

NodeJS

środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

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”.

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

You can use packages 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 needed:

- 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 than Node.js

You can update npm using npm 😊

npm doesn't control quality of hosted packages!



npm all commands

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| bundle | fund | outdated | run-script | update |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

npm basic commands

<https://docs.npmjs.com/cli/v6/commands>

No autocomplete by default 😞

> npm -v

> npm help

> npm init



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 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 doctor



npm install

- > npm install <package_name>
- > npm install
- > npm install <package_name>@<version>
- > npm install --save //default behavior since npm 5
- > npm install --no-save
- > npm install --save-dev
- > npm install --save-prod
- > npm install --global // global install is mostly for tools

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



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>

Miss

lodash

4.17.20 • Public • Published 3 months ago

 [Readme](#)

 [Explore](#) BETA

lodash v4.17.20

The **Lodash** library exported as **Node.js** modules.

Installation

Using npm:

```
$ npm i -g npm  
$ npm i --save lodash
```

In Node.js:

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.

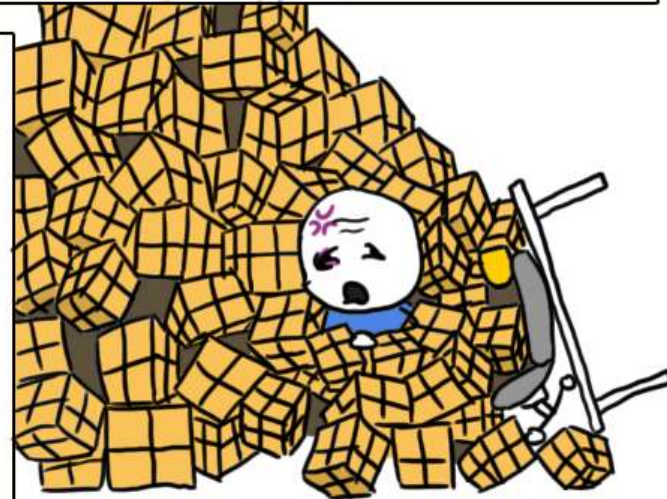
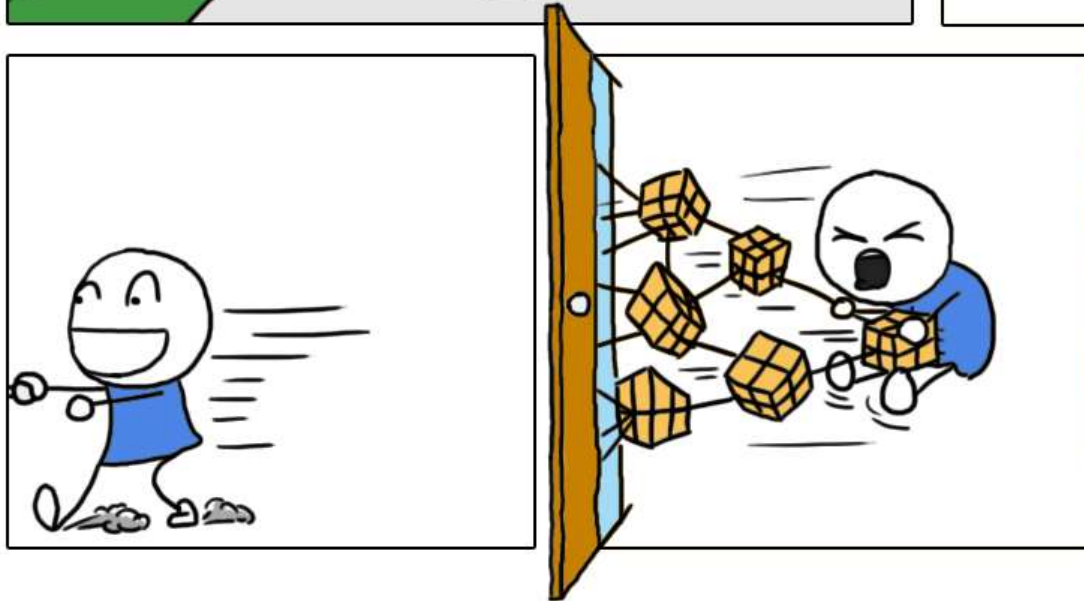
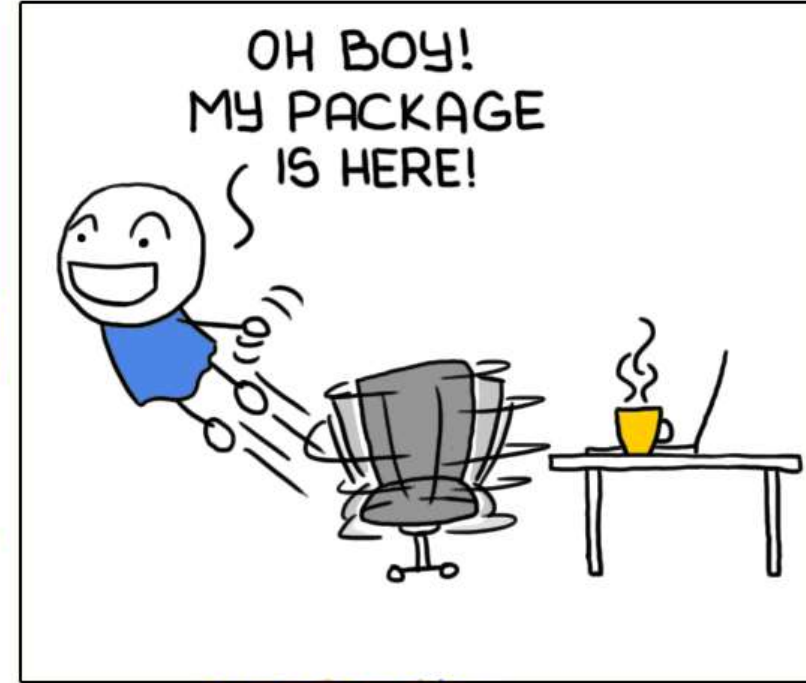
You don't commit to Git your `node_modules` folder. When you try to replicate the project on another machine by using the `npm install` command, if you specified the `~` or `^` syntax new version of packages can be installed.

