Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Curs 4

Cuprins

Controller 2
- Ce este Controller-ul 2
- Crearea unui proiect 3
- Adăugarea unui nou Controller 4
- Controller-ul implicit – HomeController 7

Actions 9
- Structura unei metode 9
- Răspunsul acțiunilor – ActionResult 11
- Parametrii unei acțiuni 14
- Selectori 14
  - ActionName 15
  - NonAction 15
  - ActionVerbs 16
- Exemplu definire acțiuni 17
  - Afișare 17
  - Adăugare 18
  - Editare 19
  - Ștergere 20

Redirect în cadrul metodelor 20
- Redirect 20
- RedirectToRoute 20
- RedirectToAction 21
- RedirectPermanent/ RedirectToRoutePermanent/ RedirectToActionPermanent 21

Returnare HTTP Status Code 22

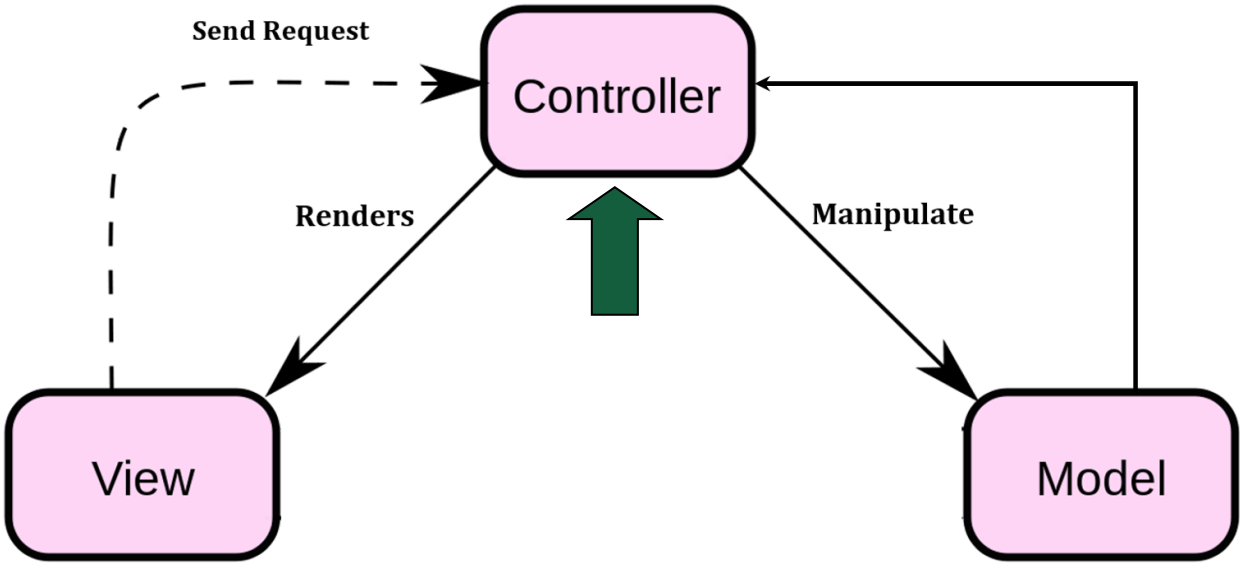
Controller

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.

