



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки

09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 17

Дисциплина: «Backend»

Тема: Изучение кэширования в веб-приложении на основе ASP.NET Core

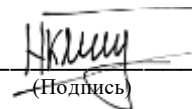
Выполнил: студент группы: 231-339

Карапетян Нвер Каренович

(Фамилия И.О.)

Дата, подпись: 04.05.25

(Дата)


(Подпись)

Проверил: _____

(Фамилия И.О., степень, звание)

(Оценка)

Дата, подпись _____

(Дата)

(Подпись)

Москва
2025

Цель:

Ознакомиться с механизмами кеширования в ASP.NET Core для оптимизации производительности веб-приложений.

Задачи:

- Изучить основные типы кэширования доступные в ASP.NET Core: внутренний кэш, кэш памяти, кэш диска, распределенный кэш.
- Реализовать простые примеры использования каждого типа кэширования в веб-приложении.
- Протестировать работу кэша при различных сценариях, включая частые запросы, изменение данных и истечение времени жизни кэша.
- Оценить влияние использования кеширования на производительность и время ответа веб-приложения.

Ход работы

Кэширование позволяет значительно снизить нагрузку на базу данных и улучшить время отклика веб-приложения за счет хранения результатов дорогостоящих операций (например, запросов к БД) в более быстром хранилище. В ASP.NET Core доступны четыре основных механизма кэширования: встроенный in-memory cache (памятный кэш), распределенный кэш (distributed cache), диск-базируемый кэш на уровне приложения и кеширование HTTP-ответов (response cache). В ходе этой лабораторной работы были последовательно реализованы и протестированы примеры каждого из этих видов.

Настройка приложения и сервисов

Перед тем как приступить к контроллеру, в файле **Program.cs** были добавлены все необходимые сервисы и настроено логирование. Сначала подключаются службы кэширования:

Листинг 1. Подключение через систему Dependency Injection сервисов кэширования.

```
builder.Services.AddMemoryCache();           // Встроенный In-Memory кэш
builder.Services.AddDistributedMemoryCache(); // Распределенный кэш
builder.Services.AddResponseCaching();        // Кэширование HTTP-ответов
```

После этого очищаются стандартные провайдеры логирования и настраивается Serilog, чтобы выводить сообщения в консоль и файлы:

Листинг 2. Настройка сервисов логирования.

```
builder.Logging.ClearProviders();

builder.Logging.AddConsole();
builder.Logging.AddDebug();

Log.Logger = new LoggerConfiguration()
    .MinimumLevel.Debug()
    .WriteTo.Console()
    .WriteTo.File("Logs/log-.txt", rollingInterval: RollingInterval.Day)
    .WriteTo.File(new Serilog.Formatting.Json.JsonFormatter(), "Logs/structured-
.json", rollingInterval: RollingInterval.Day)
    .CreateLogger();

builder.Host.UseSerilog();
```

И последним этапом настройки приложения является подключение соответствующих middleware для Response Caching и Serilog Request Logging:

Листинг 3. Подключение необходимых middleware.

```
app.UseSerilogRequestLogging();
app.UseResponseCaching();
```

Реализация In-Memory Cache

В методе `GetProducts()` контроллера сначала проверяется, есть ли в `IMemoryCache` список товаров по ключу `"ProductsList"`. Если данные найдены, в лог записывается сообщение об успешном попадании в кэш, и возвращается сохраненный список. При первом обращении кэш пуст, поэтому список загружается из базы и сохраняется в память на 30 секунд:

Листинг 4. GET-запрос с реализацией кэширования во внутренней памяти.

```
// GET: api/products
```

```
[HttpGet]
public async Task<ActionResult<List<Product>>> GetProducts()
{
    if (_memoryCache.TryGetValue(ProductsCacheKey, out List<Product> products))
    {
        _logger.LogWarning($"MemoryCache: найдено в кэше {products.Count} това-
ров");
    }
    else
    {
        _logger.LogWarning("MemoryCache: читаем из базы данных...");
        products = await _context.Products.ToListAsync();
        _memoryCache.Set(ProductsCacheKey, products, TimeSpan.FromSeconds(30));
        _logger.LogWarning($"MemoryCache: закешировано {products.Count} товаров на
30 секунд");
    }

    return Ok(products);
}
```

