Platforma .NET – Wykład 11
Technologia ASP.NET 2/2

Osoba prowadząca wykład, laboratorium i projekt:
dr hab. inż. Marek Sawerwain, prof. UZ

Instytut Sterowania i Systemów Informatycznych
Uniwersytet Zielonogórski

e-mail : M.Sawerwain@issi.uz.zgora.pl
tel. (praca) : 68 328 2321,
pok. 328a A-2,
ul. Prof. Z.Szafrana 2,
65-246 Zielona Góra

Ostatnia kompilacja pliku: Monday 5th June, 2023, t: 22:54

Notatki

Spis treści

Notatki

Plan wykładu – spotkania tydzień po tygodniu

(1) Informacje o wykładzie, pojęcie platformy, podstawowe informacje o platformie .NET
(2) Składowe platformy .NET: CLR, CTS, języki programowania, biblioteki klas, pojęcie podzespołu (ang. assembly)
(3) Programowanie w C# – środowisko VS, MonoDevelop, syntaktyka C#, wyjątki, współpraca z DLL
(4) Programowanie w C# – model obiektowy, typy uogólnione, lambda wyrażenia
(5) Programowanie w C# – aplikacje „okienkowe", programowanie wielowątkowe
(6) Programowanie w F# – podstawy, przetwarzanie danych tekstowych,
(*) "Klasówka I", czyli egzamin cześć pierwsza
(7) Dostęp do baz danych

Notatki

Plan wykładu – tydzień po tygodniu

(8) Język zapytań LINQ, Entity Framework
(9) Obsługa standardu XML
(10) Technologia ASP.NET 1/2
(11) Technologia ASP.NET 2/2
(12) Model widok i kontroler – Model View Controller
(13) Tworzenie usług sieciowych SOAP i WCF (komunikacja sieciowa)
(14) Wykład monograficzny .NET 1
(15) Wykład monograficzny .NET 2
(*) "Klasówka II", czyli egzamin cześć druga

Notatki

Plan wykładu

1. Kontrolki poprawności,
   1.1 rodzaje kontrolek poprawności,
   1.2 kontrolka podsumowania,
   1.3 własne funkcje weryfikujące.
2. Wygląd aplikacji,
   2.1 tematy,
   2.2 „skórki" w tematach,
   2.3 strona główna (master page).
3. Zarządzanie stanem aplikacji,
   3.1 stan kontrolki po stronie serwera,
   3.2 ciasteczka po stronie klienta,
   3.3 mechanizm „post-back".

Notatki

Validation Controls – kontrolki poprawności

Validation Controls, czyli kontrolki poprawności są odpowiedzialne za sprawdzenie poprawności danych wejściowych wprowadzanych przez użytkownika na formularzu aplikacji WEB. Podstawowe typy kontrolek to min.:

- **CompareValidator** – kontrolka porównująca wartości dwóch pól tekstowych,
- **CustomValidator** – kontrolka gdzie sprawdzanie poprawności jest realizowane przez własny kod programisty,
- **DynamicValidator** – kontrolka współpracuje z wyjątkami zgłaszanymi przez modele danych oraz metody rozszerzające,
- **RangeValidator** – kontrolka sprawdzająca czy dane znajdują się w określonym przedziale wartości,
- **RequiredFieldValidator** – kontrolka sprawdzająca czy w danym polu została wprowadzona „jakakolwiek" wartość,
- **RegularExpressionValidator** – sprawdzanie poprawności za pomocą wyrażenia regularnego,
- **ValidationSummary** – podsumowanie wszystkich błędów w danych jakie wprowadzono na stronie.

