

Лабораторная работа №7-8

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	4
2.	Постановка задачи.....	5
2.1.	Требования к проекту	5
2.2.	Требования к архитектуре проекта	5
2.3.	Варианты заданий	5
3.	Выполнение работы	7
3.1.	Разработка функциональных требований	7
3.2.	Требования к ресурсам системы	7
3.3.	Разработка архитектуры проекта и проектирование доменной модели	8
3.3.1.	Создание проекта .Net MAUI.	8
3.3.2.	Описание доменной модели	11
3.3.3.	Вопросы для самопроверки	15
3.4.	Разработка слоя доступа к данным (Persistence)	16
3.4.1.	Подготовка проекта	16
3.4.2.	Разработка контекста БД	16
3.4.3.	Разработка репозитория и UnitOfWork	16
3.4.4.	Разработка тестовых репозитория и UnitOfWork.....	18
3.4.5.	Внедрение зависимостей	20
3.4.6.	Вопросы для самопроверки	21
3.5.	Разработка слоя Application	22
3.5.1.	Подготовка проекта	22
3.5.2.	Описание запросов	23
3.5.3.	Вопросы для самопроверки	24
3.6.	Подготовка проекта UI	25
3.6.1.	Предварительные действия	25
3.6.2.	Вопросы для самопроверки	25
3.7.	Разработка стартовой страницы приложения	26
3.7.1.	Подготовка страницы	26
3.7.2.	Создание модели представления.....	26

3.7.3. Вывод списка групп на странице	28
3.7.4. Регистрация сервисов.....	28
3.7.5. Вывод списка объектов в группе	30
3.7.6. Выделение объектов, свойство которых не удовлетворяет условию	31
3.7.7. Вопросы для самопроверки	33
3.8. Работа с контекстом базы данных.....	34
3.8.1. Подготовка проекта	34
3.8.2. Регистрация контекста БД в качестве сервиса	35
3.8.3. Заполнение базы данными.....	35
3.8.4. Завершающие действия.....	36
3.8.5. Вопросы для самопроверки	36
3.9. Страница подробной информации об объекте	37
3.9.1. Подготовка страницы	37
3.9.2. Переход на страницу подробной информации.....	37
3.9.3. Отображение подробной информации об объекте	38
3.9.4. Вопросы для самопроверки	39
3.10. Реализация остального функционала задания.....	40

1. Предварительная информация

Проанализируйте предметную область согласно варианту задания (см. 2.3).

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)

2. Постановка задачи

2.1. Требования к проекту

Требуется создать приложение, позволяющее:

1. Выводить список 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 должна открыться 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.

2.2. Требования к архитектуре проекта

- 1) Реализовать в проекте архитектуру Clean (чистую) 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>)
- 2) Для реализации шаблона CQRS использовать библиотеку MediatR (<https://github.com/jbogard/MediatR/wiki>)
- 3) На страницах приложения реализовать шаблон MVVM
- 4) Для получения объектов сервисов, репозиторий и т.д. использовать внедрение зависимостей

2.3. Варианты заданий

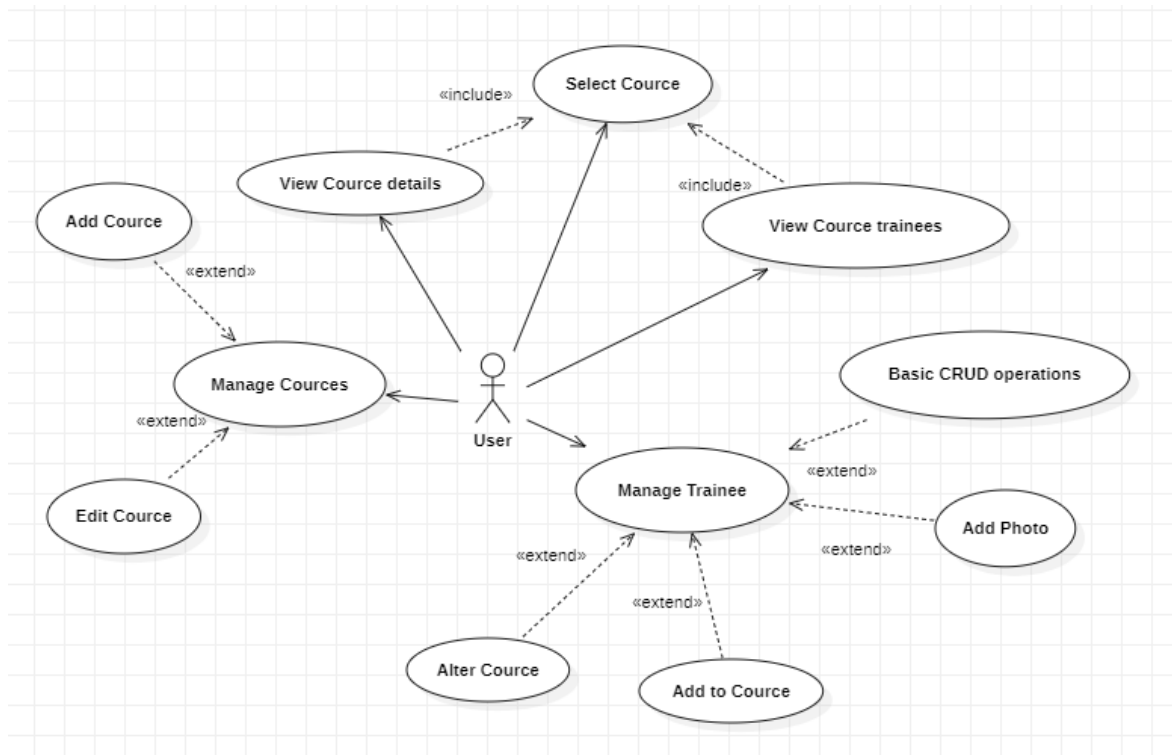
1. Спортивные команды – участники (обязательное свойство – количество очков у участника)
2. Исполнители – песни (обязательное свойство – место песни в чарте). Фото привязать к исполнителю.

3. Наборы суши – суши в наборе (обязательное свойство – количество готовых суши)
4. Супергерои – способности (обязательное свойство – степень развития способности в процентах от максимально возможной). Фото привязать к супергерою.
5. Туристические маршруты – достопримечательности (обязательное свойство – стоимость входного билета для осмотра достопримечательности).
6. Категория гостиничного номера – предоставляемый сервис (обязательное свойство – стоимость услуги).
7. Сотрудники – должностные обязанности (обязательное свойство – важность обязанности по 10-бальной шкале). Фото привязать к сотруднику.
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)
11. Больничная палата – пациенты (обязательное свойство – температура пациента).
12. Номинации конкурса – участники (обязательное свойство – количество баллов по итогам голосования).
13. Проект – задачи, выполняемые в рамках проекта (обязательное свойство – процент выполнения задачи). Фото привязать к проекту.
14. Позиции меню пиццерии – заказы клиентов (обязательное свойство – время, оставшееся до готовности заказа). Фото привязать к позиции меню.
15. Блогер – Сториз (обязательное свойство – количество лайков).
16. Марка автомобиля – объявления о продаже (обязательное свойство – цена).
17. Тематический зал музея – экспонаты (обязательное свойство – оценочная стоимость).
18. Виды процедур SPA салона – процедуры (обязательное свойство – продолжительность процедуры).
19. Виды транспорта транспортной компании – автомобили (обязательное свойство – дата прохождения техосмотра)
20. Брэнд косметики – изделия (обязательное свойство – объем, мл)

3. Выполнение работы

3.1. Разработка функциональных требований

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



3.2. 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

3.3. Разработка архитектуры проекта и проектирование доменной модели

3.3.1. Создание проекта .Net MAUI.

Создайте проект .Net MAUI.

Имя решения – номер группы и фамилия, например Ivanov_153501

В имени проекта напишите XXXX.UI, где XXXX – имя решения:

Configure your new project

.NET MAUI App C# Android iOS Mac Catalyst macOS MAUI Tizen Windows

Project name
XXXX.UI

Location
D:\Temp

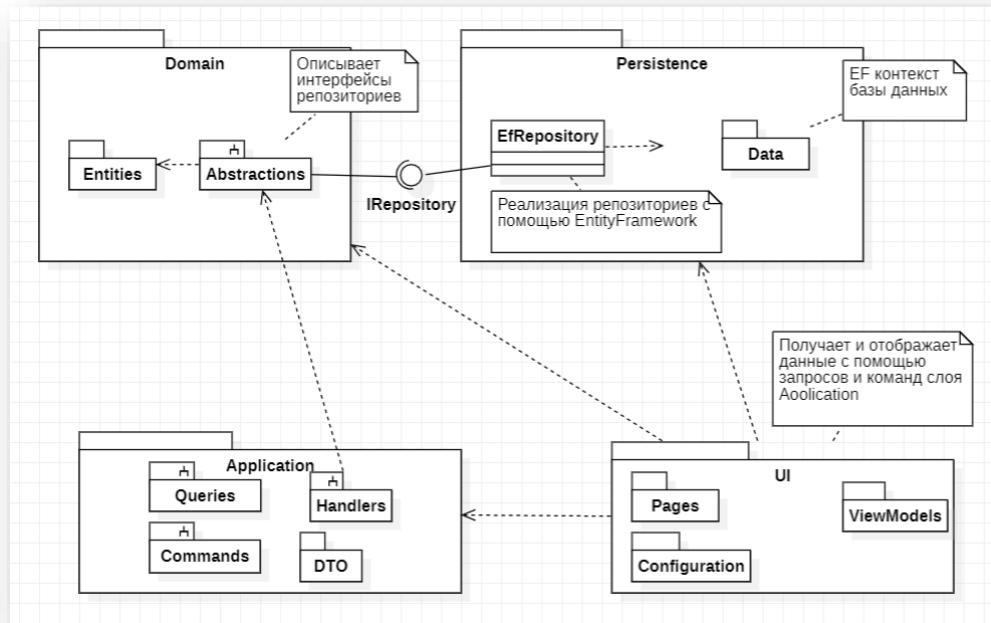
Solution
Create new solution

Solution name
XXXX

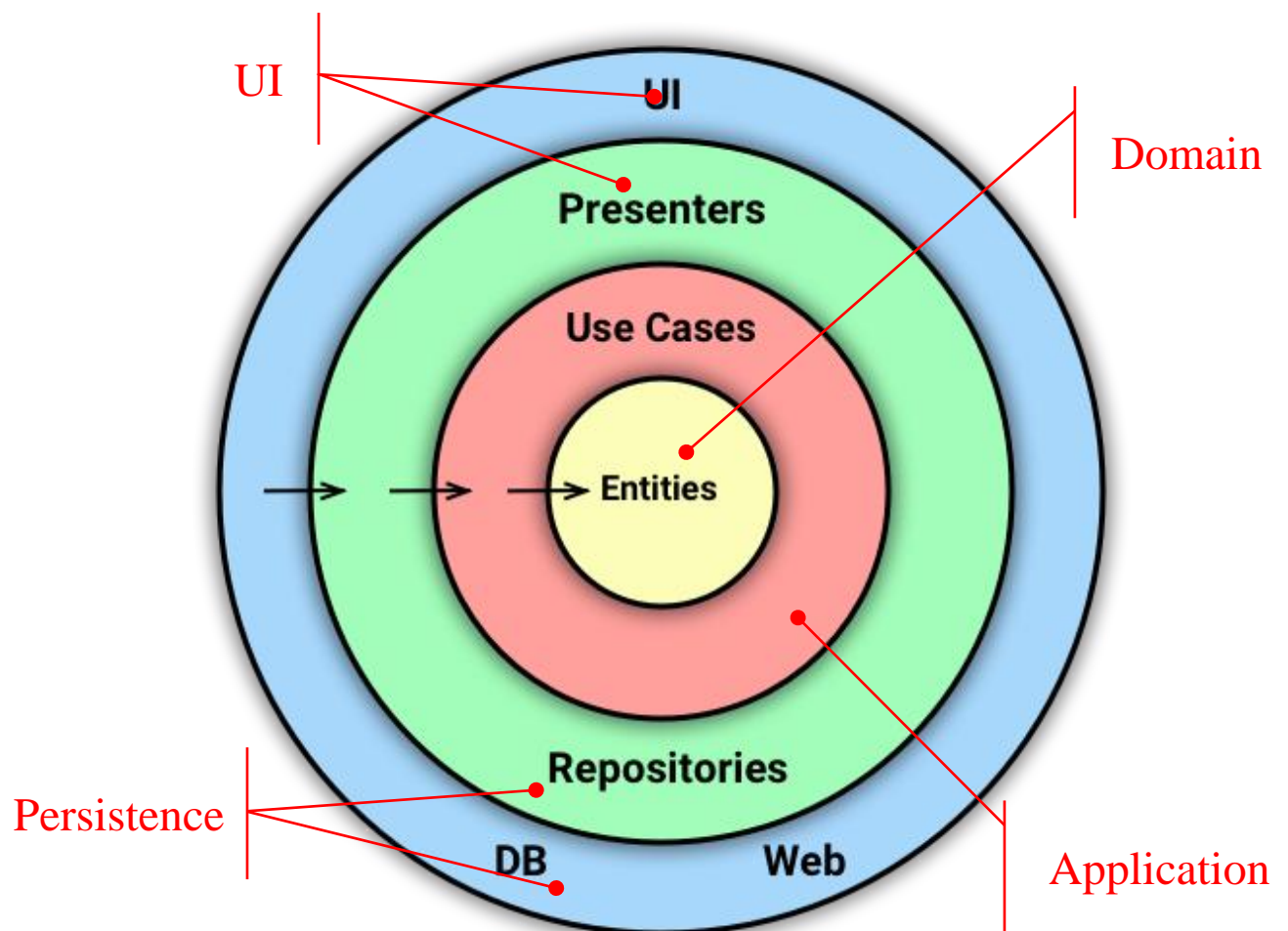
☐ Place solution and project in the same directory

Back Next

Требуется создать проект согласно схеме, приведенной ниже:



Для Clean архитектуры это :



Добавьте в решение соответствующие библиотеки классов:

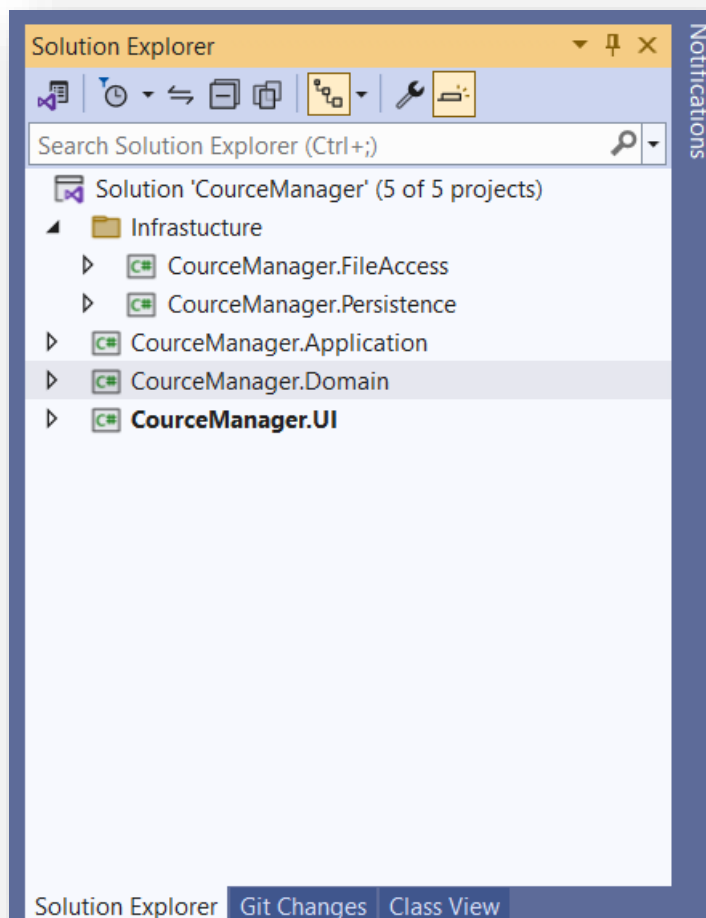
- XXXX.Domain
- XXXX.Application
- XXXX.Persistense

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

В библиотеке XXX.Application будут описаны команды и запросы библиотеки MediatR, необходимые для реализации функций вашего приложения.

В библиотеке XXXX.Persistense будут описаны классы репозитория для работы с базой данных (реализация интерфейсов репозитория, описанных в Domain) и контекст EntityFramework

Пример решения:



3.3.2. Описание доменной модели

Добавьте в проект XXXX.Domain папку Entities. Опишите в папке классы вашей предметной области. В классах опишите бизнес-методы, доступные для клиентов (шаблон DDD – Domain Driven Design). Пример:

```
public class Entity
{
    // Идентификатор сущности
    public int Id { get; set; }
}
```

Класс слушателя курсов:

```
/// <summary>
/// Слушатель курсов
/// </summary>
public class Trainee : Entity
{
    private Trainee()
    {
    }
    public Trainee(Person personalData, double? rate=0)
    {
        PersonalData = personalData;
        Rate = rate.Value;
    }
    // Имя и дата рождения описаны в отдельном классе
    public Person PersonalData { get; private set; }

    // Средний балл
    public double Rate { get; private set; }
    // Идентификатор курса
    public int? CourseId { get; private set; }
    /// <summary>
    /// Зачислить на курс
    /// </summary>
    /// <param name="courseId">Идентификатор курса</param>
    public void AddToCourse(int courseId)
    {
        if (courseId <= 0) return;
        CourseId = courseId;
    }
    /// <summary>
    /// Отчислить с курса
    /// </summary>
    public void LeaveCourse()
```

```

{
    CourseId = 0;
}
/// <summary>
/// Изменить средний балл
/// </summary>
/// <param name="rate"></param>
public void ChangeRate(double rate)
{
    if(rate<0 || rate>10) return;
    Rate = rate;
}
}

```

Персональные данные слушателя:

```
public sealed record Person(string Name, DateTime DateOfBirth);
```

Класс курса:

```

/// <summary>
/// Обучающий курс
/// </summary>
public class Course : Entity
{
    // Список слушателей курса
    private List<Trainee> _trainees = new();
    private Course()
    {
    }

    public Course(string name, DateTime commenceFrom, int duration)
    {
        Name= name;
        CommenceFrom= commenceFrom;
        Duration= duration;
    }
    // Название курса
    public string Name { get; set; }
    // Дата начала
    public DateTime CommenceFrom { get; private set; }
    // Продолжительность в днях
    public int Duration { get; private set; }
    // Публичное свойство для доступа к списку слушателей
    // (только для чтения)
    public IReadOnlyList<Trainee> Trainees {
        get => _trainees.AsReadOnly(); }
}

```

```
}
```

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

Пример:

```
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);
}

```

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

Например:

```

public interface IUnitOfWork
{
    IRepository<Course> CourseRepository { get; }
    IRepository<Trainee> TraineeRepository { get; }
    public Task SaveAllAsync();
}

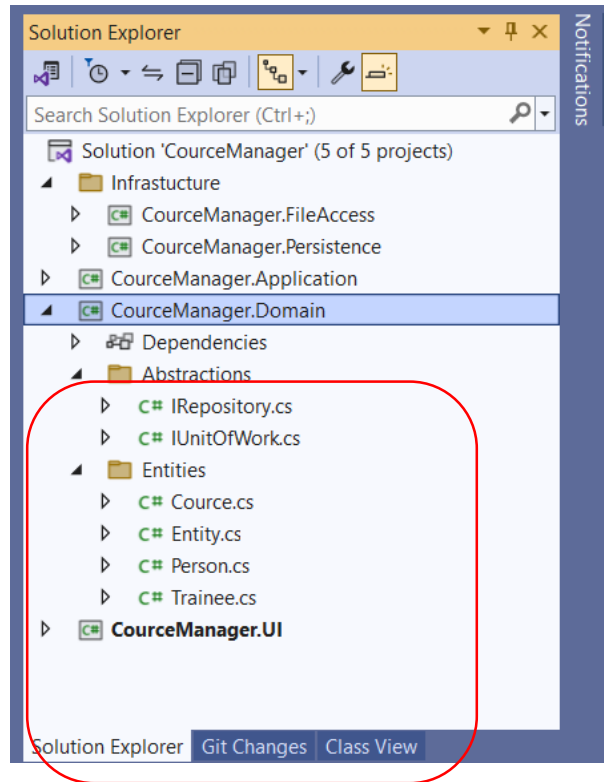
```

```

public Task DeleteDataBaseAsync();
public Task CreateDataBaseAsync();
}

```

Результат:



3.3.3. Вопросы для самопроверки

- 1) Что такое «Чистая» архитектура приложения? Ее преимущества перед трехслойной архитектурой.
- 2) Что такое «Принцип устойчивых зависимостей» компонентов приложения?
- 3) Как оцениваются метрики устойчивости?
- 4) Что такое UnitOfWork?
- 5) В каком слое чистой архитектуры находится контекст базы данных?
- 6) Шаблоны «Фабричный метод» и «Абстрактная фабрика»
- 7) Шаблон проектирования «Адаптер»
- 8) Какой механизм используется для изменения направления зависимости компонентов в приложении?
- 9) Что означает «S» в аббревиатуре SOLID?
- 10) **REP**: Reuse/Release Equivalence Principle — принцип эквивалентности повторного использования и выпусков;

3.4. Разработка слоя доступа к данным (Persistence)

3.4.1. Подготовка проекта

Добавьте в проект XXXX.Persistence ссылку на проект XXXX.Domain
 Добавьте в проект XXXX.Persistence NuGet пакет
 Microsoft.EntityFrameworkCore
 Добавьте в проект класс GlobalUsings, в котором укажите глобальное
 подключение пространств имен:

```
global using SourceManager.Domain.Entities;
global using SourceManager.Domain.Abstractions;
```

3.4.2. Разработка контекста БД

Добавьте в проект XXXX.Persistence папку Data.
 В папке Data создайте контекст базы данных (можно использовать имя
 AppDbContext). Сделайте контекст **private**.
 Конструктор контекста должен принимать параметр
 DbContextOptions<AppDbContext>
 В конструкторе контекста создайте базу данных:

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

Класс **Trainee** содержит свойство **PersonalData**, описанные в классе
Person. Для того, чтобы в базе данных не создавалась отдельная таблица Persons,
 укажем это в методе OnModelCreating:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Trainee>()
        .OwnsOne<Person>(t => t.PersonalData);
}
```

3.4.3. Разработка репозитория 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<Course>> _courseRepository;
    private readonly Lazy<IRepository<Trainee>> _traineeRepository;

    public EfUnitOfWork(AppDbContext context)
    {
        _context = context;
        _courseRepository = new Lazy<IRepository<Course>>(() =>
            new EfRepository<Course>(context));
        _traineeRepository = new Lazy<IRepository<Trainee>>(() =>
            new EfRepository<Trainee>(context));
    }

    public IRepository<Course> CourseRepository
        => _courseRepository.Value;
    public IRepository<Trainee> TraineeRepository
        => _traineeRepository.Value;

    public async Task CreateDataBaseAsync() =>
        await _context.Database.EnsureCreatedAsync();
    public async Task DeleteDataBaseAsync() =>
        await _context.Database.EnsureDeletedAsync();
    public async Task SaveAllAsync() =>
        await _context.SaveChangesAsync();
}

```

3.4.4. 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<Course>
{
    List<Course> _courses;
    public FakeCourseRepository()
    {
        _courses = new List<Course>();

        var course = new Course("Вязание крючком",
            DateTime.Now.AddDays(5), 10);
        course.Id = 1;
        _courses.Add(course);
        course = new Course("Философия мультсериала «Симпсоны»",
            DateTime.Now.AddDays(10), 5);
        course.Id = 2;
    }
}

```

```

        _courses.Add(course);
    }

    . . .

    public async Task<IReadOnlyList<Course>> ListAllAsync(CancellationToken
cancellationToken = default)
    {
        return await Task.Run(() => _courses);
    }
}

```

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()
    {
        int k = 1;
        for (int i = 1; i <= 2; i++)
            for (int j = 0; j < 10; j++)
            {
                var trainee = new Trainee(
                    new Person($"Trainee {k++}",
                        DateTime.Now
                            .AddYears(-Random.Shared.Next(30))),
                    Random.Shared.NextDouble() * 10);
                trainee.AddToCourse(i);
                _list.Add(trainee);
            }
    }

    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.

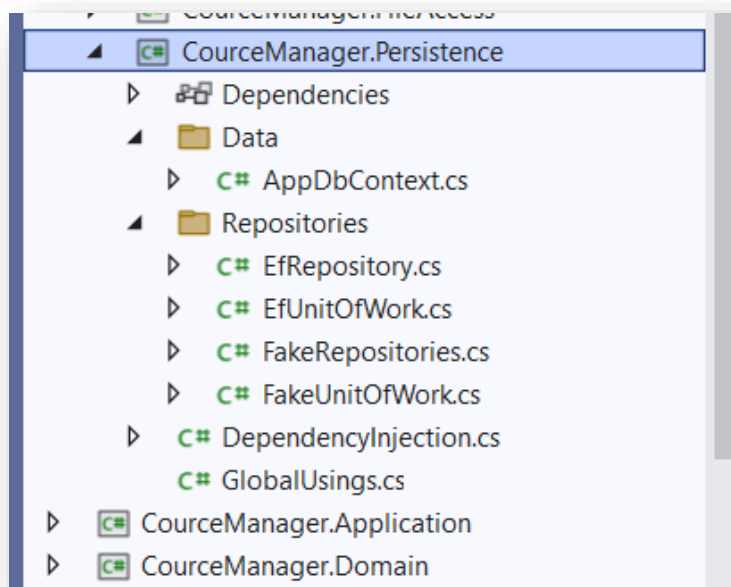
3.4.5. Внедрение зависимостей

Для реализации в проекте внедрения зависимостей добавьте в проект XXXX.Persistence класс DependencyInjection, в котором опишите метод расширения для IServiceCollection. Метод должен зарегистрировать в качестве сервисов контекст базы данных и FakeUnitOfWork:

```
public static class DependencyInjection
{
    public static IServiceCollection AddPersistence(this IServiceCollection
services)
    {
        services.AddSingleton<IUnitOfWork, FakeUnitOfWork>();
        return services;
    }
    public static IServiceCollection AddPersistence(
        this IServiceCollection services,
        DbContextOptions options)
    {
        services.AddPersistence()
            .AddSingleton<AppDbContext>(
                new AppDbContext((DbContextOptions<AppDbContext>)options));

        return services;
    }
}
```

Окончательный вид проекта:



3.4.6. 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?
- Что такое шаблон проектирования «Построитель» (Builder)?
- Для чего используются миграции EF Core?
- Чем отличается результат применения `Expression<Func<T, bool>>` от `Func<T, bool>`?

3.5. Разработка слоя Application

3.5.1. Подготовка проекта

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

Добавьте в проект NuGet пакет MediatR.

Добавьте в проект класс DependencyInjection, в котором опишите расширяющий метод для регистрации сервисов Application:

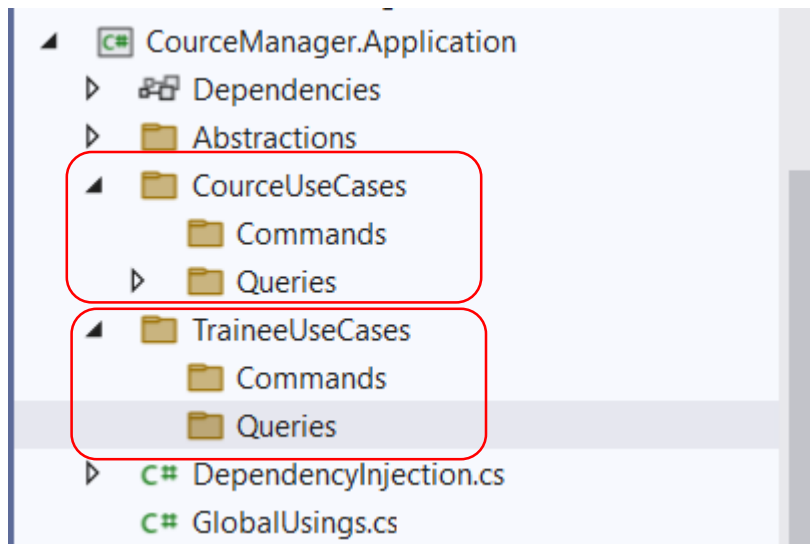
```
public static class DependencyInjection
{
    public static IServiceCollection AddApplication(this
IServiceCollection services)
    {
        services.AddMediatR(conf =>
conf.RegisterServicesFromAssembly(typeof(DependencyInjection)
.Assembly));
        return services;
    }
}
```

Добавьте в проект файл GlobalUsings.cs:

```
global using MediatR;
global using XXXX.Domain.Entities;
global using XXXX.Domain.Abstractions;
```

Добавьте в проект папки для описания вариантов использования (Use-Case) приложения. Предусмотрите отдельные папки для запросов и команд (используйте библиотеку MediatR).

Пример приведен на рисунке ниже.



3.5.2. Описание запросов

Использование библиотеки MediatR описано здесь: <https://github.com/jbogard/MediatR> и здесь: <https://github.com/jbogard/MediatR/wiki>

Опишите запросы для получения списка групп объектов и списка объектов выбранной группы.

Список объектов можно получить двумя способами – получить информацию о группе, включая список объектов группы (в приведенном примере запрос будет размещаться в папке `CourseUseCases/Queries`) или получить список объектов, отфильтрованных по Id группы (в приведенном примере запрос будет размещаться в папке `TraineeUseCases/Queries`). **Выберите один из способов.**

Реализуйте обработчики описанных запросов. Внедрите в обработчики `IUnitOfWork`.

Пример:

Запрос на получение списка слушателей курса:

```
public sealed record GetTraineesByGroupRequest(string Id)
    : IRequest<IEnumerable<Trainee>>
{
}
```

Обработчик запроса:

```
internal class GetTraineesByGroupRequestHandler(IUnitOfWork
unitOfWork) :
    IRequestHandler<GetTraineesByGroupRequest, IEnumerable<Trainee>>
```

```
{  
  
    public async Task<IEnumerable<Trainee>> Handle(  
        GetTraineesByGroupRequest request,  
        CancellationToken cancellationToken)  
    {  
        return await unitOfWork.TraineeRepository  
            .ListAsync(t => t.CourseId.Equals(request.Id),  
cancellationToken);  
    }  
}
```

3.5.3. Вопросы для самопроверки

- Что такое операции CRUD?
- Что такое CQRS?

3.6. Подготовка проекта UI

3.6.1. 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 приложения
- 6) Добавьте в проект XXXX.UI папку ViewModels. В ней будут описаны модели страниц.
- 7) В классе Program зарегистрируйте сервисы Application и Persistence:

```
builder.Services
    .AddApplication()
    .AddPersistence();
```

- 8) Добавьте класс GlobalUsings:

```
global using MediatR;
global using SourceManager.Domain.Entities;
```

3.6.2. Вопросы для самопроверки

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

3.7. Разработка стартовой страницы приложения

3.7.1. 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 будет использоваться имя Sources.

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="CourseManager.UI.AppShell"
  xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:local="clr-namespace:CourseManager.UI"
  xmlns:pages="clr-namespace:CourseManager.UI.Pages">

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

    <ShellContent
      Title="Sources list"
      ContentTemplate="{DataTemplate pages:Courses}"
      Route="Courses" />

</Shell>
```

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

3.7.2. Создание модели представления

Добавьте в папку ViewModels класс 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 CoursesViewModel : ObservableObject
{
    private readonly IMediator _mediator;
    public CoursesViewModel(IMediator mediator)
    {
        _mediator = mediator;

        public ObservableCollection<Course> Courses { get; set; } = new();
        public ObservableCollection<Trainee> Trainees { get; set; } = new();

        // Выбранный в списке курс
        [ObservableProperty]
        Course selectedCourse;

        // Команда обновления списка курсов
        [RelayCommand]
        async Task UpdateGroupList() => await GetCourses();
        // Команда обновления списка слушателей курса
        [RelayCommand]
        async Task UpdateMembersList() => await GetTrainees();

        public async Task GetCourses()
        {
            var courses = await _mediator.Send(new GetAllCoursesRequest());
            await MainThread.InvokeOnMainThreadAsync(() =>
            {
                Courses.Clear();
                foreach (var course in courses)
                    Courses.Add(course);
            });
        }

        public async Task GetTrainees()
        {
            var trainees = await _mediator.Send(new
GetTraineesByGroupRequest(SelectedCourse.Id));
            await MainThread.InvokeOnMainThreadAsync(() =>
            {
```

```

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

```

3.7.3. Вывод списка групп на странице

Примечание: выбор шаблона компоновки (Grid, StackLayout 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:CourseManager.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:CourseManager.UI.ViewModels"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:DataType="models:CourcesViewModel"
    x:Class="CourseManager.UI.Pages.Cources"
    Title="SatrtPage">
    <ContentPage.Behaviors>
        <toolkit:EventToCommandBehavior EventName="Loaded"
            Command="{Binding UpdateGroupListCommand }"/>
    </ContentPage.Behaviors>

```

3.7.4. Регистрация сервисов

Опишите расширяющие методы к IServiceCollection для регистрации страниц и моделей представления:

```

static IServiceCollection RegisterPages(

```

```
        this IServiceCollection services)
    {
        services.AddTransient<Cources>();

        return services;
    }

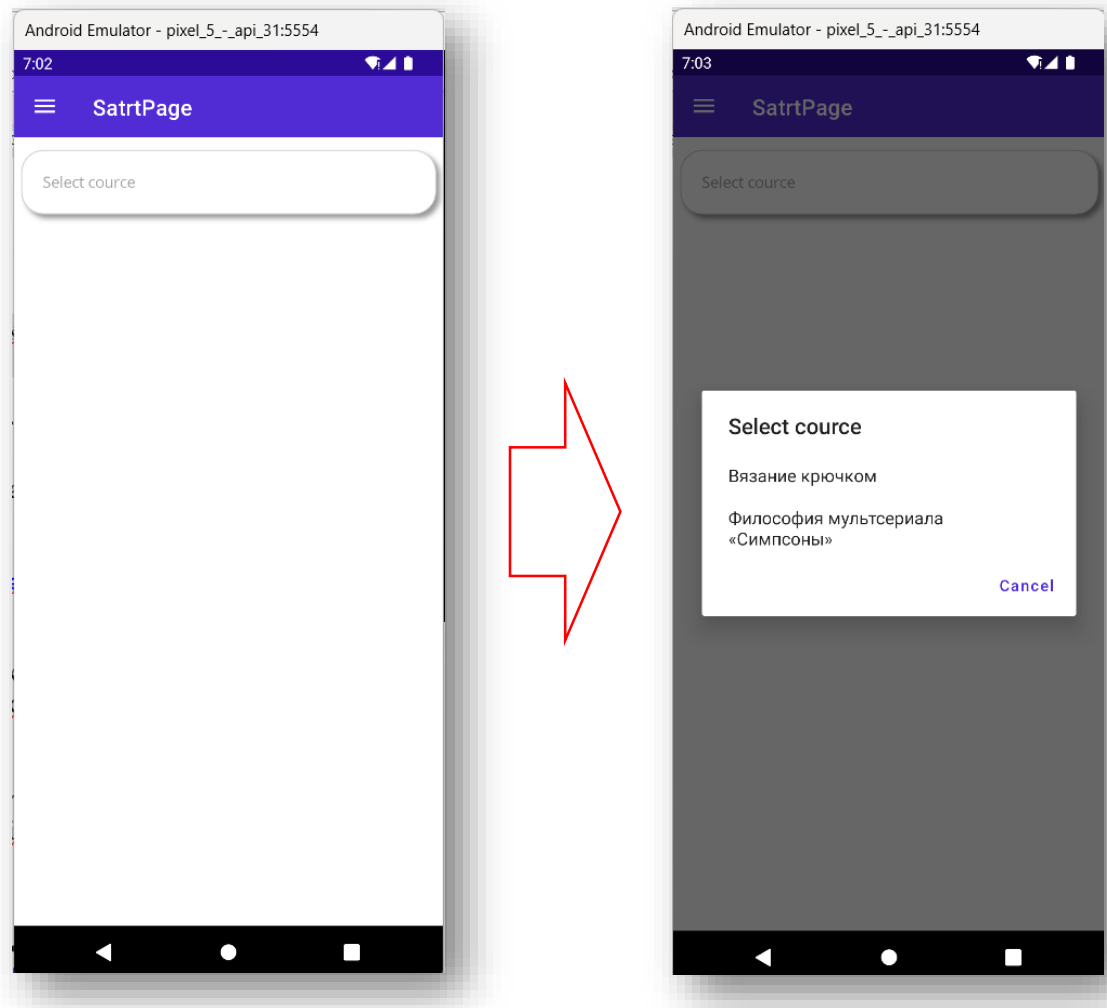
    static IServiceCollection RegisterViewModels(
        this IServiceCollection services)
    {
        services.AddTransient<CourcesViewModel>();

        return services;
    }
```

Вызовите созданные методы в классе MauiProgram:

```
builder.Services
    .AddApplication()
    .AddPersistence()
    .RegisterPages()
    .RegisterViewModels();
```

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



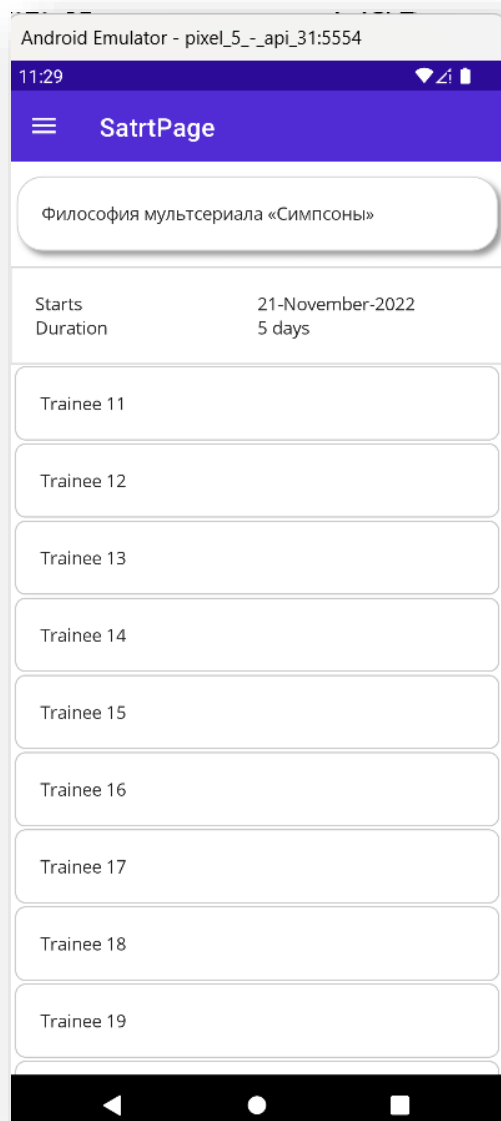
3.7.5. Вывод списка объектов в группе

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:



3.7.6. Выделение объектов, свойство которых не удовлетворяет условию

Добавьте в проект папку ValueConverters.

Добавьте 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;
    }

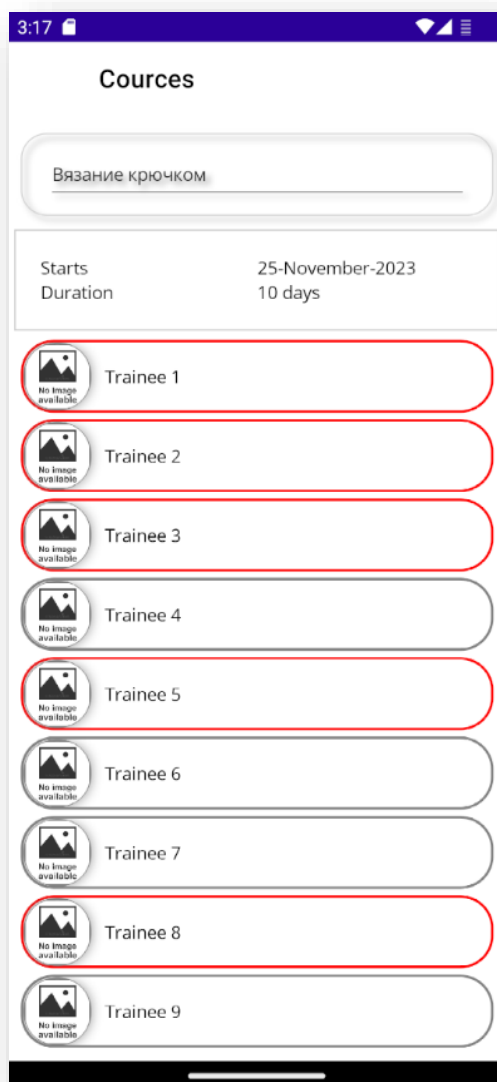
    public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
    {
        throw new NotImplementedException();
    }
}

```

Зарегистрируйте конвертер в качестве ресурса страницы.

Привяжите цвет текста или границы 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:



3.7.7. 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?

3.8. Работа с контекстом базы данных

3.8.1. Подготовка проекта

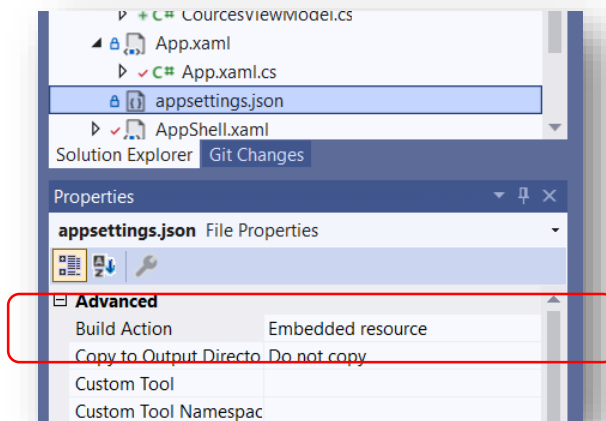
Для сохранения данных предлагается использовать **SQLite**. Если вы не планируете запускать приложение на мобильных устройствах (или их эмуляторах), можете использовать любую другую базу данных. В этом случае вам нужно добавить в проект соответствующего поставщика данных

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);
```

3.8.2. 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:

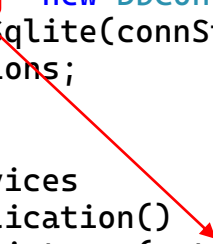
```
{
  "ConnectionStrings": {
    "SqliteConnection": "Data Source = {0}Sources.db"
  }
}
```

Создайте объект DbContextOptions, который будет передаваться в конструктор контекста. Строку подключения получить из файла appsettings.json:

```
var connStr = builder.Configuration
    .GetConnectionString("SqliteConnection");
string dataDirectory = FileSystem.Current.AppDataDirectory + "/";
connStr = String.Format(connStr, dataDirectory);

var options = new DbContextOptionsBuilder<AppDbContext>()
    .UseSqlite(connStr)
    .Options;

builder.Services
    .AddApplication()
    .AddPersistence(options)
    .RegisterPages()
    .RegisterViewModels();
```



3.8.3. Заполнение базы данными

В проекте Application создайте класс DbInitializer со статическим методом Initialize для записи в БД исходных данных. Для получения IUnitOfWork метод должен принимать в качестве параметра объект IServiceProvider. Пример:

```
public static class DbInitializer
{
    public static async Task Initialize(IServiceProvider services)
    {
        var unitOfWork = services.GetRequiredService<IUnitOfWork>();
    }
}
```

```

    // Создать базу данных заново
    await unitOfWork.DeleteDataBaseAsync();
    await unitOfWork.CreateDataBaseAsync();

    // Добавить курсы

    . . .

    await unitOfWork.SaveAllAsync();

    // добавить слушателей
    // НЕ забудьте записать слушателя на курс
    . . .
    await unitOfWork.SaveAllAsync();
}
}

```

3.8.4. Завершающие действия

В классе MauiProgram вызовите метод заполнения базы данными:

```

DbInitializer
.Initialize(builder.Services.BuildServiceProvider())
.Wait();

```

Измените регистрацию сервиса IUnitOfWork (файл DependencyInjection проекта Persistence):

```

services.AddSingleton<IUnitOfWork, EfUnitOfWork>();

```

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

3.8.5. 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?

