



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment - 1

Student Name: Himanshu Raj
Branch: BE-CSE
Semester: 6th
Subject Name: System Design

UID: 23BCS12415
Section/Group: KRG_1B
Date of Performance: 10-01-2026
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 redirection To 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 :?,
  expiry 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_:
```

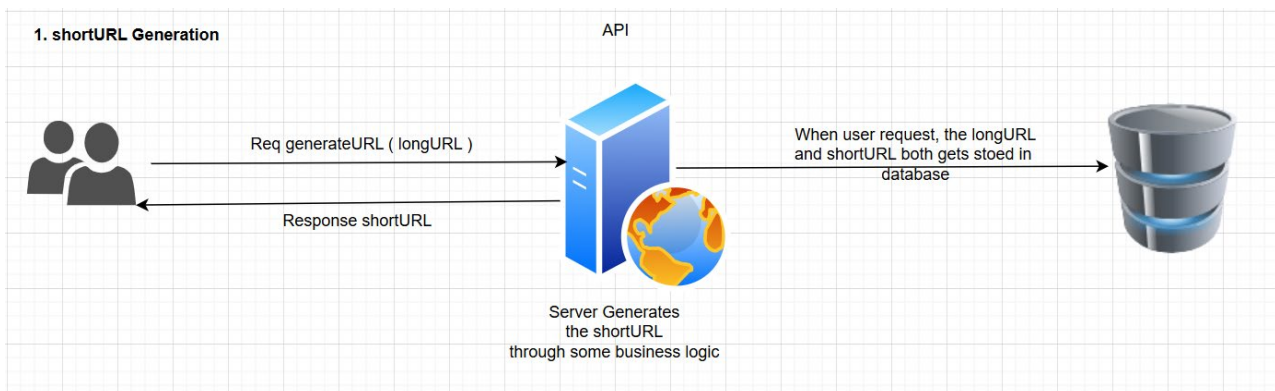
```
}
```

DATABASE SCHEMA DESIGN

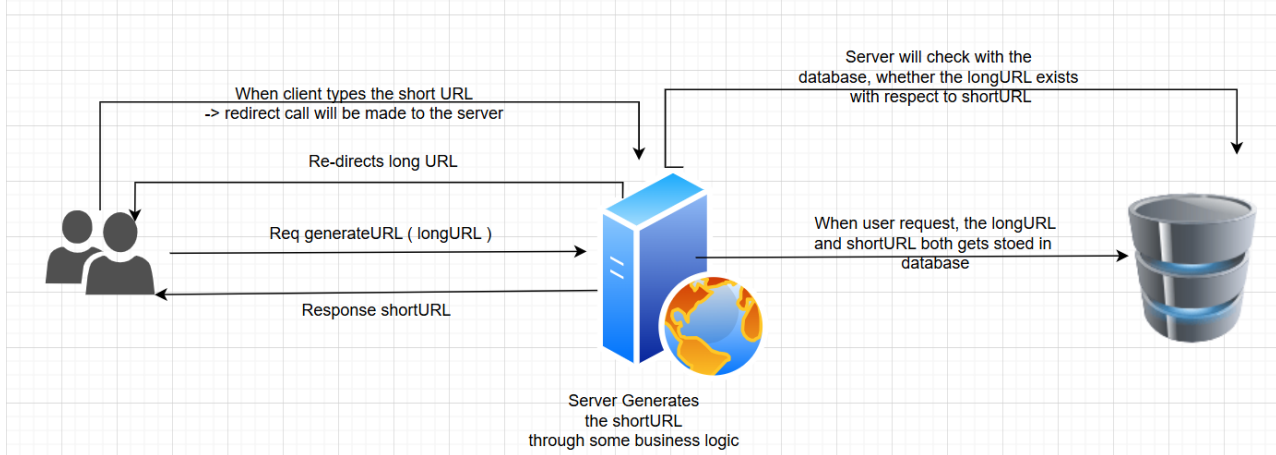
T1 USER - (META DATA OF USER)

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

HIGH-LEVEL DESIGN (HLDs):



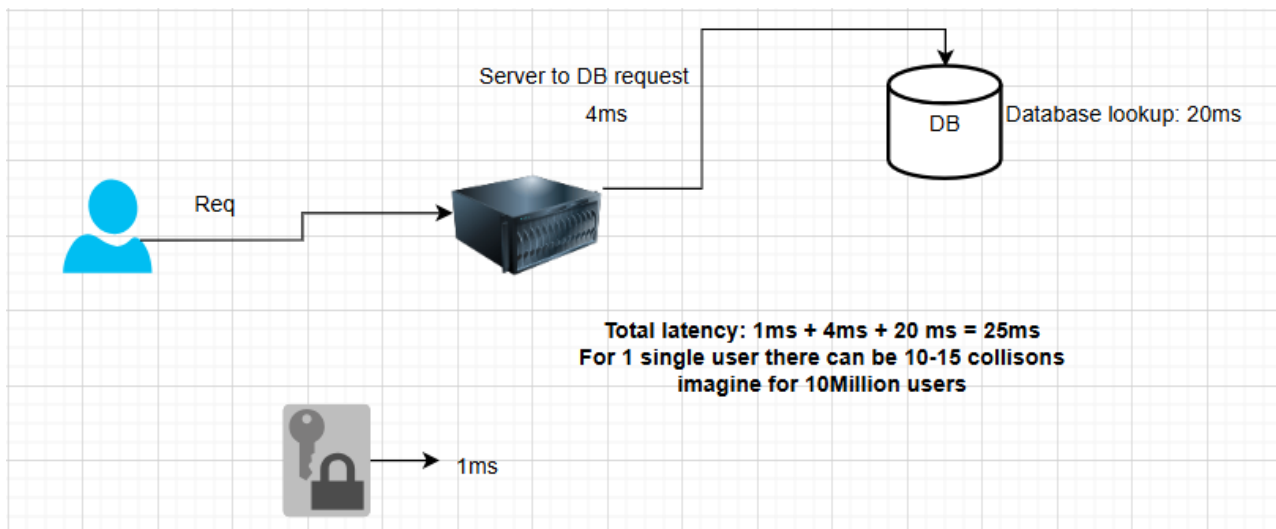
2. Re-direction: When user enters shortURL in browser



