



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 | 3 |
| ХОД РАБОТЫ | 4 |
| Планирование рефакторинга | 4 |
| Реализация веб-приложения..... | 5 |
| ВЫВОД..... | 9 |

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.

```

 11 var builder = WebApplication.CreateBuilder(args);
12
13 var connectionString = builder.Configuration.GetConnectionString(name: "TaskerDb");
14 Console.WriteLine($"[CFG] TaskerDb = {builder.Configuration.GetConnectionString(name: "TaskerDb")}");
15
16 builder.Services.AddDbContext<TaskerDbContext>(options =>
17     options.UseNpgsql(connectionString));
18 builder.Services.AddHttpContextAccessor();
19
20 builder.Services.AddScoped<ICurrentUser, HttpContextCurrentUser>();
21 builder.Services.AddScoped(typeof(IRepository<>), typeof(Repository<>));
22 builder.Services.AddScoped<IBoardRepository, BoardRepository>();
23 builder.Services.AddScoped<IColumnRepository, ColumnRepository>();
24 builder.Services.AddScoped<ICardRepository, CardRepository>();
25 builder.Services.AddScoped<IUserRepository, UserRepository>();
26
27 builder.Services.AddEndpointsApiExplorer();
28 builder.Services.AddSwaggerGen();
29 builder.Services.AddControllers();
30
31 builder.Services.AddMediatR(cfg :MediatRServiceConfiguration =>
32     cfg.RegisterServicesFromAssemblies(typeof(Program).Assembly, typeof(CreateBoardCommand).Assembly)
33 );
34
35 builder.Services.AddSingleton<Tasker.Application.Mappers.BboardsMapper>();
36
37 var app :WebApplication = builder.Build();
38
39 if (app.Environment.IsDevelopment())
40 {
41     app.UseSwagger();
42     app.UseSwaggerUI();
43 }
44 using (var scope = app.Services.CreateScope())
45 {
46     var services :IServiceProvider = scope.ServiceProvider;
47     try
48     {
49         var db = services.GetService<TaskerDbContext>();
50         await db.Database.MigrateAsync();
51     }
}

```

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

```

50         await app.Database.MigrateAsync(),
51     }
52     catch (Exception ex)
53     {
54         var logger = services.GetRequiredService<ILogger<Program>>();
55         logger.LogError(ex, "An error occurred while migrating the database.");
56         throw;
57     }
58 }
59
60
61 app.UseHttpsRedirection();
62 app.UseRouting();
63 app.UseAuthorization();
64 app.MapControllers();
65
66 app.MapGet(pattern: "/Api/IsAlive", () => Results.Ok(true));
67
68 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.

```

1  using Microsoft.EntityFrameworkCore;
2  using Tasker.Core.Bords;
3
4  namespace Tasker.Data
5  {
6      public class TaskerDbContext : DbContext
7      {
8          public TaskerDbContext(DbContextOptions<TaskerDbContext> options) : base(options) { }
9
10         public DbSet<Board> Boards => Set<Board>();
11         public DbSet<Column> Columns => Set<Column>();
12         public DbSet<Card> Cards => Set<Card>();
13
14         protected override void OnModelCreating(ModelBuilder modelBuilder)
15         {
16             base.OnModelCreating();
17
18             var board :EntityTypeBuilder<Board> = modelBuilder.Entity<Board>();
19             board.ToTable("Boards");
20             board.HasKey(b :Board => b.Id);
21             board.Property(b :Board => b.Title).IsRequired();
22             board.HasMany(b :Board => b.Columns) //CollectionNavigationBuilder<Board, Column>
23                 .WithOne()
24                 .HasForeignKey("BoardId")
25                 .OnDelete(DeleteBehavior.Cascade);
26             board.Navigation(b :Board => b.Columns).UsePropertyAccessMode(PropertyAccessMode.Field);
27             board.Metadata.FindNavigation(nameof(Board.Columns))!.SetField("_columns");
28
29             var column :EntityTypeBuilder<Column> = modelBuilder.Entity<Column>();
30             column.ToTable("Columns");
31             column.HasKey(c :Column => c.Id);
32             column.Property(c :Column => c.Title).IsRequired();
33             column.Property<System.Guid>("BoardId");
34             column.Property<int>("OrderIndex");
35             column.HasIndex(params propertyName: "BoardId");

```

Рисунок 3 – Фрагмент класса TaskerDbContext

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

```
1 {} > using ...
5
6 namespace Tasker.Application.Commands.Columns.DeleteColumn;
7
8 ^, ~ public class DeleteColumnHandler : IRequestHandler<DeleteColumnCommand, Result<BaseResponseDto>>
9 {
10     private readonly IBoardRepository _boardRepository;
11
12     ~ alexomur
13     public DeleteColumnHandler(IBoardRepository boardRepository)
14     {
15         _boardRepository = boardRepository;
16     }
17
18     ~ alexomur
19     public async Task<Result<BaseResponseDto>> Handle(DeleteColumnCommand? request, CancellationToken cancellationToken)
20     {
21         if (request is null)
22         {
23             return Result.BadRequest<BaseResponseDto>(error: "Request is null");
24         }
25
26         var board = await _boardRepository.GetByIdAsync(request.BoardId, cancellationToken);
27         if (board is null)
28         {
29             return Result.NotFound<BaseResponseDto>(message: "Board not found");
30         }
31
32         if (!await _boardRepository.RemoveColumnAsync(request.BoardId, request.ColumnId, cancellationToken))
33         {
34             return Result.NotFound<BaseResponseDto>(message: "Column not found");
35         }
36
37         return Result.Ok(message: "Column deleted");
38     }
39 }
```

Рисунок 4 – Пример использования репозитория

Демонстрация работы веб-приложения

На рисунке 5 показана успешная сборка проекта.

```
[+] Running 2/2
✓ Container pg-tasker    Healthy
✓ Container tasker-api   Started
```

Рисунок 5 – Успешная сборка и запуск проекта

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

The screenshot shows the Swagger UI interface for a .NET Core API. The main title is "Boards". Below it, a "GET /Api/Boards/All" section is displayed. Under "Parameters", there is a note "No parameters". At the bottom of this section are "Execute" and "Clear" buttons. The "Responses" section contains a "Curl" block with the command:

```
curl -X 'GET' \
  'http://localhost:5188/api/Boards/All' \
  -H 'accept: */*'
```

Below the curl command is the "Request URL": <http://localhost:5188/api/Boards/All>. The "Server response" section shows a "Code" table with one row for "200". The "Details" column under "200" shows the "Response body" as an empty array: `[]`. To the right of the array are "Copy" and "Download" buttons. The "Response headers" section lists:

```
content-type: application/json; charset=utf-8
date: Tue, 04 Nov 2025 19:17:15 GMT
server: Kestrel
transfer-encoding: chunked
```

Рисунок 5 – Результат работы API

ВЫВОД

Таким образом, проект был отрефакторен.

Исходный код проекта расположен по адресу:

<https://github.com/alexomur/Tasker>