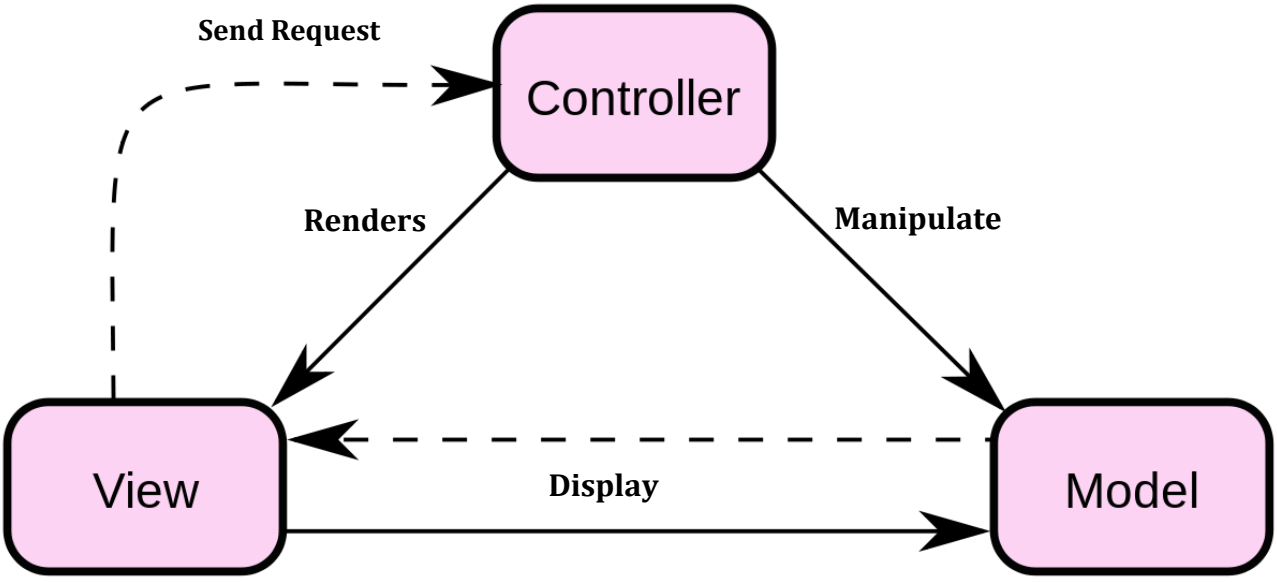
MVC – introducere, arhitectura. Avantajele MVC. Structura unui proiect MVC. Rutare.

Ce este MVC

Model-View-Controller (MVC) este un model arhitectural utilizat in dezvoltarea aplicatiilor. Succesul modelului se datoreaza izolarii logicii aplicatiei (business logic) fata de interfata cu utilizatorul, rezultand o aplicatie unde aspectul vizual si/sau nivelele inferioare ale regulilor de business sunt mai usor de modificat, fara a afecta alte nivele.

Arhitectura:



Controller (Control de intrare)

Cu ajutorul acestei componente se controleaza accesul la aplicatie.

Controller-ul este reprezentat de clase si este partea aplicatiei care se ocupa de interactiunea cu utilizatorul. In controller **se citesc datele** introduse de utilizator, **se trimit catre Model**, **se executa operatiile**, dupa care **se trimite raspunsul catre View**.

Fiecare controller contine asa numitele **Action-uri** reprezentate de metode.

Model (Stratul business – prelucrarea datelor)

Modelul este responsabil cu gestionarea datelor din aplicatie si manipularea acestora. Acesta raspunde cererilor care vin din View, cat si intructiunilor din Controller. Este cel mai de jos nivel care se ocupa cu **procesarea** si **manipularea** datelor, reprezentand nucleul aplicatiei (aici se face legatura cu baza de date).

View (Interfata cu utilizatorul)

View-ul reprezinta interfata cu utilizatorul si este partea in care se **afiseaza datele**, adica inregistrarile din baza de date si informatiile generate de aplicatie.

View-ul contine de cele mai multe ori limbaj HTML, CSS, JS si este partea vazuta de catre utilizator prin intermediul browserului.

Logica unei aplicatii MVC

Pentru o aplicatie web este necesar sa stabilim urmatoarele elemente:

- **Router** - sistemul care leaga componentele aplicatiei de anumite puncte de intrare (entrypoints) prin intermediul unui URL
- **Controller** - acesta este capabil de a manipula rute, fisiere, clase, metode si functii
- **Model** – reprezinta stratul (layer-ul) de procesare a datelor
- **View** – reprezinta componenta de afisare a informatiilor din cadrul aplicatiei

