NodeJS

środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

Rok akademicki: 2023/2024, Semestr: 1, Wykład: 2

In modern software development
you often use ready pieces
to create something new…

…and this is perfectly ok.

Build software from pieces

Those pieces in Node.js are called „packages” and „modules”.

A **package** is a file or directory that is described by a package.json file.

A **module** is any file or directory in the node_modules directory that can be loaded by the Node.js require() function.

You can use packages/modules to:

- use code written by others
- reuse code between projects
- share code with others

Package manager

A package manager is a programming language's tool to create project environments and easily import external dependencies.

Why package manager is useful:

- lot of packages in project
- indirect references
- versioning

node package manager

npm is a command-line application that helps you install modules available in the repository.

node package manager - the default package manager for the Node.js environment.

npm

npm is a:

- website https://www.npmjs.com
- CLI (Command Line Interface)
- company name

npm is part of Node.js but is updated more frequently then Node.js

You can update npm using npm ☺

npm doesn’t control quality of hosted packages!

npm all commands

access
adduser
audit
bugs
cache
ci
completion
config
dedupe
deprecate
diff
dist-tag
docs
doctor
edit
exec
explain
explore
find-dupes
fund
help
help-search
hook
init
install
install-ci-test
install-test
link
logout
ls
org
outdated
owner
pack
ping
pkg
prefix
profile
prune
publish
query
rebuild
repo
restart
root
run-script
sbom
search
shrinkwrap
star
stars
start
stop
team
test
token
uninstall
unpublish
unstar
update
version
view
whoami

npm basic commands

https://docs.npmjs.com/cli/v9/commands

No autocomplete by default ☹

> npm -v

> npm -l

> npm help

> npm <command> -help (or -h)

> npm help <command>

npm analytic commands

> npm view <package_name>

> npm view <package_name> versions

> npm view <package_name> dependencies

> npm bugs <package_name>

> npm docs <package_name>

> npm search <package_name_part>

npm init

> npm init

> npm init -y

npm file: package.json

This file holds various metadata relevant to the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

It can also contain other metadata such as a project description, the version of the project in a particular distribution, license information, even configuration data - all of which can be vital to both npm and to the end users of the package.

The package.json file is normally located at the root directory of a Node.js project.

Without this file in your project you are unable

to install dependencies using npm!

npm install

> npm install <package_name>

> npm install <package_name>@<version>

> npm install <package_name> --save //default behavior since npm 5

> npm install <package_name> --no-save

> npm install <package_name> --save-dev

> npm install <package_name> --save-prod

> npm install <package_name> --global // global install is mostly for tools

> npm install

Installation is just copying files!

npm update / uninstall

> npm update

> npm update <package_name>

> npm outdated //which packages will be updated

> npm uninstall <package_name>

npm maintenance

> npm audit

> npm audit fix

> npm prune

> npm ls

> npm ping

> npm doctor

Publish own package

It’s simple to publish a package onto npm.

There are few steps:

> npm init

// create your magnificent package

// create account on npmjs.com

> npm login

> npm publish

> npm logout //optional

https://zellwk.com/blog/publish-to-npm/

Semantic versioning

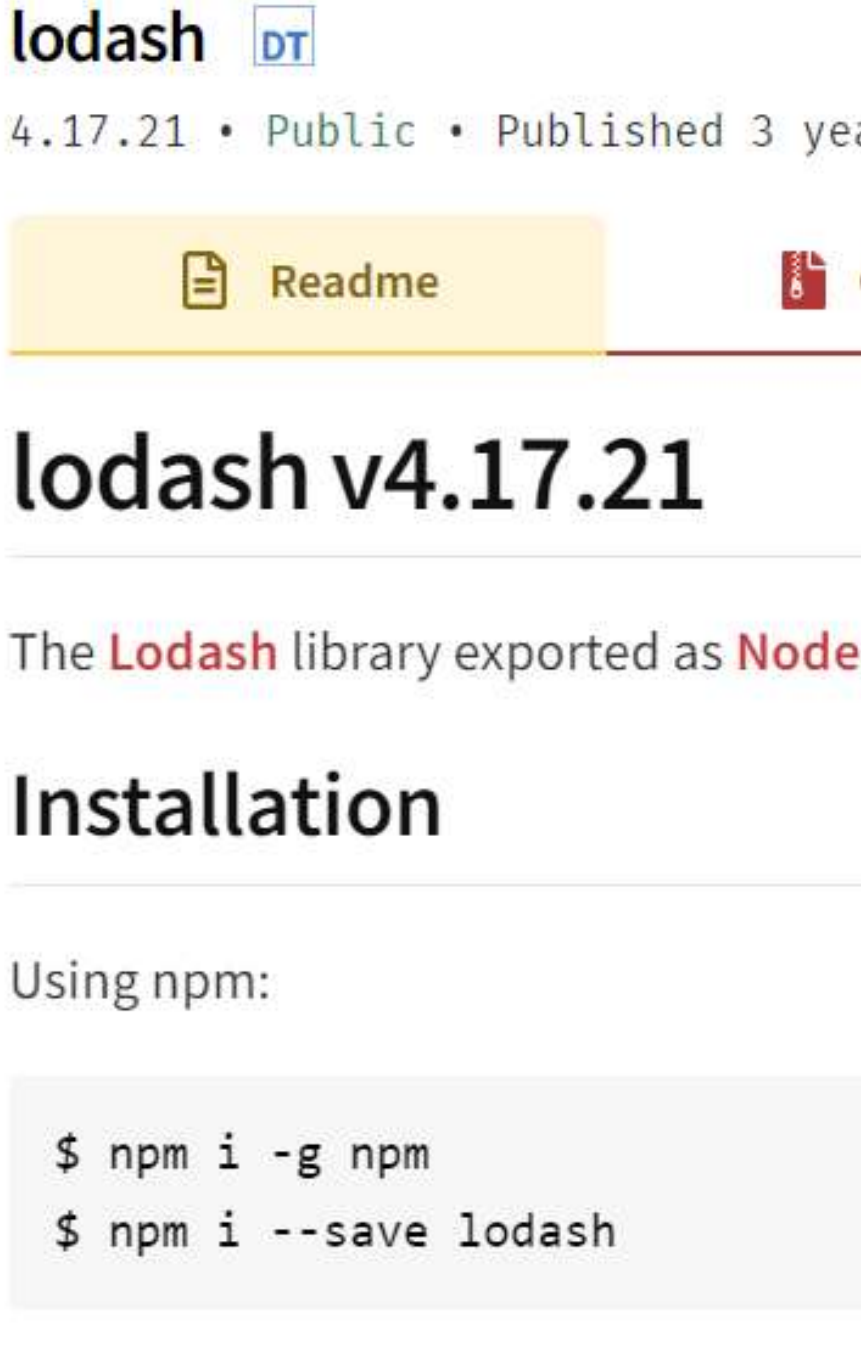
It's just convention

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards compatible manner, and
- PATCH version when you make backwards compatible bug fixes.

Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.

https://semver.org



Semantic versioning in npm

~version “Approximately equivalent to version”

will update you to all future patch versions, without incrementing the minor version.

Example: ~1.2.3 will use releases from 1.2.3 to <1.3.0.

^version “Compatible with version”

will update you to all future minor/patch versions, without incrementing the major version.

