

Dezvoltarea Aplicatiilor Web utilizand ASP.NET Core MVC

Curs 4

Cuprins

Controller	2
Ce este Controller-ul	2
Crearea unui proiect	3
Adaugarea unui nou Controller	4
Controller-ul implicit – HomeController	7
Actions	9
Structura unei metode	9
Raspunsul actiunilor – ActionResult	10
Parametrii unei actiuni	13
Selectorii	13
ActionName	14
NonAction	15
ActionVerbs	15
Exemplu definire actiuni	17
Redirect in cadrul metodelor	19
Redirect	19
RedirectToRoute	19
RedirectToAction	20
RedirectPermanent/ RedirectToRoutePermanent/ RedirectToActionPermanent	21
Returnare HTTP Status Code	21

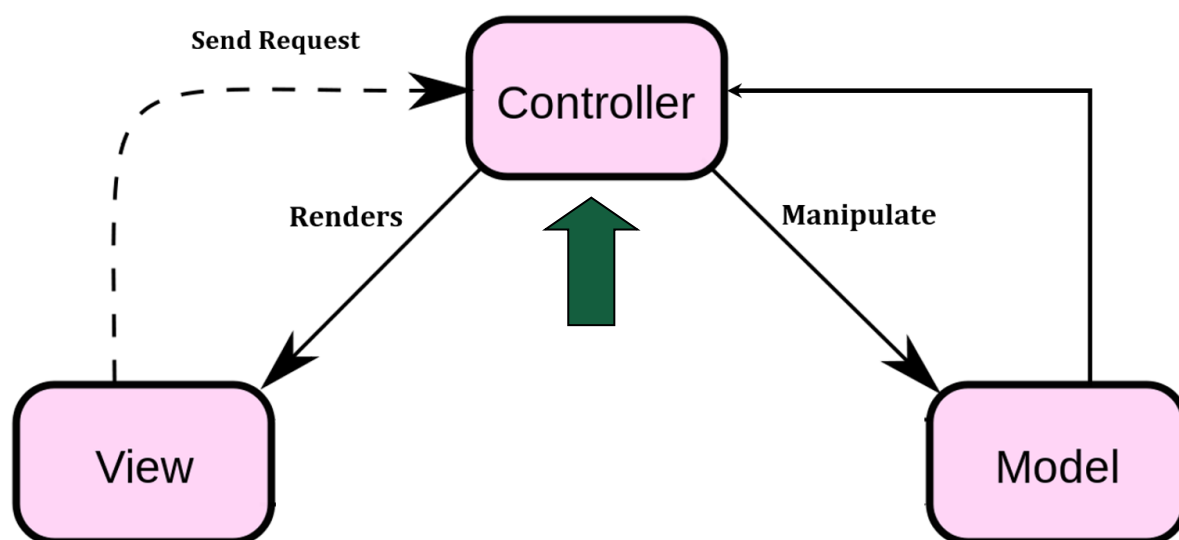
Controller

Ce este Controller-ul

În arhitectura MVC **Controller-ul** este componenta care procesează toate URL-urile aplicației. Controller-ul este o clasă, derivată din clasa de bază `Microsoft.AspNetCore.Mvc`. Această clasă conține metode publice numite **Acțiuni**. Metodele din Controller sunt responsabile pentru a procesa request-urile venite de la browser, pentru apelarea modelelor și procesarea datelor, cât și pentru a trimite răspunsul final către utilizator prin intermediul browser-ului.

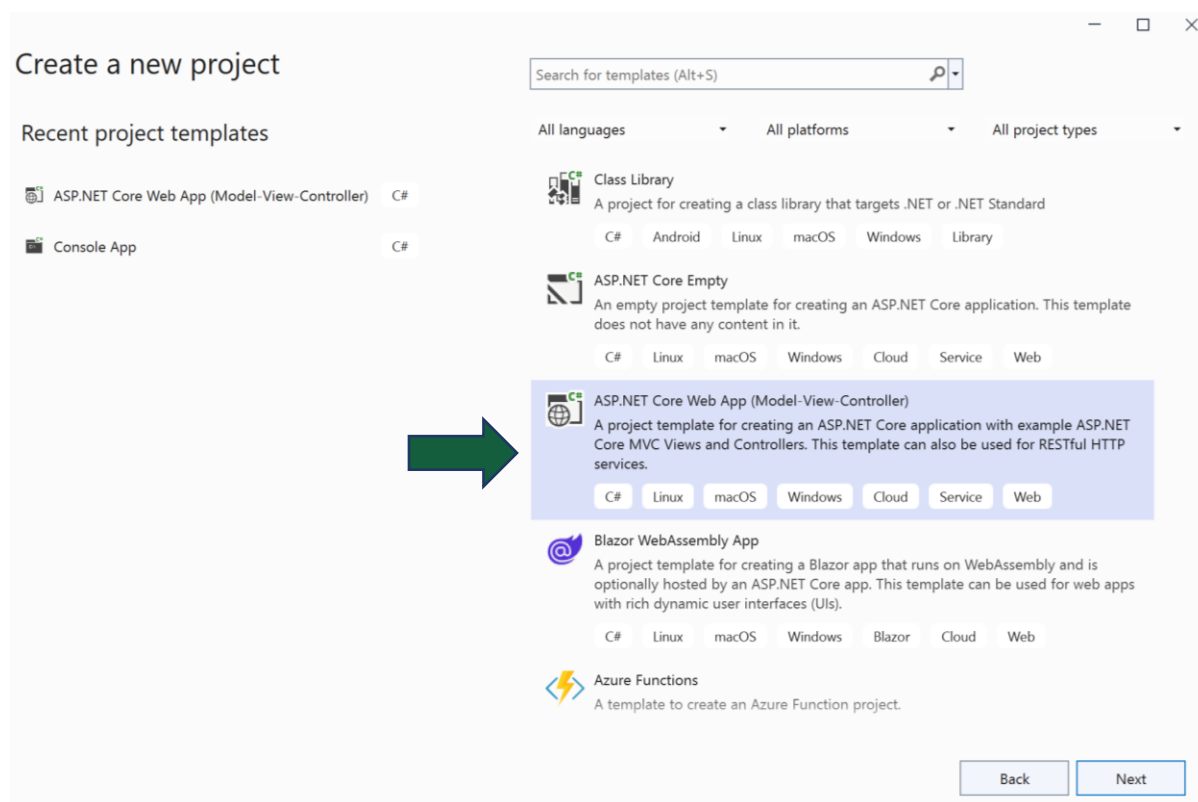
În ASP.NET MVC fiecare Controller este reprezentat de o clasă. Numele Controller-ului trebuie să se termine în cuvântul **Controller**. De exemplu, Controller-ul pentru pagina Home se poate numi **HomeController**.