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.

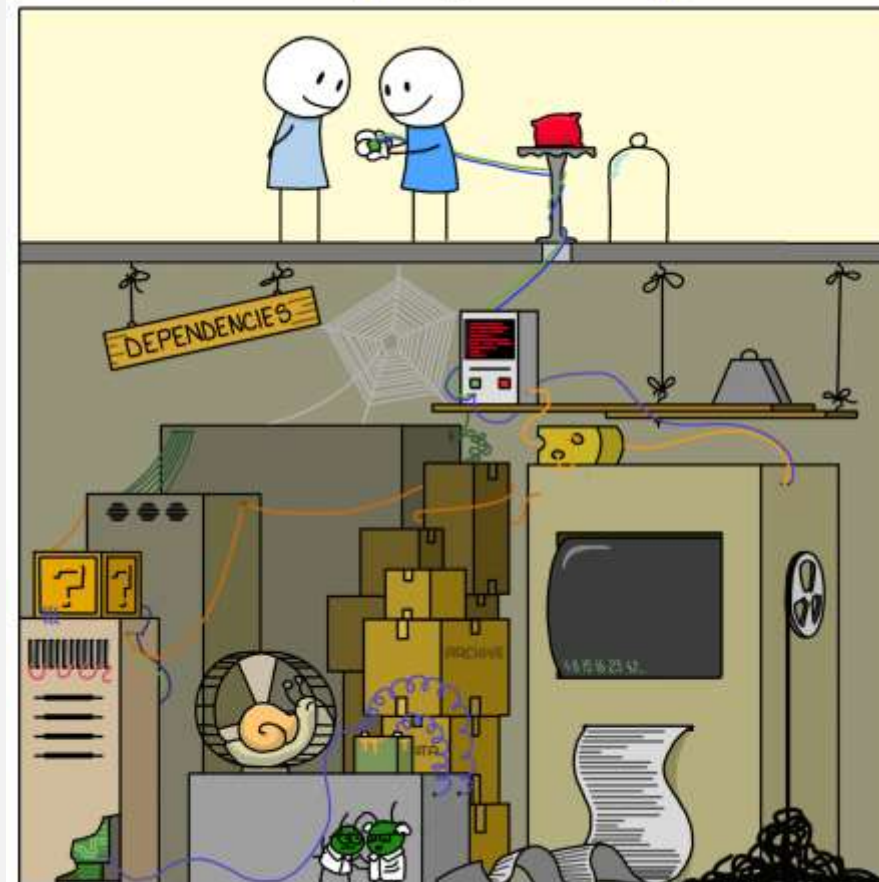
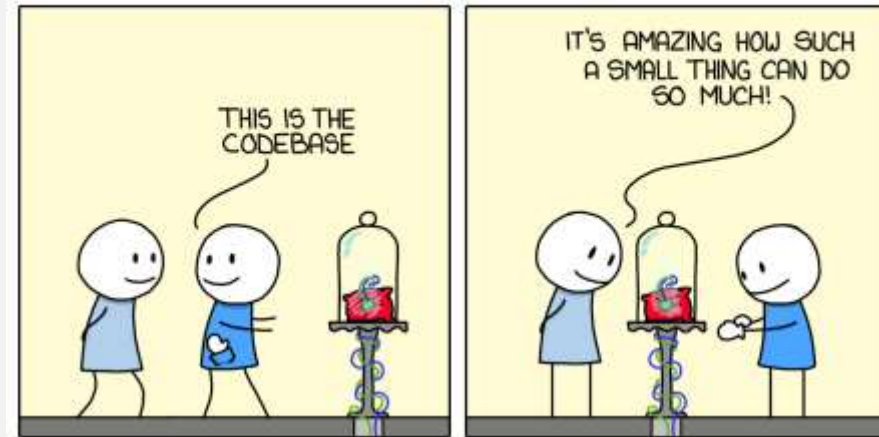
The `package-lock.json` sets your currently installed version of each package in stone, and `npm` will use those exact versions when running `npm install`.

