Web Systems I

Degree in Information Systems Engineering

Álvaro Sánchez Picot
alvaro.sanchezpicot@ceu.es

INTRODUCTION

Introduction

- Open-source
- Cross-platform
- JavaScript runtime environment
- V8 JavaScript engine (the core of Google Chrome)
- Initial version: 2009
- 2014 fork io.js (fixed in 2015)
- Leftpad incident in 2016
- More than 2,500,000 packages
- Documentation

Introduction - Business use

Netflix

- Switched its client backend in 2013 from Java to node.js
- More information

Paypal

- Changed part of its backend in 2013 from Java to node.js
- More information

Linkedin

- Switched the mobile backend in 2011 from Ruby on Rails to node.js
- Further information

And many other companies

Introduction

Same principles as JS:

- A single process
- Synchronous
- Non-blocking paradigms
  - Blocking is the exception
- Event management

Introduction

Differences with JavaScript and programming for the browser:

- No work with the DOM
- Browser variables such as *document* or *window* are missing.
- You have no file access restrictions
- No browser version problems (incompatibility with older versions of JS)

INSTALLATION

Necessary tools

- Git Bash for Windows
  - Linux and macOS now have a console
- IDE
  - Visual Studio Code
  - Webstorm
- Lightweight editors
  - Atom
  - Sublime Text
  - Notepad++
- Postman

Installation

- Go to: https://nodejs.org/
- Download and install the latest LTS version
- API

Installation

What do the numbers in version X.Y.Z mean?

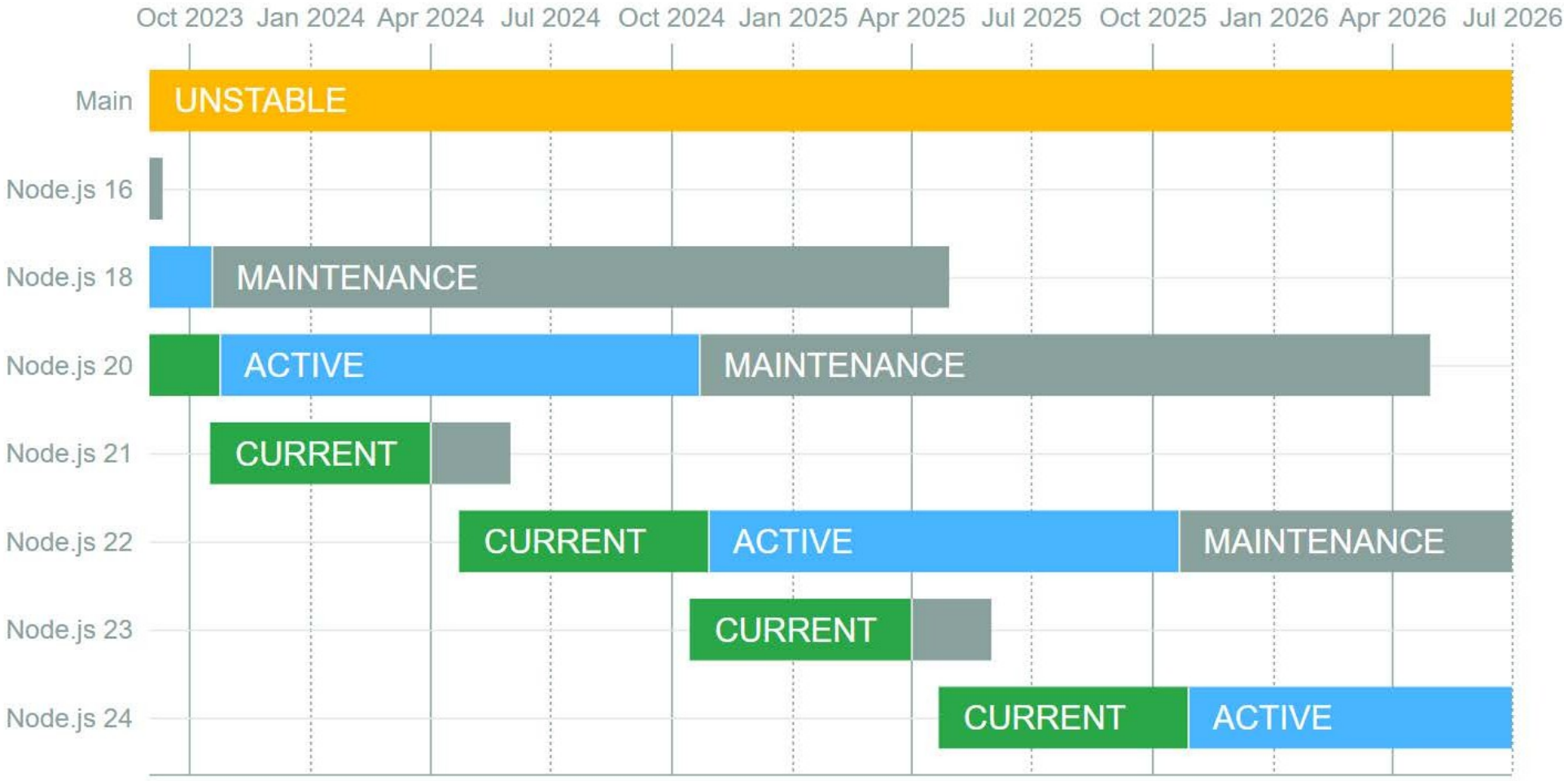
- Version numbering standard for software
- Semantic Versioning (SemVer): https://semver.org/
- MAJOR.MINOR.PATCH
  - MAJOR: Incompatible changes
  - MINOR: Added backwards compatible functionality
  - PATCH: Backwards compatible bug fixes

Installation

What is LTS?

- Long Term Support
- These are versions that are supported for much longer
- This applies to many software programs (e.g. Java, Ubuntu...).
- https://nodejs.org/en/download/releases

Installation



Installation - nvm

- Node Version Manager
- Working with different versions of Node.js
- For Windows download it here:
  - https://github.com/coreybutler/nvm-windows/releases
  - nvm-setup.zip
- For macOS and Linux, follow the instructions below:
  - https://github.com/nvm-sh/nvm#installing-and-updating

Installation - nvm

nvm list: List installed Node.js versions

nvm use x.y.z: Enables the x.y.z version of Node.js.

nvm install x.y.z: Installs the x.y.z version of Node.js.

Node.js versions: https://nodejs.org/download/release

nvm --help: Displays help

Installation - npm

- Node Package Manager
- Node.js Package Manager
- Installed together with Node.js
- Also used for JS in the frontend
- Manages project dependencies:
  - Download
  - Update
  - Versions
- Allows the definition and execution of tasks
- Web

Installation - npm

- Create the configuration file package.json

```
npm init
```

- Installation of dependencies
  - Requires package.json configuration file

```
npm install
npm i
```

- Installation of a package

```
npm install package_name
```

- Installing the x.y.z version of a package

```
npm install <package_name>@<version>
```

Installation - npm

- List installed packages along with version

```
npm list
```

```
npm ls
```

- Update all dependencies

```
npm update
```

- Upgrade a specific package

```
npm update package_name
```

- Delete a package

```
npm uninstall package_name
```

Installation - npm

- View package information

npm view package_name

- View available versions of a package

npm view package_name versions

- Command information

REPL

- Read Evaluate Print Loop
- Node.js console environment
- To run it: `node`
- The tab key can be used for auto-completion.
- Special commands:
  - `.help`: Displays help
  - `.exit`: Exits REPL (equivalent to pressing Ctrl+C twice or Ctrl+D).

npx

- Allows code to be executed
- It is not necessary to have the package installed

```
npx cowsay "Hello"

 _______
< Hello >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Example 0

Our first server, we store it in index.js:

```
const  http =  require('http');

const  port =  3000;

const  server  = http.createServer((req, res) => {
   res.statusCode = 200;
   res.setHeader('Content-Type', 'text/html');
   res.end('<h1>Hello, World!</h1>');
});

