Experiment - 1

Student Name: Yashwat Pratap Singh	**UID:** 23BCS12565
Branch: BE-CSE	**Section/Group:** KRG_2A
Semester: 6th	**Date of Performance:** 08-01-2026
Subject Name: System Design	**Subject Code:** 23CSH-314

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.

OBJECTIVES:

- To design and understand the working of URL Shortener.
- To identify Functional and Non-Functional requirements of the system
- To design a High-Level Design flow using draw.io
- To design low-level architecture for a scalable URL shortener
- To design RESTful APIs for URL shortening and redirectionTo identify core entities such as User, Short URL, and Long URL
- To analyze the trade-offs between Consistency & Availability.
- To study multiple approaches for short URL generation and compare their performance.

APPROACH:

1. Functional req
2. Non-functional req
3. API design
4. Database schema design
5. HLD of URL shortener
6. LLD of URL shortener

SYSTEM REQUIREMENTS:

Functional Requirements

- URL Shortening
  - Custom URL
  - Supports expiration date
- URL Redirection.

Non-Functional Requirements

- Low Latency - 200 ms
- Scalability: 100M daily active user & 1B URL creation per day
- Unique Shorten URL
- Availability (24 x 7 available)

API DESIGN

1. HTTPS
2. pre-defined functions:
   a. get: data retrieval
   b. put / patch: update
   c. post: to insert the data into db
   d. delete: remove the data

URL shortener system is concerned:

local host: https://127.0.0.1/shorten

app.route (/shorten) 1.

POST api call

```
http Req
{
        URL: "long url",
        custom_url :?,
        expirty date:?
}
url
https response
{
        e.g.: https://127.0.0.1/abc123
        short url:  "short URL",
        short code: abc123
}
```

