Sezione 12

I servizi di ASP.NET Core: configurazione, logging e caching

IConfiguration

- È il servizio che ci permette di accedere ai valori di configurazione;
- Ha un metodo `GetSection` per ottenere il riferimento a una sezione;
- E un metodo `GetValue<T>` per ottenere il valore.

appsettings.json

```
{
  "A": {
    "B": {
      "C": "Valore"
    }
  }
}
```

```
string c = Configuration
                .GetSection("A")
                .GetSection("B")
                .GetValue<string>("C");

string c = Configuration
                .GetValue<string>("A:B:C");

string c = Configuration["A:B:C"];
```

IConfiguration (ottenere una connection string)

- C'è un apposito metodo `GetConnectionString`

appsettings.json

```
{
  "ConnectionStrings": {
    "Default": "..."
  }
}
```

```
string c = Configuration
           .GetConnectionString("Default");
```

È una scorciatoia per:

```
string c = Configuration
           .GetSection("ConnectionStrings")
           ["Default"];
```

IOptionsMonitor<T>

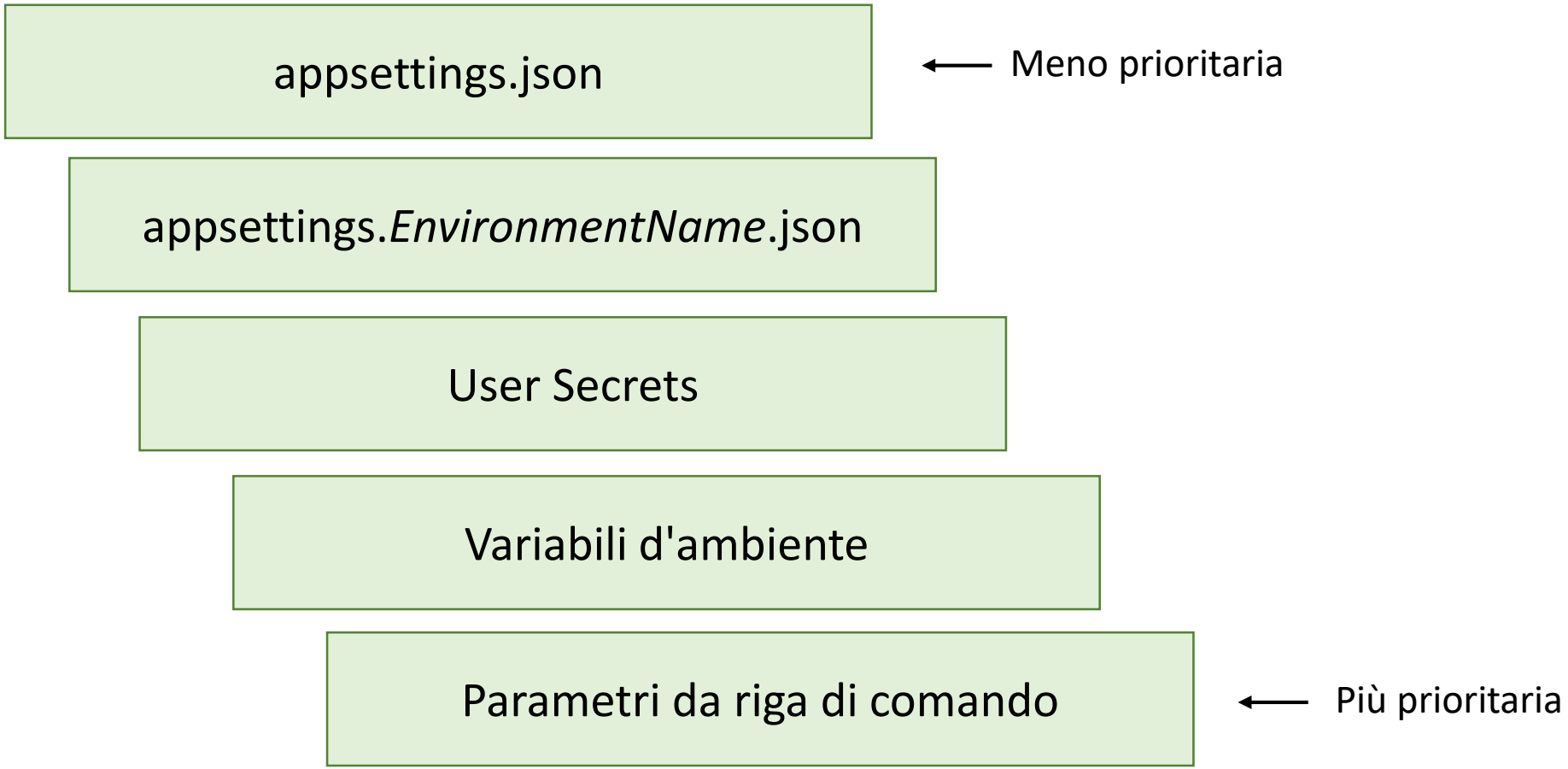
È il servizio per accedere ai valori di configurazione in maniera fortemente tipizzata.

1. Creiamoci una classe come `ConnectionStringsOptions`, in cui mettiamo proprietà chiamate come i valori di configurazione;

2. Registriamo la classe dal metodo `ConfigureServices` con:
```
services.Configure<ConnectionStringsOptions>(
                    Configuration.GetSection("ConnectionStrings"));
```

3. Riceviamo il servizio `IOptionsMonitor<ConnectionStringsOptions>` nel costruttore dei nostri componenti e leggiamo la sua proprietà `CurrentValue`.

Priorità delle fonti di configurazione di default



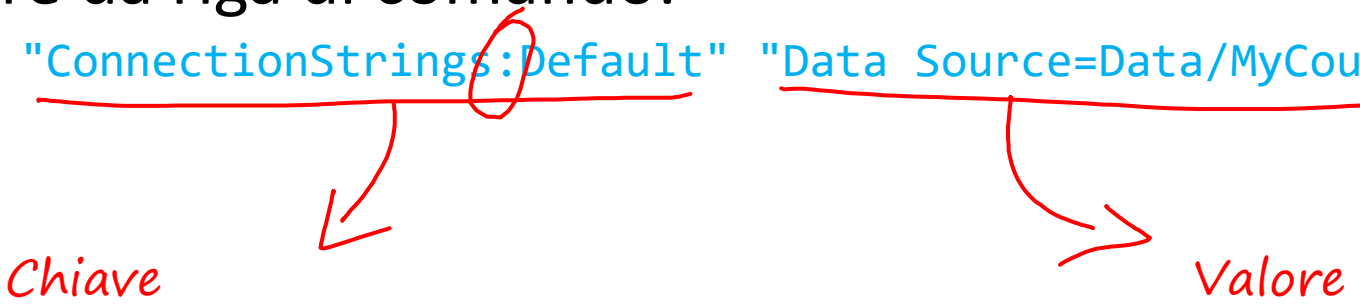
User secrets (solo per l'ambiente di sviluppo)

- I valori sono conservati in una directory esterna al progetto.

Sistema	Percorso della directory
Windows	`%APPDATA%\Microsoft\UserSecrets`
Linux e Mac	`~/.microsoft/usersecrets/`

- Per impostare un valore da riga di comando:

```
dotnet user-secrets set "ConnectionStrings:Default" "Data Source=Data/MyCourse.db"
```



Variabili d'ambiente

- Sono coppie chiave-valore impostate a livello di sistema (o a livello di utente di sistema) e perciò accessibili alle applicazioni.

Sistema	Procedura da eseguire
Windows	Lanciare il seguente comando da prompt dei comandi `setx "ConnectionStrings:Default" "DataSource=Data/Test123.db" /m`
Linux	In `/etc/environment` aggiungere: `ConnectionStrings__Default="DataSource/Test123.db"`
Mac	In `~/.bash_profile` aggiungere: `export ConnectionStrings__Default="DataSource/Test123.db"`

Parametri da riga di comando

- Si aggiungono in coda al comando di avvio dell'applicazione:

```
dotnet run --ConnectionStrings:Default="Data Source=Data/TestABC.db"

dotnet MyCourse.dll --ConnectionStrings:Default="Data Source=Data/TestABC.db"
```

ILogger<T>

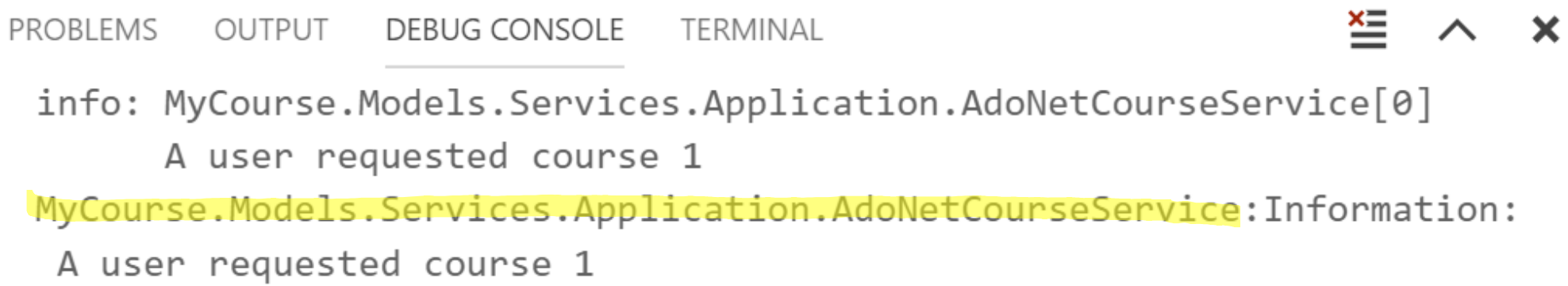
- È il servizio integrato in ASP.NET Core per loggare messaggi;
  - Come al solito lo riceviamo dal costruttore di un nostro componente.

```
private readonly ILogger<CourseService> logger;

public CourseService(ILogger<CourseService> logger)
{
  this.logger = logger;
}

public async Task<CourseDetailViewModel> GetCourseAsync(int id)
{
  logger.LogInformation("A user requested course {id}", id);
  //...
}
```

Logging strutturato!

ILogger<T>

- Il tipo T sarà usato per indicare la categoria del messaggio di log.

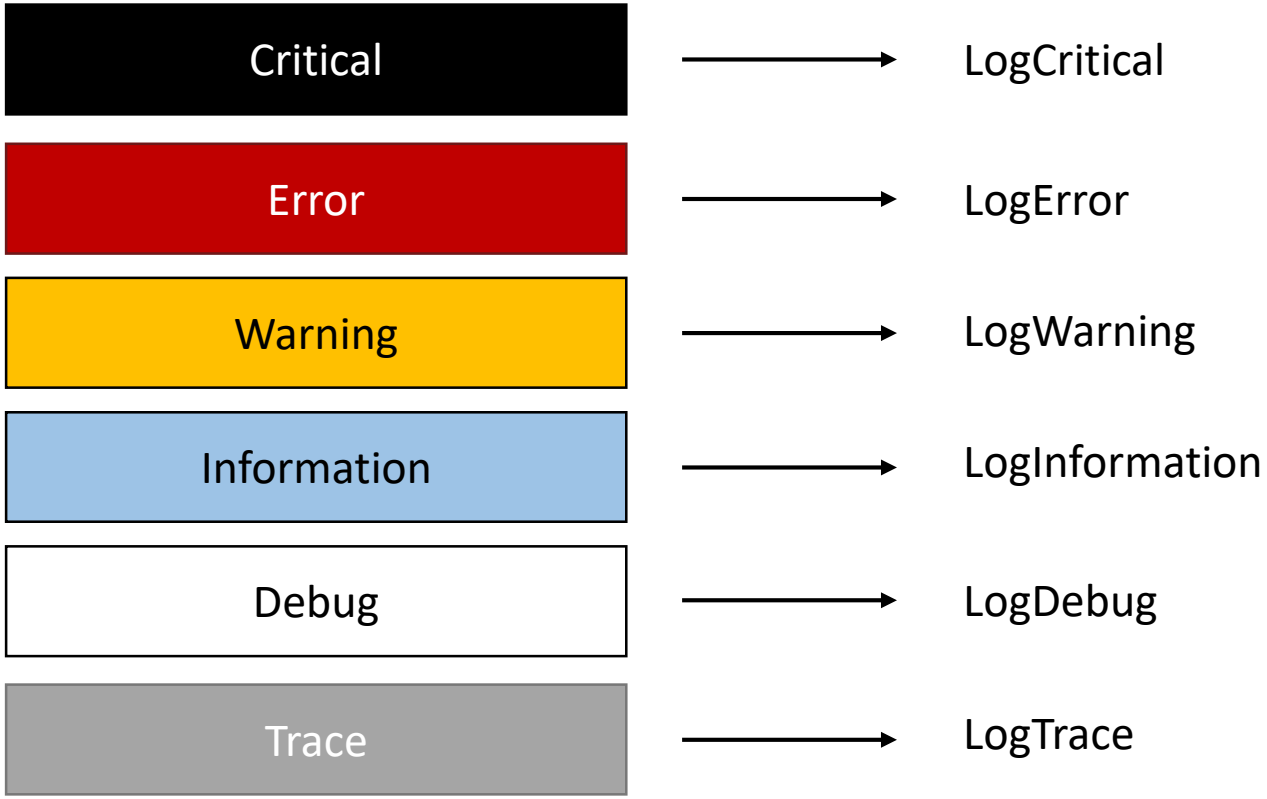


ILoggerFactory

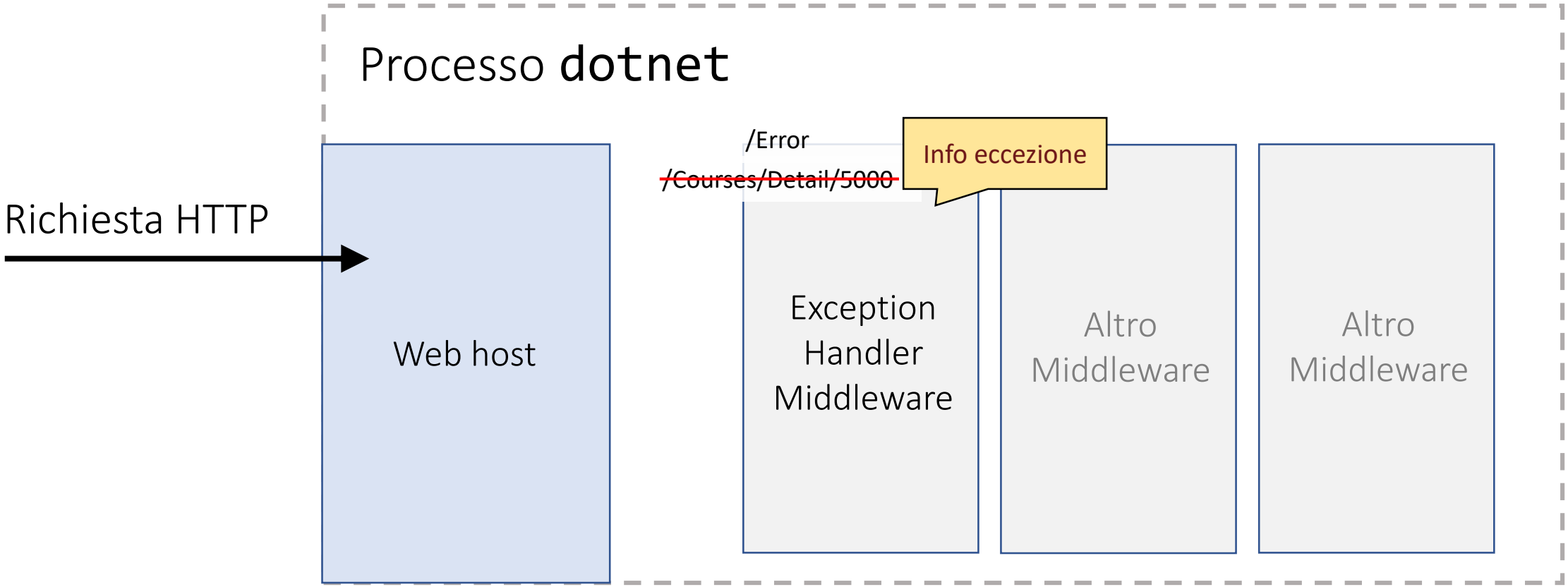
- Oppure usiamo il servizio `ILoggerFactory` per creare un `ILogger`;
  - Utile se vogliamo avere il controllo sul nome della categoria.

```
private readonly ILogger logger;

public CourseService(ILoggerFactory loggerFactory)
{
  this.logger = loggerFactory.CreateLogger("Courses");
}

public async Task<CourseDetailViewModel> GetCourseAsync(int id)
{
  logger.LogInformation("A user requested course {id}", id);
  //...
}
```

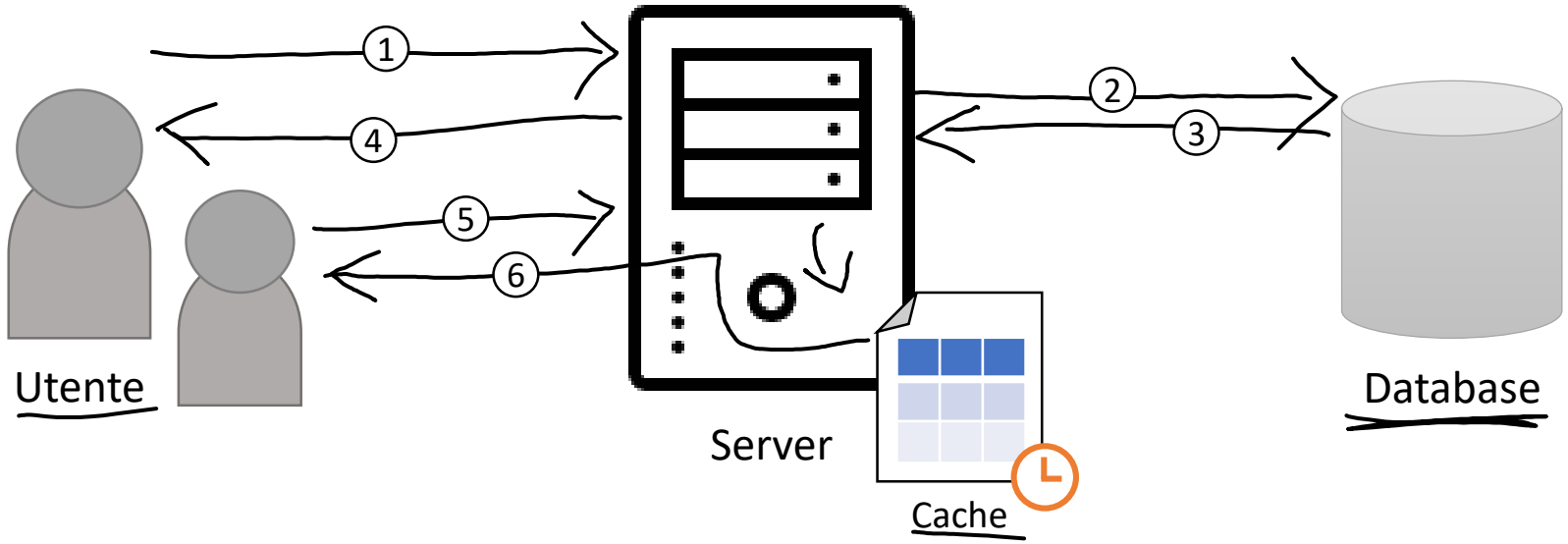
Livello dei messaggi di log



L'ExceptionHandlerMiddleware



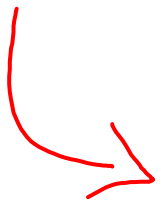
Usare la cache



Due strategie per far scadere un oggetto

- Absolute expiration
  - 30 minuti da adesso;
  - Alle 20.00 di oggi;

- Sliding expiration
  - 30 minuti da adesso, ma se l'oggetto viene letto, allora postponi la scadenza.



Attenzione perché l'oggetto potrebbe non essere mai rimosso se almeno un utente continua ad accedervi almeno una volta prima che scada

IMemoryCache

Lo usiamo per scrivere e rileggere oggetti nella memoria RAM.

- La memoria RAM è molto veloce, perciò recuperare un oggetto da lì è molto più rapido che ottenere dei risultati da un database.
- Ogni oggetto è rappresentato da una chiave.

```
var viewModel = memoryCache.GetOrCreate("Chiave", cacheEntry => {
  cacheEntry.SetAbsoluteExpiration(TimeSpan.FromSeconds(60));
  //cacheEntry.SetSlidingExpiration(TimeSpan.FromSeconds(60));
  return new CourseViewModel();
});
```

thread safe

IMemoryCache: vantaggi e svantaggi

VANTAGGI

- Miglioramento delle prestazioni

Si misura con un test di carico

SVANTAGGI

- Gli utenti potrebbero vedere dati non aggiornati
- Aumento del consumo di memoria RAM
- Effetti collaterali se si scala orizzontalmente

Absolute e sliding expiration

ABSOLUTE EXPIRATION: L'oggetto viene rimosso dalla cache alla scadenza

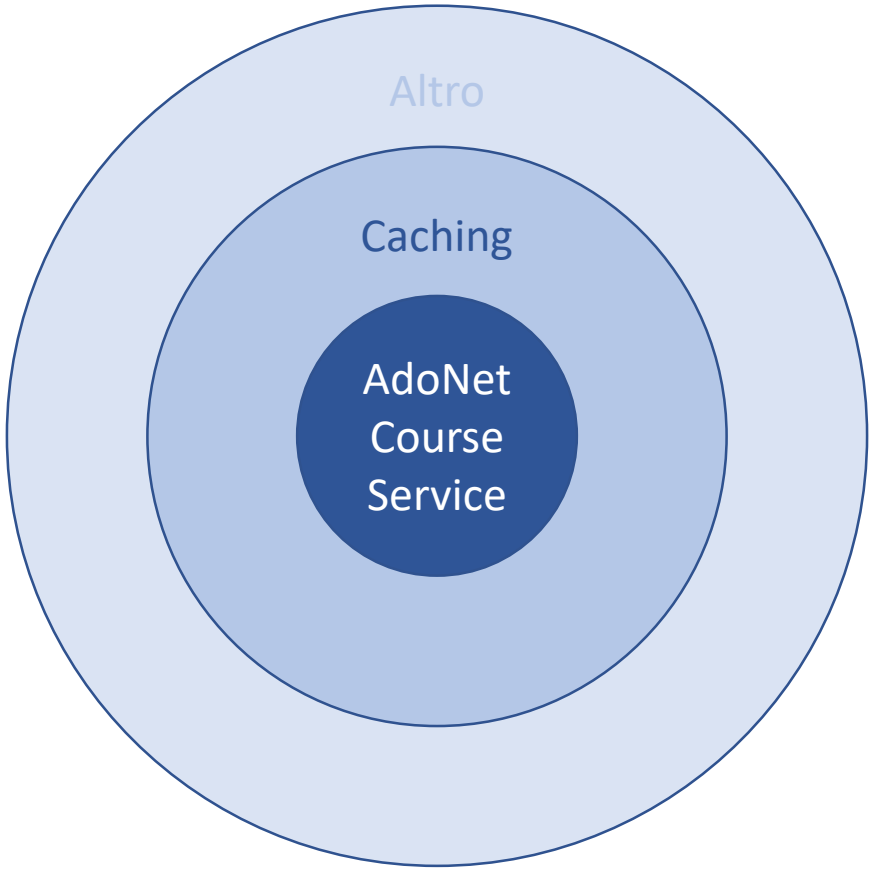
```
var result = memoryCache.GetOrCreateAsync("Chiave", cacheEntry => {
  cacheEntry.SetAbsoluteExpiration(TimeSpan.FromSeconds(60));
  return espressione;
});
```

SLIDING EXPIRATION: La scadenza continua a rinnovarsi finché la chiave cache viene acceduta

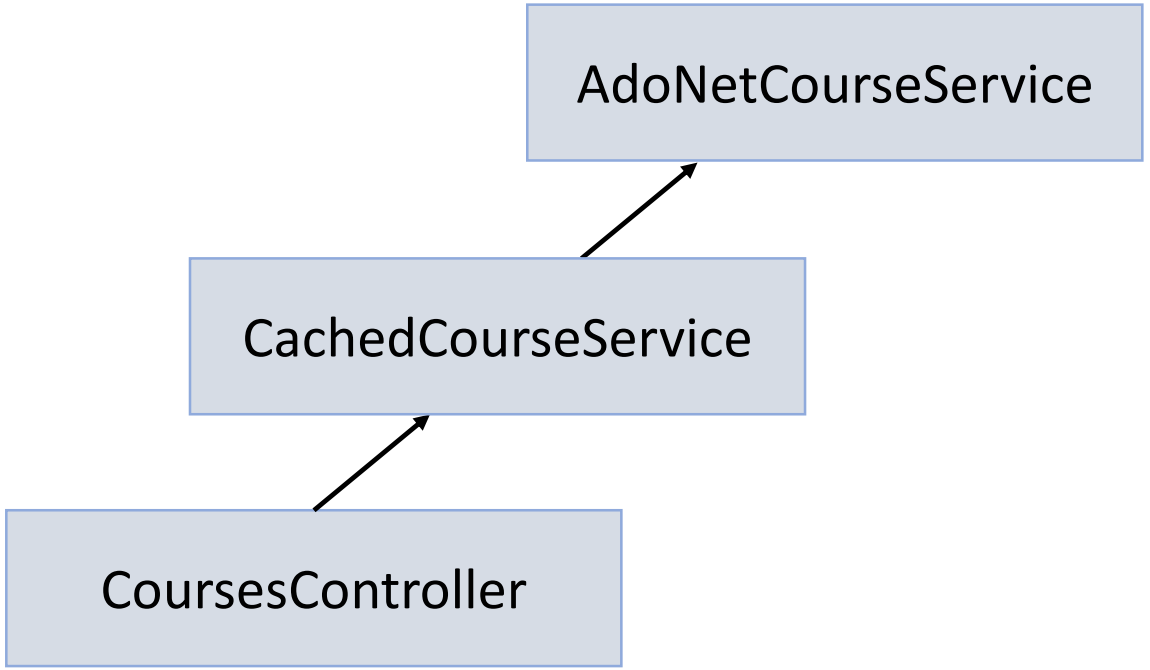
```
var result = memoryCache.GetOrCreateAsync("Chiave", cacheEntry => {
  cacheEntry.SetSlidingExpiration(TimeSpan.FromSeconds(60));
  return espressione;
});
```

Attenzione perché l'oggetto potrebbe non essere mai rimosso se almeno un utente continua ad accedervi almeno una volta prima che scada

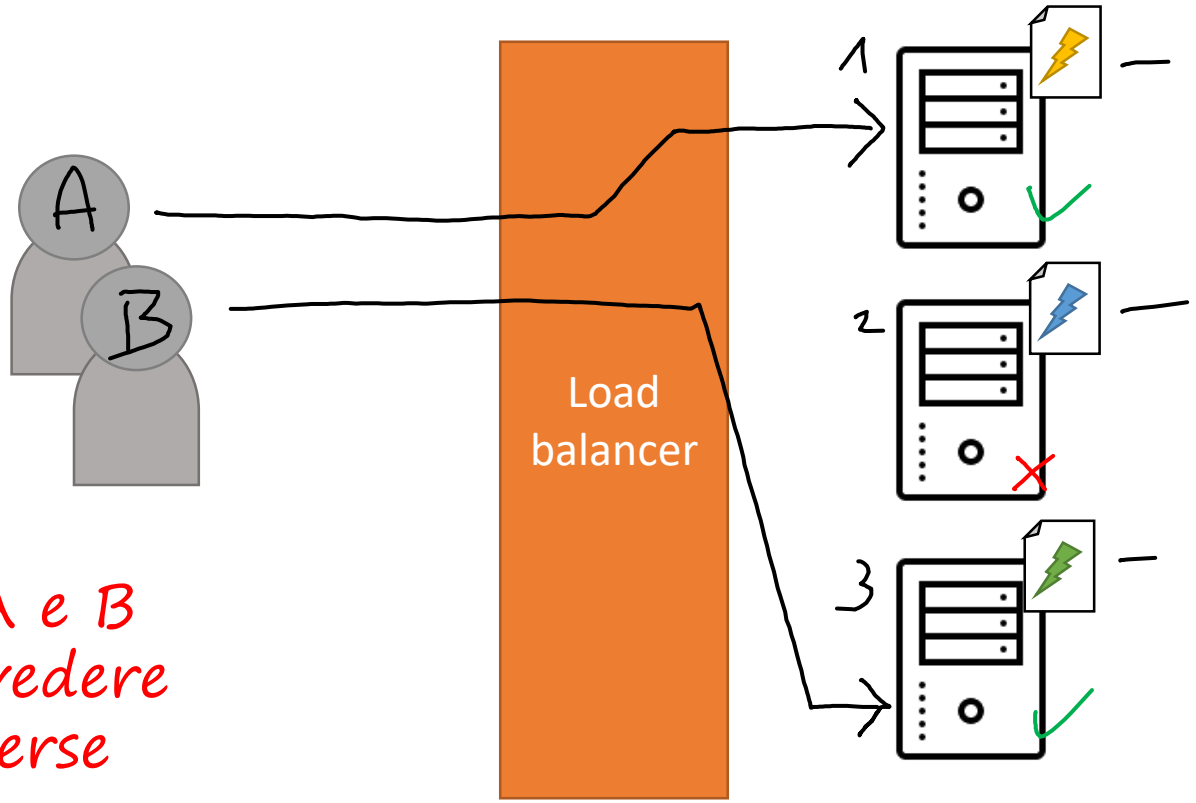
Decorator pattern



Decorator pattern

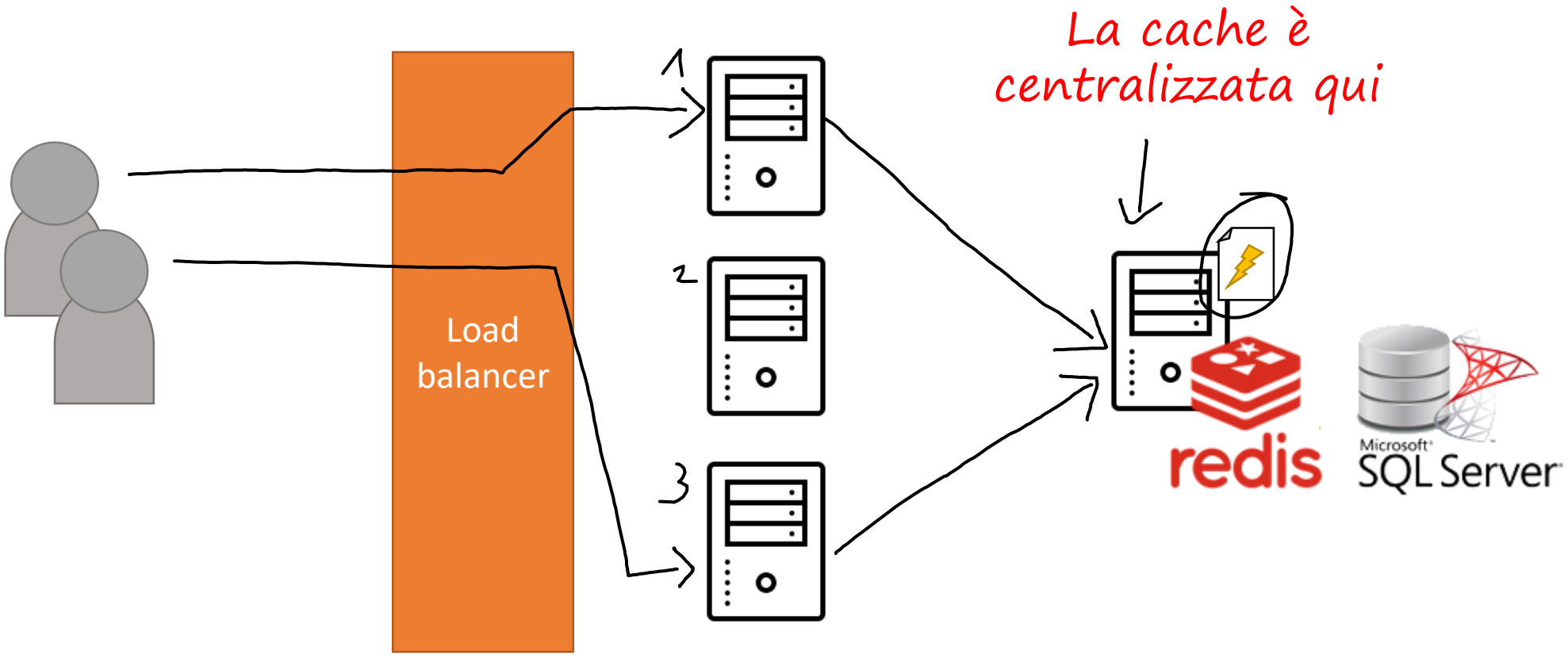


Effetti collaterali di IMemoryCache con scalabilità orizzontale

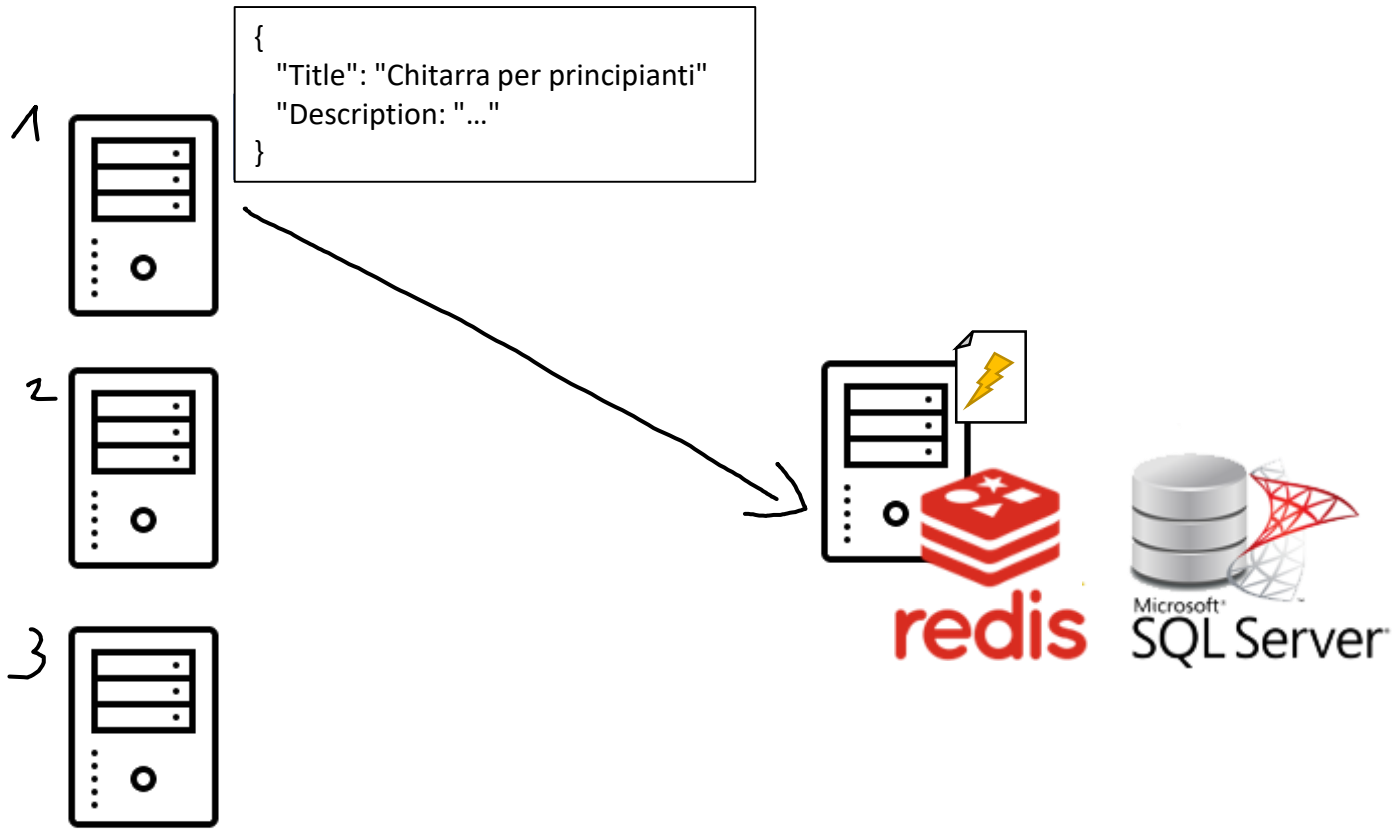


Gli utenti A e B potrebbero vedere pagine diverse

IDistributedCache



Cache distribuita: gli oggetti vanno serializzati



IDistributedCache

Recupero l'oggetto (è serializzato)

```
string key = $"Courses";
string serializedObject = await distributedCache.GetStringAsync(key);
```

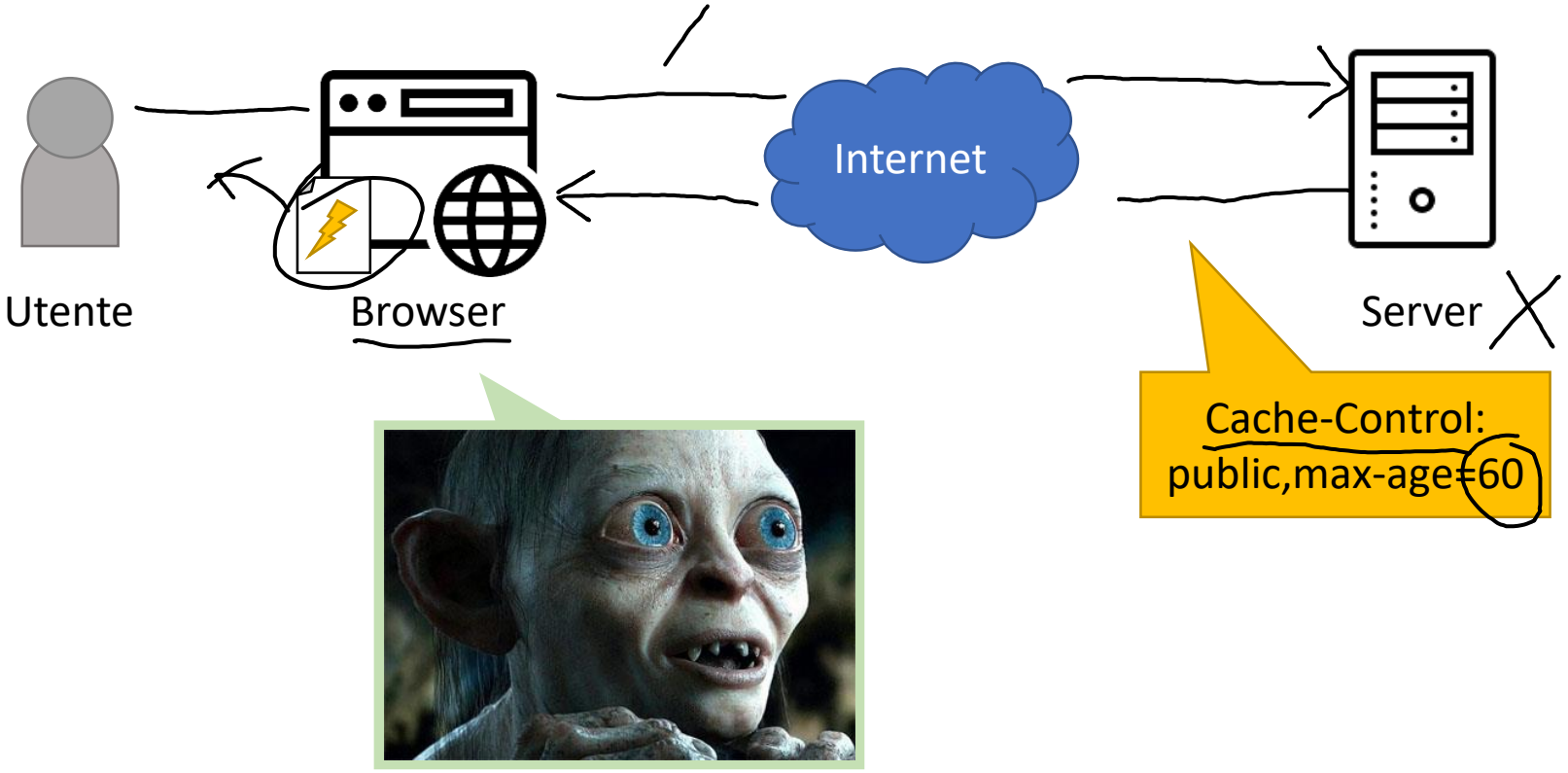
2. Se esisteva, lo deserializzo e lo restituisco

```
if (serializedObject != null) {
  return Deserialize<List<CourseViewModel>>(serializedObject);
}
List<CourseViewModel> courses = await courseService.GetCoursesAsync();
serializedObject = Serialize(courses);
await distributedCache.SetStringAsync(key, serializedObject);
return courses;
```

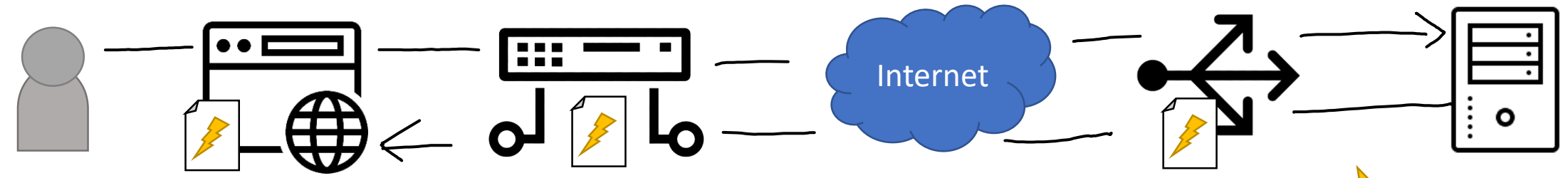
3. Se non esisteva, lo recupero dal db, lo serializzo e lo metto in cache. Poi lo restituisco

Funzionamento del response caching

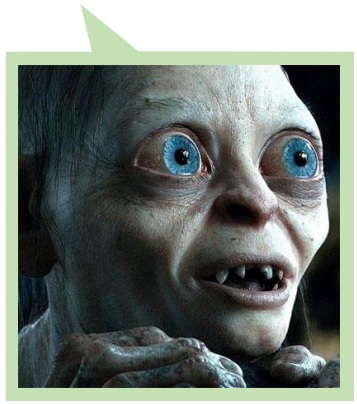
Il contenuto HTML viene messo in cache AL DI FUORI dell'applicazione.

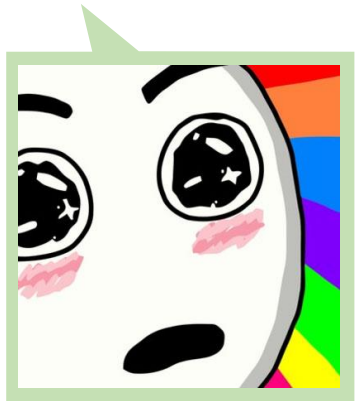


Vari dispositivi possono avvalersi della cache



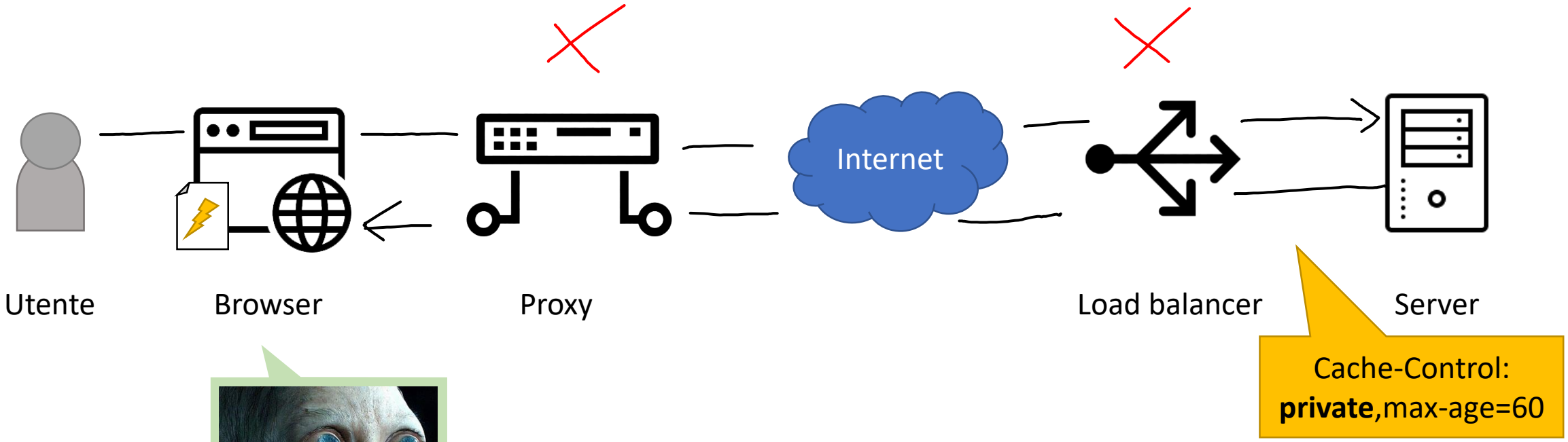
Utente Browser Proxy Load balancer Server

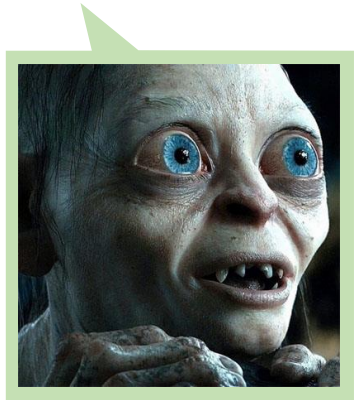




Cache-Control: **public**,max-age=60

Solo il browser può tenere il risultato in cache se Cache-Control è impostato su private





Attributo ResponseCache


```
[ResponseCache(Duration = 60, Location = ResponseCacheLocation.Any)]
public IActionResult Index()
{
  ViewData["Title"] = "Benvenuto su MyCourse!";
  return View();
}
```


```
[ResponseCache(Duration = 60, Location = ResponseCacheLocation.Client)]
public IActionResult Index()
{
  ViewData["Title"] = "Benvenuto su MyCourse!";
  return View();
}
```

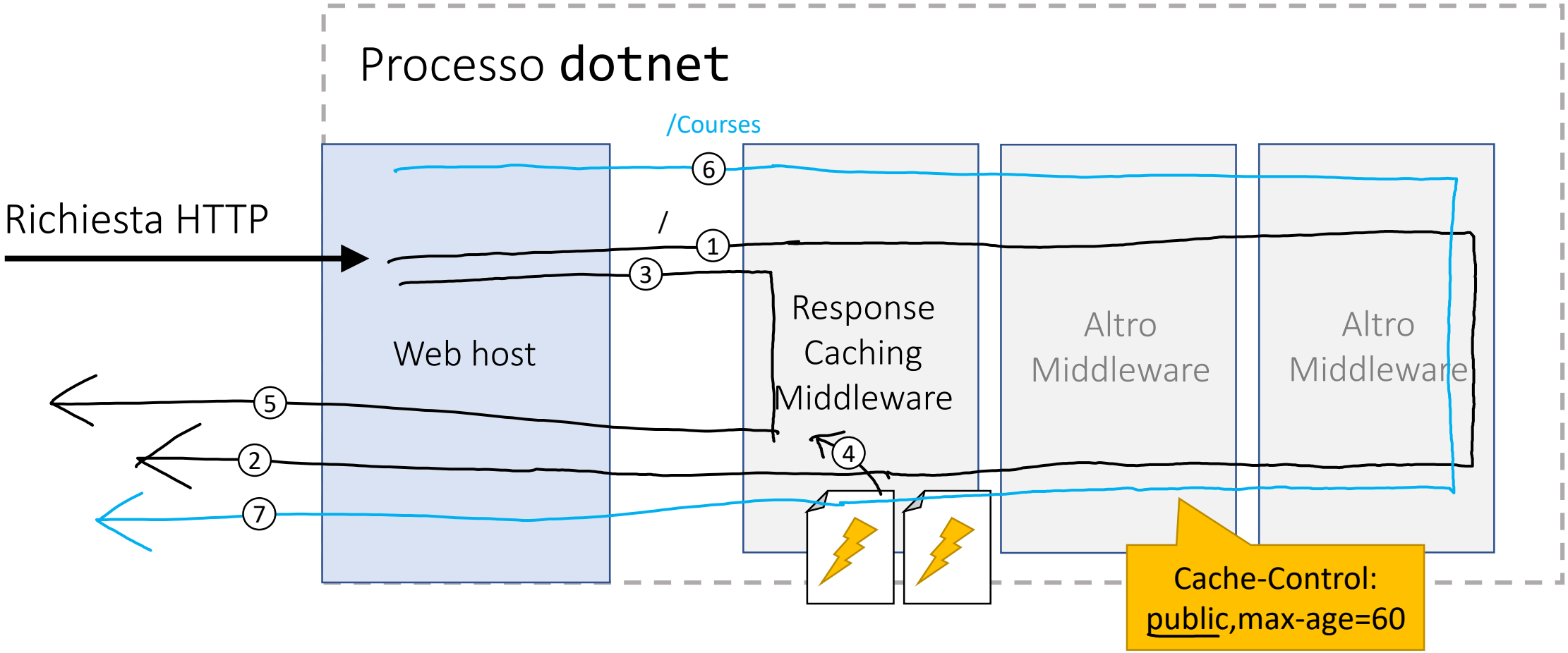
Usiamo i profili con l'attributo ResponseCache

```
[ResponseCache(CacheProfileName = "Home")]
public IActionResult Index()
{
  ViewData["Title"] = "Benvenuto su MyCourse!";
  return View();
}
```

Definiamo il profilo "Home" nel metodo ConfigureServices della classe Startup

```
services.AddMvc(options =>
{
  var homeProfile = new CacheProfile();
  homeProfile.Duration = Configuration.GetValue<int>("ResponseCache:Home:Duration");
  homeProfile.Location = Configuration.GetValue<ResponseCacheLocation>("ResponseCache:Home:Location");
  options.CacheProfiles.Add("Home", homeProfile);
}).SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

Il ResponseCachingMiddleware



Il ResponseCachingMiddleware

- Determina la chiave cache in base al "percorso" della richiesta:
  METHOD+SCHEME+HOST:PORT/PATHBASE/PATH

- http://localhost:5000/ ≠ https://localhost:5001/
- https://localhost:5001/Courses = https://localhost:5001/Courses?page=2

Usiamo il response caching per i contenuti comuni a tutti gli utenti

Risultati del test di carico

Con 200 richieste contemporanee a pagina di elenco e di dettaglio

	Richieste al secondo	Richieste fallite
Ambiente Development (logging: Information)		
Con Entity Framework Core	49	11
Ambiente Production (logging: Warning)		
Con Entity Framework Core	390	0
Con ADO.NET	407	0
Con caching in memoria	1608	0
Con caching distribuita su SQL Server	458	0
Con caching distribuita su Redis	500	0
Con ResponseCachingMiddleware	2133	0

Pacchetti Serilog

Pacchetto	A cosa serve
Serilog	Funzionalità base
Serilog.AspNetCore	Integrazione con ILogger<T>
Serilog.Sinks.Console	Scrivere su Console
Serilog.Sinks.File	Scrivere su File
Serilog.Settings.Configuration	Integrazione con IConfiguration