



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 №7**

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**

<b>ЦЕЛЬ РАБОТЫ .....</b>	<b>3</b>
<b>ХОД РАБОТЫ .....</b>	<b>4</b>
Планирование рефакторинга .....	4
Реализация веб-приложения.....	4
<b>ВЫВОД.....</b>	<b>12</b>

## **ЦЕЛЬ РАБОТЫ**

В рамках данной практической работы предполагается провести рефакторинг информационной системы, созданной в практических работах №1–6.

Рефакторинг — это процесс изменения внутренней структуры программы, не затрагивающий её внешнее поведение и направленный на упрощение понимания и сопровождения кода.

В основе рефакторинга лежит последовательность небольших эквивалентных преобразований, то есть таких, которые сохраняют исходное поведение программы. Поскольку каждое преобразование невелико, программисту проще проследить его корректность. В то же время, совокупность этих небольших шагов может привести к существенной перестройке программы, улучшению её согласованности и читаемости.

Рефакторинг системы следует проводить путём перехода от процедурной парадигмы к объектно-ориентированной (ООП) с переносом логики на определённую архитектуру.

На внешнем слое архитектуры (в случае использования DDD — Domain-Driven Design) или на слое интерфейсов (если применяется чистая архитектура) рекомендуется внедрить один из шаблонов проектирования, например MVC (Model–View–Controller) или MVP (Model–View–Presenter).

Тема проекта: Сервис для контроля выполнения задач.

# ХОД РАБОТЫ

## Планирование рефакторинга

Проект написан на ASP.NET. Использует СУБД PostgreSQL для хранения данных. Roik для преобразования данных. Swagger для тестирования API. Entity Framework для работы с реляционной базой данных. MediatR для выполнения шаблона «посредник» и др.

Решение будет разделено на 5 проектов:

- Tasker.Core – хранилище моделей. Основа DDD.
- Tasker.Api – обработчик сырых запросов. Содержит контроллеры, отправляющие запросы в Tasker.Application, настройки DI.
- Tasker.Application – обработчик команд и запросов из API, хранилище моделей DTO.
- Tasker.Data – работник с данными. Источник DbContext, репозиториев, настроек EF Core, хранилище миграций.
- Tasker.Web – Frontend составляющая решения.

## Реализация веб-приложения

На рисунке 1 показана проектная структура решения.

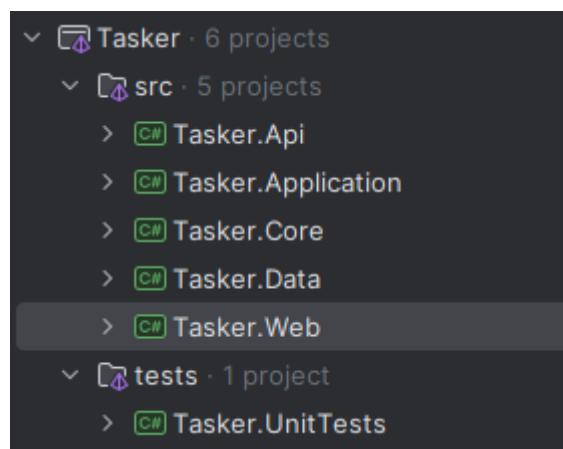
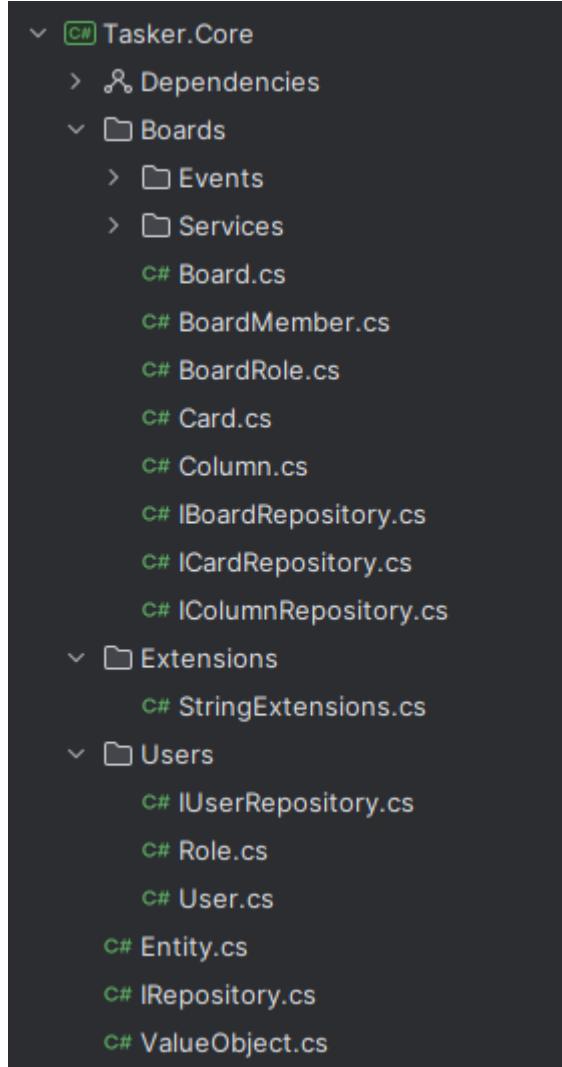


