**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**



**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-разработка

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

**Выполнил: студент группы 231-332**

**Чесноков Александр Евгеньевич**

(Фамилия И.О. обучающегося)

**Дата, подпись** 01.10.2025 

(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.

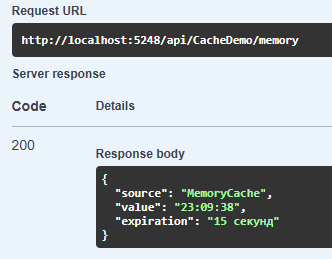
**Ход работы:**

В начале работы был создан новый проект типа ASP.NET Core Web API, в котором были зарегистрированы сервисы кеширования с помощью методов AddMemoryCache() и AddDistributedMemoryCache(). Для удобства тестирования API в проект был добавлен и настроен Swagger, обеспечивающий визуальный интерфейс для отправки запросов и просмотра ответов сервера. Конфигурация Swagger выполнялась в Program.cs, где был указан документ версии v1 и добавлено отображение по адресу корня приложения.

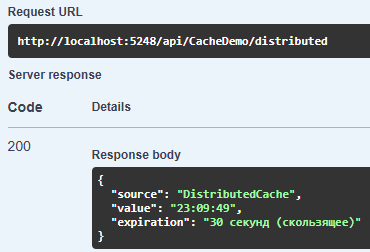
В контроллере CacheDemoController были реализованы три метода для демонстрации различных сценариев использования кеша. Метод GetMemoryCache() использует встроенный кэш памяти IMemoryCache, где при первом вызове в кэш записывается текущее время с ограничением по времени жизни в 15 секунд. При последующих обращениях возвращается сохранённое значение, что позволяет оценить работу механизма хранения данных. Метод GetDistributedCache() демонстрирует работу с IDistributedCache, где данные сохраняются в распределённом кеше с использованием скользящего времени жизни в 30 секунд. Для удобства в качестве реализации был использован встроенный DistributedMemoryCache, но при необходимости он может быть заменён на Redis или SQL Server.

Дополнительно был реализован метод ClearCache(), который очищает оба типа кеша, что упрощает тестирование сценариев обновления данных.

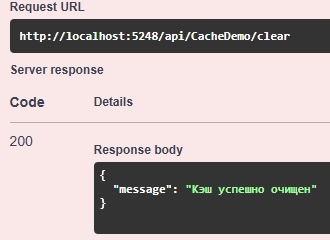
**Скриншоты работы приложения:**memory:



distributed:



Принудительная очистка кеша (также очищается по истечении времени):



**Листинг:**

**Program.cs:**

|  |
| --- |
| using Microsoft.Extensions.Caching.Memory;  using Microsoft.Extensions.Caching.Distributed;  using Microsoft.OpenApi.Models;  var builder = WebApplication.CreateBuilder(args);  // *Добавляем сервисы контроллеров и кеширования*  builder.Services.AddControllers();  builder.Services.AddMemoryCache();              // *внутренний кэш (in-memory)*  builder.Services.AddDistributedMemoryCache();   // *распределённый кэш (имитация через память)*  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen();  var app = builder.Build();  if (app.Environment.IsDevelopment())  {      app.UseSwagger();      app.UseSwaggerUI();  }  app.MapControllers();  app.Run(); |

**Controllers/CacheDemoController.cs:**

|  |
| --- |
| using Microsoft.AspNetCore.Mvc;  using Microsoft.Extensions.Caching.Memory;  using Microsoft.Extensions.Caching.Distributed;  using System.Text;  namespace Lab17.Controllers;  [ApiController]  [Route("api/[controller]")]  public class CacheDemoController : ControllerBase  {      private readonly IMemoryCache \_memoryCache;      private readonly IDistributedCache \_distributedCache;      public CacheDemoController(IMemoryCache memoryCache, IDistributedCache distributedCache)      {          \_memoryCache = memoryCache;          \_distributedCache = distributedCache;      }      // *Пример 1 — внутренний кэш (MemoryCache)*      [HttpGet("memory")]      public IActionResult GetMemoryCache()      {          const string cacheKey = "current\_time";          if (!\_memoryCache.TryGetValue(cacheKey, out string? currentTime))          {              currentTime = DateTime.Now.ToString("HH:mm:ss");              \_memoryCache.Set(cacheKey, currentTime, TimeSpan.FromSeconds(15)); // *истекает через 15 секунд*          }          return Ok(new          {              Source = "MemoryCache",              Value = currentTime,              Expiration = "15 секунд"          });      }      // *Пример 2 — распределённый кэш (DistributedCache)*      [HttpGet("distributed")]      public async Task<IActionResult> GetDistributedCache()      {          const string cacheKey = "distributed\_time";          var cachedData = await \_distributedCache.GetAsync(cacheKey);          string? timeString;          if (cachedData == null)          {              timeString = DateTime.Now.ToString("HH:mm:ss");              var bytes = Encoding.UTF8.GetBytes(timeString);              var options = new DistributedCacheEntryOptions()                  .SetSlidingExpiration(TimeSpan.FromSeconds(30)); // *обновляется при каждом обращении*              await \_distributedCache.SetAsync(cacheKey, bytes, options);          }          else          {              timeString = Encoding.UTF8.GetString(cachedData);          }          return Ok(new          {              Source = "DistributedCache",              Value = timeString,              Expiration = "30 секунд (скользящее)"          });      }      // *Пример 3 — очистка кеша*      [HttpDelete("clear")]      public async Task<IActionResult> ClearCache()      {          \_memoryCache.Remove("current\_time");          await \_distributedCache.RemoveAsync("distributed\_time");          return Ok(new          {              Message = "Кэш успешно очищен"          });      }  } |

**Ссылка на репозиторий:** <https://github.com/avoryyy/backend_mospoly>