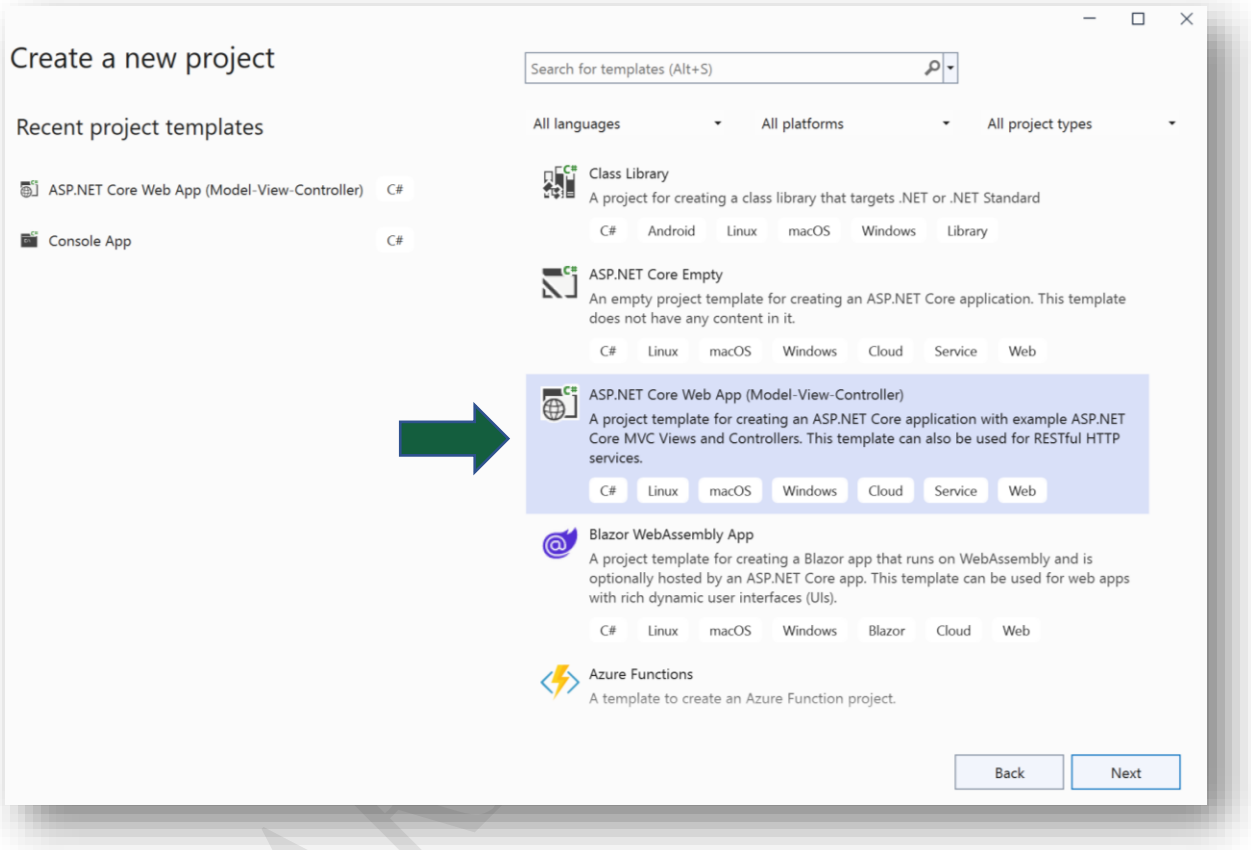
În ASP.NET MVC, fiecare Controller este reprezentat de o clasă. Numele Controller-ului trebuie să se termine în cuvântul **Controller**. De exemplu, Controller-ul pentru pagina Home se poate numi **HomeController**.

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



Crearea unui proiect

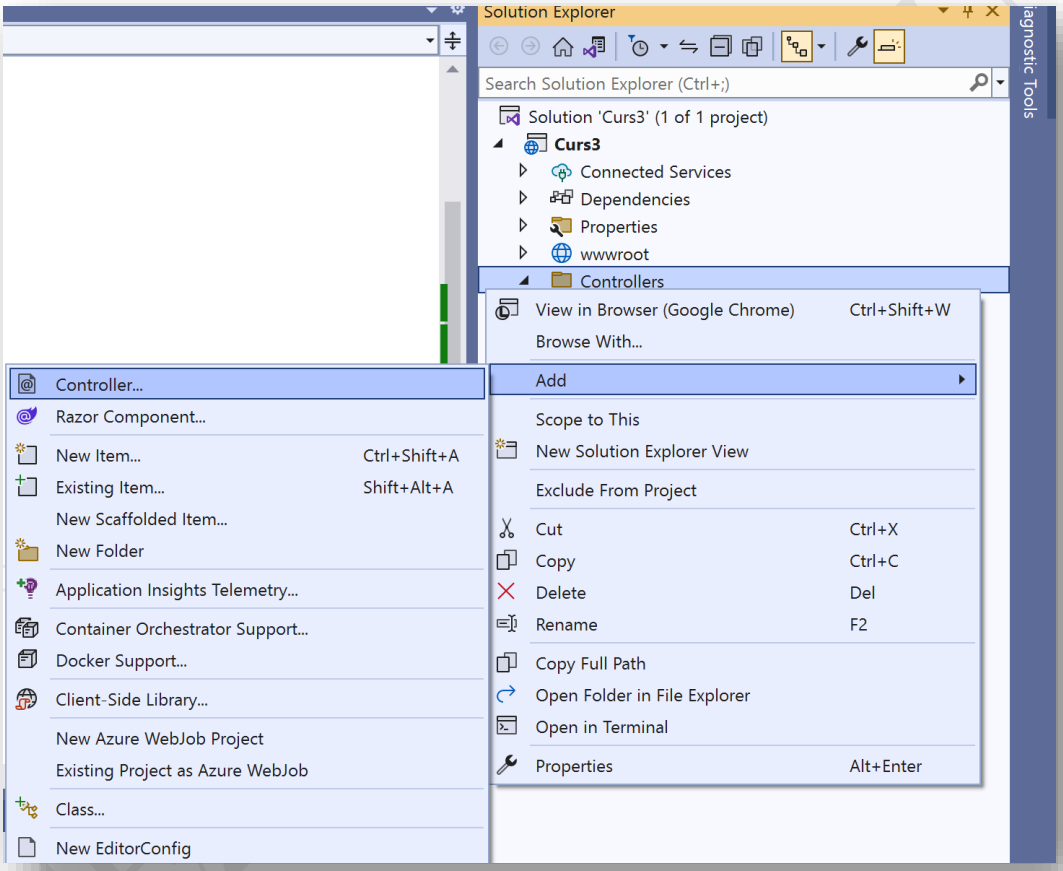
Pentru crearea unui nou proiect se utilizează opțiunea ASP.NET Core Wep App (Model-View-Controller).