Example: ^2.3.4 will use releases from 2.3.4 to <3.0.0.

https://semver.npmjs.com

npm file: package-lock.json

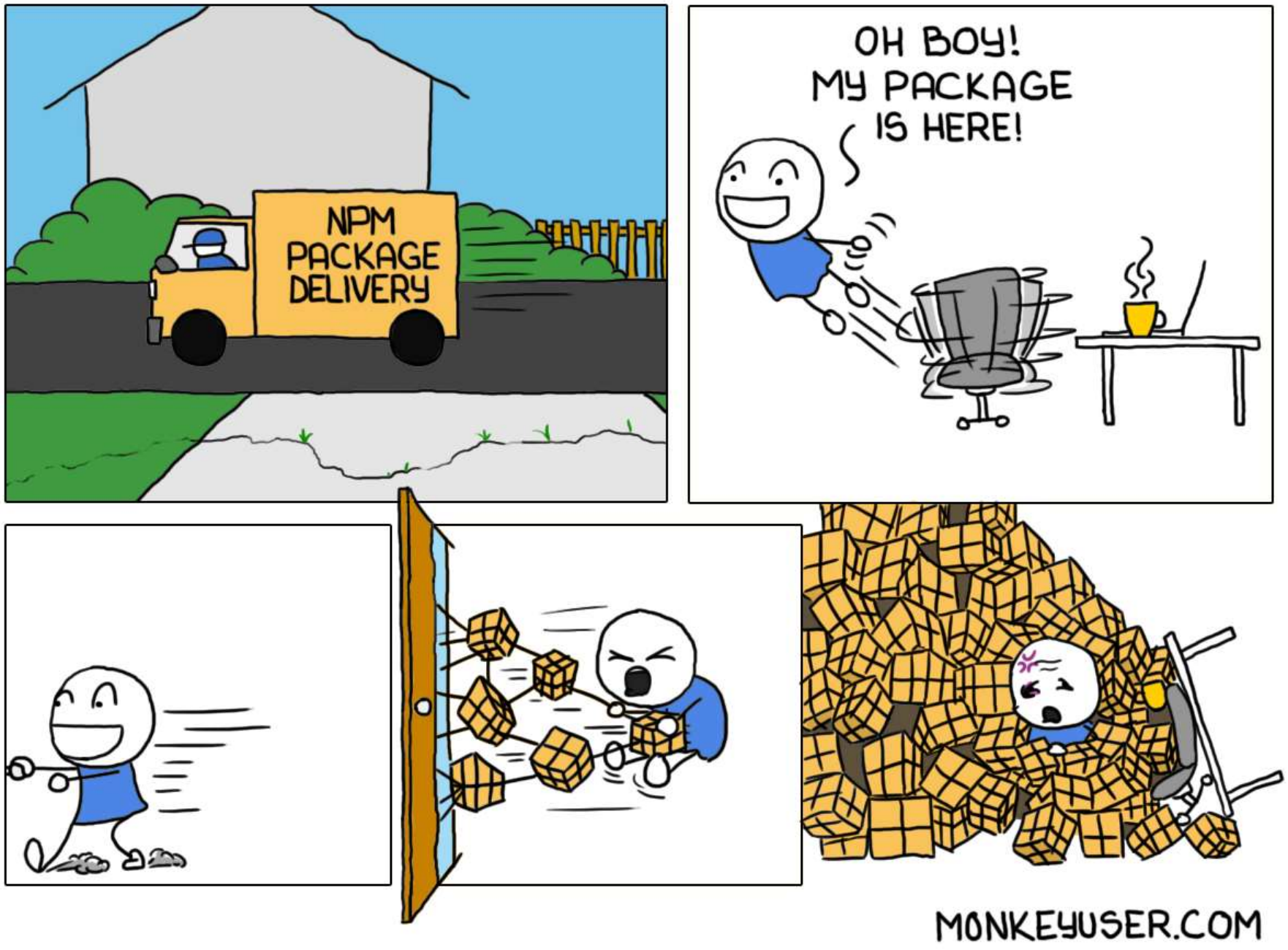
The goal of the file is to keep track of the exact version of every package that is installed so that a product is 100% reproducible in the same way even if packages are updated by their maintainers.

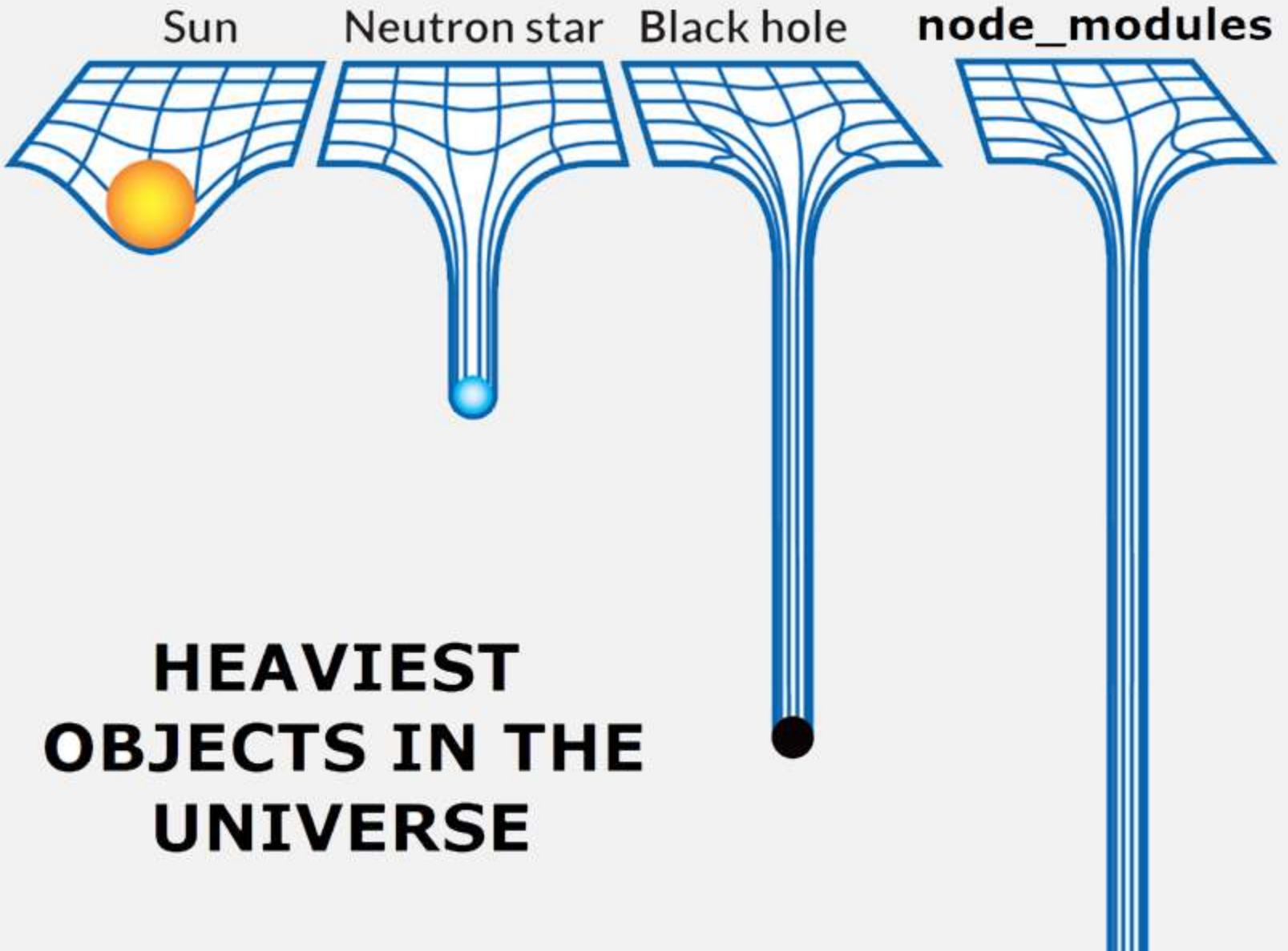
You don't commit to Git your node_modules folder. When you try to replicate the project on another machine by using the npm install command, if you specified the ~ or ^ syntax new version of packages can be installed.

So your original project and the newly initialized project can be different. Even if a patch or minor release should not introduce breaking changes, we all know bugs can (and so, they will) slide in.

The package-lock.json sets your currently installed version of each package in stone, and npm will use those exact versions when running npm install.

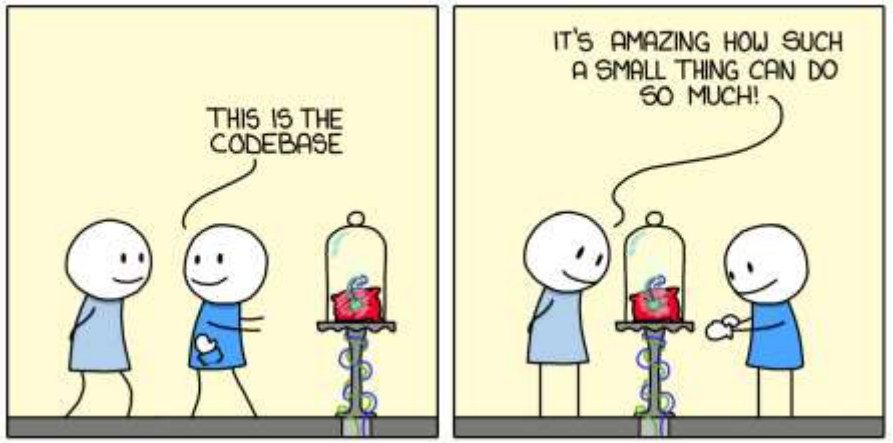
NPM DELIVERY

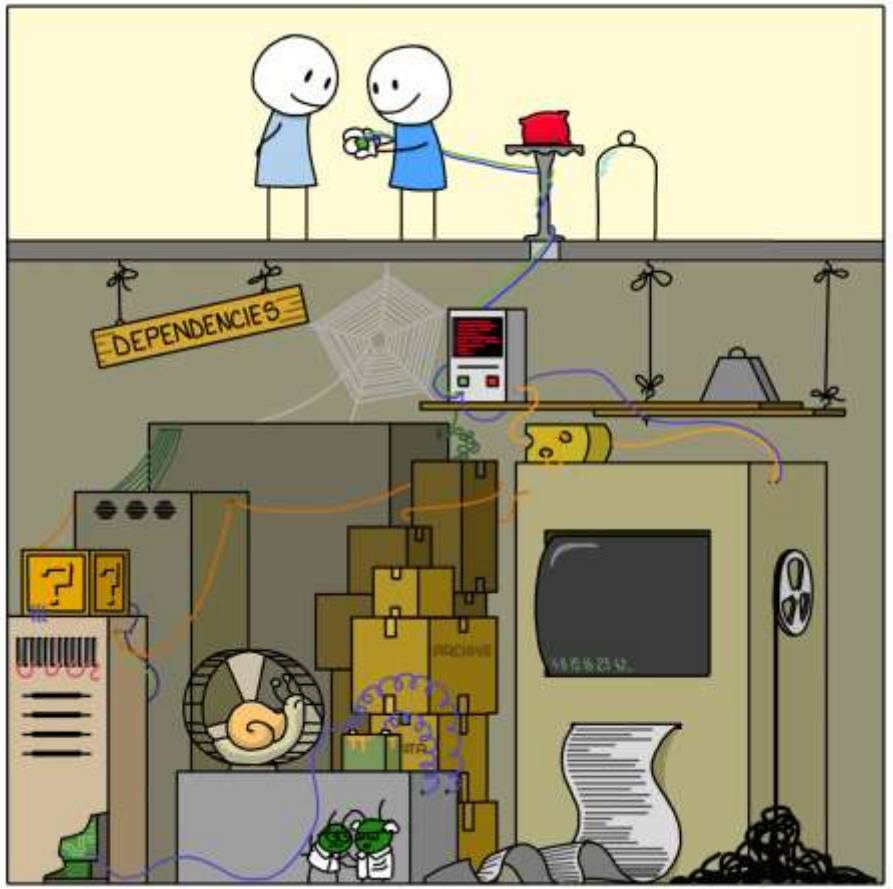




HEAVIEST OBJECTS IN THE UNIVERSE

IMPLEMENTATION





MONKEYUSER.COM

Dillema: own code or package?

Ask yourself few questions about package:

- what is license?
- is it compatible with my application?
- how well known it is?
- when was last release?
- how long will it be maintained?
- how it affect application size?
- how it affect application performance?
- is it safe?
- is it bug-free?
- how often will be used?
- what part will be used?
- [insert your concern here]

npm & Git

Should this be added to version control system?

- package.json – always (when you use npm)
- package-lock.json – yes (in 95% cases)
- node_modules – no (in 70% cases)

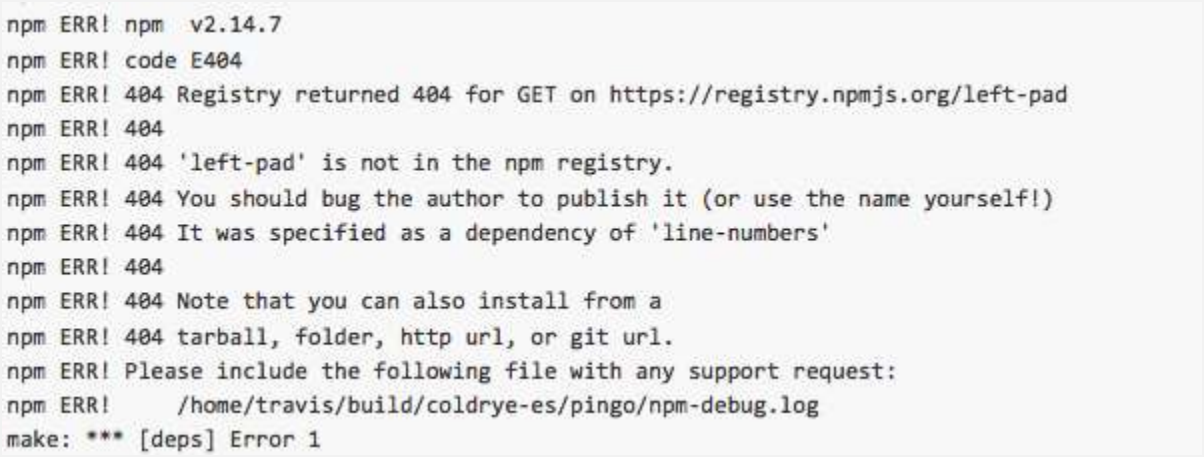
History of „leftpad” package

https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/

https://www.theregister.com/2016/03/23/npm_left_pad_chaos/

https://www.davidhaney.io/npm-left-pad-have-we-forgotten-how-to-program/

https://arstechnica.com/information-technology/2016/03/rage-quit-coder-unpublished-17-lines-of-javascript-and-broke-the-internet/

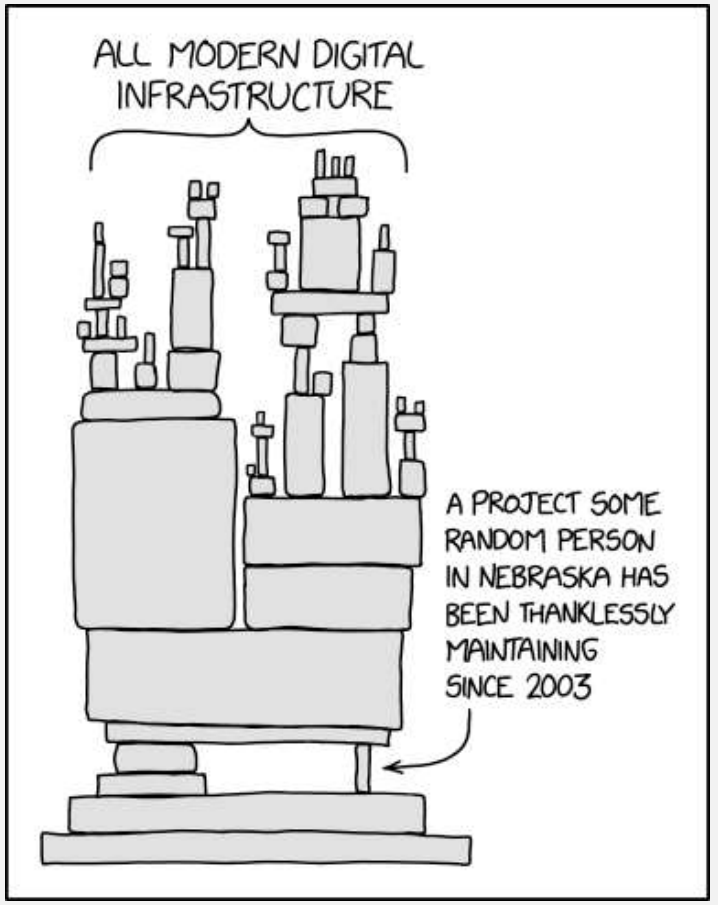


Issues with npm and free software

https://sekurak.pl/malware-w-dwoch-pakietach-npm-majacych-23-miliony-pobran-tygodniowo/