Эта схема отлично подходит для тех данных, которые часто запрашива-
ются и редко меняются в пределах 30 секунд (или любого другого промежутка
времени).

```
21:47:30 DBG: A data reader for 'Products' on server 'tcp://localhost:5432' is being disposed after spending 14ms reading results.
21:47:30 DBG: Closing connection to database 'Products' on server 'tcp://localhost:5432'.
21:47:30 DBG: Closed connection to database 'Products' on server 'tcp://localhost:5432' (11ms).
21:47:30 WRN: MemoryCache: закешировано 4 товаров на 30 секунд
21:47:30 DBG: List of registered output formatters, in the following order: ["Microsoft.AspNetCore.Mvc.Formatters.HttpNoContentOutputFormatter", "Microsoft.AspNetCore.Mvc.F
ormatters.StringOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StreamOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.SystemTextJsonOutputFormatter"]
21:47:30 DBG: No information found on request to perform content negotiation.
21:47:30 DBG: Attempting to select an output formatter without using a content type as no explicit content types were specified for the response.
21:47:30 DBG: Attempting to select the first formatter in the output formatters list which can write the result.
21:47:30 DBG: Selected output formatter 'Microsoft.AspNetCore.Mvc.Formatters.SystemTextJsonOutputFormatter' and content type 'application/json' to write the response.
21:47:30 INF: Executing ObjectResult, writing value of type 'System.Collections.Generic.List`1[[Label7.Models.Product, Label7, Version=1.0.0.0, Culture=neutral, PublicKey
Token=null]]'.
21:47:30 DBG: Response is not cacheable because it does not contain the 'public' cache directive.
21:47:30 INF: Executed action 'Label7.Controllers.ProductsController.GetProducts (Label7)' in 2530.788 ms
21:47:30 INF: Executed endpoint 'Label7.Controllers.ProductsController.GetProducts (Label7)'
21:47:30 INF: The response could not be cached for this request.
21:47:30 INF: HTTP GET /api/Products responded 200 in 2542.785 ms
21:47:30 DBG: 'HttpContext' disposed.
21:47:30 DBG: Disposing connection to database 'Products' on server 'tcp://localhost:5432'.
21:47:30 DBG: Disposed connection to database 'Products' on server 'tcp://localhost:5432' (1ms).
21:47:30 INF: Request finished HTTP/2 GET https://localhost:7039/api/Products - 200 text/plain; charset=utf-8 2788.458ms
21:47:34 DBG: Connection id "04NCE7GWRW0B5" received FIN.
21:47:34 DBG: Connection id "04NCE7GWRW0B5" is closed. The last processed stream ID was 1.
21:47:34 DBG: The connection queue processing loop for 04NCE7GWRW0B5 completed.
21:47:34 DBG: Connection id "04NCE7GWRW0B5" sending FIN because: "The socket transport's send loop completed gracefully."
21:47:34 DBG: Connection id "04NCE7GWRW0B5" stopped.
21:47:40 DBG: Connection id "04NCE7GWRW0B7" accepted.
21:47:40 DBG: Connection id "04NCE7GWRW0B7" started.
21:47:40 DBG: Connection 04NCE7GWRW0B7 established using the following protocol: Tls12
21:47:40 INF: Request starting HTTP/2 GET https://localhost:7039/api/Products - 401 400ms
21:47:40 DBG: 1 candidate(s) found for the request path '/api/Products'
21:47:40 DBG: Endpoint 'Label7.Controllers.ProductsController.GetProducts (Label7)' with route pattern 'api/Products' is valid for the request path '/api/Products'
21:47:40 DBG: Request matched endpoint 'Label7.Controllers.ProductsController.GetProducts (Label7)'
21:47:40 INF: No cached response available for this request.
21:47:40 INF: Executing endpoint 'Label7.Controllers.ProductsController.GetProducts (Label7)'
21:47:40 INF: Route matched with {action = 'GetProducts', controller = 'Products'}. Executing controller action with signature System.Threading.Tasks.Task`1[Microsoft.AspNetCore.M
vc.ActionResult`1[System.Collections.Generic.List`1[[Label7.Models.Product, Label7, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]]]] GetProducts() on controller Label7.C
ontrollers.ProductsController (Label7).
21:47:40 DBG: Execution plan of authorization filters (in the following order): ["None"]
21:47:40 DBG: Execution plan of resource filters (in the following order): ["None"]
21:47:40 DBG: Execution plan of action filters (in the following order): ["Microsoft.AspNetCore.Mvc.ModelBinding.UnsupportedContentTypeFilter (Order: -3888)", "Microsoft.As
pNetCore.Mvc.Infrastructure.ModelStateInvalidFilter (Order: -2888)"]
21:47:40 DBG: Execution plan of exception filters (in the following order): ["None"]
21:47:40 DBG: Execution plan of result filters (in the following order): ["Microsoft.AspNetCore.Mvc.Infrastructure.ClientErrorResultFilter (Order: -2000)"]
21:47:40 DBG: Executing controller factory for controller Label7.Controllers.ProductsController (Label7)
21:47:40 DBG: Executed controller factory for controller Label7.Controllers.ProductsController (Label7)
21:47:40 WRN: MemoryCache: найдено в кэше 4 товаров
21:47:40 DBG: List of registered output formatters, in the following order: ["Microsoft.AspNetCore.Mvc.Formatters.HttpNoContentOutputFormatter", "Microsoft.AspNetCore.Mvc.F
ormatters.StringOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StreamOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.SystemTextJsonOutputFormatter"]
21:47:40 DBG: No information found on request to perform content negotiation.
21:47:40 DBG: Attempting to select an output formatter without using a content type as no explicit content types were specified for the response.
```

Рисунок 1. Примеры консольных логов (строки, начинающиеся с желтой метки WRN).

Распределенный кэш (Distributed Cache)

Для метода `GetProductById(int id)` использован `IDistributedCache`. Первоначально пытаемся получить сериализованный JSON-объект товара по ключу, формируемому как `"Product_{id}"`. При отсутствии в кэше считываем товар из базы, сериализуем его в JSON и сохраняем в распределенный кэш на 60 секунд. При повторном запросе данные будут доставаться не из БД, а из кэша:

Листинг 5. GET-запрос с кэшированием данных о товаре в распределенном кэше.

```
// GET: api/products/{id}
[HttpGet("{id}")]
public async Task<ActionResult<Product>> GetProductById(int id)
{
    string cacheKey = $"Product_{id}";
    Product product = null;
    var cached = await _distributedCache.GetStringAsync(cacheKey);

    if (!string.IsNullOrEmpty(cached))
    {
        _logger.LogWarning($"DistributedCache: найден в распределенном кэше товар с
ключом {cacheKey}");
        product = JsonSerializer.Deserialize<Product>(cached)!;
    }
    else
    {
        _logger.LogWarning($"DistributedCache: в кэше не найден продукт по ключу
{cacheKey}. Читаем из базы данных...");
        product = await _context.Products.FirstOrDefaultAsync(p => p.Id == id);

        if (product is null)
            return NotFound("Продукт не найден");

        var options = new DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(60)
        };

        await _distributedCache.SetStringAsync(cacheKey, JsonSerializer.Serialize(
product), options);
        _logger.LogWarning($"DistributedCache: продукт с ключом {cacheKey} закэши-
рован на 60 секунд");
    }

    return Ok(product);
}
```

Такой подход позволяет масштабировать приложение в нескольких экземплярах: все они будут обращаться к единому распределенному хранилищу (в примере — в памяти, но в продакшене это может быть Redis).

```

21:58:17.000: Closed connection to database 'Products' on server 'tcp://localhost:5432' (10ms).
21:58:17.000: DistributedCache: manager: 1 success: 0 error: 0
21:58:17.000: List of registered output formatters, in the following order: ["Microsoft.AspNetCore.Mvc.Formatters.HttpOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StringOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StreamOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.JsonOutputFormatter"]
21:58:17.000: No information found on request to perform content negotiation.
21:58:17.000: Attempting to select an output formatter without using a content type as no explicit content types were specified for the response.
21:58:17.000: Attempting to select the first formatter in the output formatters list which can write the result.
21:58:17.000: Selected output formatter 'Microsoft.AspNetCore.Mvc.Formatters.JsonOutputFormatter' and content type 'application/json' to write the response.
21:58:17.000: Executing OwinResult, writing value of type 'Label7.Models.Product'.
21:58:17.000: Response is not cacheable because it does not contain the 'public' cache directive.
21:58:17.000: Executed action 'Label7.Controllers.ProductsController.GetProductById (Label7)' in 201.0037ms
21:58:17.000: Executed endpoint 'Label7.Controllers.ProductsController.GetProductById (Label7)'
21:58:17.000: The response could not be cached for this request.
21:58:17.000: HTTP GET /api/products/2 responded 200 in 183.1982 ms
21:58:17.000: 'ProductContext' disposed.
21:58:17.000: Disposing connection to database 'Products' on server 'tcp://localhost:5432'.
21:58:17.000: Disposed connection to database 'Products' on server 'tcp://localhost:5432' (4ms).
21:58:17.000: Request finished HTTP/2 GET https://localhost:7035/api/products/2 - 200 - application/json, charset=utf-8 201.0037ms
21:58:19.000: Connection id '0000000000000000' received FIN.
21:58:19.000: Connection id '0000000000000000' is closed. The last processed stream ID was 1.
21:58:19.000: The connection queue processing loop for 0000000000000000 completed.
21:58:19.000: Connection id '0000000000000000' sending FIN because 'The socket transport's send loop completed gracefully.'
21:58:19.000: Connection id '0000000000000000' stopped.
21:58:20.000: Connection id '0000000000000000' started.
21:58:20.000: Connection id '0000000000000000' started.
21:58:20.000: Connection 0000000000000000 established using the following protocol: HTTP
21:58:20.000: Request starting HTTP/2 GET https://localhost:7035/api/products/2 - 00000000
21:58:20.000: Candidate(s) found for the request path '/api/products/2':
21:58:20.000: Endpoint 'Label7.Controllers.ProductsController.GetProductById (Label7)' with route pattern 'api/products/{id}' is valid for the request path '/api/products/2'
21:58:20.000: Request matched endpoint 'Label7.Controllers.ProductsController.GetProductById (Label7)'
21:58:20.000: No cached response available for this request.
21:58:20.000: Executing endpoint 'Label7.Controllers.ProductsController.GetProductById (Label7)'
21:58:21.000: Route matched with {action = 'GetProductById', controller = 'Products'}. Executing controller action with signature System.Threading.Tasks.Task<Microsoft.AspNetCore.Mvc.ActionResult<IQueryable<Label7.Models.Product>>> Label7.Controllers.ProductsController.GetProductById (Label7)
21:58:21.000: Execution plan of authorization filters (in the following order): 'None'
21:58:21.000: Execution plan of resource filters (in the following order): 'None'
21:58:21.000: Execution plan of action filters (in the following order): ["Microsoft.AspNetCore.Mvc.ModelBinding.SuppressModelStateRequiredFilter (Order: -9999)", "Microsoft.AspNetCore.Mvc.Infrastructure.NoModelStateRequiredFilter (Order: -2000)"]
21:58:21.000: Execution plan of exception filters (in the following order): 'None'
21:58:21.000: Execution plan of result filters (in the following order): ["Microsoft.AspNetCore.Mvc.Infrastructure.EliminateEmptyResultFilter (Order: -2000)"]
21:58:21.000: Executing controller factory for controller 'Label7.Controllers.ProductsController (Label7)'
21:58:21.000: Executed controller factory for controller 'Label7.Controllers.ProductsController (Label7)'
21:58:21.000: Attempting to bind parameter 'id' of type 'System.Int32' ...
21:58:21.000: Attempting to bind parameter 'id' of type 'System.Int32' using the name 'id' in request data ...
21:58:21.000: Done attempting to bind parameter 'id' of type 'System.Int32'.
21:58:21.000: Done attempting to bind parameter 'id' of type 'System.Int32'.
21:58:21.000: Attempting to validate the bound parameter 'id' of type 'System.Int32' ...
21:58:21.000: Done attempting to validate the bound parameter 'id' of type 'System.Int32'.
21:58:21.000: DistributedCache: makes a noncacheable cache miss for 'Label7.Models.Product,2'
21:58:21.000: List of registered output formatters, in the following order: ["Microsoft.AspNetCore.Mvc.Formatters.HttpOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StringOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.StreamOutputFormatter", "Microsoft.AspNetCore.Mvc.Formatters.JsonOutputFormatter"]
  
```

Рисунок 2. Примеры консольных логов (строки, начинающиеся с желтой метки WRN).

Кэширование на диске

Для демонстрации файлового кэша реализован метод `GetProductByIdDiskCache(int id)`, который хранит результат в файле `CacheFiles/product_{id}.json`. Если файл существует и моложе 45 секунд, данные читаются из файла, иначе снова запрашиваются из БД и сохраняются в файл:

Листинг 6. GET-запрос с кэшированием данных о товаре на диске в формате JSON-файлов.

```

// GET: api/products/disk/{id}
[HttpGet("disk/{id}")]
public async Task<ActionResult<Product>> GetProductByIdDiskCache(int id)
{
    var cacheFile = Path.Combine("CacheFiles", $"product_{id}.json");
    Product product = null;

    if (System.IO.File.Exists(cacheFile) && DateTime.UtcNow - System.IO.File.GetCreationTimeUtc(cacheFile) < TimeSpan.FromSeconds(45))
    {
        _logger.LogWarning($"DiskCache: найден файл {cacheFile}");
        var json = await System.IO.File.ReadAllTextAsync(cacheFile);
        product = JsonSerializer.Deserialize<Product>(json);
    }
}
  
```



```

else
{
    _logger.LogWarning($"DiskCache: в кэш диске не найден продукт (ID = {id}).  
Читаем из базы данных...");
    product = await _context.Products.FirstOrDefaultAsync(p => p.Id == id);

    if (product is null)
        return NotFound("Продукт не найден");

    Directory.CreateDirectory("CacheFiles");
    await System.IO.File.WriteAllTextAsync(cacheFile, JsonSerializer.Serialize(product));
    _logger.LogWarning($"DiskCache: записан файл {cacheFile}");
}

return Ok(product);
}

```

Файловый кэш удобен для данных, которые не изменяются слишком часто и должны сохраняться между перезапусками приложения.

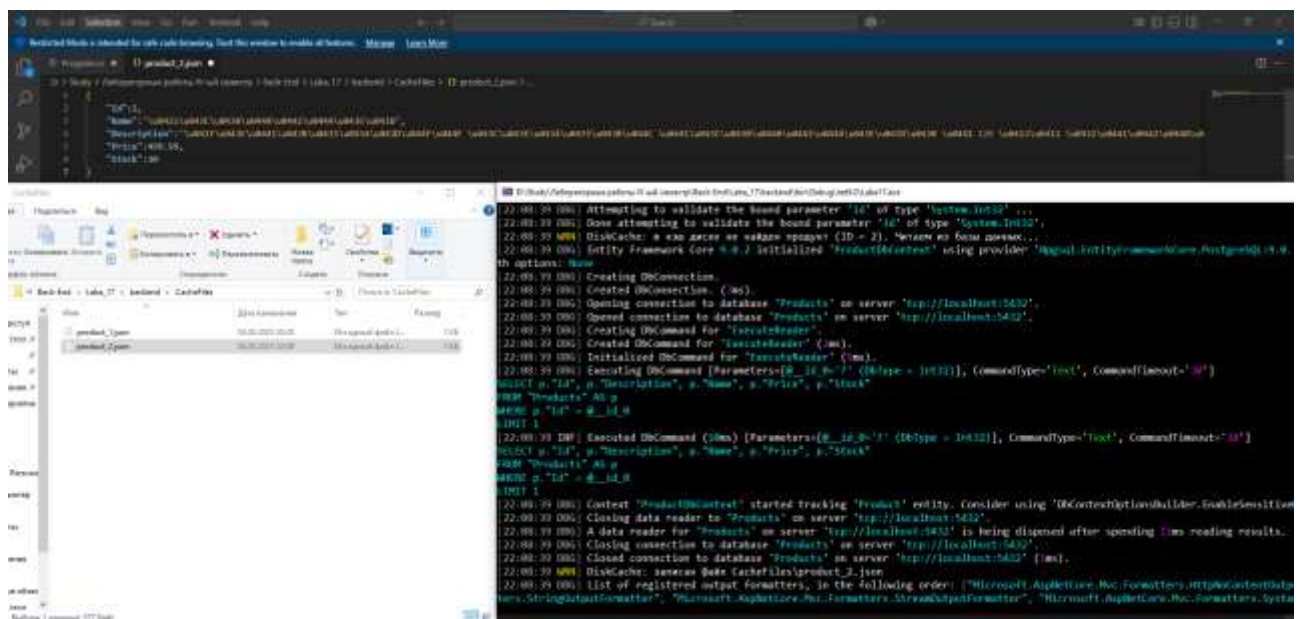


Рисунок 3. Демонстрация содержимого подобного JSON-файла. Примеры консольных логов.

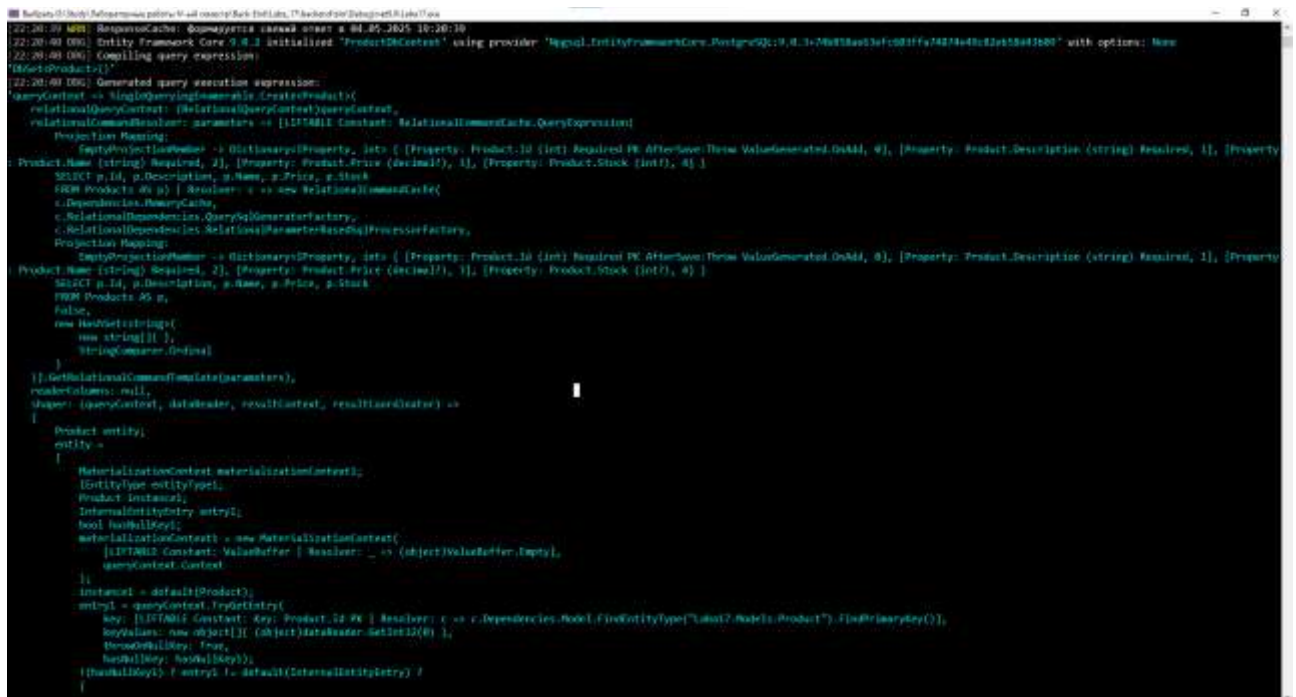
Кэширование HTTP-ответов (Response Caching)

HTTP-кэширование позволяет браузеру или прокси хранить готовый ответ. В методе `GetProductsResponseCache()` добавлен атрибут `[ResponseCache]` с параметром `Duration = 20 секунд`. `NoStore = false` означает, что заголовок `Cache-Control` не будет содержать директиву `no-store`. Если бы `NoStore = true`, сервер

