ITDLU

1

Microsoft
ASP.NET Core
Open Source

ASP.NET Core

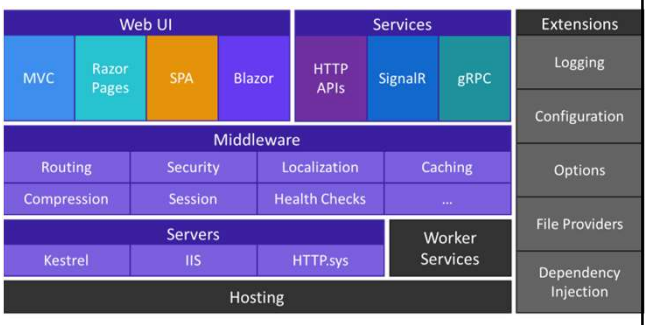
1

ITDLU

2

Agenda

- What is ASP.NET Core? How does it work?
- MVC Framework
- Middleware & Request Pipeline
- Endpoints & URL Routing
- Model Binding & Validation
- Dependency Injection
- Logging
- Caching



2

ITDLU

What is ASP.NET Core?

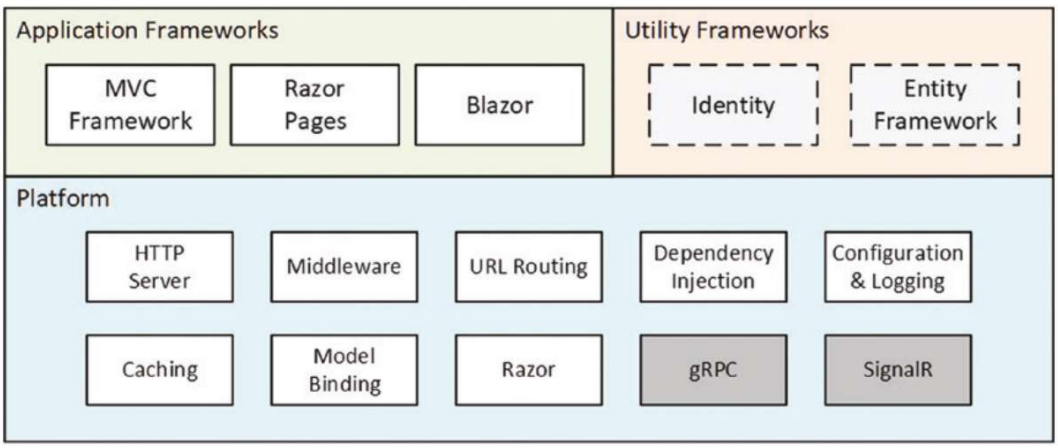
- ASP.NET Core is a cross-platform, open-source, application framework that you can use to quickly build dynamic web applications:
  - Server-rendered web applications
  - Backend server applications
  - HTTP APIs that can be consumed by mobile applications
  - …
- ASP.NET Core provides structure, helper functions, and a framework for building applications, which saves you having to write a lot of this code yourself.

3

ITDLU

What is ASP.NET Core?

- The structure of ASP.NET Core

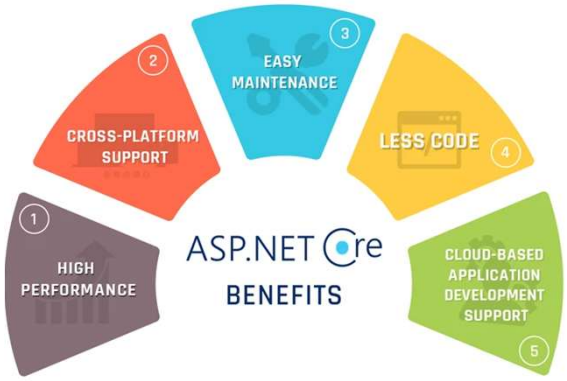


4

ITDLU

Why we choose ASP.NET Core?

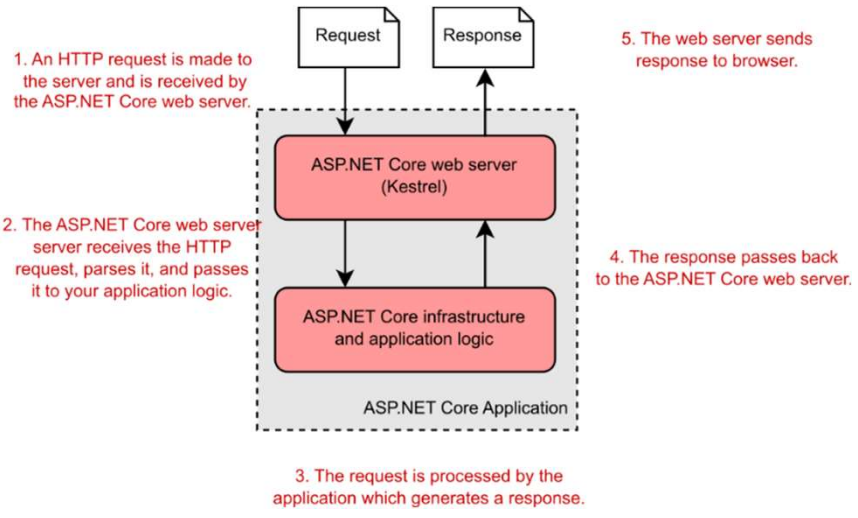
- Advantages:
  - It's a modern, high-performance, open-source web framework.
  - It uses familiar design patterns and paradigms.
  - C# is a great language (or you can use VB.NET or F# if you prefer).
  - You can build and run on any platform.



5

ITDLU

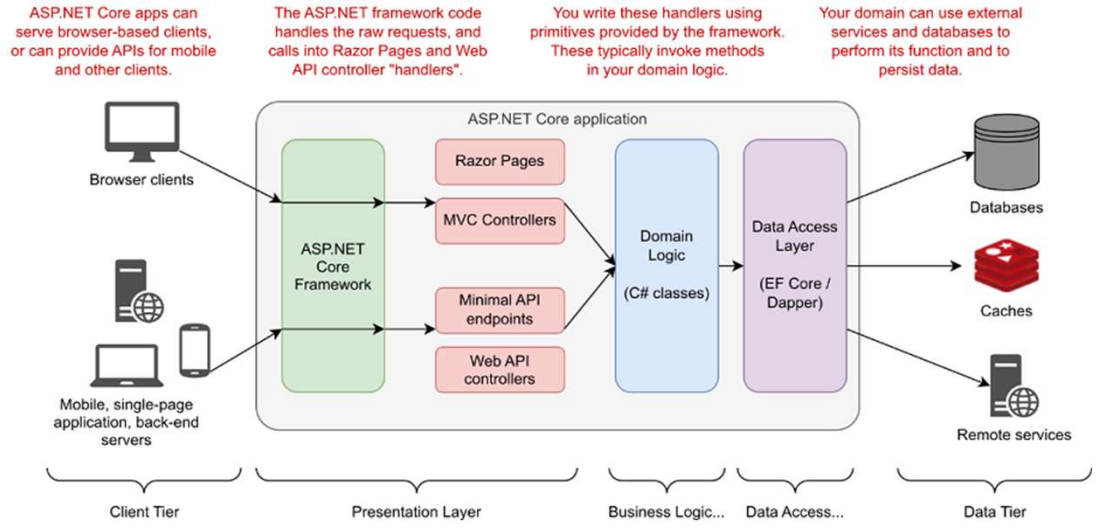
How does ASP.NET Core process a request?



6

ITDLU

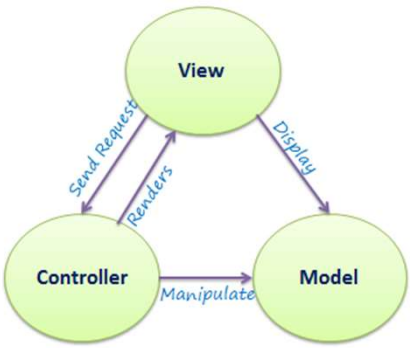
ASP.NET Core Application Layers



ITDLU

The MVC Pattern

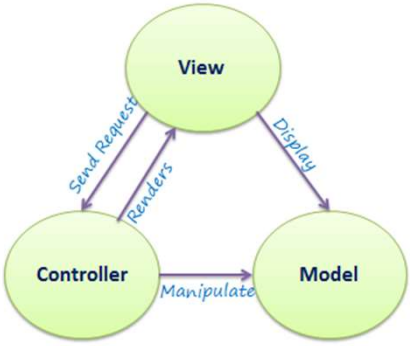
- MVC stands for Model-View-Controller
- A architecture pattern that describes the shape of an application
- Code reusability and separation of concerns



ITDLU

The MVC Pattern

- Model Responsibilities:
  - Represents the state of the application and any business logic or operations that should be performed by it.
  - Business logic should be encapsulated in the model, along with any implementation logic for persisting the state of the application.
  - Strongly-typed views typically use ViewModel types designed to contain the data to display on that view.
  - The controller creates and populates these ViewModel instances from the model.

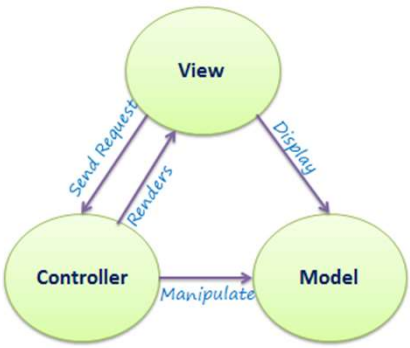


9

ITDLU

The MVC Pattern

- View Responsibilities:
  - Presents content through the user interface.
  - Use the Razor view engine to embed .NET code in HTML markup.
- Views should contain minimal logic and any logic in them should relate to presenting content.

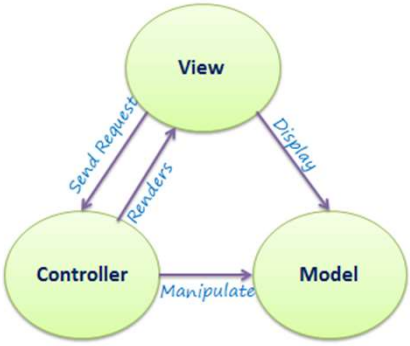


10

ITDLU

The MVC Pattern

- Controller Responsibilities:
  - Handle user interaction, work with the model, and ultimately select a view to render.
  - Handles and responds to user input and interaction
- Every controller has one or more "Actions"



11

ITDLU

ASP.NET Core MVC Framework

- The ASP.NET Core MVC framework is a lightweight, open source, highly testable presentation framework optimized for use with ASP.NET Core.
- Provides a patterns-based way to build dynamic websites that enables a clean separation of concerns. It gives you full control over markup, supports TDD-friendly development and uses the latest web standards.

12

ITDLU 13

Middleware

13

ITDLU 14

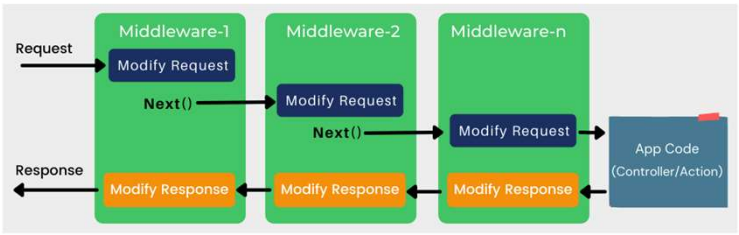
What is middleware?

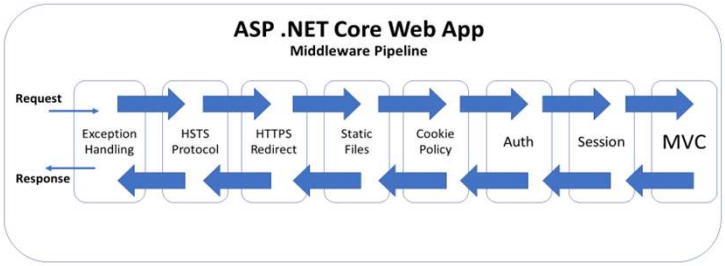
- Middleware are C# classes that can handle an HTTP request or response.
- Middleware can:
  - Handle an incoming HTTP request by generating an HTTP response
  - Process an incoming HTTP request, modify it, and pass it on to another piece of middleware
  - Process an outgoing HTTP response, modify it, and pass it on to either another piece of middleware or the ASP.NET Core web server
- Common use-cases for middleware:
  - Logging each request
  - Adding standard security headers to the response
  - Associating a request with the relevant user
  - Setting the language for the current request

14

ITDLU

Middleware Pipeline





15

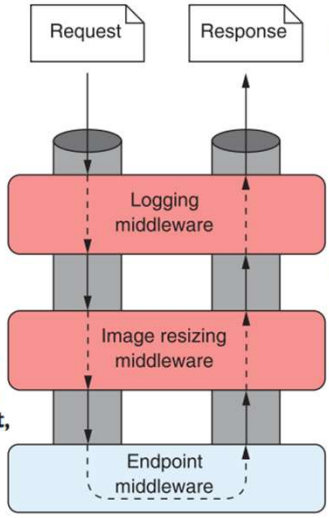
ITDLU

Example of Middleware Pipeline

1. ASP.NET Core web server passes the request to the middleware pipeline.

2. The logging middleware notes down the time the request arrived and passes the request on to the next middleware.

3. If the request is for an image of a specific size, the image resize middleware will handle it. If not, the request is passed on to the next middleware.



6. The response is returned to ASP.NET Core web server.

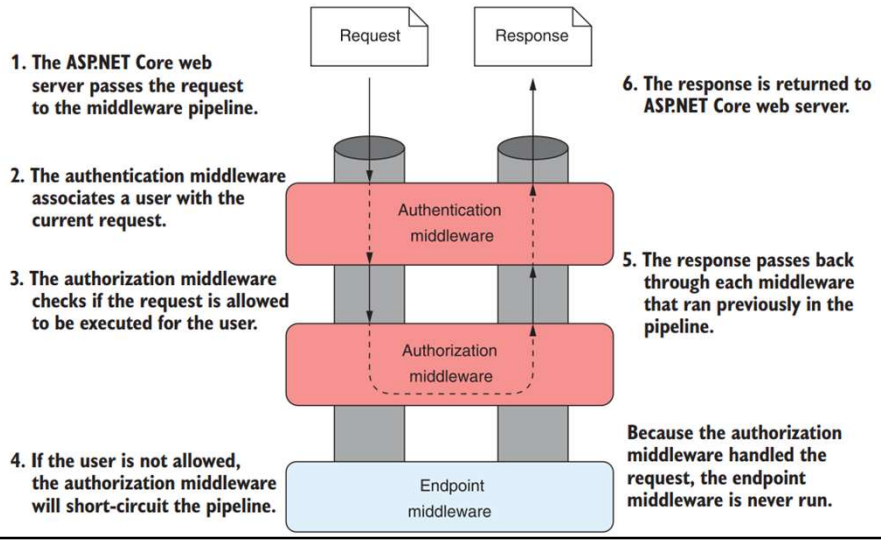
5. The response passes through each middleware that ran previously in the pipeline.

4. If the request makes it through the pipeline to the endpoint middleware, it will handle the request and generate a response.

16

ITDLU

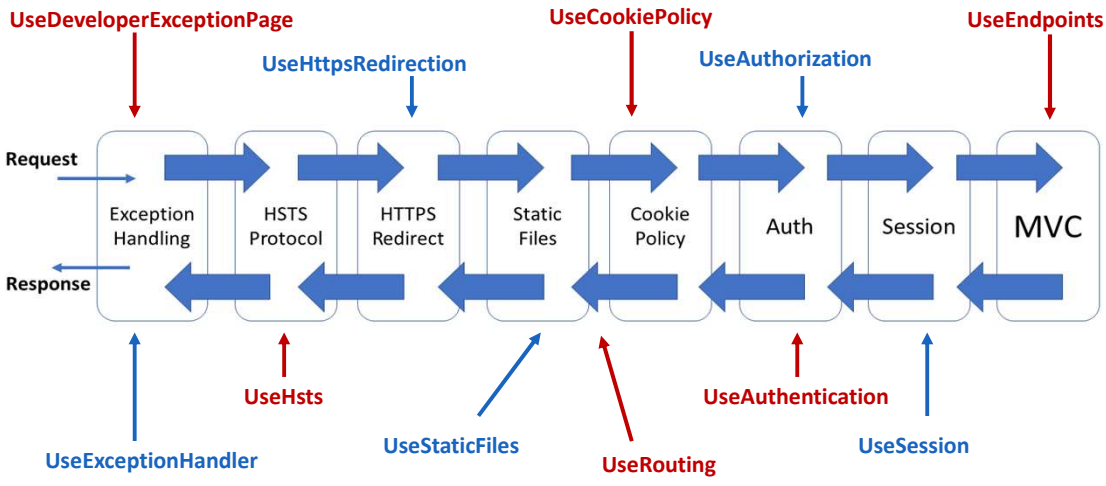
Example of Middleware Pipeline



17

ITDLU

Built-in Middleware



18

ITDLU

Custom Middleware

- Using lambda function (inline middleware)

```
app.Use(async (context, next) => {
    if (context.Request.Method == HttpMethods.Get
            && context.Request.Query["custom"] == "true") {
        context.Response.ContentType = "text/plain";
        await context.Response.WriteAsync("Custom Middleware \n");
    }
    await next();
});
```


ITDLU

Custom Middleware

- Using class

```
public class QueryStringMiddleWare {
    private RequestDelegate next;

    public QueryStringMiddleWare(RequestDelegate nextDelegate) {
        next = nextDelegate;
    }

    public async Task Invoke(HttpContext context) {
        if (context.Request.Method == HttpMethods.Get
                    && context.Request.Query["custom"] == "true") {
            if (!context.Response.HasStarted) {
                context.Response.ContentType = "text/plain";
            }
            await context.Response.WriteAsync("Class-based Middleware \n");
        }
        await next(context);
    }
}

app.UseMiddleware<Platform.QueryStringMiddleWare>();
```

ITDLU

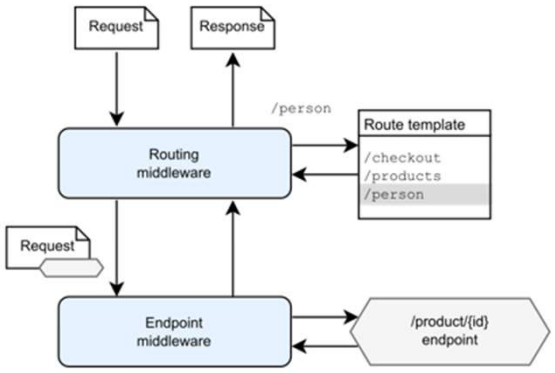
URL Routing

21

ITDLU

ASP.NET Core Routing

- Is the process of selecting a specific handler for an incoming HTTP request.
- In MVC, the handler is an action method in a controller.
- ASP.NET Core controllers use the Routing middleware to match the URLs of incoming requests and map them to actions.



22

ITDLU

ASP.NET Core Routing

- Routing uses a pair of middleware:
  - UseRouting: adds route matching to the middleware pipeline. This middleware looks at the set of endpoints defined in the app, and selects the best match based on the request.
  - UseEndpoints: adds endpoint execution to the middleware pipeline. It runs the delegate associated with the selected endpoint.

```
app.Use(async (context, next) =>
{
    // ...
    await next(context);
});

app.UseRouting();

app.MapGet("/", () => "Hello World!");
```


ITDLU

ASP.NET Core Routing

- Endpoints
  - An Endpoint is an object that contains everything that you need to execute the incoming Request.
- The endpoint object contains:
  - Metadata of the request.
  - The delegate (Request handler) that ASP.NET core uses to process the request.
- Endpoints are defined at the application startup

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

endpoints.MapGet("/hello", async context =>
{
    await context.Response.WriteAsync(helloWorld());
});
```

```
endpoints.MapControllerRoute(
    name: "archives",
    pattern: "archive/{year}/{month}",
    defaults: new { controller = "Blog", action = "Archive" });

endpoints.MapControllerRoute(
    name: "single-post",
    pattern: "post/{year}/{month}/{day}/{slug}",
    defaults: new { controller = "Blog", action = "Post" });

endpoints.MapControllerRoute(
    name: "default",
    pattern: "{action}",
    defaults: new { controller = "Blog", action = "Posts" });
```

ITDLU

25

ASP.NET Core Routing

- Endpoints that can be configure in the ASP.NET Core app
  - MVC Controller Action Method
  - Web API Controller Action Method
  - Razor page
  - SignalR
  - gRPC Services
  - Blazer on the server
  - Endpoint defined in an middleware
  - Delegates and lambdas registered with routing.

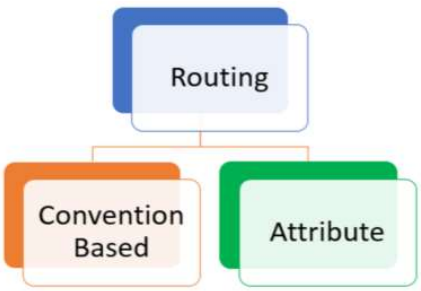
25

ITDLU

26

ASP.NET Core Routing

- Route templates
  - Are defined at startup in Program.cs or in attributes.
  - Describe how URL paths are matched to actions.
  - Are used to generate URLs for links. The generated links are typically returned in responses.



Route parameter

Route template

Route constraint

```
app.MapGet("/hello/{name:alpha}", (string name) => $"Hello {name}!");
```

26

ITDLU

27

ASP.NET Core Routing

- Convention-based routing:

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

- Attribute routing:

```
[Route("api/[controller]")]
public class ProductsController : Controller
{
    [HttpGet("{id}")]
    public IActionResult GetProduct(int id)
    {
      ...
    }
}
```

27

ITDLU

28

ASP.NET Core Routing

- URL matching:
  - Is the process by which routing matches an incoming request to an endpoint.
  - Is based on data in the URL path and headers.
  - Can be extended to consider any data in the request.
- The URL matching phases:
  - Processes the URL path against the set of endpoints and their route templates, collecting all of the matches.
  - Takes the preceding list and removes matches that fail with route constraints applied.
  - Takes the preceding list and removes matches that fail the set of MatcherPolicy instances.
  - Uses the EndpointSelector to make a final decision from the preceding list.

28

ITDLU

ASP.NET Core Routing

- URL matching:

Route Template	Example Matching URI	The request URI...
`hello`	`/hello`	Only matches the single path `/hello`.
`{Page=Home}`	`/`	Matches and sets `Page` to `Home`.
`{Page=Home}`	`/Contact`	Matches and sets `Page` to `Contact`.
`{controller}/{action}/{id?}`	`/Products/List`	Maps to the `Products` controller and `List` action.
`{controller}/{action}/{id?}`	`/Products/Details/123`	Maps to the `Products` controller and `Details` action with `id` set to 123.
`{controller=Home}/{action=Index}/{id?}`	`/`	Maps to the `Home` controller and `Index` method. `id` is ignored.
`{controller=Home}/{action=Index}/{id?}`	`/Products`	Maps to the `Products` controller and `Index` method. `id` is ignored.

ITDLU

ASP.NET Core Routing

- Route constraints
  - Route constraints execute when a match has occurred to the incoming URL and the URL path is tokenized into route values.
  - Route constraints generally inspect the route value associated via the route template and make a true or false decision about whether the value is acceptable.
- Some route constraints use data outside the route value to consider whether the request can be routed (ex. HttpMethodRouteConstraint).
- Constraints are used in routing requests and link generation.

ITDLU

ASP.NET Core Routing

- Route constraints

constraint	Example	Example Matches	Notes
`int`	`{id:int}`	`123456789`, `-123456789`	Matches any integer
`bool`	`{active:bool}`	`true`, `FALSE`	Matches `true` or `false`. Case-insensitive
`datetime`	`{dob:datetime}`	`2016-12-31`, `2016-12-31 7:32pm`	Matches a valid `DateTime` value in the invariant culture. See preceding warning.
`decimal`	`{price:decimal}`	`49.99`, `-1,000.01`	Matches a valid `decimal` value in the invariant culture. See preceding warning.
`double`	`{weight:double}`	`1.234`, `-1,001.01e8`	Matches a valid `double` value in the invariant culture. See preceding warning.
`float`	`{weight:float}`	`1.234`, `-1,001.01e8`	Matches a valid `float` value in the invariant culture. See preceding warning.

31

ITDLU

ASP.NET Core Routing

- Route constraints

`guid`	`{id:guid}`	`CD2C1638-1638-72D5-1638-DEADBEEF1638`	Matches a valid `Guid` value
`long`	`{ticks:long}`	`123456789`, `-123456789`	Matches a valid `long` value
`minlength(value)`	`{username:minlength(4)}`	`Rick`	String must be at least 4 characters
`maxlength(value)`	`{filename:maxlength(8)}`	`MyFile`	String must be no more than 8 characters
`length(length)`	`{filename:length(12)}`	`somefile.txt`	String must be exactly 12 characters long
`length(min,max)`	`{filename:length(8,16)}`	`somefile.txt`	String must be at least 8 and no more than 16 characters long
`min(value)`	`{age:min(18)}`	`19`	Integer value must be at least 18

32

ITDLU

ASP.NET Core Routing

- Route constraints

`min(value)`	`{age:min(18)}`	`19`	Integer value must be at least 18
`max(value)`	`{age:max(120)}`	`91`	Integer value must be no more than 120
`range(min,max)`	`{age:range(18,120)}`	`91`	Integer value must be at least 18 but no more than 120
`alpha`	`{name:alpha}`	`Rick`	String must consist of one or more alphabetical characters, `a`-`z` and case-insensitive.
`regex(expression)`	`{ssn:regex(^\\d{{3}}-\\d{{2}}-\\d{{4}}$)}`	`123-45-6789`	String must match the regular expression. See tips about defining a regular expression.
`required`	`{name:required}`	`Rick`	Used to enforce that a non-parameter value is present during URL generation

33

ITDLU

ASP.NET Core Routing

- URL Generation
  - Is the process by which routing can create a URL path based on a set of route values.
  - Allows for a logical separation between endpoints and the URLs that access them.
- Endpoint routing includes the LinkGenerator API (a singleton service available from DI).
- Generating a URI occurs in two steps:
  - An address is bound to a list of endpoints that match the address.
  - Each endpoint's RoutePattern is evaluated until a route pattern that matches the supplied values is found. The resulting output is combined with the other URI parts supplied to the link generator and returned.

34

ITDLU

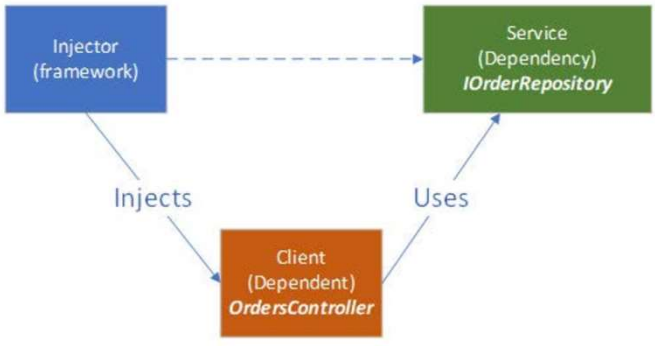
Dependency Injection

35

ITDLU

What is dependency injection?

- Dependency injection is a technique for achieving lose coupling between objects and their dependencies.
- Rather than directly instantiating dependencies, the objects needed for a class are provided to them.
- Most often class will declare their dependencies via their constructor.



36

ITDLU

Why is it useful?

- Dependency injection makes it easier to change the behavior of an application by changing the components that implement the interfaces that define application features.
- It also results in components that are easier to isolate for unit testing.
- DI is one of the most well-known methods to help achieve better maintainable code.
- DI makes it easy to extend our application as we can not only introduce new components more easily but we can also implement better versions of existing components and inject them into our applications easily.

37

ITDLU

IoC Container (DI Container)

- IoC container is a framework for implementing automatic DI.
- A container is a factory for providing instances of types that are requested from it.
- It is responsible for creating or referencing a dependency and injecting it into Client.
- .NET Core has a built-in container that is represented by the IServiceProvider interface.

38

ITDLU

Types of DI

- DI is categorized into three types:
  - Constructor injection: Dependencies are injected through a constructor while instantiating the dependent.
  - Setter injection: The dependent exposes a setter method or property that the injector uses to inject the dependency.
  - Method injection: Passing the dependency as a method parameter

```
public class UserController : ControllerBase
{
    [HttpPost("register")]
    public IActionResult RegisterUser(
        [FromServices] IMessageSender messageSender,
        string username)
    {
        messageSender.SendMessage(username);
        return Ok();
    }

    [HttpPost("promote")]
    public IActionResult PromoteUser(
        [FromServices] IPromotionService promoService,
        string username, int level)
    {
        promoService.PromoteUser(username, level);
        return Ok();
    }
}
```


ITDLU

Service Lifetimes

- Transient: Services are created every time they are requested.
- Scoped: Services are created once per request.
- Singleton: Services are created the first time they are requested. Each subsequent request uses the same instance.

```
public static IServiceCollection AddEmailSender(
    this IServiceCollection services)
{
    services.AddScoped<IEmailSender, EmailSender>();
    services.AddSingleton<NetworkClient>();
    services.AddScoped<MessageFactory>();
    services.AddSingleton(
        new EmailServerSettings
        (
            host: "smtp.server.com",
            port: 25
        ));
    return services;
}
```

By convention, return the IServiceCollection to allow method chaining.

Third-party containers

- Unity: Unity was initially built by Microsoft and is currently open sourced.
  - http://unitycontainer.org/
- Autofac: This is one of the most popular DI containers.
  - https://autofac.readthedocs.io/en/latest/.
- Simple Injector: This is one of the late entrants on the list.
  - https://docs.simpleinjector.org/en/latest/index.html
- Castle Windsor: This is one of the oldest DI frameworks available for .NET.
  - http://www.castleproject.org/projects/windsor/
- Lamar: Fast inversion of control tool and successor to StructureMap
  - https://jasperfx.github.io/lamar/guide/

41

Logging

42

ITDLU

What is logging?

- Logging helps you to record your application's behavior for different data at runtime
- Logging is to save messages for later reference, in order to help you figure out what your program is doing wrong, underperforming, …
- With the log information, we can:
  - Reproduce the problem
  - Debug production issues
  - Build very useful insights.
- .NET supports a logging API that works with a variety of built-in and third-party logging providers.

43

ITDLU

Logging Providers

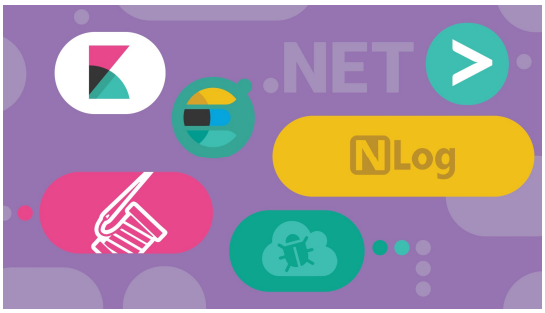
- A logging provider takes some action on logged data, such as display it on the console or store it in a file or Azure blob storage.
- Built-in logging providers:
  - Console: Logs output to the console
  - Debug: Write log output using the Debug class through the Debug.WriteLine method
  - EventSource: Writes to a cross-platform event source with the name Microsoft-Extensions-Logging.
  - Windows EventLog: Sends log output to the Windows Event Log.
- Other Microsoft logging providers:
  - Azure App Service: Writes logs to text files in an Azure App Service app's file system and to blob storage in an Azure Storage account.
  - ApplicationInsights: Writes logs to Azure Application Insights.

44

ITDLU

Logging Providers

- Third-party logging providers:
  - Elmah.io
  - Gelf
  - JSNLog
  - KissLog
  - Log4Net
  - Nlog
  - Nreco.Logging
  - Sentry
  - Serilog
  - Stackdriver

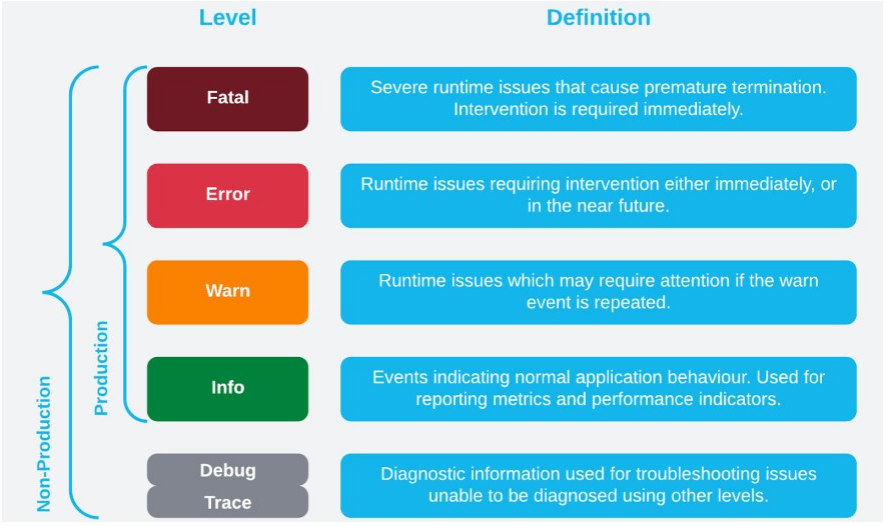


45

ITDLU

Log Levels

- Specify how important it is to see the message
- Generally, log levels should be specified in configuration and not code.



46

ITDLU

Create logs

To create logs, use an ILogger<TCategoryName> object from dependency injection (DI).

Call method Log*** to log the message

Log message template

```
public class AboutModel : PageModel
{
    private readonly ILogger _logger;

    public AboutModel(ILogger<AboutModel> logger)
    {
        _logger = logger;
    }

    public void OnGet()
    {
        _logger.LogInformation("About page visited at {DT}",
            DateTime.UtcNow.ToLongTimeString());
    }
}
```


ITDLU

Configure Logging

- Configure logging provider

```
var builder = WebApplication.CreateBuilder(args);
builder.Logging.ClearProviders();
builder.Logging.AddConsole();
```

```
var builder = WebApplication.CreateBuilder();
builder.Host.ConfigureLogging(logging =>
{
    logging.ClearProviders();
    logging.AddConsole();
});
```

- Configure minimum level
  - provided by the Logging section of appsettings.{ENVIRONMENT}.json files

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  }
}
```

ITDLU

Use Nlog in ASP.NET Core

- Install NuGet packages
  - NLog (v5+)
  - NLog.Web.AspNetCore (v5+)
- Create a Nlog.config file (download here)
- Setup NLog for Dependency injection
- Create and write logs

```
// Add services to the container.
builder.Services.AddControllersWithViews();

// NLog: Setup NLog for Dependency injection
builder.Logging.ClearProviders();
builder.Host.UseNLog();

var app = builder.Build();
```

```
public class HomeController : Controller
{
    private readonly ILogger<HomeController> _logger;

    public HomeController(ILogger<HomeController> logger)
    {
        _logger = logger;
        _logger.LogDebug(1, "NLog injected into HomeController");
    }

    public IActionResult Index()
    {
        _logger.LogInformation("Hello, this is the index!");
        return View();
    }
}
```

49

ITDLU

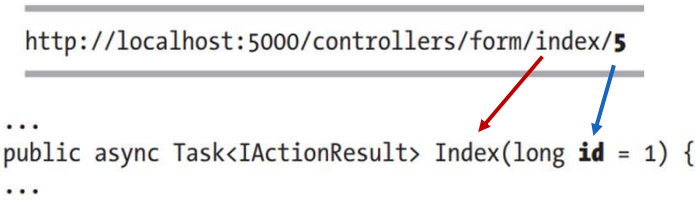
Model Binding & Validation

50

ITDLU

What is Model binding?

- Model binding is the process of creating .NET objects using the values from the HTTP request to provide easy access to the data required by action methods and Razor Pages.
- Model binding lets controllers or page handlers declare method parameters or properties using C# types and automatically receive data from the request without having to inspect, parse, and process the data directly.



ITDLU

Model Binders

- The model binding system relies on model binders, which are components responsible for providing data values from one part of the request or application.
- The default model binders look for data values in these places:
  - Form data
  - The request body
  - Routing segment variables
  - Query strings
  - Upload files
  - HTTP headers

- [FromQuery] - Gets values from the query string.
- [FromRoute] - Gets values from route data.
- [FromForm] - Gets values from posted form fields.
- [FromBody] - Gets values from the request body.
- [FromHeader] - Gets values from HTTP headers.

ITDLU

Targets

- Model binding tries to find values for the following kinds of targets:
  - Parameters of the controller action method that a request is routed to.
  - Parameters of the Razor Pages handler method that a request is routed to.
  - Public properties of a controller or PageModel class, if specified by attributes.

```
public class EditModel : PageModel
{
    [BindProperty]
    public Instructor? Instructor { get; set; }

    // ...
}
```

```
[BindProperty(Name = "ai_user", SupportsGet = true)]
public string? ApplicationInsightsCookie { get; set; }
```

```
public ActionResult<Pet> Create([FromBody] Pet pet)
```

```
public class Instructor
{
    public int Id { get; set; }

    [FromQuery(Name = "Note")]
    public string? NoteFromQueryString { get; set; }

    // ...
}
```

```
public void OnGet([FromHeader(Name = "Accept-Language")] string language)
```


ITDLU

Model Validation

- Model validation is the process of ensuring the data received by the application is suitable for binding to the model and, when this is not the case, providing useful information to the user that will help explain the problem.
- Validation can be performed automatically during the model binding process and can be supplemented with custom validation.
- The validation process consists of 2 steps
  - Checking the data received
  - Helping the user correct the problem

ITDLU

Validating Data

ModelStateDictionary object tracks details of the state of the model object

```
[HttpPost]
public IActionResult SubmitForm(Product product) {
    if (ModelState.IsValid) {
        TempData["name"] = product.Name;
        TempData["price"] = product.Price.ToString();
        TempData["categoryId"] = product.CategoryId.ToString();
        TempData["supplierId"] = product.SupplierId.ToString();
        return RedirectToAction(nameof(Results));
    } else {
        return View("Form");
    }
}
```


ITDLU

Displaying Validation Messages

```
<form asp-action="Login">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

    <div class="form-group">
        <label asp-for="Username" class="control-label"></label>
        <input asp-for="Username" class="form-control" />
        <span asp-validation-for="Username" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Password" class="control-label"></label>
        <input asp-for="Password" class="form-control" />
        <span asp-validation-for="Password" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
```

Adds messages that describe any validation errors to the view

Displays property-level validation messages

ITDLU

Specifying Validation Rules Using Metadata

- Built-in validation attributes:
  - Compare
  - Range
  - RegularExpression
  - Required
  - StringLength
  - MaxLength
  - MinLength
  - EmailAddress
  - Phone
  - Url
  - Remote

```
5 references | 0 changes | 0 authors, 0 changes
public class SignInModel
{
    [BindProperty(Name = "name")]
    [Required(ErrorMessage = "Please enter username")]
    4 references | 0 changes | 0 authors, 0 changes
    public string Username { get; set; }

    [Required(ErrorMessage = "Password is required")]
    [MinLength(6, ErrorMessage = "Password is too short")]
    4 references | 0 changes | 0 authors, 0 changes
    public string Password { get; set; }
}
```


ITDLU

Client-Side Validation

- Client-side validation prevents submission until the form is valid.
- Client-side validation avoids an unnecessary round trip to the server when there are input errors on a form.
- ASP.NET Core supports *unobtrusive client-side validation*.
  - The term *unobtrusive* means that validation rules are expressed using attributes added to the HTML elements that views generate.
- It is done by using Microsoft front-end library jQuery Unobtrusive Validation.

ITDLU

Using FluentValidation

- FluentValidation is a .NET library for building strongly-typed validation rules.
- Steps:
  - Install required packages:
    - FluentValidation.AspNetCore
    - FluentValidation.DependencyInjectionExtensions
  - Configure FluentValidation
  - Add validation to models
  - Validate models in the actions

```
public class PersonValidator : AbstractValidator<Person>
{
  public PersonValidator()
  {
    RuleFor(x => x.Id).NotNull();
    RuleFor(x => x.Name).Length(0, 10);
    RuleFor(x => x.Email).EmailAddress();
    RuleFor(x => x.Age).InclusiveBetween(18, 60);
  }
}
```

- Learn more: https://docs.fluentvalidation.net/en/latest/

ITDLU

Caching

ITDLU

61

What is caching?

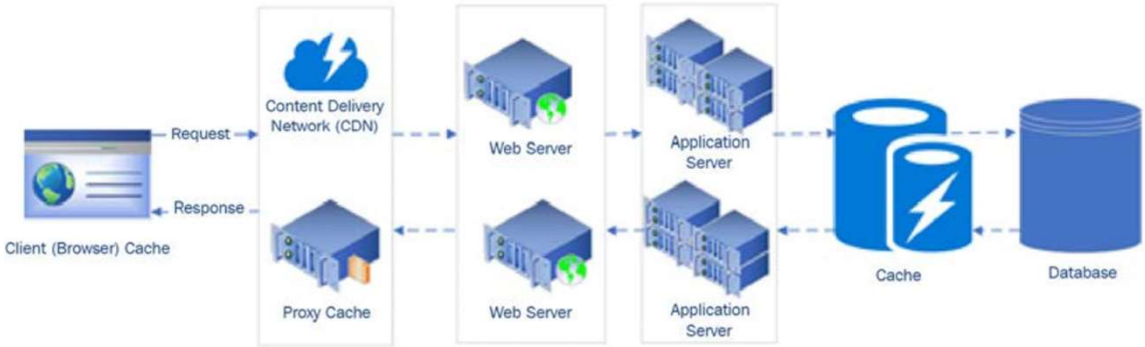
- Caching data is a technique to save frequently used data into temporary location such as RAM or disk so future requests for that data can be served faster.
- The data stored in the cache is the result of an earlier computation
- Types of caching:
  - Output caching: Full page caching, Partial (fragment) caching
  - Data caching (Application caching)

61

ITDLU

62

Cache layers in a request flow



62

ITDLU

Benefits & Costs

- Benefits of caching
  - Improve performance
    - Speed: reduce response time and page load time
    - Efficiency: reduce infrastructure usage (CPU time, DB utilization, Network bandwidth)
  - Reduce a lot of server resources at peak times
- Costs of caching
  - Staleness (out-of-date)
  - Need to check and refresh data

ITDLU

Components of Caching

- Response caching
  - A caching technique supported by HTTP to cache the response to a request made using HTTP or HTTPS either on the client (for example, a browser) or an intermediate proxy server.
  - Is controlled by setting the appropriate value for the Cache-Control header in both requests and responses.

```
Cache-Control:public,max-age=10
```

The response can be cached anywhere – client/server/intermediate proxy server. Other values: Private, No-cache

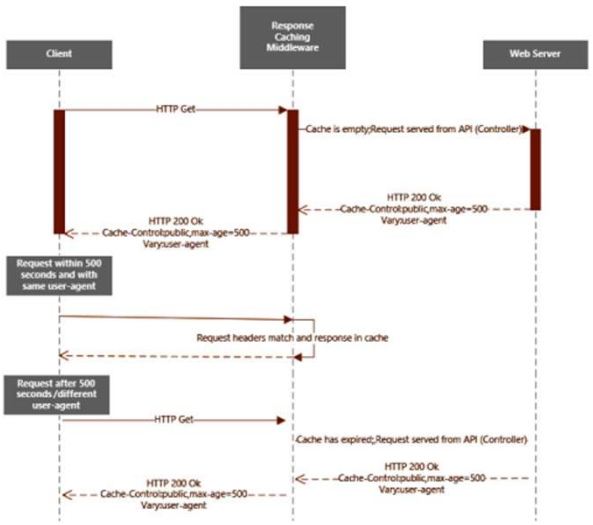
The client can cache the response for 10 seconds

ITDLU

Response caching

- Other headers:
  - Age: Indicates the duration for which an object is present in the cache (proxy/browser)
  - Vary: Tells the client the request header on which basis the responses are cached.
- Example:

  Vary: user-agent



ITDLU

Response caching

- Configure response caching
  - Add required services and middleware
    - builder.Services.AddResponseCaching();
    - app.UseResponseCaching(); ← Call after UseCors, UseRouting, before UseEndpoints
  - Handle the response to set cache headers

```
[HttpGet]
[ResponseCache(Duration = 500, VaryByHeader =
  "user-agent", Location =
  ResponseCacheLocation.Any, VaryByQueryKeys =
  new[] { "Id" })]
public async Task<IActionResult>
  Get([FromQuery]int Id = 0)
```

ITDLU 67

Response caching

- ResponseCache key properties
  - Duration: A numeric value that sets the max-age value in the response header
  - ResponseCacheLocation: An enum that takes three values – Any, Client, and None
  - VaryByHeader: A string that controls cache behavior to cache based on a specific header
  - VaryByQueryKeys: An array of strings that accepts key values on which basis data is cached

67

ITDLU 68

Response caching

- Cache profiles: apply cache policy at the controller/method level

```
builder.Services.AddControllers(options =>
{
    options.CacheProfiles.Add("Default",
      new CacheProfile {
        Duration = 500,
        VaryByHeader = "user-agent",
        Location = ResponseCacheLocation.Any,
        VaryByQueryKeys = new[] { "Id" } });
});
```

```
[ResponseCache(CacheProfileName = "Default")]
```

68

ITDLU

69

Components of Caching

- Distributed caching
  - is an extension of traditional caching in which cached data is stored in more than one server in a network.
  - is a cache strategy in which data is stored in multiple servers/nodes/shards outside the application server.
- **CAP** theorem:
  - **C** stands for consistency, meaning the data is consistent across all the nodes and each node has the same copy of the data.
  - **A** stands for availability, meaning the system is available, and the failure of one node doesn't cause the system to go down.
  - **P** stands for partition-tolerant, meaning the system doesn't go down even if the communication between nodes goes down.

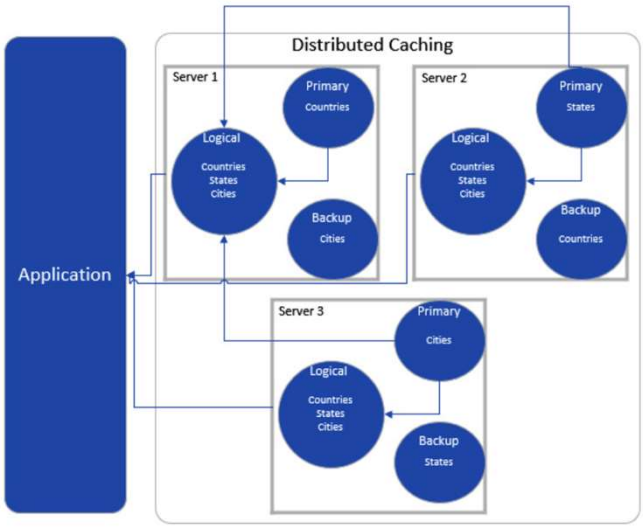
69

ITDLU

70

Distributed caching

- .NET distributed caching providers:
  - Redis cache
  - Memcached
  - Couchbase
  - SQL Server
  - NCache



70

ITDLU

Configure In-Memery Cache

- Add required services

```
builder.Services.AddMemoryCache()
```

- Create an instance of MemoryCache using constructor injection
- Use methods Get/Set/Remove exposed by IMemoryCache interface
- Configure cache options using the class MemoryCacheEntryOptions

```
DateTime? cacheEntry;
if (!cache.TryGetValue("Weather",
  out cacheEntry))
{
    cacheEntry = DateTime.Now;
    var cacheEntryOptions = new
      MemoryCacheEntryOptions()
        .SetSlidingExpiration(
         TimeSpan.FromSeconds(50))
        .SetAbsoluteExpiration(
         TimeSpan.FromSeconds(100))
        .SetPriority(
         CacheItemPriority.NeverRemove);
    cache.Set("Weather", cacheEntry,
      cacheEntryOptions);
}
cache.TryGetValue("Weather", out cacheEntry);
```


ITDLU

Learn more …

- https://learn.microsoft.com/en-us/aspnet/core/performance/caching/overview?view=aspnetcore-7.0
- https://www.c-sharpcorner.com/article/caching-mechanism-in-asp-net-core/
- https://code-maze.com/aspnetcore-in-memory-caching/
- https://learn.microsoft.com/en-us/aspnet/core/performance/caching/distributed?view=aspnetcore-7.0
- https://aspnetcore.readthedocs.io/en/stable/performance/caching/distributed.html

END