Experiment 1

Student Name: Umang Kaushik	**UID: 23BCS10712**
Branch: BE CSE	**Section/Group: KRG-1-B**
Semester: 6th	**Date of Performance:10th Jan**
Subject Name: SD	**Subject Code: 23CSH-314**

1. **Aim:** To design and analyze a URL Shortener system by identifying its functional and non- functional requirements and representing the system design using a draw.io diagram.

2. **Objectives:**

- To understand the working of a URL Shortener system
- To identify functional requirements of the system
- To identify non-functional requirements such as performance and scalability
- To design a high-level system flow using draw.io
- To improve understanding of real-world system design concepts

3. **Procedure:**

- Studied the concept of URL Shortener systems used in real-world applications.
- Identified the core functionalities required for URL shortening and redirection.
- Listed the functional requirements such as short URL creation, custom URL
- support, expiration handling, and redirection.
- Identified non-functional requirements including low latency and scalability.

- Designed a structured system diagram using draw.io, representing the requirements
- clearly.
- Reviewed the diagram to ensure clarity, correctness, and completeness.

4. Functional Requirements:

- **URL Shortening:** The system accepts a Long URL and generates a unique, shorter alias.
- **URL Redirection:** When a user visits the Short URL, the system redirects them to the original Long URL.
- **Custom Aliases (Optional):** Users should have the option to pick a specific custom string for their short URL (e.g., tiny.url/my-link).
- Expiration Management:
- **Default:** URLs expire automatically after a set period.
- **Custom:** Users can specify their own expiration date.

5. Non-Functional Requirements:

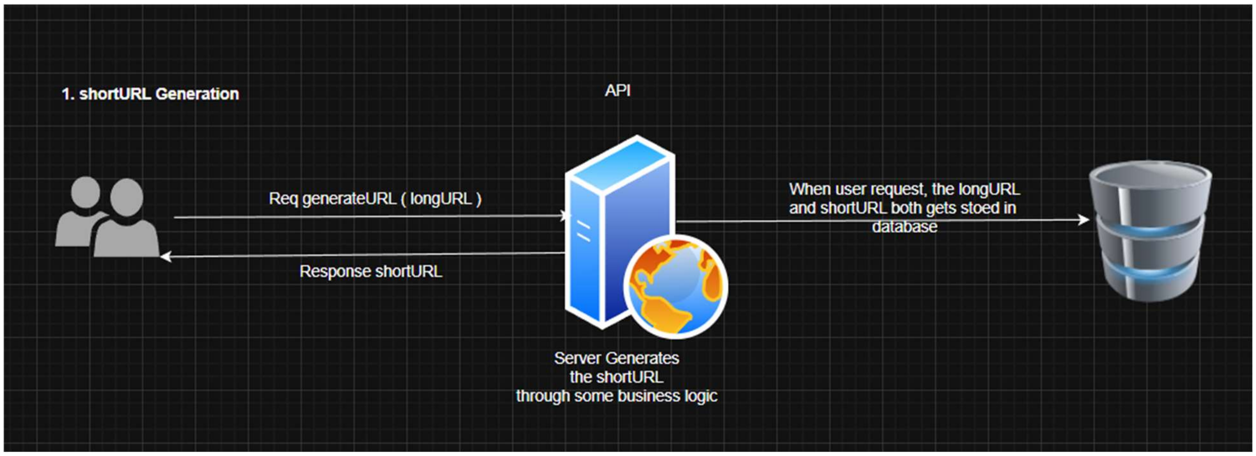
- **Low Latency:** Both creation (write) and redirection (read) requests must complete in **< 200ms**.
- **Scalability:** The system must handle **100 Million Daily Active Users (DAU)** and store/manage at least **1 Billion URLs**.
- **Uniqueness:** Every short URL generated must be unique; no collisions are allowed.
- **High Availability:** The system must be online **24/7** with zero downtime.
- **Consistency Model (CAP):** The system prioritizes **Availability over Consistency** (AP System).

