CEU | Universidad San Pablo

Sistemas Web I

Grado en Ingeniería de Sistemas de Información

Álvaro Sánchez Picot
alvaro.sanchezpicot@ceu.es

v20241028

INTRODUCCIÓN

Introducción

- Open-source
- Cross-platform
- JavaScript runtime environment
- V8 JavaScript engine (the core of Google Chrome)
- Versión inicial: 2009
- 2014 fork io.js (se solucionó en 2015)
- Leftpad incident en 2016
- Más de 2,500,000 de paquetes
- Documentación

Introducción – Uso por empresas

Netflix

- Cambió su backend de cliente en 2013 de Java a node.js
- Más información

Paypal

- Cambió parte de su backend en 2013 de Java a node.js
- Más información

Linkedin

- Cambió el backend de la parte móvil en 2011 de Ruby on Rails a node.js
- Más información

Y muchas otras empresas

Introducción

Mismos principios que JS:

- Un único proceso
- Síncrono
- Paradigmas no bloqueantes
  - Bloquear es la excepción
- Gestión de eventos

Introducción

Diferencias con JavaScript y programar para el navegador:

- No se trabaja con el DOM
- No están las variables del navegador como *document* o *window*
- No tienes restricciones de acceso a ficheros
- No tienes problemas de versiones del navegador (incompatibilidad con versiones antiguas de JS)

INSTALACIÓN

Herramientas necesarias

- Git Bash para Windows
  - Linux y macOS ya tienen consola
- IDE
  - Visual Studio Code
  - Webstorm
- Editores ligeros
  - Atom
  - Sublime Text
  - Notepad++
- Postman

Instalación

- Ir a: https://nodejs.org/
- Descargarse e instalar la última versión LTS
- API

Instalación

¿Qué significan los números de la versión X.Y.Z?

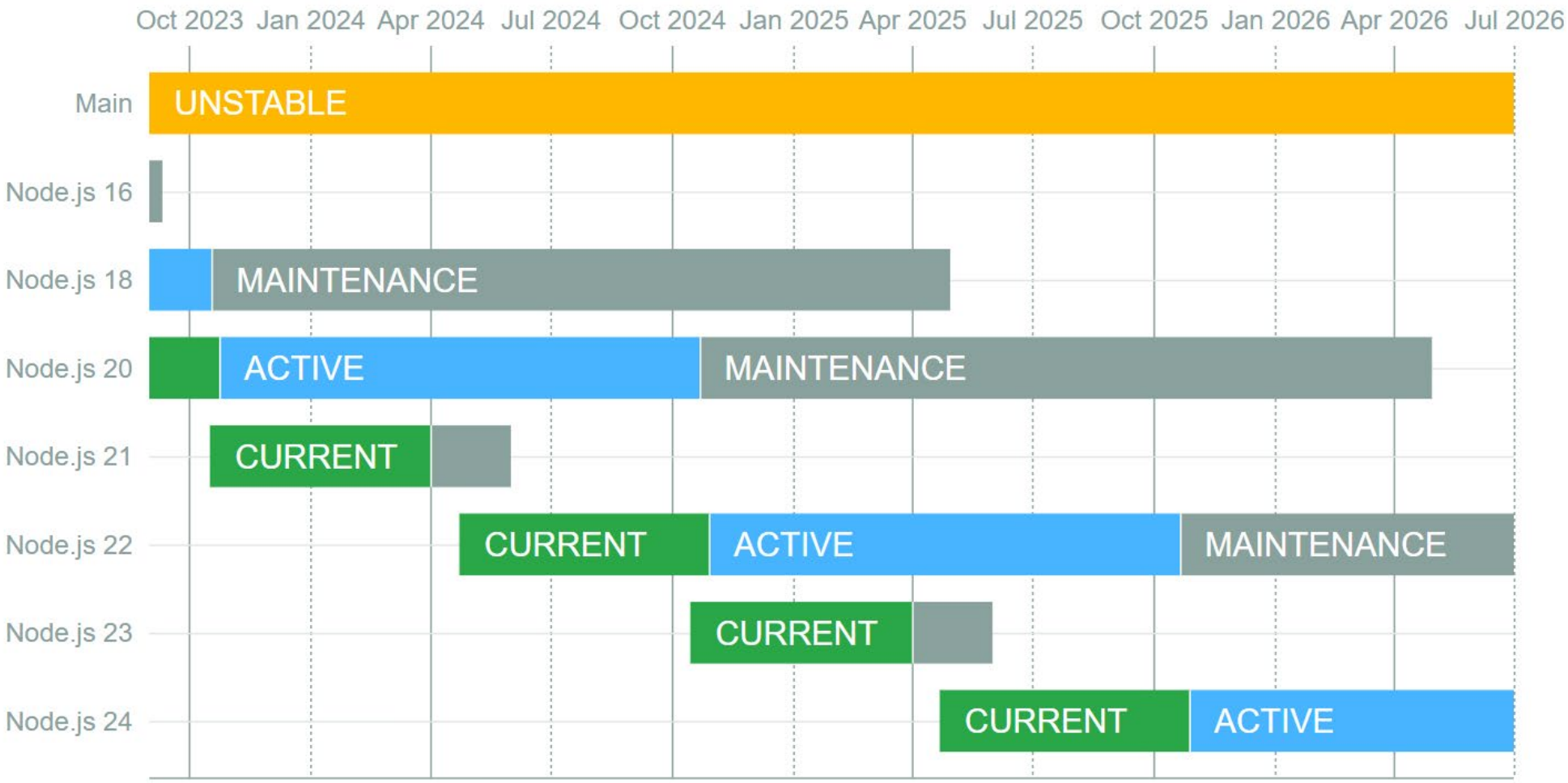
- Estándar de numeración de versiones para software
- Semantic Versioning (SemVer): https://semver.org/
- MAJOR.MINOR.PATCH
  - MAJOR: Cambios incompatibles
  - MINOR: Se añade funcionalidad retrocompatible
  - PATCH: Corrección de errores retrocompatibles

Instalación

¿Qué es LTS?

- Long Term Support
- Son versiones que tienen soporte durante mucho más tiempo
- Esto aplica a muchos programas de software (ej. Java, Ubuntu…)
- https://nodejs.org/en/download/releases

Instalación



Instalación – nvm

- Node Version Manager
- Trabajar con diferentes versiones de Node.js
- Para Windows descargarla de aquí:
  - https://github.com/coreybutler/nvm-windows/releases
  - nvm-setup.zip
- Para macOS y Linux, seguir las siguientes instrucciones:
  - https://github.com/nvm-sh/nvm#installing-and-updating

Instalación – nvm

`nvm list`: Lista las versiones de Node.js instaladas

`nvm use x.y.z`: Activa la versión x.y.z de Node.js.

`nvm install x.y.z`: Instala la versión x.y.z de Node.js.

Versiones de Node.js: https://nodejs.org/download/release

`nvm --help`: Muestra la ayuda

Instalación – npm

- Node Package Manager
- Gestor de paquetes de Node.js
- Instalado junto con Node.js
- También se usa para JS en el frontend
- Gestiona las dependencias del proyecto:
  - Descarga
  - Actualización
  - Versiones
- Permite definir y ejecutar tareas
- Web

Instalación – npm

- Crear el archivo de configuración package.json

```
npm init
```

- Instalación de las dependencias
  - Requiere archivo de configuración `package.json`

```
npm install
npm i
```

- Instalación de un paquete

```
npm install nombre_paquete
```

- Instalación de la versión x.y.z de un paquete

```
npm install <nombre_paquete>@<versión>
```

Instalación – npm

- Listar los paquetes instalados junto con la versión

```
npm list
npm ls
```

- Actualizar todas las dependencias

```
npm update
```

- Actualizar un paquete específico

```
npm update nombre_paquete
```

- Eliminar un paquete

```
npm uninstall nombre_paquete
```

Instalación – npm

- Ver información de un paquete

```
npm view nombre_paquete
```

- Ver las versiones disponibles de un paquete

```
npm view nombre_paquete versions
```

- Información de los comandos

REPL

- Read Evaluate Print Loop
- Entorno de Node.js en forma de consola
- Para ejecutarlo: `node`
- Se puede usar el tabulador para autocompletar
- Comandos especiales:
  - `.help`: Muestra la ayuda
  - `.exit`: Sale de REPL (equivalente a pulsar Ctrl+C dos veces o Ctrl+D)

npx

- Permite ejecutar código
- No hace falta tener instalado el paquete

```
npx cowsay "Hello"
```

