Requirements

Functional Requirements:

1. How long do we want out tiny url to be (7 or 9 characters)?
2. Do we want to support custom url’s and length?
3. Shorter url need to redirect to longer url
4. Should these expire or do we need to have a set timeout for these
5. Does this timeout apply for custom url’s?
6. Is it read heavy or write heavy?
7. How many users can a free user create?

Non-Functional Requirements:

1. How many users will be interacting with this system?
2. Shortened links should not be decodable
3. It should be highly available

Other Requirements:

1. Analytics, how many times accessed

API Routes and Methods

1. HTTPPOST: CreateURL (string longUrl, int expirationTime) 🡪 Return ShortUrl
   1. UserName
   2. LongUrl
   3. ExpirationTime
   4. CustomUrl
   5. Private token
2. HTTPGET: GetLongURL (string shortUrl) 🡪 307, redirection
3. HTTPDELETE: Delete (string shortUrl)

Number of users

1. Scale

Read: Write 🡪 100: 1

500M URLS in a month

500M/ 30 \* 24 \* 60 \* 60 – 200Url’s/sec

Read: 100 \* 200 🡪 20K/sec

Write: 1 \* 200 🡪 200 / sec

1. Storage

Each url takes 500bytes of memory and we want to store these URL’s for 5 years

500M \* 5 \* 12 \* 500 🡪 15TB

1. Network Bandwidth

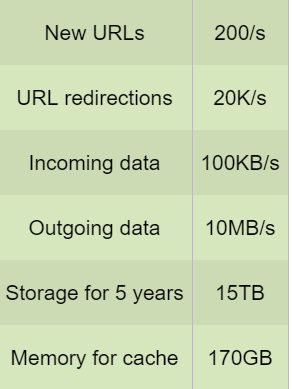
Write: 200/sec \* 500bytes 🡪 10KB/sec

Read: 20K/sec \* 500bytes 🡪 10MB/sec

1. Cache the hot urls

Let say 20% of the urls are cached

0.2 \* 200K \* 24 \* 60 \* 60 🡪 170GB



Define database models

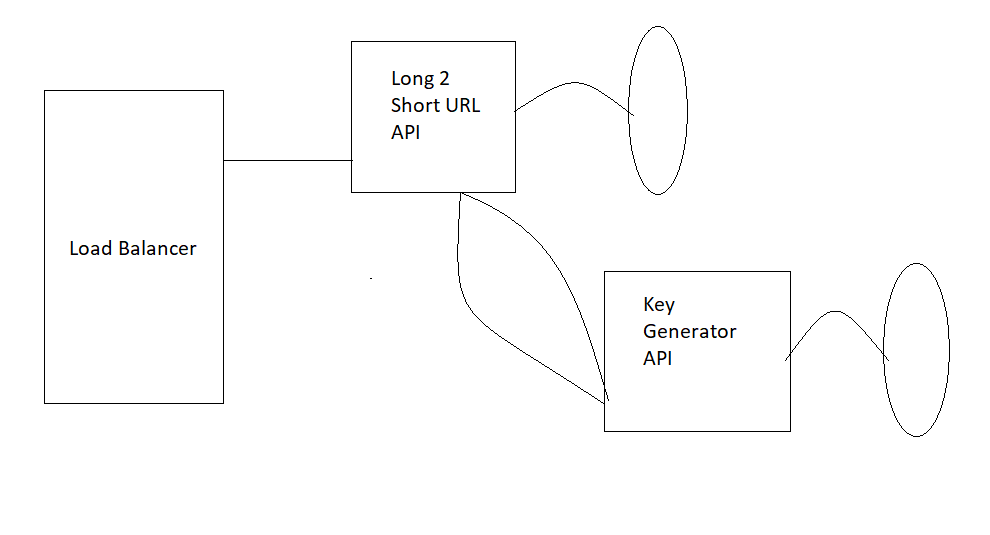
1. User Details
   1. Username
   2. Password
   3. Private token
2. Url Database
   1. ShortUrl – Primary Key
   2. LongUrl
   3. ExpirationDate
   4. UserName
   5. CreatedDate

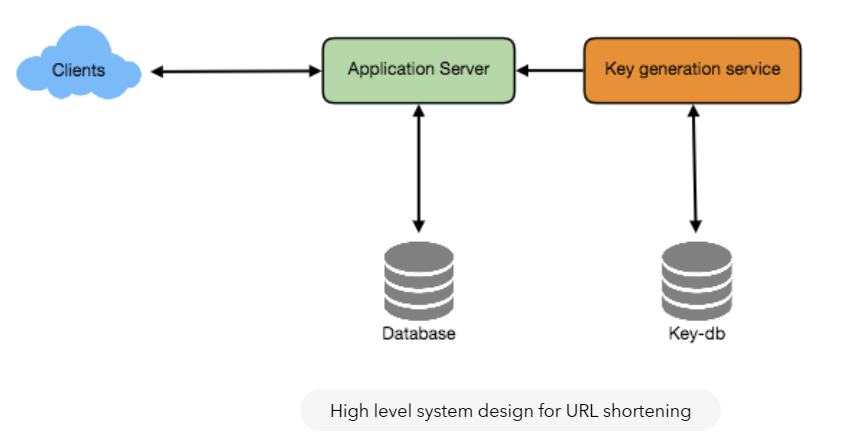
As this is read heavy and there are no major relationships exists, we can user NoSQL database like Cassandra, DynamoDB

System Design

1. We can generate a 7bit length or 9bit length url
2. We can use 0-9, a-z, A-z a total of 62 characters to form the tiny url
3. For 7 bit, 62 ^ 7 (3 trillion) and 9bit (13 quadrillion)
4. 586 years for a 7-bit url
5. We can hash the original url using MD5 or SHA256. MD5 produces 128-bit url and we can convert this into base64 (each 6bits are converted to an ACII character) encoding which produces a 21 letter. From this we can pick the first 7 characters or 9 characters
6. This will result in key duplication
   1. Two URL’s might result in same short url
      1. When these conflicts occur, we can append a unique increasing sequence number to it and encode to base64
      2. Unique username to the short url
      3. Store it in the DB, if fails generate a new one
   2. Key generation can be a different microservice which serves us the key
      1. It can store some predefined keys and store it in the database
      2. Concurrency can be an issue
         1. Mark the keys as read or delete them asap
         2. Have some in memory keys for each server

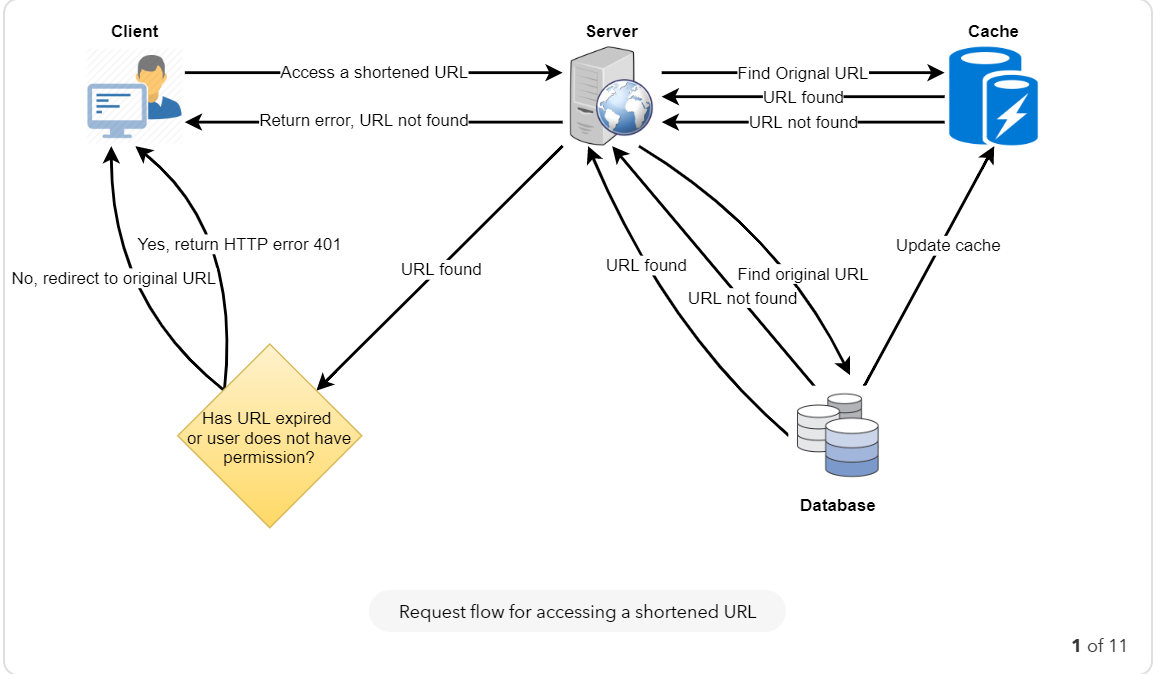
Block Diagram





Identify and Resolve bottlenecks

1. We cannot store all the URL’s in one place
   1. Sharding
      1. Based on the starting letter of the hash code
      2. Region of the user
      3. Hashing the key and selecting the partition by modulo of number of shards
      4. Consistent hashing
   2. Caching
      1. Redis
      2. LRU cache to evict the least used urls
      3. Cache 20% of the hot urls



Load Balancing:

1. Between clients and servers
2. Between servers and databases
3. Between servers and caches

Round robin or consistent hashing

