

Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC

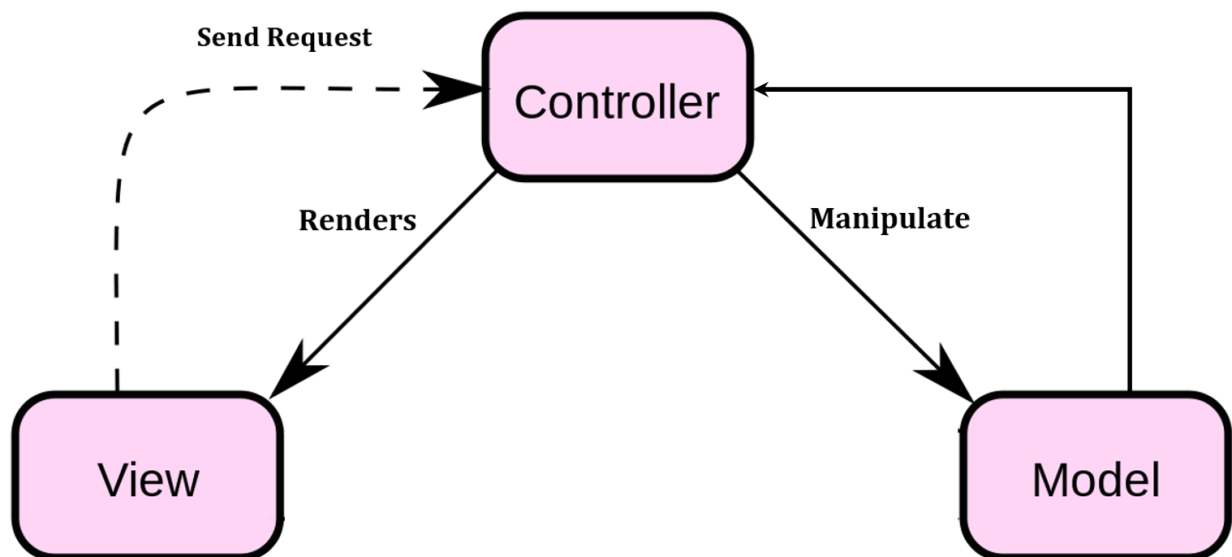
Curs 3

Cuprins:

Arhitectura MVC	2
Model (Stratul business – prelucrarea datelor)	2
Controller	3
View (interfata cu utilizatorul).....	3
Crearea unei aplicatii in ASP.NET Core 6.0 (Visual Studio 2022)	4
Structura unui proiect MVC – Sistemul de fisiere.....	7
Sistemul de rutare.....	10
Exemple de implementare a rutelor:	18
Configurarea rutelor:.....	18
Definirea rutelor custom	24
Constrangerile parametrilor	26

Arhitectura MVC

Model-View-Controller (MVC) este un model arhitectural utilizat în dezvoltarea aplicațiilor. Succesul modelului se datorează izolării logicii aplicației (business logic) față de interfața cu utilizatorul, rezultând o aplicație unde aspectul vizual și nivelele inferioare ale regulilor de business sunt mai ușor de modificat, fără a afecta alte nivele.



Model (Stratul business – prelucrarea datelor)

Modelul este responsabil cu gestionarea datelor din aplicație și manipularea acestora. Acesta răspunde cererilor care vin din View prin intermediul Controller-ului, Modelul comunicând doar cu Controller-ul. Este cel mai de jos nivel care se ocupă cu **procesarea** și **manipularea** datelor, reprezentând nucleul aplicației, fiind cel care realizează legătura cu baza de date.

Controller

Controller-ul este reprezentat de clase, fiind componenta care controleaza accesul la aplicatie.

In Controller:

- sunt procesate requesturile HTTP
- se citesc datele introduse de utilizator
- are loc procesarea prin trimiterea datelor catre Model - unde se executa operatiile
- se trimite raspunsul catre View

Asadar, Controller-ul comunica cu Modelul si cu View-ul.

Fiecare Controller contine asa numitele **Action-uri (Actions)**, reprezentate de metode. Aceste metode sunt publice, se definesc in interiorul unui Controller si sunt accesate in momentul in care are loc un request prin intermediul unei rute.

View (interfata cu utilizatorul)

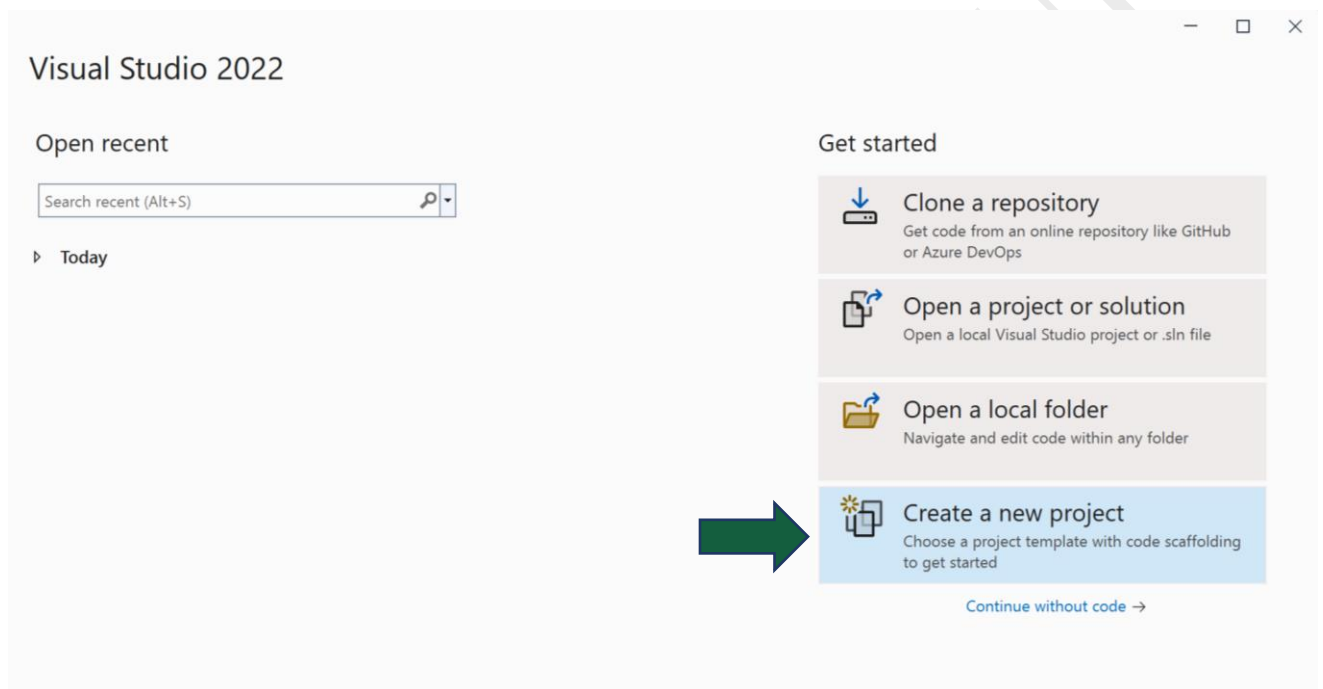
View-ul reprezinta interfata cu utilizatorul, fiind componenta arhitecturii MVC cu care utilizatorii interactioneaza prin intermediul browser-ului.

In View se afiseaza datele, adica inregistrarile din baza de date si informatiile generate de aplicatie.

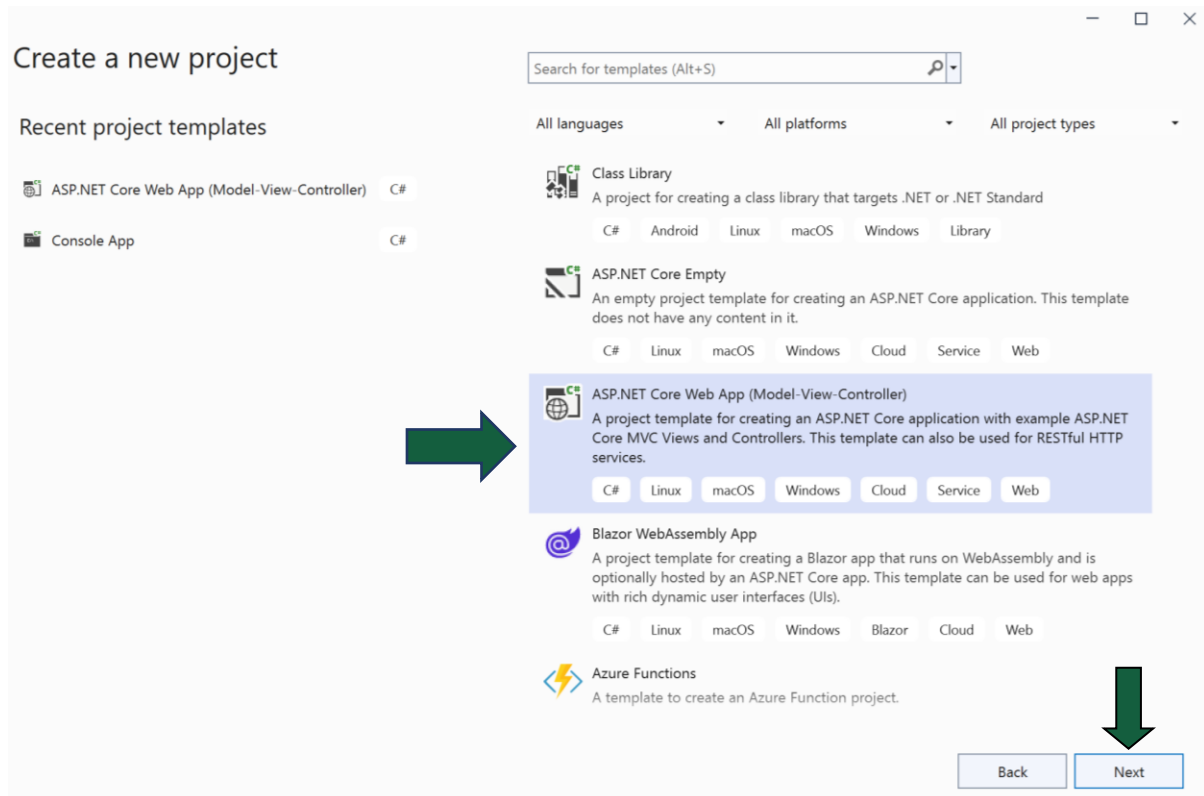
View-ul poate contine HTML static sau chiar HTML trimis din Controller (HTML dinamic). In cadrul arhitecturii MVC, View-ul comunica doar cu Controller-ul, iar cu Modelul indirect tot prin intermediul Controller-ului.

Crearea unei aplicatii in ASP.NET Core 6.0 (Visual Studio 2022)

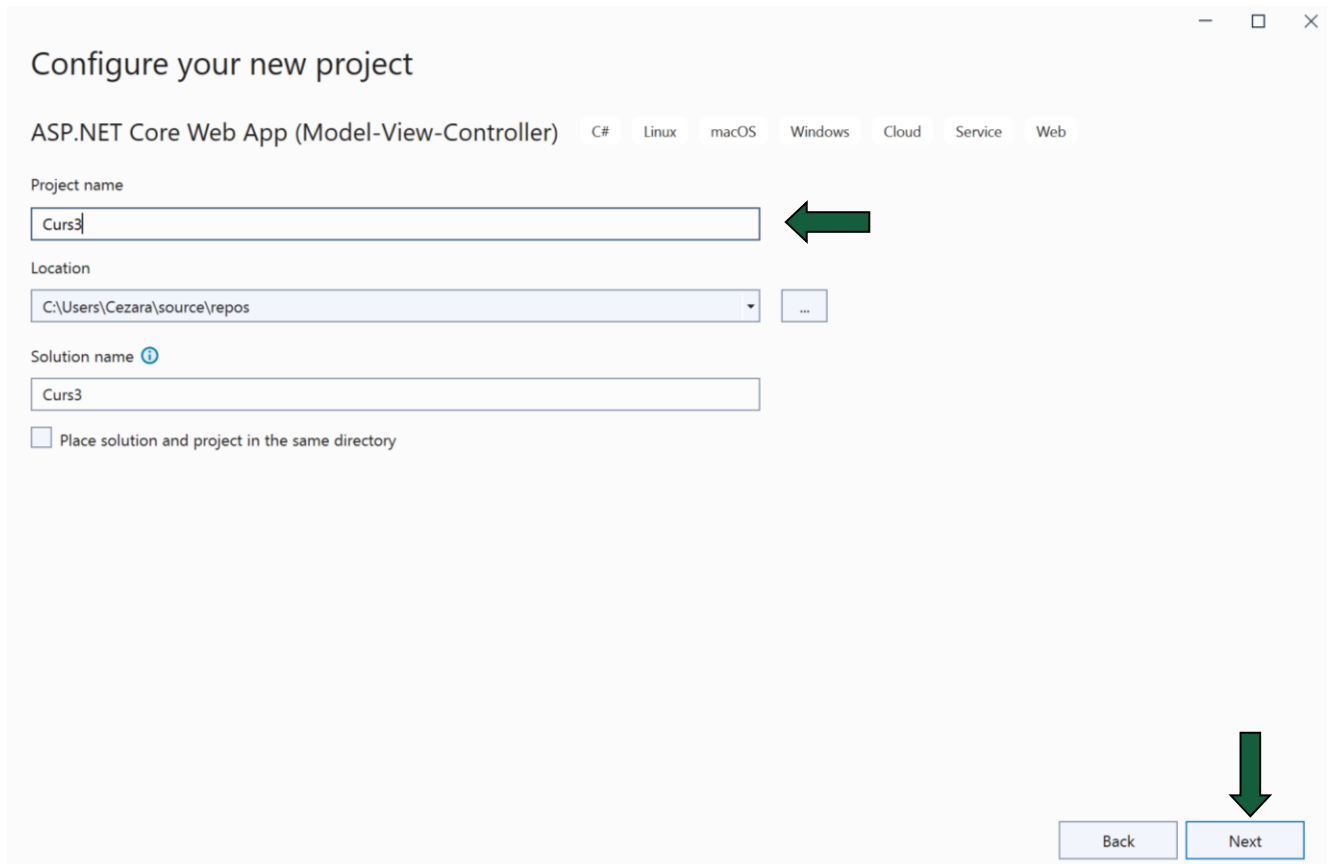
PASUL 1:



PASUL 2:



PASUL 3:



Configure your new project

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Project name

Curs3

Location

C:\Users\Cezara\source\repos

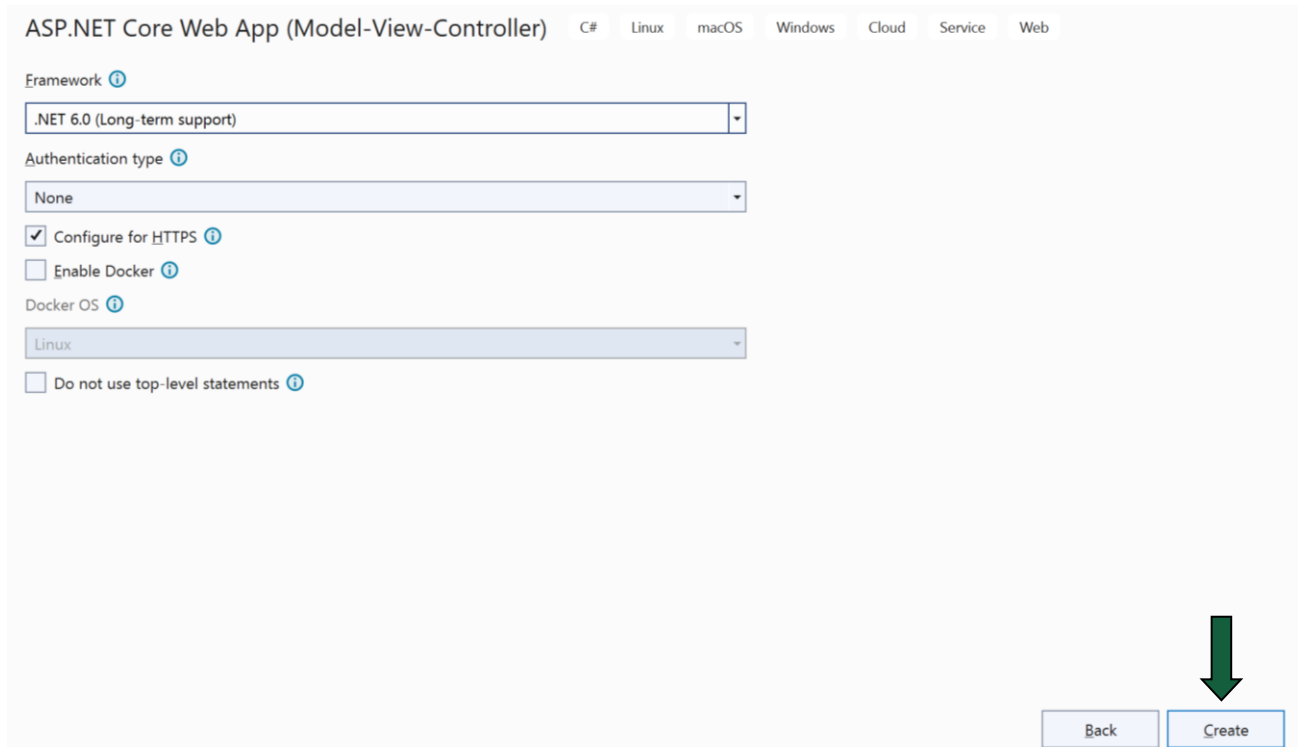
Solution name ⓘ

Curs3

☐ Place solution and project in the same directory

Back Next

PASUL 4:



ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Framework ⓘ
.NET 6.0 (Long-term support)

Authentication type ⓘ
None

☒ Configure for HTTPS ⓘ
☐ Enable Docker ⓘ

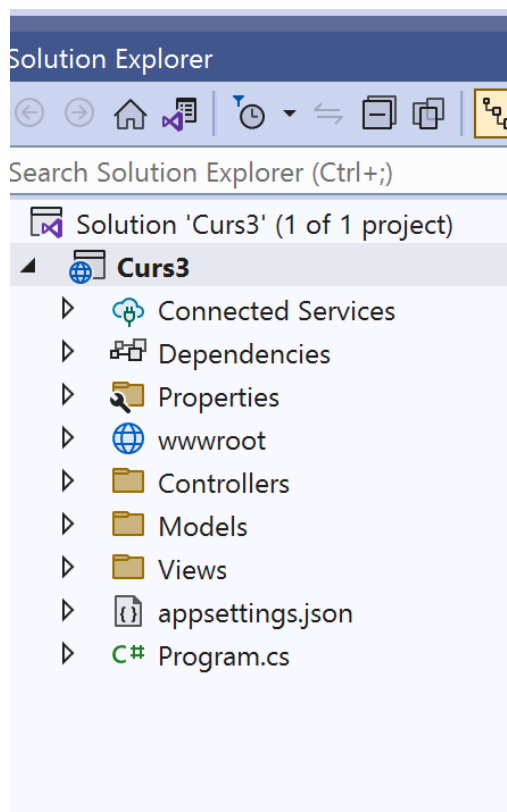
Docker OS ⓘ
Linux

☐ Do not use top-level statements ⓘ

Back Create

Structura unui proiect MVC – Sistemul de fisiere

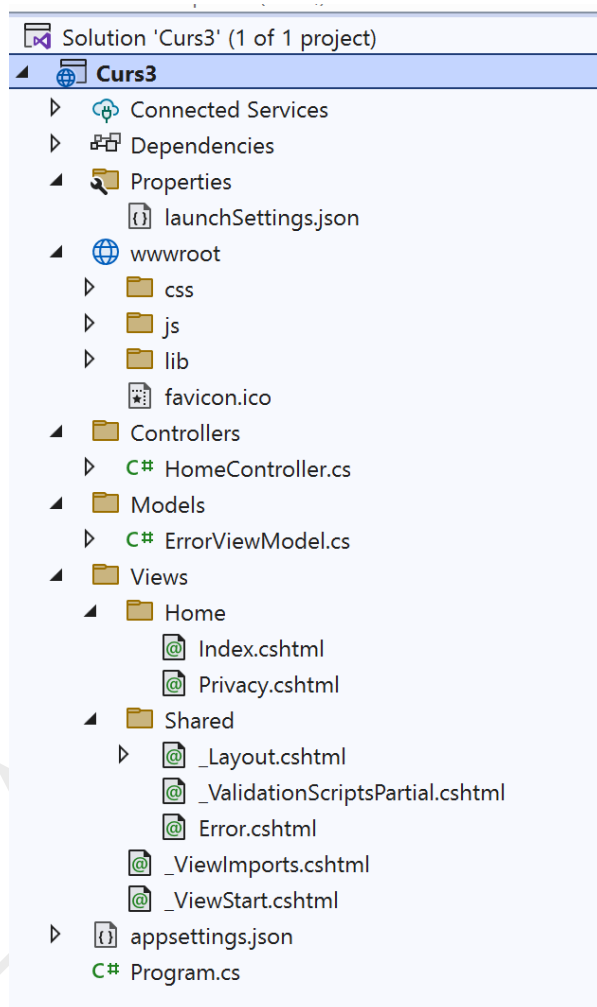
În imaginea următoare este prezentată structura unui proiect MVC, realizat în ASP.NET Core 6.0. Structura proiectului este reprezentată de sistemul de fișiere și foldere pe care le conține aplicația.



- **wwwroot** – contine fisierele statice precum librarii (ex: Bootstrap), imagini, scripturi (css, js);
- **Controllers** – include toate fisierele de tip Controller. Acestea sunt fisiere care contin cod **c#** si au extensia **.cs**. MVC impune ca numele tuturor controller-elor sa contina la final cuvantul **Controller**;
- **Models** – folderul contine modelele aplicatiei;
- **Views** – folderul contine fisierele de tip View (interfata cu utilizatorul) ale aplicatiei;
- **appsettings.json** – in acest fisier se afla setarile pentru configuratia aplicatiei. De exemplu: se utilizeaza pentru stocarea detaliilor de conectare la baza de date;
- **Program.cs** – este punctul de pornire al aplicatiei care se acceseaza imediat dupa ce aplicatia este rulata. De asemenea, in acest fisier se configureaza modulele aplicatiei: domeniul

aplicatiei (host), serverul web (IIS, Nginx, etc), modulul de autentificare, etc;

În imaginea următoare se pot observa toate folderele și fișierele existente într-un proiect nou creat.



Tot sistemul de fișiere o să fie studiat pe rând în cursurile următoare.

Sistemul de rutare

ASP.NET a introdus termenul de **Routing** (rutarea) pentru a elimina necesitatea maparii fiecarui URL cu un fisier fizic, asa cum era necesar in versiunea anterioara de ASP.NET Web Forms, unde fiecare URL trebuia sa coincida cu un fisier .aspx.

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, impreuna cu diferite metode ale acestora, in functie de URL-ul cerut de client.

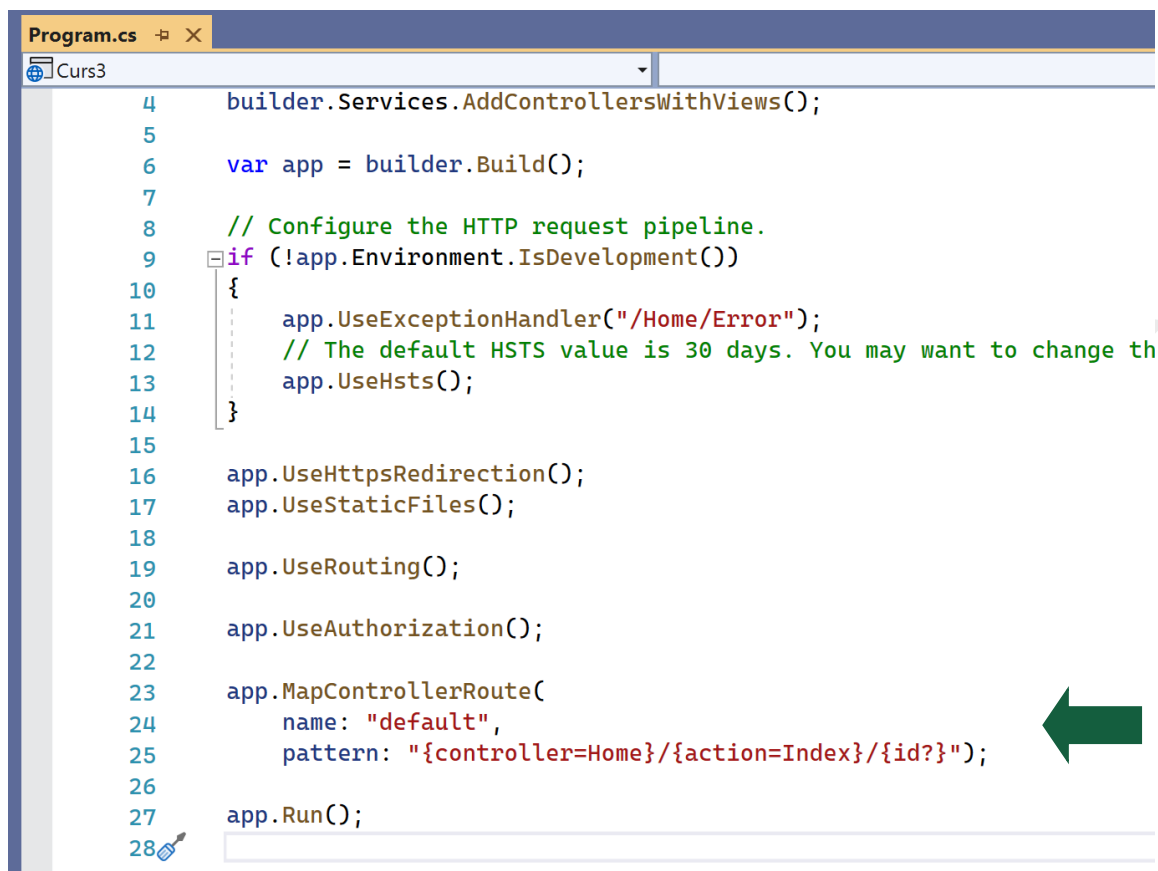
Astfel, pentru a accesa o anumita pagina, este necesar ca pentru aceasta sa existe o ruta definita, cat si un Controller care are o metoda (Action) care sa raspunda acestei resurse.

Formatul de baza al rutelor in ASP.NET este urmatorul:

`/[{NumeController}]/[{NumeActiune}]/[{Parametri}]`

Rutele se definesc in clasa **Program.cs**

In **Program.cs** se poate observa ruta default:



```

4     builder.Services.AddControllersWithViews();
5
6     var app = builder.Build();
7
8     // Configure the HTTP request pipeline.
9     if (!app.Environment.IsDevelopment())
10    {
11        app.UseExceptionHandler("/Home/Error");
12        // The default HSTS value is 30 days. You may want to change th
13        app.UseHsts();
14    }
15
16    app.UseHttpsRedirection();
17    app.UseStaticFiles();
18
19    app.UseRouting();
20
21    app.UseAuthorization();
22
23    app.MapControllerRoute(
24        name: "default",
25        pattern: "{controller=Home}/{action=Index}/{id?}");
26
27    app.Run();
28

```

Metoda **UseRouting()** se ocupa de configurarea rutelor in aplicatiei.

Definirea unei rute are urmatorul format:

```

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

```

Se observa ca variabila **app**, care este de tipul clasei **WebApplication** (clasa care se ocupa de configurarea HTTP pipeline si a rutelor), ofera mai multe metode necesare definirii rutelor.

Metoda **MapControllerRoute** adauga ruta default si primeste ca argumente 2 parametri:

- **name**: care reprezinta numele rutei (ex: name: "default")
- **pattern**: care reprezinta modelul URL-ului sau segmentele acestuia (ex: pattern: {controller=Home}/{action=Index}/{id?}).

In plus 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**, 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. Fiind urmat de "?" -> **{id?}**, inseamna ca este un parametru optional.

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

Asadar, in acest caz, metoda **app.MapControllerRoute()** mapeaza endpoint-uri cu pattern-ul:

{controller=Home}/{action=Index}/{id?}

Endpoint-ul din ASP.NET Core este reprezentat de Controller, fiind acea unitate responsabila cu procesarea request-urilor.

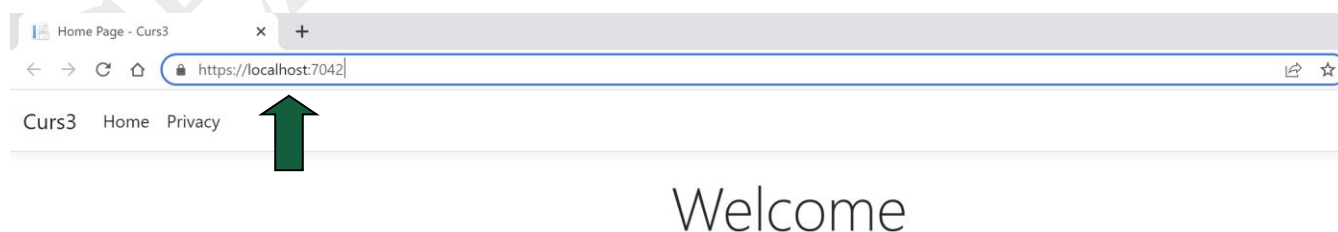
Privind exemplul anterior putem spune ca de fiecare data cand aplicatia primeste un URL care sa se potriveasca cu pattern-ul, atunci o sa se acceseze Controller-ul numit Home si Action-ul (metoda) Index. Acest lucru se intampla doar in cazul in care nu este specificat un alt Controller sau o alta metoda. In cazul in care sunt specificate explicit alte endpoints, atunci acelea o sa fie accesate.

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**”.

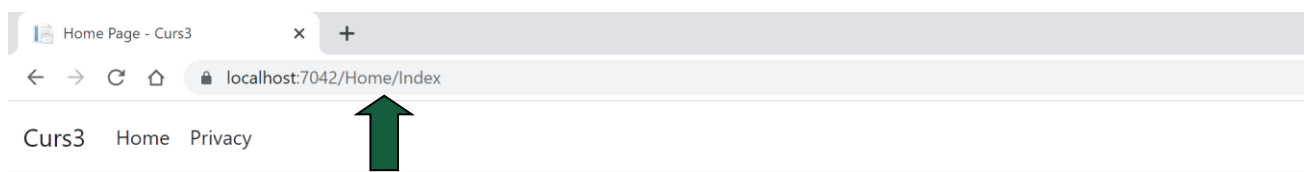
In concluzie, pattern-ul `{controller=Home}/{action=Index}/{id?}` se potriveste cu toate URL-urile de forma: /, /**Home**, /**Home/Index**, /**Students/Index**, /**Students/Index/5**, /**Students/Afisare**, etc.

In exemplul urmatoar se poate observa existenta Controller-ului **HomeController**. In cadrul clasei HomeController exista mai multe metode (Actions). Se poate observa prezenta metodei **Index()**. In momentul in care exista o metoda, trebuie sa existe si un View asociat. View-ul trebuie sa se numeasca identic cu metoda, deci **Index.cshtml**. Privind configuratia default a rutei, putem observa ca aceasta este ruta default, adica ruta care se acceseaza prima si afiseaza catre utilizatorul final interfata care se afla in Index.cshtml.

Dupa rulare se acceseaza in browser ruta default:

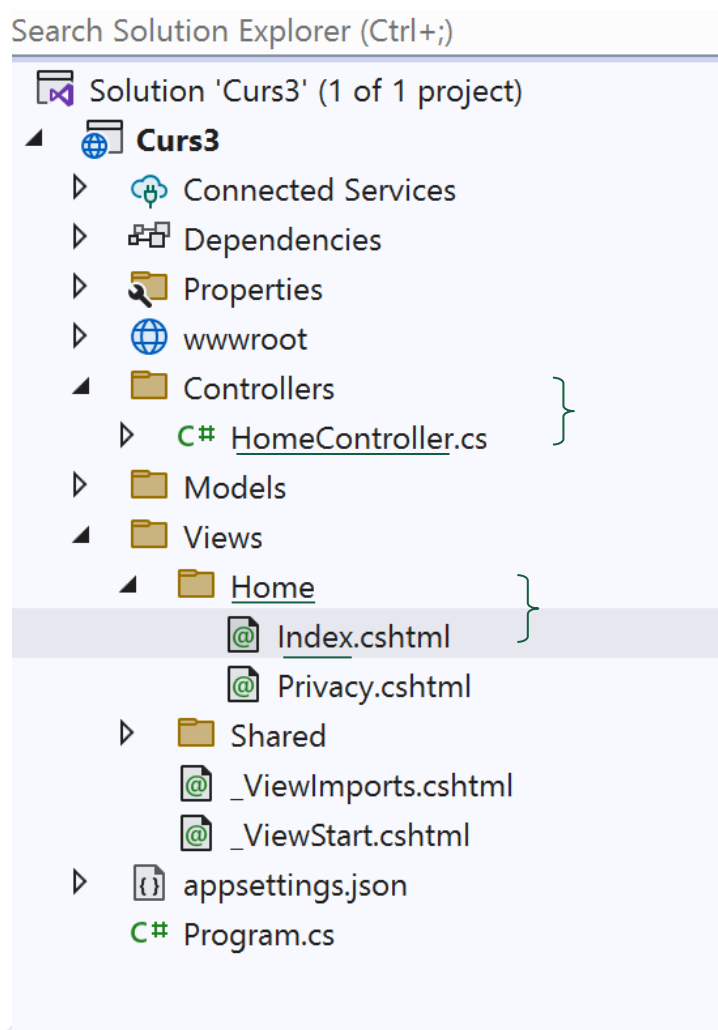


În cazul în care se accesează pagina prin scrierea întregului URL, atunci ruta o să aibă următorul format:

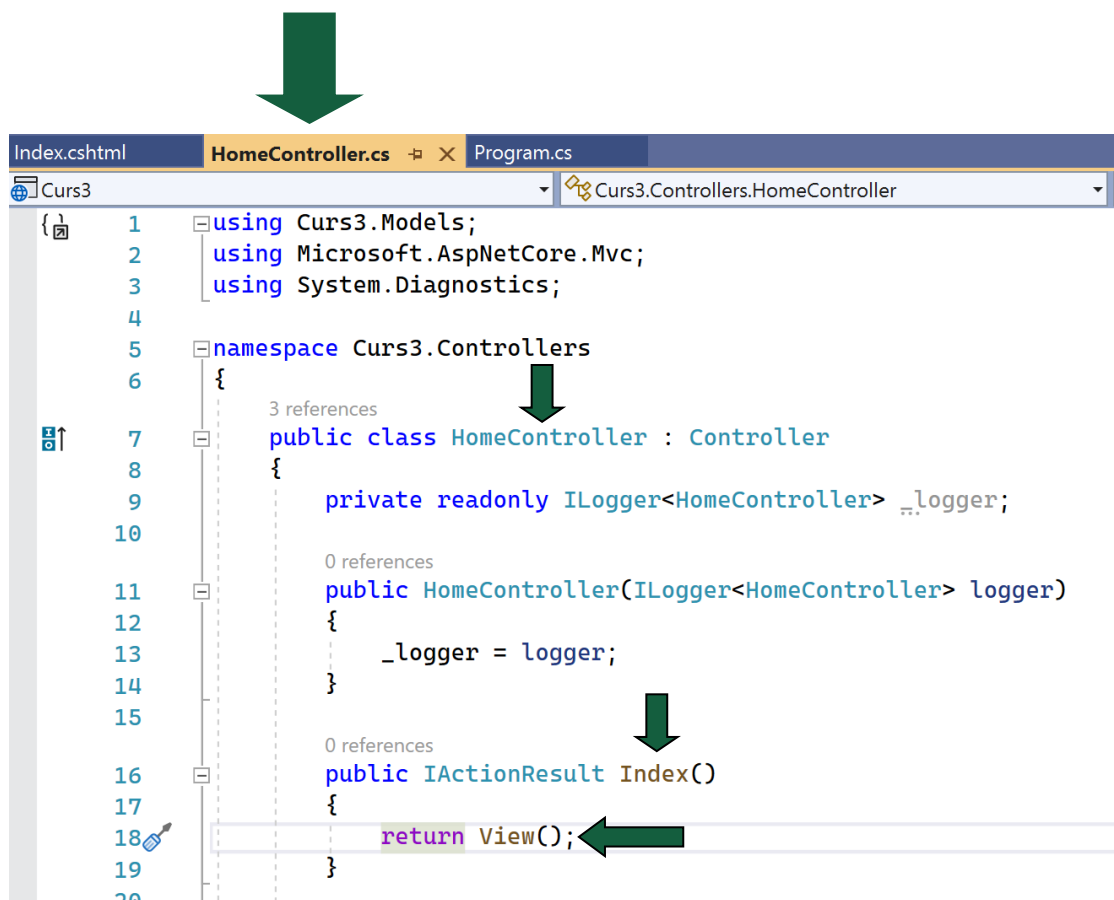


Welcome

In **Solution Explorer** se observa:



In **HomeController** exista metoda Index:

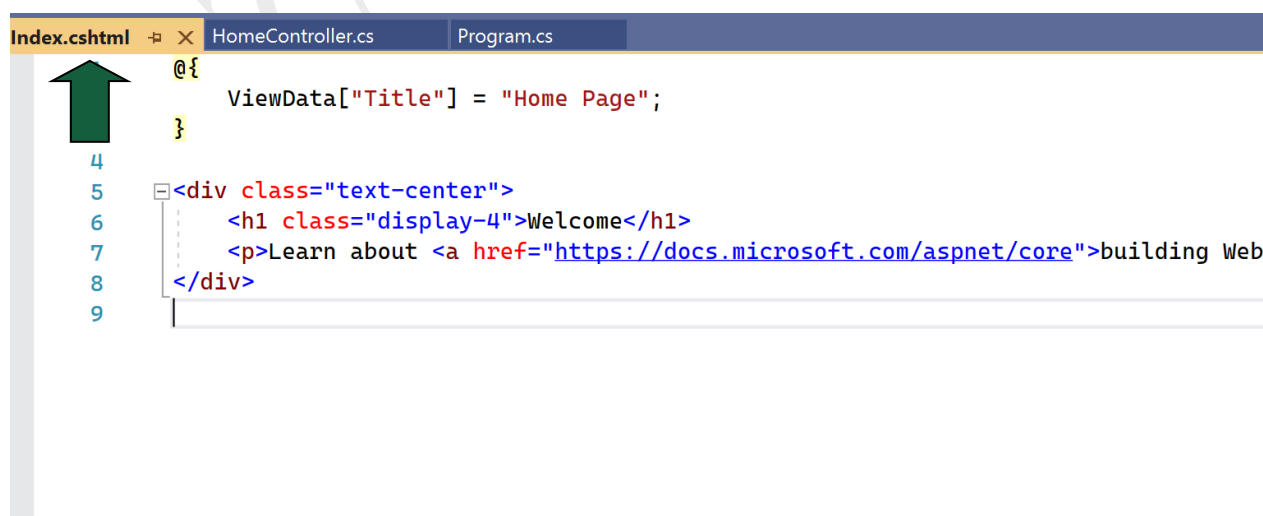


```

1  using Curs3.Models;
2  using Microsoft.AspNetCore.Mvc;
3  using System.Diagnostics;
4
5  namespace Curs3.Controllers
6  {
7      public class HomeController : Controller
8      {
9          private readonly ILogger<HomeController> _logger;
10
11         public HomeController(ILogger<HomeController> logger)
12         {
13             _logger = logger;
14         }
15
16         public IActionResult Index()
17         {
18             return View();
19         }
20     }

```

In **View** -> Folderul Home (asociat Controller-ului HomeController) -> Index.cshtml (pagina pentru codul html asociata metodei Index din Controller)



```

1  @{
2      ViewData["Title"] = "Home Page";
3  }
4
5  <div class="text-center">
6      <h1 class="display-4">Welcome</h1>
7      <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web
8  </div>
9

```

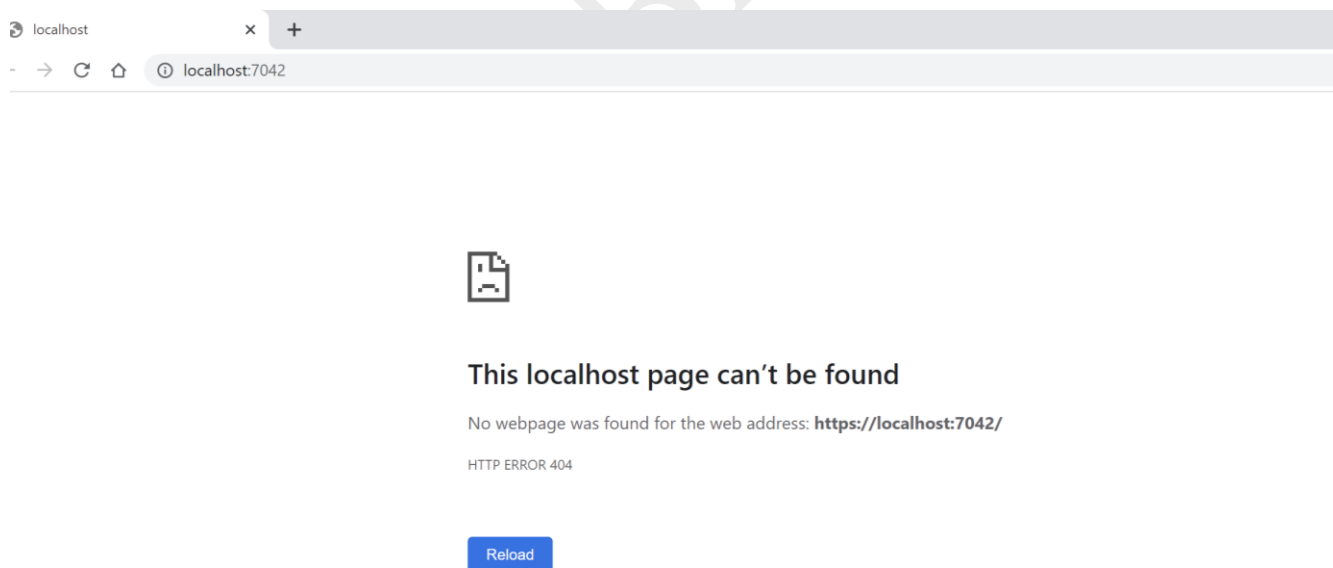

În cazul în care definiția rutei default nu conține și valorile implicite, atunci nu se pot accesa paginile aplicației.

De ex: se elimină ruta deja definită și se adaugă următoarea configurație:

```
/*
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
*/

app.MapControllerRoute(
    name: "default",
    pattern: "{controller}/{action}");
```

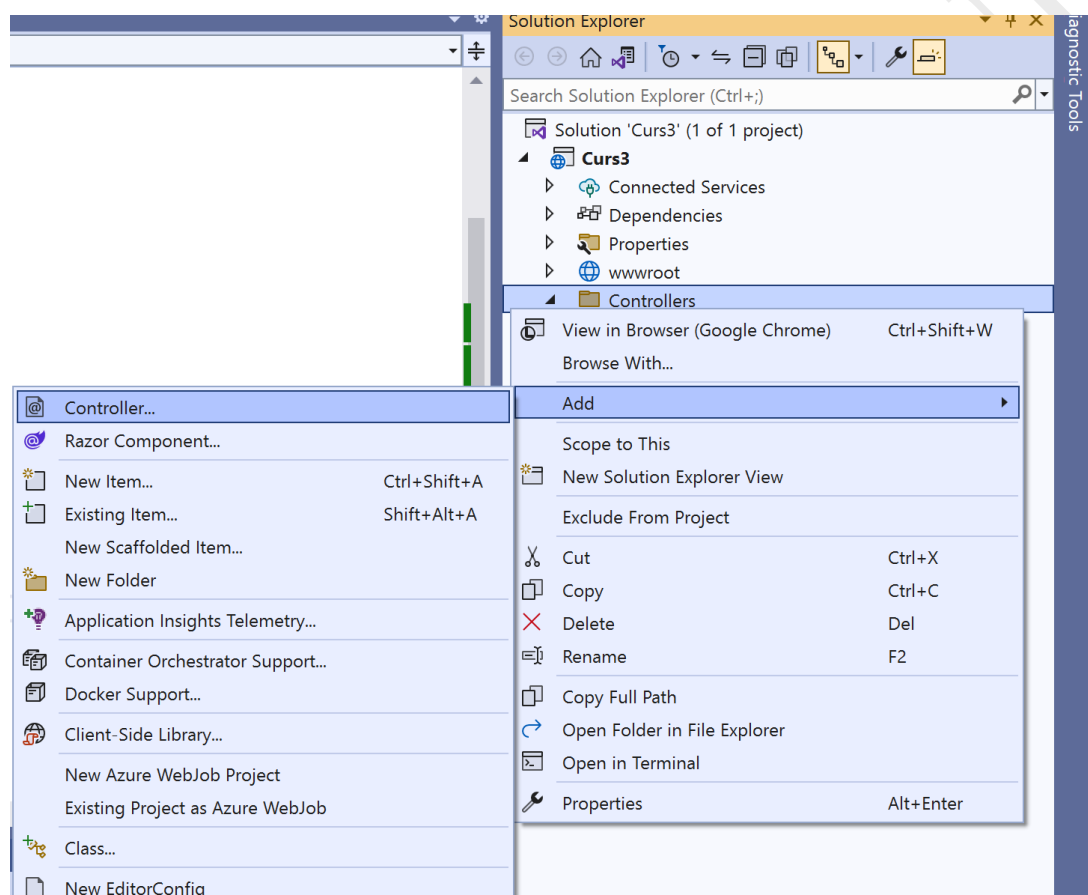
În momentul rularii apare mesajul **HTTP ERROR 404** deoarece nu găsește pagina. Acest lucru se întâmplă din cauza faptului că sistemul de rutare nu poate asocia ruta cu niciun Controller.



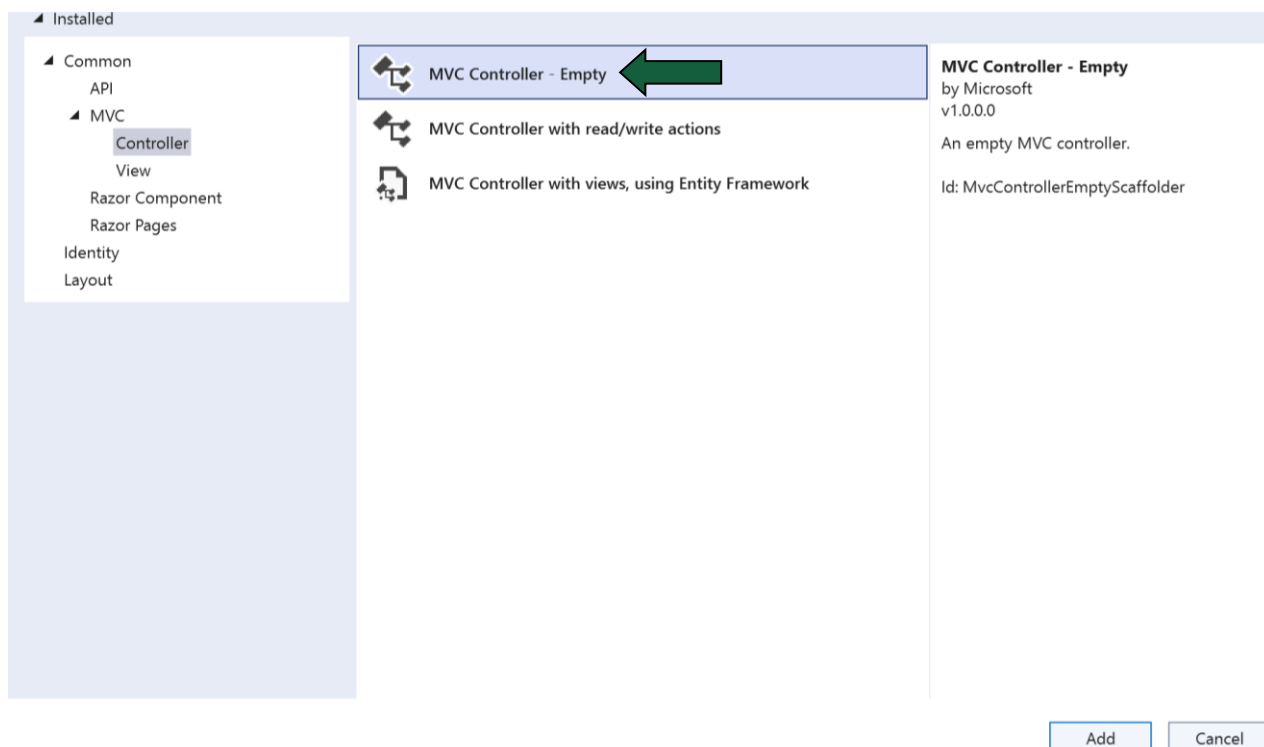
Exemple de implementare a rutelor:

Configurarea rutelor:

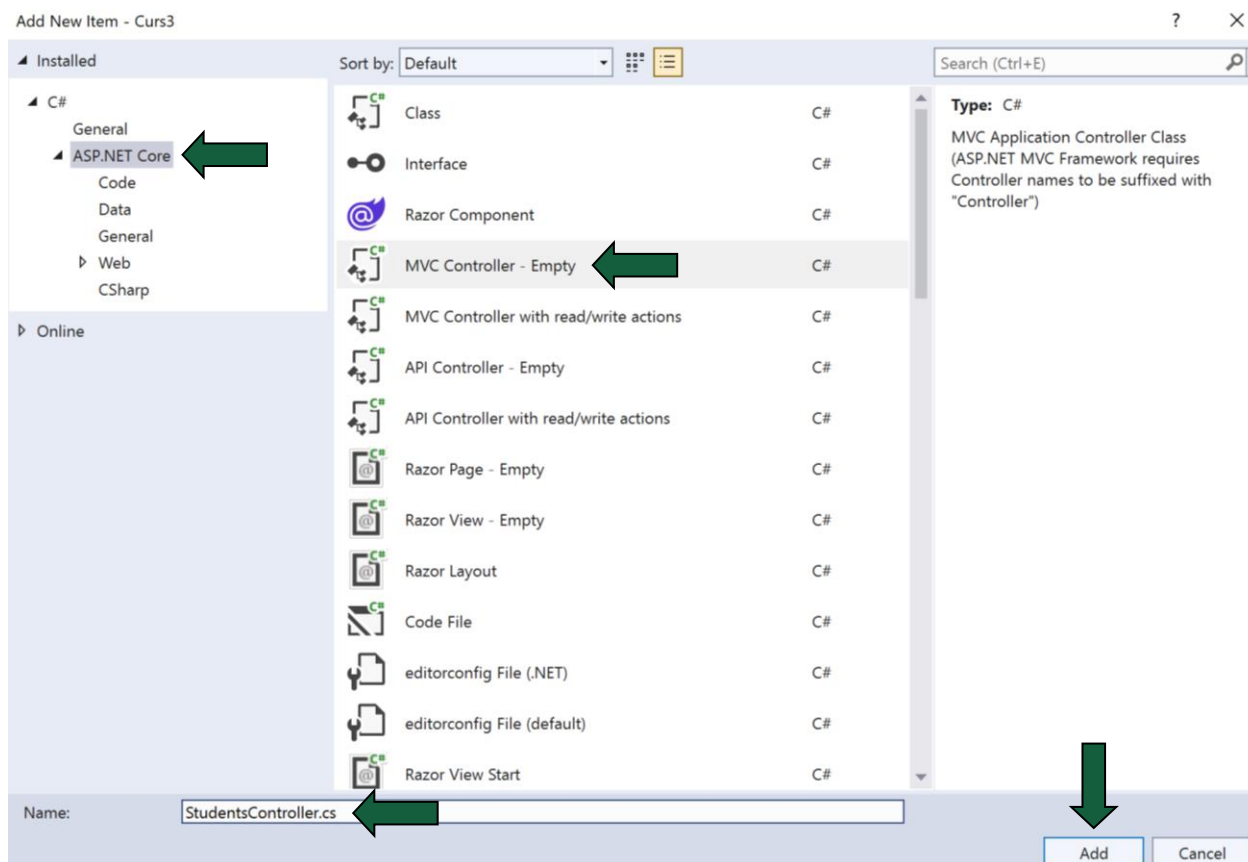
Pentru exemplele urmatoare se va crea un nou Controller, numit **StudentsController**. Click dreapta pe folderul Controller -> Add -> Controller.



Se selecteaza MVC Controller – Empty:



Se modifica numele noului Controller:



Pentru afisarea unui text, o sa se utilizeze metoda Index din cadrul Controller-ului **StudentsController**. In acest moment nu se va utiliza niciun View asociat.

```
public class StudentsController : Controller
{
    public string Index()
    {
        string response = "Hello World";
        return response;
        //return View();
    }
}
```

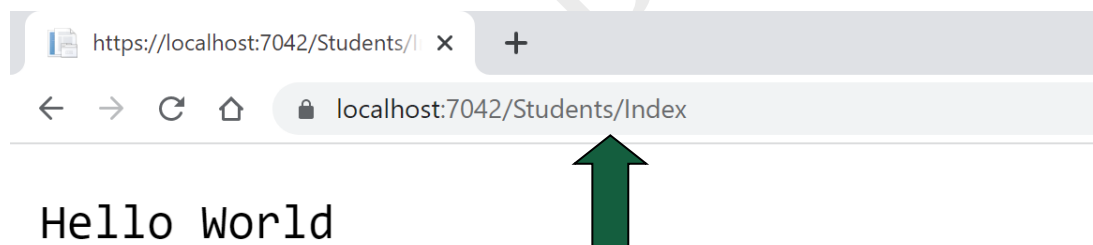
Tipul returnat de metoda Index a fost schimbat in **string** pentru a putea afisa in browser un simplu text. Astfel, valoarea de return devine valoarea variabilei **response**.

Dupa rulare, mesajul o sa fie afisat in browser, accesand ruta **/Students/Index** (**/NumeController/NumeActiune**). In momentul accesarii URL-ului, request-ul se trimite aplicatiei, dupa care se incearca maparea URL-ului cu o configuratie de ruta prezenta in Program.cs.

Astfel, pattern-ul o sa primeasca noile valori:

- controller = Students
- action = Index
- id = este optional si poate sa lipseasca

```
app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");
```



In continuare se defineste o ruta dupa cum urmeaza:

- ruta o sa contina doi parametri – name si id
- parametrul name o sa aiba o valoare implicita “World!”;
- parametrul id o sa fie optional

Metoda Index din Controller o sa afiseze mesajul “Hello World!”, folosind valoarea parametrului **name**, provenita din cadrul rutei.

Definirea rutei:

```
app.MapControllerRoute(
    name: "HelloWorld",
    pattern: "{controller=Students}/{action=Index}/{name=World!}/{id?}");
```

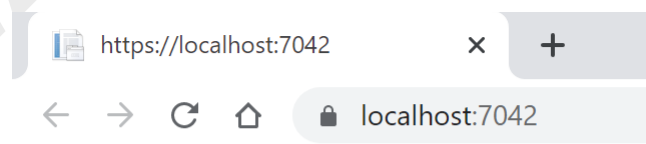
Implementarea metodei in Controller-ul **StudentsController**, metoda **Index**.

```
public string Index(string name, int? id)
{
    string response = "Hello " + name + " ";
    if (id != null)
    {
        response = response + "id = " + id;
    }

    return response;

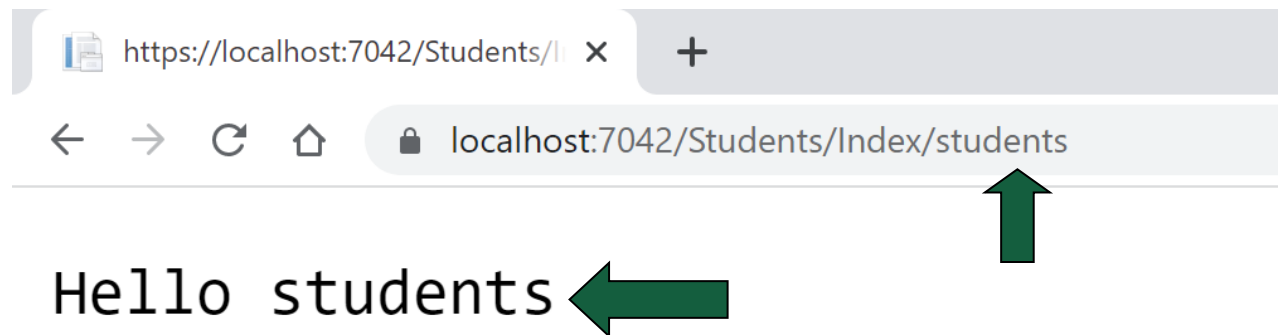
    //return View();
}
```

In momentul in care se ruleaza aplicatia, fara a introduce segmentele URL-ului, se observa ca pentru variabila **name** s-a transmis valoarea acesteia implicita: **World!**, afisandu-se mesajul Hello World!

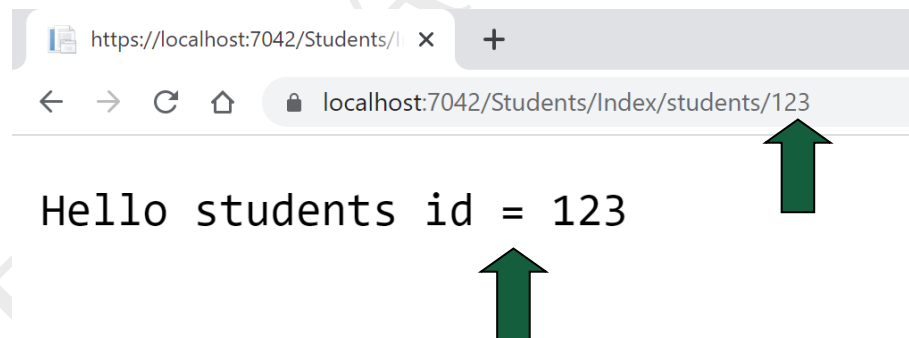


Hello World!

La adaugarea valorii pentru variabila **name** in URL, se observa cum aceasta a fost transmisa catre Controller si a fost afisata in pagina.



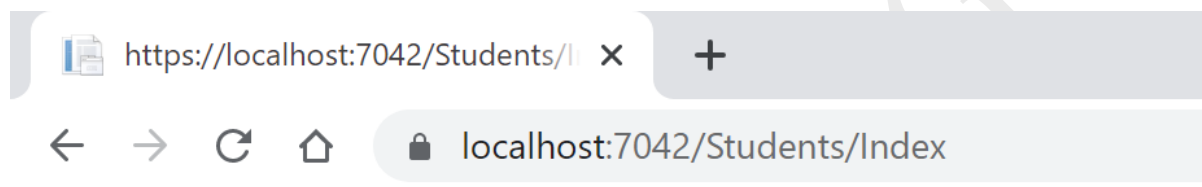
Cand parametrul optional **id** are valoare, secventa de cod specifica acestuia este executata si valoarea sa apare in raspunsul primit de la Controller:



OBSERVATIE:

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

De exemplu, daca ruta default este definita inaintea rutei creata in exemplul anterior, iar URL-ul de accesare este **/Students/Index**, atunci configuratia rutei default o sa se potriveasca si vom avea un rezultat ca cel de mai jos. Nu se afiseaza si valoarea implicita a parametrului **name**.

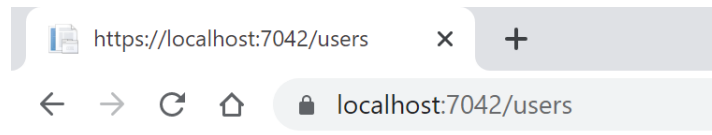


Definirea rutelor custom

Pentru fiecare **Controller** si **Actiune** in parte se pot defini si rute custom. De exemplu: daca se doreste accesarea Controller-ului Students si a metodei Index printr-un URL de forma: **/users** se poate implementa urmatoarea ruta:

```
app.MapControllerRoute(
    name: "Users",
    pattern: "users/{controller=Students}/{action=Index}/{name=world!!!}");
```

In varianta anterioara se poate accesa **/Students/Index/name** doar prin intermediul URL-ului: **/users** deoarece restul parametrilor vor prelua valorile implicite. Daca trebuie introdusa o alta valoare pentru parametrul **name**, atunci URL-ul trebuie sa fie: **users/Students/Index/abc**.



Hello world!!!

Astfel, ruta **/users** a accesat Controller-ul **StudentsController**, metoda **Index** cu paramentru implicit **name=world!!!**

In acelasi mod se poate proceda si pentru mai multe elemente in ruta:

```
app.MapControllerRoute(
    name: "HomePage",
    pattern: "Home/Page/{controller=Home}/{action=Index}");
```



Definitia anterioara a rutei functioneaza si pentru URL-uri de tipul:

```
/Home/Page/Students/Index
/Home/Page/Users/Read
```

Adica functioneaza pentru orice alt nume de Controller si Actiune.

Daca se doreste limitarea unei rute la o singura actiune, dintr-un singur Controller, atunci se foloseste urmatoarea varianta:

```
app.MapControllerRoute(
    name: "HomePage",
    pattern: "Home/Page",
    defaults: new { controller = "Home", action = "Index" });
```

Acest lucru se intampla deoarece pattern-ul nu are niciun parametru configurabil, iar ruta o sa mapeze doar Controller-ul Home si metoda Index.

De asemenea, varianta anterioara se foloseste si pentru cazul in care se doreste accesarea rutei doar printr-un URL custom. De ex: **/users**

```
app.MapControllerRoute(
    name: "Users2",
    pattern: "users/{name?}/{id?}",
    defaults: new { controller = "Students", action = "Index" });
```

OBSERVATIE:

⚠ 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.

Exista mai multe tipuri de constrangeri: de tip, length, max, min, range, regex.

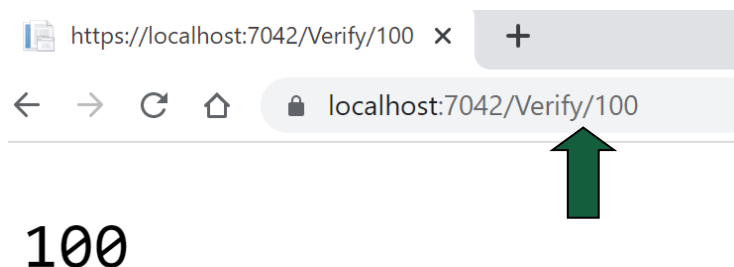
Un exemplu de constrangere este:

```
app.MapControllerRoute(
    name: "HomePage",
    pattern: "Verify/{id:range(10,100)}",
    defaults: new { controller = "Home", action = "Verify" });
```

Pentru un URL de tipul **/Verify/50** se cauta pattern-ul potrivit, dupa care se acceseaza parametrii specifici din **defaults** -> Controller-ul **Home** si metoda **Verify**. Daca parametrul **id** se afla in intervalul inchis 10, 100, atunci o sa acceseze ruta, iar in caz contrar o sa se afiseze 404 Not Found.

Pentru verificare se poate implementa in Controller-ul **Home**, metoda **Verify** care afiseaza id-ul in pagina.

```
public int Verify(int id)
{
    return id;
}
```

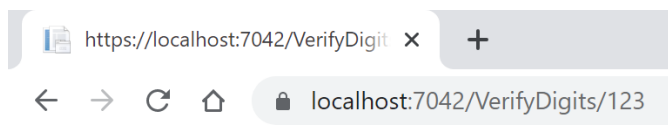


In cazul **expresiilor regulate** se configureaza ruta astfel:

```
app.MapControllerRoute(
    name: "HomePage",
    pattern: "VerifyDigits/{id:regex(\\d+)}",
    defaults: new { controller = "Home", action = "VerifyDigits" });
```

Ruta se acceseaza folosind URL-ul: **/VerifyDigits/id**, id-ul are o constrangere folosind o expresie regulata – se verifica daca este numar accesand apoi metoda **VerifyDigits** din Controller-ul **Home**.

```
public string VerifyDigits(int id)
{
    return "VerifyDigits " + id;
}
```



VerifyDigits 123