https://sekurak.pl/giga-korporacja-bierze-na-spytki-tworce-curla-bo-z-niego-korzysta-p-odpowiedz-nam-w-24h/

https://fossa.com/blog/npm-packages-colors-faker-corrupted/



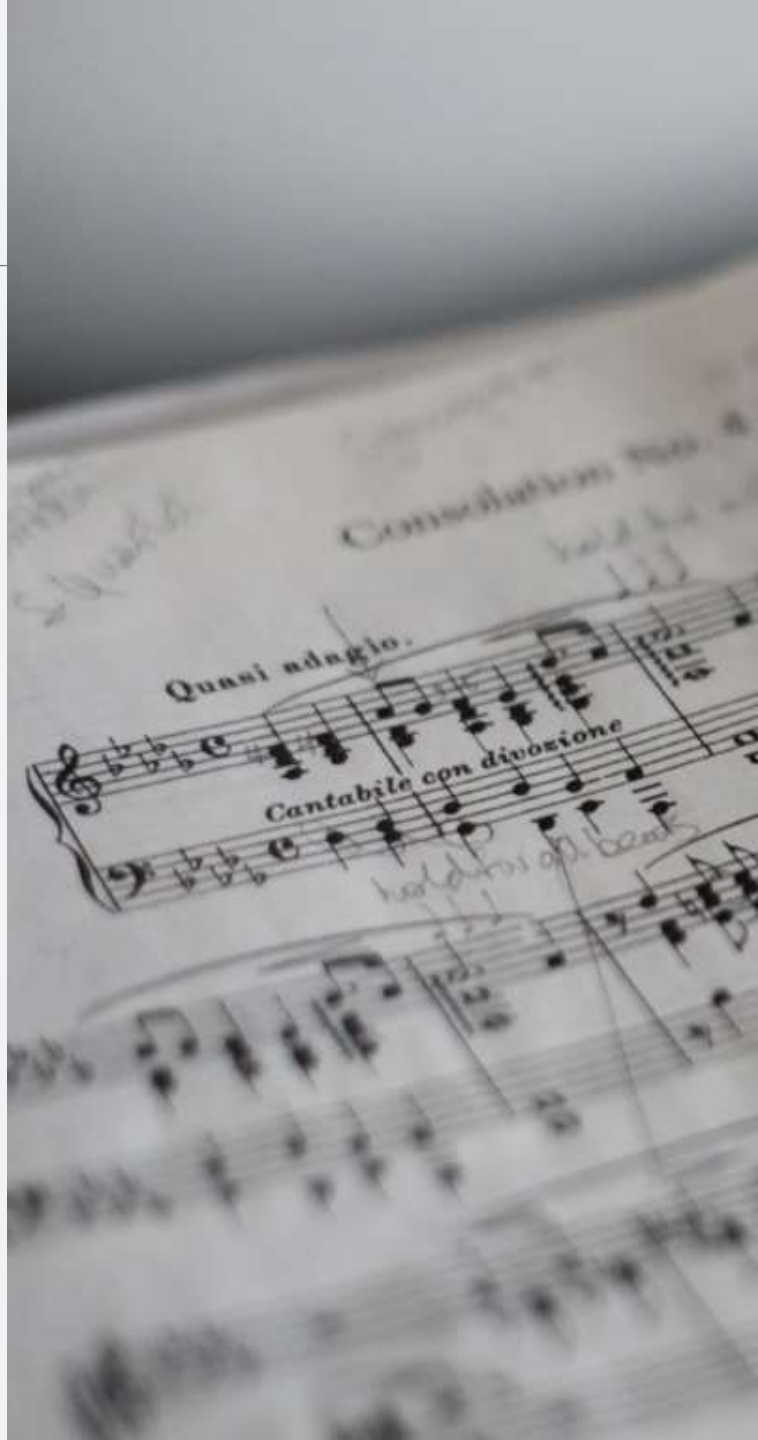

{JSON}
JavaScript Object Notation

JSON

JavaScript Object Notation, or JSON, is a lightweight data format that has become the defacto standard for the web.

JSON can be represented as either a list of values, e.g. an Array, or a hash of properties and values, e.g. an Object.

It is based upon JavaScript syntax but is distinct from it: some JavaScript is not JSON.



```
{
    "squadName": "Super hero squad",
    "formed": 2016,
    "active": true,
    "members":
    [
      {
        "name": "Atom Man",
        "age": 29,
        "powers":
        [
          "Radiation resistance",
          "Radiation blast"
        ]
      }
    ]
}
```

JSON.parse

```
const userString = '{ "id": 12, "name": "Jan", "age": 35 }';

const user = JSON.parse(userString);

console.log(userString);

console.log(userString.id);

console.log(user);

console.log(user.id)
```

JSON.parse

```
const userString = '{ "id": 12, "name": "Jan", "age": 35 }';
const user = JSON.parse(userString);
console.log(userString);
console.log(userString.id);
console.log(user);
console.log(user.id)
```

```
'{ "id": 12, "name": "Jan", "age": 35 }'
undefined
{id: 12, name: 'Jan', age: 35}
12
```

JSON.stringify

```
const user = { id: 12, name: "Jan", age: 25};

const userString = JSON.stringify(user);

console.log(user);

console.log(user.id);

console.log(userString);

console.log(userString.id);
```

JSON.stringify

```
const user = { id: 12, name: "Jan", age: 25};
const userString = JSON.stringify(user);
console.log(user);
console.log(user.id);
console.log(userString);
console.log(userString.id);
```

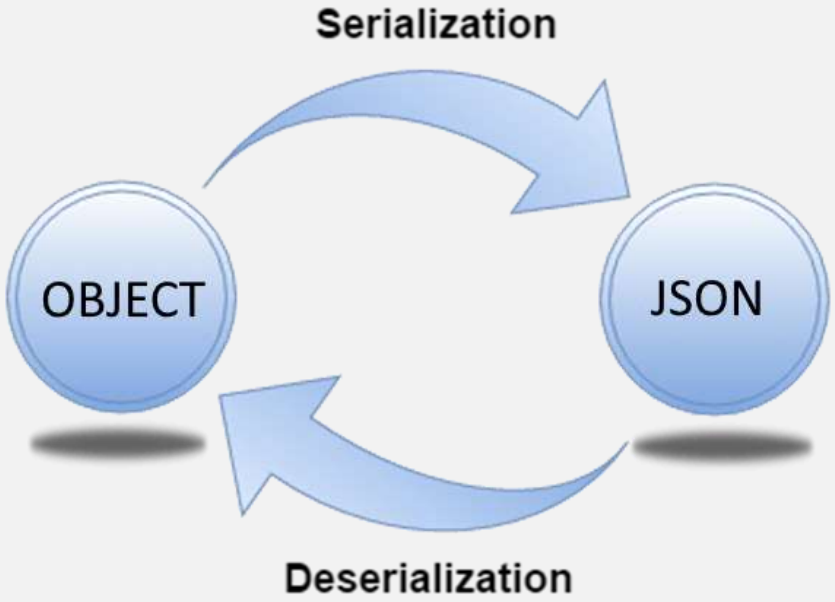
```
{id: 12, name: 'Jan', age: 25}
12
'{"id":12,"name":"Jan","age":25}'
undefined
```

Serialization

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file.

The reverse process is called deserialization.

Serialization allows the developer to save the state of an object and re-create it as needed, providing storage of objects as well as data exchange.



Debugging

https://nodejs.org/en/docs/guides/debugging-getting-started/

Possibilities:

- inspect
- --inspect & DevTools
- IDE

Inside code you can use JavaScript „debugger” instruction!

