Mini Uri

(I checked TinyUrl, and it seems to support any valid Uri[[1]](#footnote-1), so this implementation does the same)

The Long Uri to Mini Uri mapping is a Key-Value lookup problem. Since 2 users can request a “TinyUrl” for the same Uri, we need to assign a unique id/key to each request. Once an id/key has been assigned, it has to be encoded into for representation in a URL and also be a unique key that can be used for storage operations. Using a Hash will not work as we will never be able to disambiguate a collision

I am using Int64 for my request id - which can have 264 unique values. Not all systems/languages support UInt64 (ie SQL Server), so we can start the sequence at Int64.MinValue and still have the same number of unique values.

Once we have an Id, it can be base64 encoded to be represented as a string of 11 alphanumeric characters. If we restrict user keys to 10 or less, we can detect a possible user key based on the length. This allows us to use a secondary index which stores a pointer to the main index entry, which keeps all the other functions for lookup/delete/stats the same.

# Projects In Solution

This solution requires Visual Studio 2022 and Net6 to run.

## MiniUri.Common

This library contains some base classes as well as utility interfaces for a sequence generator and encoder/decoder. The plugins folder contains some concrete implementations those interfaces. The output targets Net6 and NetStandard2.0 for older C# code support.

## MiniUri.UriService.Contracts

All data models contracts and interface definitions that a client would need to connect to the ‘micro service. No logic or implementation details. Since we are in 100% in memory and no network/message protocol has been decided, serialization concerns can be ignored. Also targeting NetStandard2.0 for older C# code support.

For the data models, there are 2 types:

**Records** are immutable (new C# feature)

**Entities** are internally editable in the service layer, but exposed to the world via immutable Read Only interfaces.

**IUriService** is the service contract interface with all available methods a client would need.

## MiniUri.UriService.Implementation

This contains all the service logic. The storage layer and validation are extracted as interfaces so they can be swapped out as plugins with different implementations and unit tested easier. The service can be implemented in Net6. Since the Contracts are NetStandard2.0 compatible, new and old C# code could use this service the same transport layer was available for both.

Many of the possible thread contention points that would be of concern with the Storage Layer have been mitigated by design choices and the use of lock-free / minimal locking concurrent collections[[2]](#footnote-2)

Most of the updates are not updates as visitor information and counts are appended to a separate log for each Short Uri. Even the updates that do happen for deletion marking or sliding expiration timestamp, don’t require locking.

**Storage Layer**

|  |  |  |
| --- | --- | --- |
| **Short Uri** | **Add Operation** | **Lookup/Update/Delete Operations** |
| System Generated  (11 Characters) | All Adds should succeed | 1 Step – Decode Short Uri for storage Id to locate record and then perform operation. |
| User Request  (10 or less characters) | Some Adds can fail if not available | 2 Step – Check Secondary Index for existence of a storage Id mapping. If exists, use the mapped Id for all other operations |

**Storage Id Generator**

The storage index is maintained externally by the IdGenerator. Similar to how a Guid can be generated by the client, some architectural aspects can be simplified if the client can generate a key that doesn’t have to rely on an external service call. Different bits or key ranges can be used to partition data for scaling / performance improvements.

It should be noted, that there is a (very) small chance that id generated will be wasted. This would only occur if two users attempted to request the same desired key at such an exact time that two threads passed the first gate of the double-checked lock pattern. It is only by passing the first test that an Id is formally requested and the sequence advanced.

## MiniUri.Tests

The unit tests show usage examples of the interface and internal plugins.

## MiniUri.Web

Hosts a web user interface that uses calls the UriService. It’s not a ‘Web Service’, but it is a little nicer than a console application. It uses server-size Blazor which allows client-side web interaction with C# instead of JavaScript.

1. [What's the Difference Between a URI and a URL? - Daniel Miessler](https://danielmiessler.com/study/difference-between-uri-url/) [↑](#footnote-ref-1)
2. [FAQ :: Are all of the new concurrent collections lock-free? - .NET Parallel Programming (microsoft.com)](https://devblogs.microsoft.com/pfxteam/faq-are-all-of-the-new-concurrent-collections-lock-free/) [↑](#footnote-ref-2)