Инструменты и средства программирования (часть 2)

Лабораторная работа №5

Цель работы: получение основных навыков в разработке настольных приложений с использованием таких фреймворков, как .Net MAUI и EntityFramework Core. Знакомство с современными архитектурными стилями программирования.

Задача работы: разработка приложения с луковичной (onion) или чистой (clean) архитектурой, с хранением данных в реляционной базе данных.

Время выполнения работы: 16 часов (8 занятий)

Результат выполнения работы: приложение, обеспечивающее функционал согласно заданию.

Содержание

1. Предварительная информация

Проанализируйте предметную область согласно варианту задания.

Класс, описывающий группу объектов, должен содержать следующие свойства:

- Идентификатор
- Название
- Какое-нибудь дополнительное свойство (дату, продолжительность, стоимость, вес, оклад и т.д.)
- Навигационное свойство – коллекцию объектов-членов группы

Класс, описывающий члена группы, должен содержать следующие свойства:

- Идентификатор
- Название
- Обязательное свойство согласно варианту задания.
- 2-3 дополнительных свойства, описывающие объект
- Навигационные свойства – идентификатор группы, объект класса группы.

Отношение между классами: один-ко-многим. В одной группе может быть много объектов, но один объект не может принадлежать к разным группам.

Примечание: в приводимых ниже примерах используется предметная область «курсы – слушатели курсов». Обязательное свойство класса слушателей – средний балл (рейтинг)

1.1. Постановка задачи

Требуется создать приложение, позволяющее:

1. Вывод списка групп объектов (в виде «выпадающего» списка названий групп)
2. При выборе группы вывести **на этой же странице** полную информацию о группе, а также список объектов – членов группы.
3. **Выделить цветом** объекты, у которых обязательное свойство не соответствует какому-нибудь критерию, например, меньше допустимого значения.
4. При клике на объект в списке открывается страница с подробной информацией о выбранном объекте. На странице предусмотреть **меню** для перехода на страницу редактирования информации, а также удаление объекта.
5. Предусмотреть возможность создания и добавления объектов в группу.

6. Предусмотреть возможность добавления и редактирования групп
7. Предусмотреть сохранение файла изображения объекта. Имя файла должно соответствовать идентификатору объекта.

Требования к проекту

Реализовать в проекте архитектуру Onion и шаблон проектирования UnitOfWork. (с шаблоном UnitOfWork можно познакомиться, например, здесь: https://metanit.com/sharp/mvc5/23.3.php или здесь: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/implementing-the-repository-and-unit-of-work-patterns-in-an-asp-net-mvc-application)

На страницах приложения реализовать шаблон MVVM

Для получения объектов сервисов, репозиториев и т.д. использовать внедрение зависимостей

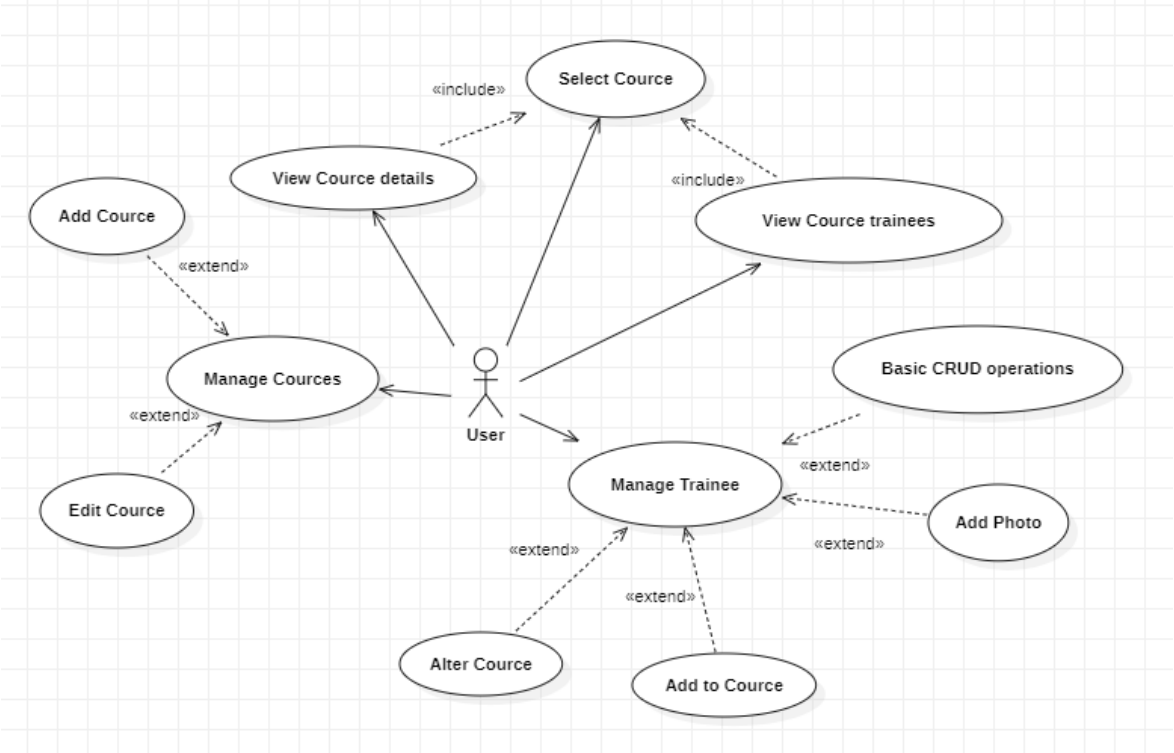
1.2. Варианты заданий

1. Спортивные команды – участники (обязательное свойство – количество очков у участника)
2. Исполнители – песни (обязательное свойство – место песни в чарте)
3. Наборы суши – суши в наборе (обязательное свойство – количество готовых суши)
4. Супергерои – способности (обязательное свойство – степень развития способности в процентах от максимально возможной)
5. Туристические маршруты – достопримечательности (обязательное свойство – стоимость входного билета для осмотра достопримечательности)
6. Категория гостиничного номера – представляемый сервис (обязательное свойство – количество свободных номеров)
7. Должности сотрудников – должностные обязанности (обязательное свойство – важность обязанности по 10-бальной шкале)
8. Бригады –выполняемые работы (обязательное свойство – качество выполнения по 10-бальной шкале)
9. Коктейли – ингредиенты (обязательное свойство – количество запаса ингредиента на складе)
10. Авторы – книги (обязательное свойство – рейтинг книги)

2. Выполнение работы

2.1. Разработка функциональных требований

Исходя из поставленной задачи выделите функции системы и представьте их в виде диаграммы вариантов использования или в виде списка, например:



2.2. Требования к ресурсам системы

Определитесь со средой разработки

Определитесь с типом готового проекта (мобильное или настольное) и целевую платформу.

Выберите ORM для работы с базой данных.

В приведенных ниже примерах используется **EntityFramework Core**. Вы можете использовать любую ORM.

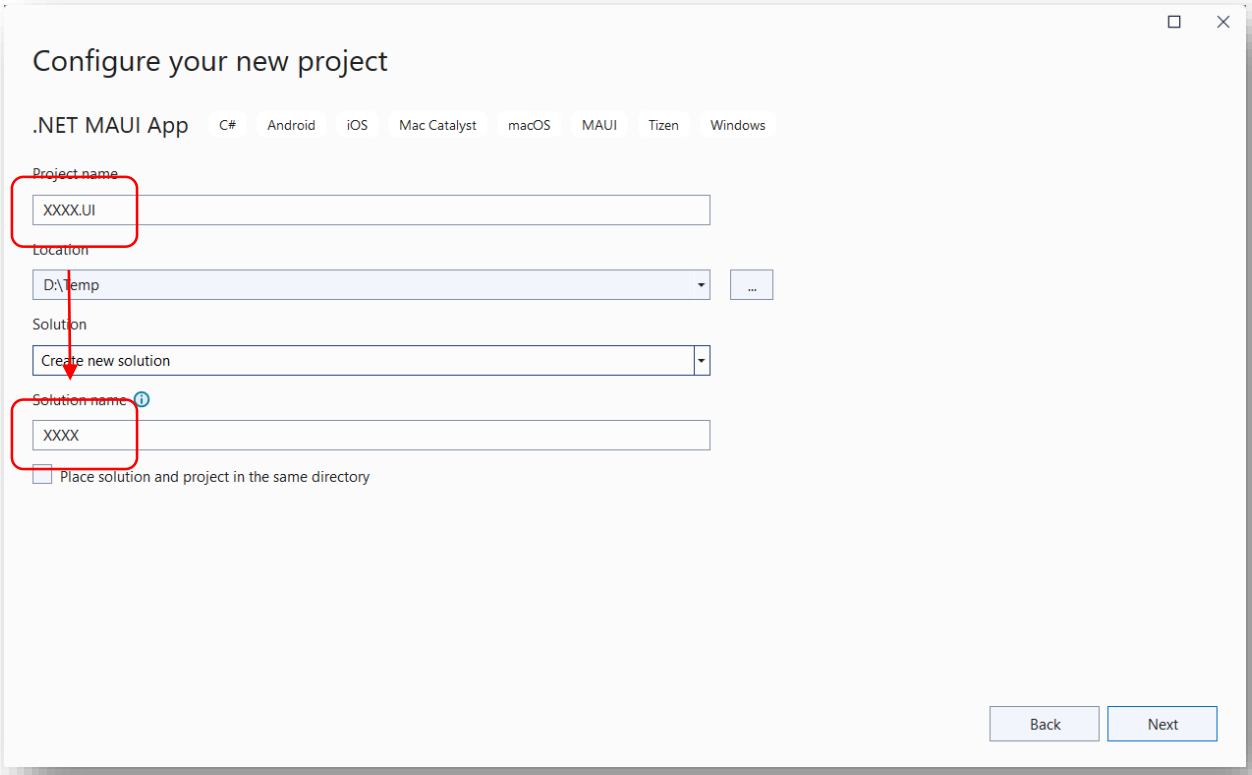
Выберите базу данных.

Для сохранения данных предлагается использовать **SQLite**. Если вы не планируете запускать приложение на мобильных устройствах (или их эмуляторах), можете использовать любую другую базу данных. В этом случае вам нужно добавить в проект соответствующего поставщика данных

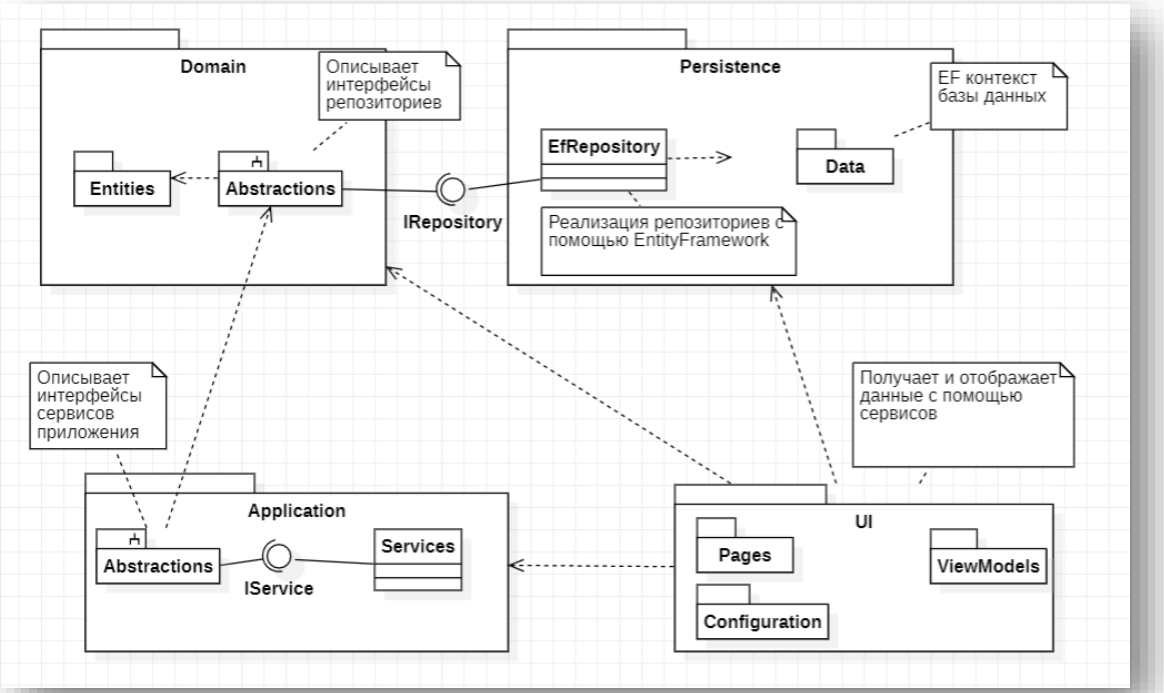
2.3. Разработка архитектуры проекта и проектирование доменой модели

2.3.1. Создание проекта .Net MAUI.

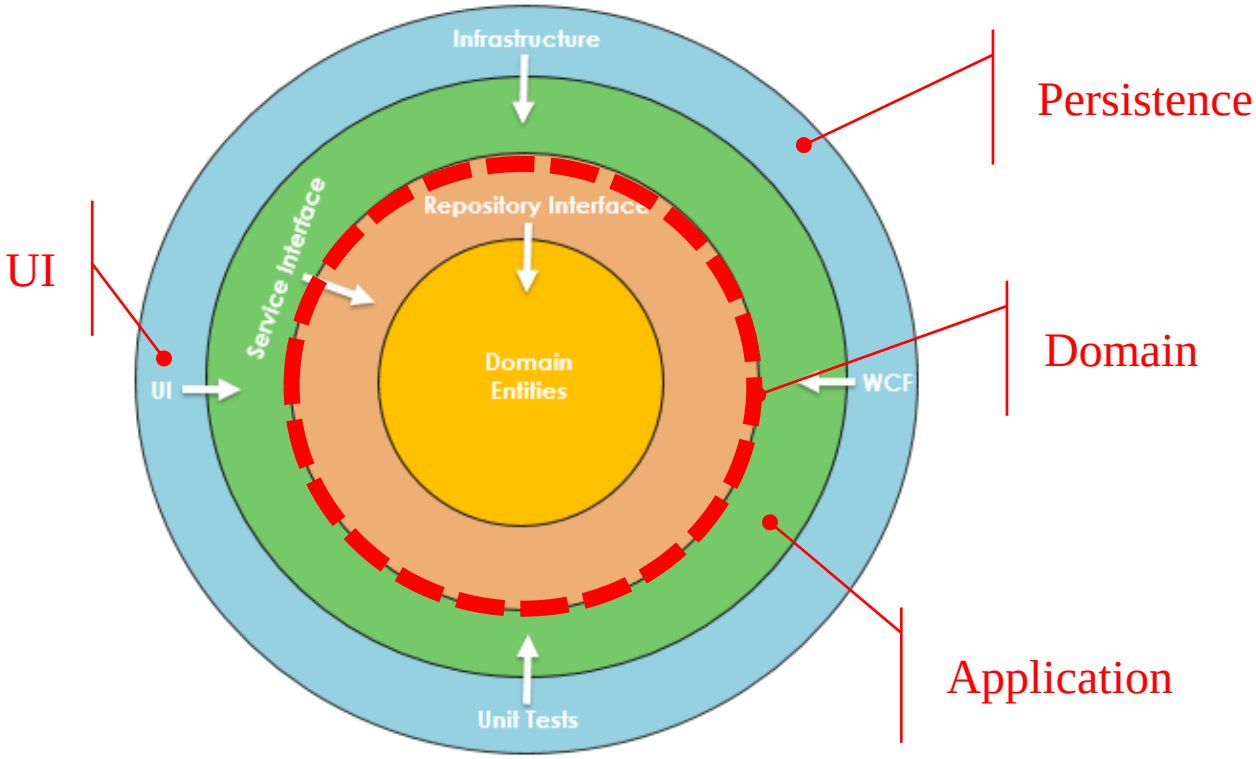
Создайте проект .Net MAUI.
Имя решения – номер группы и фамилия, например 153500_Ivanov
В имени проекта напишите XXXX.UI, где XXXX – имя решения:



Требуется создать проект согласно схеме, приведенной ниже:



Для Onion архитектуры это соответствует так:



Добавьте в решение соответствующие библиотеки классов:

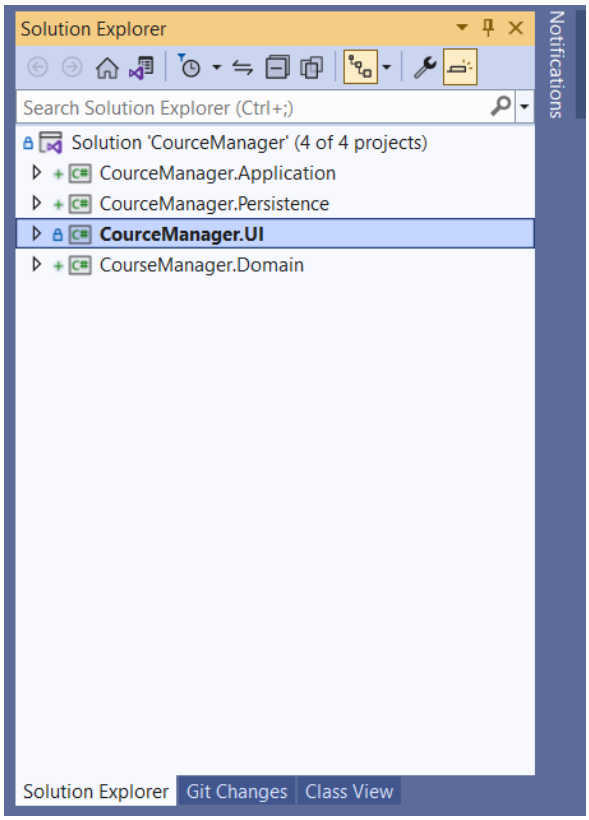
- XXXX.Domain
- XXXX.Application
- XXXX.Persistense

В библиотеке XXXX.Domain будут описаны сущностные классы Вашей предметной области и интерфейсы репозиториев, необходимых для работы с сущностями и интерфейс IUnitOfWork.

В библиотеке XXX.Application будут описаны интерфейсы сервисов, необходимых для реализации функций вашего приложения.

В библиотеке XXXX.Persistence будут описаны реальные репозитории для работы с базой данных и контекст EntityFramework

Пример решения:



2.3.2. Описание доменной модели

Добавьте в проект XXXX.Domain папку Entities. Опишите в папке классы вашей предметной области

Добавьте в проект XXXX.Domain папку Abstractions. Опишите в папке **обобщенный** (generic) интерфейс репозитория.

Например:

```
public interface IRepository<T>  where T:Entity
    {
        /// <summary>
        /// Поиск сущности по Id
        /// </summary>
        /// <param name="id">Id сущности</param>
        /// <param name="cancellationToken"></param>
        /// <param name="includesProperties">Делегаты для подключения навигационных
свойств</param>
        /// <returns></returns>
        Task<T> GetByIdAsync(int id, CancellationToken cancellationToken = default,
                             params Expression<Func<T, object>>[]? includesProperties);

        /// <summary>
        /// Получение всего списка сущностей
        /// </summary>
        /// <param name="cancellationToken"></param>
        /// <returns></returns>
        Task<IReadOnlyList<T>> ListAllAsync(CancellationToken cancellationToken =
default);

        /// <summary>
        /// Получение отфильтрованного списка
        /// </summary>
        /// <param name="filter">Делегат-условие отбора</param>
        /// <param name="cancellationToken"></param>
        /// <param name="includesProperties">Делегаты для подключения навигационных
свойств</param>
        /// <returns></returns>
        Task<IReadOnlyList<T>> ListAsync(Expression<Func<T, bool>> filter,
            CancellationToken cancellationToken = default,
            params Expression<Func<T, object>>[]? includesProperties);

        /// <summary>
        /// Добавление новой сущности
        /// </summary>
        /// <param name="entity"></param>
        /// <param name="cancellationToken"></param>
        /// <returns></returns>
        Task AddAsync(T entity, CancellationToken cancellationToken = default);

        /// <summary>
        /// Изменение сущности
        /// </summary>
        /// <param name="entity">Сущность с измененным содержимым</param>
        /// <param name="cancellationToken"></param>
        /// <returns></returns>
        Task UpdateAsync(T entity, CancellationToken cancellationToken = default);

        /// <summary>
        /// Удаление сущности
        /// </summary>
        /// <param name="entity">Сущность, которую следует удалить</param>
        /// <param name="cancellationToken"></param>
```

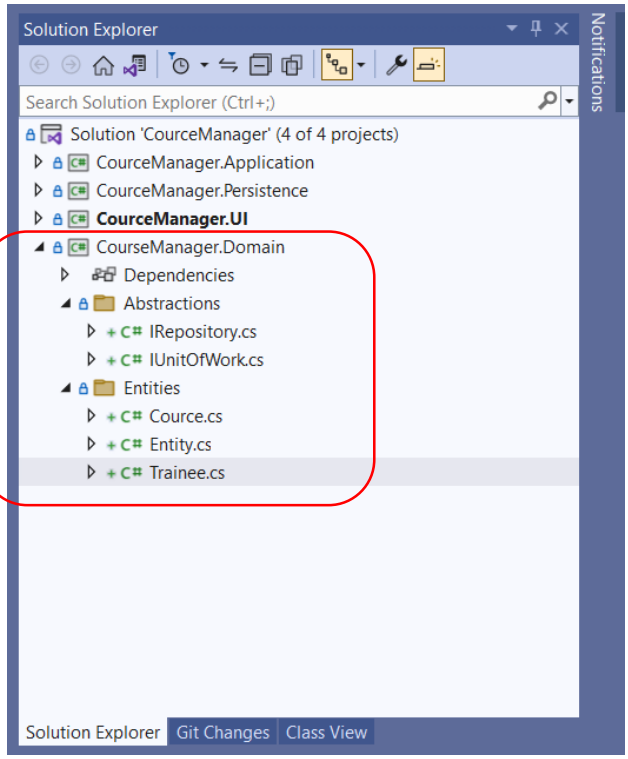
```
        /// <returns></returns>
        Task DeleteAsync(T entity, CancellationToken cancellationToken = default);


        /// <summary>
        /// Поиск первой сущности, удовлетворяющей условию отбора.
        /// Если сущность не найдена, будет возвращено значение по умолчанию
        /// </summary>
        /// <param name="filter">Делегат-условие отбора</param>
        /// <param name="cancellationToken"></param>
        /// <returns></returns>
        Task<T> FirstOrDefaultAsync(Expression<Func<T, bool>> filter, CancellationToken
cancellationToken = default);
          }
```

Примечание: в приведенном примере класс Entity – базовый класс для классов Course и Trainee. Он описывает общие для всех сущностей свойства, например, Id, Name и др.

В папке Abstractions опишите интерфейс IUnitOfWork, предоставляющий доступ к репозиториям, а также метод сохранения всех изменений. Например:

```
public interface IUnitOfWork
 {
     IRepository<Cource> CourseRepository { get; }
     IRepository<Trainee> TraineeRepository { get; }
     public Task RemoveDatbaseAsync();
     public Task CreateDatabaseAsync();
     public Task SaveAllAsync();
 }
```

Результат:



2.3.3. Вопросы для самопроверки

– Что такое «Луковая» архитектура приложения? Ее преимущества перед трехслойной архитектурой.
– Что такое «Принцип устойчивых зависимостей» компонентов приложения?
– Как оцениваются метрики устойчивости?

2.4. Разработка слоя доступа к данным (Persistence)

2.4.1. Подготовка проекта

Добавьте в проект XXXX.Persistence ссылку на проект XXXX.Domain

Добавьте в проект XXXX.Persistence NuGet пакет Microsoft.EntityFrameworkCore

2.4.2. Разработка контекста БД

Добавьте в проект XXXX.Persistence папку Data.

В папке Data создайте контекст базы данных (можно использовать имя AppDbContext).

Конструктор контекста должен принимать параметр DbContextOptions<AppDbContext>

В конструкторе контекста создайте базу данных:

```
Database.EnsureCreated();
```

2.4.3. Разработка репозиториев и UnitOfWork

1) Добавьте в проект XXXX.Persistence папку Repository.

