Platforma .NET – Wykład 12
Model widok i kontroler – Model View Controller

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: 23:35

Spis treści

1. Wprowadzenie
   - Plan wykładu
2. Informacje wstępne
   - Rozwój technologii tworzenia aplikacji WEB
   - Słabości ASP.NET
   - Zalety podejścia MVC
3. Model View Controller
   - Architektura MVC
   - AJAX, JavaScript i MVC
   - Przepływ sterowania
   - Struktura katalogów
   - Strona z pudełka
   - URL i marszruty
   - Model i Widok
   - AJAX, JavaScript i MVC
4. Już za tydzień na wykładzie

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

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

Plan wykładu

1. czym jest architektura Model, View, Controller,
   1. historia rozwoju aplikacji WEB,
   2. Ruby On Rails,
   3. wady i zalety ASP.NET i MVC,
2. architektura MVC
   1. główne elementy,
   2. rola kontrolera,
   3. model i widok,
   4. przepływ sterowania,
3. przykłady prostych aplikacji MVC,
   1. aplikacji z „pudełka”
   2. proste edytory osobowe,
   3. AJAX i strefy czasowe.

Rozwój technologii tworzenia aplikacji WEB

Rozwój technologii tworzenia aplikacji WEB od „Ery Jurajskiej" do „Współczesności":

- Era Jurajska, Common Gateway Interface (CGI), prostota, elastyczność, technologia ta zawsze dostępna w większości serwerów, jedyna opcja w tych czasach, Wady: proces jest uruchamiany poza serwerem WEB, może wymagać dużych nakładów na zasoby (powołuje nowy proces OS na każde żądanie), rozwiązanie nisko-poziomowe,
- Wiek Brązu, Microsoft Internet Database Connector (IDC), działa wewnątrz serwera WWW, Wady: jest to tylko nakładka dla zapytań SQL i szablon do formatowania rezultatów,
- Rok 1996, Active Server Pages (ASP), rozwiązanie ogólnego przeznaczenia, Wady: interpretowany w czasie wykonania, rozwiązanie sprzyja tworzeniu „spaghetti code"
- Lata 2002/2003, ASP.NET Web Forms 1.0/1.1, Zalety: kompilacja, „Statefull", UI, bogata infrastruktura, podejście obiektowe, Wady: duże wymagania co do przepustowości, tworzone pliki HTML są „brzydkie" (po stronie kodu!), niestabilność,
- Lata 2005/2010, ASP.NET Web Forms 2.0, ASP.NET AJAX, ASP.NET Web Forms 3.5, ASP.NET MVC 1.0, 2010 ASP.NET MVC 2.0, ASP.NET Web Forms 4.0.

Słabości ASP.NET

Tradycyjne aplikacje ASP.NET okazały się sporym sukcesem, i obecnie jest to dobrze wspierana i rozwijana technologia, jednakże można wskazać następujące słabe punkty:

(1) problem ViewState:
obecnie dostępny mechanizm utrzymywania stanu pomiędzy żądaniami użytkowników (ViewState) bardzo często doprowadza do konieczności transferu dużych ilości danych pomiędzy klientem a serwerem. W wielu przypadkach, gdy mamy do czynienia z bardziej rozwiniętymi aplikacji, wielkość przesyłanych danych może łatwo przekroczyć 1MB, co w efekcie przy wolniejszych łączach doprowadza do nadmiernego obciążenia serwera oraz wprowadza konieczność oczekiwania na reakcję interfejsu użytkownika. Częściowym rozwiązaniem tych problemów jest stosowanie podejścia AJAX, które oferuje możliwość częściowej aktualizacji strony,

(2) cykl życia strony: mechanizm łącznia zdarzeń generowanych przez klienta z kodem po stronie serwera, który to jest elementem cyklu życia strony, może się łatwo komplikować, szczególnie w przypadkach gdy modyfikowana jest hierarchia kontrolek w czasie wykonania bez zgłaszania błędów „ViewState" lub w momencie szukania powodów błędnej obsługi zdarzeń,

(3) separacja płaszczyzn logiki i prezentacji: model „code-behind" pozwala na łatwe rozdzielenie i separację klas implementującą logikę od warstwy prezentacji, jednakże dość często np.: po zmianie hierarchii drzewa kontrolek serwera, wymagane jest dodanie kodu osadzone w HTML'u. Ze względu na sztywny podział płaszczyzn dość często prowadzi to do otrzymania bardzo nieczytelnego kodu aplikacji.

(4) ograniczona kontrola nad kodem HTML: kontrolki serwera odpowiedzialne za rysowanie formularza/kontrolek do postaci kodu HTML nie są jego standardowym elementem, i dodatkowo nie zawsze jest to stosowana postać HTML'a, w starszych wersjach ASP.NET rezultat nie był zawsze zgodny ze standardami HTML i standardami aplikacji WEB, dodatkowo generowane identyfikatory, utrudniają wykorzystanie JavaScript'u po stronie klienta (część tych problemów zostało rozwiązanych w wersji 4.0 ASP.NET),

Słabości ASP.NET

(5) utrata abstrakcji:
formularze aplikacji WEB zazwyczaj ukrywają aspekty związane z HTML i HTTP, tworzona aplikacja jest bardziej abstrakcyjna. Jednakże przy próbie implementacji własnych zachowań, należy zajmować się wieloma szczegółami np.: mechanizmem post-back , co prowadzi do utraty abstrakcji, co więcej te elementy nie są zbyt dobrze opisane w dokumentacji, również tworzenie bogatych w interaktywne możliwości aplikacji po stronie klienta jest utrudnione, bowiem serwer może nie odzwierciedlać poprawnie całego stanu aplikacji,

(6) trudności w stosowaniu testów zautomatyzowanych: struktura aplikacji ASP.NET utrudnia stosowanie testów jednostkowych, ze względu na „grubość" i jednolitość aplikacji ASP.NET.

Kluczowe zalety podejścia MVC – Architektura MVC

Technologia ASP.NET MVC polepsza separację zadań jakie są realizowane w trakcie budowy aplikacji ASP.NET, dzięki wykorzystaniu MVC. Jednak sama idea MVC, wywodzi się z firmy Xerox, a powstała w roku 1978 i jest silnie związana z językiem Smalltalk. Obecnie podejście MVC zdobyło bardzo dużą popularność w kontekście tworzenia aplikacji WEB, niewątpliwie przyczyniły się do tego dwa następujące fakty:

- interakcje użytkownika z aplikacją MVC tworzą naturalne cykle, bowiem akcje użytkownika powodują odpowiedź aplikacji, co prowadzi do zmian w modelu danych, co skutkuje iż tworzony jest nowy widok przekazywany do użytkownika. Po wykonaniu tego cyklu, aplikacja jest gotowa do następnego cyklu, takie podejście jest bardzo dobrze dopasowane dla aplikacji WEB, bowiem współgra z protokołem HTTP (schemat request oraz response),
- niemal wszystkie aplikacji WEB są tworzone w oparciu o kilka różnych technologii (np.: bazy danych, HTML), zazwyczaj rozdzielona na poszczególne warstwy, takie podejście jest bardzo naturalne dla koncepcji MVC.

