Należy zalogować się na konto Gmail wskazane w budowanej aplikacji, następnie przejść na stronę ***https://accounts.google.com/DisplayUnlockCaptcha*** i kliknąć **Kontynuuj**.

Analogicznie podejście prezentuje metoda odpowiedzialna za anulowanie subskrypcji - listing 6-9. Tutaj także najpierw porównujemy kod Captcha oraz pobieramy dane zalogowanego użytkownika. Jeśli podany email nie istnieje w bazie danych, to przerywamy proces desubskrypcji i informujemy użytkownika o błędzie. W przeciwnym przypadku usuwamy znaleziony rekord z bazy danych i wysyłamy email z powiadomieniem o anulowaniu subskrypcji oraz możliwości jej ponowienia za pośrednictwem strony domowej serwisu.

Listing 6-9
Metoda ButtonRemove_OnClick

```
protected void ButtonRemove OnClick(object sender, EventArgs e)
{
    //Porównanie wygenerowanego kody z tekstem wpisanym
    //przez użytownika
    if (captchaTextBox.Text == Session["CaptchaImageText"].ToString())
    {
        captchaLabel.Text = string.Empty;
        captchaTextBox.Text = string.Empty;
    }
    else
    {
        captchaLabel.Text = "Niepoprawny kod!!!";
        captchaTextBox.Text = string.Empty;
```

```
        return;
    }

    string username = User.Identity.Name;
    Subscription subscription = null;
    try
    {
        //Sprawdzenie czy podany przez użytkownika adres
        //e-mail istnieje w bazie danych
        subscription =
            dataContext.Subscriptions.FirstOrDefault(
                n => n. username == username && n.email == textBoxEmail.Text);

        if (subscription != null)
        {
            dataContext.Subscriptions.DeleteOnSubmit(subscription);
            errorLabel.Text = string.Empty;
            infoLabel.Text = string.Format("Adres email {0} usunięto
                z newslettera", textBoxEmail.Text);
        }
        else
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Adres e-mail {0} nie istnieje
                w bazie subskrybentów",
                textBoxEmail.Text);
        }
    }
    catch (Exception)
    {
        infoLabel.Text = string.Empty;
        errorLabel.Text = string.Format("Wystąpił błąd podczas próby usunięcia
            adresu e-mail {0}. Spróbuj ponownie później.",
            textBoxEmail.Text);
    }

    if (subscription != null)
    {
        try
        {
            //Usunięcie rekordu z tabeli
            dataContext.SubmitChanges();
            //Stworzenie wiadomości e-mail z informacją
            //o anulowaniu subskrypcji newslettera
            var message = new MailMessage
            {
                Subject = "Newsletter",
                Body = string.Format(
                    "Twój adres e-mail został usunięty z subskrypcji
                    newslettera.)


                    Jeśli ponownie chcesz otrzymywać wiadomości
                    wejdź na stronę: {0}, podaj swój e-mail i wybierz opcję
                    'Subskrybuj'.", RootAbsolutePath)
            };

            var client = new SmtpClient();
            message.To.Add(textBoxEmail.Text);
            //Wysłanie wiadomości na podany adres e-mail
            client.Send(message);
        }
        catch (Exception)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Adres e-mail {0} usunięto
                z newslettera, ale wystąpił błąd podczas wysyłania
                potwierdzenia.", textBoxEmail.Text);
        }
```

```
    }
}
```

W ostatnim kroku konfigurujemy konto pocztowe serwisu. Z niego będą rozsyłane wszystkie powiadomienia. Konto konfigurujemy poprzez odpowiedni wpis w pliku **Web.config** - listing 6-10.

Listing 6-10
Konfiguracja konta pocztowego.

```
<system.net>
    <mailSettings>
      <smtp from="azureexamples@gmail.com" deliveryMethod="Network">
        <network host="smtp.gmail.com" port="587"
          userName="azureexamples@gmail.com"
          password="tajne_Haslo#2014" defaultCredentials="false"
          enableSsl="true" />
      </smtp>
    </mailSettings>
</system.net>
```

Wskazówka
Czasami do komunikacji z serwerem SMTP z użyciem protokołu TLS/SSL są używane porty 25 lub 465.

Po dokonaniu powyższych zmian możemy już uruchomić stronę domową serwisu w chmurze. W celu wdrożenia serwisu *Newsletter* w chmurze Azure bezpośrednio ze środowiska Visual Studio należy wykonać następujące kroki:

1. Zaloguj się do portalu zarządzania Azure (https://portal.azure.com) i w odpowiedniej lokalizacji utwórz website o adresie URL: z05*imie*.azurewebsites.net
2. Zaimportuj profil utworzonego website do lokalnego komputera.
3. Z poziomu Visual Studio 2017 opublikuj w chmurze aplikację Newsletter korzystając z zaimportowanego profilu.

~~1. Kliknij prawym przyciskiem myszy na projekcie **SerwisNewsletter** i z menu kontekstowego wybierz **Publish…**(konieczne będzie zalogowanie na konto Azure).~~

~~2. Z listy opcji **Select a publish target** wybierz **Microsoft Azure Websites**. Pojawi się nowe okno z wyborem usługi *Website*. Jeśli nie utworzyłeś wcześniej takiej usługi wybierz opcję **New…**.~~

~~3. W formularzu tworzenia witryny wypełnij pole **Site name** nazwą serwisu, z listy **Region** wybierz **North Europe**, jako serwer **Database server** wybierz serwer, na którym utworzyłeś bazę danych, podaj hasło do bazy danych – rysunek 6-9.~~

~~4. Na koniec kliknij **Create**. Zostanie utworzona usługa *Witryna sieci Web* oraz pobrany zostanie jej profil, który posłuży do wdrażania aplikacji do chmury Azure.~~