Adăugarea unui nou Controller

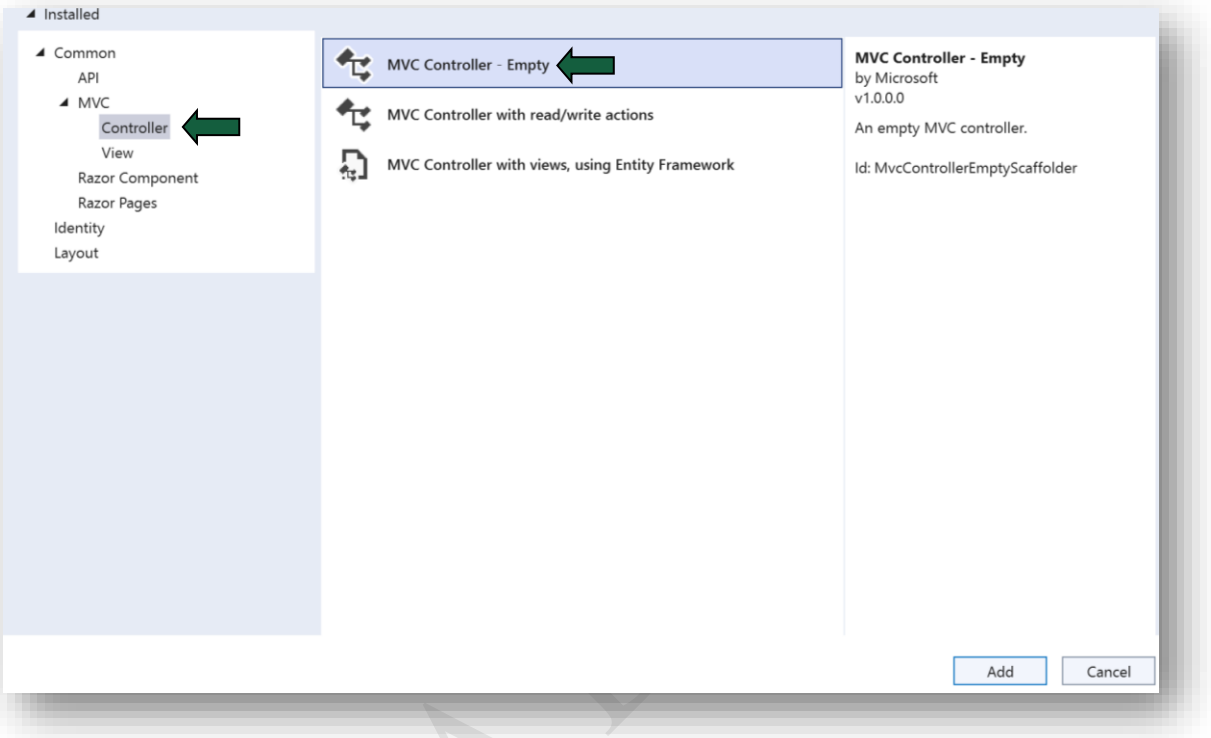
Pasul 1:

Pentru a adăuga un Controller, se procedează astfel: click dreapta pe folderul **Controllers** și din meniul **Add** se selectează **Controller**.



Pasul 2:

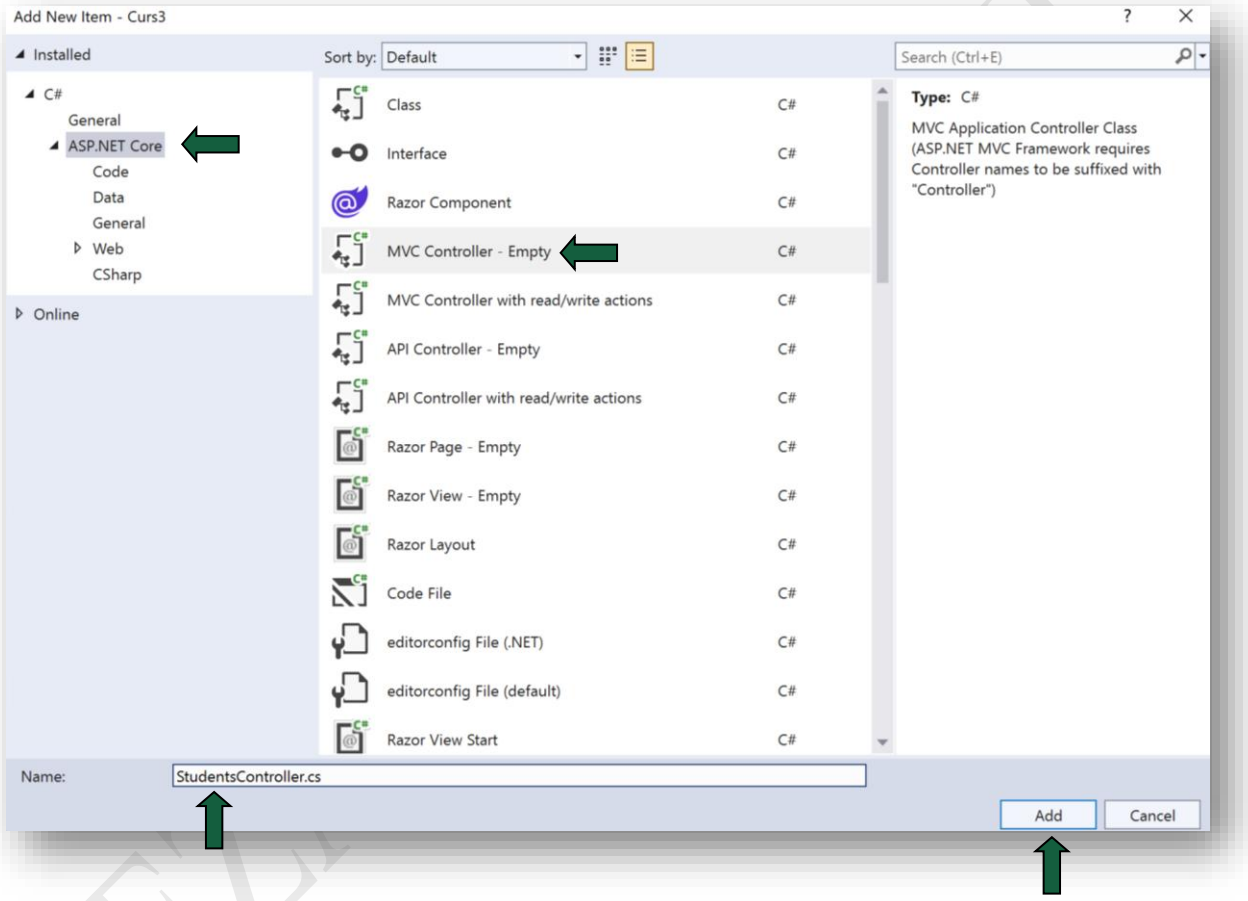
În fereastra apărută, se selectează **MVC Controller – Empty**:



Pasul 3:

Se selectează din meniul aflat în partea stângă, **ASP.NET Core**, după care opțiunea **MVC Controller – Empty.**

Se adaugă numele Controller-ului, nume care trebuie să aibă sufixul Controller. De exemplu: **StudentsController**.



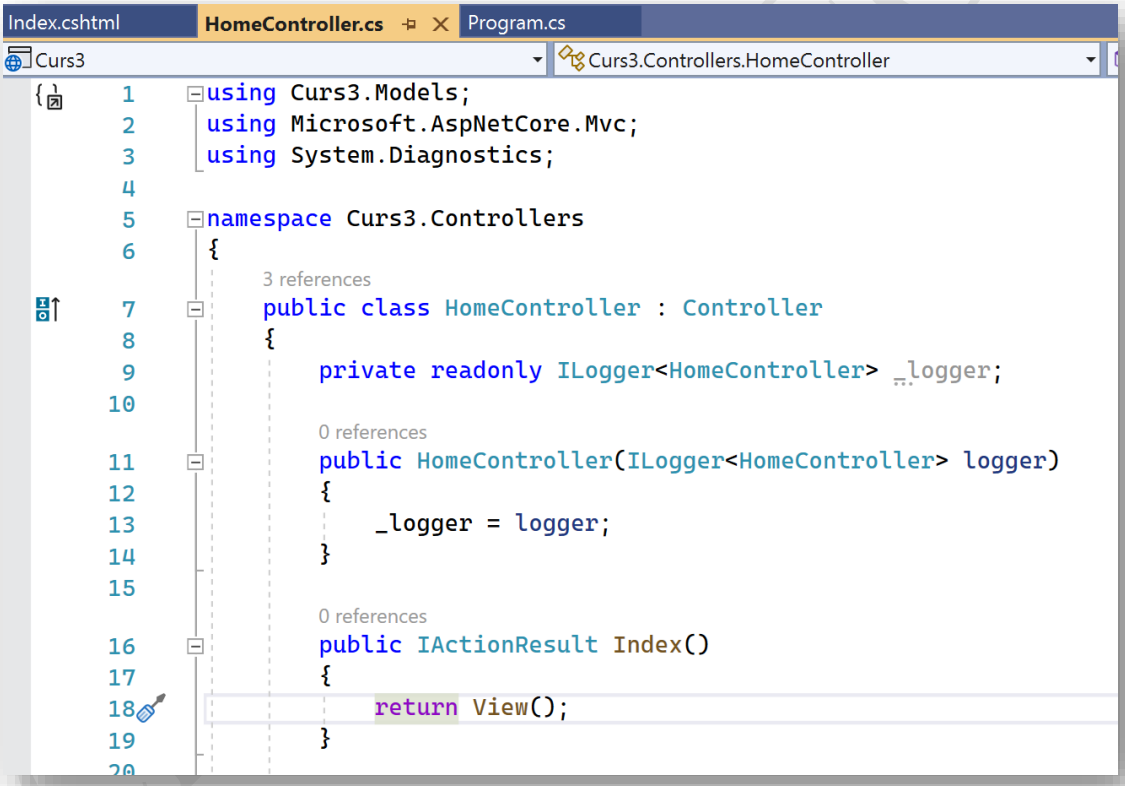
Controller-ul implicit – HomeController

În folderul Controller se observă existența unui Controller implicit. Acesta este creat automat în momentul în care se creează un nou proiect.

