<https://metanit.com/sharp/aspnet6/2.1.php>

ASP.Net Core cross-platform, high-performance, open-source

Multiple platforms: Windows, macOs, Linux

Integration client-side frameworks

Support for hosting Remote Procedures Call (gRPC)

Built-in dependency injection

Light-weight, high-performance, modular HTTP request pipeline

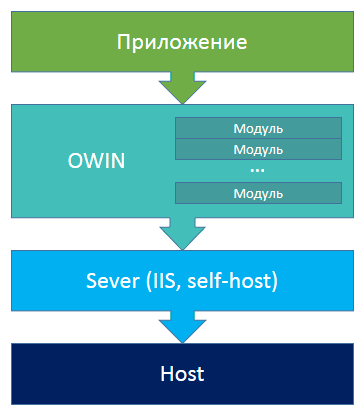
Ability to host on:

* Kestrel
* IIS
* HTTP.sys
* Ngingx
* Apache
* Docker

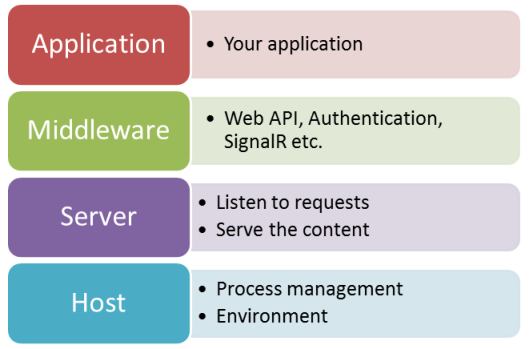
Open Web Api Interface for .Net (OWIN) – a standard interface between .Net web servers and web applications. Used for .Net apps

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/owin?view=aspnetcore-7.0>

OWIN – allows web apps to be decoupled from web servers. It defines a standard way for middleware to be used in a pipeline to handle requests and responses. Defines **Middleware** and **Servers**



Katana – implementation of OWIN interface for Microsoft .Net.



Host – is a process, which serves the application. It can be IIS server or the application file itself.

Server – implements OWIN specification, executes filling of environment dictionary with requests data.

Middleware – set of components which serve the requests.

Application – our logic itself

<https://github.com/dotnet-architecture/eShopOnContainers?WT.mc_id=dotnet-35129-website>

**WebApplication** – is a main class/object.

WebApplicationBuilder:

1. .CreateBuilder()
2. Configuration
3. Environment
4. Host / WebHost
5. Logging
6. Services

Methods:

1. .Run() / .RunAsync()
2. .Start() / .StartAsync()
3. .StopAsync()

**Middleware:**

.Use() – allows to define request delegates chain ( next.Invoke() method)

.Map() – branches the request pipeline based on matches of the given request path

.MapWhen() – branches the request based on the result of given predicate

.Run() – terminates the pipeline. If another Run delegate is added after Run delegate, it’s not called

app.Use(async (context, next) => { … })

next.Invoke() should be called before the response has been sent to client

app.Run(async context => {…})

don’t receive a next parameter. The first .Run() is always terminal and terminates the pipeline

**Program.cs**

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Map("/", () => "Home page");

app.Map("/about", () => "About page");

app.Map("/contact", () => "Contact page");

app.Use(async (context, next) =>

{

context.Response.Cookies.Append("Mentorship", "Svitlana");

context.Response.ContentType = "application/json";

context.Response.Headers.Add("Server", "PHP");

await next();

});

app.Run();

[https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-7.0#aspnet-core-middleware](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis?view=aspnetcore-7.0%23aspnet-core-middleware)

Middleware order:

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-7.0#middleware-order>

Common app scenarios for middleware:

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseDatabaseErrorPage();

}

else

{

app.UseExceptionHandler("/Error");

app.UseHsts(); //HTTP Strict Transport Security, ForwardedHeader

}

app.UseHttpsRedirection(); //Redirects All HTTP to HTTPs

app.UseStaticFiles(); //Support for serving Static Files and browse directory

app.UseCookiePolicy(); //Ensures secure, SameSite cookie fields

app.UseRouting(); //Constraint request to endpoints

app.UseAuthentication();

app.UseAuthorization();

app.UseSession(); //Support for Managing user sessions

app.MapRazorPages();

**Controllers / Actions:**

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

var app = builder.Build();

app.MapControllers();

app.Run();