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

Ejemplo 0

Nuestro primer servidor, lo guardamos en `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}`);
});
```

Ejemplo 0

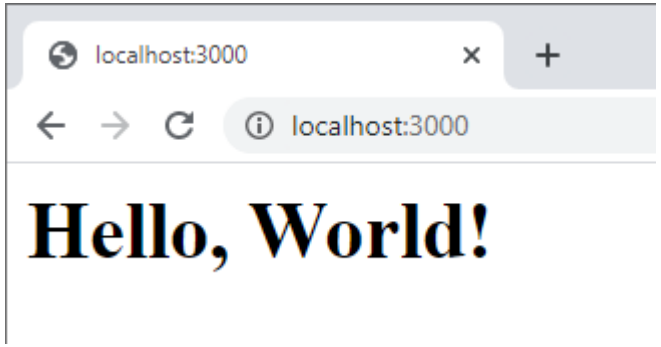
- Arrancamos el servidor con la siguiente instrucción:

```
node index.js
```

- Deberíamos ver que el servidor arranca correctamente:

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

- Desde el navegador nos conectamos a localhost:3000
- Si todo ha ido correctamente deberíamos ver:



Ejemplo 0

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

- `require` sirve para importar módulos (librerías)
  - Equivalente a import en Java
- `http` es un módulo que está incluido en el core de Node.js
  - No hace falta instalar nada extra
  - Documentación de `http`
  - Listado de módulos core:
    - `url`: para trabajar y parsear URLs
    - `path`: para trabajar y parsear paths
    - `fs`: para trabajar con I/O en archivos
    - `util`: funcionalidades diversas

Ejemplo 0

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

- `createServer` es un método para crear un servidor HTTP
  - Documentación
- `requestListener` es una función
  - Ejecutada con cada petición al servidor (request)
  - `req` es un objeto http.IncomingMessage con información de la petición
  - `res` es un objeto http.ServerResponse con información de la respuesta

Ejemplo 0

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

- `listen` es un método que arranca el servidor escuchando en un puerto
  - Documentación (idéntico al del módulo http). Varias posibilidades de invocarlo
- `port` es el puerto donde queremos escuchar
  - Si se omite, se elegirá uno aleatorio sin usar
- host
  - Por defecto la *unespecified IPv6 address* (equivalente a la 0.0.0.0 en IPv4)
- backlog es el número máximo de conexiones pendientes
  - Por defecto es 511
- `callback` se ejecuta cuando se termina de arrancar el servidor

Ejemplo 0

Vamos a añadir un log para ver las conexiones en nuestro servidor, lo arrancamos y recargamos la página:

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

¿Por qué hay dos conexiones al cargar la página en el navegador?

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

Variables de entorno

- Accesibles con el módulo `process` (disponible sin necesidad de require)

```
process.env.nombre_variable
```

- Podemos definirlas al ejecutar el programa

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

- Se suele usar por ejemplo para el puerto en el que arranca el servidor

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

- NODE_ENV
  - Para definir si estás en un entorno de producción o de desarrollo
  - Muchos módulos lo usan
  - Valores: production, development

Variables de entorno

- Se pueden cargar de un archivo .env con el módulo dotenv
  - npm install dotenv

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

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

Argumentos

- Se pueden añadir al ejecutar el programa

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

- Accesibles con `process.argv`
  - Devuelve un array que sigue el convenio UNIX
  - Posición 0: full path del comando node
  - Posición 1: full path del archivo
  - Posiciones 2 en adelante: los argumentos

Argumentos

- Si queremos usar clave=valor hay que parsearlo
- Podemos usar el módulo minimist para tratar los argumentos
  - Los argumentos se pasan con `--nombre=valor`
  - También se pueden usar opciones como en unix con `-opción valor`
    - opción es una letra
  - npm install minimist

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

process

- Objeto global con información general de la ejecución de node
- Más información

