Rate Limiting Middleware In .NET 7

Introduction

Rate limiting middleware is a software component used to control the rate of requests made to a web application or API. It is designed to prevent resource overuse, and abuse of the API, or ensure fair usage for all users. This type of middleware is added to the pipeline of an application. It is responsible for monitoring the rate of requests made by a user or system, and then blocking or throttling requests that exceed a certain threshold.

What is Rate Limit?

Rate limiting is a technique used to control the rate at which a user or system can access a resource. This is typically done to prevent resource overuse, abuse of an API, or ensure fair usage for all users. Rate limiting is often implemented by monitoring the number of requests made by a user or system within a specific time interval, and then taking action when the number of requests exceeds a specified threshold. This action can include blocking or throttling the requests.

For example, an API might allow only a certain number of requests per minute from a single user, in order to prevent that user from overwhelming the API with too many requests. Or a website might limit the number of login attempts per hour to prevent brute force attacks. Rate limiting can be implemented at various layers of a system, such as at the server or network level, or at the application level using middleware.

Rate limiting in ASP.NET Core

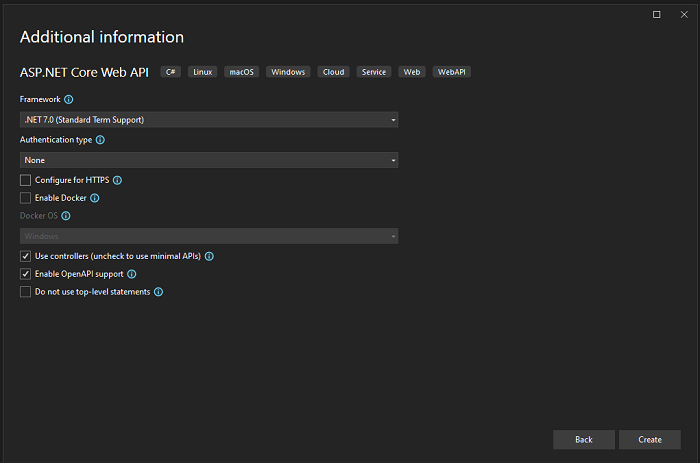
In [ASP.NET Core](https://www.c-sharpcorner.com/topics/asp-net-core), rate limiting can be implemented using middleware. Middleware is a component that sits between the client and the server and handles requests and responses. To implement rate limiting in ASP.NET Core, we can create a custom middleware component that checks the rate of requests for each user and blocks or throttles requests that exceed a certain threshold.

If your application is using .NET 7 (or higher), a rate limiting middleware is available out of the box. It provides a way to apply rate limiting to your web application and API endpoints.

Here's an example of how to implement a rate limiting middleware in ASP.NET Core.

**STEP 1**

We have to create an Asp.net Core Web API Project



**STEP 2**

Now comes in Program.cs file and add a simple rate limiter that limits all to 20 requests per minute, per authenticated username (or hostname if not authenticated):

builder.Services.AddRateLimiter(options => {

options.GlobalLimiter = PartitionedRateLimiter.Create < HttpContext, string > (httpContext => RateLimitPartition.GetFixedWindowLimiter(partitionKey: httpContext.User.Identity?.Name ?? httpContext.Request.Headers.Host.ToString(), factory: partition => new FixedWindowRateLimiterOptions {

AutoReplenishment = true,

PermitLimit = 20,

QueueLimit = 0,

Window = TimeSpan.FromMinutes(1)

}));

});

var app = builder.Build();

app.UseRateLimiter();

JavaScript

Copy

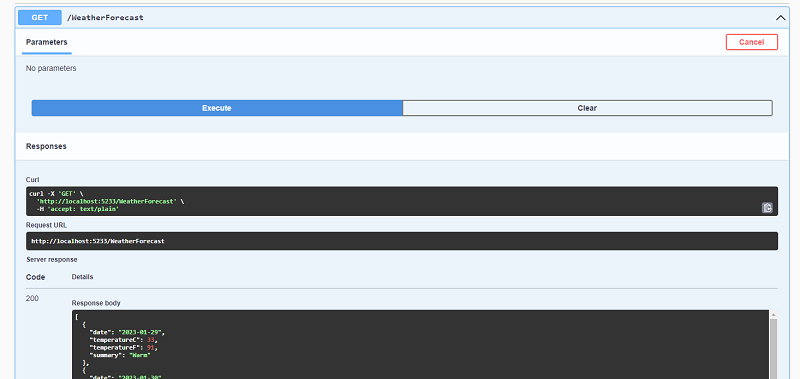
Now explain the above code.

builder.Services.AddRateLimiter

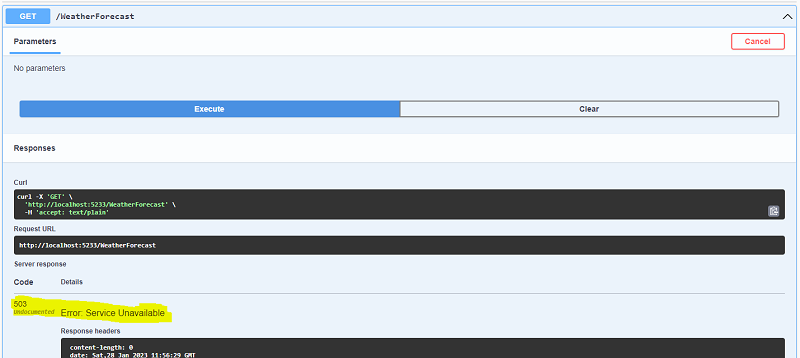
The method is used to configure and register the rate limiter service with the application's service container. Once added to the application, the rate limiter can be used to control access to certain routes or endpoints, ensuring that they are not overwhelmed by too many requests.

Now if we want to set a global rate limiter for all requests GlobalLimiter option is set to any PartitionedRateLimiter. In the above example, we have added a FixedWindowLimiter, and configured it to apply "per authenticated username (or hostname if not authenticated)" - the partition. The FixedWindowLimiter is then configured to automatically replenish permitted requests and permits 20 requests per minute.

Now run the above code and test it.



If we hit the API then the response comes and states code 200. But when we try to hit the same API again and again 20 times in 1 min. After 21 times we show the status code 503



If we want to show the meaningful response code then we have changed the status code. We Have Add

options.RejectionStatusCode = 429;

builder.Services.AddRateLimiter(options => {

options.RejectionStatusCode = 429;

JavaScript

Copy

After adding the StatusCode429, it shows the meaningful response,

*Error: Too Many Requests*

Types of RateLimit

There are several types of rate limiters that can be used to control access to a service or application:

1. Fixed window rate limiter: This type of rate limiter enforces a fixed number of requests over a fixed time window. For example, a fixed window rate limiter might allow 100 requests per minute.
2. Sliding window rate limiter: This type of rate limiter enforces a fixed number of requests over a sliding time window. For example, a sliding window rate limiter might allow 100 requests per minute, but the time window resets after each request.
3. Token bucket rate limiter: This type of rate limiter enforces a fixed number of requests over a fixed time window, but allows for bursts of requests above the limit. For example, a token bucket rate limiter might allow 100 requests per Minute but also allow up to 10 requests above the limit in a short burst.
4. Leaky bucket rate limiter: This type of rate limiter enforces a fixed number of requests over a fixed time window, but allows for a certain number of requests to be "leaked" out of the bucket over time.
5. IP rate limiter: This type of rate limiter enforces a rate limit based on the IP address of the client making the request.
6. User rate limiter: This type of rate limiter enforces a rate limit based on the user or client making the request.
7. combination of the above: you can also use a combination of these rate limiters to control access to your service or application.

Rate Limit groups of endpoints

In the above, we have checked the global limit request, But if we want to set a different Rate Limit for different endpoints.

Below is an example configuration adding 2 fixed window limiters with different settings, and a different policy name ("API" and "Web").

builder.Services.AddRateLimiter(options => {

options.RejectionStatusCode = 429;

options.AddFixedWindowLimiter("Api", options => {

options.AutoReplenishment = true;

options.PermitLimit = 10;

options.Window = TimeSpan.FromMinutes(1);

});

options.AddFixedWindowLimiter("Web", options => {

options.AutoReplenishment = true;

options.PermitLimit = 5;

options.Window = TimeSpan.FromMinutes(1);

});

});

JavaScript

Copy

In the API code, we have EnableRateLimiting

[EnableRateLimiting("Api")]

