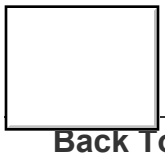


Back To Course Home

Grokking Modern System Design Interview for Engineers & Managers

0% completed

Search Course

System Design Interviews

Introduction

Abstractions

Non-functional System Characteristics

Back-of-the-envelope Calculations

Building Blocks

Domain Name System

Load Balancers

Databases

Key-value Store

☐

Content Delivery Network (CDN)

☐

Sequencer

☐

Distributed Monitoring

☐

Monitor Server-side Errors

☐

Monitor Client-side Errors

☐

Distributed Cache

☐

Distributed Messaging Queue

☐

Pub-sub

☐

Rate Limiter

☐

Blob Store

☐

Distributed Search

☐

Distributed Logging

☐

Distributed Task Scheduler

Sharded Counters

Concluding the Building Blocks Discussion

Design YouTube

Design Quora

Design Google Maps

Design a Proximity Service / Yelp

Design Uber

Design Twitter

Design Newsfeed System

Design Instagram

Design a URL Shortening Service / TinyURL

Design a Web Crawler

Design WhatsApp

Design Typeahead Suggestion

Design a Collaborative Document Editing Service / Google Docs

Spectacular Failures

Concluding Remarks

Course Certificate

Mark Course as Completed

Encoder for TinyURL

Understand the inner details of an encoder that are critical for URL shortening.

We'll cover the following

Introduction#

We've discussed the overall design of a short URL generator (SUG) in detail, but two aspects need more clarification:

1. How does encoding improve the readability of the short URL?
2. How are the sequencer and the base-58 encoder in the short URL generation related?

Why to use encoding#

Our sequencer generates a 64-bit ID in base-10, which can be converted to a base-64 short URL. Base-64 is the most common encoding for alphanumeric strings' generation. However, there are some inherent issues with sticking to the base-64 for this design problem: the generated short URL might have readability issues because of look-alike

characters. Characters like `O` (capital o) and `0` (zero), `I` (capital I), and `l` (lower case L) can be confused while characters like `+` and `/` should be avoided because of other system-dependent encodings.

So, we slash out the six characters and use base-58 instead of base-64 (includes A-Z, a-z, 0-9, `+` and `/`) for enhanced readability purposes. Let's look at our base-58 definition.

Base-58

Value	Character	Value	Character	Value	Character	Value
0	1	15	G	30	X	45
1	2	16	H	31	Y	46
2	3	17	J	32	Z	47
3	4	18	K	33	a	48
4	5	19	L	34	b	49
5	6	20	M	35	c	50
6	7	21	N	36	d	51
7	8	22	P	37	e	52
8	9	23	Q	38	f	53
9	A	24	R	39	g	54
10	B	25	S	40	h	55
11	C	26	T	41	i	56
12	D	27	U	42	j	57
13	E	28	V	43	k	
14	F	29	W	44	m	

> The highlighted cells contain the succeeding characters of the omitted ones: `0`, `O`, `I`, and `l`.

Converting base-10 to base-58#

Since we're converting base-10 numeric IDs to base-58 alphanumeric IDs, explaining the conversion process will be helpful in grasping the underlying mechanism as well as the overall scope of the SUG. To achieve the above functionality, we use the **modulus** function.

Process: We keep diving the base-10 number by 58, making note of the remainder at each step. We stop where there is no remainder left. Then we assign the character indexes to the remainders, starting from assigning the recent-most remainder to the left-most place and the oldest remainder to the right-most place.

Example: Let's assume that the selected unique ID is `2468135791013`. The following steps show us the remainder calculations:

Base-10 = 2468135791013

1. 2468135791013 % 58 = 17
2. 42554065362 % 58 = 6
3. 733690782 % 58 = 4
4. 12649841 % 58 = 41
5. 218100 % 58 = 20
6. 3760 % 58 = 48
7. 64 % 58 = 6
8. 1 % 58 = 1

Now, we need to write the remainders in order of the most recent to the oldest order.