Node inspect

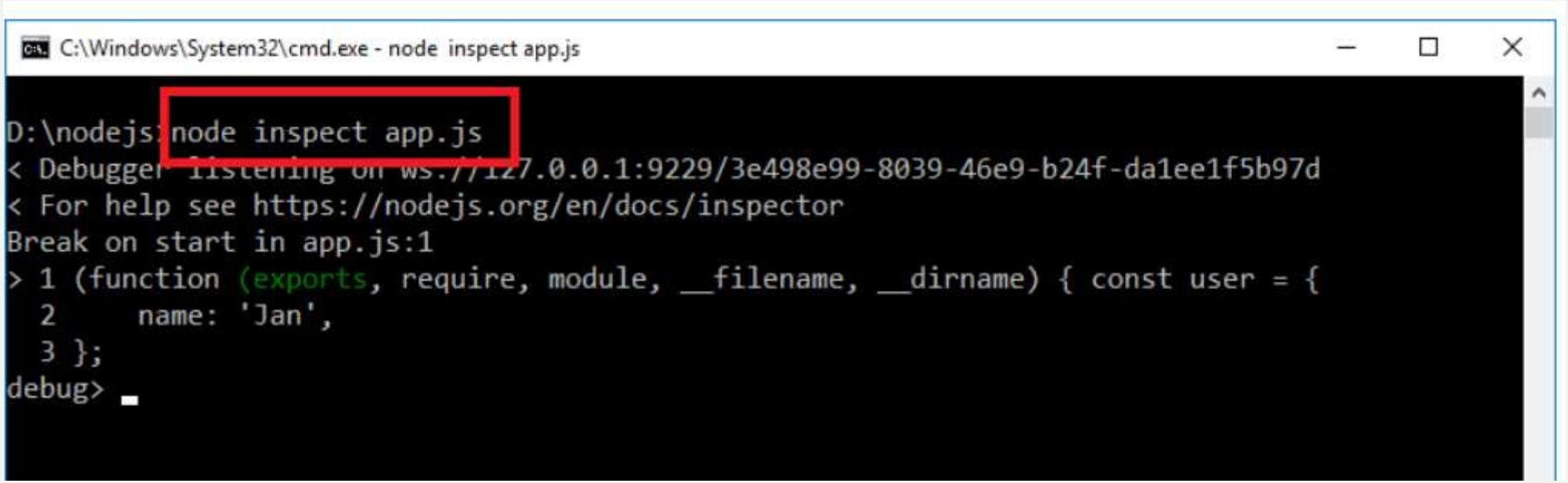
Start the interactive CLI debugger that's bundled with Node.js

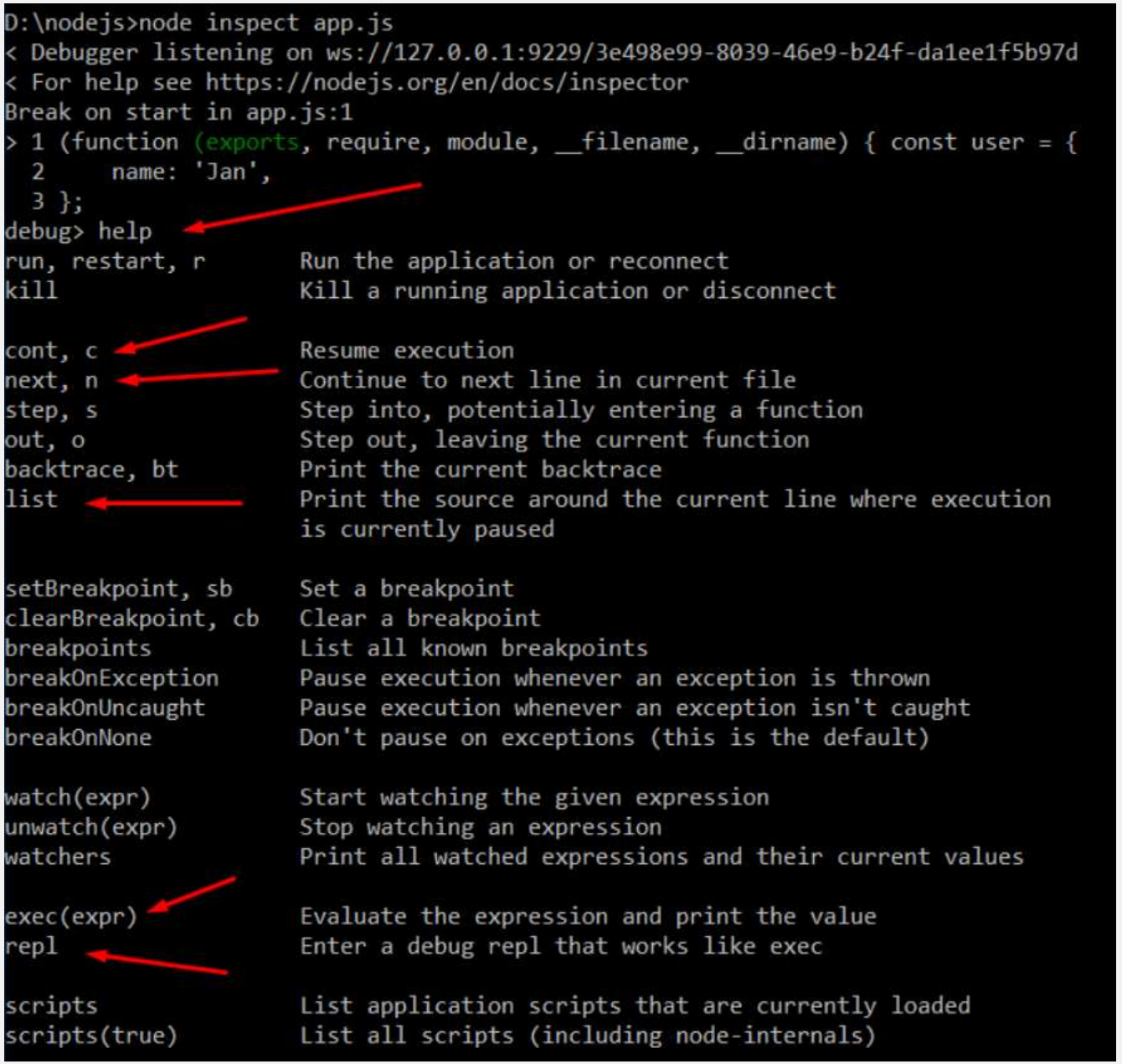
The CLI debugger will then use Node.js to start the script in a separate process and attaches to it.

