Lesson 43 - Middlewares

`Middleware` is software that's assembled into an app pipeline to handle requests and responses. Each component:

- Chooses whether to pass the request to the next component in the pipeline.
- Can perform work before and after the next component in the pipeline.

Request delegates are used to build the request pipeline. The request delegates handle each HTTP request. The ASP.NET Core request pipeline consists of a sequence of request delegates, called one after the other.

What is a Delegate?

Delegate means *"to entrust (a task or responsibility) to another person"*. In .NET, a delegate is a type that represents references to methods with a particular parameter list and return type. When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type. You can invoke (or call) the method through the delegate instance. Delegates are used to pass methods as arguments to other methods.

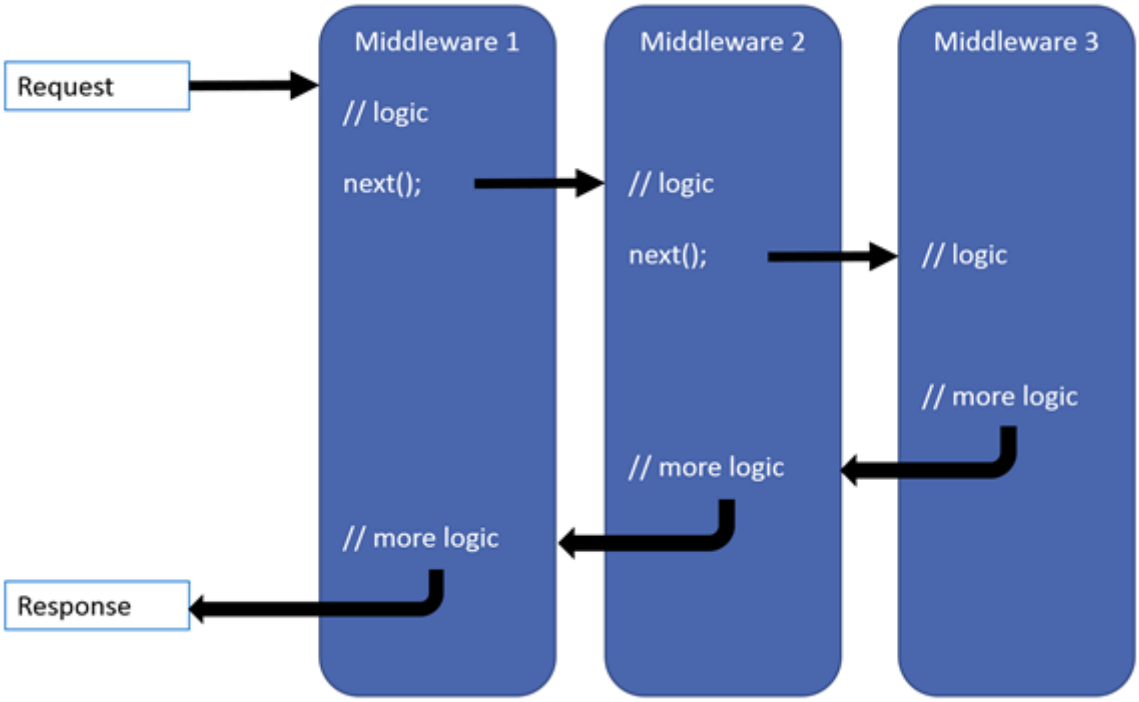
Delegates have the following properties:

- Delegates are similar to C++ function pointers, but delegates are fully object-oriented, and unlike C++ pointers to member functions, delegates encapsulate both an object instance and a method.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- Methods don't have to match the delegate type exactly.
- Lambda expressions are a more concise way of writing inline code blocks. Lambda expressions (in certain contexts) are compiled to delegate types.

What is a Request Delegate?

A function that can process an HTTP request. It takes the `HttpContext` object and processes the request. The following diagram demonstrates the concept. The thread

of execution follows the black arrows:



Each delegate can perform operations before and after the next delegate. Exception-handling delegates should be called early in the pipeline, so they can catch exceptions that occur in later stages of the pipeline.

Registering a Middleware

To register a middleware for your web API:

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

app.UseMiddleware<YourMiddleware>();

// app.UseStaticFiles();

// app.UseRouting();

// app.UseResponseCompression();

// app.MapRazorPages();

app.Run();
```

To add multiple middlewares, just add another `UseMiddleware()` statement before `Run()`. Keep in mind that the order of the statements matter. If you want to use an Exception Handler Middleware, it should be the middleware in the pipeline so that it can handle exceptions within other middlewares as well.

Custom Middlewares

Every middleware will have a `RequestDelegate` which stores the next function in the pipeline.

```
public class ExceptionHandlerMiddleware
{
        private readonly RequestDelegate _next;

        public ExceptionHandlerMiddleware(RequestDelegate next) => _next = next;
}
```

The middleware class must include:

- A public constructor with a parameter of type RequestDelegate.
- A public method named `Invoke` or `InvokeAsync`. This method must:
  - Return a `Task`.
  - Accept a first parameter of type `HttpContext`.

```
public class ExceptionHandlerMiddleware
{
        private readonly RequestDelegate _next;

        public ExceptionHandlerMiddleware(RequestDelegate next) => _next = next;

        public async Task InvokeAsync(HttpContext context) => await
_next(httpContext);
}
```