

# Curs 5

Programare asincrona

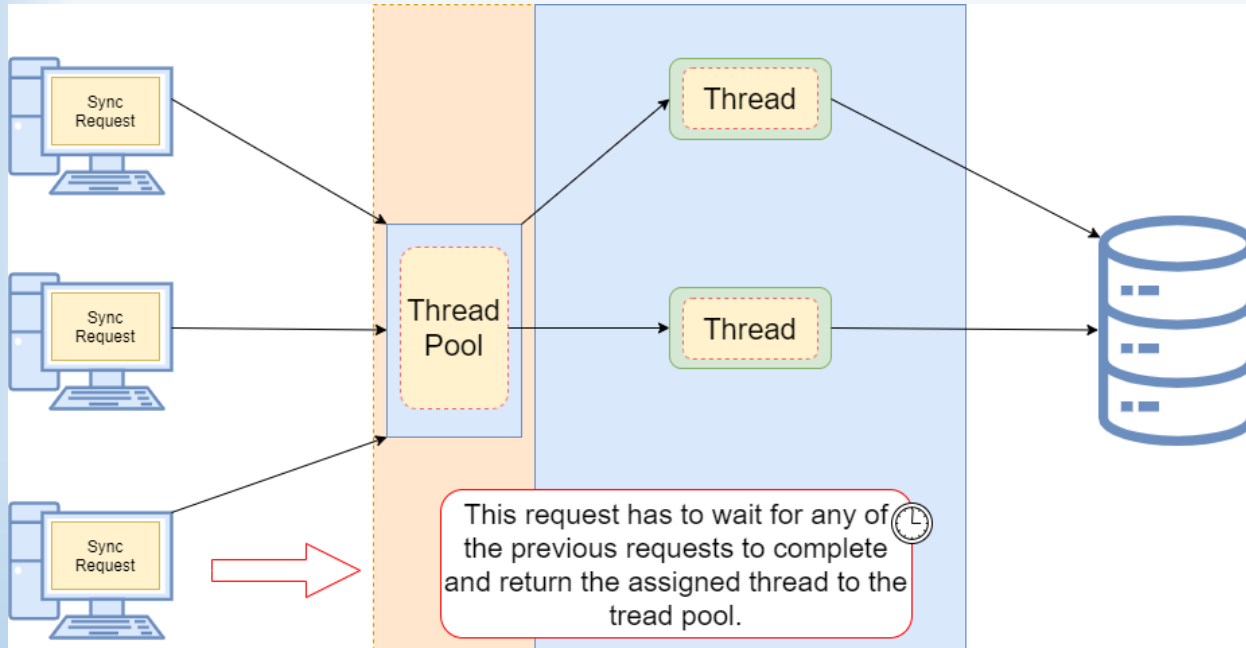
# Programare asincrona in ASP.NET Core

- 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 asincron 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?

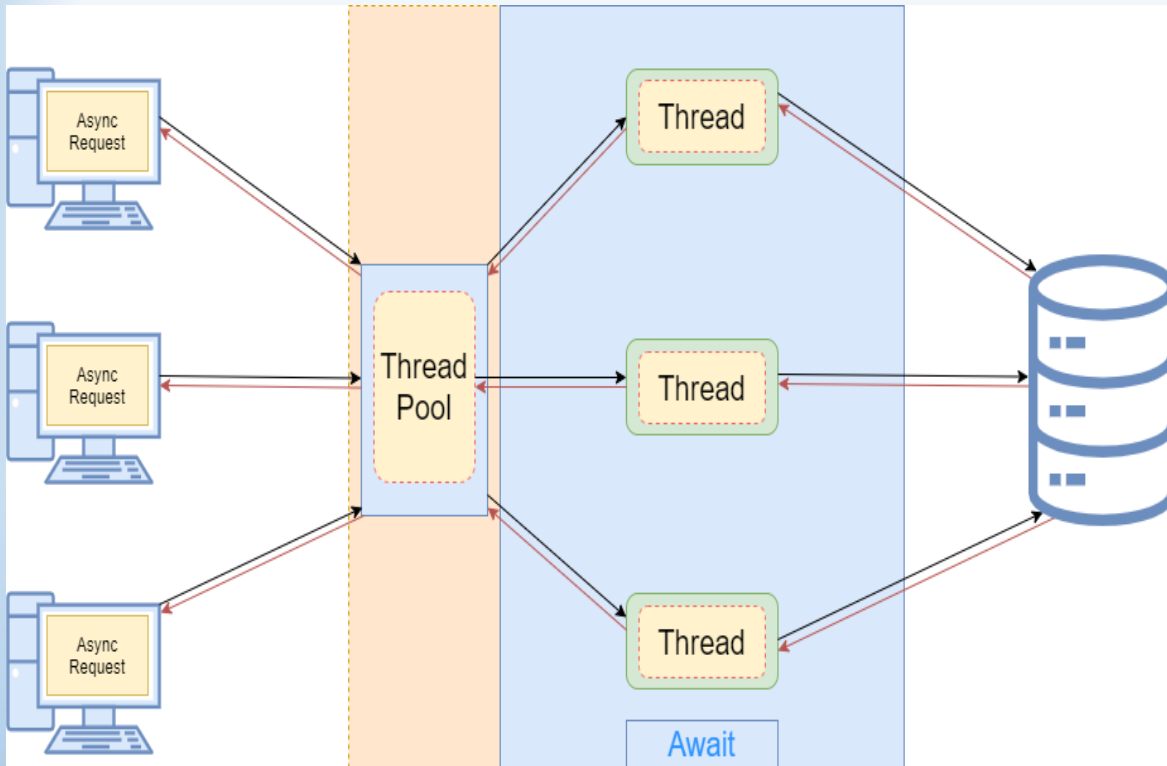
- Serverul poate gestiona un numar limitat de cereri
- Daca aplicatia primeste mai multe cereri decat poate gestiona serverul – performanta per ansamblu va avea de suferit (putem adauga un server – scalare pe orizontala)

# 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 alocă un thread
- Un al doilea client face o cerere- se alocă al doilea thread
- Dacă un al treilea client face o cerere în acest moment trebuie să aștepte una din primele doua cereri să se finalizeze și să elibereze unul din thread-uri – utilizatorul 3 experimentează o întârziere
- Deoarece clientul așteaptă o listă cu cărți din baza de date – o operație de I/O cu multe înregistrări ex. 3 secunde ; thread-ul nu face nimic decât așteaptă rezultatul și este locat trei secunde făcând-ul indisponibil 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 alocata 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 cuvintului await in acea metoda
- Nu putem utiliza await fara async specificat inainte
- Utilizarea doar a cuvintului 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<IAcțiuneRăspuns>` - pentru o metoda marcata async care returneaza o valoare (interfata `IActionResult` – definește rezultatul actiunii unei 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`).



# Overposting-Creare

- Atributul Bind din codul care a fost generat automat pentru metoda OnPostAsync este o metoda de protectie impotriva overposting
- `public async Task<ActionResult> OnPostAsync([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;}
}
```

Atributul [Bind] poate fi folosit pentru a proteja împotriva overposting în scenariile de creare. Nu funcționează bine în scenariile de editare, deoarece proprietățile excluse sunt setate la nul sau la o valoare implicită, în loc să fie lăsate neschimbate.

# Overposting- Editare

- Citirea entitatii 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,
        "Book",
        s => s.Author, s => s.Title, s => s.Price))
    {
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
}
```

# Overposting cu viewmodel (scenariu create/edit)

- Creare Viewmodel
- Facem binding la viewmodel si nu la entitati

[[BindProperty](#)]

```
public BookVM BookVM { get; set; }
```

- In viewmodel include doar proprietatile care dorim sa fie editate/create

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

# Modalitati de a citi o entitate

- **FirstOrDefaultAsync** pentru a citi o entitate. Această metodă returnează null dacă nu se găsește nimic; în caz contrar, returnează primul rând găsit care satisface criteriile din filtrul de interogare. FirstOrDefaultAsync este, în general, o alegere mai bună decât următoarele alternative:
- **SingleOrDefaultAsync** - Aruncă o excepție dacă există mai multe entități care satisfac filtrul de interogare. Pentru a determina dacă interogarea poate returna mai mult de un rând, SingleOrDefaultAsync încearcă să preia mai multe rânduri. Această muncă suplimentară este inutilă dacă interogarea poate returna o singură entitate, ca atunci când caută pentru o cheie unică.
- **FindAsync** - Găsește o entitate cu cheia primară (PK). Dacă o entitate cu PK este urmărită de context, aceasta este returnată fără o solicitare la baza de date. Această metodă este optimizată pentru a căuta o singură entitate, dar nu putem apela Include cu FindAsync. Deci, dacă sunt necesare date asociate, FirstOrDefaultAsync este cea mai bună alegere.

# Incarcarea datelor relationate

- **Date relationate:** Date pe care EF Core le încarcă în navigation properties.
- **Eager loading** - cand se citește entitatea, datele relationate sunt returnate împreună cu aceasta
- Metodele *Include* și *ThenInclude*

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

# Incarcarea datelor relationate

- **Explicit loading** – cand se citește 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

# Tracking vs. No-tracking

- Cand un obiect Dbcontext creaza obiecte entiate care le reprezinta, implicit retine daca aceste entitati 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 adeseori 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 - Metoda AsNoTracking îmbunătățește performanța în scenariile în care entitățile returnate nu sunt actualizate în contextul actual
- Scenarii de utilizare:
  - Pe durata de viață a contextului nu e nevoie să actualizăm entitățile (ex. HTTPGet action methods)
  - Executăm o interogare care returnează un volum mare de date și doar o mică parte va trebui actualizată. Dezactivăm tracking și apoi rulăm o altă interogare pentru partea de date care e necesară a fi actualizată

# 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 attributele 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

```

```

Se genereaza HTML

```

```

# 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 attributele 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 asociate 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 attribute tag-ului standard HTML (<a ... ></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>
  - <a href="mailto:support@ubbcluj.ro">support@ubbcluj.ro</a>
- ```
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 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

- Cream 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>

<a href="mailto:support@ubbcluj.ro">support@ubbcluj.ro</a>