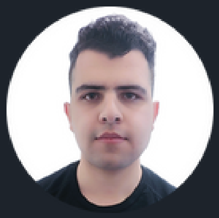
Health Checks | ASP.NET Core

Checking system components' health is vital in microservices and distributed systems.

```
builder.Services.Configure<DatabaseOptions>(
  builder.Configuration.GetSection("Database"));

builder.Services.AddHealthChecks()
  .AddCheck<RemoteHealthCheck>("Remote Endpoints Health Check")
  .AddCheck<DatabaseHealthCheck>("Database Health Check")
  .AddCheck<MemoryHealthCheck>($"Memory Health Check");
```

```
using HealthChecks.UI.Client;

app.UseHealthChecks("/health", new HealthCheckOptions()
{
  Predicate = _ => true,
  ResponseWriter =
    UIResponseWriter.WriteHealthCheckUIResponse
});
```

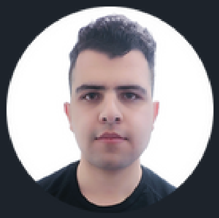


Elliot One

Health Checks JSON Output

AspNetCore.HealthChecks.UI.Client library generate JSON output for health checks.

```json
{
  "status": "Healthy",
  "totalDuration": "00:00:00.9410936",
  "entries": {
    "Remote Endpoints Health Check": {
      "data": {

      },
      "description": "Remote endpoints is healthy.",
      "duration": "00:00:00.9357820",
      "status": "Healthy",
      "tags": []
    },
    "Database Health Check": {
      "data": {

      },
      "description": "Database connection is healthy.",
      "duration": "00:00:00.1778689",
      "status": "Healthy",
      "tags": []
    },
    "Memory Health Check": {
      "data": {
        "AllocatedBytes": 7655896,
        "Gen0Collections": 0,
        "Gen1Collections": 0,
        "Gen2Collections": 0
      },
      "description": "Flags high memory use above 1073741824 bytes.",
      "duration": "00:00:00.0012378",
      "status": "Healthy",
      "tags": []
    }
  }
}
```

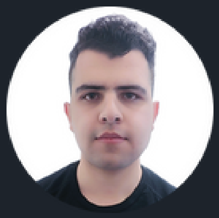


Elliot One

Remote Endpoint Health Check

Checking the health of a remote endpoint by sending an HTTP request and verifying the response status.

```
public class RemoteHealthCheck : IHealthCheck
{
  private readonly IHttpClientFactory _httpClientFactory;
  public RemoteHealthCheck(IHttpClientFactory httpClientFactory)
  {
    _httpClientFactory = httpClientFactory;
  }
  public async Task<HealthCheckResult> CheckHealthAsync(
    HealthCheckContext context,
    CancellationToken cancellationToken = new CancellationToken())
  {
    using var httpClient = _httpClientFactory.CreateClient();
    var response = await httpClient.GetAsync(
      "https://api.ipify.org", cancellationToken);

    if (response.IsSuccessStatusCode)
    {
      return HealthCheckResult.Healthy($"Remote endpoints is healthy.");
    }

    return HealthCheckResult.Unhealthy("Remote endpoint is unhealthy");
  }
}
```

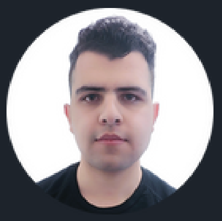


Elliot One

Database Health Check

Verifying database connectivity and reporting health status based on the connection result.

```
public class DatabaseHealthCheck : IHealthCheck
{
  private readonly IOptionsMonitor<DatabaseOptions> _options;

  public DatabaseHealthCheck(IOptionsMonitor<DatabaseOptions> options)
  {
    _options = options;
  }

  public async Task<HealthCheckResult> CheckHealthAsync(
    HealthCheckContext context,
    CancellationToken cancellationToken = default)
  {
    await using var connection =
      new SqlConnection(_options.CurrentValue.ConnectionString);

    try
    {
      await connection.OpenAsync(cancellationToken);
      return HealthCheckResult.Healthy("Database connection is healthy.");
    }
    catch (Exception ex)
    {
      return HealthCheckResult.Unhealthy("Database connection failed.", ex);
    }
  }
}

public class DatabaseOptions
{
  public string? ConnectionString { get; set; }
}
```

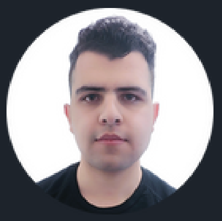


Elliot One

Memory Health Check

Checking memory usage against a specified threshold.

```csharp
public class MemoryHealthCheck : IHealthCheck
{
  private readonly IOptionsMonitor<MemoryCheckOptions> _options;

  public MemoryHealthCheck(IOptionsMonitor<MemoryCheckOptions> options)
  {
    _options = options;
  }

  public string Name => "memory_check";

  public Task<HealthCheckResult> CheckHealthAsync(
    HealthCheckContext context,
    CancellationToken cancellationToken = default(CancellationToken))
  {
    var options = _options.Get(context.Registration.Name);

    // Include GC information in the reported diagnostics.
    var allocated = GC.GetTotalMemory(forceFullCollection: false);
    var data = new Dictionary<string, object>()
    {
      { "AllocatedBytes", allocated },
      { "Gen0Collections", GC.CollectionCount(0) },
      { "Gen1Collections", GC.CollectionCount(1) },
      { "Gen2Collections", GC.CollectionCount(2) },
    };

    var status = (allocated < options.Threshold)
      ? HealthStatus.Healthy : HealthStatus.Unhealthy;
    string description =
      $"Flags high memory use above {options.Threshold} bytes.";

    return Task.FromResult(new HealthCheckResult(
      status,
      description: description,
      exception: null,
      data: data));
  }
}

public class MemoryCheckOptions
{
  // Failure threshold (in bytes)
  // Default to 1 GB
  public long Threshold { get; set; } = 1024L * 1024L * 1024L;
}
```



Elliot One

Elliot One

Enjoyed Reading This?

Reshare and Spread Knowledge.