



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 4

Дисциплина: «Backend»

Тема: *Добавление в веб-приложение на основе ASP.NET Core маршрутизации*

Выполнил: студент группы: 231-339

Карапетян Нвер Каренович

(Фамилия И.О.)

Дата, подпись: 14.03.25

(Дата)

(Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись _____

(Дата)

(Подпись)

Москва
2025

Цель:

Познакомиться с основами настройки и использования маршрутизации в веб-приложениях на основе ASP.NET Core.

Задачи:

- Расширить существующее веб-приложение на ASP.NET Core добавлением новых маршрутов.
- Настроить маршруты для различных контроллеров и действий.
- Использовать различные параметры маршрутизации, такие как шаблоны маршрутов, ограничения и атрибуты.
- Протестировать работу маршрутизации, обеспечив ее корректное функционирование при обращении к различным URL.

Инструменты и технологии:

- **ASP.NET Core** – фреймворк для разработки веб-приложений.
- **Entity Framework Core** – ORM для работы с базой данных.
- **SSMS** – инструмент для управления базами данных (MS SQL или MySQL).
- **Scalar** – инструмент для тестирования API.

Ход работы

Настройка маршрутизации в веб-приложении на основе ASP.NET Core позволяет управлять обработкой HTTP-запросов и обеспечивать доступ к различным ресурсам. В ходе работы были рассмотрены различные параметры маршрутизации, такие как шаблоны маршрутов, ограничения и атрибуты, а также их применение в CommentController.

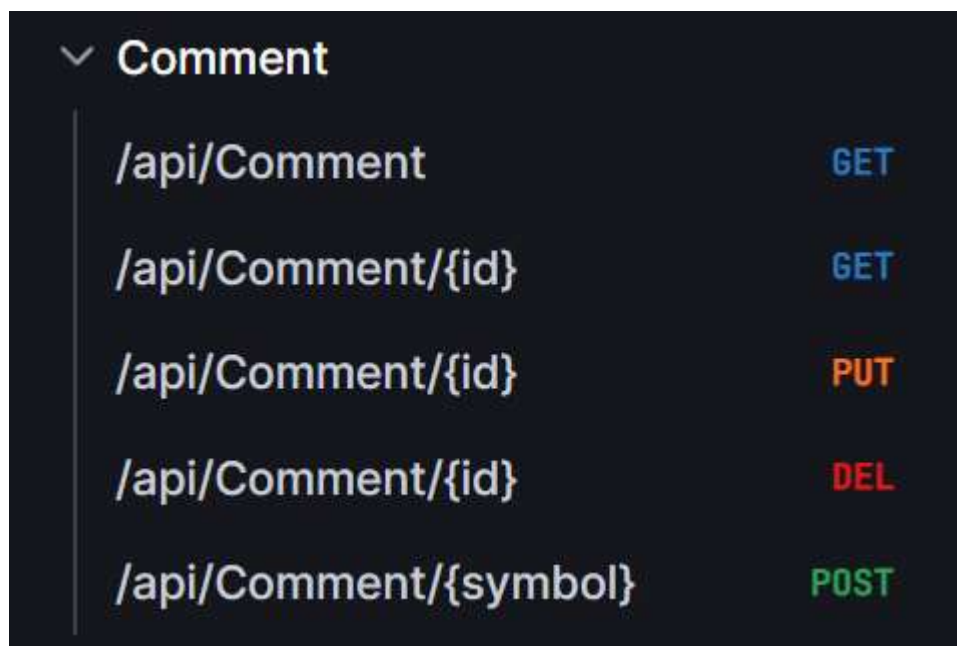
Шаблоны маршрутов

В первую очередь, обратим наше внимание на объявление самого класса контроллера. Оно содержит следующие .NET-атрибуты:

Листинг 1. Атрибуты класса CommentController.

```
[Route("api/[controller]")]
[ApiController]
public class CommentController : ControllerBase
{
    ...
}
```