```
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

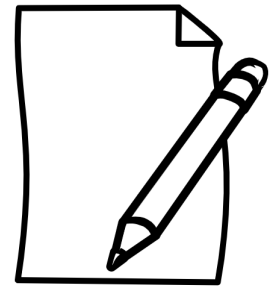
```
process.memoryUsage() // Return an object with memory usage details.
process.nextTick() // Invoke a function soon.
process.pid // The process id of the current process.
process.platform // The OS: "linux", "darwin", or "win32", for example.
process.resourceUsage() // Return an object with resource 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

- Módulo con acceso a información de bajo nivel sobre el SO
- Necesita ser cargado
- Más información

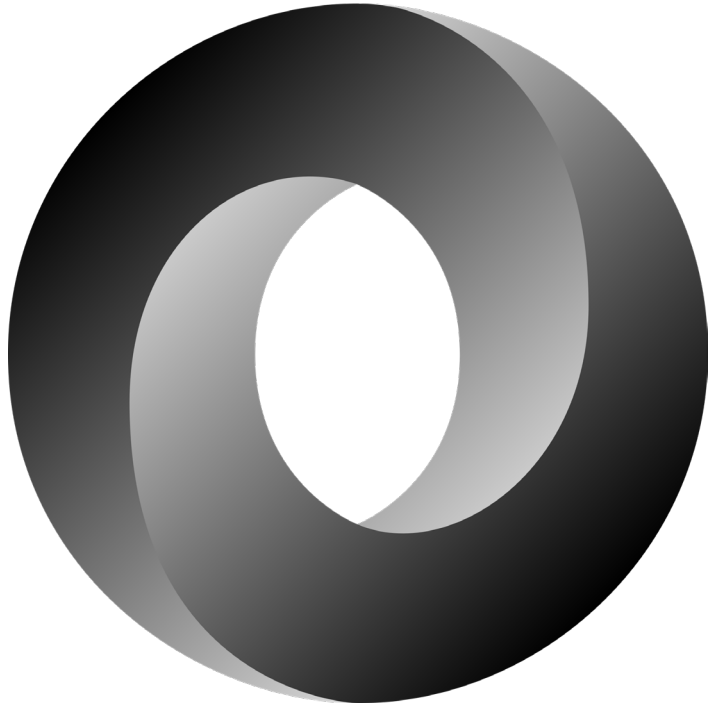
```
const os = require("os");
os.cpus() // Data about system CPU cores, including usage times.
os.freemem() // Returns the amount of free RAM in bytes.
os.homedir() // Returns the current user's home directory.
os.hostname() // Returns the hostname of the computer.
os.loadavg() // Returns the 1, 5, and 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.
```

Ejercicio 1



- Crea un servidor en node.js
- Que al iniciar muestre por consola información del sistema y versión de node.js
- De forma periódica muestre por consola la siguiente información:
  - Uso de CPU
  - Uso de memoria
  - Tiempo que el sistema lleva activo
  - Tiempo que lleva ejecutándose node.js
- Que la información que se muestra periódicamente sea configurable en un fichero, incluyendo cada cuantos segundos se muestra

JSON



JSON

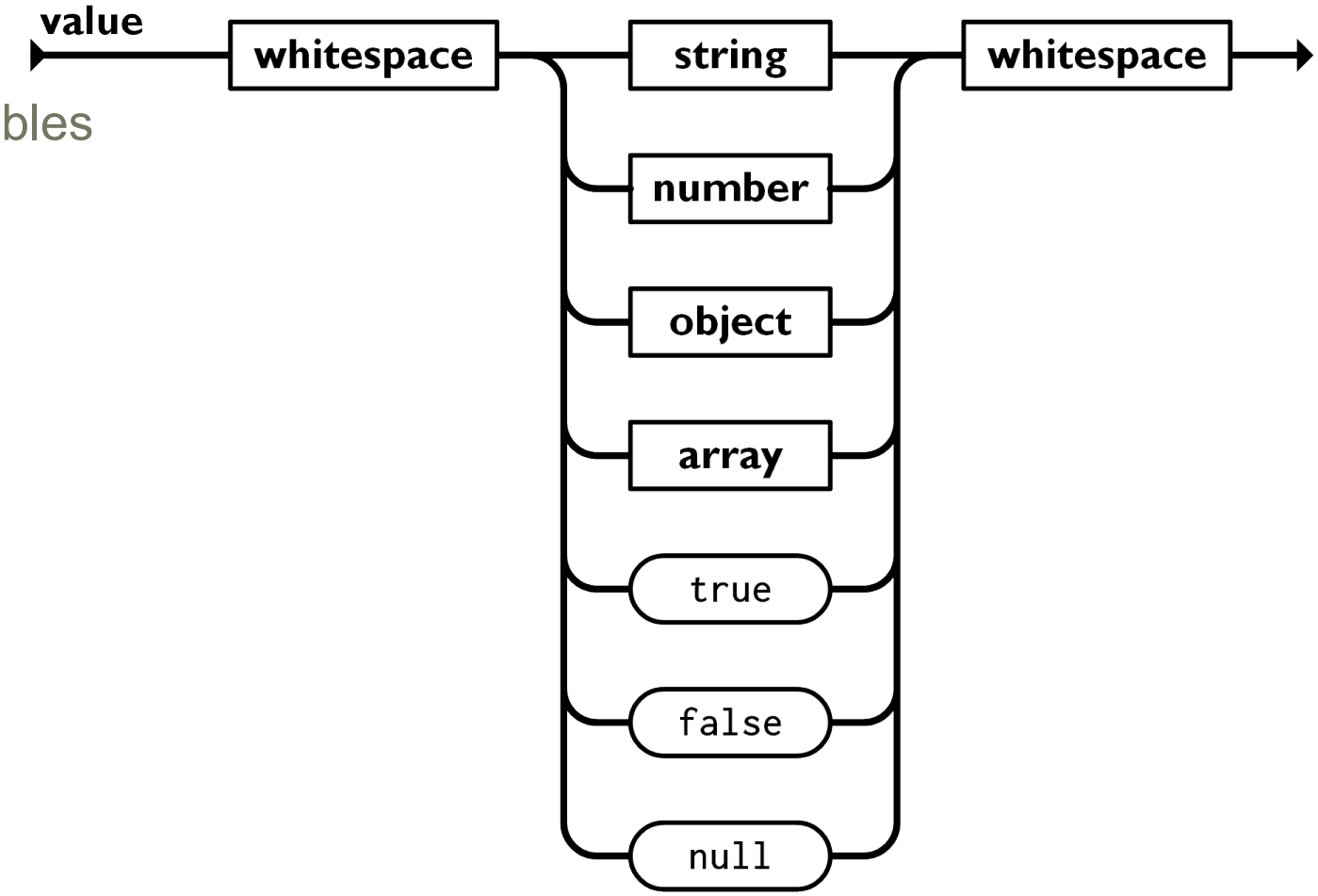
- JavaScript Object Notation
- Formato ligero para el intercambio de datos
- Basado en el estándar ECMA-262 de 1999
  - Versión más reciente ECMA-404 de 2017
- Soporta objetos, arrays y valores
- No permite comentarios
- Más información
- Validador
- Guía de estilo

JSON – Ejemplo

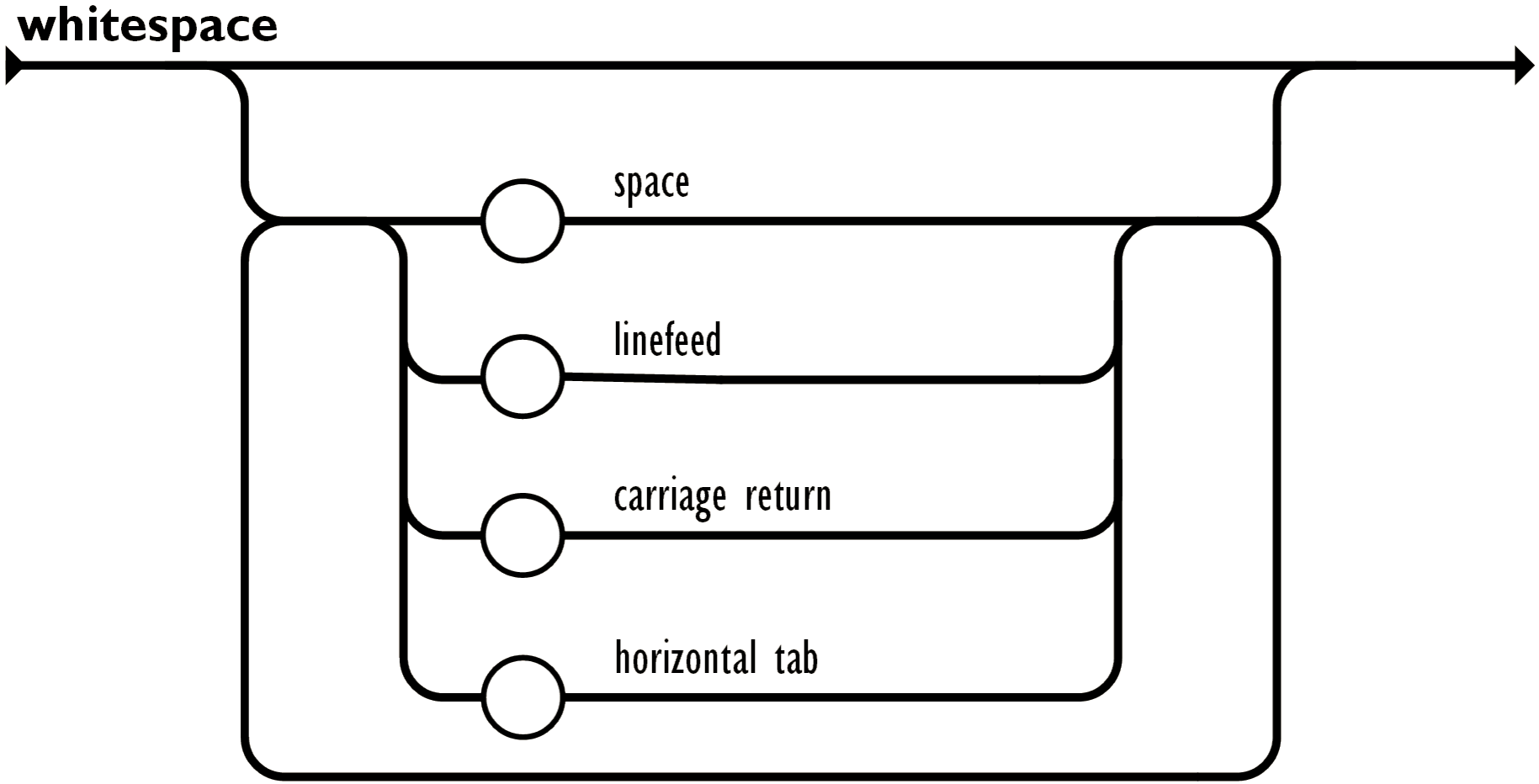
```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

JSON – Sintaxis

- Valor:
  - String: texto entre comillas dobles
  - Número
  - Objeto
  - Array
  - true
  - false
  - null



JSON – Sintaxis



JSON – Sintaxis

- Tiene que haber un único elemento raíz que tiene que ser un valor:
  - JSON válido: `{}`
  - JSON válido: `""`
  - JSON válido: `4`
  - JSON válido: `true`
  - JSON válido:

```
[
  {"nombre" : "Juan"},
  {"nombre" : "Ana"},
  {"nombre" : "Sofía"},
  {"nombre" : "Andrés"}
]
```

JSON – Sintaxis

Objeto:

- Set desordenado de pares clave / valor:

```
"clave": valor
```

- Rodeado por llaves: `{` y `}`
- La clave:
  - Es un String y tiene que tener comillas dobles
  - No puede estar repetida dentro del mismo objeto
  - Recomendable usar nomenclatura lower camel case
- Cada par clave/valor se separa por comas
  - Cuidado con las trailing commas
- Puede estar vacío: `{ }`