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

Example 0

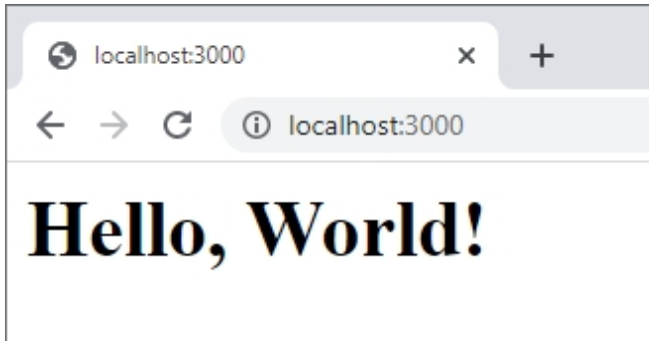
- We start the server with the following instruction:

```
node index.js
```

- We should see that the server starts up correctly:

```
$ node index.js
Server running at port 3000
```

- From the browser we connect to localhost:3000
- If everything went well we should see:



Example 0

```
const http = require('http');
```

- require is used to import modules (libraries)
  - Equivalent to import in Java
- http is a module that is included in the core of Node.js.
  - No need to install anything extra
  - http Documentation
  - List of core modules:
    - url: for working with and parsing URLs
    - path: to work and parse paths
    - fs: to work with I/O in files
    - util: various functionalities

Example 0

```
const server = http.createServer((req, res) => {/*...*/});
http.createServer([options][, requestListener]);
```

- createServer is a method to create an HTTP server.
  - Documentation
- requestListener is a function
  - Executed with every request to the server (request)
  - req is an http.IncomingMessage object with information from the request.
  - res is an http.ServerResponse object with information about the response.

Example 0

```
server.listen([port[, host[, host[, backlog]]]][, callback]);
```

- listen is a method that starts the server by listening on a port
  - Documentation (identical to that of the http module). Several possibilities to invoke it
- port is the port where we want to listen
  - If omitted, a random one shall be chosen without using
- host
  - By default the *unespecified IPv6 address* (equivalent to 0.0.0.0.0 in IPv4).
- backlog is the maximum number of outstanding connections
  - Default is 511
- callback is executed when the server is finished booting up

Example 0

We are going to add a log to see the connections on our server, start it and reload the page:

```
const http = require('http');
const server = http.createServer((req, res) => {
    console.log('New connection');
    res.statusCode = 200;
    //...
```

Why are there two connections when loading the page in the browser?

```
$ node index.js
Server running at port 3000
New connection
New connection
```

Environment variables

- Accessible with the process module (available without require)

```
process.env.variable_name
```

- We can define them when running the programme

```
USER_ID=239482 USER_KEY=foobar node app.js
```

- It is often used for example for the port on which the server starts up.

```
const PORT = process.env.PORT || 3000;
```

- NODE_ENV
  - To define whether you are in a production or a development environment
  - Many modules use it
  - Values: production, development

Environment variables

- They can be loaded from an .env file with the dotenv module.
  - npm install dotenv

```
# .env file
USER_ID="239482"
USER_KEY="foobar"
NODE_ENV="development"
NODE_ENV="development"
```

```
require('dotenv').config();
process.env.USER_ID // "239482"
process.env.USER_KEY // "foobar"
```

```
process.env.NODE_ENV // "development"
```

Arguments

- They can be added when running the programme

```
node index.js joe smith
node index.js name=joe surname=smith
```

- Accessible with process.argv
  - Returns an array following the UNIX convention
  - Position 0: full path of the node command
  - Position 1: full path of the file
  - Positions 2 onwards: the arguments

Arguments

- If we want to use key=value we have to parse it
- We can use the minimist module to deal with the arguments
  - Arguments are passed with --name=value
  - You can also use options like in unix with -option value
    - option is a letter
  - npm install minimist

```
node index.js --name=joe --surname=smith
node index.js -x 1 -y 2 -abc
```

process

- Global object with general information on the execution of node
- Further information

```
process.argv // An array of command-line arguments.
process.arch // The CPU architecture: "x64", for example.
process.cwd() // Returns the current working directory.
process.cpuUsage() // Reports CPU usage.
process.env // An object of environment variables.
process.execPath // The absolute filesystem path to the node executable.
process.exit() // Terminates the program.
process.exitCode // An integer code to be reported when the program exits.
process.kill() // Send a signal to another process.
```

process

process.memoryUsage() // Return an object with memory usage details.

process.nextTick() // Invoke a function soon.

process.pid // The process id of the current process.

process.platform // The OS: "linux", "darwin", or "win32", for example.

process.resourceUsage() // Return an object with resource usage details.

process.uptime() // Return Node's uptime in seconds.

process.version // Node's version string.

process.versions // Version strings for the libraries Node depends on.

os

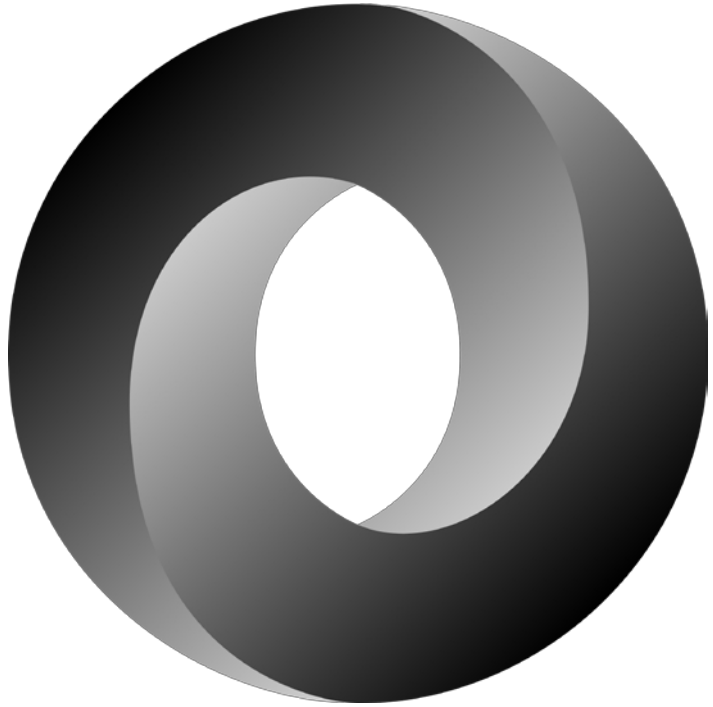
- Module with access to low-level information about the OS
- Needs to be charged
- Further information

```
const os = require("os");
os.cpus() //  Data about  system CPU cores, including usage times.
os.freemem()  // Returns  the amount of free RAM in bytes.
os.homedir()  // Returns  the current user's home directory.
os.hostname() // Returns the hostname of the computer. os.loadavg()
// Returns the 1, 5, and 15-minute load averages.
os.networkInterfaces() // Returns details about available network. connections.
os.release() // Returns the version number of the OS.
os.totalmem() // Returns the total amount of RAM in bytes.
os.uptime() // Returns the system uptime in seconds.
```

Exercise 1

- Create a node.js server
- Show system information and node.js version on startup via console.
- Periodically display the following information by console:
  - CPU usage
  - Memory usage
  - Time the system has been active
  - Time node.js has been running
- That the information displayed periodically is configurable in a file, including how many seconds it is displayed.