2) В папке Repository опишите обобщенный (generic) класс EfRepository<T>, реализующий интерфейс IRepository<T> с помощью контекста, созданного в п.2.2.2. Контекст внедрите в конструктор класса EfRepository.

Пример реализации методов `Task<IReadOnlyList<T>> ListAsync(...);` и `Task UpdateAsync`

```
public class EfRepository<T> : IRepository<T> where T:Entity
    {
        protected readonly AppDbContext _context;
        protected readonly DbSet<T> _entities;

        public EfRepository(AppDbContext context)
        {
            _context = context;
            _entities = context.Set<T>();
        }

. . .

  public async Task<IReadOnlyList<T>> ListAsync(
            Expression<Func<T, bool>>? filter,
            CancellationToken cancellationToken = default,
            params Expression<Func<T, object>>[] includesProperties)
        {
```

```
            IQueryable<T>? query = _entities.AsQueryable();
            if (includesProperties.Any())
            {
                foreach (Expression<Func<T, object>>? included in
includesProperties)
                {
                    query = query.Include(included);
                }
            }

            if (filter != null)
            {
                query = query.Where(filter);
            }


            return await query.ToListAsync();
        }

        public Task UpdateAsync(T entity,
                    CancellationToken cancellationToken = default)
        {
            _context.Entry(entity).State = EntityState.Modified;
            return Task.CompletedTask;
        }
    }
```

Опишите класс EfUnitOfWork, реализующий интерфейс IUnitOfWork с помощью контекста, созданного в п.2.2.2. Контекст внедрите в конструктор класса EfUnitOfWork. Пример:

```
public class EfUnitOfWork : IUnitOfWork
{
    private readonly AppDbContext _context;
    private readonly Lazy<IRepository<Cource>> _courceRepository;
    private readonly Lazy<IRepository<Trainee>> _traineeRepository;

    public EfUnitOfWork(AppDbContext context)
    {
        _context = context;
        _courceRepository = new Lazy<IRepository<Cource>>(() =>
                                new EfRepository<Cource>(context));
        _traineeRepository = new Lazy<IRepository<Trainee>>(() =>
                                new EfRepository<Trainee>(context));
     }
```

```
     IRepository<Cource> IUnitOfWork.CourseRepository =>
_courceRepository.Value;

     IRepository<Trainee> IUnitOfWork.TraineeRepository =>
_traineeRepository.Value;

      public async Task CreateDatabaseAsync()
      {
            await _context.Database.EnsureCreatedAsync();
        }

        public async Task RemoveDatbaseAsync()
        {
            await _context.Database.EnsureDeletedAsync();
        }

        public async Task SaveAllAsync()
        {
            await _context.SaveChangesAsync();
        }
}
```

2.4.4. Разработка тестовых репозиториев и UnitOfWork

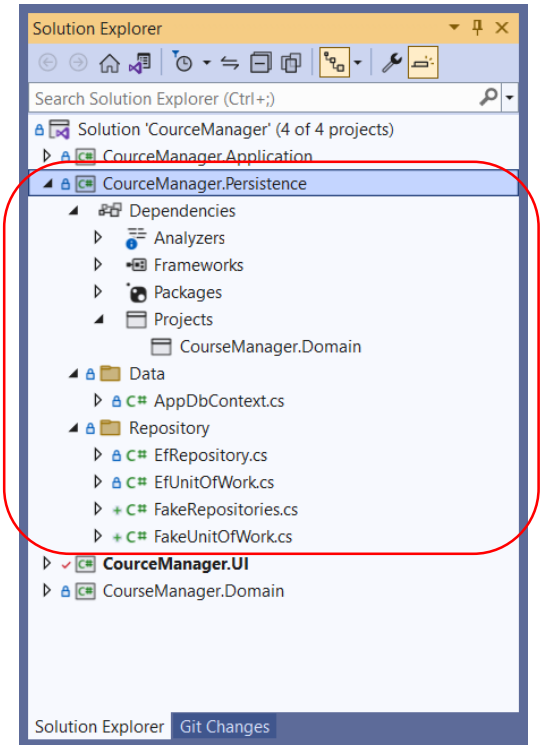
Для начальной разработки интерфейса нам понадобится имитация работы с реальной базой данных. Для этого добавьте классы Fake репозиториев. В классе FakeCourseRepository реализуем только метод `ListAllAsync`, который вернет коллекцию из двух объектов:

```
public class FakeCourseRepository : IRepository<Cource>
{
    List<Cource> _cources;
    public FakeCourseRepository()
    {
        _cources = new List<Cource>()
        {
            new Cource(){ Id=1, Name="Вязание крючком", Duration=10,
StartDate=DateTime.Now.AddDays(5), Trainees=new List<Trainee>()},
            new Cource(){ Id=2, Name="Философия мультсериала «Симпсоны»",
Duration=5, StartDate=DateTime.Now.AddDays(10), Trainees=new
List<Trainee>()}
        };
    }
        . . .
    public async Task<IReadOnlyList<Cource>> ListAllAsync(CancellationToken
cancellationToken = default)
    {
        return await Task.Run(() => _cources);
    }
}
```

В классе FakeTraineeRepository также реализуем только один метод, позволяющий получить список объектов в группе:

```
public class FakeTraineeRepository : IRepository<Trainee>
{
    List<Trainee> _list = new List<Trainee>();
    public FakeTraineeRepository()
    {
        Random rand = new Random();
        int k = 1;
        for (int i = 1; i <= 2; i++)
            for (int j = 0; j < 10; j++)
                _list.Add(new Trainee() { Id = k, Name = $"Trainee {k++}",
CourseId = i, Rate = rand.NextDouble() * 10 });
    }

   public async Task<IReadOnlyList<Trainee>>
ListAsync(Expression<Func<Trainee, bool>> filter, CancellationToken
cancellationToken = default, params Expression<Func<Trainee, object>>[]?
includesProperties)
   {
     var data = _list.AsQueryable();
     return  data.Where(filter).ToList();
   }        . . .
}
```

В классе FakeUnitOfWork используем созданные тестовые репозитории. Метод SaveAllAsync можно не реализовывать.



2.4.5. Вопросы для самопроверки

– Что такое EntityFramework Core?

– Как описать контекст БД в EntityFramework Core?

– Как в EntityFramework Core указать, какую базу данных нужно использовать?

– Как в EntityFramework Core указать отношение между таблицами 1-ко-многим?

2.5. Разработка слоя Application

2.5.1. Подготовка проекта

В проект XXXX.Application Добавьте ссылку на проект XXXX.Domain

2.5.2. Описание интерфейсов

Добавьте в проект XXXX.Application папку Abstractions.

В папке Abstractions опишите интерфейсы служебных классов для реализации бизнес-логики приложения. Терминологию выберите сами. Это могут быть классы менеджеров, сервисов и т.д. Далее по тексту будет использоваться термин «сервис».

Например, сервис для работы с группами объектов может содержать набор базовых CRUD методов (Create – Read – Update – Delete).

Сервис для работы с объектами дополнительно может содержать методы получения объектов для заданной группы, добавление объекта в группу, перемещение объекта из одной группы в другую, добавление фото.

Поскольку операции CRUD дублируются в обоих сервисах, можно описать обобщенный интерфейс IBaseService<T>:

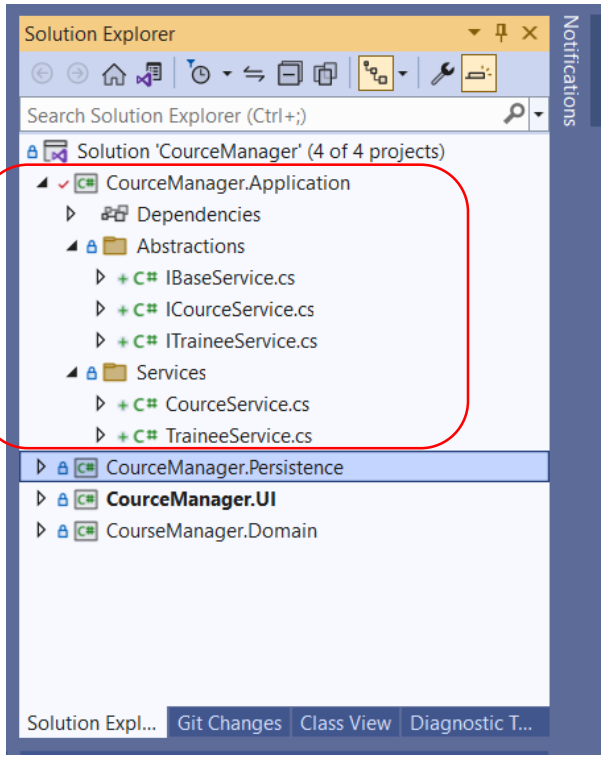
```
public interface IBaseService<T> where T : Entity
    {

        Task<IEnumerable<T>> GetAllAsync();
        Task<T> GetByIdAsync(int id);
        Task<T> AddAsync(T item);
        Task<T> UpdateAsync(T item);
        Task<T> DeleteAsync(int id);
    }
```

2.5.3. Описание сервисов

Добавьте в проект XXXX.Application папку Services.

