⚒️
Node JS

Tags apuntes

Tiene los mismos principios que JS:

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

Diferencias con JavaScript y programar para el navegador:

- No se trabaja con el DOM

  significa que esa herramienta, librería o tecnología no interactúa directamente con el **Document Object Model (DOM)** del navegador. El DOM es una representación estructurada de los elementos HTML de una página web, y normalmente, muchas tecnologías de frontend, como JavaScript, manipulan esta estructura para actualizar o cambiar la interfaz de usuario en respuesta a eventos.
- 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)

Versión → X.Y.Z → Semantic Versioning (SemVer): https://semver.org/

MAJOR.MINOR.PATCH
– MAJOR: Cambios incompatibles
– MINOR: Se añade funcionalidad retrocompatible
– PATCH: Corrección de errores retrocompatibles

¿Qué es LTS?
• Long Term Support
• Son versiones que tienen soporte durante mucho más tiempo

NVM

Node Version Manager

Trabajar con diferentes versiones de Node.js

```
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.
```

NPM

Node Package Manager

Gestor de paquetes de Node.js, instalado junto a node.

También se usa para js en el frontend

permite definir y ejecutar tareas

Crear archivo de configuración → package.json

```
npm init
```

Instalación de las dependencias (los módulos que tengas declarados en el archivo pero no este descargado en tu entorno)

```
npm install
npm i #Es lo mismo
```

Instalación de un paquete → automaticamente se añade al package.json

```
npm install nombre_paquete
npm install <nombre_paquete>@<versión> #Instalar versión especifica
```

Listar los paquetes instalados junto con la versión

```
npm list
npm ls
```

Actualizar dependencias

```
npm update #Todas
npm update nombre_paquete #Específica
```

Eliminar un paquete → ¿se borra solo del package.json?

```
npm uninstall nombre_paquete
```

Ver información de un paquete

```
npm view nombre_paquete
```

Ver las versiones disponibles de un paquete

```
npm view nombre_paquete versions
```

REPL

Red Evaluate Print Loop

Es el entorno de Node.js en forma de consola

`node` en la terminal para ejecutarlo

Se puede usar el tabulador para autocompletar

Comandos expeciales:

- `.help` → Muestra la ayuda
- `.exit` → sale de REPL (o pulsar ctrl D dos veces o ctrl D)

Permite ejecutar código

```
//Primer servidor index.js
const http = require('http'); //Importar librería (módulos)
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}`);
});
```

Para ejecutarlo:

```
node index.js
```

Módulos core de nodejs que no hace falta instalar:

url → trabajar y pasear urls

path → para trabajar y pasear paths

fs → trabajar con I/O en archivos

util → funcionalidades diversas

Explícación código ejemplo:

`http.createServer([options], [requestListener])`;

createServer es un método para crear un servidor HTTP

requestListener es una función ejecutada con cada petición al servidor (request)

req → objeto `http.IncommingMessage` con información de la peticion

res → objeto `http.ServerResponse` con información de la respuesta

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

listen es un metodo que arranca el servidor escuchando en un puerto

host → por defecto unspecified ipv6 → 0.0.0.0 en ipv4

backlog → númerous máximo de conexiones pendientes

callback → se ejecuta cuando se termina de arrancar el servidor

Pregunta importante

```
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? -> imprime x2
```

Porque los navegadores hacen una primera solicitud para pedir el contenido de la página y una segunda para obtener el favicon xd

Variables de entorno

forma de almacenar datos de configuración fuera del código fuente

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, si hay una definida, la definida, sino el 3000

```
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

Se pueden cargar de un archivo .env con el módulo dotenv (necesario instalarlo)

`npm install dotenv`

Creamos archivo .env con las variables

