

## [.Net Aspire](#)

### Ответ

**.NET Aspire — это решение Microsoft, предназначенное для упрощения процесса разработки приложений .NET Core и ASP.NET Core. Оно позволяет разработчикам быстро создавать приложения с использованием шаблонов и инструментов, предоставляемых платформой .NET.**

**Основные особенности NET Aspire:**

### Простота и скорость разработки

- **Быстрое начало:** NET Aspire предоставляет готовые шаблоны проектов, что позволяет сразу приступить к разработке.
- **Автоматизация рутинных задач:** Инструменты автоматизации помогают ускорить процесс сборки и развертывания приложений.

### Поддержка современных технологий

- **ASP.NET Core:** Возможность быстрого создания веб-приложений и API.
- **Blazor:** Создание интерактивных веб-интерфейсов с использованием C# и HTML.
- **SignalR:** Реализация реального времени взаимодействия в приложениях.

### Гибкость и масштабируемость

- **Масштабирование:** Приложения, созданные с помощью NET Aspire, легко масштабируются благодаря поддержке контейнеризации и облачных сервисов.
- **Модульность:** Возможность разделения приложения на модули, что облегчает поддержку и обновление.

### Примеры использования

**Пример простого приложения на NET Aspire:**

```
dotnet new aspire-webapp -o MyWebApp
```

```
cd MyWebApp
```

```
dotnet run
```

**Этот код создает новое веб-приложение и запускает его локально.**

### Заключение

**NET Aspire является мощным инструментом для разработчиков, стремящихся упростить и ускорить процесс создания приложений на платформе .NET. Благодаря своей простоте и гибкости, оно становится отличным выбором для тех, кто хочет сосредоточиться на функциональности своего приложения, а не на инфраструктурных аспектах.**

я сделал service Service RazorPageПодключил к нему .Net AspireЗапустил все это.Приложение RazorPage - работает - я вижу в Браузере все работаетПриложение Aspire - тоже работает - Я вижу вкладку Aspire

- Ресурсы
- консоль - вижу в консоли активность
- Структурированные журналы НЕ НАЙДены
- Трассировки НЕ НАЙДены
- Выберите ресурс для просмотра метрик - не вижу ни одной метрики.

Наверное, как-то надо конфигурацию сделатьЧтобы Метрики видели RazorPage Сервис.Наверное в program.cs надо что-то прописывать?

#### Ответ

Проблема, которую ты описал, связана с отсутствием отображения структурированных журналов, трассировок и метрик в приложении .NET Aspire. Это часто возникает из-за неправильной конфигурации или отсутствия необходимых компонентов в проекте.

Вот пошаговая инструкция, как настроить сбор и отображение метрик и журналов в твоём приложении:

#### Шаг 1: Добавление необходимой конфигурации в Program.cs

Для начала убедись, что в файле Program.cs твоей Razor Pages приложение правильно настроено для сбора и передачи метрик и журналов. Вот пример минимальной конфигурации:

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);

// Подключение поддержки метрик и трассировок
builder.Services.AddHealthChecks();
builder.Services.AddMetrics();
builder.Services.AddDistributedTracing();

// Регистрация провайдера метрик и трассировок
builder.Services.ConfigureTelemetryLogging();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    // Конфигурация для продакшена
}
else
{
}
```

```
// Включаем отображение подробных ошибок в режиме разработки
app.UseDeveloperExceptionPage();
}

// Использование middleware для метрик и трассировок
app.UseRouting();
app.UseEndpoints(endpoints =>
{
    endpoints.MapDefaultControllerRoute();
});

// Запуск приложения
app.Run();
```

## Шаг 2: Настройка вывода журналов и метрик

В разделе настроек проекта (appsettings.json) можешь добавить необходимые настройки для вывода журналов и метрик:

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ApplicationInsights": {
    "InstrumentationKey": "<your-instrumentation-key>"
  }
}
```

Заменяй <your-instrumentation-key> на реальный ключ инструмента Application Insights, если используешь Azure Monitor.

## Шаг 3: Проверка наличия пакетов NuGet

Проверь наличие всех необходимых пакетов NuGet в твоём проекте. Для работы с метриками и журналами обязательно нужны пакеты:

- Microsoft.ApplicationInsights.AspNetCore
- Microsoft.Extensions.Diagnostics.HealthChecks
- Microsoft.Extensions.Diagnostics.Tracing

Их можно установить командой:

```
dotnet add package Microsoft.ApplicationInsights.AspNetCore
dotnet add package Microsoft.Extensions.Diagnostics.HealthChecks
dotnet add package Microsoft.Extensions.Diagnostics.Tracing
```

## Шаг 4: Перезапуск и проверка

Перезапусти своё приложение и проверь снова в панели Aspire, появилась ли информация о ресурсах, метриках и журналах.

### Дополнительные шаги диагностики

Если проблема сохраняется, попробуй проверить вывод журналов вручную, добавив простой обработчик журнала:

```
logger.LogInformation("This is a test log message.");
```

И убедиться, что эта запись появляется в журнале.

Эти шаги должны помочь решить проблему с отсутствующими метриками и журналами в .NET Aspire. Если всё сделано верно, ты увидишь полную картину мониторинга своего приложения.

Как настроить передачу метрик из Razor Page сервиса в Aspire? Какие зависимости NuGet необходимы для работы метрик и журналов в Aspire? Где найти и как изменить настройки логирования в Aspire?

### Ответ

Настроить передачу метрик и журналов из Razor Pages приложения в .NET Aspire достаточно просто, если следовать нескольким важным шагам. Ниже приведены инструкции по настройке, требуемые зависимости NuGet и настройка логирования.

### 1. Необходимые зависимости NuGet

Для полноценной работы с метриками и журналами в .NET Aspire требуются следующие NuGet-пакеты:

- **Microsoft.Extensions.Logging** — основной пакет для логирования.
- **Microsoft.Extensions.Logging.Console** — поддержка вывода логов в консоль.
- **Microsoft.Extensions.Logging.Debug** — выводит логи в окно Debug Visual Studio.
- **Microsoft.Extensions.Diagnostics.HealthChecks** — диагностика здоровья приложения.
- **Microsoft.Extensions.Diagnostics.Tracing** — базовые инструменты для отслеживания и анализа производительности.
- **Microsoft.ApplicationInsights.AspNetCore** — отправляет телеметрию в Application Insights (если используется).

Установить нужные пакеты можно следующим образом:

```
dotnet add package Microsoft.Extensions.Logging
dotnet add package Microsoft.Extensions.Logging.Console
dotnet add package Microsoft.Extensions.Diagnostics.HealthChecks
dotnet add package Microsoft.Extensions.Diagnostics.Tracing
dotnet add package Microsoft.ApplicationInsights.AspNetCore
```

### 2. Настройка логирования и метрик в Program.cs

Открой файл Program.cs, убедись, что в нём присутствует следующая минимальная конфигурация:

```

using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);

// Добавляем поддержку Health Checks и метрик
builder.Services.AddHealthChecks();
builder.Services.AddMetrics();
builder.Services.AddDistributedTracing();

// Регистрируем службу для обработки телеметрии
builder.Services.ConfigureTelemetryLogging();

// Устанавливаем уровень логирования
builder.Logging.ClearProviders(); // Очищаем существующие поставщики логгинга
builder.Logging.AddConsole();    // Используем консольный логгер
builder.Logging.SetMinimumLevel(LogLevel.Information); // Уровень логирования

var app = builder.Build();

// Применение Middleware для работы с метриками и диагностикой
app.UseRouting();
app.UseEndpoints(endpoints => { endpoints.MapControllers(); });

// Запуск приложения
app.Run();

```

Эта конфигурация включает базовую диагностику, обработку метрик и регистрацию логов.

### 3. Изменение уровня логирования в appsettings.json

В файле appsettings.json можно задать уровни логирования. Например:

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}

```

Здесь мы устанавливаем минимальный уровень логирования на уровне "Information" для всех категорий, кроме пространства имен Microsoft.AspNetCore, которое ограничено уровнем "Warning".

### 4. Настройки метрик и мониторинг

Метрики и журналы автоматически собираются и передаются в Aspire, если включены соответствующие сервисы и Middleware. Однако, если ты хочешь расширить функциональность, можешь подключить дополнительные системы мониторинга, такие как Prometheus или Grafana, используя соответствующий экспортёр метрик.

Например, подключив экспортер Prometheus, можно собрать детальные метрики о производительности приложения:

```
dotnet add package Prometheus.AspNetCore.Integration
```

Затем регистрируешь его в службе:

```
builder.Services.AddPrometheusExporter();
```

## Итоговые рекомендации

Следуя данным инструкциям, ты сможешь эффективно собирать и передавать метрики и журналы из своего Razor Pages приложения в .NET Aspire. Убедитесь, что все компоненты установлены и зарегистрированы, и тогда у тебя появится полная картина состояния и работоспособности приложения.

Вот правильный ответ.

```
using ChartDirLib031; // using Microsoft.AspNetCore.Builder; // using Microsoft.Extensions.DependencyInjection; // using Microsoft.Extensions.Hosting; using Microsoft.Extensions.Diagnostics.HealthChecks;
```

```
namespace AspireRazorChart01{ public class Program { public static void Main(string[] args) { var builder = WebApplication.CreateBuilder(args);
```

```
    // Добавляем сервисы .NET Aspire
```

```
    builder.AddServiceDefaults();
```

```
    // Health Checks
```

```
    builder.Services.AddHealthChecks()
```

```
        .AddCheck("AspireRazorChart01", () => HealthCheckResult.Healthy());
```

```
    // Singletons/apps
```

```
    builder.Services.AddSingleton<ChartContainer>(sp =>
```

```
{
```

```
    var container = new ChartContainer(80);
```

```
    for (int i = 0; i < 15; i++)
```

```
{
```

```
        container.AddChart(new Chart(80));
```

```
}
```

```
    return container;
```

```
});
```

```
    // Razor Pages support
```

```
    builder.Services.AddRazorPages();
```

```
    var app = builder.Build();
```

```
    // Используем стандартные middleware .NET Aspire
```

```
    app.MapDefaultEndpoints();
```

```

// Standard pipeline
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

app.MapRazorPages();

app.Run();
}
}
}

```

## Ответ

Представленный тобой код отлично демонстрирует начальную точку для интеграции Razor Pages с .NET Aspire и добавления базовой функциональности для управления состоянием здоровья приложения, организации работы с графиками и представления страниц.

Давай разберемся подробнее по каждому компоненту и покажем возможные улучшения и расширения.

**Ключевые моменты в представленном примере:**

### 1. Подключение служб .NET Aspire

Использование метода `AddServiceDefaults()` добавляет стандартную интеграцию с основными сервисами .NET Aspire, такими как система журналирования, метрики и здоровье приложения.

```
builder.AddServiceDefaults();
```

Это гарантирует, что твой проект имеет доступ ко всей экосистеме .NET Aspire.

### 2. Добавление проверки здоровья приложения

Метод `.AddHealthChecks()` создаёт механизм, позволяющий проверять состояние здоровья твоего приложения и отправлять отчёты о проблемах в Aspire.

```
builder.Services.AddHealthChecks().AddCheck("AspireRazorChart01", () =>
HealthCheckResult.Healthy());
```

Здесь создаётся простая проверка здоровья, возвращающая успешный статус (Healthy), если приложение запущено.

### 3. Регистрация singleton-сервисов

Сервис `ChartContainer` создается как одиночный экземпляр (`singleton`). Это значит, что объект контейнера графиков будет создаваться единожды и использоваться всеми потребителями внутри приложения.