Avantajele MVC

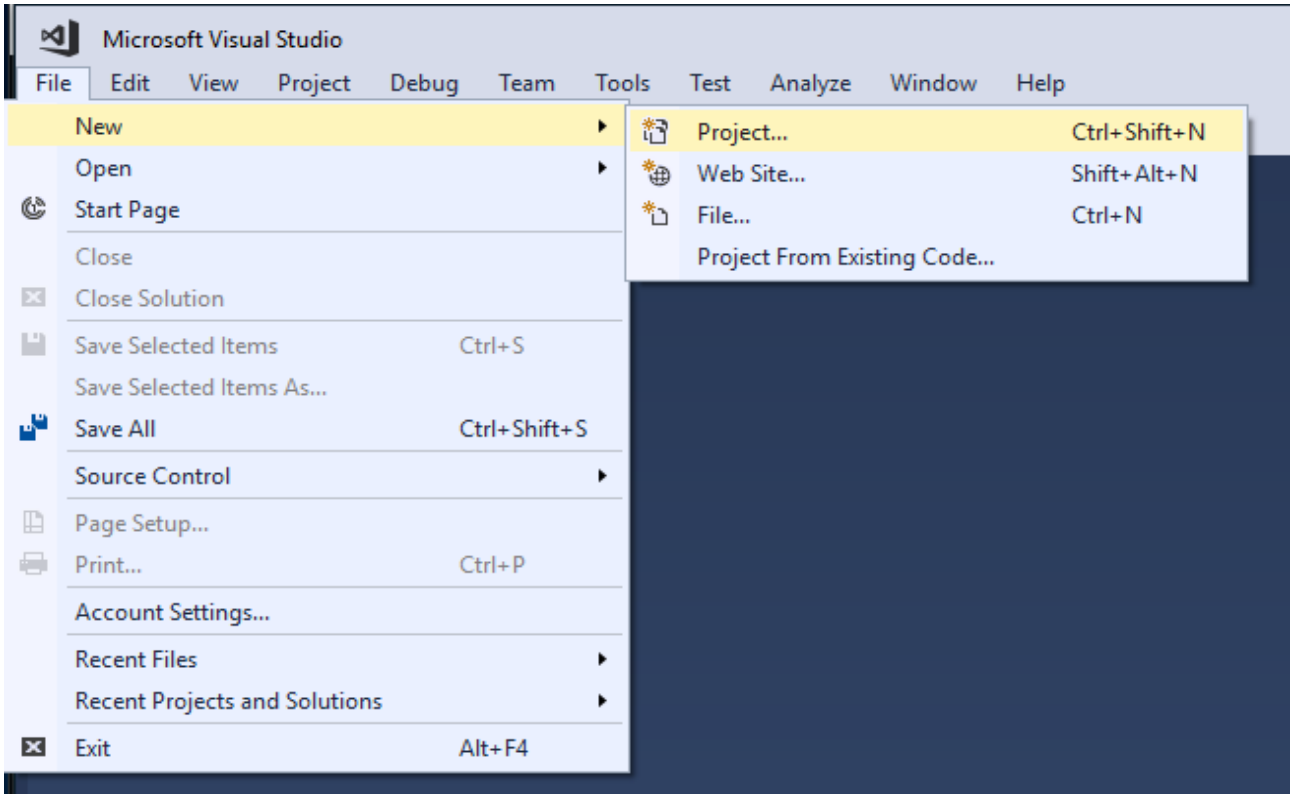
- **Organizare** datorita structurii Model-View-Controller
- **Dezvoltare rapida** – se poate realiza o dezvoltare rapida si paralela deoarece fiecare componenta poate fi dezvoltata de catre alt programator
- **Reutilizarea codului**
- **Flexibilitate** – modificarile nu afecteaza intregul Model deoarece partea modelului nu depinde de partea vizuala, prin urmare modificarile din Model nu vor afecta intreaga arhitectura.

Dezavantajele MVC

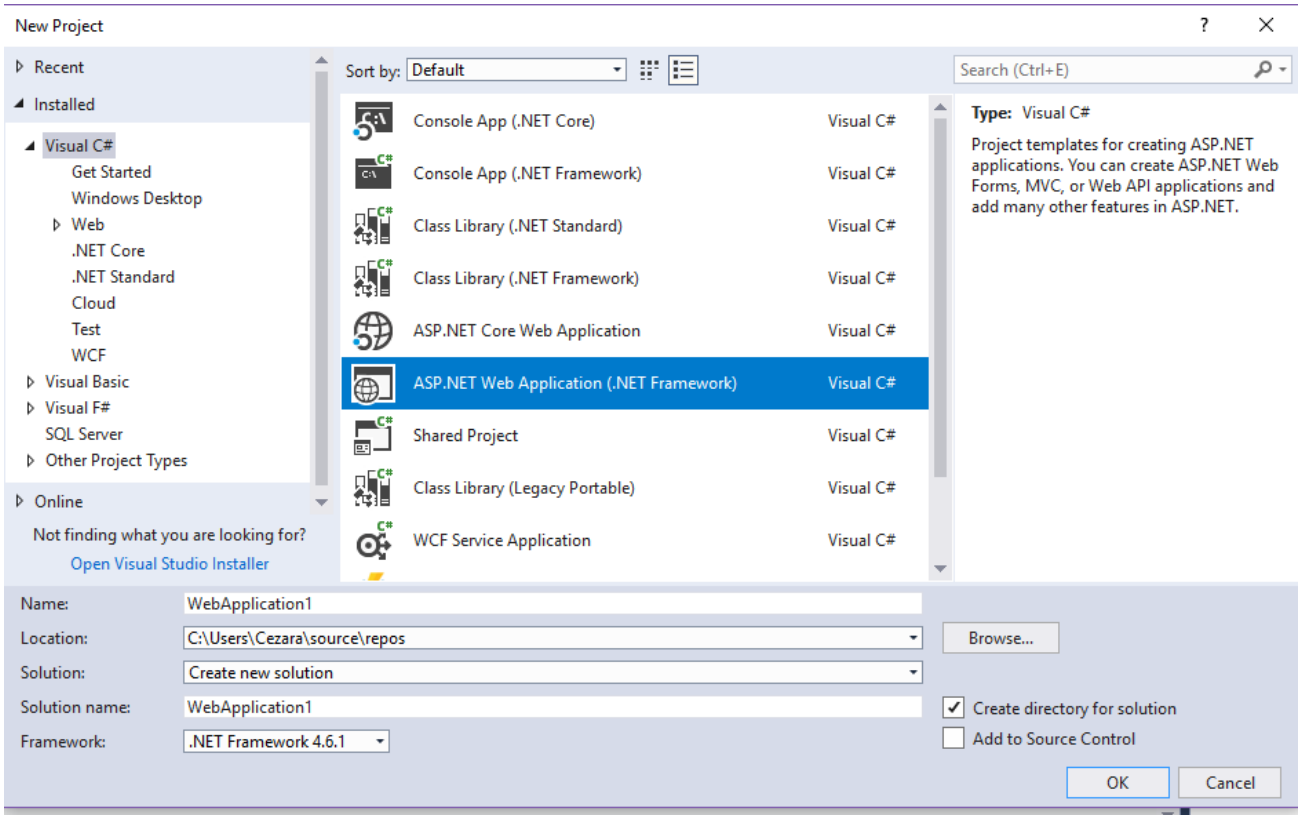
- Timpul de finalizare al unui proiect este mare la dezvoltarea aplicatiilor folosind acest model, fara cunostintele corespunzatoare
- Structura modelului creste complexitatea dezvoltarii

