Request pipeline & middleware

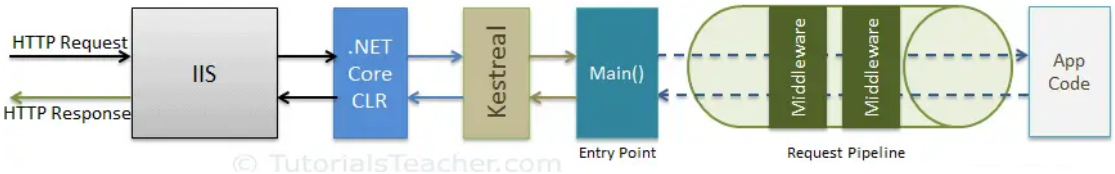
בעליית האפליקציה נבנה כמין צינור המגדיר איך לטפל ב request שהגיע לשרת.

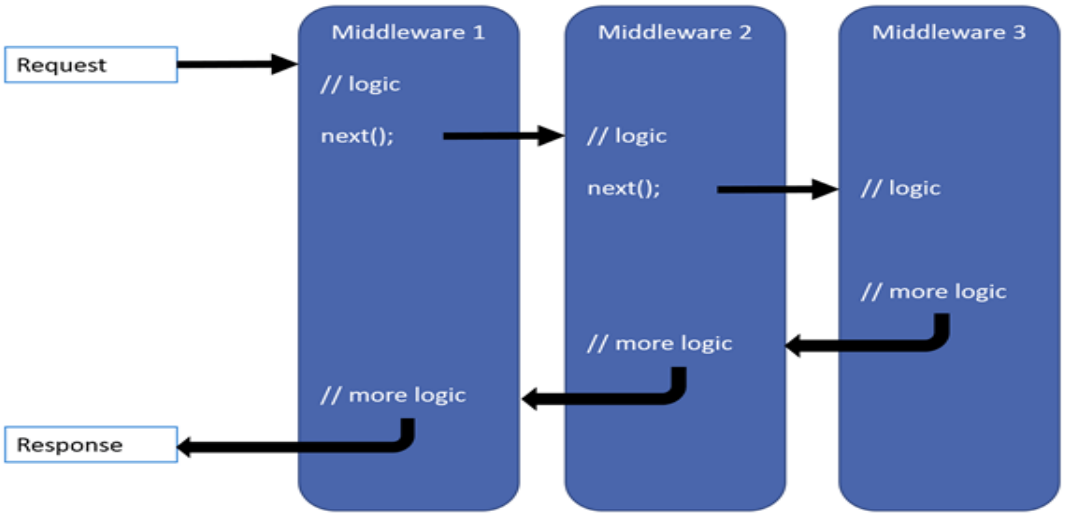
זהו כמין פס יצור שדרכו עובר ה response ומורכב ממספר לא קבוע של חוליות שהן פונקציות הנקראות לביצוע בזו אחר זו.

בכל חוליה ניתן להגדיר האם לטפל בבקשה להעביר לחוליה הבאה או להחזיר response.

הגדרת מבנה ה request pipeline נעשית בקובץ program.cs (מ net >6.)

תהליך מעבר ה request לשרת:





Middleware

Middleware זהו delegate דרכו עוברת הבקשה. כל middleware יכול :

1. לטפל בבקשה ולהחזיר ל"צינור"
2. לטפל ולהעביר ל middleware הבא
3. להעביר ל middleware הבא בתור בלבד.

כל middleware יכול לבצע "עבודה" לפני או אחרי ה middleware הבא בתור שהרי התהליך הוא פנימה והחוצה.

ה"צינור" קובע את סדר ה middlewares כך שכל אחד יודע רק מי הבא אחריו. לכן הסדר של הגדרתם ב program.cs חשוב.
למשל UseExceptionHandler המטפל בשגיאות יהיה מוכרח להיות הראשון כדי לבדוק את התוצאה ולהעביר לקליינט את השגיאה.

ה middleware האחרון נקרא "terminal middleware" זוהי התחנה הסופית שממנה התשובה מתחילה לחזור...

ישנם middlewares רבים שמגיעים כחלק מה framework או מ NuGet וניתן גם לכתוב custom middleware המותאמים אישית.

דוגמה לmiddleware מNuGet: UseSwagger .

שימי לב לעיתים מלבד שורת ה use יש לצרוך גם service במקביל.

דוגמאות לbuild in middlewares:

Middleware	Description	Order
Authentication	מספק תמיכה בזיהוי המשתמש	Before HttpContext.User is needed. Terminal for OAuth callbacks.
Authorization	מספק תמיכה באזור השליטה של המשתמש	Immediately after the Authentication Middleware.
Cookie Policy	מטפל הנושא הcookies	Before middleware that issues cookies. Examples: Authentication, Session, MVC (TempData).
CORS	האפליקציה משתפת. Originמגדיר לאיזה סוגי	Before components that use CORS. UseCors currently must go before UseResponseCaching .
DeveloperExceptionPage	יוצר דף עם מידע מפורט על שגיאות עבור המפתח.	Before components that generate errors. The project templates automatically register this middleware as the first middleware in the pipeline when the environment is Development.
HTTPS Redirection	מעביר את כל בקשות HTTPל HTTPS.	Before components that consume the URL.
Request Localization	מספק תמיכה בניהול שפות	Before localization sensitive components. Must appear after Routing Middleware when using RouteDataRequestCultureProvider.
Request Timeouts	מספק תמיכה לtimeout	UseRouting must come before UseRequestTimeouts.
Endpoint Routing	Defines and constrains request routes.	Terminal for matching routes.

Session	Provides support for managing user sessions.	Before components that require Session.
Static Files	Provides support for serving static files and directory browsing.	Terminal if a request matches a file.
URL Rewrite	Provides support for rewriting URLs and redirecting requests.	Before components that consume the URL.
W3CLogging	Generates server access logs in the W3C Extended Log File Format.	At the beginning of the middleware pipeline.

איך כותבים custom middleware?

דרך אחת היא anonymous function . ניתן גם לכתוב מחלקה.

הגדרת ה middleware ב pipeline יכולה להעשות באמצעות אחת משלוש פונקציות:

Run- מקבלת פרמטר context מסוג HTTPContext ואין לה אופציה להעביר הלאה – ללא next. זה בעצם להגדיר את ה..terminal middleware.

Use - מקבלת פרמטר context מסוג HTTPContext וגם פרמטר next שהוא delegate ל middleware הבא.

Map - כדי להבדיל בין ברנצים וניתובים שונים מקבלת pattern ו delegate לפונקציה שתופעל בעת ניתוב ל pattern הספציפי.

דוגמאות לanonymous function

```
app.Map("/map2",() => "HandleMapTest2");
app.Use(async (context, next) =>
{
await context.Response.WriteAsync("Before Invoke from 1st app.Use()\n");
await next();
await context.Response.WriteAsync("After Invoke from 1st app.Use()\n");
});

app.Use(async (context, next) =>
{
await context.Response.WriteAsync("Before Invoke from 2nd app.Use()\n");
await next();
```

```
await context.Response.WriteAsync("After Invoke from 2nd app.Use()\n");
});

app.Run(async (context) =>
{
await context.Response.WriteAsync("Hello from 1st app.Run()\n");
});

// the following will never be executed
app.Run(async (context) =>
{
await context.Response.WriteAsync("Hello from 2nd app.Run()\n");
});
```

ניתן להעביר את ה custom middleware לclass:

```
 using System.Globalization;

namespace Middleware.Example;

public class RequestCultureMiddleware
{
private readonly RequestDelegate _next;

public RequestCultureMiddleware(RequestDelegate next)
{
_next = next;
}

public async Task InvokeAsync(HttpContext context)
{
```

```
var cultureQuery = context.Request.Query["culture"];
if (!string.IsNullOrWhiteSpace(cultureQuery))
{
var culture = new CultureInfo(cultureQuery);

CultureInfo.CurrentCulture = culture;
CultureInfo.CurrentUICulture = culture;
}

// Call the next delegate/middleware in the pipeline.
await _next(context);
}
}
```

המחלקה חייבת לכלול public constructor עם פרמטר מסוג RequestDelegate.

וכן public methodבשם Invokeאו InvokeAsync שמחזירה task ומקבלת כפרמטר ראשון מסוג HTTPContext

פרמטרים נוספים ל ctor או ה invoke יגיעו באמצעות DI.

עמ"נ לייצא את ה middleware נוסיף פונקציית הרחבה extension method

```
public static class RequestCultureMiddlewareExtensions
{
public static IApplicationBuilder UseRequestCulture(
this IApplicationBuilder builder)
{
return builder.UseMiddleware<RequestCultureMiddleware>();
}
}
```

השלב הבא יהיה להוסיף את ה middlewareהחדש ל request pipeline ב: program.cs

```
app.UseRequestCulture();
```

ניתן גם במקום פונקציית ההרחבה להוסיף בצורה כזו:

```
app.UseMiddleware<MyMiddleware>()
```