JSON



JSON

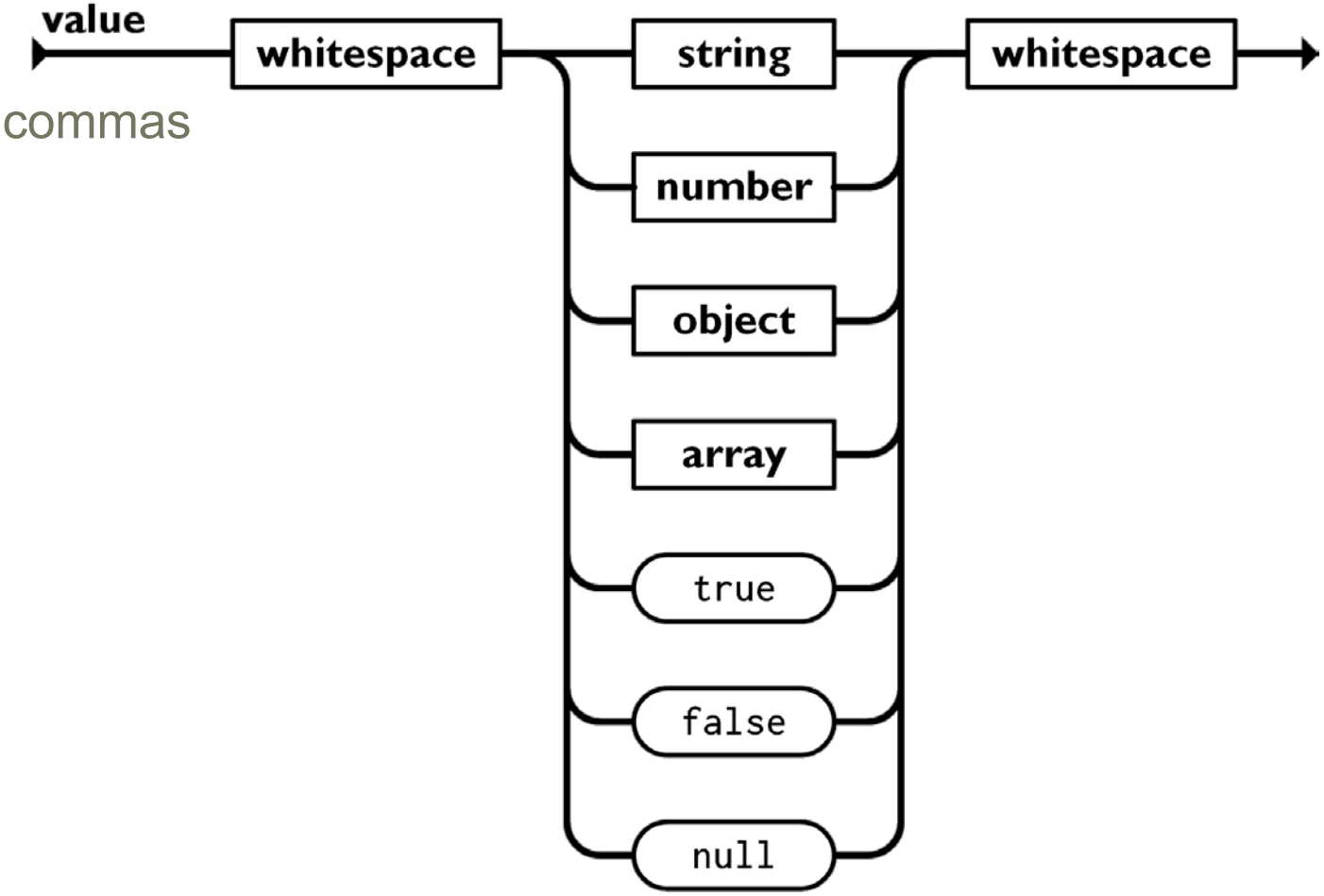
- JavaScript Object Notation
- Lightweight format for data exchange
- Based on the ECMA-262 standard of 1999.
  - Latest version ECMA-404 of 2017
- Supports objects, arrays and values
- No comments allowed
- Further information
- Validator
- Style guide

JSON - Example

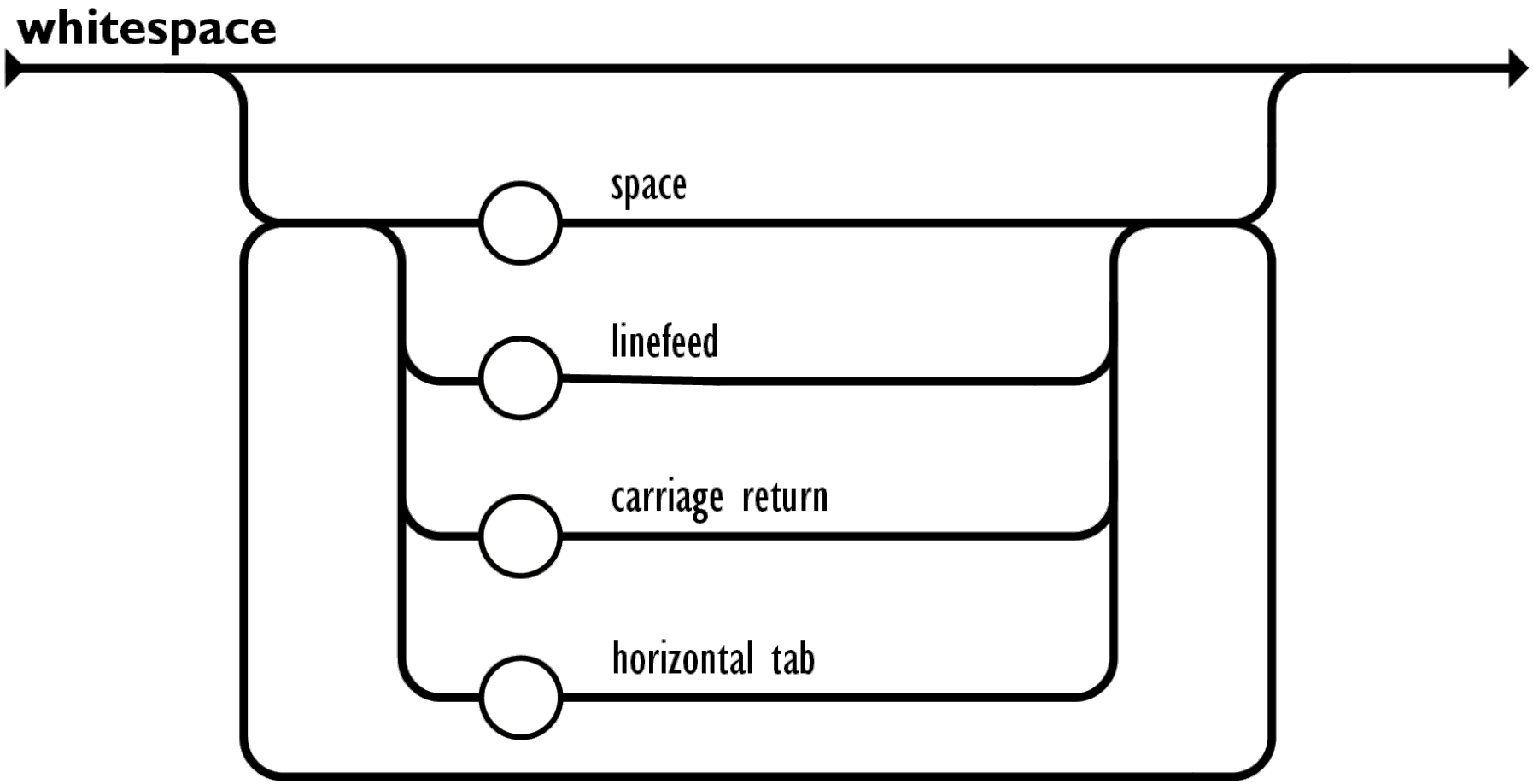
```
{
   "firstName": "John",
   "lastName": "Smith",

   "isAlive": true,
   "age: 27,
   "address": {
      "streetAddress:   "21  2nd  Street",

      "city": "New York",
      "state": "NY",
      "postalCode": "10021-3100".
   },
   "phoneNumbers": [
      {
         "type": "home",
         "number": "212 555-
         1234".
      },
      {
         "type": "office",
         "number": "646 555-
         4567".
      }
   ],
   "children": [],
   "spouse": null
}
```

JSON - Syntax

- Value:
  - String: text in double inverted commas
  - Number
  - Object
  - Array
  - true
  - false
  - null



JSON - Syntax


whitespace
space
linefeed
carriage return
horizontal tab

JSON - Syntax

- There has to be a single root element which has to be a value:
  - Valid JSON: {}
  - Valid JSON: ""
  - Valid JSON: 4
  - Valid JSON: true
  - Valid JSON:

```
[
   { "name" : "John"},
   { "name" : "Ana"},
   { "name" : "Sofia"},
   { "name" : "Andrés"}
]
```

JSON - Syntax

Object:

- Unordered set of key/value pairs:

  `"key": value`

- Surrounded by braces: { and }
- The key:
  - It is a String and must have double inverted commas
  - May not be repeated within the same object
  - Recommended to use lower camel case nomenclature.
- Each key/value pair is separated by commas
  - Beware of trailing commas
- Can be empty: { }

