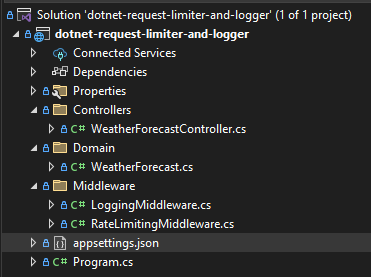
**.NET Request Logger & Rate Limiter**

**Overview**

This project is a **.NET Core Web API** extension that provides **request logging** and **rate limiting** capabilities. It logs incoming requests, including method, IP address, response time, and status, while blocking excessive requests from the same IP to prevent server overload.



**Features**

* **Request Logging**: Logs HTTP requests with details (method, IP, duration, status code).
* **Rate Limiting**: Blocks requests exceeding a configured threshold within a time window and requests that do not contain an IP address.
* **Configuration via appsettings.json**: Allows customizing limits and logging levels.
* **Swagger Integration**: API testing via Swagger UI.

**Technologies Used**

* .NET 8
* ASP.NET Core Web API
* C#
* Middleware Architecture
* Logging (Microsoft.Extensions.Logging)

**Setup Instructions**

1. **Clone the Repository**

* git clone <https://github.com/nichifor-dragos14/framework-design-assignment.git>
* cd dotnet-request-limiter-and-logger

1. **Install Dependencies**

* Ensure you have the .NET SDK installed. If not, download it from [Microsoft .NET](https://dotnet.microsoft.com/en-us/).

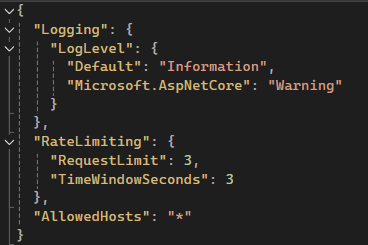
1. **Run the Application**

* dotnet run

The API will start at <https://localhost:7209> (or the default assigned port).

**Configuration**

Modify the RateLimiting section in appsettings.json to adjust rate limiting settings:

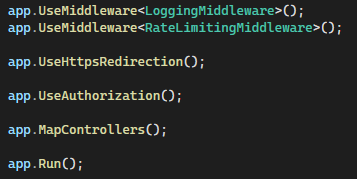


* RequestLimit: Maximum number of requests per IP in the given time window.
* TimeWindowSeconds: Time window for rate limiting (in seconds).

**Middleware Implementation**

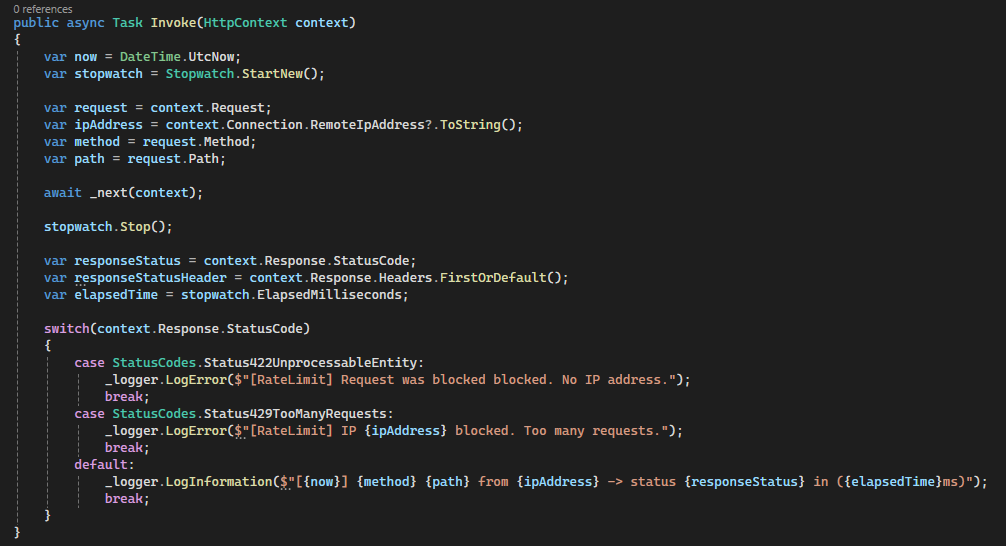
1. **Middleware usage**

* In the Program.cs file, the middleware components are registered to handle HTTP requests and responses.
* The order in which middleware components are registered determines how they interact. In this case, LoggingMiddleware is executed first. A stopwatch will be used for tracking the execution time of every incoming request. Once the request details are recorded and the stopwatch is initialized, the request proceeds to the RateLimitingMiddleware, where the system checks whether the client has exceeded the allowed number of requests. If the request is within the limit, it continues further into the application. Otherwise, it is blocked, and an appropriate response is returned.



1. **Logging Middleware**

* Captures and logs incoming request details, such as the HTTP method, path, IP address, response status, and duration.
* If a request is blocked due to exceeding the rate limit, it logs an error indicating the blocked IP. An error is logged when the incoming IP address cannot be identified as well.
* Other messages can be configured by adding different cases inside the switch statement.



1. **Rate Limiting Middleware**

* Monitors and restricts the number of requests per IP address based on configuration settings.
* Maintains a record of recent requests per IP and blocks further requests if the limit is exceeded within the specified time frame. This is done with a concurrent dictionary in order to prevent asynchronous operations.
* Returns an HTTP 429 (Too Many Requests) status when a request is blocked, along with a message indicating the rate limit breach. Returns an HTTP 422 (Unprocessable Entity) status when a request is blocked due to not identifying the incoming IP address.



**Usage & Testing**

1. **Test with Swagger UI**

* Run the API and navigate to <https://localhost:7209/swagger> to test endpoints.

1. **Send API Requests**

* Test the rate limiting by sending multiple requests:

curl -X GET "https://localhost:7209/api/test" -H "Accept: application/json"’

If you exceed the limit, you'll receive: [Rate Limit] IP ::1 blocked. Too many requests

If the IP address could not be identified, you'll receive: [RateLimit] Request was blocked blocked. No IP address.

**Conclusion**

This project provides a robust logging and request-limiting middleware that helps secure APIs from abuse and ensures performance stability. 🚀

For further improvements, consider:

* Implementing dynamic configuration changes without restarting the app.
* Logging request details into a database for analytics.
* Adding JWT-based authentication to prevent abuse from unauthorized users.