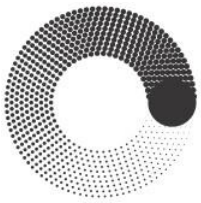
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки 09.03.02
«Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № _18__

Дисциплина: BackEnd-разработка
_____

_____

Тема: *Мониторинг веб-приложения на основе ASP.NET Core*_____

Выполнил(а): студент(ка) группы __221-3711_____

_____Костоваров А.С._____
(Фамилия И.О.)

Дата, подпись _____ _____
(Дата) (Подпись)

Проверил: _____ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись _____ _____
(Дата) (Подпись)

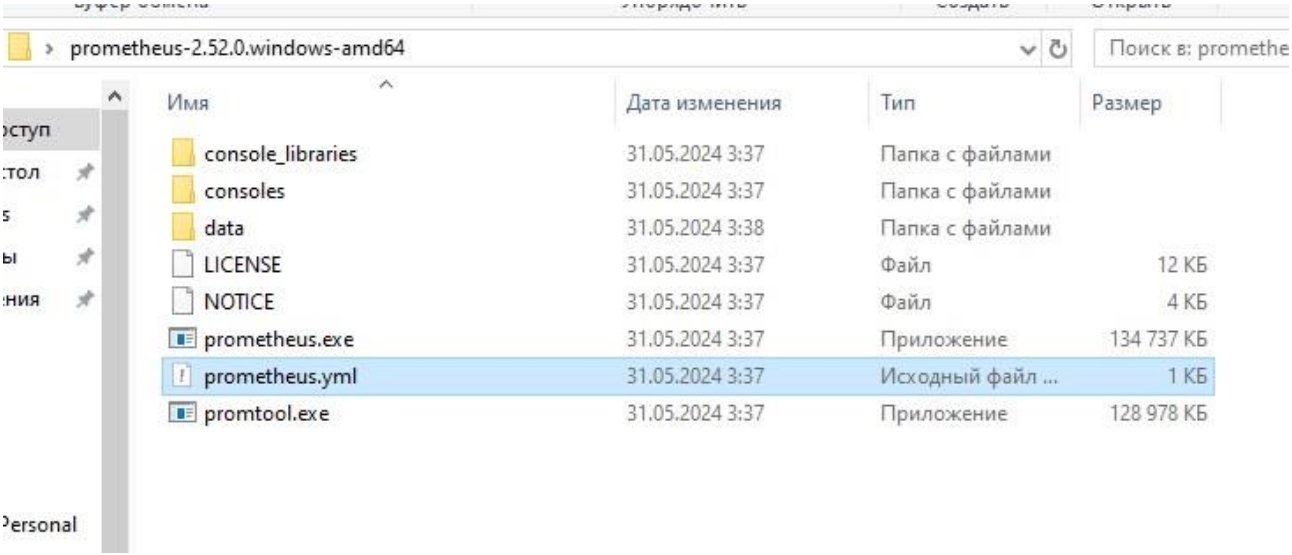
Замечания: _____
_____

Москва

2024

Для мониторинга работы приложения, возьмем приложение из лабораторной работы номер 17, будем использовать Prometheus для сбора метрик и их визуализации.

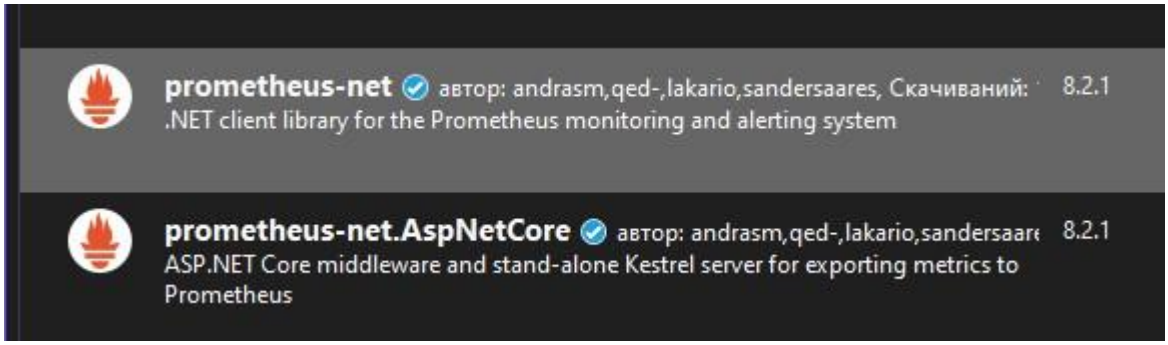
Скачиваем Prometheus с официального сайта и распаковываем архив.



