МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

ПРАКТИЧЕСКАЯ РАБОТА №8

по дисциплине «Проектирование и разработка серверных частей интернет-ресурсов»

Студент группы ИКБО-21-23 Муравьев А.О.

_____

(подпись студента)

Руководитель практической работы Благирев М.М.

_____

(подпись руководителя)

Работа представлена «___»_____2025 г.

Допущен к работе «___» _____2025 г.

Москва 2025

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

В рамках данной практической работы предполагается результат практических работ 1-7 поместить в какой-либо фреймворк для разработки интернет-ресурсов.

Тема проекта: Сервис для контроля выполнения задач.

ХОД РАБОТЫ

Планирование переноса проекта на фреймворк

Проект написан на C# с использованием СУБД PostgreSQL.

Для переноса на фреймворк нам нужны:

- Фреймворк для API;
- Фреймворк ORM;
- Фреймворк для Frontend.

В мире .NET дела с фреймворками обстоят весьма консервативно.

Варианты фреймворка для написания API:

- **ASP.NET Core** – современный полномасштабный фреймворк для создания серверных частей веб-приложений;
- ASP.NET / Web API – устаревшее решение, работающее только на системах Windows (слишком старый);
- NancyFX – лёгкий фреймворк для создания приложений с декларативным синтаксисом маршрутов (для совсем простых приложений).

Выбор очевиден – ASP.NET Core.

Варианты фреймворка ORM:

- **Entity Framework Core (EF Core)** – современный полномасштабный ORM-фреймворк, поддерживающий оба шаблона разработки Code First и Database First; сам генерирует SQL-запросы;
- Entity Framework – устаревшая версия фреймворка EF Core;
- Dapper – легковесный быстрый ORM-фреймворк без генерации SQL.

Выбор понятен – EF Core.

С фреймворком для фронта всё сложнее, поскольку их сильно меньше при использовании чистого C# для фронтэнда.

- **Blazor Server** – код исполняется на сервере, обновления UI идут через SignalR (выполняется быстро, но требуется постоянное соединение с интернетом);
- Blazor WASM – код C# компилируется в WebAssembly и выполняется в браузере (медленная загрузка, возможен оффлайн-режим);

Для лучшей скорости работы приложения был выбран Blazor Server.

Реализация веб-приложения

На рисунках 1-2 показана настройка API (DI) с применением инструментов ASP.NET Core.

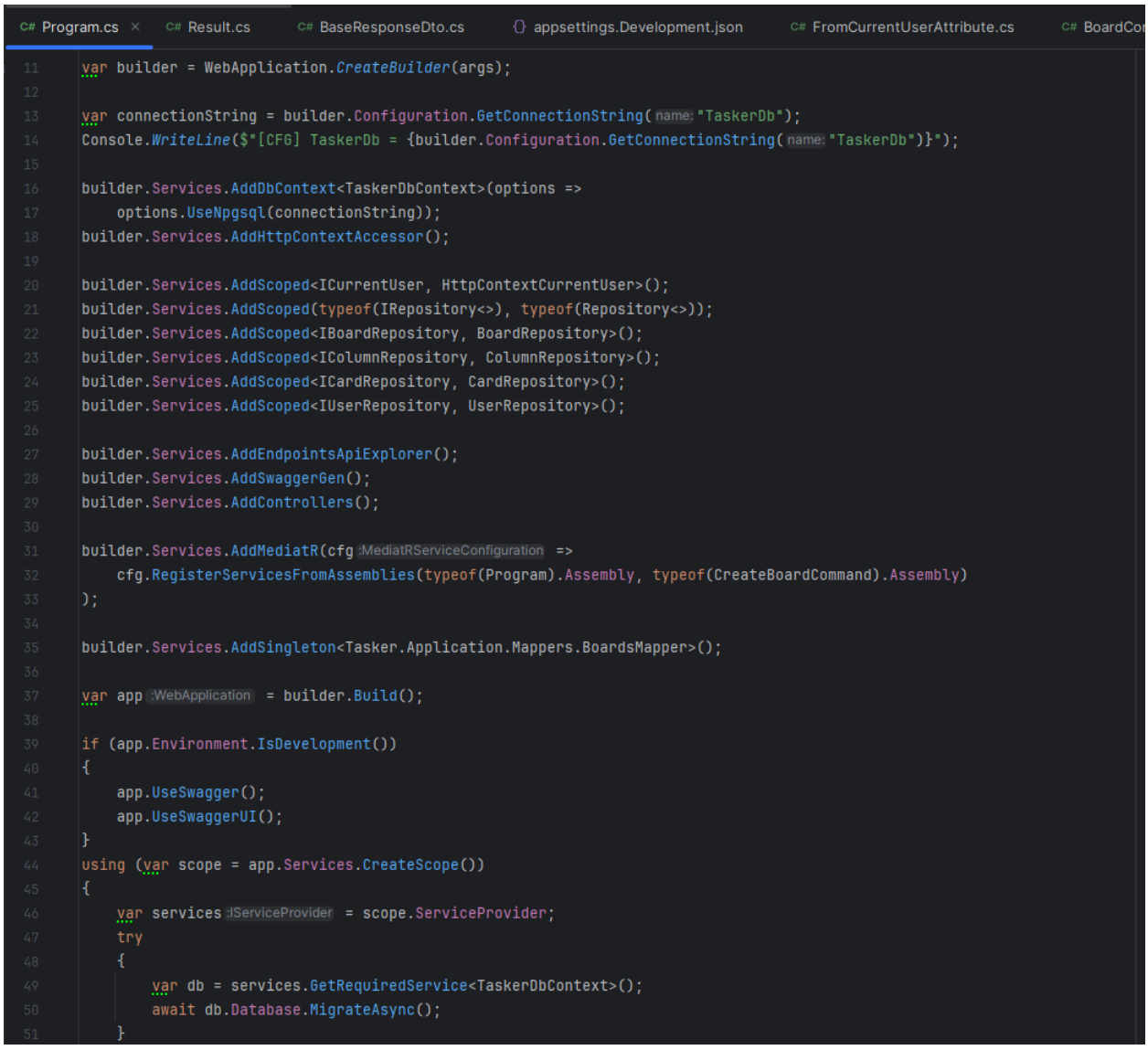

Рисунок 1 – Настройка API (DI), часть 1 из 2

```
    }
    catch (Exception ex)
    {
        var logger = services.GetRequiredService<ILogger<Program>>();
        logger.LogError(ex, message: "An error occurred while migrating the database.");
        throw;
    }
}


app.UseHttpsRedirection();
app.UseRouting();
app.UseAuthorization();
app.MapControllers();

app.MapGet( pattern: "/Api/IsAlive", () => Results.Ok(true));

app.Run();
```

Рисунок 2 - Настройка API (DI), часть 2 из 2

На рисунке 3 показан фрагмент класса TaskerDbContext, который наследуется от DbContext, предоставленного EF Core.

```
using Microsoft.EntityFrameworkCore;
using Tasker.Core.Boards;

namespace Tasker.Data
{
    12 usages  alexomur  1 exposing API
    public class TaskerDbContext : DbContext
    {
        alexomur
        public TaskerDbContext(DbContextOptions<TaskerDbContext> options) : base(options) { }

        3 usages  alexomur
        public DbSet<Board> Boards => Set<Board>();
        9 usages  alexomur
        public DbSet<Column> Columns => Set<Column>();
        5 usages  alexomur
        public DbSet<Card> Cards => Set<Card>();

        alexomur
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);

            var board :EntityTypeBuilder<Board> = modelBuilder.Entity<Board>();
            board.ToTable("Boards");
            board.HasKey(b :Board => b.Id);
            board.Property(b :Board => b.Title).IsRequired();
            board.HasMany(b :Board => b.Columns) // CollectionNavigationBuilder<Board,Column>
                .WithOne()
                .HasForeignKey("BoardId")
                .OnDelete(DeleteBehavior.Cascade);
            board.Navigation(b :Board => b.Columns).UsePropertyAccessMode(PropertyAccessMode.Field);
            board.Metadata.FindNavigation(nameof(Board.Columns))!.SetField("_columns");

            var column :EntityTypeBuilder<Column> = modelBuilder.Entity<Column>();
            column.ToTable("Columns");
            column.HasKey(c :Column => c.Id);
            column.Property(c :Column => c.Title).IsRequired();
            column.Property<System.Guid>("BoardId");
            column.Property<int>("OrderIndex");
            column.HasIndex( params propertyNames: "BoardId");
```

Рисунок 3 – Фрагмент класса TaskerDbContext

Использование репозиториев не изменено. На рисунке 4 показано использование репозитория для получения данных.

