**Middleware In ASP.NET Core**

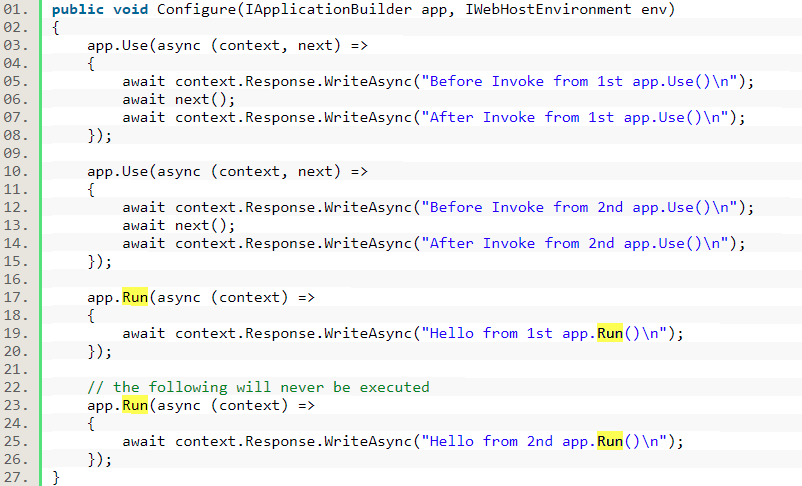
1. **Mis on Middleware?**

Middleware is software that enables one or more kinds of communication or connectivity between two or more applications or application components in a distributed network.

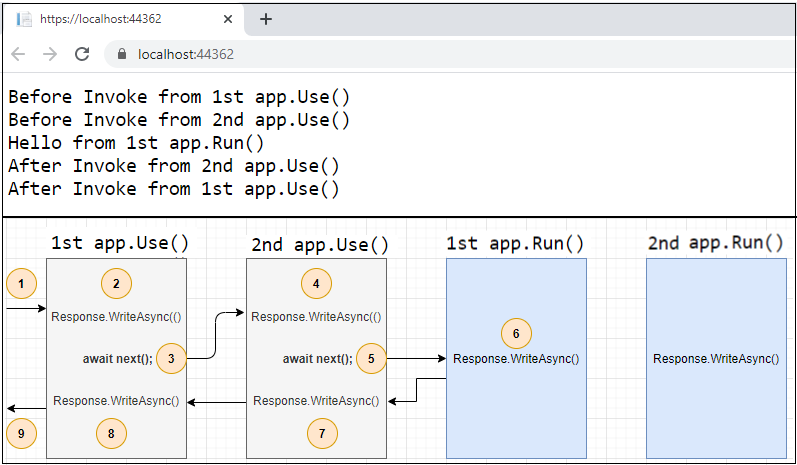
1. **Missuguseid extensione meetodeid kasutab?**
   * Run;
   * Use;
   * Map;
2. **Iseloomusta Middleware käskude loogikat.**

* **app.Run();**

This middleware component may expose Run[Middleware] methods that are executed at the end of the pipeline. Generally, this acts as a terminal middleware and is added at the end of the request pipeline, as it cannot call the next middleware.

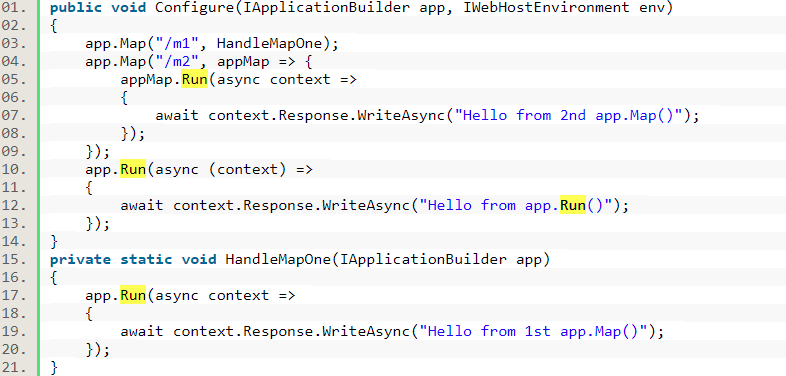


* **app.Use();**

This is used to configure multiple middleware. Unlike app.Run(), We can include the next parameter into it, which calls the next request delegate in the pipeline. We can also short-circuit (terminate) the pipeline by not calling the next parameter.

*The first app.Run() delegate terminates the pipeline. In the following example, only the first delegate (“Hello from 1st app.Run()”) will run and the request will never reach the second Run method.*

* **app.Map();**

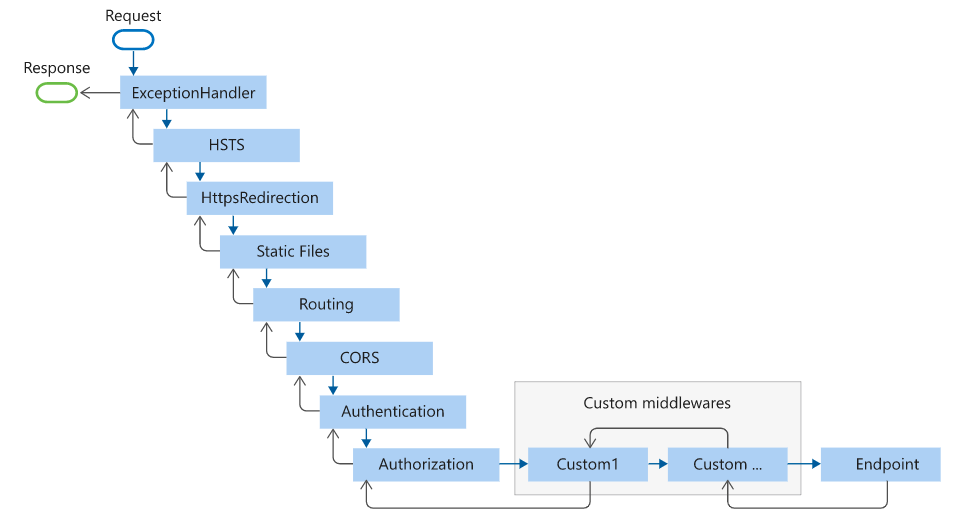
These extensions are used as a convention for branching the pipeline. The map branches the request pipeline based on matches of the given request path. If the request path starts with the given path, the branch is executed.

1. **Kuidas kutsutakse delegaati, kui see ei edasta päringut (request) järgmisele delegaadile?**

Request delegates are used to build the request pipeline. The request delegates handle each HTTP request. Request delegates are configured using Run, Map, and Use extension methods. An individual request delegate can be specified in-line as an anonymous method (called in-line middleware), or it can be defined in a reusable class. These reusable classes and in-line anonymous methods are middleware, also called middleware components. Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline or short-circuiting the pipeline. When a middleware short-circuits, it's called a terminal middleware because it prevents further middleware from processing the request.

1. **Kirjelda täieliku päringuprotsessi järjestust ASP.NET Core MVC näite puhul.**

The following diagram shows the complete request processing pipeline for ASP.NET Core MVC and Razor Pages apps. You can see how, in a typical app, existing middlewares are ordered and where custom middlewares are added. You have full control over how to reorder existing middlewares or inject new custom middlewares as necessary for your scenarios.



The Endpoint middleware in the preceding diagram executes the filter pipeline for the corresponding app type—MVC or Razor Pages.

1. **Kuidas mõjutab middleware koodi järjestus komponeneti tööd?**

The order that middleware components are added in the Startup. Configure method defines the order in which the middleware components are invoked on requests and the reverse order for the response. The order is critical for security, performance, and functionality.

1. **Mida teeb Developer Exception Page Middleware, UseExceptionHandler, UseHsts, UseHttpsRedirection, UseStaticFile, UseCookiePolicy, UseRouting, UseAuthentication, UseAuthorization, UseSession ja UseEndpoints koos MapRazorPages, UseSpaStaticFiles?**

The following Startup.Configure method adds middleware components for common app scenarios:

* Exception/error handling

When the app runs in the Development environment:

* Developer Exception Page Middleware (UseDeveloperExceptionPage) reports app runtime errors.
* Database Error Page Middleware reports database runtime errors.

When the app runs in the Production environment:

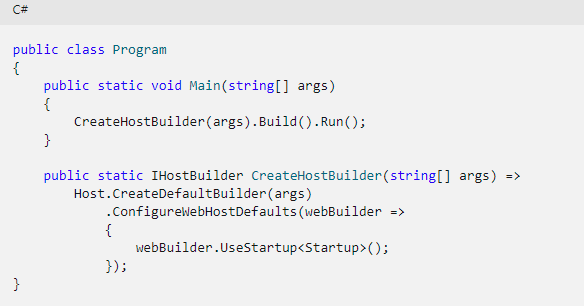
* Exception Handler Middleware (UseExceptionHandler) catches exceptions thrown in the following middlewares.
* HTTP Strict Transport Security Protocol (HSTS) Middleware (UseHsts) adds the Strict-Transport-Security header.
* HTTPS Redirection Middleware (UseHttpsRedirection) redirects HTTP requests to HTTPS.
* Static File Middleware (UseStaticFiles) returns static files and short-circuits further request processing.
* Cookie Policy Middleware (UseCookiePolicy) conforms the app to the EU General Data Protection Regulation (GDPR) regulations.
* Routing Middleware (UseRouting) to route requests.
* Authentication Middleware (UseAuthentication) attempts to authenticate the user before they're allowed access to secure resources.
* Authorization Middleware (UseAuthorization) authorizes a user to access secure resources.
* Session Middleware (UseSession) establishes and maintains session state. If the app uses session state, call Session Middleware after Cookie Policy Middleware and before MVC Middleware.
* Endpoint Routing Middleware (UseEndpoints with MapRazorPages) to add Razor Pages endpoints to the request pipeline.

For Single Page Applications (SPAs), the SPA middleware UseSpaStaticFiles usually comes last in the middleware pipeline. The SPA middleware comes last:

* To allow all other middlewares to respond to matching requests first.
* To allow SPAs with client-side routing to run for all routes that are unrecognized by the server app.

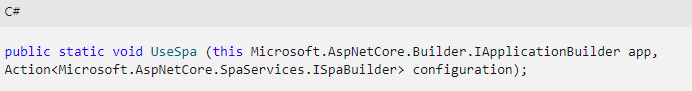
For more details on SPAs, see the guides for the React and Angular project templates.

1. **Valige välja viis Built-in Middleware ja iseloomustage neid.**
   * Static Files - Provides support for serving static files and directory browsing.

Static files are stored within the project's [web root](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-5.0#web-root) directory. The default directory is {content root}/wwwroot, but it can be changed with the [UseWebRoot](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.hostingabstractionswebhostbuilderextensions.usewebroot) method.

* + WebSockets - Enables the WebSockets protocol.

A protocol that enables two-way persistent communication channels over TCP connections. It's used in apps that benefit from fast, real-time communication, such as chat, dashboard, and game apps.

* + URL rewrite - Provides support for rewriting URLs and redirecting requests.
* URL rewriting is the act of modifying request URLs based on one or more predefined rules. URL rewriting creates an abstraction between resource locations and their addresses so that the locations and addresses aren't tightly linked. URL rewriting is valuable in several scenarios to:
* Move or replace server resources temporarily or permanently and maintain stable locators for those resources.
* Split request processing across different apps or across areas of one app. Remove, add, or reorganize URL segments on incoming requests.
* Optimize public URLs for Search Engine Optimization (SEO).
* Permit the use of friendly public URLs to help visitors predict the content returned by requesting a resource.
* Redirect insecure requests to secure endpoints.
* Prevent hotlinking, where an external site uses a hosted static asset on another site by linking the asset into its own content.
* SPA - Handles all requests from this point in the middleware chain by returning the default page for the Single Page Application (SPA)
* This middleware should be placed late in the chain, so that other middleware for serving static files, MVC actions, etc., takes precedence.
* MVC - Processes requests with MVC/Razor Pages.
* The Model-View-Controller (MVC) architectural pattern separates an application into three main groups of components: Models, Views, and Controllers. This pattern helps to achieve separation of concerns. Using this pattern, user requests are routed to a Controller which is responsible for working with the Model to perform user actions and/or retrieve results of queries. The Controller chooses the View to display to the user, and provides it with any Model data it requires. The following diagram shows the three main components and which ones reference the others:

