



Experiment 1

Student Name: Jetavya Singh Naruka

UID: 23BCS11171

Branch: BE-CSE

Section: KRG_3B

Semester: 6th

Date of performance: 06-01-2026

Subject name: System Design

Subject code: 23CSH-314

Aim : To study and design a scalable URL Shortener system by analyzing its functional requirements, non-functional requirements, API design, database schema, and low-level design approaches including counter-based short URL generation for distributed systems.

Objective :-

1. To understand the working of a URL shortening service.
2. To design REST APIs for URL creation and redirection.
3. To identify suitable database and server choices.
4. To analyze different short URL generation techniques.
5. To understand scalability challenges and their solutions.

Problem Statement : Design a URL Shortener system that converts long URLs into short URLs and redirects users from short URLs to original long URLs with **low latency, high availability, and scalability**.

Requirements :-

A. Functional Requirements

1. Generate a short URL from a long URL.
2. Should provide unique short URL.
3. Support custom URLs (optional).
4. Support URL expiration (default and custom).

5. Redirect short URL to the original long URL.
6. User registration and login using REST APIs.

B. Non-Functional Requirements :-

- **Low Latency:** URL creation and redirection < 200 ms.
- **Scalability:** 1. 100M active Users.
2. 1B URL shortening capacity.
- **Availability:** System should be available 24×7.

System Entities

1. **User**
2. **Long URL**
3. **Short URL**

API Design:-

1. **POST API call :** For providing the Long URL to Server.
API body : {
 URL : “//Long_URL” – Insert Database;
}
Response : Short URL
2. **GET API call :** For retrieving the actual Long URL using the Short URL from Database.

High Level Design (HLD)

Components:

1. Client (Browser / App)
2. Server (Business Logic)
3. Database (Storage)

Flow:



Workflow

1. Short URL Generation

- i) Client sends long URL request.
- ii) Server generates short URL.
- iii) Server stores long URL + short URL in database.
- iv) Server responds with short URL.

2. Redirection

- i. Client enters short URL.
- ii. Server validates short URL from database.
- iii. Server redirects to original long URL.

Database Design

Table T1 – User Metadata

Column	Description
User_Id	Primary key
Name	User name
Email	Unique
Password	Encrypted
Created_at	Timestamp

Table T2-URL Table:-

Column	Description
Id	Primary key
ShortURL	Unique
LongURL	Original URL
CustomeURL	Optional
Expiration_date	Expiry Time
User_Id	Foreign key

Low Level Design (LLD) :-

Approach 01: Hashing

(Encryption)

Method:

Short_URL = encrypt(Long_URL)

Algorithms: MD5, SHA-1, Base64

Problems

1. Long hash length (not short).
2. High collision probability.
3. Multiple DB lookups.
4. High latency at scale.

Not suitable for large-scale systems

Approach 02: Counter-Based Approach (Recommended)

Logic:

1. Generate a global counter value.
2. Convert counter to Base62.
3. Use Base62 value as short URL.

Example:

Counter = 10000

Base62 = 2bi

Short URL = short.ly/2bi

Advantages

- No collision
- Fast generation
- Predictable length

Scaling Challenges & Solutions

Problem 1: Monolithic Server

- Cannot handle 100M users.

Solution: Horizontal Scaling (Multiple Servers)

Problem 2: Distributed Counter Conflict

- Multiple servers may generate same counter.

Solution:

Use **Global Counter in Cache (Redis)** with atomic increment.

Type of Database Used

Recommended

- **NoSQL (Key-Value Store)**
 - Redis (Cache)
 - DynamoDB / Cassandra (Persistent)

Reason:

- Fast read/write
- High availability
- Easy horizontal scaling

Type of Server

- Stateless REST API Server
- Deployed behind Load Balancer
- Performs:
 - URL validation
 - Counter fetch
 - Base62 conversion
 - DB read/write

Result :-

A scalable and highly available **URL Shortener System** was successfully designed using **counter-based Base62 encoding**, REST APIs, NoSQL database, and distributed architecture principles.