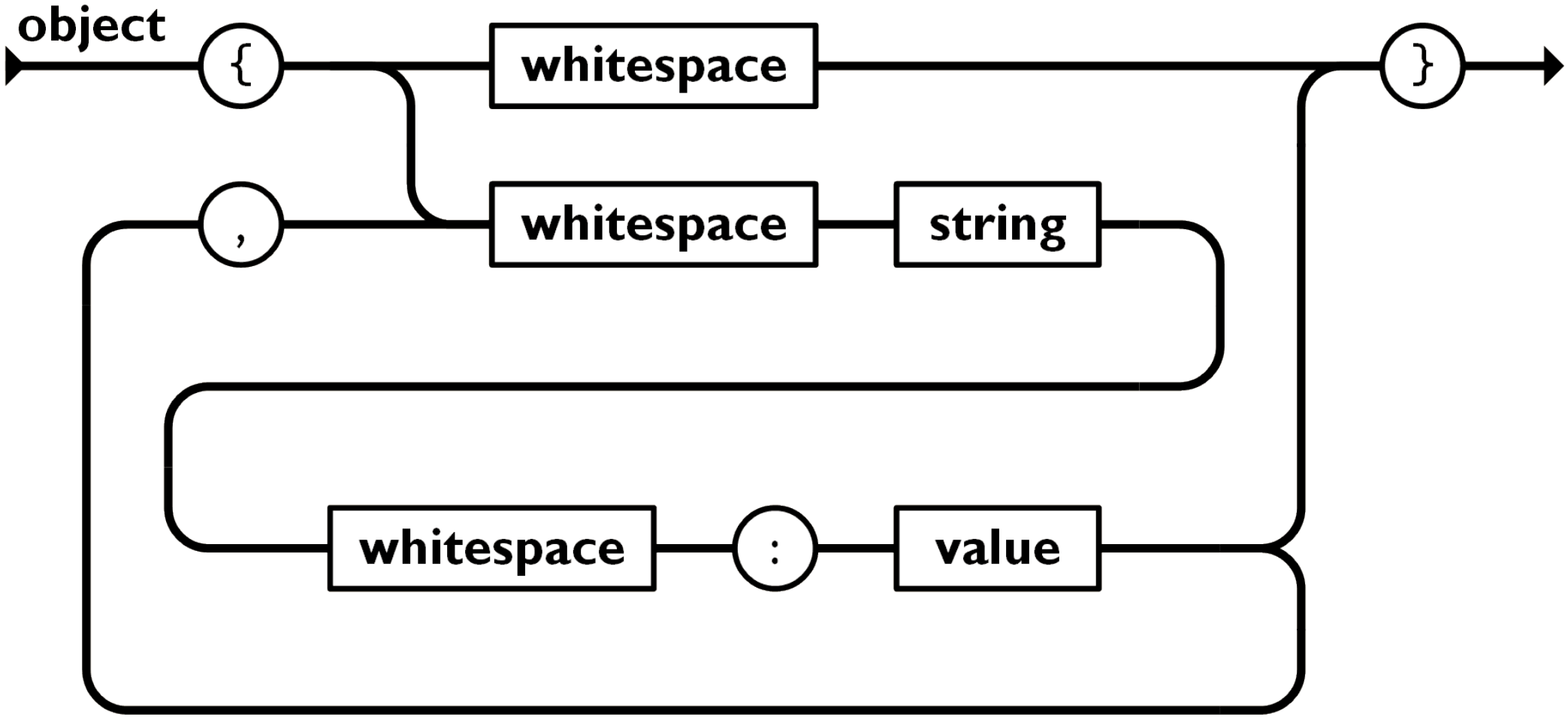
JSON – Sintaxis

Objeto:

```
{
  "nombreCompleto": "Juan Pérez Rodríguez",
  "edad": 27
}
```

Cuidado con la trailing comma

JSON – Sintaxis



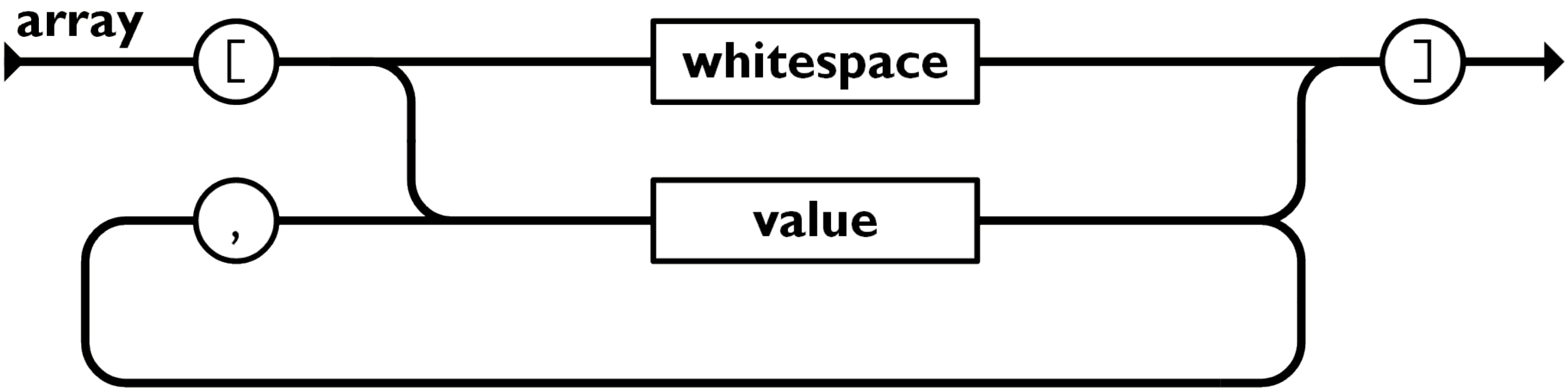
JSON – Sintaxis

Array:

- Colección ordenada de valores
- Rodeado por corchetes: [y]

```
[
  {"movil": 612345678},
  {"fijo": 912345678}
]
```

JSON – Sintaxis



JSON – Sintaxis

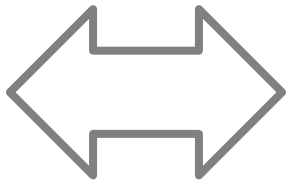
- Se pueden anidar elementos:

```
{
  "nombre": "Juan",
  "direccion": {
    "calle": "Avenida Ciudad de Barcelona 23",
    "ciudad": "Madrid"
  },
  "telefonos": [
    {"movil": 612345678},
    {"fijo": 912345678}
  ],
  "edad": 27
}
```

JSON

- No se garantiza el orden del contenido de los objetos

```
{
   "nombre": "Juan",
   "edad": 27
}
```

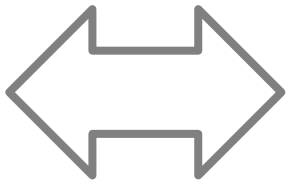


```
{
   "edad": 27,
   "nombre": "Juan"
}
```

JSON

- Los elementos de un array sí que están ordenados

```
[
  {"nombre" : "Juan"},
  {"nombre" : "Ana"}
]
```



```
{
   "edad": 27,
   "nombre": "Juan"
}
```

JSON – Ejercicio

- ¿Está bien formado el siguiente JSON?

```
{
  "nombre": "Juan",
  "telefonos": [
    {"movil": 612345678},
    {"fijo": [912345678]}, false
  ]
  edad: 27, mayorDeEdad: "true",
  "direccion": {
    "calle": "Avenida Ciudad de Barcelona 23",
    "ciudad": Madrid, null
  },
}
```

JSON con JS

- Paquete JSON
  - No es específico de Node.js
- Convertir un objeto JSON a texto

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

- Convertir un texto en formato JSON a objeto

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

JSON con JS

- Recorrer un objeto JSON

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

JSON con JS – Ejemplo

```
let persona = {"nombre": "Juan", "edad": 23};
console.log("Persona:\n", persona);
let text = JSON.stringify(persona);
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(persona, null, 2));
```

package.json

- Formato json
- Usado por npm
- Metadatos del proyecto
  - Dependencias
  - Descripción
  - Versión
  - Licencia
  - Configuración
  - …
- Se puede crear manualmente o usando `npm init`
- Documentación

package.json

Ejemplo de package.json:

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

package.json

```
"name": "test-project"
```

- Nombre del paquete
- Hasta 214 caracteres
- Solo minúsculas, '-' o '_'

```
"version": "1.0.0"
```

- Versión del paquete
- Formato semver

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

- Descripción del paquete

package.json

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

- Punto de entrada del paquete
- Donde se buscará para exportarlo como módulo
- Si no existe el valor por defecto es `index.js`
- Se ejecuta el archivo con `node .`

package.json

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

- Define scripts ejecutables
- Se ejecutan con: `npm run XXXX`
- Algunos especiales se pueden ejecutar sin el run:
  - `npm start`
  - `npm test`

package.json

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

- Array de palabras clave
- Útil para buscar el paquete en npm

package.json

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

- Solo puede haber uno
- Email y url opcionales

package.json

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

- Puede haber varios

package.json

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

- Dependencias del paquete
- Siguen un formato especial para indicar como se actualiza
- Se añaden automáticamente al hacer:

```
npm install nombre_paquete
```

package.json

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

- Dependencias solo para desarrollo
- No necesarias en producción
- Se añaden automáticamente al hacer:

```
npm install --save-dev nombre_paquete
```

package.json

- Y muchas opciones más:
  - `"bugs": "https://github.com/whatever/package/issues"`
  - `"homepage": "https://whatever.com/package"`
  - `"license": "MIT"`
  - `"repository": "github:whatever/testing"`
  - `"private": true`
  - y más…

package.json

- Hay un problema con package.json:
  - El proyecto no es reproducible al 100%

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

- Alguien podría usar la versión 2.5.2, otro la 2.5.3 y otro la 2.6.0
  - Puede dar lugar a inconsistencias en el proyecto
- Alternativa: subir la carpeta node_modules al repositorio
  - No recomendable porque puede llegar a ocupar mucho (varios cientos de megas)