В файле prometheus.yml настроим подключение к адресу нашего приложения.

```
> Users > mikha > Desktop > prometheus-2.52.0.windows-amd64 > ! prometheus.yml
# my global config
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
  # scrape_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:9090"]

  # Scrape configuration for ASP.NET Core application
  - job_name: "aspnetcore_app"
    scrape_interval: 5s
    metrics_path: "/metrics"
    scheme: "https"
    static_configs:
      - targets: ["localhost:7015"]
```

Добавляем наш локал хост. Далее перейдем в приложение и добавим использование сбора метрик, для этого сначала устанавливаем 2 необходимых пакета.



Добавляем данные строки в файл program.

```
// Настройка Prometheus для сбора метрик
app.UseMetricServer();
app.UseHttpMetrics();
```

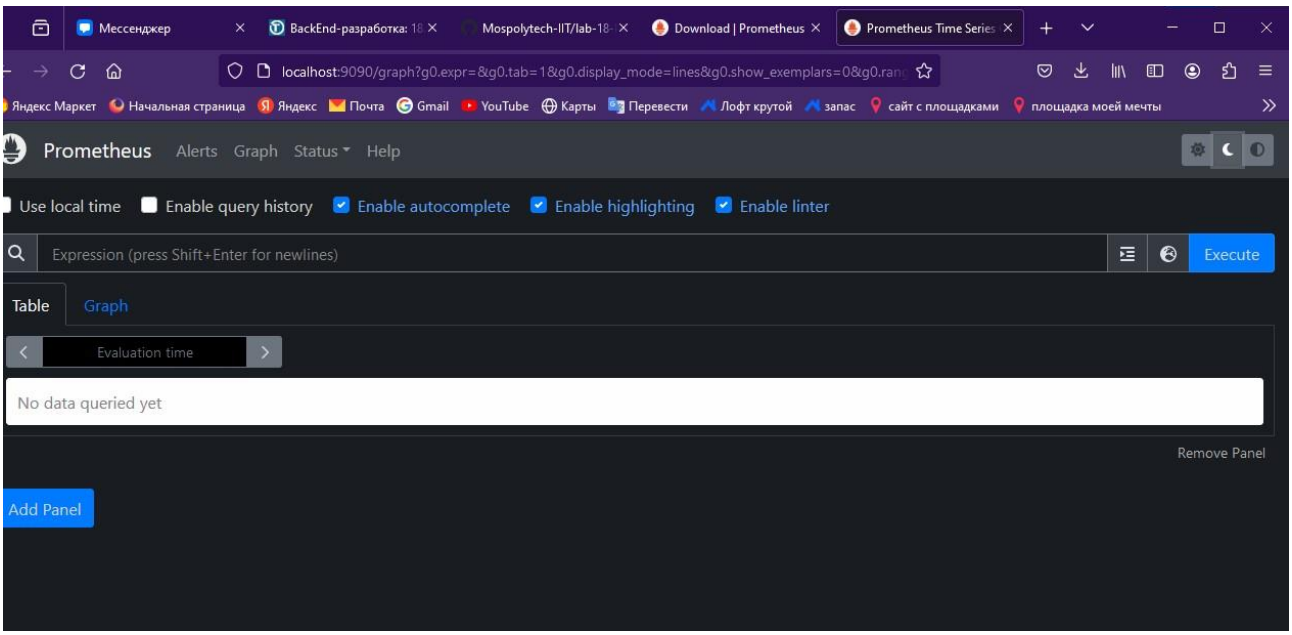
А так же в котроллере настроим кастомные метрики для получения информации о количестве запросов, использовании памяти и использовании ресурсов процессора.

```
private static readonly Counter TotalRequests = Metrics.CreateCounter("total_requests", "Total number of requests");
private static readonly Gauge MemoryUsage = Metrics.CreateGauge("memory_usage_bytes", "Current memory usage in bytes");
private static readonly Gauge CpuUsage = Metrics.CreateGauge("cpu_usage_seconds_total", "Total CPU usage in seconds");
```