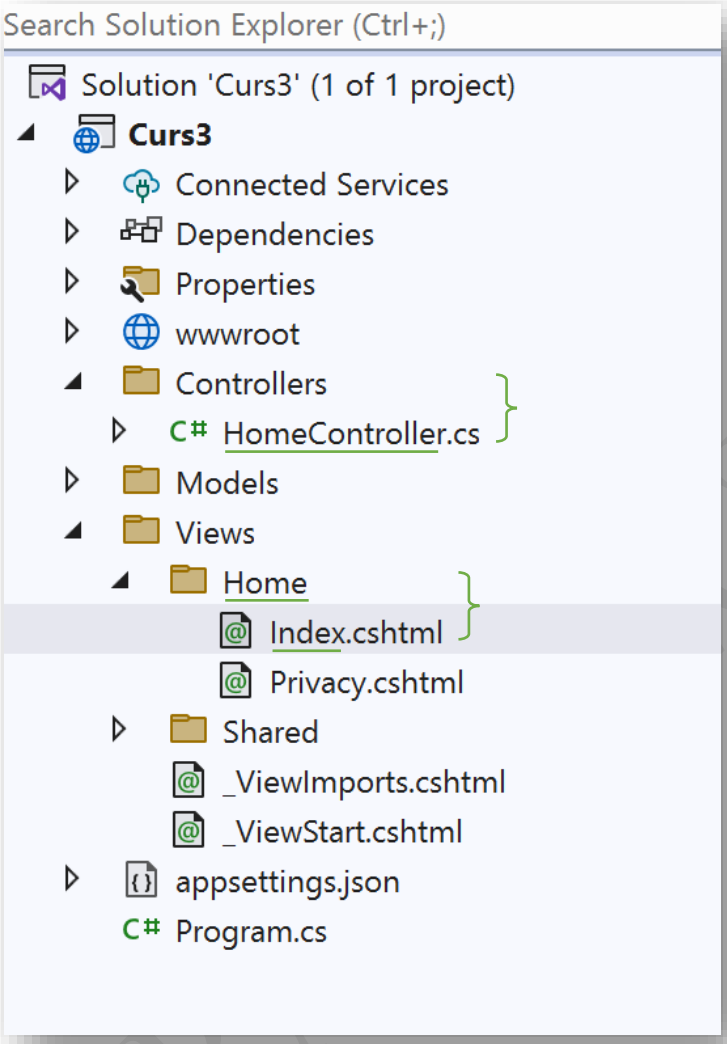
/!\ Observație

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

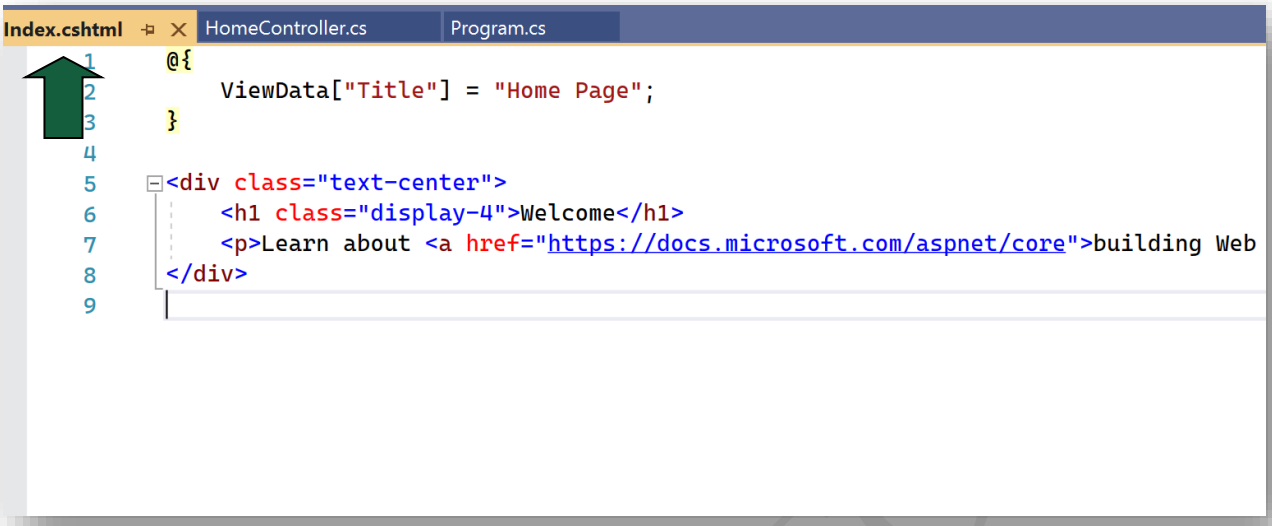
Controller-ul numit ***HomeController*** este, de fapt, o clasă care conține mai multe metode publice (Actions).