Controller-ele trebuie să fie adăugate în cadrul folderului **Controllers** din proiectul ASP.NET.



Crearea unui proiect

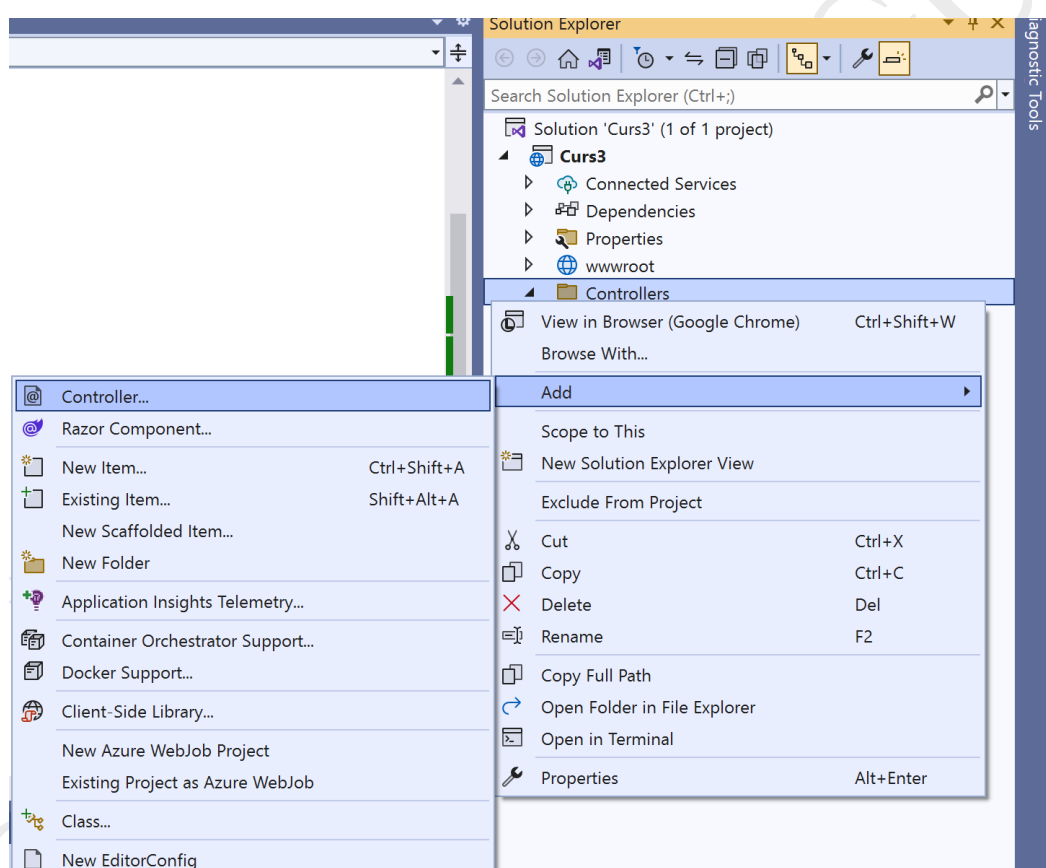
Pentru crearea unui nou proiect se utilizeaza optiunea ASP.NET Core Wep App (Model-View-Controller).



Adaugarea unui nou Controller

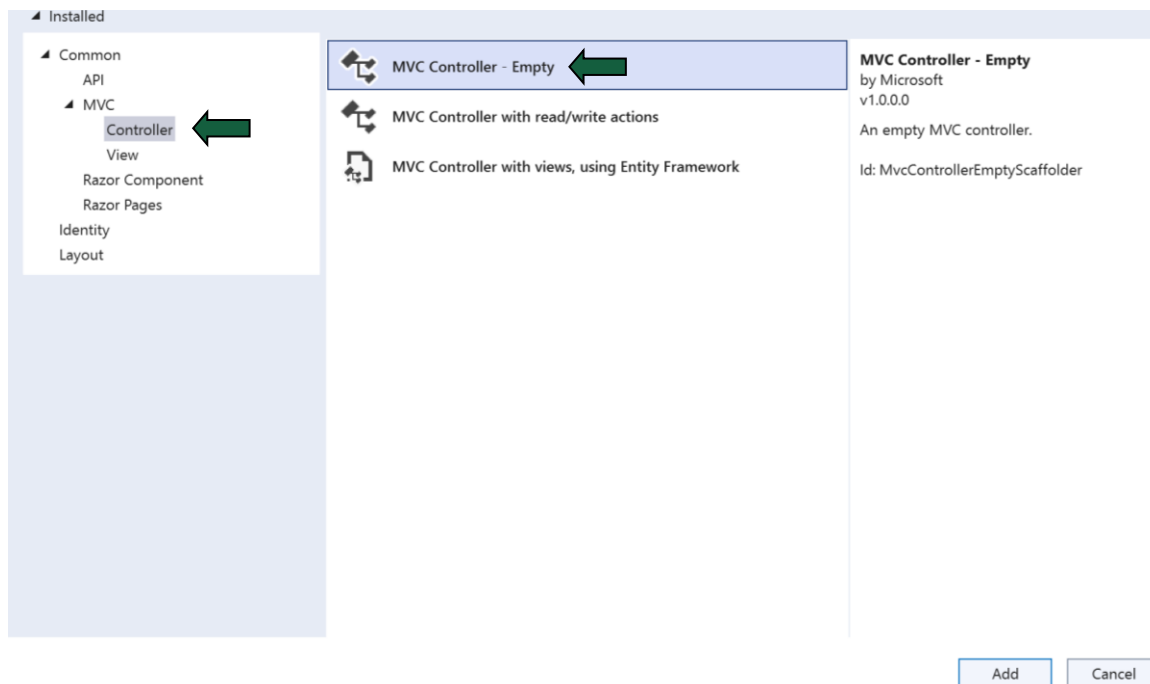
Pas 1:

Pentru a adauga un Controller se procedeaza astfel: click dreapta pe folderul **Controllers** si din meniul **Add** se selecteaza **Controller**.



Pas 2:

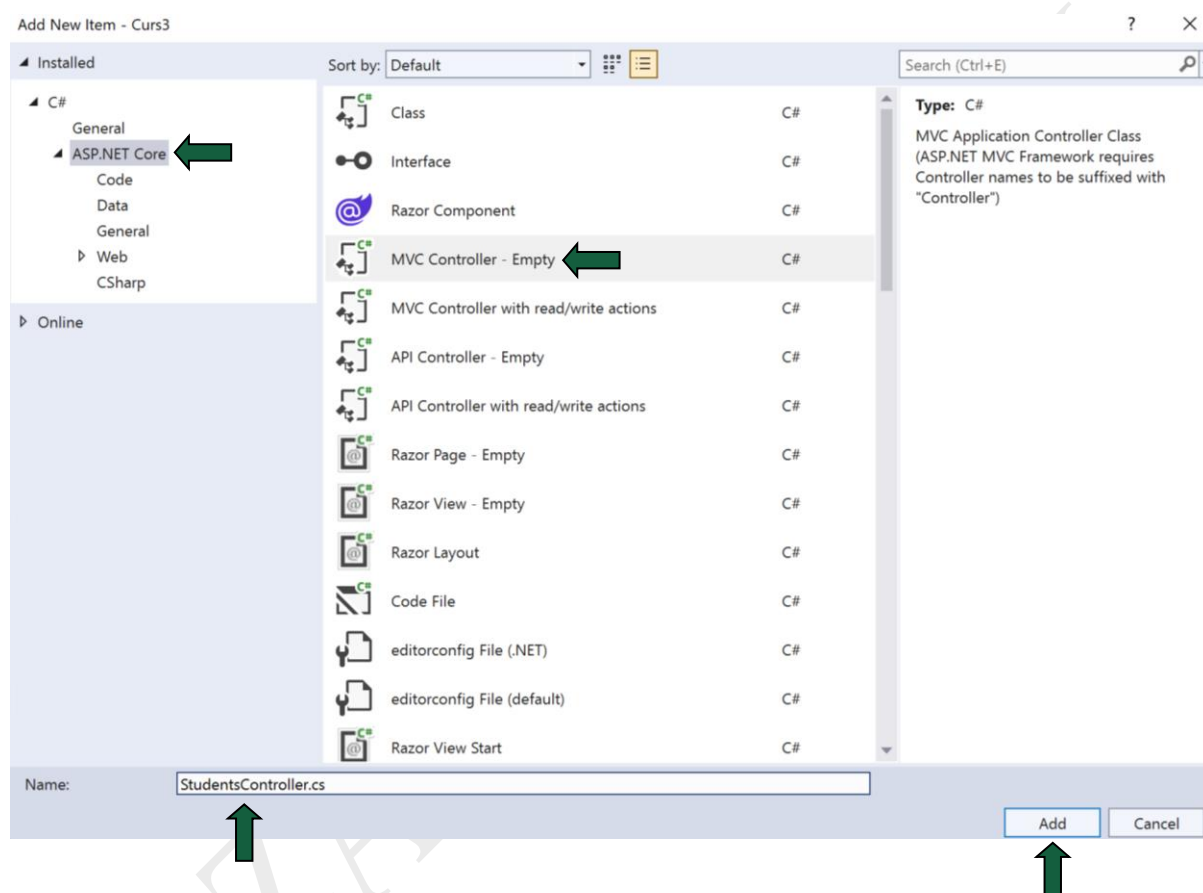
În fereastra aparuta se selecteaza **MVC Controller – Empty**:



Pas 3:

Se selecteaza din meniul aflat in partea stanga **ASP.NET Core**, dupa care optiunea **MVC Controller – Empty**.

Se adauga numele Controller-ului, nume care trebuie sa aiba sufixul Controller. De exemplu: **StudentsController**.



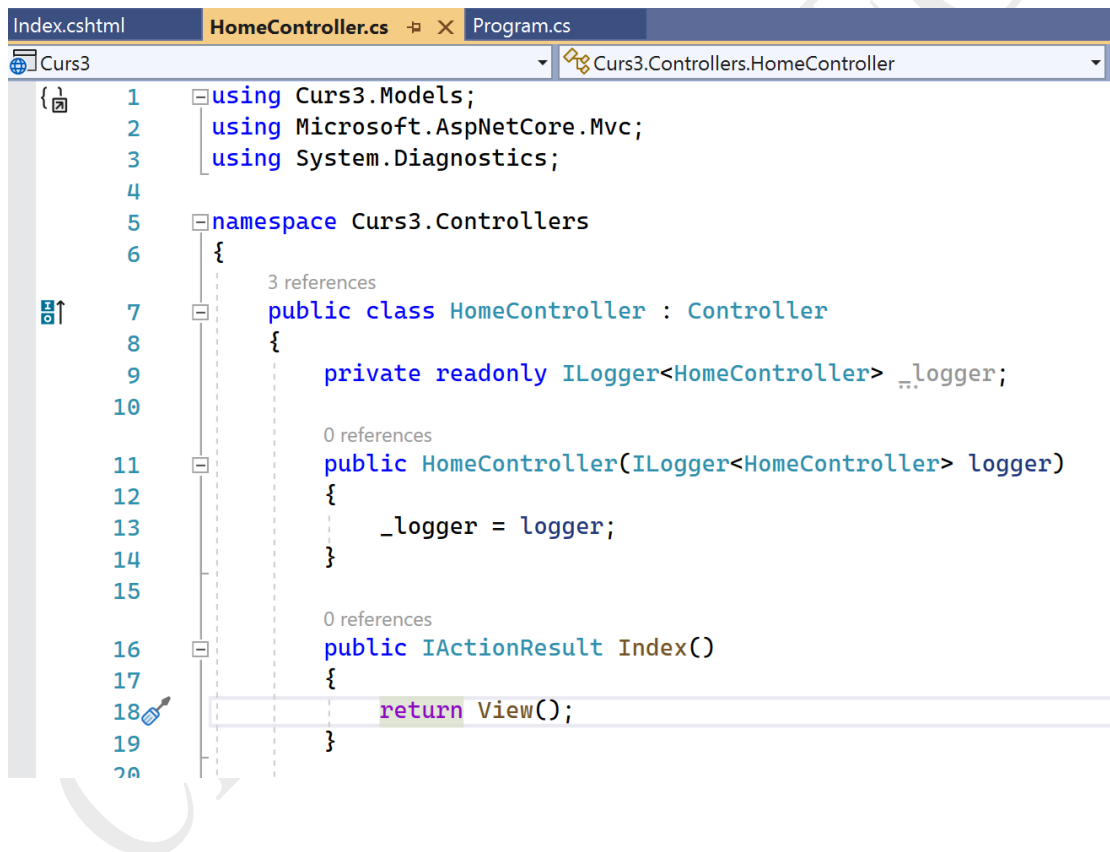
Controller-ul implicit – HomeController

În Folderul Controller se observa existenta unui Controller implicit. Acesta este creat automat în momentul în care se creează un nou proiect.

⚠ Observatie

Numele unui Controller trebuie să conțină sufixul Controller (ex: HomeController).

Controller-ul numit HomeController este de fapt o clasă care conține mai multe metode publice (Actions).

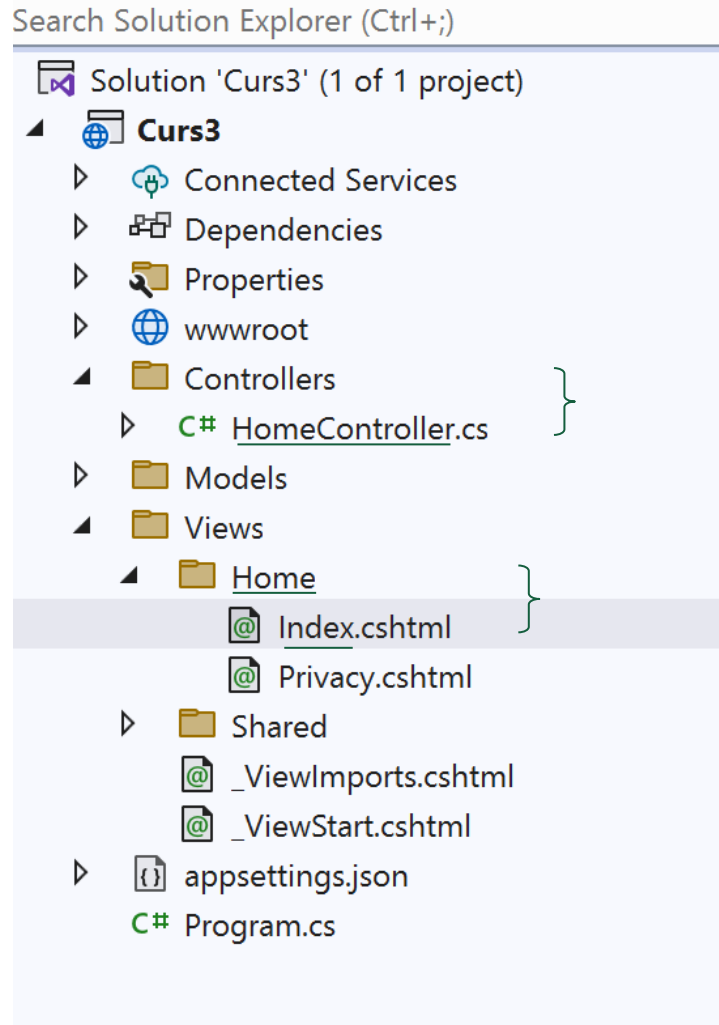


```

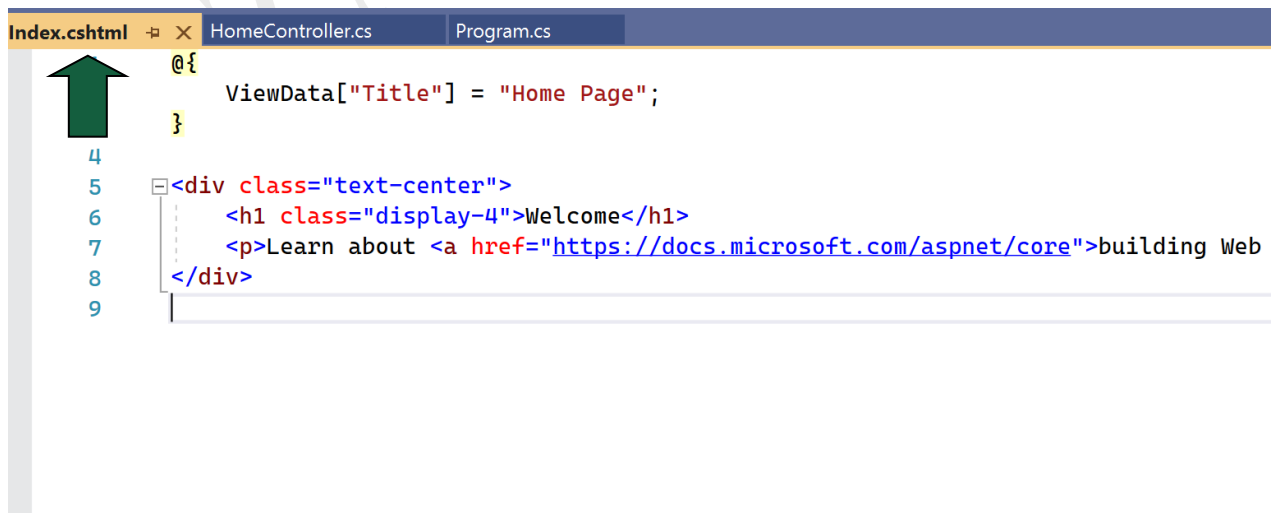
1  using Curs3.Models;
2  using Microsoft.AspNetCore.Mvc;
3  using System.Diagnostics;
4
5  namespace Curs3.Controllers
6  {
7      3 references
8      public class HomeController : Controller
9      {
10         private readonly ILogger<HomeController> _logger;
11
12         0 references
13         public HomeController(ILogger<HomeController> logger)
14         {
15             _logger = logger;
16
17         0 references
18         public IActionResult Index()
19         {
20             return View();
21         }
22     }

```

Se poate observa metoda **Index()** care returnează un View. Așadar, metoda Index o să aibă propriul View, numit exact ca metoda → **Index.cshtml**, aflat în folderul View → Folderul Home (deoarece acesta este numele Controller-ului) **(VEZI imaginea de mai jos)**



Index.cshtml



View-ul anterior, numit **Index.cshtml**, contine doua tipuri de cod:

- O expresie Razor care contine cod C# – un obiect de tip dictionar **ViewData** (vom studia in cursul destinat View-urilor).
Codul C# este inclus folosind simbolul @
- Elemente de html pentru interfata (UI)

Actions

Structura unei metode

```
public class StudentsController : Controller
{
    public IActionResult Index()
    {
        return View();
    }
}
```

In Controller-ul **StudentsController** se observa metoda Index() de tipul **IActionResult**. Aceasta este definita ca fiind **public** pentru a putea fi accesata de framework.

De asemenea, metodele *trebuie sa fie publice*, deoarece ele vor fi apelate in urma request-urilor primite din browser (request-uri HTTP). Pot exista si metode *private*, dar acestea se utilizeaza doar intern, nefiind accesibile prin intermediul URL-urilor.

Actiunile *pot fi supraincarcate* (adica sa existe metode cu acelasi nume, in aceeasi clasa) doar daca se utilizeaza cu parametri de intrare diferiti (tip de date, numar).

Tipul returnat **IActionResult** este o interfata, reprezentand raspunsul actiunii trimis de catre Controller la browser, fiind cel mai frecvent tip utilizat deoarece intotdeauna se va apela un View pentru a afisa informatiile catre utilizatorul final.

Metoda **View()** din interiorul actiunii este definita in clasa abstracta de baza **Controller** si este de tipul **ViewResult : ActionResult**.

Deoarece clasa **ActionResult** implementeaza interfata **IActionResult** este suficient ca tipul de return al metodei sa fie **IActionResult**.

Raspunsul actiunilor – ActionResult

Framework-ul MVC include diverse tipuri de rezultat care pot fi returnate prin intermediul **ActionResult**. Aceste clase de rezultat pot fi tipuri de date diferite: html, fisiere, string-uri, json, obiecte, etc.

Posibilele tipuri de date returnate pentru **ActionResult** sunt:

- **ViewResult** – aceasta clasa returneaza continut HTML
- **EmptyResult** – aceasta clasa returneaza un raspuns gol – pagina returnata nu are niciun continut (ex: returneaza status code 200 -> adica request-ul a fost executat corect, dar raspunsul este gol)
- **ContentResult** - poate fi folosit pentru a returna text
- **FileContentResult** / **FileStreamResult** - reprezinta continutul unui fisier (folosit pentru descarcarea fisierelor)
- **JsonResult** – reprezinta un JSON care poate fi cerut prin AJAX sau alte metode
- **RedirectResult** – reprezinta redirectionarea catre un nou URL

- **RedirectToRouteResult** – reprezinta redirectionarea catre o alta actiune in acelasi Controller sau in alt Controller
- **PartialViewResult** – returneaza HTML-ul dintr-un partial
- **StatusCodeResult** – returneaza un raspuns de tip: BadRequest (400), Unauthorized (401), Forbidden (403), NotFound (404).

ActionResult este clasa de baza a tuturor claselor enumerate mai sus. ActionResult implementeaza interfata IActionResult. Deci, indiferent de raspunsul folosit, actiunea poate sa aiba tipul de raspuns **IActionResult**.

Pentru a returna tipurile de date mentionate mai sus clasa de baza Controller are urmatoarele metode implementate:

- ViewResult -> View()
- ContentResult -> Content() – primeste ca parametru un string care va fi afisat in browser
- FileContentResult/FilePathResult/FileStreamResult -> File()
- JsonResult -> Json() – primeste ca parametru orice tip de date si va returna un raspuns sub forma JSON (se va serializa parametrul primit sub forma unui string JSON)
- RedirectResult -> Redirect() – primeste ca parametru un URL (in format string) si va redirectiona browser-ul catre acel URL
- RedirectToRouteResult -> RedirectToRoute() – primeste ca parametru un **nume de ruta** (care este definita in fisierul Program.cs) si va redirectiona browser-ul catre acea ruta
- PartialViewResult -> PartialView() – returneaza continutul unui partial

Avantajul utilizarii ca tip de date de return **IActionResult** este acela ca se poate utiliza oricare dintre tipurile de raspuns enumerate mai sus fara a schimba tipul de date al actiunii.

De exemplu, pentru a descarca un fisier prin intermediul unei rute putem folosi urmatoarea secventa de cod:

```
public IActionResult Download()
{
    byte[] fileBytes =
        System.IO.File.ReadAllBytes(@"c:\folder\myfile.ext");
    string downloadName = "myfile.ext";
    return File(fileBytes,
        System.Net.Mime.MediaTypeNames.Application.Octet, downloadName);
}
```

Pasul 1: se citește fisierul ca secvență de bytes de la o cale cunoscută

Pasul 2: setăm un nume de fisier pentru fisierul care va fi descărcat – downloadName

Pasul 3: returnăm metoda File care primește 3 parametri:

- Array-ul de bytes care stochează conținutul fisierului
- MIMEType-ul (MediaType) fisierului descărcat (ex: pentru fișiere **.mp3** -> **audio/mpeg**; pentru **imagini de tip JPEG** -> **image/jpeg**; pentru **imagini de tip PNG** -> **image/png**) pentru encodarea corectă a fisierului salvat
- Numele fisierului care va fi salvat pe client

Parametrii unei actiuni

Fiecare actiune poate sa primeasca parametrii unei rute in cadrul semnaturii acesteia. Parametrii rutei pot fi de orice tip (string, int, float sau chiar de tipul clasei unui Model).

```
public IActionResult Show(Student student)
{
    // obiectul student de tipul Modelului Student va
    // contine informatiile unui student din baza de date
}
```

In exemplul anterior, pentru ruta “/Students/Show/2” obiectul student va fi instantiat in mod automat cu valorile studentului cu ID-ul 2 din baza de date, prin intermediul modelului. Acest lucru va fi detaliat in cursul referitor la Model.

!OBSERVATIE:

Numele parametrilor definiti in semnatura Actiunii **trebuie sa fie acelasi** cu numele parametrilor definiti in **Program.cs**. Diferenta dintre numele parametrilor va conduce catre nerezolvarea acestora (vor avea valoarea null).

Selectorii

Framework-ul ASP.NET Core MVC ofera posibilitatea adaugarii unor attribute actiunilor, pentru a ajuta sistemul de rutare sa aleaga actiunea corecta in momentul procesarii unui request. Aceste attribute se numesc **Selectorii** si sunt:

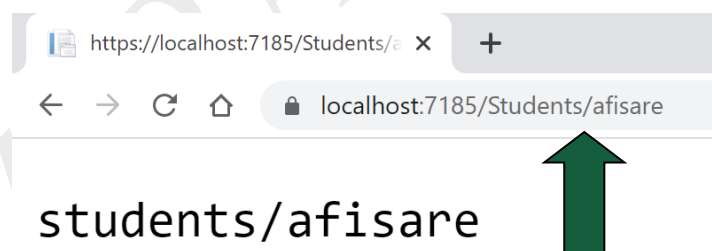
- ActionName
- NonAction
- ActionVerbs

ActionName

Atributul **ActionName** ofera posibilitatea de a alocu un nume unei actiuni, care este diferit de numele acesteia. De exemplu, daca avem o metoda numita **Index** in Controller-ul **StudentsController** putem redenumi aceasta actiune prin intermediul atributului ActionName astfel:

```
[ActionName("afisare")]
public IActionResult Index()
{
    return Content("students/afisare");
}
```

Inainte de adaugarea atributului ActionName pagina se putea accesa prin URL-ul: **/Students/Index**. Dupa adaugarea atributului ActionName cu valoarea "afisare" pagina poate fi accesata prin intermediul URL-ului **/Students/afisare**. Astfel, acest atribut ne ofera posibilitatea rescrierii numelui actiunii. Acest lucru se intampla fara a aduce modificari fisierului Program.cs.



NonAction

Atributul **NonAction** indica faptul ca o metoda a unui Controller nu este o actiune. Acest atribut se foloseste in momentul in care dorim ca o metoda publica a unui Controller sa nu poata fi accesata prin intermediul unei rute.

Exemplu:

```
[NonAction]
public Student GetStudent(int id)
{
    return ...;
}
```

Aceasta metoda nu poate fi accesata prin intermediul unei rute, desi este publica. In schimb, ea poate fi accesata din celelalte metode ale aceluiasi Controller sau ale unui alt Controller.

ActionVerbs

Atributul **ActionVerbs** este folosit in momentul in care se doreste accesarea unei actiuni in functie de **verbul HTTP**. De exemplu, se pot defini doua actiuni cu acelasi nume, inasa care raspund la un verb HTTP diferit si au parametri diferiti.

Verbele HTTP acceptate sunt urmatoarele: **GET, POST, PUT, PATCH, HEAD, OPTIONS** si **DELETE**.

!OBSERVATIE:

In cazul in care atributul ActionVerbs este omis, verbul default folosit este GET.

Aceste verbe sunt folosite in urmatoarele contexte:

- **GET**: este folosit in accesarea unei resurse (cererea unei pagini de la server) [clientul cere de la server](#)
- **POST** – este folosit in crearea unei resurse sau trimiterea datelor la server prin intermediul unui formular [serverul cere de la client](#)
- **PUT/PATCH** – verbul este folosit pentru modificarea (totala sau partiala) a unei resurse. De exemplu: cand se editeaza o intrare deja existenta in baza de date, se foloseste unul dintre aceste verbe
- **DELETE** – verb folosit pentru stergerea unei resurse
- **HEAD** – este identic cu GET, dar returneaza doar antetele pentru raspuns, nu si continutul raspunsului. De obicei se foloseste pentru a verifica daca exista o resursa sau daca poate fi accesata
- **OPTIONS** – returneaza metodele HTTP acceptate de server pentru o adresa URL specificata

!OBSERVATIE:

In ASP.NET Core MVC pentru editare (PUT) si stergere (DELETE) se foloseste tot POST → [HttpPost].

Exemplu definire actiuni

Exemplu de definire a actiunilor (metodelor) folosind **verbele HTTP** corespunzatoare:

```
public class StudentsController : Controller
{
    // GET: lista tuturor studentilor
    public IActionResult Index()
    {
        return View();
    }

    // GET: vizualizarea unui student
    public IActionResult Show(int id)
    {
        return View();
    }
}
```

Aceste doua actiuni, **Index()** si **Show()** afiseaza informatii despre studenti. **Index** va afisa **lista tuturor studentilor**, iar **Show** va afisa **informatii despre un singur student** in functie de ID-ul primit ca parametru. Deoarece aceste pagini afiseaza informatii si nu trimit nimic la server, vom folosi verbul **GET**. Paginile care au verbul GET se pot accesa direct prin intermediul URL-ului aferent acestora.

```
// GET: se afiseaza formularul de creare a unui student
public IActionResult New()
{
    return View();
}
```

Metoda **New()** care are verbul HTTP GET va afisa prin intermediul view-ului un formular prin care introducem datele aferente unui student. Pentru a trimite datele catre server, formularul trebuie sa defineasca metoda prin care trimite datele (adica verbul HTTP → `<form action="/students/new" method="post">`)

```

[HttpPost]
public IActionResult New(Student student)
{
    // cod creare student
    // dupa crearea studentului, se preia ID-ul nou inserat din
    baza de date
    // se redirectioneaza browser-ul catre studentul nou creat

    return Redirect("/students/" + id);
}

```

Datele introduse in formular vor fi trimise catre server prin intermediul metodei POST. Astfel, putem sa definim o ruta cu acelasi nume, dar cu verbul POST [**HttpPost**]. Aceasta ruta necesita un parametru prin care o sa primeasca datele din formular.

Deoarece metoda are acelasi nume atat in momentul afisarii formularului, cat si in momentul trimiterii datelor catre server, este necesar un tip de date diferit pentru parametrii acestuia, dar si un verb Http diferit.

```

// GET: se doreste editarea unui student
public IActionResult Edit(int ID)
{
    return View();
}

// POST: se trimite modificarile la server si se stocheaza
[HttpPost]
public IActionResult Edit(Student ID)
{
    // cod modificare date student
    // se redirectioneaza browser-ul catre studentul editat
    return Redirect("/Students/Edit" + ID);
    //return RedirectToRoute("students_show", new { id = ID
});
}

```

Aceasta metoda modifica datele studentului si primeste datele prin intermediul verbului HTTP POST. Metodele care creeaza, modifica sau sterg date nu au de obicei un View. Dupa finalizarea procesarii datelor, acestea redirectioneaza utilizatorul la o pagina aferenta actiunii. **De exemplu:** in actiunea de mai sus redirectionam la pagina de afisare a datelor studentului pentru a vedea modificarile efectuate.

```
[HttpPost]
public IActionResult Delete(int id)
{
    // cod stergere student din baza de date
    // redirectionare browser la pagina index a studentilor
    return RedirectToRoute("students_index");
}
} // se inchide clasa StudentsController
```

Redirect in cadrul metodelor

Redirect

Metoda **Redirect** se foloseste in momentul in care se doreste realizarea unui redirect temporar (HTTP 302). Primeste ca argument un string, reprezentand URL-ul pe care trebuie sa il acceseze.

```
Ex: Redirect("/Students/Edit" + ID);
    Redirect("/Home/Index");
```

RedirectToRoute

Metoda **RedirectToRoute** realizeaza un redirect temporar. Primeste ca argument denumirea rutei, denumire existenta in fisierul Program.cs in momentul configurarii sistemului de rutare.

```
Ex: return RedirectToRoute("Nume_Ruta");
```

De asemenea, metoda **RedirectToRoute** mai poate fi utilizata astfel:

```
return RedirectToRoute(new { controller = "Home", action = "Index" });
return RedirectToRoute("students_show", new { id = ID });
```

students_show -> numele rutei din Program.cs

id -> parametrul din Program.cs

ID -> parametrul primit ca parametru de intrare in metoda

RedirectToAction

Metoda **RedirectToAction** se utilizeaza in momentul in care se doreste redirect temporar catre o metoda din acelasi Controller sau dintr-un alt Controller.

Ex: `return RedirectToAction("Edit");` // metoda din acelasi Controller

Se poate redirectiona si catre o metoda dintr-un alt Controller astfel:

```
return RedirectToAction("Nume_Actiune", "Nume_Controller"); →
→ return RedirectToAction("Index", "Students");
// redirect catre metoda Index din Controller-ul Students
```

Si in acest caz se pot trimite parametrii din ruta:

```
return RedirectToAction("Nume_Actiune", "Nume_Controller", new {
    paramDinRuta })
```

RedirectPermanent/ RedirectToRoutePermanent/ RedirectToActionResultPermanent

Aceste metode se utilizeaza la fel ca in exemplele anterioare, singura diferenta fiind starea redirect-ului. In acest caz se realizeaza un redirect permanent (HTTP 301). Raspunsul o sa fie stocat in memoria cache a browser-ului, iar serverul nu o sa mai fie interogat pentru accesari ulterioare.

Returnare HTTP Status Code

Pentru a returna un status al request-ului HTTP, se poate proceda astfel:

```
public StatusCodeResult BadRequest()
{
    return StatusCode(StatusCodes.Status400BadRequest);
}

public StatusCodeResult Unauthorized()
{
    return StatusCode(StatusCodes.Status401Unauthorized);
}

public IActionResult Forbidden()
{
    return StatusCode(StatusCodes.Status403Forbidden);
}

public IActionResult NotFound()
{
    return StatusCode(StatusCodes.Status404NotFound);
    //sau return NotFound();
}
```

Ca tip de returnare, la fel ca in cazul celorlalte tipuri, se poate utiliza direct IActionResult.