Notatki

Dodanie kontrolki sprawdzającej

Kolejne kroki to min.: dodatnie kontrolki sprawdzającej, wybór kontrolki która będzie weryfikowana, ustawienie parametrów kontroli. Dodanie kontrolki typu **TextBox**:

```
<asp:TextBox id="TextBoxCTRL" runat="server" />
```

Kontrolka sprawdzająca poprawność:

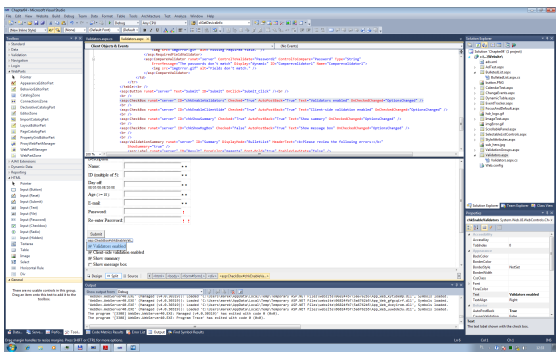
```
<asp:ValidatorCTRLType
    id="ValidatorID"
    runat="server"
    ControlToValidate="TextBoxCTRL"
    ErrorMessage="wiadomość o błędzie"
    Display="static|dynamic|none"
    Text="Text_to_display_by_input_control">
</asp:ValidatorCTRLType
```

Własność **Display** pozwala określić miejsce wyświetlania komunikatów generowanych przez kontrolkę poprawności:

- **Static** - do strony zostanie dodane miejsce przeznaczone na wyświetlanie informacji,
- **Dynamic** - miejsce na wyświetlanie informacji kontrolki zostanie dodane dynamicznie, jeśli dane okażą się niepoprawne,

Notatki

Przykład weryfikatorów



Notatki

Przykład weryfikatorów

Pole imię i nazwisko:

```
<asp:TextBox runat="server" Width="200px" ID="Name" />

<asp:RequiredFieldValidator ID="ValidateName" ControlToValidate="Name"
          ErrorMessage="Pole imię i nazwisko jest potrzebne."
                    Display="dynamic"  runat="server" >*
</asp:RequiredFieldValidator>

<asp:RegularExpressionValidator ID="ValidateName2" ControlToValidate="Name"
      validationExpression="[a-z A-Z]*" ErrorMessage="Name cannot contain digits"
                Display="dynamic" runat="server" >*
</asp:RegularExpressionValidator>
```

Pole ID podzielne przez pięć:

```
<asp:TextBox runat="server" Width="200px" ID="EmpID" />

<asp:RequiredFieldValidator  ID="ValidateEmpID" ControlToValidate="EmpID"
          ErrorMessage="ID is required" Display="dynamic" runat="server" >*
</asp:RequiredFieldValidator>

<asp:CustomValidator ID="ValidateEmpID2" ControlToValidate="EmpID"
          ClientValidationFunction="EmpIDClientValidate" runat="server"
ErrorMessage="ID must be a multiple of 5"
        Display="dynamic" OnServerValidate="ValidateEmpID2_ServerValidate">*
</asp:CustomValidator>
```


Notatki

Przykład weryfikatorów

Funkcja po stronie klienta weryfikującą poprawność ID:

```
<head runat="server">
<script type="text/javascript">
function EmpIDClientValidate(ctl, args) {
    args.IsValid=(args.Value % 5 == 0);
}
</script>
</head>
```

Funkcja weryfikacji ID po stronie serwera:

```
protected void ValidateEmpID2_ServerValidate(object source, ServerValidateEventArgs args) {
    try { args.IsValid = (int.Parse(args.Value) % 5 == 0); }
    catch { args.IsValid = false; }
}
```

Przycisk „Submit":

```
<asp:Button runat="server" Text="Submit" ID="Submit" OnClick="Submit_Click" />
```


Obsługa przycisku „Submit":

Notatki

Przykłady z kontrolką CompareValidator

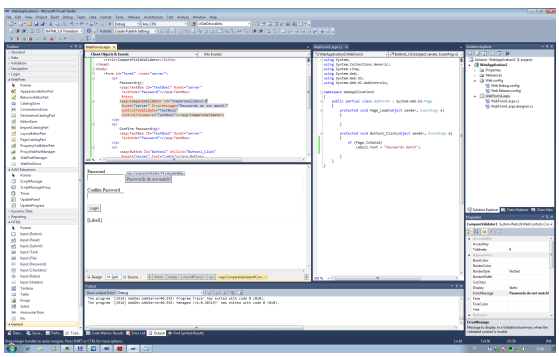
Podanie hasła:

```
<form id="Form1" runat="server">
<p>Password<br>
  <asp:TextBox ID="TextBox1" Runat="server"
          TextMode="Password"></asp:TextBox>  
  <asp:CompareValidator ID="CompareValidator1"
        Runat="server" ErrorMessage="Passwords do not match!"
        ControlToValidate="TextBox2"  ControlToCompare="TextBox1">
  </asp:CompareValidator> </p>
<p>Confirm Password<br>
  <asp:TextBox ID="TextBox2" Runat="server"
          TextMode="Password"></asp:TextBox> </p>
<p><asp:Button ID="Button1" OnClick="Button1_Click"
          Runat="server" Text="Login"></asp:Button></p>
<p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
```