Technologia ASP.NET MVC dostarcza nowoczesny wariant MVC, przeznaczony do tworzenia aplikacji WEB. Technologia ta wprowadza rozwiązania dotąd dostępne w pakiecie Ruby On Rails (i podobne technologie i platformy) na poziom platformy .NET, uwypuklając i wprowadzając znane techniki i rozwiązania aplikacji MVC z innych platform na poziom .NET, co powoduje iż programiści MVC dotąd korzystający z innych rozwiązań łatwo rozpoczną pracę w MVC w ramach platformy .NET.

Rozszerzalność i kontrola HTML/HTTP

Rozszerzalność, projektanci MVC założyli, że w każdym dostępnym dla programisty komponencie będą dostępne następujące trzy opcje rozwoju danego komponentu:

- domyślna implementacja danego komponentu może być stosowana w typowych aplikacjach MVC,
- zmiana domyślnego zachowania powinna wymagać tylko utworzenia klasy pochodnej,
- możliwe jest całkowite zastąpienie komponentu nowym, przy wykorzystaniu klasy abstrakcyjnej bądź interfejsu.

Kontrola nad HTML oraz HTTP

Jednym z ważnych założeń ASP.NET MVC jest tworzenie kodu HTML w czystej postaci zgodnej ze obecnie uznanymi standardami. Powstały kod HTML łatwo też dopasować do określonej postaci graficznej dzięki CSS, dzięki temu iż tworzony przez MVC kod HTML jest bardzo czytelny.
Strony generowane przez ASP.NET MVC nie zawierają danych typu **ViewState**, toteż są znacznie mniejsze niż typowe formularze ASP.NET. Dodatkowo mniejsze strony, są zawsze szybciej przetwarzane oraz przesyłane przez sieć, co przyczynia się do dalszej poprawy ogólnej jakości aplikacji WEB.

Analogicznie jak pakiet Ruby on Rails, ASP.NET MVC pracuje bezpośrednio na poziomie HTTP. Co oznacza, iż mamy pełną kontrolę nad żądaniami pomiędzy przeglądarką a serwerem. Umożliwia to lepszą kontrolę nad interfejsem użytkownika. Tym bardziej iż technologia AJAX, nie wprowadza utrudnień przy obsłudze zdarzeń „postback", ze zdarzeniami po stronie klienta.

System marszrut (ang. routing system)

Obecnie, w aplikacjach WEB wymaga się, aby stosowane adresy były czytelne również dla człowieka, toteż zamiast następującego adresu:

```
/App_v2/User/Page.aspx?action=show%20prop&prop_id=82742
```

lepiej byłoby zastosować poniższy URL, który naturalnie jest znacznie bardziej czytelny

```
/to-rent/chicago/2303-silver-street
```

Czyste i eleganckie adresy URL są trudne do uzyskania, szczególnie we wcześniejszych rozwiązaniach, obecnie ASP.NET MVC używa rozwiązania dostępnego w **System.Web.Routing** do otrzymywania czystego adresu URL. W ten sposób można uzyskać pełną kontrolę nad URL i jego odwzorowaniach na kontrolery i akcje, bez konieczności stosowania dodatkowych wspomagających rozwiązań.

Inne ważne zalety to: zdolność do zautomatyzowanych testów, wykorzystanie najlepszych rozwiązań dostępnych w ASP.NET, nowoczesne API, oraz ASP.NET MVC jest rozwiązaniem o otwartym kodzie.

Ruby On Rails

Ruby on Rails (często nazywany RoR lub po prostu Rails – `http://www.rubyonrails.org/`) – to pakiet do szybkiego tworzenia aplikacji WEB. RoR został opracowany w języku Ruby z użyciem architektury MVC (ang. Model-View-Controller).
Główne założenia jakie zostały przyjęte w trakcie tworzenia RoR są następujące:

- szybkość, łatwość oraz co najważniejsze to przyjemność tworzenia kodu,
- reguła DRY (ang. Don't Repeat Yourself), co polega na unikaniu powtarzania tej samej pracy w różnych miejscach aplikacji,
- reguła COC (ang. Convention Over Configuration), polegająca na sprowadzeniu do minimum niezbędnej konfiguracji aplikacji poprzez zastępowanie jej gotowymi i zalecanymi wzorcami,
- możliwość użycia wtyczek, które rozszerzają aplikacje o nowe funkcje, jak np.: logowanie, skalowanie obrazów i etc.

Ruby On Rails

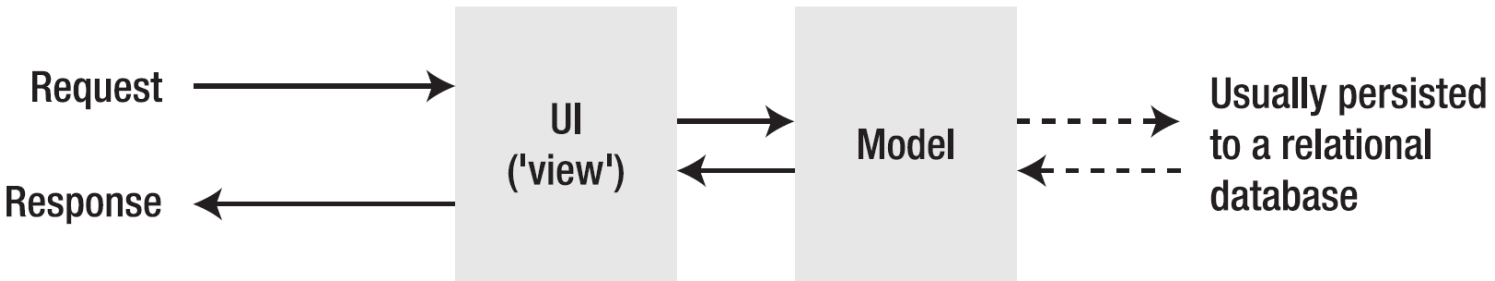
Na framework Rails składają się cztery główne elementy:

- ActiveRecord – mechanizm ORM (Object-Relational mapping) dla Ruby, odpowiada za tworzenie modeli w architekturze MVC,
- ActionPack – biblioteka zawierająca klasy ActionController i ActionView, które odpowiadają za tworzenie odpowiednio kontrolerów i widoków,
- ActiveSupport – zbiór użytecznych dodatków do standardowej biblioteki Ruby, zawiera m.in. rozszerzenia klas String czy Time,
- ActionMailer – biblioteka służąca do wysyłania wiadomości email (w ramach aplikacji Rails).

