MPD

# Introducere in .NET, WPF, XAML

## Istoria

**2002 -** .NET framework 1.0 – C# 1.0

* C# - nou limbaj de prg dezvoltat pt .NET framework – Anders Hejlsberg
* .NET framework continea 3000 de clase la momentul cand a fost creat

**2005 - .NET framework 2 - C#2**

* Generics – crearea de parametrii generic fara a se declara tipul acestora

**2005 - .NET framework 3**

* **WPF, WCF, WF –** librarii noi

**2008 .NET 3.5 - C#3**

* **LINQ –** interogari folosind aceeasi sintaxa pt colectii de obiecte, baze de date si fisiere XML

**2010 .NET 4 - C#4**

* Facilitate noi pt programare paralela

**2012 .NET 4.5 - C#5**

* **Programare asincrona –** device-uri cu touch screen

**2015 .NET core 1.0 - C#6**

* **Runtime distribuit** cu fiecare aplicatie, **open-source, multi-platforma**

**2017 .NET core 2.0 - C#7**

* **Noi** facilitati pt **prg functionala**

**2019 .NET core 3.0 - C#8**

* Imbunatatiri pt prg functionala si asincrona

# Aplicatii de tip Desktop

* Nu e nevoie de conexiune la Internet
* Mai buna interactiune cu utilizatorul
* Pot oferi o performanta mai buna decat app web
* Utilizarea firelor de executie e mai facila

## Clasa, Namespace, Assembly

* Mai multe Clase = Namespace
* Mai multe Namespaces = Assembly

## Windows Presentation Foundation (WPF) & XAML

* WPF ofera posibilitatea de **separare a interfetei logice de programare**
* **Interfata** – definita folosind **XAML markup**; acest markup poate fi **conectat** la un **fisier cu cod C# (code-behind) pt functionalitate**
* XAML – definire de el simple, grafica interactiva, animatii, functionalitati multimedia

## Namespace-ul System.Windows

* = namespace-ul principal al WPF
* Contine **2 clase principale: Application, Window**

## Clasa Application

* App.xaml.cs
* **App.xaml – fereastra cu care porneste app StartupUri**
* Obiect din clasa Application – o instanta a unei applicatii WPF care ruleaza
* Metoda **Run() – porneste app**
* **Proprietati:**
  + **StartupUri –** seteaza sau furnizeaza val pentru un URI care specifica fereastra care se deschide automat cand o app porneste
  + **Current –** permite accesul din orice parte a app la ob app care ruleaza
  + **Windows –** acces la o colectie care reprez ferestrele din memorie pt app WPF curenta

## Clasa Window

* Un obiect din clasa System.Windows.Window = o fereastra detinuta de ob din clasa Application (derivata)
* **System.Windows.Controls.ContentControl** – poate **găzdui o singură piesă de conținut**, care se referă la datele vizuale plasate în interiorul suprefeței controlului prin intermediul proprietății Content
  + **Toate controalele WPF mostenesc clasa de baza Control**
  + **Propr de baza:**
    - **Marimea el**
    - **Opacitate**
    - **Ordinea de accesare a controalelor**
    - **Culoare fundal…**

## XML Namespaces

* http://schemas.microsoft.com/winfx/2006/**xaml**/**presentation -**  **mapează** System.Windows, System.Windows.Controls, System.Windows.Data, System.Windows.Ink, System.Windows.Media, System.Windows.Navigation, etc.
* http://schemas.microsoft.com/winfx/2006/**xaml - include cuvinte cheie XAML**

## Property element syntax

<DefiningClass>

<DefiningClass.PropertyOnDefiningClass>

<! -- Value for prop here -->

</DefiningClass.PropertyOnDefiningClass>

</DefiningClass>

## Attached Properties

* Permite unui copil sa seteze valoarea pt o proprietate care e definita intr-un el parinte

<ParentElement>

<ChildElement ParentElement.PropertyOnParent = “Value” >

</ParentElement>

## WPF Styles

* **Clasa System.Windows.Style** – proprietatea Setters
* **Triggers** – permite **captarea diverselor evenimente**
* **BasedOn** – construirea unui **nou stil pe baza unui stil existent**
* **TargetType** – aplicarea unor **constrangeri** care **definesc unde se aplica stilul**

# Manageri de continut

* Gestiunea **pozitiei controalelor** in fereastra
* Gestiunea **comportamentului controalelor la** **redimensionarea** ferestrei

## Canvas

* **Pozitionare absoluta**
* Controalele raman acolo unde au fost plasate in design time