~~5. W kolejnym oknie wybierz **Publish** – rysunek 6-10. Jeśli nie wystąpią żadne błędy otworzy się okno przeglądarki ze stroną domową serwisu.~~

~~Rysunek 6-9~~
~~Dodawanie Witryny sieci Web~~

~~Rysunek 6-10~~
~~Ekran wdrażania aplikacji do chmury Azure~~

6.7 Logowanie użytkowników

Teraz kiedy mamy już zbudowaną i testowo wdrożoną stronę domową serwisu możemy przystąpić do realizacji logowania użytkowników. Jak wcześniej wspomnieliśmy do autoryzacji użytkowników wykorzystamy konta utworzone przez nich na portalach społecznościowych. Aby móc skorzystać z takiej możliwości musimy dodać budowany serwis do aplikacji na każdym portalu społecznościowym. Na każdym z dużych portali społecznościowych taki proces przebiega bardzo podobnie.

OPISAĆ CAŁY PROCES SZCZEGÓŁOWO. WYKORZYSTAĆ UMIEJĘTNOŚCI z 04_zestaw/zad. 3

Listing 6-13
Widok strony Login.aspx

```
<%@ Page Title="Zaloguj się" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Login.aspx.cs"
Inherits="SerwisNewsletter.Account.Login" Async="true" %>

<%@ Register Src="~/Account/OpenAuthProviders.ascx" TagPrefix="uc"
TagName="OpenAuthProviders" %>

<asp:Content runat="server" ID="BodyContent"
ContentPlaceHolderID="MainContent">
    <h2>Zaloguj się przy użyciu zewnętrznego serwisu.</h2>
    <div class="row">
        <div class="col-md-4">
            <section id="socialLoginForm">
                <uc:OpenAuthProviders runat="server" ID="OpenAuthLogin" />
            </section>
        </div>
    </div>
</asp:Content>
```

Ponieważ używamy tylko logowania poprzez portale społecznościowe, to kod z pliku **Login.aspx.cs** należy skasować pozostawiając jedynie pustą klasę **Login**.

6.8 Przeglądanie listy aktywnych subskrypcji

Zajmiemy się teraz stronami administratora serwisu. Na początku utworzymy stronę, która pozwoli na przeglądanie listy aktywnych subskrypcji, aktualizację ich danych lub usuwanie wybranych subskrypcji z bazy danych. W tym celu musimy dodać do projektu nową stronę analogicznie jak to robiliśmy w wypadku **CaptchaImage** w rozdziale 6.6. Nową stronę nazwiemy **Subscriptions** - listing 6-14. Jej głównym elementem jest kontrolka GridView, która pozwala wyświetlić dane z bazy danych w tabelarycznym układzie.

Listing 6-14
Widok strony Subscriptions.aspx

```
<%@ Page Title="Subskrypcje" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Subscriptions.aspx.cs"
Inherits="SerwisNewsletter.Subscriptions" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent"
runat="server">
    <div class="jumbotron">
        <h1>Usługi Microsoft Azure</h1>
        <h2>Serwis Newsletter - automatyczne powiadomienia email</h2>
    </div>
    <div class="panel panel-default">
        <div class="panel-heading">Lista subskrybentów serwisu.</div>
```

```
        <div class="panel-body">
            <asp:Label runat="server" ID="infoLabel" CssClass="label label-
success"></asp:Label><br />
            <asp:Label runat="server" ID="errorLabel" CssClass="label label-
danger"></asp:Label><br />
            <asp:GridView runat="server" ID="gridViewSubscriptions"
AutoGenerateColumns="False"
                DataKeyNames="id" AllowPaging="True" PageSize="5"
CssClass="table table-striped table-bordered">
                <Columns>
                    <asp:TemplateField HeaderText="#">
                        <ItemTemplate>
                            <%# Container.DataItemIndex + 1 %>
                        </ItemTemplate>
                    </asp:TemplateField>
                    <asp:BoundField DataField="userName"
HeaderText="Użytkownik"/>
                    <asp:BoundField DataField="email" HeaderText="Email"/>
                    <asp:CommandField CancelText="Anuluj" DeleteText="Usuń"
ShowDeleteButton="True"
                        EditText="Edytuj" ShowEditButton="True"
UpdateText="Zapisz" />
                </Columns>
            </asp:GridView>
        </div>
    </div>
</asp:Content>
```

Z podaną kontrolką wiążemy i obsługujemy zdarzenia, co pozwoli obsłużyć operacje wykonywane przez administratora. Główne funkcje są zapisane w pliku **Subscriptions.aspx.cs** - listing 6-15. Standardowo rozpoczynamy od inicjalizacji klasy kontekstowej, która odpowiada za operacje na bazie danych. Następnie, korzystając z wbudowanych zdarzeń kontroli GridView, podpinamy pod nie potrzebne metody. Metoda **GridViewSubscriptionsOnRowUpdating** pozwala na aktualizację danych subskrypcji. Rozpoczynamy od wyszukiwania subskrypcji o podanym id w bazie danych. Następnie aktualizujemy rekord tabeli, po czym zatwierdzamy zmiany na bazie danych. Ostatnim etapem jest wysłanie emaila z powiadomieniem o dokonanej aktualizacji na adres podany w subskrypcji. Analogicznie przebiega wykonanie metody **GridViewSubscriptionsOnRowDeleting**, która usuwa wybraną subskrypcję z bazy danych. W tym przypadku usuwamy rekord tabeli **Subscription** o podanym id, a następnie wysyłamy do użytkownika powiadomienie o dokonanej akcji. Po każdej wykonanej komendzie wywoływana jest metoda **PopulateGridViewSubscriptions**, która odświeża dane w kontrolce GridView.

