DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

EXPERIMENT - 1

Student Name: Jayesh Saini
UID: 23BCS11619
Branch: BE-CSE
Section/Group: KRG-2A
Semester: 6th
Date of Performance: 11/01/2026
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 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:

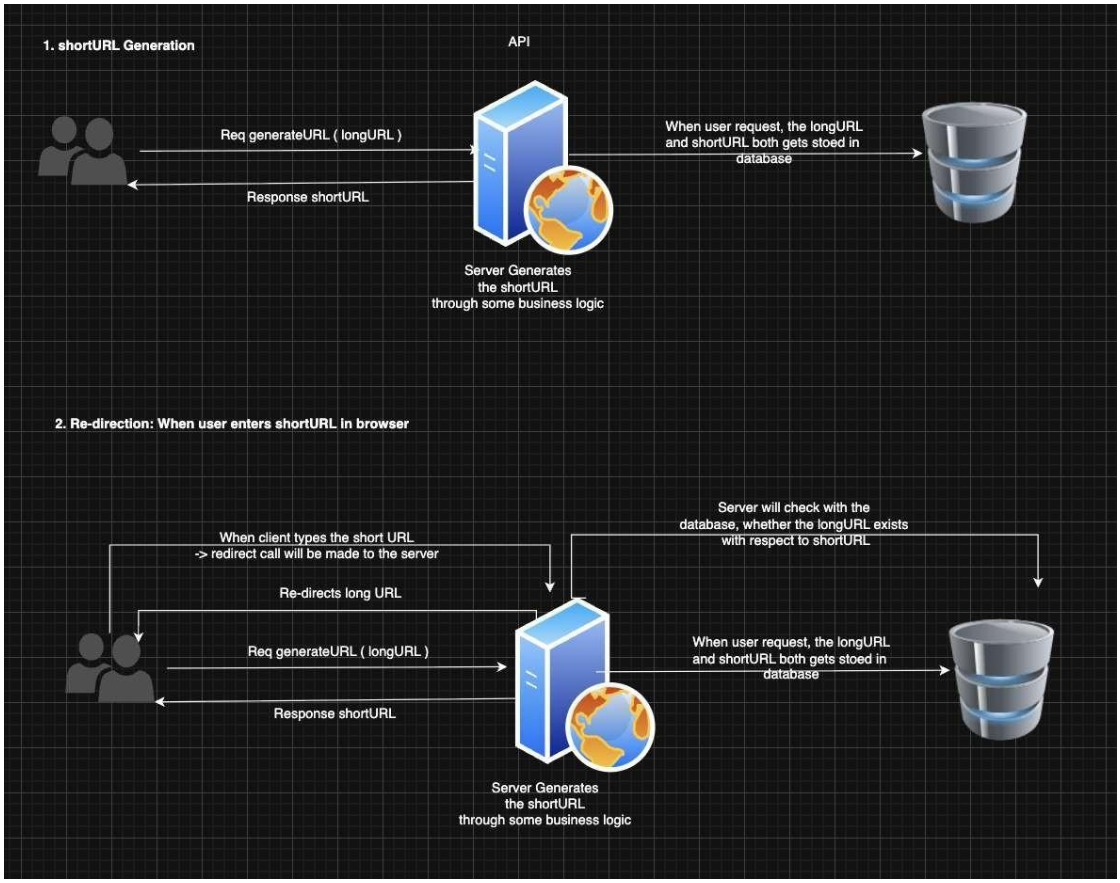
A. Functional Requirements

- 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.

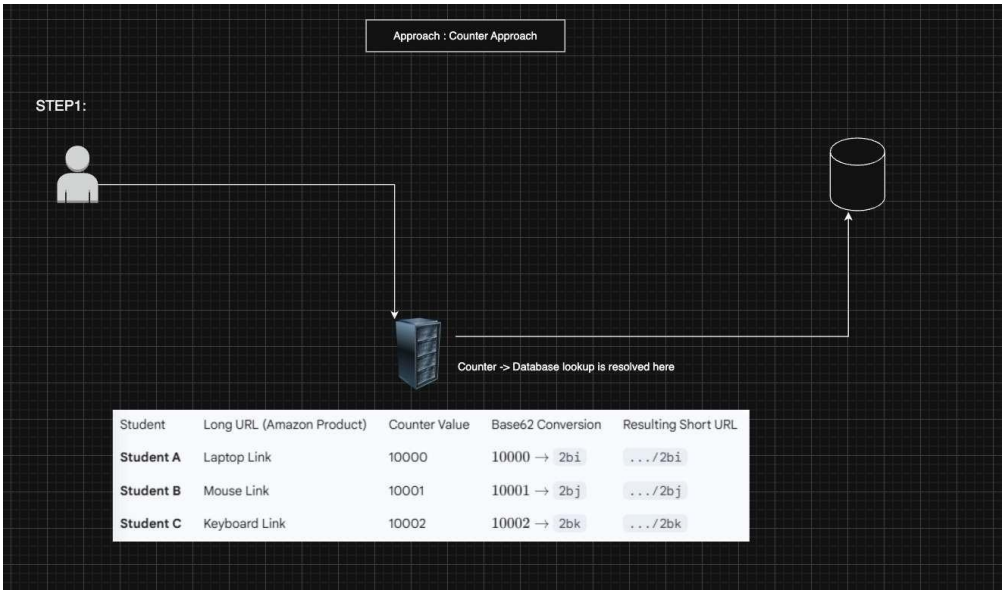
6. Low Level Design (LLD):

Approach : Counter-Based

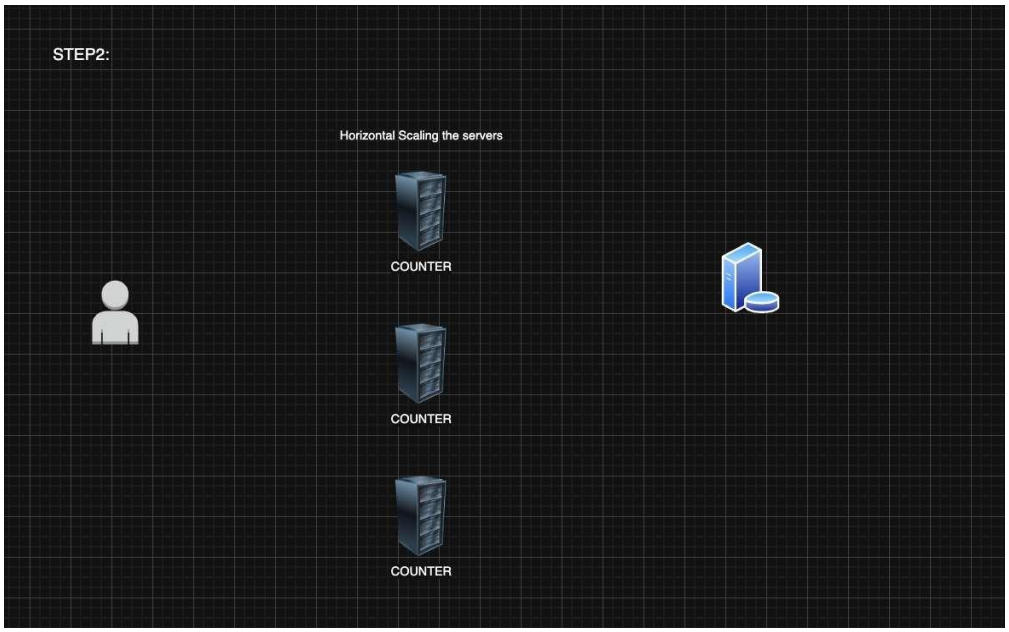
- Uses an auto-increment counter.
- Counter value converted to Base62 for a short URL. **Example** :

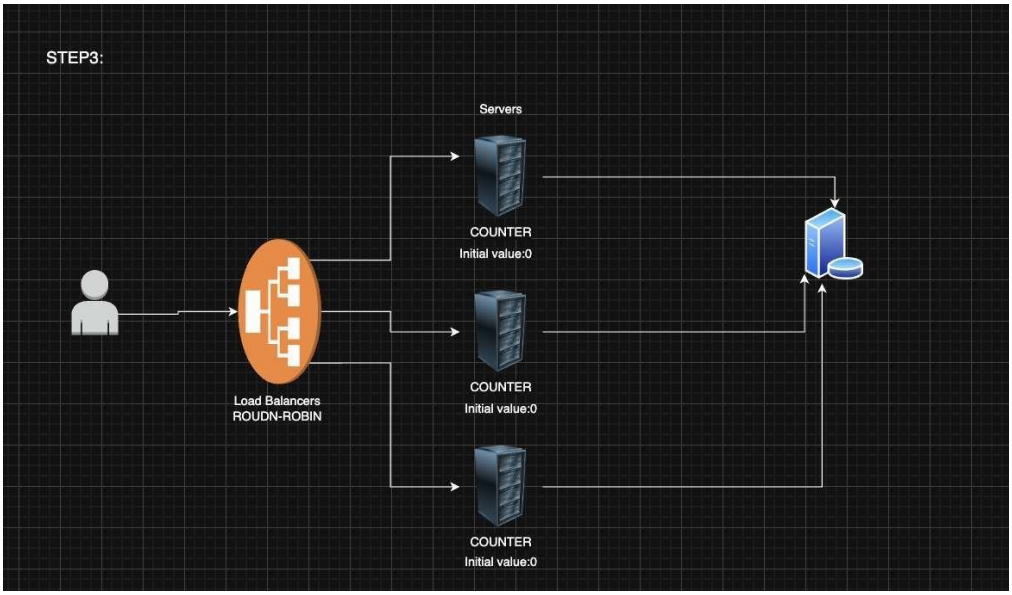
Counter: 10000 → Base62: 2bi → Short URL •

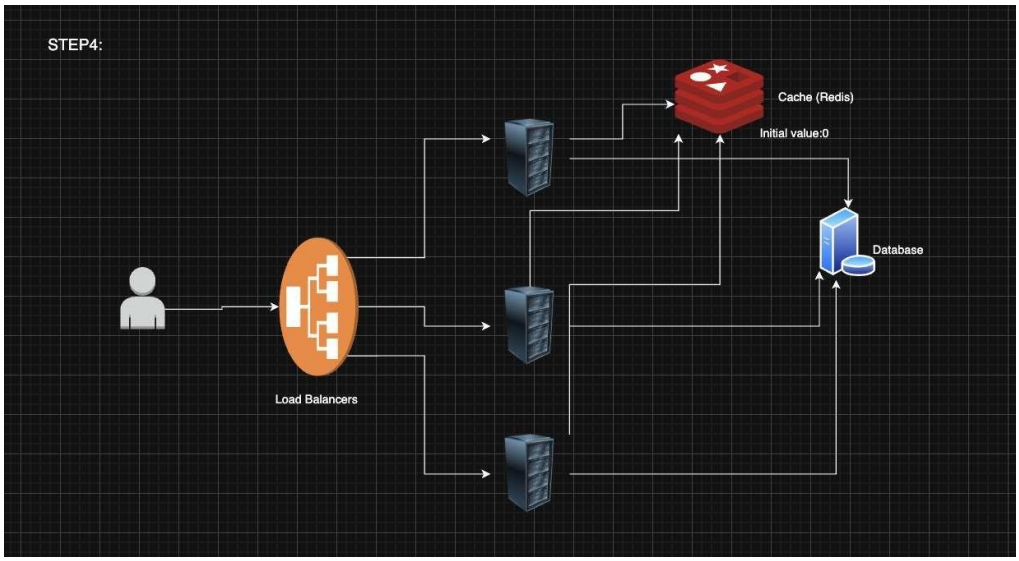
Issue: Single counter causes scalability issues.



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


STEP2:
Horizontal Scaling the servers
COUNTER
COUNTER
COUNTER


STEP3:
Servers
COUNTER
Initial value:0
COUNTER
Initial value:0
COUNTER
Initial value:0
Load Balancers
ROUDN-ROBIN


STEP4:
Cache (Redis)
Initial value:0
Database
Load Balancers

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:

- Gained an understanding of how to design scalable, real-world systems.
- Learned the principles and best practices of REST API design.
- Developed knowledge of the CAP theorem and the concept of eventual consistency.
- Explored different URL shortening techniques along with their trade-offs.
- Understood the role of horizontal scaling, caching, and load balancing in system performance.
- Learned the importance of building systems with low latency and high availability.