GET / POST / PUT / DELETE

Filter is used to perform logic before or after action method

**Routing:**

.Map(), .MapGet(), .MapPost(), .MapPut()

app.MapGet("/", () => "This is a GET");

app.MapPost("/", () => "This is a POST");

app.MapPut("/", () => "This is a PUT");

app.MapDelete("/", () => "This is a DELETE");

Conventional routing:

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

.MapAreaRoute(…)

Attribute routing:

[Route("")]

[Route("Home")]

[Route("Home/Index")]

[Route("Home/Index/{id?}")]

public IActionResult Index(int? id)

{

return ControllerContext.MyDisplayRouteInfo(id);

}

Areas:

app.MapAreaControllerRoute(name: "duck\_route",

areaName: "Duck",

pattern: "Manage/{controller}/{action}/{id?}");

app.MapControllerRoute(name: "default",

pattern: "Manage/{controller=Home}/{action=Index}/{id?}");

[Area("Duck")]

public class UsersController : Controller

{

// GET /Manage/users/GenerateURLInArea

public IActionResult GenerateURLInArea()

{

// Uses the 'ambient' value of area.

var url = Url.Action("Index", "Home");

// Returns /Manage/Home/Index

return Content(url);

}

// GET /Manage/users/GenerateURLOutsideOfArea

public IActionResult GenerateURLOutsideOfArea()

{

// Uses the empty value for area.

var url = Url.Action("Index", "Home", new { area = "" });

// Returns /Manage

return Content(url);

}

}

app.Use(async (context, next) =>

{

if (context.Request.Path == "/date")

await context.Response.WriteAsync($"Date: {DateTime.Now.ToShortDateString()}");

else

await next.Invoke();

});

**Dependency Injection:**

Constructor injection:

Method injection:

public IActionResult About([FromServices] IDateTime dateTime)

{

return Content( $"Current server time: {dateTime.Now}");

}

IServiceProvider

Объект IServiceCollection.Services

**ServiceType**: тип сервиса

**ImplementationType**: тип реализации сервиса

**Lifetime**: жизненный цикл сервиса

* **Transient**
* **Scoped**
* **Singleton**

Inject into Middleware

**Configuration:**

Interface IConfiguration

[https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.c onfiguration.iconfiguration?view=dotnet-plat-ext-7.0](https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.c%09%09onfiguration.iconfiguration?view=dotnet-plat-ext-7.0)

Interface IConfigurationRoot

<https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.configuration.iconfigurationroot?view=dotnet-plat-ext-7.0>

Providers:

* Command line arguments
* Environment variables
* Add Ini File, Ini Stream
* Add InMemory Collection
* Files (Json, xml)
* User Secrets
* Azure (Key Vault)
* Custom providers

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// установка настроек конфигурации

app.Configuration["name"] = "Tom";

app.Configuration["age"] = "37";

app.Run(async (context) =>

{

// получение настроек конфигураци

string name = app.Configuration["name"];

string age = app.Configuration["age"];

await context.Response.WriteAsync($"{name} - {age}");

});

app.Run();

1. In-Memory configuration

builder.Configuration .AddInMemoryCollection(new Dictionary<string, string>

{

{"name", "Tom"},

{"age", "37"}

});

app.Map("/", (IConfiguration appConfig) => $"{appConfig["name"]} - {appConfig["age"]}");

1. Command Line

builder.Configuration.AddCommandLine(args);

launchSettings.json:

"commandLineArgs": "name=Bob age=37",

1. Json/XML files

builder.Configuration.AddJsonFile("config.json");

app.Map("/", (IConfiguration appConfig) => $"{appConfig["person"]} - {appConfig["company"]}");

builder.Configuration.AddXmlFile("config.xml");

Источники конфигурации загружаются в порядке, в котором подключены

Самый последний перетирает первый

appConfig.GetSection("ConnectionStrings:DefaultConnection").Value;

appConfig["ConnectionStrings:DefaultConnection"];

appConfig.GetConnectionString("DefaultConnection");

**IOptions**

Позволяет передавать конфигурацию как объекты определенных классов

var builder = WebApplication.CreateBuilder();

builder.Configuration.AddJsonFile("person.json");

// устанавливаем объект Person по настройкам из конфигурации

builder.Services.Configure<Person>(builder.Configuration);

var app = builder.Build();

app.Map("/", (IOptions<Person> options) =>

{

Person person = options.Value; // получаем переданные через Options объект Person

return person;

});

app.Run();

**Logging**

ILogger<T> interface

var app = builder.Build();

app.Logger.LogDebug()

Log Level:

Trace (Verbose) – наиболее детализированный уровень

Debug – информация, которая будет полезна для отладки

Information -

Warning

Error

Critical

app.Run(async (context) =>

{

var path = context.Request.Path;

app.Logger.LogCritical($"LogCritical {path}");

app.Logger.LogError($"LogError {path}");

app.Logger.LogInformation($"LogInformation {path}");

app.Logger.LogWarning($"LogWarning {path}");

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

});

**ILogger**: is responsible to write a log message of a given Log Level.

**ILoggerProvider**: is responsible to create an instance of ILogger (you are not supposed to use ILoggerProvider directly to create a logger)

**ILoggerFactory**: you can register one or more ILoggerProviders with the factory, which in turn uses all of them to create an instance of ILogger. ILoggerFactory holds a collection of ILoggerProviders.

----

ILoggerFactory factory = new LoggerFactory().AddConsole(); // add console provider

factory.AddProvider(new LoggerFileProvider("c:\\log.txt")); // add file provider

Logger logger = factory.CreateLogger(); // creates a console logger and a file logger

----

ILoggerFactory loggerFactory = LoggerFactory.Create(builder => builder.AddConsole());

ILogger logger = loggerFactory.CreateLogger<Program>();

app.Run(async (context) =>

{

logger.LogInformation($"Requested Path: {context.Request.Path}");

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

});

Custom ILogger, custom ILoggerProvider

**Context, Cookies, Sessions**

HttpContext:

1. Request
2. Response
3. Items
4. Session
5. Connection
6. Features
7. TraceIdentifier
8. User

HttpContext.Request:

1. Body
2. Form
3. Method
4. Host
5. Query
6. Headers
7. Path

HttpContext.Response:

1. Body
2. Cookies
3. Headers
4. HttpContext
5. StatusCode

HttpContext.Items

HttpContext.Request.Cookies

HttpContext.Respopnse.Cookies

HttpContext.Session.SetString

HttpContext.Session.Set<Person>

HttpContext.Items – designed to share short-lived per request data

HttpContext.Features – designed to share various HTTP feature to allow middleware to create or modify the application hosting pipeline.

A request Feature is an entity created during a user HTTP request within middleware. The purpose is to allow modify the current application hosting pipeline. **Performance**

[https://learn.microsoft.com/en-us/aspnet/core/fundamentals/request-features?view=aspnetcore-3.0#feature-interfaces](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/request-features?view=aspnetcore-3.0%23feature-interfaces)

**Comparing with HttpContext.Items:**

1. Generic keys items (instead of Object in HttpContext.Items)
2. Constraint, because of **Type** key
3. Micro-optimization: there is no boxing/unboxing because of generic

**HttpContext.Features** //list of supported feature interfaces

1. [IHttpRequestFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.ihttprequestfeature)
2. [IHttpResponseFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.ihttpresponsefeature)
3. [IHttpResponseBodyFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.ihttpresponsebodyfeature)
4. [IHttpAuthenticationFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.authentication.ihttpauthenticationfeature)
5. [IFormFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.iformfeature)
6. [IQueryFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.iqueryfeature)
7. [IRequestBodyPipeFeature](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.http.features.irequestbodypipefeature)

context.Features.Set(…);

context.Features.Get<>()

context.Request.Body.CopyToAsync();

context.Request.EnableBuffering();

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/middleware/request-response?view=aspnetcore-7.0>

**Stream** and **Pipe** (2 abstractions for request/response bodies)

Request reading: HttpRequest.Body (Stream) and **HttpRequest.BodyReader** is a **PipeReader** (Pipe)

Request writing: HttpResponse.Body (stream) and **HttpResponse.BodyWriter** is a **PipeWriter** (Pipe)

[https://learn.microsoft.com/en-us/dotnet/standard/io/pipelines#what-problem-does-systemiopipelines-solve](https://learn.microsoft.com/en-us/dotnet/standard/io/pipelines%23what-problem-does-systemiopipelines-solve)

Pipelines are recommended. Pipes are designed as high-performance parsing of streaming data (Kestrel implementation detail).

Streams example:

Stream requestBody

while (true)

{

var bytesRemaining = await requestBody.ReadAsync(buffer, offset: 0, buffer.Length);

if (bytesRemaining == 0)

{

break;

}

….

