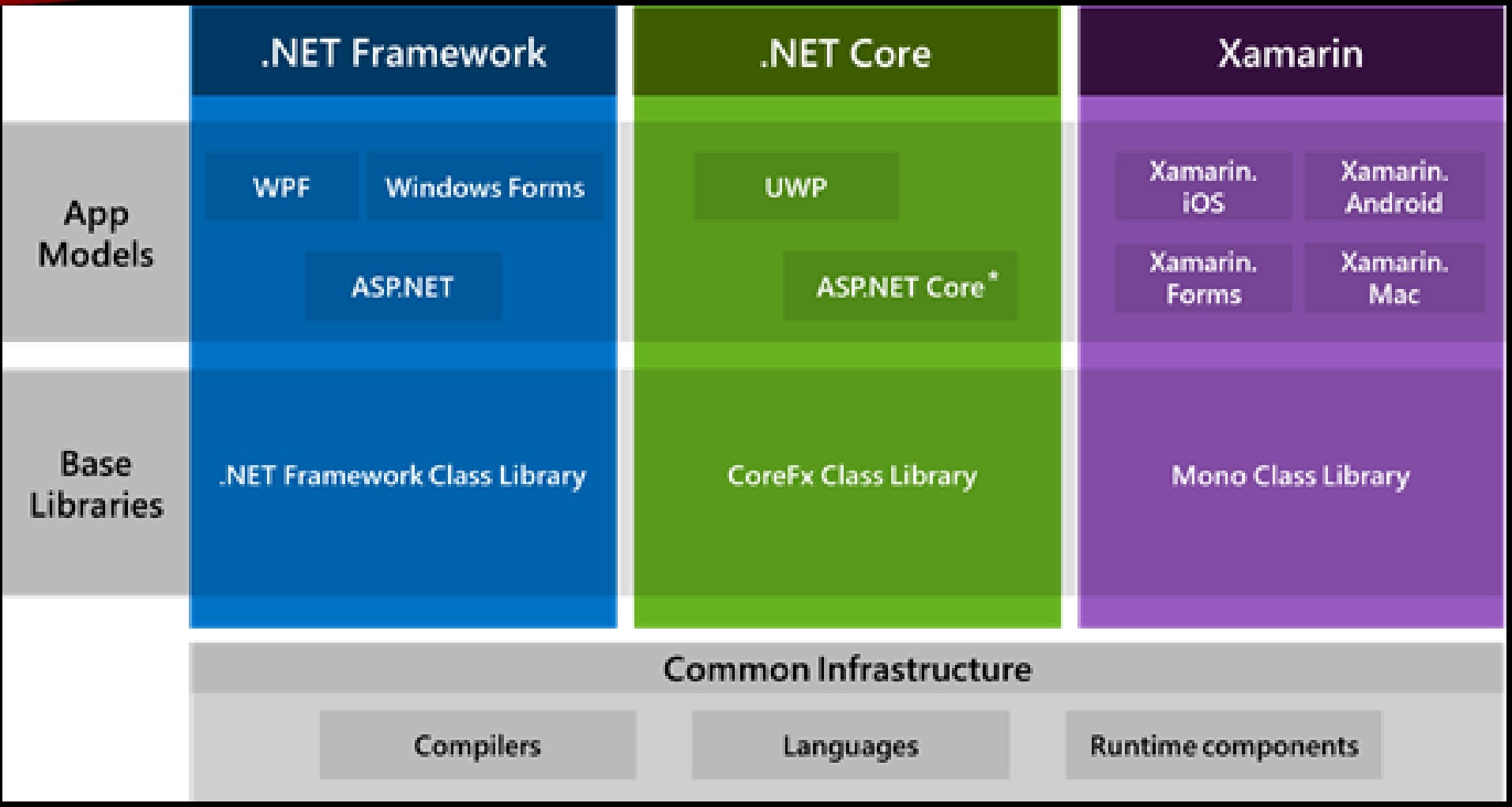
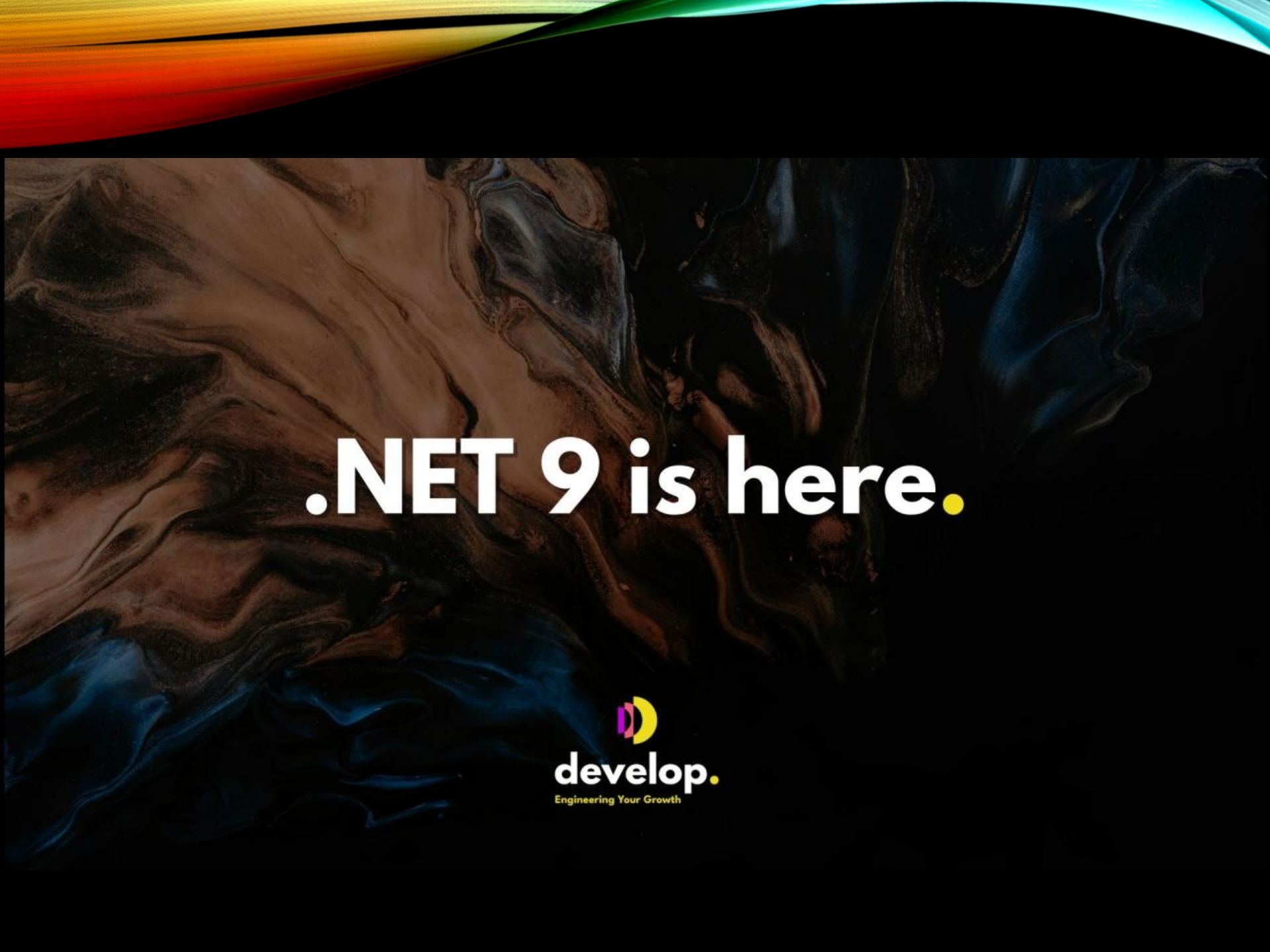
УЕБ ПРИЛОЖЕНИЯ
С ENTITY FRAMEWORK CORE