2. GET (</short code>)

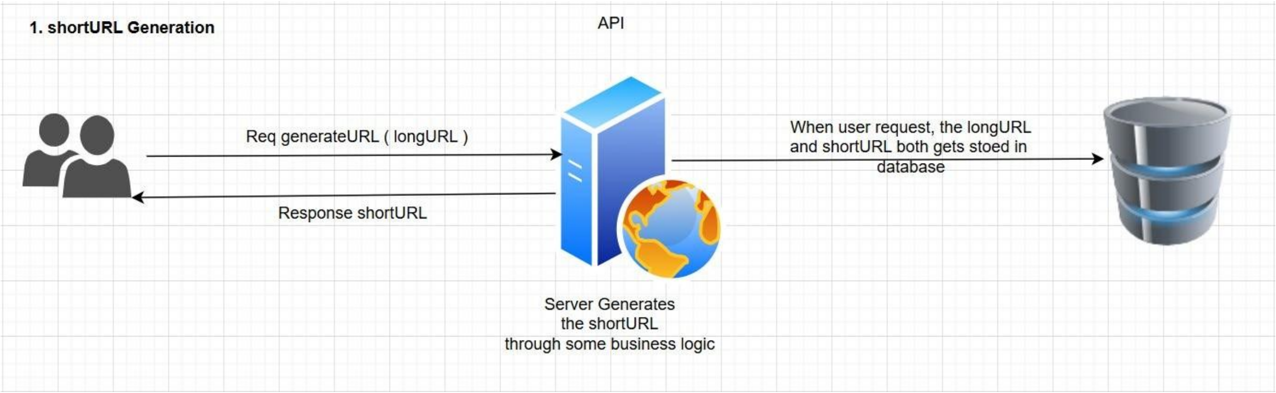
```
https response
{
long_:
}
```

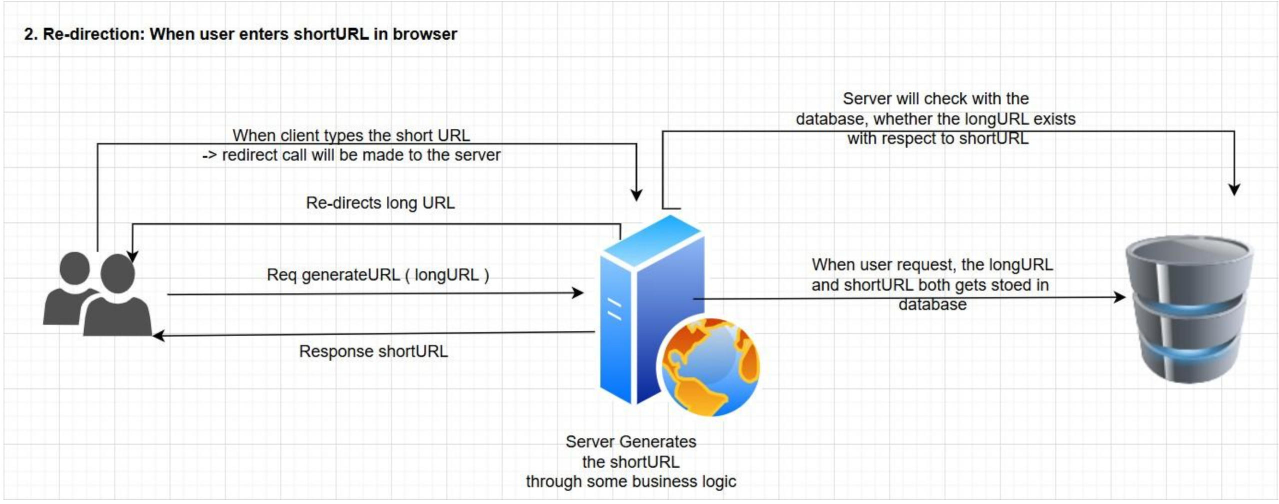
DATABASE SCHEMA DESIGN

T1 USER - (META DATA OF USER)

T2 API_MAPPING - (LONG_URL, SHORT_URL, CUSTOM_URL, EXPIRY_DATE)

HIGH-LEVEL DESIGN (HLDs):

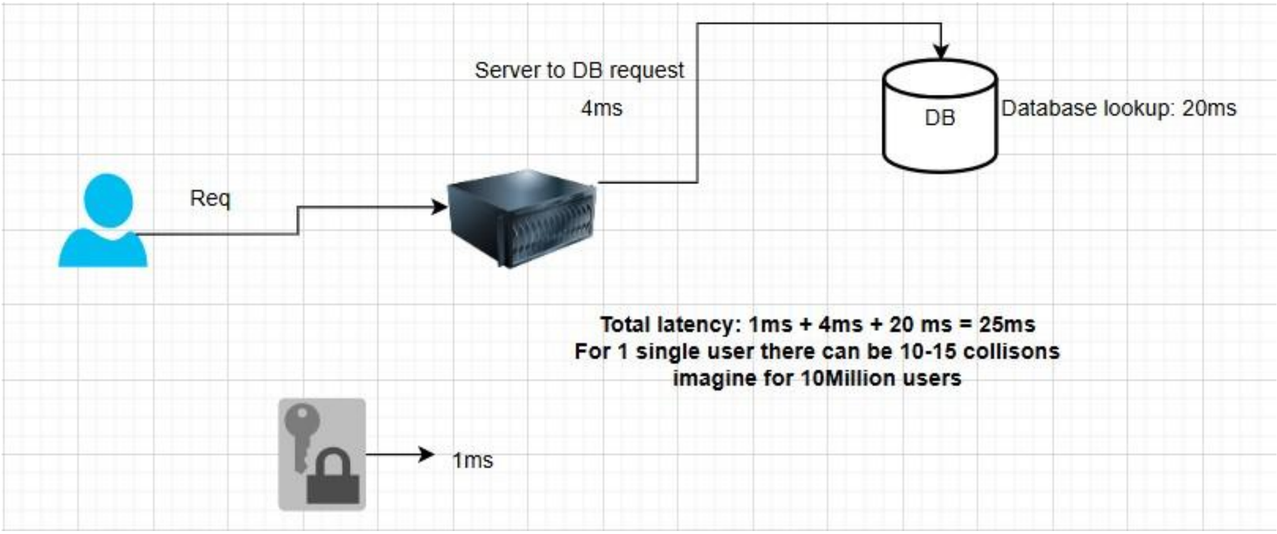

1. shortURL Generation
API
Req generateURL (longURL)
Response shortURL
When user request, the longURL and shortURL both gets stoed in database
Server Generates the shortURL through some business logic


2. Re-direction: When user enters shortURL in browser
When client types the short URL -> redirect call will be made to the server
Server will check with the database, whether the longURL exists with respect to shortURL
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