Se poate observa metoda **Index()** care returnează un View. Așadar, metoda Index o să aiba 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**, conține două tipuri de cod:

- O expresie **Razor** care conține cod C# – un obiect de tip dicționar **ViewData** (vom studia în cursul destinat View-urilor).
  Codul C# este inclus folosind simbolul **@**

- Elemente de html pentru interfață (UI)

Actions

Structura unei metode

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

În Controller-ul **StudentsController** se observă metoda Index() de tipul **IActionResult**. Aceasta este definită ca fiind **public** pentru a putea fi accesată de framework.

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

Acțiunile ***pot fi supraîncărcate*** (adică să existe metode cu același nume/semnătură, în aceeași clasa) doar dacă se utilizează cu parametri de intrare diferiți (număr diferit de parametri sau tipuri de parametri diferite). Aceste metode trebuie să se distingă prin semnătură, nu prin tipul de returnare.

Tipul returnat **IActionResult** este o **interfață abstractă**, reprezentând răspunsul acțiunii trimis de către Controller la browser, fiind cel mai frecvent tip utilizat, deoarece întotdeauna se va apela un View pentru a afișa informațiile către utilizatorul final. Ea face parte din cadrul de lucru **MVC** (Model-View-Controller) și este utilizată pentru a indica cum ar trebui să răspundă serverul HTTP la o cerere făcută de un client.

Metoda **View()** din interiorul acțiunii este definită în clasa abstractă de bază - **Controller** - și este de tipul **ViewResult : ActionResult**.

Deoarece **clasa ActionResult** implementează **interfața IActionResult** este suficient ca tipul de return al metodei să fie **IActionResult**. Cu alte cuvinte, IActionResult permite diferite tipuri de răspunsuri HTTP.

Răspunsul acțiunilor – 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, fișiere, string-uri, json, obiecte, etc.

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

- **ViewResult** – această clasă returnează conținut HTML. Răspunde cu un view HTML, bazat pe un fișier Razor (.cshtml);
- **EmptyResult** – această clasă returnează un răspuns gol – pagina returnată nu are niciun conținut (ex: returnează status code 200 → adică request-ul a fost executat corect, dar răspunsul este gol);
- **ContentResult** – poate fi folosit pentru a returna un conținut simplu ca răspuns (text);
- **FileContentResult** / **FileStreamResult** – reprezintă conținutul unui fișier (folosit pentru manipularea fișierelor);
- **JsonResult** – răspunde cu date serializate în format JSON;
- **RedirectResult** – reprezintă redirecționarea către un nou URL;
- **RedirectToRouteResult** – reprezintă redirecționarea către o altă acțiune în același Controller sau în alt Controller;
- **PartialViewResult** – returnează HTML-ul dintr-un parțial;
- **StatusCodeResult** – returnează un răspuns de tip: BadRequest (400), Unauthorized (401), Forbidden (403), NotFound (404);

ActionResult este clasa de bază a tuturor claselor enumerate mai sus. ActionResult implementează interfața IActionResult. Deci, indiferent de raspunsul folosit, acțiunea poate să aibă tipul de răspuns **IActionResult**.

Pentru a returna tipurile de date menționate mai sus, clasa de bază Controller are următoarele metode implementate:

- ViewResult → View();
- ContentResult → Content() – primește ca parametru un string care va fi afișat în browser;
- FileContentResult/FilePathResult/FileStreamResult → File();
- JsonResult → Json() – primește ca parametru orice tip de date și va returna un răspuns sub forma JSON (se va serializa parametrul primit sub forma unui string JSON);
- RedirectResult → Redirect() – primește ca parametru un URL (în format string) și va redirecționa browser-ul către acel URL;
- RedirectToRouteResult → RedirectToRoute() – primește ca parametru un **nume de rută** (care este definită în fișierul Program.cs) și va redirecționa browser-ul către acea rută;
- PartialViewResult → PartialView() – returnează conținutul unui parțial;

Avantajul utilizării ca tip de date de return a interfeței **IActionResult** este acela că se poate utiliza oricare dintre tipurile de răspuns enumerate mai sus, fără a schimba tipul de date al acțiunii.