}

Pipe example:

PipeReader reader

while (true)

{

ReadResult readResult = await reader.ReadAsync();

var buffer = readResult.Buffer;

SequencePosition? position = null;

do

{

// Look for a EOL in the buffer

position = buffer.PositionOf((byte)'\n');

}

while (position != null);

if (readResult.IsCompleted && buffer.Length > 0)

{

…

}

reader.AdvanceTo(buffer.Start, buffer.End);

if (readResult.IsCompleted)

{

break;

}

}

HttpContext – инкапсулирует всю информацию касательно Request/Response

**Exceptions**

1. Developer Exception Page – generates detailed stack traces for server errors. Displayed by default for Developer environment.

if (context.HostingEnvironment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

app.UseDeveloperExceptionPage();

app.Run(async (context) =>

{

int a = 5;

int b = 0;

int c = a / b;

await context.Response.WriteAsync($"c = {c}");

});

app.Run();

1. Exception handler

Generates an error payload:

app.UseExceptionHandler(exceptionHandlerApp

=> exceptionHandlerApp.Run(async context

=> await Results.Problem()

.ExecuteAsync(context)));

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

app.Environment.EnvironmentName = "Production"; // меняем имя окружения

// если приложение не находится в процессе разработки

// перенаправляем по адресу "/error"

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

}

// middleware, которое обрабатывает исключение

app.Map("/error", app => app.Run(async context =>

{

context.Response.StatusCode = 500;

await context.Response.WriteAsync("Error 500. DivideByZeroException occurred!");

}));

// middleware, где генерируется исключение

app.Run(async (context) =>

{

int a = 5;

int b = 0;

int c = a / b;

await context.Response.WriteAsync($"c = {c}");

});

app.Run();

1. StatusCodePagesMiddleware

var builder = WebApplication.CreateBuilder();

var app = builder.Build();

// обработка ошибок HTTP

app.UseStatusCodePages(async statusCodeContext =>

{

var response = statusCodeContext.HttpContext.Response;

var path = statusCodeContext.HttpContext.Request.Path;

response.ContentType = "text/plain; charset=UTF-8";

if (response.StatusCode == 403)

{

await response.WriteAsync($"Path: {path}. Access Denied ");

}

else if (response.StatusCode == 404)

{

await response.WriteAsync($"Resource {path} Not Found");

}

});

app.Map("/hello", () => "Hello ASP.NET Core");

app.Run();

1. DeveloperExceptionPageMiddleware

Adding .AddProblemDetails() extension

var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

DeveloperExceptionPageOptions developerExceptionPageOptions = new DeveloperExceptionPageOptions

{

SourceCodeLineCount = 5

};

app.UseDeveloperExceptionPage(developerExceptionPageOptions);

}

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapGet("/", async context =>

{

//Throwing an Exception

throw new Exception("Error Occurred while processing your request");

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

});

});

//This will Run the Application

app.Run();

**Results API** – специальный класс для упрощения отправки ответа

app.Map("/results", () => Results.Text("Hello, world!"));

**Controllers:**

[ApiController]

[Route(“[controller”])

: **ControllerBase** interface is used for web pages (not Web API)

: **Controller** interface is used for both Web pages and Web API

[FromBody], [FromForm], [FromHeader], [FromQuery], [FromRoute], [FromServices] - injection

Accepted(), .BadRequest(), .Challenge() – returns 401 error or redirects to login page

Content(), Created(), Json(), Ok(), Unauthorized()

**ControllerBase:**

* CreatedAtAction
* BadRequest
* NotFound
* PhysicalFile
* TryUpdateModelAsync
* TryValidateModelAsync

**Results**.Json()

**Results**.LocalRedirect()

[**Produces**] – specifies the expected Type the action and return supported content types

**[Consumes] -** specifies the data types that action accepts to consume

Attributes:

[Route]

[Bind]

[HttpGet, [HttpPost], [HttpPut], [HttpDelete], [HttpPatch]

Swagger:

builder.Services.AddSwaggerGen();

app.UseSwagger();

app.UseSwaggerUI();

builder.Services.AddSwaggerGen(options =>

{

options.SwaggerDoc("v1", new OpenApiInfo

{

Version = "v1",

Title = "ToDo API",

Description = "An ASP.NET Core Web API for managing ToDo items",

TermsOfService = new Uri("https://example.com/terms"),

Contact = new OpenApiContact

{

Name = "Example Contact",

Url = new Uri("https://example.com/contact")

},

License = new OpenApiLicense

{

Name = "Example License",

Url = new Uri("https://example.com/license")

}

});

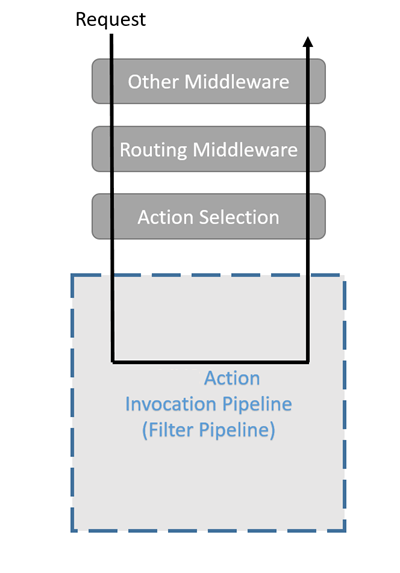
// using System.Reflection;

var xmlFilename = $"{Assembly.GetExecutingAssembly().GetName().Name}.xml"; options.IncludeXmlComments(Path.Combine(AppContext.BaseDirectory, xmlFilename));

});

**Filters:**

<https://learn.microsoft.com/en-us/aspnet/core/fundamentals/minimal-apis/min-api-filters?view=aspnetcore-7.0>



* Authorization
* Resource (OnResourceExecuting, OnResourceExecuted)
* Action – (OnActionExecuting, OnActionExecuted) – can changes arguments, can change result
* Endpoint – can change arguments, can change results
* Exception – global exception policy
* Result – run before and after execution of action results, only if method executed successfully.

**IActionFilter / IAsyncActionFilter** – фильтры действий (привязка модели, изменение входных данных)

OnActionExecuting / OnActionExecuted

**IResourceFilter : Attribute -** ресурсные фильтры, могут переписывать респонс

OnResourceExecuting / OnResourceExecuted

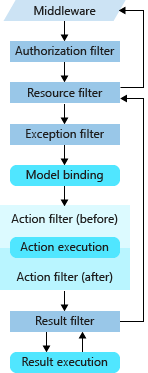
**IResultFilter / IAsyncResultFilter –** вызываются после всех остальных фильтров

OnResultExecuting / OnResultExecuted (context.Result, context.Cancel)

**IExeceptionFilter –** исключение, которое возникает только в Actions конроллера

OnException (context.ExceptionHandled = true)

**AuthorizationFilter** – первый фильтр в пайплайне, нельзя бросать исключение



public class ResponseHeaderAttribute : ActionFilterAttribute

{

private readonly string \_name;

private readonly string \_value;

public ResponseHeaderAttribute(string name, string value) =>

(\_name, \_value) = (name, value);

public override void OnResultExecuting(ResultExecutingContext context)

{

context.HttpContext.Response.Headers.Add(\_name, \_value);

base.OnResultExecuting(context);

}

}

[ResponseHeader("Filter-Header", "Filter Value")]

public class ResponseHeaderController : ControllerBase

{

public IActionResult Index() =>

Content("Examine the response headers using the F12 developer tools.");

// ...

builder.Services.AddControllersWithViews(options =>

{

options.Filters.Add<GlobalSampleActionFilter>();

});

builder.Services.AddDistributedMemoryCache();

public ProductController(IMemoryCache memoryCache)

{

}

this.memoryCache.Set("12345678", new { Id = 1, Name = "Bla-bla-bla" });

var cacheItem = this.memoryCache.Get("12345678");

**Authorization, Authentication**

Authentication is a process of determining a user’s identity.

Authorization is a process of determining whether user has an access.

Authentication:

* Cookies
* JWT Token

Authorization:

* Claims
* Role-based
* Policy-based
* Resource-based

Authentication schema:

CookieAuthenticationDefaults.AuthenticationScheme “Cookies”

JwtBearerDefaults.AuthenticationScheme “Bearer”

builder.Services.AddAuthentication(<schema>)

.AddJwtBearer();

//or .AddCookie();

app.UseAuthentication();

app.UseAuthorization();

Role-based authorization (Roles are Claims, Roles are group of identical collection of users)

<https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-7.0>

Claims-based authorization (Not all Claims are Roles, Claims are information about user)

<https://learn.microsoft.com/en-us/aspnet/core/security/authorization/claims?view=aspnetcore-7.0>

Policy-based authorization

<https://learn.microsoft.com/en-us/aspnet/core/security/authorization/policies?view=aspnetcore-7.0>

Resource-based authorization

<https://learn.microsoft.com/en-us/aspnet/core/security/authorization/resourcebased?view=aspnetcore-7.0>

IAuthenticationService

<https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.authentication.iauthenticationservice?view=aspnetcore-7.0>