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

EXPERIMENT NO: 1

Student Name: AYUSH PREM
UID: 23BCS10482
Branch: BE CSE
Section/Group: KRG 1B
Semester:6th
Date of Performance:10/01/ 2026
Subject Name: SYSTEM DESIGN
Subject Code: 23CSH-314

Aim: To design and document a scalable **URL Shortener System** by defining its functional requirements, non-functional requirements, API design, database schema, high-level and low-level architecture.

Definition: A URL Shortener is a system that converts a long URL into a shorter, unique URL. When a user accesses the short URL, they are redirected to the original long URL.
Example:
Long URL → https://example.com/articles/system-design/url-shortener
Short URL → https://short.ly/ABC123

Need:

- Shortens long URLs, making them easier to share
- Enhances overall user experience
- Allows tracking and performance analytics
- Ideal for social media and messaging platforms
- Offers custom and time-limited links for premium users

Approach:

1. Functional Requirements
2. Non-Functional Requirements
3. API Design
4. Database Schema Design
5. High-Level Design (HLD)
6. Low-Level Design (LLD)

1. Functional Requirements

Common System Features

- User registration (Sign Up)
- User authentication (Login)

Core Functionality

A. URL Shortening

- Input: Long URL
- Output: Generated short URL

Premium User Options:

- Custom short URL aliases
- Configurable URL expiration date

B. URL Redirection

- Input: Short URL
- Output: Automatic redirection to the corresponding original URL

2. Non-Functional Requirements

User Scale

- Supports up to 100 million registered users
- Handles 1 million active URL creation requests

Queries Per Second (QPS)

- High read QPS for URL redirection
- Moderate write QPS for URL creation

Availability

- System must be operational 24×7

Consistency

- Strong consistency for short-to-long URL mappings

Performance (Latency)

- URL creation: ≤ 20 ms
- URL redirection: ≤ 20 ms

Scalability

- Designed for horizontal scaling

Uniqueness Constraints

- Each short URL must map to exactly one long URL
- A single long URL may map to the same short URL

Transactions

- Must be ACID compliant
- Dirty reads must be prevented

3. API Design

Communication Protocol

- HTTPS

Supported HTTP Methods

- GET: Retrieve data
- POST: Create new records
- PUT / PATCH: Update existing records
- DELETE: Remove records

URL Shortener APIs

1. Generate Short URL

Endpoint:

POST https://127.0.0.1/shorten

Request Body:

```
{
  "url": "LONG_URL",
  "custom_url": "optional",
  "expiry_date": "optional"
}
```

Response:

{
"short_url": "https://127.0.0.1/ABC123",
"short_code": "ABC123"
}

2. Redirect to Original URL

Endpoint:

GET https://127.0.0.1/{short_code}

Response:

{
"long_url": "LONG_URL"
}

4. Database Schema Design

Table 1: USER

Stores user-related metadata.

Field Name	Description
user_id	Unique user identifier
email	User email address
password_hash	Encrypted password
is_premium	Indicates premium subscription
created_at	Account creation timestamp

Table 2: URL_MAPPING

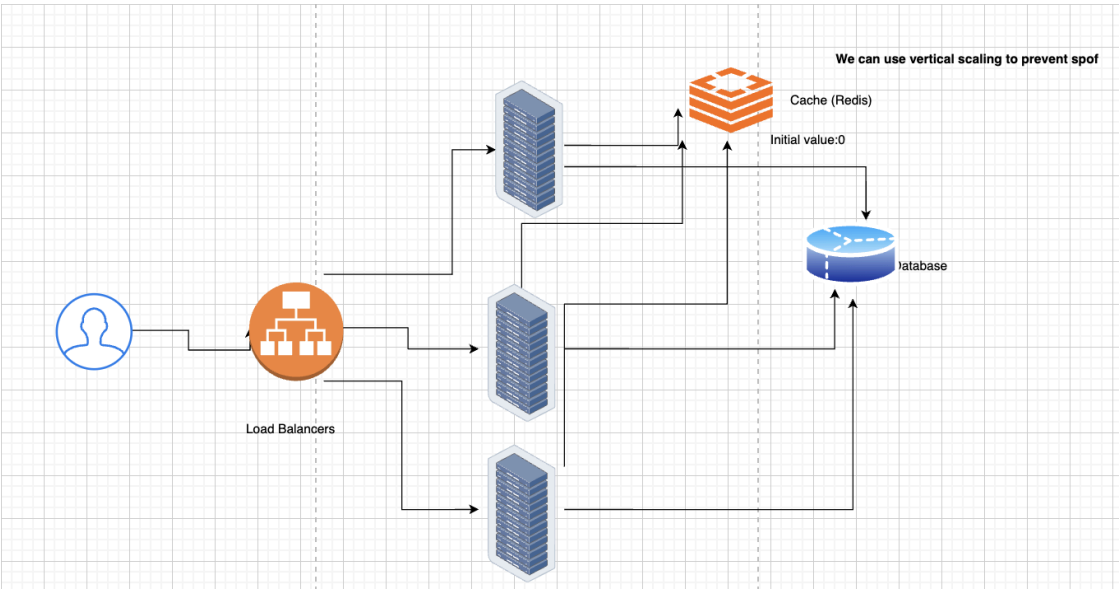
Stores mappings between long and short URLs.

Field Name	Description
id	Primary key
user_id	Foreign key referencing USER table
long_url	Original URL
short_code	Generated short code
custom_url	Optional custom alias
expiry_date	Optional expiration date
created_at	Record creation timestamp

5. High-Level Design (HLD)

The system architecture includes:

- Load Balancer
- Application Servers
- Caching Layer
- Database



6. Low-Level Design (LLD)

- Short code generation mechanism
- Cache lookup and fallback strategy
- Database read/write workflow
- URL expiration validation logic

Result: The URL Shortener System design was successfully documented with clear requirements, APIs, database schema, and architecture.

Conclusion: This experiment demonstrates how real-world scalable systems are designed by balancing functionality, performance, and reliability