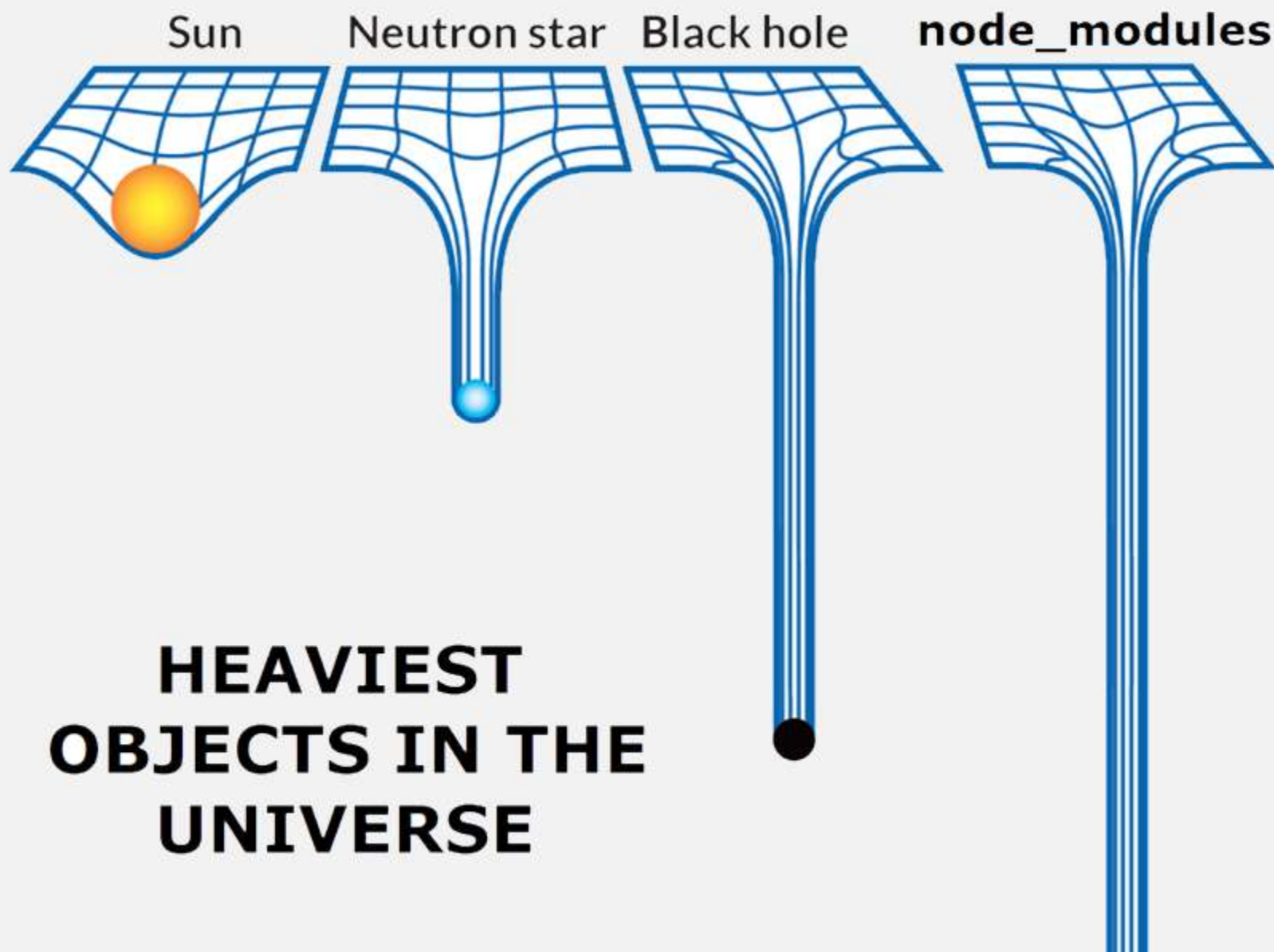


NPM DELIVERY



IMPLEMENTATION





Dilemma: 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]

<https://www.npmjs.com/browse/depended>



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 99% 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/>

```
npm ERR! npm v2.14.7
npm ERR! code E404
npm ERR! 404 Registry returned 404 for GET on https://registry.npmjs.org/left-pad
npm ERR! 404
npm ERR! 404 'left-pad' is not in the npm registry.
npm ERR! 404 You should bug the author to publish it (or use the name yourself!)
npm ERR! 404 It was specified as a dependency of 'line-numbers'
npm ERR! 404
npm ERR! 404 Note that you can also install from a
npm ERR! 404 tarball, folder, http url, or git url.
npm ERR! Please include the following file with any support request:
npm ERR!    /home/travis/build/coldrye-es/pingo/npm-debug.log
make: *** [deps] Error 1
```

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

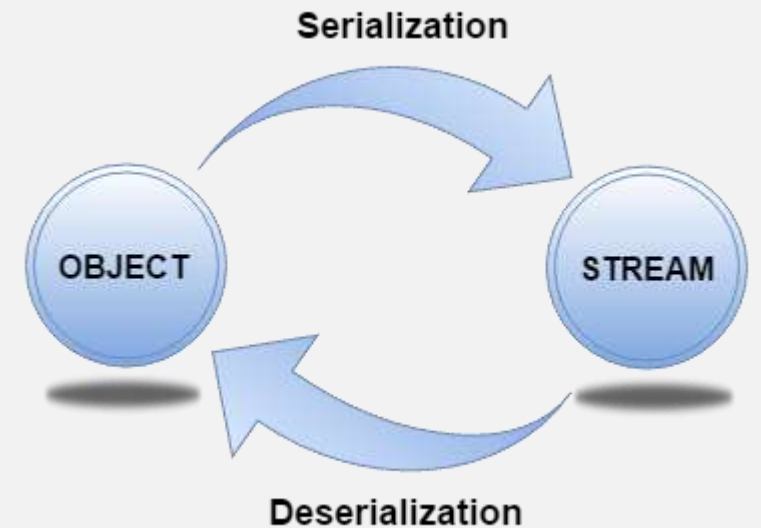
Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file.

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.

