

Curs 1

Aplicatii Web cu ASP.NET Core
- Razor Pages -

Objective

- Deprinderea principalelor concepte ale programării în .NET
- Programarea aplicațiilor Web utilizând ASP.NET Core
- Programarea aplicațiilor multiplatforma utilizând .NET MAUI
- Programarea accesului la date
- Limbaj de programare C# (principiile de programare orientată obiect învățate în anul 2 la POO sunt aceleași)
- Toate materialele sunt disponibile pe Platforma Moodle:
<https://econ.elearning.ubbcluj.ro/moodle> (Medii de programare și dezvoltare_IE3+Opt_23-24) – Curs comun, laboratoarele sunt marcate cu (IE+OPT) sau (IE)

Echipa:

- Lect. Dr. Florina Covaci, Giorgia Cociș, Mihai Negrisan



Masterat

- Continuarea traiectoriei de studii la unul din cele 3 masterate ale departamentului IE:
 - Business Analytics si Managementul Informatiilor (cu accent pe data science, data engineering, big data)
 - EBusiness (cu accent pe tehnologii web și sisteme distribuite)
 - Business Modelling and Distributed Computing (în limba engleză, cu accent pe modelare de business si calcul distribuit cu componenta de cercetare)

Evaluare

- IE – Disciplina obligatorie
 - Probe examen:
 - Probă teoretică: 40%
 - Examen grila în sesiune (20 intrebari*0.45=9+1pct oficiu) – o singura varianta de raspuns corecta
 - Proiect : 40% (cerinte minime proiect – vezi Moodle)
 - Examinat prin sustinere în perioada 8-19 ian. 2024
 - Activitate laborator 20% - 1 data pe sapt; laboratorul cu sarcina efectuata se incarca in saptamana curenta! (in assignment dedicat)
- SPE+CIG+MK+ECTS+FB+EAI etc. – Disciplina Optionala
 - Proiect : 80% (cerinte minime proiect – vezi Moodle)
 - Examinat prin sustinere în perioada 8-19 ian. 2024
 - Activitate laborator 20% - 1 data la doua sapt; laboratorul cu sarcina efectuata se incarca in saptamana curenta!) (in assignment dedicat marcat cu OPT)
- Promovare examen: **minim 5** la fiecare probă de examen
- !!Atentie – media ponderata ≥ 4.5 (ex. $0.4*5+0.4*5+0.2*1=4.2$)



Cerinte minime pentru proiect IE

- 1. Realizarea unui sistem informatic (cu o tematica unitara – ex. Sistem informatic pentru o clinica stomatologica) care va contine doua proiecte de urmatoarele tipuri utilizand tehnologiile indicate:
 - I. aplicatie web (ASP.NET Core) ex. Programare pacienti; Review medic dupa efectuarea consultatiei/tratamentului
 - II. aplicatie mobila (.NET MAUI) ex. Programare pacienti; Trimitere notificare cu o zi inainte de data programata
- 2. Structura bazei de date pentru intreg sistemul informatic: existenta a cel putin 5 tabele cu relatii intre ele. Nu e obligatoriu ca cele 5 tabele sa fie utilizate toate in fiecare proiect ci se pot utiliza doar o parte din ele/proiect unele din tabele putand fi comune celor doua proiecte (minim 2-3 tabele/tip proiect)
- 3. Crearea interfetei pentru fiecare proiect si executarea de operatii CRUD (Create, Read, Update, Delete) asupra TUTUROR tabelelor din fiecare proiect conform relatiilor dintre aceste tabele -> adaugare, modificare, stergere, vizualizare a datelor
- 4. Validarea datelor de intrare pentru fiecare tip de proiect
- 5. Autentificare si autorizare
- Observatii:
 - 1. TEMATICA TREBUIE SA FIE DIFERITA DE CEA DIN LABORATOARE! (Atentie proiectul va fi notat cu nota 1 daca doar traducem numele de entitati din engleza in romana (ex. Books vs. Carti sau utilizam entitati foarte similare ex. Movies vs Books) !)
 - 2. Proiectul se realizeaza intr-o ECHIPA formata din DOUA persoane
 - 3. Pentru fiecare tip de aplicatie aveti exemple de implementari in fisierele aferente laboratoarelor.
 - 4. Implementarea nu trebuie neaparat sa fie identica cu cea din laboratoare, dar e bun punct de sprijin
 - 5. Proiectele copiate primesc automat nota 1



Cerinte minime pentru proiect (MPD ca disciplina OPTIONALA):

- 1.Realizarea unei aplicatii pe o tematica la alegere, tematica diferita fata de exemplele din laboratoare
- 2.Tipul aplicatiei la alegere: aplicatie web (ASP.NET Core)/aplicatie mobila (.NET MAUI)
- 3.Structura bazei de date: existenta a cel putin 3-4 tabele cu relatii intre ele
- 4.Crearea interfetei in functie de tipul aplicatiei alese si executarea de operatii CRUD (Create, Read, Update, Delete) asupra TUTUROR tabelelor conform relatiilor dintre aceste tabele -> adaugare, modificare, stergere, vizualizare a datelor
- 5. Implementarea unui mecanism de autentificare pentru aplicatiile web
- Observatii:
 - 1.TEMATICA TREBUIE SA FIE DIFERITA DE CEA DIN LABORATOARE! (Atentie proiectul va fi notat cu nota 1 daca doar traducem numele de entitati din engleza in romana (ex. Books vs. Carti sau utilizam entitati foarte similare ex. Movies vs Books) !)
 - 2. Proiectul se realizeaza INDIVIDUAL
 - 3.Pentru fiecare tip de aplicatie aveti exemple de implementari in fisierele aferente laboratoarelor.
 - 4.Implementarea nu trebuie neaparat sa fie identica cu cea din laboratoare, dar e bun punct de sprijin
 - 5.Proiectele copiate primesc automat nota 1



Lucrare licenta

- [Ghidul lucrărilor de licență cu specific de Informatică Economică](#)

<https://econ.ubbcluj.ro/departamente/studenti.php?c=4>

Tipuri:

- A. Proiectare și implementare de aplicații (Web, desktop, mobile etc.) sau de componente software
 - B. Cercetare științifică orientată empiric (formularea de ipoteze de cercetare și validarea/testarea lor prin analiză non-trivială de date colectate sau experimente)
 - C. Studii de caz privind administrarea și configurarea de medii/platforme IT
- Sablon de redactare FSEGA
- https://econ.ubbcluj.ro/n1.php?id_m=14&m=Examen%20finalizare%20studii



Concurs participate CodeCamp

https://codecamp.ro/conferences/codecamp_cluj-napoca/

Conference

Codecamp_Cluj-Napoca

📅 2 November 2023

📍 Cluj-Napoca 🎫 Ticket: ~~99 euro~~ 79 euro

[register](#)

Cluj Innovation Park
Strada Franklin Delano Roosevelt 2/16, Cluj-Napoca 400227,
Romania

Agenda

9:45 – 10:00

Hello, Cluj!

10:00 – 10:45

Technical Neglect

Kevin Henney

11:00 – 11:45

Rediscovering JavaScript

Vincent Stransman

12:00 – 12:45

Testing Infrastructure as Code

Radu Vornicu

12:45 – 14:00

Lunch break

14:00 – 14:45

The road to hell is paved with agile mindsets, empowerment, and embracing change

James Coplan

15:00 – 15:45

The Intersection of Architecture and Implementation

Mark Richards

16:00 – 16:45

Building Evolutionary Architectures

Neal Ford

16:45 – 17:00

Till next time, Cluj!



UNIVERSITATEA
BABEȘ-BOLYAI

CONCURS : "Ghiceste pentru 10!"

- Studentii care participa activ la cursul de Medii de programare vor primi pe emailul institutional (stud.ubbcluj.ro) cate o litera pentru fiecare activitate notata in cadrul intalnirilor de la curs.
- Litera primita face parte dintr-un cuvant. Cuvantul reprezinta un termen/notiune invatata pe parcursul semestrului 1 la disciplina Medii de programare si dezvoltare.
- Primii 3(4) studenti (in functie de data si ora email-ului) care trimit raspunsul corect pe email florina.covaci@econ.ubbcluj.ro pana in Ian 2024 primesc nota 10 la proba teoretica pentru disciplina Medii de programare si dezvoltare.
- Pentru a gasi si trimite raspunsul puteti lucra individual sau in echipe de maxim 2 persoane. Pentru ca echipa sa fie valida, fiecare membru din echipa trebuie sa fii primit pe email minim o litera.
- Raspunsurile vor fi trimise utilizand email-ul institutional (stud.ubbcluj.ro). In cazul in care lucratii individual raspunsul corect va fi atribuit studentului caruia ii apartine adresa de email. In cazul in care lucratii in echipa, studentul care trimite emailul va mentiona in email si componenta echipei.
- Succes!



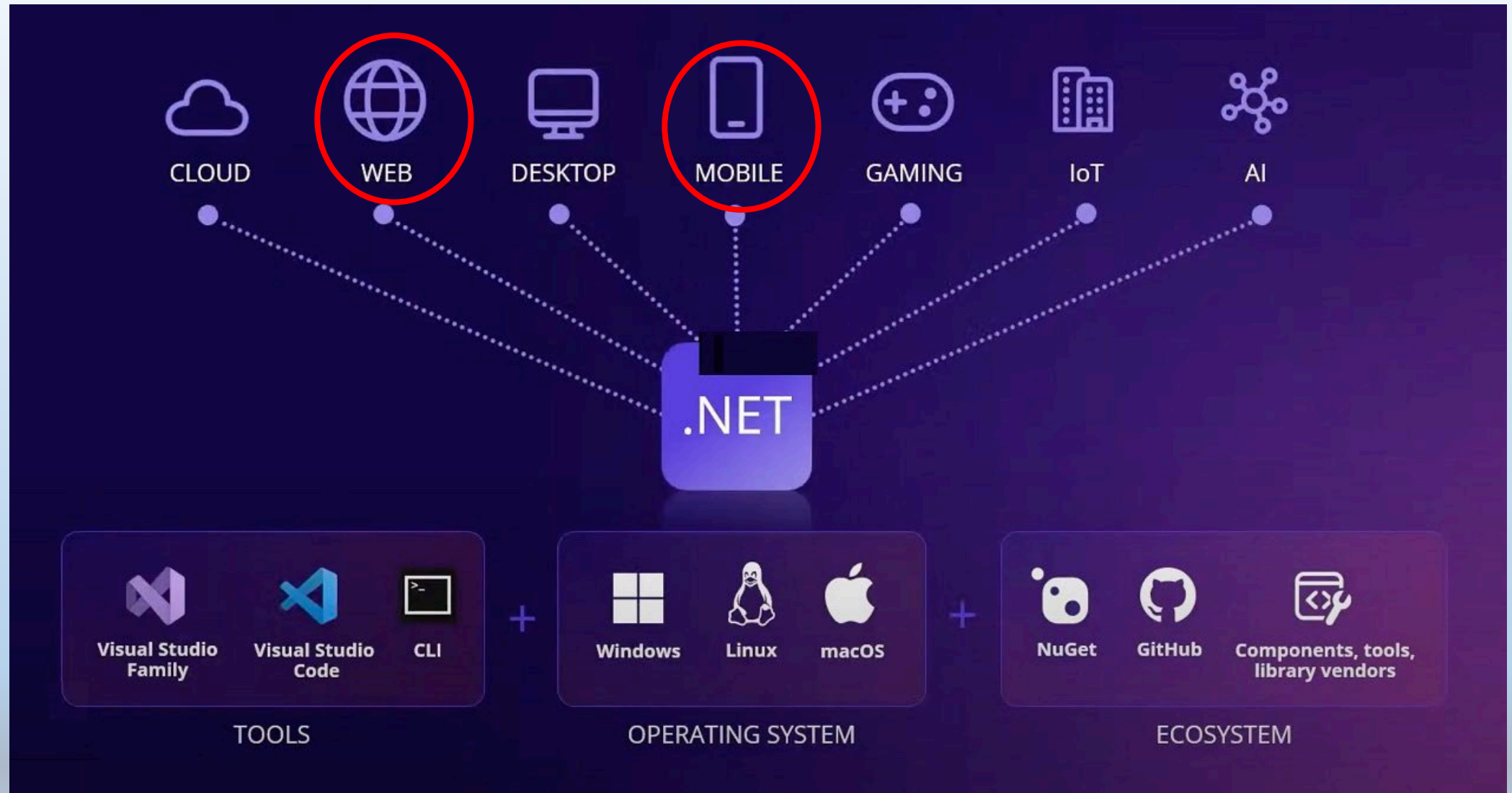
ASP.NET

- ASP.NET - O unealta Microsoft C# pentru dezvoltarea de aplicatii web
- Competitori: Java Spring, PHP Laravel, Node.js Express
- .NET Platforma de dezvoltare Microsoft din anul 2001
- ASP – Active Server Pages- pagini web dinamice, de cele mai multe ori conectate la o baza de date
- Core – open source, cross-platform

ASP.NET Core

- Necesitatea ca ASP.NET si Entity Framework sa ruleze pe alte SO in afara de Windows
- Arhitectura care usureaza testarea
- Integrarea de framework-uri client-side – Blazor –creare de UIs utilizand C# in loc de JavaScript.
- Suporta instalarea de versiuni diferite ale .Net Core – side by side versioning





Tehnologii Web

- HTML
- CSS
- JavaScript si TypeScript
- Librarii de scripting

HTML si CSS

- Cascade Style Sheets - definesc felul in care arata pagina web
- HTML tagul pentru list item definea modul in care este vizualizat-cerc,disc,patrat ->CSS
- Se pot utiliza template-uri Bootstrap - o colectie de conventii CSS si HTML care pot fi adaptate facil
- www.getbootstrap.com - documentatie si template-uri

JavaScript si TypeScript

- Putem modifica dynamic elemente client-side
- ECMAScript – standard care defineste functionalitatile curente si vitoare pentru JavaScript.
- Implementarea Microsoft pentru JavaScript se numeste JScript.
- TypeScript similar cu JavaScript. Sintaxa TypeScript se bazeaza pe JavaScript, dar aduce noi functionalitati (ex. Cod puternic tipizat, adnotari)
- Documentatie la : www.typescriptlang.org.

Librarii de scripting

- Pot fi utilizate server-side impreuna cu ASP.NET Core
- jQuery (<http://www.jquery.org>) –utilizata pentru a gestiona unitar modul in care diferite browsere gestioneaza evenimentele
- Angular (<https://angular.io>) librerie de la Google bazata pe patternul MVC creata pentru a simplifica dezvoltarea si testarea aplicatiilor web de tip single-page (Spre deosebire de ASP.NET MVC, Angular ofera patternul MVC in codul client-side)
- React (<https://reactjs.org>) librerie de la FaceBook care ofera functionalitati prin care se pot actualiza facil interfetele utilizator pe masura ce datele se modifica in background
- Visual Studio ofera template-uri pentru Angular si React

Aplicatii Web cu ASP.NET Core

- De tip Server-side
 - Razor Pages
 - MVC
- De tip Client-side
 - Blazor
 - Angular
 - React

Curs 2

Pagini Razor și sintaxa

Structura unei aplicatii Web – cu Razor

- In directorul `wwwroot` regasim continut de tip client-side CSS, JavaScript, imagini, si orice alt continut non-programatic
- Directorul `Pages` contine pagini Razor si fisiere suport. Fiecare pagina este o pereche a urmatoarelor fisiere:
 - Un fisier `.cshtml` care contine markup Razor, HTML si cod C#.
 - Un fisier `.cshtml.cs` care contine cod C# pentru a gestiona evenimentele la nivel de pagina.
- Fisierele suport au nume care incepe cu “_”. Ex fisierul “_Layout.cshtml” configureaza elemente UI comune tuturor. Ex. acest fisier seteaza navigation menu in partea de sus a paginii si informatia despre copyright in partea de jos a paginii.

Pagini Layout

- Majoritatea site-urilor prezintă același conținut pe fiecare pagină sau într-un număr mare de pagini (ex. antet, subsol, bara meniu de navigare, scripturile, css etc.)
- Adăugarea aceluiași conținut încalcă principiul DRY (Don't Repeat Yourself) => trebuie să modificați aspectul antetului, trebuie să editați fiecare pagină
- Pagina Layout - acționează ca un șablon pentru toate paginile care fac referire la aceasta.
- Paginile care fac referire la pagina de Layout se numesc pagini de conținut. Paginile de conținut nu sunt pagini web complete. Acestea conțin doar conținutul care variază de la o pagină la alta.

Exemplu pagină de Layout foarte simplă:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<title></title>
<link href="/css/site.css" rel="stylesheet" type="text/css" />
</head>
  <body>
    @RenderBody()
  </body>
</html>
```

Fisiere configurare aplicatii Web – cu Razor

- **appSettings.json**
- Contine date de configurare, precum string-uri de conexiune

- **Program.cs**

Punctul de start al aplicatiei.

Sintaxa Razor

- Razor este o sintaxa de tip markup pentru a incorpora cod de tip server-side in pagini web
- Sintaxa Razor consta in Razor markup, C#, si HTML
- Limbajul Razor implicit este HTML. Interpretarea HTML din markup-ul Razor este similara cu interpretarea HTML dintr-un fisier HTML.
- Pentru a trece la limbaj C# - simbolul @ - Razor evalueaza expresiile C# si le randeaza in output HTML
- Cand simbolul @ este urmat de un cuvnt cheie Razor – tranzitia se face la markup specific Razor

Expresii implicite/explicite Razor

- Expresii implicite

<p>@DateTime.Now</p>

<p>Last week: @DateTime.Now - TimeSpan.FromDays(7)</p>

Se afiseaza

<p>Last week: 7/7/2016 4:39:52 PM - TimeSpan.FromDays(7)</p>

- Expresii explicite ()

<p>Last week this time: @(DateTime.Now - TimeSpan.FromDays(7)) </p>

Continutul @() este evaluat si afisat

Expresii explicite

- Expresiile explicite pot fi utilizate pentru a concatena text cu un rezultat al unei expresii

@

{

```
var joe = new { Name="Joe", Age=33 };
```

}

<p>Age@joe.Age</p>

e interpretata ca o adresa de email , se afiseaza <p>Age@joe.Age</p>

<p>Age@(joe.Age)</p> se afiseaza <p>Age33</p>

Functii locale in blocuri de cod

- Se pot declara functii locale cu markup pentru a fi utilizate ca metode template

```
@{  
void RenderName(string name)  
{ <p>Name: <strong>@name</strong></p>  
}  
RenderName("John Keller");  
RenderName("Martin Johnson");  
}  
<p>Name: <strong>John Keller</strong></p>  
<p>Name: <strong>Martin Johnson</strong></p>
```

Tranzitie implicita

- Limbajul implicit intr-un bloc de cod este C#, dar poate face implicit tranzitia la HTML

```
@{  
var inCSharp = true;  
<p>Now in HTML, was in C# @inCSharp</p>  
}
```

Tranzitie explicita

- Delimitata - Pentru a defini o sectiune dintr-un bloc de cod care trebuie sa afiseze HTML

```
@for (var i = 0; i < people.Length; i++)  
{ var person = people[i];  
<text>Name: @person.Name</text>  
}
```

- Inline - @:

```
@for (var i = 0; i < people.Length; i++)  
{  
var person = people[i];  
@:Name: @person.Name  
}
```

Structuri de control

- Conditionale

```
@if (value % 2 == 0)
```

```
{ <p>The value was even.</p> }
```

```
else
```

```
if (value >= 1337)
```

```
{ <p>The value is large.</p> }
```

```
else
```

```
{ <p>The value is odd and small.</p> }
```

Structuri de control - ciclice

```
@{  
    var people = new Person[]  
    { new Person("Weston", 33),  
      new Person("Johnathon", 41),  
      ... };  
}  
  