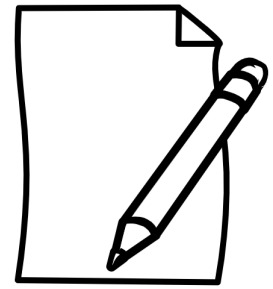
package-lock.json

- Especifica la versión exacta de cada dependencia
  - El paquete es 100% reproducible
- Se genera y actualiza de forma automática
  - Al hacer `npm install`
  - Al hacer `npm update`
- Recordatorio, se pueden ver las dependencias con: `npm list`
- Documentación

package-lock.json

```
{
  "name": "prueba_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…"
    }
  }
}
```

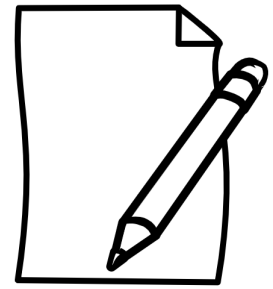
Ejercicio 2



Tenemos actualmente una aplicación web sobre entradas de cine que queremos extender diseñando una API que dé acceso a parte de la información de la base de datos. Para ello vamos a empezar diseñando alguno de los mensajes que se envían.

Condense la información que aparece en la siguiente página en un único mensaje JSON, sin duplicar información innecesaria.

Ejercicio 2



La película Spider-Man: No Way Home ha sido dirigida por Jon Watts y los actores principales son Tom Holland y Zendaya

La película Spider-Man: No Way Home se puede ver en la Sala 17 a las 17:00 y a las 20:15.

La película Spider-Man: No Way Home se puede ver en la Sala 20 a las 18:00 y a las 21:15.

La película The Matrix Resurrections ha sido dirigida por Lana Wachowski y los actores principales son Keanu Reeves y Carrie-Anne Moss.

La película The Matrix Resurrections se puede ver en la Sala 5 a las 19:15 y en la Sala 22 a las 22:30.

EVENT LOOP

Event Loop

- Un único thread
- Se repite (loop)
- Cada iteración es un tick
- Gestiona la ejecución de eventos
- Una cola FIFO de callbacks
  - En cada tick se ejecuta hasta que se vacía
- Gestionado internamente por la librería libuv
- Explicación más detallada
  - Hay varias fases
  - Cada una tiene su propia cola FIFO

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

- No debemos bloquear el event loop
- Cada tick debe ser corto
- El trabajo asociado a cada cliente tiene que ser breve
- Intentar dividir las tareas más intensivas
- Más información

FICHEROS

path

- Módulo con utilidades para trabajar con ficheros y rutas
- Hay que importarlo:

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

- Más información

path

`path.dirname(path)`

- Devuelve el nombre del directorio

`path.sep`

- Separador del SO: \ en Windows, / en UNIX

`path.normalize(path)`

- Elimina separadores duplicados y procesa .. y .

`path.join([…paths])`

- Une los paths usando el separador del sistema y lo normaliza

`path.resolve([…paths])`

- Procesa de derecha a izquierda y genera una ruta absoluta

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
- Acceso e interacción con el sistema de ficheros
- Core de Node.js
- Los métodos son async por defecto
  - Hay versión sync añadiendo Sync al final del método
    - `fs.access()` ↔ `fs.accessSync()`
- Documentación

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: modo lectura (por defecto)
- r+: modo lectura y escritura. No se creará el archivo si no existe
- w+: modo lectura y escritura (al principio). Se crea el archivo si no existe
- a: escritura al final del archivo. Se crea el archivo si no existe

fs – Lectura de ficheros

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

fs – Escritura de ficheros

```
const fs = require('fs');
const content = 'Algo de contenido del fichero';
fs.writeFile('test.txt', content, err => {
  if (err) {
    console.error(err);
    return;
  }
  console.log('El fichero se ha escrito correctamente');
});
```

fs – Promesas

`fs.promises`

- Funciones equivalentes que trabajan con promesas
- Más información

```
fs.promises.open(path, flags[, mode])
fs.promises.readFile(path[, options])
fs.promises.writeFile(file, data[, options])
Y muchos más…
```

fs – Promesas

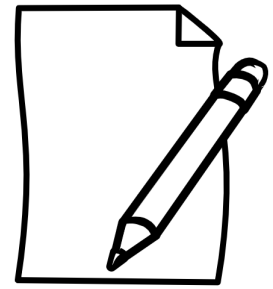
```
fs.promises
  .readFile("data.csv", "utf8")
  .then(processFileText)
  .catch(handleReadError);

// Alternativamente, usando async / await
async function processText(filename, encoding="utf8") {
  let text = await fs.promises.readFile("data.csv", "utf8");
  // Procesar el texto
}
```

fs – stat

```
const fs = require("fs");
let stats = fs.statSync("diccionario.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
```

Ejercicio 3



- Crea un servidor en node.js
- Que al cargar la página principal muestre una contraseña aleatoria
- Generada a partir de X palabras aleatorias seleccionadas de un diccionario
- El número de palabras (X) está definido como parámetro en la query
  - Accesible en `req.url`

PETICIONES HTTP

Peticiones HTTP

- Hacer una petición HTTP:
  - GET
  - POST
  - PUT
  - DELETE
  - …
- Paquetes https y http

Peticiones HTTP – Ejemplo 1 GET

```
const https = require('https');
const options = {
  hostname: 'www.google.com',
  port: 443,
  path: '/',
  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();
```

Peticiones HTTP – Ejemplo 1 GET

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

- `options`: String y objeto con las opciones
- `callback`: Se ejecuta cuando se recibe la respuesta
- Para el método GET se puede usar `https.get(options[, callback])`

Peticiones HTTP – Ejemplo 1 GET

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

- Define un callback para un evento
- Otros valores posibles:
  - `timeout`
  - `data`
- Si no se especifica el evento 'error' podría saltar una excepción y detener el programa
- Más información

Peticiones HTTP – Ejemplo 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);
});
```

Peticiones HTTP – Ejemplo 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}`);});
```

Servidor para procesar mensajes POST

Peticiones HTTP – Ejemplo 3 POST

```
const https = require('http');
const data = "Mensaje";
const options = {
  hostname: 'localhost',
  port: 3000,
  path: '/todos',
  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();
```

Cliente para enviar un mensaje POST

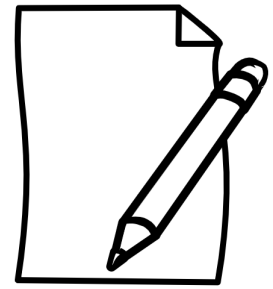
Peticiones HTTP – Ejemplo 3 POST

- Equivalente al GET
- Hay que añadir cabeceras
- Hay que añadir el mensaje
- PUT/DELETE serían equivalentes al POST

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

- Procesar la respuesta que llega en binario

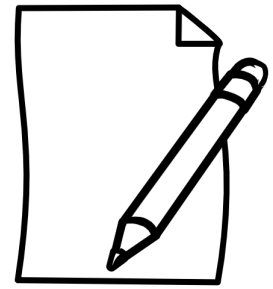
Ejercicio 4



Web scraping

- Extraer datos de páginas web
- De forma automática haciendo uso de bots
- Una vez extraída la información se procesa
- Se repite el proceso a lo largo del tiempo para comparar la evolución de los datos
- Ejemplos:
  - Google extrayendo información de las webs para el buscador
  - Internet Archive para conservar webs antiguas
  - Páginas de comparación de precios

Ejercicio 4



- Crea un servidor
  - Inicializa package.json
  - Que descargue periódicamente el HTML de una web
  - Que lo procese para extraer una información específica
  - Opcional: usa algún paquete como cheerio para manipular el DOM
- Cuidado
  - Algunas webs requieren JS para visualizarse
  - No hagas muchas peticiones en poco tiempo

