Controller. Actiuni. Selectori. Filtre

Ce este Controller-ul

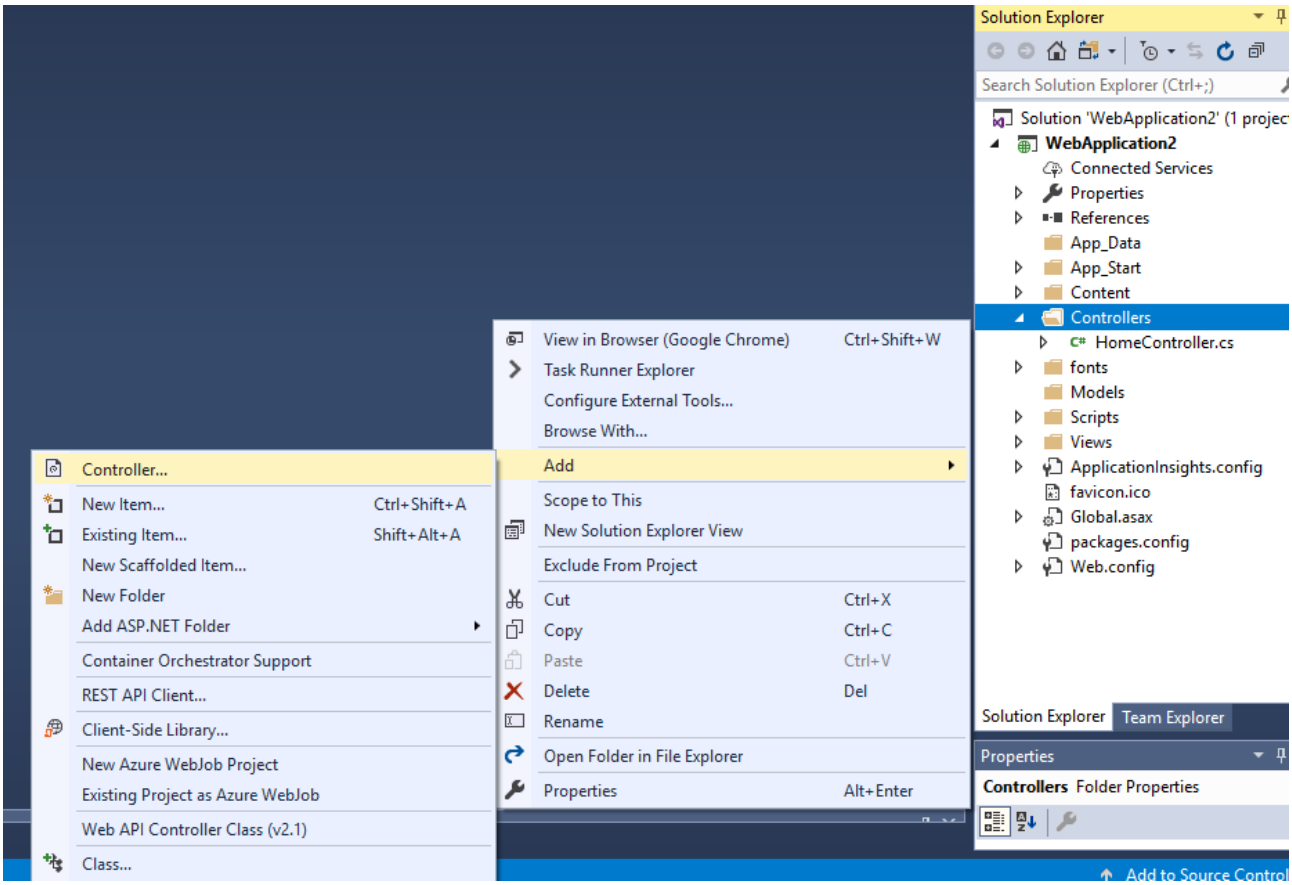
In arhitectura MVC **Controller-ul** este componenta care proceseaza toate URL-urile aplicatiei. Controller-ul este o clasa, derivata din clasa de baza **System.Web.Mvc.Controller**. Aceasta clasa contine metode publice numite **Actiuni**. Metodele din Controller sunt responsabile pentru a procesa request-urile venite de la browser, pentru apelarea modelelor si procesarea datelor, cat si pentru a trimite raspunsul final catre utilizator prin intermediul browser-ului.