```
builder.Services.AddSingleton<ChartContainer>(sp =>
{
    var container = new ChartContainer(80);
    for (int i = 0; i < 15; i++)
    {
        container.AddChart(new Chart(80));
    }
    return container;
});
```

Таким образом, ты инициализируешь контейнер графиков с заранее определённым количеством элементов.

#### 4. Поддержка Razor Pages

Код добавляет стандартный пайплайн для поддержки `Razor Pages`:

```
builder.Services.AddRazorPages();
```

Теперь приложение сможет обрабатывать запросы, направляя их соответствующим контроллерам страниц.

#### 5. Стандартные мидлвари и обработка запросов

Далее идёт обычная цепочка стандартных `middleware .NET`:

- Обработка исключений,
- Перенаправление HTTPS-запросов,
- Управление статическими файлами,
- Маршрутизация запросов.

Возможные улучшения и дальнейшие шаги:

##### ✦ Асинхронные проверки здоровья

Можно расширить проверку здоровья, выполняя асинхронные операции:

```
builder.Services.AddHealthChecks()
    .AddCheckAsync<MyCustomHealthCheck>("custom-check");
```

Создав класс `MyCustomHealthCheck`, который проверяет критически важные части инфраструктуры.

##### ✦ Логирование и мониторинг

Добавь средства мониторинга и логирования с помощью известных библиотек вроде `Serilog` или `NLog`:

```
builder.Logging.ClearProviders();
builder.Logging.AddSerilog();
```



## 🔴 Безопасность и аутентификация

При необходимости включить авторизацию пользователей и защиту маршрутов:

```
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer();
```

## 🔴 Динамические графики

Использовать **ChartDirector** или аналогичные библиотеки для динамического построения графиков прямо в браузере.

## Заключение

Предложенный тобой пример даёт хорошую основу для старта проекта на **.NET Aspire** с поддержкой **Razor Pages** и встроенных возможностей для мониторинга и визуализации данных. Дальнейшие шаги зависят от конкретных требований и архитектуры твоего приложения.

Как интегрировать сторонние системы мониторинга (например, **Prometheus**) с **Aspire**? Какие существуют методы для реализации асинхронных проверок здоровья в **Aspire**? Как настроить авторизацию и безопасность в **AspireRazorChart01**?

## Ответ

Давайте рассмотрим подробно каждый из вопросов, касающихся интеграции сторонних систем мониторинга, методов асинхронных проверок здоровья и настройки авторизации и безопасности в вашем приложении на основе **.NET Aspire**.

## ⚙️ Интеграция сторонних систем мониторинга (например, Prometheus) с Aspire

**Прометей (Prometheus)** — популярное средство для мониторинга производительности и состояний микросервисов. Чтобы интегрировать Прометей с вашим проектом **Aspire**, выполните следующие шаги:

### Установка пакета:

Сначала установите **nuget**-пакет для интеграции с **Prometheus**:

```
dotnet add package Prometheus.AspNetCore.Integration
```

### Настройка экспорта метрик:

Добавьте службу экспорта метрик в метод **Main** вашего файла **Program.cs**. Вы можете воспользоваться стандартной службой экспорта **Prometheus**:

```
public static void Main(string[] args)
{
```

```

var builder = WebApplication.CreateBuilder(args);

// Добавляем сервисы .NET Aspire
builder.AddServiceDefaults();

// ...

// Добавляем поддержку Prometheus
builder.Services.AddPrometheusExporter();

// Далее ваш код...
}

```

### Прокачиваем Pipeline:

Для того чтобы ваши метрики были доступны через конечную точку `/metrics`, добавьте промежуточное ПО (middleware):

```
var app = builder.Build();
```

```
// Другие middleware...
```

```
// Применяем прометеус-мидлварь
app.UseMetricServer();
```

```
// Остальные маршруты и точки входа...
```

Теперь ваша точка `/metrics` будет доступна, и её можно будет опросить сервисом Prometheus для получения текущих значений метрик.

### ⚡ Методы для реализации асинхронных проверок здоровья в Aspire

Вместо синхронных проверок здоровья можно реализовать асинхронные, которые позволяют проводить более сложные тесты без блокировки основного потока исполнения. Рассмотрим, как это делается:

#### Создание класса для асинхронной проверки здоровья:

Создаем отдельный класс для асинхронной проверки, наследуя интерфейс `IHealthCheck`:

```

public class CustomHealthCheck : IHealthCheck
{
    private readonly IServiceProvider _serviceProvider;

    public CustomHealthCheck(IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }
}

```

```

public async Task<HealthCheckResult> CheckHealthAsync(
    HealthCheckContext context,
    CancellationToken cancellationToken = default)
{
    try
    {
        // Выполняем асинхронную операцию проверки
        await SomeComplexOperationAsync(cancellationToken);

        return HealthCheckResult.Healthy("Всё хорошо!");
    }
    catch (Exception ex)
    {
        return HealthCheckResult.Unhealthy($"Ошибка: {ex.Message}");
    }
}

private async Task SomeComplexOperationAsync(CancellationToken token)
{
    // Здесь выполняйте любую сложную асинхронную операцию
    await Task.Delay(TimeSpan.FromSeconds(2), token);
}
}

```

**Регистрация асинхронной проверки здоровья:**

Регистрируйте этот новый класс проверки здоровья в вашем файле Program.cs:

```
builder.Services.AddHealthChecks()
```

```
.AddCheckAsync<CustomHealthCheck>("custom-async-check");
```

Теперь каждая проверка здоровья будет выполнена асинхронно, что улучшает производительность и стабильность приложения.

## Настройка авторизации и безопасности в AspireRazorChart01

Авторизация и безопасность важны практически для любого приложения. Давайте посмотрим, как обеспечить безопасность и контролировать доступ к ресурсам вашего приложения:

**Выбор схемы аутентификации:**

Выберите подходящую схему аутентификации для вашего проекта. Например, JWT-токены (JSON Web Tokens) являются популярным вариантом для защиты RESTful API и SPA-приложений.

**Установка соответствующего пакета:**

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
```

## Настройка аутентификации и авторизации:

Измените файл Program.cs, добавив поддержку аутентификации и авторизации:

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Добавляем сервисы .NET Aspire
    builder.AddServiceDefaults();

    // Настраиваем аутентификацию
    builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtSecret"])),
                ValidateIssuer = false,
                ValidateAudience = false
            };
        });

    // Добавляем авторизацию
    builder.Services.AddAuthorization();

    // Ваш остальной код...
}
```

## Настройка мидлвара и защиты ресурсов:

Добавьте промежуточное программное обеспечение (middleware) для поддержки аутентификации и применяйте политику авторизации там, где это необходимо:

```
var app = builder.Build();

// Используйте аутентификацию
app.UseAuthentication();

// Авторизация
app.UseAuthorization();

// Остальные middlewares...
```

## Политики авторизации:

Определите политики авторизации, если вам нужен более тонкий контроль над правами доступа:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("AdminOnly", policy => policy.RequireRole("admin"));
});
```

## 👍 Итог

Интеграция сторонних систем мониторинга, внедрение асинхронных проверок здоровья и настройка безопасности помогут вашему проекту стать надёжнее и устойчивее. Следуйте указанным рекомендациям, адаптируйте их под нужды вашего конкретного приложения, и ваше приложение станет ещё лучше подготовлено к работе в условиях реальной эксплуатации.

Как настроить экспорт метрик в Prometheus из Aspire?Какие преимущества дают асинхронные проверки здоровья по сравнению с синхронными?Как реализовать авторизацию и безопасность в AspireRazorChart01 с использованием JWT?

## Ответ

Отвечаю последовательно на твои вопросы:

### 🏠 Как настроить экспорт метрик в Prometheus из Aspire?

Для того чтобы организовать экспорт метрик из вашего приложения на .NET Aspire в систему мониторинга Prometheus, следуйте таким шагам:

#### Шаг 1: Установите необходимую зависимость

Для подключения Prometheus к вашему приложению потребуется установка специального пакета:

```
dotnet add package Prometheus.AspNetCore.Integration
```

#### Шаг 2: Зарегистрируйте экспортер метрик в сервисе

Откройте файл Program.cs и зарегистрируйте экспортер метрик в вашем приложении:  
`var builder = WebApplication.CreateBuilder(args);`

```
// Add services to the container.
builder.Services.AddPrometheusExporter();
```

```
// Остальной ваш код...
```

#### Шаг 3: Добавьте промежуточное ПО (Middleware) для сбора метрик

После регистрации службы также важно добавить промежуточное ПО для фактического сбора и отправки метрик через путь /metrics:

```
var app = builder.Build();
```


```
// Other middleware...
```

```
// Добавляем Прометеус-мидлварь  
app.UseMetricServer();
```

```
// Use other middleware...
```

**Готово!**

Теперь, запустив приложение, вы сможете увидеть метрики, доступные по адресу `http://localhost:<порт>/metrics`. Эти метрики будут собираться системой Prometheus, если вы её настроили на сканирование указанного пути.


