

precio fishbone



Agenda Asp.Net Core Web Api – EF (4 days)

- Overview
- Startup
- Middleware (Logging)
- Routing
- Format Response Data
- Authentication (Azure AD)
- Static files
- Dependency Injection
- EF – Overview
- EF – Create Model(Migration – AutoMapper)
- EF – Query Data (No Tracking)
- EF – Save Data (Tracking)
- Assignment

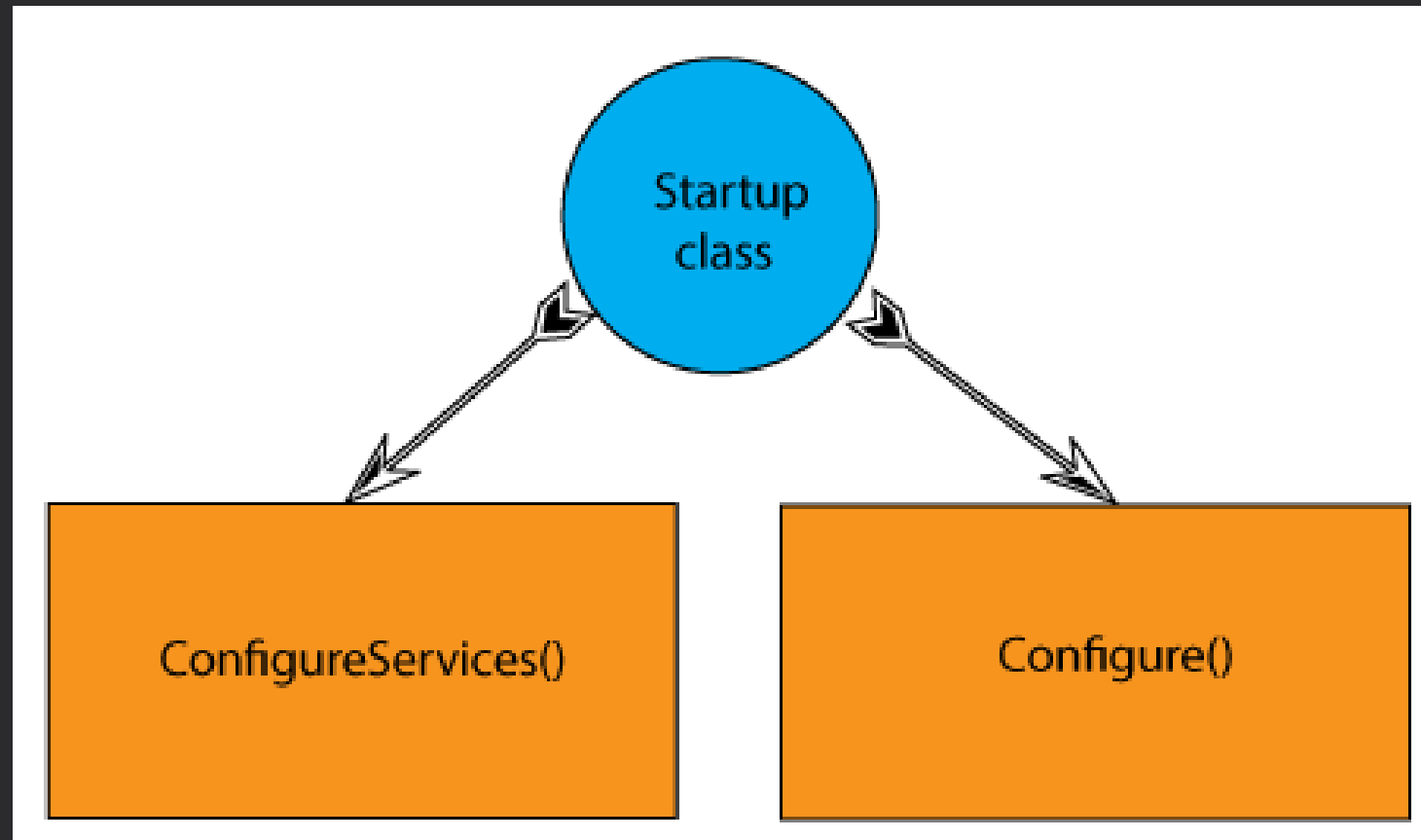


Overview

Why choose ASP.NET Core?



Startup

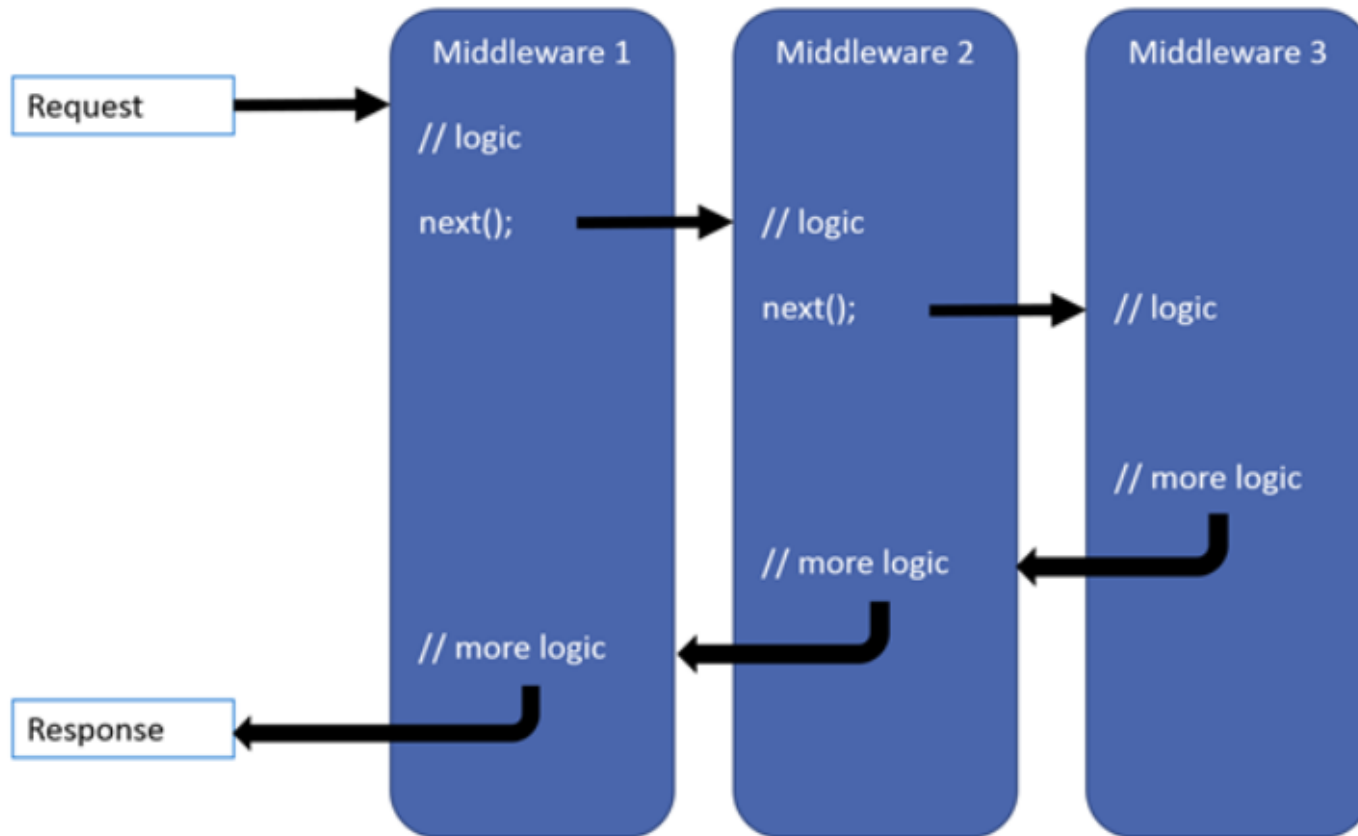


Middleware

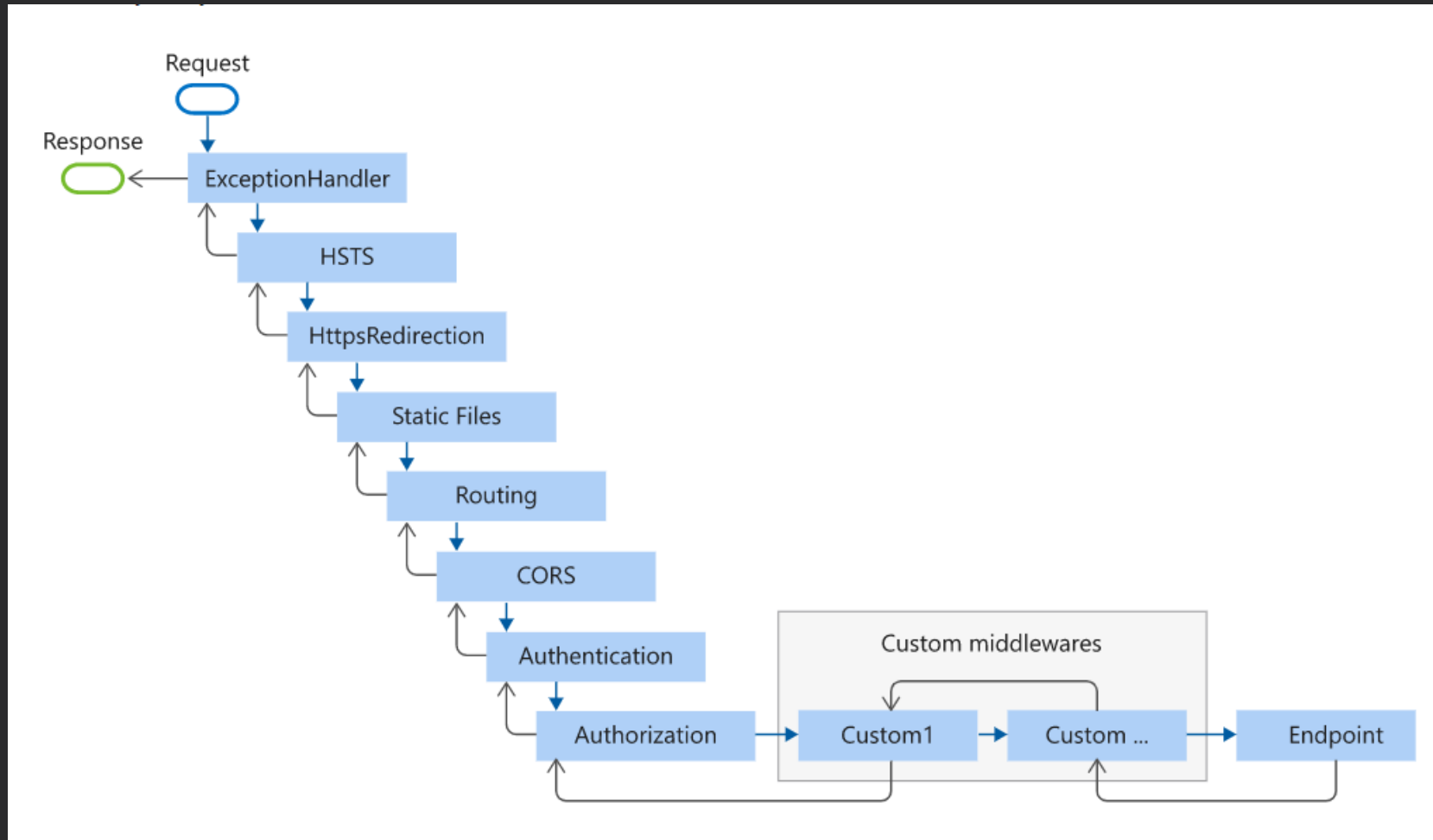
Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:

- Chooses whether to pass the request to the next component in the pipeline.
- Can perform work before and after the next component in the pipeline.

Middleware



Middleware Order



Middleware

- Run
 - The simplest possible ASP.NET Core app sets up a single request delegate that handles all requests. This case doesn't include an actual request pipeline. Instead, a single anonymous function is called in response to every HTTP request.

```
public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello, World!");
        });
    }
}
```


Middleware

- Use
 - Chain multiple request delegates together

```
app.Use(async (context, next) =>
{
    // Do work that doesn't write to the Response.
    await next.Invoke();
    // Do logging or other work that doesn't write to the Response.
});
```

Middleware

- Map

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.

```
private static void HandleMapTest1(IApplicationBuilder app)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Map Test 1");
    });
}

private static void HandleMapTest2(IApplicationBuilder app)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Map Test 2");
    });
}

public void Configure(IApplicationBuilder app)
{
    app.Map("/map1", HandleMapTest1);

    app.Map("/map2", HandleMapTest2);
}
```

Implement IMiddleware

```
public class FactoryActivatedMiddleware : IMiddleware
{
    private readonly AppDbContext _db;

    public FactoryActivatedMiddleware(AppDbContext db)
    {
        _db = db;
    }

    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
        var keyValue = context.Request.Query["key"];

        if (!string.IsNullOrEmpty(keyValue))
        {
            _db.Add(new Request()
            {
                DT = DateTime.UtcNow,
                MiddlewareActivation = "FactoryActivatedMiddleware",
                Value = keyValue
            });


            await _db.SaveChangesAsync();
        }

        await next(context);
    }
}
```

Implement IMiddleware

Extensions are created for the middlewares:

C#

 Copy

```
public static class MiddlewareExtensions
{
    public static IApplicationBuilder UseConventionalMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<ConventionalMiddleware>();
    }

    public static IApplicationBuilder UseFactoryActivatedMiddleware(
        this IApplicationBuilder builder)
    {
        return builder.UseMiddleware<FactoryActivatedMiddleware>();
    }
}
```

Implement IMiddleware

The factory-activated middleware is added to the built-in container in `Startup.ConfigureServices`:

```
C# Copy  
  
public void ConfigureServices(IServiceCollection services)  
{  
    services.AddDbContext<AppDbContext>(options =>  
        options.UseInMemoryDatabase("InMemoryDb"));  
  
    services.AddTransient<FactoryActivatedMiddleware>();  
  
    services.AddRazorPages();  
}
```

Both middlewares are registered in the request processing pipeline in `Startup.Configure`:

```
C# Copy  
  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Error");  
    }  
  
    app.UseConventionalMiddleware();  
    app.UseFactoryActivatedMiddleware();  
}
```


Basic routing

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapGet("/", async context =>
        {
            await context.Response.WriteAsync("Hello World!");
        });
    });
}
```

Routing to controller

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}");
});
```

```
public class ProductsController : Controller
{
    public IActionResult Details(int id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }
}
```

Attribute routing for REST APIs

```
public class HomeController : Controller
{
    [Route("")]
    [Route("Home")]
    [Route("Home/Index")]
    [Route("Home/Index/{id?}")]
    public IActionResult Index(int? id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }

    [Route("Home/About")]
    [Route("Home/About/{id?}")]
    public IActionResult About(int? id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }
}
```

Attribute routing with Http verb attributes

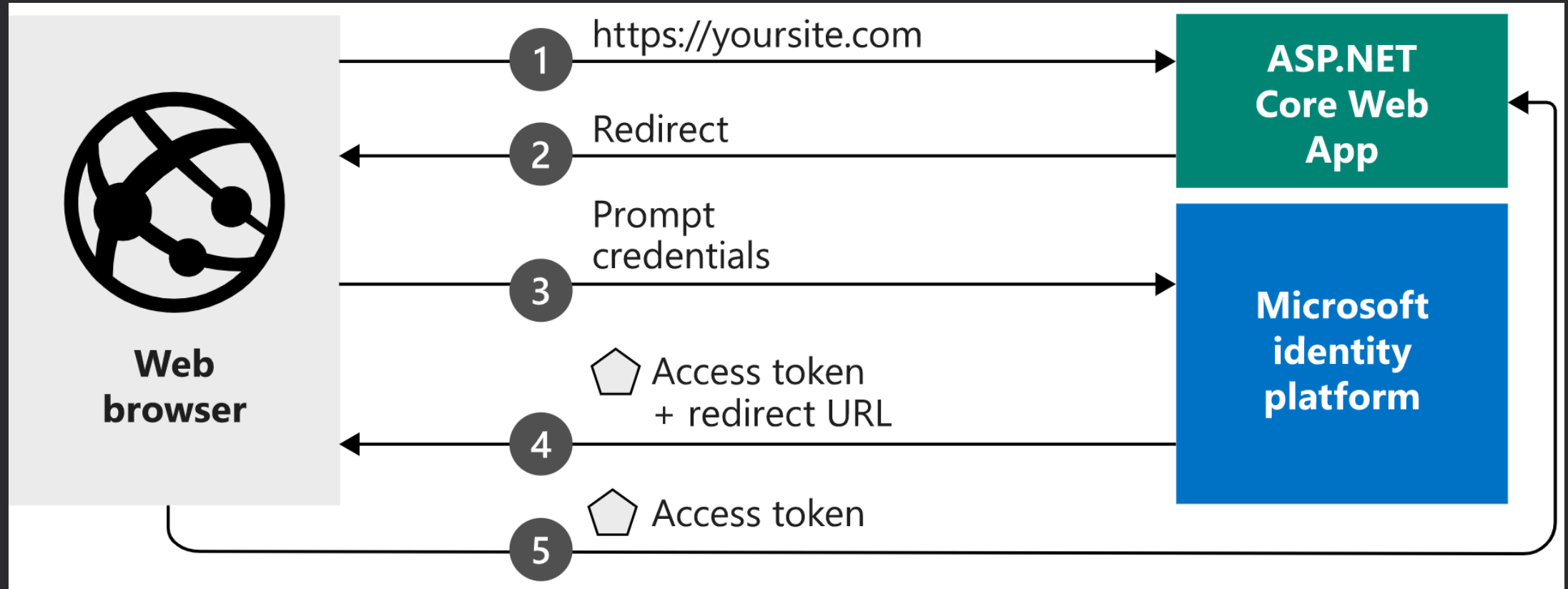
- Common HTTP verb

- [HttpGet]
- [HttpPost]
- [HttpPut]
- [HttpDelete]

```
[ApiController]
[Route("products")]
public class ProductsApiController : ControllerBase
{
    [HttpGet]
    public IActionResult ListProducts()
    {
        return ControllerContext.MyDisplayRouteInfo();
    }

    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
        return ControllerContext.MyDisplayRouteInfo(id);
    }
}
```

Authentication (Azure AD)



Authentication (Azure AD)

appsettings.json

```
"ClientId": "Enter_the_Application_Id_here"  
"TenantId": "Enter_the_Tenant_Info_Here"
```

Authentication (Azure AD)

```
public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        // This lambda determines whether user consent for non-essential cookies is needed for a given request.
        options.CheckConsentNeeded = context => true;
        options.MinimumSameSitePolicy = SameSiteMode.None;
    });

    services.AddAuthentication(AzureADDefaults.AuthenticationScheme)
        .AddAzureAD(options => Configuration.Bind("AzureAd", options));

    services.Configure<OpenIdConnectOptions>(AzureADDefaults.OpenIdScheme, options =>
    {
        options.Authority = options.Authority + "/v2.0/";           // Microsoft identity platform

        options.TokenValidationParameters.ValidateIssuer = false; // accept several tenants (here simplified)
    });

    services.AddMvc(options =>
    {
        var policy = new AuthorizationPolicyBuilder()
            .RequireAuthenticatedUser()
            .Build();
        options.Filters.Add(new AuthorizeFilter(policy));
    })
    .SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
}
```

Authentication (Azure AD)

```
[Authorize]
public class AccountController : Controller
{
    public ActionResult Login()
    {
    }

    public ActionResult Logout()
    {
    }
}
```

Format response data in ASP.NET Core Web API

Format-specific Action Results

- ActionResult
 - Using built-in helper method `Ok` returns JSON-formatted data
 - Response header content-type: `application/json`
- ContentResult
 - Using `Content` helper method to return plain text
 - Response header content-type: `text/plain`

For actions with multiple return types, return `ActionResult`.

For example, returning different HTTP status codes based on the result of operations performed.

Static files

- What is static files?
 - HTML, CSS, images, and JavaScript files.
 - ASP.NET Core app serves directly to clients by default.
- Where it was store?:
 - Default directory is `/wwwroot`.
 - We can change to other directory using `UseWebRoot`.

Static files

- How to use?:
 - Call the `UseStaticFiles` method in `Startup.Configure`, which enables static files to be served.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseRouting();
}
```

- Serve files outside of web root using Static File Middleware:

```
// using Microsoft.Extensions.FileProviders;
// using System.IO;
app.UseStaticFiles(new StaticFileOptions
{
    FileProvider = new PhysicalFileProvider(
        Path.Combine(env.ContentRootPath, "MyStaticFiles")),
    RequestPath = "/StaticFiles"
});
```

- `UseDefaultFiles`: Setting a default page provides visitors a starting point on a site
 - `AddDirectoryBrowser`, `UseDirectoryBrowser`: allows directory listing within specified directories

Static files



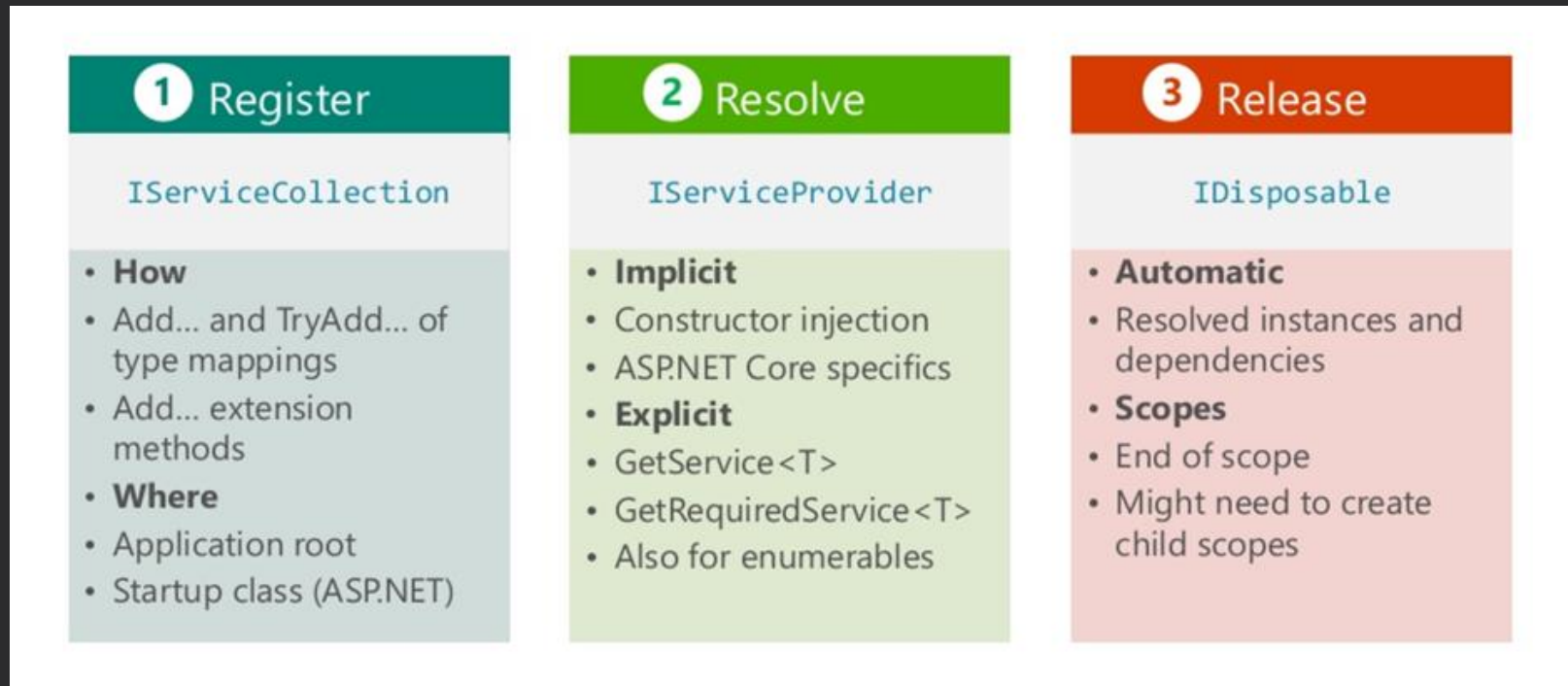
- Authorization
 - Static File Middleware doesn't provide authorization checks, all files are publicly accessible.
- Serve files based on authorization:
 - Store them outside of `wwwroot` and any directory accessible.
 - Serve them via an action method to which authorization is applied.

- Security:
 - UseDirectoryBrowser and UseStaticFiles can leak secrets
 - Disabling directory browsing in production is highly recommended.
 - Review public directories carefully



Dependency Injection

- A technique in which an object receives other objects that it depends on
- Create dependent objects outside of a class and provides those objects to a class through different ways

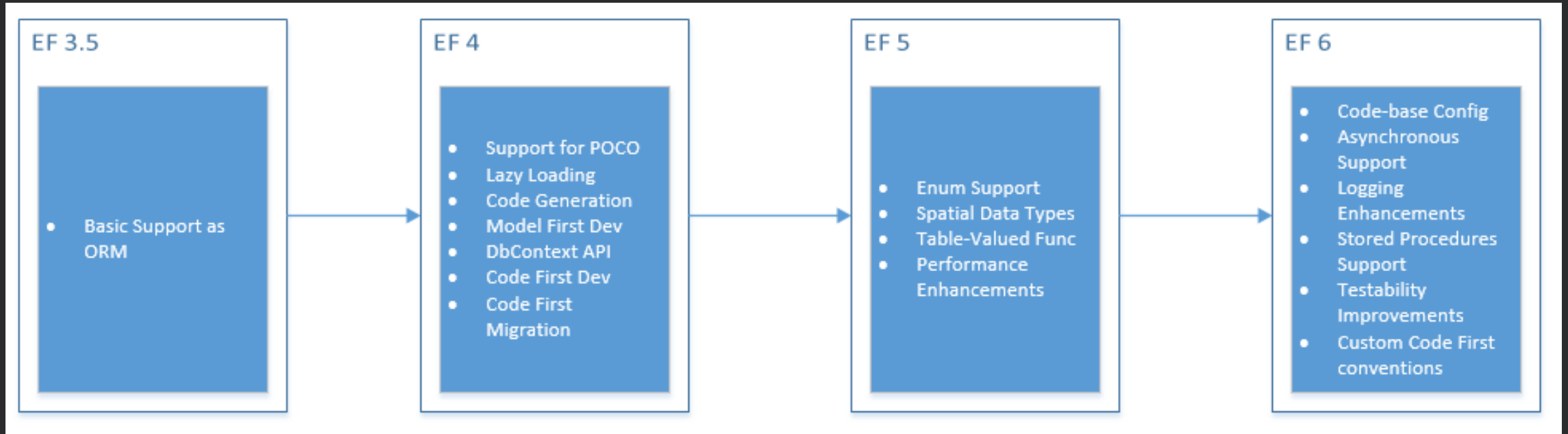


Entity Framework – Overview

- An object-relational mapper to work with relational data using domain-specific objects
 - Writing and managing ADO.Net code for data access
- ORM includes 3 main parts:
 - Domain class objects
 - Relational database objects
 - Mapping information on how domain objects map to relational database objects
- Keep our database design separate from our domain class design => maintainable and extendable
- The main ORM, coupled with the Language-Integrated Query (LINQ) framework, that Microsoft provides for the .NET Framework

Entity Framework – Overview

- History



Entity Framework – Create Model

Add New Item - NGO.Core.Repositories

Installed

Visual C# Items

Code

Data

General

Web

Windows Forms

WPF

SQL Server

Storm Items

Workflow

Online

Sort by: Default

Class

Class for U-SQL

Interface

Windows Form

User Control

Component Class

User Control (WPF)

About Box

ADO.NET Entity Data Model

Application Configuration File

Application Manifest File

Assembly Information File

Bitmap File

Code Analysis Rule Set

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Visual C# Items

Entity Data Model Wizard

Choose Model Contents

What should the model contain?

EF Designer from database

Empty EF Designer model

Empty Code First model

Code First from database

Creates a model in the EF Designer based on an existing database. You can choose the database connection, settings for the model, and database objects to include in the model. The classes your application will interact with are generated from the model.

Entity Framework – Create Model

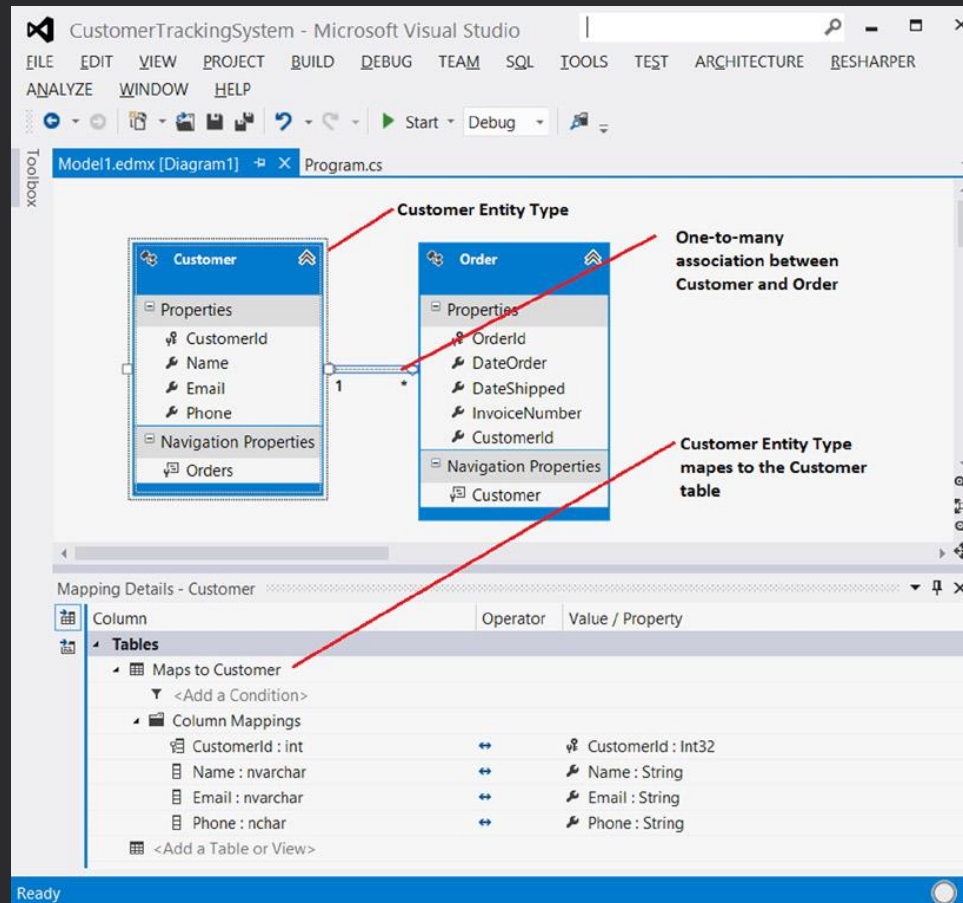
- Code First – FluentAPI
 - Create your models *first* and *Entity framework* will create database according to mappings for you automatically
 - Code first with existing database
- Code first conventions
 - Primary key convention: by default if a property on a class is named ID (ignore case-sensitive)
 - Removing conventions: can remove any of the conventions defined by override *OnModelCreating* method (ex: plural table name)
 - Decimal property convention: decimal(18,2) column
 - Relationship convention
 - Foreign key convention
 - Type discovery
- Code first data annotations

```
public class Employee
{
    public int Id { get; set; }
    public string Surname { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public short Age { get; set; }
    public decimal MonthlySalary { get; set; }

    public string FullName
    {
        get
        {
            return $"{Surname} {FirstName} {LastName}";
        }
    }
}
```

Entity Framework – Create Model

- DB First
 - Generate model from database using T4 code-generation strategy



Connection Properties

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) Change...

Server name: Refresh

Log on to the server

Authentication: Windows Authentication

User name:

Password:

☐ Save my password

Connect to a database

☒ Select or enter a database name:

☐ Attach a database file: Browse...

Logical name:

Advanced...

Test Connection OK Cancel

Entity framework – Tracking queries

- By default, queries that return entity types are **tracking**
- This means you can make changes to those entity instances and have those changes persisted by `SaveChanges()`
- Example:
 - The change to the **book** rating will be detected and persisted to the database during `SaveChanges()`.

```
using (var context = new BookContext())
{
    var book = context.Books.FirstOrDefault(b => b.BookId == 1);
    book.Rating = 5;
    context.SaveChanges();
}
```

Entity framework – Tracking queries

Custom projections

- Even if the result type of the query isn't an entity type, EF Core will still track entity types contained in the result by default. In the following query, which returns an anonymous type, the instances of **Blog** in the result set will be tracked.
- If the result set contains entity types coming out from LINQ composition, EF Core will track them.
-

```
var blog = context.Blogs
    .Select(b =>
        new
        {
            Blog = b,
            PostCount = b.Posts.Count()
        });
```

```
var blog = context.Blogs
    .Select(b =>
        new
        {
            Blog = b,
            Post = b.Posts.OrderBy(p => p.Rating).LastOrDefault()
        });
```

Entity framework – Tracking queries

- Custom projections
 - If the result set doesn't contain any entity types, then no tracking is done. In the following query, we return an anonymous type with some of the values from the entity (but no instances of the actual entity type). There are no tracked entities coming out of the query.











```
var blog = context.Blogs
    .Select(b =>
        new
        {
            Id = b.BlogId,
            Url = b.Url
        });
```

Entity framework – No tracking queries

- No tracking queries are useful when the results are used in a **read-only** scenario
- They are **quicker to execute** because there is no need to setup change tracking information

```
using (var context = new BookContext())  
{  
    var books = context.Books.AsNoTracking().ToList();  
}
```

Documents

 Name ▾	Modified ▾	Modified By ▾	+ Add column
 CAS	April 30	Megan Bowen	
  CoasterAndBargeLoading.xlsx	A few seconds ago	Administrator MOD	
  RevenueByServices.xlsx	A few seconds ago	Administrator MOD	
  RevenueByServices2016.xlsx	A few seconds ago	Administrator MOD	
  RevenueByServices2017.xlsx	A few seconds ago	Administrator MOD	