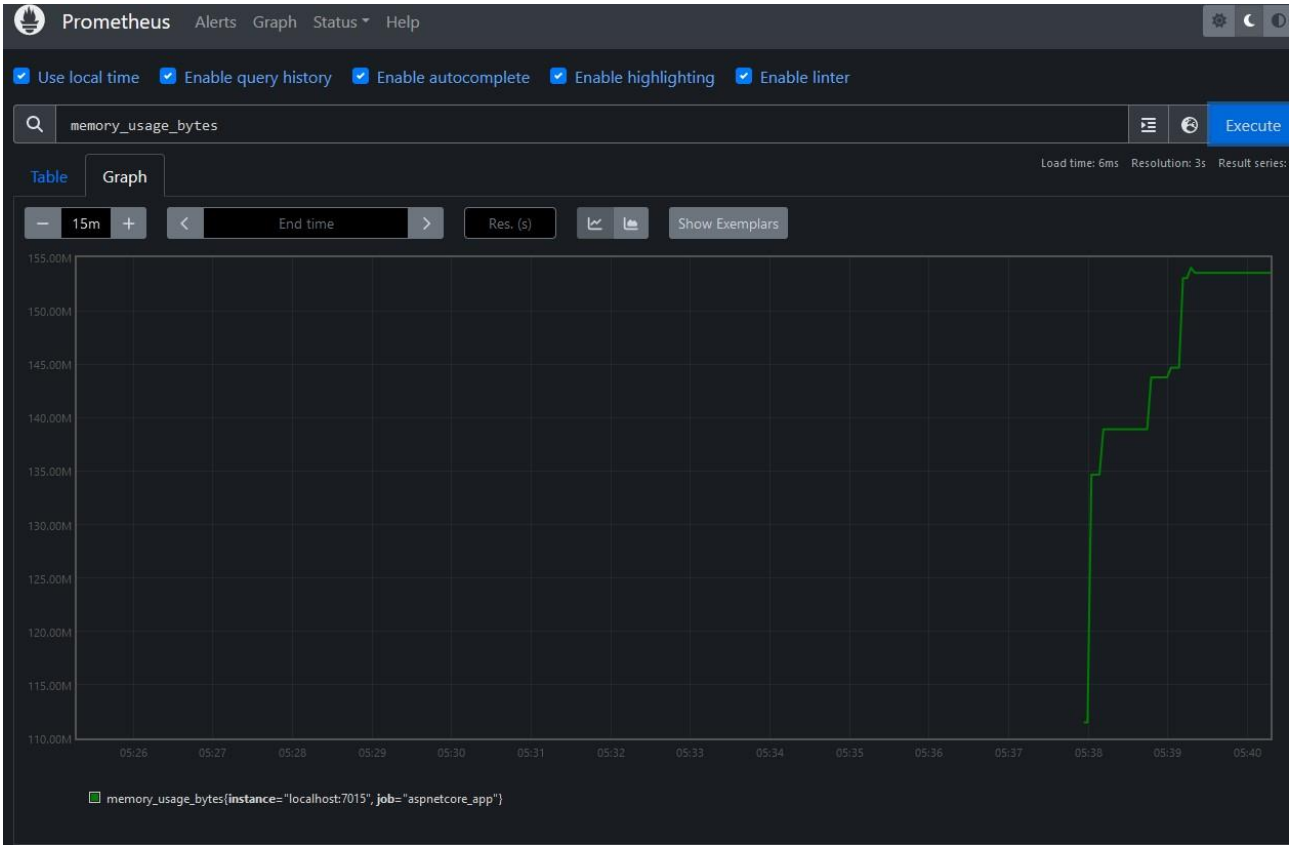
И добавим их ко всем действиям контроллера.

```
// Получение всех примеров из кэша памяти
[HttpGet("MemoryCache")]
Ссылок: 0
public IActionResult GetExamplesMemoryCache()
{
    TotalRequests.Inc();
    MemoryUsage.Set(Process.GetCurrentProcess().WorkingSet64);
    CpuUsage.Set(Process.GetCurrentProcess().TotalProcessorTime.TotalSeconds);

    // Измерение времени выполнения запроса
    Stopwatch sw = new Stopwatch();

    sw.Start();
    var examples = exservice.GetExamplesMemoryCache();
    sw.Stop();

    // Передача данных во View
    ViewData["AllTablecache"] = examples;
    ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

    return View("Index");
}
```