Object => serialization => JSON

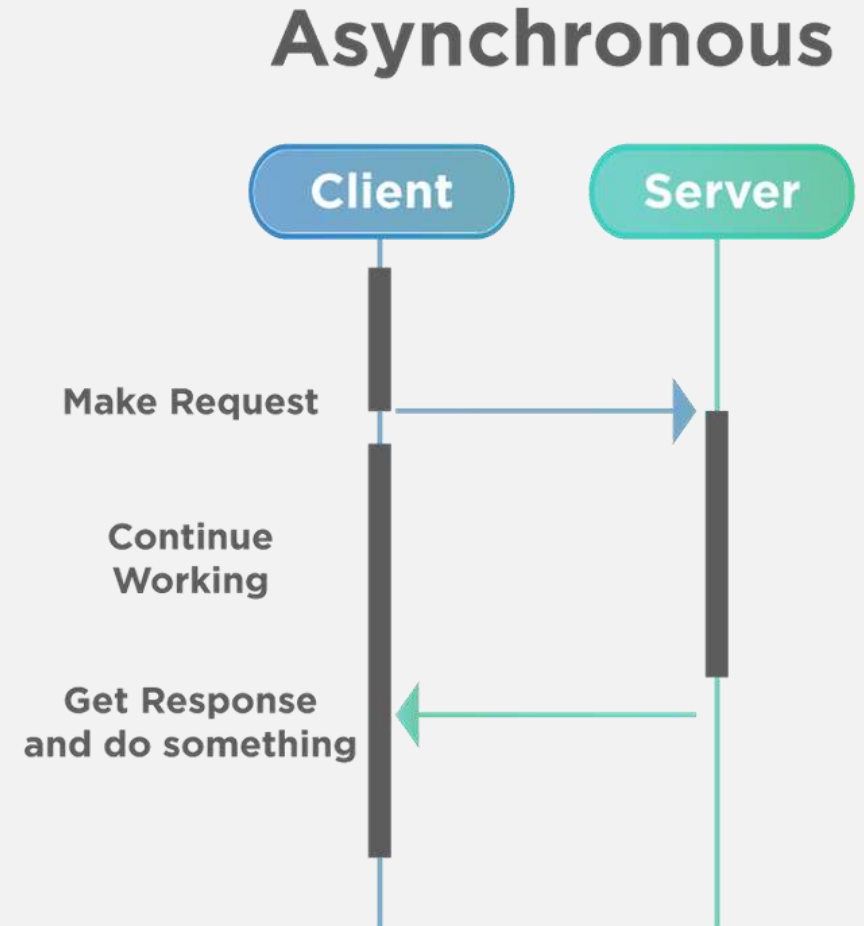
JSON => deserialization => Object



Async in action

For example, a function to read a file may start reading file and return the control to the execution environment immediately so that the next instruction can be executed.

Once file I/O is complete, it will call another function.



Callback

A Callback is simply a function passed as an argument to another function which will then use it (call it back)

Callback function allows other code to run in the meantime.

Node.js makes heavy use of callbacks - all the APIs of Node.js are written in such a way that they support callbacks. It allows Node.js to process a large number of requests without waiting for any function to return the result which makes Node.js highly scalable.

For example: In Node.js, when a function start reading file, it returns the control to execution environment immediately so that the next instruction can be executed. Once file I/O gets completed, callback function will get called to avoid blocking or wait for File I/O.



Callback example

```
const fs = require("fs");

var myCallbackFunction = function (err, data) {
  console.log(data.toString());
}

fs.readFile("input.txt", myCallbackFunction);

console.log("Program Ended");
```



Callback example #2

```
const fs = require("fs");

fs.readFile("input.txt", function (err, data) {
  console.log(data.toString());
});

console.log("Program Ended");
```



Error-First callback pattern

In Node.js, it is considered standard practice to handle errors in asynchronous functions by returning them as the first argument to the current function's callback.

If there is an error, the first parameter is passed an Error object with all the details. Otherwise, the first parameter is null.

```
var callback = function (error, retval) {  
    if (error) {    // something went wrong  
        console.log(error); // log error  
        return; // and leave  
    }  
  
    console.log(retval); // ok, we can process returned value  
}
```


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

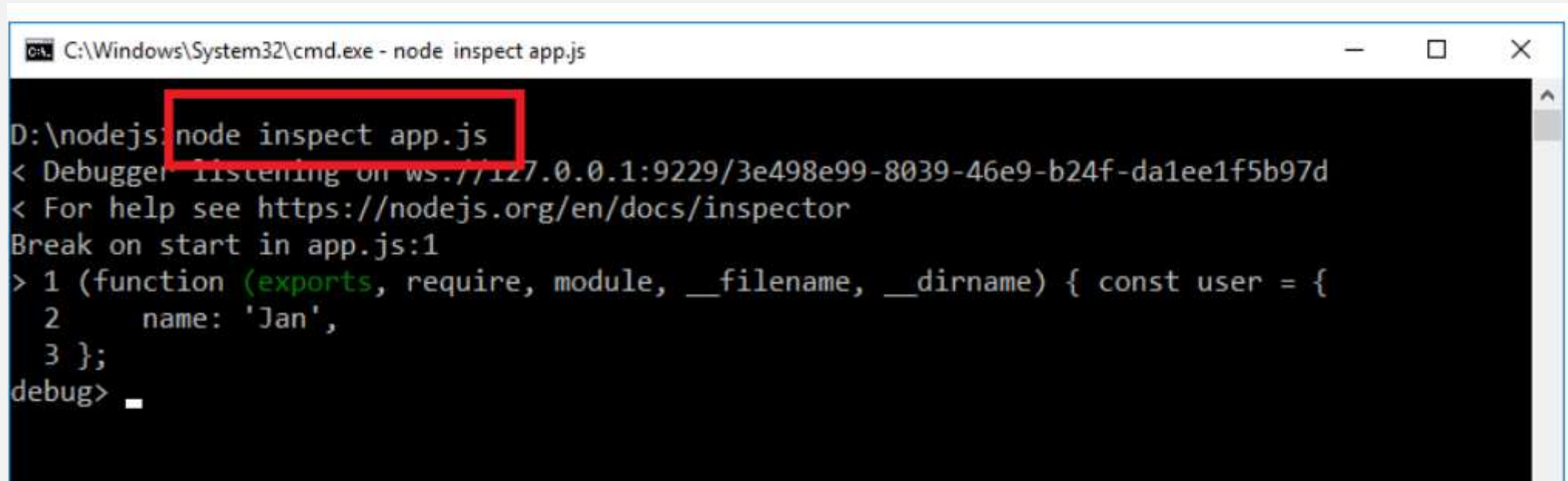
The CLI debugger will then use Node.js to start the script in a separate process and attaches to it.

```
> node inspect app.js
```

```
> node inspect --port=xxxx app.js
```



Node inspect



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\System32\cmd.exe - node inspect app.js". The command prompt shows the following text:

```
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> _
```

