**MemoryCache**

Caching ist das **temporäre Speichern von Daten im Arbeitsspeicher**, um:

* **Antwortzeiten** zu reduzieren (z. B. bei häufigen Datenbankabfragen),
* **Serverlast** zu verringern,
* **Performanz** zu verbessern.

**IMemoryCache in ASP.NET Core**

**Abhängigkeit registrieren (automatisch in ASP.NET Core vorhanden)**

// In Program.cs oder Startup.cs

builder.Services.AddMemoryCache();

**Controller mit Dependency Injection**

*public class HomeController : Controller*

*{*

*private readonly IMemoryCache \_cache;*

*public HomeController(IMemoryCache cache)*

*{*

*\_cache = cache;*

*}*

*...*

*}*

**Beispiel: Aktuelle Zeit mit Ablaufzeit cachen**

*public IActionResult Zeit()*

*{*

*string zeit = \_cache.GetOrCreate("zeit", entry =>*

*{*

*entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(60);*

*return DateTime.Now.ToString("T");*

*});*

*return Content($"Gecachte Zeit: {zeit}");*

*}*

**Ergebnis:**

* Beim ersten Aufruf: neue Zeit gespeichert
* Bei weiteren Aufrufen innerhalb 60 Sekunden: gleiche Zeit angezeigt

**4. Weitere Caching-Optionen**

**Absolute Expiration**

entry.AbsoluteExpiration = DateTimeOffset.Now.AddMinutes(5);

**Sliding Expiration**

entry.SlidingExpiration = TimeSpan.FromMinutes(3);

**Manuelles Setzen:**

\_cache.Set("stadt", "Berlin", TimeSpan.FromMinutes(10));

**5. Beispiel: Datenbankabfrage simulieren**

*public IActionResult Produkte()*

*{*

*var produkte = \_cache.GetOrCreate("produkte", entry =>*

*{*

*entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(10);*

*// Datenbankzugriff simulieren*

*Thread.Sleep(1000); // künstliche Verzögerung*

*return new List<string> { "Tisch", "Stuhl", "Lampe" };*

*});*

*return View(produkte);*

*}*

Vorteil: Der „teure“ Datenbankzugriff wird nur alle 10 Minuten durchgeführt.

**6. Übung**

**Ziel:** Caching auf einen langsamen Service anwenden

1. Simulieren Sie eine Methode GetWitze() mit Thread.Sleep(2000), die eine Liste von Witzen zurückgibt.
2. Cachen Sie die Liste für 2 Minuten mit IMemoryCache.
3. Geben Sie die Witze in einer View aus.
4. Zeigen Sie die aktuelle Uhrzeit zum Vergleich an.

**View mit „Cache-Treffer“-Hinweis**

*public IActionResult Zufallszahl()*

*{*

*if (!\_cache.TryGetValue("zahl", out int zahl))*

*{*

*zahl = new Random().Next(1000);*

*\_cache.Set("zahl", zahl, TimeSpan.FromSeconds(30));*

*ViewBag.Info = "Zahl neu generiert!";*

*}*

*else*

*{*

*ViewBag.Info = "Zahl aus dem Cache.";*

*}*

*return View(zahl);*

*}*

*@model int*

*<p>@ViewBag.Info</p>*

*<p>Zufallszahl: <strong>@Model</strong></p>*

Der MemoryCache (IMemoryCache) in einer **ASP.NET Core Webanwendung** gilt **global für alle Benutzer der Anwendung** – **nicht pro Benutzer**, sondern **anwendungsweit**.

* Der Cache liegt **im Arbeitsspeicher des Servers**.
* Wenn **Benutzer A** eine Ressource speichert, kann **Benutzer B** sie aus dem Cache erhalten (wenn der Key gleich ist).
* Alle Zugriffe auf denselben Key greifen auf denselben Wert zu – unabhängig vom Benutzer.

**Beispiel zur Veranschaulichung:**

public IActionResult Uhrzeit()

{

var zeit = \_cache.GetOrCreate("uhrzeit", entry =>

{

entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromSeconds(30);

return DateTime.Now.ToString("T");

});

return Content($"Gecachte Uhrzeit: {zeit}");

}

Wenn Benutzer A die Seite aufruft, wird DateTime.Now gespeichert.  
Wenn **innerhalb von 30 Sekunden** Benutzer B dieselbe Seite aufruft, sieht er dieselbe Uhrzeit – obwohl er ein anderer Benutzer ist.

**Was tun bei benutzerspezifischem Caching?**

Wenn du **benutzerspezifische Daten** cachen willst, kannst du den Cache-Key mit einer Benutzerkennung kombinieren, z. B.:

string benutzerId = User.Identity.Name; // oder UserId aus Claims

string key = $"profil\_{benutzerId}";

var daten = \_cache.GetOrCreate(key, entry =>

{

entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5);

return LadeBenutzerdaten(benutzerId);

});

|  |  |  |
| --- | --- | --- |
| Cachetyp | Sichtbarkeit | Gültigkeit |
| MemoryCache | global (alle Benutzer) | während Laufzeit der App |
| Session | benutzerspezifisch | pro Benutzer + Sitzung |
| ViewData, ViewBag, TempData | nur für aktuelle Request/Redirect | sehr kurzlebig |
| DistributedCache (z. B. Redis) | anwendungsübergreifend | auch für mehrere Instanzen |

|  |  |  |  |
| --- | --- | --- | --- |
| Methode | Beschreibung | Beispiel | Wann verwenden? |
| SetSlidingExpiration | Verlängert die Lebensdauer bei jedem Zugriff. Läuft ab, **wenn sie nicht benutzt wird**. | .SetSlidingExpiration (TimeSpan.FromMinutes(5)) | Wenn der Cache **nur bei Inaktivität** ablaufen soll. |
| SetAbsoluteExpiration | Legt eine **feste maximale Lebensdauer** fest – **unabhängig von Zugriffen**. | .SetAbsoluteExpiration(TimeSpan.FromMinutes(30)) | Wenn die Daten **immer nach einer bestimmten Zeit** neu geladen werden sollen. |