Experiment No. 1

Student Name: Nagesh Kumar	**UID:** 23BCS10608
Branch: BE CSE	**Section/Group:** KRG 1A
Semester: 6th	**Subject Code:** 23CSH-314
Subject Name: System Design	**Date of Performance:** 10/01/2026

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 understand functional and non-functional requirements of a large-scale distributed system.
- To design RESTful APIs for URL creation and redirection.
- To identify core entities such as User, Short URL, and Long URL.
- To analyze CAP theorem trade-offs and apply eventual consistency.
- To design high-level and low-level architecture for a scalable URL shortener.
- To study multiple approaches for short URL generation and compare their performance.

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.

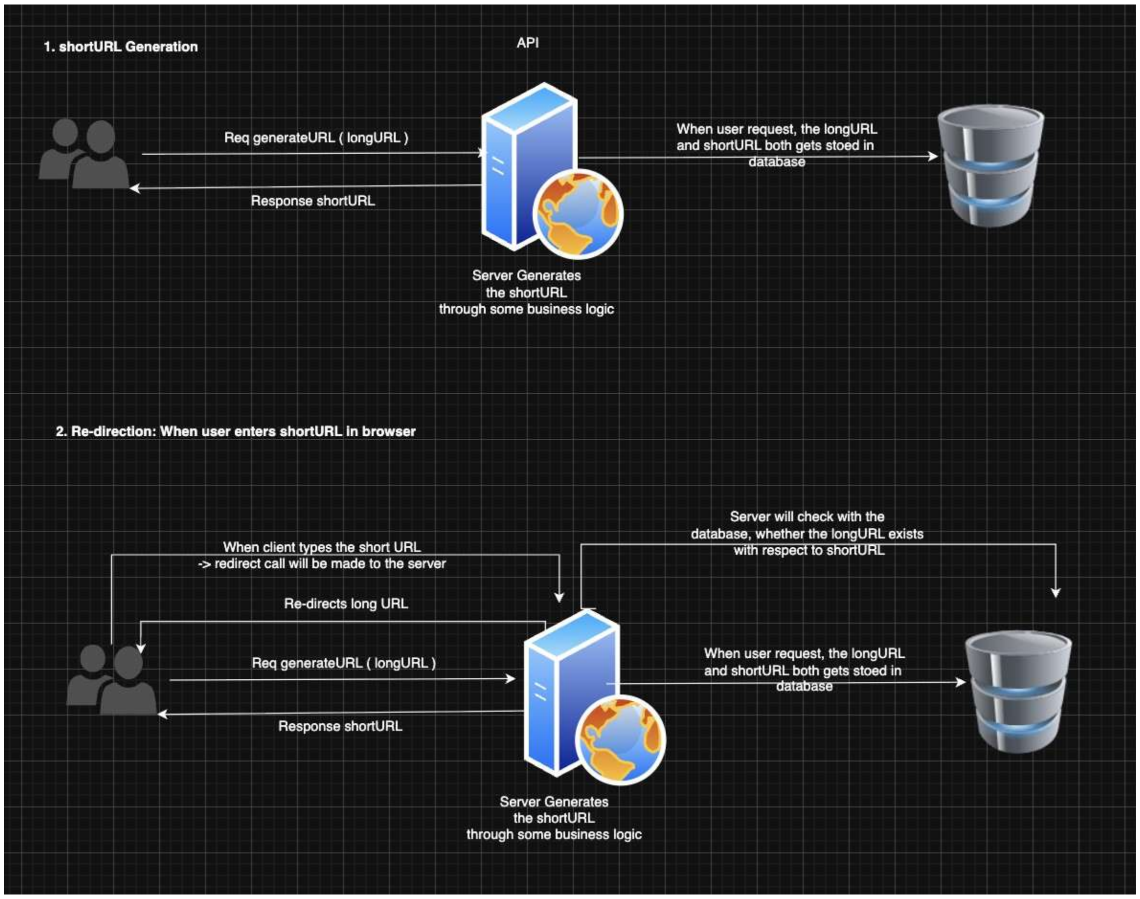
B. Non-Functional Requirements

- Low latency (≤ 20 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.
- The server processes business logic and generates a short URL.
- Database stores mappings of short URL and long URL.
- On redirection, the server fetches a long URL and redirects the user.


