Functional Requirements

1. Given a long URL, return a short URL
2. Given a short URL, redirect to long URL

Non -Functional Requirements

1. Very Low latency
2. Highly available

Back of the envelope estimation

- Traffic? How many long URL will come to our server. Write request /day = 100milliom
- For how long we must store the URL -> 10 years.
- Maximum number of records -> 100m*10*365 = 365 billion.
- Avg length of long URL = 100bytes. Total volume we need to store is: 100bytes*365billion
- Read request : ratio with write is 10:1
- Number of characters: Consider 0-9, a-z and A-Z. Total number of characters is 62.
- Maximum length of short URL: 62^7 = 3.5 trillion. Max length = 7

API's

1. GET or POST: given long URL give short URL
2. GET : short URL: redirect to long URL. Status code can be 301 permanently moved (browser will cache the response and will not call the API for the same URL) or 302 (temporary) browsers will send the request to the API always.

Architecture

Hash Function

- Hash function that takes a long URL and gives a 7-character string.
- Either can use MD5 or sha1 or CRC32. Problem is they are more than 7 characters.
- If we pick the first 7 characters they can lead to collision.
- Another can be to map long URL to a integer value and then convert it to base62 representation (base10 to base62). Where a-z will be from 10-36 and A-Z will be from 37-62.

Database Choice

- SQL -> Can use. Primary key on ID to avoid 2 different longURL map to same

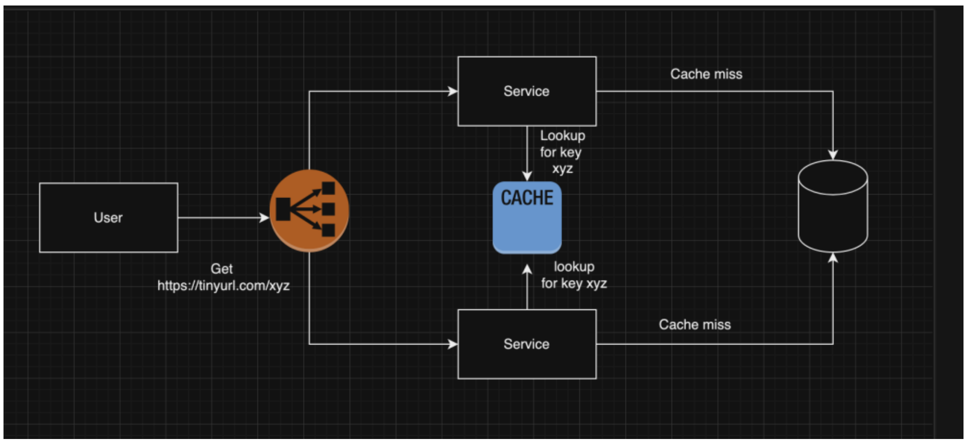
  Unique id generated. Although the chances of unique id generated same is pretty low.

- Challenges is volume of record that we might need to store in the database.
- Can use NOSQL to leverage. To tackle collison as explained above, we can use distributed

  Locking.

ceation_date	expired_date	id	long_url	short_url

Diagram


Service
Cache miss
Lookup for key xyz
CACHE
User
Get https://tinyurl.com/xyz
lookup for key xyz
Service
Cache miss