EXPRESS

Express

- Web framework
- Permite definir métodos HTTP
- Permite definir las rutas
- Permite trabajar con middleware
- Unopinionated
  - No define el template engine
  - No define la base de datos
- Instalación:

```
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… (lista completa)
- `path`: String, patrón, expresión regular…
- `callback`: Una o más funciones

Express

Middleware

- Código que ejecutamos en medio de otra ejecución
- Necesitamos invocar el método `next()` para continuar la cadena
- Información
- Express middleware

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

Express – templates

Template engines

- Plantillas estáticas para generar las vistas
- Facilita el diseño de una página HTML
- En tiempo de ejecución:
  - Se reemplazan las variables por sus valores
  - Se transforma a un archivo HTML que se le envía al cliente
- Express soporta varios
- Te puedes crear tu propio engine
- Más información

Express – templates

Pug

- Antiguamente Jade
- Basado en Haml (HTML abstraction markup language)
- Información

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

Express – templates

EJS

- Embedded JavaScript templates
- Usa HTML
- Etiquetas extras para flujo de control y variables

  `<% … %>`

  `<%= … %>`
- Información
- Demo

Express – templates

EJS

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

Express – express-generator

- Información
- Ejecutar `express-generator`:

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

- Alternativamente instalarlo en global:

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

- Luego:

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

Express – express-generator

- Estructura de archivos:

```
.
|-- 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"
  },
  "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"
  }
}
```

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
```

- Archivos estáticos
- Podríamos añadir archivos html
- Enlazado en app.js con:

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

Express

Carpeta `routes`:

- Define las rutas
- Cómo responde la aplicación a un endpoint (URI + método HTTP)
- Enlazado en 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

Carpeta views

- Define los templates
- Enlazado en app.js:

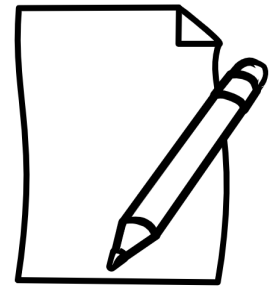
```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

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>
```

Ejercicio 5



- Crea un proyecto nuevo usando express-generator
- Crea una página inicial que tenga lo siguiente:
  - Una lista de elmentos que se carguen desde el servidor (imágenes, array de texto...)
  - Un botón para hacer login
- Crea una página que sea login

Express - Ejemplo

Ejemplo de servidor con login:

- Página inicial
- Página de login
- Página restringida (solo accesible si el usuario ha hecho login)

Express

`res.locals`

- Objeto con variables locales para la petición
- Disponibles en las vistas
- Solo disponible durante la petición
- Info

`app.locals`

- Objeto con variables locales de la aplicación
- Disponibles en las vistas
- Disponible durante toda la aplicación
- Info

SOCKET.IO

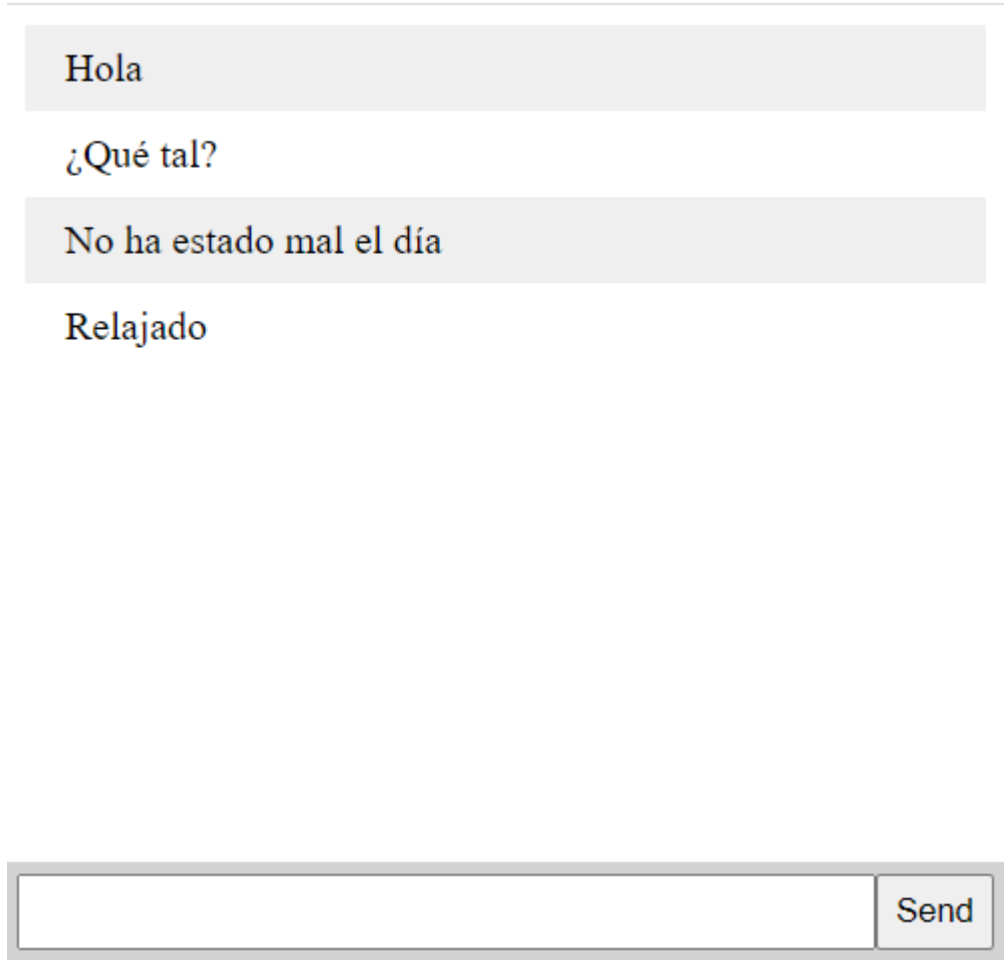
Socket.IO

- Comunicación servidor ⟷ cliente
- Bidireccional
- WebSockets
- Reconexión en caso de fallo
- Escalable
- Enviar y recibir eventos
- Link
- Instalación:

```
npm install socket.io
```

Socket.IO - Ejemplo

- Aplicación de chat
- chat.html



Socket.IO

```
//servidor
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

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

Socket.IO

- Arrancar en modo verbose:

```
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

- Enviar un mensaje:

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

- Enviar un mensaje a todos los clientes conectados:

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

- Hacer broadcast (enviar un mensaje a todos menos a uno mismo):

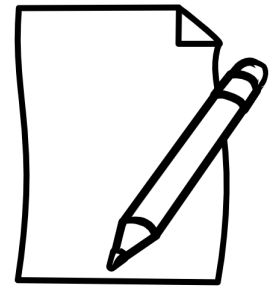
```
socket.broadcast.emit(/* ... */);
```

- Rooms
- Emit cheatsheet

Socket.IO

- Nombres de eventos reservados:
  - connect
  - connect_error
  - disconnect
  - disconnecting
  - newListener
  - removeListener

Ejercicio 6



Integra el ejemplo de Socket.IO con el ejemplo de login de express para conseguir las siguientes funcionalidades:

- Que sólo los usuarios logeados tengan acceso al chat
- Que aparezca el nombre del usuario en cada mensaje que se envía
- Histórico de los últimos mensajes que se han enviado

Extras:

- Chats privados entre usuarios
- Chat siempre disponible, independiente de la página en la que esté el usuario

LOGGING

¿Qué es logging?

¿Qué es hacer logging?

- Guardar información relevante de nuestro programa para su posterior análisis
- Objetivos:
  - Comprobar que el programa funciona correctamente
  - Solucionar bugs
  - Análisis de parámetros

¿Qué información queremos loggear?

¿Qué información queremos loggear?

- Cambios en el estado del programa
- Interacción con el usuario
- Interacción con otros programas
- Interacción con ficheros
- Comunicaciones
- Cada vez que se entra en un método
- Cada vez que se sale de un método
- Errores
- Excepciones

¿Qué información NO queremos loggear?

¿Qué información NO queremos loggear?

- Información sensible
  - Información personal (DNI, Nombres y apellidos, Email…)
  - Información médica
  - Información financiera (Tarjetas de crédito, dinero…)
  - Contraseñas
  - Direcciones IP
- Cuidado con las URL porque podrían tener información sensible: `/user/<email>`
- Cuidado con el tamaño del log

¿Por qué no usar console.log()?