Poprawność strony oznacza poprawność hasła:

```
protected void Button1_Click(Object sender, EventArgs e) {
  if (Page.IsValid) Label1.Text = "Passwords match";
}
```


Notatki



Notatki

Przykłady z kontrolką CompareValidator

Sprawdzanie czy podano wartość liczbową:

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

Własność **Type** pozwala na określenie typu danych jaki będzie stosowany w porównaniach. Możliwe wartości są następujące:

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

Notatki

Przykłady z kontrolką CompareValidator

Sprawdzanie czy podano wartość większą niż określona stała:

```
Wiek:
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
 
<asp:CompareValidator ID="CompareValidator1" runat="server"
  Operator="GreaterThan" ValueToCompare="18"
  ControlToValidate="TextBox1"
  Text="Należy mieć skończone 18 lat" Type="Integer">
</asp:CompareValidator>
```

Własność **Operator** określa rodzaj porównania:

- Equal
- NotEqual
- GreaterThan
- GreaterThanEqual
- LessThan
- LessThanEqual
- DataTypeCheck

Notatki

Własna kontrolka poprawności

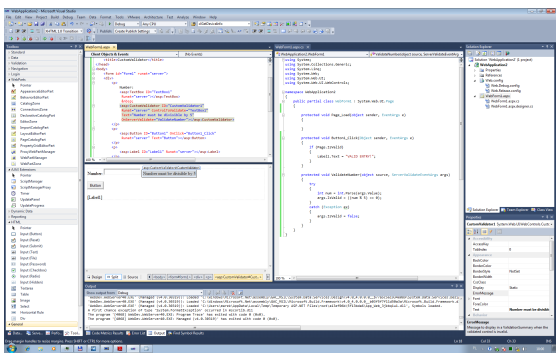
Liczba ma być podzielna przez pięć:

```
<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="Button"></asp:Button>
  </p>
  <p> <asp:Label ID="Label1" Runat="server"></asp:Label> </p>
</div>
</form>
```

Metoda sprawdzająca, czy liczba jest podzielna przez pięć:

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


Notatki



Notatki

Kontrolka ValidationSummary

Kontrolka wyświetla listę komunikatów o błędach, zgłoszonych przez znajdujące się na stronie kontrolki sprawdzające poprawność danych:

- wyświetla zawartość atrybutów **Text** oraz **ErrorMessages**.
- atrybut `Text="*"` jest stosowany do wskazania lokalizacji błędu

Kontrolka podsumowania poprawności danych **Summary** oraz etykieta **Result**:

```
<asp:ValidationSummary runat="server" ID="Summary" DisplayMode="BulletList"
            HeaderText="<b>Please review the following errors:</b>"
                                                ShowSummary="true" />

<asp:Label runat="server" ID="Result" ForeColor="magenta" Font-Bold="true"
            EnableViewState="False" />
```

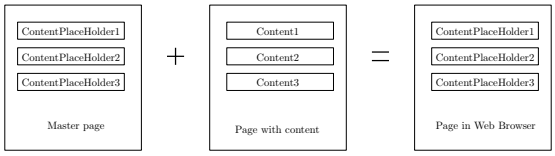
Najważniejsze atrybuty kontrolki podsumowania:

- DisplayMode –– format wyświetlania podsumowania, List – jako listę w osobnych wierszach, BulletList – jako listę wypunktowaną, SingleParagraph – w jednym paragrafie,
- EnableClientScript - włączenie/wyłączenie kodu generowanego po stronie klienta z podsumowania błędów sprawdzania poprawności,
- ShowSummary – gdy przyjmuje wartość **true**, to podsumowanie zostanie wyświetlone na formularzu aplikacji sieciowej.