Listing 6-15
Kod klasy Subscriptions.aspx.cs

```
public partial class Subscriptions : System.Web.UI.Page
{
    #region Private Definitions

    /// <summary>
    /// Obiekt odpowiedzialny za zarządzanie tabelą subskrybentów
    /// </summary>
    private readonly SubscriptionsDataContext dataContext =
        new SubscriptionsDataContext();

    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        //Jeśli zalogowany użytkownik nie jest
        //administratorem przekierujmy go do strony logowania
        if (!User.IsInRole("administrator"))
        {
            Server.Transfer("~/Account/AdminLogin.aspx");
```

```
        }

        if (!IsPostBack)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Empty;
            PopulateGridViewSubscriptions();
        }
        //Powiązanie metod ze zdarzeniami kontrolki GridView
        gridViewSubscriptions.RowEditing +=
            GridViewSubscriptionsOnRowEditing;
        gridViewSubscriptions.RowDeleting +=
            GridViewSubscriptionsOnRowDeleting;
        gridViewSubscriptions.RowUpdating +=
            GridViewSubscriptionsOnRowUpdating;
        gridViewSubscriptions.RowCancelingEdit +=
            GridViewSubscriptionsOnRowCancelingEdit;
        gridViewSubscriptions.PageIndexChanging +=
            GridViewSubscriptionsOnPageIndexChanging;
    }

    private void GridViewSubscriptionsOnRowCancelingEdit(object sender,
        GridViewCancelEditEventArgs gridViewCancelEditEventArgs)
    {
        gridViewSubscriptions.EditIndex = -1;
        PopulateGridViewSubscriptions();
    }

    private void GridViewSubscriptionsOnRowUpdating(object sender,
        GridViewUpdateEventArgs gridViewUpdateEventArgs)
    {
        var subscriptionId = (int)gridViewUpdateEventArgs.Keys["id"];
        var email = (string)gridViewUpdateEventArgs.NewValues["email"];
        var username = (string)gridViewUpdateEventArgs.NewValues["userName"];
        try
        {
            //Wyszukanie subskrypcji o podanym ID
            var subscription = dataContext.Subscriptions.
                First(n => n.Id == subscriptionId);
            subscription.email = email;
            subscription.username = username;
            //Aktualizacja danych subskrypcji
            dataContext.SubmitChanges();
            gridViewSubscriptions.EditIndex = -1;
            //Odświeżenie tabeli subskrypcji
            PopulateGridViewSubscriptions();
            errorLabel.Text = string.Empty;
            infoLabel.Text = "Dane zostały zapisane";
        }
        catch (Exception)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Wystąpił błąd podczas
                próby aktualizacji danych związanych z adresem
                email {0}. Spróbuj ponownie później.", email);
            return;
        }

        try
        {
            //Przygotowanie maila z powiadomieniem
            //o aktualizacji danych subskrypcji
            var message = new MailMessage
            {
                Subject = "Newsletter",
                Body = string.Format("Twoje dane, związane
                    z subskrybcją newslettera, zostały zaktualizowane.")
            };
            //Zastosowano klienta poczty z przestrzeni
           //nazw System.Net.Mail
```

```
            var client = new SmtpClient();
            message.To.Add(email);
            client.Send(message);
        }
        catch (Exception)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Wystąpił błąd podczas próby
                wysłania powiadomienia o aktualizacji danych związanych
                z adresem email {0}. Spróbuj ponownie później.", email);
        }
    }

    /// <summary>
    /// Wypełnia danymi subskrypcji kontrolkę GridView
    /// </summary>
    private void PopulateGridViewSubscriptions()
    {
        gridViewSubscriptions.DataSource = dataContext.Subscriptions;
        gridViewSubscriptions.DataBind();
    }
    private void GridViewSubscriptionsOnRowDeleting(object sender,
        GridViewDeleteEventArgs gridViewDeleteEventArgs)
    {
        var subscriptionId = (int)gridViewDeleteEventArgs.Keys["id"];
        var email = (string)gridViewDeleteEventArgs.Values["email"];
        try
        {
            //Wyszukanie subskrypcji o podanym ID
            var subscription = dataContext.Subscriptions.
                First(n => n.Id == subscriptionId);
            dataContext.Subscriptions.DeleteOnSubmit(subscription);
            //Usunięcie subskrypcji z bazy danych
            dataContext.SubmitChanges();
            //Odświeżenie tabeli subskrypcji
            PopulateGridViewSubscriptions();
            errorLabel.Text = string.Empty;
            infoLabel.Text = string.Format("Adres email {0} usunięto
                z newslettera.", subscription.email);
        }
        catch (Exception)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Wystąpił błąd podczas próby
                usunięcia danych związanych z adresem email {0}. Spróbuj
                ponownie później.", email);
            return;
        }
        try
        {
            //Przygotowanie maila z powiadomieniem o usunięciu subskrypcji
            var message = new MailMessage
            {
                Subject = "Newsletter",
                Body = string.Format("Twój adres email {0} został usunięty
                     z subskrypcji newslettera.", email)
            };

            var client = new SmtpClient();
            message.To.Add(email);
            client.Send(message);
        }
        catch (Exception)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Format("Wystąpił błąd podczas próby
                wysłania powiadomienia o usunięciu danych związanych
                z adresem email {0}. Spróbuj ponownie później.", email);
        }
    }
```

```
    private void GridViewSubscriptionsOnRowEditing(object sender,
        GridViewEditEventArgs gridViewEditEventArgs)
    {
        gridViewSubscriptions.EditIndex = gridViewEditEventArgs.NewEditIndex;
        PopulateGridViewSubscriptions();
    }

    private void GridViewSubscriptionsOnPageIndexChanging(object sender,
        GridViewPageEventArgs e)
    {
        gridViewSubscriptions.PageIndex = e.NewPageIndex;
        PopulateGridViewSubscriptions();
    }
}
```

6.9 Przeglądanie listy aktywnych subskrypcji

Kolejną stroną administratora serwisu jest strona odpowiedzialna za rozsyłanie wiadomości do subskrybentów, czyli głównej funkcjonalności serwisu *Newsletter*. Podobnie jak poprzednio musimy dodać do projektu nową stronę o nazwie **NewsletterTemplate**. Widok tej strony przedstawia listing 6-16. Sama strona składa się z prostego formularza, w którym podajemy tytuł wiadomości oraz możemy załączyć treść wiadomości w postaci pliku txt lub html. W przypadku wiadomości html znaczniki zostaną wyrenderowane podobnie jak to się dzieje w przeglądarce, dzięki czemu możemy wystylizować treść wiadomości.