Гл. ас. д-р Елица Ибрямова

EIbryamova@ecs.uni-ruse.bg

ENTITY FRAMEWORK CORE


.NET Framework
.NET Core
Xamarin
App Models
WPF
Windows Forms
ASP.NET
UWP
ASP.NET Core*
Xamarin. iOS
Xamarin. Android
Xamarin. Forms
Xamarin. Mac
Base Libraries
.NET Framework Class Library
CoreFx Class Library
Mono Class Library
Common Infrastructure
Compilers
Languages
Runtime components


.NET 9 is here.
develop.
Engineering Your Growth

.NET Aspire 9.0

Azure Functions preview support

More secure defaults

"WaitFor" resources to spin up

Start/Stop Resources

Improved Azure Configuration

Simplified acquisition

Improved Azure Container Apps integration

Persistent Containers

Support for .NET 8 & .NET 9

OpenAI Integration

AWS Stable Integrations

Visual Studio & C# Dev Kit enhancements

.NET Aspire Community Toolkit

Build anything with a unified platform

.NET

Cloud | Web | Desktop | Mobile | Gaming | IoT | AI

Tools


Visual Studio


Visual Studio Code


CLI


GitHub Copilot

+

Operating systems


Windows


Linux

macOS

+

Ecosystem

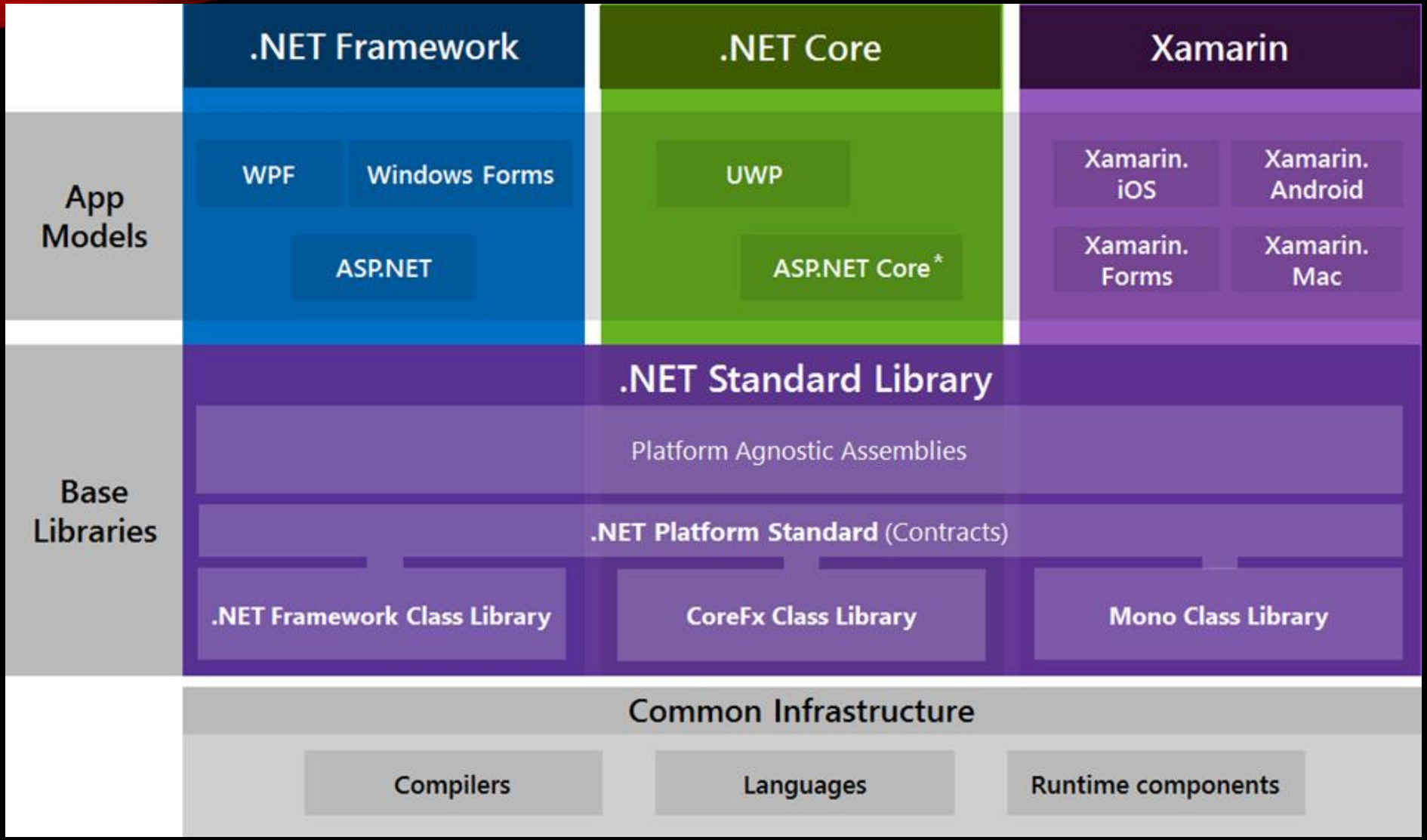

NuGet


GitHub

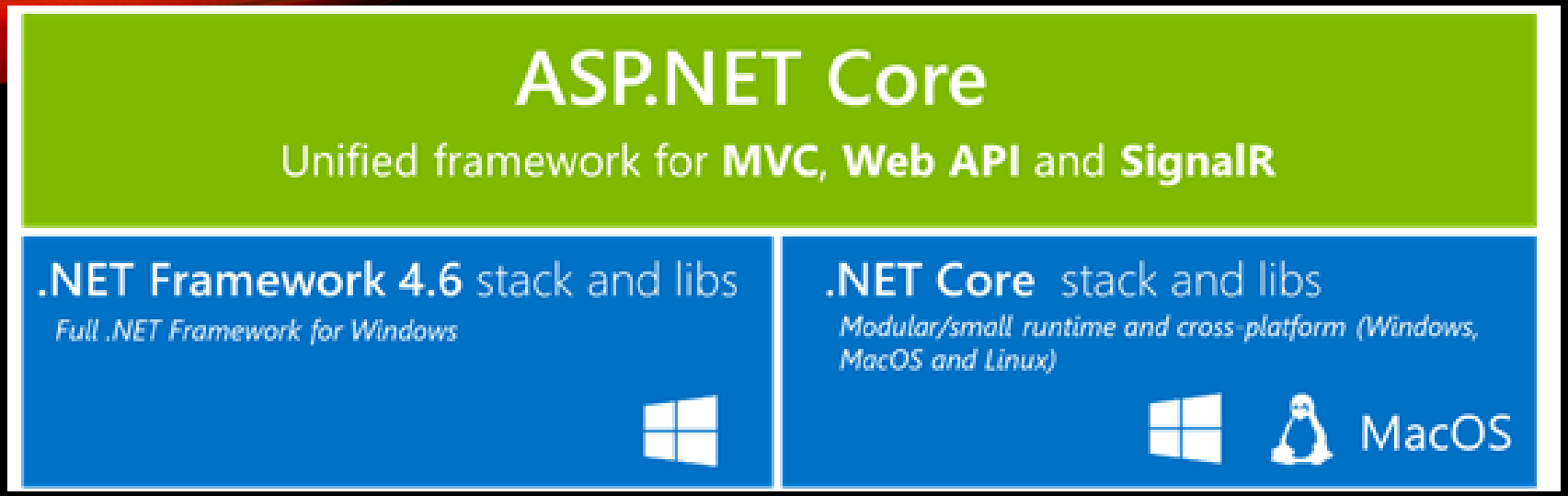

.NET Aspire


Components, tools, library vendors

.NET – A unified development platform


.NET Framework
.NET Core
Xamarin
App Models
WPF
Windows Forms
ASP.NET
UWP
ASP.NET Core*
Xamarin. iOS
Xamarin. Android
Xamarin. Forms
Xamarin. Mac
Base Libraries
.NET Standard Library
Platform Agnostic Assemblies
.NET Platform Standard (Contracts)
.NET Framework Class Library
CoreFx Class Library
Mono Class Library
Common Infrastructure
Compilers