Notatki

Idea strony wzorcowej

Strona wzorcowa albo stron główna - pozwala na łatwe tworzenie wspólnego wyglądu dla całej witryny internetowej, bowiem zawiera tylko wspólne elementy dla całego serwisu, element szczególne są umieszczane na tzw. stronach z zawartością określone przez kontrolkę **ContentPlaceHolder**:



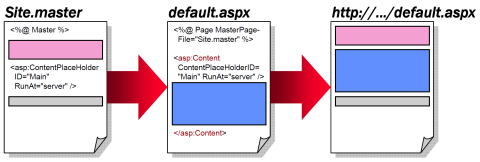
Plik .master zawiera szablon wykorzystywany przez wszystkie strony aplikacji natomiast inne strony są reprezentowane przez kontrolki umieszczone na stronie „Master Page".

Notatki

Strona wzorcowa

Utworzenie strony głównej można zrealizować na następujące trzy sposoby:

- poprzez proste dodawanie strony wzorcowej do projektu –– za pomocą opcji Add New Item" z menu „Website", wybierając szablon „Master Page",
- poprzez dodawanie strony zawartości i zaznaczenie opcji „Select master page". Nowa strona zawartości korzystająca ze strony wzorcowej musi zawierać kontrolkę **Content**, której atrybut **ContentPlaceHolderID** jest równy wartości **ContentPlaceHolderID** strony wzorcowej,
- poprzez modyfikowanie istniejącej strony aplikacji WEB do postaci strony używającej strony wzorcowej.



Notatki

Przykład strony Master Page

Definicja strony „master page":

```
<%@ Master %>
<html>
<body>
  <table width="100%">
    <tr>
      <td bgcolor="darkblue" align="center">
        <span style="font-size: 36pt; color: white">ACME Enterprise Inc.</span>
      </td>
    </tr>
  </table>
  <asp:ContentPlaceHolder ID="Main" RunAt="server" />
</body>
</html>
```

Strona z zawartością:

```
<%@ Page MasterPageFile="~/Site.master" %>
  <asp:Content ContentPlaceHolderID="Main" RunAt="server">
    This content fills the place holder "Main" defined in the master page
  </asp:Content>
```


Notatki

Tematy w aplikacjach WEB

Zadaniem techniki tematów jest umożliwienie łatwego ujednolicenia wyglądu aplikacji WEB, w sposób podobny do tematów aplikacji w ramach systemu Windows. Tematy naturalnie są podobne do technologii CSS, jednakże istnieją pewne różnice:

- tematy mogą określać wiele dodatkowych własności kontrolki bądź strony, nie tylko własności bezpośrednio związane ze stylem, np.: stosując tematy mogą określić postać grafiki wyświetlanej w kontrolce **TreeView**, możliwe jest także określenie układu kontrolki **GridView**,
- tematy mogą zawierać grafikę,
- temat nie są kaskadowe jak arkusze stylów, domyślnie własności określone w temacie odnoszą się do strony w której to we własności Theme zastosowano określony temat, zastosowanie stylu kaskadowego jest możliwe przez użycie StyleSheetTheme,
- tylko jeden temat może zostać zastosowany do określonej strony, nie można stosować wielu tematów na jednej stronie.

Elementy tematów

Główne elementy tematów to min.: skórki, kaskadowe arkusze stylów (cascading style sheets – CSS), obrazy oraz inne zasoby. Najmniejszym elementem, który jest zawarty w temacie to „skórka". Pliki opisujące tematy są zapisywane w specjalnym katalogu po stronie serwera aplikacji.

Notatki

Tematy w aplikacjach WEB

Nakazanie stosowanie istniejącego tematu:

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

Wyłączenie działania tematu:

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

Bezpośrednie określenie parametrów np.: kolorów pola tekstowego:

```
<asp:Textbox ID="TextBox1" runat="server" BackColor="#000000"
                                          ForeColor="#ffffff" />
```

Wyłącznie kontrolki z obszaru działania tematu:

```
<asp:Textbox ID="TextBox1" runat="server" BackColor="#000000"
                   ForeColor="#ffffff" EnableTheming="false" />
```


Włącznie działania tematu jeśli był wyłączony dla wskazanej kontrolki:

Notatki

Jeden temat i różne „skórki"

Tematy mogą posiadać różne „skóry" jej wybór jest wykonywany przez podanie odpowiedniej wartości w atrybucie:

```
<form id="form1" runat="server">

<p><asp:Textbox ID="TextBox1" runat="server">
                                Textbox1</asp:Textbox> </p>

<p><asp:Textbox ID="TextBox2" runat="server"
             SkinId="TextboxDotted">Textbox2</asp:Textbox></p>
<p><asp:Textbox ID="TextBox3" runat="server"
            SkinId="TextboxDashed">Textbox3</asp:Textbox> </p>

</form>
```


Notatki

Temat ze „skórami"

Postać pliku opisującego temat z różnymi skórkami dla kontrolek **Textbox**:

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
          Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
          Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
          BorderColor="#004000" Font-Bold="True" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"
          Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
          BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Arial"
          Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
          BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
          Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
          BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```


Notatki

Zarządzanie stanem

Zarządzanie stanem to zdolność do przechowywania oraz przekazywania informacji np. pomiędzy żądaniami pojedynczego użytkownika aplikacji WEB.
Wyróżnia się cztery główne kryteria mechanizmów przechowywania oraz zarządzania stanem, a są to: zasięg, czas życia, dopuszczalny rozmiar przechowywanych danych oraz po jakiej stronie stan aplikacji jest przechowywany.
W przypadku aplikacji ASP.NET stan może dotyczyć następujących obszarów:

- stan aplikacji (application state) - do przechowywania danych i stosowania w obrębie całej aplikacji (niezależnie dla każdego użytkownika tej aplikacji) są np. dane konfiguracyjne, liczba sesji, i etc,
- stan sesji (session state) – do przechowywania danych i dostępu do danych w obrębie sesji (dla pojedynczego użytkownika) np. stan karty zakupów, e-mail klienta, i etc,
- stan strony (page state) - do przechowywania informacji o strony pomiędzy kolejnymi przetworzeniami strony np. zawartość pól tekstowych TextBoxes, stan CheckBoxe, i etc.

Uwaga

Na aplikację ASP.NET składają się pliki, strony, procedury obsługi zdarzeń, moduły kodu wykonywalnego (programów i bibliotek), kodu wykonywanego lub uruchamianego w obrębie danego katalogu wirtualnego (i jego podkatalogów) na serwerze WWW.

Notatki

Zapamiętywanie stanu kontrolki

Wszystkie kontrolki dziedziczą własność ViewState z klasy bazowej Control. Ze stanem kontrolek związane są dwa podstawowe problemy:

- wysoki czas ładowania: strona musi być serializowana i deserializowana we względu na swój stan oraz wszystkich kontrolek potomnych,
- zwiększenie wielkości strony: ponieważ stan kontrolek jest umieszczany w ukrytym formularzu o nazwie **_VIEWSTATE**.

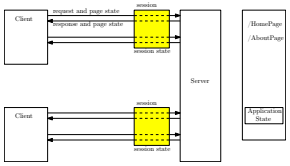
Ukryta kontrolka, o nazwie _ViewState, przechowuje dane pochodzące z kontrolek formularza w postaci zaszyfrowanej. W tym kontekście pojawia się problem z wydajnością, dlatego stosowane jest selektywne włączanie oraz wyłączanie mechanizmu zapamiętywania stanu:

```
<input type="hidden" name="_VIEWSTATE" value="dDwtMTA4MzEOMjEwNTs7Pg==" />
...
<%@ Page EnableViewState="False" %>
<asp:ListBox id="ListName" EnableViewState="true" runat="server">
</asp:ListBox>
```