Crearea unui proiect MVC

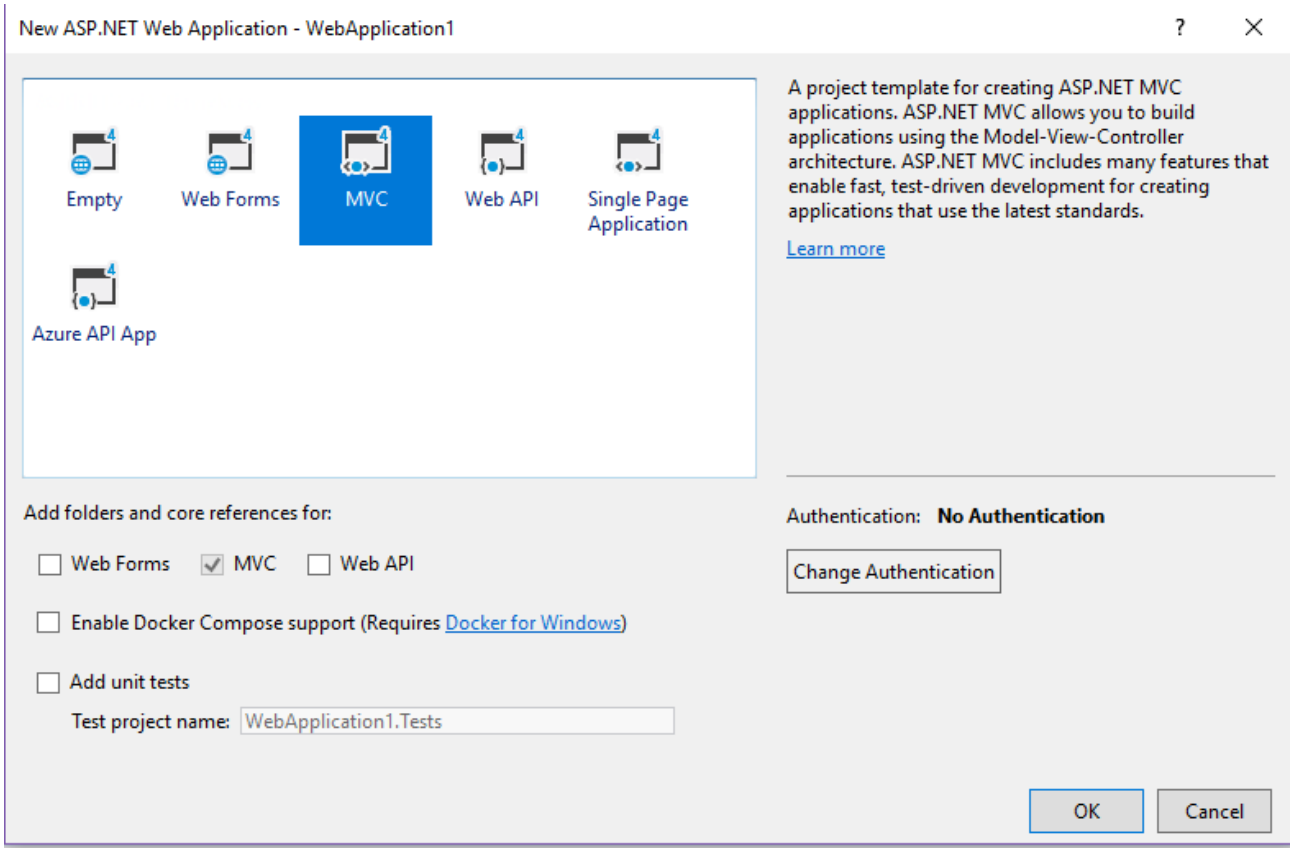
Din meniul **File** selectam **New > Project**