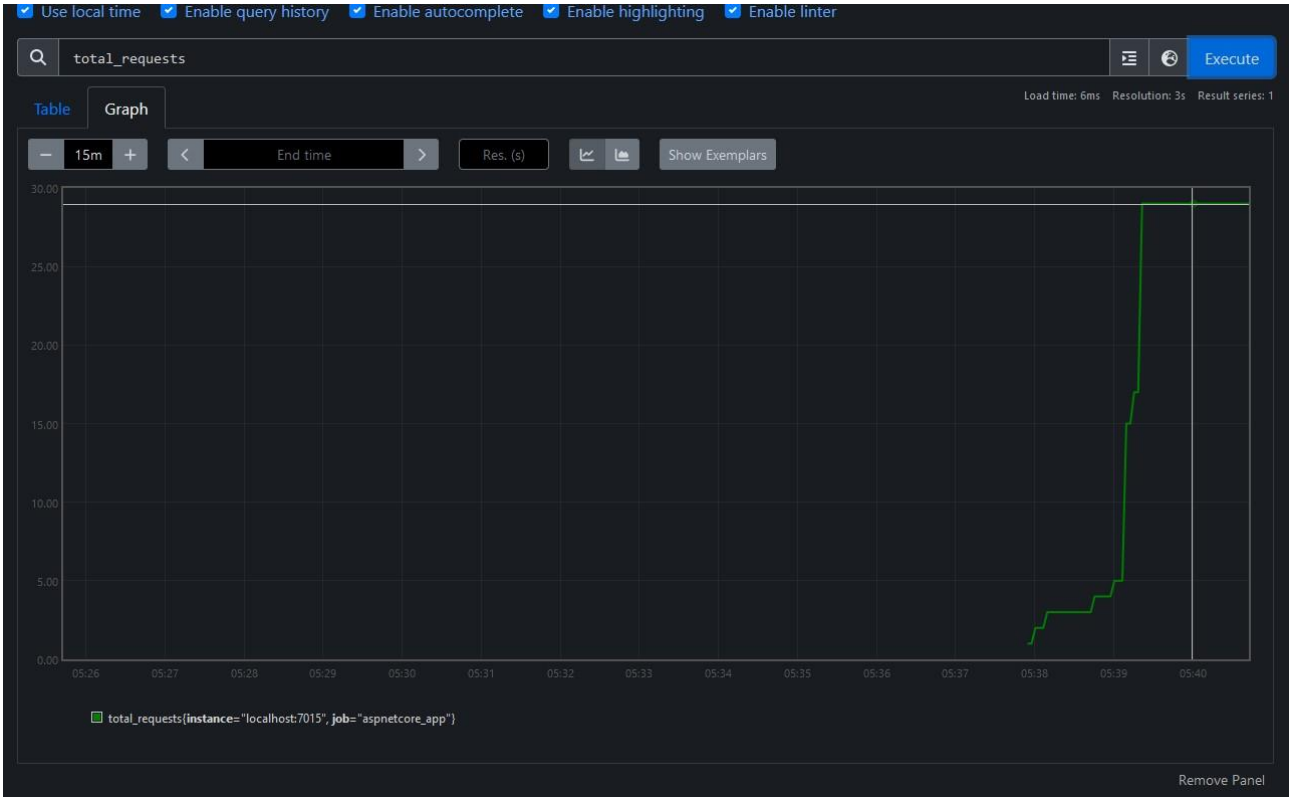
Теперь в консоли запустим Prometheus и перейдем по адресу в браузере.