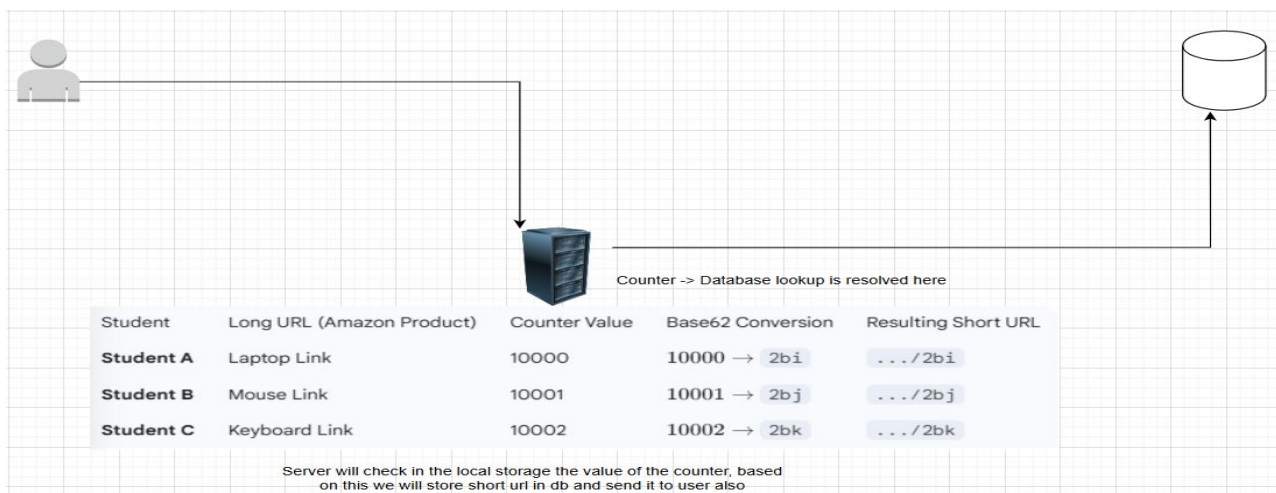
LOW-LEVEL DESIGN

Conversion of longURL into shortURL

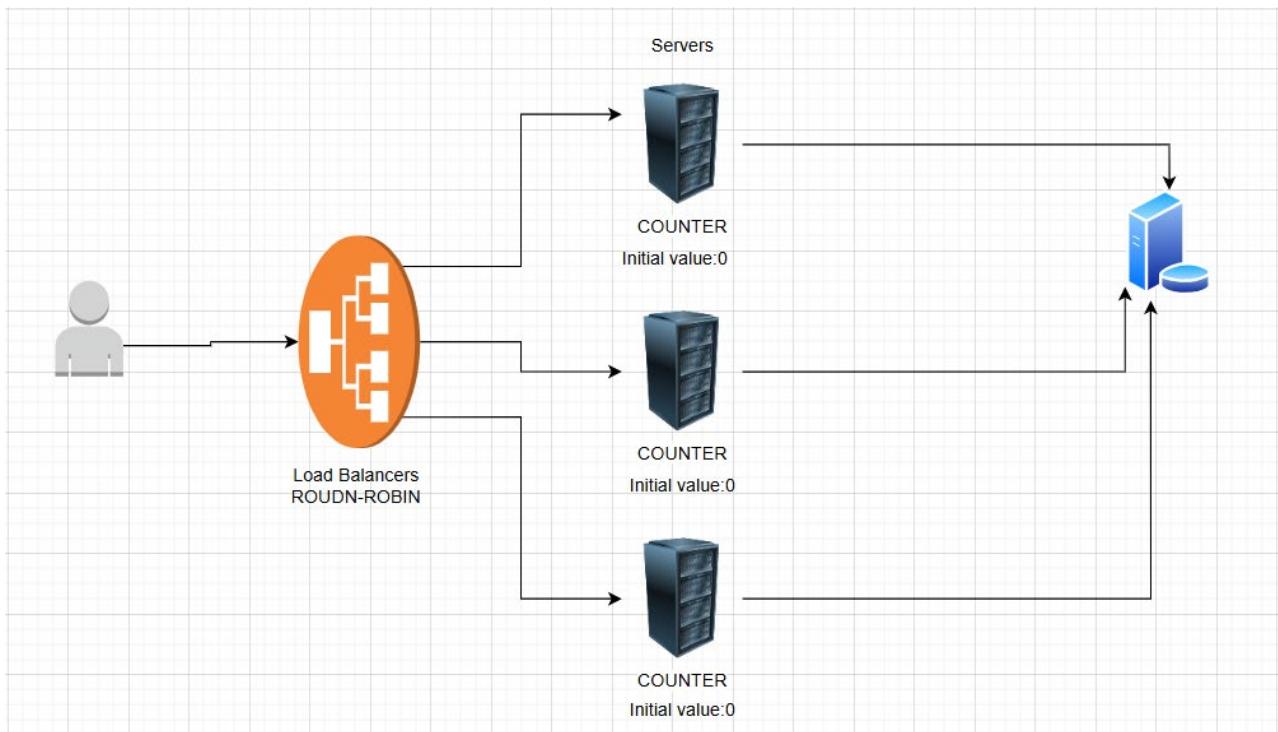
Approach 1: Encryption



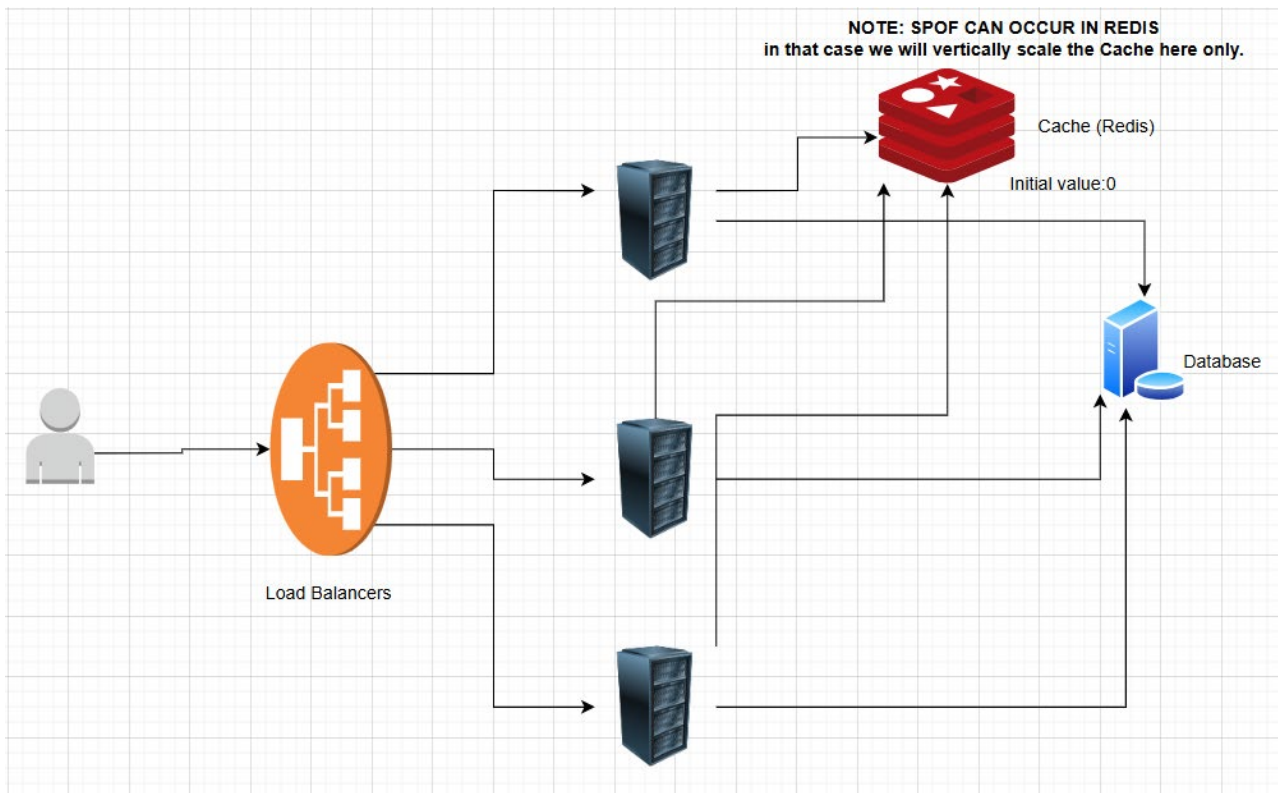
Approach 2: Count Approach



Horizontal Scaling



Redis Implementation



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.