Notatki

Dwa sposoby zarządzania stanem

Server-Side State Management	Client-Side State Management
Stan aplikacji – informacje są dostępne dla wszystkich użytkowników aplikacji WEB	Cookies – Plik tekstowy zawierający informację do utrzymania poprawnego stanu
Stan sesji – Informacje są dostępne tylko dla użytkownika określonej sesji	Własność ViewState – Utrzymuje wartości pomiędzy wielokrotnymi żądaniami do tej samej strony
Bazy danych – W pewnych przypadkach, można stosować bazę danych do utrzymywania stanu aplikacji WEB	Łańcuchy zapytań – Informacja dodawana na koniec URL



Notatki

Obiekty do zarządzania stanem ASP.NET

Główne pojęcia charakteryzujące stan aplikacji:

- Zasoby – szybki dostęp (stan aplikacji jest przechowywany w pamięci), jednakże przechowywanie dużych bloków danych bezpośrednio w stanie aplikacji może wypełnić pamięć serwera,
- Ulotność – stan aplikacji jest przechowywany w pamięci, ale jest usuwany z niej w momencie zatrzymania lub restartu aplikacji lub w momencie awarii serweram
- Skalowalność – stan aplikacji nie jest dzielony na serwery dostępne w farmie serwerów,
- Współbieżność – do stanu aplikacji może jednocześnie odwoływać się wiele wątków (ważne jest zapewnienie mechanizmów bezpiecznej aktualizacji przechowywanych obiektów).

Dane o stanie aplikacji są przechowywane w obiekcie o klasie **HttpApplicationState**, który jest obiektem typu słownikowego zawierającego pary typu klucz-wartość.

- Zawartość tego obiektu jest dostępna poprzez właściwości **Session** klas **Page** i **HttpContext**,
- Niepowtarzalny identyfikator sesji, służy do identyfikacji kolejnych żądań pochodzących od tego samego klienta.
- Właściwość Session.Mode wskazuje gdzie przechowywane są dane o sesji: InProc, StateServer lub SqlServer.

Uwaga

Plik global.asax umożliwia definiowanie procedur obsługi zdarzeń na poziomie sesji i aplikacji.

Notatki

Stan aplikacji zapisywany po stronie serwera

Stan po stronie serwera: inicjalizacja i uruchomienie aplikacji oraz stan zmiennych sesji są inicjalizowane w pliku Global.asax

- obiekt **Application** współdzieli informacje pomiędzy wszystkimi użytkownikami aplikacji WEB,
- obiekt **Session** zapamiętuje informacje do określonej sesji użytkownika.

```
protected void Application_Start(Object sender,EventArgs e) {
  Application["NumberofVisitors"] = 0;
}
```

Ustalenie sesji oraz zmiennych aplikacji:

```
Session["BackColor"] = "blue";
Application.Lock();
Application["NumberOfVisitors"] = (int)Application["NumberOfVisitors"] + 1;
Application.UnLock();
```

Odczyt sesji oraz zmiennych aplikacji:

```
strBgColor = (string)Session["BackColor"];
lblNbVisitor.Text = Application["NumberOfVisitors"].ToString();
```


Notatki

Stan aplikacji zapisywany po stronie serwera

Server-Side State: Dostęp do informacji o stanie, parametry zmiennych sesji, aplikacji:

- Page state
  - zapis: `ViewState["counter"] = counterVal;`
  - odczyt: `int counterVal = (int) ViewState["counter"];`
- Session state
  - zapis: `Session["cart"] = shoppingCart;`
  - odczyt: `DataTable shoppingCart = (DataTable) Session["cart"];`
- Application state
  - zapis: `Application["database"] = databaseName;`
  - odczyt: `string databaseName = (string) Application["databaseName"];`

Czas działania aplikacji i sesji:

- okres trwania zmiennych sesji po ostatnim dostępie (domyślnie 20 minut),
- czas sesji może zostać zmienione w pliku „web.config",
- zmienne aplikacji istnieją do czasu pojawienia się zdarzenia Application_End.

```
<configuration>
<system.web>
<sessionState timeout="10" />
</system.web>
</configuration>
```


Notatki

Przechowywanie stanu sesji w aplikacji WEB

Dwa podstawowe sposoby:

- InProc
  - Przechowywane w przestrzeni adresowej ASP.NET worker process
  - Szybkie działanie,
  - Dane są tracone gdy proces jest restartowany
- Out-of-Proc
  - State Server
    - Usługa Windows ASP.NET State Service (Instalowana domyślnie, ale nie uruchomiona)
    - Niezależne od IIS – proces aspnet_state.exe
    - Konfiguracja IP i portu `stateConnectionString="tcpip=127.0.0.1:42424"`
  - – SQL Server
    - Dane sesyjne przechowywane w SQL Server
    - Wymaga stworzenia baz danych InstallSqlState.sql / InstallPersistantSqlState.sql
    - Największa niezawodność – możliwość klastrowania

Notatki

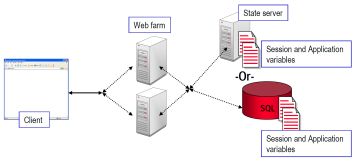
Przechowywanie stanu sesji w urządzeniach masowych

Server-Side State: skalowanie zasobów aplikacji oraz zmiennych sesji, jednakże domyślnie

- stan sesji jest zarządzany przez proces,
- wadą jest brak skalowalności.

Lepszym rozwiązaniem jest

- stan może być zapamiętany przy wykorzystaniu bazy SQL Server oraz serwera stanów,
- zaletą jest przeniesienie stanu z procesu oraz pełna skalowalność.



- określenie gdzie stan ma być zapisywane jest realizowane w pliku **Web.config** (możliwe wartości sqlserver or stateserver),
- jeśli stan ma być zapisywany do bazy danych (SQL server), to postać wpisu do **Web.config** jest następująca:

```
<sessionState mode="SQLServer" sqlConnectionString=
            "data source=SQLServerName; Integrated security=true" />
```


Notatki

Zapisywanie stanu po stronie klienta

Client-Side State: stosowanie „ciasteczek" do zapamiętywania stanu sesji, utworzenie ciasteczka jest realizowane w następujący sposób:

```
HttpCookie objCookie = new HttpCookie("myCookie");
DateTime now = DateTime.Now;

objCookie.Values.Add("Time", now.ToString());
objCookie.Add("ForeColor", "White");
objCookie.Values.Add("BackColor", "Blue");

objCookie.Expires = now.AddHours(1);

Response.Cookies.Add(objCookie);
```

Ciasteczka, zawierają informacje o domenie:

```
Set-Cookie: Username=FName+SName; path=/; domain=microsoft.com;
Expires=Tuesday, 04-Feb-11 00.00.02 GMT
```


Notatki

Odczytanie stanu po stronie klienta

Client-Side State: Odczytanie informacji z ciasteczka, ale w pierwszej kolejności należy utworzyć obiekt ciasteczka:

```
HttpCookie objCookie = Request.Cookies[["myCookie"];
```

Odczytanie informacji z ciasteczka:

```
lblTime.Text = objCookie.Values["Time"];
lblTime.ForeColor = System.Drawing.Color.FromName
                        (objCookie.Values["ForeColor"]);
lblTime.BackColor = System.Drawing.Color.FromName
                        (objCookie.Values["BackColor"]);
```


Notatki

„Sesja bez ciastek"

Każda aktywna sesja jest identyfikowana i śledzona przez ID sesji. ID sesji są przekazywane przez żądania klient/serwera stosując ciasteczka HTTP lub dołączane do URL, w tym ostatnim przypadku można mówić o sesjach bez ciasteczek (cookieless sessions):

- ID sesji jest zakodowane w URL'u:

```
http://localserver/(h44a1e55c0breu552yrecobl)/page.aspx
```

- nie można używać bezwzględnych URL'i
- większość przeglądarek ogranicza wielkość URL do 255 znaków, co naturalnie ogranicza wielkość ID sesji.