In urmatorul ecran selectam **ASP.NET Web Application (.NET framework)**

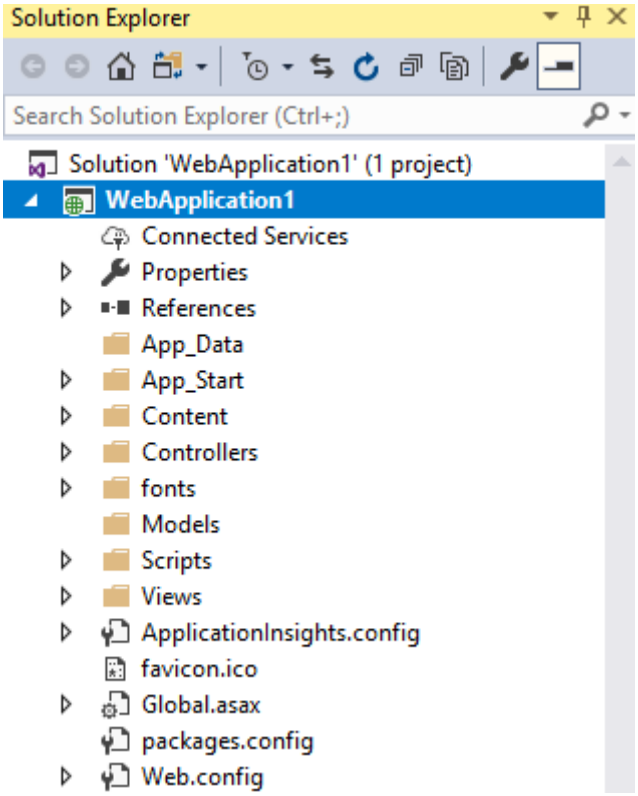


In cadrul ferestrei de mai jos trebuie selectat tipul aplicatiei. Se selecteaza MVC pentru a adauga un nou proiect.



Arhitectura unui proiect MVC

Proiectul MVC nou creat are urmatoarele componente:



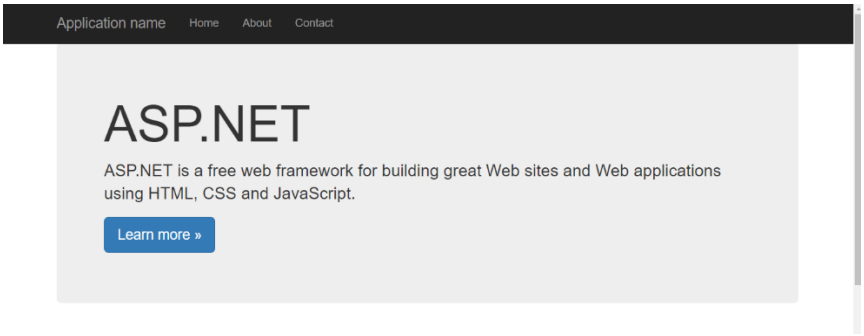
- **App_Data** – in acest folder se stocheaza informatii relevante in ceea ce priveste datele aplicatiei (ex: baza de date, fisierele .mdf, fisierele xml, etc)

- **App_Start** – acest folder contine fisiere necesare (BundleConfig.cs, FilterConfig.cs, RouteConfig.cs) pentru configuratia initiala a aplicatiei.
  - In fisierul **RouteConfig.cs** dezvoltatorul poate configura rutele aplicatiei

- **Content** – acest folder contine asset-urile aplicatiei (CSS si imagini)

- **Controllers** – acest folder contine Controller-ele aplicatiei. MVC impune ca numele tuturor controller-elor sa contina la final cuvantul Controller

- **Fonts** – in acest folder se pot pune fisiere care contin fonturi si care pot fi folosite in CSS
- **Models** – folderul contine modelele aplicatiei
- **Scripts** – folderul contine fisiere si librarii JavaScript
- **Views** – folderul contine fisierele de tip View (interfata cu utilizatorul) ale aplicatiei



