Experiment 1

Student Name: Gourav Sharma **UID: 23BCS10857**
Branch: CSE **Section/Group: KRG 3-A**
Semester: 6th **Date of Performance:08/01/2026**
Subject Name: System Design **Subject Code: 23CSH-314**

1. **Aim**: To design and analyze a **URL Shortener System** that converts long URLs into short, unique URLs while ensuring high availability, scalability, low latency, and efficient redirection. The system also supports optional custom URLs, expiration dates, and user authentication.

2. **Objective**:
   - To design a system that converts long URLs into short, unique, and collision-free links.
   - To ensure high availability and scalability for handling millions of URL redirections efficiently.
   - To provide low-latency redirection for a fast and seamless user experience.
   - To support advanced features such as custom aliases, link expiration, and analytics.
   - To implement secure user authentication and access control for managing shortened URLs.

3. **Tools Used:**

   - **Python** – Backend logic implementation and URL generation algorithms.
   - **Flask** – Lightweight web framework for developing RESTful APIs.
   - **Draw.io** – Designing system architecture diagrams (HLD & LLD).

4. **System Requirements:**

   ## A. Functional Requirements
   - Create a short URL from a given long URL.
   - Support optional custom short URLs.
   - Support default and user-defined expiration dates.
   - Redirect users from short URL to the original long URL.
   - Provide REST APIs for URL creation and redirection.
   - Support user registration and login using REST APIs.

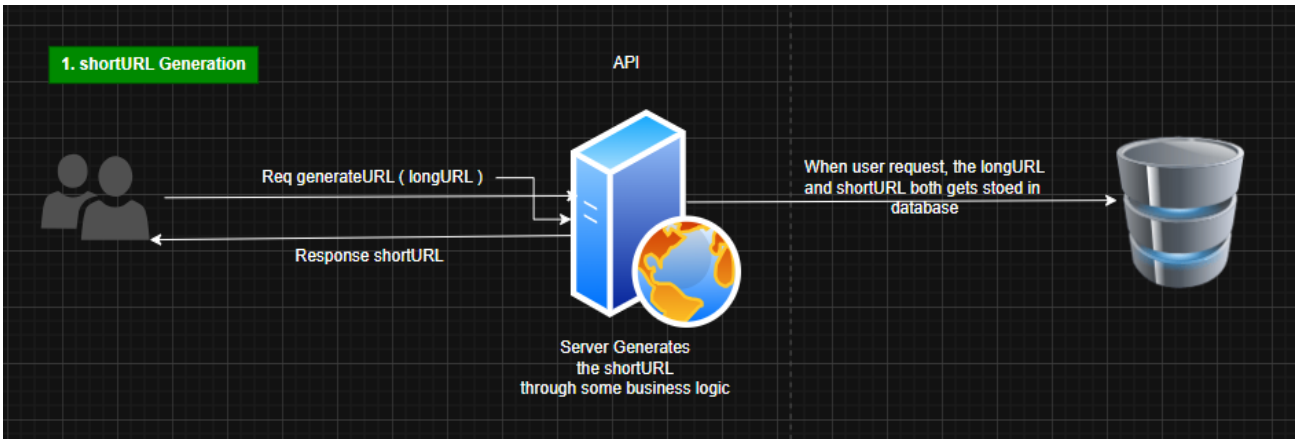
B. Non-Functional Requirements

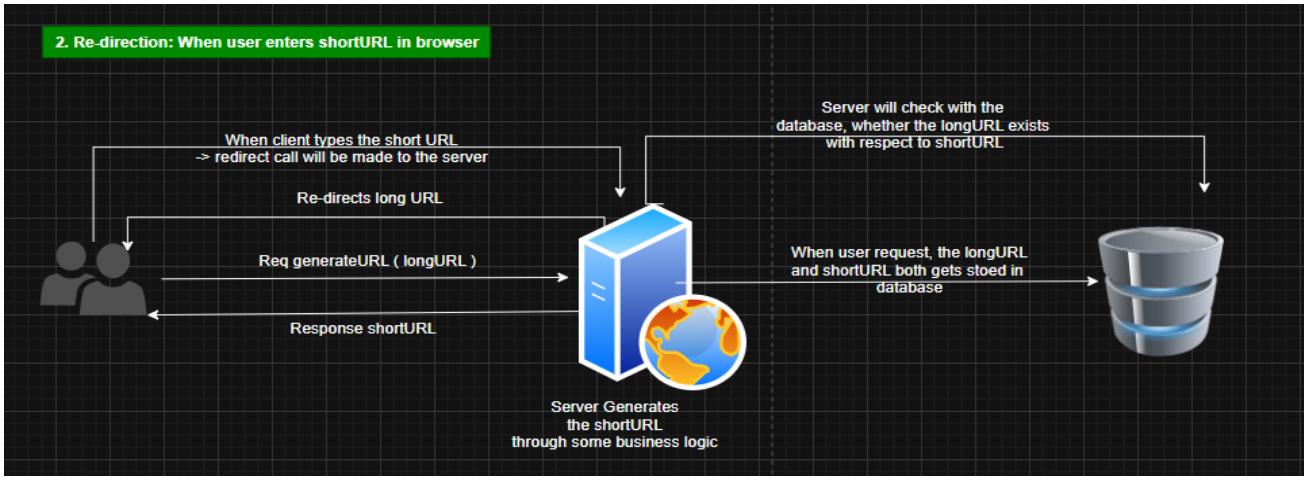
- Low latency (≤ 200 ms for URL creation and redirection).
- High scalability (100M daily active users, 1B URLs).
- High availability (24×7).
- Uniqueness of short URLs.
- High availability preferred over strict consistency (Eventual Consistency).

5. High Level Design (HLD):

The system follows a **Client–Server–Database architecture**:

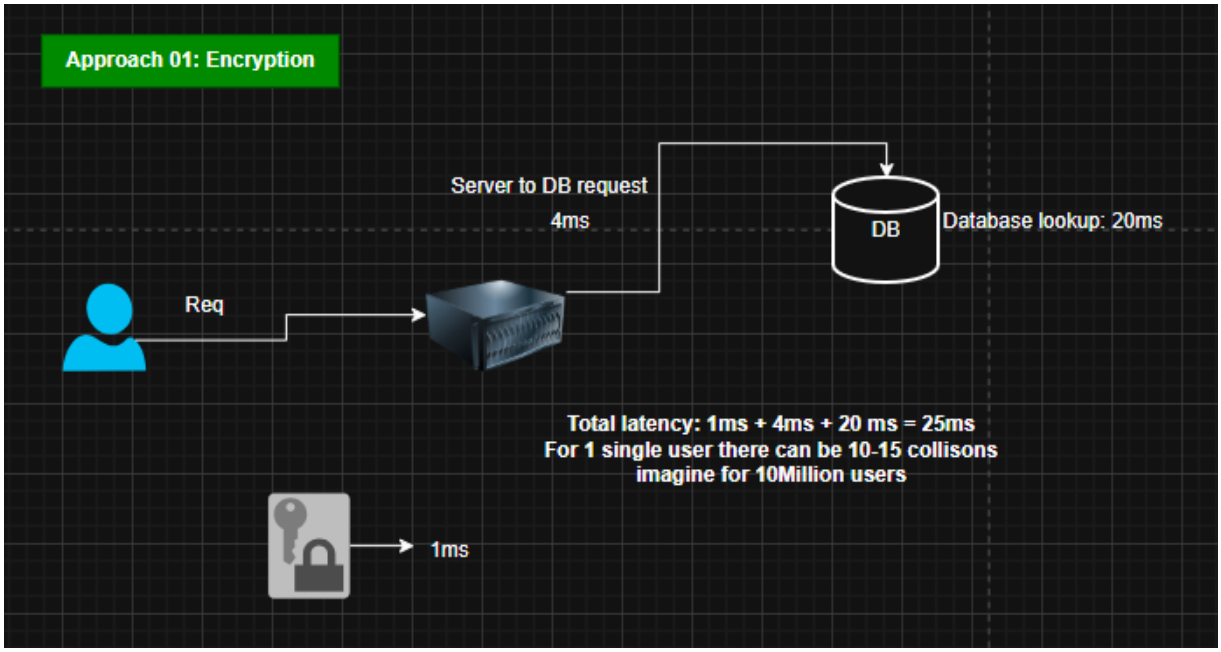
- Client sends request to generate or access short URL.
- Server processes business logic and generates short URL.
- Database stores mappings of short URL and long URL.
- On redirection, server fetches long URL and redirects the user.





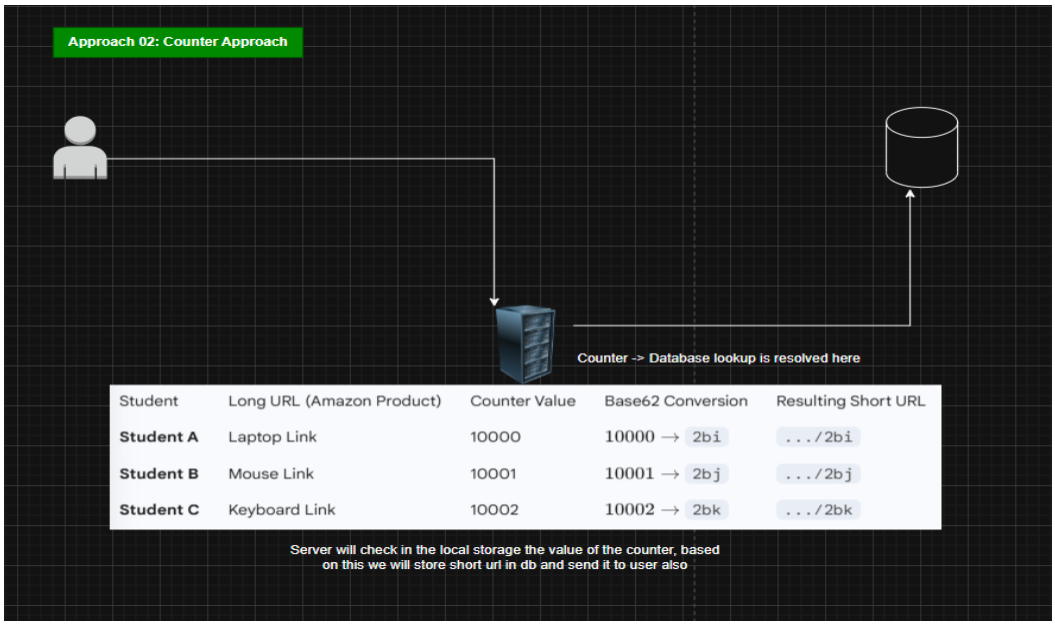
6. Low Level Design (LLD):

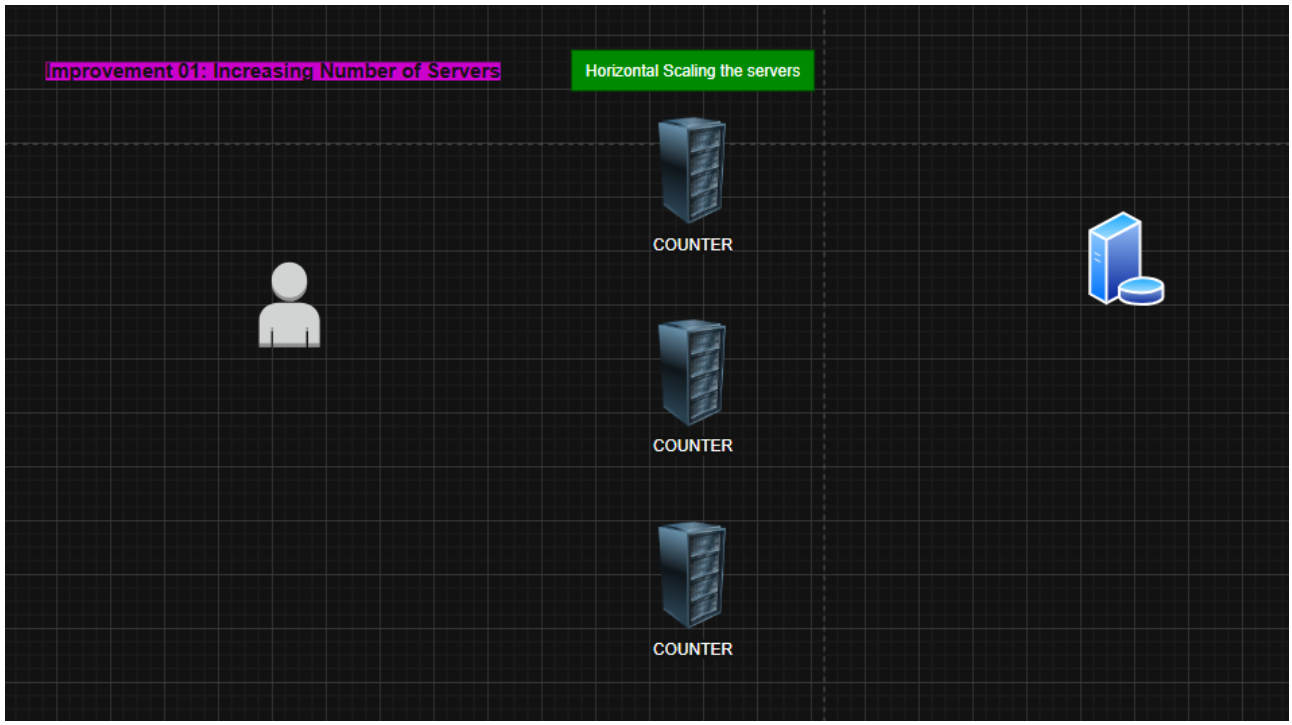
Approach 1: Encryption

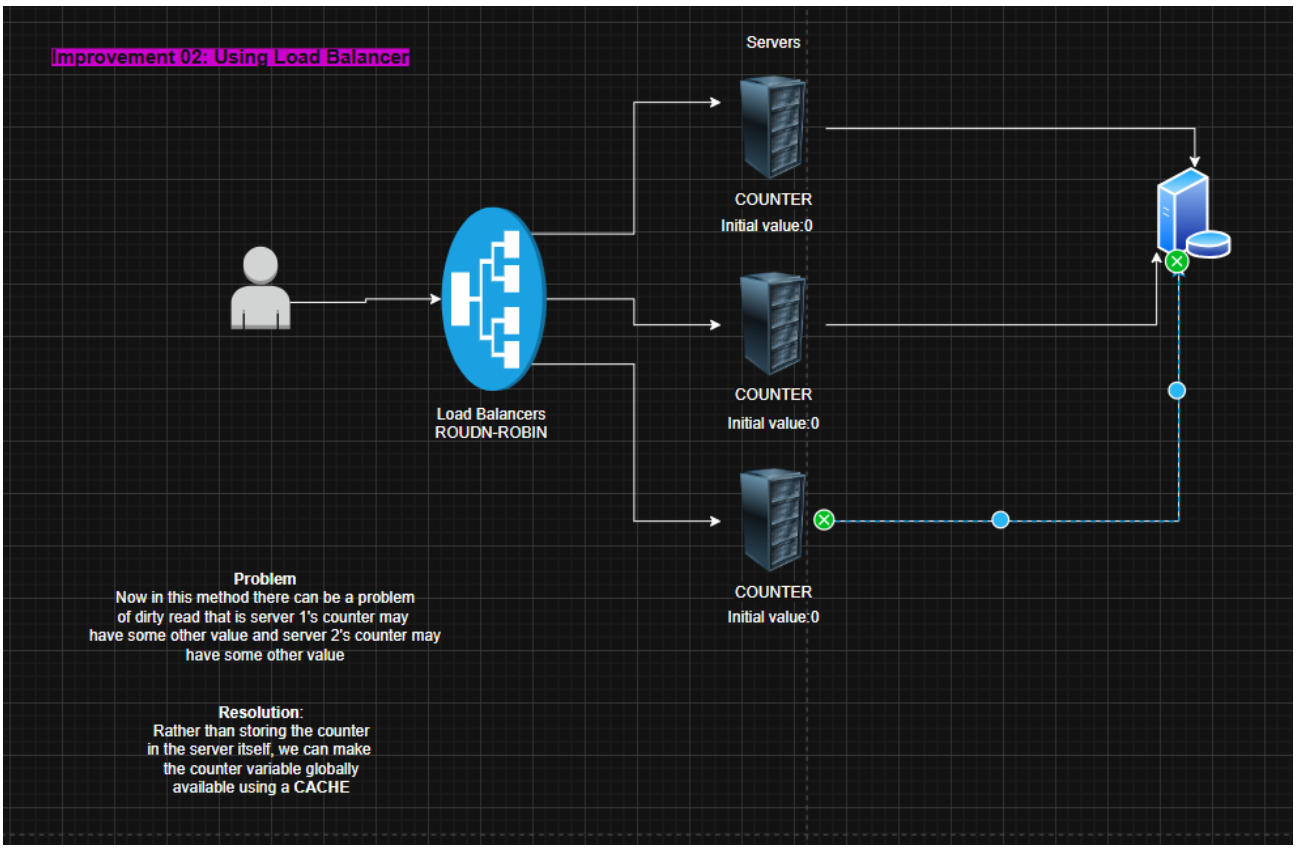


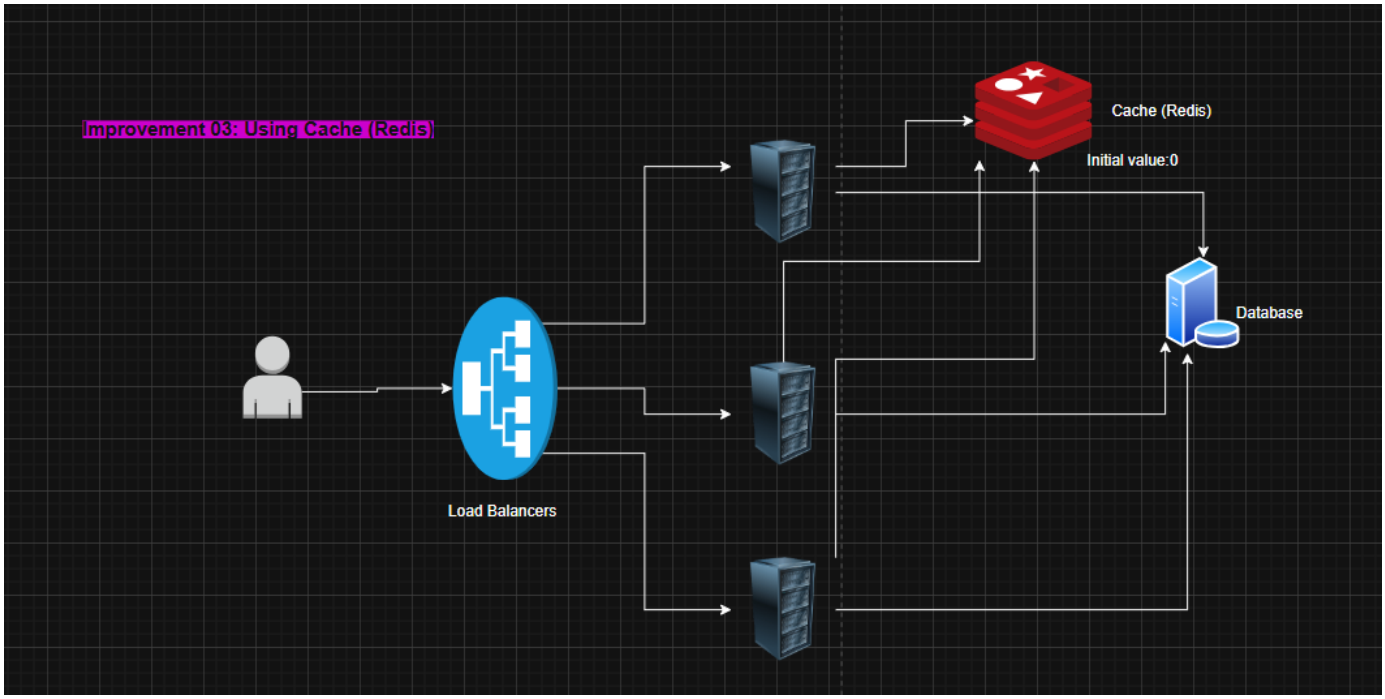
Approach 2: Counter-Based

- Uses auto-increment counter.
- Counter value converted to Base62 for short URL.
- Issue: Single counter causes scalability issues.




Improvement 01: Increasing Number of Servers
Horizontal Scaling the servers
COUNTER
COUNTER
COUNTER


Improvement 02: Using Load Balancer
Servers
COUNTER
Initial value:0
COUNTER
Initial value:0
COUNTER
Initial value:0
Load Balancers
ROUDN-ROBIN
Problem
Now in this method there can be a problem of dirty read that is server 1's counter may have some other value and server 2's counter may have some other value
Resolution:
Rather than storing the counter in the server itself, we can make the counter variable globally available using a CACHE



7. Scalability Solution

- Horizontal scaling of application servers.
- Use of Load Balancer (Round Robin).
- Centralized counter stored in Redis cache.
- Redis ensures fast access and atomic increments.
- Database stores final URL mappings.

8. Learning Outcomes (What I Have Learnt)

- Learned how to design a real-world scalable system.
- Understood REST API design principles.
- Gained knowledge of CAP theorem and eventual consistency.
- Learned multiple URL shortening techniques and their trade-offs.
- Understood horizontal scaling, caching, and load balancing.
- Learned importance of low latency and high availability systems.