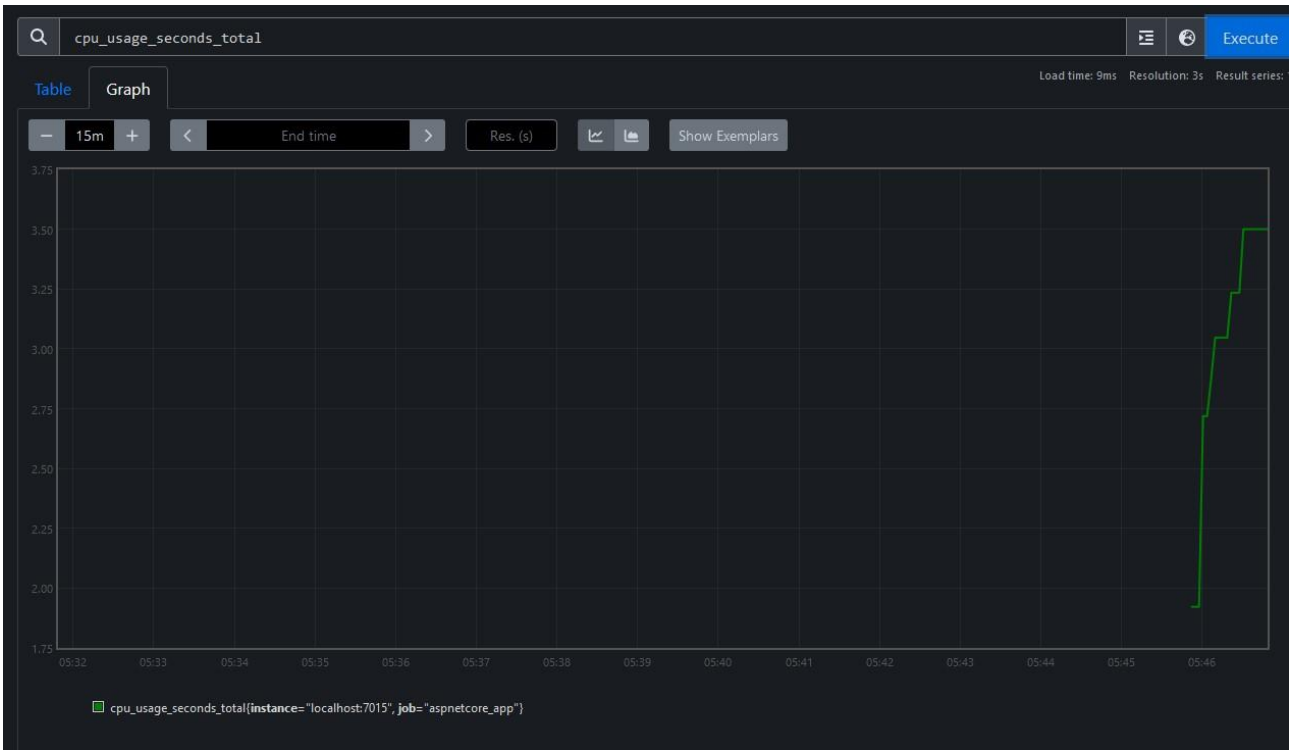
Видим окно для контроля метрик.



Введем название метрики и перейдем на вкладку граф, теперь мы видим график по нужной нам метрике, в данном случае использование памяти в течении последних 15 минут.



Так выглядит общее количество запросов.



А так время использования процессора.

Таким образом в реальном приложении будет удобно отслеживать активность на серверной части, в какое время больше всего запросов поступает, насколько загружены ресурсы сервера и тд.

