Advanced Topics

Web Host, Logging, Cache, Sessions, TempData, Areas, Performance, SEO, GDPR

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

Have a Question?

Software University

sli.do

#csharp-web

WebHost

...and WebApplication

WebHost

- ASP.NET Core apps configure and launch a host
  - The host is responsible for **app startup** and **lifetime management**
  - At minimum, the host configures a **server** and **request pipeline**
    - Can also set up logging, dependency injection and configuration

WebApplication

- Before .NET 6, the webhost is set up first and then the app is built
  - In .NET 6 we do those actions **simultaneously** in **`Program.cs`**
  - **`WebApplication`** is an abstraction of **`WebHost`**
    - Returned by the **`Build()`** method of the **`WebApplicationBuilder`**
    - Defines the way the app communicates with its environment

CreateBuilder()

- `CreateBuilder()` initializes a new instance of the **`WebApplicationBuilder`** class
  - Performs several essential tasks
    - Configures Kestrel server, loads host and app configuration
    - Configures logging, IIS integration, sets the content root, etc.
- This sets up **default** config which you can **modify**:

```
var builder = WebApplication.CreateBuilder(args);
```

```
builder.Host.ConfigureLogging(logging =>
{
    logging.SetMinimumLevel(LogLevel.Warning);
});
```

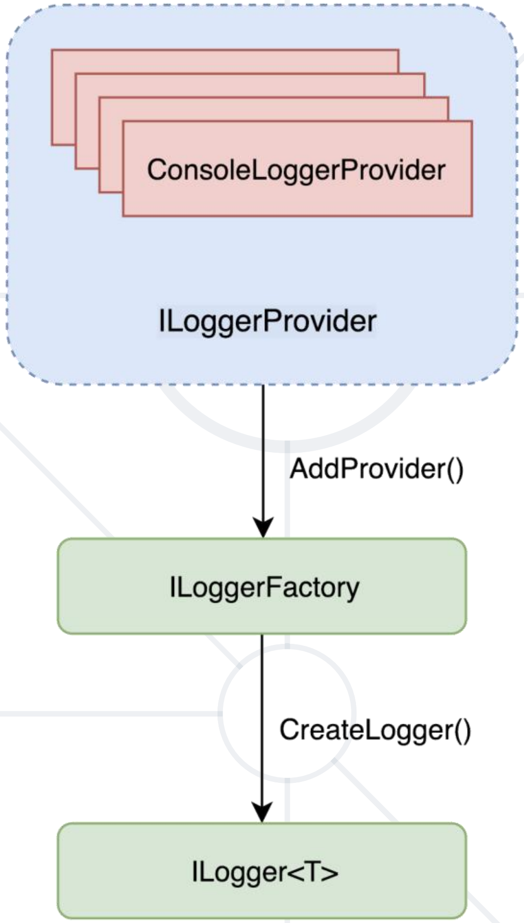
```
builder.Host.ConfigureServices((context, services)
=>
{
    services.Configure<KestrelServerOptions>(

context.Configuration.GetSection("Kestrel"));
});
```

Logging

ILoggerFactory & ILogger

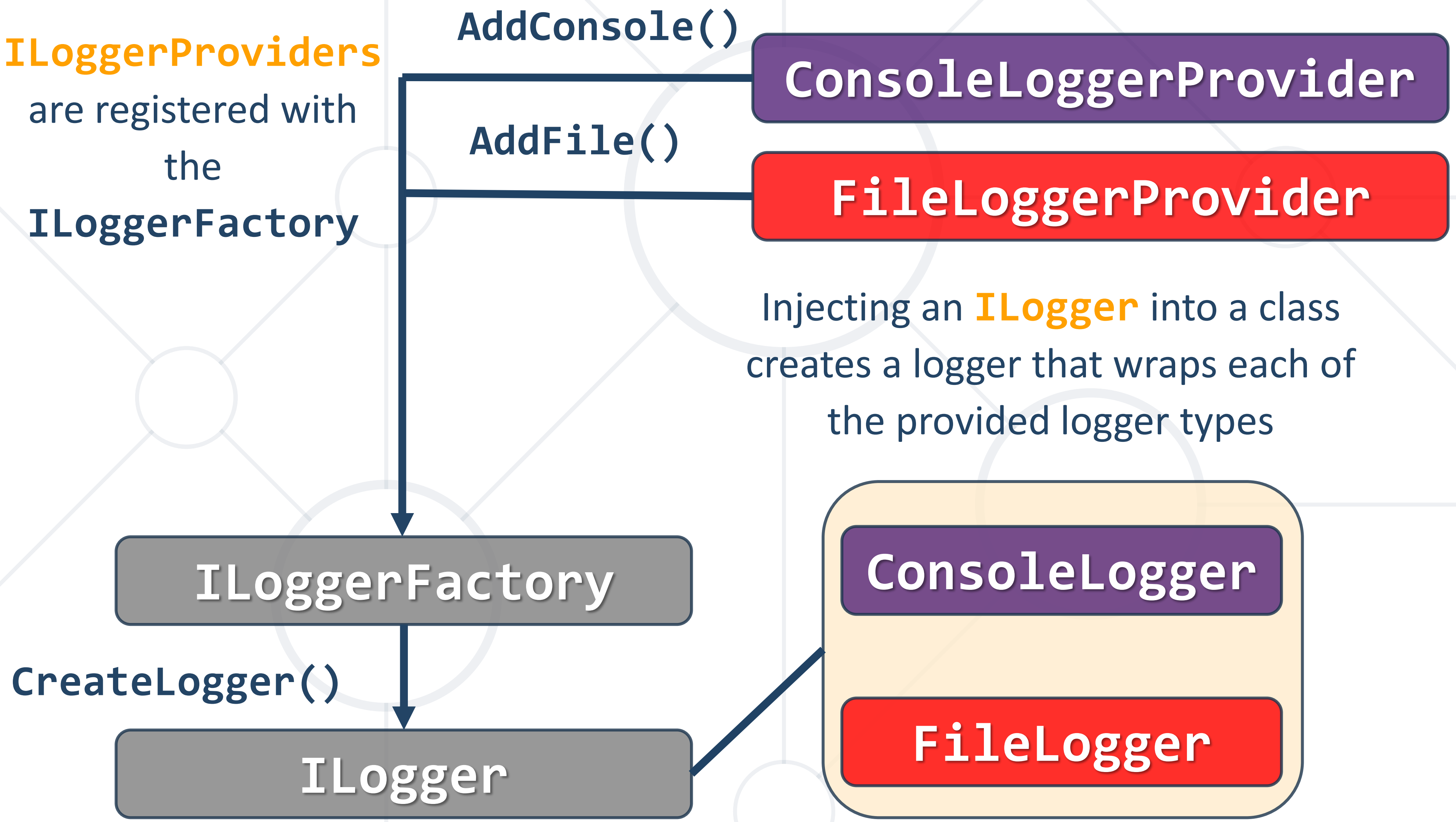
Logging

- ASP.NET Core supports a logging API
  - It works with a variety of logging providers
- The ASP.NET Core logging infrastructure consists of 3 main components:
  - `ILogger` – used by the app to create log messages
  - `ILoggerFactory` – creates instances of `ILogger`
  - `ILoggerProvider` – controls where log messages are output
- An application may have multiple logging providers



ILogger, ILoggerFactory and ILoggerProvider

ILoggerProviders are registered with the **ILoggerFactory**



Ilogger Providers are used to create loggers that output to a specific destination

Injecting an **ILogger** into a class creates a logger that wraps each of the provided logger types

Logging Configuration

- Logging **configuration** is commonly provided by the **app settings**
  - **`Logging`** property can have **`LogLevel`**
  - **`LogLevel`** specified the minimum level to log
  - Other properties under **`Logging`** can specify **logging providers**
- Sample Logs

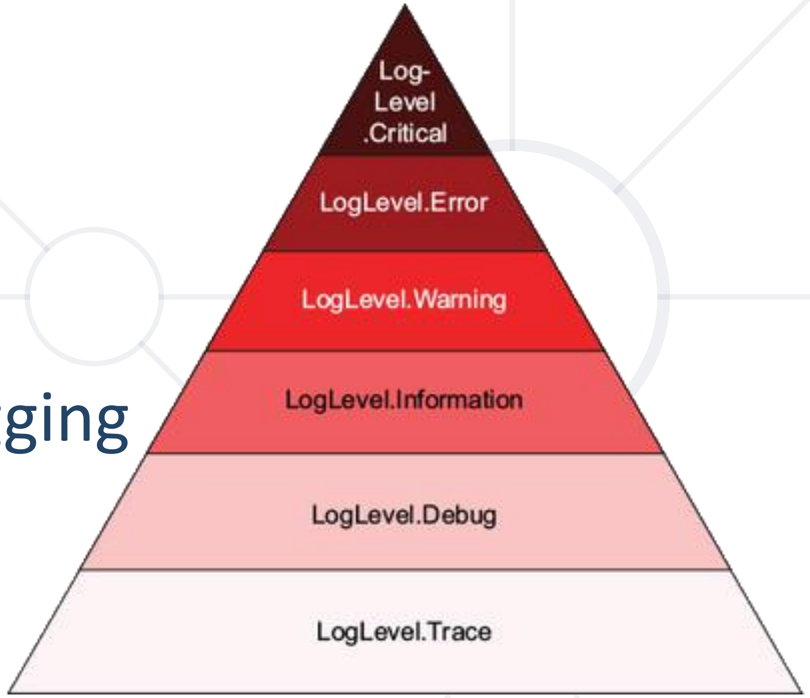
appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  ...
}
```

```
info: TodoApi.Controllers.TodoController[1002]
      Getting item 0
warn: TodoApi.Controllers.TodoController[4000]
      GetById(0) NOT FOUND
```

Logging Levels

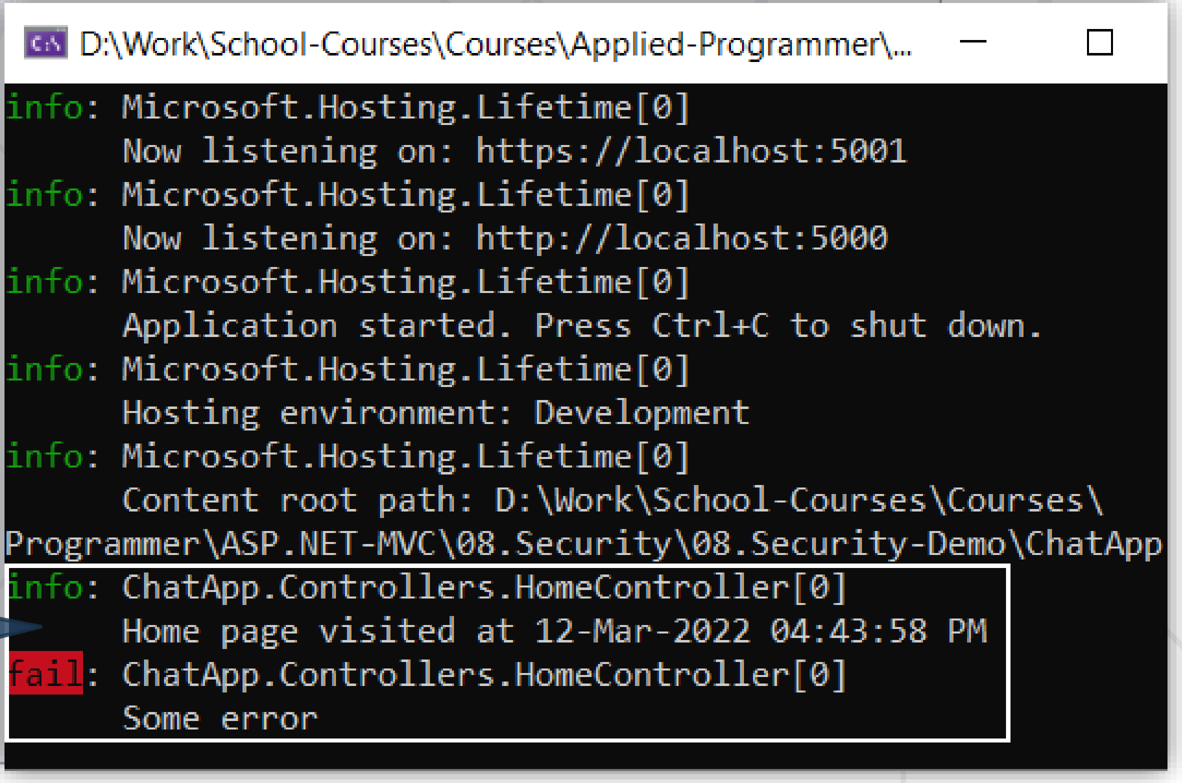
- **Logging Levels** are **not** technology-specific
  - It is important to know the levels and their use
- **Logging Levels** and their description:
  - **`Trace`** – for information, valuable only for debugging
  - **`Debug`** – for information, useful in development and debugging
  - **`Information`** – for tracking the general flow of the app
  - **`Warning`** – for abnormal and unexpected events in the app flow
  - **`Error`** – for errors and exceptions that cannot be handled
  - **`Critical`** – for failures that require immediate attention



How to Log Messages from Your Code?

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

   public HomeController(ILogger<HomeController> logger)
      => this.logger = logger;

   public IActionResult Index()
   {
      var message = $"Home page visited at
      this.logger.LogInformation(message);

      var error = "Some error";
      this.logger.LogError(error);

      return View();
   }
}
```

