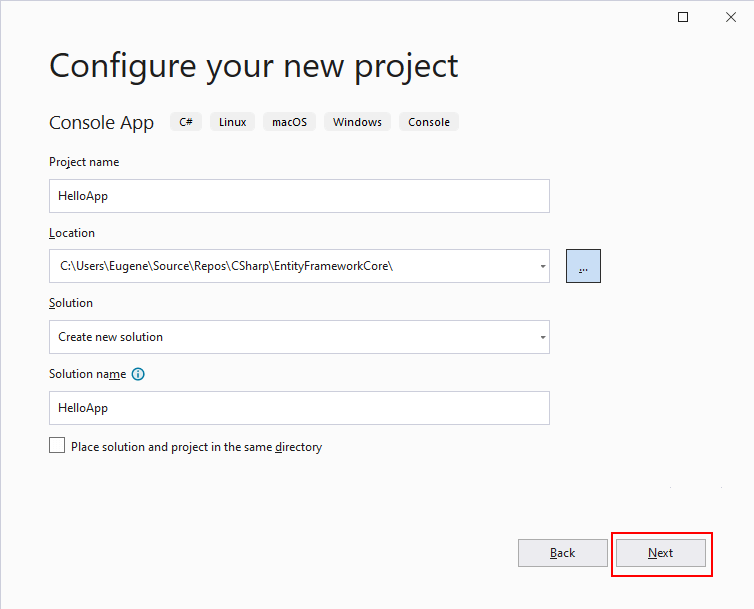
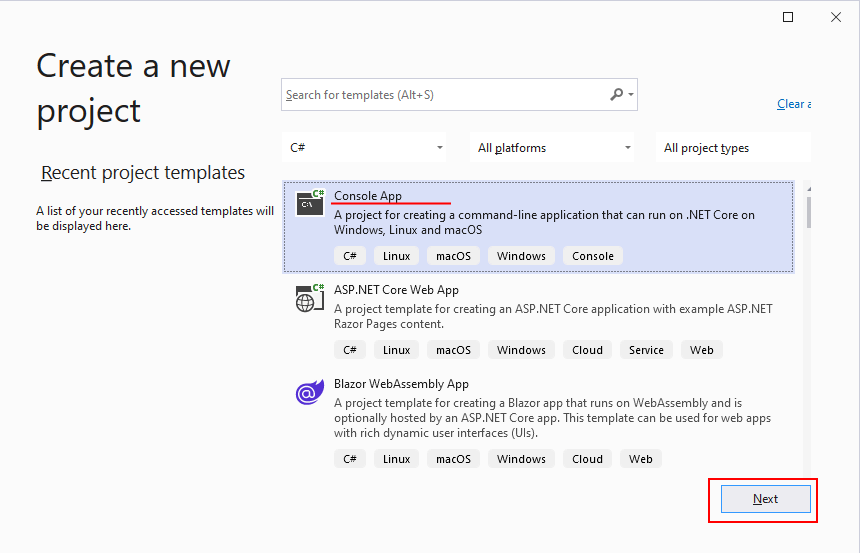
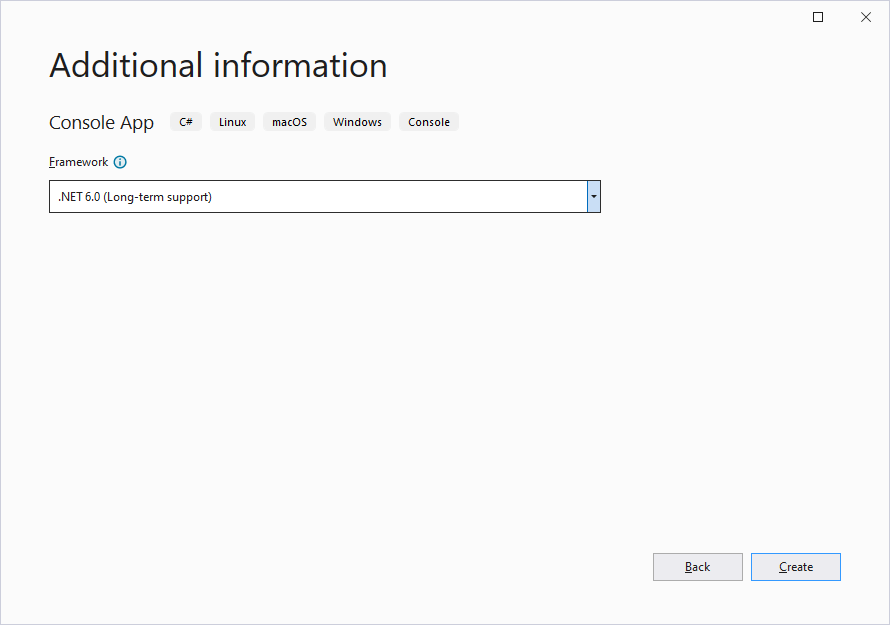
**Введение в Entity Framework Core**

═╬**1.1** ╬═ **Entity Framework** - представляет ORM-технологию (object-relational mapping отображения данных на реальные объекты) от компании Microsoft для доступа к данным. Entity Framework Core позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами классом независимо от типа хранилища. Если на физическом уровне мы оперируем таблицами, индексами, первичными и внешними ключами, но на концептуальном уровне, который нам предлагает Entity Framework, мы уже работаем с объектами.