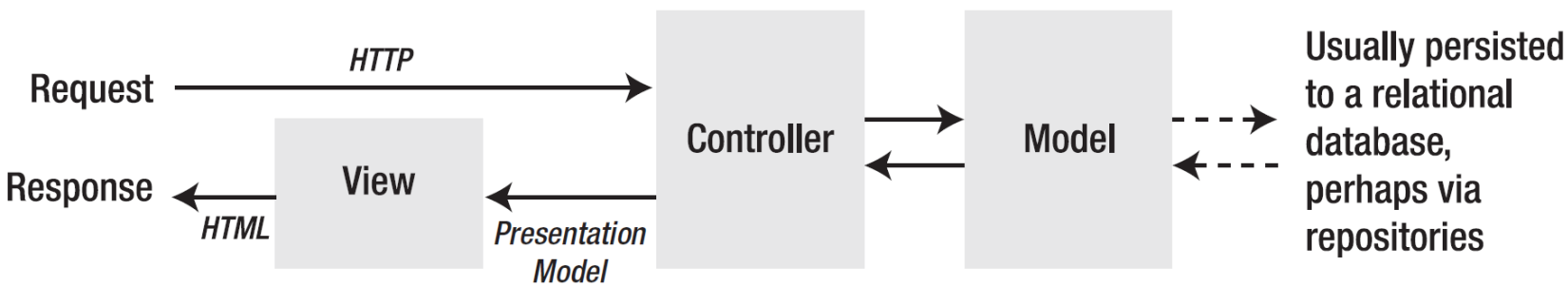
Każdy z głównych elementów może zostać zainstalowana oddzielnie, (za pomocą menadżera pakietów RubyGems). W składzie dystrybucji Rails znajduje się także program **rails** generujący drzewo katalogów nowego projektu.

Architektura MVC

Podejście Model, Widok rozdziela źródło danych od sposobu jego prezentacji:



Dodanie dodatkowego obiektu kontrolera pomiędzy prezentację a źródło danych, w efekcie daje architekturę Model, Widok, Kontroler:



Czym jest ASP.NET MVC

Czym jest ASP.NET MVC w kilku prostych punktach:

- „framework" przeznaczony do budowy aplikacji WEB,
- technologia ta jest oparta o podejście Model-View-Controller,
  - model zarządza danymi aplikacji oraz wymusza ograniczenia dla tego modelu,
  - widok to zazwyczaj pasywna reprezentacja stanu aplikacji (widoki tworzą żądania przesyłana do kontrolera bazując na akcjach klienta aplikacji WEB),
  - kontroler tłumaczy żądania na akcje wykonywane na modelu danych oraz tworzy odpowiedni widok.

Podstawowy cykl przetwarzania informacji w MVC:

- klient żąda wykonania nazwanej akcji na określonym kontrolerze, np.:

```
http://localhost/aController/anAction
```

- żądanie jest kierowane przez metodę anAction do kontrolera aController (metoda zadecyduje jak obsłużyć żądanie, być może wymagać to będzie dostępu do stanu modelu, co w efekcie doprowadzi do przesłania informacji w postaci widoku).

Model i Widok

Opis modeli jest zawarty w plikach zawierających kod w C# i odnosi się do źródła danych np.: bazy SQL bądź też pliku XML,

- plik z kodem C# zarządza dostępem do danych aplikacji poprzez różnego rodzaju obiekty,
- „LINQ to SQL" oraz „LINQ to XML" są wykorzystywane do tworzenia zapytań do źródeł danych,
  - dostęp do danych może być bezpośredni,
  - najczęściej jednak dostęp do danych jest pośredni poprzez dodatkowe obiekty opakowujące bądź dane w plikach XML.

Widok to zazwyczaj pliku aspx zawierające kod HTML oraz kod inline np.: `<% ... C# code here ... %>`:

- kod jest stosowany tylko do wpierania prezentacji i nie jest stosowany po przetwarzania kodu po stronie aplikacji,
- HTML jest wspieranych przez odpowiednie obiekty, ASP.NET MVC dostarcza ułatwienie dla najczęściej stosowanych konstrukcji HTML,
- ASP.NET MVC dostarcza także bibliotekę jQuery dla wpierania akcji klienta oraz komunikacji AJAX z serwerem.

Kontroler

Kontroler jest implementowany jako klasa w C# i bazuje na klasie **Controller**

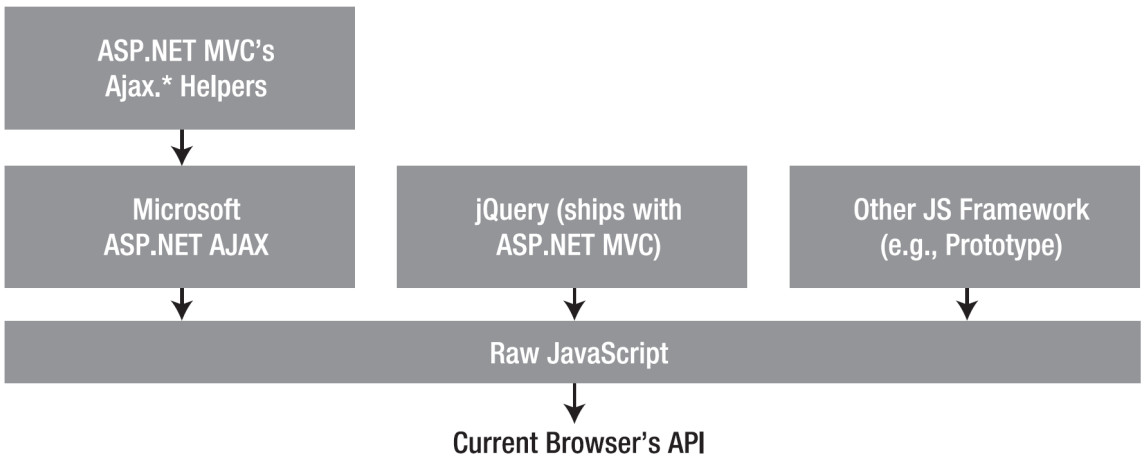
- kontroler określa kategorie przetwarzania w aplikacji,
- metody są odpowiedzialne za detale w przetwarzaniu.
- marszruty do kontrolera są określone w pliku Global.Asax.cs, zazwyczaj domyślne przetwarzanie jest wystarczające w większości aplikacji.