Languages
Runtime components


ASP.NET
VS
.NET
Core



ASP.NET Core in .NET 9

Quality & fundamentals

Developer experience

Cloud native

ASP.NET Core in .NET 9

Optimized static web asset handling

Improvements to exception handling and debugging

Authentication enhancements

New Blazor Hybrid Templates

Improved Kestrel connection metrics

SignalR improved distributed tracing

Dictionary debugging improvements

Built-in OpenAPI support

Detect Blazor component render mode

Improvements to DataProtection

SignalR AOT support

Blazor reconnection improvements

Trust Developer certs on Linux

Up to **25%** faster Blazor startup

Keyed service support in middleware

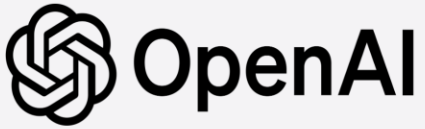
.NET AI Ecosystem

Semantic Kernel

Azure OpenAI

Azure AI Search

Azure Inference



OpenAI library for .NET

LlamaParse C# Client

GitHub Models

Qdrant Solutions

C# Client

C# Client

Smart Components

Smart Components

Smart Components

Entity Framework

Core

ENTITY FRAMEWORK CORE

- Стандартната рамка за **ORM (Object Relational Mapping)** за .NET и .NET Core;
  - Предоставя **LINQ (Language-Integrated Query)-**базирани заявки за данни и **CRUD (Create, read, update and delete)** операции;
- Предоставя **автоматично проследяване** на промяната на обекти в паметта;
- Работи с релационни бази от данни (с различни системи за управление на бази от данни);
  - Отворен код с независим цикъл на пускане на версии;
- **Entity Framework Core**
  - https://introprogramming.info/intro-csharp-book/read-online/glava22-lambda-izrazi-i-linq-zayavki/#_Toc298864633