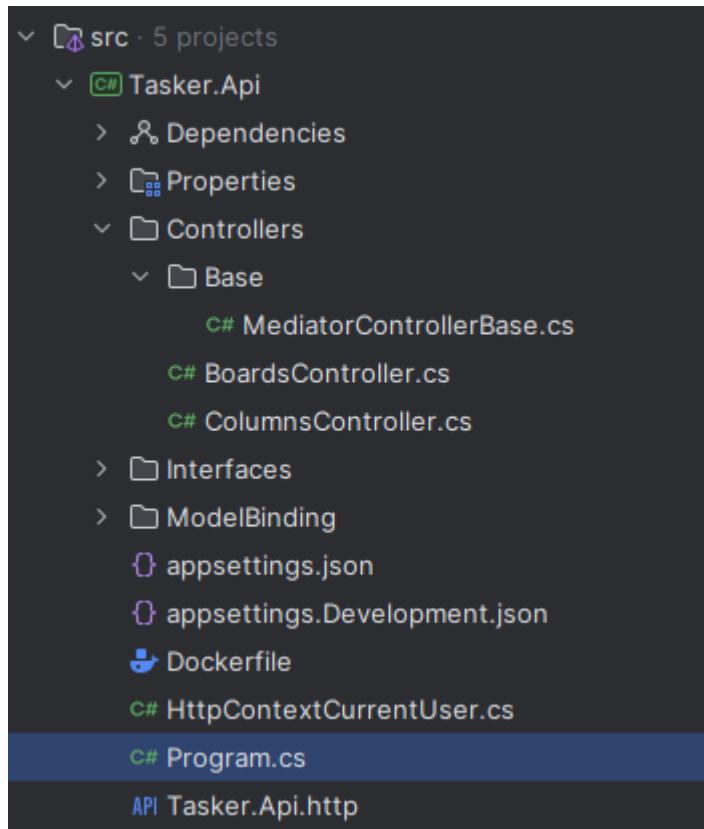
Рисунок 1 – Проектная структура решения

На рисунке 2 показана структура Tasker.Core – основы DDD решения.



**Рисунок 2 – Файловая структура Tasker.Core**

На рисунке 3 показана файловая структура Tasker.Api.



