

The `IWebHost` is created in `Program` using the Builder pattern, and the `CreateDefaultBuilder` helper method.

The `WebHostBuilder` calls out to `Startup` to configure your application.

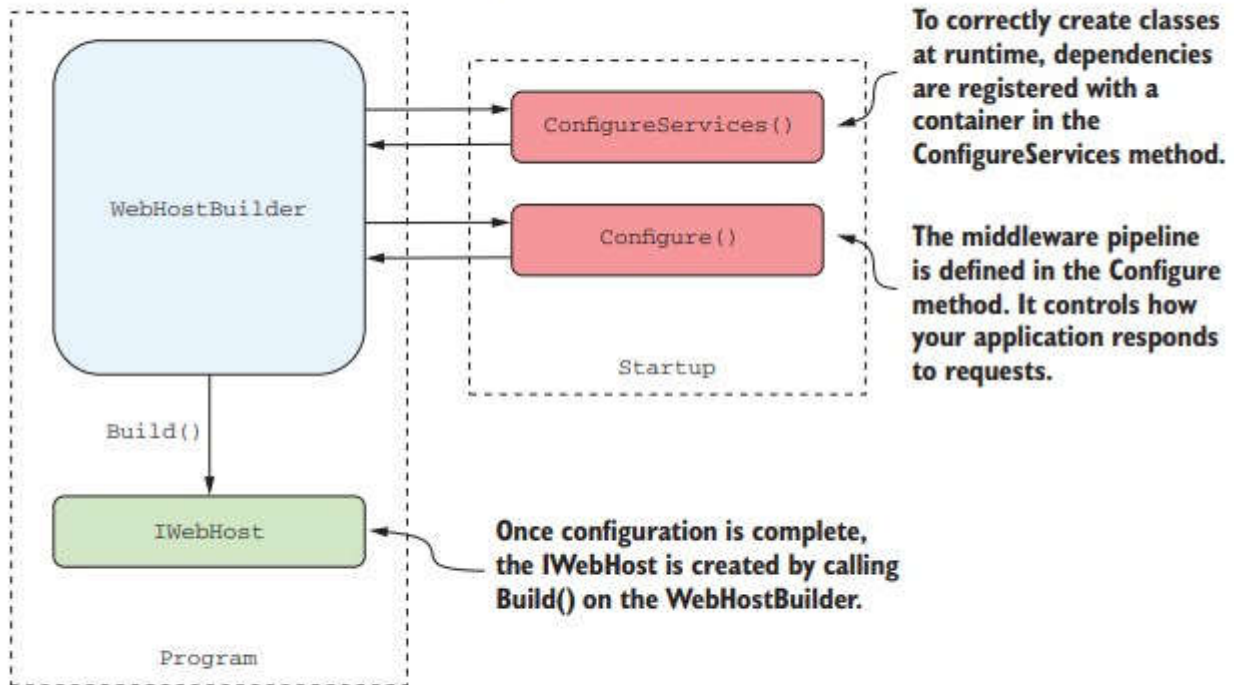


Figure 2.9 The `WebHostBuilder` is created in `Program.cs` and calls methods on `Startup` to configure the application's services and middleware pipeline. Once configuration is complete, the `IWebHost` is created by calling `Build()` on the `WebHostBuilder`.

`IHostBuilder` this generic host introduced with ASP.NET Core 3.0 and .NET Core 3.0 basically replaces the previous `IWebHost` and `IWebHostBuilder`

prvo se povikuje `ConfigureServices()` pa `Configure()` (vo `Configure` se postavuvaat samo middlewares) `Run()`; runs the `IWebHost`, start listening for requests and generating responses.

IServiceCollection services is the Dependency Injection container you can add services with services.Add.... and they can be configured with anonymous method in the Add.... method

```
public void ConfigureServices(IServiceCollection services)
{
    // The following example adds support for controllers, API-related features, and views, but
    not pages.

    services.AddControllersWithViews();

    services.AddAuthentication(); // manages authentication of User

    services.AddAuthorization(); // manages authorization of User

    services.AddIdentity<IdentityUser, IdentityRole>() // Adds the default identity system configuration
    for the specified User and Role types and manages them.

    services.ConfigureApplicationCookie() // Configure the app's cookie for the asp.net core identity
    in Startup.ConfigureServices. ConfigureApplicationCookie must be
    called after calling AddIdentity or AddDefaultIdentity.

    services.AddIdentity<UserIdentityUser, IdentityRole>().AddEntityFrameworkStores<AppDbContext>();
    // The Entity Framework database context to use.

    services.AddIdentity<UserIdentityUser, IdentityRole>().AddDefaultTokenProviders(); // Adds the
    default token providers used to generate tokens for reset passwords, change email and change
    telephone number operations, and for two factor authentication token generation.

    // Add Controllers without Views (usually for APIs)
    services.AddControllers();

    // AddMvc in the ConfigureServices added Controllers and Razor Pages, and now they've been
    separated:
    services.AddRazorPages(); // Adds services for pages

    var con_string = configuration["ConnectionStrings:MyConnectionString"]; //getting the connection
    string
```

```
services.AddDbContext<AppDbContext>({options =>
    options.UseSqlServer(configuration.GetConnectionString("MyConnectionString"))});
// Your DbContext type can be added to the service container by using
the AddDbContext<TContext> method.
```

```
}
```

Configure the middleware pipeline

app.Use() may call next middleware component in the pipeline. On the other hand, middleware defined using app.Run() will never call subsequent middleware they are terminal.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage(); // When an exception is thrown and propagates up the
        pipeline to this middleware, it will be captured. The middleware then generates a friendly HTML
        page, which it returns with a 500 status code to the user
    }
    else
    {
        app.UseExceptionHandler(); //Provides a user-friendly generic error page in production
        app.UseHsts();
    }

    app.UseStaticFiles(); // by default asp.net core will not serve static files
```

the default directory for static files is wwwroot to serve static files use middleware `app.UseStaticFiles()`; This extension method enables support for serving static content from the `wwwroot` folder

`app.UseDefaultFiles()`; // To serve a default page from wwwroot without a fully qualified (empty url) URI in wwwroot, it changes the url to `default.htm` `default.ht` `index.htm` `index.html` but does not serve it.

`app.UseFileServer()`; //combines `app.UseStaticFiles()` and `app.UseDefaultFiles()`;

`UseRouting()` //Will try to select a route to execute but doesn't actually execute the route. It will select the endpoint for the current request.

`UseAuthentication()` //Populates the `User` (code that runs before this won't have a valid `HttpContext.User` property)

`UseAuthorization()` //Will look at the populated user and the current endpoint to determine if an authorization policy needs to be applied.

`UseEndpoints()` //Executes the current endpoint

//`app.UseRouting()`; raboti samo so `app.UseEndpoints()` se koristat zaedno
`app.UseRouting()`;

```
app.UseEndpoints(endpoints =>
{
```

```
    endpoints.MapGet("/ping/pong", async context =>
    {
        await context.Response.WriteAsync("Ping-Pong");
    });
```

// Mapping of controllers now takes place inside `UseEndpoints`.

```
endpoints.MapRazorPages(); // Adds endpoints for Razor Pages
```

```

endpoint.MapControllers(); // adds support for attribute-routed controllers.

endpoints.MapDefaultControllerRoute(); // Adds endpoints for controller actions to
the IEndpointRouteBuilder and adds the default route {controller=Home}/{action=Index}/{id?}.

endpoints.MapControllerRoute();
// Adds endpoints for controller actions to the IEndpointRouteBuilder and specifies a route with the
given name, pattern, defaults, constraints, and dataTokens.

});

app.UseStatusCodePages(); //Converts raw error status codes into simple page with satus code.

app.UseStatusCodePagesWithRedirects("/Error/{0}"); // to return a custom error view.

app.UseStatusCodePagesWithReExecute("/Error{0}"); // to return a custom error view.


app.UseHttpsRedirection();
    app.UseCookiePolicy();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UseSession();
    app.UseDefaultFiles();

}

```

Routing

These are added at two distinct points in the middleware pipeline, as they serve two distinct roles. Generally speaking, you want the routing middleware to be *early* in the pipeline, so that subsequent middleware has access to the information about the endpoint that will be executed. The invocation of the endpoint should happen at the *end* of the pipeline. For example:

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();

    // Add the EndpointRoutingMiddleware
    app.UseRouting();

    // All middleware from here onwards know which endpoint will be invoked
    app.UseCors();

    // Execute the endpoint selected by the routing middleware
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapDefaultControllerRoute();
    });
}
```

Branching

Listing 19.2 Using the Map extension to create branching middleware pipelines

```
public void Configure(IApplicationBuilder app)
{
    app.UseDeveloperExceptionPage();
    app.Map("/ping", branch =>
    {
        branch.UseExceptionHandler();
        branch.Run(async (context) =>
        {
            context.Response.ContentType = "text/plain";
            await context.Response.WriteAsync("pong");
        });
    });
    app.UseMvc();
}
```

Every request will pass through this middleware.

The Map extension method will branch if a request starts with /ping.

This middleware will only run for requests matching the /ping branch.

The MvcMiddleware will run for requests that don't match the /ping branch.

The Run extension always returns a response, but only on the /ping branch.

Configuration of services in Dependency Injection Container

```
services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(configuration.GetConnectionString("MyConnectionString"))
    .EnableSensitiveDataLogging() //default parameters are hidden this will enable them to be
    showed
);

services.AddControllersWithViews(configure=> {
    var policy = new AuthorizationPolicyBuilder()
    .RequireAuthenticatedUser() // /// adding Authorization(the user must be logged in)
    .AddRequirements(new SomeClass) //za custom SomeClass koja ke bide Requirement odnosno ke
    nasledi od IAuthorizationRequirement
    .Build();
    configure.Filters.Add(new AuthorizeFilter(policy)); /// adding Filters globally, za da se iskoristat
    policy treba [Authorize("policy name")]
    }). AddNewtonsoftJson(opt =>
    {
/*
Install-Package Microsoft.AspNetCore.Mvc.NewtonsoftJson
ReferenceLoopHandling.Ignore;      ako e ignore ne pecati loop, ne go pecati prop
ReferenceLoopHandling.Error;      error frla isklucok
ReferenceLoopHandling.Serialize;   pecati loop
*/
        opt.SerializerSettings.ReferenceLoopHandling = ReferenceLoopHandling.Ignore;

        opt.SerializerSettings.NullValueHandling = Newtonsoft.Json.NullValueHandling.Ignore);
        //ignore null values not tested

    });

// UserIdentityUser e custom moze i IdentityUser
services.AddIdentity<UserIdentityUser, IdentityRole>(setupAction=> {
    setupAction.Password.RequiredLength = 4;
    setupAction.Password.RequiredUniqueChars = 1;
    setupAction.User.RequireUniqueEmail = true; // Requires each user to have a unique email.
    default is false
```

```
options.SignIn.RequireConfirmedEmail = true; //
```

Requires a confirmed email to sign in. default is false

```
}).AddEntityFrameworkStores<AppDbContext>();
```

//Configure the app's cookie for asp.net core identity
in Startup.ConfigureServices. ConfigureApplicationCookie must be
called after calling AddIdentity or AddDefaultIdentity.

```
services.ConfigureApplicationCookie(options =>
{
    options.AccessDeniedPath = "/Identity/Account/AccessDenied";
    options.Cookie.Name = "YourAppCookieName";
    options.Cookie.HttpOnly = true;
    options.ExpireTimeSpan = TimeSpan.FromMinutes(60);
    options.LoginPath = "/Identity/Account/Login";
    // ReturnUrlParameter requires
    //using Microsoft.AspNetCore.Authentication.Cookies;
    options.ReturnUrlParameter = CookieAuthenticationDefaults.ReturnUrlParameter;
    options.SlidingExpiration = true;
});
```

/// adding Filters globally

```
services.AddControllersWithViews(configure=> {
    var policy = new AuthorizationPolicyBuilder()
.RequireAuthenticatedUser() // /// adding Authorization(the user must be logged in)
.AddRequirements(new SomeClass) //za custom SomeClass koja ke bide Requirement odnosno ke
nasledi od IAuthorizationRequirement
.Build();
configure.Filters.Add(new AuthorizeFilter(policy)); /// adding Filters globally
});
```

```
services.AddControllers(configure =>
{
    //so true ako go nema vo lista na outputformatter ke vrati status code 406 Not Acceptable
    //so false vraka default
    //json go ima vo listata toj e default
    configure.ReturnHttpNotAcceptable = true;
```



```
configure.OutputFormatters.Add(new XmlDataContractSerializerOutputFormatter());
});
```

//za da se iskoristat policy treba [Authorize("policy name")]

```
services.AddAuthorization(configure =>
```

```
{
    configure.AddPolicy("CanEnterSecurity", policyBuilder =>
```

```
        policyBuilder.RequireClaim("BoardingPassNumber")
        .RequireRole("Admin") //the user must have roleName Admin
```

.RequireClaim("key", "value") //The user must have the specified claim. Optionally, with one of the specified values.

.RequireClaim("key", "value", "value1", "value2") //A list of allowed values can also be specified. the user must have key claim with a value of claim1 or claim2, or claim3

.RequireAuthenticatedUser() //The required user must be authenticated. Creates a policy similar to the default[Authorize] attribute, where you don't set a policy

```
.RequireUserName("username") //The user must have the specified username.
```

.RequireAssertion(context => //Executes the provided lambda function, which returns a bool, indicating whether the policy was satisfied.

```
    context.User.IsInRole("Admin") &&
    context.User.HasClaim(claim => claim.Type == "Edit Role" && claim.Value ==
    "true") ||
    context.User.IsInRole("Super Admin"))
```

```
.Requirements(new ManageAdminRolesAndClaimsRequirement()) //custom requirement
```

.AuthenticationSchemes.Add(CookieAuthenticationDefaults.AuthenticationScheme)//Selecting the scheme can be done with policies only this policy only runs against the identity created by the cookie authentication

options.InvokeHandlersAfterFailure = false; // If you do not want the rest of the handlers to be called, when a failure is returned, set InvokeHandlersAfterFailure property to false. The default is true.

```
);
```

```
});
```

```

services.AddAuthentication("AuthScheme") //we specify the default auth Scheme,the scheme to use
by default when a specific scheme isn't requested. with [Authorize] attribute
//or
services.AddAuthentication(config=> {
    //default auth scheme // here we use Cookie for authentication
    config.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
});
.AddCookie(); //if asp.net core identity is used must be called for cookie explicitly if
services.AddAuthentication() is used asp.net core uses cookie internally if you want to configure call
ConfigureApplicationCookie AddCookie() can be used without asp.net core identity

.AddCookie(); vs ConfigureApplicationCookie
when you use asp.net core identity call ConfigureApplicationCookie to configure the cookie
when you do not use asp.net core identity call .AddCookie(); to configure cookie
.AddJwtBearer(); //must be installed from nuget packet manager

```

Configuration on Middlewares

```

endpoints.MapControllerRoute(
    name: "people",
    pattern: "People/{ssn}",
    constraints: new { ssn = "^\\d{3}-\\d{2}-\\d{4}$", },
    defaults: new { controller = "People", action = "List", });
});

app.UseDefaultFiles();
DefaultFilesOptions defaultFilesOptions = new DefaultFilesOptions();
defaultFilesOptions.DefaultFileNames.Clear();
defaultFilesOptions.DefaultFileNames.Add("mydefaults.html"); //adding custom defaults files
app.UseDefaultFiles(defaultFilesOptions);

app.UseFileServer();
FileServerOptions fileServerOptions = new FileServerOptions();
fileServerOptions.DefaultFileOptions.DefaultFileNames.Clear();
fileServerOptions.DefaultFileOptions.DefaultFileNames.Add("mydefault.html"); //adding custom
files
app.UseFileServer(fileServerOptions);

```

```
endpoints.MapControllerRoute();
```

```
// Adds endpoints for controller actions to the IEndpointRouteBuilder and specifies a route with the  
given name, pattern, defaults, constraints, and dataTokens.
```

name [String](#) The name of the route.

pattern [String](#) The URL pattern of the route.

defaults [Object](#) An object that contains default values for route parameters. The object's properties represent the names and values of the default values.

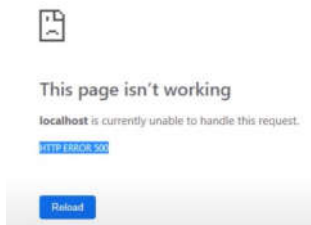
constraints [Object](#) An object that contains constraints for the route. The object's properties represent the names and values of the constraints.

dataTokens [Object](#) An object that contains data tokens for the route. The object's properties represent the names and values of the data tokens.

2 types of error pages

exceptions and error status codes.

exception generic page with no exception middleware



generic status code error page with no status page middleware

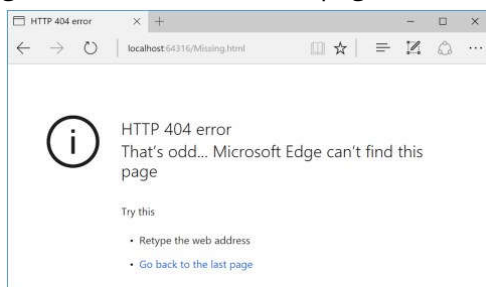


Figure 3.17 A generic browser error page. If the middleware pipeline can't handle a request, it will return a 404 error to the user. The message is of limited usefulness to users and may leave many confused or thinking your web application is broken.

default behavior when you don't add error-handling middleware to your application return just status code

Without handling these status codes, users will see a generic error page, such as in figure 3.17, which may leave many confused and thinking your application is broken. A better approach would be to

handle these error codes and return an error page that's in keeping with the rest of your application or, at the very least, doesn't make your application look broken

exceptions middleware are `DeveloperExceptionPageMiddleware` and `ExceptionHandlerMiddleware`
exceptions middleware catches exceptions

exception middleware responds with body and status code 500

error status code middleware are `UseStatusCodePages` `UseStatusCodePagesWithRedirects` and `UseStatusCodePagesWithReExecute`

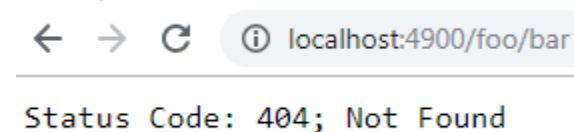
error status code middleware catches response that has an HTTP Status code that starts with 4xx or 5xx and has no response body

error status code middleware responds with body and status code (404 if not) if you don't want body for Api example you should Disable Status Code Pages the default behavior when you don't add error-handling middleware to your application return just status code but if you do add status code middleware and want to only return status code

Disable status code middleware and maybe works for exceptions

```
public string Index()
{
    var statusCodePagesFeature = HttpContext.Features
        .Get<IStatusCodePagesFeature>();
    if (statusCodePagesFeature != null)
    {
        statusCodePagesFeature.Enabled = false;
    }
    return StatusCode(500);
}
```

`app.UseStatusCodePages();` //Converts raw error status codes into simple page with status code
the middleware will intercept any response that has an HTTP Status code that starts with 4xx or 5xx and has no response body. For the simplest case, where you don't provide any additional configuration, the middleware will add a plain text response body, indicating the type and name of the response



`app.UseStatusCodePagesWithRedirects("/Error/{0}");` //to return a custom error view.

if there is a 404 error, the user is redirected to `/Error/404`. The placeholder `{0}`, in `/Error/{0}` will automatically receive the http status code.

// If there is 404 status code, the route path will become Error/404

```
[Route("Error/{statusCode}")]
```

```
public IActionResult HttpStatusCodeHandler(int statusCode)
```

```
{
```

```
    switch (statusCode)
```

```
    {
```

```
        case 404:
```

```
            ViewBag.ErrorMessage = "Sorry, the resource you requested could not be found";
```






```
            break;
```

```
    }
```

```
    return View("NotFound");
```

```
}
```

/foo/bar does not exists

| Name | Status | Initiator |
|---|--------|--------------------------------|
|  bar | 302 | Other |
|  404 | 200 | :15410/foo/bar |
|  bootstrap.mi... | 200 | 404 |
|  employees.png | 200 | 404 |
|  site.css | 200 | 404 |

app.UseStatusCodePagesWithReExecute("/Error{0}") //to return a custom error view.

With the following line in place, if there is a 404 error, the user is redirected to /Error/404. The placeholder {0}, in "/Error/{0}" will automatically receive the http status code.

// If there is 404 status code, the route path will become Error/404

```
[Route("Error/{statusCode}")]
```

```
public IActionResult HttpStatusCodeHandler(int statusCode)
```

```
{
```

```
    switch (statusCode)
```

```
    {
```

```
        case 404:
```

```
            ViewBag.ErrorMessage = "Sorry, the resource you requested could not be found";
```

```
            break;
```

```
    }
```

```
    return View("NotFound");
```

```
}
```

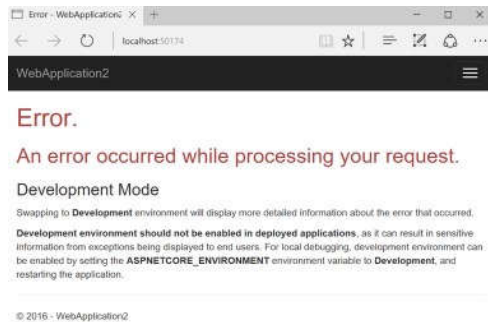
| Name | Status | Initiator |
|---|--------|---------------------|
|  bar | 404 | Other |
|  bootstrap.mi... | 200 | bar |
|  site.css | 200 | bar |
|  employees.png | 200 | bar |

```

app.UseExceptionHandler(); //Provides a user-friendly generic error page in production
app.UseExceptionHandler(appBuilder => { //provides custom error response
    appBuilder.Run(async context =>
    {
        context.Response.StatusCode = 500;
        await context.Response.WriteAsync("Exception occurred try later");
    });
});

```

app.UseExceptionHandler("/Error"); // custom error page



you can get exception details

```

[Route("Error")]
public IActionResult Error()
{
    // Retrieve the exception Details
    var exceptionHandlerPathFeature =
        HttpContext.Features.Get<ExceptionHandlerPathFeature>();

    ViewBag.ExceptionPath = exceptionHandlerPathFeature.Path;
    ViewBag.ExceptionMessage = exceptionHandlerPathFeature.Error.Message;
    ViewBag.StackTrace = exceptionHandlerPathFeature.Error.StackTrace;

    return View("Error");
}

```

```

app.UseDeveloperExceptionPage();
DeveloperExceptionPageOptions developerExceptionPageOptions = new
DeveloperExceptionPageOptions();

```