Inject ILogger through the constructor



```
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: https://localhost:5001
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://localhost:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\Work\School-Courses\Courses\
Programmer\ASP.NET-MVC\08.Security\08.Security-Demo\ChatApp
info: ChatApp.Controllers.HomeController[0]
      Home page visited at 12-Mar-2022 04:43:58 PM
fail: ChatApp.Controllers.HomeController[0]
      Some error
```

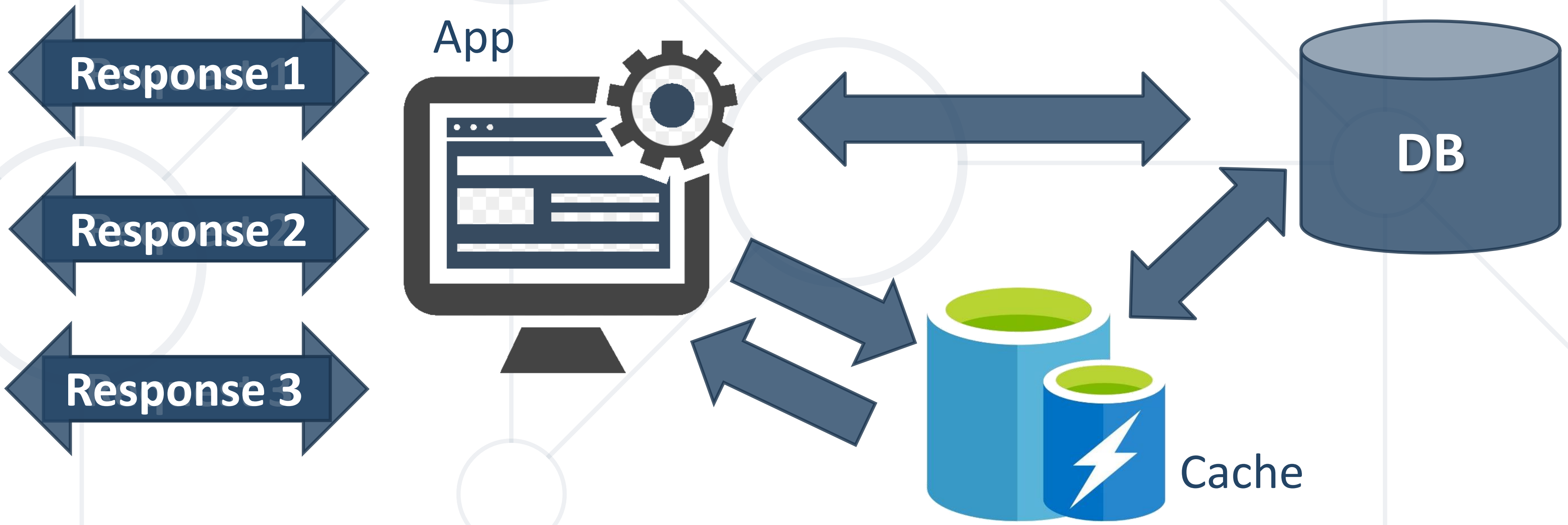
Messages are displayed on the console

Cache

An Efficient Way to Store Data

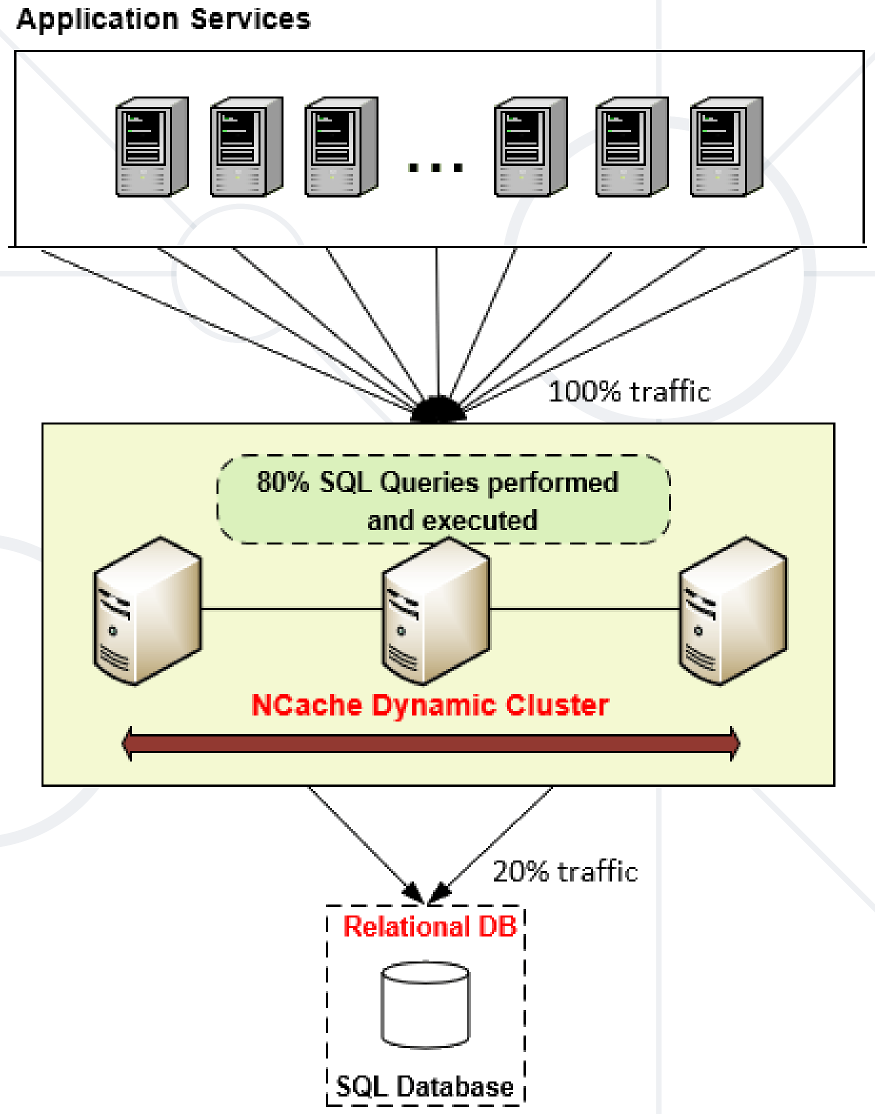
Cache

- Cache makes a copy of a piece of data and stores it
  - Can be extracted much faster than from its original source
  - Significantly improves application performance
  - Works best with data that does **not change frequently**