De exemplu, pentru a descărca un fișier prin intermediul unei rute putem folosi următoarea secvență de cod:

```
public IActionResult Download()

{
    // Definim calea completă către fișierul pe care vrem să-l
descărcăm
    string filePath = @"C:\folder\myfile.ext";


    // Verificăm dacă fișierul există pe server
    if (!System.IO.File.Exists(filePath))
    {
        return NotFound("Fișierul nu a fost găsit pe server");
    }


    // Se citește fișierul ca secvență de bytes de la o cale
cunoscută - filePath
    byte[] fileBytes = System.IO.File.ReadAllBytes(filePath);


    // Specificăm numele sub care fișierul va fi descărcat
    string downloadName = Path.GetFileName(filePath);


    // Tipul de conținut - MimeType-ul (MediaType) fișierului
descărcat
    // Exemplu: pentru fisiere .mp3 -> audio/mpeg;
    // pentru imagini de tip JPEG -> image/jpeg;
    // pentru imagini de tip PNG -> image/png;
    // pentru encodarea corectă a fișierului salvat
    string contentType = MediaTypeNames.Application.Octet;


    // Returnăm fișierul către client ca răspuns HTTP
    // fileBytes - array-ul de bytes care stochează conținutul
fișierului
    // downloadName - numele fișierului care va fi salvat pe
client
    return File(fileBytes, contentType, downloadName);

}
```

Parametrii unei acțiuni

Fiecare acțiune poate să primească parametrii unei rute în cadrul semnăturii acesteia. Parametrii rutei pot fi de orice tip (string, int, float sau chiar de tipul clasei unui **Model**).

```
public IActionResult Show(int id)
{
    …
}
```

În exemplul anterior, pentru ruta "**/Students/Show/2**", o să se acceseze Controller-ul ***Students***, actiunea ***Show***, iar ***id-ul*** o să primească valoarea 2.

/!\OBSERVAȚIE:

Numele parametrilor definiți în semnătura Acțiunii **trebuie să fie același** cu numele parametrilor definiți în **Program.cs**. Diferența dintre numele parametrilor va conduce către nerezolvarea acestora (vor avea valoarea null).

Selectori

Framework-ul ASP.NET Core MVC oferă posibilitatea adăugării unor atribute acțiunilor, pentru a ajuta sistemul de rutare să aleagă acțiunea corectă în momentul procesării unui request. Aceste atribute se numesc **Selectori** și sunt:

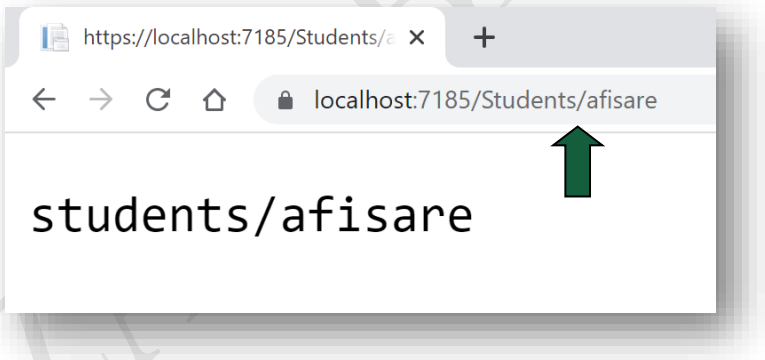
- ActionName;
- NonAction;
- ActionVerbs;

ActionName

Atributul **ActionName** oferă posibilitatea de a aloca un nume unei acțiuni, care este diferit de numele acesteia. De exemplu, dacă avem o metodă numită **Index** în Controller-ul **StudentsController**, putem redenumi această acțiune prin intermediul atributului ActionName, astfel:

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

Înainte de adăugarea atributului ActionName, pagina se putea accesa prin URL-ul: **/Students/Index**. După adăugarea atributului ActionName cu valoarea "afisare", pagina poate fi accesată prin intermediul URL-ului **/Students/afisare**. Astfel, acest atribut ne oferă posibilitatea rescrierii numelui acțiunii. Acest lucru se intâmplă fără a aduce modificări fișierului Program.cs.



NonAction

Atributul **NonAction** indică faptul că o metodă a unui Controller nu este o acțiune. Acest atribut se folosește în momentul în care dorim ca o metodă publică a unui Controller să nu poată fi accesată prin intermediul unei rute.

Exemplu:

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

Această metodă nu poate fi accesată prin intermediul unei rute, deși este publică. In schimb, ea poate fi accesată din celelalte metode ale aceluiași Controller sau ale unui alt Controller.

ActionVerbs

Atributul **ActionVerbs** este folosit în momentul în care se dorește accesarea unei acțiuni în funcție de **verbul HTTP**. De exemplu, se pot defini două acțiuni cu același nume, însă care raspund la un verb HTTP diferit și au parametri diferiți.

Verbele HTTP acceptate sunt următoarele: **GET**, **POST**, **PUT**, **PATCH**, **HEAD**, **OPTIONS** și **DELETE**.

/!\OBSERVAȚIE:

În cazul în care atributul ActionVerbs este omis, verbul default folosit este **GET**.

