UUID – Universally Unique ID

When any part of the information system can create unique id:s for new objects and events

27.10.2022

Haaga-Helia

Database can generate autoincrement id:s

- MariaDB/MySQL: AUTOINCREMENT, Oracle: Sequence, MS SQL Server: Identity, etc. etc.
- Works fine when it's database that creates the id:s
- Challenges:
  - If database is distributed, how to make sure each item will get unique id?
  - When database creates id:s, how to get that correct id back to frontend that initiated the creation of that new resource/object?
    - Well, e.g. the Knex library returns an array of newly created database row primary keys if success
    - => Return that to the frontend too, e.g. in an object as JSON in body of the response
  - What if we have a totally distributed architecture and there is no certain database? Or we have an IoT system where devices are e.g. underwater and collect information and events and have access to the main system only occasionally?
- Then we start to have a case for system where _everyone_ needs to be able to create id:s …

UUID – Universally Unique ID

- *How everybody would be able to create id:s that are not clashing with ones created by others?*
    - *Let’s create so 1. **long** 2. so **random** character sequence from 3. **hex**ademical digits 0-9,a-f that two random generations would basically never yield the same value.*
- There was a set of similar ideas (e.g. Microsoft’s GUID), which were in 2005 standardized to be a 128-bits (presented in text format like this: 123e4567-e89b-12d3-a456-426614174000).
- Read more here: https://en.wikipedia.org/wiki/Universally_unique_identifier#History
- Technical explanation here: https://www.rfc-editor.org/rfc/rfc4122.html
- Each programming language and environment has implementations for this. They allow the creation of valid UUIDs.
- For JavaScript you can start from the normal good source by searching for e.g.:
    - ”MDN uuid”
    - ”MDN javascript uuid”

While testing manually, just print shorter version

- For testing by human beings you can select some substring of that uuid to be displayed.
- E.g. if you take four hex digits from the part that is fully random, the chance for getting same sequence is
  - 1 / 16^4 = one out of 65536
  - which is apparently enough for some initial manual testing
  - Still for human it’s rather easy to spot the values as different id:s and connect the one created e.g. by one client and shown by other part of the system as the same id
- Similarly in git we can safely use just the start of the commit hash to refer to a certain commit in history:
  - **ca82a6d**ff817ec66f44342007202690a93763949 => **ca82a6d**

Random generation is complicated for computers

- **Note:** Really random generation is difficult for exact logical digital devices like computers.
- Implementing your own would be working fine if you are a PhD specialized in cryptography or related mathematics.
  - And even then it would probably fail to be as close to random as the ones we get from established libraries.
- So let’s just use those from libraries.
  - Even they are faulty if computers are so fast that random values get the same computer time based seed
    - like we have seen before in programming.
    - We might get numerous same values in row or other kind of unwanted non-random patterns.
  - Basically we would need to create a single random generator, and continue to ask more randomized values from the same generator to get more real pseudo random values
  - (or make the system time change by adding so much delay that the used seed changes, but that’s us ’doctoring’ again, so not good …)
  - Of course if we are just creating some funny test values for an app we can accept non-fully random values