✔Основная функциональность Entity Framework Core сосредоточена в следующих пакетах:

* Microsoft.EntityFrameworkCore:основной пакет EF Core
* Microsoft.EntityFrameworkCore.SqlServer: представляет функциональность провайдера для Microsoft SQL Server и SQL Azure
* Microsoft.EntityFrameworkCore.SqlServer.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQL Server
* Microsoft.EntityFrameworkCore.Sqlite: представляет функциональность провайдера для SQLite и включает нативные бинарные файлы для движка базы данных
* Microsoft.EntityFrameworkCore.Sqlite.Core: представляет функциональность провайдера для SQLite, но в отличие от предыдущего пакета не содержит нативные бинарные файлы для движка базы данных
* Microsoft.EntityFrameworkCore.Sqlite.NetTopologySuite: предоставляет поддержку географических типов (spatial types) для SQLite
* Microsoft.EntityFrameworkCore.Cosmos: представляет функциональность провайдера для Azure Cosmos DB
* Microsoft.EntityFrameworkCore.InMemory: представляет функциональность провайдера базы данных в памяти
* Microsoft.EntityFrameworkCore.Tools: содержит команды EF Core PowerShell для Visual Studio Package Manager Console; применяется в Visual Studio для миграций и генерации классов по готовой бд
* Microsoft.EntityFrameworkCore.Design: содержит вспомогательные компоненты EF Core, применяемые в процессе разработки
* Microsoft.EntityFrameworkCore.Proxies: хранит функциональность для так называемой "ленивой загрузки" (lazy-loading) и прокси остлеживания изменений
* Microsoft.EntityFrameworkCore.Abstractions: содержит набор абстракций EF Core, которые не зависят от конкретной СУБД
* Microsoft.EntityFrameworkCore.Relational: хранит компоненты EF Core для провайдеров реляционных СУБД
* Microsoft.EntityFrameworkCore.Analyzers: содержит функционал анализаторов C# для EF Core

**Работа с проектом**



Чтобы начать работать с EntityFramework Core, нам необходимо вначале добавить в проект пакет EntityFramework Core.

Для добавления пакетов можно использовать **Package Manager Console**. Для этого в меню Visual Studio перейдем к пункту **Tools** -> **NuGet Package Manager** -> **Package Manager Console.**

В открывшемся внизу в Visual Studio окне Package Manager Console введем команду:

► **Install-Package Microsoft.EntityFrameworkCore.Sqlite**

Добавим в проект новый класс **User**:

▌public class User

▌{

▌    public int Id { get; set; }

▌    public string? Name { get; set; }

▌    public int Age { get; set; }

▌}

Взаимодействие с базой данных в Entity Framework Core происходит посредством специального класса - контекста данных. Поэтому добавим в наш проект новый класс, который назовем **ApplicationContext** и который будет иметь следующий код:

▌using Microsoft.EntityFrameworkCore;

▌

▌public class ApplicationContext : DbContext

▌{

▌    public DbSet<User> Users => Set<User>();

▌    public ApplicationContext() => Database.EnsureCreated();

▌

▌    protected override void OnConfiguring(DbContextOptionsBuilder ▌optionsBuilder)

▌    {

▌        optionsBuilder.UseSqlite("Data Source=helloapp.db");

▌    }

▌}

Основу функциональности Entity Framework Core для работы с MS SQL Server составляют классы, которые располагаются в пространстве имен **Microsoft.EntityFrameworkCore**. Среди всего набора классов этого пространства имен следует выделить следующие:

* **DbContext**: определяет контекст данных, используемый для взаимодействия с базой данных
* **DbSet/DbSet<TEntity>**: представляет набор объектов, которые хранятся в базе данных
* **DbContextOptionsBuilder**: устанавливает параметры подключения

Теперь определим сам код программы, который будет взаимодействовать с созданной БД. Для этого изменим файл **Program.cs** следующим образом:

▌using (ApplicationContext db = new ApplicationContext())

▌{

▌    // создаем два объекта User

▌    User tom = new User { Name = "Tom", Age = 33 };

▌    User alice = new User { Name = "Alice", Age = 26 };

▌

▌    // добавляем их в бд

▌    db.Users.Add(tom);

▌    db.Users.Add(alice);

▌    db.SaveChanges();

▌    Console.WriteLine("Объекты успешно сохранены");

▌

▌    // получаем объекты из бд и выводим на консоль

▌    var users = db.Users.ToList();

▌    Console.WriteLine("Список объектов:");

▌    foreach (User u in users)

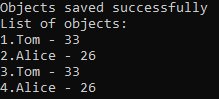
▌    {

▌        Console.WriteLine($"{u.Id}.{u.Name} - {u.Age}");

▌    }

▌}

В результате после запуска программа выведет на консоль:

****