## WrapPanel

* **Modificare pozitie** pe masura ce fereastra e redimensionata

## StackPanel

* Similar cu WrapPanel
* **Wrap vs Stretch**

## Grid

* Definire si configurarea fiecarei **coloane/rand**
* Atribuirea de continut fiecarei celule folosind **attached property**
* **Dimensionarea liniilor si coloanelor**
  + **Absoluta**
  + **Autodimensionare**
  + **Relativa**
* **GridSplitter** – permit **utilizatorilor sa redimensioneze liniile sau coloanele**

## DockPanel

* La redimensionarea ferestrei, **fiecare element ramane pe partea specificata a panelului (Top, Bottom, Left, Right)**

## Scrolling

# Delegates

## Delegates

* **Contin referinte la metode** care **au acelasi tip de return si acelasi tip de parametrii ca si delegate-ul**

### Declararea Delegates

* Precizam compilatorului ce fel de metoda va referentia acel delegate apoi cream instante din acel delegate
* Definim **detalii despre semnatura si tipul de return al metodei pe care o referentiaza**

### Action<T>

* Referentiaza o metoda cu **tip de return void**
* Action<in T, in T2….> - 16 param maxim

### Func<T>

* Referentiaza o **metoda care are tip de return**
* Func<**in** T, … **out** Result> - 16 param maxim

### Multicast Delegate

* Un delegate poate referential mai multe metode
* **Apeleaza succesiv** fiecare metoda
* **Return type void**

## Expresii Lambda

*Ex: Func<double, double> square = x => { return x \* x; }*

*Ex: Func<double, double, double> twoParams = (x,y) => x\*y;*

*Console.WriteLine(twoParams(3,2));*

## Evenimente

* Un **mesaj trimis** de catre un **obiect** la care **programul raspunde prin executarea unei bucati de cod**
* Evenimentele pot fi lansate de catre controale/alte ob ale app
* Bazate pe Delegates

*Ex: Button – Ev Click*

* O **metoda handler** (handler de evenimente) trebuie definita – e **invocata cand apare ev Click**
* Cand apare un ev app trebuie sa stie ce metoda sa execute – transmiterea metodei care gestioneaza ev ca si parametru la un delegate

### Pasi

1. **Declararea delegate-ului**

*public delegate void EventHandler(object sender, EventArgs e);*

1. **Declararea evenimentului gestionat prin delegate**
2. **Scrierea metodei care sa raspunda la eveniment (**in clasa controller care raspunde la eveniment**)**

*private void button1\_Click(object sender, System.EventArgs e) { }*

1. **Legarea evenimentului de handlerul sau**

*this.button1.Click += new System.EventHandler(this.button1\_Click);*

***element.Eveniment += delegate(metoda ca argument la delegate)***

### Arhitectura de gestiune a evenimentelor

* Fiecare **eveniment are**:
  + Un **obiect** (control) **trimitator** care **arunca (lanseaza) evenimentul**
  + Un **obiect** (control) **receptor** care **are metoda handler** a ev
* **Delegate = obiect intermediar intre trimitator si receptor**
* **Un delegate memoreaza o referinta catre metoda care va gestiona evenimentul**
  + **Defineste numele handlerului si tipul args**
* Sunt **multicast**

### Handlere de evenimente

* **Metode legate de evenimente**
* la aparitia evenimentului se executa codul din handler
* **Argumentele handlerului**
  + Obiect care a aruncat ev (sender)
  + Ob care contine **info specifica evenimentului** (tip EventArgs)

### Tipuri de evenimente

* **Directe** – declansat si gestionat de obiectul de origine
* **Bubbling** – de la obiectul de origine catre nivelurile superioare din arbore
* **Tunneling** – de la elementul parinte (ex. Window) catre obiectul de origine

### Evenimente rutate

* **Rafinarea** modului standard de gestiune a ev
* **Modelul propaga automat un eveniment** (sus/jos) intr-un **arbore** de obiecte, **executand handlerul de eveniment corespunzator**
* un **eveniment creat de catre un control** sa **apara la un alt control** dintre containerele in care sta controlul sursa
* permite programatorului sa **decida nivelul la care va prinde evenimentul** lansat in GUI
* **se simplifica scrierea codului**
* **se asigura consistenta la tratarea ev**

### Evenimente Bubbling

* aruncate intai de controalele unde isi au originea
* ev sunt **prinse si tratate mai intai la control si apoi la containerul care il contine**
* permit unui **grup complex de obiecte** sa se **comporte ca un singur element**