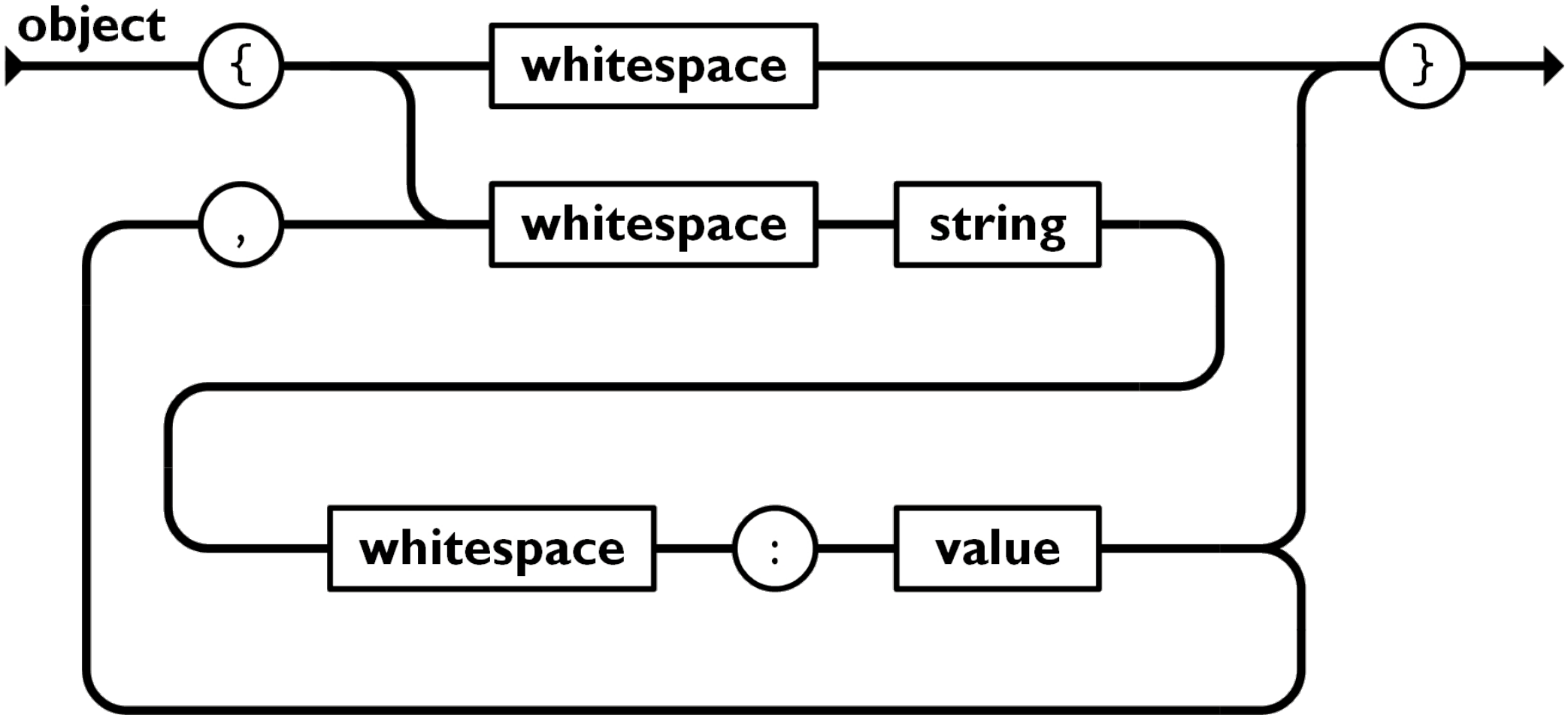
JSON - Syntax

Object:

```
{
   "fullName": "Juan Pérez Rodríguez", "age": 27
}
```

Beware of trailing comma

JSON - Syntax



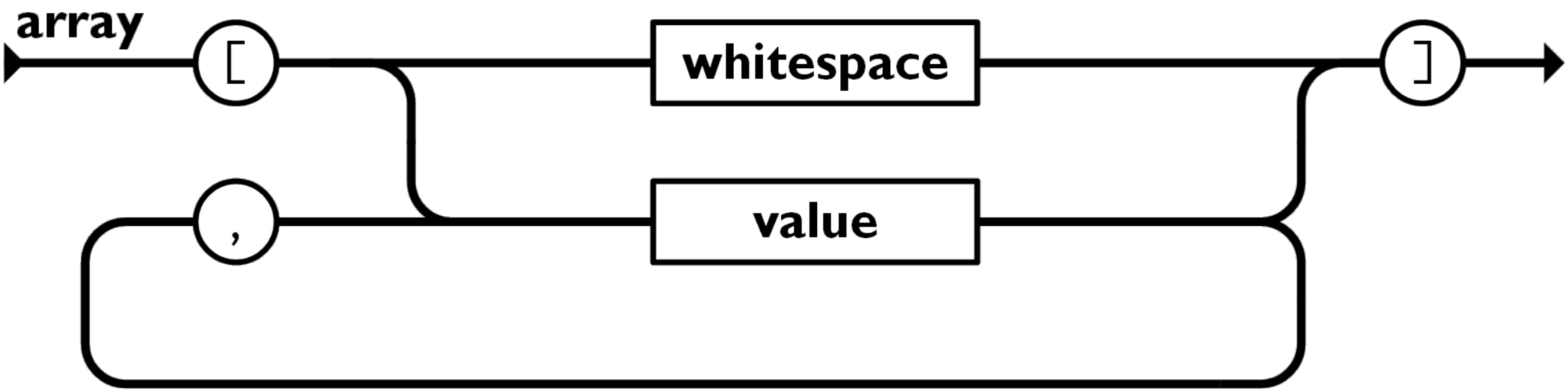
JSON - Syntax

Array:

- Orderly collection of securities
- Surrounded by square brackets: [and]

```
[
   { "mobile": 612345678},
   {"fixed": 912345678}
]
```

JSON - Syntax

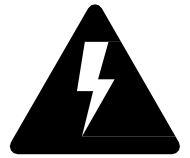


JSON - Syntax

- Elements can be nested:

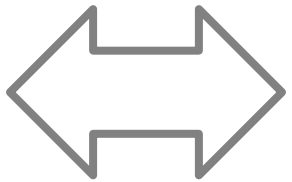
```
{
  "name": "John",
  "address": {
    "street": "Avenida Ciudad de Barcelona
    23", "city": "Madrid".
  },
  "telephones": [
    { "mobile": 612345678},
    {"fixed": 912345678}
  ],
  "age: 27
}
```

JSON



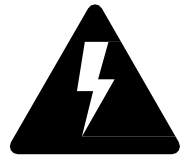
- The order of the content of the objects is not guaranteed.

```
{
   "name": "John",
   "age": 27
}
```



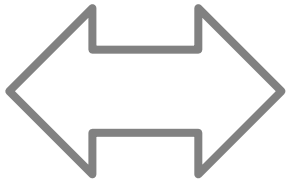
```
{
   "age: 27,
   "name": "Juan".
}
```

JSON



- The elements of an array are in order

```
[
   { "name" : "John"},
   { "name" : "Ana"}
]
```



```
{
   "age: 27,
   "name": "Juan".
}
```

JSON - Exercise

- Is the following JSON well formed?

```
{
   "name": "Juan",
   "telephones": [
      { "mobile": 612345678},
      { "fixed": [912345678]}, false
   ]
   age: 27, oldestAge: "true", "address": {
      "street": "Avenida Ciudad de Barcelona 23",
      "city": Madrid, null
   },
}
```

JSON with JS

- JSON package
  - Not specific to Node.js
- Convert a JSON object to text

```
let text = JSON.stringify(obj);
```

- Convert a text in JSON format to an object

```
let obj = JSON.parse(text);
```

JSON with JS

- Traversing a JSON object

```
for(let key in jsonObj) {
    console.log("key: "+ key +", value: "+ jsonObj[key]);
}
```

JSON with JS - Example

```
let person = {"name": "John", "age": 23};
console.log("Person:Person", person);
let text = JSON.stringify(person);
console.log("\nText:\n", text);
let obj = JSON.parse(text);
console.log("\nObj:\n", obj);
console.log("\nKeys:")
for(let key in obj) {
      console.log("key: "+ key +", value: "+ obj[key]);
}
console.log("\nStringify:\n", JSON.stringify(person, null, 2));
```

package.json

- json format
- Used by npm
- Project metadata
  - Units
  - Description
  - Version
  - Licence
  - Configuration
  - ...
- It can be created manually or using `npm init`
- Documentation

package.json

Example of package.json:

```
{
   "name": "test_node", "version":
   "1.0.0",
   "description": "Node.js testing exercise", "main":
   "index.js",
   "scripts": {"test": "echo "Error: no test specified" && exit 1"},
   "keywords": ["test", "test"],
   "author": "Álvaro",
   "license": "ISC",
   "dependencies": {
      "minimist": "^1.2.5": "^1.2.5".
   }
}
```

package.json

"name: "test-project".

- Package name
- Up to 214 characters
- Lower case only, '-' or '_'.

"version": "1.0.0".

- Package version
- Semver format

"description": "A package to work with strings".

- Package description

package.json

main": "src/main.js": "src/main.js".

- Point of entry of the package
- Where it will be searched for export as a module
- If it does not exist the default value is index.js
- The file is executed with node .

package.json

```
"scripts: {
 "start": "npm run  dev",
 "unit": "jest --config test/unit/jest.conf.js --coverage",
 "test": "npm run unit", "unit": "jest --config
 test/unit/jest.conf.js --coverage", "test": "npm run unit".
}
```

- Define executable scripts
- Run with: npm run XXXX
- Some specials can be run without the run:
  - npm start

- npm test

package.json

```
"keywords: [
    "email",
    machine learning",
    "ai", "ai",
    "machine learning
]
```

- Array of keywords
- Useful for finding the package in npm

package.json

```
"author": "Joe <joe@whatever.com> (https://whatever.com)"
"author": {
    "name": "Joe",
    "email": "joe@whatever.com",
    "url": "https://whatever.com"
  }
```

- There can only be one
- Optional email and url

package.json

```
"contributors": ["Joe <joe@whatever.com>
(https://whatever.com)"]
```

```
"contributors": [
    {
      "name": "Joe",
      "email": "joe@whatever.com",
      "url": "https://whatever.com"
    }
  ]
```

- There may be several

package.json

```
"dependencies": {
   "vue": "^2.5.2": {
   "vue": "^2.5.2".
}
```

- Dependencies of the package
- They follow a special format to indicate how it is updated.
- They are added automatically when making:

```
npm install package_name
```

package.json

```
"devDependencies": {
   "autoprefixer":"^7.1.2",
   "babel-core": "^6.22.1".
}
```

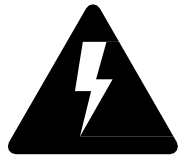
- Development-only units
- Not required in production
- They are added automatically when making:

```
npm install --save-dev package_name
```

package.json

- And many more options:
  - "bugs": "https://github.com/whatever/package/issues"
  - "homepage": "https://whatever.com/package"
  - "license": "MIT
  - repository": "github:whatever/testing": "github:whatever/testing": "github:whatever/testing".
  - "private": true
  - and more...

package.json



- There is a problem with package.json:
  - The project is not 100% replicable

```
"dependencies": {
   "vue": "^2.5.2": {
   "vue": "^2.5.2".
}
```

- One might use version 2.5.2, another 2.5.3 and another 2.6.0.
  - It may lead to inconsistencies in the project.
- Alternative: upload node_modules folder to repository
  - Not recommended because it can take up a lot of space (several hundred megabytes).

package-lock.json

- Specifies the exact version of each dependency
  - The package is 100% reproducible
- Automatically generated and updated
  - By doing npm install
  - When doing npm update
- Reminder, you can view dependencies with: npm list
- Documentation

package-lock.json

```
{
  "name": "test_node",
  "version": "1.0.0",
  "lockfileVersion": 1,
  "requires": true,
  "dependencies": {
    "minimist": {
      "version": "1.2.5",
      "resolved": "https://registry.npmjs.org/minimist/-/minimist-1.2.5.tgz",
      "integrity": "sha512-FM9nNUYrRBAELZQT3xeZQ7fmM..."
    }
  }
}
```

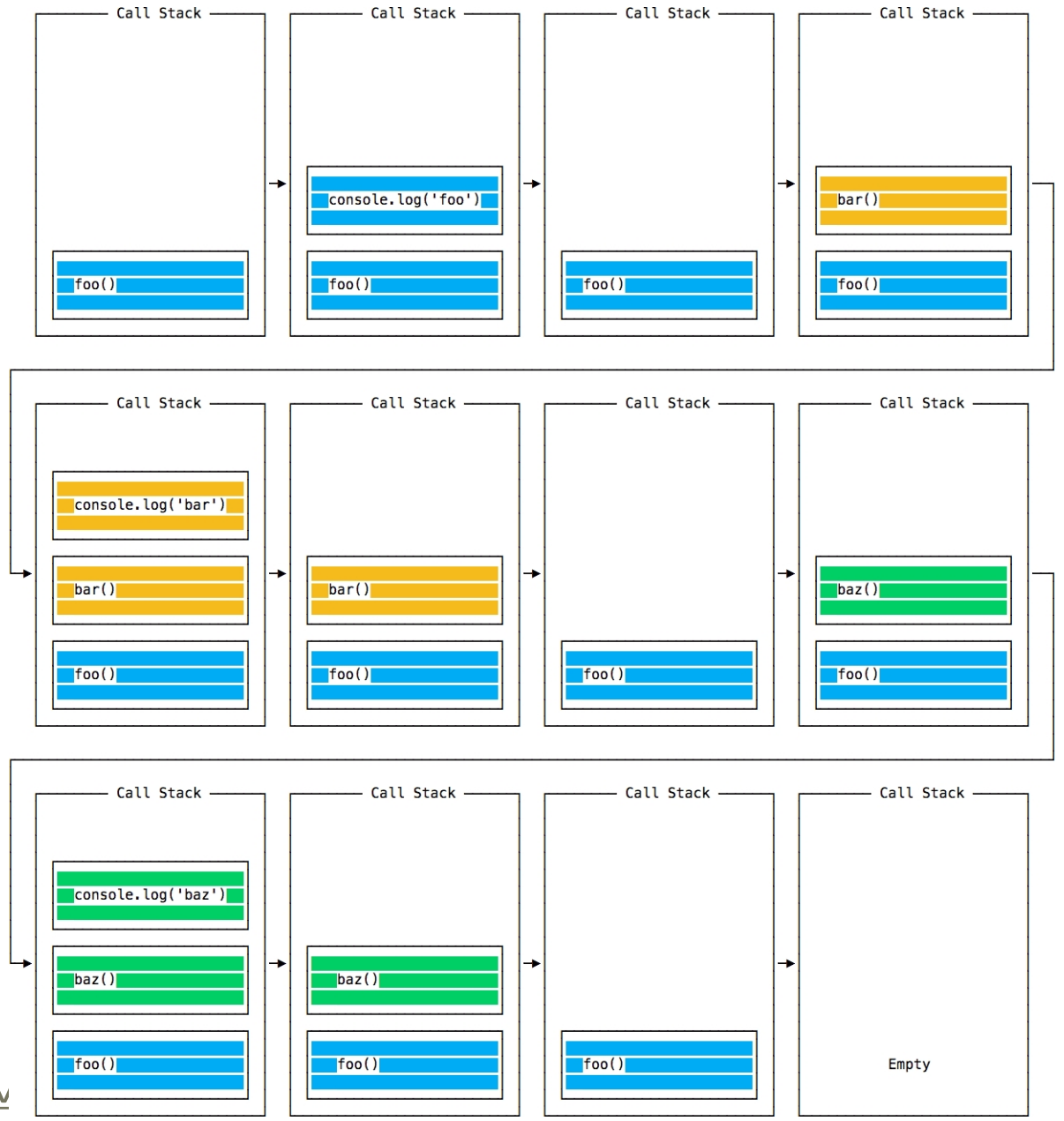
EVENT LOOP

Event Loop

- A single thread
- Repeats (loop)
- Each iteration is a tick
- Manages the execution of events
- A FIFO queue of callbacks
  - On each tick it runs until it is empty
- Managed internally by the libuv library
- More detailed explanation
  - There are several phases
  - Each has its own FIFO queue

Event Loop

```
const bar = () => console.log('bar');
const baz = () => console.log('baz');
const foo = () => {
  console.log('foo');
  bar();
  baz();
}
foo();
```



source: nodejs.dev

Don't block the event loop!