[HttpGet("GetWeatherForecast")]

public IEnumerable < WeatherForecast > Get() {

return Enumerable.Range(1, 5).Select(index => new WeatherForecast {

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

}).ToArray();

}

[EnableRateLimiting("Web")]

[HttpGet("GetWeatherForecast1")]

public IEnumerable < WeatherForecast > Get1() {

return Enumerable.Range(1, 5).Select(index => new WeatherForecast {

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

}).ToArray();

}

JavaScript

Copy

So if you want to disable RateLimit to some Action, Then you should just on the top of the action [DisableRateLimiting]

Conclusion

In conclusion, rate limiting is a technique used to control the rate of requests made to a server or service. It is used to prevent abuse, protect resources and ensure that service remains available and responsive for legitimate users. There are several different algorithms and techniques for implementing rate limiting, which can vary depending on the use case and the goals of the service. Overall, rate limiting is an important method for maintaining the performance and stability of a service in the face of high traffic or malicious activity.

# How To Use Rate Limiting In ASP.NET Core

APRIL 8, 2023

**READ TIME - 5 MINUTES**

**Thank you to our sponsors who keep this newsletter free to the reader:**

Today's issue is sponsored by [**Rebus Pro.**](https://pro.rebus.fm/) Rebus is a free .NET "service bus", and [**Rebus Pro is the perfect one-up for serious Rebus users.**](https://pro.rebus.fm/) Use Fleet Manager to get Slack alerts when something fails and retry dead-lettered messages with a click of the mouse.

And by [**Hasura**](https://hasura.io/graphql/database/sql-server/?utm_source=newsletter&utm_medium=email&utm_campaign=milan-dotnet-weekly), an open-source engine that gives [**instant GraphQL and REST APIs on new or existing SQL Server**](https://hasura.io/graphql/database/sql-server/?utm_source=newsletter&utm_medium=email&utm_campaign=milan-dotnet-weekly), enabling teams to ship apps faster.

Rate limiting is a technique to limit the number of requests to a server or an API.

A limit is introduced within a given time period to prevent server overload and protect against abuse.

In ASP.NET Core 7, we have a built-in rate limiter middleware that's easy to integrate into your API.

We're going to cover four rate limiter algorithms:

* [Fixed window](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#fixed-window-limiter)
* [Sliding window](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#sliding-window-limiter)
* [Token bucket](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#token-bucket-limiter)
* [Concurrency](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#concurrency-limiter)

Let's see how we can work with rate limiting.

## [What Is Rate Limiting?](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#what-is-rate-limiting)

Rate limiting is about restricting the number of requests to an API, usually within a specific time window or based on other criteria.

This is practical for a few reasons:

* Prevents overloading of servers or applications
* Improves security and guards against DDoS attacks
* Reduces costs by preventing unnecessary resource usage

In a multi-tenant application, each unique user can have a limitation on the number of API requests.

## [Configuring Rate Limiting](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#configuring-rate-limiting)

ASP.NET Core 7 introduced built-in rate limiting middleware in the **Microsoft.AspNetCore.RateLimiting** namespace.

To add rate limiting to your application, you first need to register the rate limiting services:

builder.Services.AddRateLimiter(options =>

{

options.RejectionStatusCode = StatusCodes.Status429TooManyRequests;

// We'll talk about adding specific rate limiting policies later.

});

I suggest updating the **RejectionStatusCode** to **429 (Too Many Requests)** because it's more correct. The default value is **503 (Service Unavailable)**.

And you also have to apply the **RateLimitingMiddleware**:

app.UseRateLimiter();

That's everything you'll need.

Let's see the rate limiting algorithms we can use.

## [Fixed Window Limiter](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#fixed-window-limiter)

The **AddFixedWindowLimiter** method configures a fixed window limiter.

The **Window** value determines the time window.

When a time window expires, a new one starts, and the request limit is reset.

builder.Services.AddRateLimiter(rateLimiterOptions =>

{

rateLimiterOptions.AddFixedWindowLimiter("fixed", options =>

{

options.PermitLimit = 10;

options.Window = TimeSpan.FromSeconds(10);

options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;

options.QueueLimit = 5;

});

});

## [Sliding Window Limiter](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#sliding-window-limiter)

The sliding window algorithm is similar to the fixed window, but it introduces segments in a window.

Here's how it works:

* Each time window is divided into multiple segments
* The window slides one segment each segment interval
* The segment interval is (window\_time)/(segments\_per\_window)
* When a segment expires, the requests taken in that segment are added to the current segment

The **AddSlidingWindowLimiter** method configures a sliding window limiter.

builder.Services.AddRateLimiter(rateLimiterOptions =>

{

rateLimiterOptions.AddSlidingWindowLimiter("sliding", options =>

{

options.PermitLimit = 10;

options.Window = TimeSpan.FromSeconds(10);

options.SegmentsPerWindow = 2;

options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;

options.QueueLimit = 5;

});

});

## [Token Bucket Limiter](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#token-bucket-limiter)

The token bucket algorithm is similar to the sliding window, but instead of adding back the requests from the expired segment, a fixed number of tokens are added after each replenishment period.

The total number of tokens can never exceed the token limit.

The **AddTokenBucketLimiter** method configures a token bucket limiter.

builder.Services.AddRateLimiter(rateLimiterOptions =>

{

rateLimiterOptions.AddTokenBucketLimiter("token", options =>

{

options.TokenLimit = 100;

options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;

options.QueueLimit = 5;

options.ReplenishmentPeriod = TimeSpan.FromSeconds(10);

options.TokensPerPeriod = 20;

options.AutoReplenishment = true;

});

});

When **AutoReplenishment** is **true**, an internal timer will execute every **ReplenishmentPeriod** and replenish the tokens.

## [Concurrency Limiter](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#concurrency-limiter)

The concurrency limiter is the most straightforward algorithm, and it just limits the number of concurrent requests.

The **AddConcurrencyLimiter** method configures a concurrency limiter.

builder.Services.AddRateLimiter(rateLimiterOptions =>

{

rateLimiterOptions.AddConcurrencyLimiter("concurrency", options =>

{

options.PermitLimit = 10;

options.QueueProcessingOrder = QueueProcessingOrder.OldestFirst;

options.QueueLimit = 5;

});

});

There's no time component involved in this case. The only parameter is the number of concurrent requests.

## [Using Rate Limiting In Your API](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#using-rate-limiting-in-your-api)

Now that we have configured our rate limiting policies, let's see how we can use them in our API.

There are slight differences between controllers and minimal API endpoints, so I'll cover them in separate examples.

**Controllers**

To add rate limiting on a controller we use the **EnableRateLimiting** and **DisableRateLimiting** attributes.

**EnableRateLimiting** can be applied on the controller or on the individual endpoints.

[EnableRateLimiting("fixed")]

public class TransactionsController

{

private readonly ISender \_sender;

public TransactionsController(ISender sender)

{

\_sender = sender;

}

[EnableRateLimiting("sliding")]

public async Task<IActionResult> GetTransactions()

{

return Ok(await \_sender.Send(new GetTransactionsQuery()));

}

[DisableRateLimiting]

public async Task<IActionResult> GetTransactionById(int id)

{

return Ok(await \_sender.Send(new GetTransactionByIdQuery(id)));

}

}

In the previous example:

* All endpoints in the **TransactionsController** will use a **fixed window** policy
* The **GetTransactions** endpoint will use a **sliding window** policy
* The **GetTransactionById** endpoint won't have any rate limiting applied

**Minimal APIs**

In a Minimal API endpoint you can configure the rate limit policy by calling **RequireRateLimiting** and specifying the policy name.

We're using the **token bucket** policy in this example.

app.MapGet("/transactions", async (ISender sender) =>

{

return Results.Ok(await sender.Send(new GetTransactionsQuery()));

})

.RequireRateLimiting("token");

## [Closing Thoughts](https://www.milanjovanovic.tech/blog/how-to-use-rate-limiting-in-aspnet-core#closing-thoughts)

It's great that we can quickly introduce **rate limiting** in ASP.NET Core.

You can choose from one of the existing rate limiter algorithms:

* Fixed window
* Sliding window
* Token bucket
* Concurrency

Here are some resources if you want to learn more about rate limiting:

* [Rate Limiting pattern](https://learn.microsoft.com/en-us/azure/architecture/patterns/rate-limiting-pattern)
* [Announcing Rate Limiting for .NET](https://devblogs.microsoft.com/dotnet/announcing-rate-limiting-for-dotnet/)