



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 №4

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
СТРУКТУРА ВЕБ-ПРИЛОЖЕНИЯ	4
РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ	4
РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ... Ошибка! Закладка не определена.	
ВЫВОД.....	10

ЦЕЛЬ РАБОТЫ

Предполагается реализация интерфейса прикладного программирования для доступа к некоторым данным по варианту. Предполагается реализация серверной части обработки запросов и тестирование данного интерфейса с использованием программы Postman. Для реализации данного сервиса предлагается использовать серверную конфигурацию, модернизированную в течение первых трех практических работ. Важной частью данной практической работы является сохранение функциональности реализованной в практической работе №3. То есть интерфейс предлагается создать уже в существующем веб-приложении. Также предполагается использование темы практической работы №3 для продолжения модернизирования собственной системы. Изменение темы согласовывается отдельно с преподавателем. Хранение данных предполагается уже в существующей базе данных.

Тема проекта: Сервис для контроля выполнения задач.

Парадигма API: REST.

ХОД РАБОТЫ

Структура веб-приложения

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.

Реализация веб-приложения

Для реализации CRUD были созданы команды для работы с бордами (Board), показанные на рисунке 1.

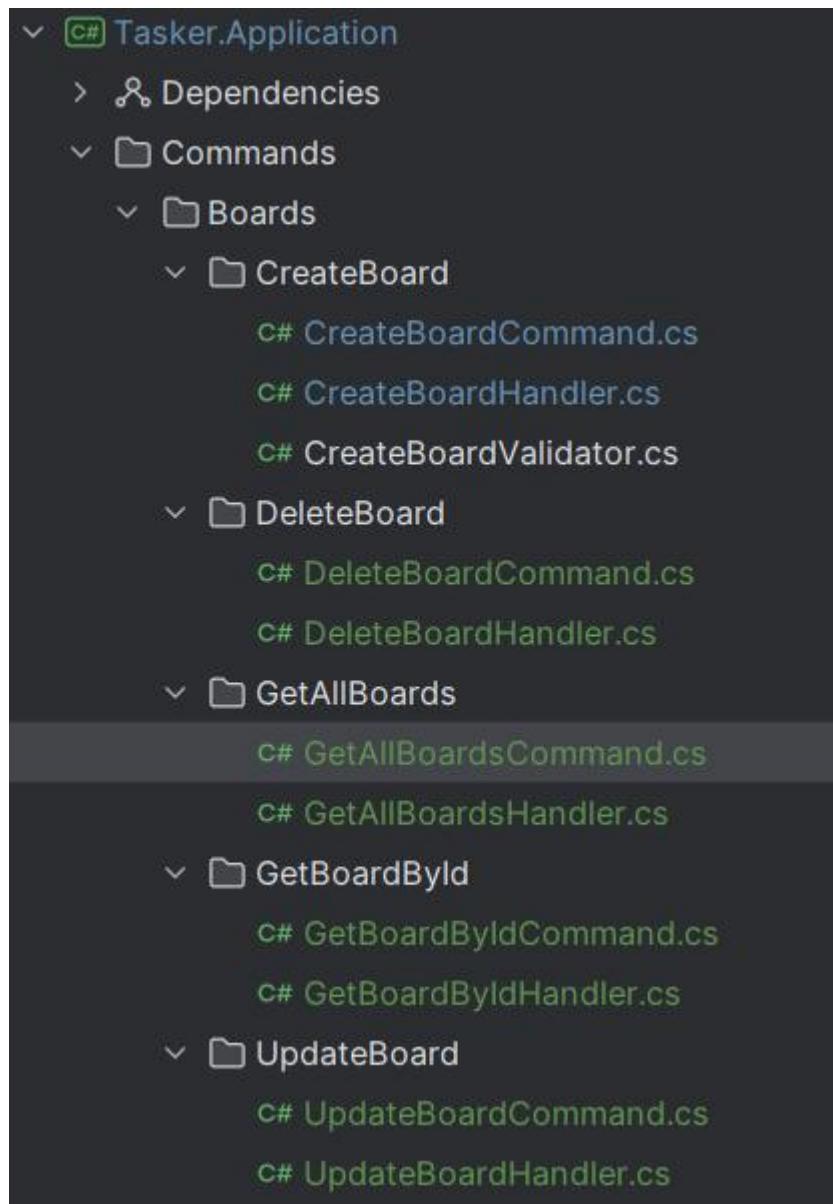


Рисунок 1 – Файлы созданных команд

Каждая команда состоит из Command-файла (объявление команды), Handler-файла (описание обработки его вызова) и может иметь Validator-файл (для валидации запроса).

Каждый вид запроса необходимо утвердить в MediatR и в самом ASP.NET. Для этого нужно создать API-контроллер и его основу (рисунок 2).

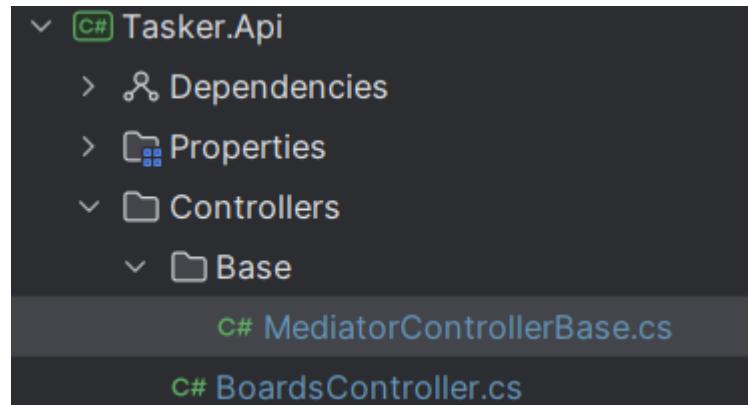


Рисунок 2 – Файлы контроллеров

Все команды регистрируются в контроллере, что показано на рисунке 3.

```

16 public class BoardsController : MediatorControllerBase
17 {
18
19     [HttpGet(template: "All")]
20     & new *
21     public async Task<IActionResult> GetAll()
22         => await ExecuteCommand<GetAllBoardsCommand, List<BoardDto>>(new GetAllBoardsCommand());
23
24     [HttpGet(template: "{boardId:guid}")]
25     & new *
26     public async Task<IActionResult> Get(Guid boardId)
27         => await ExecuteCommand<GetBoardByIdCommand, BoardDto?>(new GetBoardByIdCommand(boardId));
28
29     [HttpPost]
30     & new *
31     public async Task<IActionResult> CreateBoard([FromQuery] string title, [FromQuery] string? description)
32         => await ExecuteCommand<CreateBoardCommand, BoardDto>(new CreateBoardCommand(title, description));
33
34     [HttpPut(template: "{boardId:guid}")]
35     & new *
36     public async Task<IActionResult> UpdateBoard(Guid boardId, [FromQuery] string title, [FromQuery] string? description)
37         => await ExecuteCommand<UpdateBoardCommand, BaseResponseDto>(new UpdateBoardCommand(boardId, title, description));
38
39     [HttpDelete(template: "{boardId:guid}")]
40     & new *
41     public async Task<IActionResult> DeleteBoard(Guid boardId)
42         => await ExecuteCommand<DeleteBoardCommand, BaseResponseDto>(new DeleteBoardCommand(boardId));
43 }

```

Рисунок 3 – Регистрация команд в контроллере

Тестирование веб-приложения

Для тестирования API был использован Swagger UI.

На рисунке 4 показан результат GET запроса по адресу /Api/Boards/All.

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 <pre>[{ "id": "698b09df-c0b3-4189-9f1d-d3bddca0e077", "title": "TextText", "description": "TextTextText" }]</pre> <p>Copy Download</p>

Рисунок 4 – Запрос всех Boards

На рисунке 5 показан результат запроса по адресу /Api/Boards/{boardId}.

Parameters

Name	Description
boardId * required	string(\$uuid) (path)

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5188/Api/Boards/698b09df-c0b3-4189-9f1d-d3bddca0e077' \
-H 'accept: */*'
```

Request URL

```
http://localhost:5188/Api/Boards/698b09df-c0b3-4189-9f1d-d3bddca0e077
```

Server response

Code	Details
200	Response body <pre>{ "id": "698b09df-c0b3-4189-9f1d-d3bddca0e077", "title": "TextText", "description": "TextTextText" }</pre> <p>Copy Download</p>

Рисунок 5 – Запрос определённого Board

На рисунке 6 показан результат POST запроса по адресу /Api/Boards/.

The screenshot shows a REST API interface with the following details:

- Request Body:**
 - title: Тест 1
 - description: Тест 11
- Buttons:** Execute, Clear
- Responses:**
 - Curl:**

```
curl -X 'POST' \
'http://localhost:5188/api/Boards?title=%D0%A2%D0%85%D1%81%D1%82%D0%81&description=%D0%A2%D0%85%D1%81%D1%82%D0%81'
```
 - Request URL:** <http://localhost:5188/api/Boards?title=%D0%A2%D0%85%D1%81%D1%82%D0%81&description=%D0%A2%D0%85%D1%81%D1%82%D0%81>
 - Server response:**

Code	Details
200	Response body <pre>{ "id": "f5bb019b-dd6d-4f0a-b7b5-9fd5aa90459", "title": "Тест 1", "description": "Тест 11" }</pre>

Рисунок 6 – Запрос на создание нового Board

На рисунке 7 показан результат PUT запроса по адресу /Api/Boards/{boardId}.

The screenshot shows a REST API interface with the following details:

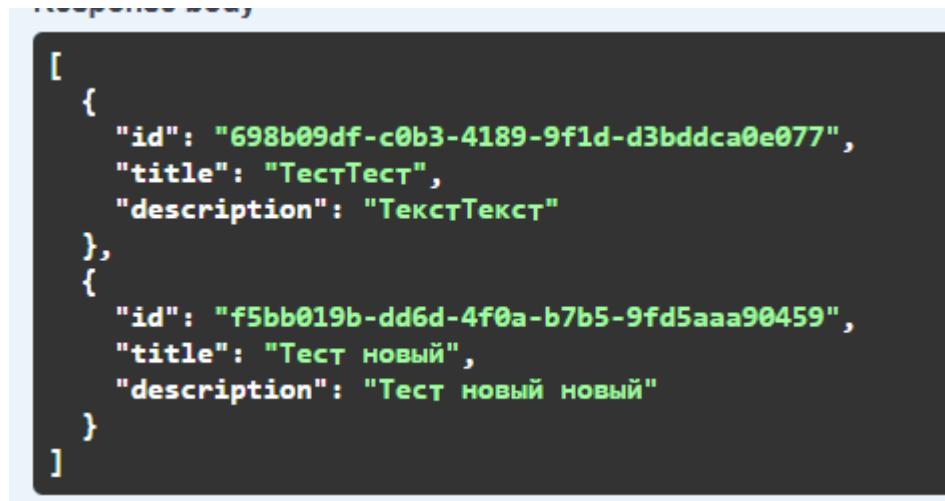
- Request Body:**
 - title: Тест новый
 - description: Тест новый новый
- Buttons:** Execute, Clear
- Responses:**
 - Curl:**

```
curl -X 'PUT' \
'http://localhost:5188/api/Boards/f5bb019b-dd6d-4f0a-b7b5-9fd5aa90459?title=%D0%A2%D0%85%D1%81%D1%82%D0%80%D0%BD%D0%BE%D0%82%D1%8B&description=%D0%A2%D0%85%D1%81%D1%82%D0%80%D0%BD%D0%BE%D0%82%D1%8B'
```
 - Request URL:** <http://localhost:5188/api/Boards/f5bb019b-dd6d-4f0a-b7b5-9fd5aa90459?title=%D0%A2%D0%85%D1%81%D1%82%D0%80%D0%BD%D0%BE%D0%82%D1%8B&description=%D0%A2%D0%85%D1%81%D1%82%D0%80%D0%BD%D0%BE%D0%82%D1%8B>
 - Server response:**

Code	Details
200	Response body <pre>{ "message": "Board updated." }</pre>

Рисунок 7 – Обновление существующего Board

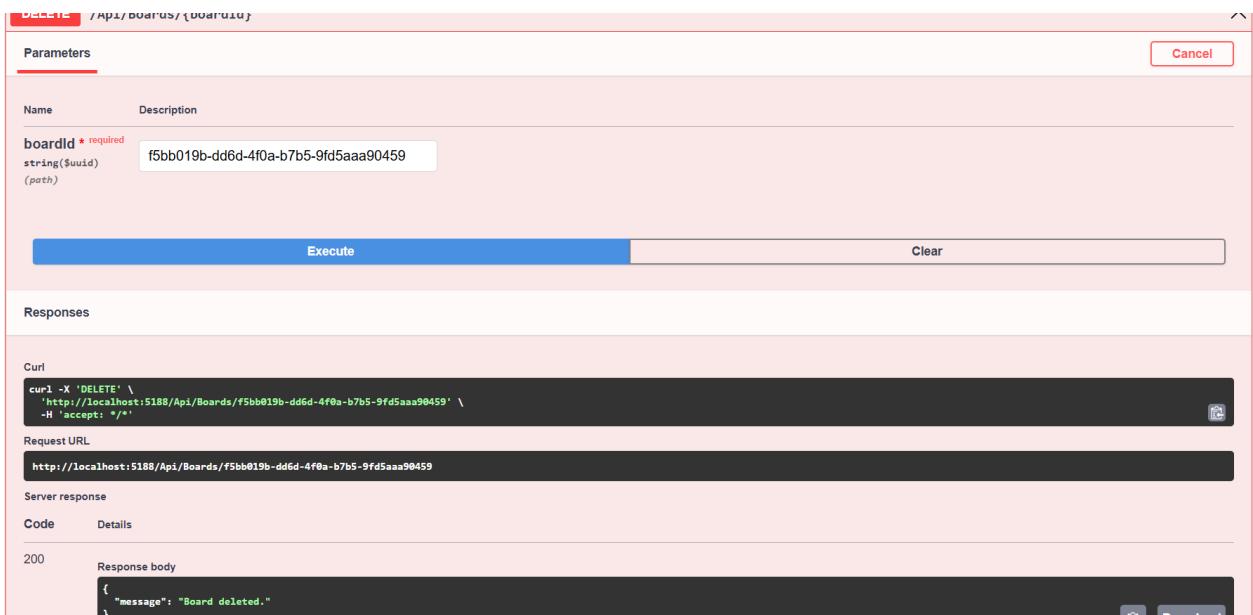
Проверка изменения после PUT запроса показана на рисунке 8.