Event Loop

```
const  http =   require('http');
const  crypto  = require('crypto');
const   port =  3000;
const  server  = http.createServer((req, res) => {
   let  time =  Date.now();
   res.statusCode = 200;
   res.setHeader('Content-Type', 'text/html');
   res.end('<h1>Hello, World!</h1>');
   let array = [];
   for (let i  = 0; i  < 1000000; i++) {
      array.push(crypto.randomBytes(1000).toString('hex'));
   }
   console.log(Date.now()-time);
});
server.listen(port, () => {console.log(`Server running at port ${port}`);});
```

Event Loop

- We must not block the event loop
- Each tick must be short
- The work associated with each client has to be brief.
- Try to divide up the more intensive tasks
- Further information

FILES

path

- Module with utilities for working with files and routes
- It must be imported:

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

- Further information

path

path.dirname(*path*)

- Returns the name of the directory

path.sep

- OS separator: on Windows, / on UNIX

path.normalize(*path*)

- Removes duplicate separators and processes ... and .

path.join([...*paths*])

- Joins the paths using the system separator and normalises it.

path.resolve([...*paths*])

- Processes from right to left and generates an absolute path.

path

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

console.log(path.sep);
let p = "src/pkg/test.js"; console.log(path.basename(p))
// => "test.js" console.log(path.extname(p)) // => ".js"
console.log(path.dirname(p)) // => "src/pkg"
console.log(path.basename(path.dirname(p))) // => "pkg"
console.log(path.dirname(path.dirname(p))) // // => "src"

console.log(path.normalize("a/b/c/../d/")) // => "a/b/d/"
console.log(path.normalize("a/./b")) // => "a/b"
console.log(path.normalize("//a//b//") // // => "/a/b/"

console.log(path.join("src", "pkg", "t.js")) // => "src/pkg/t.js"

console.log(path.resolve()) // => process.cwd()
console.log(path.resolve("t.js")) // => path.join(process.cwd(), "t.js")
console.log(path.resolve("/tmp", "t.js")) // => "/tmp/t.js"
console.log(path.resolve("/a", "/b", "t.js")) // => "/b/t.js"
```

fs

- File system
- Accessing and interacting with the file system
- Node.js Core
- The methods are async by default
  - There is sync version by adding Sync to the end of the method
    - fs.access() ↔ fs.accessSync()
- Documentation

fs

fs.access(): check if the file exists and Node.js can access it with its permissions fs.appendFile(): append data to a file. If the file does not exist, it's created fs.close(): close a file descriptor

fs.copyFile(): copies a file

fs.mkdir(): create a new folder

fs.open(): set the file mode

fs.readdir(): read the contents of a directory

fs.readFile(): read the content of a file

fs.realpath(): resolve relative file path pointers (., ..) to the full path

fs.rename(): rename a file or folder

fs.rmdir(): remove a folder

fs.stat(): returns the status of the file identified by the filename passed

fs - open

```
fs.open(path[, flags[, mode]], callback);
fs.open('/Users/joe/test.txt', 'r', (err, fd) => {
  //fd is our file descriptor
});
```

flags:

- r: read mode (default)
- r+: read and write mode. The file will not be created if it does not exist
- w+: read and write mode (at the beginning). The file is created if it does not exist
- a: write to the end of the file. The file is created if it does not exist

fs - File reading

```
const fs = require('fs');
fs.readFile('test.txt', 'utf8', (err, data) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(data);
});
```

fs - File writing

```
const fs = require('fs');
const content = 'Some content of file'; fs.writeFile('test.txt',
content, err => {
   if (err) {
      console.error(err);
      return;
   }
   console.log('The file was written successfully');
});
```

fs - Promises

fs.promises

- Equivalent functions working with promises
- More information

fs.promises.open(path, flags[, mode])

fs.promises.readFile(path[, options])

fs.promises.writeFile(file, data[, options])

And many more...

fs - Promises

```
fs.promises
   .readFile("data.csv", "utf8")
   .then(processFileText)
   .catch(handleReadError);

// Alternatively, using async / await
async function processText(filename, encoding="utf8") {
   let text = await fs.promises.readFile("data.csv", "utf8");
   // Text processing
}
```

fs - stat

```
const fs = require("fs");
let stats = fs.statSync("dictionary.txt");
console.log(stats);
stats.isFile() // => true: this is an ordinary file
stats.isDirectory() // => false: it is not a directory
stats.size // file size in bytes
stats.atime // access time: Date when it was last read stats.mtime
// modification time: Date when it was last written stats.uid // the
user id of the file's owner
stats.gid // the group id of the file's owner
stats.mode.toString(8) // the file's permissions, as an octal string
```

Exercise 2

- Create a node.js server
- When loading the homepage, display a random password.
- Generated from X random words selected from a dictionary
- The number of words (X) is defined as a parameter in the query.
  - Accessible at `req.url`

HTTP REQUESTS

HTTP requests

- Make an HTTP request:
  - GET
  - POST
  - PUT
  - DELETE
  - ...
- https and http packages

HTTP requests - Example 1 GET

```
const https = require('https');
const options = {
   hostname: 'www.google.com',
   port: 443,
   path: '/',
   method: 'GET',
   method: 'GET'.
};
const req = https.request(options, res => {
   console.log(`statusCode: ${res.statusCode}`);
   res.on('data', d => {
      process.stdout.write(d);
   });
});
req.on('error', error => {console.error(error);})
```

```
req.end();
```

HTTP requests - Example 1 GET

```
const req = https.request(options[, callback]);
```

- options: String and object with options
- callback: Executed when the response is received.
- For the GET method you can use https.get(options[, callback])

HTTP requests - Example 1 GET

```
req.on('error', error => {console.error(error);});
emitter.on(eventName, listener);
```

- Define a callback for an event
- Other possible values:
  - timeout
  - data
- Failure to specify the 'error' event could raise an exception and stop the program.
- Further information

HTTP Requests - Example 2 GET

```
const https = require('https');

https.get('https://api.nasa.gov/planetary/apod?api_key=DEMO_KEY', (resp) => {
   let data = '';
   resp.on('data', (chunk) => {
      data += chunk;
   });
   resp.on('end', () => {
      console.log(JSON.parse(data));
   });
}).on("error", (err) => {
   console.log("Error: " + err.message);
});
```

HTTP requests - Example 3 POST

```
const  http =  require('http');
const  port =  3000;
const  server  = http.createServer((req,    res)  =>  {
   req.on('data', d => {
      process.stdout.write(d);
   });
   req.on('end', () => {
      console.log('\nNo more data');
   });
   res.statusCode = 200;
   res.setHeader('Content-Type', 'text/html');
   res.end('<h1>Hello, World!</h1>');
});
server.listen(port, () => {console.log(`Server running at port ${port}`);});
```

Server for processing POST messages

HTTP requests - Example 3 POST

```
const https = require('http');
const data = "Message";
const options = {
   hostname: 'localhost',
   port: 3000,
   path:      '/all',
   method: 'POST',
   headers: {
      Content-Type':  'text/html',
      'Content-Length': data.length
   }
};
const req = https.request(options, res => {
   console.log(`statusCode: ${res.statusCode}`);
   res.on('data', d => {process.stdout.write(d);});
});
req.on('error', error => {console.error(error);});
req.write(data);
req.end();
```

Client to send a POST message

HTTP requests - Example 3 POST

- Equivalent to GET
- Headers need to be added
- The message should be added
- PUT/DELETE would be equivalent to POST

```
res.on('data', d => {
  process.stdout.write(d);
});
```

- Process the incoming response in binary

Exercise 3

Web scraping

- Extracting data from web pages
- Automatically using bots
- Once the information has been extracted, it is processed
- The process is repeated over time to compare the evolution of the data.
- Examples:
  - Google extracting information from websites for the search engine
  - Internet Archive to preserve old websites
  - Price comparison sites

Exercise 3

- Create a server
  - Initialise package.json
  - Periodically downloading HTML from a website
  - Process it to extract specific information
  - Optional: use a package like cheerio to manipulate the DOM.
- Care
  - Some websites require JS to display
  - Don't make too many requests in too short a time

EXPRESS

Express

- Web framework
- Allows HTTP methods to be defined
- Allows you to define the routes
- Enables working with middleware
- Unopinionated
  - Does not define the template engine
  - Does not define the database
- Installation:

```
npm install express
```

- API

Express

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => {
   res.send('Hello World!');
});

app.listen(port, () => {
   console.log(`Example app listening at http://localhost:${port}`);
});
```

Express

app.METHOD(path, callback [, callback ...])

