**Short ID Generation in JavaScript**

Nov 16th, 2014

Tags: [javascript](http://fiznool.com/blog/categories/javascript/)

A ‘short’ or ‘hash’ ID is a seemingly random sets of characters, popularised by services which need unique but readable (and typeable) URLs. For example, YouTube use short IDs for their video URLs:

<http://www.youtube.com/watch?v=bmpONxJ7JSw>

I found myself needing to generate short IDs for URLs in a recent project. These were the requirements:

1. Be short enough to type
2. Be easy enough to speak (e.g. over the phone in support situations)
3. Be unique (or close to unique as feasible)
4. Not contain any rude words

I wasn’t looking at generating millions of IDs per day, all I needed was something simple.

**Existing solutions**

Since I was using node.js to generate these IDs, the first port of call was npm, to see if there were any projects which already existed. My search yielded two main results:

* [shortid](https://github.com/dylang/shortid)
* [node.hashids](http://hashids.org/node-js/)

These libraries didn’t really fit the requirements, however:

* shortid looks like a good fit, but it is over-engineered for what I need. You need an alphabet size of 64 characters, and there is no control over the minimum number of chars generated.
* node.hashids generates IDs from a sequence, and is intended to be a reverse-lookup for a number. I din’t need this, and so again it felt over-engineered.

**Rolling my own**

**Initial implementation**

Under the guise of the [KISS principle](http://en.wikipedia.org/wiki/KISS_principle), I set about creating my own generator. Turns out, for my use case, it was rather straightforward.

I started by using JavaScript’s Math.random() to generate an 8-character ‘random’ ID, using a pre-defined alphabet.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var ALPHABET = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';  var ID\_LENGTH = 8;  var generate = function() {  var rtn = '';  for (var i = 0; i < ID\_LENGTH; i++) {  rtn += ALPHABET.charAt(Math.floor(Math.random() \* ALPHABET.length));  }  return rtn;  } |

* Since Math.random creates a number between 0 (inclusive) and 1 (exclusive), Math.random() \* alphabet.length creates a number anywhere between 0 and just less than alphabet.length.
* Math.floor() takes the floating point number and returns the nearest integer, thus ending up with an integer between 0 and alphabet.length - 1 (in this case, 62).
* alphabet.charAt(index) returns the character in the string alphabet at the given index.

This was a good start, and satisfied requirement 1) – it creates an 8 character ID which is easy to type.

**Modifying alphabet size**

Requirement 2) specifies it should be easy to speak over the phone. This will be useful in support situations – e.g. if a user needs assistance, they should be able to speak the ID to an operator without too many difficulties.

The first thing to trim is the lowercase/uppercase, which could cause confusion. IDs look nicer with lowercase, so the uppercase was removed.

|  |  |
| --- | --- |
| 1 | var ALPHABET = '0123456789abcdefghijklmnopqrstuvwxyz'; |

Next up, numbers that look like letters. This is another source of confusion, and so the numbers 0 and 1 and the letters o and l were removed.

|  |  |
| --- | --- |
| 1 | var ALPHABET = '23456789abcdefghijkmnpqrstuvwxyz'; |

Lastly, rude words, aka requirement 3). It could be quite embarrassing to generate an ID with a rude word. Sticking to English profanities, it turns out that most rude words are formed from the majority of the following characters: cfhistu. These characters were also removed.

|  |  |
| --- | --- |
| 1 | var ALPHABET = '23456789abdegjkmnpqrvwxyz'; |

Our final alphabet contains 25 characters.

**Uniqueuess**

Uniqueness is a difficult goal to achieve. I was quite relaxed about requirement 4) – my ID generator would not need to generate millions of IDs per day.

Since we are generating an 8-character ID, the theoretical probability of a clash using the alphabet above of 25 chars is 258 = 152,587,890,625 – over **152 billion** to 1.

However, since Math.random() is a *pseudorandom* generator, which means it is not guaranteed to always produce random numbers, the probability will drop slightly. This being said, it is good enough for my needs.

As a failsafe, a function can be added to check that the generated ID is indeed unique:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | var UNIQUE\_RETRIES = 9999;  var generateUnique = function(previous) {  previous = previous || [];  var retries = 0;  var id;  // Try to generate a unique ID,  // i.e. one that isn't in the previous.  while(!id && retries < UNIQUE\_RETRIES) {  id = generate();  if(previous.indexOf(id) !== -1) {  id = null;  retries++;  }  }  return id;  }; |

For this to work, you should pass in an array of previously generated IDs.

If the list of IDs was stored in a database, you could generate the ID and then check for its existence in the DB before continuing.

**Summary**

Generating your own ‘short’ ID needn’t be complicated. Depending on your requirements, a simple solution can often suffice.

The full code for this generator can be [found here](https://gist.github.com/fiznool/73ee9c7a11d1ff80b81c).