Zadanie kontrolera to zwrócenie rezultatu, można wyróżnić następujące typu rezultatów:

- EmptyResult – pusta odpowiedź (null), nie są podejmowane żadne dodatkowe akcje,
- ContentResult – odpowiedź stanowi zawartość, która zostanie przekazana w sposób bezpośredni,
- JsonResult – obiekt zostanie zapisany (serializowany) do formatu JSON i przekazany jako odpowiedź,
- RedirectResult – przekierowanie użytkownika pod wskazany URL,
- RedirectToRouteResult – przekierowanie użytkownika pod URL określony w parametrach marszrut,
- ViewResult – wywołanie systemy Widoków i utworzenie widoku odpowiedniego dla odpowiedzi,
- PartialViewResult – odpowiedź podobna do ViewResult, przy czym widok jest tworzony częściowo i jest to odpowiedź na żądanie AJAX,
- FileResult – jest to klas bazowa dla rezultatów które mają zwracać dane binarne jako strumień, jest szczególnie przydatna jeśli akcje użytkownika wymagają przesłania pliku,
- FilePathResult – klasa pochodna FileResult, wynikiem jest plik uwzględniający ścieżkę dostępu,
- FileContentResult – klasa pochodna FileResult, wynikiem jest ciąg bajtów stanowiący zawartość pliku,
- FileStreamResult – klasa pochodna FileResult, rezultatem jest strumień,
- JavaScriptResult – rezultat jest stosowany do natychmiastowego wykonania kodu JavaScript przez klienta, a kod jest przesyłany z serwera.

AJAX, JavaScript i MVC

JavaScript po stronie klienta aplikacji WEB pozwala na tworzenie interaktywnych aplikacji, z przetwarzaniem po stronie klienta, wraz z technologią AJAX, obniża konieczność komunikacji z serwerem oraz polepsza jakość UI.

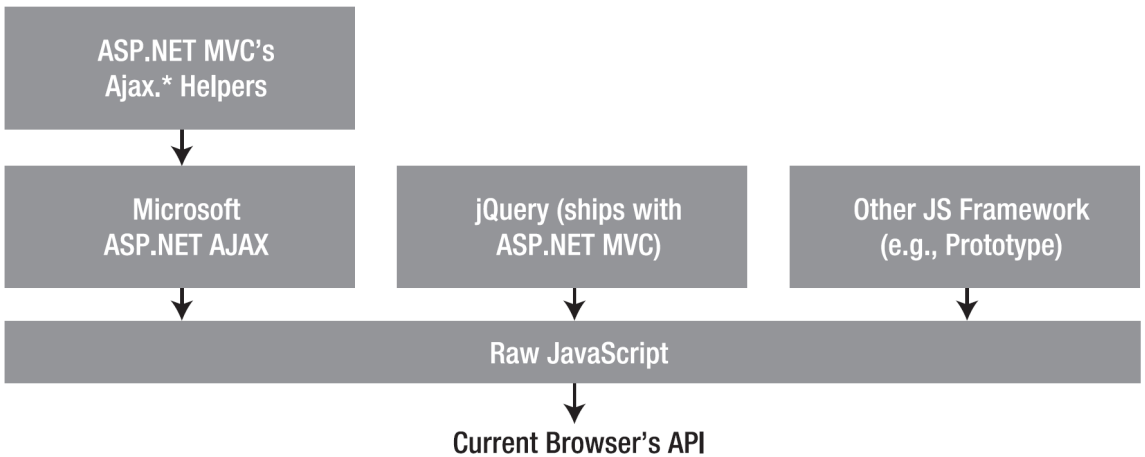


Framework MVC oferuje kilka elementów wspomagających tworzenie asynchronicznych uaktualnień stron:

- Ajax.ActionLink() tworzy znacznik odnośnika, podobnie do Html.ActionLink(), po kliknięciu odbiera i tworzy nową zawartość dla istniejącej strony HTML,
- Ajax.BeginForm() tworzy formularz HTML, podobnie jak Html.BeginForm(), po jego wysłaniu, odbiera i tworzy nową zawartość które jest dodawana do istniejącej strony HTML.

AJAX, JavaScript i MVC

Jak już to wcześniej powiedziano JavaScript, po stronie klienta aplikacji WEB pozwala na tworzenie interaktywnych aplikacji, z przetwarzaniem po stronie klienta. Wraz z technologią AJAX obniża, to konieczność komunikacji z serwerem oraz polepsza jakość UI.

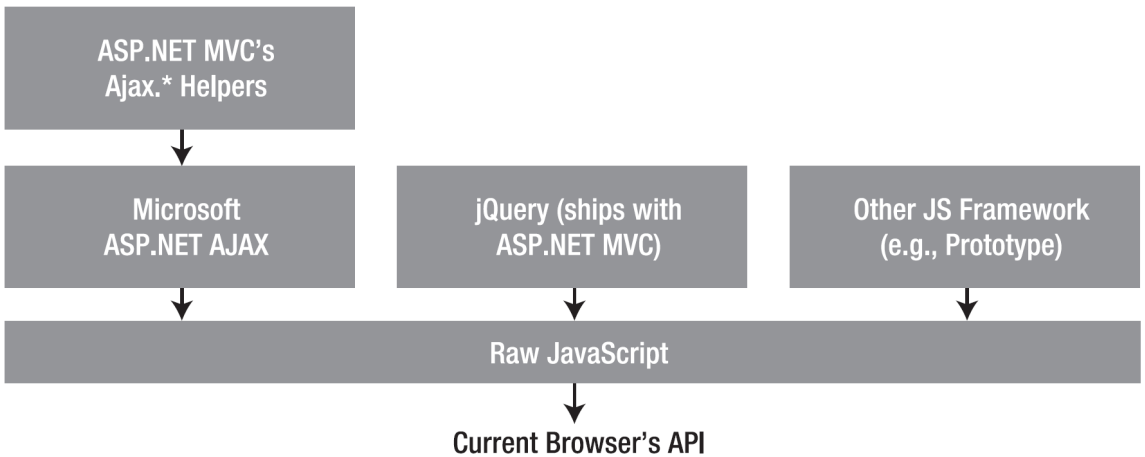


Framework MVC oferuje kilka elementów wspomagających tworzenie asynchronicznych uaktualnień stron:

- Ajax.RouteLink() funkcjonuje podobnie jak Ajax.ActionLink(), jednakże URL jest tworzony na podstawie zbioru parametrów dla marszrut, nie muszą one być związane z konkretną wywoływaną akcją. Stanowi to odpowiednik wyrażenia Html.RouteLink() w technologii AJAX. Polecenie Ajax.RouteLink() jest bardzo użyteczne w bardziej zaawansowanych scenariuszach, gdzie wykorzystuje się własny kontroler oparty o IController, w którym może nie pojawić się metoda związana z akcją.

AJAX, JavaScript i MVC

JavaScript po stronie klienta aplikacji WEB pozwala na tworzenie interaktywnych aplikacji, z przetwarzaniem po stronie klienta, wraz z technologią AJAX obniża, to konieczność komunikacji z serwerem oraz polepsza jakość UI.