@for (var i = 0; i < people.Length;  
i++)  
{ var person = people[i];  
<p>Name: @person.Name</p>  
<p>Age: @person.Age</p>  
}
```

```
@foreach (var person in people)  
{ <p>Name: @person.Name</p>  
  <p>Age: @person.Age</p>  
}
```

Clasa PageModel

- Scopul: de a oferi o separare clară între stratul UI (fișierul de vizualizare .cshtml) și logica de procesare a paginii.
- Beneficii:
 - Reduce complexitatea stratului UI, făcându-l mai ușor de întreținut.
 - Permite o mai mare flexibilitate pentru echipe, deoarece un membru poate lucra la vizualizare, în timp ce altul poate lucra la logica de procesare.
- Clasa PageModel este declarată într-un fișier de clasă separat - un fișier cu extensia .cs
- Prin conventie numele clasei este denumit după șablonul <NumePagina>Model - O clasă PageModel pentru About.cshtml va fi numită Model și va fi generată într-un fișier numit About.cshtml.cs.



PageModel

- gestionează o solicitare pentru o anumită pagină sau acțiune pentru o aplicație web (echivalent cu rolul Controller-ului în MVC)
- O clasă Razor PageModel este o implementare a pattern-ului Page Controller-caracterizat prin faptul că există o mapare unu-la-unu între pagini și controlerele acestora.
- Rol:
 - a accepta input de la pagină
 - de a se asigura că toate operațiunile solicitate asupra modelului (date) sunt aplicate
 - de a determina vizualizarea corectă de utilizat pentru pagina rezultată.

Directive Razor

```
@page  
@model IndexModel  
{  
}
```

- **@page** - permite ca pagina sa poata gestiona cereri
- trebuie sa fie prima directiva care apare in fisier
- **@model** – specifica modelul transmis catre pagina Razor
- reprezentat de clasa derivata din PageModel

Clasa PageModel

Mosteneste clasa de baza PageModel

Create.cshtml - Create.cshtml.cs

```
public class CreateModel : PageModel
{
    public IActionResult OnGet()
    {
        return Page();
    }

    [BindProperty]
    public Book Book { get; set; }
}
```

- Procesarea cererilor se realizeaza prin metode handler (On<verb> cu Async ataşat opţional):
 - OnPost/OnPostAsync – ruleaza cand exista cereri POST (cand utilizatorul trimite un formular)
 - OnGet/OnGetAsync – initializeaza starea paginii
- Se pot adauga metode handler pentru orice verb HTTP (PUT, DELETE etc.)
- Sufixul Async este optional dar este folosit adeseori pentru a indica faptul că metoda este destinată să ruleze asincron
- Proprietatile clasei PageModel sunt disponibile ca si proprietati ale modelului in Pagina Razor
- Create.cshtml `<input asp-for="Book.Title" class="form-control" />`



Metode Handler in Pagini Razor

- OnGet si OnGetAsync reprezinta aceasi metoda handler-> nu pot exista ambele in aceeaasi pagina.

Se va arunca exceptie: InvalidOperationException: Multiple handlers matched. The following handlers matched route data and had all constraints satisfied: OnGetAsync(), OnGet()

- Parametrii nu joacă nici un rol în dezambiguizarea între handler-e bazate pe aceeași metodă HTTP, în ciuda faptului că compilatorul o va permite. Prin urmare, aceeași excepție va fi ridicată chiar dacă metoda OnGet preia parametri și metoda OnGetAsync nu
- Metodele de gestionare trebuie să fie publice și pot returna *void*, *Task* dacă sunt asincrone sau *IActionResult* (sau *Task<IActionResult>*).

HTTP este stateless

```
public class IndexModel : PageModel
{
    public string Message { get; set; }
    public void OnGet()
    {
        Message = "Get used";
    }
    public void OnPost()
    {
        Message = "Post used";
    }
}
```

Orice valoare inițializată în handlerul OnGet nu este disponibilă în handlerul OnPost

```
<h3>@Message</h3>
```

```
<form method="post"><button class="btn btn-  
default">Click to post</button></form>
```

```
<p><a href="/" class="btn btn-default">Click to Get</a></p>
```

Metode Handler cu parametrii

- Delete.cshtml.cs

```
public IActionResult OnPost(int? id)
{
}
```

- Delete.cshtml

```
<form method="post">
    <input type="hidden" asp-for="Book.id" />
    <input type="submit" value="Delete" class="btn btn-danger" /> |
    <a asp-page="./Index">Back to List</a>
</form>
```

ActionResult

```
public IActionResult OnGet (int? id)
{
    if (id == null || _context.Book == null)
    {
        return NotFound();
    }

    var book = _context.Book.FirstOrDefault (m => m.ID == id);
    else
    {
        Book = book;
    }
    return Page();
}
```

ActionResult - tip de returnare a metodelor handler; sunt responsabile pentru generarea de răspunsuri și coduri de stare adecvate (clasa abstractă Microsoft.AspNetCore.Mvc.ActionResult sau interfața Microsoft.AspNetCore.Mvc.IActionResult)

-returnarea conținutului unei pagini Razor (PageResult), redirectionarea către o altă resursă (de exemplu, RedirectResult) sau pur și simplu returnează un anumit cod de stare HTTP (ex. NotFoundResult, OkResult).

Pagina Layout

- Pages/Shared/_Layout.cshtml
- Furnizeaza un Layout unitar pentru intreaga aplicatie
- Permite containerului HTML pentru layout
 - Sa fie specificat intr-un singur loc
 - Sa fie utilizat in pagini multiple
- @RenderBody ()- placeholder care permite afisare continutului specific pentru fiecare pagina

Setarea Layout-ului

- Se realizeaza in Pages/_ViewStart.cshtml

```
@{
```

```
    Layout = "_Layout";
```

```
}
```

- Seteaza pentru toate paginile din directorul Pages, layout-ul definit in Pages/Shared/_Layout.cshtml

ViewData si Layout

```
@{  
    ViewData["Title"] = "Home page";  
}
```

- Clasa de baza Pagemodel contine o proprietate de tip dictionar, care este utilizata pentru a trimite date la un View
- Obiectele sunt adaugate folosind perechi de tipul cheie/valoare
- Proprietatea Title este adaugata la dictionarul ViewData

ViewData si Layout

- Proprietatea title este folosita in Pages/Shared/_Layout.cshtml

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="utf-8" />
```

```
    <meta name="viewport" content="width=device-  
width, initial-scale=1.0" />
```

```
    <title>@ViewData["Title"] - Sample App</title>
```

ViewData

- container pentru a trimite date de la PageModel la pagina de continut
- dictionar de obiecte cu chei de tip string

```
public class IndexModel :  
    PageModel  
{  
    public void OnGet()  
    {  
        ViewData["MyNumber"] = 42;  
        ViewData["MyString"] = "Hello  
        World";  
    }  
}
```

```
@page  
@model IndexModel  
@{  
    }  
    <h2>@ViewData["MyString"]</h2>  
    <p>The correct number is @ViewData["MyNumber"]  
    </p>
```

ViewBag

Un wrapper asupra dictionarului ViewData

Ofera o alternativa pentru accesarea continutului din ViewData utilizand proprietati in locul cheilor de tip string

```
@page
```

```
@model IndexModel
```

```
@{
```

```
    ViewBag.Title = "My Home Page";
```

```
}
```

```
<title>@ViewBag.Title</title>
```

```
<title>@ViewData["Title"]</title>
```

Curs 3

Accesul la date – Entity Framework Core

Entity Framework - ADO.NET API

- Introdus începând cu versiunea .NET 3.5
- EF Core – versiune multi-platforma a lui EF
- EF Core – (O/RM)- Object-Relational Mapper:
 - Permite dezvoltatorilor .NET sa lucreze cu bd utilizand obiecte .NET
 - Elimina necesitatea scrierii de cod specific pentru accesul la date
- Un set de date - o colecție de rânduri și coloane => colecție obiecte tipizate denumite entități
- Aceste entități pot fi interogate utilizând LINQ, motorul EF va traduce LINQ în interogări SQL.



Rolul entitatilor

- Entitatile reprezinta un model conceptual al bazei de date - EDM (Entity data model)
- EDM – un set de clase client-side care sunt mapate la o baza de date pe baza conventiilor definite in EF si pe baza unor configurari
- Creare Model – abordari:
 - Generarea unui model dintr-o baza de date existenta
 - Scrierea de mana a claselor care reprezinta modelul -> Migrarea pentru a crea baza de date din model.

Componente EF

1. Clasa DbContext - utilizata pentru interogarea bazei de date si pentru a grupa modificarile pentru a putea fi scrise in bloc
 - Metoda SaveChanges – salvează in baza de date toate modificările făcute in context. Returneaza numarul de entitati efectuate
 - Proprietatea Database – ofera un mechanism pentru creare/stergerea/verificarea bazei de date, execută proceduri stocate si expune funcționalități legate de tranzacții
 - Evenimente SavingChanges – se apeleaza când modificările se salveaza în baza de date, dar înainte de a deveni persistente



Componente EF

2. Clasa derivată din DbContext – se trimite la constructor numele string-ului de conexiune pentru clasa context

```
public Nume_Pren_Lab2Context (DbContextOptions<Nume_Pren_Lab2Context>  
options)  
    : base(options)  
{  
  
}
```

Componente EF

3. Entity Sets DbSet<T> - adaugarea de tabele in context

- `public DbSet<Nume_Pren_Lab2.Models.Book> Book { get; set; }`
- `public DbSet<Nume_Pren_Lab2.Models.Publisher> Publisher { get; set; }`
- `public DbSet<Nume_Pren_Lab2.Models.Category> Category { get; set; }`

Membrii DbSet<T>:

- Add – adaugarea unui obiect in colectie ; acestea vor fi marcate cu starea Added si vor fi inserate in baza de date cand se apeleaza SaveChanges pentru DbContext
- Find – gasește un rand dupa cheia primară și returnează un obiect reprezentând acel rând
- Remove – marcheaza un obiect pentru ștergere

Componente EF

4. DbChangeTracker - realizează tracking-ul automat al stării oricărui obiect DbSet<T> în cadrul unui DbContext

Stările entităților:

- Detached – obiectul există, dar nu se face încă tracking pe el; se afla în această stare imediat ce a fost creat și înainte să fie adăugat la obiectul context
- Unchanged – obiectul nu a fost modificat de când a fost atașat la context sau de la ultimul apel a lui SaveChanges()
- Added – obiectul este nou și a fost adăugat la obiectul context, iar metoda SaveChanges() nu a fost apelată.
- Deleted – obiectul a fost sters marcat pentru stergere
- Modified – una din proprietățile obiectului a fost modificată și metoda SaveChanges() nu a fost apelată

Componente EF

5. Adnotări – reprezintă attribute utilizate pentru modelarea entităților in vederea maparii cu bd

- Key – definește cheia primară. Nu este necesara daca proprietatea se numeste Id sau combina numele clasei cu Id- ex.
- Required –proprietățile nu pot lua valori null
- ForeignKey –definește o proprietate care este utilizată ca și cheie străină
- NotMapped – o proprietate nu este mapată pe un câmp al bazei de date
- ConcurrencyCheck – marchează un camp pentru a fi verificat in cazuri de concurență când se realizează inserări, actualizări sau ștergeri
- Table/ Column – permite numirea claselor si campurilor diferit față de numele din baza de date. Atributul table permite specificarea inclusiv a numelui schemei bazei de date
- DatabaseGenerated – specifică faptul că un câmp este generat din baza de date cum ar fi Identity



Modelarea relatiilor intre entitati

- O relatie defineste modul in care doua entitati se raporteaza una la cealaltata
- Intr-o baza de date relationala ->constrangere de cheie straina

Definire termeni

- Entitate dependenta : O entitate care contine proprietati de tip cheie straina. Este referita uneori ca si entitatea copil din relatie
- Entitate principala: Entitatea care contine proprietatea de tip cheie primara. Este referita uneori ca si entitatea parinte din relatie
- Navigation property : O proprietate definita in entitatea principala si/sau dependenta care referentiaza entitatea relationata
 - **Collection navigation property:** Contine referinta la mai multe entitati
 - **Reference navigation property:** Contine referinta la o singura entitate

Entitate principala-dependenta

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public List<Post> Posts { get; set; }
}
```

```
public class Post {
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public int BlogId { get; set; }
    public Blog Blog { get; set; }
}
```

Relatii definite complet

- Avem navigation properties definite la ambele capete ale relatiei si o proprietate de tip cheie straina definite in entitatea dependenta

```
public class Blog {  
    public int BlogId { get; set; }  
    public string Url { get; set; }  
    public List<Post> Posts { get; set; }  
}  
  
public class Post {  
    public int PostId { get; set; }  
    public string Title { get; set; }  
    public string Content { get; set; }  
    public int BlogId { get; set; }  
    public Blog Blog { get; set; }  
}
```


No foreign key property

- Desi este recomandat sa definim o proprietate de tip cheie straina in entitatea dependenta, nu este obligatoriu
- Daca nu este gasita o cheie straina se creaza automat o proprietate de tip cheie straina shadow
- Proprietatile shadow – proprietati care nu sunt definite in clasa entitate .NET dar sunt definite pentru acea entitate in modelul EF.
 - valoarea si starea acelor entitati sunt gestionate de ChangeTracker

No foreign key property

```
public class Blog
{ public int BlogId { get; set; }
  public string Url { get; set; }
  public List<Post> Posts { get; set; }
}

public class Post {
  public int PostId { get; set; }
  public string Title { get; set; }
  public string Content { get; set; }
  public Blog Blog { get; set; }
}
```

Configurare manuala prin Fluent API

```
class MyContext : DbContext {  
    public DbSet<Blog> Blogs { get; set; }  
    public DbSet<Post> Posts { get; set; }  
    protected override void OnModelCreating(ModelBuilder modelBuilder)  
    {  
        modelBuilder.Entity<Post>()  
            .HasOne(p => p.Blog)  
            .WithMany(b => b.Posts);  
    }  
}
```

Cascade Delete

```
protected override void OnModelCreating(ModelBuilder
modelBuilder)
{
    modelBuilder.Entity<Post>()
        .HasOne(p => p.Blog)
        .WithMany(b => b.Posts)
        .onDelete(DeleteBehavior.Cascade);
}
```

Relatie one-to-one

-au un reference navigation property la ambele capete ale relatiei

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
    public BlogImage BlogImage {
        get; set; }
}
```

```
public class BlogImage
{
    public int BlogImageId { get;
        set; }
    public byte[] Image { get; set;
        }
    public string Caption { get;
        set; }
}
```

```
public int BlogId { get; set; }
public Blog Blog { get; set; } }
```

Configurare manuala cu Fluent API

```
class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
    public DbSet<BlogImage> BlogImages { get; set; }
    protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<Blog>()
            .HasOne(b => b.BlogImage)
            .WithOne(i => i.Blog)
    }
}
```

Relatie many-to-many

-necesita un collection navigation property la ambele capete

```
public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }
    public ICollection<Tag> Tags { get; set; }
}
```

```
public class Tag
{
    public string TagId { get; set; }
    public ICollection<Post> Posts {
        get; set; }
}
```

Relatie many-to-many

entitate de tip join

```
public class Post
{ public int PostId { get; set; }
  public string Title { get; set; }
  public string Content { get; set; }
  public List<PostTag> PostTags { get;
  set; }
}
```

```
public class Tag
{ public string TagId { get; set; }
  public List<PostTag> PostTags { get;
  set; }
}
```

```
public class PostTag {
  public DateTime PublicationDate {
  get; set; }
  public int PostId { get; set; }
  public Post Post { get; set; }
  public string TagId { get; set; }
  public Tag Tag { get; set; }
}
```


Configurare manuala cu Fluent API

```
public class MyContext : DbContext
{
    public MyContext(DbContextOptions<MyContext> options) : base(options) {
    }

    protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<PostTag>()
            .HasKey(t => new { t.PostId, t.TagId });

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Post)
            .WithMany(p => p.PostTags)
            .HasForeignKey(pt => pt.PostId);

        modelBuilder.Entity<PostTag>()
            .HasOne(pt => pt.Tag)
            .WithMany(t => t.PostTags)
            .HasForeignKey(pt => pt.TagId); } }
```

Migrarea

- In procesul de dezvoltarea a unei aplicatii modelul se schimba frecvent si nu mai este sincronizat cu baza de date
- La modificarea modelului-adaugare, stergere, modificare de entitati stergem baza de date si EF creaza o noua baza de date corespunzatoare modelului si apeleaza Seed Data
- In productie avem date in baza de date -> nu putem sterge baza de date
- EF Migration – actualizeaza baza de date

Add-Migration ExtendedModel

- EF genereaza cod care va creaza baza de date de la 0
- Directorul Migrations - fisier *<timestamp>_ExtendedModel.cs*

```
public partial class ExtendedModel : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Book",
            columns: table => new
            {....
        }
    }
}
```

Remove-Migration ExtendedModel

```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Books");
}
```

Metoda Down se apeleaza cand fac rollback la migrare

SnapShot pentru model

- Migrarea creaza un *snapshot* a schemei bazei de date curente
- Directorul *Migrations/LibraryContextModelSnapshot.cs*
- Cand creem o noua migrare, EF determina ce s-a modificat comparand modelul curent cu modelul din snapshot

Curs 4

LINQ – Language Integrated Query



LINQ

- Motivatie LINQ
 - Aplicatiile vor accesa date la un anumit moment in timpul executiei
 - Datele pot fi regasite in fisiere XML, bd relationale, colectii in memorie, siruri
- LINQ – introdus in .NET 3.5 – ofera un modalitate puternic tipizata de a accesa date in diverse formate prin intermediul unui layer de abstractizare

Structuri sintactice specifice

- LINQ poate fi inteles ca un limbaj de interogare puternic tipizat incorporat in gramatica C#
- Putem construi expresii asemanatoare cu interogările SQL, dar interogările SQL se pot aplica unor surse de date diferite inclusiv celor care nu au nimic de a face cu bd relationale
- Desi interogările LINQ sunt similare cu interogările SQL, sintaxa nu este identica. De fapt multe interogari LINQ folosesc un format diferit de cel utilizat la bd
- Nu incercam sa mapam sintaxa LINQ to SQL, ci mai degraba le privim ca interogari care “prin coincidenta” seamana cu SQL

Mecanisme LINQ

- Cand LINQ a fost introdus in .NET, limbajul C# continea o serie de mecanisme pe care se bazeaza tehnologia LINQ
- Astfel, limbajul C# utilizeza urmatoarele mecanisme care se afla la baza LINQ:
 - Variabile cu tip implicit
 - Sintaxa de initializare a obiectelor/colectiilor
 - Expresii Lambda
 - Metode extinse
 - Tipuri anonime

Variabile cu tip implicit

- Cuvantul cheie **var** permite declararea unei variabile locale fara a specifica explicit tipul variabilei
- Totusi, variabila este puternic tipizata, deoarece compilatorul va determina tipul de date corect bazandu-se pe atribuirea valorii

```
static void DeclareImplicitVars()
{
    // Implicitly typed local variables.
    var myInt = 0;
    var myBool = true;
    var myString = "Time, goes on...";
    // Print out the underlying type.
    Console.WriteLine("myInt is a: {0}", myInt.GetType().Name);
    Console.WriteLine("myBool is a: {0}", myBool.GetType().Name);
    Console.WriteLine("myString is a: {0}", myString.GetType().Name);
}
```

- Acest feature este foarte util cand utilizam LINQ – multe interogari LINQ vor returna o secventa de tipuri de date, care nu sunt cunoscute pana la momentul compilarii, deci nu vom putea declara tipul unei variabile explicit

Sintaxa de initializare a obiectelor/colectiilor

- Sintaxa de initializare colectie pentru a umple o lista `List<T>` cu obiecte `Rectangle`, fiecare avand doua obiecte `Point` care reprezinta o pozitie determinata de doua coordonate (x,y):

```
List<Rectangle> myListOfRects = new List<Rectangle>
{
    new Rectangle {TopLeft = new Point { X = 10, Y = 10 },
    BottomRight = new Point { X = 200, Y = 200}},
    new Rectangle {TopLeft = new Point { X = 2, Y = 2 },
    BottomRight = new Point { X = 100, Y = 100}},
    new Rectangle {TopLeft = new Point { X = 5, Y = 5 },
    BottomRight = new Point { X = 90, Y = 75}}
};
```

- Aceasta sintaxa combinata cu variabilele cu tip implicit ne permit declararea de tipuri anonime, utile la crearea de proiectii LINQ
- Proiectii LINQ- transformarea unui obiect intr-o forma noua care de obicei consta intr-un subset de proprietati. Se pot realiza proiectii si cu obiectul original fara modificari

Expresii Lambda

- Utilizam expresii lambda pentru a crea o functie anonima
- operator (\Rightarrow) permite construirea unei expresii lambda- separa lista de parametrii de corpul functiei
- $(\text{ArgumentsToProcess}) \Rightarrow \{ \text{StatementsToProcessThem} \}$
- Expresiile lambda simplifica modul de lucru in .NET, reducand numarul de linii de cod care trebuie scrise

Metode extinse

- Metode extinse permit adaugarea de functionalitati la clasele existente fara a folosi mostenirea
- La crearea unei metode extinse primul parametru este calificat cu cuvantul cheie **this** marcand tipul care se extinde.
- Metodele extinse trebuie definite in cadrul unei clase statice si trebuie declarate folosind cuvantul cheie **static**

```
namespace MyExtensions
{
    public static class IntExtensions {
        public static bool IsGreaterThan(this int i, int value)
        { return i > value; }
    }
}

using MyExtensions;

class Program {
    static void Main(string[] args)
    { int i = 10;
      bool result = i.IsGreaterThan(100); Console.WriteLine(result);
    } }
```

Cand creem interogari LINQ, utilizam metode extinse definite in .NET.