ОСНОВЕН РАБОТЕН ПРОЦЕС

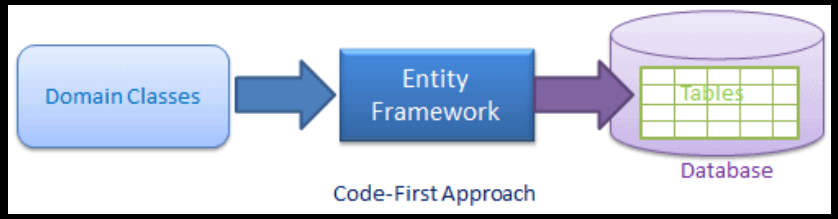
- Определя модела на данни (Code First / Scaffold from DB)
- Изписва и изпълнява заявки върху IQueryable
- EF генерира и изпълнява SQL заявка в БД
- EF преобразува резултатите от заявката в .NET обекти
- Промяна на данните със C# -извиква се "**SaveChanges()**“
- EF генерира и изпълнява SQL команда за промяна на БД

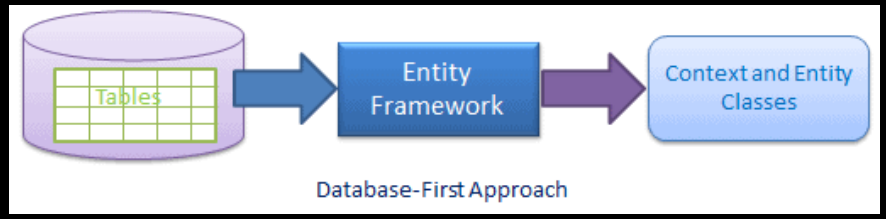
ENTITY FRAMEWORK CORE: КОНФИГУРАЦИЯ

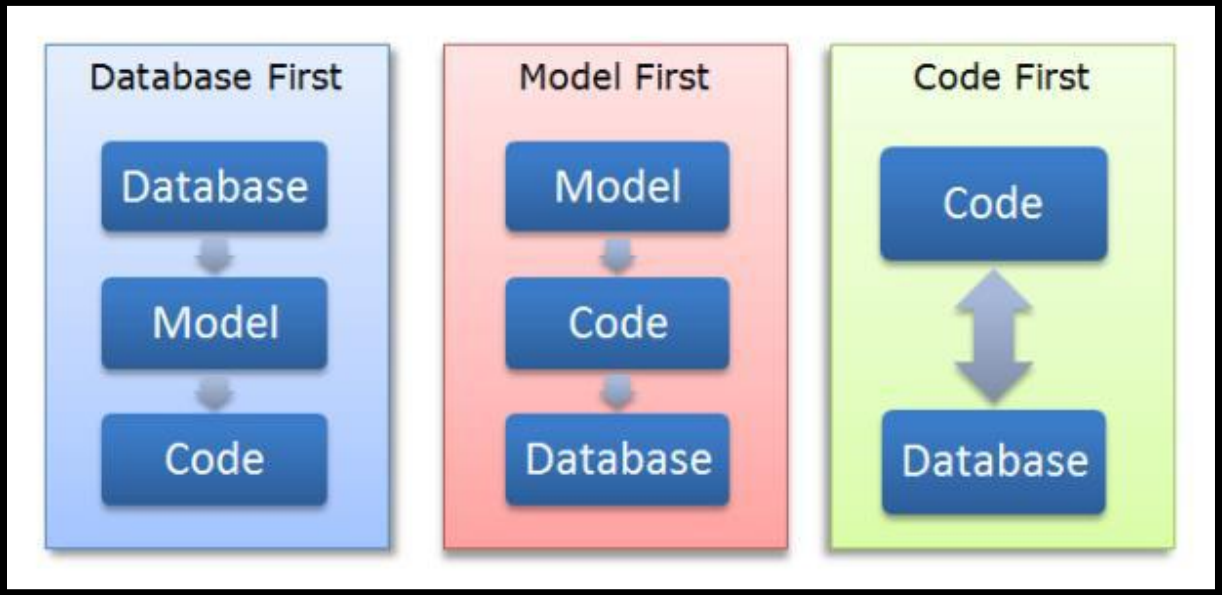
- За да добавите поддръжка на EF Core към проект във Visual Studio:
  - Инсталирайте го от **Package Manager Console**
  - EF Core е модулен -различни допълнителни пакети могат да бъдат инсталирани:

    - **Install-Package Microsoft.EntityFrameworkCore**
    - **Install-Package Microsoft.EntityFrameworkCore.SqlServer**

МЕТОД CODE-FIRST И МЕТОД DATABASE FIRST

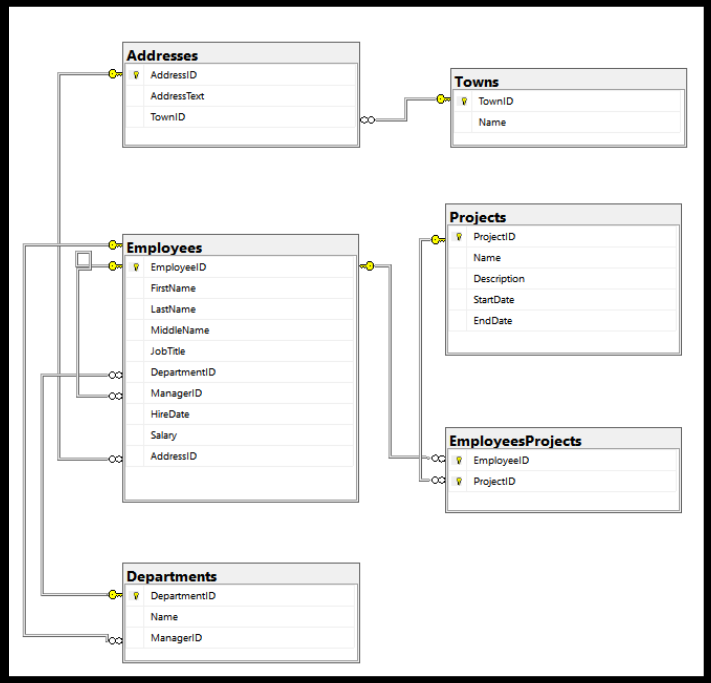


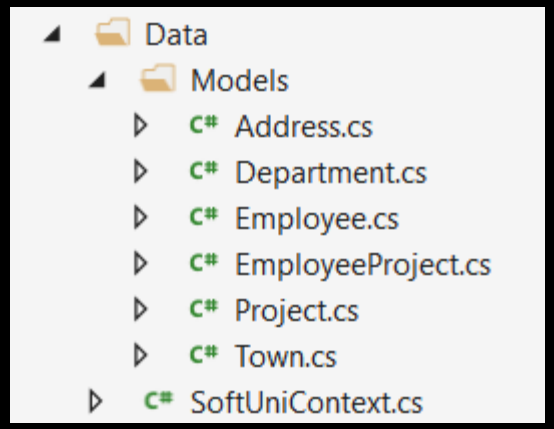




МЕТОДЪТ DATABASE FIRST

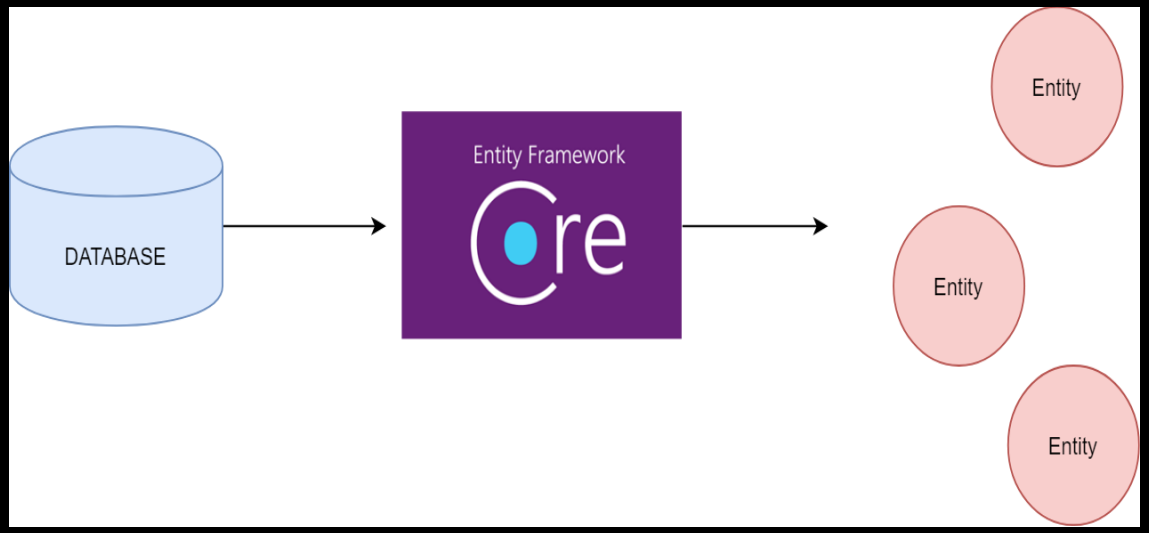
- Методът "Database First" моделира класовете с обекта след като базата данни е създадена:





МЕТОДЪТ “DATABASE FIRST” SCAFFOLDING DB

- Scaffolding (транслиране) DbContext е процес на изграждане на модел на данни, така че Entity Framework Core да може да използва съществуваща база от данни.
- Scaffolding проверява съществуваща база от данни и автоматично създава модел на данни.



ИЗПОЛЗВАНЕ НА МЕТОДА "DATABASE FIRST"

- Scaffolding DbContext от DB с командата Scaffold-DbContext в конзолата за управление на пакети:

```
Scaffold-DbContext
-Connection "Server=.;Database=…;Integrated Security=True"
-Provider Microsoft.EntityFrameworkCore.SqlServer
-OutputDir Data
```

- Scaffolding предварително изисква следните пакети:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
Install-Package Microsoft.EntityFrameworkCore.SqlServer.Design
```


ЖИВОТЪТ Е ТВЪРДЕ КРАТЪК! БЪДИ В ГЛАВНАТА РОЛЯ!
СВОБОДЕН ИГРАЧ
ДЕКЕМВРИ В КИНАТА

МЕТОДЪТ “CODE FIRST“

- Създава съответните таблици в БД, следвайки тяхно описание чрез класове. Как да го използвате?
  - Създават се класове, описващи таблиците в базата от данни, като чрез специален синтаксис се указват връзките между тях;
  - Създава се клас, който задължително наследява DbContext;
  - В него се добавят като DbSet <T> класовете, описващи таблиците;
  - Пренаписва се метода OnConfiguring(), за да се укаже, по какъв начин да се свърже с БД.

МЕТОДЪТ “CODE FIRST“

- Накрая използвайки Package Manager конзолата се добавя поддръжка на миграции и с команда създаваме база от данни
  - Нужни пакети
    - Microsoft.EntityFrameworkCore.Tools
    - Microsoft.EntityFrameworkCore.Design

МЕТОДЪТ “CODE FIRST“

- Създаване на таблиците в БД
- EF е достатъчно умен, за да създаде първичен (Primary key) или вторичен ключ (Foreign key), забелязвайки имената на свойства,завършващи на ID, Id
- **Методът “Code First“**

```
public class OrderDetail
{
    public int OrderDetailID { get; set; }
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public int Quantity { get; set; }
    public Order Order { get; set; }
}
```

OrderID, ще е референция към първичния ключ на поръчката, за която сме добавили връзка

- **EF разбира този синтаксис и ще създаде връзка между двете таблици.**

МЕТОДЪТ “CODE FIRST“

- Съответно една поръчка ще има много на брой артикули за себе си
- Типа на връзката е един-към-много

```
public class Order
{
    public int OrderID { get; set; }
    public int CustomerID { get; set; }
    public int EmployeeID { get; set; }
    public DateTime OrderDate { get; set; }
    public List<OrderDetail> OrderDetails { get; set; }
}
```

EF разбира, че една поръчка има много детайли и ще създаде връзка един-към-много

МЕТОДЪТ “CODE FIRST“

- Накрая се добавят класовете като DbSet<T>в контекста
- Описват по какъв начин да стане връзката с БД

