**Middleware In ASP.NET Core**

***1. Mis on Middleware?***

Middleware helps developers build applications more efficiently. It acts like the connective tissue between applications, data, and users. For organizations with multi-cloud and containerized environments, middleware can make it cost-effective to develop and run applications at scale.

***2. Missuguseid extensione meetodeid kasutab?***

Run, Use, and Map.

***3. Iseloomusta Middleware käskude loogikat.***

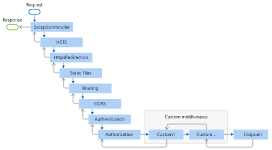
* The Run method is an extension method on the IApplicationBuilder interface and accepts a parameter of RequestDelegate delegate type which actually handles the request.
* The Use Extension Method in ASP.NET Core Application allows us to add a new middleware component which may call the next middleware component in the request processing pipeline. The Use extension method adds a middleware delegate defined in-line to the application’s request pipeline. The Use method is also implemented as an extension method on the IApplicationBuilder interface.
* Map extensions are used as a convention for branching the pipeline. 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.

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

When a delegate doesn't pass a request to the next delegate, it's called short-circuiting the request pipeline. Short-circuiting is often desirable because it avoids unnecessary work. For example, Static File Middleware can act as a terminal middleware by processing a request for a static file and short-circuiting the rest of the pipeline. Middleware added to the pipeline before the middleware that terminates further processing still processes code after their next.Invoke statements.

***5. 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.



***6. 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.

***7. 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.

***8. Valige välja viis Built-in Middleware ja iseloomustage neid.***

* Health checks in ASP.NET Core

Health checks are created by implementing the IHealthCheck interface. The CheckHealthAsync method returns a HealthCheckResult that indicates the health as Healthy, Degraded, or Unhealthy. The result is written as a plaintext response with a configurable status code (configuration is described in the Health check options section). HealthCheckResult can also return optional key-value pairs.

* Introduction to Identity on ASP.NET Core

ASP.NET Core Identity:

Is an API that supports user interface (UI) login functionality.

Manages users, passwords, profile data, roles, claims, tokens, email confirmation, and more.

Users can create an account with the login information stored in Identity or they can use an external login provider. Supported external login providers include Facebook, Google, Microsoft Account, and Twitter.

* AuthorizationAppBuilderExtensions

Adds the AuthorizationMiddleware to the specified IApplicationBuilder, which enables authorization capabilities. When authorizing a resource that is routed using endpoint routing, this call must appear between the calls to app.UseRouting() and app.UseEndpoints(...) for the middleware to function correctly.

* Configure ASP.NET Core to work with proxy servers and load balancers

In the recommended configuration for ASP.NET Core, the app is hosted using IIS/ASP.NET Core Module, Nginx, or Apache. Proxy servers, load balancers, and other network appliances often obscure information about the request before it reaches the app:

When HTTPS requests are proxied over HTTP, the original scheme (HTTPS) is lost and must be forwarded in a header.

Because an app receives a request from the proxy and not its true source on the Internet or corporate network, the originating client IP address must also be forwarded in a header.

This information may be important in request processing, for example in redirects, authentication, link generation, policy evaluation, and client geolocation.

* Response Caching Middleware in ASP.NET Core

The middleware determines when responses are cacheable, stores responses, and serves responses from cache.

Responses containing content for authenticated clients must be marked as not cacheable to prevent the middleware from storing and serving those responses.