Listing 6-16
Widok strony NewsletterTemplate.aspx

```
<%@ Page Title="Szablon newslettera" Language="C#"
MasterPageFile="~/Site.Master" AutoEventWireup="true"
CodeBehind="NewsletterTemplete.aspx.cs"
Inherits="SerwisNewsletter.NewsletterTemplete" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent"
runat="server">
    <div class="jumbotron">
        <h1>Usługi Microsoft Azure</h1>
        <h2>Serwis Newsletter - automatyczne powiadomienia email</h2>
        <p class="lead">
            Wypełnij szablon newslettera i roześlij do wszystkich
subskrybentów.
        </p>
    </div>
    <div class="panel panel-default">
        <div class="panel-heading">Formularz zgłoszeniowy</div>
        <div class="panel-body">
            <asp:Label runat="server" ID="infoLabel" CssClass="label label-
success"></asp:Label><br />
            <asp:Label runat="server" ID="errorLabel" CssClass="label label-
danger"></asp:Label><br />
            <asp:ValidationSummary runat="server" ValidationGroup="Validation"
CssClass="alert alert-danger" />
            <asp:RequiredFieldValidator runat="server"
ValidationGroup="Validation" Display="None"
                ErrorMessage="Treść wiadomości jest wymagana"
ControlToValidate="fileUploadContent" />
            <asp:RegularExpressionValidator runat="server"
ControlToValidate="fileUploadContent"
                ErrorMessage="Proszę wybrać plik .txt lub .html"
ValidationGroup="Validation" Display="None"
                                
ValidationExpression="(.*\.([Tt][Xx][Tt]|[Hh][Tt][Mm][Ll])$)"></asp:RegularExpr
essionValidator>
            <div class="form-group">
                <asp:Label runat="server" CssClass="control-label col-md-3"
```

```
AssociatedControlID="textboxTopic" Text="Temat wiadomości:" />
                <div class="col-md-9">
                    <asp:TextBox runat="server" ID="textboxTopic"
CssClass="form-control" />
                </div>
            </div>
            <div class="form-group">
                <asp:Label runat="server" CssClass="control-label col-md-3"
AssociatedControlID="fileUploadContent" Text="Treść wiadomości" />
                <div class="col-md-9">
                    <asp:FileUpload runat="server" ID="fileUploadContent"
CssClass="form-control" />
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-offset-3 col-md-9">
                    <asp:Button runat="server" ID="buttonSend"
ValidationGroup="Validation" CssClass="btn btn-default" Text="Wyślij"
OnClick="ButtonSend_OnClick" />
                </div>
            </div>
        </div>
    </div>
</asp:Content>
```

Z kolei plik **NewsletterTemplate.aspx.cs** - listing 6-17 – przedstawia klasę, której metody służą do rozsyłania wiadomości do subskrybentów portalu. Tak jak poprzednio najpierw tworzymy obiekt klasy kontekstowej, który pozwala pobrać z bazy danych aktywne subskrypcje newslettera. Przy załadowaniu strony sprawdzamy, czy zalogowany użytkownik jest administratorem. Jeśli nie jest, to odsyłamy go do strony logowania administratora, w przeciwnym razie może on rozesłać wiadomość do subskrybentów. Główną metodą w tej klasie jest **ButtonSend_OnClick**. To ona odpowiada za proces rozsyłania wiadomości. W pierwszej kolejności należy sprawdzić poprawność danych wejściowych, typ wiadomości - txt lub html - oraz rozmiar pliku. Maksymalny dopuszczalny rozmiar pliku to 1MB. Próba dodania większego kończy się błędem i powiadomieniem administratora. Następnie sprawdzane jest rozszerzenie pliku, a potem na podstawie jego zawartości przygotowywana jest treść newslettera. Później korzystając z obiektu klasy kontekstowej pobieramy listę wszystkich subskrypcji i rozsyłamy przygotowaną wiadomość na adresy email użytkowników. Jeśli w trakcie rozsyłania nastąpi błąd to zapisujemy informację o tym na liście wyjątków. Na koniec wyświetlamy administratorowi krótkie podsumowanie z liczbą poprawie i błędnie rozesłanych wiadomości oraz tworzymy bardziej szczegółowy raport w postaci pliku xml - metoda **SaveMailErrors**.

Listing 6-17
Kod klasy NewsletterTemplate.aspx.cs

```
public partial class NewsletterTemplete : System.Web.UI.Page
{
    #region Private Definitions

    /// <summary>
    /// Obiekt odpowiedzialny za zarządzanie tabelą subskrybentów
    /// </summary>
    private readonly SubscriptionsDataContext dataContext =
        new SubscriptionsDataContext();

    #endregion

    protected void Page_Load(object sender, EventArgs e)
    {
        //Jeśli zalogowany użytkownik nie jest
        //administratorem przekierujmy go do strony logowania
        if (!User.IsInRole("administrator"))
        {
            Server.Transfer("Account/AdminLogin.aspx");
        }
```

```
        if (!IsPostBack)
        {
            infoLabel.Text = string.Empty;
            errorLabel.Text = string.Empty;
        }
    }

    protected void ButtonSend_OnClick(object sender, EventArgs e)
    {
        infoLabel.Text = string.Empty;
        errorLabel.Text = string.Empty;
        //Sprawdzenie czy plik został wgrany
        if (!fileUploadContent.HasFile)
        {
            errorLabel.Text = "Plik z treścią newslettera jest wymagany";
            return;
        }

        HttpPostedFile file = fileUploadContent.PostedFile;
        //Ograniczenie maksymalnego rozmiaru pliku do 1MB
        if (file.ContentLength > 1024 * 1024)
        {
            errorLabel.Text = "Plik może mieć max. 1MB";
            return;
        }

        string fileContent;
        using (var reader = new StreamReader(file.InputStream))
        {
            fileContent = reader.ReadToEnd();
        }

        if (string.IsNullOrWhiteSpace(fileContent))
        {
            errorLabel.Text = "Plik nie może być pusty";
            return;
        }
        //Pobranie rozszerzenia pliku
        //Dadano przestrzeń nazw System.IO
        string extension = Path.GetExtension(file.FileName).ToLower();
        //Utworzenie szablonu wiadomości
        var message = new MailMessage
        {
            Subject = textboxTopic.Text,
            Body = fileContent,
            IsBodyHtml = extension == ".html"
        };
        //Zastosowano klienta poczty z przestrzeni
       //nazw System.Net.Mail
        var client = new SmtpClient();
        var mailErrors = new List<MailError>();
        //Wysłanie wiadomości na adresy email aktywnych subskrypcji
        foreach (var subscription in dataContext.Subscriptions)
        {
            try
            {
                message.To.Clear();
                message.To.Add(subscription.email);
                client.Send(message);
            }
            catch (Exception ex)
            {
                //W przypadku błędu przy rozsyłaniu newslettera
                //dodanie komunikatu do listy błędów
                var mailError = new MailError()
                {
                    Mail = subscription.email,
                    UserName = subscription.username,
                    ErrorMessage = ex.Message
```

```
                };
                mailErrors.Add(mailError);
            }
        }
        infoLabel.Text = string.Format("Newsletter został rozesłany.
            Poprawnie wysłanych wiadomości: {0}. Błędów podczas wysyłania:
            {1}.", dataContext.Subscriptions.Count() - mailErrors.Count,
            mailErrors.Count);
        //Zapis raportu wysyłki do pliku XML
        SaveMailErrors(mailErrors);
    }

    /// <summary>
    /// Tworzy raport wysyłki newslettera w formacie XML
    /// </summary>
    /// <param name="mailErrors">Lista błędów jakie wystąpiły
        podczas rozsyłania</param>
    private void SaveMailErrors(IEnumerable<MailError> mailErrors)
    {
        //Zapis informacji o dacie rozesłania newsletter
        //Dodano przestrzeń nazw System.Xml.Linq
        var newsletter = new XElement("Newsletter",
            new XAttribute("SendDate",
                DateTime.Now.ToString("yyyy-mm-dd HH:mm")));
        //Zapis komunikatów błędów jakie wystąpiły
        //podczas rozsyłania
        foreach (MailError mailError in mailErrors)
        {
            var error = new XElement("MailError",
                new XAttribute("email", mailError.Mail),
                new XAttribute("username", mailError.UserName));
            error.SetValue(mailError.ErrorMessage);
            newsletter.Add(error);
        }
        //Sprawdzenie czy plik raportu już istnieje, jeśli nie
        //stworzenie nowego, w przeciwnym wypadku aktualizacja
        string file = Server.MapPath("~/App_Data/NewsletterReport.xml");
        if (File.Exists(file))
        {
            var xDocument = XDocument.Load(file);
            var xRoot = xDocument.Element("Report");
            if (xRoot != null)
            {
                xRoot.Add(newsletter);
            }
            xDocument.Save(file);
        }
        else
        {
            var xRoot = new XElement("Report");
            xRoot.Add(newsletter);
            var xDocument = new XDocument();
            xDocument.Add(xRoot);
            xDocument.Save(file);
        }
    }
}
```

Wskazówka
Przed wdrożeniem aplikacji do chmury należy w katalogu **App_Data** koniecznie utworzyć plik o nazwie **NewsletterReport.xml**.

Jeśli przyjrzymy się powyższemu kodowi to zauważymy wykorzystanie w nim klasy **MailError** – listing 6-18. Jest to bardzo prosta struktura, grupująca podstawowe informacje o błędzie jaki wystąpił podczas wysyłania newslettera. Do projektu dołączamy ją analogicznie

jak robiliśmy to z klasą **RandomImage** w podpunkcie 6.6.

Listing 6-18
Kod klasy MailError.cs

```
public class MailError
{
    public string Mail { get; set; }
    public string UserName { get; set; }
    public string ErrorMessage { get; set; }
}
```

6.10 Logowanie administratora serwisu

Administrator portalu *Newsletter* loguje się za pomocą danych jakimi baza została zasilona przy pierwszym uruchomieniu serwisu. Odmienny sposób logowania powoduje, że administrator otrzymał osobną stronę logowania. Dodajemy ją do projektu analogicznie jak inne strony. W tym przypadku dodamy ją do podfolderu **Account**, a sam plik będzie nosił nazwę **AdminLogin.aspx**. Z powodu tego, że administrator nie korzysta z logowania za pomocą portali społecznościowych to i widok strony logowania jest odmienny, niż w przypadku logowania zwykłego użytkownika - listing 6-19. Sama strona składa się z formularza z polami do wprowadzenia loginu i hasło oraz jednego przycisku.

Listing 6-19
Widok strony AdminLogin.aspx

```
<%@ Page Title="Zaloguj się" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="AdminLogin.aspx.cs"
Inherits="SerwisNewsletter.Account.AdminLogin" Async="true" %>

<asp:Content runat="server" ID="BodyContent"
ContentPlaceHolderID="MainContent">
    <h2>Zaloguj się przy użyciu lokalnego konta.</h2>
    <div class="row">
        <div class="col-md-8">
            <section id="loginForm">
                <div class="form-horizontal">
                    <h3>Na potrzeby testów dane do logowania to:<br />
                        Login - Admin, Hasło - P4$$word</h3>
                    <hr />
                    <asp:PlaceHolder runat="server" ID="ErrorMessage"
Visible="false">
                        <p class="text-danger">
                            <asp:Literal runat="server" ID="FailureText" />
                        </p>
                    </asp:PlaceHolder>
                    <div class="form-group">
                        <asp:Label runat="server" AssociatedControlID="Login"
CssClass="col-md-2 control-label">Login</asp:Label>
                        <div class="col-md-10">
                            <asp:TextBox runat="server" ID="Login"
CssClass="form-control" />
                            <asp:RequiredFieldValidator runat="server"
ControlToValidate="Login"
                                CssClass="text-danger" ErrorMessage="Login jest
wymagany." />
                        </div>
                    </div>
                    <div class="form-group">
                        <asp:Label runat="server"
AssociatedControlID="Password" CssClass="col-md-2 control-
label">Hasło</asp:Label>
                        <div class="col-md-10">
                            <asp:TextBox runat="server" ID="Password"
TextMode="Password" CssClass="form-control" />
                            <asp:RequiredFieldValidator runat="server"
```

```
ControlToValidate="Password" CssClass="text-danger" ErrorMessage="Hasło jest
wymagane." />
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="col-md-offset-2 col-md-10">
                            <div class="checkbox">
                                <asp:CheckBox runat="server" ID="RememberMe" />
                                <asp:Label runat="server"
AssociatedControlID="RememberMe">Zapamiętaj mnie</asp:Label>
                            </div>
                        </div>
                    </div>
                    <div class="form-group">
                        <div class="col-md-offset-2 col-md-10">
                            <asp:Button runat="server" OnClick="LogIn"
Text="Zaloguj się" CssClass="btn btn-default" />
                        </div>
                    </div>
                </div>
            </section>
        </div>
    </div>
</asp:Content>
```

Dane logowania administratora są sprawdzane za pomocą pojedynczej metody - listing 6-20 – zapisanej w pliku **AdminLogin.aspx.cs**.

Listing 6-20
Metoda LogIn

```
protected void LogIn(object sender, EventArgs e)
{
    if (IsValid)
    {
        //Walidacja hasła użytkownika
        //Dodana przestrzeń nazw Microsoft.AspNet.Identity.Owin
        var manager = Context.GetOwinContext()
            .GetUserManager<ApplicationUserManager>();
        var signinManager = Context.GetOwinContext()
            .GetUserManager<ApplicationSignInManager>();
        var result = signinManager.PasswordSignIn(Login.Text,
            Password.Text, RememberMe.Checked, shouldLockout: false);

        //W zależności od wyniku logowania wykonaj jedną z akcji
        switch (result)
        {
            case SignInStatus.Success:
                Response.Redirect("~/Subscriptions.aspx");
                break;
            case SignInStatus.LockedOut:
                Response.Redirect("/Account/Lockout.aspx");
                break;
            case SignInStatus.RequiresVerification:
                Response.Redirect(String.Format(
"/Account/TwoFactorAuthenticationSignIn?ReturnUrl={0}&RememberMe={1}",
                Request.QueryString["ReturnUrl"], RememberMe.Checked), true);
                break;
            default:
                FailureText.Text = "Nieudana próba logowania.
                    Spróbuj ponownie.";
                ErrorMessage.Visible = true;
                break;
        }
    }
}
```

Uwaga
Metoda **LogIn** prezentuje domyślą, dostarczoną przez szablon projektu Visual Studio obsługę konta użytkownika podczas logowania. W wersji z listingu 6-20 metoda została zaadaptowana do logowania administratora serwisu.

Teraz musimy zasilić bazę danych ustawieniami początkowymi, tak aby po uruchomieniu serwisu administrator miał już swoje konto. W tym celu wykorzystamy drugą bazę danych, która została automatycznie stworzona podczas dodawania serwisu na portalu zarządzającym w rozdziale 6.6. Standardowo jej nazwa powinna pokrywać się z nazwą naszego serwisu zakończoną przyrostkiem **_db**. Analogicznie jak poprzednio aby móc korzystać z tej bazy danych musimy dodać odpowiedni wpis w pliku konfiguracyjnym **Web.config**. Potrzebny wpis przedstawia listing 6-21. Należy pamiętać aby uzupełnić wpis o odpowiednie hasło do bazy. Dwie osobne bazy danych, jedna przechowujące dane użytkowników, a druga dla danych aplikacji pozwolą na wygodniejsze zarządzanie i testowanie serwisu.

Listing 6-21
Konfiguracja bazy danych użytkowników

```
<add name="admindatabaseConnectionString"
connectionString="Server=tcp:hgpi04lvu0.database.windows.net,1433;Database=news
leterexample_db;User ID=azure@hgpi04lvu0;Password=
super_Tajne;Trusted_Connection=False;Encrypt=True;Connection Timeout=30;"
providerName="System.Data.SqlClient" />
```

Kolejnym krokiem jest zasilenie bazy danymi logowania administratora, tak aby mógł korzystać z serwisu zaraz po jego uruchomieniu. W tym celu w katalogu **App_Start** dodajemy do pliku **IdentityConfig** nową klasę, którą przedstawia listing 6-22.

Listing 6-22
Zasilenie bazy danych użytkowników

```
//Dodano przestrzeń nazw System.Data.Entity
public class MyDbInitializer :
DropCreateDatabaseIfModelChanges<ApplicationDbContext>
{
    protected override void Seed(ApplicationDbContext context)
    {
        InitializeIdentityForEF(context);
        base.Seed(context);
    }

    private void InitializeIdentityForEF(ApplicationDbContext context)
    {
        var userManager = new UserManager<ApplicationUser>(
            new UserStore<ApplicationUser>(context));
        var roleManager = new RoleManager<IdentityRole>(
            new RoleStore<IdentityRole>(context));

        //Utwórz rolę administratora jeśli nie istnieje
        if (!roleManager.RoleExists("administrator"))
        {
            roleManager.Create(new IdentityRole("administrator"));
        }

        //Utwórz nowego użytkownika
        var user = new ApplicationUser {UserName = "Admin"};
        IdentityResult result = userManager.Create(user, "P4$$word");

        //Dodaj stworzonego użytkownika do roli administratora
        if (result.Succeeded)
        {
            userManager.AddToRole(user.Id, "administrator");
        }
    }
```

```
}
```

Wskazówka
W środowisku produkcyjnym należy login i hasło administratora pobierać z pliku konfiguracyjnego **Web.config** zamiast umieszczać bezpośrednio w kodzie aplikacji.

Klasa ta dziedzicząc po klasie **DropCreateDatabaseIfModelChanges** inicjalizuje bazę danych przy pierwszym uruchomieniu serwisu. Tworzy w niej strukturę tabel odpowiednią dla frameworku *Identity*. Operacja ta nie będzie powtarzana jeśli nie zmienimy modelu danych. Sama klasa udostępnia metodę **Seed**, odpowiedzialną właśnie za wprowadzanie danych początkowych. W niej właśnie tworzymy nową rolę administratora, a następnie nowego użytkownika. Na końcu dodajemy stworzonego użytkownika do roli administratora i w ten prosty sposób już przy pierwszym uruchomieniu serwisu administrator będzie mógł zalogować się do serwisu. Ostatnim elementem jest wymuszenie logowania przy dostępie do stron budowanego serwisu. Ma to uniemożliwić użytkownikom anonimowym korzystanie z jego usług. W tym celu musimy dodać odpowiednie wpisy w **Web.config** - listing 6-24. W pierwszym wpisie blokujemy dostęp do zasobów serwisu użytkownikom anonimowym, a następnie zezwalamy na dostęp do określonych strony tylko użytkownikom o roli administratora.

Dodatkowo należy wskazać jeszcze bazę danych, która będzie automatycznie wypełniania danymi administratora. Aby tego dokonać należy otworzyć plik **Models\IdentityModels.cs** i w klasie **ApplicationDbContext** zmienić wpis **DefaultConnection** na dodany wcześniej **admindatabaseConnectionString**. W ten sposób zasilimy danymi prawidłową bazę danych.

Wymagana jest jeszcze modyfikacja kodu w pliku **Global.asax.cs**, tak aby zainicjować bazę danych z danymi logowania dla administratora. Wynikowa postać pliku jest przedstawiona na listingu 6-23.

Listing 6-23
Metoda Application_Start z pliku Global.asax.cs

```
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    //Dodano przestrzeń nazw System.Data.Entity
    Database.SetInitializer(new MyDbInitializer());
}
```

Listing 6-24
Konfiguracja dostępu do zasobów serwisu

```
<configuration>
…
<!--
Blokujemy dostęp użytkownikom anonimowym
-->
<location path="Default.aspx">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</location>
<!--
Następnie zezwalamy administratorom na dostęp do wybranych stron
-->
<location path="Subscriptions.aspx">
  <system.web>
    <authorization>
```

```
      <allow roles="administrator" />
    </authorization>
  </system.web>
</location>
<location path="NewsletterTemplete.aspx">
  <system.web>
    <authorization>
      <allow roles="administrator" />
    </authorization>
  </system.web>
</location>
…
</configuration>
```

6.11 Przygotowanie strony wzorcowej

Ostatnim elementem jaki musimy skonfigurować jest strona wzorcowa. Strona wzorcowa pełni rolę szablonu dla wspólnych elementów stron serwisu *Newsletter*. Często używana jest do dodawania nagłówka i stopki serwisu gdyż te elementy najczęściej są identyczne na każdej ze stron aplikacji webowej. Powinna ona także zawierać odnośniki do stron, które wcześniej dodaliśmy. W stopce strony dodamy też informację o dacie aktualizacji wyświetlanej strony. Widok strony wzorcowej przedstawia listing 6-25. Strona składa się ze standardowych kontrolek ASP.NET, dzięki którym możemy w prosty sposób skonfigurować nagłówek, menu i stopkę strony.