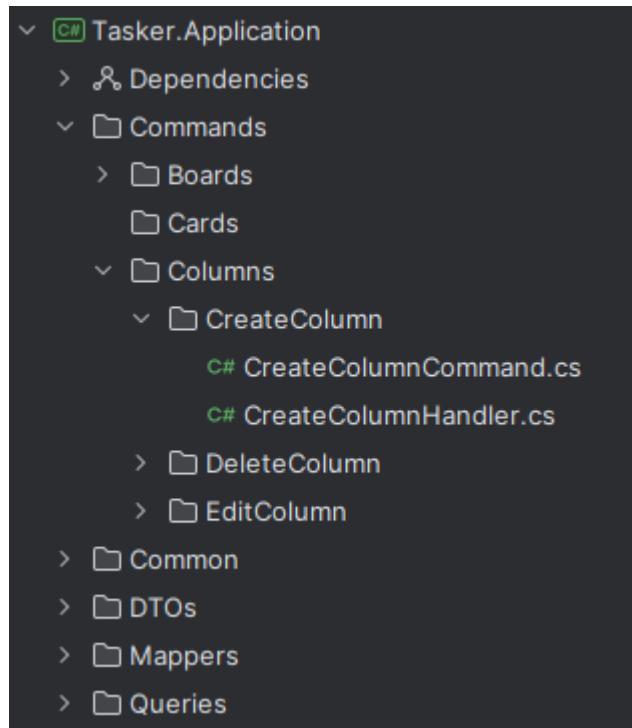
**Рисунок 3 – Файловая структура Tasker.Api**

На рисунке 4 показан контроллер запросов, связанных с колонками.

```
13  namespace Tasker.Api.Controllers;
14 
15  [ApiController]
16  [Area("Boards")]
17  [Route("Api/{area}/{boardId:guid}/{controller}")]
18  ^o public class ColumnsController : MediatorControllerBase
19  {
20      ^o     public ColumnsController(IMediator mediator) : base(mediator)
21      {
22      }
23 
24      [HttpGet(template: "All")]
25      ^o     public async Task<IActionResult> GetAllByBoardId([FromRoute] Guid boardId)
26          => await ExecuteCommand<GetAllColumnsCommand, List<ColumnDto>>(new GetAllColumnsCommand(boardId));
27 
28      [HttpGet(template: "{columnId:guid}")]
29      ^o     public async Task<IActionResult> GetById([FromRoute] Guid boardId, [FromRoute] Guid columnId)
30          => await ExecuteCommand<GetColumnByIdCommand, ColumnDto>(new GetColumnByIdCommand(boardId, columnId));
31 
32      [HttpPost(template: "Create")]
33      ^o     public async Task<IActionResult> Create([FromRoute] Guid boardId, [FromQuery] string title, [FromQuery] string?
34          => await ExecuteCommand<CreateColumnCommand, ColumnDto?>(new CreateColumnCommand(boardId, title, description));
35 
36      [HttpPatch(template: "{columnId:guid}/Edit")]
37      ^o     public async Task<IActionResult> Edit([FromRoute] Guid boardId, [FromRoute] Guid columnId, [FromQuery] string?
38          => await ExecuteCommand<EditColumnCommand, ColumnDto>(new EditColumnCommand(boardId, columnId, title, description));
39 
40      [HttpDelete(template: "{columnId:guid}/Delete")]
41      ^o     public async Task<IActionResult> Delete([FromRoute] Guid boardId, [FromRoute] Guid columnId)
42          => await ExecuteCommand<DeleteColumnCommand, BaseResponseDto>(new DeleteColumnCommand(boardId, columnId));
43 }
```

**Рисунок 4 – Контроллер ColumnsController**

На рисунке 5 показана файловая структура Tasker.Application.



**Рисунок 5 – Файловая структура Tasker.Application**

Каждая команда (или запрос) состоит из определения команды и её обработчика.

На рисунке 6 показаны команда `CreateColumnCommand` и обработчик `CreateColumnCommand`.