> node inspect app.js

> node inspect --port=xxxx app.js

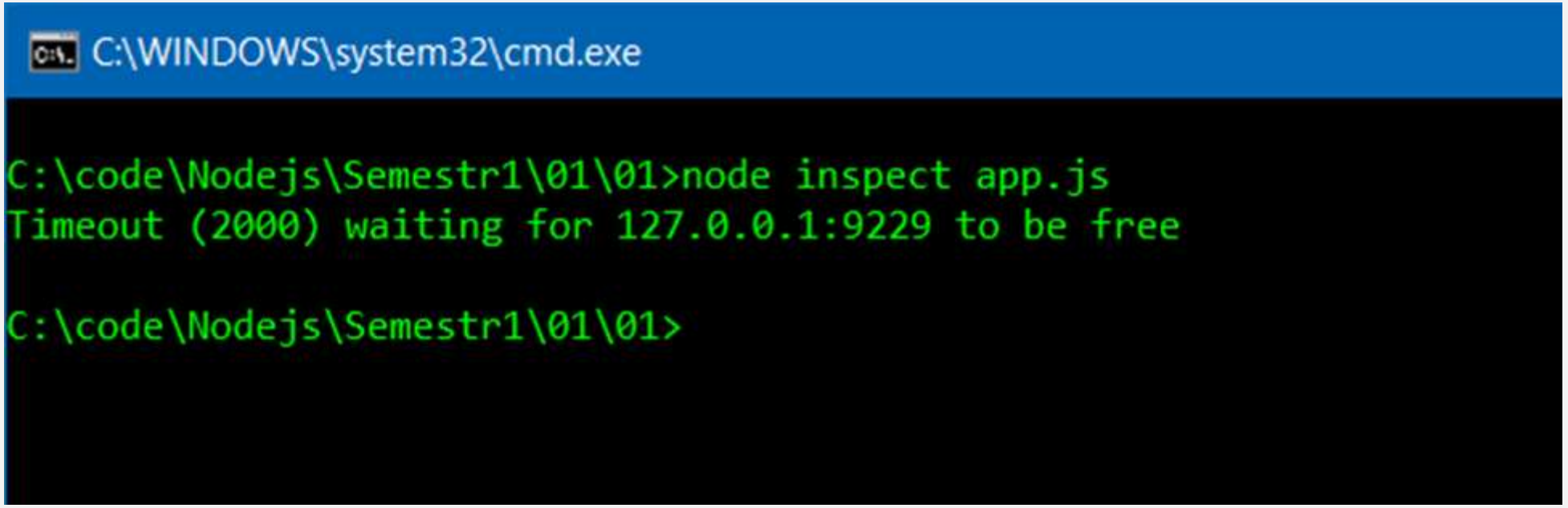
Node inspect




D:\nodejs>node inspect app.js
< Debugger listening on ws://127.0.0.1:9229/3e498e99-8039-46e9-b24f-da1ee1f5b97d
< For help see https://nodejs.org/en/docs/inspector
Break on start in app.js:1
> 1 (function (exports, require, module, __filename, __dirname) { const user = {
2 name: 'Jan',
3 };
debug> help
run, restart, r Run the application or reconnect
kill Kill a running application or disconnect
cont, c Resume execution
next, n Continue to next line in current file
step, s Step into, potentially entering a function
out, o Step out, leaving the current function
backtrace, bt Print the current backtrace
list Print the source around the current line where execution
is currently paused
setBreakpoint, sb Set a breakpoint
clearBreakpoint, cb Clear a breakpoint
breakpoints List all known breakpoints
breakOnException Pause execution whenever an exception is thrown
breakOnUncaught Pause execution whenever an exception isn't caught
breakOnNone Don't pause on exceptions (this is the default)
watch(expr) Start watching the given expression
unwatch(expr) Stop watching an expression
watchers Print all watched expressions and their current values
exec(expr) Evaluate the expression and print the value
repl Enter a debug repl that works like exec
scripts List application scripts that are currently loaded
scripts(true) List all scripts (including node-internals)

Node inspect

Sometimes it does not work ☹ - timeout when opening port



https://github.com/nodejs/node-inspect/issues/48#issuecomment-520246415

Node --inspect

Run application but expose the remote debugging interface that things like VS Code, Chrome DevTools, WebStorm, etc. can attach to.

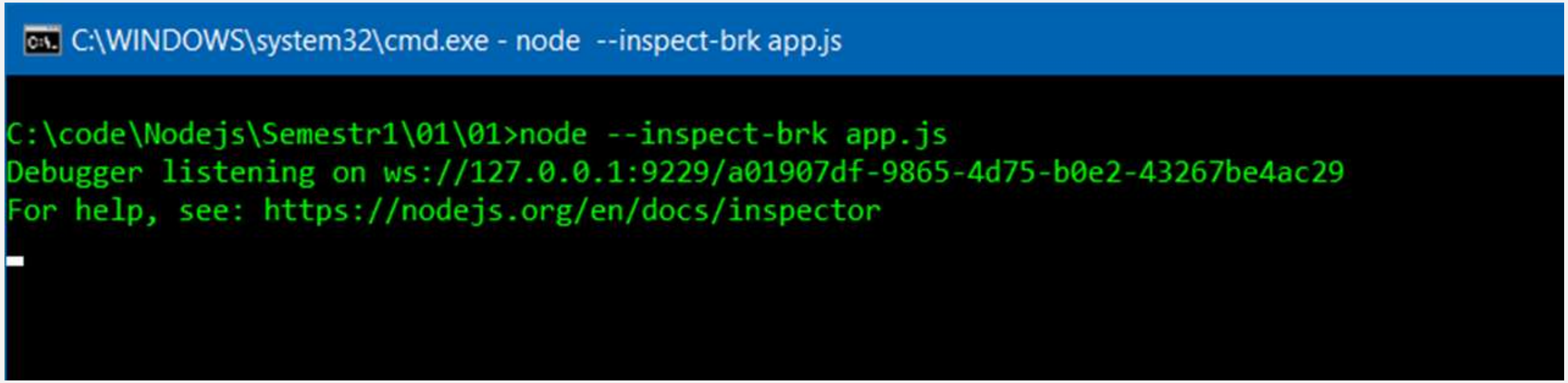
> node --inspect app.js

> node --inspect=[host:port] app.js

> node --inspect=127.0.0.1:5050 app.js

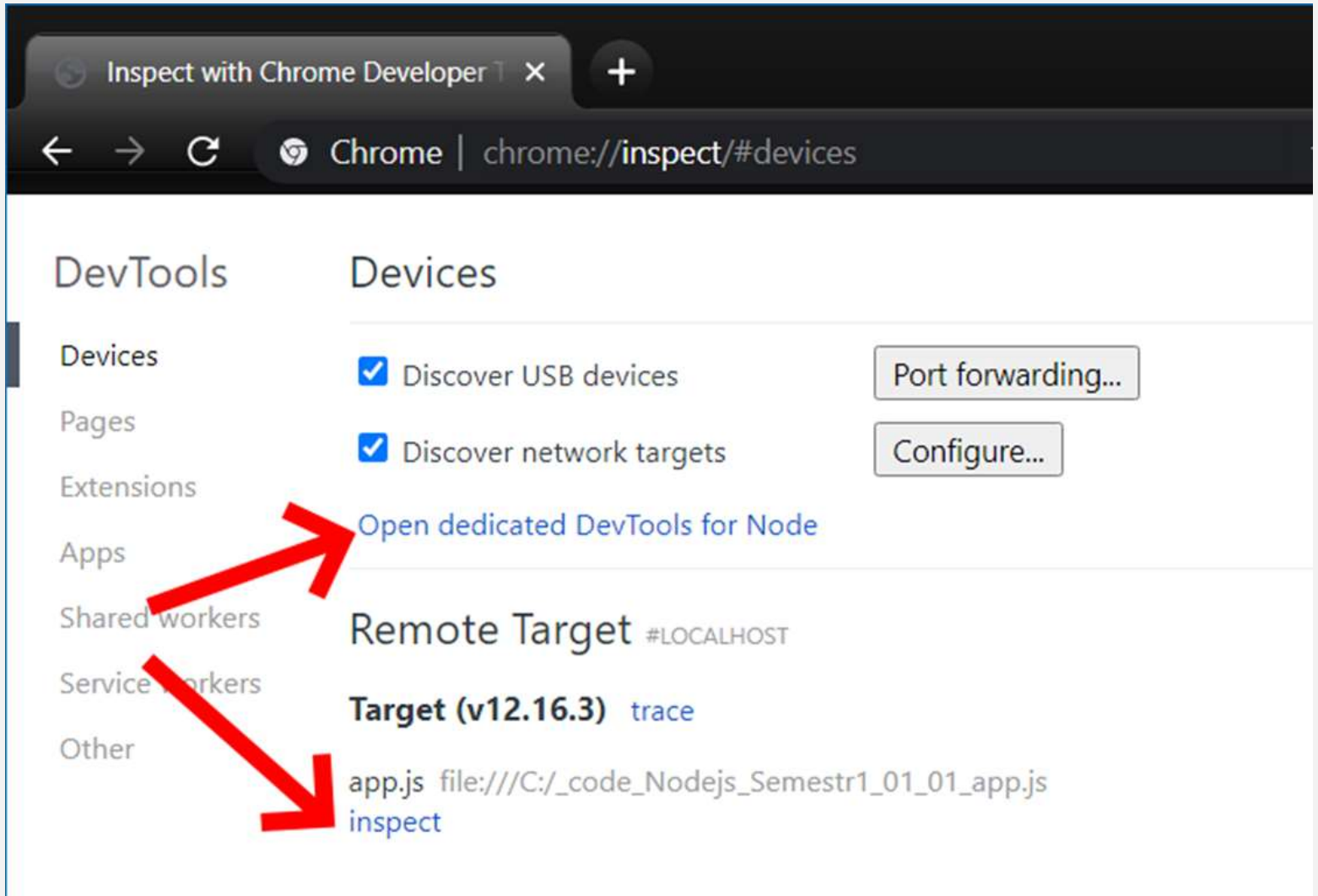
> node --inspect-brk app.js //break before user code starts

