DEPARTMENT OF
COMPUTER SCIENCE &ENGINEERING
Discover. Learn. Empower.

Experiment 1

Student Name: Vishal Saini
UID: 23BCS10163
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. Support custom URLs (optional).
3. Support URL expiration (default and custom).
4. Redirect short URL to the original long URL.
5. User registration and login using REST APIs.

B. Non-Functional Requirements

1. **Low Latency:** URL creation and redirection < 200 ms.
2. **Scalability:**
    - 100M daily active users
    - 1B shortened URLs
3. **Availability:** System should be available 24×7.
4. **Uniqueness:** Each short URL must be unique.
5. **CAP Theorem:**
    - Availability > Consistency
    - Eventual Consistency model.

System Entities

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

API Design

1. Create Short URL

Endpoint: POST /v1/url

Request Body:

```
{
  "longURL": "string",
  "customURL": "string (optional)",
  "expirationDate": "date"
}
```

Response:

```
{
  "shortURL": "https://short.ly/2bi"
}
```

2. Redirect Short URL

Endpoint: GET /v1/url/{shortURL}

Action: Redirects to original long URL.

3. User APIs

- POST /v1/register
- POST /v1/login

High Level Design (HLD)

Components:

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

Flow:
Client → Server → Database → Server → Client

Workflow

1. Short URL Generation

1. Client sends long URL request.
2. Server generates short URL.
3. Server stores long URL + short URL in database.
4. Server responds with short URL.

2. Redirection

1. Client enters short URL.
2. Server validates short URL from database.

3. 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
customURL	Optional
expirationDate	Expiry Time
user_id	Foreign Key

Low Level Design (LLD)

Approach 01: Hashing (Encryption)

Method:
shortURL = encrypt(longURL)

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.

CU
CHANDIGARH UNIVERSITY

DEPARTMENT OF

COMPUTER SCIENCE &ENGINEERING