В папке Services опишите сервисы, реализующие интерфейсы, описанные в папке Abstractions.

Каждый сервис должен принимать в конструктор объект IUnitOfWork для возможности манипуляции данными.



2.5.4. Вопросы для самопроверки

– Что такое операции CRUD?

2.6. Подготовка проекта UI

2.6.1. Предварительные действия

1) Добавьте в проект XXXX.UI ссылки на проекты XXXX.Domain, XXXX.Application и XXXX.Persistense
2) Загрузите в проект XXXX.UI NuGet пакет **CommunityToolkit.Maui**.
3) В классе MauiProgram добавьте:

```
builder
    .UseMauiApp<App>()
    .UseMauiCommunityToolkit()
```

4) Загрузите в проект XXXX.UI NuGet пакет **CommunityToolkit.Mvvm**
5) Добавьте в проект XXXX.UI папку Pages. В ней будут описаны все страницы приложениия
6) Добавьте в проект XXXX.UI папку ViewModels. В ней будут описаны модели страниц.

2.6.2. Регистрация сервисов

В классе MauiProgram зарегистрируйте сервисы из проекта XXXX.Application и IUnitOfWork. В качестве реализации IUnitOfWork используйте FakeUnitOfWork, Например:

```
public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf",
"OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf",
"OpenSansSemibold");
            });

        SetupServices(builder.Services);
        return builder.Build();
    }
    private static void SetupServices(IServiceCollection services)
    {
        //Services
```

```
services.AddSingleton<IUnitOfWork, FakeUnitOfWork>();
services.AddSingleton<ITraineeService, TraineeService>();
services.AddSingleton<ICourceService, CourceService>();

//Pages


//ViewModels
}
```

2.6.3. Вопросы для самопроверки

– Что такое шаблон проектирования MVVM?
– Как настроить контейнер IoC в приложении .Net MAUI?
– Чем отличается сервис «singleton» от «transient»?

2.7. Разработка стартовой страницы приложения

2.7.1. Подготовка страницы

В папку Pages добавьте новый элемент - .Net MAUI Content Page (XAML). Данная страница будет выводить список групп объектов (см. п.1.1, требование 1). Имя страницы должно соответствовать ее назначению. В примерах ниже бует использоваться имя Cources.

В разметке AppShell:

- уберите директиву `Shell.FlyoutBehavior="Disabled"`
- добавьте пространство имен для папки Pages
- добавьте в разметку созданную страницу:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="CourceManager.UI.AppShell"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:CourceManager.UI"
    xmlns:pages="clr-namespace:CourceManager.UI.Pages">

    <ShellContent
        Title="Home"
        ContentTemplate="{DataTemplate local:MainPage}"
        Route="MainPage" />
    <ShellContent
        Title="Cources list"
        ContentTemplate="{DataTemplate pages:Cources}"
        Route="Cources" />

</Shell>
```

Запустите проект и проверьте, что созданная страница появилась в навигации.

2.7.2. Создание модели представления

Добавьте в папку ViewModels сласс модели представления для страницы, созданной в п. 2.7.1. Имя класса – имя страницы с суффиксом ViewModel.

Класс должен наследоваться от класса **ObservableObject** из пространства имен **CommunityToolkit.Mvvm.ComponentModel.** Используйте модификатор **partial.**

Внедрите в созданный класс сервисы: для доступа к данным групп объектов и классам-членам групп.

Опишите две коллекции – коллекция объектов и коллекция групп объектов (согласно индивидуальному заданию). Эти коллекции будут источником данных для вывода на страницу.

Добавьте поле, описывающее выбранную группу объектов. Используйте атрибут **[ObservableProperty]**

Опишите метод, заполняющий данными список групп объектов (с помощью соответствующего сервиса)

Опишите метод, заполняющий данными список объектов группы (с помощью соответствующего сервиса). Id группы брать из объекта выбранной группы, описанного ранее.

Создайте команды UpdateGroupList и UpdateMembersList, вызывающие созданные методы (используйте атрибут **[RelayCommand])**

Пример:

```
public partial class CourcesViewModel : ObservableObject
    {
        private readonly ICourceService _courceService;
        private readonly ITraineeService _traineeService;

        public CourcesViewModel(ICourceService courceService, ITraineeService
traineeService)
        {
            _courceService = courceService;
            _traineeService = traineeService;
        }

        public ObservableCollection<Cource> Cources { get; set; } = new()
        public ObservableCollection<Trainee> Trainees { get; set; } = new()

        [ObservableProperty]
        Cource selectedCource;

        [RelayCommand]
        async void UpdateGroupList() => await GetCources();
        [RelayCommand]
        async void UpdateMembersList() => await GetTrainees();

        public async Task GetCources()
        {
            var cources = await _courceService.GetAllAsync();
            await MainThread.InvokeOnMainThreadAsync(() =>
            {
                Cources.Clear();
                foreach (var cource in cources)
                    Cources.Add(cource);
            });
        }

        public async Task GetTrainees()
        {
            var trainees = await _traineeService.GetCourseMembers(SelectedCource.Id);
            await MainThread.InvokeOnMainThreadAsync(() =>
            {
```

```
                Trainees.Clear();
                foreach (var trainee in trainees)
                    Trainees.Add(trainee);
            });
        }
}
```

Зарегистрируйте созданный класс в качестве сервиса в класса MauiProgram.

