Caching in ASP.NET Core

Using LRU Cache (Least Recently Used) and LFU Cache (Least Frequently Used)

ⓘ Introduction to Caching

Caching is a technique that stores frequently accessed data in a **fast-access storage layer** to reduce latency and improve performance.

Why Cache?

```
Without Cache:    User Request → Database (Slow) → Response (200-500ms)
With Cache:       User Request → Cache (Fast) → Response (5-20ms)
                                    ↓ (if not found)
                                Database → Store in Cache
```

LRU Cache (Least Recently Used)

How LRU Cache Works

Concept: "If you haven't used it recently, you probably won't need it soon"

```
Data Structure:
- Hash Map (Dictionary) for O(1) lookups
- Doubly Linked List for O(1) reordering

Operations:
1. GET(key): Move item to front (most recent)
2. PUT(key, value):
   - If exists: Update value, move to front
   - If full: Remove tail (least recent), add to front
```

When to Use LRU:

- ✓ **Browser Cache** - Caching web pages and resources
- ✓ **Operating System Page Cache** - Memory management
- ✓ **Database Query Cache** - Frequently executed queries
- ✓ **Session Management** - User session data
- ✓ **API Response Caching** - REST API responses

LFU Cache (Least Frequently Used)

How LFU Cache Works

Concept: "If you haven't used it much, you probably don't need it"

```
Data Structure:
1. Hash Map: key → (value, frequency, node)
2. Frequency Map: frequency → Doubly Linked List
3. Min Frequency tracker

Operations:
1. GET(key): Increase frequency, move to next freq list
2. PUT(key, value):
   - If exists: Update value, increase frequency
   - If full: Remove from min frequency list
```

When to Use LFU:

- ✓ **Content Delivery Networks (CDN)** - Popular static content
- ✓ **Search Engine Results** - Frequently searched queries
- ✓ **E-commerce Product Catalog** - Hot products
- ✓ **Advertisement Serving** - Frequently displayed ads
- ✓ **Compiler Optimization** - Frequently used code paths

LRU vs LFU Comparison

Choose LRU When:	Choose LFU When:
• Recent access matters more than frequency	• Frequency of access matters more than recency
• Sequential access patterns exist	• Stable access patterns with identifiable popular items
• Memory is constrained	• Can tolerate additional memory overhead
• Simple implementation is needed	• Popular content needs to stay cached longer
• Cache pollution from one-time accesses is a concern	• Long-term popularity is important

Key Takeaways

Performance Impact

Caching can reduce response times from 200-500ms to 5-20ms.

Implementation Complexity

LRU is simpler to implement than LFU, which requires frequency tracking.

Temporal Locality

LRU exploits temporal locality (recently used items will likely be used again).

Frequency Patterns

LFU exploits frequency patterns (frequently used items will likely be used again).

Implementation Summary

LRU Implementation Tips:

- Use Dictionary + LinkedList in C#
- Implement cache size limit
- Update recency on every access
- Consider thread safety for web applications

LFU Implementation Tips:

- Use Dictionary + Dictionary of LinkedLists
- Track minimum frequency
- Handle frequency promotion carefully
- Watch for memory overhead

Suraj Goud