The screenshot shows a code editor with two tabs open. The top tab is 'CreateColumnCommand.cs' and the bottom tab is 'CreateColumnHandler.cs'. Both files are part of the 'Tasker.Application.Commands.Columns.CreateColumn' namespace.

```
1 {} > using ...
4
5 namespace Tasker.Application.Commands.Columns.CreateColumn;
6
7 ^ public record CreateColumnCommand(Guid BoardId, string Title, string? Description) : IRequest<Result<ColumnDto?>>;
8
9
C# CreateColumnHandler.cs ×
1 {} > using ...
2
3 namespace Tasker.Application.Commands.Columns.CreateColumn;
4
5 ^ public class CreateColumnHandler : IRequestHandler<CreateColumnCommand, Result<ColumnDto?>>
6 {
7     private readonly IBoardRepository _boardRepository;
8
9     ^ public CreateColumnHandler(IBoardRepository boardRepository)
10    {
11        _boardRepository = boardRepository;
12    }
13
14    ^ public async Task<Result<ColumnDto?>> Handle(CreateColumnCommand? request, CancellationToken cancellationToken)
15    {
16        if (request is null)
17        {
18            return Result.BadRequest<ColumnDto?>(error: "Request is null");
19        }
20
21        var column = await _boardRepository.AddColumnAsync(
22            request.BoardId,
23            request.Title,
24            request.Description,
25            cancellationToken); // Task<Column>
26
27        return Result.Ok<ColumnDto?>(new ColumnsMapper().ToDto(column));
28    }
29
30
31
32
33 }
```

**Рисунок 6 – Команда и её обработчик**

На рисунке 7 показана файловая структура проекта Tasker.Data.

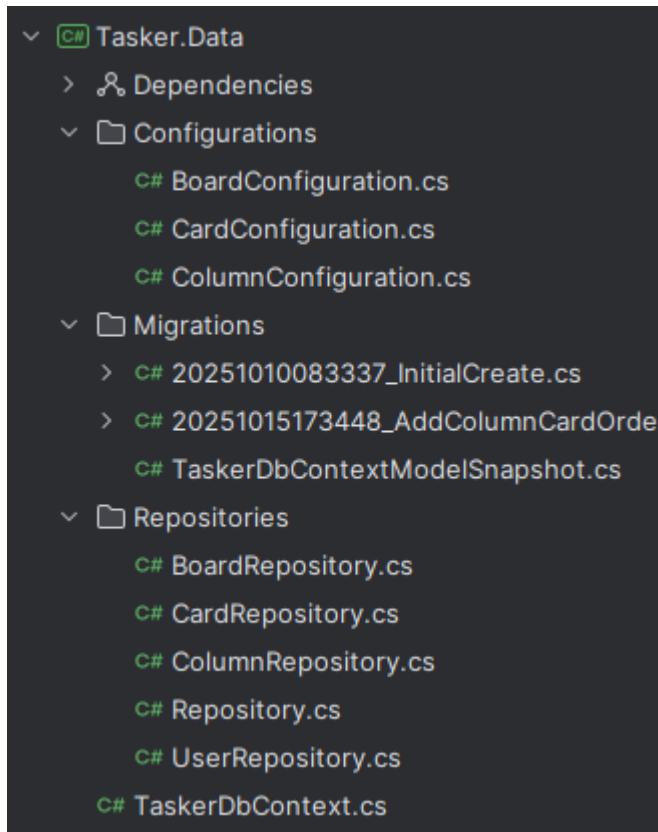


Рисунок 7 – Файловая структура проекта Tasker.Data

### Демонстрация работы веб-приложения

На рисунке 8 показана успешная сборка проекта.

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

Рисунок 8 – Успешная сборка и запуск проекта

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

localhost:5188/swagger/index.html

Boards

GET /Api/Boards/All

Parameters

No parameters

Responses

Curl

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

Request URL

http://localhost:5188/Api/Boards/All

Server response

Code	Details
200	Response body [] Response headers content-type: application/json; charset=utf-8 date: Tue, 04 Nov 2025 19:17:15 GMT server: Kestrel transfer-encoding: chunked

Responses

Cancel

Execute

Clear

Download

**Рисунок 9 – Результат работы API**

## **ВЫВОД**

Таким образом, проект был отрефакторен.

Исходный код проекта расположен по адресу:

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