Framework MVC oferuje kilka elementów wspomagających tworzenie asynchronicznych uaktualnień stron:

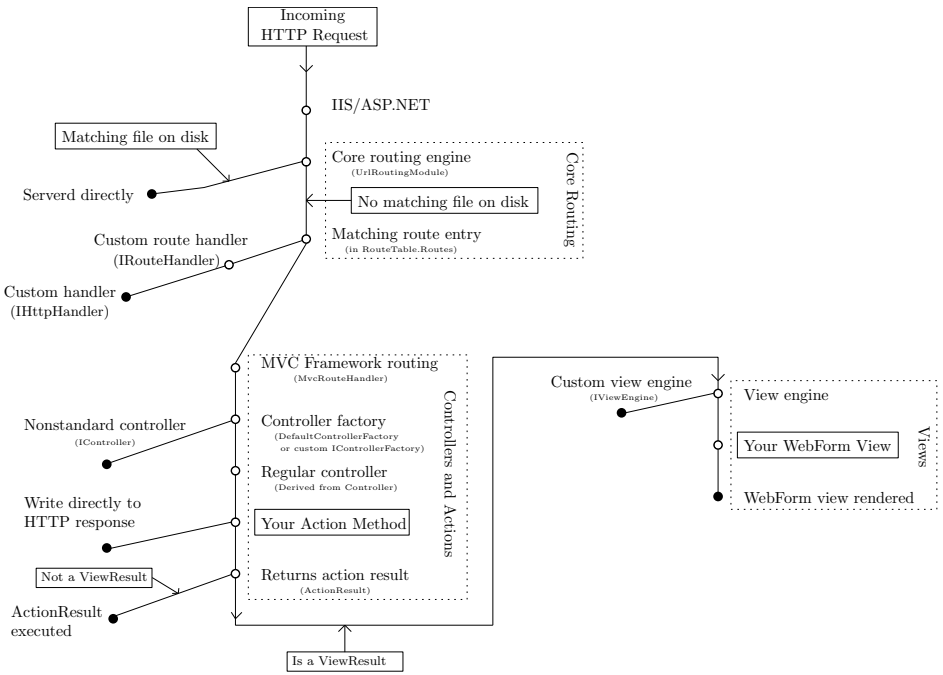
- podobnie jak, Ajax.BeginRouteForm() metoda ta jest podobna do Ajax.BeginForm(), jednakże generowany URL jest oparty na zbiorze parametrów marszrut, nie koniecznie związanych z wywołaną akcją, i stanowi odpowiednik metody Html.BeginRouteForm().

Czy warto angażować się kolejną technologię?

Struktura aplikacji ASP.NET MVC jest bardzo elastyczna min.:

- łatwo dodawać nowe kategorie zachowań do aplikacji poprzez dodawanie nowych kontrolerów,
- obiekty kontrolera wprowadzają porządek w aplikacji,
- łatwo tworzyć nowe widoki, których ilość jest nieograniczona, nawigacja jest prosta i we większości przypadku wystarcza infrastruktura obecna w MVC, np.: reguły marszrut w pliku **Global.asax.cs**,
- również liczba modeli nie jest sztucznie ograniczona, należy tylko dodać nowe klasy i stosować LINQ do realizacji dostępu do danych.

Przepływ sterowania w aplikacji ASP.NET MVC

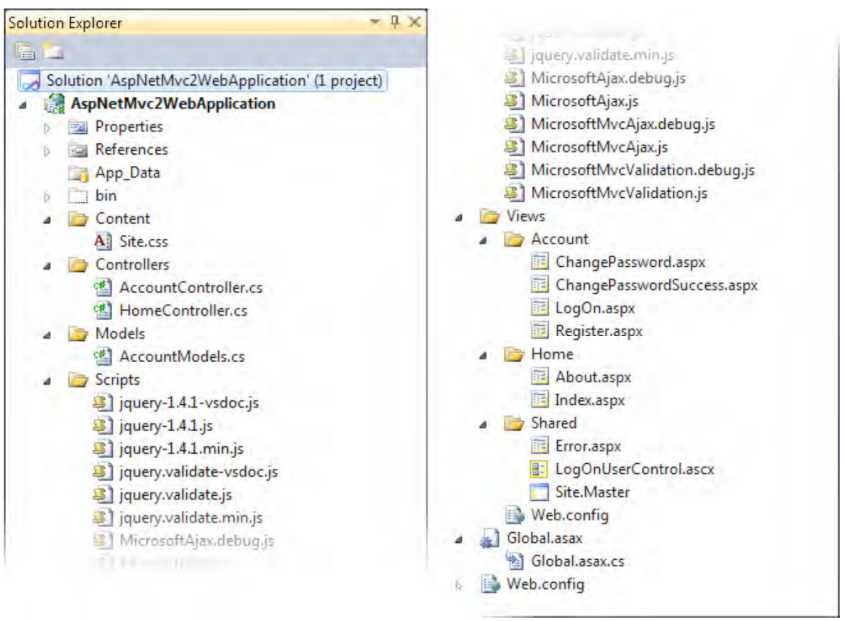


Rysunek pochodzi z książki autorstwa Stevena Sandersona, pt. *Pro ASP.NET MVC 2 Framework*, APress.

Nowy projekt w Visual Studio

Zakładając nowy projekt w Visual Studio należy wybrać nowo dostępy projekt „ASP.NET MVC 2 project" , wpierając opcję File ⇒ New ⇒ Project, otrzymujemy dwie możliwości:

- ASP.NET MVC 2 Web Application,
- ASP.NET MVC 2 Empty Web Application.