бы добавил Cache-Control: no-store, запрещая любому узлу кэшировать ответ вовсе. Поскольку мы устанавливаем NoStore = false, кэширование разрешено, и в HTTP-заголовках мы увидим, например: Cache-Control: public, max-age=20. При первом запросе ответ формируется и кэшируется, а при повторном в течение 20 секунд сервер не будет выполнять бизнес-логику, а просто выдаст сохраненный ответ:

Листинг 7. GET-запрос с кэшированием HTTP-ответа.

```
// GET: api/products/response
[HttpGet("response")]
[ResponseCache(Duration = 20, Location = ResponseCacheLocation.Any, NoStore = false)]
public async Task<ActionResult<List<Product>>> GetProductsResponseCache()
{
    _logger.LogWarning($"ResponseCache: формируется свежий ответ в {DateTime.UtcNow}");
    var products = await _context.Products.ToListAsync();
    return Ok(products);
}
```



```
22:20:00.000 [INFO] ResponseCache: формируется свежий ответ в 04.05.2025 10:20:00
22:20:00.000 [INFO] Entity Framework Core 9.0.1 initialized 'ProductDbContext' using provider 'Microsoft.EntityFrameworkCore.Sqlite' with options: None
22:20:00.000 [INFO] Compiling query expression:
[HttpGetProducts()]
22:20:00.000 [INFO] Generated query execution expression:
QueryContext => SingleQuerying<IEnumerable<Product>>().CreateQueryContext(
    relationalQueryContext: [RelationalQueryContext]QueryContext,
    relationalCommandBuilder: parameters => [15776411 Constants RelationalCommandCache.QueryExpression]
)
Projection Mapping:
    QueryProjectionValue -> Dictionary<Property, ISet> { [Property: Product.ID (int) Required PK AfterSave: Throw ValueGenerated.OnAdd, 0], [Property: Product.Description (string) Required, 1], [Property: Product.Name (string) Required, 2], [Property: Product.Price (decimal), 1], [Property: Product.Stock (int), 0] }
    SELECT p.ID, p.Description, p.Name, p.Price, p.Stock
    FROM Products AS p | Resolver: c => new RelationalCommandCache(
        c.Dependencies, QueryContext,
        c.RelationalDependencies, QueryGeneratorFactory,
        c.RelationalDependencies, RelationalParameterFactory,
        c.RelationalDependencies, RelationalCommandCache
    )
    Projection Mapping:
    QueryProjectionValue -> Dictionary<Property, ISet> { [Property: Product.ID (int) Required PK AfterSave: Throw ValueGenerated.OnAdd, 0], [Property: Product.Description (string) Required, 1], [Property: Product.Name (string) Required, 2], [Property: Product.Price (decimal), 1], [Property: Product.Stock (int), 0] }
    SELECT p.ID, p.Description, p.Name, p.Price, p.Stock
    FROM Products AS p
    false
    new HashSet<string>()
    new string[] { }
    string.Empty.ToString()
    }
    }
    GetRelationalCommandCache(parameters),
    readerColumns: null,
    (queryContext, dataReader, resultContext, resultDataReader) =>
    {
        Product entity;
        entity =
        {
            MaterializationContext materializationContext;
            EntityType entityType;
            Product instance;
            InternalEntityType entityType;
            bool hasNullKey;
            materializationContext = new MaterializationContext(
                [15776411 Constants ValueBuffer [ Resolver: _ => (object)ValueBuffer.Empty],
                queryContext.Context
            );
            instance = default(Product);
            entityType = queryContext.TryGetEntityType(
                key: [15776411 Constants Key: Product.ID PK | Resolver: c => c.Dependencies.Model.FindEntityType("Lama7.Models.Product").FindPrimaryKey(),
                keyValues: new object[] { (object)dataReader.GetString(0) },
                throwOnNullKey: true,
                hasNullKey: hasNullKey);
            hasNullKey = entityType != default(InternalEntityType) ?
        }
```

Рисунок 4. Примеры консольных логов.