Curs 8

Accesul la date in aplicatiile .NET MAUI

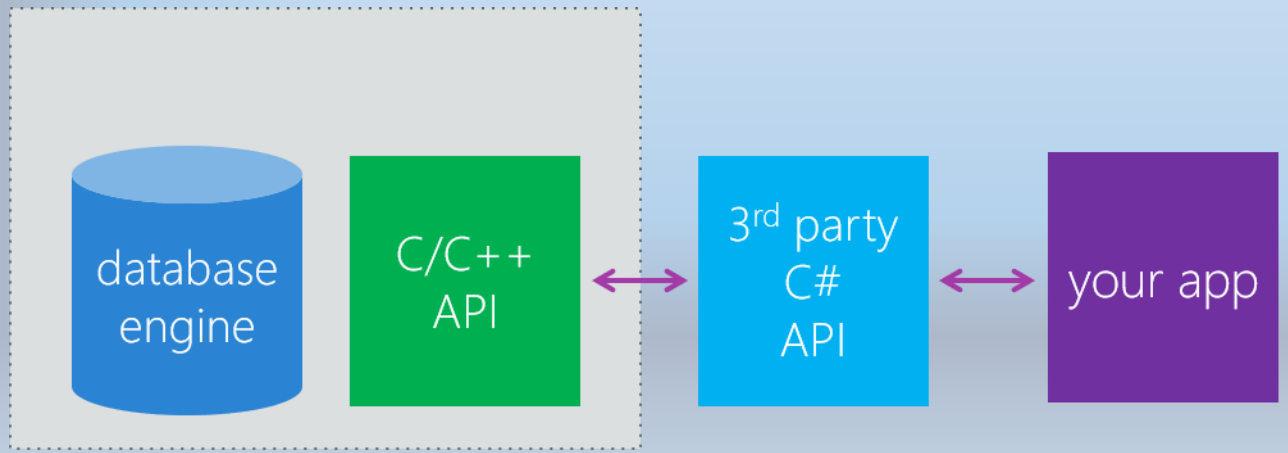
Accesul la date

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 utilizam modelele care sunt definite în proiect pentru a servi drept schema a bd
- SQLite-net este un pachet NuGet – pentru a-l utiliza instalam 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

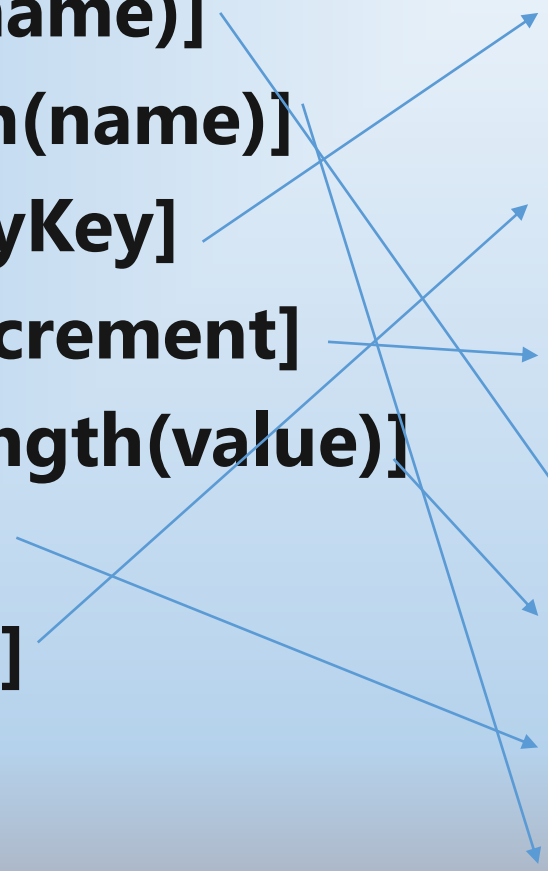
Creare Tabel

- Constructorul primeste ca si parametru calea spre fisierul de tip baza de date

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

Atribute SQLite-net

1. **[Table(name)]**
2. **[Column(name)]**
3. **[PrimaryKey]**
4. **[AutoIncrement]**
5. **[MaxLength(value)]**
6. **[Ignore]**
7. **[Unique]**



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)

B. Specifica ca valorile stocate in acea coloana trebuie sa aiba valori unice (pot exista mai multe coloane setate unique)

C. Specifica ca valoarea acelei coloane se va autoincrementa la inserarea unui nou rand in tabel.

D. Specificam numele tabelului care va fi creat (daca dorim sa fie diferit de numele clasei)

E. Specifica numarul maxim de caractere care pot fi salvate in coloana respectiva.

F. Proprietatea nu va fi mapata cu o coloana din table

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 utilizam un obiect SQLiteAsyncConnection
- Această clasă este definită în namespace-ul SQLite
- Când instanțiam un obiect SQLiteConnection, specificam 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 asyncron

- Dacă executam interogări într-o bază de date într-un mod sincron=>probleme de performanță și aplicații care nu răspund.