Код с настройками Prometheus.

```
using Laba17;
using Microsoft.EntityFrameworkCore; using
Microsoft.Extensions.Caching.Distributed; using
Microsoft.Extensions.DependencyInjection; using
Prometheus;

var builder = WebApplication.CreateBuilder(args);

// Настройка подключения к базе данных MySQL var
connectionString =
"Server=127.0.0.1;Port=3306;Database=exampleshema;User=root;Password=pugpug12;";
builder.Services.AddDbContext<LABAcontext>(options =>
    options.UseMySql(connectionString, new MySqlServerVersion(new Version(8, 0, 36))));

// Настройка кэширования
builder.Services.AddMemoryCache(); // Внутренний кэш и кэш памяти
builder.Services.AddDistributedMemoryCache(); // Использование распределенного кэша в
памяти
builder.Services.AddControllersWithViews(); // Регистрация контроллеров и представлений
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(builder =>
    {
        builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
    });
});
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
app.UseRouting(); app.UseCors();
app.UseAuthentication();
app.UseAuthorization();
app.UseResponseCaching(); // Использование кэширования ответов

// Настройка Prometheus для сбора метрик
app.UseMetricServer(); app.UseHttpMetrics();

// Настройка маршрутизации app.UseRouting();

// Определение маршрута по умолчанию для контроллеров
app.MapControllerRoute(     name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.UseStaticFiles();

app.Run();


using Microsoft.AspNetCore.Mvc; using
Microsoft.Extensions.Caching.Memory; using
Microsoft.Extensions.Caching.Distributed; using
Prometheus; using System.Diagnostics;

namespace Laba17.Controllers
{
```

```
    [ApiController]
    [Route("api/[controller]")]
    public class HomeController : Controller
    {
        private static readonly Counter TotalRequests =
Metrics.CreateCounter("total_requests", "Total number of requests");
private static readonly Gauge MemoryUsage =
Metrics.CreateGauge("memory_usage_bytes", "Current memory usage in bytes");
private static readonly Gauge CpuUsage =
Metrics.CreateGauge("cpu_usage_seconds_total", "Total CPU usage in seconds");
        // Сервис для работы с кэшем
        ExampleService exservice;

        // Конструктор контроллера
        public HomeController(IMemoryCache cache, LABAcontext _context,
IDistributedCache distributedCache)
        {
            // Инициализация сервиса с кэшем памяти и распределенным кэшем
exservice = new ExampleService(cache, _context, distributedCache);         }

        // Получение всех примеров из базы данных
        [HttpGet("Database")]
        public IActionResult GetExamplesDatabase()
        {
            TotalRequests.Inc();
            MemoryUsage.Set(Process.GetCurrentProcess().WorkingSet64);
            CpuUsage.Set(Process.GetCurrentProcess().TotalProcessorTime.TotalSeconds);
            // Измерение времени выполнения запроса
            Stopwatch sw = new Stopwatch();

            sw.Start();
            var examples = exservice.GetExamplesDatabase();
sw.Stop();

            // Передача данных во View
            ViewData["AllTable"] = examples;
            ViewData["AllTime"] = sw.ElapsedMilliseconds;

            return View("Index");
        }

        // Получение всех примеров из кэша памяти
        [HttpGet("MemoryCache")]
        public IActionResult GetExamplesMemoryCache()
        {
            TotalRequests.Inc();
            MemoryUsage.Set(Process.GetCurrentProcess().WorkingSet64);
            CpuUsage.Set(Process.GetCurrentProcess().TotalProcessorTime.TotalSeconds);
            // Измерение времени выполнения запроса
            Stopwatch sw = new Stopwatch();

            sw.Start();
            var examples = exservice.GetExamplesMemoryCache();
sw.Stop();

            // Передача данных во View
            ViewData["AllTablecache"] = examples;
            ViewData["AllTimecache"] = sw.ElapsedMilliseconds;
```

```
            return View("Index");
        }

        // Получение всех примеров из распределенного кэша
        [HttpGet("DistributedCache")]
        public async Task<IActionResult> GetExamplesDistributedCache()
        {
            TotalRequests.Inc();
            MemoryUsage.Set(Process.GetCurrentProcess().WorkingSet64);
            CpuUsage.Set(Process.GetCurrentProcess().TotalProcessorTime.TotalSeconds);
            // Измерение времени выполнения запроса
            Stopwatch sw = new Stopwatch();

            sw.Start();
            var examples = await exservice.GetExamplesDistributedCacheAsync();
sw.Stop();

            // Передача данных во View
            ViewData["AllTablecache"] = examples;
            ViewData["AllTimecache"] = sw.ElapsedMilliseconds;

            return View("Index");
        }
    }
}
```