Tipuri anonime

- Generarea definitiei unei clase la compilare prin specificarea unui set de perechi nume-valoare
- Pentru a defini un tip anonim declaram o variabila cu tip implicit si specificam datele folosind sintaxa de initializare a obiectelor

// Creaza un tip anonim compus din alt tip anonim

```
var purchaseItem = new {  
    TimeBought = DateTime.Now,  
    ItemBought = new {Color = "Red", Make = "Saab", CurrentSpeed = 55},  
    Price = 34.000};
```

- LINQ foloseste tipuri anonime cand dorim sa proiectam noi feluri de date “on the fly”
- O colectie de obiecte Persoana – dorim sa utilizam LINQ pentru a obtine info despre varsta si CNP
Folosind o proiectie LINQ, permitem compilatorului sa genereze un nou tip anonim care contine informatia dorita

Termenii LINQ

- Utilizand LINQ putem crea expresii de interogare folosind limbajul C#
- LINQ poate fi folosit in mai multe cazuri iar termenii folositi difera in functie de acest lucru:
 - *LINQ to Objects*: utilizarea interogarilor LINQ la siruri si colectii
 - *LINQ to XML*: utilizarea LINQ pentru a manipula si interoga documente XML
 - *LINQ to Entities*: utilizarea interogarilor LINQ in cadrul Entity Framework.
 - *Parallel LINQ (PLINQ)*: procesarea paralela a datelor returnate de o interogare LINQ

Interogari

- Specifica informatiile care trebuie incarcate din sursa de date
- Optional specifica cum trebuie sortate/grupate
- Sunt stocate intr-o variabila de interogare si initializate cu o expresie de interogare

```
var result = from matchingItem in container select matchingItem;
```

- Interogarile sunt separate de executia acestora

Sintaxa de baza

- Corectitudinea sintactica a unei interogarii LINQ este validata la compilare => ordinea operatorilor este importanta
- Expresiile LINQ sunt construite utilizand operatorii **from**, **in**, **select**
- Template-ul general:

```
var result = from matchingItem in container select matchingItem;
```
- Identificatorul aflat dupa operatorul **from** reprezinta un item care corespunde criteriilor introgarii LINQ – poate avea orice nume.
- Identificatorul aflat dupa operatorul **in** reprezinta containerul de date in care se face cautarea (sir, colectie, document XML, etc.).

Selectarea itemilor din container

```
static void
SelectEverything(ProductInfo[]
products)
{
    Console.WriteLine("All product
details:");
    var allProducts = from p in
products select p;
    foreach (var prod in
allProducts)
    {
        Console.WriteLine(prod.ToString
());
    }
}
```

```
static void
ListProductNames(ProductInfo[]
products)
{
    Console.WriteLine("Only
product names:");
    var names = from p in products
select p.Name;
    foreach (var n in names)
    {
        Console.WriteLine("Name: {0}",
n);
    }
}
```

Obtinerea de subseturi de date

- Template-ul general:

```
var result = from item in container where BooleanExpression select item;
```

```
static void GetOverstock(ProductInfo[] products)
{
    Console.WriteLine("The overstock items!");
    var overstock = from p in products where p.NumberInStock > 25
    && p.ExpDate=DateTime.Today.AddDays(7) select p;
    foreach (ProductInfo c in overstock)
    {
        Console.WriteLine(c.ToString());
    }
}
```

Proiectarea unor noi tipuri de date

```
static void
GetNamesAndDescriptions(ProductInfo[]
] products)
{
    Console.WriteLine("Names and
Descriptions:");
    var nameDesc = from p in products
select new { p.Name, p.Description
}; // tip de date anonim
foreach (var item in nameDesc)
{
    Console.WriteLine(item.ToString());
}}
```

- Cand interogarea LINQ creaza o proiectie nu va cunoaste tipul de date- e obligatorie utilizarea **var**

```
static var
GetProjectedSubset(ProductInfo[]
products)
{
    var nameDesc = from p in products
select new { p.Name, p.Description
};
    return nameDesc; // Nu!
}
• Nu putem scrie metode care
returneaza tipuri implicite
return nameDesc.ToArray(); return
type Array
```

Numarul de itemi returnati

```
static void GetCountFromQuery()
{
    string[] currentVideoGames = {"Morrowind", "Uncharted 2", "Fallout 3", "Daxter", "System Shock 2"};
    int numb = (from g in currentVideoGames where g.Length > 6 select g).Count();
    Console.WriteLine("{0} items for the LINQ query.", numb);
}
```

- Count() - metoda extinsa a clasei Enumerable

Inversarea ordinii din setul obtinut

```
static void ReverseEverything(ProductInfo[] products)
{
    Console.WriteLine("Product in reverse:");
    var allProducts = from p in products select p;
    foreach (var prod in allProducts.Reverse())
    {
        Console.WriteLine(prod.ToString());
    }
}
```

- Reverse() - metoda extinsa a clasei Enumerable

Sortarea

```
static void AlphabetizeProductNames(ProductInfo[] products)
{
    // produse in ordine alfabetica.
    var subset = from p in products orderby p.Name select p;
    Console.WriteLine("Ordered by Name:");
    foreach (var p in subset)
    {
        Console.WriteLine(p.ToString());
    }
}
```

- var subset = from p in products orderby p.Name **ascending** select p;
- var subset = from p in products orderby p.Name **descending** select p;

Gruparea

```
var queryCustomersByCity =  
    from cust in customers  
    group cust by cust.City;  
  
foreach (var customerGroup in queryCustomersByCity)  
{  
    Console.WriteLine(customerGroup.Key);  
    foreach (Customer customer in customerGroup)  
    {  
        Console.WriteLine(" {0}", customer.Name);  
    }  
}
```


Join

Gasim clientii si distribuitori care se afla in aceeasi locatie

```
var innerJoinQuery =  
    from cust in customers  
    join dist in distributors on cust.City equals dist.City  
    select new { CustomerName = cust.Name, DistributorName =  
dist.Name };
```

- creeaza asocieri intre secvente care nu sunt explicit modelate in sursa de date
- Clauza join se aplica colectiilor de obiecte nu direct tabelelor din bd
- Cheile straine din model – colectie de item-uri
- from order in Customer.Orders

Diferenta intre doua containere

```
static void DisplayDiff()
{
    List<string> myCars = new List<String> {"Yugo", "Aztec", "BMW"};
    List<string> yourCars = new List<String>{"BMW", "Saab", "Aztec" };
    var carDiff = (from c in myCars select c).Except(from c2 in yourCars
    select c2);
    Console.WriteLine("Here is what you don't have, but I do:");
    foreach (string s in carDiff)
    Console.WriteLine(s);}

```

- Except() - metoda extinsa a clasei Enumerable

Intersectia a doua containere

```
static void DisplayIntersection()
{
    List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
    List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec"
};
    // Get the common members.
    var carIntersect = (from c in myCars select c).Intersect(from c2 in
yourCars select c2);
    Console.WriteLine("Here is what we have in common:");
    foreach (string s in carIntersect)
        Console.WriteLine(s);}
• Intersect() - metoda extinsa a clasei Enumerable
```

Reuniunea

```
static void DisplayUnion()
```

```
{
```

```
List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
```

```
List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };
};
```

```
// Get the union of these containers.
```

```
var carUnion = (from c in myCars select c).Union(from c2 in yourCars select c2);
```

```
Console.WriteLine("Here is everything:");
```

```
foreach (string s in carUnion)
```

```
Console.WriteLine(s);
```

- Union() - metoda extinsa a clasei Enumerable

Concatenarea

```
static void DisplayConcat()
{
    List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
    List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };
    var carConcat = (from c in myCars select c).Concat(from c2 in yourCars
    select c2);
    foreach (string s in carConcat)
        Console.WriteLine(s);
}
```

- Concat() - metoda extinsa a clasei Enumerable

Eliminarea duplicatelor

```
static void DisplayConcatNoDups()
{
    List<string> myCars = new List<String> { "Yugo", "Aztec", "BMW" };
    List<string> yourCars = new List<String> { "BMW", "Saab", "Aztec" };
    var carConcat = (from c in myCars select c).Concat(from c2 in
yourCars select c2);
    foreach (string s in carConcat.Distinct())
        Console.WriteLine(s);
}
```

- Distinct() - metoda extinsa a clasei Enumerable

Operatori de agregare LINQ

```
static void AggregateOps()
{
    double[] winterTemps = { 2.0, -21.3, 8, -4, 0, 8.2 };
    // Exemple de agregare
    Console.WriteLine("Max temp: {0}", (from t in winterTemps select
    t).Max());
    Console.WriteLine("Min temp: {0}", (from t in winterTemps select
    t).Min());
    Console.WriteLine("Average temp: {0}", (from t in winterTemps
    select t).Average());
    Console.WriteLine("Sum of all temps: {0}", (from t in winterTemps
    select t).Sum());
}
```



Cu LINQ vs. Fara LINQ

```
static void
QueryOverStrings()
{
    string[] currentVideoGames =
    {"Morrowind", "Uncharted 2",
    "Fallout 3", "Daxter",
    "System Shock 2"};
    var subset = from g in
    currentVideoGames where
    g.Contains(" ") orderby g
    select g;
    foreach (string s in subset)
    Console.WriteLine("Item:
    {0}", s);
}
```

```
static void QueryOverStringsLongHand()
{
    string[] currentVideoGames = {"Morrowind",
    "Uncharted 2", "Fallout 3", "Daxter", "System
    Shock 2"};
    string[] gamesWithSpaces = new string[5];
    for (int i = 0; i < currentVideoGames.Length;
    i++)
    { if (currentVideoGames[i].Contains(" "))
    gamesWithSpaces[i] = currentVideoGames[i];
    }
    Array.Sort(gamesWithSpaces);
    foreach (string s in gamesWithSpaces)
    { if( s != null)
    Console.WriteLine("Item: {0}", s);
    }}
}
```


Relatii intre tipurile interogarilor LINQ

```
List<string> names =  
    new List<string>{"John", "Rick", "Maggie", "Mary"};  
  
IEnumerable<string> nameQuery = from name in names  
                                where name[0] == 'M'  
                                select name;  
  
foreach (string str in nameQuery)  
{  
    Console.WriteLine(str);  
}
```

The diagram illustrates the relationships between LINQ types in the provided code. It shows three numbered arrows: 1. From the `List<string>` type in the first line to the `name` variable in the `from` clause of the second line. 2. From the `name` variable in the `select` clause of the second line to the `string` type in the `foreach` loop of the third line. 3. From the `string` type in the `foreach` loop of the third line to the `IEnumerable<string>` type in the second line.

Interogari care nu transforma sursa de date

1. Argumentul tip al sursei de date determină tipul identificatorului.
2. Tipul obiectului care este selectat determină tipul variabilei de interogare. Aici `name` este un `string`. Prin urmare, variabila de interogare este un `IEnumerable<string>`.
3. Variabila de interogare este iterata în instrucțiunea `foreach`. Deoarece variabila de interogare este o secvență de string-uri, variabila de iterație este, de asemenea, un șir.

Relatii intre tipurile interogarilor LINQ

Interogari care transforma sursa de date: Interogarea preia o secvență de obiecte Customer ca intrare și selectează numai proprietatea Nume din rezultat

1. Argumentul tip al sursei de date determină tipul identicatorului.
2. Instrucțiunea select returnează proprietatea Name în loc de obiectul Customer complet - argumentul de tip al custNameQuery este string, nu Customer.
3. Deoarece custNameQuery este o secvență de string-uri, variabila de iterație a buclei foreach trebuie să fie, de asemenea, un string.

```
Table<Customer> Customers = db.GetTable<Customers>();  
  
IQueryable<string> custNameQuery =  
    from cust in Customers  
    where cust.City == "London"  
    select cust.Name;  
  
foreach (string str in custNameQuery)  
{  
    Console.WriteLine(str);  
}
```

Relatii intre tipurile interogarii LINQ

```
Table<Customer> Customers = db.GetTable<Customers>();
```

1

```
var namePhoneQuery =  
    from cust in Customers  
    where cust.City == "London"  
    select new { name = cust.Name,  
                phone = cust.Phone };
```

2

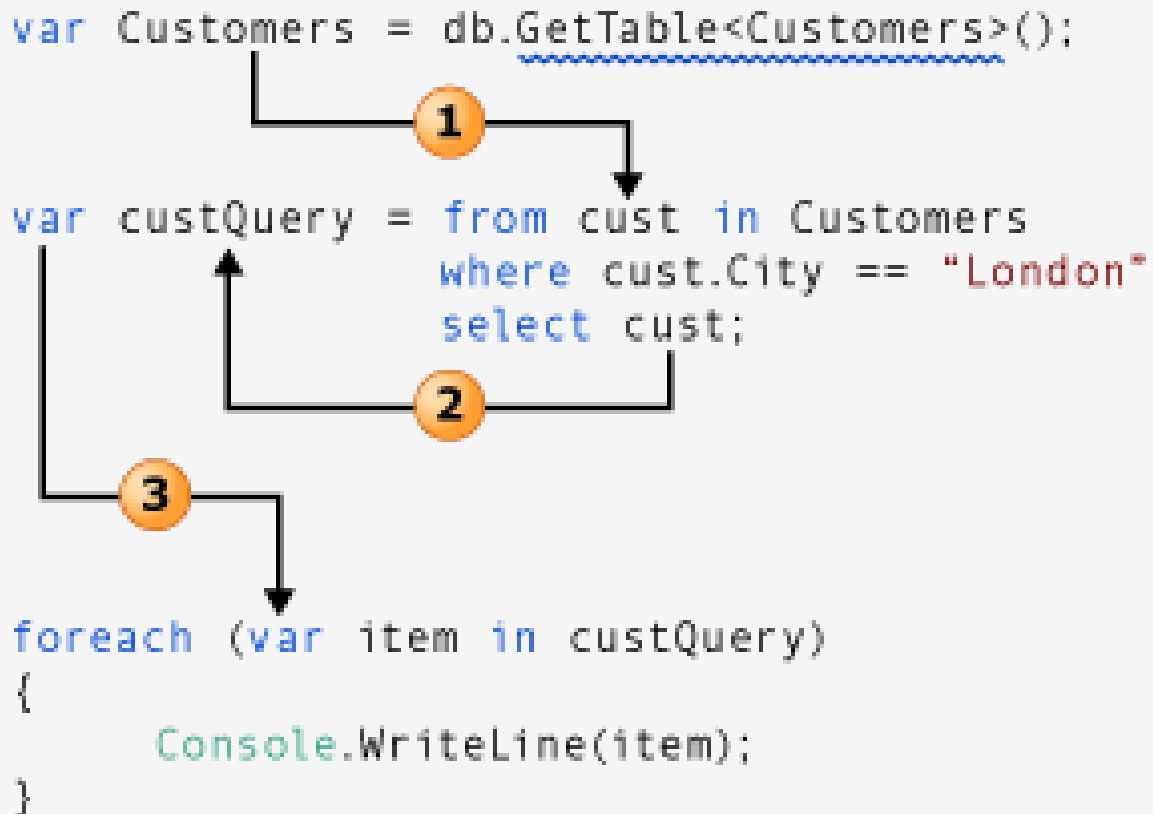
3

```
foreach (var item in namePhoneQuery)  
{  
    Console.WriteLine(item);  
}
```

1. Argumentul tip al sursei de date este întotdeauna tipul identificatorului.
2. Deoarece instrucțiunea select produce un tip anonim, variabila de interogare trebuie introdusă implicit folosind var.
3. Deoarece tipul variabilei de interogare este implicit, variabila de iterație din bucla foreach trebuie, de asemenea, să fie implicită.

