In-Memory Caching

IMemoryCache interface provides the facility for in-memory caching.

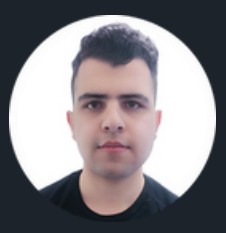
```
public class ArticleService : IArticleService
{
    private readonly IMemoryCache _memoryCache;
    private const string CacheKey = "articles";

    public ArticleService(IMemoryCache memoryCache)
    {
        _memoryCache = memoryCache;
    }

    public async Task<List<Article>> GetArticles()
    {
        if (!_memoryCache.TryGetValue(CacheKey, out List<Article>? articles))
        {
            articles = await GetValuesFromDbAsync();

            var cacheExpiryOptions = new MemoryCacheEntryOptions
            {
                AbsoluteExpiration = DateTime.Now.AddMinutes(5),
                Priority = CacheItemPriority.High,
                SlidingExpiration = TimeSpan.FromMinutes(2),
                Size = 1024,
            };

            _memoryCache.Set(CacheKey, articles, cacheExpiryOptions);
        }
        return articles ?? new List<Article>();
    }
}
```



Elliot One

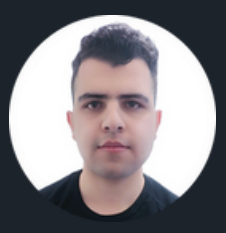
Using MemoryCache Service

MemoryCache service should be registered in Program.cs.

```
builder.Services.AddScoped<IArticleService, ArticleService>();
builder.Services.AddMemoryCache();
```

```
[Route("api/[controller]")]
[ApiController]
public class ArticleController : ControllerBase
{
    private readonly IArticleService _articleService;
    public ArticleController(IArticleService articleService)
    {
        _articleService = articleService;
    }

    [HttpGet]
    public async Task<ActionResult<List<Article>>> GetArticles()
    {
        return await _articleService.GetArticles();
    }
}
```



Elliot One

Distributed Redis Caching

IDistributedCache interface provides the facility for distributed caching.

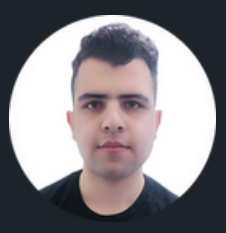
```
public class ArticleService : IArticleService
{
    private readonly IDistributedCache _distributedCache;
    private const string CacheKey = "articles";

    public ArticleService(IDistributedCache distributedCache)
    {
        _distributedCache = distributedCache;
    }

    public async Task<List<Article>> GetArticles()
    {
        var cachedArticles = await _distributedCache.GetStringAsync(CacheKey);
        if (cachedArticles != null)
        {
            return JsonConvert.DeserializeObject<List<Article>>(
              cachedArticles) ?? new List<Article>();
        }

        var articles = await GetValuesFromDbAsync();
        if (articles.Any())
        {
            await SetArticles(articles);
            return articles;
        }
        return new List<Article>();
    }

    private async Task SetArticles(List<Article> articles)
    {
        var cacheOptions = new DistributedCacheEntryOptions
        {
            AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(5),
            SlidingExpiration = TimeSpan.FromMinutes(2),
        };
        await _distributedCache.SetStringAsync(
          CacheKey, JsonConvert.SerializeObject(articles), cacheOptions);
    }
}
```



Elliot One

Using RedisCache Service

StackExchangeRedisCache service should be registered in Program.cs.

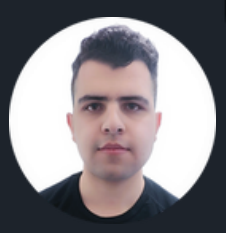
```
builder.Services.AddStackExchangeRedisCache(options =>
{
  options.Configuration = "Your_Redis_Connection_String";
  // Optional: Instance name for distinguishing multiple caches
  options.InstanceName = "SampleInstance";
});

builder.Services.AddScoped<IArticleService, ArticleService>();
```

```
[Route("api/[controller]")]
[ApiController]
public class ArticleController : ControllerBase
{
  private readonly IArticleService _articleService;

  public ArticleController(IArticleService articleService)
  {
    _articleService = articleService;
  }

  [HttpGet]
  public async Task<ActionResult<List<Article>>> GetArticles()
  {
    var articles = await _articleService.GetArticles();
    return Ok(articles);
  }
}
```



Elliot One

Elliot One

Enjoyed Reading This?

Reshare and Spread Knowledge.