¿Por qué no usar console.log()?

console.log()	Logging
No se puede desactivar. Tendríamos que borrar todas las líneas.	Se puede activar/desactivar
No es granular. Se imprime todo.	Se pueden usar distintos niveles y solo imprimir los que nos convenga
Se mezcla con otra información en la consola	Se distingue en la consola
No es persistente	Podemos usar archivos u otros soportes
No hay más información que la que añadimos nosotros	Se pueden añadir metadatos

Niveles

- **Fatal**: Situación catastrófica de la que no nos podemos recuperar
- **Error**: Error en el sistema que detiene una operación, pero no todo el sistema
- **Warn**: Condiciones que no son deseables, pero no son necesariamente errores
- **Info**: Mensaje informativo
- **Debug**: Información de diagnóstico
- **Trace**: Todos los posibles detalles del comportamiento de la aplicación

console

- Simple
- Similar a la versión de navegador
- Objeto global
- Documentación
- Especificación
- Varias versiones:

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

Logging

Conviene usar una librería para facilitar el logging:

- Ligero
- Dar formato
- Distribuir los logs
  - Terminal
  - Fichero
  - Base de datos
  - HTTP
  - …

Logging

Librerías de logging en Node.js:

- Winston
- Pino
- Y muchas más

Logging – Morgan

- Desarrollada por expressjs (info)
- Usada para logging automático (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
- Soporte para múltiples transportes
- Desacopla partes del proceso de logging
- Flexibilidad en el formato
- Permite definir tus propios niveles
- Instalación: `npm install winston`
- Info
- express-winston

Logging – Winston

- Ejemplo de configuración (archivo `logger.js`):

```
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

- Ejemplo de uso:

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

logger.info('Otro mensaje de info');
logger.warn("Mensaje de aviso");
logger.error("Mensaje de error");
```

BASES DE DATOS

Bases de datos

- Node.js puede trabajar con cualquier DB
- SQL:
  - My SQL
  - Oracle
  - SQLite
  - PostgreSQL
- NoSQL:
  - MongoDB
  - Cassandra
  - Regis
- Funciona mejor con NoSQL DBs (lo veremos en SW2)

Bases de datos

- Dos librerías mySQL muy parecidas
  - mysql (API)
  - mysql2 (API)
- Diferencias

Bases de datos

- Require

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

- Configuración:

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

- Conexión inicial:

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

Bases de datos

- 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) {/*…*/}
);
```

- Cerrar la conexión:

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

Bases de datos

```
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");
  });
});
```

Bases de datos

```
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
  }
);
```

Bases de datos - Sequelize

- ORM (Object–relational mapping)
- Permite trabajar con múltiples DBs:
  - Oracle
  - Postgres
  - MySQL
  - SQLite
  - …
- Instalación: `npm install sequelize`
- Luego instalar el driver, por ejemplo: `npm install sqlite3`
- Web

Sequelize – Ejemplo

- Partiendo del ejemplo Login del tema de Express
- Añadida página de registro
- Añadida gestión de usuarios con Sequelize y SQLite

PASSPORT

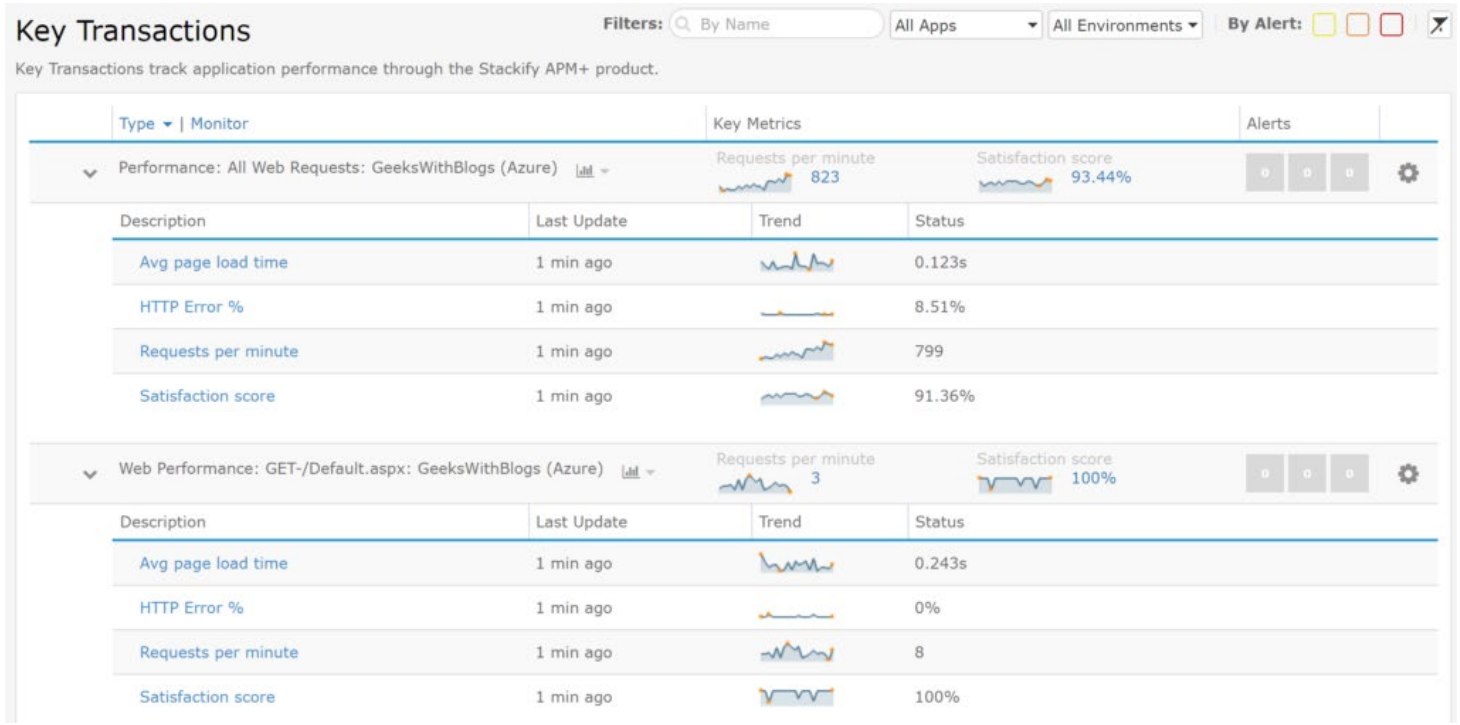
Passport

- Authentication middleware
- Fácilmente integrable con Express
- Soporta muchas estrategias:
  - username/password
  - OAuth(Facebook, Twitter, Google, …)
  - OpenID
  - …
- Info

APM

APM

- Application Performance Monitoring/Management
- Gestión de:
  - Rendimiento
  - Disponibilidad
  - Logs
  - Tráfico
  - Uso de recursos
  - Tasa de errores
  - Latencia
  - …



APM

- Varias opciones en Node.js. Algunos ejemplos:
  - App Metrics
  - Retrace
  - PM2
  - Clinic.js
  - Express-status-monitor

APM – PM2

- Process Manager 2
- Built-in load balancer
- Ofrece extras de pago
- Require instalación global:

```
npm install pm2 -g
```

- Info

APM – PM2

- Arrancar un proceso:

```
pm2 start index.js
```

```
┌────┬──────┬─────────────┬─────────┬──────┬─────┬────────┬──┬────────┬─────┬────────┬──────┬──────────┐
│ id │ name │ namespace   │ version │ mode │ pid │ uptime │ ↺ │ status │ cpu │ mem    │ user │ watching │
├────┼──────┼─────────────┼─────────┼──────┼─────┼────────┼──┼────────┼─────┼────────┼──────┼──────────┤
│ 0  │ www  │ default     │ 0.0.0   │ fork │ 23244 │ 0s   │ 0 │ online │ 0%  │ 28.0mb │ AsaP │ disabled │
└────┴──────┴─────────────┴─────────┴──────┴─────┴────────┴──┴────────┴─────┴────────┴──────┴──────────┘
```

- Reiniciar un proceso

```
pm2 restart <id>
```

- Detener un proceso

```
pm2 stop <id>
```

APM – PM2

- Información de procesos en ejecución:

```
pm2 ls
```

```
┌────┬──────┬─────────────┬─────────┬──────┬───────┬────────┬───┬────────┬──────┬────────┬──────┬──────────┐
│ id │ name │ namespace   │ version │ mode │ pid   │ uptime │ ↺ │ status │ cpu  │ mem    │ user │ watching │
├────┼──────┼─────────────┼─────────┼──────┼───────┼────────┼───┼────────┼──────┼────────┼──────┼──────────┤
│ 0  │ www  │ default     │ 0.0.0   │ fork │ 23244 │ 73s    │ 0 │ online │ 0%   │ 28.1mb │ AsaP │ disabled │
└────┴──────┴─────────────┴─────────┴──────┴───────┴────────┴───┴────────┴──────┴────────┴──────┴──────────┘
```

- Dashboard:

```
pm2 monit
```

TESTING

Testing

- Buscar bugs
- Evitar futuros errores
- Ejecución automática

Testing

- Librerías:
  - Selenium
  - Mocha
  - Jest
  - Tape

Testing – Mocha

- Instalación:

```
npm install -g mocha
```

- Añadir a package.json:

```
{
  "scripts": {
    "test": "mocha"
  }
}
```

- Crear una carpeta test en el directorio raíz

Testing – Mocha

- Escribir tests requiere de una assertion library, pej Chai

```
npm install chai
```

- Múltiples formas de assertion (API):
  - Test Driven Development (TDD):

```
assert.equal(3,3)
```

  - Behavior Driven Development (BDD):

```
expect(3).to.equal(3)
```

Testing – Ejemplo 1

```
/* test/sum.js */
let sum = require('../sum.js');
let assert = require('chai').assert;
describe('#sum()', function() {
  // add a test hook
  beforeEach(function() {
    // ...some logic before each test is run
  });
  // test a functionality
  it('should add numbers', function() {
    // add an assertion
    assert.equal(sum(1, 2, 3, 4, 5),15);
    //expect(sum(1, 2, 3, 4, 5)).to.equal(15);
  });
  // ...some more tests
})
```

Testing – Ejemplo 1

```
/* sum.js */
module.exports = function() {
  // Convert arguments object to an array
  let args = Array.prototype.slice.call(arguments);
  // Throw error if arguments contain non-finite number values
  if (!args.every(Number.isFinite)) {
    throw new TypeError('sum() expects only numbers.')
  }
  // Return the sum of the arguments
  return args.reduce(function(a, b) {
    return a + b;
  }, 0);
}
```

Testing – Ejemplo 2

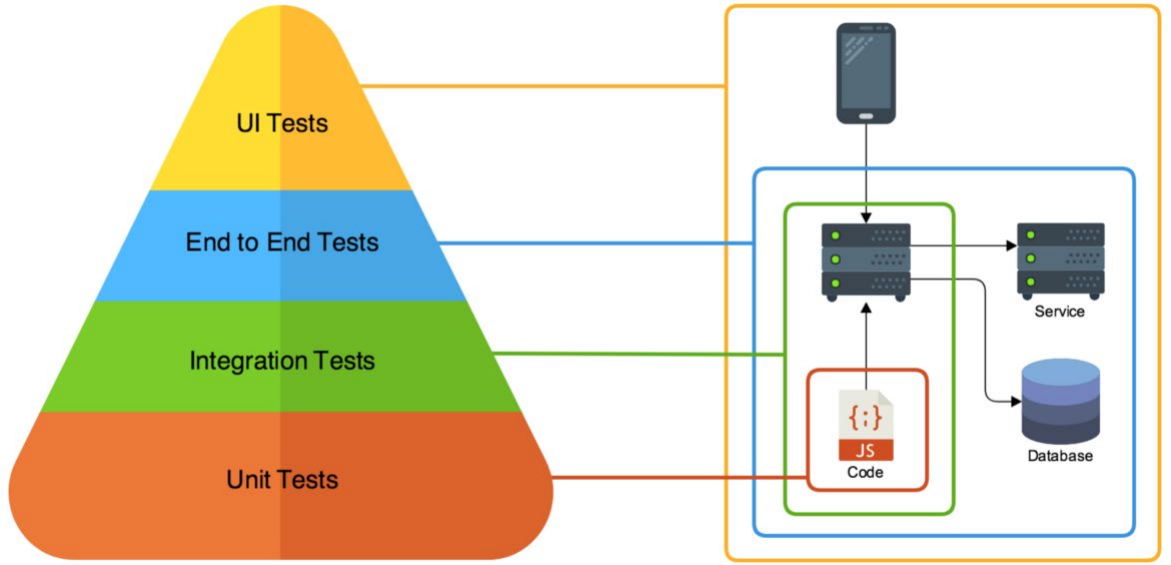
```
// test/server.js
const expect  = require("chai").expect;
const request = require("request");
describe("Color Code Converter API", function() {
  describe("RGB to Hex conversion", function() {
    const url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";
    it("returns status 200", function(done) {
      request(url, function(error, response, body) {
        expect(response.statusCode).to.equal(200);
        done();
      });
    });
    it("returns the color in hex", function(done) {
      request(url, function(error, response, body) {
        expect(body).to.equal("ffffff");
        done();
});});});});});
```

Testing – Ejemplo 2

```
// server.js
const express = require("express");
const app = express();
app.get("/rgbToHex", function(req, res) {
  const red   = parseInt(req.query.red, 10);
  const green = parseInt(req.query.green, 10);
  const blue  = parseInt(req.query.blue, 10);
  const hex = rgbToHex(red, green, blue);
  res.send(hex);
});
function rgbToHex(red, green, blue){
  const redHex   = red.toString(16);
  const greenHex = green.toString(16);
  const blueHex  = blue.toString(16);
  return pad(redHex) + pad(greenHex) + pad(blueHex);
};
function pad(hex) {  return (hex.length === 1 ? "0" + hex : hex); }
app.listen(3000);
```

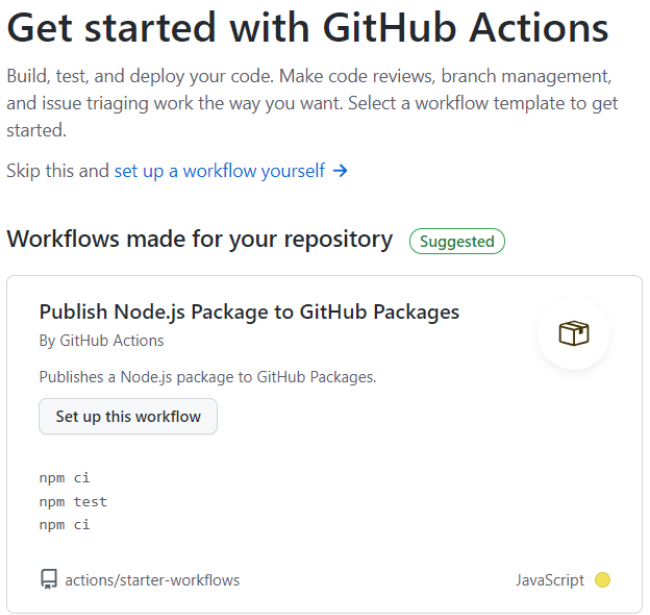
Testing – Continuous Integration

- Cada merge del código en la rama principal implica:
  - Code Build:
    - Descarga de dependencias
    - Instalación de herramientas
    - Compilación del código
    - Linting: errores de estilo
    - Generación de la versión final
  - Test:
    - Unit tests
    - Integration tests
    - End-to-end tests
    - UI tests



Testing – Continuous Integration

- Si el CI falla, se genera un informe
- Si el CI tiene éxito se publicará la versión en producción:
  - Continuous Delivery(CD)
- Los repositorios ofrecen herramientas para CI/CD:
  - GitHub
  - GitLab
  - Bitbucket



Testing

- Testing tolos:

https://developer.okta.com/blog/2020/01/27/best-nodejs-testing-tools

- Tutoriales:

https://blog.logrocket.com/a-quick-and-complete-guide-to-mocha-testing-d0e0ea09f09d/

https://semaphoreci.com/community/tutorials/getting-started-with-node-js-and-mocha

- Otros:

https://semaphoreci.com/continuous-integration

https://semaphoreci.com/blog/build-stage

https://semaphoreci.com/blog/automated-testing-cicd

REFERENCIAS

Referencias

- API
- https://nodejs.dev/
- https://www.twilio.com/blog
- Node.js Handbook by Flavio Copes