Node --inspect commandline

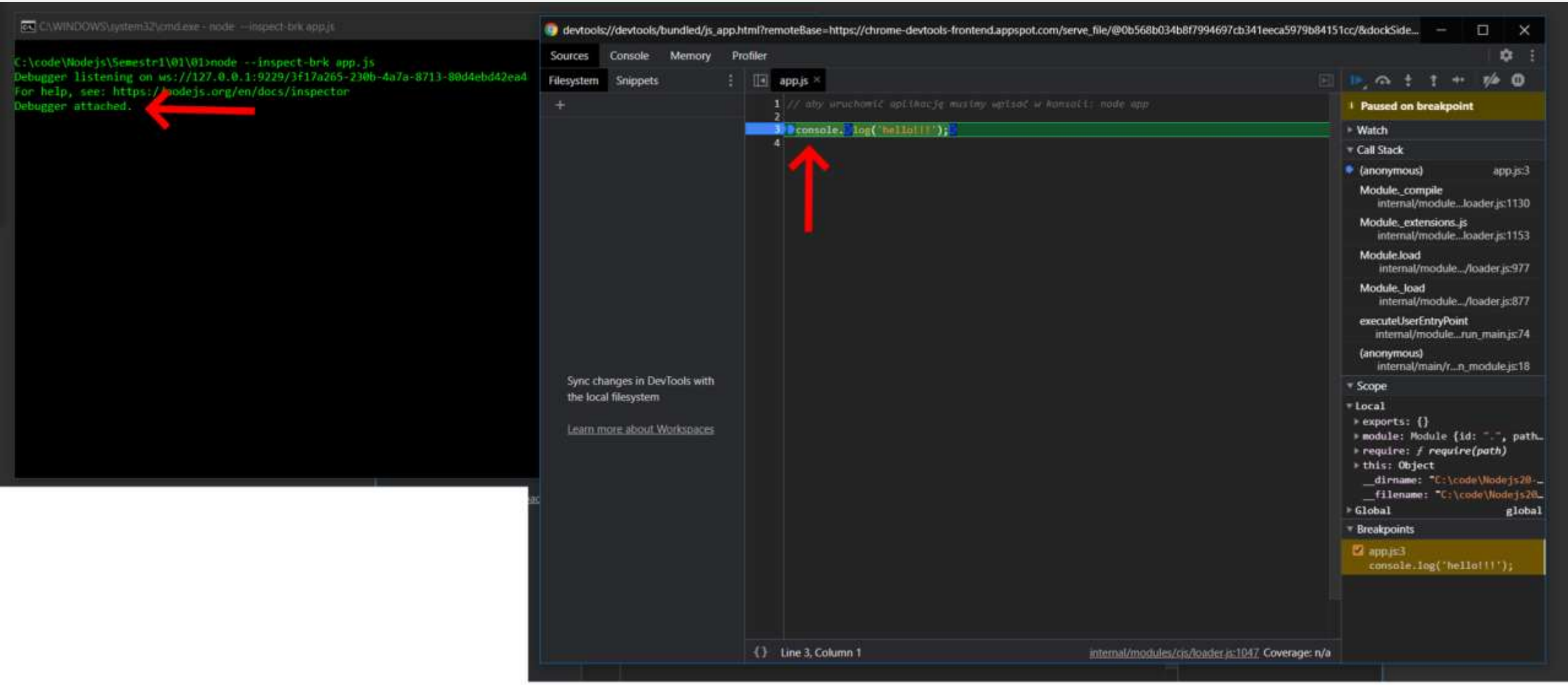


Debugging in Chrome DevTools

User Chrome to open page: chrome://inspect/ and click „inspect” or „Open dedicated…”



Node --inspect debugging



Security warning

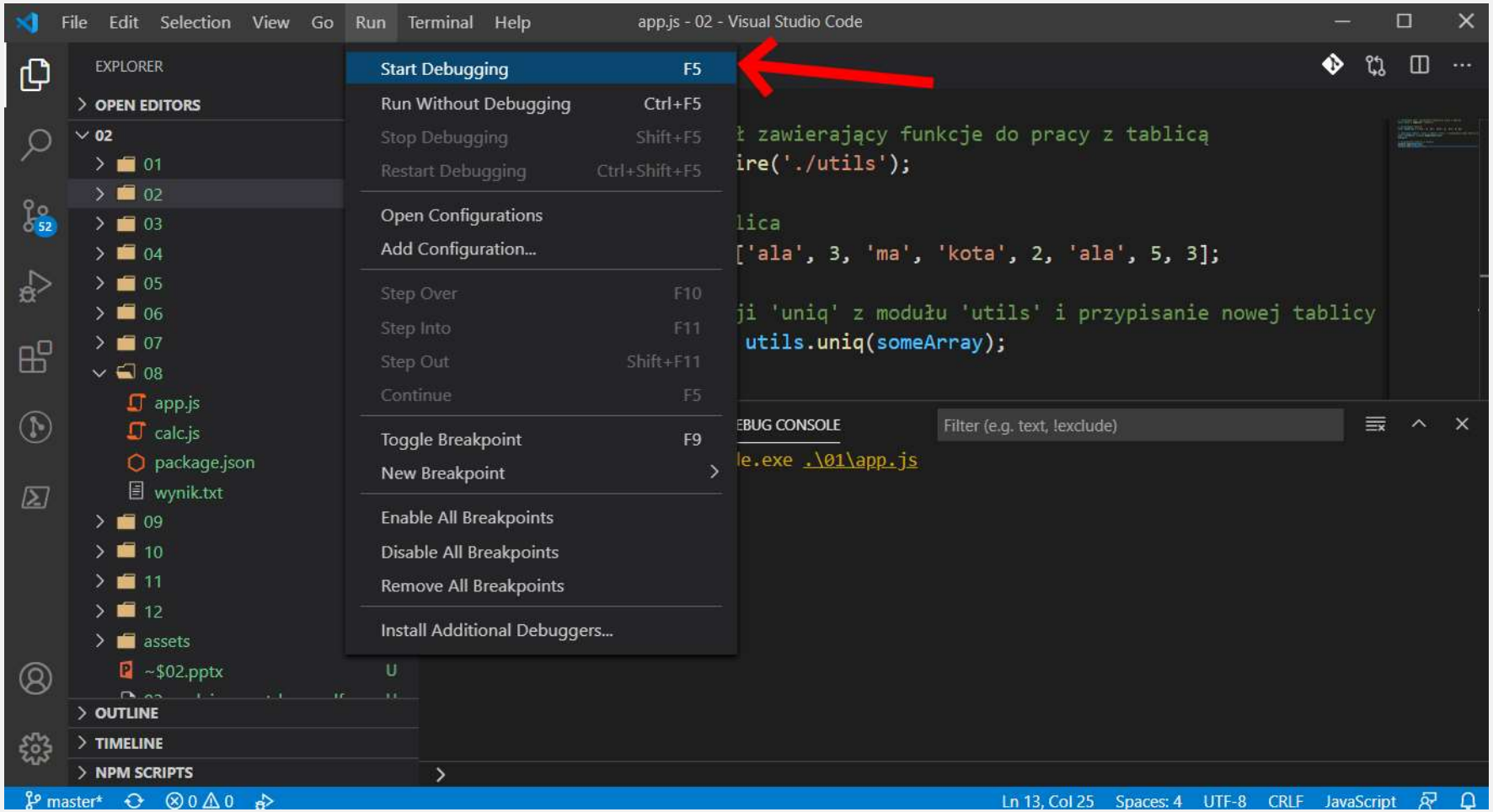
Since the debugger has full access to the Node.js execution environment, a malicious actor able to connect to this port may be able to execute arbitrary code on behalf of the Node.js process.