В CommentController маршрут задаётся с помощью атрибута [Route("api/[controller]")], который определяет базовый URL для всех методов внутри контроллера. Директива [controller] автоматически подставляет имя контроллера без суффикса "Controller", что в данном случае превращает маршрут в **api/comment**. Это позволяет унифицировать маршруты без необходимости указывать их вручную в каждом контроллере. Открыв Swagger, можно наглядно проследить единоначалие адресов маршрутов всех конечных точек («эндпоинтов») различных HTTP-запросов:



Comment	
/api/Comment	GET
/api/Comment/{id}	GET
/api/Comment/{id}	PUT
/api/Comment/{id}	DEL
/api/Comment/{symbol}	POST

Рисунок 1. Все «эндпоинты» контроллера начинаются одинаково.

Атрибут [ApiController] дополняет работу маршрутизации, автоматически добавляя проверку модели (ModelState.IsValid), упрощая обработку запросов и управление данными. Он также позволяет ASP.NET Core автоматически определять источник параметров ([FromBody], [FromRoute] и др.), если это явно не указано.

Таким образом, базовый маршрут `api/comment` служит префиксом для всех запросов, а конечные маршруты определяются внутри методов контроллера, расширяя базовый маршрут и позволяя обрабатывать различные запросы.

Ограничения маршрутов

В `CommentController` используются различные ограничения маршрутизации, которые позволяют определять типы передаваемых параметров, ограничивая набор допустимых значений. Рассмотрим для примера атрибуты .NET перед такими HTTP-запросами, как GET, PUT и DELETE.

Листинг 2. Атрибуты GET-, PUT- и DELETE-методов

```
[HttpGet("{id:int}")]
public async Task<IActionResult> GetById([FromRoute] int id)
{
    ...
}

[HttpPut("{id:int}")]
public async Task<IActionResult> Update([FromRoute] int id, [FromBody] UpdateCom-
mentRequestDto updateDto)
{
    ...
}

[HttpDelete("{id:int}")]
public async Task<IActionResult> Delete([FromRoute] int id)
{
    ...
}
```

В приведённых примерах маршрутная модель маршрутизации включает в себя параметр `id`, для которого задано ограничение типа `int`. Это означает, что `id` в URL должен представлять собой целое число. Если клиент попытается передать нечисловое значение, сервер автоматически вернет ошибку 404 без выполнения метода:

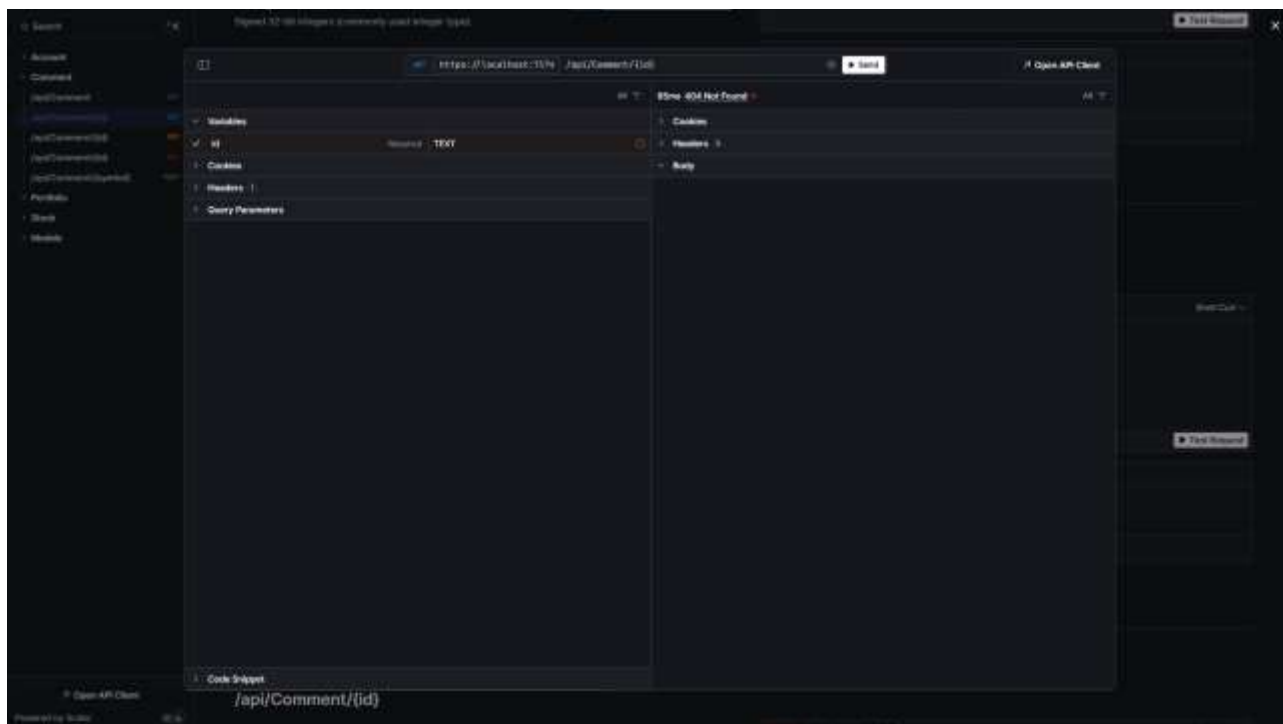


Рисунок 2. Попытка передать в качестве параметра строку обвенчалась ошибкой 404.

В то же время метод `HttpPut("{id:int}")` принимает `id` комментария как числовой параметр и обновляет соответствующую запись, а `[HttpDelete("{id:int}")]` определяет маршрут для удаления комментария с указанным идентификатором.

Рассмотрим .NET-атрибуты POST-запроса:

Листинг 3. Атрибуты POST-запроса.

```
[HttpPost("{symbol:alpha}")]
[Authorize]
public async Task<IActionResult> Create([FromRoute] string symbol, CreateCommentDto commentDto)
{
    ...
}
```

У принимаемого методом параметра `symbol` стоит ограничение `alpha`. Это ограничение гарантирует, что параметр соответствует символам латинского алфавита верхнего или нижнего регистра (`a–z`, `A–Z`). Соответственно, если через `Scalar` попытаться отправить запрос, передав в качестве параметра число, мы получим практически мгновенную ошибку 404:

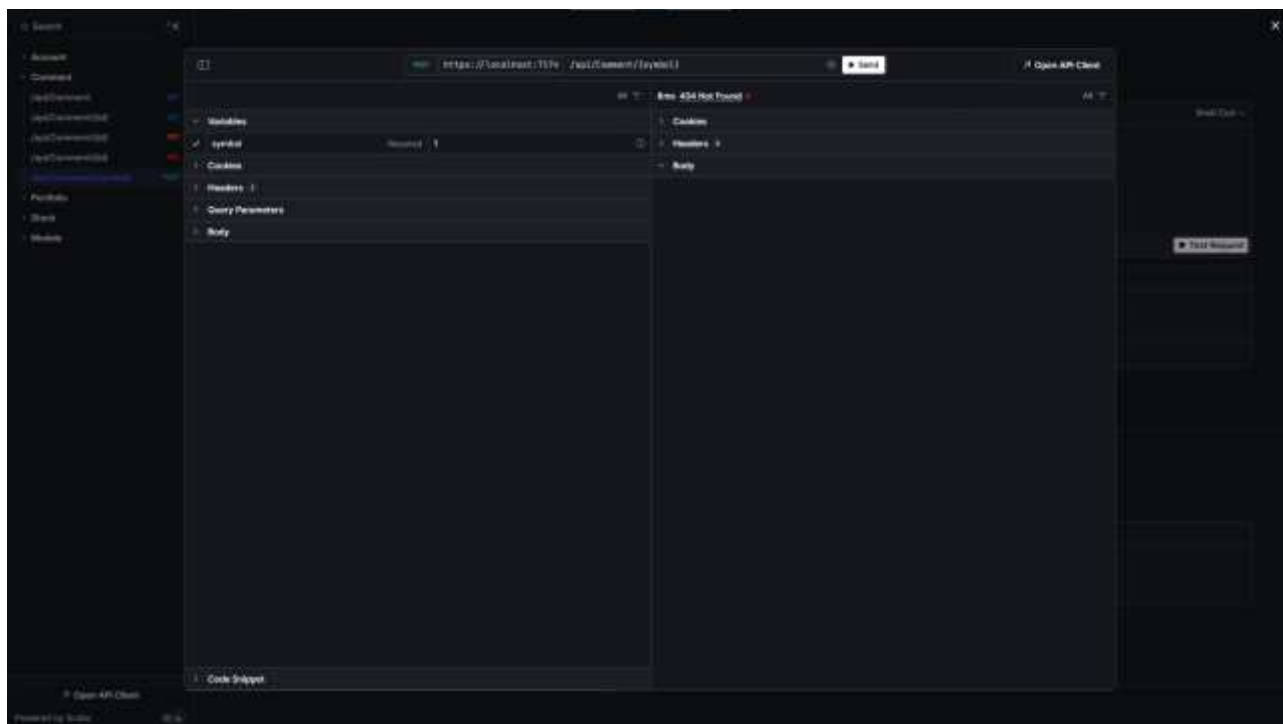


Рисунок 3. Передача в качестве параметра числа вместо ожидаемого строкового типа.
Всевозможные виды поддерживаемых ограничений перечислены в официальной документации Microsoft:

Ограничение	Описание	Пример
alpha	Соответствует символам латинского алфавита верхнего или нижнего регистра (a–z, A–Z)	{x:alpha}
bool	Соответствует логическому значению.	{x:bool}
datetime	Соответствует значению DateTime .	{x:datetime}
decimal	Соответствует десятичному значению.	{x:decimal}
double	Соответствует 64-разрядному значению с плавающей запятой.	{x:double}
float	Соответствует 32-разрядному значению с плавающей запятой.	{x:float}
guid	Соответствует значению GUID.	{x:guid}
int	Соответствует 32-разрядному целочисленному значению.	{x:int}
length	Соответствует строке с указанной длиной или в пределах заданного диапазона длин.	{x:length(6)} {x:length(1,20)}
long	Соответствует 64-разрядному целочисленному значению.	{x:long}
max	Сопоставляет целое число с максимальным значением.	{x:max(10)}
maxlength	Соответствует строке с максимальной длиной.	{x:maxlength(10)}
min	Сопоставляет целое число с минимальным значением.	{x:min(10)}
minlength	Соответствует строке с минимальной длиной.	{x:minlength(10)}
range	Соответствует целым числам в диапазоне значений.	{x:range(10,50)}
regex	Соответствует регулярному выражению.	{x:regex(^\\d{3}-\\d{3}-\\d{4}\$)}

Обратите внимание, что некоторые ограничения, например "min", принимают аргументы в скобках. К параметру можно применить несколько ограничений, разделенных двоеточием.

Рисунок 4. Поддерживаемые ограничения маршрутов.

Атрибуты маршрутизации

Взглянем немного под другим углом на методы GET и PUT, но в этот раз обратим внимание на параметры самого метода:

Листинг 4. Атрибуты GET- и PUT-метода.

```
[HttpGet]
[Authorize]
public async Task<IActionResult> GetAll([FromQuery] CommentQueryObject queryObject)
{
    ...
}

[HttpPut("{id:int}")]
public async Task<IActionResult> Update([FromRoute] int id, [FromBody] UpdateCommentRequestDto updateDto)
```

```
{  
  ...  
}
```

Атрибуты маршрутизации также включают [FromRoute], [FromQuery] и [FromBody], которые управляют способом передачи параметров в методы контроллера.