Uruchomienie sesji bez ciasteczek:

- stan sesji jest konfigurowant w sekcji <SessionState> pliku Web.config,
- ustawienie atrybutu `cookieless = true`

```
<sessionState cookieless="true" />
```


Notatki

Komunikacja zwrotna – „page postback"

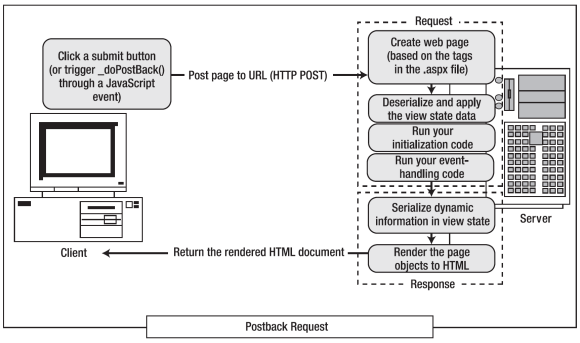
Komunikacja zwrotna albo odsyłanie (ang. postback) polega na powracaniu do określonej postaci strony WWW w czasie trwania sesji z serwerem:

- model ASP.NET Postback – mechanizm do przekazywania własności kontrolek, z przeglądarki do serwera aplikacji, oraz przywracania tych wartości po uzyskaniu odpowiedzi z serwera aplikacji (umożliwia to kontrolkom aplikacji utrzymywanie wartości własności nie jako ponad żądaniami do serwera, i ponad bez stanowością protokołu HTTP),
- własność AutoPostBack – ta własność kontroluje czy interakcja użytkownika powinna powodować wywołanie tzw. „round-trip" względem serwera, niektóre kontrolki WEB oferują wsparcie dla auto-post-back oferując własność AutoPostBack,
- własność EnableViewState – określa, czy dana kontrolka WEB powinna utrzymywać swój stan podczas trwania akcji postback,
- Cross-Page Postbacks – można także określić iż dana kontrolka wysyła żądanie, do innej strony poprzez zmianą wartości własności PostbackUrl.

IsPostBack – właściwość strony umożliwiająca programowe rozróżnienie między żądaniami przesyłanymi zwrotnie, a pierwszymi wywołaniami strony.

Notatki

Komunikacja zwrotna – „page postback"



Notatki

Kontrolki są źródłem efektu „page postback"

Kontrolki które powodują natychmiastowy „round-trip":

- przycisk – zdarzenie Click

```
<asp:Button Text="click me" Runat="server" OnClick="DoClick" />‘
```

Kontrolki o opóźnionej obsłudze, będą przetwarzane w następnym „round-trip":

- pole tekstowe – zdarzenie TextChanged

```
<asp:TextBox Runat="server" OnTextChanged="DoTextChanged" />
```

- lista – zdarzenie SelectedIndexChanged

```
<asp:ListBox Rows="3" Runat="server" OnSelectedIndexChanged="DoSIChanged" />
```

Kontrolki z automatycznym post-back,

- pole tekstowe – zdarzenie TextChanged

```
<asp:TextBox Runat="server" AutoPostBack="true" OnTextChanged="DoTextChanged" />
```


Notatki

Komunikacja zwrotna – „cross-page postback"

Strony mogą wykonać post back do innych stron, wykorzystywane są wtedy następujące właściwości:

- **control.PostBackUrl** – adres docelowy żądania zwrotnego,
- **Page.PreviousPage** – zwraca referencję do strony, która generowała żądanie zwrotne,
- **PreviousPage.IsCrossPagePostBack** – informacja, czy wystąpiło żądanie zwrotne z innej strony.

```
<html>
<body>
  <form runat="server">
    <asp:TextBox ID="Input" RunAt="server" />
    <asp:Button Text="Test" PostBackUrl="PageTwo.aspx" RunAt="server" />
  </form>
</body>
</html>
```


Notatki

W następnym tygodniu między innymi

1. Model View Controller,
2. źródła technologii MVC,
3. pierwsza aplikacja,
4. podstawowe elementy,
5. przekierowania, adresy, kontrolery oraz widoki,
6. AJAX i MVC.

Proponowane tematy prac pisemnych:

1. zarządzanie stanem w dużych aplikacjach ASP.NET,
2. współpraca z bazami danych aplikacji ASP.NET,
3. tworzenie własnych kontrolek WEB,
4. strona graficzna aplikacji ASP.NET, tematy, skórki, strona główna.

Dziękuje za uwagę!!!

Notatki