Platforma .NET

8. seminář

Radek Janoštík

Univerzita Palackého v Olomouci

13. 11. 2025

Reakce na úkoly

- Nějaké doplňující dotazy k zadání?
- Jak s konzultacemi?

Vydání .NET 10

- V úterý 11. 11. byla vydána nová verze – .NET 10
- `https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-10/overview`
- Runtime – spousta změn v kompilaci (JIT, AoT, instrukce AVX10.2, . . .)
- Knihovny – nová kryptografická API (postkvantová kryptografie), JSON serializace, `System.Numerics`
- SDK – `Microsoft.Testing.Platform`, jednorázové spuštění *tool*, pořadí příkazů, . . .
- C# 14 – `https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-14`
- F# 10 – `https://learn.microsoft.com/en-us/dotnet/fsharp/whats-new/fsharp-10`

Object Relation Mapping (ORM)

- Automatické mapovaní objektů do relační databáze
- Náš pohled – transparentní práce s databází pomocí objektů
- Odstínění od konkrétní implementace databáze (MSSQL, MySQL, PostgreSQL, . . .)
- Dva přístupy
  - `Code First` – Databáze se generuje z programu
  - `Database First` – Programový kód se generuje dle struktury DB
- Zde ukážeme pouze `Code First`
- Zmatečnost názvů (neintuitivní skutečný význam)

Entity Framework

- První verze (2008) součástí .NET Frameworku, nad LINQ to SQL
- Od verze 6 (2013) Opensource, mimo .NET Framework
- `https://docs.microsoft.com/en-us/ef/`
- Dvě verze:
  - EF6 - Frameworková `https://github.com/dotnet/ef6`
  - EF Core – pro .NET Core `https://github.com/dotnet/efcore`
- `CodeFirst` i `DatabaseFirst`
- Líné vyhodnocování
- Podpora LINQ
- Podpora MSSQL, Oracle, MySQL, ... (`https://docs.microsoft.com/en-us/ef/core/providers/?tabs=dotnet-core-cli`

EF – instalace – NuGet Package Manager

- Ruční linkování knihoven pracné (verze, závislosti, hledání knihoven v různých zdrojích)
- `NuGet Package Manager` – správce balíčků s repozitáři (podobně jako v Linuxu)
- Pravý klik na projekt → Manage NuGet Packages
- Správa verzí, licencí, závislostí, možno více zdrojů, vytvářet vlastní balíčky
- Balíčky „per projekt“ nebo „per solution“
- Potřebné balíčky:
  - Microsoft.EntityFrameworkCore
  - Microsoft.EntityFrameworkCore.Design
  - Microsoft.EntityFrameworkCore.Proxies
  - Microsoft.EntityFrameworkCore.Sqlite

EF – vytvoření modelu

- = definice tříd, které chceme uchovávat v databázi
- Pozor na `CodeFirst` konvence `https://docs.microsoft.com/en-us/ef/core/modeling/relationships` např.
  - Property obsahující „id“ bude primární klíč
  - Odkaz na jinou entitu pomocí `virtual` + typ + název
- Např.:

```
public class Student {
   public int Id {get; set;}
   public string Name {get; set;}
   public int AddressId {get; set;}
   public virtual Address Address {get; set}
}
```

EF – vytvoření kontextu

- Objekt reprezentující připojení k databázi

```
using Microsoft.EntityFrameworkCore
```

- Předek: `DbContext`, pro každou entitu `DbSet`
- Např.:

```
public class UniversityContext : DbContext
{
    public UniversityContext()
    { }
    public DbSet<Address> Addresses { get; set; }
    public DbSet<Student> Students { get; set; }
    protected override void OnConfiguring(DbContextOptionsBuilder options) {
            options.UseLazyLoadingProxies()
                .UseSqlite("Data Source=/cesta/k/databazi/testDb.db");
   }
}
```

- Lze dodefinovat názvy sloupců pomocí atributů:

```
[Column("jmeno_studenta")]
public string Name { get; set; }
```

- ...

Vytvoření databáze

- Můžeme potřebovat EF nástroje pro `dotnet`

  ```
  dotnet tool install --global dotnet-ef
  ```

- Případně dodat do `PATH`

  ```
  export PATH="$PATH:/home/radek/.dotnet/tools"
  ```

- Přidání úvodní migrace (viz dále)

  ```
  dotnet ef migrations add InitialCreate
  ```

- Vytvoření databáze

  ```
  dotnet ef database update
  ```

Přístup k datům – vložení záznamu

- Vytvoření záznamu

```
try {
   using (UniversityContext ctx = new UniversityContext()) {
       Address a = new Address() { Street = "17. Listopadu",
                                   Number = 14 };
       Student s = new Student() { Address = a, Name = "Karel  Vomáčka" };
       ctx.Students.Add(s);
       ctx.SaveChanges();
   }
} catch (Exception e){
   Console.WriteLine(e);
}
```

- Vždy potřeba uložit změny `ctx.SaveChanges()`
- Automatické vložení/uložení všech dotyčných entit

Přístup k datům – čtení dat

- LINQ dotaz na `DbSet`

```
try {
   using (UniversityContext ctx = new UniversityContext()) {
      foreach (Student st in ctx.Students.Where(p =>p.Address.Street == "17. listopadu"
         Console.WriteLine($"{st.Name}, {st.Id}");
      }
   }
} catch (Exception e) {
   Console.WriteLine(e);
}
```

- Zobrazení vygenerovaných dotazů – úprava options kontextu a doinstalování `Microsoft.Extensions.Logging.Console`:

```
options.UseLoggerFactory(LoggerFactory.Create(b =>
                                    { b.AddConsole(); }));
```

Změna modelu – migrace (1/2)

- Při změně modelu(definice tříd) musíme „protlačit“ změnu i do databáze
- Systém verzování databáze a *migrací*
- Při automatickém založení databáze nám vznikla i tabulka `_EFMigrationHistory`, která zaznamenává změny v DB.
- Zjistili jsme, že jsme zapomněli na město

```
public class Address {
   public int Id { get; set; }
   public string Street { get; set; }
   public int Number { get; set; }
   public string City { get; set; }
}
```

Změna modelu – migrace (2/2)

- Změnu musíme dostat do databáze
- Zadáme příkaz: `dotnet ef migrations add City` (City je název dané migrace)
- a aktualizujeme databázi: `dotnet ef database update`
- Databáze se aktualizovala, dodal se sloupec pro město

Vazba 1:N

- K adrese pouze doplníme:
  `public virtual ICollection<Student> Students { get; set; }`
- Pokud jsou vazby zřejmé, máme automaticky k dispozici seznam studentů na dané adrese
- Jinak potřeba dodat atribut `[InverseProperty("Address")]` se specifikací inverzní vlastnosti
- Případně pomocí FluentAPI
  `https://docs.microsoft.com/en-us/ef/core/modeling/` v `DbContext`

```
protected override void OnModelCreating(ModelBuilder modelBuilder) {
        modelBuilder.Entity<Student>()
            .HasOne(s => s.Address)
            .WithMany(a => a.Students);
    }
```

- Priority: Konvence, atributy, FluentAPI

Vazba M:N

- Vazbu bychom mohli namodelovat pomocí vazební tabulky

```
public class StudentSubject {
    public int StudentId { get; set; }
    public virtual Student Student { get; set; }
    public int SubjectId { get; set; }
    public virtual Subject Subject { get; set; }
}
```

- A dodefinovat navigační property:

```
protected override void OnModelCreating(ModelBuilder modelBuilder) {
modelBuilder.Entity<StudentSubject>()
                .HasKey(s => new {s.StudentId, s.SubjectId});
    modelBuilder.Entity<StudentSubject>().HasOne(ss => ss.Student)
                .WithMany(s => s.Subjects)
                .HasForeignKey(ss => ss.StudentId);
    modelBuilder.Entity<StudentSubject>().HasOne(ss => ss.Subject)
                .WithMany(s => s.Students)
                .HasForeignKey(ss => ss.SubjectId);
}
```

Vazba M:N

- Při vhodném pojmenování EF automaticky vytvoří vazební tabulku
- V kódu nám nevznikají nové třídy

```
public class Subject
{
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual List<Student> Students { get; set; } = [],
}
```

- Obě kolekce musí být `virtual`
  - Proč?

Vazby

- Problematika vazeb mezi objekty je složitější
- Někdy nám to podchytí dobré pojmenování
- Jindy je potřeba ručně dodefinovat (atributy, FluentAPI)
- Více k vazbám: `https://docs.microsoft.com/en-us/ef/core/modeling/relationships`

Editace dat

- Stačí změnit property u entity a uložit
- Pozor abychom pracovali s DbEntitou a ne „obyčejným objektem“

```
try {
using (UniversityContext ctx = new UniversityContext()) {
   // Vytvoříme nový předmět (automaticky se vloží)
   Subject sub = new Subject() {
      Credits = 6,
      Name = "Platforma .NET"
   };
   // Najdeme DbEntitu studenta
   Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);
   if (s != null) { // Tu upravíme
      s.Name = "Dave Lister";
      s.Subjects.Add(new StudentSubject() {Student = s, Subject = sub});
      ctx.SaveChanges();
   }
}
} catch (Exception e) {
   Console.WriteLine(e);
}
```

Mazání dat

- Nejdříve najdeme entitu

```
Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);
```

- Poté ji odstraníme

```
ctx.Students.Remove(s);
ctx.SaveChanges();
```

- Můžeme také jen změnit její stav:

```
Student s = ctx.Students.FirstOrDefault(p => p.Id == 1);
ctx.Entry(s).State = EntityState.Deleted;
ctx.SaveChanges();
```

- Smaže jen tuto entitu, ty které obsahuje nemusí
- Jde nastavit smazání všech souvisejících, mohou být ale navázány jinam – více podrobností:
  `https://docs.microsoft.com/en-us/ef/core/saving/cascade-delete`

EF – závěrem

- Entity Framework Core je velmi rozsáhlé téma
- Ukázány pouze základy, důležité je samostudium
- Dát pozor, zda je dokumentace pro EFCore nebo pro EF6
  - Neustále se vyvíjí (odlišnosti ve verzích)
  - Modulárnost (ne všechny návody uvádí `using`)
- Některé databázové connectory nemusí podporovat vše
- Dobrý pomocník pro programátora, ale má i svá úskalí

Úkol

- Pomocí Entity Framework vytvořte model pro „malý Instagram“ – následující entity
  - User (Id, Firstname, LastName, Age, Following, Followed)
  - Post (Id, Text, ImgPath, Tags, Likes, ReleaseDate)
  - Comment (Id, Author, Likes, Text)
  - Message (pro soukromé zprávy mezi uživateli)
  - Vhodně navrhněte vazby
- Model by měl umožňovat, komentování postů, lajkování (kdo lajkoval co), chat mezi dvěma uživateli (soukromé zprávy)
- Vytvořte nějaká data a poté je vypište
  - Všechny příspěvky nějakého uživatele, které jsou starší než týden a v textu neobsahujou slovo “já” ve všech možných podobách (stačí třeba velká i malá písmena)
  - Všechny příspěvky, které určitý uživatel lajknul
  - Všechny příspěvky, které lajknuli dva různí uživatelé
  - Všechny příspěvky všech uživatelů, které:
    - určitý uživatel sleduje
    - jsou staré maximálně 3 dny
    - seřazené od nejnovějšího po nejstarší