В приведенных выше методах наблюдается тонкая настройка атрибутов маршрутизации, позволяющая разделить источники данных, передаваемых клиентом. Например, метод получения всех комментариев использует атрибут [FromQuery] для объекта запроса, что дает возможность извлекать параметры непосредственно из строки запроса. Такой подход особенно полезен для реализации фильтрации, пагинации или сортировки, поскольку параметры могут быть переданы динамически через URL, не затрагивая основное тело запроса (например, URL-адрес GET-запроса может выглядеть следующим образом: 'https://localhost:7174/api/Comment?Symbol=msft&IsDescending=true').

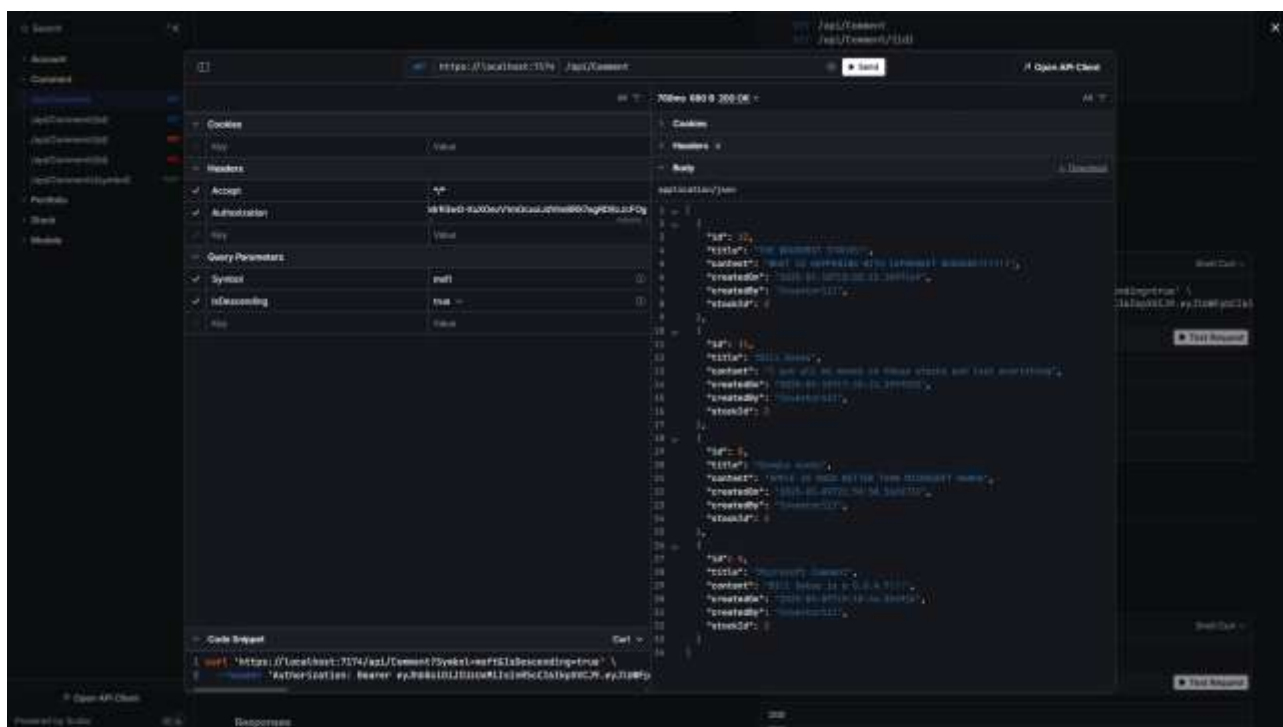


Рисунок 5. Пример GET-запроса.

В случае метода обновления комментария (Update) применяется комбинация атрибутов: [FromRoute] для извлечения числового идентификатора, который задается ограничением {id:int} в маршруте, и [FromBody] для получения дан-

ных обновления, переданных в теле HTTP-запроса. Это четкое разделение источников данных позволяет не только обеспечить корректную обработку каждого отдельного параметра, но и способствует повышению безопасности, исключая возможность подмены данных, полученных из URL, и не допуская ошибок при их обработке.

Приложение

Листинг 5. Листинг контроллера CommentController.

```
using backend.Dtos.Comment;
using backend.Extensions;
using backend.Helpers;
using backend.Interfaces;
using backend.Mappers;
using backend.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace backend.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class CommentController : ControllerBase
    {
        private readonly ICommentRepository _commentRepo;
        private readonly IStockRepository _stockRepo;
        private readonly UserManager<AppUser> _userManager;
        private readonly IFMPService _fmpService;

        public CommentController(ICommentRepository commentRepo, IStockRepository stockRepo, UserManager<AppUser> userManager, IFMPService fmpService)
        {
            _commentRepo = commentRepo;
            _stockRepo = stockRepo;
            _userManager = userManager;
            _fmpService = fmpService;
        }

        [HttpGet]
        [Authorize]
        public async Task<IActionResult> GetAll([FromQuery] CommentQueryObject queryObject)
        {
            if (!ModelState.IsValid)
                return BadRequest(ModelState);
        }
    }
}
```

```

        var comments = await _commentRepo.GetAllAsync(queryObject);

        var commentsDto = comments.Select(c => c.ToCommentDto());

        return Ok(commentsDto);
    }

    [HttpGet("{id:int}")]
    public async Task<IActionResult> GetById([FromRoute] int id)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        var comment = await _commentRepo.GetByIdAsync(id);

        if (comment is null)
            return NotFound();

        return Ok(comment.ToCommentDto());
    }

    [HttpPost("{symbol:alpha}")]
    [Authorize]
    public async Task<IActionResult> Create([FromRoute] string symbol, Create-
CommentDto commentDto)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        var stock = await _stockRepo.GetBySymbolAsync(symbol);

        if (stock is null)
        {
            stock = await _fmpService.FindStockBySymbolAsync(symbol);

            if (stock is null)
                return BadRequest("Stock does not exist!");
            else
                await _stockRepo.CreateAsync(stock);
        }

        var username = User.GetUsername();
        var appUser = await _userManager.FindByNameAsync(username);

        var commentModel = commentDto.ToCommentFromCreate(stock.Id);
        commentModel.AppUserId = appUser.Id;

        await _commentRepo.CreateAsync(commentModel);
    }

```

```

        return CreatedAtAction(nameof(GetById), new { id = commentModel.Id },
commentModel.ToCommentDto());
    }

    [HttpPut("{id:int}")]
    public async Task<IActionResult> Update([FromRoute] int id, [FromBody] Up-
dateCommentRequestDto updateDto)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        var comment = await _commentRepo.UpdateAsync(id, updateDto.ToCom-
mentFromUpdate());

        if (comment is null)
            return NotFound("Comment not found!");

        return Ok(comment.ToCommentDto());
    }

    [HttpDelete("{id:int}")]
    public async Task<IActionResult> Delete([FromRoute] int id)
    {
        if (!ModelState.IsValid)
            return BadRequest(ModelState);

        var commentModel = await _commentRepo.DeleteAsync(id);

        if (commentModel is null)
            return NotFound("Comment does not exist!");

        return NoContent();
    }
}
}

```

Список источников

1. YouTube. Плейлист "Backend-разработка API на ASP.NET Core". URL: <https://www.youtube.com/playlist?list=PL82C6-O4XrHcNJd4ejg8pX5fZaIDZmXyn> (дата обращения: 13.03.2025).
2. Microsoft Learn. Настройка маршрутизации в ASP.NET Web API [Электронный ресурс]. URL: <https://learn.microsoft.com/ru-ru/aspnet/web-api/overview/web-api-routing-and-actions/> (дата обращения: 14.03.2025).

3. Нуман Балох. Использование маршрутизации в ASP.NET Web API [Электронный ресурс]. URL: <https://nouman.microsoft.com/ru-ru/aspnet/web-api/overview/web-api-routing-and-actions/> (дата обращения: 14.03.2025).