Cache Types

- **ASP.NET Core** supports several different caches
  - The simplest cache is based on the **`IMemoryCache`**
    - An **in-memory** cache stored on the app server's memory
    - Can store any type – **primitive** or **complex** (object)
  - **`IDistrubutedCache`** – cache shared by multiple app servers

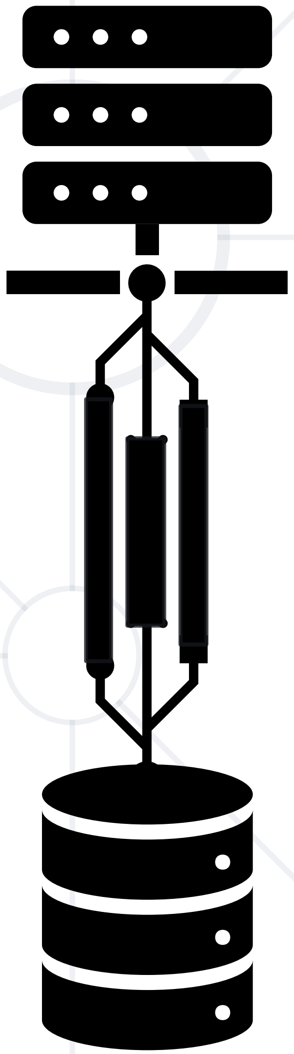


In-Memory Cache

- In-memory Cache is configured as a simple service

```
// Add the IMemoryCache as a dependency to the DI
builder.Services.AddMemoryCache();
```

```
public class HomeController : Controller
{
    private IMemoryCache cache;

    public HomeController(IMemoryCache memoryCache)
    {
        // Inject the IMemoryCache through DI
        this.cache = memoryCache;
    }
    ...
}
```



In-Memory Cache – Example

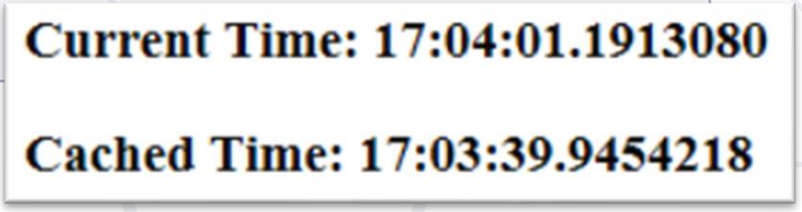
- Here is an example of a cache `DateTime` value

```
public IActionResult GetCachedData()
{
    DateTime cacheEntry;

    if (!this.cache.TryGetValue(CacheKeys.Entry, out cacheEntry)) // Look for cache key.
    {
        cacheEntry = DateTime.Now; // Key not in cache, so get data.

        var cacheEntryOptions = new MemoryCacheEntryOptions() // Set cache options.
                .SetSlidingExpiration(TimeSpan.FromSeconds(3)); // Keep in cache for this time.
                // Reset time if accessed.

        // Save data in cache.
        this.cache.Set(CacheKeys.Entry, cacheEntry, cacheEntryOptions);
    }

    return View("Cache", cacheEntry);
}
```

In-Memory Cache – Example (2)

- The cached `DateTime` value **remains in the cache**
  - Its value is untouched, from the moment of caching

```
<h3>Current Time: @DateTime.Now.TimeOfDay.ToString()</h3>
<h3>Cached Time: @(Model == null
             ? "No cached entry found"
             : Model.Value.TimeOfDay.ToString())
</h3>
```



Current Time: 17:04:01.1913080

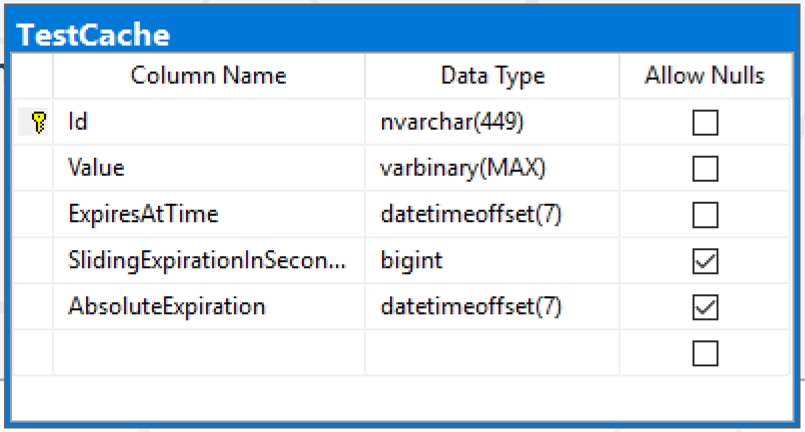
Cached Time: 17:03:39.9454218

- There are requests within the **timeout period**
  - No eviction is done due to **memory pressure**

Distributed Cache

- We can persist cache data in a SQL server database

```
builder.Services.AddDistributedSqlServerCache(
        options =>
        {
            options.ConnectionString = Configuration.GetConn
            options.SchemaName = "dbo";
            options.TableName = "TestCache";
        });
    // services.AddDistributedRedisCache()
builder.Services.AddSession();
```


TestCache

	Column Name	Data Type	Allow Nulls
🔑	Id	nvarchar(449)	☐
	Value	varbinary(MAX)	☐
	ExpiresAtTime	datetimeoffset(7)	☐
	SlidingExpirationInSecon...	bigint	☑
	AbsoluteExpiration	datetimeoffset(7)	☑
			☐

- The **cache table** is created using the **sql-cache** command

```
dotnet sql-cache create "Data Source=(localdb)\\mssqllocaldb;Initial Catalog=DistCache;Integrated Security=True;" dbo TestCache
```

Cache Tag Helpers

- The framework also supplies you with a convenient **Tag Helper**
  - The **Cache Tag helper** caches content to the internal cache provider

```
<cache>
    Current Time: @DateTime.Now
</cache>
```

```
<cache expires-on="new DateTime(2025,1,29,17,02,0)">
    Current Time: @DateTime.Now
</cache>
```

```
<cache enabled="true">
    Current Time: @DateTime.Now
</cache>
```

```
<cache expires-after="TimeSpan.FromSeconds(120)">
    Current Time: @DateTime.Now
</cache>
```

```
<cache expires-sliding="TimeSpan.FromSeconds(60)">
    Current Time Inside Cache Tag Helper: @DateTime.Now
</cache>
```

HTTP Response Cache (1)

- There are other types of Cache, **like HTTP-based Response Caching**
  - The primary **HTTP header** for caching is **Cache-Control**
  - It is used to specify caching **directives**
  - These directives control caching behavior during communication
- **Response Caching** in **ASP.NET Core** is controlled by a simple **middleware**

```
builder.Services.AddResponseCaching();
```

```
app.UseResponseCaching();
```

HTTP Response Cache (2)

- Once enabled, you can configure it:
  - Manually in **Request Handlers**
  - With attributes on **Controller Actions**
- The convenient built-in **ResponseCache** attribute is quite useful

```
[ResponseCache(Duration = 30, Location = ResponseLocation.None, NoStore = true)]
public IActionResult Error()
{
    ...
}
```

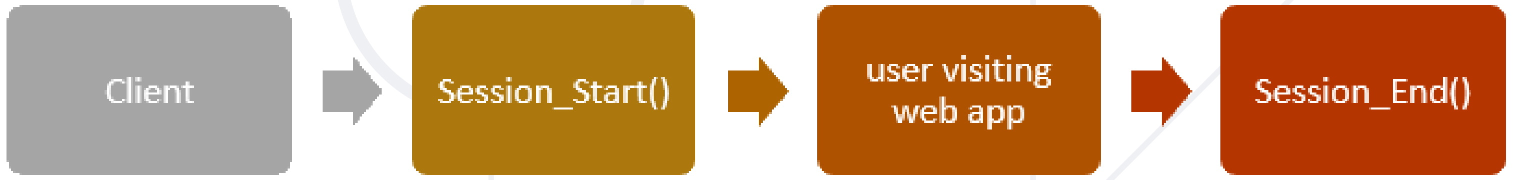
  - The attribute's properties are used to configure the **Caching**

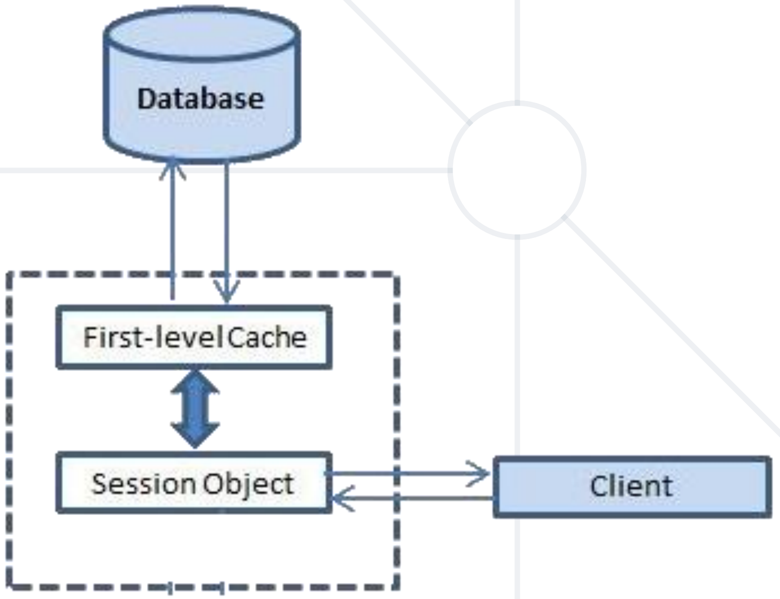
Sessions

Application state

Sessions

- Session state is an ASP.NET Core scenario for storage of client data
  - Each client has a unique Session ID which the framework stores
  - Data can be maintained while the client browsers the application
- Session data is backed by cache, and is considered ephemeral
  - The application should continue to function without session data





Configure Session

- Enabling **Session middleware**, setting up in-memory **session provider**

```
// services.AddMemoryCache(); // Default in-memory cache – provides IMemoryCache
// Provides IDistributedCache
builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options =>
{
     // Set a short timeout for easy testing
     options.IdleTimeout = TimeSpan.FromSeconds(10);

     // XSS security
     options.Cookie.HttpOnly = true;
});

builder.Services.AddControllersWithViews();

// UseSession() Middleware must be called before UseMvc()
app.UseSession();
```

Set and Use Session

- After the **Session state** is **configured**, **HttpContext.Session** is available
- **ASP.NET Core Sessions** store **byte array** values to ensure **serialization**
  - You may need specific approaches in order to store **complex data**

```
public IActionResult GetShoppingCart()
{
    if (this.HttpContext.Session.Get("Cart") == null)
    {
        this.HttpContext.Session.SetString("Cart", JsonConvert.SerializeObject(new Cart()));
    }

    this.ViewData["Cart"] = this.HttpContext.Session.GetString("Cart") == null
                ? null
                : JsonConvert.DeserializeObject(this.HttpContext.Session.GetString("Cart"));

    return View();
}
```

Extend Session

- You can implement **Session Extension methods** to ease your work

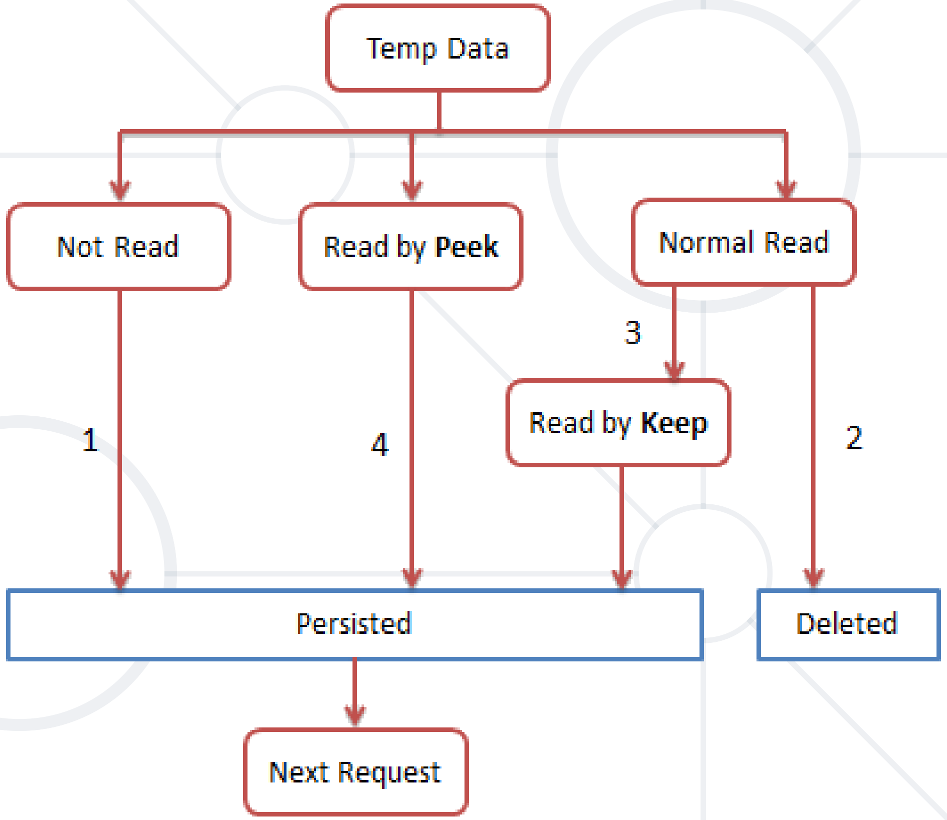
```
public static class SessionExtensions
{
    public static void Set<T>(this ISession session, string key, T value)
    {
        session.SetString(key, JsonConvert.SerializeObject(value));
    }

    public static T Get<T>(this ISession session, string key)
    {
        var value = session.GetString(key);

        return value == null
                ? default(T)
                : JsonConvert.DeserializeObject<T>(value);
    }
}
```

Temp Data

Store data until it’s read

TempData

- **ASP.NET Core** exposes the **TempData** property in:
  - **Razor Page** page models
  - **MVC Controllers**
- The property stores data until it's read
  - **Keep()** and **Peek()** methods avoid deletion when data is examined
- **TempData** is:
  - Particularly used for **redirection**
  - When data is required for **more than** a **single** request



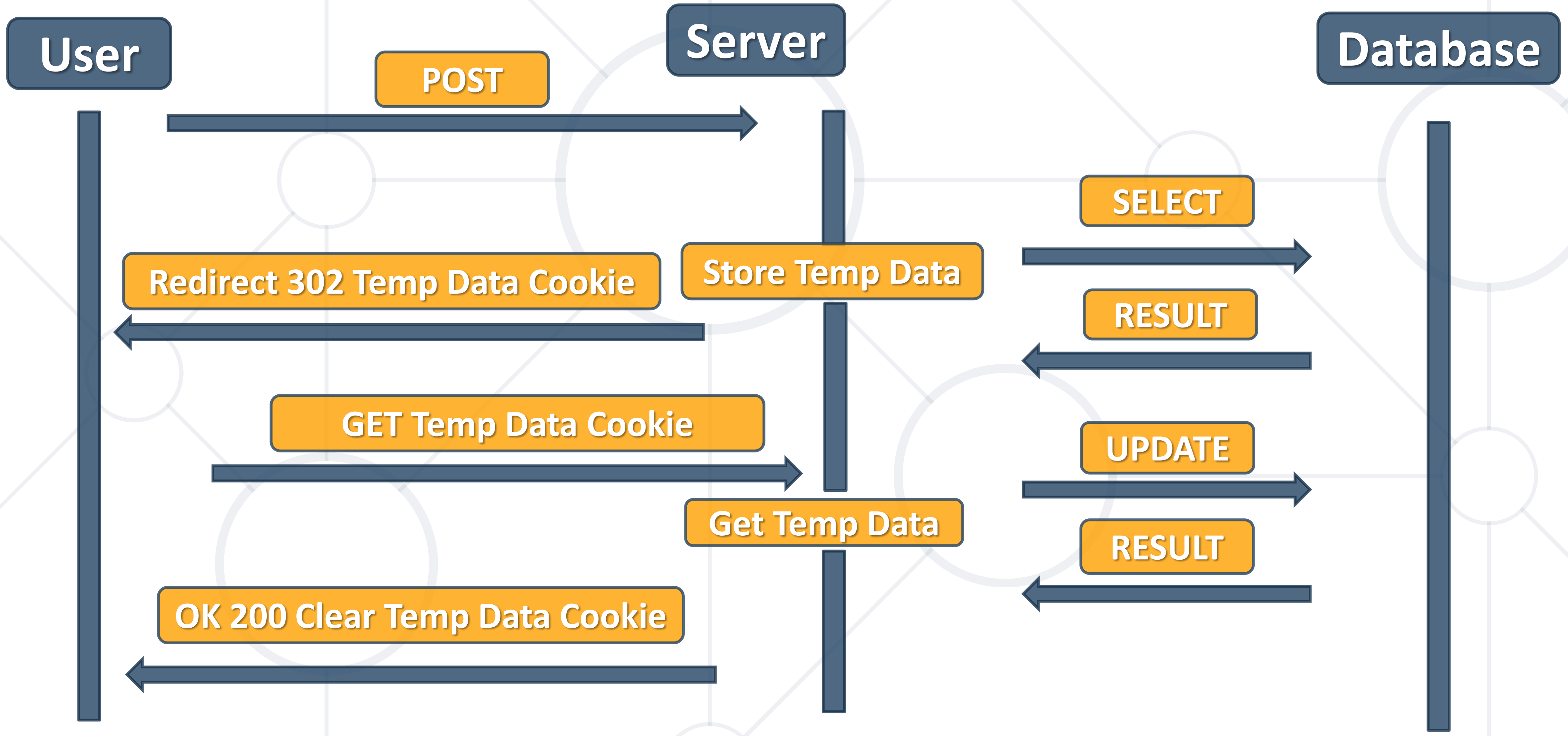
TempData

- TempData is implemented by TempData providers
  - Using either cookies or session state
  - Since ASP.NET Core 2.0, the default TempData provider is cookie-based

```
builder.Services.AddControllersWithViews()
        .AddSessionStateTempDataProvider();

builder.Services.AddSession(...);

. . .

app.UseSession();
```

Not needed when working with cookies

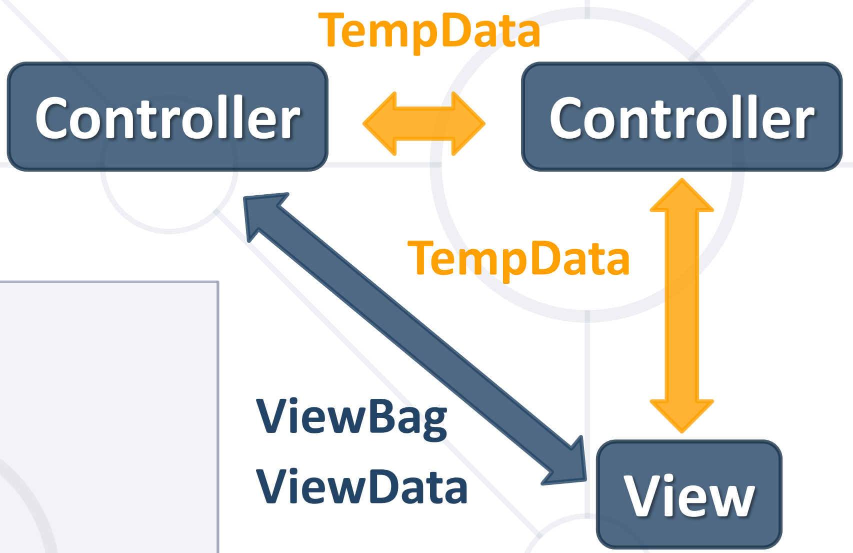
TempData with Cookies Workflow



Enable and Access TempData

- After enabling the **TempData**, you can access it in:
  - Your **Controller** and **Actions**
  - Your **Razor Page** page model

```
public IActionResult RedirectToTempData()
{
    this.TempData["Previous"] = "/Home/RedirectToTempData";
    return this.RedirectToAction("AccessTempData");
}

public IActionResult AccessTempData()
{
    this.HttpContext.Response.Headers.Add("Previous",
       this.TempData["Previous"].ToString());
    // Add a HttpHeader ("Previous") with the previous Action URL
    return this.View();
}
```



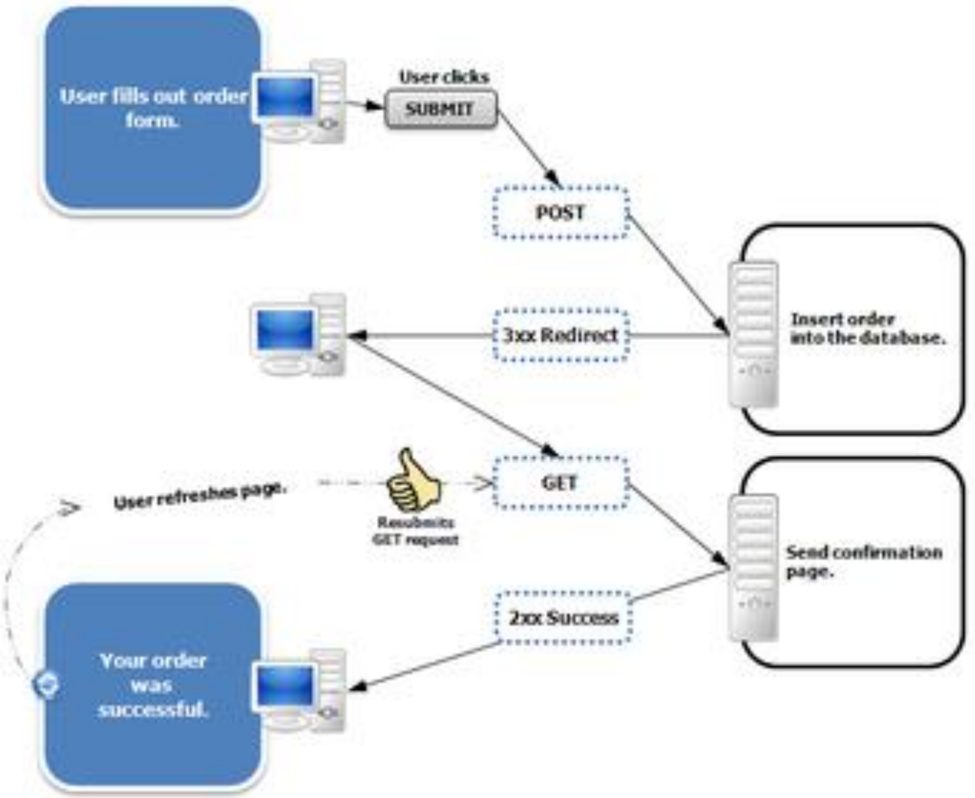
Post-redirect-Get

- **Post-redirect-Get** (**PRG**) is a design pattern in web development
  - **POST** requests should be answered with a **REDIRECT**
  - The **REDIRECT** response should trigger a **GET** request in the client
- **Post-redirect-Get** is designed to reduce **duplicate form submissions**
  - These are caused by clients **refreshing** or **navigating** back and forth
- **Post-redirect-Get** has a major role in most applications
  - Duplicate form submissions can be critical in **Store** applications
  - Duplicate form submissions may cause undesired **Data spam**

Post-redirect-Get

- **PRG** is a pattern, and easy to implement

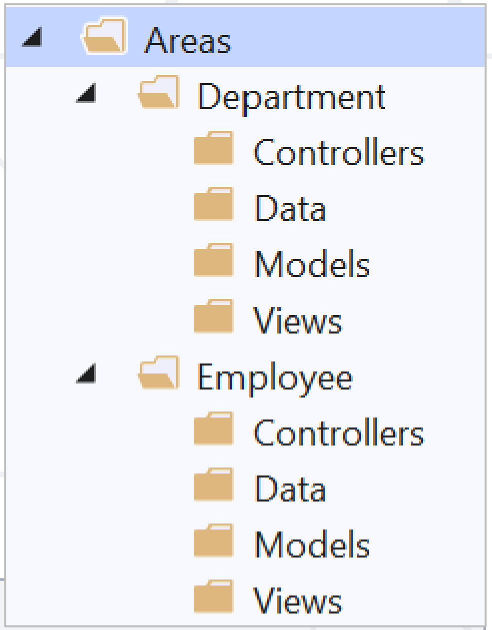
```
[HttpGet]
public IActionResult Create()
{
    return View(new ProductModel());
}

[HttpPost]
public IActionResult Create(ProductModel productModel)
{
    if (!ModelState.IsValid)
    {
        return View(productModel);
    }

    // Do magic with productModel
    return RedirectToAction("Details", { id });
}
```



Areas

Areas

- Some applications can have a **large number** of **components**
- We can partition Web applications into smaller units (**Areas**)
  - An **Area** is effectively an **MVC structure** inside an application
- Example: large e-commerce application
  - Store, users, blog, forum, administration
- To use areas you should change the **default route template**:

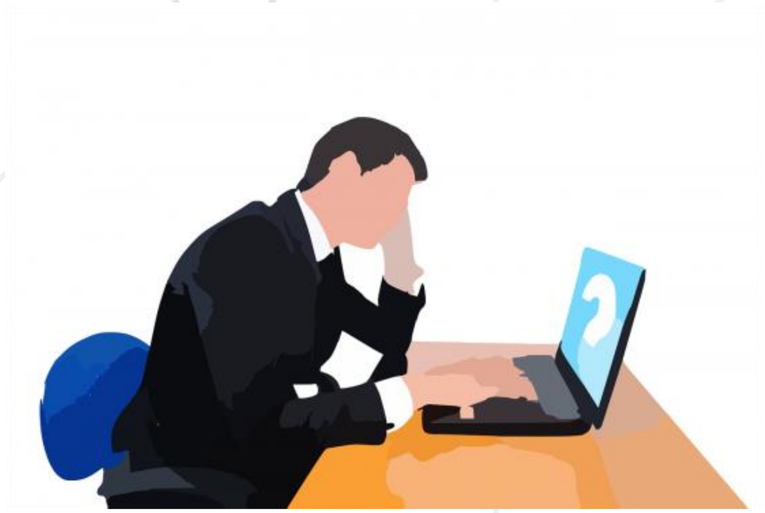


```
routes.MapRoute(
    name: "areas",
    template: "{area:exists}/{controller=Home}/{action=Index}/{id?}"
);
```

Performance

Performance

- **Performance** is an important topic in Web app development
  - Slow-loading discomforts your clients and drives them elsewhere
- There is **no magic** functionality which optimizes your app
  - There are many tips, though, on how to speed up your app



Performance Tips (1)

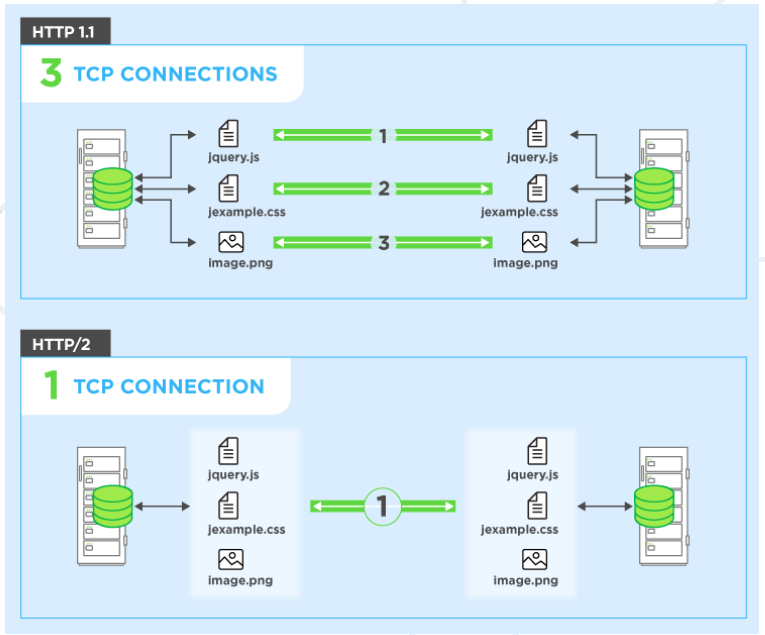
- **Measure everything (Application Insights, dotTrace)**
  - Gather diagnostics for your application
  - Localize which are the slow components of your application
  - Analyze what slows down these components
  - Order and prioritize the most malicious slow-pokes
- **Pick the low-hanging fruit first**
  - Once you've determined the slowest component, prioritize it

Performance Tips (2)

- **Enable Compression**
  - HTTP Protocol is not particularly efficient
  - You can enable Response Compression to increase app efficiency
    - ConfigureServices: `services.AddResponseCompression();`
    - Configure: `app.UseResponseCompression();`
- **Reduce HTTP Requests**
  - HTTP Communication is quite slow
  - Reduce amount of HTTP Requests needed for each functionality
  - Use sprites for images and fonts instead of images

Performance Tips (3)

- **HTTP/2 over SSL (enabled by default)**
  - Binary protocol, Compression of headers
  - Request multiplexing, HTTP 1.1 compatible
- **Minify your files (bundleconfig.json)**
  - Compression is a great tool
  - Your third-party resources are unnecessarily slowing your app
  - You can minify them in order to reduce the data traffic



Performance Tips (4)

- **Load CSS First**
  - CSS Content must be loaded first, preferably in the head section
  - Browsers tend to do unnecessary actions when rendering pages
- **Load JS Last**
  - Pages need to be rendered as quickly as possible
  - JavaScript is not particularly needed for the rendering of pages
  - Of course, this is only applicable to non-heavy JavaScript sites

Performance Tips (5)

- **Cache your pages**
  - There is a lot of static, unchangeable content on web app pages
  - The process of its slow retrieval does not need to be repeated
- **Content Delivery Network (CDN)**
  - CDN outsources a bit of work from your application
  - There are plenty of CDNs closely-located to your clients
  - CDNs are a preferred resource in Production Environment

SEO

Search Engine Optimization

Search Engine Optimization (SEO)

- **Search Engine Optimization** is very important in web apps
  - Common users tend to use Google/Bing to look for services
  - There are ways to boost your place in the results of SEs
- There are several simple **tips** you can follow:
  - Unique content, external links from trustworthy sites
  - Make your application crawlable and fast
  - Make your URLs meaningful
  - Unique and relevant title and description with keywords

GDPR

GDPR

- **General Data Protection Regulation** (**GDPR**) is a regulation in **EU** law
  - Addresses **data protection** and **privacy** of individuals within the **EU**
  - It also addresses export of personal data outside of the EU
- **GDPR** aims to:
  - Provide individuals with more control over their **personal data**
  - Simplify the regulatory environment for **international businesses**
- **ASP.NET Core** provides **APIs** to help meet some **GDPR** requirements
  - There is also a sample app in GitHub here

GDPR

- There are **several individual rights** you must provide your clients
  - Right to be **informed** – inform your clients how you use their personal data
  - Right of **access** – if a client requests their data, you must provide it
  - Right to **rectification** – allow clients to correct inaccurate personal data
  - Right to **erasure** – provide clients with the ability to erase their data
  - Right to **restrict processing** – allow clients to block processing of their data
  - Right to **portability** – allow clients to obtain and reuse their data
  - Right to **object** – allow clients to object to the use of their personal data
  - Rights related to **automatic decision making**, including **profiling**

Summary

- WebHost and WebApplication
- Logging
- Cache
- Sessions
- TempData
- Areas
- Performance
- SEO
- GDPR

Software University

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License