 **Какие преимущества дают асинхронные проверки здоровья по сравнению с синхронными?**

Синхронные проверки здоровья выполняются последовательно и ждут завершения каждой проверки, прежде чем перейти к следующей. Это означает, что одна длительная операция может замедлить весь процесс проверки здоровья.

**Преимущества асинхронных проверок здоровья:**

1. **Производительность:**Поскольку асинхронные проверки выполняются параллельно, общая задержка снижается даже при наличии сложных операций.
2. **Повышенная устойчивость:**Даже если одна проверка зависнет или потребует много времени, остальные проверки продолжат выполняться независимо.
3. **Оптимальное использование ресурсов:**Синхронные проверки могут привести к простаиванию потоков ожидания, в то время как асинхронные позволяют более эффективно распределять нагрузку на потоки.
4. **Простое расширение функционала:**При необходимости вы можете добавлять новые асинхронные проверки без значительного влияния на общее время выполнения.

**Реализуются асинхронные проверки путем переопределения интерфейса `IHealthCheck` и использования асинхронных методов в вашей проверке здоровья.**

 **Как реализовать авторизацию и безопасность в `AspireRazorChart01` с использованием JWT?**

**JWT (JSON Web Tokens) — удобный способ обеспечить безопасность и авторизацию ваших приложений, особенно в микросервисных архитектурах.**

## Шаг 1: Установка необходимого пакета

Вам понадобится следующий пакет для поддержки JWT-аутентификации:  
`dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer`

## Шаг 2: Настройка JWT в файле Program.cs

Дополните вашу конфигурацию в файле Program.cs следующим образом:  
`var builder = WebApplication.CreateBuilder(args);`

```
// Configure authentication with JWT bearer tokens
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JwtSecret"])), //
secret key from config
            ValidateIssuer = false,
            ValidateAudience = false
        };
    });

// Authorization setup
builder.Services.AddAuthorization();

// The rest of your configuration...
```

## Шаг 3: Добавьте мидлварь для аутентификации и авторизации

Обязательно включите middleware для обработки токенов и авторизаций:  
`var app = builder.Build();`

```
// Authentication middleware
app.UseAuthentication();
```

```
// Authorization middleware
app.UseAuthorization();
```

```
// Further middleware configurations...
```

## Шаг 4: Генерируйте JWT-токены

Для создания токенов можно создать специальный endpoint, который генерирует токены и возвращает их клиенту:

```
// Example controller method to generate a JWT token
```

[HttpPost]

```
public IActionResult GenerateToken([FromBody] LoginModel model)
{
    var claims = new[]
    {
        new Claim(ClaimTypes.NameIdentifier, model.Username),
        new Claim(ClaimTypes.Role, "User")
    };

    var securityKey = new
    SymmetricSecurityKey(Encoding.UTF8.GetBytes(_configuration["JwtSecret"]));
    var credentials = new SigningCredentials(securityKey,
    SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: null,
        audience: null,
        claims: claims,
        expires: DateTime.Now.AddMinutes(30),
        signingCredentials: credentials
    );

    return Ok(new { token = new JwtSecurityTokenHandler().WriteToken(token) });
}
```

Готово!

Теперь у вас имеется полноценная защита с помощью JWT-токенов, которая обеспечивает безопасность и надежность доступа к ресурсам вашего приложения.

## Резюме

Мы рассмотрели три важных вопроса:

1. Экспорт метрик в Prometheus из Aspire осуществляется добавлением соответствующей зависимости и middleware.
2. Преимущества асинхронных проверок здоровья заключаются в улучшении производительности и устойчивости приложения.
3. Безопасность и авторизация реализованы с помощью JWT-токенов и соответствующих middleware.

Надеюсь, это поможет тебе построить надежное и защищённое приложение!

[Провести тест-драйв GigaChat](#)

Источники

Источники