```
//ViewModels
services.AddSingleton<CourcesViewModel>();
```

2.7.3. Вывод списка групп на странице

Примечание: выбор шаблона компоновки (Grid, CtackLayout и т.д.) – на ваше усмотрение

Внедрите созданную модель представления в конструктор класса страницы и назначьте ее в качестве BindingContext

В разметке XAML страницы укажите модель представления в качестве типа данных:

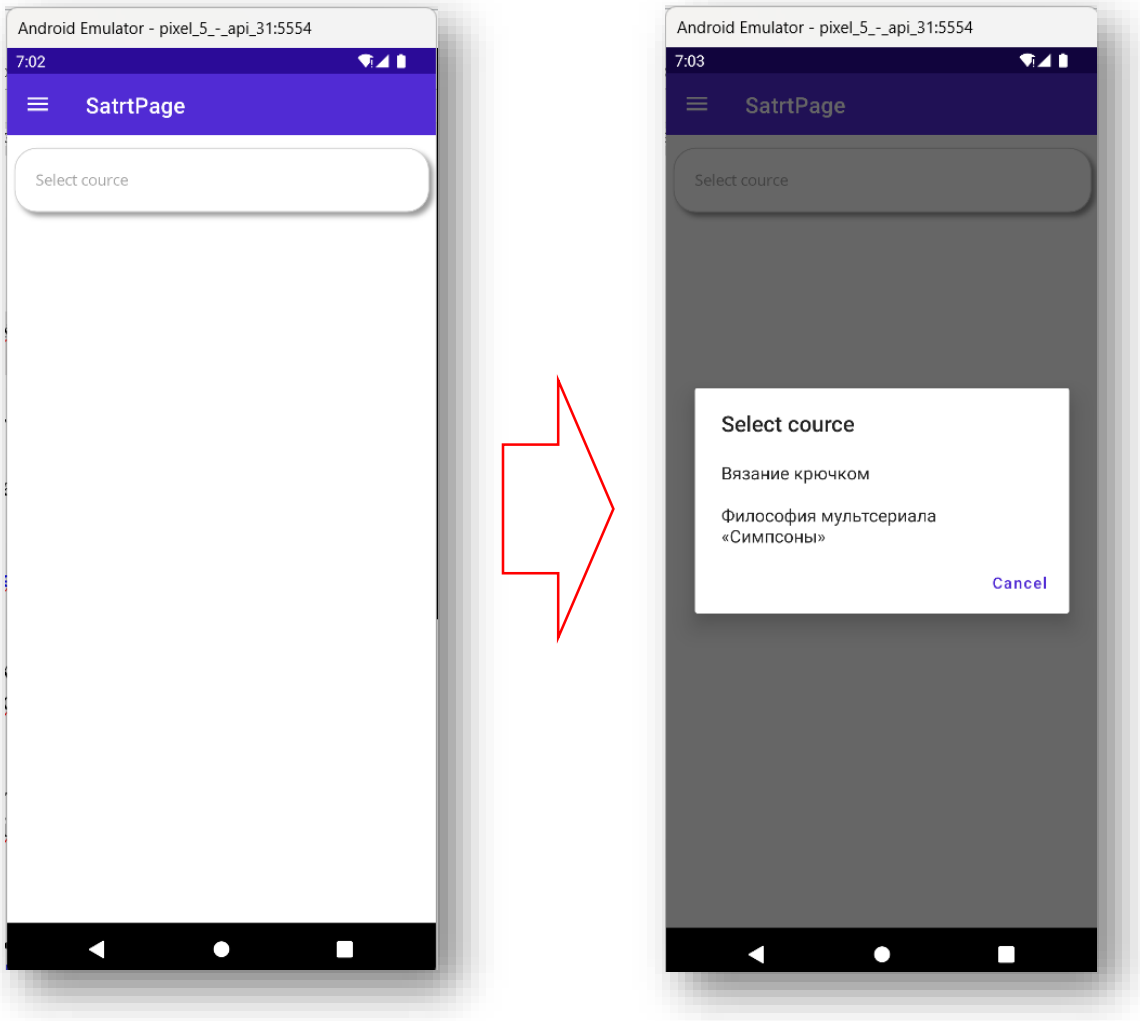
```
xmlns:models="clr-namespace:CourceManager.UI.ViewModels"
x:DataType="models:CourcesViewModel"
```

Разместите на странице элемент управления Picker, в котором отобразите список из коллекции групп объектов, описанной во ViewModel. Список должен показывать названия групп объектов.

Список групп должен загружаться при загрузке страницы. Свяжите событие «Loaded» страницы с командой UpdateGroupList во ViewModel. Используйте для этого EventToCommandBehavior из библиотеки CommunityToolkit.Maui:

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:models="clr-namespace:CourceManager.UI.ViewModels"
             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
             x:DataType="models:CourcesViewModel"
             x:Class="CourceManager.UI.Pages.Cources"
             Title="SatrtPage">
    <ContentPage.Behaviors>
        <toolkit:EventToCommandBehavior EventName="Loaded"
                                        Command="{Binding UpdateGroupListCommand }"/>
    </ContentPage.Behaviors>
```

Запустите проект. Проверьте, что на странице предлагается выбор группы из списка FakeRepository.



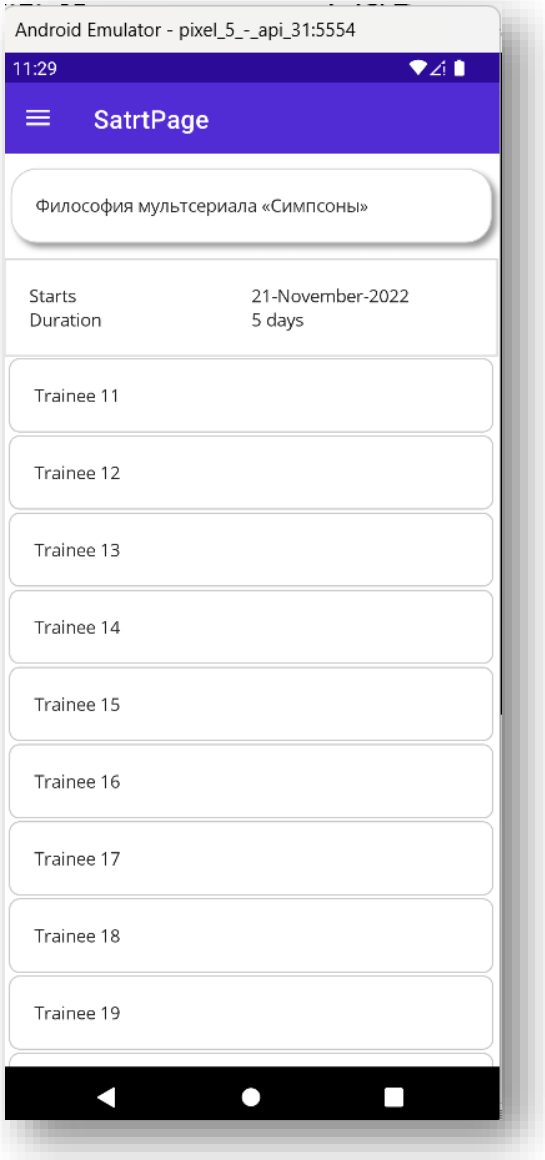
2.7.4. Вывод списка объектов в группе

Список объектов должен загружаться при выборе объекта в элементе «Picker». Свяжите событие SelectedIndexChanged с командой UpdateMembersList в модели представления. Используйте для этого EventToCommandBehavior из библиотеки CommunityToolkit.Maui (см. п. 2.7.3.).

Добавьте на страницу вывод информации о выбранной группе.

Добавьте на страницу элемент CollectionView для вывода списка объектов группы.

Запустите проект. Проверьте, что при выборе группы выводится информация о группе и список объектов группы:



2.7.5. Выделение объектов, свойство которых не удовлетворяет условию

Добавьте в проект папку ValueConverters.

Добавите в созданную папку конвертер, преобразующий значение обязательного свойства объекта (см. индивидуальное задание) в цвет.

Пример:

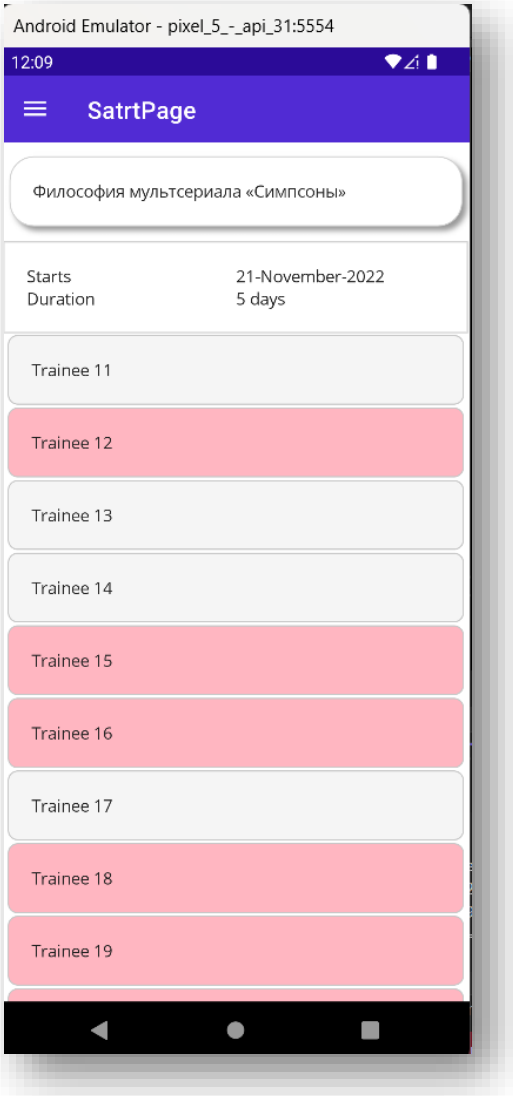
```
internal class RateToColorValueConverter : IValueConverter
{
     public object Convert(object value, Type targetType, object parameter,
                           CultureInfo culture)
        {
            if ((double)value < 6)
                return Colors.LightPink;
            return Colors.WhiteSmoke;
```

```
        }

        public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            throw new NotImplementedException();
        }
}
```

Зарегистрируйте конвертер в качестве ресурса страницы.