```
public class MyContext : DbContext
{
        public DbSet<OrderDetail> OrderDetails { get; set; }
        public DbSet<Order> Orders { get; set; }

        protected override void OnConfiguring (DbContextOptionsBuilder
        optionsBuilder)
        {
                optionsBuilder.UseSqlServer(@"Data
                Source=(localdb)\ProjectsV13;Initial Catalog=StoreDB;");
        }
}
```

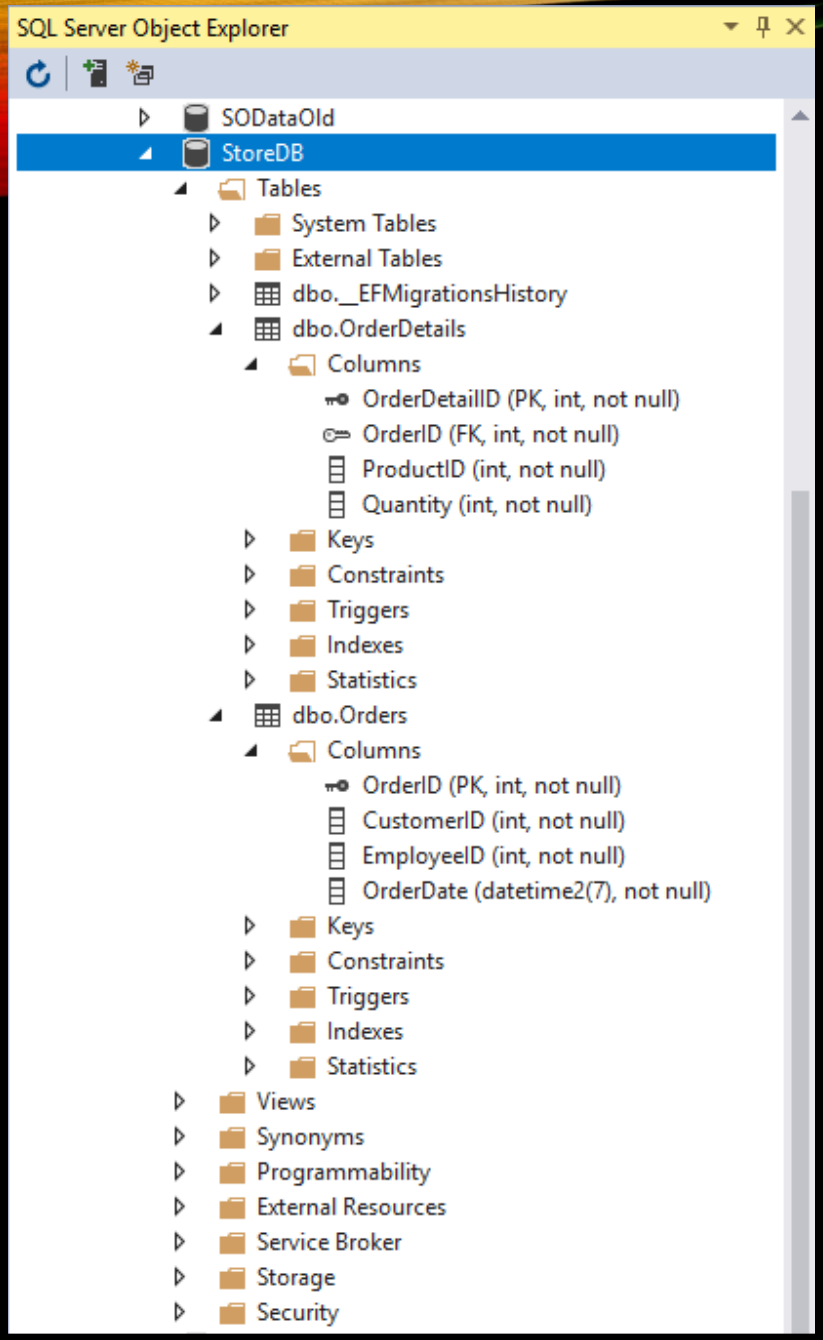
МЕТОДЪТ “CODE FIRST“

- Добавяне на първата миграция
- Създаване на БД по описаните таблици

- Използвайки Package Manager Console:
  $ Add-Migration Initial
  $ Update-Database

- Вече съществува съответната БД, описана чрез нашите класове

МЕТОДЪТ "CODE FIRST"



МЕТОДЪТ “CODE FIRST“

- Fluent API –начин да се конфигурират по-сложни връзки в БД, както и да се наложат ограничения за колоните
- Пренаписва се методът OnModelCreating(ModelBuildermodelBuilder) в нашият клас, служещ ни за контекст
- Чрез него може да се конфигурират връзки като 1-към-много, много-към-много и тн.

МЕТОДЪТ “CODE FIRST“

- Операциите за добавяне, изтриване, промяна и извличане се извършват по същия начин, както при Database First метода

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
        modelBuilder.Entity<BookCategory>()
                .HasKey(bc => new { bc.BookId, bc.CategoryId });
        modelBuilder.Entity<BookCategory>()
                .HasOne(bc => bc.Book)
                .WithMany(b => b.BookCategories)
                .HasForeignKey(bc => bc.BookId);
        modelBuilder.Entity<BookCategory>()
                .HasOne(bc => bc.Category)
                .WithMany(c => c.BookCategories)
                .HasForeignKey(bc => bc.CategoryId);
}
```

EF КОМПОНЕНТИ

Класът DbContext:

- Съдържа връзката към базата данни и преобразуваните класове
- Осигурява достъп до данни, базиран на LINQ
- Осигурява проследяване на идентичността, проследяване на промените и API за CRUD операции

Entity classes:

- Всяка таблица от база данни се свежда до C# клас

EF КОМПОНЕНТИ(2)

Асоциации (връзки между таблиците):

- Асоциацията е базирана на първичен ключ / чужд ключ между два класа
- Позволява навигация от един обект към друг

Concurrency контрол–контрол на едновременните заявки:

- Entity Framework използва optimistic concurrency контрол
- Няма заключване по подразбиране
- Автоматично открива concurrency конфликти


How did you fix that Bug?
SENIOR DEV
Commented the code.
INTERN

SoftUni Foundation

ЧЕТЕНЕ НА ДАННИ

ЗАЯВКИ КЪМ БД С ПОМОЩТА НА EF CORE

КЛАСЪТ DBCONTEXT

DbContext предоставя:

- CRUD Операции
  - Начин за достъпване на записите
  - Метод за добавяне на нови записи (методът Add())
  - Възможност за манипулиране на данни от база данни чрез промяна на обекти
- Изпълнение на LINQ заявки като SQL заявки
- Управление на база данни създаване/изтриване/миграция

ИЗПОЗЛВАНЕ НА КЛАСА DBCONTEXT

- Първо създайте инстанция на класа DbContext:
  var context = new SoftUniDbContext();
- В конструктора може да бъде подаден низ за свързване към БД;
- Свойствата на класа DbContext:
  - Database–EnsureCreated/Deleted методи, DBвръзка
  - ChangeTracker –Съдържа информация за вградения тракерза промени
  - Всички таблици са изредени като свойства в следния формат :
  - DbSet<Employee> Employees { get; set; }

ЧЕТЕНЕ НА ДАННИ С LINQ ЗАЯВКИ

- Изпълненние на LINQ заявка:

```
using (var context = new SoftUniEntities())
{
  var employees = context.Employees
    .Where(e => e.JobTitle == "Design Engineer")
    .ToArray();
}
```

EF превежда това до SQL заявка

- Employees е свойство към класа DbContext:

```
public partial class SoftUniEntities : DbContext
{
  public DbSet<Employee> Employees { get; set; }
  public DbSet<Project> Projects { get; set; }
  public DbSet<Department> Departments { get; set; }
}
```

ЧЕТЕНЕ НА ДАННИ С LINQ ЗАЯВКИ (2)

- Може да се използват и extension методи в заявката

```
using (var context = new SoftUniEntities())
  var employees = context.Employees
      .Where(c => c.JobTitle == "Design Engineering")
      .Select(c => c.FirstName)
      .ToList();
```

- Намиране на запис по ID

```
using (var context = new SoftUniEntities())
{
  var project = context.Projects.Find(2);
  Console.WriteLine(project.Name);
}
```

ПРОСТИ ОПЕРАЦИИ С LINQ

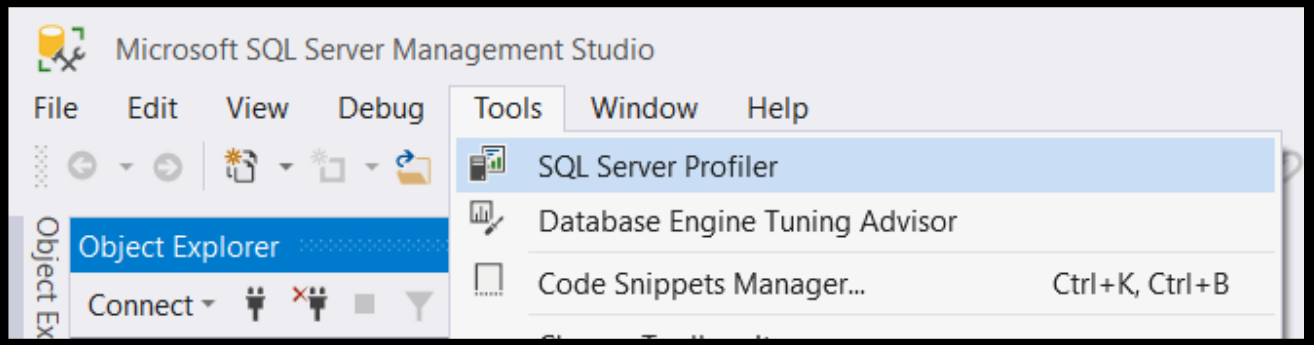
- Where() -Търси по дадено условие
- First/Last() / FirstOrDefault/LastOrDefault()
  - Получава първия / последния елемент, който съответства на условието
  - Хвърля InvalidOperationExceptionгрешка без OrDefault
- Select()
  - Преобразува колекция до друг тип
- OrderBy() / ThenBy() / OrderByDescending()
  - Сортира колекция по дадено условие

ПРОСТИ ОПЕРАЦИИ С LINQ(2)

- Any()
  - Проверява дали някой елемент съответства на условие
- All()
  - Проверява дали всички елементи съответстват на условие
- Distinct()
  - Връща само уникалните елементи от колекция
- Skip() / Take()
  - Пропуска / взима X на брой елементи

ПРОСЛЕДЯВАНЕ НА SQL ЗАЯВКИТЕ

- Заявки, изпратени до SQL Server, могат да бъдат наблюдавани с SQL Server Profiler
- Включено е в SQL Server Management Studio:



- Заявките също могат да бъдат наблюдавани с Express Profiler
- **Проследяване на SQL Заявките**
  https://expressprofiler.codeplex.com/

SoftUni Foundation

CRUD ОПЕРАЦИИ С EF CORE

СЪЗДАВАНЕ НА НОВИ ДАННИ

- За да създадете нов ред в БД, използвайте метода Add(...) на съответния DbSet:

```
var project = new Project()
{
	Name = "Judge System",
	StartDate = new DateTime(2015, 4, 15),
};
context.Projects.Add(project);
context.SaveChanges();
```

КАСКАДНИ ДОБАВЯНИЯ

- Можем да добавяме и каскадно:

```
Employee employee = new Employee();
employee.FirstName = "Petya";
employee.LastName = "Grozdarska";
employee.Projects.Add(new Project { Name = "SoftUni Conf"} );
softUniEntities.Employees.Add(employee);
softUniEntities.SaveChanges();
```

- Проектът ще бъде добавен, когато служителя бъде добавен в базата от данни

ПРОМЯНА НА СЪЩЕСТВУВАЩИ ДАННИ

- DbContext позволява промяна на свойствата на обект и запазване на промяната в базата данни
- Просто заредете записа, променете го и извикайте SaveChanges()
- DbContext автоматично проследява всички промени

```
Employees employee =
    softUniEntities.Employees.First();
employee.FirstName = "Alex";
context.SaveChanges();
```

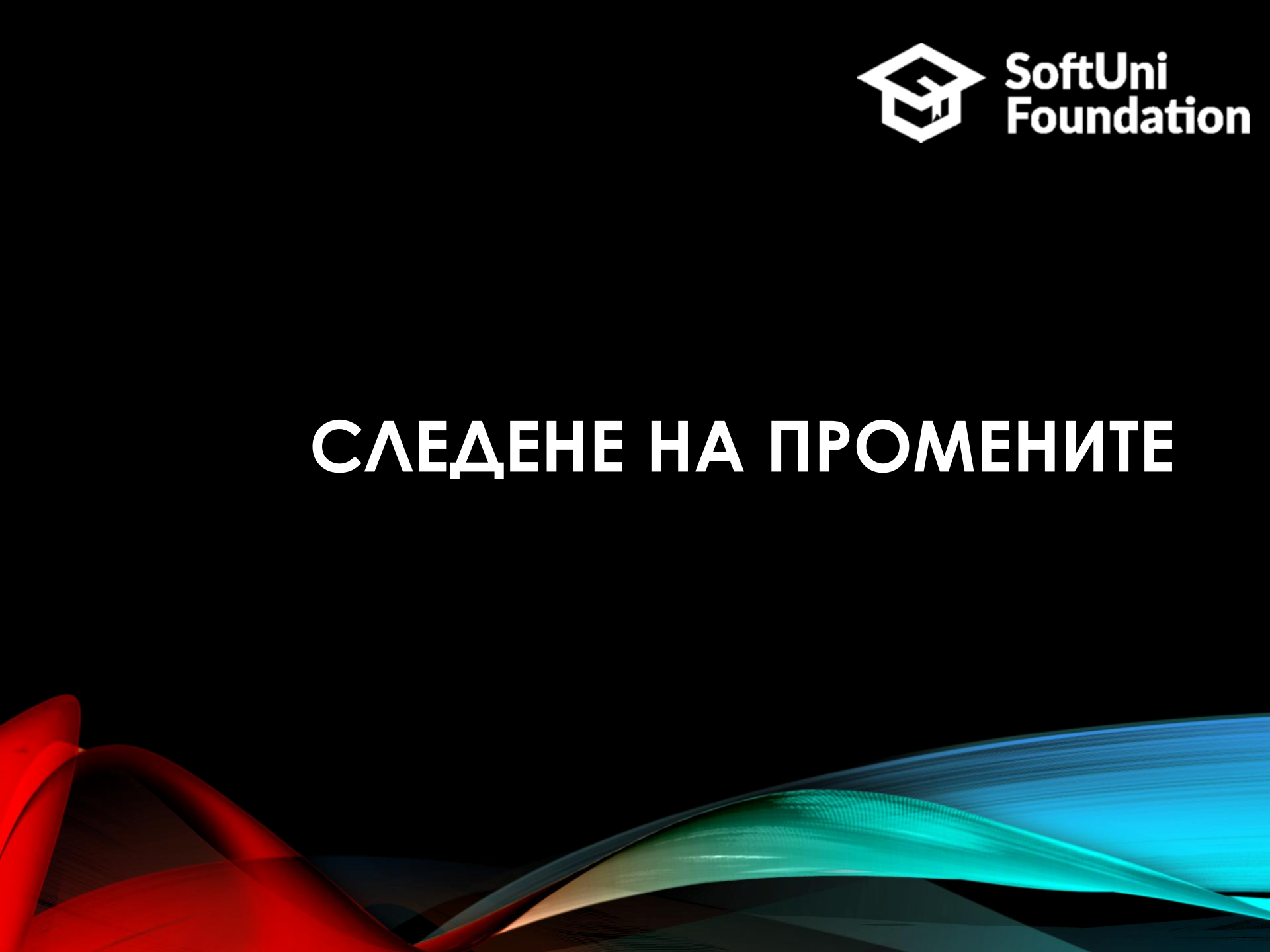
ИЗТРИВАНЕ НА СЪЩЕСТВУВАЩИ ДАННИ

- Изтриването се извършва чрез Remove() върху зададена колекция

```
Employees employee =
softUniEntities.Employees.First();
softUniEntities.Employees.Remove(employee);
softUniEntities.SaveChanges();
```

Маркира обекта за изтриване при следващото записване

Изпълнете командата за изтриване в SQL



СЛЕДЕНЕ НА ПРОМЕНИТЕ

СЛЕДЕНЕ НА ПРОМЕНИТЕ

- Методът SaveChanges()
  - Минава през вътрешна колекция на DbContextкласа
  - В зависимост от състоянието (Unchanged, Modified, Added, Deleted)генерира чрез type reflection заявка към БД
- Как се пази състоянието?
  - В списък от сложни обекти, които по същество пазят две важни свойства –референция към следените обекти, в която се пази даден обект и състоянието му (обектите са от тип DbEntityEntry)
  - Запазената референция е към следения обект

СЛЕДЕНЕ НА ПРОМЕНИТЕ

- При инициализация на обект (нека за пример вземем User) в Stackпаметтасе запазва референция сочеща към паметта (Heap), в която този обект се съхранява
- При направени промени в тази памет съответния на инстанцията DbEntityEntryобект променя съхраняваното състояние

СЛЕДЕНЕ НА ПРОМЕНИТЕ

- **User u = context.Users.Find(1);**

- Към момента нашият потребител u се пазивъв вътрешния списък на контекста със състояние Unchanged

- **u.Username = “New_Username"**

- След като сме променили свойства на обекта и в паметта (Heap-а)се отразяват нашите промени, те ще се отразят и в списъка на контекста
- Нашият потребител вече се пази със състояние Modified

СЛЕДЕНЕ НА ПРОМЕНИТЕ

- **context.Users.Remove(u);**

- Ако премахнем потребителя от контекста, той не се изтрива от базата –състоянието му се променя на Deleted

- **User newUser = new User();**
- **context.Users.Add(newUser);**

- Подобно, ако добавим нов потребител в контекста, той не се добавя в базата от данни, а само започва да се следи (състоянието му е Added)

СЛЕДЕНЕ НА ПРОМЕНИТЕ

- **context.SaveChanges();**
- Методът SaveChanges()обхожда вътрешния списък с DbEntityEntry елементи и за всеки следен обект изпълнява съответната заявка в зависимост от състоянието на записа

Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"

- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Принципи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA