Для обработки запросов теперь используется новый конвейер HTTP, который основан на компонентах Katana и спецификации OWIN. А его модульность позволяет легко добавить свои собственные компоненты.

var host = new WebHostBuilder()

        .UseKestrel()               // настраиваем веб-сервер Kestrel

        .UseContentRoot(Directory.GetCurrentDirectory())    // настраиваем корневой каталог приложения

        .UseIISIntegration()        // обеспечиваем интеграцию с IIS

        .UseStartup<Startup>()    // устанавливаем главный файл приложения

        .Build();                   // создаем хост

    host.Run();                     // запускаем приложение

**StartUp]**

public void ConfigureServices(IServiceCollection services)//Optional either //like constructor

        {

        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)

        {

            if (env.IsDevelopment())

            {

                app.UseDeveloperExceptionPage();

            }

            app.Run(async (context) =>

            {

                await context.Response.WriteAsync("Hello World!");

            });

        }

    }

Метод Configure выполняется один раз при создании объекта класса Startup, и компоненты middleware создаются один раз и живут в течение всего жизненного цикла приложения:

public void Configure(IApplicationBuilder app)

    {

        int x = 2;

        app.Run(async (context) =>

        {

            x = x \* 2;  //  2 \* 2 = 4 //Next request = 8

            await context.Response.WriteAsync($"Result: {x}");

        });

    }

Перегрузка метода Use, которая в качестве параметров принимает контекст запроса - объект HttpContext и делегат Func<Task>, который представляет собой ссылку на следующий в конвейере компонент middleware. Делится на две части (до next.Invoke() и после next.Invoke().)

Run не вызывают никакие другие компоненты и дальше обработку запроса не передают.

Метод Map (и методы расширения MapXXX()) применяется для сопоставления пути запроса с определeнным делегатом, который будет обрабатывать запрос по этому пути. Например: app.Map("/index", Index);

private static void Index(IApplicationBuilder app)

{

    app.Run(async context =>

    {

        await context.Response.WriteAsync("Index");

    });

}

Вложенные методы Map:

app.Map("/home", home =>

    {

        home.Map("/index", Index);

        home.Map("/about", About);

    });

Метод MapWhen

app.MapWhen(context => {

        return context.Request.Query.ContainsKey("id") &&

                context.Request.Query["id"] == "5";

    }, HandleId);

Class component

public class TokenMiddleware

{

    private readonly RequestDelegate \_next;

    public TokenMiddleware(RequestDelegate next)

    {

        this.\_next = next;

    }

    public async Task InvokeAsync(HttpContext context)

    {        if (token!="12345678")

        {//some code        }

        else

        {

            await \_next.Invoke(context);

        }

    }

}

!!!!!!!!! In StartUp > Configure > app.UseMiddleware<TokenMiddleware>() or use extension method like UseToken(this IApplicationBuilder builder).If you want to pass parameter you can use constructor.

**static**

WebHost.CreateDefaultBuilder(args)

            .UseStartup<Startup>()          // установка класса Startup как стартового

            .UseWebRoot("static")   // установка папки static

*http://localhost:55234 /index.html – from static*

**UseDefaultFiles() ->**  [*http://localhost/*](http://localhost/) *->* default.htm, default.html,index.htm, index.html or DefaultFilesOptions options = new DefaultFilesOptions();

        options.DefaultFileNames.Clear(); // удаляем имена файлов по умолчанию

        options.DefaultFileNames.Add("hello.html"); // добавляем новое имя файла

        app.UseDefaultFiles(options); // установка параметров

**app.UseDirectoryBrowser() ->**  позволяет пользователям просматривать содержимое каталогов на сайте

app.UseDirectoryBrowser(new DirectoryBrowserOptions()

    {

        FileProvider = new PhysicalFileProvider(Path.Combine(Directory.GetCurrentDirectory(), @"wwwroot\html")),

        RequestPath = new PathString("/pages")

    });

OWIN (Open Web Interface for .NET), которая позволяет отвязать веб-приложение от конкретного веб-сервера и по сути создать самохостирующееся приложение. OWIN определеляет механизм использования компонентов middleware для обработки конкретных запросов и отправки ответа.

В соответствии со спецификацией OWIN компоненты middleware OWIN должны принимать объект IDictionary<string, object>, который представляет набор параметров, связанных с запросом и возвращаемым ответом.