- METHOD: get, post, delete... (full list)
- path: String, pattern, regular expression...
- callback: One or more functions

Express

Middleware

- Code that we execute in the middle of another execution
- We need to invoke the next() method to continue the chain.
- Information
- Express middleware

```
var myLogger = function (req, res, next) {
    console.log('LOGGED');
    next();
}
app.use(myLogger);
```

Express - templates

Template engines

- Static templates for generating views
- Facilitates the design of an HTML page
- At runtime:
  - Variables are replaced by their values
  - It is transformed into an HTML file that is sent to the client.
- Express supports several
- You can create your own engine
- Further information

Express - templates

Pug

- Formerly Jade
- Based on Haml (HTML abstraction markup language)
- Information

```
//index.pug
html
    head
        title= title
    body
        h1= message
```

Express - templates

EJS

- Embedded JavaScript templates
- Use HTML
- Extra labels for control flow and variables

  <% ... %>

  <%= ... %>
- Information
- Demo

Express - templates

EJS

```
//index.ejs
<!DOCTYPE html>
<html>
    <head>
        <title><%= title %></title>
    </head>
    <body>
        <h1><%= message %></h1>
    </body>
</html>
```

Express - express-generator

- Information
- Run express-generator:

```
npx express-generator -v ejs express_test
```

- Alternatively install it in global:

```
npm install -g express-generator
express-generator -v ejs express_test
```

- Later:

```
cd express_test
npm install
DEBUG=express-test:* npm start
```

Express - express-generator

- File structure:

```
.
|-- app.js
|-- bin
|   `-- www
|-- package.json
|-- public
|   |-- images
|   |-- javascripts
|   `-- stylesheets
|       `-- style.css
|-- routes
|   |-- index.js
|   `-- users.js
`-- views
    |-- error.ejs
    `-- index.ejs
```

Express

package.json

```
{
  "name": "express-test",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    start": "node ./bin/www": "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser":"~1.4.4",
    "debug": "~2.6.9",
    "ejs":"~2.6.1",
    "express": "~4.16.1",
    "http-errors":"~1.6.3",
    "morgan": "~1.9.1": "~1.9.1".
  }
}
```

Express

- /bin/www

```
#!/usr/bin/env node
var app = require('../app');
var debug = require('debug')('express-test:server');
var http = require('http');
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
var server = http.createServer(app);
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
function  normalizePort(val) { /*..*/   }
function  onError(error) { /*..*/ }
function  onListening() { /*..*/ }
```

Express

app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join( dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join( dirname, 'public')));
app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler app.use(function(req,
res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page res.status(err.status
  || 500); res.render('error');
});
module.exports = app;
```

Express

```
public
|     |-- images
|     |-- javascripts
|     `-- stylesheets
|         `-- style.css
```

- Static files
- We could add html files
- Linked in app.js with:

```
app.use(express.static(path.join( dirname, 'public')));
```

Express

Routes folder:

- Define the routes
- How the application responds to an endpoint (URI + HTTP method)
- Linked in app.js:

```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
//...
app.use('/', indexRouter);
app.use('/users', usersRouter);
```

Express

- routes/index.js:

```
var express = require('express');
var router = express.Router();
/* GET home page. */
router.get('/', function(req, res, next) {
   res.render('index', { title: 'Express' });
});
module.exports = router;
```

Express

Folder views

- Define the templates
- Linked in app.js:

```
app.set('views', path.join( dirname, 'views'));
app.set('view engine', 'ejs');
```

Express

- views/index.ejs

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Exercise 4

- Create a new project using express-generator
- Create a home page that has the following:
  - A list of items to be loaded from the server (images, text arrays...).
  - A login button
- Create a login page

Express - Example

Example of a server with login:

- Home page
- Login page
- Restricted page (only accessible if the user has logged in)

Express

res.locals

- Object with local variables for the request
- Available in the views
- Only available during the request
- Info

app.locals

- Object with local application variables
- Available in the views
- Available throughout the application
- Info

SOCKET.IO

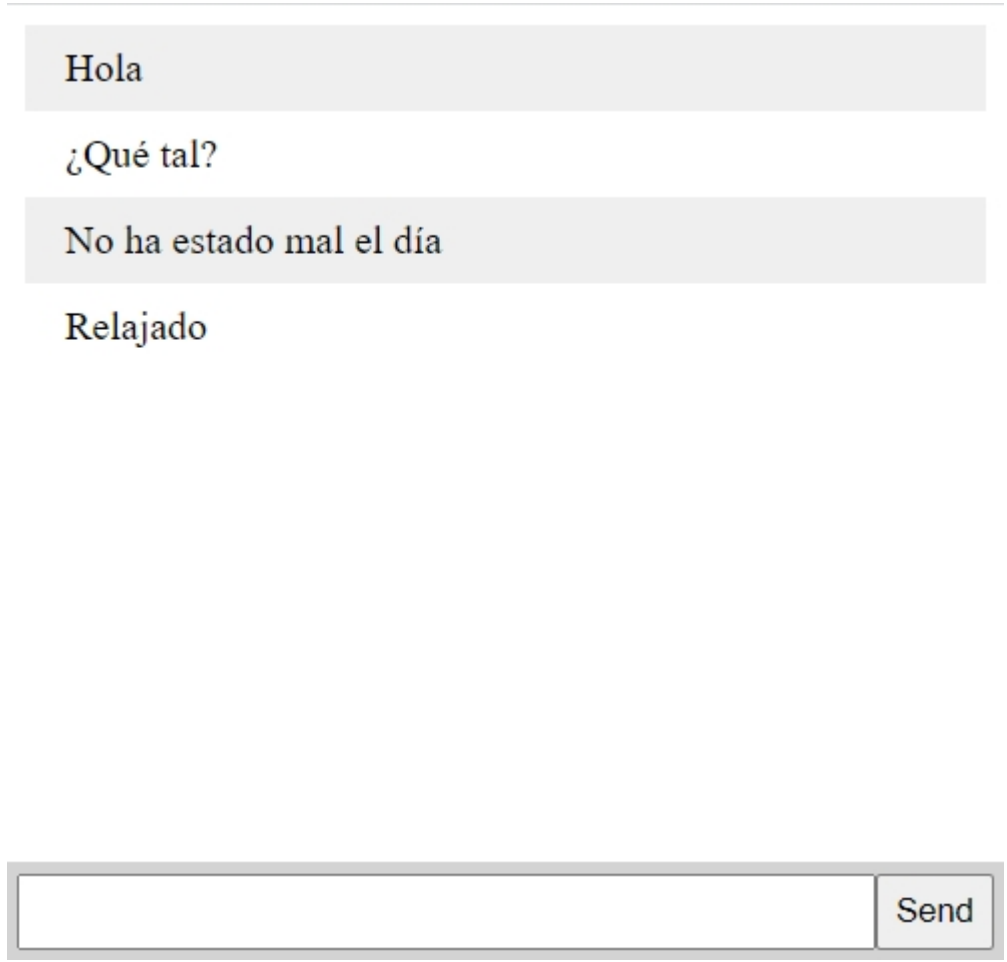
Socket.IO

- Communication server ⟷ client
- Bidirectional
- WebSockets
- Reconnection in case of failure
- Scalable
- Send and receive events
- Link
- Installation:

```
npm install socket.io
```

Socket.IO - Example

- Chat application
- chat.html



Socket.IO

```
//server
const express = require("express"); const
{ createServer } = require("http"); const
{ Server } = require("socket.io"); const
app = express();
const httpServer = createServer(app);
const io = new Server(httpServer, { /* options */ });
//…
io.on('connection', (socket) => {
   console.log('a user connected');
   socket.on('disconnect', () => {
      console.log('user disconnected');
   });
});
httpServer.listen(3000);
```

Socket.IO