Katalogi i ich przeznaczenie w aplikacjach MVC

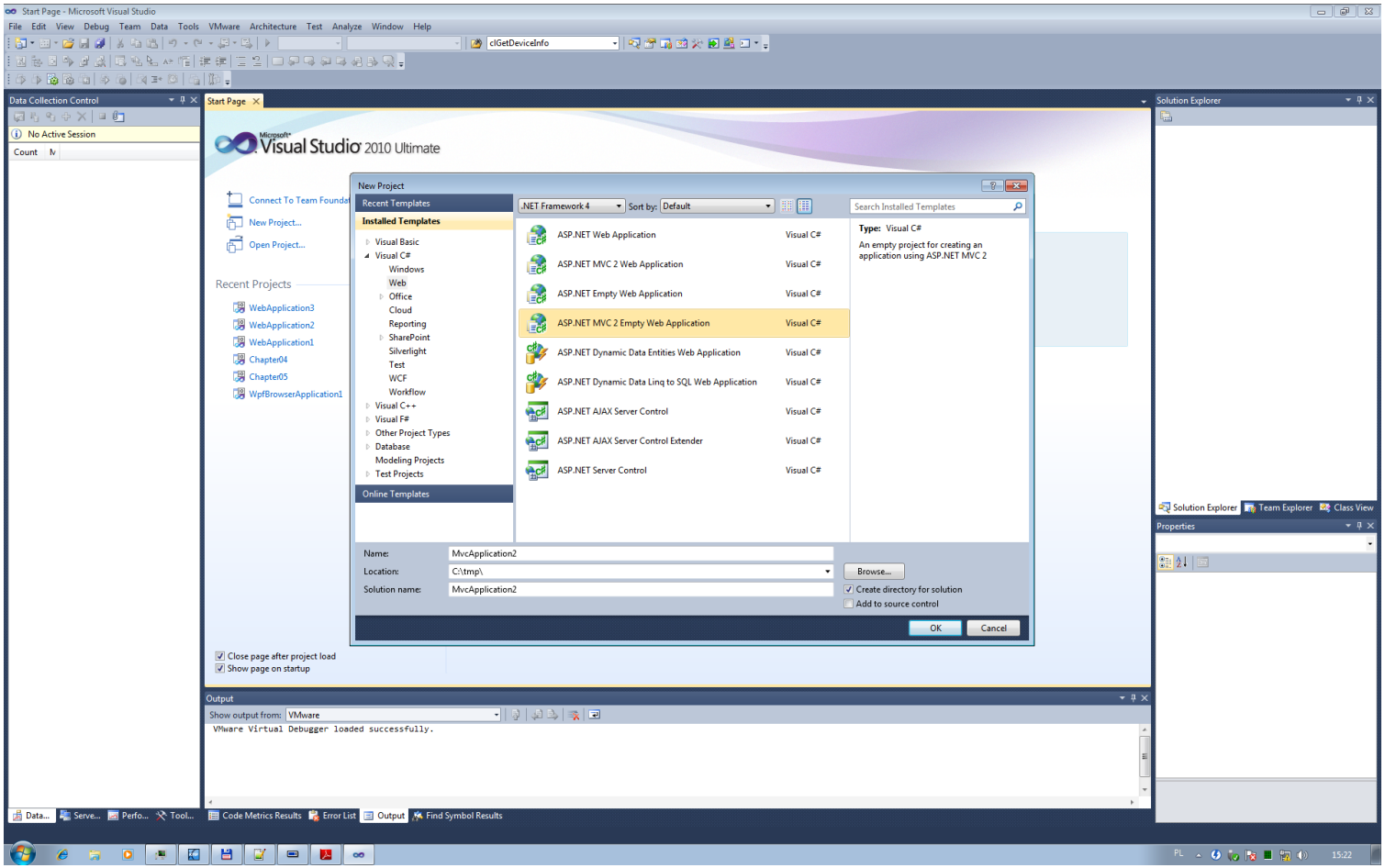
- `/App_Data` – katalog do przechowywania danych, w szczególności jest dane są zapisane w plikach *.mdf w przypadku bazy danych SQL Server Express Edition bądź *.mdb dla programu Microsoft Access), można także przechowywać w tym katalogu także inne prywatne pliki np.: pliki w formacie XML, pliki z tego katalogu nie są udostępniane przez serwer IIS, ale można z nich korzystać z poziomu kodu C# albo VB z poziomu aplikacji,
- `/bin` – w katalogu znajdują się podzespoły .NET z rozwijanej aplikacji MVC (a także inne podzespoły z których korzysta aplikacja WEB),
- `/Content` – katalog jest przeznaczony do przechowywania statycznych, publicznie dostępnych plików jak np.: *.css albo pliki z danymi graficznymi,
- `/Controllers` – katalog jest przeznaczony do przechowywania klas kontrolerów,
- `/Models` – ten katalog jest przeznaczony do przechowywania klas reprezentujących dane, zaleca się aby model był określony przez klasę znajdującą się w oddzielnym pliku.

Katalogi i ich przeznaczenie w aplikacjach MVC

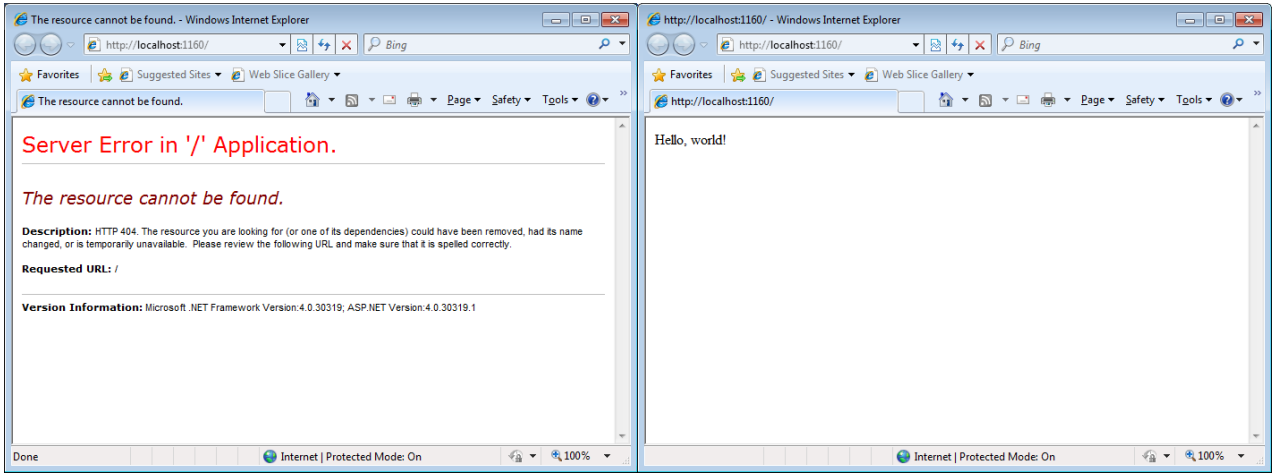
- `/Scripts` – jest to katalog dla statycznych plików dostępnych publicznie, głównymi plikami jakie są tam przechowywane są pliki JavaScript (*.js) związane ze wsparciem technologii AJAX, obecne są także pliki JQuery oraz pliki odpowiedzialne za poprawność po stronie klienta,
- `/Views` – katalog przechowuje widoki (zazwyczaj pliki *.aspx) oraz widoki częściowe (zazwyczaj pliki *.ascx),
- `/Views/Shared` – znajdują się tu pliki, które nie są związane z określonym kontrolerem, np.: strona główna (*.Master) i inne dzielone oraz częściowe widoki,
- `/Views/Web.config` – nie jest to plik zawierający konfigurację aplikacji, zawiera opis konfiguracji niezbędnej, aby poprawnie prezentować zwykłe strony ASP.NET ASPX,
- `/Global.asax` – określa globalny obiekt aplikacji ASP.NET (kod C# zawarty jest w /Global.asax.cs), gdzie znajdują się reguły trasowania (marszrut) oraz dodatkowy kod uruchomiany podczas inicjalizacji oraz deinicjalizacji aplikacji, a także kod wykonywany w momencie przechwycenia nieobsłużonego wyjątku,
- `/Web.config` – opis konfiguracji aplikacji MVC.

