Approach

First off, we would want to separate the process of maintaining the links from the web application that is serving those links, since we have approximately 10K entries a database table is an acceptable solution to maintain that list of links.

We would design a small utility program (CLI program) that is designed to run on a periodic bases to update the links periodically, the executable of which shall be run as an scheduled task [0] on Windows systems (Cron Tab on *nix Systems) lets say once at the initiation of this program (with force refresh flag set to true) and once every 1 hour thereafter (with force refresh flag set to false) to keep the links and cached documents in-sync.

Our database table needs to maintain the last timestamp a document was retrieved inorder to be able to signal the user on how fresh a particular document is (as the application logic requires us to mark things that was changed in the last 3 days)

This presents a new issue to consider, how do we tell if a document changed?
An appropriate measure here would be to maintain a content hash, lets avoid md5 to avoid potential collisions however lets not get too carried away so I assume sha-256 shall suffice. Every document we scrape & download we will compute the sha-256 for and maintain that hash along with the time stamp on which that document was retrieved.

Thus telling if a document has changed in the last 3 days would boil down to:

- Timestamp of Now - Last Update Timestamp in DB <= 3 days
- Current sha-256 != sha-256 of content_hash field in DB

Which should trigger the UI signal to the user to show that the document has changed in the last 3 days, and perhaps activate a link to allow for a refreshed version of the document to be retrieved.

Once we retrieve the document we have 2 options (or trade-offs) to consider.

a) Add a suffix or use datetime-stamp named folders to keep all versions of a particular document (may be it would useful from a regulatory perspective to keep all docs at the cost of storage)
b) Simply overwrite the old doc with the new version.

For time purpose I have chosen the server's own file system as a flat / document store being fully aware of the limitations [1] of a regular file system such as NTFS or FAT32 being as follows:

FAT32

- Maximum disk size: 2 terabytes

- Maximum file size: 4 gigabytes
- Maximum number of files on disk: 268,435,437
- Maximum number of files in a single folder: 65,534

NTFS:

- Maximum disk size: 256 terabytes
- Maximum file size: 256 terabytes
- Maximum number of files on disk: 4,294,967,295
- Maximum number of files in a single folder: 4,294,967,295

Improvements

As a next step we can consider abstracting away the file system operations in both the utility program and the web app to utilize either a distributed file system [2] over the network or a blob/object store such as S3 or Microsoft Azure Block storage.

We may also want to implement more rigorous checks to validate that the files indeed are of the correct type (MIME-type) checks, and are within reasonable file sizes, say in the handful of MBs.

Perhaps adding document tagging, summaries and a search bar would make it easier to use.

A history page to show all available cached versions of a doc and how often it changes would be a good indicator for what files to update more frequently.

References

[0] Task Scheduler for developers
[1] Max files per directory on NTFS vol vs FAT32
[2] Comparison of distributed file systems

Appendix I

Utility program is placed in the data output path as the default behaviour is to write the files in the same directory it is located.

The built output CLI program is to be located within the ClientApp/public/Data folder of the web application codebase.

```
.\ChemiCleanFiles.exe -h

ChemiCleanFiles: Refresh the DB

Usage:
  ChemiCleanFiles [options]

Options:
  --output-dir-path <output-dir-path>    The output path
  --version                              Show version information
  -?, -h, --help                         Show help and usage information
```

Appendix II

The SQL database must have the following structure of the tables.

```
CREATE TABLE [dbo].[tblProduct] (
    [ProductName]  NVARCHAR (250) NOT NULL,
    [SupplierName] NVARCHAR (250) NULL,
    [Url]          NVARCHAR (300) NOT NULL,
    [UserName]     NVARCHAR (50)  NULL,
    [Password]     NVARCHAR (50)  NULL,
    [ContentHash]  NVARCHAR (MAX) NULL,
    [Updated]      DATETIME2 (7)  NULL,
    [Uri]          NVARCHAR (MAX) NULL,
    [Id]           INT            IDENTITY (1, 1) NOT NULL,
    CONSTRAINT [PK_tblProduct] PRIMARY KEY CLUSTERED ([Id] ASC)
);
```