URL Shortener System Design

Author: Chandrateja

Date: Jan 9 2025

Repository: https://github.com/chandrateja5227/UrlShortner-App

LinkedIn: https://www.linkedin.com/in/chandra-teja-6782a2225/

1. System Overview

The URL Shortener is a web application that allows users to convert long URLs into shorter, more manageable links. The system is designed with a microservices architecture, leveraging modern web technologies to provide a robust and scalable solution.

2. Scale and Capacity Planning

2.1 Traffic Estimation

- Daily URL Shortening Requests: 10 million
- Request Rate:
  - Per Day: 10,000,000 requests
  - Per Second: ~115 requests/second (10,000,000 / 86,400 seconds)

2.2 Storage Estimation

- Projection Period: 5 years
- Total Expected URLs: 18 billion (10 million * 365 * 5)

2.3 Short Code Generation Strategy

Character Set

- Allowed Characters: 62 characters
  - Uppercase Letters: A-Z (26 characters)
  - Lowercase Letters: a-z (26 characters)
  - Digits: 0-9 (10 characters)

Short Code Length Calculation

- Combinations with 6-character code: 62^6 = 68,719,476,736 (68 billion)
  - Provides ample unique combinations for 18 billion URLs
  - Allows for potential collisions and regeneration attempts

2.4 URL Metadata Estimation

- Assuming each URL mapping record:
  - Short Code: 6 bytes

  - Original URL: Average 100 bytes
  - Metadata: 50 bytes
  - Total per record: ~156 bytes

Storage Requirements

- 18 billion URLs * 156 bytes   2.8 TB of storage over 5 years

3. Low-Level Architecture Diagram

```
+-------------------+
|   Client Browser  |
+--------+----------+
         |
         v
+-------------------+
|   Vue.js Frontend |
|   Components:     |
|   - UrlShortner   |
|   - Router        |
+--------+----------+
         |
         v
+-------------------+
| .NET Core Backend |
| +---------------+ |
| | URL Controller| |
| +-------+-------+ |
|         |         |
|         v         |
| +---------------+ |
| |  URL Service  | |
| | - Generate    | |
| | - Validate    | |
| | - Map URLs    | |
| +-------+-------+ |
|         |         |
|    +----+----+    |
|    |         |    |
|    v         v    |
+-------------------+
| Caching Layer     |
| +---------------+ |
| | Redis Cache   | |
| | - Check URL   | |
| | - Store URL   | |
| +-------+-------+ |
|         |         |
|         v         |
+-------------------+
| Data Storage      |
| +---------------+ |
| |  Apache Solr  | |
| | - Persistent  | |
| | - Indexing    | |
| +---------------+ |
+-------------------+
```

4. Short Code Generation Algorithm

4.1 Unique Code Generation Approach

```
private string GenerateShortCode(string longUrl, int attempt)
{
    // Inputs to ensure uniqueness:
    // 1. Original Long URL
    // 2. Attempt Number
    // 3. Current Timestamp
    var input = $"{longUrl}_{attempt}_{DateTime.UtcNow.Ticks}";

    // Use SHA-256 for consistent, pseudo-random generation
    using (var sha256 = SHA256.Create())
    {
        var hash = sha256.ComputeHash(Encoding.UTF8.GetBytes(input));
        var shortCode = new StringBuilder(ShortCodeLength);

        // Map hash bytes to allowed characters
        for (int i = 0; i < ShortCodeLength; i++)
        {
            shortCode.Append(AllowedChars[hash[i] % AllowedChars.Length]);
        }

        return shortCode.ToString();
    }
}
```

4.2 Collision Handling

- Maximum Regeneration Attempts: 3
- Strategies:
    1. Check cache for existing short code
    2. Verify with search service
    3. Regenerate with incremented attempt number

5. Component Interactions

5.1 Frontend (Vue.js)

- `UrlShortner.vue`: Primary component for URL shortening
- Responsibilities:
  - Capture long URL input
  - Send URL to backend API
  - Display shortened URL
  - Handle user interactions

5.2 Backend (.NET Core)

URL Controller

- Endpoint: `/st/shorten`
- Methods:
  - `ShortenUrl(string longUrl)`
  - `RedirectToLongUrl(string shortCode)`

URL Service

- Core business logic for URL management
- Key operations:
  1. Validate input URL
  2. Generate unique short URL
  - Uses `GenerateShortCode` method which is regenetate sh256 hash from long url+time+coillion index to ensure no duplicates exist and takes the first 6char convert to the 62 base encoding using allowed characters
  - Ensures uniqueness
  3. Manage URL mappings

5.3 Caching Layer (Redis)

- Interface: `ICacheService`
- Responsibilities:
  - Quick URL lookup
  - Reduce database load
  - Caching strategies:
    - ∗ Cache short URL mappings
    - ∗ Set expiration policies

5.4 Data Storage (Apache Solr)

- Interface: `ISearchService`
- Responsibilities:
  - Persistent URL storage

    - Indexing URL metadata
    - Support complex queries
    - Distributed data management

6. Performance Optimization

6.1 Caching Strategy

- Cache Expiry: 24 hours
- Key Structure: `url:{shortCode}`
- Reduces database load
- Improves response time

6.2 Click Tracking (currenlty we are storing the info but UI is not implemented)

- Increment click count on each URL access
- Supports analytics and usage monitoring

7. Scalability Considerations(Based on research)

7.1 Horizontal Scaling

- Stateless backend design
- Distributed caching with Redis
- Solr for scalable data storage

7.2 Load Distribution

- Expected Peak: 115 requests/second
- Recommended Minimum:
    - 3-4 backend instances
    - Distributed Redis cluster
    - Multiple Solr shards

8. Future Scaling Strategies

- Implement consistent hashing
- Advanced sharding techniques

9. Future Improvements

- User authentication
- analytics UI
- Custom URL aliases
- Link expiration

Note on Cloud deployment

- Deployed in free tier aws server using the docker containerization approach, currently it not up and running due to resource limitations.