Relatii intre tipurile interogarii LINQ

```
var Customers = db.GetTable<Customers>();  
var custQuery = from cust in Customers  
                where cust.City == "London"  
                select cust;  
foreach (var item in custQuery)  
{  
    Console.WriteLine(item);  
}
```



Similar cu exemplul 2, dar folosim variabile cu tip implicit - var

Executia amanata

- Interogările LINQ nu sunt evaluate decat in momentul in care se itereaza secventa
- Beneficiu – putem utiliza o interogare LINQ de mai multe ori pentru acelasi container si putem fi siguri ca vom obtine rezultate actualizate

```
static void QueryOverInts()
{
    int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };

    var subset = from i in numbers where i < 10 select i;
    // LINQ se evalueaza aici!
    foreach (var i in subset)
        Console.WriteLine("{0} < 10", i);
    Console.WriteLine();
    // Modificam datele
    numbers[0] = 4;
    // Se reevalueaza!
    foreach (var j in subset)
        Console.WriteLine("{0} < 10", j);
    Console.WriteLine();
}
```

Executia Imediata

- Cand e nevoie sa evaluam o expresie LINQ fara a itera colectia putem utiliza metode extinse ale clasei Enumerable `ToArray<T>()`, `ToDictionary<TSource,TKey>()`, and `ToList<T>()`.
- Aceste metode vor face ca interogarea LINQ sa se execute in momentul apelarii metodei pentru a obtine datele

```
static void ImmediateExecution()
```

```
{
```

```
int[] numbers = { 10, 20, 30, 40, 1, 2, 3, 8 };
```

```
// obtinem datele IMEDIAT ca int[].
```

```
int[] subsetAsIntArray = (from i in numbers where i < 10 select i).ToArray<int>();
```

```
// obtinem datele IMEDIAT ca List<int>.
```

```
List<int> subsetAsListOfInts = (from i in numbers where i < 10 select i).ToList<int>();
```

```
}
```

Interogari cu agregari

```
var subset = from i in numbers where i < 10  
select i;
```

```
int NumCount = subset.Count();
```

Interogari care realizeaza agregari se executa fara o instructiune foreach pentru ca interogarea in sine foloseste un foreach pentru a returna rezultatul

Query Syntax vs. Method Syntax

```
int[] numbers = { 5, 10, 8,  
3, 6, 12};
```

//Query syntax:

```
IEnumerable<int> numQuery1 = from  
num in numbers
```

```
where num % 2 == 0
```

```
orderby num select num;
```

//Method syntax:

```
IEnumerable<int> numQuery2 =  
numbers.Where(num => num % 2 ==  
0).OrderBy(n => n);
```

```
foreach (int i in numQuery1)  
{ Console.Write(i + " "); }
```

```
Console.WriteLine(System.Environment.NewLine);
```

```
foreach (int i in numQuery2)  
{ Console.Write(i + " "); }
```


Curs 5

Async si await

Tag Helpers

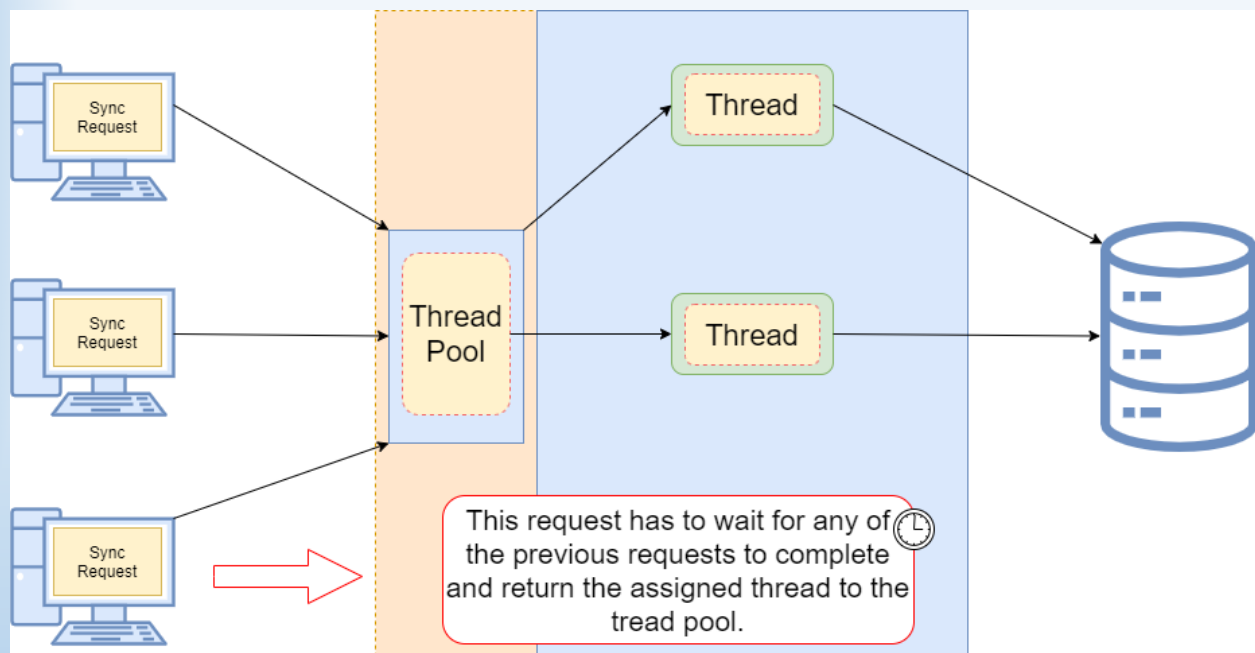
Programare asincrona in ASP.NET Core

- 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 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?

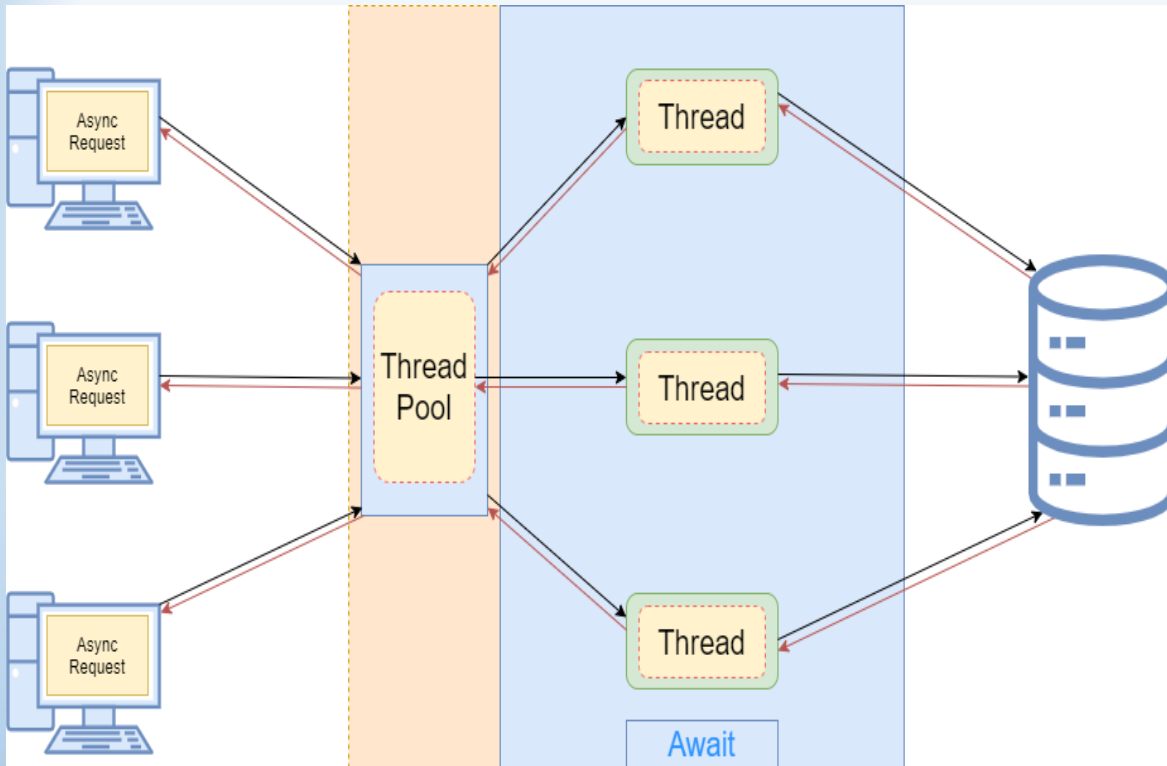
- Serverul poate gestiona un numar limitat de cereri
- Daca aplicatia primeste mai multe cereri decat poate gestiona serverul – performanta per ansamblu va avea de suferit (putem adauga un server – scalare pe orizontala)

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<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`).

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;}
}
```

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,
        "",
        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 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 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 ... >) 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 crea 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 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 = "ubbc1uj.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>
```

# Curs 6

Validarea datelor  
Autentificare si Autorizare

# DRY

- Un principiu cheie al dezvoltării software se numește DRY (Don't repeat yourself).
- Razor Pages încurajează dezvoltarea în care funcționalitatea este specificată o singură dată și se reflectă în întreaga aplicație.
- DRY poate ajuta:
  - Reduceți cantitatea de cod într-o aplicație.
  - Faceți codul mai puțin predispus la erori și mai ușor de testat și întreținut.
- Suportul de validare oferit de Razor Pages și Entity Framework este un bun exemplu al principiului DRY: Regulile de validare sunt specificate declarativ într-un singur loc, în clasa modelului. Regulile sunt aplicate peste tot în aplicație



# DataAnnotations

- Namespace-ul `System.ComponentModel.DataAnnotations` furnizează:
  - Un set de attribute de validare încorporate care sunt aplicate declarativ unei clase sau proprietăți.
  - Attribute de formatare precum `[DataType]` care ajută la formatare și nu oferă nicio validare.

# Atribute de validare

```
public class Book {
 public int Id { get; set; }
 [StringLength(60, MinimumLength = 3)]
 [Required]
 [RegularExpression(@"^[A-Z]+[a-zA-Z\s]*$")]
 public string Title { get; set; }
 [DataType(DataType.Date)]
 public DateTime PublishingDate { get; set; }
 [Range(1, 100)]
 [Column(TypeName = "decimal(6, 2)")]
 public decimal Price { get; set; }
}
```

- [Required] și [MinimumLength] indică faptul că o proprietate trebuie să aibă o valoare. Nimic nu împiedică un utilizator să introducă spațiu alb pentru a satisface această validare.
- [RegularExpression] este folosit pentru a limita caracterele care pot fi introduse. Title: Trebuie să folosească numai litere. Prima literă trebuie să fie majusculă. Spațiile albe sunt permise în timp ce numerele și caracterele speciale nu sunt permise.
- Atributul [Range] constrânge introducerea unei valori într-un interval specificat.
- Atributul [StringLength] poate seta o lungime maximă a unei proprietăți și opțional lungimea minimă a acesteia.
- Tipurile de valori, cum ar fi decimal, int, float, DateTime, sunt obligatorii în mod inerent și nu au nevoie de atributul [Required].

# [DataType]

- Oferă indicii pentru ca motorul de vizualizare să formateze datele.
- Utilizați atributul [RegularExpression] pentru a valida formatul datelor.
- Atributul [DataType] este utilizat pentru a specifica un tip de date care este mai specific decât tipul intrinsec al bazei de date.
- Atributele [DataType] nu sunt atribute de validare
- Poate furniza un selector de dată DataType.Date în browserele care acceptă HTML5.
- DataType.Date nu specifică formatul datei care este afișată. În mod implicit, câmpul de date este afișat conform formatelor implicite bazate pe CultureInfo a serverului.

# Mesaje de eroare

- Atributele de validare permit specificarea unui mesaj de eroare care va fi afisat pentru input invalid

```
[StringLength(8, ErrorMessage = "Name length can't be more than 8.")]
```

```
[StringLength(8, ErrorMessage = "{0} length must be between {2} and {1}.", MinimumLength = 6)
```

```
public string Name { get; set; }
```

```
"Name length must be between 6 and 8.".
```

# Validare Client - side

- Previne trimiterea formularului daca formularul nu este valid
- Previne round-trip care nu este necesar la server in cazul in care exista erori de input la un formular
- Scripturile specificate in *\_ValidationScriptsPartial.cshtml* asigura support pentru validarea de tip client-side

```
<script src="~/lib/jquery-validation/dist/jquery.validate.min.js"></script>
```

```
<script src="~/lib/jquery-validation-unobtrusive /jquery.validate.unobtrusive.min.js">
</script>
```

# jQuery Unobtrusive Validation

- Scriptul jQuery Unobtrusive Validation este o librărie de front-end custom Microsoft bazat pe pluginul jQuery Validation
- Fără această librărie ar trebui să realizăm logica de validare atât server-side cât și client-side
- TagHelpers utilizează atribute de validare ale proprietăților modelului și afișează atribute data- HTML5 pentru elementele din formular care necesită validare
- jQuery Unobtrusive Validation parsează atributele data- și le trimite la jQueryValidation



## CSHTML

```
<div class="form-group">
```

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

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

```

```

```
</div>
```

- HTML

```
<div class="form-group"> <label class="control-label" for="Book_Title">Book Title</label> <input
class="form-control" type="text" data-val="true" data-val-regex="The field Book Title must match
the regular expression '^[A-Z]+[a-zA-Z0-9\s.]*$'." data-val-regex-pattern="^[A-
Z]+[a-zA-Z0-9\s.]*$" data-val-required="The Book Title field is required." id="Book_Title"
name="Book.Title" value="" /> <span class="text-danger field-validation-valid" data-valmsg-
for="Book.Title" data-valmsg-replace="true"> </div>
```

# Attribute predefinite

- [CreditCard]: valideaza ca proprietatea respecta formatul unui card de credit.  
Necesita validari aditionale cu jQuery
- [EmailAddress]: valideaza ca proprietatea respecta formatul unei adrese de email.
- [Phone]: valideaza ca proprietatea respecta formatul unui numar de telefon
- [Url]: valideaza ca proprietatea respecta formatul unui URL



# Atribute pentru validare custom

```
public class Book {
 public int Id { get; set; }
 [StringLength(60, MinimumLength = 3)]
 [Required]
 [RegularExpression(@"^[A-Z]+[a-zA-Z\s]*$")]
 public string Title { get; set; }
 [ClassicBook(1960)]
 [DataType(DataType.Date)]
 public DateTime PublishingDate { get; set; }
 [Range(1, 100)]
 [Column(TypeName = "decimal(6, 2)")]
 public decimal Price { get; set; }
}
```

# Attribute custom

- Mostenim clasa ValidationAttribute
- Suprascriem metoda IsValid
- Argumente ale metodei IsValid:
  - Value – inputul care trebuie validat
  - ValidationContext – obiect care ofera informatii despre instanta modelului
- Ruleaza pe server

# Attribute custom - exemplu

```
public class ClassicBookAttribute : ValidationAttribute
{
 public ClassicBookAttribute(int year)
 { Year = year; }
 public int Year { get; }
 public string GetErrorMessage() => $"Classic books must have a publishing year no later than {Year}.";
 protected override ValidationResult IsValid(object value, ValidationContext validationContext)
 {
 var book = (Book)validationContext.ObjectInstance;
 var releaseYear = ((DateTime)value).Year;
 if (releaseYear > Year)
 { return new ValidationResult(GetErrorMessage()); }
 return ValidationResult.Success;
 }
}
```



# Validation TagHelpers

- Exista doi taghelpers pentru validare:
  - Validation message tag helper – afiseaza un mesaj de validare pentru o singura proprietate din model
  - Validation summary tag helper – specifica daca validarea se va face pe baza modelului (ModelOnly)/tipurile setate in baza de date (All)
- Input Tag Helper adauga client-side validation bazate pe adnotarile de la nivelul modelului
- Validarea se face la nivel de server daca se dezactiveaza JavaScript

# Tag Helper pentru validare

- Validation Message Tag Helper

- afiseaza un mesaj de validare pentru proprietatea aferenta din modelul nostru
- este utilizat cu atributul asp-validation-for a unui element HTML <span>
- in general il utilizam dupa un tag helper input pentru acceasi proprietate, pentru a afisa erorile de validare langa input-ul care a cauzat eroarea

```
<input asp-for="Book.Author" />
```

```

```

- Validation Summary Tag Helper afiseaza un rezumat al erorilor de validare

```
<div asp-validation-summary="ModelOnly"></div>
```

- Validarea se realizeaza pe baza constrangerilor si atributelor adnotarilor in clasele din model

# Autorizarea

- Procesul care determina ce actiuni ii sunt premise utilizatorului
- Independenta de autentificare dar necesita un mecanism de autentificare
- Tipuri de autorizare:
  - declararea rolului
  - pe baza unei politici

# Identity

- ASP.NET Core Identity:
  - API care ofera functionalitati de tip login
  - Gestioneaza utilizatori, parole, roluri, confirmare email etc.
  - Configurat sa utilizeze o baza de date SQL Server pentru a stoca useri, parole si date de profil
  - Sub forma unei librarii de clase Razor

# Areas

- Reprezinta un feature ASP.NET utilizat pentru organizarea unor functionalitati relationate intr-un singur grup ca o structura de foldere
- Oferă o modalitate de partitionare a unei aplicatii web in grupuri functionale mai mici, fiecare cu propriul set de pagini razor, controller, view-uri si modele
- Reprezinta efectiv o structura in interiorul unei aplicatii – pentru aplicatii complexe e avantajos sa partitionam aplicatia in functionalitati (arii) de nivel inalt
- Ex. Aplicatie de e-commerce – checkout, billing si search fiecare cu propria arie care contine clase si paginile aferente



# Configurarea Lockout

```
public async Task<IActionResult> OnPostAsync(string
returnUrl = null) {
 if (ModelState.IsValid) {
 var result = await
_signInManager.PasswordSignInAsync(Input.Email,
Input.Password, Input.RememberMe, lockoutOnFailure:
false);
...
}
```

# Optiuni Lockout

- Program.cs

```
builder.Services.Configure<IdentityOptions>(options => {
 options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);
 options.Lockout.MaxFailedAccessAttempts = 5;
});
```

- [DefaultLockoutTimeSpan](#) – perioada pentru care este blocat un utilizator cand apare lockout
- [MaxFailedAccessAttempts](#) – numarul de incercari permise pana la lockout

# Configurare Parola

- Implicit – Identity solicita ca parola sa contina un caracter litera mare, un caracter litera mica, un numar, si un caracter non-alfanumeic
- Lungimea trebuie sa fie minim sase caractere – atributul [StringLength] din InputModel (Areas/Identity/Pages/Account/Register.cshtml.cs si ResetPassword.cshtml.cs)
- [PasswordOptions](#) in Program.cs.

# Optiuni Parola

Program.cs

```
builder.Services.Configure<IdentityOptions>(options =>
{
 options.Password.RequireDigit = true;
 options.Password.RequireLowercase = true;
 options.Password.RequireNonAlphanumeric = true;
 options.Password.RequireUppercase = true;
 options.Password.RequiredLength = 6;});
```



# Configurare User

```
builder.Services.Configure <IdentityOptions>(options => {
 // Default User settings.
 options.User.AllowedUserNameCharacters =
 "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123
 456789-._@+";
 options.User.RequireUniqueEmail = false; });
```

# Autorizarea

- Procesul care determina ce actiuni ii sunt premise utilizatorului
- Independenta de autentificare dar necesita un mecanism de autentificare
- Tipuri de autorizare in Pagini Razor:
  - Conventie
  - Atributul [Authorize]

# Autorizarea prin conventie

- In fisierul Program.cs

```
builder.Services.AddRazorPages(options =>
{
 options.Conventions.AuthorizeFolder("/Books");
 options.Conventions.AllowAnonymousToPage("/Books/Index");
 options.Conventions.AllowAnonymousToPage("/Books/Details");
 options.Conventions.AuthorizeFolder("/Members", "AdminPolicy");
});
```

- Specificarea unei politici

```
builder.Services.AddAuthorization(options =>
{
 options.AddPolicy("AdminPolicy", policy =>
 policy.RequireRole("Admin"));
});
```

# Combinare autorizare cu acces anonim

- Valid

```
.AuthorizeFolder("/Books").AllowAnonymousToPage
("/Books/Index");
```

- Invalid

```
.AllowAnonymousToFolder("/Books").AuthorizePage
("/Books/Index");
```



# Autorizarea prin atribut

- Aplicarea atributului Authorize la pagina

```
[Authorize(Roles = "Admin")]
public class EditModel : PageModel
{...
 public async Task<IActionResult> OnPostAsync(){ }
 ...
}
```

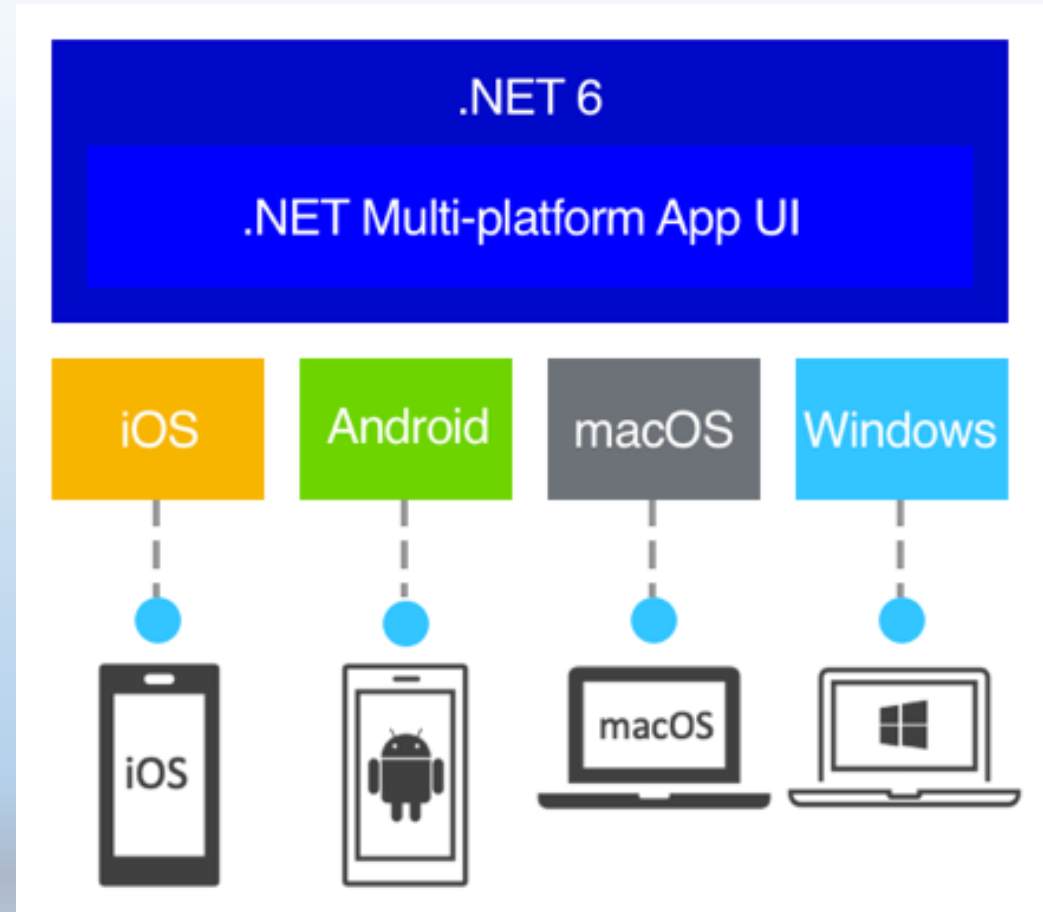
- Nu se poate aplica la nivel de metoda, doar la nivel de pagina

# Curs 7

Aplicatii multi-platforma cu .NET  
MAUI

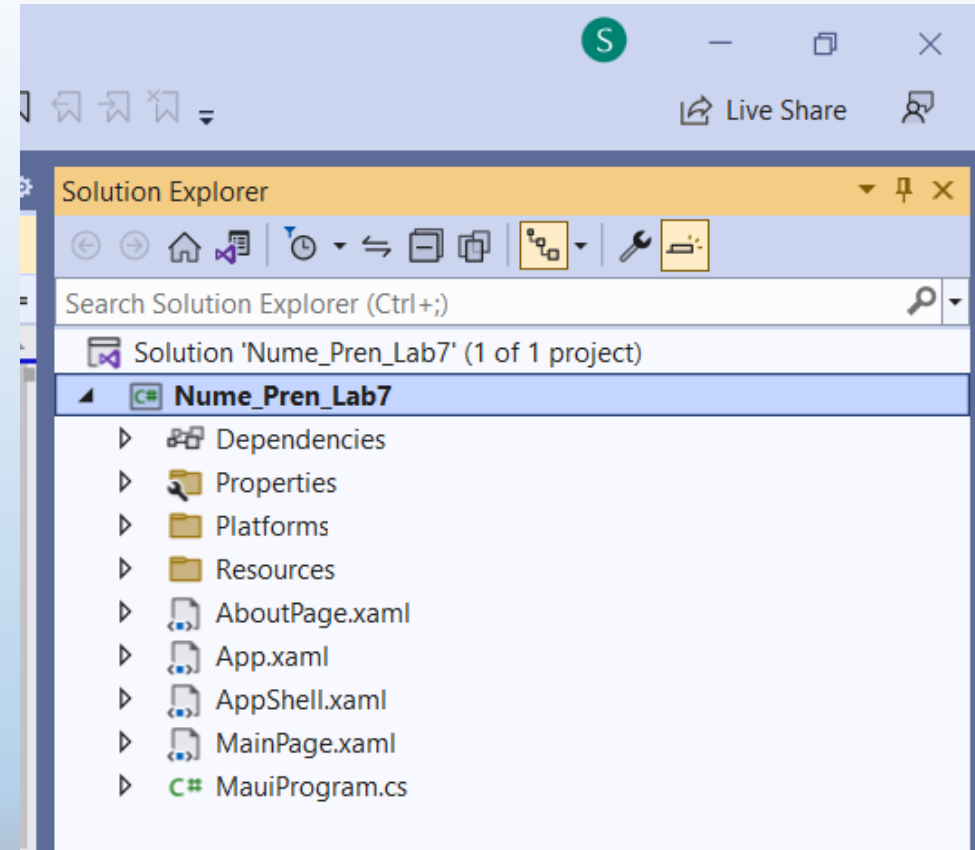
# Ce este .NET MAUI?

- Este un framework multi-platforma pentru crearea de aplicații mobile și desktop cu C# și XAML.
- Putem dezvolta aplicații care pot rula pe Android, iOS, macOS și Windows dintr-o singură bază de cod partajată.
- Reprezintă evoluția lui Xamarin.Forms (fiind extins de la aplicații mobile la scenarii de tip desktop)



# Structura unei aplicatii .NET MAUI

- *MauiProgram.cs*
  - fișierul care configureaza si pornește aplicația.
  - punct de intrare al aplicației
  - indica clasa App definită de fișierul App.xaml.



# Structura unei aplicatii .NET MAUI

- *App.xaml si App.xaml.cs*
  - Fișierul .xaml conține marcaj XAML, iar fișierul de cod conține cod pentru a interacționa cu marcajul XAML.
  - Fișierul App.xaml conține resurse XAML la nivel de aplicație, cum ar fi culori, stiluri sau șabloane.
  - Fișierul App.xaml.cs conține în cod care instanțiază aplicația Shell
- *AppShell.xaml și AppShell.xaml.cs*
  - Acest fișiere definesc ierarhia vizuală a aplicației.
- *MainPage.xaml și MainPage.xaml.cs*
  - Aceasta este pagina de pornire afișată de aplicație. Fișierul MainPage.xaml definește UI (interfața de utilizator) a paginii. MainPage.xaml.cs conține codul din spatele XAML

# XAML

- XAML (eXtensible Application Markup Language) este un limbaj bazat pe XML permite organizarea obiectelor în ierarhii părinte-copil.
- XAML permite dezvoltatorilor să definească UI în aplicațiile .NET MAUI utilizând markup în loc de cod.
- Avantaje:
  - XAML este adesea succint
  - Ierarhia părinte-copil inherentă în XML permite XAML să imite cu o mai mare claritate vizuală ierarhia părinte-copil a obiectelor interfeței cu utilizatorul.
- Dezavantaje:
  - Gestiunea evenimentelor trebuie să fie realizată într-un fișier separat cu cod sursă
  - Nu există designer vizual pentru elementele XAML. Toate elementele de interfață trebuie editate manual (putem folosi însă hot reload pentru a vizualiza UI pe măsura ce îl edităm)

# Anatomia unui fisier XAML

- <ContentPage> este obiectul rădăcină pentru clasa AboutPage. ContentPage poate avea un singur obiect copil.
- <VerticalStackLayout> este singurul obiect copil al ContentPage. Tipul VerticalStackLayout poate avea mai mulți copii. Acest control aranjează copiii pe verticală, unul după altul.
- <Image> afișează o imagine, în acest caz folosește imaginea dotnet\_bot.png care vine cu fiecare proiect .NET MAUI.
- <Label> afișează text
- <Button> declasează evenimentul Clicked. Putem rula cod ca răspuns la evenimentul Clicked Clicked="OnCounterClicked"
- Evenimentul Clicked al butonului este atribuit handler-ului de evenimente OnCounterClicked, definit în fișierul code-behind.

# Sintaxa XAML

- XAML este conceput în principal pentru instanțierea și inițializarea obiectelor
- Label este un *object element* obiect .NET MAUI reprezentat ca un element XML.
- Text, VerticalOptions, FontAttributes and FontSize sunt *property attributes* – proprietati reprezentate ca attribute XML
- In cel de al doilea exemplu TextColor a devenit *property element*

```
<Label Text="Hello, XAML!"
 VerticalOptions="Center"
 FontAttributes="Bold"
 FontSize="18"
 TextColor="Aqua" />
```

```

<Label Text="Hello, XAML!"
VerticalOptions="Center"
FontAttributes="Bold" FontSize="18">
 <Label.TextColor>
Aqua
 </Label.TextColor>
</Label>
```



# Proprietati semantice

- Folosite pentru a defini informații despre controale
- Clasa `SemanticProperties` definește următoarele proprietăți atașate:
  - `Description` (tip `string`)- reprezintă o descriere
  - `Hint` (tip `String`) care este similar cu `Description`, dar oferă context suplimentar, cum ar fi scopul unui control
  - `HeadingLevel` care permite ca un element să fie marcat ca titlu pentru a organiza interfața de utilizare și a facilita navigarea

# Attached Properties

```
<Grid>
<Grid.RowDefinitions>
<RowDefinition Height="Auto" />
<RowDefinition Height="100" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition Width="100" />
</Grid.ColumnDefinitions>
<Label Text="My Text"
Grid.Row="1" Grid.Column="1"
TextColor="Purple">
</Grid>
```

- Sunt definite de clasa Grid, dar sunt setate la nivelul copiilor Gridului

# Content properties

<ContentPage

```
xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlSamples.XamlPlusCodePage"
Title="XAML + Code Page">
```

<ContentPage.Content>

<Grid> ... </Grid>

</ContentPage.Content>

</ContentPage>

-nu sunt obligatorii pentru ca toate elementele .NET MAUI pot o avea un singur element Content

# XAML Markup extensions

- permit setarea proprietăților la obiecte sau valori care sunt referite indirect din alte surse (ex. de obicei, utilizăm XML pentru a seta proprietățile unui obiect la valori explicite, cum ar fi un string, un număr, etc.)
- sunt deosebit de importante pentru partajarea obiectelor și constantele de referință utilizate în cadrul unei aplicații
- markup extensions XML sunt denumite astfel deoarece sunt susținute de cod în clasele care implementează `IMarkupExtension`. Putem să definim markup extensions custom

# Resurse comune

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlSamples.SharedResourcesPage" Title="Shared Resources Page">
 <StackLayout>
 <Button Text="Do this!" HorizontalOptions="Center" VerticalOptions="Center"
 BorderWidth="3" Rotation="-15" TextColor="Red" FontSize="24" />
 <Button Text="Do that!" HorizontalOptions="Center" VerticalOptions="Center"
 BorderWidth="3" Rotation="-15" TextColor="Red" FontSize="24" />
 </StackLayout>
</ContentPage>
```

Dacă una dintre aceste proprietăți trebuie schimbată, preferam să facem modificarea o singură dată.



# Dictionar de resurse

- un dicționar cu perechi de tip chei string și valori asociate

```
<ContentPage
xmlns="http://schemas.microsoft.com/dotnet/2021/
maui"
xmlns:x="http://schemas.microsoft.com/winfx/2009/
xaml" x:Class="XamlSamples.SharedResourcesPage"
Title="Shared Resources Page">
```

```
<ContentPage.Resources>
```

```
<LayoutOptions x:Key="horzOptions"
Alignment="Center" />
```

```
<LayoutOptions x:Key="vertOptions"
Alignment="Center" /> </ContentPage.Resources>
```

```
...
```

```
</ContentPage>
```

```
<Button Text="Do this!"
HorizontalOptions="{StaticResource horzOptions}"
VerticalOptions="{StaticResource vertOptions}"
BorderWidth="3"
Rotation="-15"
TextColor="Red"
FontSize="24" />
```

StaticResource este întotdeauna delimitată cu {} și include cheia din dicționarul de resurse

DynamicResource - pentru cheile de dicționar asociate cu valori care s-ar putea modifica în timpul execuției, în timp ce StaticResource accesează elementele din dicționar o singură dată când elementele de pe pagină sunt construite

# Pagini Shell in .NET MAUI

- Un obiect ShellContent reprezintă obiectul ContentPage pentru fiecare FlyoutItem sau Tab
- Paginile sunt create de obicei la cerere, ca răspuns la navigare
- Acest lucru se realizează prin utilizarea extensiei de markup *DataTemplate* pentru a seta proprietatea *ContentTemplate* a fiecărui obiect ShellContent la un obiect ContentPage

```
<Shell ...>

 <TabBar>
 <ShellContent
 Title="Welcome"
 ContentTemplate="{DataTemplate
local:MainPage}"
 />

 <ShellContent
 Title="About"
 ContentTemplate="{DataTemplate
local:AboutPage}"
 />

 <ShellContent
 Title="My Shopping Lists"
 ContentTemplate="{DataTemplate
local:ListEntryPage}"
 />
 </TabBar>
</Shell>
```

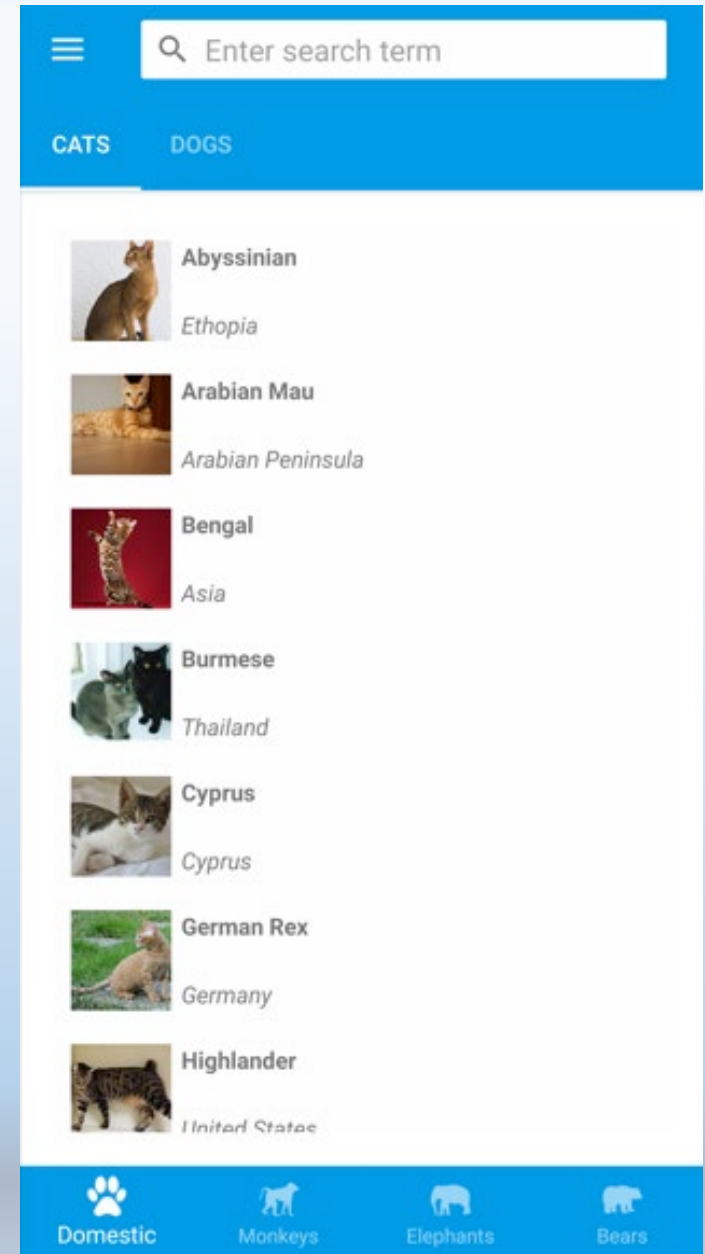
# Descrierea ierarhiei vizuale a unei aplicatii

- Ierarhia vizuală a unei aplicații .NET MAUI Shell este descrisă în clasa Shell, pe care șablonul de proiect o numește AppShell.
- O clasă Shell subclasată poate conține din trei obiecte ierarhice principale:
  - FlyoutItem sau TabBar. Un FlyoutItem reprezintă unul sau mai multe elemente de tip flyout. Un TabBar reprezintă bara de file și ar trebui utilizată atunci când modelul de navigare pentru aplicație începe cu file și nu necesită un instrument derulant. Fiecare obiect FlyoutItem sau obiect TabBar este un copil al obiectului Shell.
  - Tab, care reprezintă conținut grupat, navigabil. Fiecare obiect Tab este un copil al unui obiect FlyoutItem.
  - ShellContent, care reprezintă obiectele ContentPage pentru fiecare filă. Fiecare obiect ShellContent este un copil al unui obiect Tab. Atunci când într-o filă sunt prezente mai multe obiecte ShellContent, obiectele vor fi navigabile prin filele de sus.



# Descrierea ierarhiei - exemplu

- `<Shell xmlns="http://schemas.microsoft.com/dotnet/2021/maui" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:views="clr-namespace:Xaminals.Views" x:Class="Xaminals.AppShell"> ...`
- `<FlyoutItem FlyoutDisplayOptions="AsMultipleItems">`
  - `<Tab Title="Domestic" Icon="paw.png">`
    - `<ShellContent Title="Cats" Icon="cat.png" ContentTemplate="{DataTemplate views:CatsPage}" /> <ShellContent Title="Dogs" Icon="dog.png" ContentTemplate="{DataTemplate views:DogsPage}" />`
  - `</Tab>`
  - `<ShellContent Title="Monkeys" Icon="monkey.png" ContentTemplate="{DataTemplate views:MonkeysPage}" /> <ShellContent Title="Elephants" Icon="elephant.png" ContentTemplate="{DataTemplate views:ElephantsPage}" /> <ShellContent Title="Bears" Icon="bear.png" ContentTemplate="{DataTemplate views:BearsPage}" />`
- `</FlyoutItem>`
- `... </Shell>`
- Aceste obiecte nu reprezintă nicio interfață cu utilizatorul, ci mai degrabă organizarea ierarhiei vizuale a aplicației. Shell va prelua aceste obiecte și va produce interfața de navigare cu utilizatorul pentru conținut.



# Navigarea in .NET MAUI

- navigare bazată pe URI, care utilizează rute pentru a naviga la orice pagină din aplicație
- navigare bazata pe ierarhie

# Navigarea bazata pe URI

- Navigarea se realizează prin invocarea metodei `GoToAsync`, din clasa `Shell`
- Când navigarea este pe cale să fie efectuată, evenimentul *Navigating* este declanșat și evenimentul *Navigated* este declanșat când navigarea se termină.
- Navigarea cu ruta absoluta
  - `await Shell.Current.GoToAsync("//animals/monkeys");`
- Navigarea cu ruta relativa
  - `await Shell.Current.GoToAsync("monkeydetails");`
  - animals
    - domestic
      - cats
      - dogs
    - monkeys
    - elephants
    - bears
  - about

# Rute în .NET MAUI

- URI-urile de navigare pot avea trei componente:
  - O rută, care definește calea către conținut care există ca parte a ierarhiei vizuale Shell.
  - O pagina. Paginile care nu există în ierarhia vizuală Shell pot fi introduse în stiva de navigare de oriunde dintr-o aplicație Shell. De exemplu, o pagină de detalii nu va fi definită în ierarhia vizuală Shell, dar poate fi introdusă în stiva de navigare după cum este necesar.
  - Unul sau mai mulți parametri de interogare. Parametrii de interogare sunt parametri care pot fi transferați către pagina de destinație în timpul navigării.
- Când un URI de navigare include toate cele trei componente, structura este: `//route/page?queryParameters`

# Inregistrarea rutelor

- `<Shell ...>`
  - `<FlyoutItem ... Route="animals">`
    - `<Tab ... Route="domestic">`
      - `<ShellContent ... Route="cats" />`
      - `<ShellContent ... Route="dogs" />`
    - `</Tab>`
    - `<ShellContent ... Route="monkeys" />`
    - `<ShellContent ... Route="elephants" />`
    - `<ShellContent ... Route="bears" />`
  - `</FlyoutItem>`
  - `<ShellContent ... Route="about" />`
  - `</Shell>`

- Rutele pot fi definite pentru obiectele FlyoutItem, TabBar, Tab și ShellContent
- Se seteaza prin proprietatea Route
  - animals
    - domestic
      - cats
      - dogs
    - monkeys
    - elephants
    - bears
  - about
- Ruta absoluta pentru dogs:  
//animals/domestic/dogs
- ArgumentException va fi aruncata la pornirea aplicației dacă este detectată o rută duplicată

# Inregistrarea rutelor paginilor detaliu

- Paginile de tip detaliu nu se afla in ierarhia vizuala a clasei Shell
- În constructorul clasei Shell, rute suplimentare pot fi înregistrate în mod explicit pentru orice pagini de detalii care nu sunt reprezentate în ierarhia vizuală Shell.
- Acest lucru se realizează cu metoda `Routing.RegisterRoute`:

```
public AppShell()
{
 InitializeComponent();
 Routing.RegisterRoute("monkeydetails", typeof(MonkeyDetailPage));
 Routing.RegisterRoute("beardetails", typeof(BearDetailPage));
 Routing.RegisterRoute("catdetails", typeof(CatDetailPage));
 Routing.RegisterRoute("dogdetails", typeof(DogDetailPage));
 Routing.RegisterRoute("elephantdetails", typeof(ElephantDetailPage));
}
```

- Aceste pagini de detalii pot fi apoi navigate folosind navigarea bazată pe URI, de oriunde în cadrul aplicației
- Rutele pentru astfel de pagini sunt cunoscute ca rute globale

# Rute Relative

- Navigarea poate fi efectuată și prin specificarea unui URI relativ, valid ca argument pentru metoda GoToAsync.
- Sistemul de rutare va încerca să potrivească URI-ul cu un obiect ShellContent.
- Prin urmare, dacă toate rutele dintr-o aplicație sunt unice, navigarea poate fi efectuată prin specificarea numelui unic al rutei ca URI relativ. Sunt acceptate următoarele formate de rută relative:
  - route Ierarhia rutei va fi căutată pentru ruta specificată, în sus de la poziția curentă. Pagina potrivită va fi adăugată în stiva de navigare.
  - /route Ierarhia rutei va fi căutată de pe ruta specificată, în jos de la poziția curentă. Pagina potrivită va fi adăugată în stiva de navigare.
  - //route Ierarhia rutei va fi căutată pentru ruta specificată, în sus de la poziția curentă. Pagina potrivită va înlocui stiva de navigare.
  - ///route Ierarhia rutei va fi căutată pentru ruta specificată, în jos de la poziția curentă. Pagina potrivită va înlocui stiva de navigare.

# Navigarea ierharhica

- Pagini
  - Modale
  - NeModale
- In designul UI “modal” se refera necesitatea ca utilizatorul sa interactioneze cu aplicatia inainte ca aplicatia sa continue
- Cand o fereastră modala este afisata utilizatorului, utilizatorul nu poate sa se intoarca la fereastră principala, ci trebuie sa intractioneze cu fereastră modala mai intai
- In general utilizam ferestre modale cand aplicatia are nevoie de informatii de la utilizator si nu dorim ca utilizatorul sa se intoarca la pagina anterioara decat dupa ce furnizeaza informatiile
- O fereastră care nu este modala se numeste fereastră nemodala



# Navigarea ierarhica

- Mecanismul de navigare intre pagini implementeaza pagini modale si nemodale prin definirea a doua metode care pot fi apelate de o pagina pentru a naviga catre alta pagina

Task PushAsync(Page page) – navigheaza catre o pagina nemodala

Task PushModalAsync(Page page) - navigheaza catre o pagina modala

- Pentru a naviga catre pagina anterioara sunt definite metodele:

Task<Page> PopAsync()

Task<Page> PopModalAsync()

# Navigarea catre o alta pagina

- Utilizam proprietatea Navigation a obiectului pagina, codul pentru a naviga catre alta pagina va arata asa:

```
await Navigation.PushAsync(new MyNewPage());
await Navigation.PushModalAsync(new MyNewModalPage());
```

# Curs 8

Accesul la date in aplicatiile  
.NET MAUI

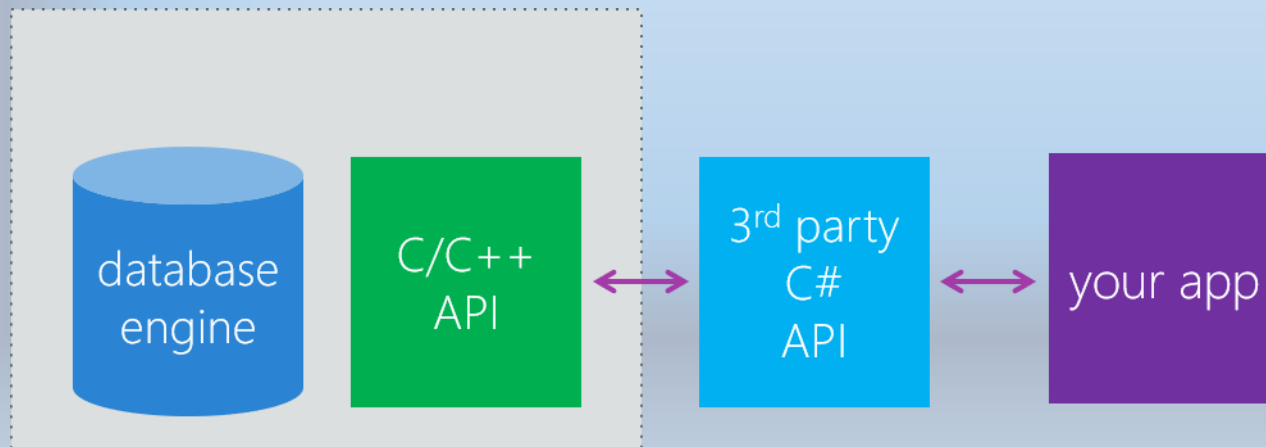
# Accesul la date

Scenarii de utilizare:

- .NET MAUI - baza de date locala – SQLite
- .NET MAUI – baza de date centralizata – consumarea unui serviciu REST intr-o aplicatie mobila

# SQLite

- Motor de baze de date SQL
- SQLite nu necesită un server
- Baza de date este stocată într-un singur fișier disc pe sistemul de fișiere al dispozitivului pe care ruleaza
- Multi-platforma -poate rula pe diverse SO



# Integrarea SQLite in aplicatiile .NET MAUI

Etape:

1. Install the NuGet package: sqlite-net-pcl
2. Configurarea: nume baza de date, calea spre baza de date

```
if (database == null)
{
 database = new
 ShoppingListDatabase(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.
 LocalApplicationData), "ShoppingList.db3"));
}
```

Expunem baza de date ca un singleton – o singura conexiune la baza de date este creata si este mentinuta deschisa cat timp aplicatia ruleaza, evitand operatia costisitoare de deschidere/inchidere db la fiecare operatie pe bd

3. Crearea unei clase pentru acces la date: centralizează logica interogărilor și simplifică gestionarea inițializării bazei de date, facilitând extinderea operațiilor de date pe măsură ce aplicația crește (ex. clasa `ShoppingListDatabase`)



# SQLite-net

- Este un object-relational mapper (ORM)
- Ajută la simplificarea procesului de definire a schemei bazei de date
- Permite să utilizăm modelele care sunt definite în proiect pentru a servi drept schema a bd
- SQLite-net este un pachet NuGet – pentru a-l utiliza instalăm pachetul **sqlite-net-pcl**

```
public class ShopList
{
 [PrimaryKey, AutoIncrement]
 public int ID { get; set; }
 [MaxLength(250), Unique]
 public string Description { get; set; }
 public DateTime Date { get; set; }
}
```

Clasa ShopList=> Tabel ShopList in BD

# Creare Tabel

- Constructorul primește ca și parametru calea spre fișierul de tip bază de date

```
public ShoppingListDatabase(string dbPath)
{
 database = new SQLiteAsyncConnection(dbPath);
 database.CreateTableAsync<ShopList>().Wait();
}
```



# Attribute SQLite-net

- |                              |   |    |                                                                                                                                                                  |
|------------------------------|---|----|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. <b>[Table(name)]</b>      | → | A. | Specifica faptul ca acea coloana este cheie primara (nu accepta valori NULL) (o singura coloana poate fi marcata cheie primara; nu suporta chei primare compuse) |
| 2. <b>[Column(name)]</b>     | → | B. | Specifica ca valorile stocate in acea coloana trebuie sa aiba valori unice (pot exista mai multe coloane setate unique)                                          |
| 3. <b>[PrimaryKey]</b>       | → | C. | Specifica ca valoarea acelei coloane se va autoincrementa la inserarea unui nou rand in tabel.                                                                   |
| 4. <b>[AutoIncrement]</b>    | → | D. | Specificam numele tabelului care va fi creat (daca dorim sa fie diferit de numele clasei)                                                                        |
| 5. <b>[MaxLength(value)]</b> | → | E. | Specifica numarul maxim de caractere care pot fi salvate in coloana respectiva.                                                                                  |
| 6. <b>[Ignore]</b>           | → | F. | Proprietatea nu va fi mapata cu o coloana din table                                                                                                              |
| 7. <b>[Unique]</b>           | → | G. | Specificam un nume diferit pentru coloana (daca dorim sa fie diferit de numele proprietatii).                                                                    |

# Conectarea la o BD SQLite

- Pentru a crea o conexiune la o bază de date SQLite utilizăm un obiect `SQLiteAsyncConnection`
- Această clasă este definită în namespace-ul `SQLite`
- Când instanțiam un obiect `SQLiteConnection`, specificăm numele fișierului pentru fișierul bazei de date. Constructorul va deschide apoi fișierul dacă acesta există sau îl va crea dacă nu este prezent.

```
SQLiteAsyncConnection _database = new SQLiteAsyncConnection(dbPath);
```

# Operatii CRUD cu SQLite

```
public Task<int> SaveProductAsync(Product product)
{
 if (product.ID != 0)
 {
 return _database.UpdateAsync(product);
 }
 else
 {
 return _database.InsertAsync(product);
 }
}

public Task<int> DeleteProductAsync(Product product)
{
 return _database.DeleteAsync(product);
}

public Task<List<Product>> GetProductsAsync()
{
 return _database.Table<Product>().ToListAsync();
}
```

Metoda Table returneaza toate randurile dintr-un tabel

# SQLite cu LINQ

- SQLite suporta :

- Where
- Take
- Skip
- OrderBy
- OrderByDescending
- ThenBy
- ElementAt
- First
- FirstOrDefault
- ThenByDescending
- Count

```
public Task<ShopList> GetShopListAsync(int id)
{
 return _database.Table<ShopList>()
 .Where(i => i.ID == id)
 .FirstOrDefaultAsync();
}
```

```
public Task<ShopList> GetShopListAsync(int id)
{
 var user = from u in
 _database.Table<ShopList>() where u.ID == id
 select u;
 return user.FirstOrDefault();
}
```

# SQLite asincron

- Dacă executăm interogări într-o bază de date într-un mod sincron=>probleme de performanță și aplicații care nu răspund.
- În mod sincron operațiile se execută pe Thread-ul UI – UI freeze sau operațiile durează mai mult pentru a fi executate

# Task-uri

- Clasa Task din namespace-ul System.Threading.Tasks
- Sunt gestionate de un task scheduler pentru a fi rulate pe firul de executie din background sau firul de executie al UI
- Task scheduler-ul default detine un set de fire de executie de background si utilizeaza urmatorul fir de executie disponibil pentru a executa urmatorul task
- Similare cu o comanda care contine o actiune care poate fi executata
- Spre deosebire de o comanda, nu sunt declansate de actiunea unui utilizator ci de un task scheduler care declanseaza actiunea pe firul de executie corespunzator

# Async si Await (I)

- Cuvintele cheie *async* si *await* – introduse in C# 5.0
- Acestea nu fac nimic/nu executa o actiune in termeni de cod
- Sunt folosite de compilator, codul marcat de aceste cuvinte fiind gestionat diferit
- *Async* – prescurtarea de la *asynchronous*—se pot executa mai multe actiuni simultan (cod multithreaded – cod *asynchron*)

```
async void OnShopListAddedClicked(object sender, EventArgs e)
{
 await Navigation.PushAsync(new ListPage
 {
 BindingContext = new ShopList()
 });
}
```

# Async si Await (II)

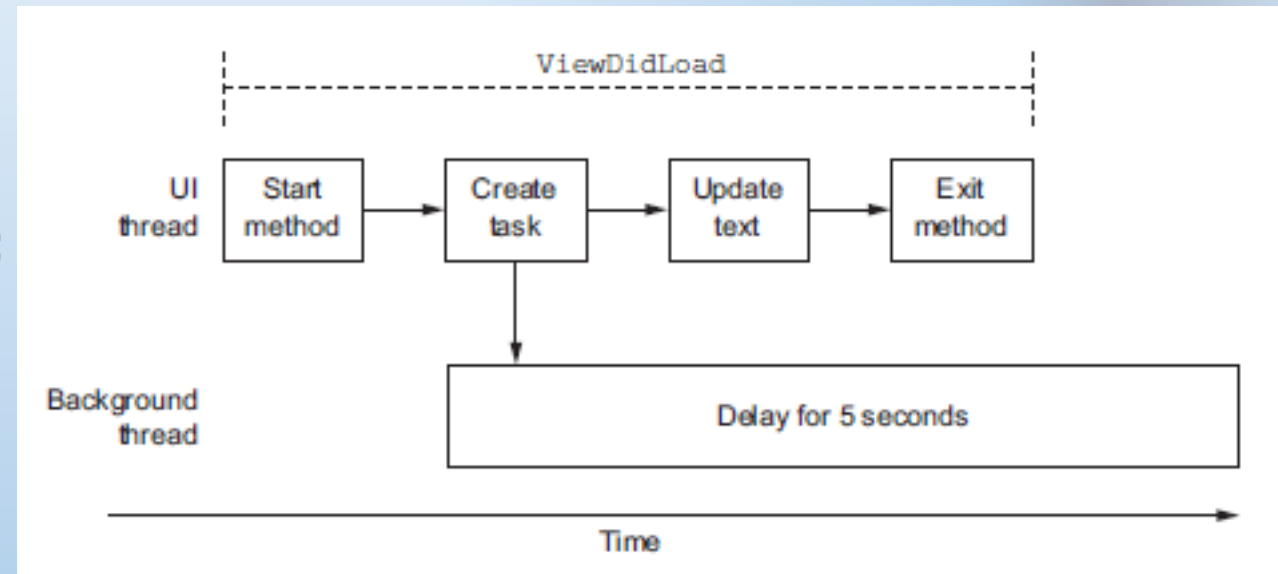
- Permit modelarea urmatorului pattern “ruleaza codul x in background apoi ruleaza codul y in contextul curent sincronizat”
- Marcam o metoda *async* si cand dorim sa apelam alte metode async din aceasta metoda marcam aceste apeluri cu *await*
- *Await* specifica compilatorului ca in metoda pe care o apelam, o parte din cod se va executa pe un fir de executie din background utilizand un task si sa astepte rularea codului din metoda pana cand metoda marcata await termina ce are de facut
- Intre timp, thread-ul curent poate procesa cealalta parte a codului si dupa finalizarea executiei metodei marcate cu await se executa si cealalta parte



# Task in background

```
public override void ViewDidLoad()
{
 ...
 Task.Delay(TimeSpan.FromSeconds(5));
 TextField.Text = "Foo";
}
```

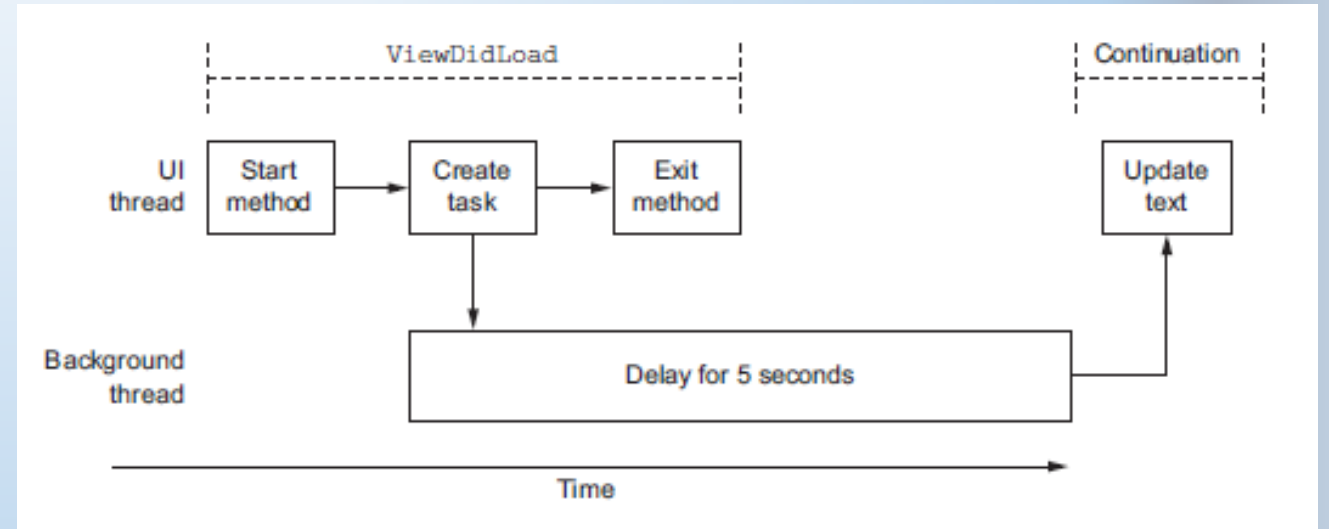
Task-ul Delay task este creat ,  
este pornit intr-un thread de background, UI este actualizat



# Adaugam `async` si `await`

```
public override async void
ViewDidLoad()
{
 ...
 await
 Task.Delay(TimeSpan.FromSeconds(5));
 TextField.Text = "Foo";
}
```

- Se iese din metoda `ViewDidLoad` de indata ce metoda `await` este apelata, iar restul codului se executa in firul de executie original



- Contextul de sincronizare UI are un singur thread, deci daca marcam `await` o metoda din thread-ul UI, codul care urmeaza dupa metoda va rula pe thread-ul UI
- Daca marcam `await` o metoda dintr-un thread de background, codul de dupa `await` s-ar putea sa ruleze pe un thread diferit in background

# Data Binding - context

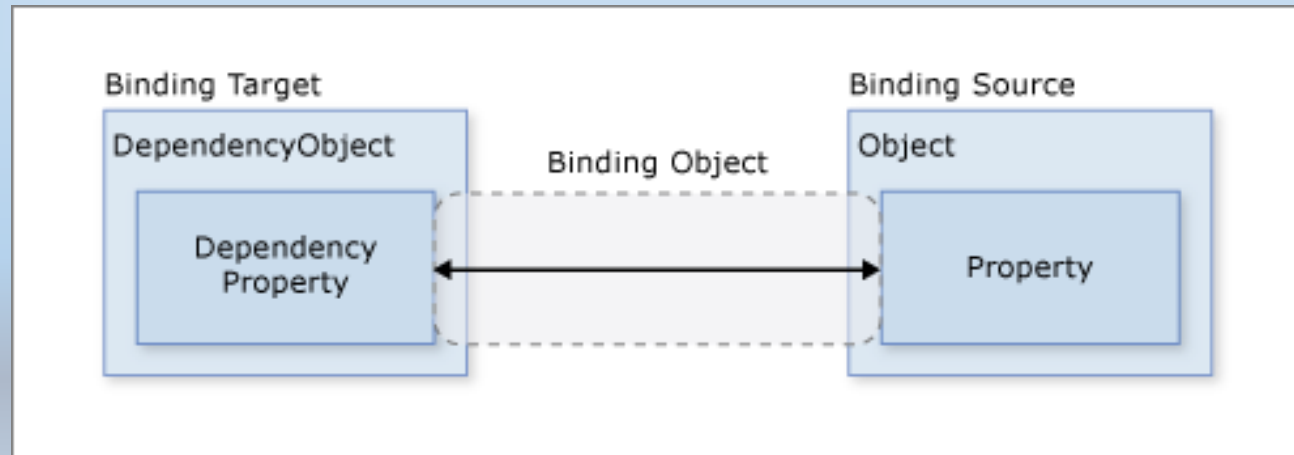
- O aplicație .NET MAUI constă dintr-una sau mai multe pagini, fiecare dintre acestea conține de obicei mai multe obiecte de UI
- Una dintre sarcinile principale ale aplicației este să mențină aceste obiecte UI sincronizate cu sursa de date și să țină evidența diferitelor valori sau selecții pe care le reprezintă
- Pentru a gestiona acest lucru cu succes, aplicația trebuie să fie notificată cu privire la modificările datelor sursa. Posibile soluții:
  - definirea evenimentelor care semnalează când are loc o schimbare-> un handler de evenimente care efectuează transferul de date de la un obiect la altul.
  - Utilizăm databinding - automatizează această sarcină și face ca handlerii de evenimente să nu fie necesari. DataBinding poate fi implementată fie în XAML, fie în cod

# Data Binding - scop

- Oferă aplicațiilor o modalitate consistentă de a afișa date și de a interacționa cu date diverse
- DataBinding- procesul care stabilește conexiunea dintre UI și datele pe care le afișează aceasta
  - când datele din sursa de date își modifică valoarea elementele care sunt legate la acele date se modifică automat
  - când se modifică datele în elementele de interfață, sursa de date se poate modifica automat pentru a reflecta modificările

# Tinta si sursa

- Data binding leagă o pereche de proprietăți între două obiecte, dintre care cel puțin unul este de obicei un obiect de interfață cu utilizatorul.
- Aceste două obiecte sunt numite țintă și sursă:
  - Ținta este obiectul (și proprietatea) asupra careia este setat DataBind-ul.
  - Sursa este obiectul (și proprietatea) referențiat de databinding.



# Operatii data-binding

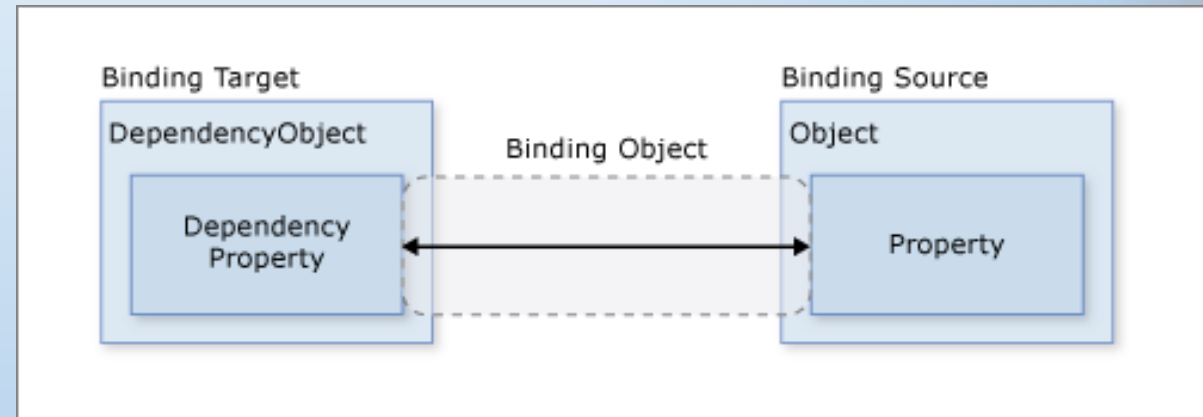
```
<Editor Placeholder="Enter shop name" Margin="20"
Text="{Binding ShopName}"
HeightRequest="50" />
```

Target – Editor

Target Property- Text

Source – Obiectul shop

Source object value path – shop.ShopName



În cel mai simplu caz, datele circulă de la sursă la țintă, ceea ce înseamnă că valoarea proprietății țintă este setată din valoarea proprietății sursă. Cu toate acestea, în unele cazuri, datele pot curge alternativ de la țintă la sursă sau în ambele direcții.

# Tipuri de data-binding

- Setam **Binding.Mode:**

1. One-time

2. One-way

3. Two-way

4. One-way-to-source

A. Utilizatorul poate modifica datele prin interfata . Bindingul se realizeaza in ambele directii. Utile in scenarii cu formulare editabile

B. Bindingul se face de la sursa la destinatie si se efectueaza o singura data cand se porneste aplicatia

C. Bindingul se face de la sursa la destinatie. Este util pentru datele read-only deoarece nu se pot modifica datele din controale UI

D. daca se modifica destinatia, sursa se modifica

# Binding la colectii

- Un obiect binding source poate fi tratat
  - colectie de date grupate impreuna (ex. rezultatul unei interogari asupra bd)
  - `<Picker Title="Select a Shop"  
ItemsSource="{Binding Shop}"  
ItemDisplayBinding="{Binding ShopName}" />`
- Setam Proprietatea ItemsSource si ItemDisplayBinding
- Pentru proprietatea ItemsSource bindingul implicit e OneWay



# Data Template

Ceea ce specificam in DataTemplate devine structura vizuala a unui obiect

```
<ListView x:Name="listView"
 Margin="20"
 ItemSelected="OnListViewItemSelected">
 <ListView.ItemTemplate>
 <DataTemplate>
 <TextCell Text="{Binding
ShopName}"
 Detail="{Binding
Adress}" />
 </DataTemplate>
 </ListView.ItemTemplate>
</ListView>
```

```
<ListView.ItemTemplate>
 <DataTemplate>
 <Grid> ...
 <Label Text="{Binding ShopName}"
FontAttributes="Bold" />
 <Label Grid.Column="1" Text="{Binding Adress}" />
 </Grid>
</DataTemplate> </
<ListView.ItemTemplate>
```

# Curs

.NET MAUI – Elemente avansate

# Binding cu BindingContext

```
<StackLayout Padding="10, 0">
 <Label Text="TEXT"
 FontSize="80"
 HorizontalOptions="Center"
 VerticalOptions="CenterAndExpand"
 BindingContext="{x:Reference Name=slider}"
 Rotation="{Binding Path=Value}" />

 <Slider x:Name="slider"
 Maximum="360"
 VerticalOptions="CenterAndExpand" />
```

# Binding fara BindingContext

```
<StackLayout Padding="10, 0">
 <Label Text="TEXT"
 FontSize="40"
 HorizontalOptions="Center"
 VerticalOptions="CenterAndExpand"
 Rotation="{Binding Source={x:Reference slider},
 Path=Value}" />

 <Slider x:Name="slider"
 Maximum="360"
 VerticalOptions="CenterAndExpand" />
</StackLayout>
```

# Mostenire BindingContext

```
<StackLayout Padding="10">
 <StackLayout VerticalOptions="FillAndExpand"
 BindingContext="{x:Reference slider}">
 <Label Text="TEXT"
 FontSize="80"
 HorizontalOptions="Center"
 VerticalOptions="EndAndExpand"
 Rotation="{Binding Value}" />
 <BoxView Color="#800000FF"
 WidthRequest="180"
 HeightRequest="40"
 HorizontalOptions="Center"
 VerticalOptions="StartAndExpand"
 Rotation="{Binding Value}" />
 </StackLayout>
 <Slider x:Name="slider"
 Maximum="360" />
</StackLayout>
```

# String format

- Setarea proprietatii StringFormat pentru markup extension-ului Binding catre un string de formatare care contine un placeholder:

```
<TextCell Text="{Binding Description}"
 Detail="{Binding Date, StringFormat='Date is {0:dddd, MMMM dd}'}" />
```

```
<Slider x:Name="slider" /> <Label Text="{Binding Source={x:Reference slider},
Path=Value, StringFormat='The slider value is {0:F2}'}" />
```

- În XAML, șirul de formatare este delimitat de caractere cu ghilimele simple
- Utilizarea proprietății StringFormat are sens numai atunci când proprietatea țintă este de tip string, iar modul de binding este OneWay sau TwoWay.

# Behaviors

- Permit adaugarea de functionalitati la controalele de interfata
- Functionalitatea este implementata intr-o clasa de tip Behaviour si atasata controlului respectiv ca si cum ar fi o parte a controlului
- Exemple de utilizare:
  - Crearea unui validari pentru un control de tip Entry
  - Controlul unui animatii
  - Adaugarea de efecte la un control

# .NET MAUI behaviors

- Cream o clasa care mosteneste clasa Behavior
- Suprascriem metoda OnAttachedTo
- Suprascriem metoda OnDetachingFrom
- Implementam functionalitatea behavior-ului



- ```

class ValidationBehaviour : Behavior<Editor>
{

    protected override void OnAttachedTo(Editor entry)
    {
        entry.TextChanged += OnEntryTextChanged;
        base.OnAttachedTo(entry);
    }

    protected override void OnDetachingFrom(Editor entry)
    {
        entry.TextChanged -= OnEntryTextChanged;
        base.OnDetachingFrom(entry);
    }

    void OnEntryTextChanged(object sender, TextChangedEventArgs args)
    {
        ((Editor)sender).BackgroundColor = string.IsNullOrEmpty(args.NewTextValue) ? Color.FromRgba("#AA4A44") :
        Color.FromRgba("#FFFFFF");
    }
}

```

- Metoda OnAttachedTo este apelata imediat ce behavior este atasat unui control
- Metoda OnDetachingFrom este apelata imediat ce behavior-ul este inlaturat (entry.Behaviors.Clear();)
- Primesc ca parameter o referinta catre controlul care este atasat

Consumarea unui behavior

```
<Editor Placeholder="Enter the description of the shopping list"  
    Text="{Binding Description}"  
    HeightRequest="50" >  
  
    <Editor.Behaviors>  
        <local:ValidationBehaviour />  
    </Editor.Behaviors>  
</Editor>
```

Clasa Geocoding

- Oferă posibilitatea de a găsi coordonatele pentru o adresă și invers: găsirea adresei pentru un set de coordonate
- Proprietăți: Latitudinea, Longitudinea și Altitudinea
- Altitudinea nu este întotdeauna disponibilă – proprietatea Altitude poate fi null sau 0. Dacă este disponibilă, valoarea indică metri deasupra nivelului mării

Clasa Geocoding - exemplu

```
var address = "FSEGA Theodor Mihali Cluj-Napoca";  
  
var locations = await Geocoding.GetLocationsAsync(address);  
  
    var options = new MapLaunchOptions { Name = "Facultatea mea" };  
  
    var location = locations?.FirstOrDefault();  
  
Console.WriteLine($"Latitude: {location.Latitude}, Longitude:  
{location.Longitude}, Altitude: {location.Altitude}");
```



Reverse Geocoding

- Furnizeaza detalii legate de un obiectiv, pentru un set de coordonate furnizat:
- AdminArea - Judet
- CountryCode – Codul tarii (Ex. RO)
- CountryName – Nume tara
- FeatureName – Numele obiectivului de la acele coordonate/ Numarul
- Locality - Localitate
- PostalCode – Cod Postal
- SubAdminArea – Nume municipiu
- SubLocality – Nume Comuna
- SubThoroughfare – Numarul/ Interval Numeric
- Thoroughfare – Strada/Artera



Reverse Geocoding (II)

```
private async Task<string> GetGeocodeReverseData(double latitude = 47.673988,  
double longitude = -122.121513)  
{  
    IEnumerable<Placemark> placemarks = await  
    Geocoding.Default.GetPlacemarksAsync(latitude, longitude);  
    Placemark placemark = placemarks?.FirstOrDefault();  
    if (placemark != null) {  
        return $"AdminArea: {placemark.AdminArea}\n" + $"CountryCode:  
{placemark.CountryCode}\n" + $"CountryName: {placemark.CountryName}\n" +  
        $"FeatureName: {placemark.FeatureName}\n" + $"Locality: {placemark.Locality}\n"  
        + $"PostalCode: {placemark.PostalCode}\n" + $"SubAdminArea:  
{placemark.SubAdminArea}\n" + $"SubLocality: {placemark.SubLocality}\n" +  
        $"SubThoroughfare: {placemark.SubThoroughfare}\n" + $"Thoroughfare:  
{placemark.Thoroughfare}\n";  
    }  
    return "";
```

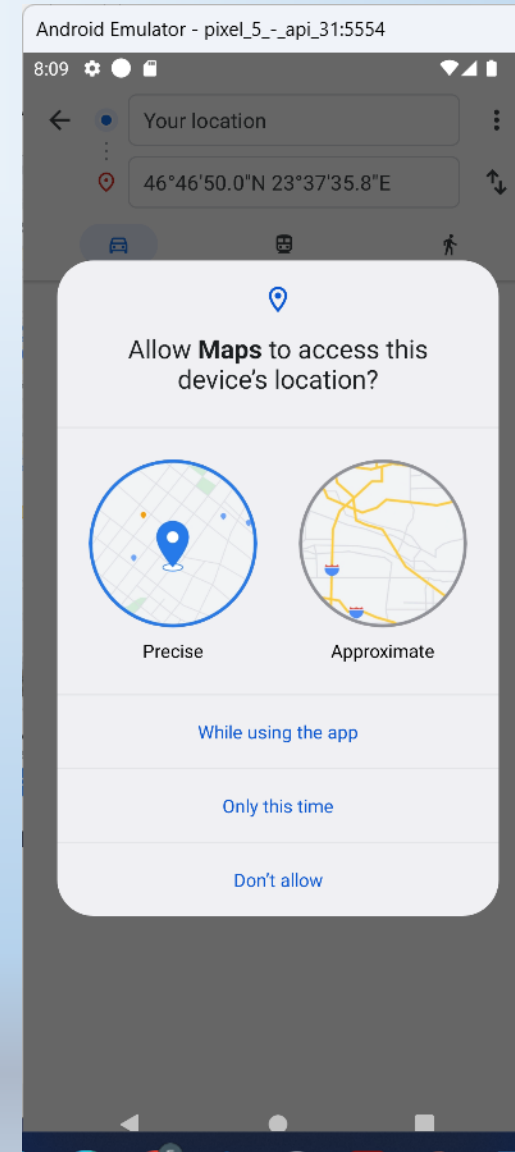
Clasa Geolocation

- Putem afla coordonatele curente de geolocalizare a dispozitivului
- Pentru a folosi functia de geolocalizare sunt necesare anumite actiuni de setup specific platformei
- Ex. Android adaugam in AndroidManifest.xml:

```
<uses-permission  
android:name="android.permission.ACCESS_FINE_LOCATION"  
"/>
```

```
<uses-permission  
android:name="android.permission.ACCESS_COARSE_LOCATION"  
"/>
```

API-ul pentru geolocalizare va cere permisiunea utilizatorului atunci cand este necesar



Gelocalizare

- Putem gasi ultima locatie cunoscuta a dispozitivului prin apelul metodei `GetLastKnownLocationAsync`.
- Este mai rapida decat a executa o interogare noua, dar mai putin exact si poate returna null daca nu exista locatii salvate in cache

```
public async Task<string> GetCachedLocation()
{
    try
    {
        Location location = await Geolocation.Default.GetLastKnownLocationAsync();
        if (location != null) return $"Latitude: {location.Latitude}, Longitude: {location.Longitude}, Altitude: {location.Altitude}";
    }
    catch (FeatureNotSupportedException fnsEx) { // Handle not supported on device exception }
    catch (PermissionException pEx) { // Handle permission exception }
    catch (Exception ex) { // Unable to get location }
    return "None";
}
```


Geolocalizare (II)

- Pentru a afla coordonatele locatiei curente utilizam **GetLocationAsync**

```
try {
```

```
var request = new  
GeolocationRequest(GeolocationAccuracy.Medium);
```

```
var location = await  
Geolocation.GetLocationAsync(request);
```

```
if (location != null) {  
Console.WriteLine($"Latitude: {location.Latitude},  
Longitude: {location.Longitude}, Altitude:  
{location.Altitude}");  
}
```

```
} catch (FeatureNotSupportedException fnsEx) { //  
Handle not supported on device exception }
```

Acuratetea geolocalizarii

	Lowest	Low	Medium	High
Android	500	500	100-500	0-100
iOS	3000	1000	100	10
Windows	1000-5000	300-3000	30-500	≤ 10

Distanța între două locații

- Clasa `Location` definește metoda `CalculateDistance` – permite să calculăm distanța între două locații geografice. Distanța calculată nu ia în considerare drumurile ci distanța între două puncte pe suprafața Pământului

```
Location boston = new Location(42.358056, -71.063611);  
Location sanFrancisco = new Location(37.783333, -122.416667);  
double miles = Location.CalculateDistance(boston,  
sanFrancisco, DistanceUnits.Kilometers);
```

Clasa Map (I)

- Permite unei aplicatii sa deschida o aplicatie de tip “harti” instalata la o locatie specifica
- Utilizeaza metoda OpenAsync cu parameterii de tip Location sau Placemark si optional un parametru de tip MapLaunchOptions

```
public class MapTest
{ public async Task NavigateToBuilding25() {
var location = new Location(47.645160, -122.1306032); var
options = new MapLaunchOptions { Name = "Microsoft Building
25" };
await Map.OpenAsync(location, options);
}
}
```

Clasa Map (II)

- Cand utilizam metoda OpenAsync cu un Placemark este necesara furnizarea urmatoarelor informatii:
 - CountryName
 - AdminArea
 - Thoroughfare
 - Locality

```
public class MapTest {  
    public async Task NavigateToBuilding25()  
    {  
        var placemark = new Placemark { CountryName =  
            "United States", AdminArea = "WA", Thoroughfare =  
            "Kingston Avenue", Locality = "Redmond" };  
        var options = new MapLaunchOptions { Name =  
            "Microsoft Building 25" };  
        await Map.OpenAsync(placemark, options); } }
```

Harta- mod navigare

- Daca se apeleaza OpenMapAsync fara parametrul MapLaunchOptions, harta va fi lansata cu locatia specificata. Optional putem sa calculam ruta de navigare din pozitia curenta la care se afla dispozitivul prin setare NavigationMode la MapLaunchOptions

```
public class MapTest {  
    public async Task NavigateToBuilding25()  
    {  
        var location = new Location(47.645160, -122.1306032);  
        var options = new MapLaunchOptions { NavigationMode =  
            NavigationMode.Driving };  
        await Map.OpenAsync(location, options);  
    }  
}
```

- NavigationMode suporta Bicycling, Driving si Walking

Curs 11

Servicii Web

API web

- Scop - transferul de date, independent de limbaj sau framework-ul folosit
- ASP.NET Core suporta crearea de servicii REST, cunoscute si sub denumirea de API-uri web
- Pentru a gestiona cereri, un API web utilizeaza controale
- Controalele intr-un API web sunt clase care deriva din clasa ControllerBase

Representational State Transfer

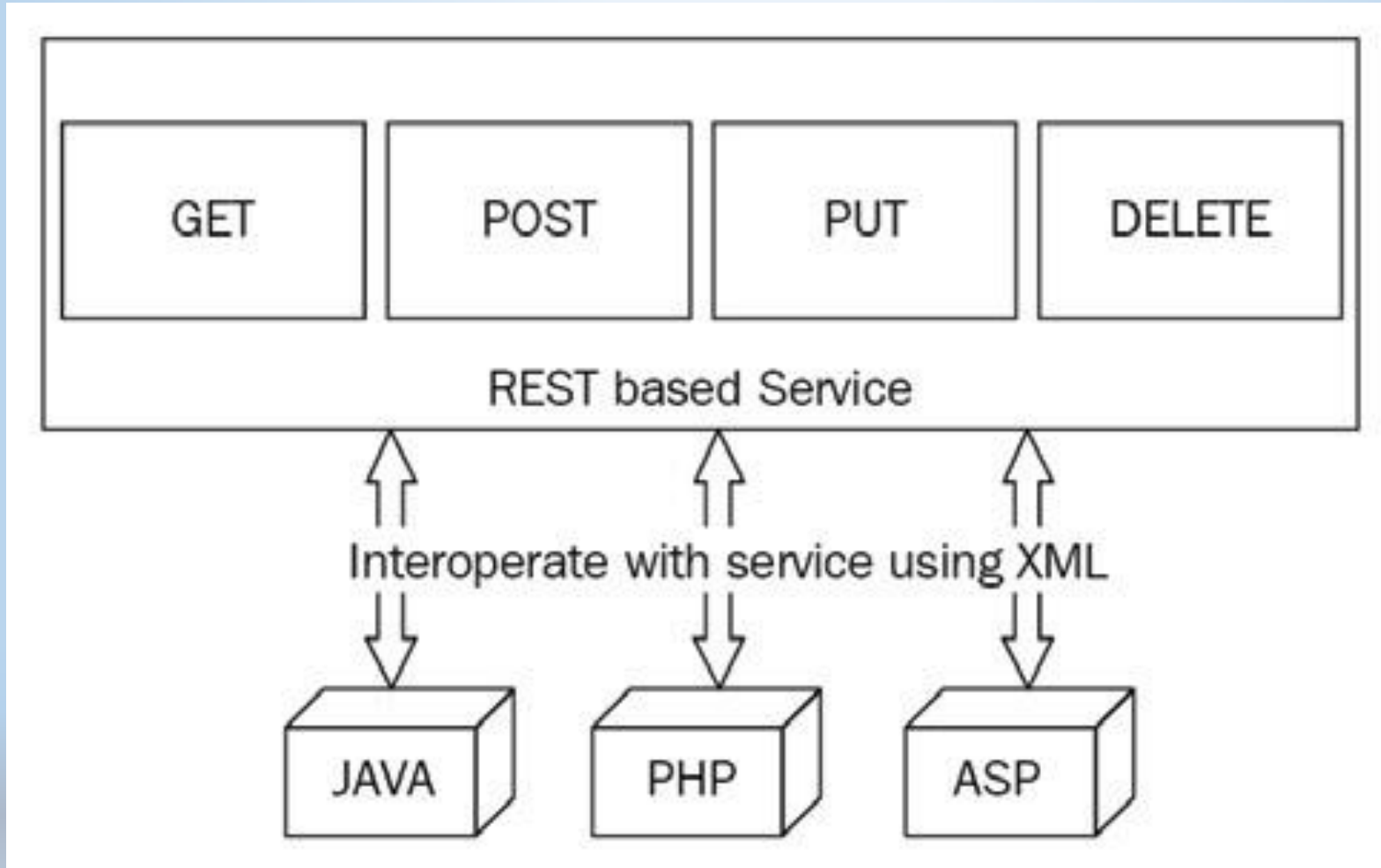
- Introduction de Roy Fielding
- Architectural Styles and the Design of Network-based Software Architectures, 2000

<https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

REST

- Un stil arhitectural care defineste un set de reguli pentru a construi servicii web
- Stil arhitectural = concept cu principii predefinite
- Implementare principiilor care reprezinta componente in aplicatie
- Implementarea REST va fi diferita de la dezvoltator la dezvoltator – nu exista un stil de implementare fixat
- Diferit de pattern-uri arhitecturale precum MVC, care reprezinta concepte si implementari concrete. MVC este un pattern arhitectural avand o structura fixata care defineste cum componentele interactioneaza unele cu altele si nu pot fi implementate diferit

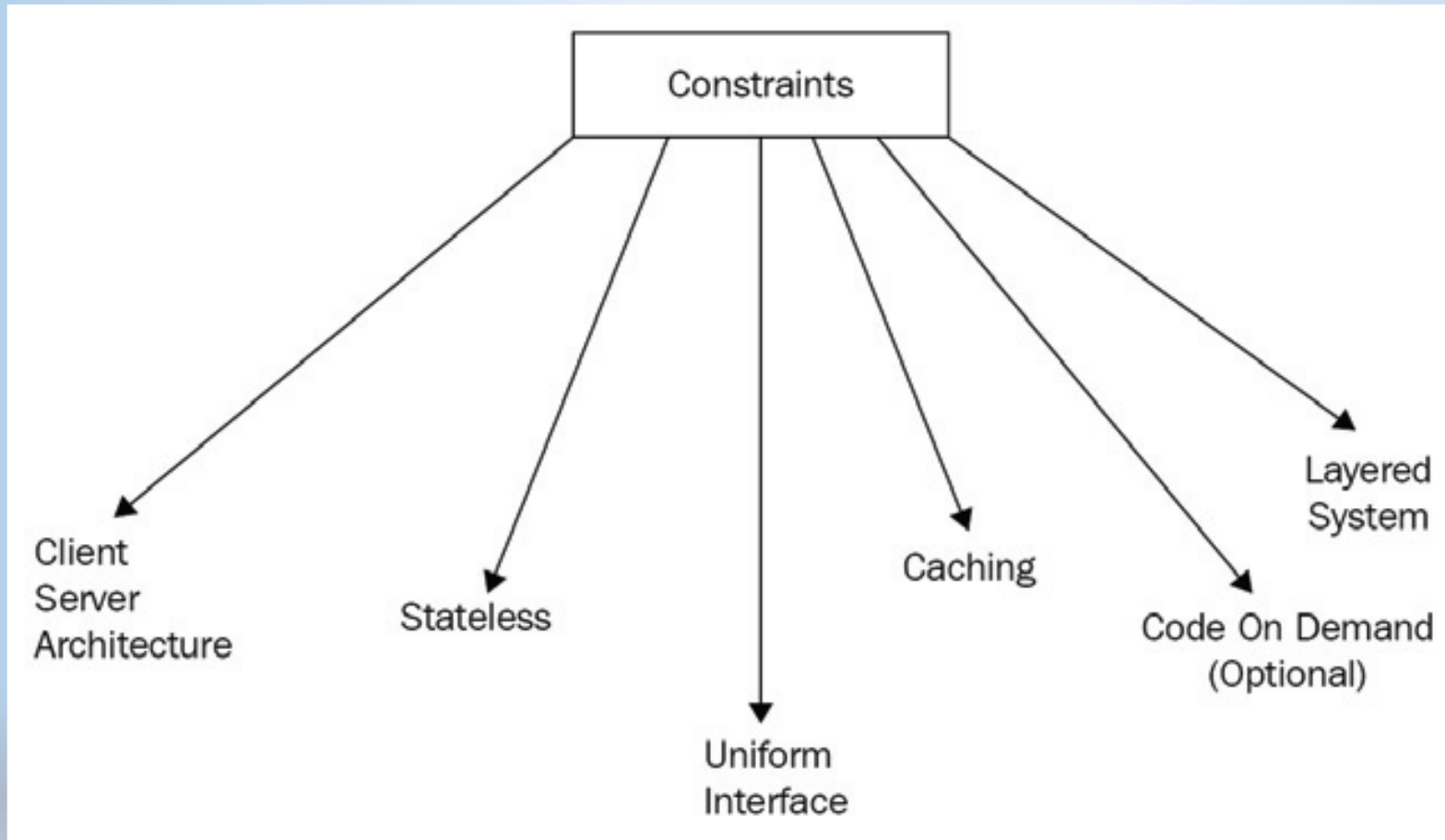
Serviciu REST



Caracteristici REST

- Raspunsul trimis de la server ca urmare a cererii clientului este o resursa intr-un anumit format
- Cele mai comune formate sunt: .json, .xml, .pdf, .doc
- REST este stateless- starea sistemului este intodeauna diferita : cand se primeste o cerere la server, cererea este gestionata si apoi uitata, astfel ca urmatoarea cerere nu depinde de starea celei anterioare.
- Fiecare cerere este gestionata de server in mod independent
- Cererile sunt realizate folosind o conexiune HTTP, fiecare continand un URI (Uniform Resource Identifier) pentru localizarea resursei solicitate

Constrangeri REST



1. Arhitectura client-server

- Clientul (consumatorul serviciului) nu trebuie sa cunoasca detalii despre cum serverul proceseaza datele si le stocheaza in baza de date
- Serverul nu trebuie sa depinda de implementarea clientului, in special de UI
- Clientul si serverul nu sunt una si aceasi entitate si fiecare poate exista in lipsa celuilalt
- Serviciul cand interactioneaza cu clientii, ofera suficiente informatii: cum sa fie consumat, ce operatii se pot efectua la utilizarea lui
- Clientul si serverul pot fi complet decuplate daca adera la toate constrangerile REST

2. Stateless

- Un serviciu REST nu mentine starea aplicatiei =>Stateless
- O cerere intr-un serviciu REST nu depinde de cererile anterioare. Serviciul trateaza fiecare cerere independent
- Fiecare cerere de la client către server trebuie să conțină toate informațiile necesare pentru înțelegerea și finalizarea cererii.
- Serverul nu poate utiliza orice informații de context stocate anterior pe server. Din acest motiv, aplicația client trebuie să păstreze în întregime starea sesiunii.

3. Caching

- Caching trebuie sa fie aplicat cand este posibil, iar resursele trebuie sa se autodeclare cacheable.
- Daca sunt cacheable, serverul stie durata pentru care raspunsul este valid
- Daca clientul are acces la un raspuns din cache valid, nu se mai executa cererea si se utilizeaza copia din cache
- Specificare explicita a unei resurse cachable se face in headerul Cache-Control unde se poate seta si durata valabilitatii copiei
- Imbunatatirea performantei pe partea de client si o mai buna scalabilitate pe partea de server deoarece se reduce incarcarea

4. Code on demand (optional)

- De cele mai multe ori serviciul returneaza reprezentari statice de resurse - .json, .xml
- Serviciul poate returna cod executabil pentru a extinde functionalitatea aplicatiei

5. System multi nivel

- Fiecare nivel este restrictionat sa acceseze doar urmatorul nivel din ierarhie
- Deployment-ul API-urilor pe serverul A, stocarea datelor pe serverul B si cererile de autentificare pe serverul C – clientul nu stie unde e conectat
- Arhitectura bazata pe layere – ajuta la o mai buna gestionare a complexitatii si imbunatateste mentenabilitatea codului.

6. Interfata uniforma

- Decuplarea clientului de serviciu
- REST este definit de patru constrangeri pentru interfata:
- **Identificarea resurselor:** Un URI este utilizat pentru a identifica o resursa; Interfața trebuie să identifice în mod unic fiecare resursă implicată în interacțiunea dintre client și server
- **Manipularea resurselor prin reprezentari:** Cand un client detine o resursa are suficienta informatie pentru a modifica sau sterge resursa; Resursele ar trebui să aibă reprezentări uniforme în răspunsul serverului. Consumatorii API ar trebui să folosească aceste reprezentări pentru a modifica starea resursei de pe server.
- **Mesaje auto-descriptive:** Mesajele trimise trebuie sa contina suficienta informatie despre datele procesate. Fiecare reprezentare a resursei ar trebui să conțină suficiente informații pentru a descrie modul de procesare a mesajului. De asemenea, ar trebui să furnizeze informații despre acțiunile suplimentare pe care clientul le poate efectua asupra resursei
- **Hypermedia ca si motor al starii aplicatiei:** Reprezentarea returnata de serviciu poate sa contina link-uri catre alte resurse; Clientul ar trebui să aibă doar URI inițial al aplicației. Aplicația client ar trebui să conducă în mod dinamic toate celelalte resurse și interacțiuni prin utilizarea hyperlinkurilor.
- REST definește o interfață uniformă pentru interacțiunile dintre clienți și servere. De exemplu, API-urile REST bazate pe HTTP folosesc metodele standard HTTP (GET, POST, PUT, DELETE etc.) și URI-urile (Uniform Resource Identifiers) pentru a identifica resursele

Metode

Metoda	Operatia realizata pe server	Tipul Metodei
GET	Citeste/Incarca o resursa	Safe
PUT	Insereaza sau actualizeaza o resursa daca aceasta exista	Idempotent
POST	Insereaza sau actualizeaza o resursa daca aceasta exista	Nonidempotent
DELETE	Sterge o resursa	Idempotent
OPTIONS	Afiseaza o lista cu operatiile permise pentru o resursa	Safe
HEAD	Returneaza doar Headerul fara body pentru cererea respectiva	Safe

Tipuri de metode

- *Safe* – operatia executata nu afecteaza valoarea initiala a resursei
- GET, OPTIONS, HEAD – doar incarca sau citesc resursa
- *idempotent* (poate fi repetata) – operatia executata are acelasi rezultat indiferent de cate ori o executam
- DELETE si PUT opereaza pe o resursa specifica, operatia putand fi repetata

POST vs. PUT

- Ambele pot sa adauge sau sa actualizeze o resursa
- POST este nonidempotent, nu este repetabil – va crea de fiecare data o noua resursa daca nu se trimite URI-ul exact
- PUT valideaza existenta resursei prima data, iar daca exista o actualizeaza, in caz contrar o creaza

- PUT <https://localhost:44327/api/shoplists>- nu merge pt ca nu are ID-ul resursei specificat
- PUT <https://localhost:44327/api/shoplists/5> - create/update
- POST <https://localhost:44327/api/shoplists>- creeaza o noua resursa; la apeluri ulterioare – inregistrari duplicat, creaza o noua resursa cu aceleasi date
- POST <https://localhost:44327/api/shoplists/18>- update
- PUT creaza sau actualizeaza o resursa pentru un URI specificat
- PUT si POST se comporta la fel daca resursa exista deja
- POST fara specificarea ID-ului creaza o resursa de fiecare data cand se executa



Avantaje REST

- Independenta fata de o platforma sau un limbaj de programare
- Metode standardizate prin utilizarea HTTP
- Nu pastreaza starea clientului pe server
- Suporta caching
- Accesibil tuturor tipurilor de aplicatii client: Mobile, web, desktop

Dezavantaje REST

- Daca standardele nu sunt aplicate corect vor exista dificultati pentru clientul care consuma serviciul
- Securitatea este o preocupare, daca nu exista procese care sa restrictioneze accesul la resurse

Coduri de stare

- Cand serverul returneaza raspunsul returneaza si codul de stare pentru a informa clientul despre cum s-a executat cererea pe server
- **200 OK** Raspuns standard pentru cereri HTTP executate cu success
- **201 CREATED** Raspuns standard pentru o cerere cand o resursa a fost creata cu success
- **204 NO CONTENT** Raspuns standard pentru cereri executate cu success dar nu s-a returnat nimic in body
- **400 BAD REQUEST** Cererea nu poate fi procesata datorita sintatxei gresite, marimii prea mari sau alt motiv
- **403 FORBIDDEN** Clientul nu are permisiunea sa acceseze resursa solicitata
- **404 NOT FOUND** Resursa nu exista
- **500 INTERNAL SERVER ERROR** Cand apare o eroare sau o exceptie la procesarea codului pe server

Coduri de stare implicite

- GET: Returneaza **200 OK**
- POST: Returneaza **201 CREATED**
- PUT: Returneaza **200 OK**
- DELETE: Returneaza **204 NO CONTENT** daca operatia a esuat

RESTful APIs

- Un serviciu web care respecta principiile REST se numeste RESTful API
- In mod uzual serviciile web RESTful utilizeaza mesaje JSON pentru a returna date clientului.
- JSON – format de date de tip text de dimensiuni reduse – data-interchange

```
{  
  "id":1  
  "amount": 8.50,  
  "date":"2020-05-09T00:00:00Z",  
  "description": "tea"  
}
```

Maparea URI

- Cand un Web API primeste o cerere ruteaza cererea respectiva la o actiune
- Actiunile – metode in clasa Controller
- Gaseste URI pe baza specificarilor de rutare definite astfel:
 - Pentru a gasi controllerul adauga “controller” la valoarea variabilei {controller}-shoplists
 - Pentru a gasi actiunea cauta in actiunile definite in controller care sunt marcate cu acelasi atribut HTTP
 - {id} este mapat cu parametrul unei actiuni.
- <https://192.168.1.9:45455/api/shoplists/{0}>

Atribute

- [ApiController] – aplicat la controale specifice

[ApiController]

[Route("[controller]")]

```
public class WeatherForecastController : ControllerBase
```

[ApiController] – controale multiple

- Cream o clasa de baza

```
[ApiController] public class MyControllerBase :  
ControllerBase { }
```

```
[Route("[controller]")]  
public class PetsController : MyControllerBase
```

Atributul [Route]

- Cand utilizam atributul [ApiController] este necesar atributul [Route]

```
[ApiController]
```

```
[Route("[api/controller]")]
```

```
public class WeatherForecastController : ControllerBase
```

Action methods sunt inaccesibile prin rutele conventionale specificate in Program.cs

Tipul de return

- ActionResult

Mai multe tipuri de ActionResult sunt posibile intr-o actiune.

Tipurile ActionResult reprezinta diferite coduri HTTP status:

- BadRequestResult(400) / BadRequest()
- NotFoundResult(404)
- OkObjectResult(201)

Clasa HttpClient si HttpResponseMessage

- Clasa HttpClient este utilizata pentru a trimite si primi cereri folosind protocolul HTTP
- Oferă functionalitati pentru a trimite cereri HTTP si a primi raspunsuri HTTP de la un URI
- Clasa HttpResponseMessage reprezinta un mesaj de tip raspuns primit de la un serviciu web in urma realizarii unei cereri HTTP
- Contine informatii despre raspuns, incluzand codul de stare
- Continutul poate fi citit cu metoda ReadAsStringAsync

GET

```
[Produces("application/json")]
[ApiController] //indica faptul ca clasa va fi utilizata pentru raspunsuri HTTP
[Route("[controller]")] //conventie de nume - la runtime se va concatena cu numele textului din URL
public class ShopListsController : ControllerBase
{
    [HttpGet("{id}")] //identifica o actiune asociata cu metoda GET
    public async Task<ActionResult<ShopList>> GetShopList(int id)
    {
        var shopList = await _context.ShopList.FindAsync(id);
        if (shopList == null)
        {
            return NotFound();
        }
        return shopList;
    }
}
```

POST

[HttpPost]

```
        public async Task<ActionResult<ShopList>>
PostShopList(ShopList shopList)
    {
        _context.ShopList.Add(shopList);
        await _context.SaveChangesAsync();

        return CreatedAtAction("GetShopList", new { id =
shopList.ID }, shopList); // returneaza codul de stare 201
    }
```

PUT

[HttpPut("{id}")] //primeste ID-ul din URL

```
public async Task<IActionResult> PutShopList(int id, ShopList shopList)
{
    if (id != shopList.ID)
    {
        return BadRequest();
    }

    _context.Entry(shopList).State = EntityState.Modified;

    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        if (!ShopListExists(id))
        {
            return NotFound();
        }
        else
        {
            throw;
        }
    }

    return NoContent();
}
```

DELETE

```
[HttpDelete("{id}")]
public async Task<ActionResult<ShopList>> DeleteShopList(int id)
{
    var shopList = await _context.ShopList.FindAsync(id);
    if (shopList == null)
    {
        return NotFound();
    }

    _context.ShopList.Remove(shopList);
    await _context.SaveChangesAsync();

    return shopList;
}
```

Consumarea serviciilor web

- Serviciile REST pot sa fie sau nu parte a unei aplicatii Web
- O aplicatie Web poate sa apeleze sau sa consume servicii web externe sau servicii care fac parte din aceasi aplicatie
- Programul care permite interactiunea (cerere, raspuns) intre serviciu si aplicatia care consuma acel serviciu se numeste *client*