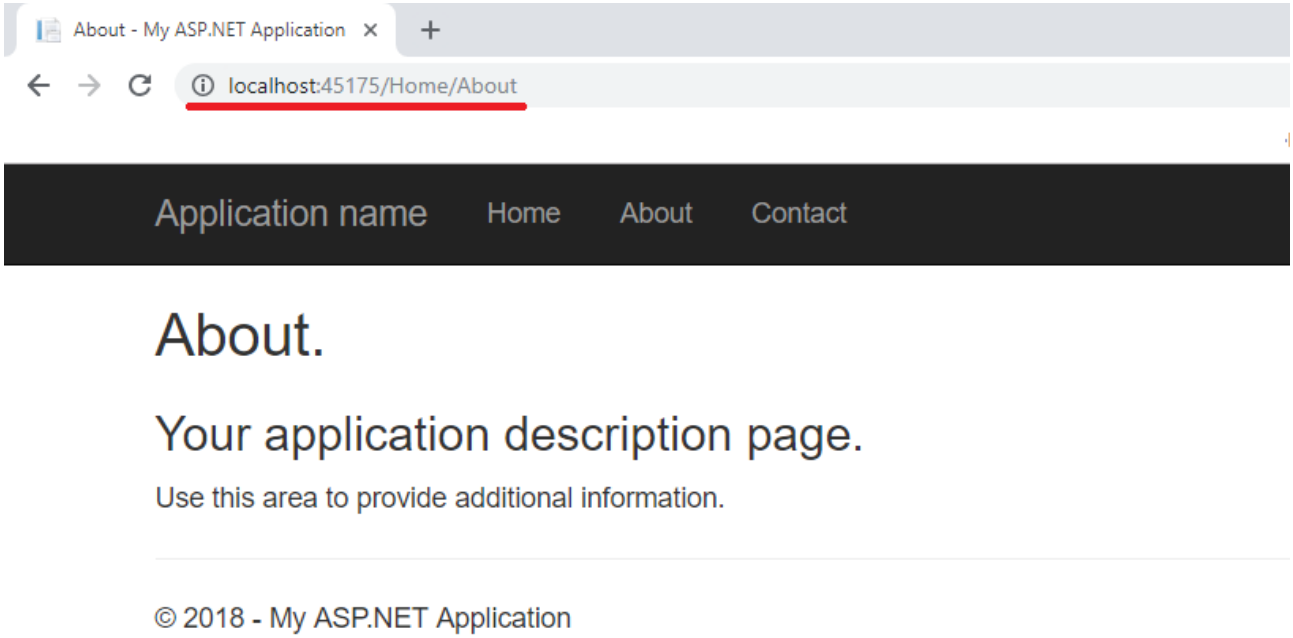
Sistemul de rutare

In ASP.NET Web Forms fiecare URL trebuie sa coincida cu un fisier .aspx. De exemplu, un URL de tipul: **http://domain/Default.aspx** trebuie sa coincida cu un fisier **Default.aspx** care contine codul.

ASP.NET a introdus termenul de "**Routing**" si implicit rutarea pentru a elimina necesitatea maparii fiecarui URL cu un fisier fizic.

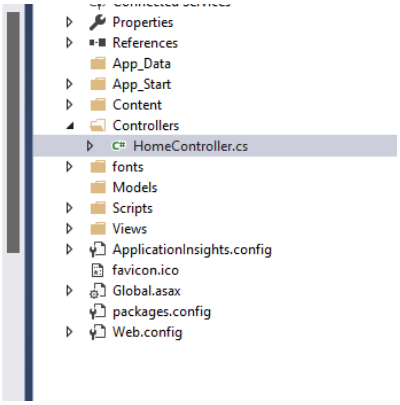
Rutele reprezinta diferite URL-uri din aplicatie care sunt procesate de un anumit controller si de o anumita metoda pentru generarea unui raspuns catre client. Framework-ul ASP.NET MVC invoca diverse clase de tip Controller (si diferite metode ale acestora) in functie de URL-ul cerut de client.

Astfel, pentru a accesa o anumita pagina, este necesar ca pentru aceasta sa existe o ruta definita, cat si un Controller care are o metoda care sa raspunda pentru aceasta resursa. Formatul de baza al rutelor in ASP.NET este urmatorul: **/[NumeController]/[NumeActiune]/[Parametrii]**.



Codul aferent acestei rute se regaseste in **HomeController.cs** din folderul **Controllers**:

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";
        return View();
    }
```



Aceste rute se pot defini in fisierul **RouteConfig.cs** din folderul **App_Start** aflat in directorul de baza al aplicatiei construite.

Fisierul are urmatorul format:

```
public class RouteConfig
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",     Numele rutei
                url: "{controller}/{action}/{id}"  Modelul(pattern-ul) URL-ului
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }    Ruta default
            );
        }
    }
```

In acest fisier observam ca variabila **routes**, care este de tipul clasei **RouteCollection**, ofera mai multe metode necesare definirii rutelor.

Metoda **MapRoute** primeste ca argumente 3 parametri:

- **name**: care reprezinta numele rutei
- **url**: care reprezinta schema URL-ului sau segmentele acestuia
- metoda **defaults** care primeste o colectie in care sunt definite detaliile dupa cum urmeaza:
    - **controller** - primeste ca valoare numele controller-ului care sa raspunda la aceasta ruta
    - **action** - primeste ca valoare numele metodei din controller care sa raspunda la aceasta ruta
    - pentru fiecare parametru adaugat in ruta, defineste **tipul parametrilor** sau daca acestia sunt **necesari** sau **optionali** sau se pot seta valorile implicite

In exemplul anterior, se defineste ruta **/Home/Index/{id}** care este procesata de controller-ul **HomeController** prin metoda **Index**. Parametrul ID este optional si poate fi omis din URL.

In acest sistem de rutare, pot fi mai multi parametri delimitati prin caracterul "/".

In cazul in care aplicatia este accesata fara segmentele necesare in cadrul URL-ului, adica se cere pagina principala a aplicatiei "/", framework-ul ASP.NET MVC va raspunde in mod implicit cu metoda "**Index**" din controllerul "**Home**".

Valoarea implicita a unui parametru se poate seta folosind urmatoarea secventa de cod:

```
routes.MapRoute(
        "Default",
"{controller}/{action}/{id}",
new { controller = "Home", action = "Index", id = "5" }
);
```

In Aplicatiile ASP MVC folosirea parametrilor in URL este preferata fata de trimiterea argumentelor ca si query string-uri (modul preferat pentru ASP.NET Web Forms). Astfel, pentru a trimite parametri unei rute, se prefera ca acestia sa fie setati in ruta ca si variabile, conform exemplului de mai jos:

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");


        routes.MapRoute(
            name: "Hello",
            url: "{ controller}/{ action}/{ name}/{ id}",
            defaults: new { controller = "Home", action = "Index", name =
UrlParameter.Optional, id = UrlParameter.Optional }
        );


        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
        );

    }
}
```

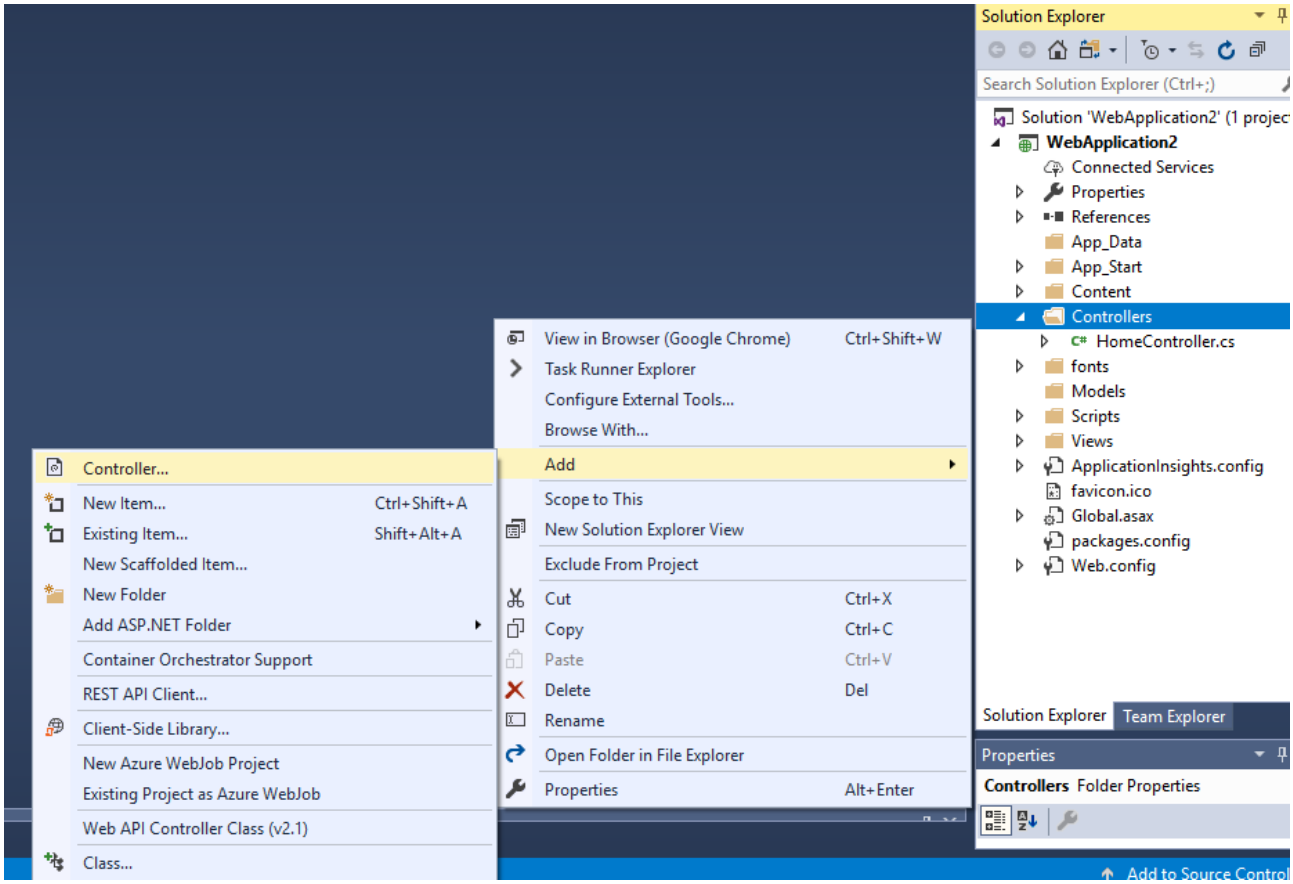
In acest exemplu, pentru ruta **Hello** se poate accesa pagina folosind urmatorul url: **www.exemplu.com/Home/Index/nume/id**.

In cazul in care se va accesa pagina folosind doar numele Controller-ului, adica **Home** (**www.exemplu.com/Home**) atunci metoda implicita care va raspunde solicitarii este metoda **Index**.

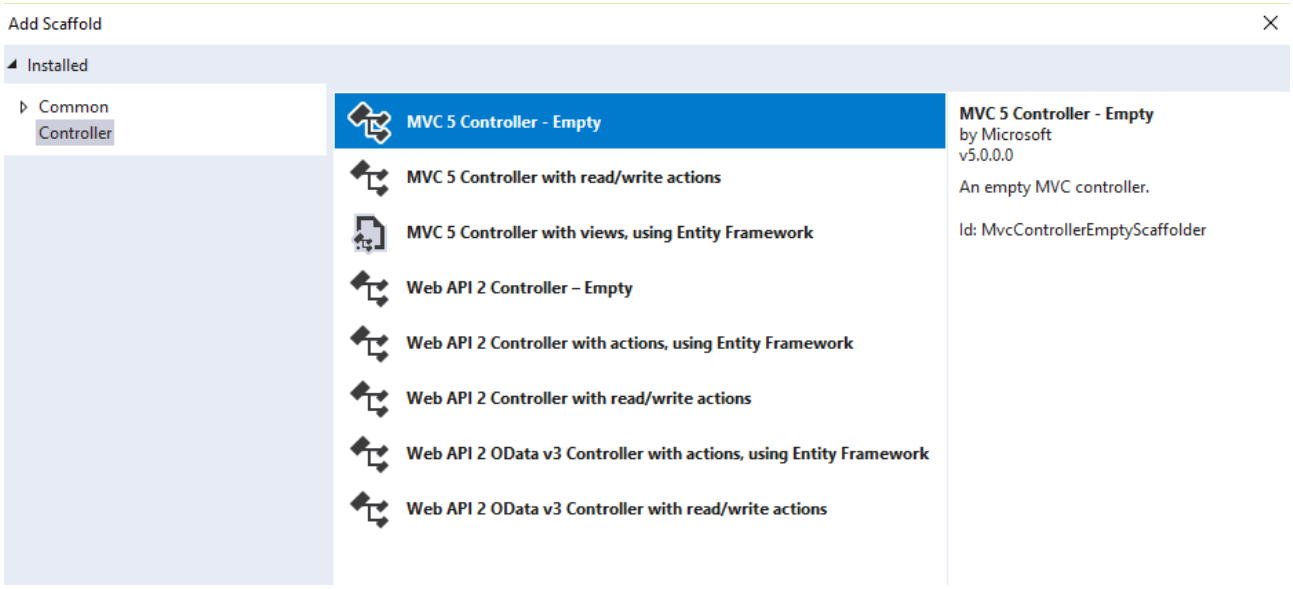
In cazul ASP.NET MVC parametrii sunt delimitati prin caracterul "/" intr-o ruta, de exemplu *www.exemplu.com/controller/actiune/nume/id* fata de modul traditional care foloseste query strings:
www.exemplu.com/Pagina.aspx?nume=valoare&id=valoare

Exemplu:

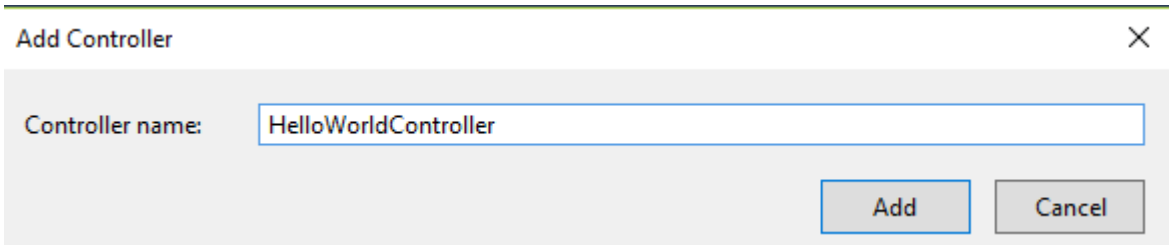
Pentru exemplul urmator avem nevoie de un nou Controller in care preluam parametrii unei rute pentru a fi afisati in browser. Astfel, facem click dreapta pe folderul **Controllers** si din meniul **Add** selectam "**Controller**..."



In fereastra aparuta, selectam **MVC 5 Controller – Empty**:



Adaugam numele Controllerului – **HelloWorldController** si apasam butonul Add.



Codul noului Controller generat contine o metoda numita **Index()**. In aceasta metoda vom face modificarile necesare pentru acest exemplu.

HelloWorldController.cs

WebApplication2 — WebApplication2.Controllers.HelloWorldController

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Web;
5   using System.Web.Mvc;
6
7   namespace WebApplication2.Controllers
8   {
9       public class HelloWorldController : Controller
10      {
11          // GET: HelloWorld
12          public ActionResult Index()
13          {
14              return View();
15          }
16      }
17  }
```

Pentru a putea afisa continutul, vom face o noua ruta in fisierul **RouteConfig.cs** dupa cum urmeaza:

```
routes.MapRoute(
    name: "HelloWorld",
    url: "{controller}/{action}/{name}/{id}",
    defaults: new { controller = "HelloWorld", action = "Index", name = "World",
      id = UrlParameter.Optional }
);
```

Ruta adaugata va contine 2 parametri – **name** si **ID**. Pentru noua ruta, configuram Controller-ul de raspuns cu valoarea “HelloWorld”, iar pentru Action configuram valoarea “Index”. Astfel, Controller-ul HelloWorld creat va raspunde la aceasta ruta.

OBSERVATIE:

/!\ Ruta trebuie definita **inaintea rutei default**, deja existenta in fisierul RouteConfig.cs, deoarece rutele sunt interpretate in mod cascada (de sus in jos). Framework-ul utilizeaza prima configuratie din fisier care contine acelasi numar de parametrii ca ruta accesata din browser.

Parametrii **name si id** au urmatoarele valori:

- **name** are valoarea default "world". In cazul in care acesta nu apare in URL va fi trimis in mod automat la Controller cu aceasta valoare

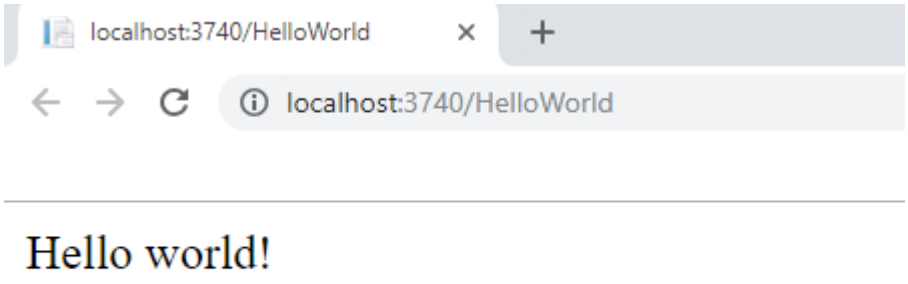
- **id** este parametru optional. Daca parametrul nu a fost trimis prin URL, in Controller el este null. In cazul in care apare in URL, in Controller se poate prelua aceasta valoare

In cadrul Controller-ului, in metoda care va raspunde pentru ruta definita, putem prelua parametrii din URL sub forma argumentelor metodei Index(). Astfel, pentru cei doi parametri definiti ({name} si {id}) metoda Index devine:

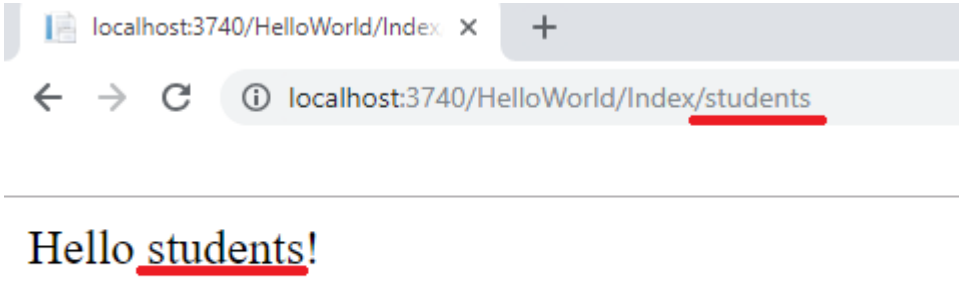
```
public class HelloWorldController : Controller
{
    // GET: HelloWorld
    public string Index(string name, int? id)
    {
        string response = "Hello " + name + "!";
        if(id != null)
        {
            response += "; ID = " + id.ToString();
        }

        return response;
        //return View();
    }
}
```

Tipul returnat de metoda Index a fost schimbat in String pentru a putea afisa in browser un simplu text. Astfel, valoarea de return devine valoarea variabilei “response” care inglobeaza valorile parametrilor.

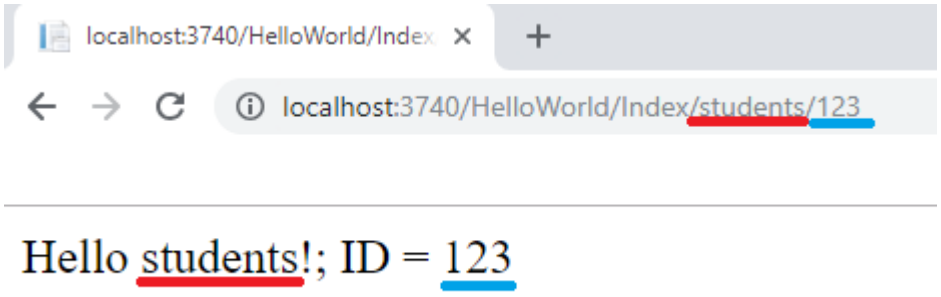
La accesarea paginii fara niciun parametru, observam ca pentru variabila “name” s-a transmis valoarea acesteia implicita: “world”. Dupa cum stim, daca in URL nu apare numele metodei, (se foloseste doar numele controller-ului), metoda Index este accesata in mod implicit.



La adaugarea valorii pentru variabila “name” in URL, putem observa cum aceasta a fost transmisa la Controller si a fost afisata in pagina.



Cand parametrul optional ID are valoare, secventa de cod specifica acestuia este executata si valoarea sa apare in raspunsul primit de la controller:



Pentru fiecare **Controller** si **Actiune** in parte se pot defini si rute customizate. Daca avem un controller “Users” si o metoda “Show” care va afisa detaliile unui utilizator, putem sa modificam URL-ul prin scrierea unei rute astfel:

```
            routes.MapRoute(
                  name: "Users",
                  url: "users/{user_id}",
                  defaults: new { controller = "Users", action = "Show", user_id
= UrlParameter.Optional }
              );
```

Accesarea resursei se face prin URL-ul “/users/{user_id}” (ex: “/users/123”). Deoarece rutele necesita un Controller si o Actiune pentru a putea fi procesate, rutele definite trebuie scrise inainte de ruta **Default** pentru a fi procesate corect.

Daca rutele scrise de utilizator sunt scrise dupa ruta Default, acestea nu se vor accesa corect deoarece cele 3 segmente (Controller, Action si paremetrul ID optional) sunt identice cu ruta Default. Astfel, procesarea se va face de catre ruta Default si cautarea unei rute compatibile se va termina.

In momentul accesarii “/users/123” de catre ruta Default, pagina va fi executata de Controllerul Users prin metoda 123. Acest lucru se datoreaza faptului ca in ruta default se asteapta atat un nume de Controller, cat si un nume de actiune.

```
routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id
= UrlParameter.Optional }
            );
```

Pentru a elimina aceasta problema este necesara scrierea tuturor rutelor inainte de ruta default. Acelasi lucru se aplica si pentru rutele scrise de dezvoltator. ***In momentul scrierii rutelor, dezvoltatorul trebuie sa se asigure ca nu exista ambiguitate intre definitiile acestora***.

Constrangerile parametrilor:

Pentru a asigura un anumit tip de date sau un anumit format pentru parametrii transmisi catre Controller, este necesara declararea unor constrangeri in fisierul RoutesConfig.cs. Astfel, pentru a constrange parametrul, in definirea rutei se adauga proprietatea "**constraints**" care defineste printr-o **expresie regulata** caracterele acceptate pentru acel parametru:

```
routes.MapRoute(
                name: "Default",
                url: "users/{user_id}",
                defaults: new { controller = "Users", action = "Show",
user_id = UrlParameter.Optional },
                constraints: new { user_id = @"\d+" }
            );
```

Conform constrangerii, parametrul "user_id" accepta doar cifre. Astfel, valoarea parametrului ajunsa la Controller va fi formata numai din cifre.

/!\ Pana **duminica**, **11 Oct**, trebuie sa completati pe drive echipele impreuna cu proiectele alese.