It is worth to know the security implications of exposing the debugger port on public and private networks.

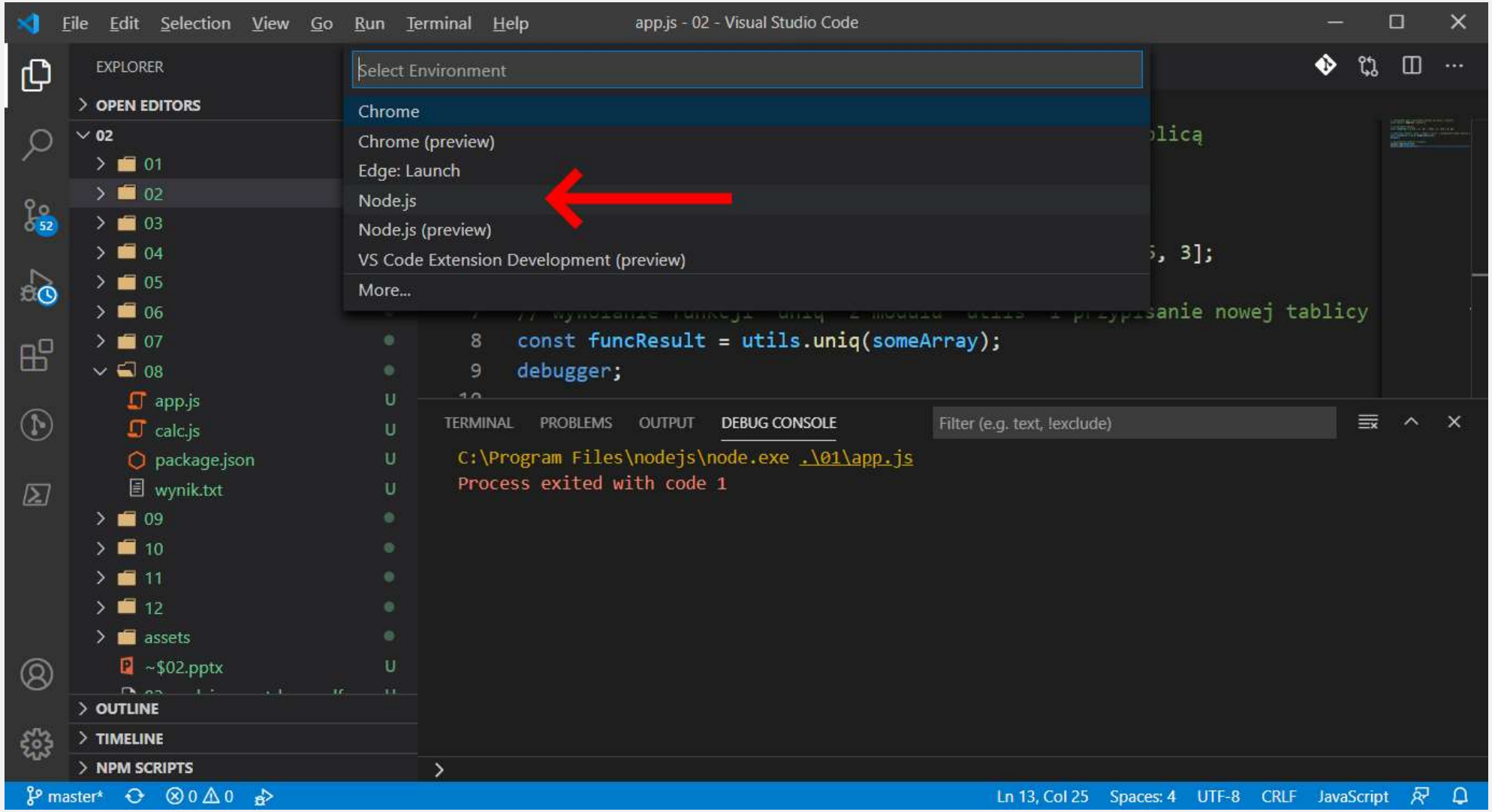
Debugging in IDE

You can debug your Node application in IDE – VS Code, Webstorm etc.

First approach – start debug session

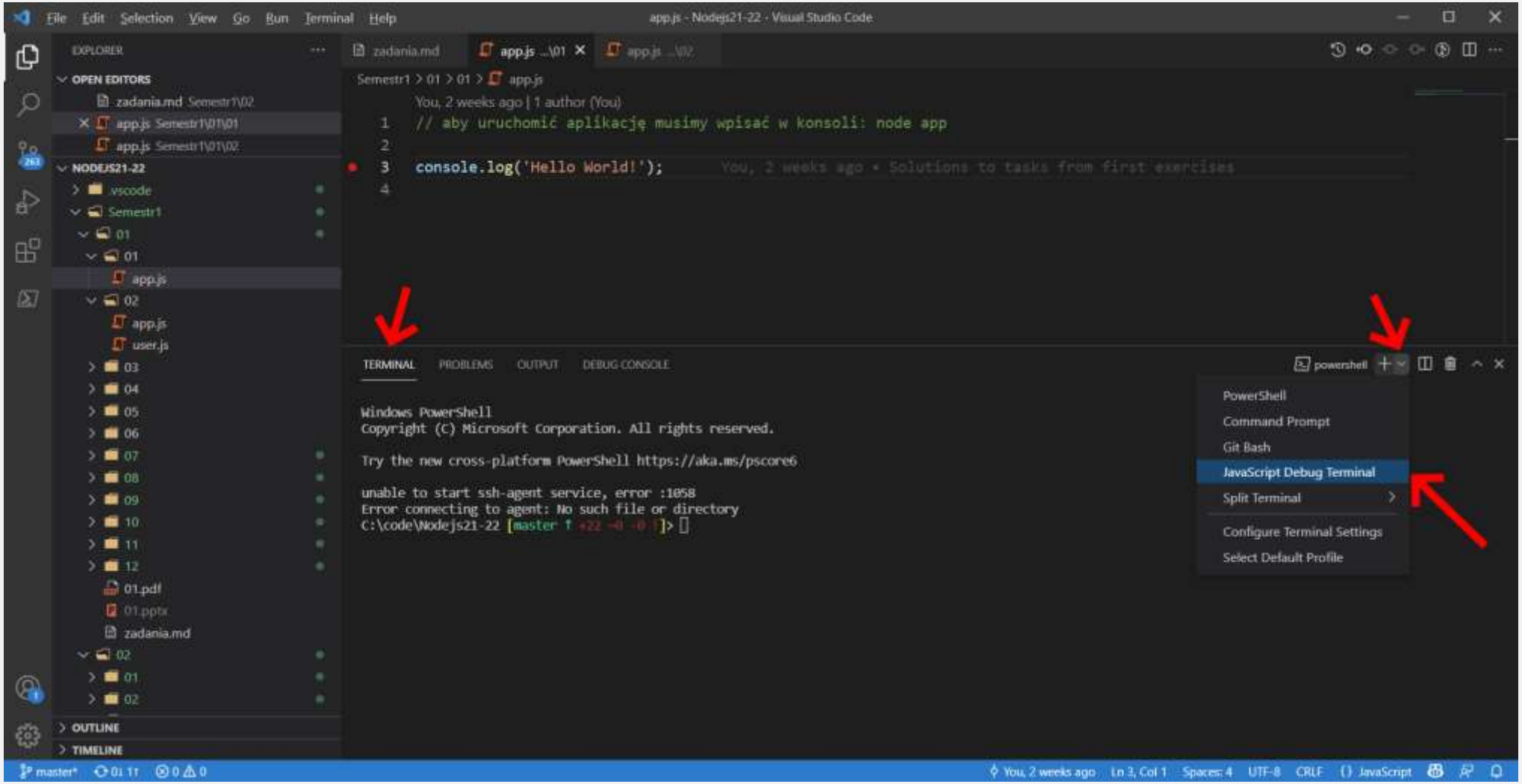


Debugging in IDE



Debugging in IDE

Second approach – use debug terminal



Debugging in IDE