Привяжите цвет текста или фона в списке объектов к обязательному свойству объекта (см. индивидуальное задание). Используйте созданный конвертер значений.

Запустите проект. Проверьте, что в списке объектов группы некоторые объекты показаны другим цветом:



2.7.6. Вопросы для самопроверки

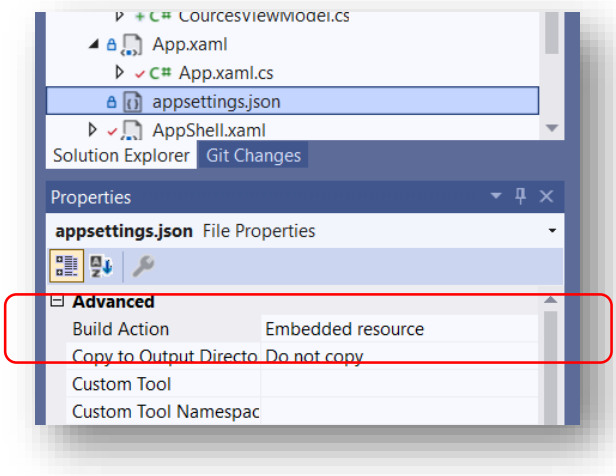
- Для чего используется интерфейс INotifyPropertyChanged? Пример реализации.
- Для чего используется класс ObservableCollection<T>?
- Как привязать значение XAML к свойству модели представления?
- Для чего используется интерфейс ICommand? Пример реализации?
- Что такое конвертер значений (Value Converter)?
- Что такое Binding Context представления?
- Как можно связать событие с командой?

2.8. Работа с контекстом базы данных

2.8.1. Подготовка проекта

Для сохранения данных предлагается использовать **SQLite**. Если вы не планируете запускать приложение на мобильных устройствах (или их эмуляторах), можете использовать любую другую базу данных. В этом случае вам нужно добавить в проект соответствующего поставщика данных

1) Загрузите в проект XXXX.UI NuGet пакет **Microsoft.Extensions.Configuration.Json**
2) Загрузите в проект XXXX.UI NuGet пакет **Microsoft.EntityFrameworkCore.Sqlite**
3) Добавьте в проект XXXX.UI файл **appsettings.json**. Этот файл будет использоваться для хранения конфигурационных данных, в частности, строки подключения к базе данных.
4) В свойствах файла укажите Build Action – Embedded resource



5) Настройте использование созданного файла конфигурации в файле MauiProgram:

```
string settingsStream = "CourceManager.UI.appsettings.json";

var builder = MauiApp.CreateBuilder();
          . . .

var a = Assembly.GetExecutingAssembly();
using var stream = a.GetManifestResourceStream(settingsStream);
builder.Configuration.AddJsonStream(stream);
```

2.8.2. Регистрация контекста БД в качестве сервиса

Примечание. Поскольку используется база данных SQLite, то база данных представляет собой файл, а строка подключения к БД содержит путь к файлу. Нужно предусмотреть различные пути в зависимости от платформы: Android или Windows (для iOs пример не приводится в виду отсутствия компьютера Mac для проверки правильности задания)

Укажите строку подключения в файле appsettings.json:

```
{
  "ConnectionStrings": {
       "SqliteConnection": " Data Source = {0}Cources.db"
  }
}
```

Опишите метод, добавляющий контекст базы данных в качестве сервиса. Для этого нужно создать объект DbContextOptions, который будет передаваться в конструктор контекста. Строку подключения получить из файла appsettings.json:

```
private static void AddDbContext(MauiAppBuilder builder)
{
     var connStr = builder.Configuration
                          .GetConnectionString("SqliteConnection");
     string dataDirectory=String.Empty;
#if ANDROID
        dataDirectory = FileSystem.AppDataDirectory+"/";
#endif
     connStr = String.Format(connStr, dataDirectory);

     var options = new DbContextOptionsBuilder<AppDbContext>()
               .UseSqlite(connStr)
               .Options;
     builder.Services.AddSingleton<AppDbContext>((s)=>
                                    new AppDbContext(options));
}
```

2.8.3. Заполнение базы данными

Опишите метод для записи в БД исходных данных:

```
public async static void SeedData(IServiceCollection services)
{
     using var provider = services.BuildServiceProvider();

     var unitOfWork = provider.GetService<IUnitOfWork>();
```

```
    await unitOfWork.RemoveDatbaseAsync();
    await unitOfWork.CreateDatabaseAsync();

    // Add cources
    IReadOnlyList<Cource> cources = new List<Cource>()
       {
          new Cource(){ Name="Вязание крючком", Duration=10,
                  StartDate=DateTime.Now.AddDays(5)},
          new Cource(){ Name="Философия мультсериала «Симпсоны»",
                  Duration=5, StartDate=DateTime.Now.AddDays(10)}
        };

    foreach (var cource in cources)
         await unitOfWork.CourseRepository.AddAsync(cource);
    await unitOfWork.SaveAllAsync();

    //Add trainees
    Random rand = new Random();
    int k = 1;
    foreach(var cource in cources)
       for (int j = 0; j < 10; j++)
          await unitOfWork.TraineeRepository.AddAsync(new Trainee()
          {
          Name = $"Trainee {k++}",
          Rate = rand.NextDouble() * 10,
          Cource=cource,
          DateOfBirth= DateTime.Now.AddDays(-rand.Next(20,40))});
    await unitOfWork.SaveAllAsync();
}
```

2.8.4. Завершающие действия

Последовательно вызовите методы `AddDbContext` и `SeedData`:

```
AddDbContext(builder);
SetupServices(builder.Services);
SeedData(builder.Services);
return builder.Build();
```

Примечание: в приведенном примере регистрация всех сервисов (кроме контекста базы данных) описана в отдельном методе SetupServices.

Измените регистрацию сервиса IUnitOfWork:

```
services.AddSingleton<IUnitOfWork, EfUnitOfWork>();
```

Запустите проект. Убедитесь, что приложение работает с базой данных.

2.8.5. Вопросы для самопроверки

- Как получить значение любой секции в файле appsettings.json?
- Для чего используются миграции в EntityFrameworkCore?
- Где находится контекст базы данных в трехслойной архитектуре?
- Где по умолчанию сохраняются файлы приложения в зависимости от платформы?
- Как используется интерфейс IPreferences и реализующий его класс Preferences?

2.9. Страница подробной информации об объекте

2.9.1. Подготовка страницы

Создайте файлы страницы подробной информации об объекте и модели представления этой страницы в соответствующих папках. (Далее по тексту страница «XxxDetails» и «XxxDetailsViewModel»)

Внедрите модель представления в конструктор класса страницы и назначьте ее в качестве BindingContext.

Зарегистрируйте страницу и ее модель представления в качестве **Transient** сервисов в классе MauiProgram.

Зарегистрируйте маршрут к странице в классе AppShell:

```
Routing.RegisterRoute(nameof(TraineeDetails),
                      typeof(TraineeDetails));
```

2.9.2. Переход на страницу подробной информации

Согласно заданию (см. п.1.1.) страница должна открываться при клике на объект в списке.

В модели представления стартовой страницы добавьте команду для перехода на страницу подробной информации:

```
[RelayCommand]
async void ShowDetails() => await GotoDetailsPage();

private async Task GotoDetailsPage()
{
    await Shell.Current.GoToAsync(nameof(TraineeDetails));
}
```

В разметке CollectionView стартовой страницы добавьте элемент распознавания жестов TapGestureRecognizer. Привяжите свойство Command к описанной выше команде.

Например:

```
<CollectionView.ItemTemplate>
    <DataTemplate x:DataType="entities:Trainee">
        <Frame Margin="5,0" BackgroundColor="{Binding Rate,
                                    Converter={StaticResource RateToColor} }">
            <Frame.GestureRecognizers>
                <TapGestureRecognizer Command="{Binding
                                    Source={RelativeSource AncestorType={x:Type
models:CourcesViewModel} },
                                    Path=ShowDetailsCommand}"/>
            </Frame.GestureRecognizers>
</CollectionView.ItemTemplate>
```

Запустите проект. Проверьте, что при клике на элемент в списке объектов открывается страница XxxDetails.