### Evenimente de tip tunel

* invers ca si evenimentele bubbling
* e **aruncat intai de catre cel mai de sus container** din arborele visual si apoi **in jos** de catre fiecare container **pana la controlul de origine**
* permit ca ev sa fie prinse si tratate mai **intai la container si apoi la control**
* sunt marcate cu cuvantul **Preview**
* ev de tip tunel sunt **definite in pereche cu cate un ev bubbling**
* **intai se arunca ev de tip tunel si apoi cel bubbling**
  + **argumentele sunt partajate intre acestea**

### RoutedEventArgs

* fiecare eveniment include o **instanta a clasei RoutedEventArgs** (sau care **mosteneste**)
* contine **info** despre **eveniment si sursa acestuia**
* **Proprietati:**
  + **Handled –** daca ev a fost tratat sau nu; prin setarea la True se poate opri tratarea ulterioara a ev de tip bubbling sau tunel
  + **OriginalSource –** se obtine obiectul care a aruncat prima data ev
  + **Source –** obiectul care a aruncat ev
  + **RoutedEvent – evenimentul care a fost ridicat**

### Tratarea unui eveniment de tip tunel sau bubbling

* Pt a **inhiba tratarea unui ev** de acest tip la vreun **nivel in ierarhie**, se foloseste **proprietatea Handled = true**.
* Daca se seteaza Handled la un ev de tip tunel => Handled si ev bubbling pereche

### Evenimente la nivel de aplicatie

* **Activated**: aplicatia primeste focus de la un alt program
* **Deactivated**: aplicatia lasa focusul unui alt program
* **DispatcherUnhandledException:** cand o exceptie nu e tratata
* **Exit:** cand app e inchisa
* **SessionEnding:** se inchide sesiunea Windows adica la **logoff** sau **shutdown**
* **Startup:** app e pornita

# Comenzi WPF

* Un eveniment este specific unei clase de baza si poate fi utilizat doar pentru clase care extend aceasta clasa de baze
* **Comenzile WPF sunt asemanatoare ev dar sunt independete de un control specific si pot fi aplicate mai multor tipuri de controale**
* **Permit mai multor surse sa invoce acceasi logica a comenzii**
* O comanda poate specifica daca acea actiune e disponibila (**CanExecute**)
* **Comenzile**
  + sunt **taskuri de nivel inalt** care sunt executate in aplicatie
  + se pot asocia orice nr de ob UI sau inputuri cu o comanda a.i. sa se lege comanda de un handler care e executat atunci cand controalele sunt active
  + **comanda poate fi dezactivata** (cand el de interfata asociat e dezactivat)

### Utilizare

* **RoutedCommand** predefinite
* Un control care are suport native pentru gestiunea/invocarea comenzilor

### Comenzi – arhitectura

* **Obiectul Command: actiunea** care se **executa**
* **Sursa: obiectul** care **invoca** comanda
* **Targetul: obiectul** pe care se **executa** comanda
* **CommandBinding: obiectul** care **mapeaza logica** comenzii la comanda

**Comenzi predefinite**

* **ApplicationCommands:** Close, Copy, Cut, Delete, Find, Open, Paste, Save, SaveAs, Redo, Undo
* **ComponentCommands:** MoveDown, MoveFocusBack, MoveLeft/Right, ScrollToEnd/Home
* **EditingCommands:** AlignCenter, CorrectSpellingError, DecreaseFontSize, EnterLineBreak
* **MediaCommands:** FastForward, NextTrack, Play, Rewind
* **NavigationCommands:** BrowseBack/Forward, Favorites, Last/NextPage, Zoom

*Pentru a putea utiliza aceste comenzi trebuie sa cream binding-uri si handlere pt acestea.*

### Implementarea unei comenzi

1. **Comanda** care se doreste a se utiliza
2. **Asociare** comanda cu control/input
3. **Creare metoda** care gestioneaza comanda
4. Creare control **CommandBinding**
5. Se **adauga CommandBinding** la colectia **Commands** a controlului sau la controlul **Window**

### Handlere de comenzi

* CommandBinding leaga comanda de un handler
* Handler corect al unei comenzi:
  + *private void myCommandHandler(object sender, ExecutedRoutedEventArgs e) { handle the command here }*
    - *ExecutedRoutedEventArgs – derivate din RoutedEventArgs*
* Creare CommandBinding
  + *CommandBinding abinding = new CommandBinding();*
  + *abinding.Command = ApplicationCommands.Find;*
  + *abinding.Executed += new ExecutedRoutedEventHandler(myCommandHandler);*
  + *this.CommandBindings.Add(abinding);*

### Comenzi Bubbling

* se rostogolesc catre containerele din varful ierarhiei
* fiecare control are propria colectie CommandBinding
* Se poate opri rostogolirea prin Handled = true

### Dezacticarea comenzilor

* Comenzile care **nu sunt associate in CommandBinding sunt automat dezactivate**
* Fiecare **control care are in proprietatea Command asignat o comanda disables devine dezactivat**
* Pentru dezactivare se utilizeaza Command.CanExecute

# WPF Data-Binding

* Data Binding = conectarea proprietatilor controalelor cu date care se modifica pe parcursul ciclului de viata al app

Operatii data-binding:

*Sursa: ex: Proprietatea de tip Boolean property sau o BDR*

*Destinatie: prop controlului UI care utilizeaza datele (CheckBox sau TextBox)*

## Tipuri de data-binding

* **One-time:** de la sursa -> destinatie si se face o singura data cand se porneste app
* **One-way:** de la sursa -> destinatie; util pt datele readonly deoarece nu se pot modifica datele
* **Two-way:** utilizatorul poate modifica datele prin interfata (binding realizat in ambele directii)
* **One-way-to-source:** destinatie -> sursa; daca se modifica destinatia, sursa se modifica

# Accesul la date – Entity Framework

## Entity Framework - ADO .NET API

* Introdus incepand cu versiune .NET 3.5
* EF Core – versiune multi-platforma a lui EF
* Permite **interactiunea cu datele din BDR** utilizand un **model mapat** cu functionalitatea de business din aplicatie
* Un **set de date** – o **colectie de randuri si coloane** => **colectie obiecte** tipizate denumite **entitati**
* **Entitatile** pot fi **interogate utilizand LINQ**

## Rolul entitatilor

* **Entitatile** = **model conceptual** al BD – **EDM** (entity data model)
* **EDM** – un **set de clase client-side** care sunt **mapate la o BD** pe baza **conventiilor** definite in EF si pe baza unor **configurari**

## Componente EF

1. Clasa **DbContext**
   * + Utilizata pt **interogarea BD** si pentru a **grupa modificarile** pentru a putea fi scrise in bloc
     + **GetValidationErrors** – **valideaza intrarile** si returneaza o colectie DbEntityValidationResults
     + **SaveChanges – salveaza in BD toate modificarile** facute in context; returneaza nr de entitati afectate
     + **Configuration –** ofera **acces la proprietatile de configurare ale contextului**
     + **Database –** ofera un **mecanism pt creare/stergerea/verificarea BD,** executa **procedure stocate, expune functionalitati** legate de **tranzactii**
   * **Evenimente**
     + **ObjectMaterialized –** se apeleaza cand un nou obiect entitate este creat din BD ca rezultat al unei interogari
     + **SavingChanges – se apeleaza cand modificarile se salveaza in BD, dar inainte de a deveni persistente**
2. **Clasa derivate din DbContext –** se trimite la **constructor** **numele string-ului de conexiune** pt clasa context
3. **Entity Sets DbSet<T> -** **adaugarea de tabele in context**
   * + *Membrii DbSet<T>*
       - ***Add*** *– marcate cu starea Added, inserate in BD cand se apeleaza SaveChanges*
       - ***Create*** *– creare instanta de tipul specificat*
       - ***Find*** *– gaseste un rand dupa PK si returneaza un obiect reprezentand acel rand*
       - ***Remove*** *– marcheaza un obiect pt stergere*
4. **DbChangeTracker –** realizeaza tracking-ul automat al starii oricarui obiect DbSet<T> in cadrul unui DbContext

**Starile entitatilor:**

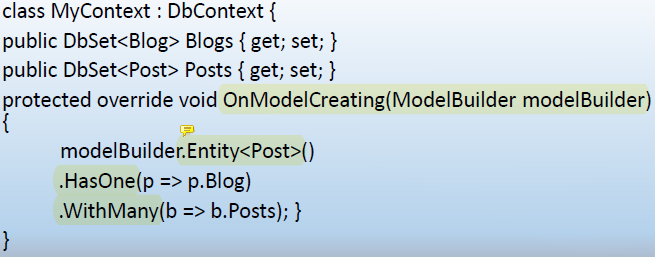
* + - **Detached –** obiectul exista, dar nu se face inca tracking pe el (se afla in starea aceasta imediat ce a fost creat si inainte de a fi adaugat la ob context)
    - **Unchanged –** obiectul **nu a fost modificat** de cand a fost atasat la context sau de la ultimul apel a lui SaveChanges()
    - **Added –** obiectul este **nou** si a fost **adaugat** la obiectul context iar metoda **SaveChanges() nu a fost apelata**
    - **Deleted –** obiectul a fost **marcat pentru stergere**
    - **Modified –** una din **proprietatile obiectului a fost modificata** si metoda **SaveChanges() nu a fost apelata**

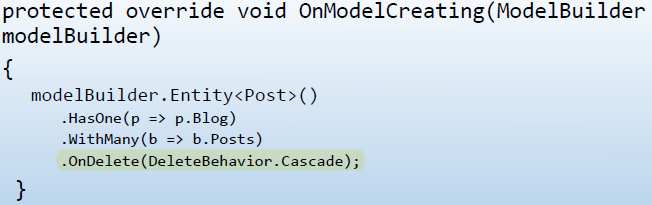
1. **Adnotari –** attribute utilizate pentru modelarea entitatilor in vederea maparii cu BD
   * + **Key**
     + **Required**
     + **ForeignKey**
     + **NotMapped –** o prop nu e mapata pe un camp al BD
     + **ConcurrencyCheck –** marcheaza un camp pt a fi verificat in cazuri de concurenta cand se realizeaza inserari/actualizari/stergeri
     + **Table/Column**
     + **DatabaseGenerated –** spec faptul ca un camp e generat din BD, cum ar fi Identity

## Modelarea relatiilor intre entitati

* O relatie defineste modul in care 2 entitati se raporteaza una la cealalta
* Intr-o BDR -> constrangere de cheie straina

### Definire termini

* ***Entitate dependenta –*** o entitate care contine proprietati de tip **FK**. Este referita uneori ca si entitatea **copil** din relatie
* ***Entitate principala –*** entitatea care contine proprietatea de tip PK. E 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 –*** *ref la* ***mai multe*** *entitati*
  + ***Reference navigation property –*** *ref la o* ***singura*** *entitate*
* ***Relatii definite complet – navigation properties*** *definite la ambele capete ale relatiei si o proprietate de tip PK definite in entitatea dependent*
* ***No foreign key property – nu e obligatoriu*** *sa definim o prop* ***FK*** *in* ***entitatea dependent***
  + *Daca nu e gasita o FK se* ***creaza automat*** *o proprietate de tip* ***FK shadow***
  + *Valoarea si starea acelor entitati sunt gestionate de ChangeTracker*
* ***Configurare manuala prin Fluent API***
* ***Cascade Delete***



* ***Relatie one-to-one:*** au un reference navigation property la ambele capete ale relatiei
* ***Relatie many-to-many:*** necesita un **collection navigation property la ambele capete**
  + Sau cu **entitate de tip join**
* ***Stergere***

***Metoda normala de DELETE:*** *Cautam inregistrarea cu un anumit ID prin metoda FIND. Daca o gasim o stergem cu REMOVE. Aruncam o exceptie daca nu i s-a schimbat starea in DELETED (=> NU s-a putut sterge)*

***Metoda folosind EntityState*** *- Cream o instanta a clasei avand Id = id cautat Modificam starea obiectului prin context.Entry(instanta).State = EntityState.Deleted; - Daca putem salva nu avem probleme de concurenta, altfel avem si tratam exceptia*

* ***SavingChanges***
  + Evenimentul se declanseaza dupa ce metoda SaveChanges() a fost apelata, dar inainte ca BD sa fie actualizata
    - ***CurrentValues***
    - ***OriginalValues***
    - ***Entity –*** entitatea reprez de ObjectStateEntry object
    - ***State***
    - ***AcceptChanges –*** accepta valorile curente
    - ***ChangeState***
    - ***GetModifiedProperties***
    - ***RejectPropertyChanges –*** respinge modificarile

## Incarcarea datelor

* ***Lazy*** – bazate pe setarile de context
  + Daca se face o cerere pentru o proprietate, EF creeaza un apel la BD, il executa si populeaza obiectul cu detaliile. O interogare – se aduc toate masinile, si apoi pentru fiecare masina o alta interogare cu care aduc comenzile
  + **Avantaj –** se aduc doar datele de care e nevoie
  + **Dezavantaj –** probleme de performanta in cazul in care nu este utilizat corespunzator
  + **Se poate dezactiva:** context.Configuration.LazyLoadingEnabled = false;
* ***Eager –*** cand e nevoie de incarcarea datelor relationate pt un obiect, interogari multiple – inefficient; un admin de bd-join sql
* ***Explicit*** – controlat de dezvoltator
  + ***Folosing Collection()***.***Load()***

## Migrarea

* Modelul se schimba frecvent si nu mai este sincronizat cu BD
* EF Migration – actualizeaza BD
* Metoda **Down** se apeleaza cand fac rollback la migrare: Remove-Migration ExtendedModel
* **Snapshot pentru model** – migrarea creaza un snapshot a schemei BD curente
  + Directorul Migrations/LibraryContextModelSnapshot.cs
  + Cand creem o noua migrare, EF determina ce s-a modificat comparand modelul curent cu modelul din snapshot

# LINQ – Language Integrated Query

## Motivatie LINQ – Structuri sintactice specifice

* Aplicatiile vor accesa date la un anumit moment in timpul executiei
* Datele pot fi regasite in fisiere XML, BDR, colectii in memorie, siruri
* LINQ ofera o **modalitate puternic tipizata** de a **accesa date in diverse formate** prin intermediul unui **layer de abstractizare**
* **LINQ = limbaj de interogare puternic tipizat incorporat in gramatica C#**
* Putem construi expresii asemanatoare cu interogarile SQL, dar acestea pot fi aplicate si BD non-relationale
* Interogarile LINQ – similare cu interogarile SQL, dar nu identice (seamana prin coincidenta)

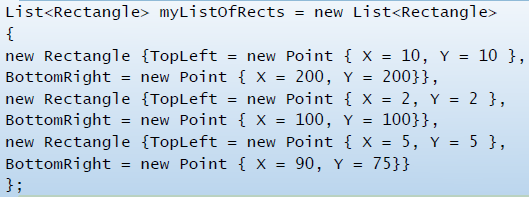
## Mecanisme LINQ

* Limbajul C# utilizeaza urmatoarele **mecanisme care se afla la baza LINQ**:
  + ***Variabile cu tip implicit***
  + ***Sintaxa de initializare a obiectelor/colectiilor***
  + ***Expresii Lambda***
  + ***Metoda extinse***
  + ***Tipuri anonime***

1. ***Variabile cu tip implicit***

* ***Cuvantul cheie “var” (***compilatorul va determina tipul de date corect bazandu-se pe atribuirea valorii***)***
* Acest feature e foarte util cand utilizam LINQ – multe *interogari 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

1. ***Sintaxa de initializare a obiectelor/colectiilor***

* Sintaxa de intializare colectie pentru a umple o lista List<T> cu obiecte Rectangle, fiecare avand doua obiecte Point care reprezinta o pozitie determinate de doua coordonate(x,y):
* ***Aceasta sintaxa combinata cu variabilele cu tip implicit ne permit declararea de tipuri anonime, utile la crearea de proiectii LINQ***

1. ***Expresii Lambda***

* ***“=>”*** *permite construirea unei expresii lambda*
* Expresiile lambda simplifica modul de lucru in .NET, reducand numarul de linii de cod care trebuie scrise
* ***( ArgumentsToProcess ) => { StatementsToProcessThen }***

1. ***Metode extinse***

* Permit adaugarea de functionalitati la clasele existente fara a folosi mostenirea, si permit adaugarea de noi functionalitati la clase sealed

1. ***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 folosinf sintaxa de initializare a obiectelor*
* ***LINQ*** foloseste tipuri anonime cand dorim sa proiectam noi feluri de date **“on the fly”**

## Termenii LINQ

* LINQ pot fi folosite in mai multe cazuri:
  + LINQ to Objects – siruri si colectii
  + LINQ to XML – documente XML
  + LINQ to DataSet – obiecte de tip ADO.NET DataSet
  + LINQ to Entities – ADO.NET Entity Framework
  + Parallel LINQ (PLINQ) – procesarea paralela a datelor returnate de o interogare LINQ

## Interogari

* Specifica info care trebuie incarcate din sursa de date
* Optional: cum trebuie sortate/grupate
* Stocate intr-o variabila de interogare si initializate cu o expresie de interogare
* ***Interogarile sunt separate de executia acestora***
* ***Template general:***
  + ***var result = from matchingItem in container select matchingItem;***
  + dupa from – item care corespunde criteriilor interogarii
  + dupa in – containerul de date in care se face cautarea
* **Corectitudinea sintactica** a unei interogari LINQ e validate la compilare => **ordinea operatorilor e importanta**
* Operatori ***from, in, select, where, orderby…***
* *!!!!! TIPURI DE INTEROGARI IN CURSUL 6!!!!!*

## Executia amanita

* **Interogarile 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

## 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>(), ToList<T>().** ---- *se adauga aceste metode la finalul interogarii LINQ* ----- ***.().ToArray<int>(); ….***
* Aceste metode vor face ca interogarea LINQ sa se execute in momentul apelarii metodei

## Interogari cu agregari

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

# Aplicatii WEB – ASP.NET Core

## ASP.NET Core

* Unificarea modului de dezvoltare a web app si web API
* Arhitectura care usureaza testarea
* App Windows, macOS, Linux
* Open-source si centrat pe comunitate
* Integrarea de framework-uri client-side – Blazor (UIs folosind C# si nu JS)
* Suporta instalarea de versiuni diferite ale .Net Core – side by side versioning

## JavaScript si TypeScript

* Putem modifica dinamic elemente client-side
* ECMAScript – standard care defineste functionalitatile curente si viitoare pt JS
* Implementarea Microsoft pt JS e Jscript
* TypeScript
  + Similar cu JS
  + Sintaxa se bazeaza pe JS, dar aduce noi functionalitati
  + Exista multe similaritati intre C# si TypeScript

## Librarii de scripting

* jQuery – gestioneaza unitary modul in care diferite browsere gestioneaza evenimentele
* Angular – librarie de la Google bazata pe pattern-ul MVC; pagini single-page
* React – librarie de la FB, ofera functionalitati prin care se pot actualize facil interfetele utilizator pe masura ce datele se modifica in background

## Structura unei app Web – cu Razor

* in dir wwwroot – continut de tip client-side CSS, JS, imagini, continut non-programatic
* dir Pages – pagini Razor si fisiere suport (incep cu “\_”);
* pereche de:
  + fisier .cshtml – markup HTML & C# utilizand sintaxa Razor
  + fisier .cshtml.cs – cod C# pt gestiune evenimente la nivel de pagina
* appSettings.json – date de configurare, string-uri de conexiune
* Program.cs – punctul de start al app; la pornire se creeaza un obiect “host”, incarca configurarile din appsettings.json
* Startup.cs – cod care configureaza comportamentul app
  + Metoda ***ConfigureServices*** – se configureaza, in containerul ***dependency injection*** ***servicii*** necesare app
  + Metoda ***Configure*** – definire mod de gestionare a cererilor ca o serie de ***componente middleware***
* ***Serviciu = componenta reutilizabila, furnizeaza functionalitate app (consumate in app prin Dependency Injection)***

## Dependency Injection

* + ***Design pattern*** *care ofera o* ***cuplare slaba***
  + *Un serviciu e definit doar cu o* ***interfata, tipul concret care implementeaza interfata e injectat***
  + ***Injectarea*** *se face cu* ***constructori cu argumente de tip interfata***

## Middleware

* Componenta soft, integrata in pipeline-ul app
* Gestioneaza cereri
* Fiecare componenta decide daca trebuie sa trimita mai departe cererea la urmatoarea componenta din pipeline
* Fiecare componenta din pipeline e responsabila pentru a invoca urmatoarea componenta sau pt a scurt-circuita pipeline-ul
* Componenta aferenta gestiunii exceptiilor trebuie sa fie apelata devreme in pipeline

## Creare de app web cu Razor

* Razor e o sintaxa de tip markup pentru a incorpora ***cod de tip server-side*** in pagini web
* Consta in Razor ***markup, C# si HTML***
* Limbajul Razor ***implicit e HTML***
* Paginile Razor sunt derivate din ***PageModel***
* ***Cand simbolul @ e urmat de un cuvant cheie Razor, trecerea e facuta la mark-up specific Razor, altfel la C#***
* ***Blocurile de cod Razor incep cu @ si sunt marcate cu {} - bloc de cod C#***
* ***Se pot declara functii locale cu markup pt a fi utilizate ca metode template in blocuri de cod***
* Razor evalueaza expresiile C# si le interpreteaza in output HTML
* ***@\*Comentariu\*@*** in Razor
* ***Sintaxa Razor:***
  + ***Expresii implicite nu pot contine spatii:*** *@DateTime.Now*
  + ***Expresii explicite ():*** *@(DateTime.Now – TimeSpan.FromDays(7))*
    - *Pot fi utilizate pt a concatena text cu un rezultat al unei expresi*
  + ***Tranzitie implicita***
    - *Limbajul implicit intr-un bloc de cod e C#, dar poate face implicit tranzitia la HTML: @*
  + ***Tranzitie explicita – tag-ul <text> sau “ @:” pt inline***

### Directive Razor

* ***@page*** – permite ca pagina sa poate ***gestiona cereri*** (prima directiva din fisier)
* ***@model*** – specifica ***modelul transmis*** catre pagina Razor (repr de clasa derivate din PageModel)

### Pagina Layout

* Furnizeaza un ***Layout unitar*** pt intreaga app
* Permite containerului ***HTML pt layout***
  + Sa fie ***specificat intr-un singur loc***
  + Sa fie utilizat in ***pagini multiple***
* ***RenderBody()*** – ***placeholder*** care permite ***afisarea continutului specific*** pt ***fiecare pagina***
* *Clasa* ***PageModel*** *contine o* ***proprietate de tip dictionar****, utilizata pentru a* ***trimite date la un View*** *(folosind* ***perechi cheie/valoare****) –* ***ViewData***
* **Setare Layout**
  + In Pages/\_ViewStart.cshtml: @{ Layout=”\_Layout”; }

### Tag Helpers

* Permit codului de tip server-side sa participe la crearea si afisarea elementelor HTML in fisiere Razor
* Folosesc limbajul C# si targeteaza el HTML, bazandu-se pe ***numele elementului si numele atributului***
* ***Prefixati cu “asp-”***
* ***Tipizare puternica cu proprietatea modelului***
* ***Ofera:***
  + ***Experienta de dezvoltare asemanatoare HTML***
  + ***Suport IntelliSense pt crearea de markup HTML si Razor***
  + ***Cod mai robust & facil, mentenabil utilizand doar info de la server***
* ***Image Tag Helper***
  + Adauga versiune la imagine
  + Image Tag Helper adauga tag-ului <img> elemente privind comportamentul cache pt fisiere statice de tip imagine
  + Un string cu o caloare unica de hash e 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 e generat care include string-ul actualizat ***(<img … asp-append-version=”true”>***)
* ***Directiva @addTagHelper***
  + Se adauga in *\_ViewImports.cshtml*
  + Efect: Tag Helpers ***devin disponibili*** in cadrul paginilor
  + Microsoft.AspNetCore.Mvc.TagHelpers e assembly-ul pt tag helpers predefiniti
  + Se pot crea tag helpers ***custom***
* ***Anchor Tag Helper***
  + *Adauga* ***noi atribute*** *tag-ului standard HTML (<a asp-page=”./Edit” asp-route-id=”10”> Edit </a>)*
* ***Tag Helper pentru validare***
  + ***Validation Message Tag Helper***
    - *Mesaj de validare pt proprietatea aferenta din modelul nostrum*
    - *Utilizat cu asp-validation-for a unui el <span>*
    - *In general dupa un <inpu>*
  + ***Validation Summary Tag Helper***
    - *Afiseaza un rezumat al erorilor de validare (pe baza constrangerilor/atributelor adnotarilor in clasele din model)*
    - *Asp-validation-for = “”*
  + ***Tag Helpers Custom***
    - ***Conventie de nume*** *referitor la elementul pe care il targeteaza: el targetat e radacina EmailTagHelper – email -> <email>*
    - *Clasa EmailTagHelper* ***mosteneste TagHelper***
    - ***Metoda suprascrisa Process*** *controleaza ceea ce face tag helper-ul cand este executat*
    - ***SetAttribute, SetContent***

### PageModel

* Prin conventie numele clasei e denumit dupa ***sablonul <NumePagina>Model***
* *PageModel* permite *separarea logicii paginii fata de prezentarea acesteia*
* ***MetodeHandler:***
  + ***OnPost –*** ruleaza cand exista cereri POST
  + ***OnGet –*** initializeaza starea paginii
  + ***Se pot adauga pt orice verb HTTP***
  + ***Sufixul “Async” e optional, dar folosit pt functionare asincrona***

### Binding

* [BindProperty]
* Legarea proprietarilor nu mai e necesara conversia datelor HTTP catre tipul modelului

*Exemplu binding cu prop. Title:*

*<form method=”post”>*

*<input asp-for=”Book.Title” />*

*</form>*

### Antiforgery Token

*<input name=”\_RequestVerificationToken” type=”hidden” value=”<Antiforgery Token>” />*

* FormTagHelper injecteaza token-uri antiforgery in el HTML form