Listing 6-25
Widok strony wzorcowej Site.Master

```
<%@ Master Language="C#" AutoEventWireup="true" CodeBehind="Site.master.cs"
Inherits="SerwisNewsletter.SiteMaster" %>

<!DOCTYPE html>
<html lang="en">
<head runat="server">
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title><%: Page.Title %> - Usługi Microsoft Azure</title>
    <asp:PlaceHolder runat="server">
        <%: Scripts.Render("~/bundles/modernizr") %>
    </asp:PlaceHolder>
    <webopt:BundleReference runat="server" Path="~/Content/css" />
    <link href="~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
</head>
<body>
    <form runat="server" class="form-horizontal" role="form">
        <asp:ScriptManager runat="server">
            <Scripts>
                <asp:ScriptReference Name="MsAjaxBundle" />
                <asp:ScriptReference Name="jquery" />
                <asp:ScriptReference Name="bootstrap" />
                <asp:ScriptReference Name="respond" />
                <asp:ScriptReference Name="WebForms.js" Assembly="System.Web"
Path="~/Scripts/WebForms/WebForms.js" />
                <asp:ScriptReference Name="WebUIValidation.js"
Assembly="System.Web" Path="~/Scripts/WebForms/WebUIValidation.js" />
                <asp:ScriptReference Name="MenuStandards.js"
Assembly="System.Web" Path="~/Scripts/WebForms/MenuStandards.js" />
                <asp:ScriptReference Name="GridView.js" Assembly="System.Web"
Path="~/Scripts/WebForms/GridView.js" />
                <asp:ScriptReference Name="DetailsView.js"
Assembly="System.Web" Path="~/Scripts/WebForms/DetailsView.js" />
                <asp:ScriptReference Name="TreeView.js" Assembly="System.Web"
Path="~/Scripts/WebForms/TreeView.js" />
                <asp:ScriptReference Name="WebParts.js" Assembly="System.Web"
Path="~/Scripts/WebForms/WebParts.js" />
                <asp:ScriptReference Name="Focus.js" Assembly="System.Web"
```

```
Path="~/Scripts/WebForms/Focus.js" />
                <asp:ScriptReference Name="WebFormsBundle" />
            </Scripts>
        </asp:ScriptManager>

        <div class="navbar navbar-inverse navbar-fixed-top">
            <div class="container">
                <div class="navbar-header">
                    <button type="button" class="navbar-toggle" data-
toggle="collapse" data-target=".navbar-collapse">
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                        <span class="icon-bar"></span>
                    </button>
                </div>
                <div class="navbar-collapse collapse">
                    <ul class="nav navbar-nav">
                        <li><a runat="server" id="HomeLink" href="~/">Strona
główna</a></li>
                        <li><a runat="server" id="SubscriptionsLink"
href="~/Subscriptions.aspx" visible="False">Lista subskrypcji</a></li>
                        <li><a runat="server" id="NewsletterTempleteLink"
href="~/NewsletterTemplete.aspx" visible="False">Szablon newslettera</a></li>
                        <li><a runat="server"
href="~/About.aspx">Informacje</a></li>
                        <li><a runat="server"
href="~/Contact.aspx">Kontakt</a></li>
                    </ul>
                    <asp:LoginView runat="server" ViewStateMode="Disabled">
                        <AnonymousTemplate>
                            <ul class="nav navbar-nav navbar-right">
                                <li><a id="LoginLink" runat="server"
href="~/Account/Login.aspx">Zaloguj się</a></li>
                                <li><a id="AdminLink" runat="server"
href="~/Account/AdminLogin.aspx">Admin</a></li>
                            </ul>
                        </AnonymousTemplate>
                        <LoggedInTemplate>
                            <ul class="nav navbar-nav navbar-right">
                                <li><a runat="server"
href="~/Account/Manage.aspx" title="Ustawienia konta">Witaj, <%:
Context.User.Identity.GetUserName()  %> !</a></li>
                                <li>
                                    <asp:LoginStatus runat="server"
LogoutAction="Redirect" LogoutText="Wyloguj" LogoutPageUrl="~/Default.aspx"
OnLoggingOut="Unnamed_LoggingOut" />
                                </li>
                            </ul>
                        </LoggedInTemplate>
                    </asp:LoginView>
                </div>
            </div>
        </div>
        <div class="container body-content">
            <asp:ContentPlaceHolder ID="MainContent" runat="server">
            </asp:ContentPlaceHolder>
            <hr />
            <footer>
                <p>
                    &copy; Usługi Microsoft Azure<br />
                    Aktualizacja:
                    <asp:Label ID="Updated" runat="server" Text="" /><br />
                    Adres IP:
                    <asp:Label ID="IP" runat="server" Text="" />
                </p>
            </footer>
        </div>
    </form>
</body>
</html>
```

Jeśli zaś chodzi o modyfikację pliku **Site.Master.cs** to dotyczą one jedynie pozycji menu, które wyświetlają się w zależności od typu użytkownika: anonimowego, zwykłego lub administratora oraz informacji o dacie aktualizacji strony. Aby dostosować wygląd menu do typu użytkownika musimy na końcu metody **Page_Init** dodać kod przedstawiony na listingu 6-26. Z kolei datę aktualizacji strony wyznaczymy na podstawie daty ostatniej modyfikacji pliku aspx strony głównej serwisu, co przedstawia metoda **Page_Load** z listingu 6-27.

Listing 6-26
Dynamiczne wyświetlanie zakładek w metodzie Page_Init

```
if (Page.User.IsInRole("administrator"))
{
    HomeLink.Visible = false;
    SubscriptionsLink.Visible = true;
    NewsletterTempleteLink.Visible = true;
}
```

Listing 6-27
Metoda Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{
    //Dodano przestrzeń nazw System.IO
    Updated.Text = File.GetLastWriteTime(
        Server.MapPath("~/Default.aspx")).ToString();
    IP.Text = Request.UserHostAddress;
}
```

Po tych modyfikacjach oraz ponownym wdrożeniu zmian w chmure Azure główna strona serwisu *Newsletter* powinna wyglądać jak na rysunku 6-15.

Uwaga
Publikując końcową wersję witryny w chmurze należy odznaczyć pola wyboru przy opcjach **Use this connection string at runtime** w sekcji **Settings**. Dzięki temu dane połączenia do baz danych zostaną pobrane bezpośrednio z pliku konfiguracyjnego **Web.config**.

Rysunek 6-15
Strona domowa serwisu Newsletter

6.12 Podsumowanie

Przykład opisany w tym rozdziale miał na celu praktyczne wprowadzenie do zagadnień tworzenia aplikacji webowych z wykorzystaniem …