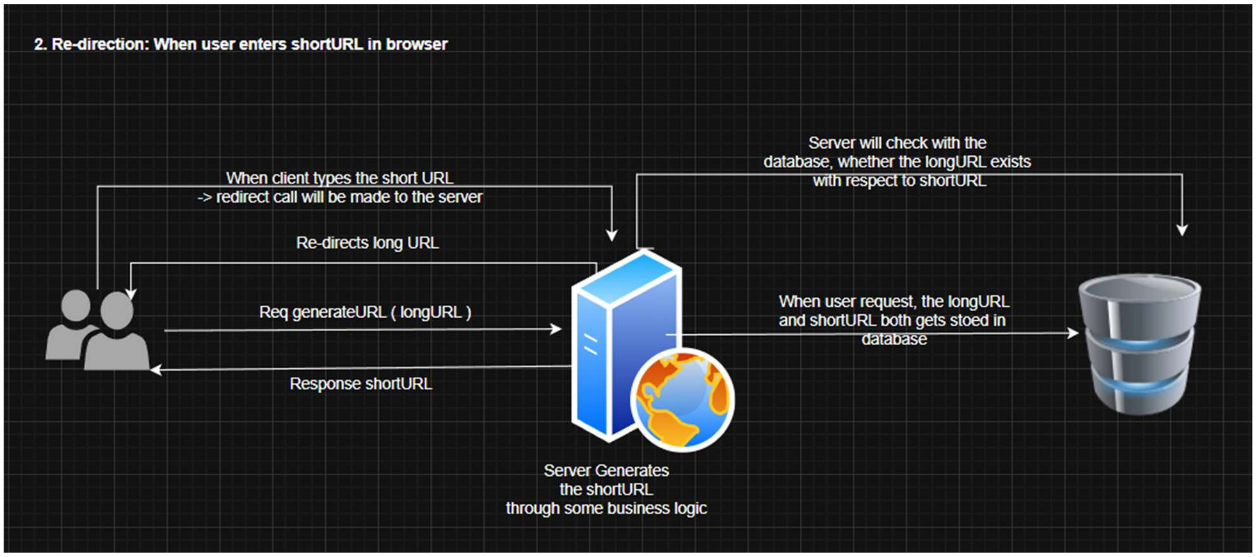
- Updates are **Eventually Consistent** (it is acceptable if a newly created link takes a few moments to propagate to all server nodes).

6. Outcome:

Low Latency: Both creation (write) and redirection (read) requests must complete in < 200ms.
Scalability: The system must handle 100 Million Daily Active Users (DAU) and store/manage at least 1 Billion URLs.
Uniqueness: Every short URL generated must be unique; no collisions are allowed.
High Availability: The system must be online 24/7 with zero downtime.
Consistency Model (CAP): The system prioritizes Availability over Consistency (AP System). Updates are Eventually Consistent (it is acceptable if a newly created link takes a few moments to propagate to all server nodes).

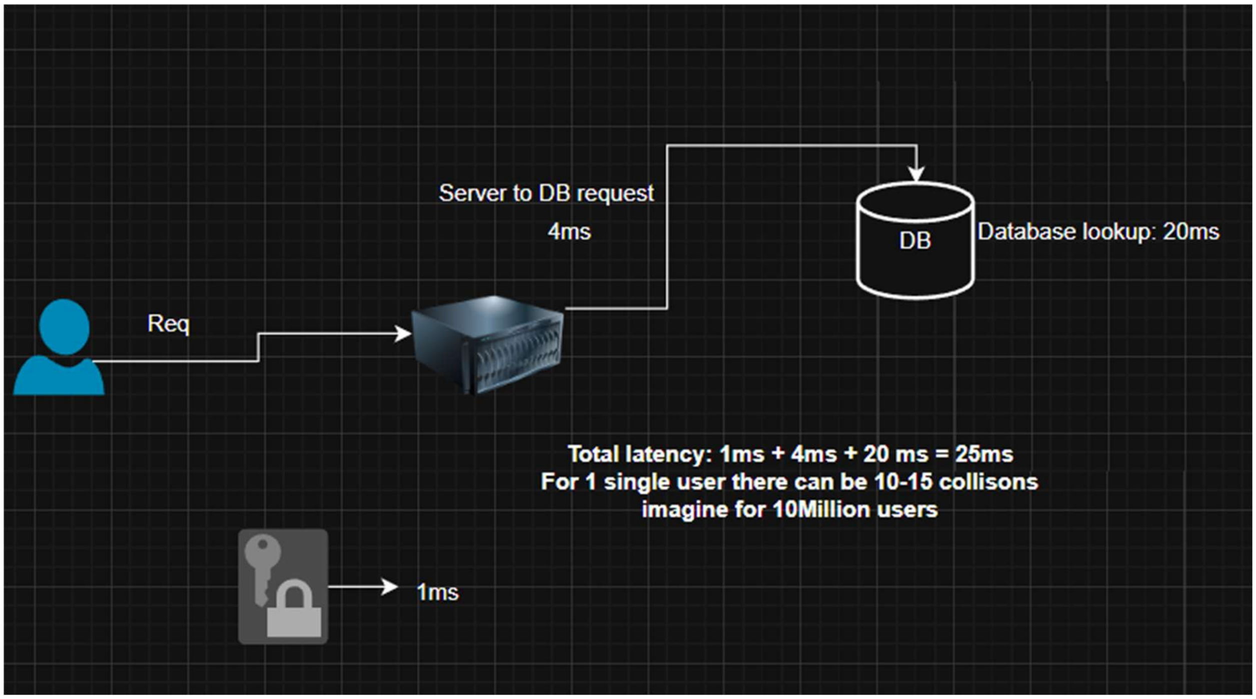
7. Required System Design:


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

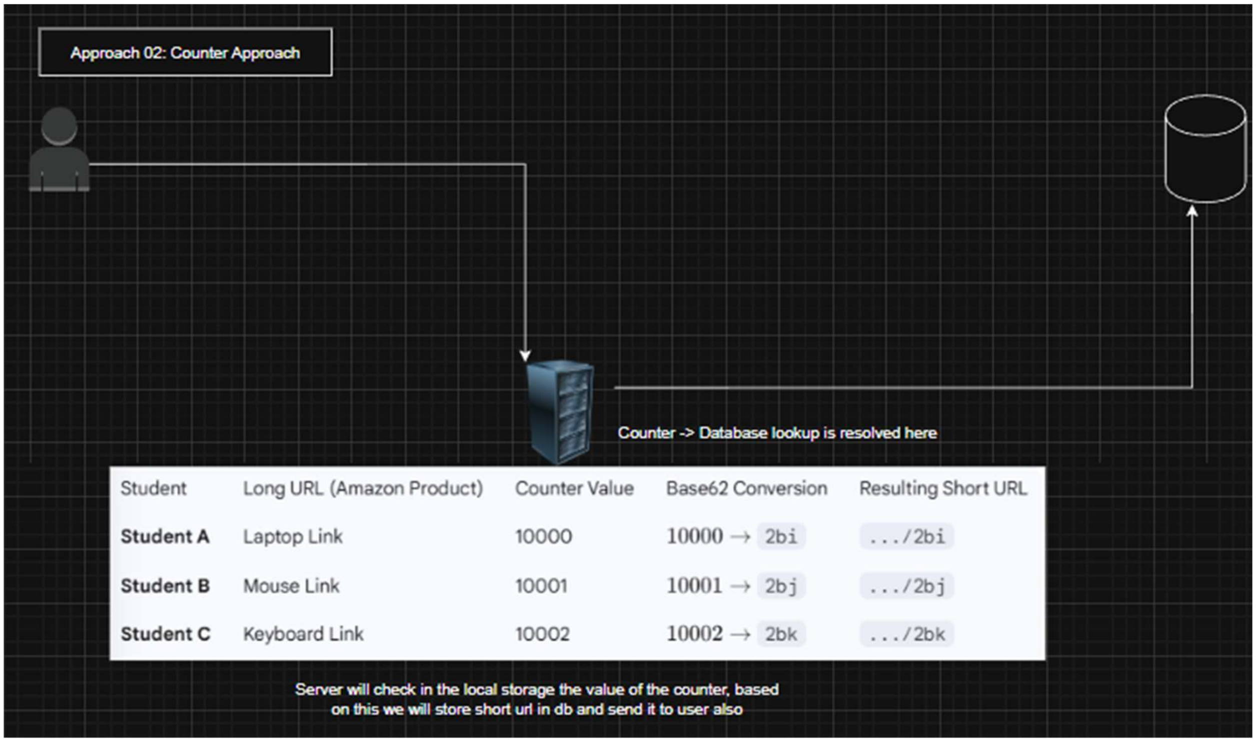


2. Re-direction: When user enters shortURL in browser

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

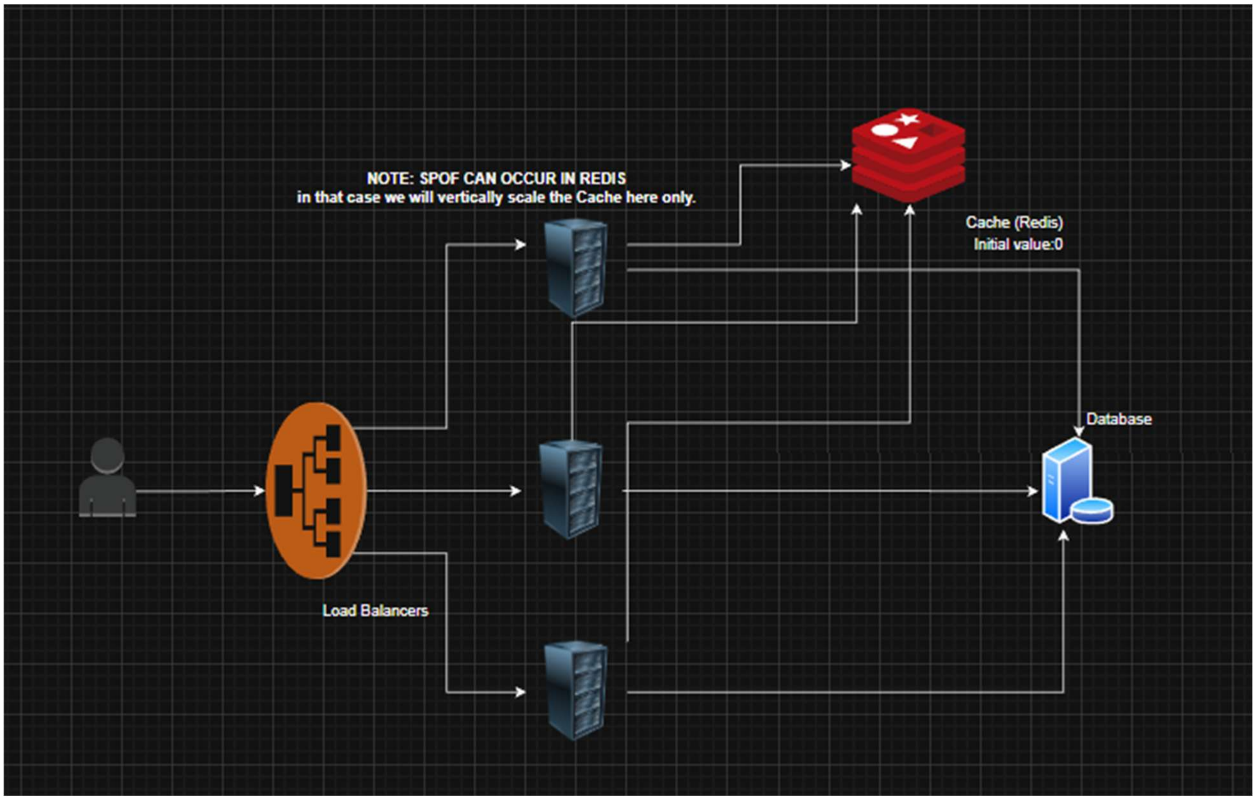


Total latency: 1ms + 4ms + 20 ms = 25ms
For 1 single user there can be 10-15 collisons
imagine for 10Million users


Approach 02: Counter Approach
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

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


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