PROGRAMMING ON THE TOILET

What is it?

Programming on the Toilet is a document which contains tips, tricks, snippets or explanations about various programing technologies and paradigms, which could help developers in their everyday job. This episode was brought to you by Senadin Terović, Junior Software Engineer at Atlantbh.

Express middleware

Middleware functions are functions that have access to the request object (`req`), the response object (`res`), and the `next` function in the Express application's request-response cycle. The `next` function is a function in the Express router which, when invoked, executes the middleware succeeding the current middleware in the middleware chain. If the current middleware function does not end the request-response cycle, it must call `next()` to pass control to the next middleware function. Otherwise, the request will be left hanging.

```
var express = require('express');
var app = express();

// define middleware function
var loggerMiddleware = function(req, res, next) {
  console.log(`{req.method} ${req.url} — ${new Date()}`);
  next();
}
// bind middleware function to the instance of the app object
app.use(loggerMiddleware);
// every time the app receives a request, it prints some request details to
the     // terminal
app.get('/', function(req, res) {
  res.send('Hello, world!');
});
```

Memoization

In computing, memoization is an optimization technique used primarily to speed up computer programs by storing the results of expensive function calls and returning the cached result when the same inputs occur again. Dynamic programming and memoization work together.

```
// direct recursive solution
function fib(n) {
  if (n < 1) {
    return n;
  } else {
    return fib(n - 1) + fib(n - 2);
  }
}
```

```
// solution using memoization
var memo = {};
function fib(n) {
  if (n <= 1) {
    return n;
  }
  if (n in memo) {
    return memo[n];
  }
  memo[n] = fib(n - 1) + fib(n - 2);
  return memo[n];
}
```

Most of the time, referring to the previous calculation output is cheaper than recomputing in terms of CPU cycles and reduces the time complexity of the solution.
There is also a handy function available in the Underscore.js and Lodash library that relies on this technique.

```
var fibonacci = _.memoize(function(n) {
  return n < 2 ? n: fibonacci(n - 1) + fibonacci(n - 2);
});
```