The command `node inspect app.js` is highlighted with a red rectangle.

```
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)

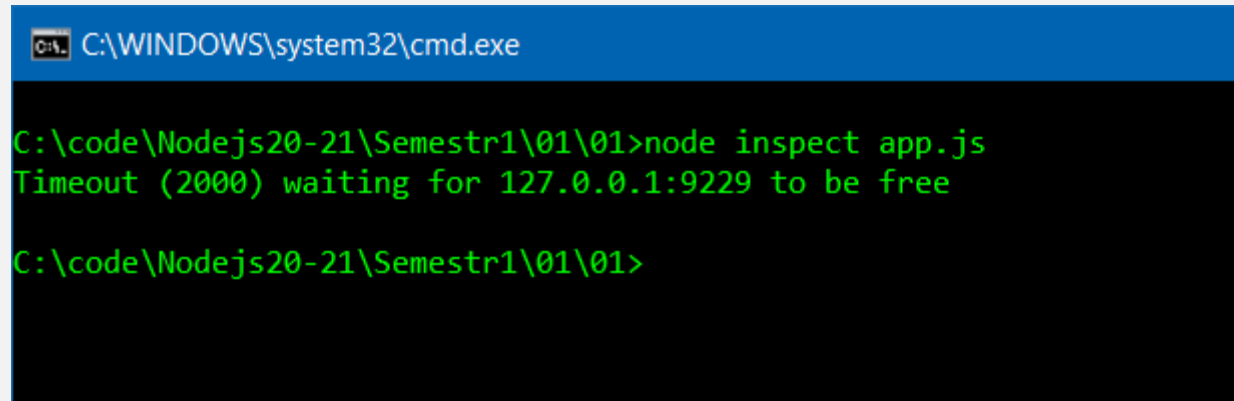
watch(expr)           Start watching the given expression
unwatch(expr)         Stop watching an expression
watchers              Print all watched expressions and their current values

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



```
C:\WINDOWS\system32\cmd.exe

C:\code\Nodejs20-21\Semestr1\01\01>node inspect app.js
Timeout (2000) waiting for 127.0.0.1:9229 to be free

C:\code\Nodejs20-21\Semestr1\01\01>
```

<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
```

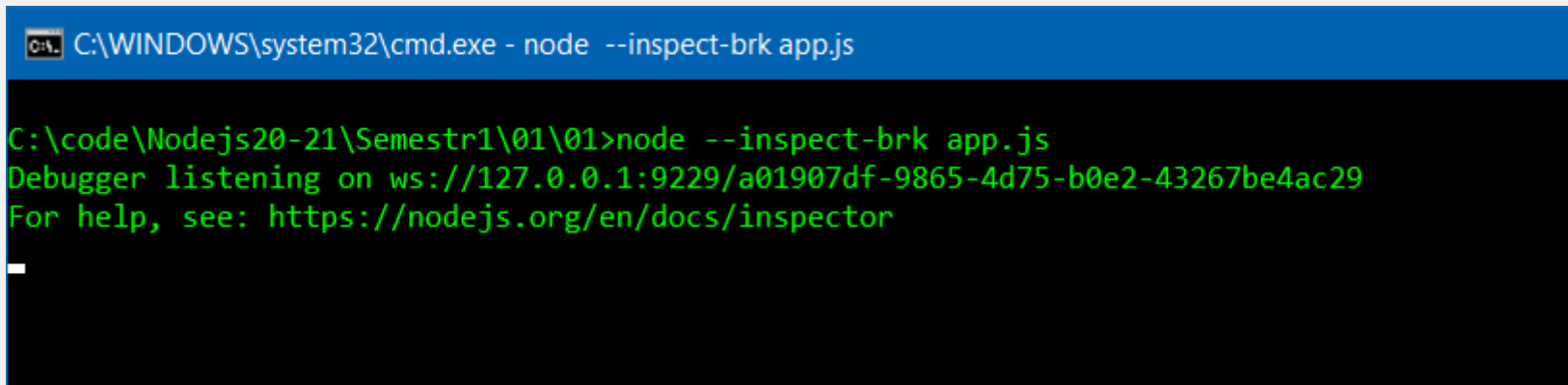
```
> node --inspect=[host:port] app.js
```