```
//client
<html>
   <body>
      //...
      <script src="/socket.io/socket.io.js"></script>
      <script>
         const socket = io();
      </script>
   </body>
</html>
```

Socket.IO

- Start in verbose mode:

DEBUG=* node index.js

```
  socket.io-parser decoded 0 as {"type":0,"nsp":"/"} +0ms
  socket.io:client connecting to namespace / +0ms
  socket.io:namespace adding socket to nsp / +0ms
  socket.io:socket socket connected - writing packet +0ms
  socket.io:socket join room kyqKxJzvWCDwRpqsAAAB +0ms
  socket.io-parser encoding packet {"type":0,"data":{"sid":"kyqKxJzvWCDwRpqsAAAB"},"nsp":"/"} +3ms
  socket.io-parser encoded {"type":0,"data":{"sid":"kyqKxJzvWCDwRpqsAAAB"},"nsp":"/"} as 0{"sid":"kyqKxJzvWCDwRpqsAAAB"} +0ms
  engine:socket sending packet "message" (0{"sid":"kyqKxJzvWCDwRpqsAAAB"}) +4ms
a user connected
  engine upgrading existing transport +7ms
  engine:transport readyState updated from undefined to open (websocket) +22ms
  engine:socket might upgrade socket transport from "polling" to "websocket" +3ms
  engine intercepting request for path "/socket.io/" +1ms
  engine handling "GET" http request "/socket.io/?EIO=4&transport=polling&t=NrdXDVX&sid=X1AwAPP5Rvmkw7kDAAAA" +0ms
  engine setting new request for existing client +0ms
  engine:polling setting request +7ms
  engine:socket flushing buffer to transport +1ms
  engine:polling writing "40{"sid":"kyqKxJzvWCDwRpqsAAAB"}" +1ms
  engine:socket executing batch send callback +0ms
  engine:ws received "2probe" +0ms
  engine:socket got probe ping packet, sending pong +4ms
  engine:ws writing "3probe" +2ms
  engine intercepting request for path "/socket.io/" +6ms
```

Socket.IO

- Send a message:

```
socket.emit(/* message */);
```

- Send a message to all connected clients:

```
io.emit(/* message */);
```

- Broadcasting (sending a message to everyone but oneself):

```
socket.broadcast.emit(/* ... */);
```

- Rooms
- Emit cheatsheet

Socket.IO

- Reserved event names:
  - connect
  - connect_error
  - disconnect
  - disconnecting
  - newListener
  - removeListener

Exercise 5

Integrate the Socket.IO example with the express login example to achieve the following functionality:

- Only logged-in users have access to chat
- The user's name to appear on every message sent.
- History of the last messages sent Extras:
- Private chats between users
- Chat is always available, regardless of the page the user is on.

LOGGING

What is logging?

What is logging?

- Save relevant information from our programme for further analysis.
- Objectives:
  - Check that the programme is working properly
  - Fixing bugs
  - Parameter analysis

What information do we want to log?

What information do we want to log?

- Changes in programme status
- User interaction
- Interaction with other programmes
- Interaction with files
- Communications
- Each time a method is entered
- Every time you leave a method
- Errors
- Exceptions

What information do we NOT want to log?

What information do we NOT want to log?

- Sensitive information
  - Personal information (DNI, Names and surnames, Email...)
  - Medical information
  - Financial information (credit cards, money...)
  - Passwords
  - IP Addresses
- Beware of URLs that may contain sensitive information: `/user/<email>`
- Be careful with the size of the log

Why not use console.log()?

Why not use console.log()?

console.log()	Logging
It cannot be deactivated. We would have to delete all lines.	Can be activated/deactivated
It is not granular. Everything is printed.	You can use different levels and only print the ones that suit you.
Mixed with other information in the console	Distinguishable on the console
Not persistent	We can use files or other media
There is no information other than what we add.	Metadata can be added

Levels

- **Fatal**: Catastrophic situation from which we can't recover
- **Error**: Error in the system that stops an operation, but not the whole system.
- **Warn**: Conditions that are not desirable, but are not necessarily errors.
- **Info**: Information message
- **Debug**: Diagnostic information
- **Trace**: All possible details of the application behaviour

console

- Simple
- Similar to browser version
- Global object
- Documentation
- Specification
- Various versions:

```
console.error()
console.warn()
console.log()
console.debug()
```

Logging

A library should be used to facilitate logging:

- Slight
- Formatting
- Distribute logs
  - Terminal
  - File
  - Database
  - HTTP
  - ...

Logging

Node.js logging libraries:

- Winston
- Pine
- And many more

Logging - Morgan

- Developed by expressjs (info)
- Used for automatic logging (middleware)

```
const logger = require('morgan');
if (app.get('env') == 'production') {
   app.use(logger('common', {
      skip: function(req, res) {return res.statusCode < 400},
      stream: dirname + '/../morgan.log'.
   }));
} else {
   app.use(logger('dev'));
}
```

Logging - Winston

- Simple
- Support for multiple transports
- Uncouples parts of the logging process
- Flexibility in format
- Allows you to define your own levels
- Installation: `npm install winston`
- Info
- express-winston

Logging - Winston

- Example configuration (logger.js file):

```
const winston = require('winston');
const logger = winston.createLogger({
   level: 'info',
   format: winston.format.json(),
   defaultMeta: { service: 'user-service' },
   transports: [
      new winston.transports.File({ filename: 'error.log', level: 'error' }),
      new winston.transports.File({ filename: 'combined.log' })  ],
});
if (process.env.NODE_ENV !== 'production') {
   logger.add(new winston.transports.Console({
      format: winston.format.simple(),
   }));
module.exports = logger;
}
```

Logging - Winston

- Example of use:

```
const logger = require('../logger'); //O require('logger');
logger.log({
   level: 'info',
   message: 'Hello distributed log files!
});
logger.log('info', 'Info message');

logger.info('Another info message');
logger.warn("Warning message");
logger.error("Error message");
```

DATA BASE

Databases

- Node.js can work with any DB
- SQL:
  - My SQL
  - Oracle
  - SQLite
  - PostgreSQL
- NoSQL:
  - MongoDB
  - Cassandra
  - Regis
- Works best with NoSQL DBs (we will see this in SW2).

Databases

- Two very similar mySQL libraries
  - mysql (API)
  - mysql2 (API)
- Differences

Databases

- Require

```
//npm install mysql2
const mysql = require('mysql2');
```

- Configuration:

```
const con = mysql.createConnection({/*Options*/)
```

- Initial connection:

```
con.connect(function(err) {/*...*/});
```

Databases

- Query:

```
con.query("Query", funtion(err) {/*...*/});
```

- Prepared statements:

```
con.execute(
   SELECT * FROM `table` WHERE `name` = ? AND `age` > ?', ['Rick
   C-137', 53],
   function(err, results, fields) {/*...*/}
);
```

- Close the connection:

```
con.end();
```

Databases

```
const mysql = require('mysql2'); const
con = mysql.createConnection({
   host: "localhost", user:
   "yourusername",
   password:
   "yourpassword".
});
con.connect(function(err) {
   if (err) throw err;
   console.log("Connected!");
   con.query("CREATE DATABASE mydb", function (err, result) { if
      (err) throw err;
      console.log("Database created");
   });
});
```

Databases

```
const mysql = require('mysql2'); const
con = mysql.createConnection({
   host: "localhost",
   user: "yourusername",
   password: "yourpassword",
   database: "mydb".
});
con.query(
   SELECT * FROM `table` WHERE `name` = "Page" AND `age` > 45',
   function(err, results, fields) {
      if (err) throw err;
      console.log(results); // results contains rows returned by server console.log(fields);
      // fields contains extra meta data about results, if available
   }
);
```

Databases - Sequelize

- ORM (Object-relational mapping)
- It allows working with multiple DBs:
  - Oracle
  - Postgres
  - MySQL
  - SQLite
  - ...
- Installation: `npm install sequelize`
- Then install the driver, for example: `npm install sqlite3`
- Web

Sequelize - Example

- Based on the example Login from the Express theme
- Added registration page
- Added user management with Sequelize and SQLite