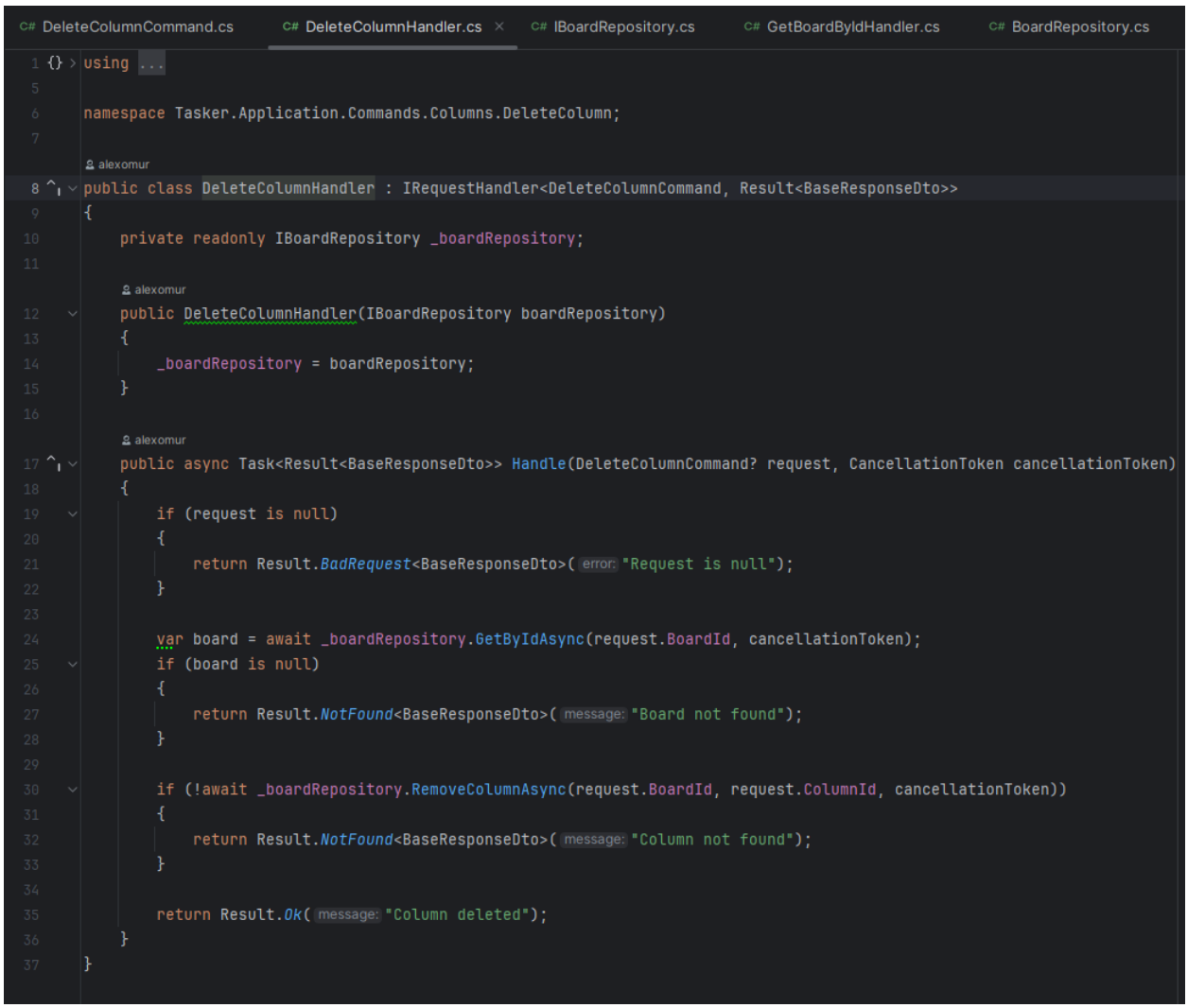

Рисунок 4 – Пример использования репозитория

Демонстрация работы веб-приложения

На рисунке 5 показана успешная сборка проекта.

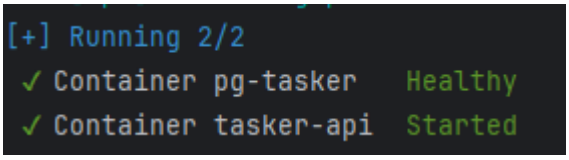

Рисунок 5 – Успешная сборка и запуск проекта

Swagger запускается, запросы выполняются. Результат показан на рисунке 9.

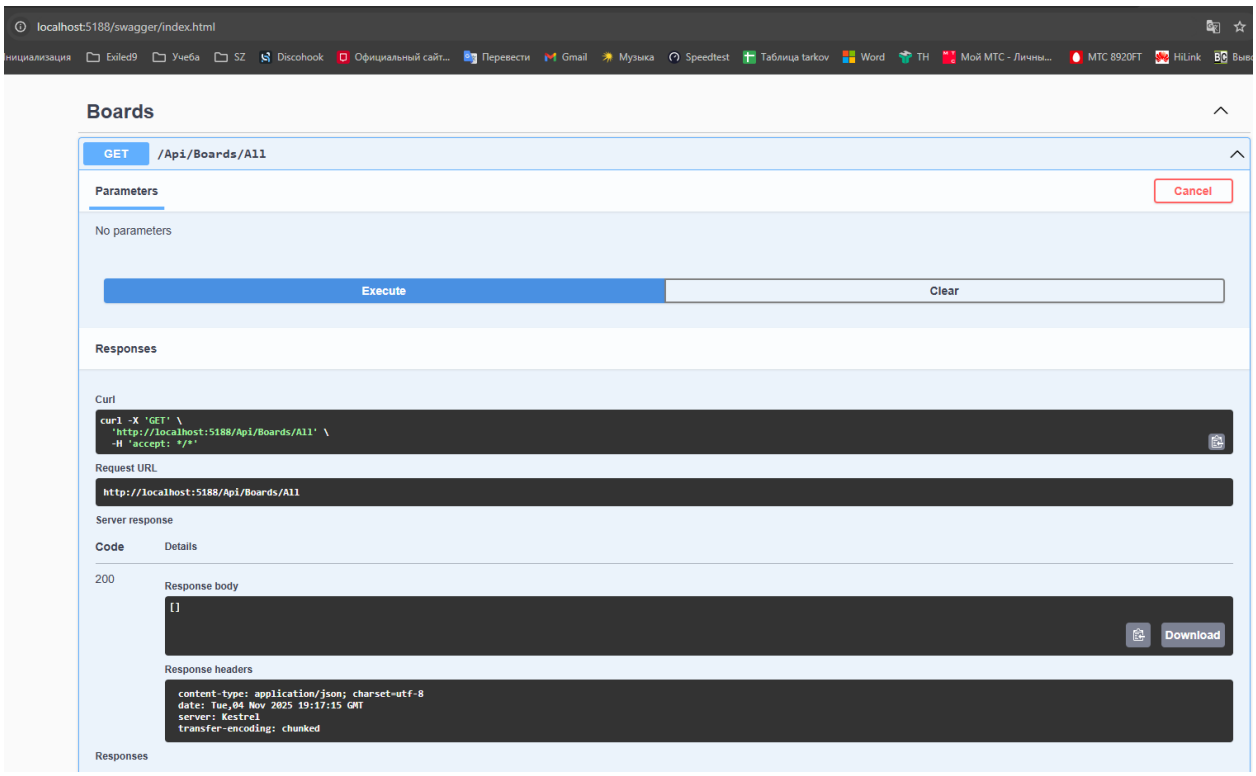

Рисунок 5 – Результат работы API

ВЫВОД

Таким образом, проект был отрефакторен.

Исходный код проекта расположен по адресу:

https://github.com/alexomur/Tasker