- In mod sincron operatiile se executa pe Thread-ul UI – UI freeze sau operatiile dureaza 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)*

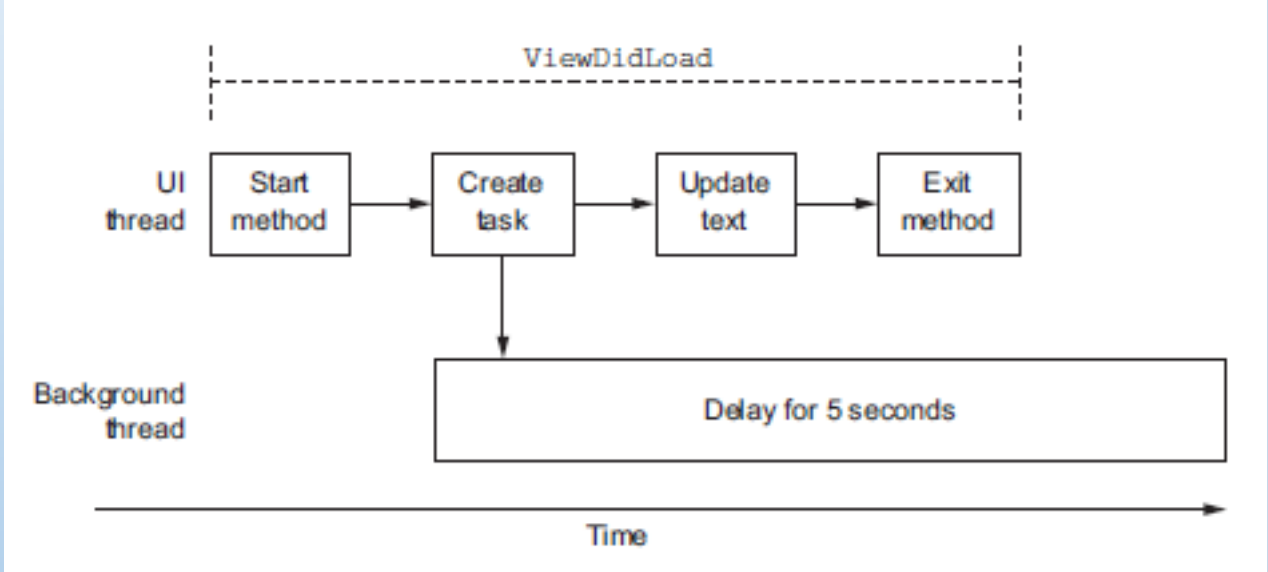
```
async void OnShopListAddedClicked(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new ListPage
        {
            BindingContext = new ShopList()
        });
    }
```

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";
}
```



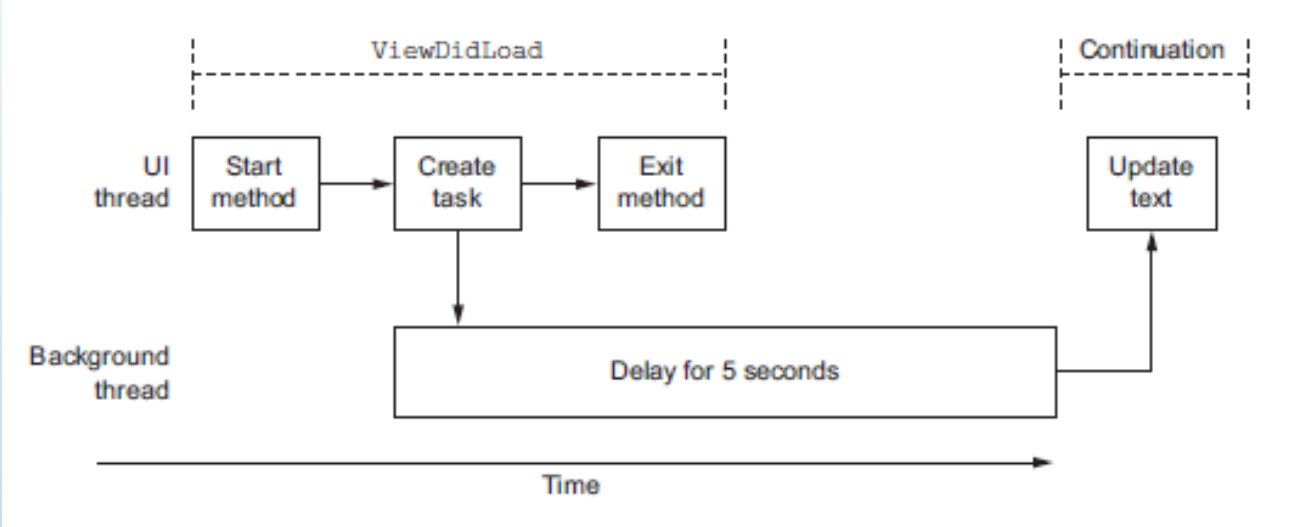
Task-ul Delay task este creat ,

este pornit intr-un thread de background, UI este actualizat

Adaugam **async** si **await**

```
public override async void
ViewDidLoad()
{
...
await
Task.Delay(TimeSpan.FromSecon
ds(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

UNIVERSITATEA BABEȘ-BOLYAI

Data Binding - context

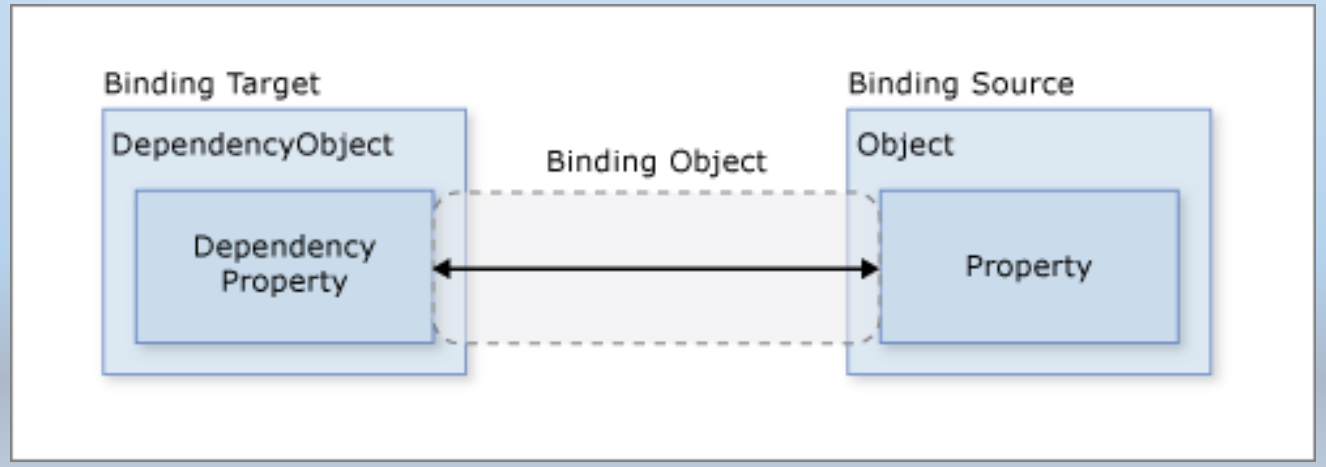
- O aplicație .NET MAUI constă dintr-una sau mai multe pagini, fiecare dintre acestea conține de obicei mai multe obiecte de UI
- Una dintre sarcinile principale ale aplicației este să mențină aceste obiecte UI sincronizate cu sursa de date și să țină evidența diferitelor valori sau selecții pe care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

Data Binding - scop

- Ofera aplicatiilor o modalitate consistenta de a afisa date si de a interactiona cu date diverse
- DataBinding- procesul care stabileste conexiunea dintre UI si datele pe care le afiseaza aceasta
  - cand datele din sursa de date isi modifica valoarea elementele care sunt legate la acele date se modifica automat
  - cand se modifica datele in elementele de interfata, sursa de date se poate modifica automat pentru a reflecta modificarile

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) referentiat de databinding.



Operatii data-binding

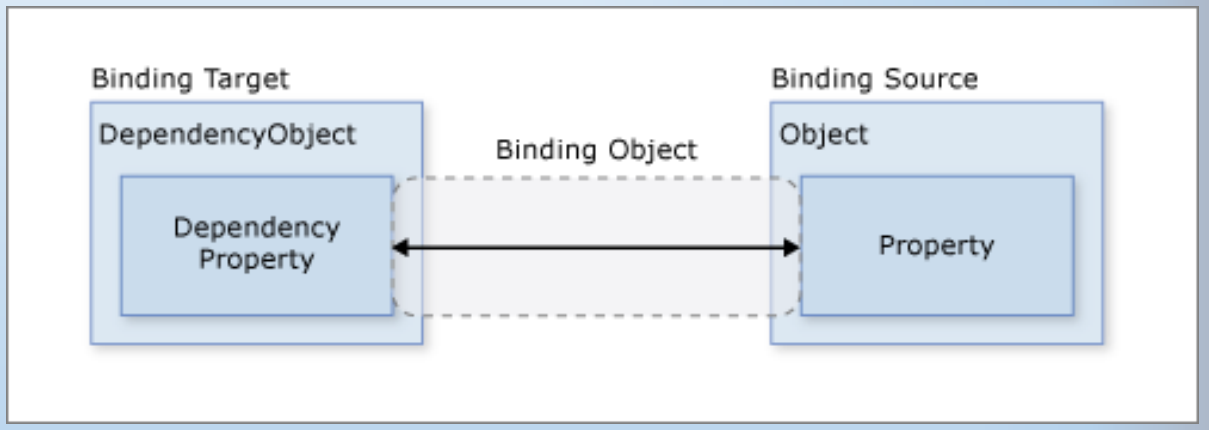
```
<Editor Placeholder="Enter shop name" Margin="20"
                Text="{Binding ShopName}"
                HeightRequest="50" />
```

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>
```