2.9.3. Отображение подробной информации об объекте

Для отображения подробной информации необходимо передать странице (модели представления) выбранный объект.

Для получения данных в модели представления XxxDetailsViewModel добавьте соответствующее свойство. Используйте атрибут [QueryProperty]

Для передачи данных:

В разметке TapGestureRecognizer на стартовой странице передайте параметр команды:

```
CommandParameter="{Binding}"
```

В модели представления стартовой странице получите объект в качестве параметра команды. Передайте объект при переходе на страницу XxxDetails:

```
async void ShowDetails(Trainee trainee) =>
                                await GotoDetailsPage(trainee);

private async Task GotoDetailsPage(Trainee trainee)
{
      IDictionary<string, object> parameters =
                                    new Dictionary<string, object>()
            {
                { "Trainee", trainee }
            };
            await
Shell.Current.GoToAsync(nameof(TraineeDetails),parameters);
}
```

Выполните разметку страницы XxxDetails.

Запустите проект. Проверьте, что выводится подробная информация о выбранном объекте в списке.

2.9.4. Вопросы для самопроверки

– Что такое «Словарь ресурсов» (Resource dictionary)? Как описать словарь ресурсов?
– Как использовать ресурс стиля в разметке XAML?

- Для чего используется CollectionView? Как настроить внешний вид CollectionView?
- Как зарегистрировать маршрут для навигации по стеку (два варианта)?
- Как перейти на другую страницу в коде модели представления?
- Как передать параметры странице при навигации по стеку?
- Чем отличается переход по адресу “MainPage” от “//MainPAge” ?

2.10. Реализация остального функционала задания

1) На стартовой странице разместите **MenuItems (для Windows), или ToolBarItems (для Android), или просто кнопки**, предоставляющие возможность:

- создание новой группы объектов
- создание нового объекта с добавлением в текущую группу

Создать соответствующие страницы

2) На странице XxxDetails разместите **MenuItems (для Windows), или ToolBarItems (для Android)**, или просто кнопки, предоставляющие возможность:

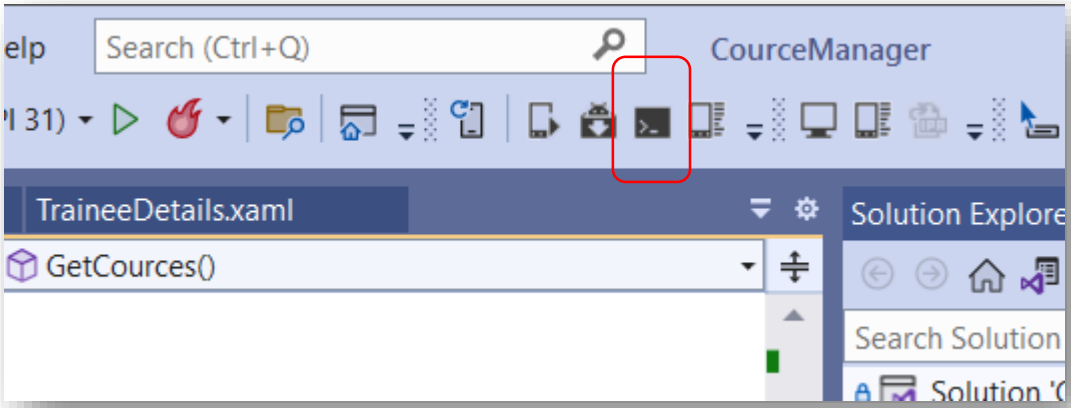
- Редактировать информацию об объекте
- Перемещать объект в другую группу
- Выбирать изображение для объекта и сохранять его в специально созданной папке «Images» (имя файла должно соответствовать Id объекта в БД). Это изображение затем должно выводиться на странице XxxDetails. Если файл отсутствует, то вывести изображение-«заместитель». Файл изображения-заместителя хранить в папке Resources/Images.
- При выводе изображения привязаться к Id, описать ValueConverter, который преобразует Id в изображение из файла.

Примечание 1: Предусмотреть автоматический выбор места для папки «Images» в зависимости от целевой платформы.

Примечание 2. Для работы с файловой системой настройте соответствующие разрешения (https://learn.microsoft.com/ru-ru/dotnet/maui/platform-integration/storage/file-picker?view=net-maui-6.0&tabs=android)

Примечание 3. При использовании эмулятора Android можно предварительно скопировать файлы изображений в папку Downloads. Для этого:

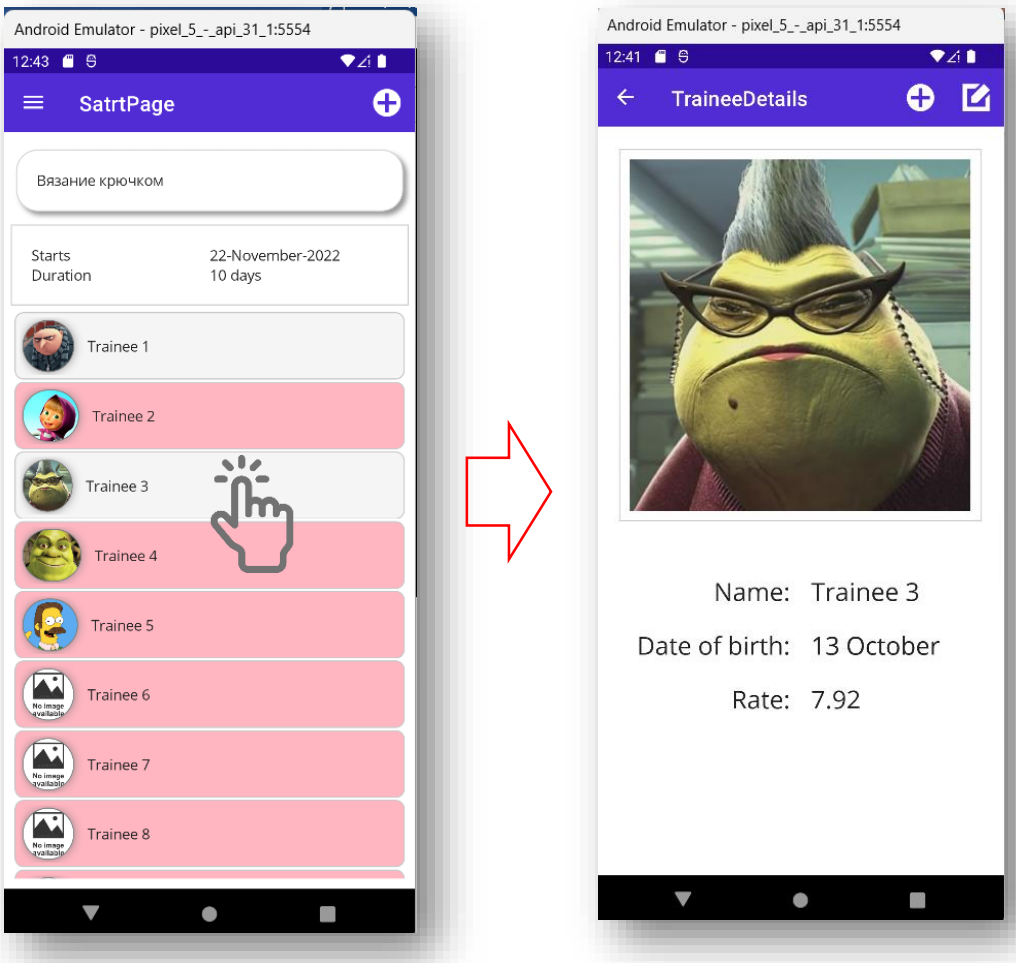
Запустите VisualStudio в режиме администратора.
Откройте «Android Adb Command Prompt»



Пример копирования:

xxxx>**adb push "D:\Images\xxx.jpg" "/storage/emulated/0/download"**

Пример вывода изображения на странице



Дополнительное задание (необязательно)

В навигации Shell добавьте иконки к элементам FlyOutItem. Предлагается использовать любой шрифт с иконками доступный для бесплатного скачивания (например, FontAwesome или IcoFont).

Использование шрифтов описано здесь: https://learn.microsoft.com/ru-ru/dotnet/maui/user-interface/fonts?view=net-maui-6.0#display-font-icons

Пример: