EXPERIMENT NO: 1

Student Name: Shiva Gupta
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, and high-level/low-level architecture.

1. **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

2. **Need:**

   - Reduces URL length for easy sharing
   - Improves user experience
   - Enables tracking and analytics
   - Useful for social media and messaging platforms
   - Supports custom and expiring links for premium users

3. **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 Requirements

- User Sign Up
- User Login

Core Features

A. URL Shortening - Input: Long URL - Output: Short URL

Optional (Premium Users): - Custom URL support - URL expiry date

B. URL Redirection - Input: Short URL - Output: Redirect to original long URL

2. Non-Functional Requirements

- **User Scale:**
  - 100 million total users
  - 1 million active URL creation requests
- **QPS:**
  - High read QPS (redirection)
  - Moderate write QPS (URL creation)
- **Availability:** 24 × 7
- **Consistency:** Strong consistency for URL mapping
- **Latency:**
  - URL shortening ≤ 20 ms
  - URL redirection ≤ 20 ms
- **Scalability:**
  - Horizontal scaling preferred
- **Uniqueness:**
  - One short URL maps to exactly one long URL
  - Same long URL may map to the same short URL
- **Transactions:**
  - ACID compliant
  - Avoid dirty reads

3. API Design

Protocol

- HTTPS

HTTP Methods

- GET: Retrieve data
- POST: Insert data

- PUT / PATCH: Update data
- DELETE: Remove data

URL Shortener APIs

1. Create 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 Long URL

Endpoint:

GET https://127.0.0.1/{short_code}

Response:

```
{
 "long_url": "LONG_URL"
}
```

4. Database Schema Design

Table 1: USER

Stores metadata related to users.

Field Name	Description
user_id	Unique user identifier

Field Name	Description
email	User email
password_hash	Encrypted password
is_premium	Premium user flag
created_at	Account creation time

Table 2: URL_MAPPING

Stores URL mappings.

Field Name	Description
id	Primary key
user_id	Reference to USER table
long_url	Original URL
short_code	Generated short code
custom_url	Optional custom alias
expiry_date	Optional expiry
created_at	Creation timestamp

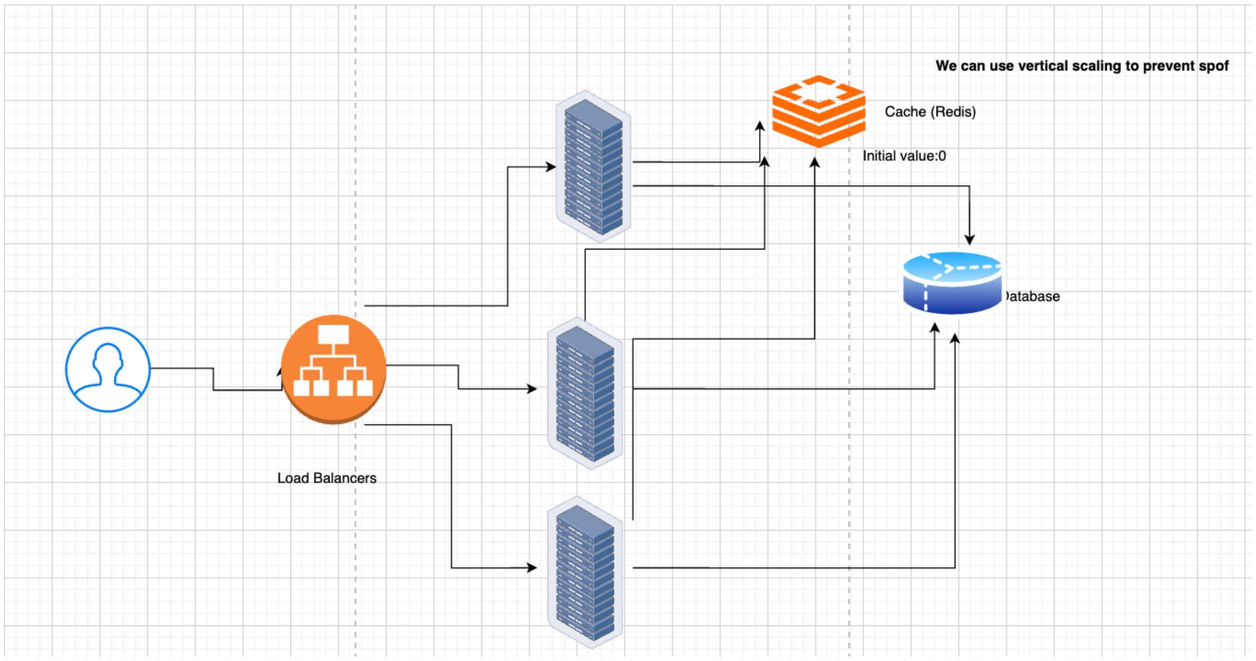
5. High-Level Design (HLD)

Refer to **Experiment 01 Draw.io Diagram** for: - Load balancer - Application servers - Cache layer - Database

6. Low-Level Design (LLD)

- Short code generation logic
- Cache lookup flow
- Database read/write flow
- URL expiry validation

7. High-Level Design (HLD)



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