```
[  
  {  
    "id": "698b09df-c0b3-4189-9f1d-d3bddca0e077",  
    "title": "ТестТест",  
    "description": "ТекстТекст"  
  },  
  {  
    "id": "f5bb019b-dd6d-4f0a-b7b5-9fd5aaa90459",  
    "title": "Тест новый",  
    "description": "Тест новый новый"  
  }  
]
```

Рисунок 8 – Демонстрация изменения Board

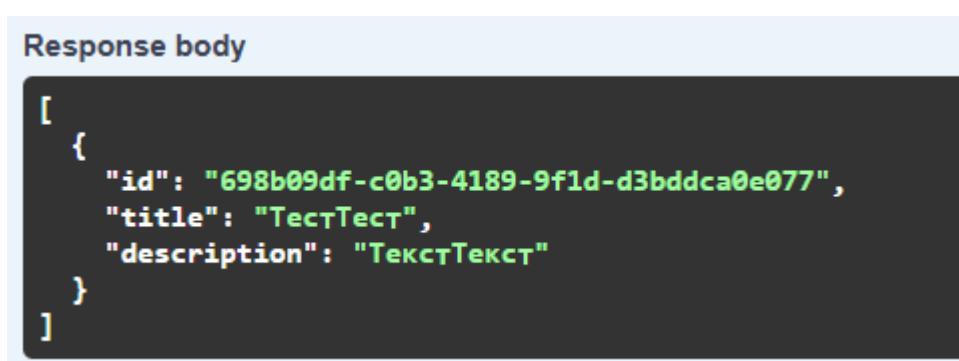
На рисунке 9 показан результат отправки DELETE запроса по адресу /Api/Boards/{boardId}.



The screenshot shows a DELETE request configuration in a tool like Postman. The URL is set to `/api/boards/f5bb019b-dd6d-4f0a-b7b5-9fd5aaa90459`. The `boardId` parameter is specified as `f5bb019b-dd6d-4f0a-b7b5-9fd5aaa90459`. The `Method` dropdown is set to `DELETE`. The `Execute` button is visible at the bottom left, and the `Cancel` button is at the top right. Below the request details, there are sections for `Curl`, `Request URL`, and `Server response`.

Рисунок 9 – Удаление Board

На рисунке 10 показан результат выполнения DELETE запроса.



```
[  
  {  
    "id": "698b09df-c0b3-4189-9f1d-d3bddca0e077",  
    "title": "ТестТест",  
    "description": "ТекстТекст"  
  }  
]
```

Рисунок 10 – Демонстрация удаления Board

ВЫВОД

Таким образом, был разработан API для описанного сервиса.

Исходный код проекта расположен по адресу:

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