Aceste verbe sunt folosite în următoarele contexte:

- **GET**: este folosit în accesarea unei resurse (cererea unei pagini de la server);
- **POST** – este folosit în crearea unei resurse sau trimiterea datelor la server prin intermediul unui formular;

- **PUT/PATCH** – verbul este folosit pentru modificarea (totală sau parțială) a unei resurse. De exemplu: când se editează o intrare deja existentă în baza de date, se folosește unul dintre aceste verbe;
- **DELETE** – verb folosit pentru ștergerea unei resurse;
- **HEAD** – este identic cu GET, dar returnează doar antetele pentru răspuns, nu și conținutul răspunsului. De obicei se folosește pentru a verifica dacă există o resursă sau dacă poate fi accesată;
- **OPTIONS** – returnează metodele HTTP acceptate de server pentru o adresă URL specificată;

/!\OBSERVAȚIE:

În ASP.NET Core MVC pentru editare (PUT) și ștergere (DELETE) se folosește tot POST → [HttpPost].

Exemplu definire acțiuni

Exemplu de definire a acțiunilor (metodelor) folosind **verbele HTTP** corespunzătoare:

Afișare

```
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 două acțiuni, **Index()** și **Show()**, afișează informații despre studenți. **Index** va afișa **lista tuturor studenților**, iar **Show** va afișa **informații despre un singur student**, în funcție de ID-ul primit ca parametru. Deoarece aceste pagini afișeaza informații, și nu trimit nimic la server, vom folosi verbul **GET**. Paginile care au verbul GET se pot accesa direct prin intermediul URL-ului aferent acestora.

Adăugare

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

Metoda **New()**, care are verbul **HTTP GET**, va afișa prin intermediul view-ului un formular prin care introducem datele aferente unui student. Pentru a trimite datele către server, formularul trebuie să definească metoda prin care trimite datele (adică 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/show/" + id);
        }
```

Datele introduse în formular vor fi trimise către server, prin intermediul metodei **POST**. Astfel, putem să definim o rută cu același nume, dar cu verbul POST [**HttpPost**]. Această rută necesită un parametru prin care o să primească datele din formular.

Deoarece metoda are același nume, atât în momentul afișării formularului, cât și în momentul trimiterii datelor către server, este necesar un tip de date diferit pentru parametri acesteia, dar și un verb Http diferit.

Editare

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


// POST: se trimit modificarile la server si se stocheaza
    [HttpPost]
    public IActionResult Edit(Student ID)
    {
        // cod modificare date student
        // se redirectioneaza browser-ul catre studentul
editat

        return RedirectToRoute("students_show", new { id =
ID });

    }
```

Această metodă modifică datele studentului și primește datele prin intermediul verbului **HTTP POST**. Metodele care creează, modifică sau șterg date nu au, de obicei, un View. După finalizarea procesării datelor, acestea redirecționează utilizatorul la o pagină aferentă acțiunii. **De exemplu**: în acțiunea de mai sus redirecționăm la pagina de afișare a datelor studentului pentru a vedea modificările efectuate. În acest context se utilizează numele rutei, nume pe care îl are parametrul ***name*** în **Program.cs**.

Ștergere

```
[HttpPost]
public IActionResult Delete(int id)
{
        // cod stergere student din baza de date
        // redirectionare browser la pagina index a studentilor

        return RedirectToRoute("students_index");

}
```

Redirect în cadrul metodelor

Redirect

Metoda **Redirect** se folosește în momentul în care se dorește realizarea unui redirect temporar (HTTP 302). Primește ca argument un **string**, reprezentând URL-ul pe care trebuie să îl acceseze.

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

RedirectToRoute

Metoda **RedirectToRoute** realizează un redirect temporar. Primește ca argument denumirea rutei, denumire existentă în fișierul Program.cs și utilizată în momentul configurării sistemului de rutare.

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

De asemenea, metoda **RedirectToRoute** mai poate fi utilizată 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 în metodă
```

RedirectToAction

Metoda **RedirectToAction** se utilizează în momentul în care se dorește redirect temporar către o metoda din același Controller sau dintr-un alt Controller.

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

Se poate redirecționa și către o metodă dintr-un alt Controller, astfel:

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

Și în acest caz se pot trimite parametrii din rută:

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

RedirectPermanent/ RedirectToRoutePermanent/ RedirectToActionPermanent

Aceste metode se utilizează la fel ca în exemplele anterioare, singura diferență fiind starea redirect-ului. În acest caz, se realizează un redirect permanent (HTTP 301). Răspunsul o să fie stocat în memoria cache a browser-ului, iar serverul nu o să mai fie interogat pentru accesările ulterioare.

Se utilizează acest tip de redirect în situații precum:

- **Schimbarea definitivă a URL-ului** – în acest caz se urmărește ca toate cererile viitoare să fie direcționate la noul URL;

- **SEO (optimizare pentru motoarele de căutare)** – motoarele de căutare precum Google vor transfera încrederea de la vechiul URL la noul URL, ceea ce este important pentru menținerea clasamentului paginii;

Returnare HTTP Status Code

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

```
public StatusCodeResult BadRequst()
   {
       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 în cazul celorlalte tipuri, se poate utiliza direct **IActionResult**.