0 Contents

1 NodeJS and other WebServers

This is the interface between the operating system and the actual website itself. -> Apache

- Runs everywhere
- async with events
- easy to deploy
- Lots of APIs for different usecases
  HTTP/HTTPS, URL, FileSystem, Console, UDP, Cryptography, DNS
- modular
- not that much "magic"
  you need to write more yourself

negative: IT IS JS, FFS...

1.1 Async via events

The idea of events is *essentially a queue that doesn't block actual functionality*, this means that your UI still does what it should, as it will be prioritized compared to async functionaity.
Just like bevy, the events are handled in the next frame, or more precise *whenever there is nothing else to do*.
This means we will get the illusion of async/multi-threading, without js actually being able to do so.

1.2 Callback hell with async events

If you constantly need to check for finished events, then you will end up with unreadable code.
The solution is simple: Promises
The default in NodeJS is still callbacks, but there is work on this to change.
Example for callback

```
button.addEventListener('click', function (event) {
  console.log("1. subscription");
});
```

1.3 Example for NodeJS

```
const http = require('node:http'); // lua?

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200; // HTTP response
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

2 Modules

2.1 NPM

- npm init
  - -g for global installation
  - –save-dev for a module that will only be used during development
-
- item 3
- item 4

2.2 Exporting Modules

```
function add() { return ++counter; }
function get() { return counter; }
// default export
export default {count: add, get: get};
// named
export {add, get};
```

2.3 Importing Modules

```
// default
import counterA from './counter.mjs';
// Named
import {add, get} from './counter.mjs';
```

2.4 package.json

```
{
  "name": "my_package",
  "description": "",
  "version": "1.0.0",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "https://github.com/mgfeller/my_package.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

```
  "bugs": {
    "url": "https://github.com/mgfeller/my_package/issues"
  },
  "homepage": "https://github.com/mgfeller/my_package"
}
```

2.5 package-lock.json

This handles dependency problems, where one dependency will be updated but the first package still wants an older one.

```
{
  "name": "my_package",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "fancy-calc-demo": {
      "version": "5.0.2",
      "resolved": "https://registry.npmjs.org/fancy-calc-demo/-/fancy-
      calc-demo-5.0.2.tgz",
      "integrity": "sha512-
      93xBMjZMU6HfGLXlwi1uYtQKL6eiNqOsWqkuFmlWMGJqKVBMF3+jb/dSrsRHrvpplIIDxU
      klwLNUzeZ5BTMjwQ=="
    }
  }
}
```

This *must also be in the git repository!*

2.6 Native Modules

These are modules that are performance critical and are therefore written in other languages such as c++.
The problem with this is that you will now need OS specific versions again!

2.7 nvm/nodenv

This is a tool to handle different node versions, you prob don't need this crap on arch, btw...

2.8 API versions

typically you will have 2 versions of a specific API, one that is synchronous, and one that is async.
The async will not throw an exception, as it is either handled with callbacks or with promises, the synchronous variant however will throw an exception if it fails.

```
// async
let fs = require('fs');
let path = "test.txt";
fs.readFile(path, function(err, content) {
  if (err) return console.error(err);
  console.log('content of file: %s\n', path);
  console.log(content.toString());
});
// the synchronous variant
// readFileSync

// note this specific problem can be done better with streams, as you might have continous input/ouput
let server = http.createServer(function (req, res) {
  let stream = fs.createReadStream(__dirname + '/data.txt');
  stream.pipe(res);
});
```