NODEJS: Crash Course

by
Sergii
Tsegelnyk

APPLICATION ARCHITECTURE

Start with npm init

```
Sergiis-MacBook-Pro-2:test-project serg$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (test-project)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
license: (ISC)
About to write to /Users/serg/projects/ciklum/rdss/fe/temp/test-project/package.json:

{
  "name": "test-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Sergii Tsegelnyk <dr3am3r.ua@gmail.com> (http://serhiy.co/)",
  "license": "ISC"
}


Is this ok? (yes)
```

Use a smart .npmrc

- create .npmrc file in a project root
- define needed things i.e.

```
Sergiis-MacBook-Pro-2:architecture serg$ cat .npmrc
save=true
save-exact=true
```

Stick with lowercase

- let MyClass = require('my-class');
- MyClass.js and myclass.js will be treated differently across platforms

Avoid garbage

- node (V8) uses a lazy and greedy garbage collector
- it sometimes waits until it absolutely has to before reclaiming unused memory
- you can provide flags for V8:
- node --optimize_for_size --max_old_space_size=920 --gc_interval=100 server.js
- for more control you can call GC manually, i.e.
- node --expose-gc server.js

Use Clusterisation

- node runtime is limited to a single CPU core
- and about 1.5 GB of memory
- on a large server bake Cluster support into your app
- choose a cluster abstraction for your needs, i.e. forky, throng, etc

Be environmentally aware

- leverage the usage of environment variables with .env
- best practice: DO NOT put it in git
- use loader for it, like dotenv (npm i dotenv)

Utilise npm’s lifecycle scripts

- before: `preinstall`
- after: `postinstall`
- "postinstall": "if [$BUILD_ASSETS]; then npm run build-assets; fi"
- "build-assets": "bower install && grunt build"

Remove shit from git

- create .gitignore and put unnecessary stuff there
- i.e. node_modules, npm-debug.log, etc

Use CommonJS Power

- Create index.js in the root of a directory
- require('./path/to/dir')
- alternative: NODE_PATH=.
- then anywhere require('app/stuff/morestuff')

Encapsulate reusable parts

- Keep them in a separate repository
- Cover with tests
- Declare in package.json
- Versioning as a bonus

Consuming modules

- Define your way (callback, promises, generators)
- Keep it consistent throughout an Application
- Jsdoc generation will save time for newcomers

File structure

- DO NOT overcomplicate: keep it relevant
- By component, by type, mixed
- Number of layers depends on the App complexity

Examples in ExpressJs

- Routes-Handlers-Models
- Routes-Handlers-Services-Models
- Routes-Handlers-Services-MoreServices-…
- take a look at KrakenJs

Use general rules of good coding

- Keep functions small and testable
- Keep modules small
- If something gets too big - separate it
- Apply coding standards