3.9. Страница подробной информации об объекте

3.9.1. 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.

```
static IServiceCollection RegisterPages(
    this IServiceCollection services)
{
    services
        .AddSingleton<Courses>()
        .AddTransient<TraineeDetails>();
    return services;
}

static IServiceCollection RegisterViewModels(
    this IServiceCollection services)
{
    services
        .AddSingleton<CoursesViewModel>()
        .AddTransient<TraineeDetailsViewModel>();
    return services;
}
```

Зарегистрируйте маршрут к странице в классе AppShell:

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

3.9.2. 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">
    <Border Stroke="{Binding Rate, Converter={StaticResource RateToColor} }"
      Margin="0,1"
      StrokeThickness="2"
      StrokeShape="RoundRectangle 25,25,25,25">
      <HorizontalStackLayout>
        <HorizontalStackLayout.GestureRecognizers>
          <TapGestureRecognizer Command="{Binding
            Source={RelativeSource
              AncestorType={x:Type models:CourcesViewModel} },
            Path=ShowDetailsCommand}"
            CommandParameter="{Binding}"/>
        </HorizontalStackLayout.GestureRecognizers>
      </Border Stroke="Gray" StrokeThickness="2">
```

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

3.9.3. 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.

3.9.4. 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” ?

3.10. Реализация остального функционала задания

1) В проекте Application опишите команды и их обработчики (в папках Commands) для реализации функций приложения.

Пример команды добавления слушателя:

```
public sealed record AddTraineeCommand(
    string Name,
    DateTime DateOfBirth,
    int? GroupId) : IRequest<Trainee>
{ }
```

Пример обработчика команды добавления слушателя:

```
internal class AddTraineeCommandHandler :
    IRequestHandler<AddTraineeCommand, Trainee>
{
    private readonly IUnitOfWork _unitOfWork;

    public AddTraineeCommandHandler(IUnitOfWork unitOfWork)
    {
        _unitOfWork = unitOfWork;
    }

    public async Task<Trainee> Handle(
        AddTraineeCommand request,
        CancellationToken cancellationToken)
    {
        Trainee newTrainee = new Trainee(new Person(
            request.Name,
            request.DateOfBirth));

        if (request.GroupId.HasValue)
        {
            newTrainee.AddToCourse(request.GroupId.Value);
        }

        await _unitOfWork.TraineeRepository.AddAsync(
            newTrainee,
            cancellationToken);

        return newTrainee;
    }
}
```

2) 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

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

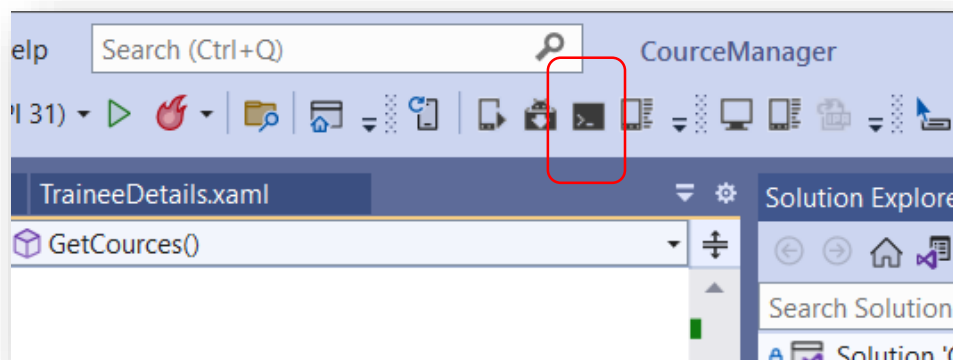
- Редактировать информацию об объекте
- Перемещать объект в другую группу
- Выбирать изображение для объекта и сохранять его в специально созданной папке «Images» (имя файла должно соответствовать Id объекта в БД). Это изображение затем должно выводиться на странице XxxDetails. Если файл отсутствует, 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:

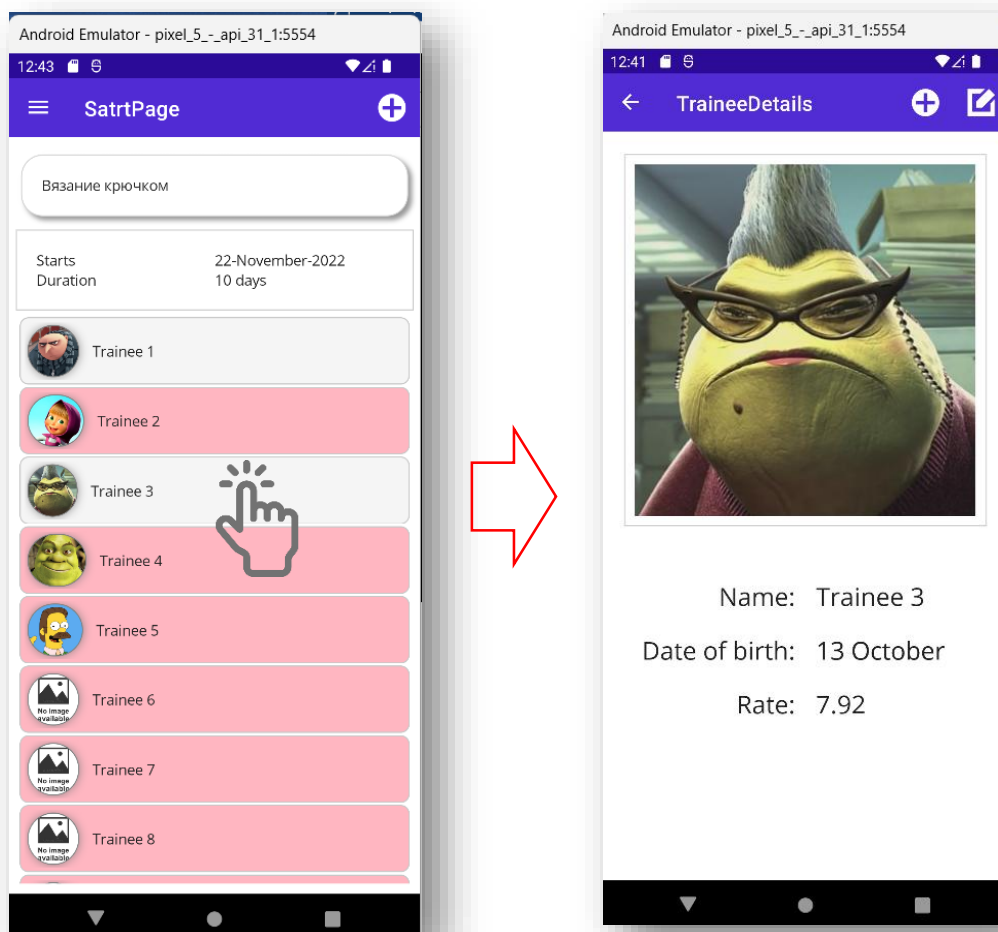
Запустите VisualStudio в режиме администратора.
Откройте «Android Adb Command Prompt»



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:

