Experiment No: 1

Student Name: Kanishk Bhandari	UID: 23BCS11100
Branch: BE CSE	Section/Group: 23BCS_KRG-2_B
Semester: 6th	Date of Performance: 07/01/2026
Subject Name: System Design	Subject Code: 23CSH-314

1- **Aim** - URL Shortner Designing

2- **Requirements**: Functional & Non-Functional

A- Functional Requirement

- Create a short URL from a Long URL.
- Optional:
  Support custom URL,
  Supports expiration date : Default + Custom expiration
- User should get redirected to the original URL from short URL.

B- Non-Functional Requirement

- Low Latency: response in least amount of time: (On URL-creation, on URL Redirect) - 200 ms
- Scalability: 100M daily active user & application shoudl be able to short 1B URL
- Unique: Shorten URL should be Unique.
- Availability: 24 x 7 available
- CAP: Tradeoff btw Consistency & Availability: We need high availability here as per requirement.
  Availability >> Consistency

  We will achieve consitency having some delay - This system is called Eventual system.

3- **Core-entities of System**

- Short-URL
- Long-URL
- User

4- **API endpoint creation**

- POST:

```
{
    longURL,
    customURL ? (optional),

    expirationDate
}
```

- GET:

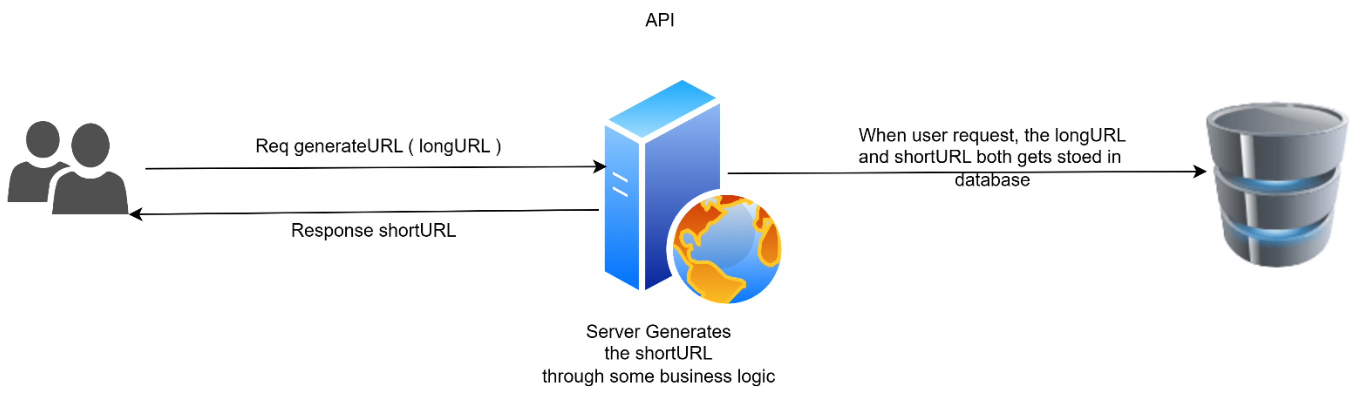
  /v1/url/{short-url} -> will redirect me to longURL

5- High-Level Design

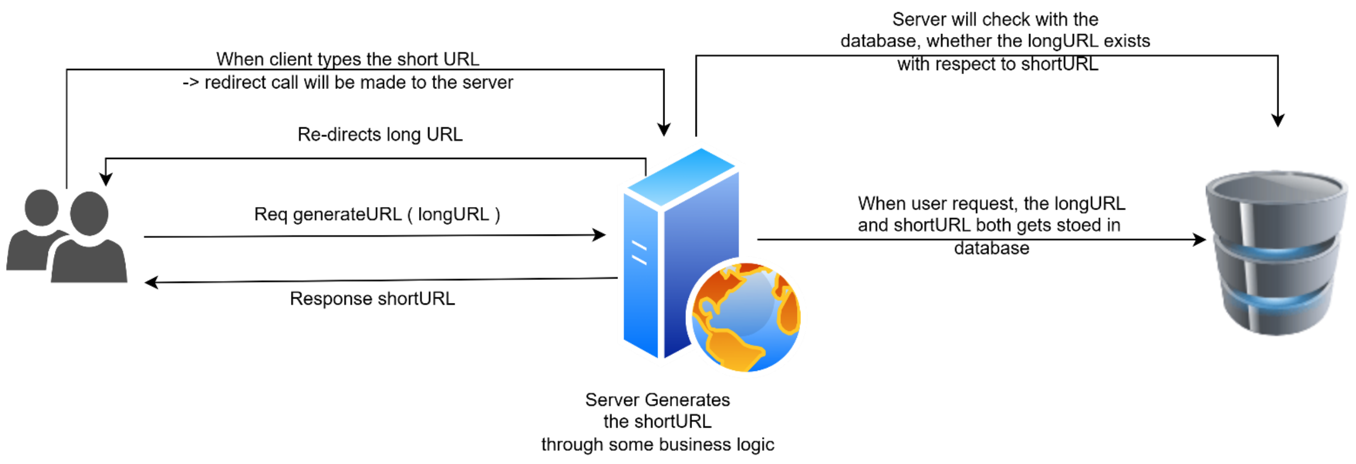
Now According to the functional requirement of the system, we can identify that :

There will be a client who is requesting, then there will a server upon which computation will be going on, and lastly there will be an database in which storage will be done

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

3. We need to have REST APIs for client registration and login as well

4. Database Schema Design

- T1(col1, col2, col3... coln) User Meta Data Storage
- T2(id, shortURl, longURL, customURL, expirationDate) URL_Table

6- Low-Level Design

longURL : https://youtu.be/MmZjpFVMsqk?si=3B1TjNRkhrIi4cFO

shortURL : https://bit.ly/5PLcymn

Approach 1-

encrypt(longURL) = shortURL

Most Popular alogrithm for encryption lib are : MD5, SHA1, BASE64

Limitations –

- Encrypted length is very large & the requirement was of short lenght url.
- To resolve this, what we can do is we can take first 4 letters from encrypted text. But in this also a problem can occur.

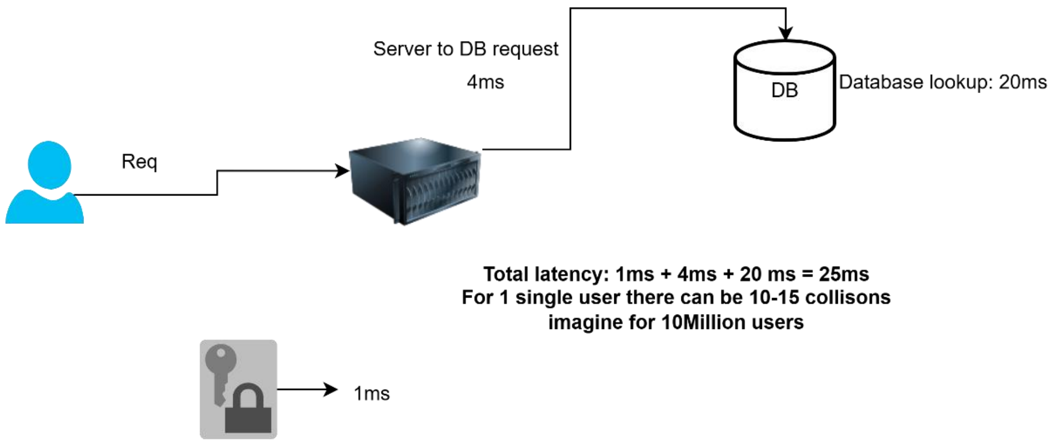
  Link1: https://www.youtube.com/watch?v=**HHUi**8F_qAXM&t=1s06e2116c3e064129e81226fb7b5750ac

  Link2: https://www.youtube.com/watch?v=**HHUi**dbi8793hch?2fdgvkk165f903040b075b90ed40ad15a4f6e5a184a2

- Starting four characters are same, pointing to same link in DB i.e. Duplication of links

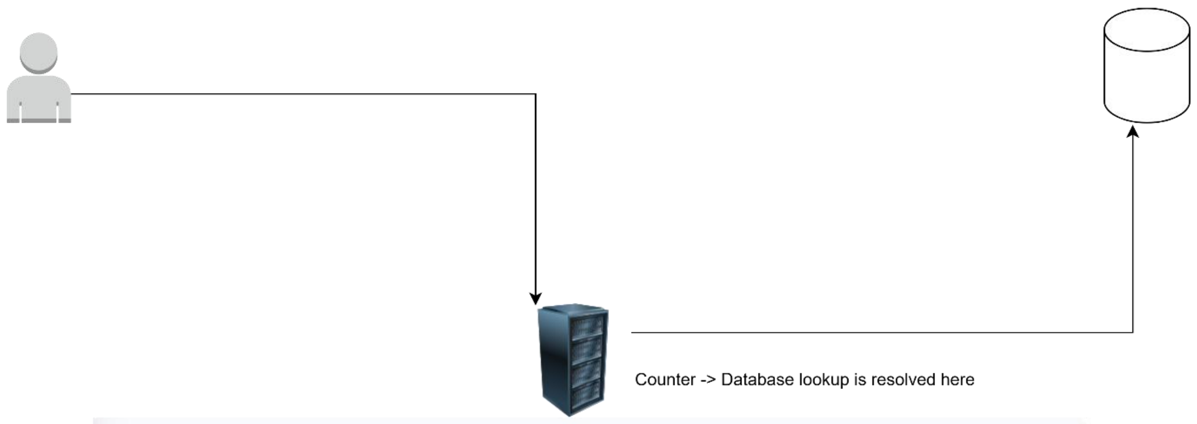
Resolution: At first we can store the 4 byte code & corresponding longURL in DB. Second time, if same 4 byte code is generate, it wll be compared in the DB, if code already exist again then generate a MD5 4byte unique code.

BUT this results into high latency issue 16ms + 16ms

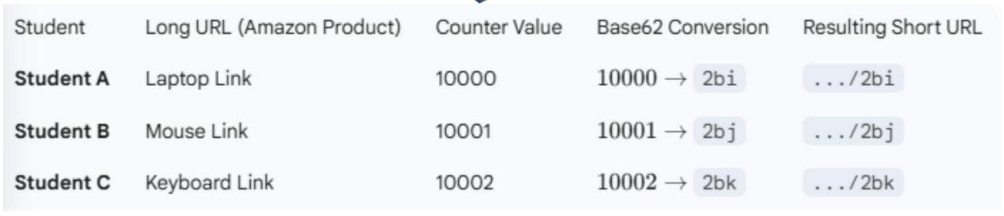


Total latency: 1ms + 4ms + 20 ms = 25ms
For 1 single user there can be 10-15 collisons
imagine for 10Million users

Approach 2 (Counter Approach)-



Counter -> Database lookup is resolved here



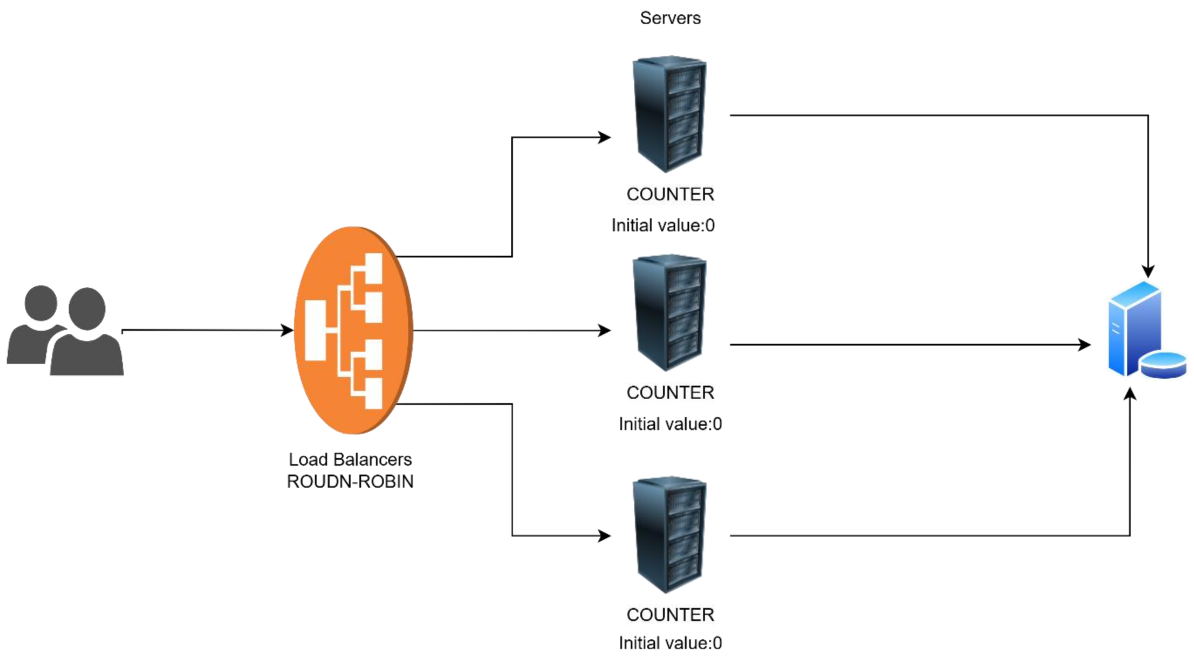
Student	Long URL (Amazon Product)	Counter Value	Base62 Conversion	Resulting Short URL
Student A	Laptop Link	10000	$10000 \to$ `2bi`	`.../2bi`
Student B	Mouse Link	10001	$10001 \to$ `2bj`	`.../2bj`
Student C	Keyboard Link	10002	$10002 \to$ `2bk`	`.../2bk`

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

Limitations –

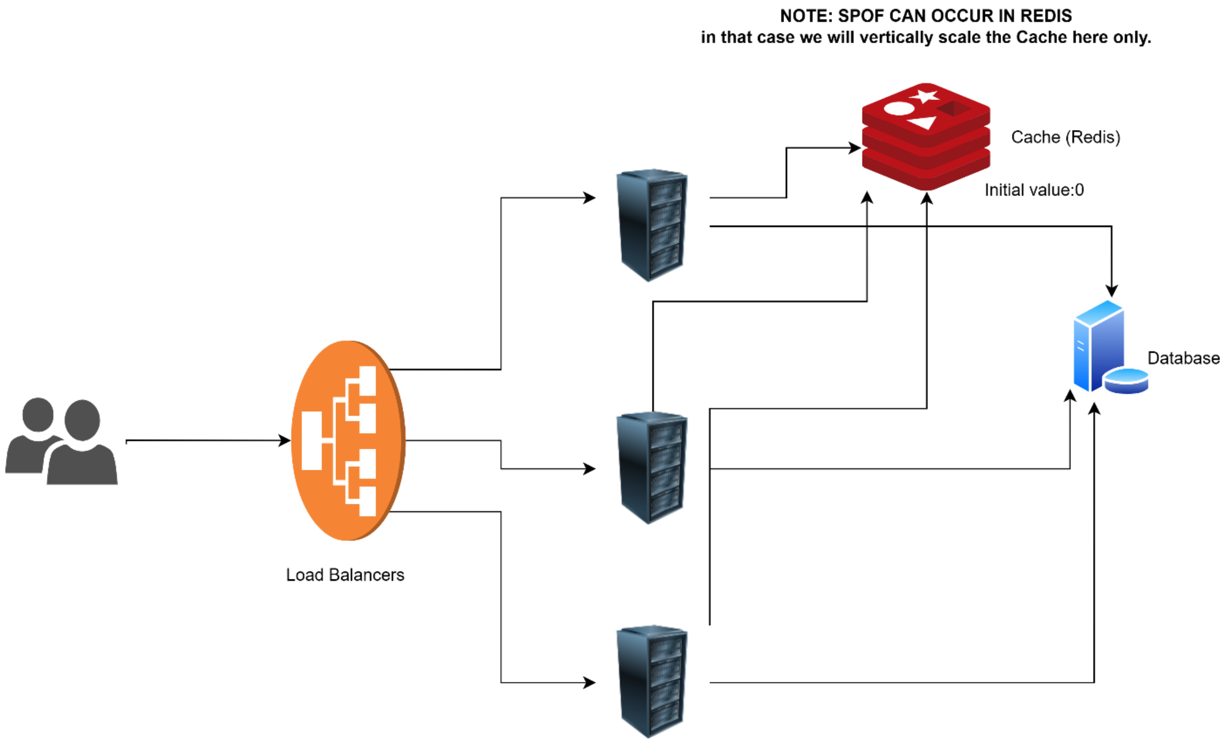
- Server follows monolithic architecture, it can process the request for 1 user easily by managing counter value, but what happens when 100 Million user comes.
- **Resolution – Scaling**

  Vertical scaling here is not possible as number of users are 100 M, So horizontal scaling is done.

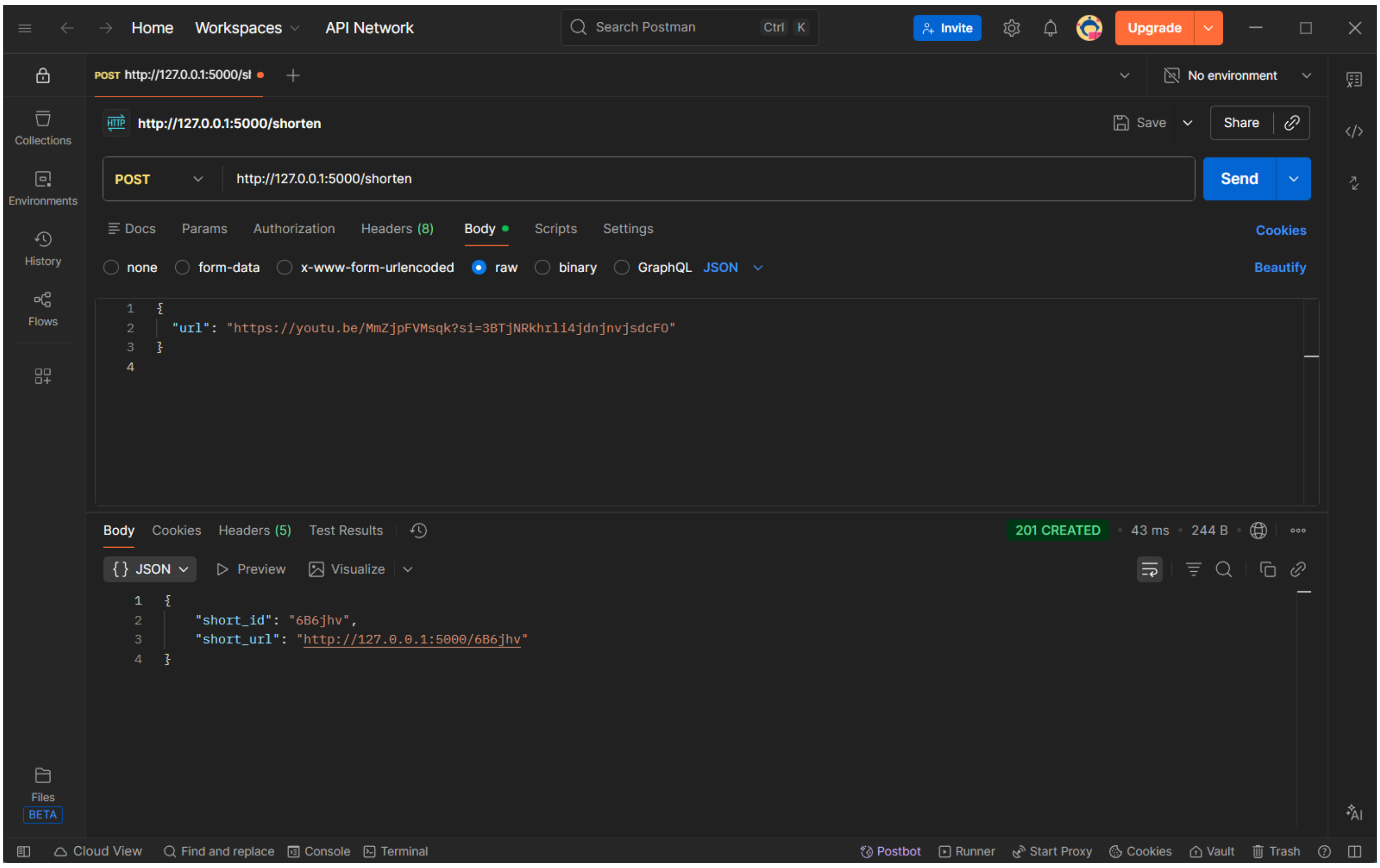


Problem - Now in this method there can be a problem of dirty read that is server 1's counter may have some other value and server 2's counter may have some other value.

Resolution: Rather than storing the counter in the server itself, we can make the counter variable globally available using a cache.


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

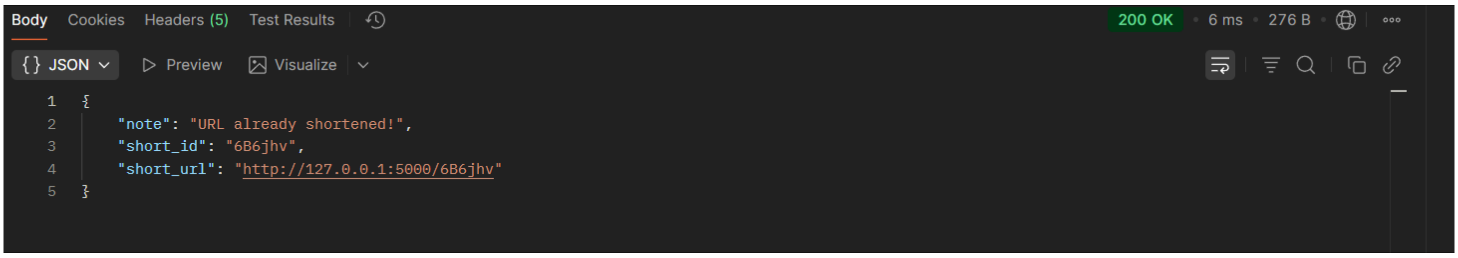
7- Output-


POST http://127.0.0.1:5000/shorten

```
{
    "url": "https://youtu.be/MmZjpFVMsqk?si=3BTjNRkhrli4jdnjvjsdcFO"
}
```

201 CREATED · 43 ms · 244 B

```
{
    "short_id": "6B6jhv",
    "short_url": "http://127.0.0.1:5000/6B6jhv"
}
```


200 OK · 6 ms · 276 B

```
{
    "note": "URL already shortened!",
    "short_id": "6B6jhv",
    "short_url": "http://127.0.0.1:5000/6B6jhv"
}
```