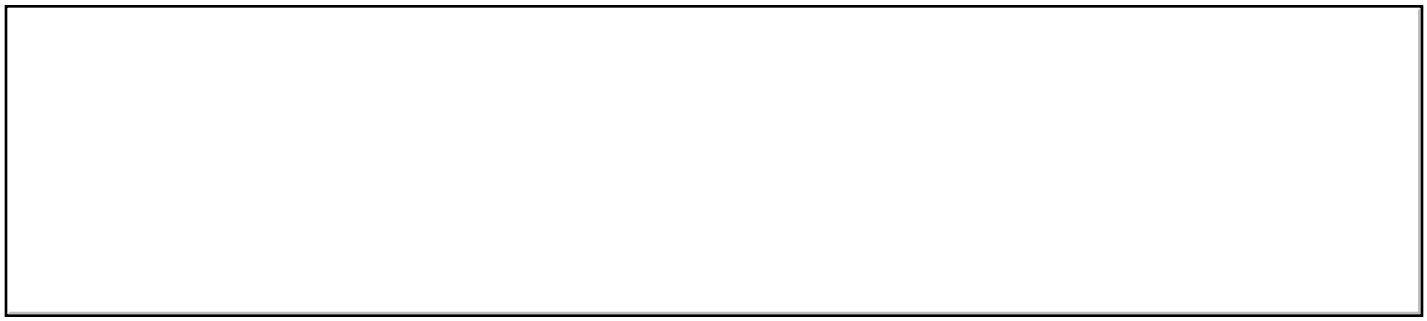
Base-58 = [1] [6] [48] [20] [41] [4] [6] [17]

Using the table above, we can write the associated characters with the remainders written above.

Base-58 = 27qMi57J

> **Note:** Both the base-10 numeric IDs and base-64 alphanumeric IDs have 64-bits, as per our sequencer design.

Let's see the example above with the help of the following illustration.



How a base-10 number is converted into a base-58 alphanumeric short URL

Converting base-58 to base-10#

The decoding process holds equal importance as the encoding process, as we used a decoder in case of custom short URLs generation, as explained in the design lesson.

Process: The process of converting a base-58 number into a base-10 number is also straightforward. We just need to multiply each character index (value column from the table above) by the number of 58s that position holds, and add all the individual multiplication results.

Example: Let's reverse engineer the example above to see how decoding works.

Base-58: 27qMi57J

$$2_{58} = 1 \times 58^7 = 2207984167552$$

$$7_{58} = 6 \times 58^6 = 228412155264$$

$$q_{58} = 48 \times 58^5 = 31505124864$$

$$M_{58} = 20 \times 58^4 = 226329920$$

$$i_{58} = 41 \times 58^3 = 7999592$$

$$5_{58} = 4 \times 58^2 = 13456$$

$$7_{58} = 6 \times 58^1 = 348$$

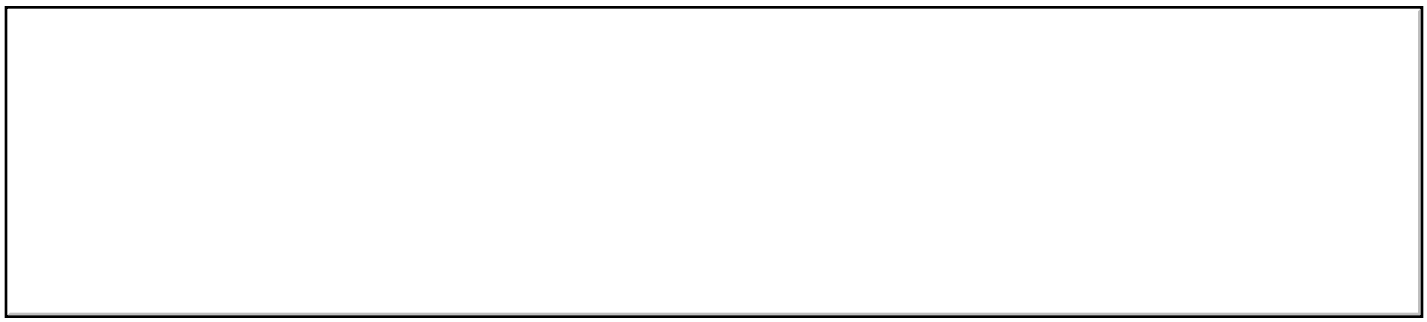
$$J_{58} = 17 \times 58^0 = 17$$

$$\text{Base-10} = 17 + 348 + 13456 + 7999592 + 226329920 + 31505124864 + 228412155264 + 2207984167552$$

$$\text{Base-10} = 2468135791013.$$

This is the same unique ID of base-10 from the previous example.

Let's see the example above with the help of the following illustration.



How a base-58 number is converted into a base-10 numeric ID

The scope of the short URL generator#

The short URL generator is the backbone of our URL shortening service. The output of this short URL generator depends on the design-imposed limitations, as given below:

- The generated short URL should contain alphanumeric characters.
- None of the characters should look alike.
- The minimum default length of the generated short URL should be six characters.

These limitations define the scope of our short URL generator. We can define the scope, as shown below:

- **Starting range**: Our sequencer can generate a 64-bit binary number that ranges from $1 \rightarrow (2^{64} - 1)$. To meet the requirement for the minimum length of a short URL, we can select the sequencer IDs to start from at least 10 digits, i.e., 1 Billion.
- **Ending point**: The maximum number of digits in sequencer IDs that map into the short URL generator's output depends on the maximum utilization of 64 bits, that is, the largest base-10 number in 64-bits. We can estimate the total number of digits in any base by calculating these two points:
    1. The numbers of bits to represent one digit in a base-n. This is given by $log_2 n$.
    2. $Number\ of\ digits\ = \frac{Total\ bits\ available}{Number\ of\ bits\ to\ represent\ one\ digit}$

Let's see the calculations above for both the base-10 and base-58 mathematically:

- **Base-10:**
    - The number of bits needed to represent one decimal digit = $log_2 10 = 3.13$
    - The total number of decimal digits in 64-bit numeric ID = $\frac{64}{3.13} = 20$
- **Base-58:**
    - The number of bits needed to represent one decimal digit = $log_2 58 = 5.85$
    - The total number of base-58 digits in a 64-bit numeric ID = $\frac{64}{5.85} = 11$

> **Maximum digits**: The calculations above show that the maximum digits in the sequencer generated ID will be 20 and consequently, the maximum number of characters in the encoded short URL will be 11.

Quiz

Question 1

Since we're using the 10 digits and beyond sequencer IDs, is there a way we can use the sequencer IDs shorter than 10 digits?

Show Answer

The sequencer's lifetime#

The number of years that our sequencer can provide us with unique IDs depends on two factors:

- Total numbers available in the sequencer = $2^{64} - 10^9$ (starting from 1 Billion as discussed above)
- Number of requests per year = $200\ Million\ per\ month \times 12 = 2.4\ Billion$ (as assumed in *Requirements*)

So, taking the above two factors into consideration, we can calculate the expected life of our sequencer.

The lifetime of the sequencer = $\frac{total\ numbers\ available}{yearly\ requests} = \frac{2^{64}-10^9}{2.4\ Billion} = 7,686,143,363.63\ years$

Life expectancy for sequencer

Number of requests per month	200	Million
Number of requests per year	2.4	Billion

Lifetime of sequencer	f 7686143363.63	years

> Therefore, our service can run for a long time before the range depletes.

Design and Deployment of TinyURL

Next

Evaluation of TinyURL's Design

Mark as Completed

Report an Issue