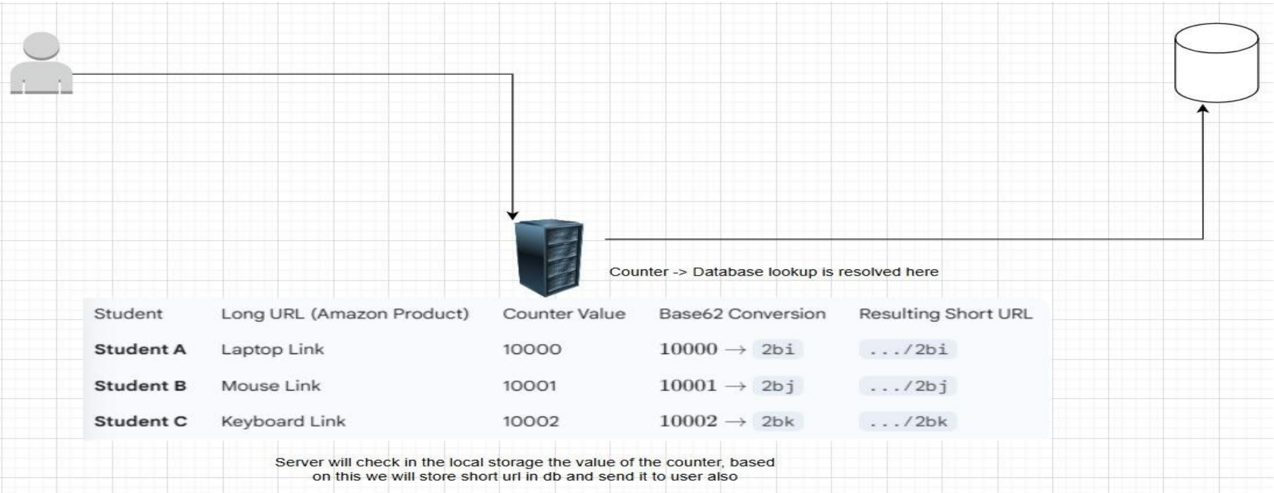
LOW-LEVEL DESIGN

Conversion of longURL into shortURL

Approach 1: Encryption


Server to DB request
4ms
DB
Database lookup: 20ms
Req
Total latency: 1ms + 4ms + 20 ms = 25ms
For 1 single user there can be 10-15 collisons
imagine for 10Million users
1ms