1. shortURL Generation
API
Req generateURL (longURL)
Response shortURL
When user request, the longURL and shortURL both gets stoed in database
Server Generates the shortURL through some business logic
2. Re-direction: When user enters shortURL in browser
Server will check with the database, whether the longURL exists with respect to shortURL
When client types the short URL -> redirect call will be made to the server
Re-directs long URL
Req generateURL (longURL)
Response shortURL
When user request, the longURL and shortURL both gets stoed in database
Server Generates the shortURL through some business logic

6. Low Level Design (LLD):

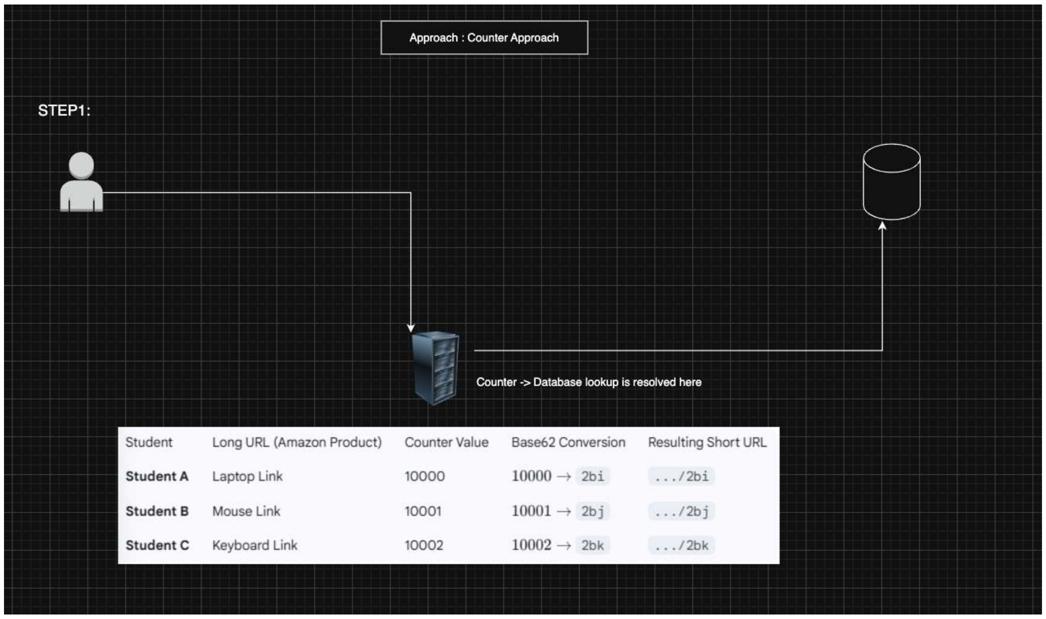
Approach : Counter-Based

- Uses an auto-increment counter.
- Counter value converted to Base62 for a short URL.

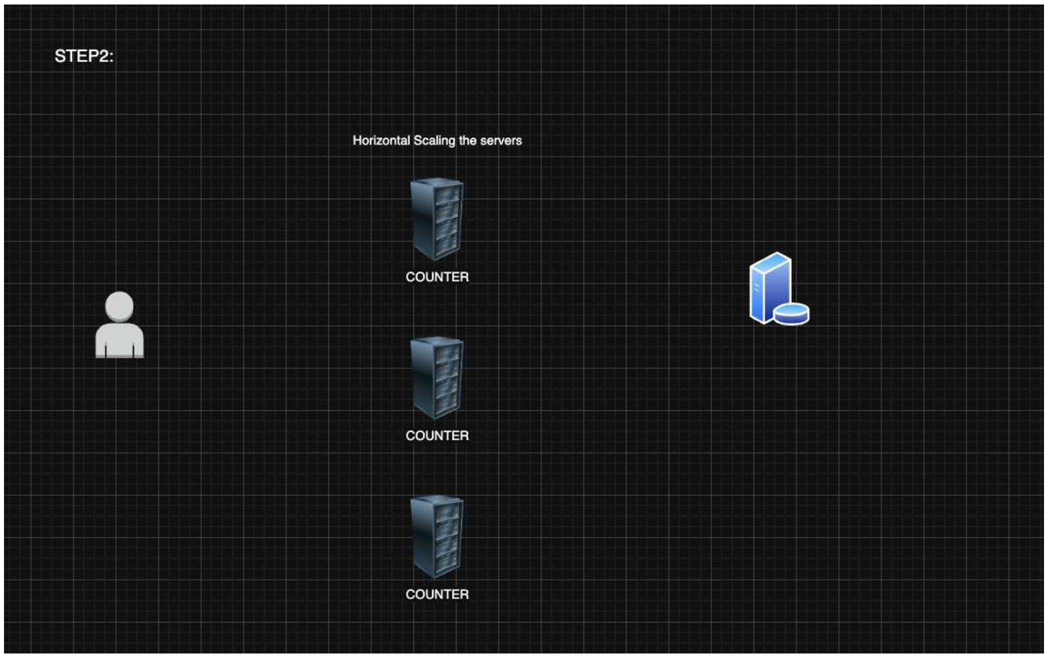
Example :

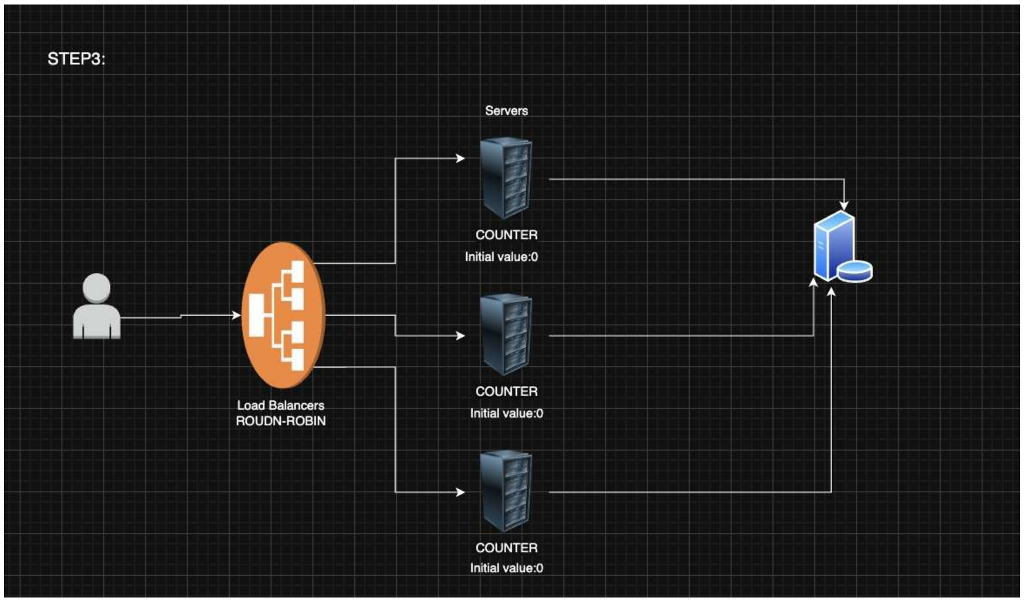
Counter: 10000 → Base62: 2bi → Short URL

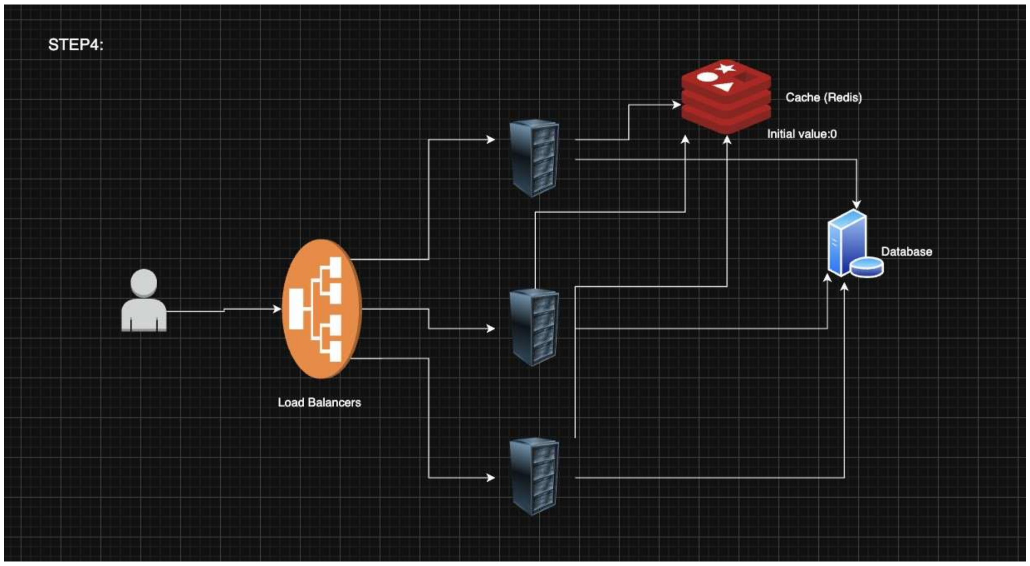
- Issue: Single counter causes scalability issues.


Approach : Counter Approach
STEP1:
Counter -> Database lookup is resolved here

Student	Long URL (Amazon Product)	Counter Value	Base62 Conversion	Resulting Short URL
Student A	Laptop Link	10000	10000 → `2bi`	`.../2bi`
Student B	Mouse Link	10001	10001 → `2bj`	`.../2bj`
Student C	Keyboard Link	10002	10002 → `2bk`	`.../2bk`







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.