Nasza pierwsza strona w ASP.NET MVC

Pusta strona MVC w Visual Studio 2010



Pusta aplikacja po uruchomieniu:

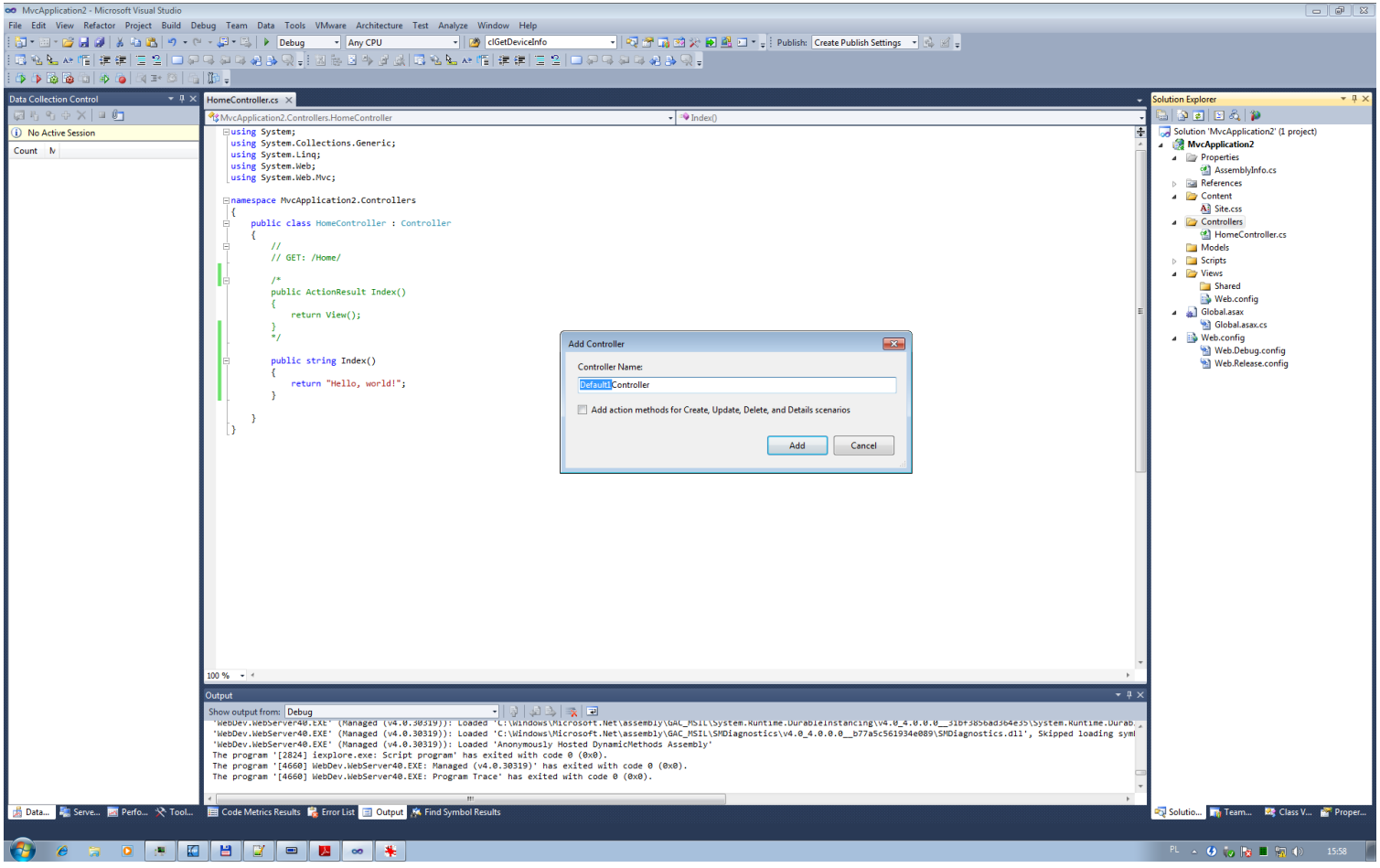


Aby to zmienić trzeba utworzyć kontroler „domowy” – HomeController (katalog Controllers):

```
namespace MvcApplication2.Controllers {
    public class HomeController : Controller {
        //
        // GET: /Home/

        public string Index(){
            return "Hello, world!";
        }
    }
}
```

Strona z pudełka



Widok jako plik Index.ascx

Plik widoku Index.ascx powinien być umieszczony w katalogu ~/Views/Home albo ~/Views/Shared.

```
<%@ Control Language="C#" Inherits="System.Web.Mvc.ViewUserControl<dynamic>" %>
<h1>Tytuł strony</h1>
<p>Dzień Dobry!</p>
<p>(View)</p>
```

Kontroler przedstawia się w następujący sposób:

```
namespace MvcApplication2.Controllers {
    public class HomeController : Controller {
        //
        // GET: /Home/


      public ActionResult Index() {
            return View();
      }
   }
}
```

Zawartość tworzona dynamicznie

W zależności od wartości godziny (na serwerze) będzie generowane inne przywitanie:

```
public ViewResult Index() {
   int hour = DateTime.Now.Hour;
   ViewData["greeting"] = (hour < 12 ? "Dzień Dobry!" : "Dobry Wieczór!");
   return View();
}
```

Na stronie widoku (Index.ascx) należy dodać jedną linię:

```
<%: ViewData["greeting"] %>, world (from the view)!
```

Jeśli stosowane jest Visual Studio 2008 bądź platforma .NET w wersji 3.5, linia kodu przedstawia się nieco inaczej:

```
<%= Html.Encode(ViewData["greeting"]) %>, world (from the view)!
```

URL i marszruty

Tradycyjne podejście do URL:

- `http://mysite.com/default.aspx -> e:\webroot\default.aspx`
- `http://mysite.com/admin/login.aspx -> e:\webroot\admin\login.aspx`
- `http://mysite.com/articles/AnnualReview -> File not found! Send error 404.`

Podejście w architekturze MVC:

- `http://mysite.com/photos -> { controller = "Gallery", action = "Display" },`
- `http://mysite.com/admin/login.aspx -> { controller = "Auth", action = "Login" },`
- `http://mysite.com/articles/AnnualReview ->`
  `{ controller = "Articles", action = "View", contentItemName = "AnnualReview" }`

Domyślne adresy w ASP.NET MVC:

- `/ -> { controller = "Home", action = "Index" },`
- `/Forum -> { controller = "Forum", action = "Index" },`
- `/Forum/ShowTopics -> { controller = "Forum", action = "ShowTopics" },`
- `/Forum/ShowTopics/75 -> { controller = "Forum", action = "ShowTopics", id = "75" }.`

Rejestrowanie domyślnych dróg

Domyślne rejestrowany jest ogólny schemat adresowania dla aplikacji MVC, w pliku Global.asax.cs:

```
namespace MvcApplication2 {
    public class MvcApplication : System.Web.HttpApplication {
        public static void RegisterRoutes(RouteCollection routes)  {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index",
                          id = UrlParameter.Optional }
            );
        }

        protected void Application_Start() {
            AreaRegistration.RegisterAllAreas();
            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

Dodanie nowej drogi

Dodanie nowej marszruty:

```
routes.Add(new Route("Catalog", new MvcRouteHandler()) {
  Defaults = new RouteValueDictionary(
    new { controller = "Products", action = "List" }
)});
```

Model i Widok

Domyślnie aplikacje ASP.NET MVC, oferują wbudowane obiekty do realizacji typowych zadań, co pomaga w szybkiemu tworzeniu podstawowego szkieletu aplikacji.
Dla dwóch następujących klas:

```
public class Person {
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
    public Address HomeAddress { get; set; }
    public bool IsApproved { get; set; }
}

public class Address {
    public string Line1 { get; set; }
    public string Line2 { get; set; }
    public string City { get; set; }
    public string PostalCode { get; set; }
    public string Country { get; set; }
}
```

Model i Widok

Utworzenie widoku do edycji danych osobowych:

```
<h2>Edit this person</h2>
<% using(Html.BeginForm()) { %>
<%: Html.EditorForModel() %>
<p><input type="submit" value="Save" /></p>
<% } %>
```

Wykorzystanie atrybutów do określenia dodatkowych informacji o danych:

```
public class Person {
    [HiddenInput (DisplayValue = false)]  public int PersonId { get; set; }
    [DisplayName("First name")] public string FirstName { get; set; }
    [DisplayName("Last name")] public string LastName { get; set; }
    [DataType(DataType.Date)] // Show only the date, ignoring any time data
    [DisplayName("Born")] public DateTime BirthDate { get; set; }
    public Address HomeAddress { get; set; }
    [DisplayName("May log in")] public bool IsApproved { get; set; }
}
```

Edycja poszczególnych pól

Własny edytor osobowy, gdzie samodzielnie wyszczególniamy pola do edycji:

```
<% using(Html.BeginForm()) { %>
  <fieldset>
     <legend>Person</legend>
      <div class="field">
           <label>Name:</label>
           <%: Html.EditorFor(x => x.FirstName) %>
           <%: Html.EditorFor(x => x.LastName) %>
      </div>
      <div class="field">
           <label>Born:</label>
           <%: Html.EditorFor(x => x.BirthDate) %>
      </div>
      <div align="center"><%: Html.EditorFor(x => x.IsApproved) %>May log in</div>
      <fieldset>
         <legend>Home address</legend>
         <div class="addressEditor">
           <%: Html.EditorFor(x => x.HomeAddress) %>
         </div>
      </fieldset>
  </fieldset>
  <p><input type="submit" value="Save" /></p>
<% } %>
```

Dostępne „udogodnienia” w prezentacji danych

- Display, Html.Display(”Title”), rysowanie widoku tylko do odczytu dla własności w danym modelu, wybierając wzorzec właściwy dla typu własności i dowolnych metadanych związanych z modelem,
- DisplayFor, Html.DisplayFor(x => x.Title), odmiana poprzedniego „pomocnika” o silnej kontroli typów,
- DisplayForModel, Html.DisplayForModel(), skrót dla Html.DisplayFor(x=> x), rysuje widok tylko do odczytu dla całego modelu niż tylko dla wybranej własności,
- Editor, Html.Editor(”Title”), rysowanie kontrolki do edycji dla wskazanej własności modelu, wybierając wzorzec odpowiedni dla typu własności i metadanych,
- EditorFor, Html.EditorFor(x => x.Title), odmiana poprzedniego „pomocnika” o silnej kontroli typów,
- EditorForModel, Html.EditorForModel(), odmiana Html.EditorFor(x=> x), innymi słowy, rysuje kontrolki dla całego modeli niż tylko dla wskazanych własności,
- Label, Html.Label(”Title”), tworzenie etykiety HTML <label> odnoszącej się do własności modelu,
- LabelFor, Html.LabelFor(x => x.Title), odmiana poprzedniego „pomocnika” o silnej kontroli typów,
- LabelForModel, Html.LabelForModel(), skrót dla Html.LabelFor(x => x), tworzenie etykiety HTML <label> odnoszącej się do całego modelu niż tylko do wybranego elementu,
- DisplayText, Html.DisplayText(”Title”), reprezentacja w postaci ciągu znaków wskazanej własności modelu,
- DisplayTextFor, Html.DisplayTextFor(x => x.Title), odmiana poprzedniego „pomocnika” o silnej kontroli typów.

AJAX i JavaScript i aplikacjach MVC

W pierwszej kolejności do strony widoku trzeba dodać dwie linie wczytujące pliki JavaScript:

```
<body>
<!-- Rest the page goes here -->
    <script type="text/javascript"
        src="<%: Url.Content("~/Scripts/MicrosoftAjax.js") %>"></script>
    <script type="text/javascript"
        src="<%: Url.Content("~/Scripts/MicrosoftMvcAjax.js") %>"></script>
</body>
```

Elementy strony odpowiedzialne za wyświetlanie czasu w trzech różnych strefach czasowych:

```
<h2>Którą mamy godzinę?</h2> <p> Pokaż czas w strefie:
<%: Ajax.ActionLink("UTC", "GetTime", new { zone = "utc" },
                new AjaxOptions { UpdateTargetId = "myResults" }) %>
<%: Ajax.ActionLink("BST", "GetTime", new { zone = "bst" },
                new AjaxOptions { UpdateTargetId = "myResults" }) %>
<%: Ajax.ActionLink("MDT", "GetTime", new { zone = "mdt" },
                new AjaxOptions { UpdateTargetId = "myResults" }) %>
</p> <div id="myResults" style="border: 2px dotted red; padding: .5em;">
Aktualny czas  </div>
<p>Strona została wygenerowana o godzinie
                    <%: DateTime.UtcNow.ToString("h:MM:ss tt") %> (UTC)</p>
```

Obsługa strefy czasowej

Poniższy kod powinien być umieszczony w klasie kontrolera:

```
...
 private Dictionary<string, int> offsets = new Dictionary<string, int> {
                { "utc", 0 }, { "bst", 1 }, { "mdt", -6 } };
public string GetTime(string zone)  {
    DateTime time = DateTime.UtcNow.AddHours(offsets[zone]);

    return string.Format("<div>The time in {0} is {1:h:MM:ss tt}</div>",
                        zone.ToUpper(), time);
}
...
```

W następnym tygodniu między innymi

1. czym są usługi sieciowe (WEB services) w .NET,
2. przykłady usług sieciowych .NET,
3. usługi sieciowe oparte o XML (SOAP),
4. technologia Windows Communication Foundation (WCF),
5. model programowania WCF dla usług sieciowych,
6. aplikacje zorientowane na usługi.

Proponowane tematy prac pisemnych:

1. przedstawienie architektury MVC,
2. tworzenie modelu danych i łączenie go z widokiem,
3. architektura i model obiektu kontrolera.

Dziękuje za uwagę!!!