In ASP.NET MVC fiecare Controller este reprezentat de o clasa. Numele Controller-ului trebuie sa se termine in cuvantul "**Controller**". De exemplu, controller-ul pentru pagina home se poate numi **HomeController**.

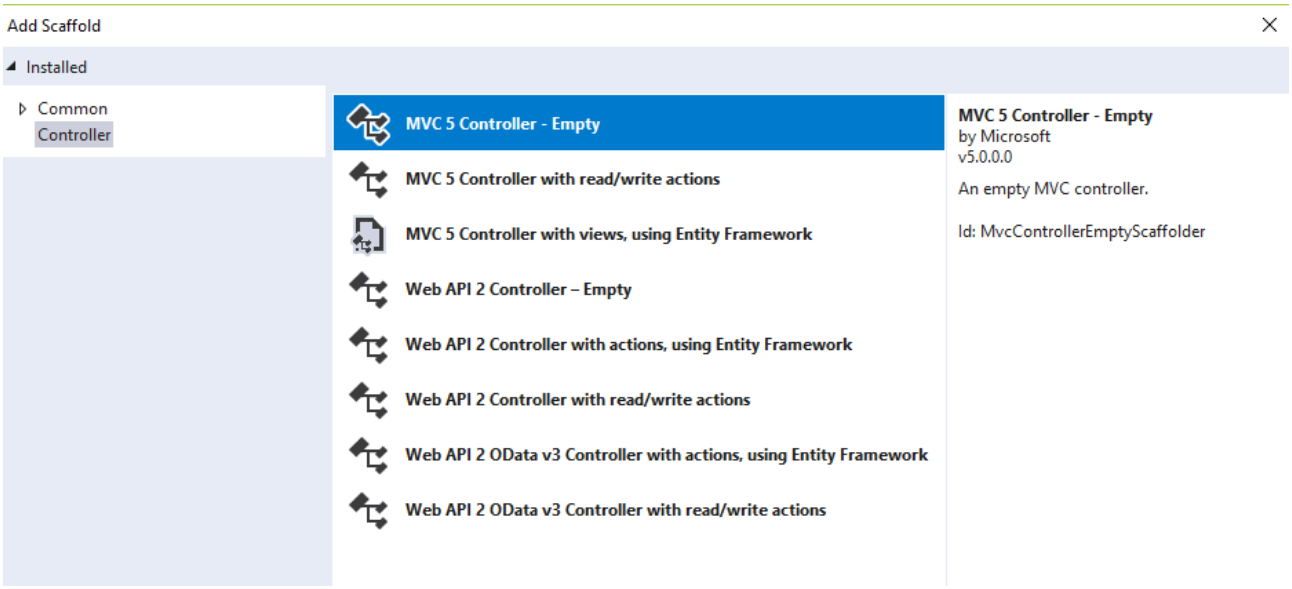
Controller-ele trebuie sa fie adaugate in cadrul folderului "Controllers" din proiectul ASP.NET.

Adaugarea unui Controller

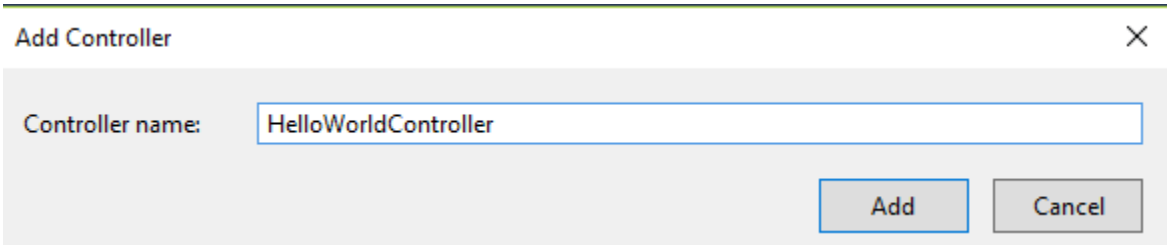
Pentru adaugarea unui Controller procedam astfel: facem click dreapta pe folderul **Controllers** si din meniul **Add** selectam "**Controller**..."



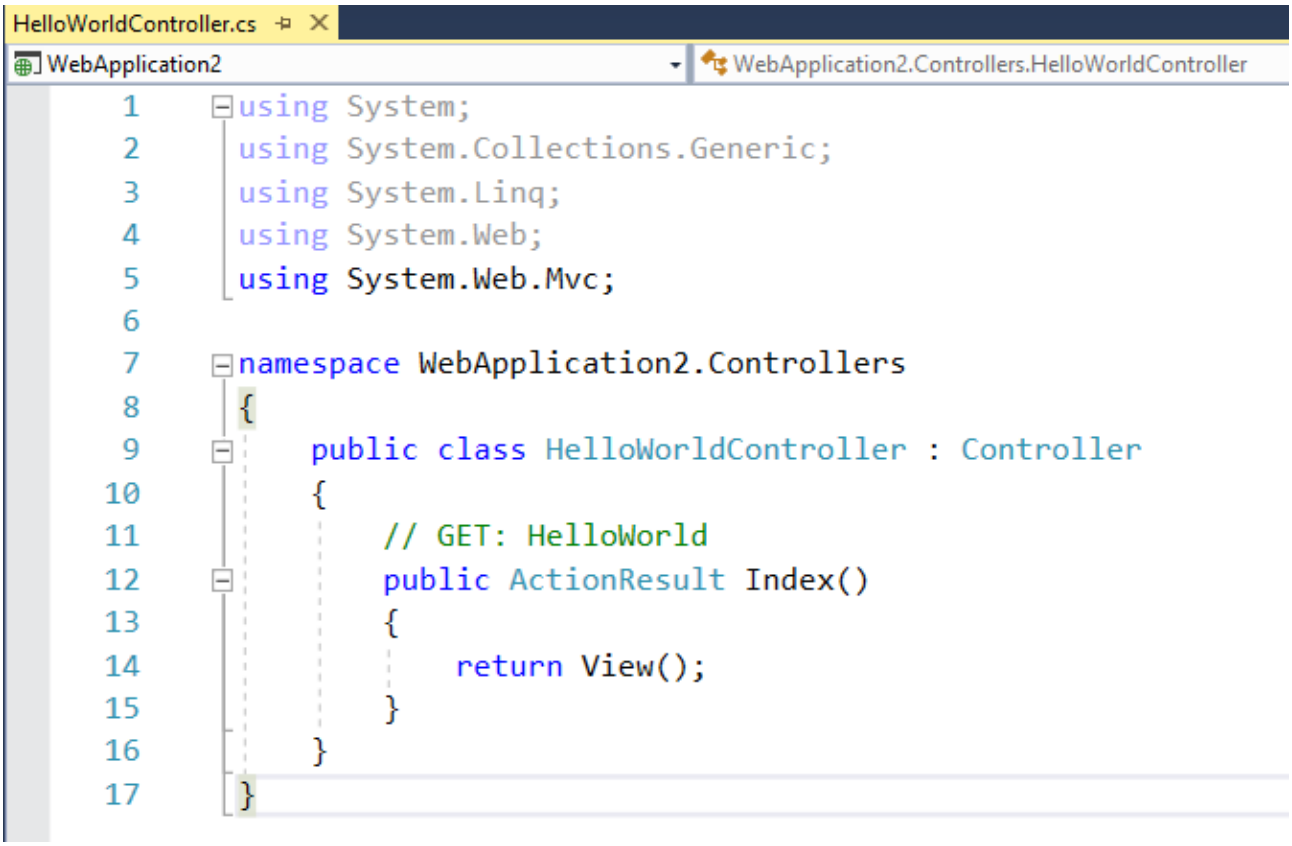
In fereastra aparuta, selectam **MVC 5 Controller – Empty**:



Adaugam numele Controller-ului – de exemplu: **HelloWorldController** si apasam butonul Add.



Codul noului Controller generat contine o metoda numita **Index()**.



Dupa cum putem observa, noul Controller adaugat este reprezentat de o clasa cu numele "HelloWorld**Controller**" (**OBS:** terminatia "**Controller**" din numele Controller-ului este obligatorie. Lipsa acesteia conduce la neidentificarea Controller-ului) care este derivata din clasa de baza abstracta Controller. Aceasta clasa de baza contine un numar de metode care pot fi folosite pe parcursul dezvoltarii unei aplicatii.

Actiuni (Actions)

Toate metodele publice ale unui Controller se numesc **Actiuni**. Acestea sunt metode normale ale unei clase, dar care au urmatoarele restrictii:

- Actiunile **trebuie sa fie publice**. Ele nu pot fi *private* sau *protected*
- Actiunile **nu pot fi supraincarcate** (overloaded)
- Actiunile **nu pot fi statice**

Structura unei actiuni

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

O actiune este reprezentata de o metoda. In exemplul de mai sus actiunea este **Index()**. Putem observa ca aceasta este definita ca fiind **public** pentru a putea fi accesata de framework.

Tipul de date returnat, **ActionResult**, reprezinta raspunsul actiunii trimis de catre Controller la browser-ul utilizatorului. Metoda **View()** din interiorul actiunii este definita in clasa abstracta de baza **Controller** si este de tipul **ViewResult : ActionResult**. Aceasta metoda returneaza un raspuns HTML dintr-un fisier de tip View.

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, fisiere javascript, 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
- **ContentResult** - poate fi folosit pentru a returna text
- **FileContentResult / FileStreamResult** - reprezinta continutul unui fisier (folosit pentru descarcarea fisierelor)
- **JavaScriptResult** – reprezinta un script JS
- **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
- **HttpUnauthorizedResult** – returneaza un raspuns de tip HTTP 403 – Forbidden

ActionResult este clasa de baza a tuturor claselor enumerate mai sus. Astfel, indiferent de raspunsul folosit, actiunea poate sa aiba tipul de raspuns ActionResult.

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**()
- JavaScriptResult -> JavaScript() – primeste ca parametru un string ce reprezinta cod JavaScript. Acesta va fi returnat catre browser si executat imediat
- 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 RouteConfig) si va redirectiona browser-ul catre acea ruta
- PartialViewResult -> PartialView() – returneaza continutul unui partial.

Avantajul utilizarii ca tip de date de return ActionResult este acela ca putem 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 ActionResult 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 citeste fisierul ca secventa de bytes de la o cale cunoscuta
Pasul 2: setam un nume de fisier pentru fisierul care va fi descarcat – downloadName
Pasul 3: returnam metoda File care primeste 3 parametri:

- Array-ul de bytes care stocheaza continutul fisierului
- MimeType-ul (MediaType) fisierului descarcat (ex: pentru fisiere **.mp3 -> audio/mpeg**; pentru **imagini de tip JPEG -> image/jpeg**; pentru **imagini de tip PNG -> image/png**) pentru encodarea corecta 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).

Exemplu:

```
public ActionResult Show(Student student)
{
    // variabila student va contine informatiile unui student din baza de date
}
```

In exemplul anterior, pentru ruta "**/students/2/show**" 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 **RouteConfig.cs**. Diferenta dintre numele parametrilor va conduce catre nerezolvarea acestora (vor avea valoarea null).

Selectori

Framework-ul ASP.NET MVC ofera posibilitatea adaugarii unor atribute actiunilor, pentru a ajuta sistemul de rutare sa aleaga actiunea corecta in momentul procesarii unui request. Aceste atribute se numesc **Selectori**. MVC 5 ofera urmatorii selectori atribuiti actiunilor:

- ActionName
- NonAction
- ActionVerbs

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

```
[ActionName("about")]
public ActionResult AboutPage()
    {
        return View();
    }
```

Inainte de adaugarea atributului ActionName pagina se putea accesa prin URL-ul **/Pages/AboutPage**. Dupa adaugarea atributuiui ActionName cu valoarea “about” pagina poate fi accesata prin intermediul URL-ului **/Pages/about**. Astfel, acest atribut ne ofera posibilitatea rescrierii numelui actiunii. Acest lucru se intampla fara a aduce modificari fisierului RouteConfig.cs

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 a unui alt Controller. **In cazul in care atributul ActionVerbs este omis, verbul default folosit este GET.**

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, insa care raspund la un verb HTTP diferit.

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

Aceste verbe sunt folosite in urmatoarele contexte:

- **GET**: este folosit in accesarea unei resurse (cererea unei pagini de la server)
- **POST** – este folosit in crearea unei resurse sau trimiterea datelor la server prin intermediul unui formular. Acest verb este similar cu runat=”server” din WebForms (postback).
- **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

Exemplu de definire a rutelor folosind **verbele HTTP** corespunzatoare:

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


        // GET: vizualizarea unui student
        public ActionResult 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: afisam formularul de creare a unui student
 public ActionResult New()
 {
     return View();
 }
```

Pagina 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"**>)

```
// POST: trimitem datele studentului catre server pentru creare
 [HttpPost]
 public ActionResult New(Student student)
 {
  // cod creare student
 // dupa ce am creat studentul luam ID-ul nou inserat din baza de date
 // redirectionam browser-ul catre studentul nou creat

     int id = 123;
     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 primeaca 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 acesteia.

```
// GET: vrem sa editam un student
 public ActionResult Edit(int ID)
 {
     return View();
 }
```

Pentru faptul ca in cadrul aplicatiilor web verbele PUT|PATCH|DELETE sunt pseudo-verbe si nu se pot folosi in tag-ul form pentru atributul method, in formularul de editare este necesar sa adaugam urmatoarea secventa de cod:

```
<form action="/Students/Edit/{id}" method="post">
    @Html.HttpMethodOverride(HttpVerbs.Put)
    . . .
<form />
```

```
// PUT: vrem sa trimitem modificarile la server si sa le salvam
 [HttpPut]
 public ActionResult Edit(Student ID)
 {
     // cod modificare date student
     // redirectionam browser-ul catre studentul editat
     return Redirect("/students/" + ID);
     //return RedirectToRoute("students_show", new { id = ID });
 }
```

Aceasta metoda modifica datele studentului si primeste datele prin intermediul verbului HTTP PUT. 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.

```
        [HttpDelete]
         public ActionResult Delete(int id)
         {
             // cod stergere student din baza de date
             // redirectionam browserul la pagina index a studentilor
             return RedirectToRoute("students_index");
         }
    }
```

Metoda nu va putea fi accesata prin intermediul URL-ului deoarece are verbul HttpDelete la care raspunde. Astfel, va fi nevoie in View-ul din care vrem sa efectuam stergerea de un formular care sa trimita catre aceasta pagina cu verbul HTTP corespunzator:

```
<form action="/Students/Delete/{id}" method="post">
    @Html.HttpMethodOverride(HttpVerbs.Delete)
    <button type="submit">Delete student</button>
</form>
```

Formularul va trimite un request catre URL-ul "**/Students/Delete/{id}**" prin metoda POST. Insa, datorita existentei helper-ului **@Html.HttpMethodOverride** formularul va include urmatorul hidden field in corpul sau:

```
40 <form action="/Students/Delete/3" method="post">
41     <input name="X-HTTP-Method-Override" type="hidden" value="DELETE" />
42     <button type="submit">Delete student</button>
43 </form>
```

Acest camp ascuns va fi transmis catre framework-ul MVC care, prin intermediul componentelor sale, va face legatura cu ruta care accepta ca si verb DELETE (HttpDelete). Analog este si pentru verbele PUT si PATCH.

/!\OBSERVATIE:

Fara folosirea acestui helper in cadrul formularelor HTML, accesarea rutelor care au verbele PUT|PATCH|DELETE este imposibila.

Filtre

In ASP.NET MVC exista posibilitatea de a executa anumite secvente de cod inainte si/sau dupa executarea unei actiuni dintr-un Controller. Aceste secvente de cod se numesc **Filtre**. Ele sunt reprezentate de clase care sunt executate inainte sau dupa executia unei metode.

Filtrele de actiune sunt reprezentate de atribute care pot fi aplicate fie unei actiuni dintr-un Controller, fie unui Controller intreg. Framework-ul ASP.NET MVC ofera cateva filtre pentru actiuni default:

- **OutputCache** – acest filtru ofera un sistem de caching pentru rezultatul unei actiuni pe o perioada configurabila de timp
- **HandleError** – acest filtru poate executa cod in momentul in care o exceptie este aruncata de actiunea unui Controller
- **Authorize** – ofera posibilitatea restrictionarii unei anumite actiuni pentru un utilizator sau pentru un rol

In ASP.NET MVC filtrele sunt de 4 tipuri:

- **Authorization** filters – filtre care implementeaza componenta de autorizare
- **Action** filters – filtre care se pot aplica actiunilor
- **Result** filters – filtre care se pot aplica rezutatelor unei actiuni sau ale unui Controller
- **Exception** filters – filtre care pot trata exceptiile unei actiuni

Acestea se executa cu prioritate in ordinea listei de mai sus. De exemplu, filtrele pentru autorizare se vor executa intotdeauna inainte de Action Filters, Result Filters sau Exception Filters.

Exemplu:

```
[OutputCache(Duration = 3600)]
public string Index()
{
    return "This will be cached for one hour!";
}
```

Actiunea Index() va returna timp de o ora acelasi continut, chiar daca valoarea de return din interiorul acesteia se va schimba. Acest lucru se intampla deoarece primul raspuns oferit de catre actiune va fi cache-uit (adica o sa fie generat un fisier static cu raspunsul acesteia, care timp de o ora o sa fie servit in detrimentul executarii acelei metode si returnarii valorii introduse in secventa de return).

Pe langa filtrele implicite, ASP.NET MVC ofera posibilitatea de a scrie si filtre proprii care pot fi folosite in cadrul aplicatiei. Filtrele scrise de dezvoltatori sunt reprezentate de o clasa care mosteneste clasa de baza numita **ActionFilterAttribute**. Cele 4 metode in care se pot executa secvente de cod pentru un filtru sunt:

- **OnActionExecuting** – aceasta metoda se va apela inainte de executarea actiunii
- **OnActionExecuted** – aceasta metoda se va apela dupa executarea actiunii
- **OnResultExecuting** – aceasta metoda se va executa inaintea afisarii rezultatului
- **OnResultExecuted** – aceasta metoda se va executa dupa afisarea rezultatului

Exemplu:

In root-ul proiectului se va crea un folder numit "**ActionFilters**". In cadrul acestuia se va adauga o noua clasa cu urmatorul continut:

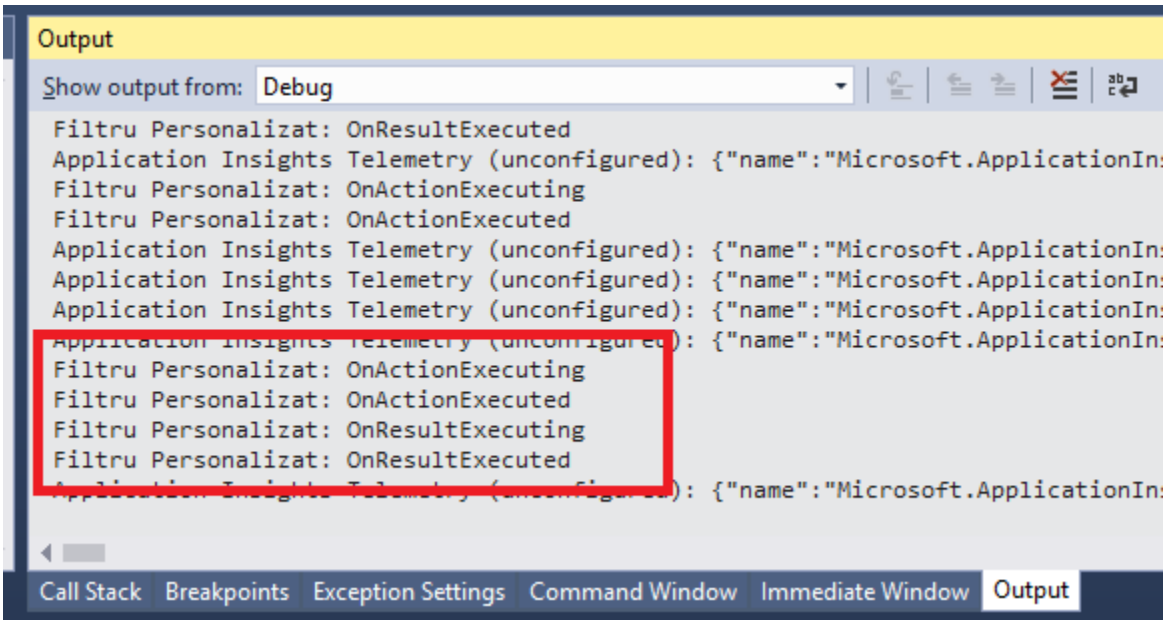
```
public class FiltruAfisareMesaj : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext
filterContext)
    {
        Mesaj("OnActionExecuting");
    }

    public override void OnActionExecuted(ActionExecutedContext
filterContext)
    {
        Mesaj("OnActionExecuted”);
    }

    public override void OnResultExecuting(ResultExecutingContext
filterContext)
    {
        Mesaj("OnResultExecuting");
    }

    public override void OnResultExecuted(ResultExecutedContext
filterContext)
    {
        Mesaj("OnResultExecuted");
    }

    // Afisam un mesaj in consola
    private void Mesaj(string mesaj)
    {
        Debug.WriteLine(mesaj, "Filtru Personalizat");
    }
}
```

Clasa adaugata va afisa cate un mesaj in consola aplicatiei pentru toate cele 4 evenimente care se vor desfasura in momentul aplicarii acestui filtru.

Aplicarea filtrului la nivel de Controller se poate face dupa cum urmeaza:

```
[FiltruAfisareMesaj]
public class StudentsController : Controller
```

In momentul rularii acestui Controller putem sa vedem mesajele din filtrul creat in consola:



De asemenea, filtrele pot fi folosite pentru executarea oricarui tip de cod pentru Actiuni sau Controllere. Prin intermediul acestora se pot implementa: sisteme de logging, sisteme customizate de autorizare si autentificare, sisteme de executare a unor actiuni in functie de anumite considerente ale aplicatiei, etc.

Fiecare parametru al actiunilor care se executa in cadrul unui filtru poate oferi informatii despre: parametrii, rute, request, raspuns, datele rutei, etc.