**Содержание**

1. **Проектирование предметной области**
   1. Описание предметной области
   2. Создание диаграммы классов
   3. Создание GitHub-репозитории проекта
2. **Введение в ASP.NET Web API**
   1. Организация структуры проекта по Clean Architecture
   2. Создание моделей приложения
3. **Работа с базой данных**
   1. Реализация архитектурного паттерна репозиторий
   2. Использование EntityFramework для работы с базой данных
   3. Миграция данных
4. **Проектирование предметной области**
   1. Описание предметной области

Социальная сеть – это онлайн-платформа для общения, обмена информацией и создания связей между людьми. Пользователи создают профили, общаются друг с другом, делятся контентом, создают группы и сообщества, а также находят людей по интересам. Существует множество типов социальных сетей, от универсальных до тематических.

* 1. Создание диаграммы классов

Создания диаграммы классов системы управления социальной сетью был начат с определения основных классов, которые будут представлять различные сущности системы. Пример классов:

**Комментарий:** Этот класс будет отражать меню ресторана и включать в себя информацию о блюдах, их ценах, категориях и т. д.

**Друг:** Этот класс представляет собой друга пользователя. Он содержит информацию о том, кто является другом, и когда была установлена дружба.

**Группа:** Этот класс представляет собой группу пользователей. Он содержит информацию о названии группы, описании, создателе и членах группы.

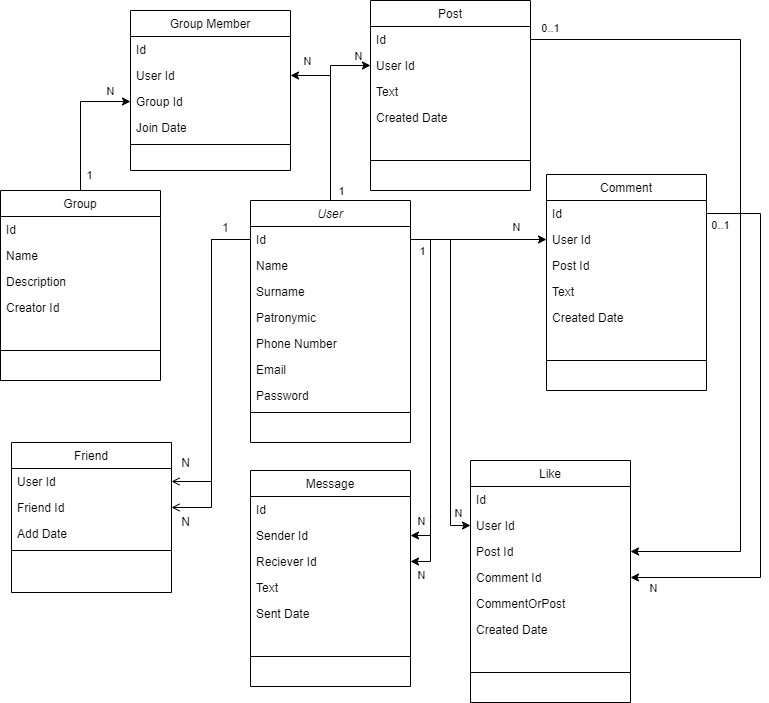
**Участник группы:** Этот класс представляет собой связь между пользователем и группой. Он содержит информацию о том, кто является членом группы и когда он присоединился к ней.

**Понравившийся:** Этот класс представляет собой лайк к публикации или сообщению. Он содержит информацию о том, кто поставил лайк и когда.

**Сообщение:** Этот класс представляет собой сообщение, отправленное одним пользователем другому. Он содержит информацию о том, кто отправил сообщение, кому оно было отправлено, текст сообщения и время отправки.

**Публикация:** Этот класс представляет собой публикацию в социальной сети. Он содержит информацию о том, кто ее опубликовал, когда она была опубликована, текст публикации, а также изображения, видео и другие вложения.

**Пользователь:** Этот класс представляет собой пользователя социальной сети. Он содержит информацию о имени пользователя, пароле, адресе электронной почты и других данных о пользователе.



**Рис 1. Диаграмма классов**

* 1. Создание GitHub-репозитории проекта

Краткий процесс создания GitHub-репозитория для проекта под названием «SocialSite».

Создание нового репозитория: после входа в аккаунт, была нажата на кнопку "New" (Новый) в верхнем правом углу страницы репозиториев.

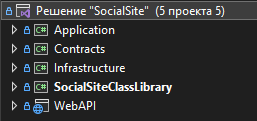
Заполнение информации о репозитории: на странице создания нового репозитория была указана название проекта ("SocialSite "), описание проекта (например, "Система управления рестораном"), было выбрано уровень доступа (**общедоступный** или приватный), а также другие настройки по нашему усмотрению.

Создание репозитория: была нажата на кнопку "Create repository" (Создать репозиторий) для завершения создания репозитория. Клонирование репозитория: после создания репозитория скопирован его URL. Затем открыв терминал на вашем компьютере, перейден в папку, где сохранен проект, и было выполнено команда `git clone <https://github.com/MirzoDadokhonov/SocialSite.git >` для клонирования репозитория в наш компьютер.

1. **Введение в ASP.NET Web API**
   1. Организация структуры проекта по Clean Architecture

Clean Architecture - это методология проектирования программного обеспечения, предложенная Робертом Мартином (также известным как Uncle Bob). Она призвана обеспечить высокую степень разделения ответственностей и улучшить расширяемость, тестируемость и общую чистоту кода.

Применение этой методологии можно увидеть в следующей картинке, в структуре проекта «Социальная сеть».



**Рис 2. Организация структуры проекта по Clean Architecture**

По рисунку можно понять, как проект разделён на слои: Presentation (WebAPI), Application (одноимённый проект), Infrastructure (одноимённый проект), Contracts (одноимённый проект c моделями dto) и Domain (SocialSiteClassLibrary).

* 1. Создание моделей приложения

Были созданы все сущности исходя из схема диаграмма классов (рис. 1).

**Листинг 1. Код на языке программирования с# (User Entity).**

public class User : BaseEntity

{

public string Name { get; set; }

public string Surname { get; set; }

public string Nickname { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string PhoneNumber { get; set; }

}

Аналогичным образом можно рассмотреть и другие классы. Далее будет рассмотрен один пример создания класса – перечисления значений.

**Листинг 2. Код на языке программирования с# (BanReason Enum).**

public class Message : CancellableEntity

{

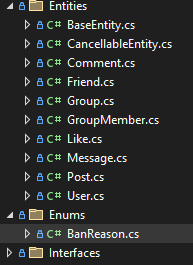
BadContent,

BadWordsUsed,

ContentCannotBeUnderstood

}

В итоге все классы были готовы к использованию. Далее можно будет сделать миграцию данных, а также запустить проект API.



**Рис 3. Cтруктура моделей проекта**

1. **Работа с базой данных**
   1. Реализация архитектурного паттерна репозиторий

Архитектурный паттерн репозиторий является часто используемым в приложениях, особенно там, где требуется взаимодействие с хранилищем данных, таким как база данных. Вот как можно реализовать этот паттерн:

**Определение интерфейса репозитория**:

Создайте интерфейс, который будет определять методы для основных операций доступа к данным, таких как добавление, чтение, обновление и удаление (CRUD).

**Листинг 3. Определение интерфейса репозитория:**

public interface IBaseService<TEntity> where TEntity : BaseEntity

{

Task<TEntity> GetAsync(Guid id, CancellationToken token = default);

Task<IEnumerable<TEntity>> GetAllAsync(CancellationToken token = default);

Task<TEntity> InsertAsync(TEntity entity, CancellationToken token = default);

Task<TEntity> UpdateAsync(TEntity entity, CancellationToken token = default);

Task<bool> DeleteAsync(TEntity entity, CancellationToken token = default);}

**Листинг 4. Создание конкретной реализации репозитория**:

public class BaseRepository<T> : IBaseRepository<T> where T : BaseEntity

{

private readonly ApplicationDbContext \_context;

private readonly DbSet<T> \_dbSet;

public BaseRepository(ApplicationDbContext context)

{

\_context = context;

\_dbSet = context.Set<T>();

}

public async Task<T> InsertAsync(T entity, CancellationToken token = default)

{

await \_dbSet.AddAsync(entity, token);

await \_context.SaveChangesAsync(token);

return entity;

}

public async Task<bool> DeleteAsync(T entity, CancellationToken token = default)

{

\_dbSet.Remove(entity);

return await \_context.SaveChangesAsync(token) > 0;

}

public async Task<IEnumerable<T>> GetAllAsync(CancellationToken token = default)

{

return await \_dbSet.ToListAsync(token);

}

public async Task<T> GetAsync(Guid id, CancellationToken token = default)

{

return await \_dbSet.FindAsync(id, token);

}

public async Task<T> UpdateAsync(T entity, CancellationToken token = default)

{

\_dbSet.Update(entity);

await \_context.SaveChangesAsync(token);

return entity;

}

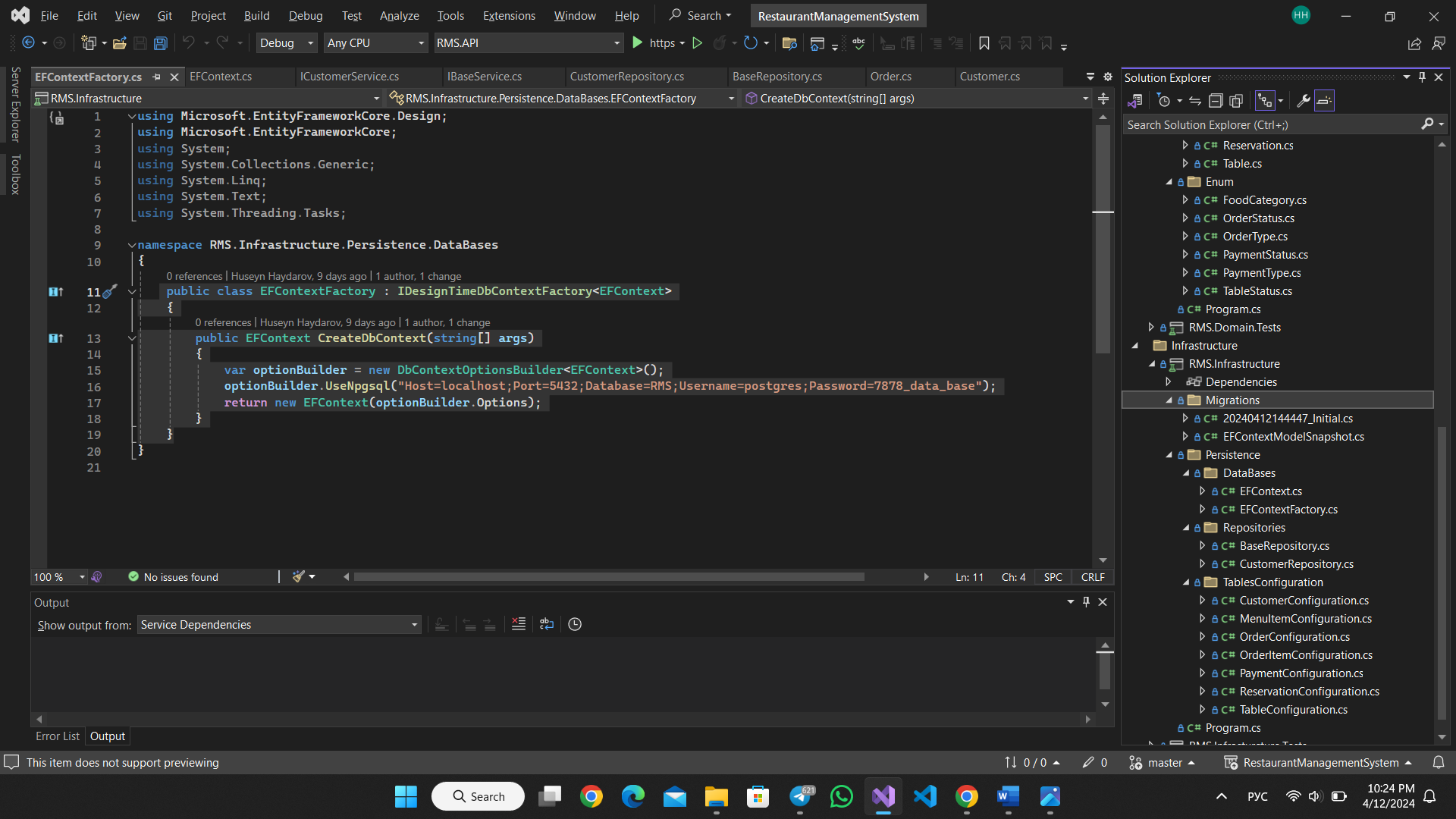
}

* 1. Использование EntityFramework для работы с базой данных

Entity Framework (EF) — это набор технологий в Microsoft .NET Framework, который позволяет разработчикам работать с данными в виде объектов и свойств, без прямой работы с базой данных. EF позволяет создавать схему базы данных, основанную на моделях объектов приложения, и автоматически генерировать SQL-код для взаимодействия с базой данных.

* 1. Миграция данных

После того как все конфигурации для всех моделей были написан, можно набрав команду add-migration CreateInitialize создать классы миграции.



**Рис 4. Классы миграции проекта**