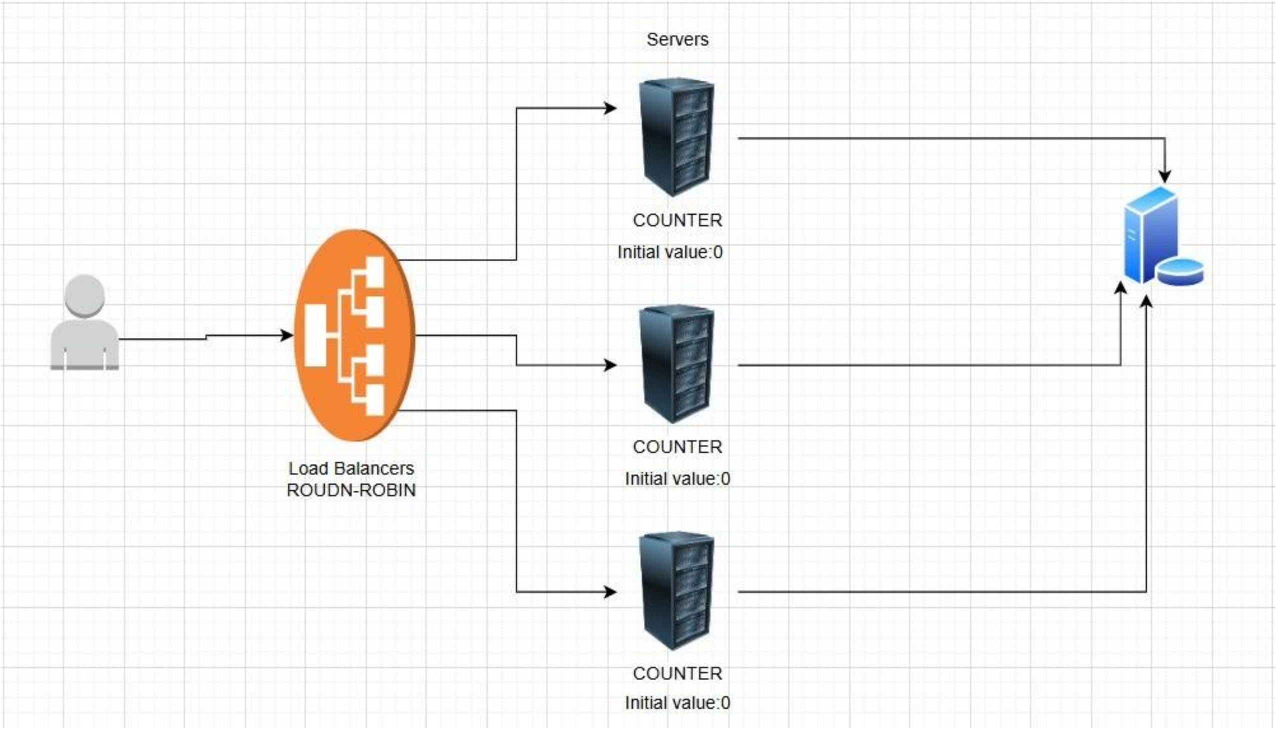
Approach 2: Count Approach



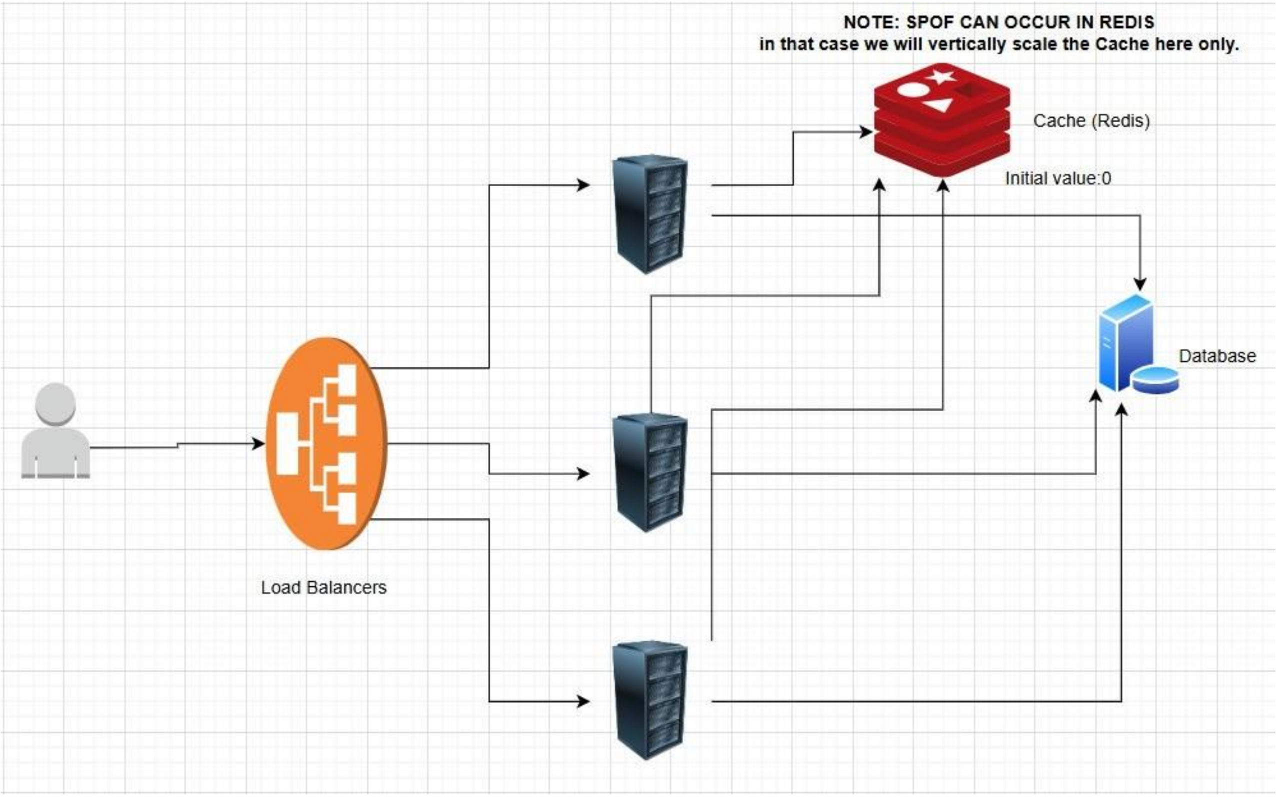
Student	Long URL (Amazon Product)	Counter Value	Base62 Conversion	Resulting Short URL
Student A	Laptop Link	10000	10000 → `2bi`	`.../2bi`
Student B	Mouse Link	10001	10001 → `2bj`	`.../2bj`
Student C	Keyboard Link	10002	10002 → `2bk`	`.../2bk`

Server will check in the local storage the value of the counter, based on this we will store short url in db and send it to user also

Horizontal Scaling



Redis Implementation


NOTE: SPOF CAN OCCUR IN REDIS
in that case we will vertically scale the Cache here only.
Cache (Redis)
Initial value:0
Database
Load Balancers

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.

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.