Curs 5

Async si await
Tag Helpers

Programare asincrona in ASP.NET Core

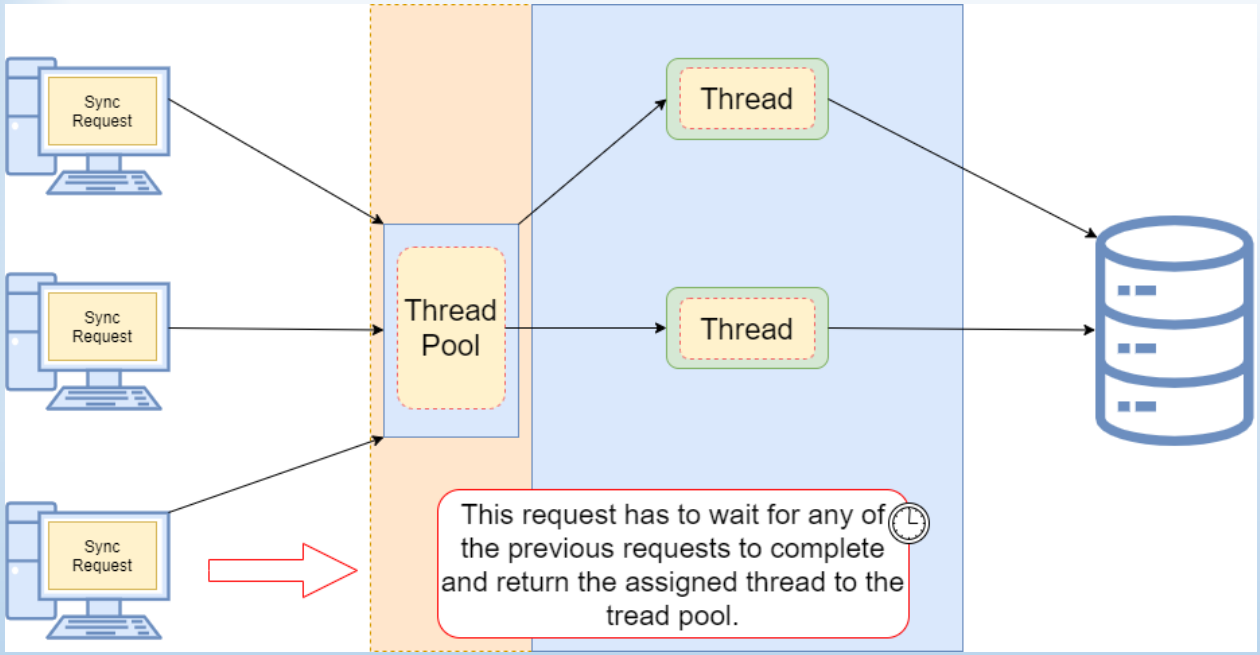
- Prin utilizarea programarii asincrone evitam probleme legate de performanta aplicatiei si crestem responsivitatea acesteia
- Programarea asincrona reprezinta o tehnica care permite executarea instructiunilor fara blocarea acesteia
- Nu mareste performata in termeni de viteza a aplicatiei – daca o interogare a bazei de date dureaza trei secunde – codul asyncron nu o va face mai rapida
- Imbunatatire indirecta a performantei cu privire la cate cereri concurente poate gestiona serverul => cu alte cuvinte creste scalabilitatea aplicatiei

Scalabilitate- importanta?

- Serverul poate gestiona un numar limitat de cereri

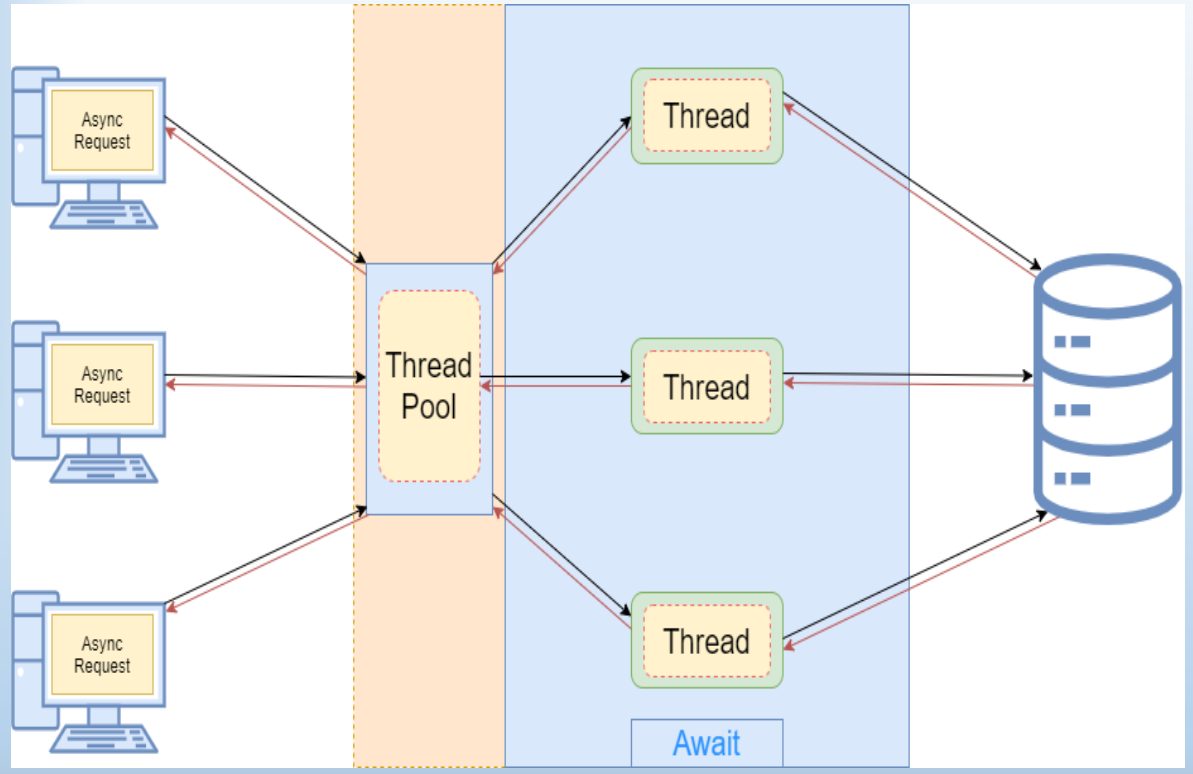
- Daca aplicatia primeste mai multe cereri decat poate gestiona serverul – performanta per ansamblu va avea de suferit (putem adauga un server – scalare pe orizontala)

Cereri sincrone



- Presupunem ca avem doar doua thread-uri pe server
- Un client trimite o cerere catre o pagina care interogheaza lista de carti din baza de date - se aloca un thread
- Un al doilea client face o cerere- se aloca al doilea thread
- Daca un al treilea client face o cerere in acest moment trebuie sa astepte una din primele doua cererei sa se finalizeze si sa elibereze unul din thread-uri – utilizatorul 3 experimenteaza o intarziere
- Deoacere clientul asteapta o lista cu carti din baza de date – o operatie de I/O cu multe inregistrari ex. 3 secunde ; thread-ul nu face nimic decat asteapta rezultatul si este locat trei secunde facand-ul indisponbil pentru alte cereri

Cereri asincrone



- In cazul cererilor asincrone de indata ce cererea ajunge in punctul in care executa operatie de I/O unde baza de date proceseaza rezultatul cateva secunde, thread-ul este returnat la pool thread si poate fi utilizat de alte cereri
- Cand baza de date returneaza rezultatul, thread pool aloca un thread din nou pentru a intoarce raspunsul la client

Async si await in ASP.NET Core

- Async- se utilizeaza la declararea metodei, scopul acesteia fiind de a permite utilizarea cuvantului await in acea metoda
- Nu putem utiliza await fara async specificat inainte
- Utilizarea doar a cuvantului async nu face ca metoda respectiva sa fie asincrona – codul va fi tot sincron daca nu utilizam await
- Await realizeaza o asteptare(wait) asincrona a expresiei specificate dupa acesta:
  - Verifica daca operatia este finalizata
  - Daca este finalizata va continua executia sincron, altfel va pune in pauza executia metodei async, iar cand operatia este gata metoda async isi va continua executia

Async-await

```
public async Task OnGetAsync()
        {
            Publisher = await _context.Publisher.ToListAsync();
        }
```

Daca baza de date necesita timp pentru a interoga tabelul Publishers, cuvantul await va pune pe pauza executia metodei OnGetAync si va returna un task incomplet

In acest timp thread-ul va fi returnat la thread pool si va disponibil pentru alte cereri

Dupa ce baza de date proceseaza operatia, metoda async va relua executia si va returna lista de Publishers

Task

- Tipuri de return:
- Task<IActionResult> - pentru o metoda marcata async care returneaza o valoare (interfata IActionResult – defineste rezultatul actiunii unui action method)
- Task - pentru o metoda async care **nu** returneaza o valoare

- Task reprezinta o executie a unei metode asincrone si nu un rezultat
- Task are o serie de proprietati care indica daca o operatie a fost finalizata cu success sau nu (Status, IsCompleted, IsCanceled, IsFaulted).

Tracking vs. No-tracking

- Cand un obiect db context creaza obiecte entiate care le reprezinta, implicit retine daca aceste entiati sunt sincronizate cu baza de date
- Datele din memorie functioneaza ca un cache si sunt utilizate cand se face update la o entitate
- In aplicatiile web- acest cache adesori nu este necesar pentru ca durata de viata a obiectelor db context este scurta (se creaza unul nou la fiecare cerere) – ob. dbContext care a realizat cererea este disposed inainte sa fie utilizat din nou

Dezactivarea tracking

- Apelul metodei AsNoTracking
- Scenarii de utilizare:
  - Pe durata de viata a contextului nu e nevoie sa actualizam entitatile (ex. HTTPGet action methods)
  - Executam o interogare care returneaza un volum mare de date si doar o mica parte va trebui actualizata. Dezactivam tracking si apoi rulam o alta interogare pentru partea de date care e necesar a fi actualizata

Overposting-Creare

- Atributul Bind din codul care a fost generat automat pentru metoda Create este o metoda de protectie impotriva overposting
- public async Task<IActionResult> Create([Bind("Title,Author,Price")] Book book)

```
public class Book
   {
        public int ID { get; set; }
        public string Title { get; set; }
        public string Author { get; set; }
        public decimal Price { get; set; }
        public decimal DiscountPrice{get;set;}


}
```

Overposting- Editare

- Citirea entiatii din baza de date si apoi apelam TryUpdateModel cu o lista de proprietati permise pentru a fi editate

```
var bookToUpdate = await _context.Books.FirstOrDefaultAsync(s
=> s.ID == id);
            if (await TryUpdateModelAsync<Book>(
                bookToUpdate,
                "",
                s => s.Author, s => s.Title, s => s.Price))
            {
await _context.SaveChangesAsync();
                    return RedirectToAction(nameof(Index));
                }
}
```

Overposting cu viewmodel (scenario create/edit)

- Creare Viewmodel
- Facem binding la viewmodel si nu la entiati
- In viewmodel include doar proprietatile care dorim sa fie editate/create

Incarcarea datelor relationate

- **Eager loading** - cand se citeste entitatea, datele relationate sunt returnate impreuna cu aceasta
- Metodele *Include* si *ThenInclude*

```
var publisherToUpdate =_context.Publishers
                .Include(i => i.PublishedBooks)
                .ThenInclude(i => i.Book)
```

Incarcarea datelor relationate

- **Explicit loading** – cand se citeste entitatea datele relationate nu sunt aduse
- Incarcam explicit un navigation property cu DbContext.Entry(...)
- Cand e nevoie de acestea folosim metoda *Load*()
- Multiple interogari trimise catre bd

```
var publishers = _context.Publishers
foreach(Publisher p in publishers)
{
_context.Entry(p).Collection(i=>i.PublishedBooks.Load();
}
```

Considerente legate de performanta

- Cand stim ca vom avea nevoie de datele relationate – eager loading- o singura interogare trimisa la bd mai eficienta decat a trimite interogari separate
- Ex. Fiecare Publisher are 10 carti publicate- eager loading va produce o singura interogare de tip join –>o singura round-trip la baza de date

- Utilizam explicit loading in anumite scenarii cand eager loading poate cauza o interogare cu join foarte complex – interogari separate vor performa mai bine; sau atunci cand avem nevoie doar de un subset al datelor relationate, interogari separate vor performa mai bine pentru ca eager loading va aduce mai multe date decat avem nevoie

Tag Helpers

- Tag Helpers permit codului de tip server-side sa participe la crearea si afisarea elementelor HTML in fisiere Razor
- Tag Helpers folosesc limbaj C# si targeteaza elemente HTML bazandu-se pe numele elementului si numele atributului
- Sunt prefixati cu “asp-”

Ex. Label Tag Helper targeteaza elementul HTML <label> cand sunt aplicate atributele tag helperului

```
<label asp-for="Title"></label>
```

Ce ofera Tag Helpers?

1. **Experienta de dezvoltare asemanatoare HTML –** de cele mai multe ori, markup-ul Razor cu tag helpers este asemanator cu cel standard HTML. Designeri front-end familiarizati cu HTML/CSS/JavaScript pot edita markup Razor fara a invata sintaxa Razor

2. **Suport IntelliSense pentru crearea de markup HTML si Razor** – productivitate mai buna cand se utilizeaza tag helpers decat scrierea de markup C# Razor

Ce ofera Tag Helpers?

3. **Cod mai robust si mai facil mentabil utilizand doar informatia de la server**

Ex. Uzual la actualizarea imaginilor se modifica numele imaginii si fiecare referinta la imagine necesita sa fie actualizata. Imaginile se salveaza in cache din motive de performanta, iar daca nu se schimba numele imaginii exista riscul ca la client sa se afiseze o copie a imaginii din cache

- Image Tag helper – adauga un numar de versiune la numele imaginii, astfel ca la modificarea imaginii, serverul genereaza automat o noua versiune pentru imagine; se garanteaza afisarea imaginii curente

Image Tag Helper

- Image Tag Helper adauga tag-ului <img> elemente privind comportamentul cache pentru fisiere statice de tip imagine.
- Un string cu o valoare unica de hash este adaugata la URL permite reincarcarea imaginii de la server si nu din cache-ul clientului
- Daca se modifica imaginea de pe server, un URL unic este generat care include string-ul actualizat

```
<img src="~/images/asplogo.png" asp-append-version="true">
```

Se genereaza HTML

```
<img src="/images/asplogo.png?v=Kl_dqr9NVtnMdsM2MUg4qthUnWZm5T1fCEimBPWDNgM">
```

Input Tag Helper

```
public class Book
    {

      public int ID { get; set; }

     [Required, StringLength(150, MinimumLength = 3)]
     [Display(Name = "Book Title")]
     public string Title { get; set; }

     [RegularExpression(@"^[A-Z][a-z]+\s[A-Z][a-z]+$", ErrorMessage = "Numele autorului trebuie sa fie de forma 'Prenume Nume'"), Required, StringLength(50, MinimumLength = 3)]
     public string Author { get; set; }
}
```

```
<input asp-for="Book.Title" class="form-control" />
```

Se genereaza atributele id si name pentru expresia specificata in asp-for astfel:

```
<input class="form-control" type="text" data-val="true" data-val-length="The field Book Title must be a string with a minimum length of 3 and a maximum length of 150." data-val-length-max="150" data-val-length-min="3" data-val-required="The Book Title field is required." id="Book_Title" maxlength="150" name="Book.Title" value="" />
```

Genereaza attribute de validare pe baza adnotarilor din model

Atributul **id** generat este consistent cu atributul **for** de la label tag helper pentru a fi associate corect

Label TagHelper

```
public class Book
    {

      public int ID { get; set; }

     [Required, StringLength(150,
MinimumLength = 3)]
     [Display(Name = "Book Title")]
     public string Title { get; set; }

     [RegularExpression(@"^[A-Z][a-
z]+\s[A-Z][a-z]+$", ErrorMessage = "Numele
autorului trebuie sa fie de forma 'Prenume
Nume'"), Required, StringLength(50,
MinimumLength = 3)]
     public string Author { get; set; }
}
```

```
 <label asp-for="Book.Title"
class="control-label"></label>
```

Se genereaza caption si atributul for (vizualizare sursa pagina):

```
<label class="control-label"
for="Book_Title">Book Title</label>
```

Avantaje fata de elementul HTML<label>:

- obtinem automat valoarea atributului Display(daca nu e setat va afisa numele proprietatii). Daca se modifica Display label tag helper va modifica automat
- Mai putin cod sursa
- Cod puternic tipizat legat de proprietatea din model

Select Tag Helper

```
public class Book
    {
     public int ID { get; set; }
     [Required, StringLength(150, MinimumLength
= 3)]
     [Display(Name = "Book Title")]
     public string Title { get; set; }


     [RegularExpression(@"^[A-Z][a-z]+\s[A-
Z][a-z]+$", ErrorMessage = "Numele autorului
trebuie sa fie de forma 'Prenume Nume'"),
Required, StringLength(50, MinimumLength = 3)]
     public string Author { get; set; }
….
    public int PublisherID { get; set; }
}
```

```
<select asp-for="Book.PublisherID"
class="form-control" asp-
items="ViewBag.PublisherID"></select>
```

Asp-for specifica numele proprietatii din model si asp-items specifica optiunile

```
<select class="form-control" data-
val="true" data-val-required="The
PublisherID field is required."
id="Book_PublisherID"
name="Book.PublisherID"><option
value="1">Nemira</option> <option
value="2">Humanitas</option> <option
value="5">Arthur</option> </select>
```

Anchor Tag Helper

- Adauga noi atribute tag-ului standard HTML (<a ... >) pentru a seta atributul href la o pagina specifica

```
<a asp-page="./Index">Back to List</a>
<a asp-page="./Edit" asp-route-id="10">Edit</a>
```

Se genereaza HTML

```
<a href="/Index">Back to List</a>
<a href="/Edit? id=10">Edit</a>
```

Directiva @addTagHelper

- La crearea unei aplicatii web ASP.NET Core in fisierul *_ViewImports.cshtml* se adauga directiva

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

- efect - Tag Helpers devin disponibili in cadrul paginilor.
- Fisierul *_ViewImports.cshtml* este mostenit de toate paginile
- Sintaxa wildcard ("*") specifica ca toate Tag Helpers din assembly-ul (*Microsoft.AspNetCore.Mvc.TagHelpers*) vor fi disponibile pentru fiecare pagina
- Primul parametru @addTagHelper specifica care tag helpers sa fie disponibili, al doilea parametru "Microsoft.AspNetCore.Mvc.TagHelpers" specifica numele assembly-ului in care se gasesc Tag Helpers.
- Microsoft.AspNetCore.Mvc.TagHelpers este assembly-ul pentru tag helpers predefiniti
- Se pot creea tag helpers custom

Tag Helpers custom

- Tag helper pentru tag-ul <email>Support</email>
- support@ubbcluj.ro

```
using Microsoft.AspNetCore.Razor.TagHelpers;
public class EmailTagHelper : TagHelper
 { public override void Process(TagHelperContext
context, TagHelperOutput output)
 { output.TagName = "a"; // Inlocuieste <email> cu <a>
tag }
}
```

Tag Helpers custom

- Tag helpers utilizeaza o conventie de nume referitor la elementul pe care il targeteaza
- Elementul targetat este radacina **EmailTagHelper – email -> element targetat <email>**
- clasa EmailTagHelper mosteneste TagHelper. Clasa TagHelper ofera metodele si proprietatile necesare pentru a crea tag helpers
- metoda suprascrisa Process controleaza ceea ce face tag helper-ul cand este executat

Utilizare

- Pentru a fi vizibil in paginile Razor utilizam directiva @addTagHelper

- In *_ViewImports.cshtml*

```
@addTagHelper namespace.TagHelpers.EmailTagHelper, numeassembly
```

SetAttribute si SetContent

- Creem o ancora valida pentru email

```
public class EmailTagHelper : TagHelper
    {
        private const string EmailDomain = "ubbcluj.ro";

        public string MailTo { get; set; }
        public override void Process(TagHelperContext context,
TagHelperOutput output)
        {
            output.TagName = "a";    // inlocuieste <email> cu tag-ul <a>
            var address = MailTo + "@" + EmailDomain;
            output.Attributes.SetAttribute("href", "mailto:" + address);
            output.Content.SetContent(address);
        }
<email mail-to="Support"></email>
```