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

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



Node --inspect commandline

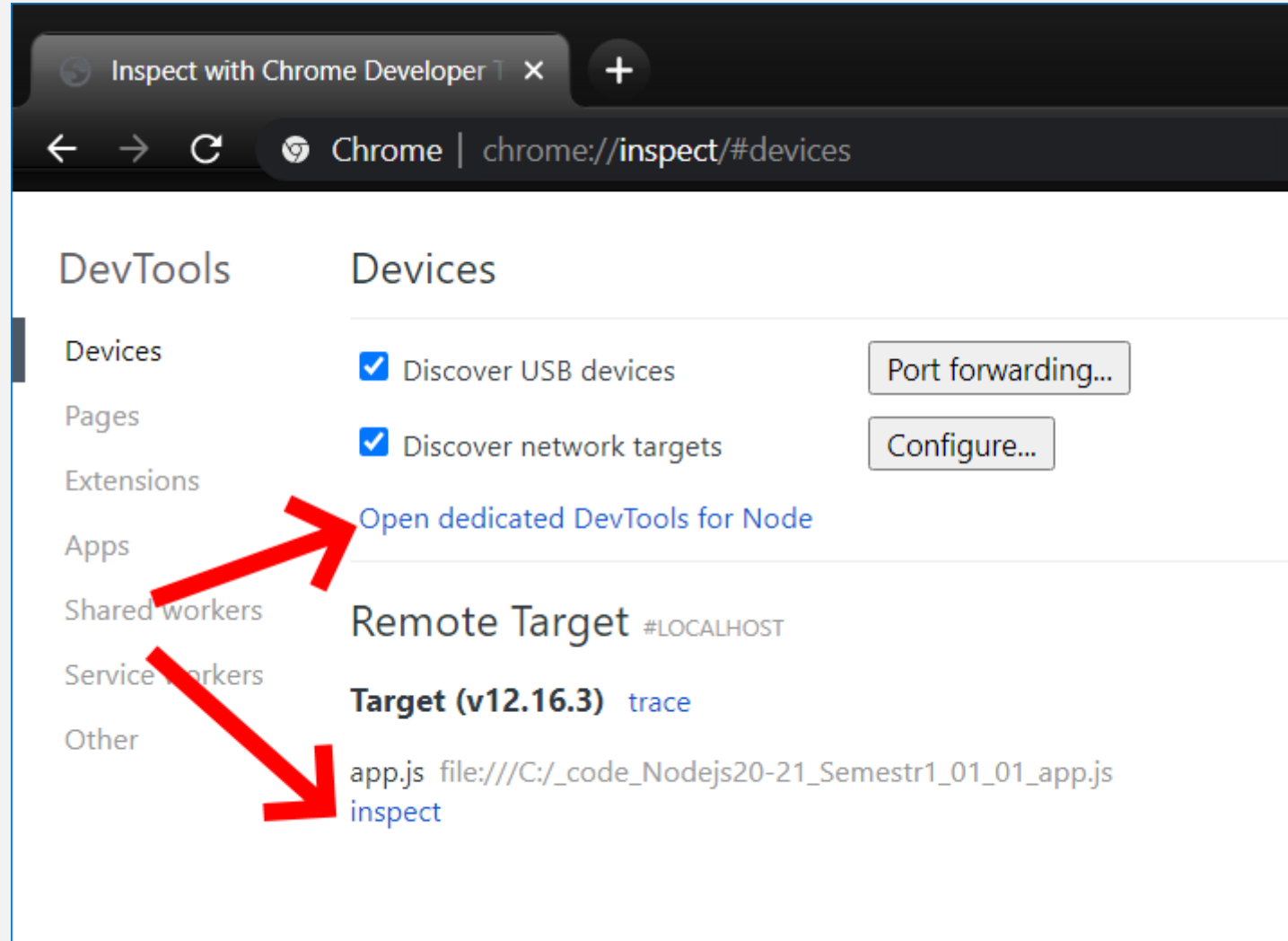


```
C:\WINDOWS\system32\cmd.exe - node --inspect-brk app.js

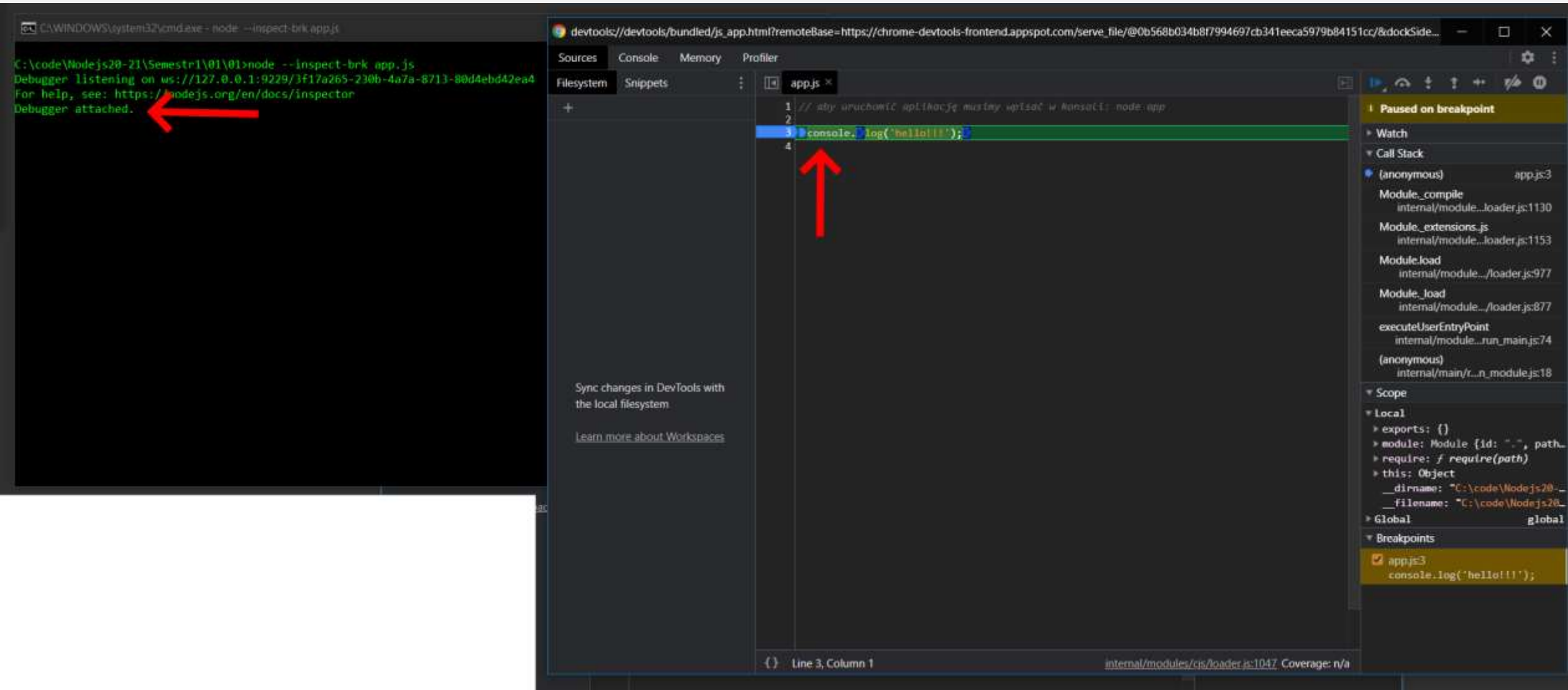
C:\code\Nodejs20-21\Semestr1\01\01>node --inspect-brk app.js
Debugger listening on ws://127.0.0.1:9229/a01907df-9865-4d75-b0e2-43267be4ac29
For help, see: https://nodejs.org/en/docs/inspector
_
```

Debugging in Chrome DevTools

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



Node --inspect debugging



Security warning

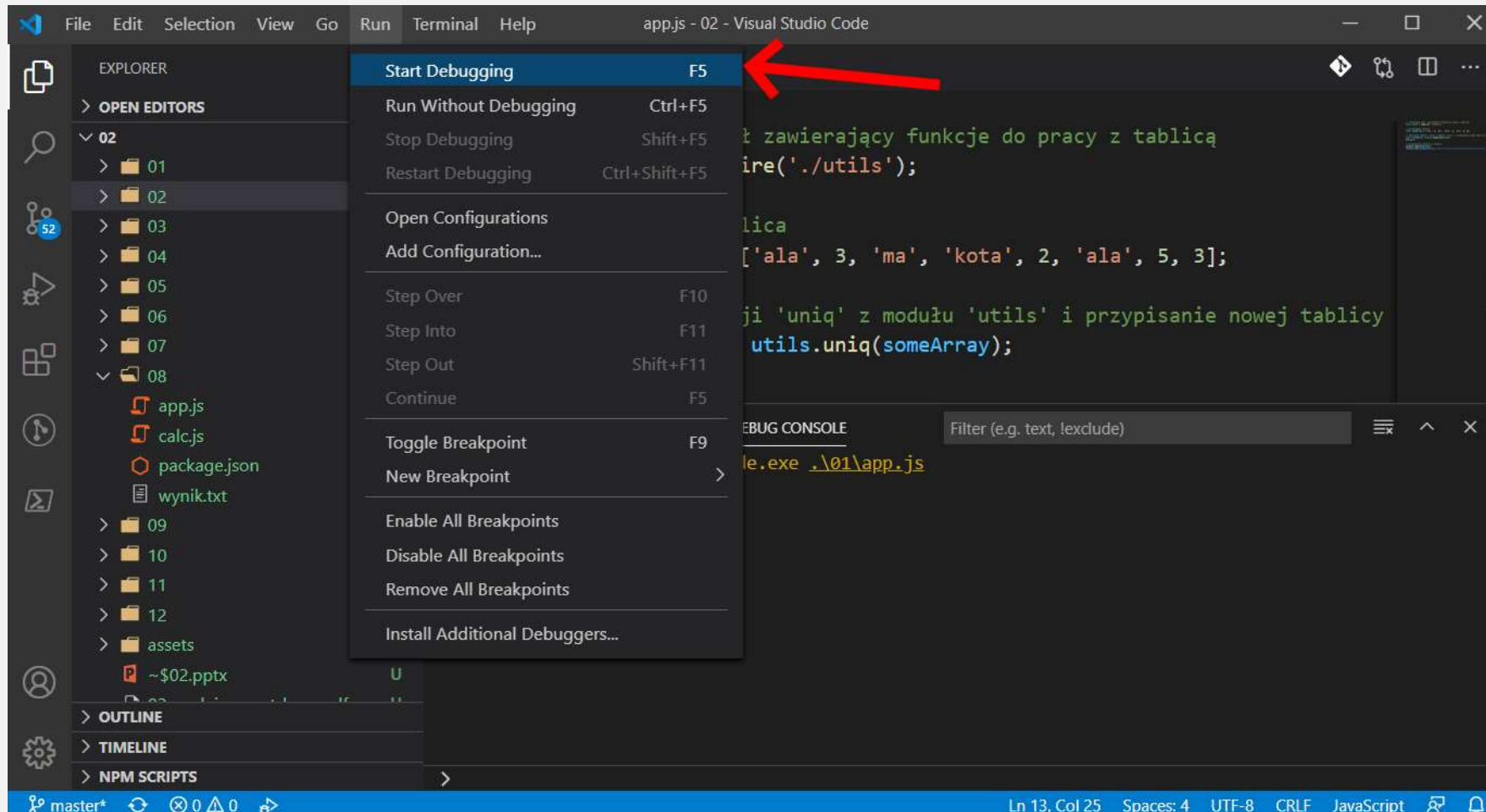
Since the debugger has full access to the Node.js execution environment, a malicious actor able to connect to this port may be able to execute arbitrary code on behalf of the Node.js process.

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

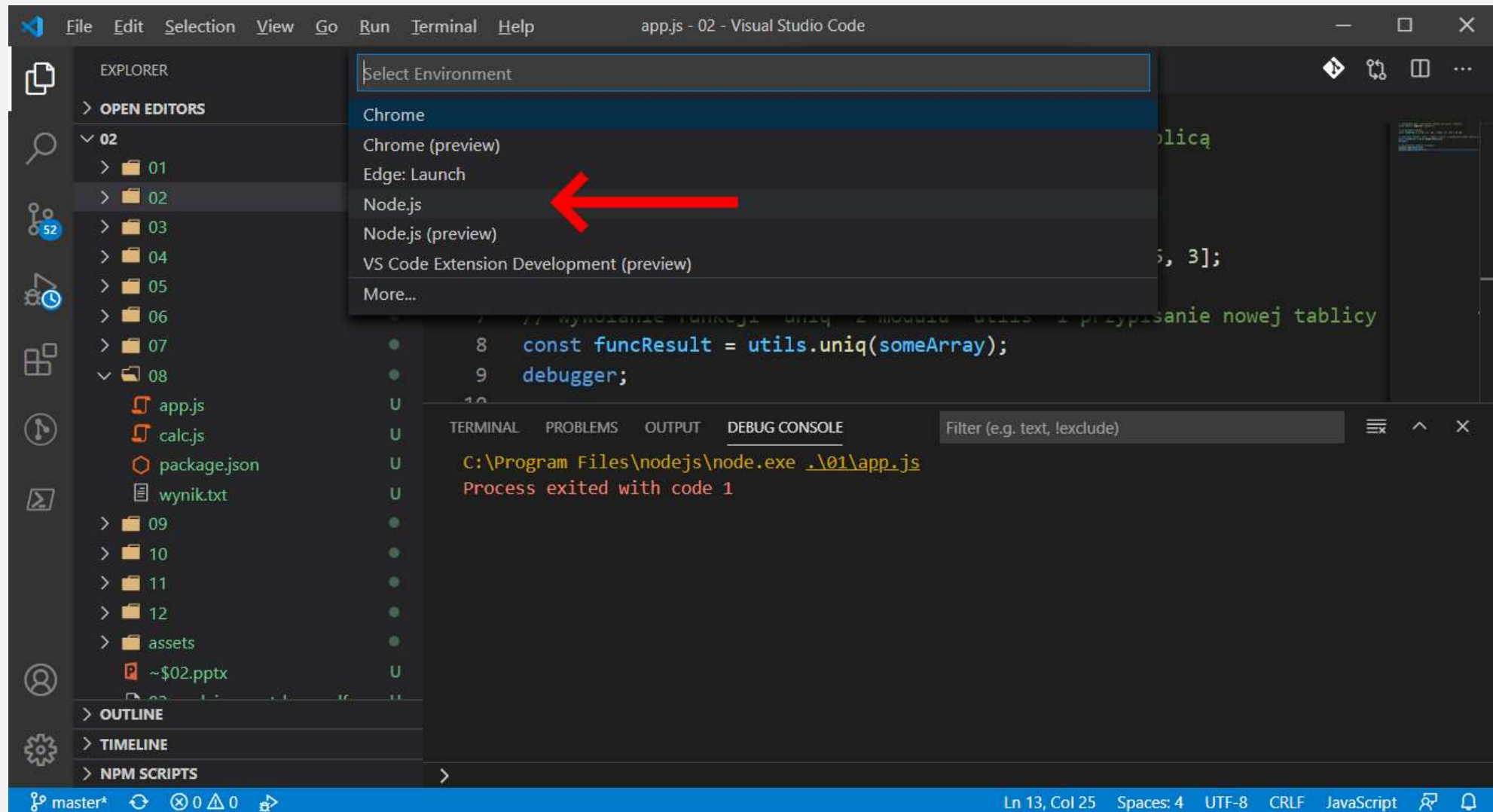


Debugging in IDE

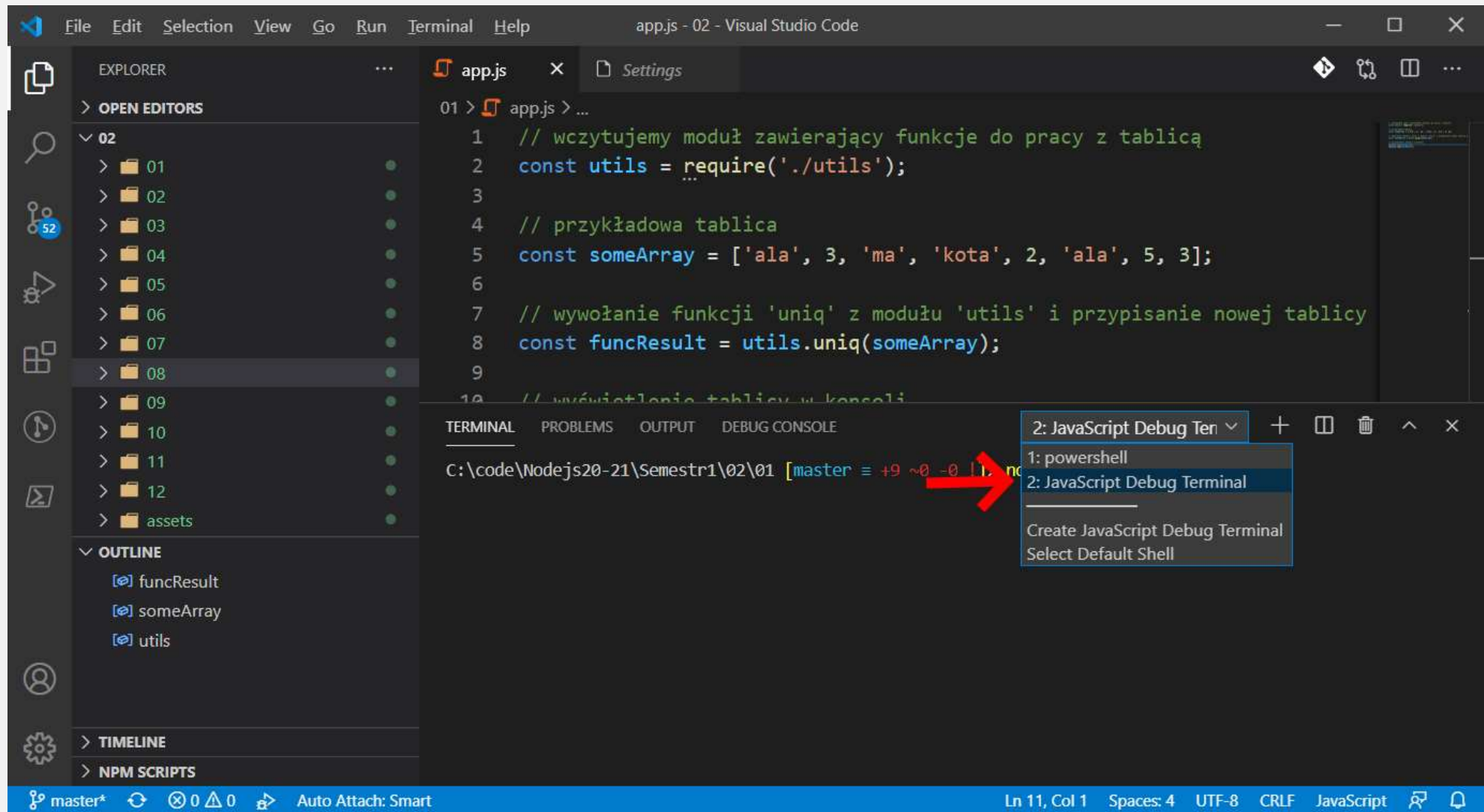
You can also debug your Node application in IDE – VS Code, Webstorm etc.



Debugging in IDE



Debugging in IDE



Debugging in IDE