```
PORT=3000
NODE_ENV=development
DB_USER=root
DB_PASS=secretpassword
```

Entonces, en el código cargamos las variables:

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

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

Argumentos

Se pueden añadir argumentos al ejecutar el programa

`node index.js joe smith`

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

Accesible con process.argv → devuelve todo con el siguiente convenio UNIX

- Posición 0: full path del comando node
- Posición 1: full path del archivo
- Posiciones 2 en adelante: los argumentos

Si queremos usar clave=valor hay que parsearlo

Los argumentos se pasarían con `--nombre=valor`

También se pueden usar opciones en unix como `-opción valor` (opcional es una letra)

Para esto habrá que instalar minimist `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

```
// no necesita ser cargado
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.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

Módulos con acceso a información de bajo nivel sobre el sistema operativo

```
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.
```

Event Loop

es un componente clave del entorno de ejecución de JavaScript (incluido Node.js) que se encarga de gestionar la ejecución de código, la realización de operaciones asíncronas y la gestión de eventos. Es el mecanismo que permite a Node.js ser no bloqueante y altamente eficiente.

Basado en:

- un único thread
- Se repite (loop)
- Cada iteración es un tick
- Gestiona la ejecución de eventos
- Una cola FIFO de callbacks
- Gestionado internamente por la librería libuv

Ejemplo:

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

DON'T BLOCK THE 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++) { //Tarea super intensiva, bloquea el Event loop
        //Mientrasa estas aqui el servidor no puede manejar otras peticiones
        array.push(crypto.randomBytes(1000).toString('hex'));
    }
    console.log(Date.now()-time);
});
server.listen(port, () => {console.log(`Server running at port ${port}`);});
```

- No debeos bloquear el event loop
- Cada tic debe ser corto
- El trabajo asociado a cada cliente tiene que ser breve
- Intentar dividir ñas tareas más imtensivas

Ficheros

PATH

Modulo con utilidades para trabajar con ficheros y rutas

Hay que importarlo → `const path = require("path")`

```
path.dirname(path) // Devuelve el nombre del directorio
path.sep //Separador del SO  \ en windows y / en UNIX
path.normalize(path) //Elimina separadores duplicados y procesa .. y .
path.join([..paths]) //Une los paths usando el separador y lo normaliza
path.resolve([..paths]) // Procesa de derecha a izquierda y genera ruta absoluta
```

```
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, en el code de node.js

Los métodos son async por defecto → si añades Sync al final se hace sync

`fs.access() → fs.accessSync()`

```
fs.access(): //check if the file exists and 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

Sintaxis → 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

OJO

r+ sobreescribe → Ejemplo:

```
Hola Mundo
```

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

fs.open('texto.txt', 'r+', (err, fd) => {
  if (err) throw err;

  const buffer = Buffer.from('Adiós');
  //buffer almacena los datos en binario en bloques fijps de memoria
  //Si hacemos console.log(buffer):
  //<Buffer 48 6f 6c 61 20 4d 75 6e 64 6f>
  fs.write(fd, buffer, 0, buffer.length, null, (err) => {
  //fs.write(fd, buffer, offset, length, position, callback)
    if (err) throw err;
    console.log('Escritura completada');
    fs.close(fd, () => {});
  });
});
```

```
Adiós Mundo
```

No hace falta escribir con un buffer xd

```
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');
    });
```

PROMESAS `fs.promises`

- Funciones equivalentes que trabajan con promesas
- recordemos → trabajar de forma asincrona sin usar callbacks → evitar callback hell

```
fs.promises.open(path, flags[, mode])
fs.promises.readFile(path[, options])
fs.promises.writeFile(file, data[, options])
fs.promises.readFile("data.csv", "utf8")
fs.promises.then(processFileText)
fs.promises.catch(handleReadError)
```

Alternativamente, usando async / await (recordemos que lo necesitamos patra las promesas, sino .then() y .catch())

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

stat `fs.stat`

metadatos lol

```
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
```

Peticiones HTTP

podemos usar:

- GET
- POST
- PUT
- DELETE
- ...

Ejemplo petición:

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

Otro ejemplo:

```
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);
});
```

Ejemplo ahora con POST:

SERVIDOR QUE ENVIA LA INFO CON 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}`);});
```

CLIENTE QUE RECIBE LA INFO CON 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();
```

EXPRESS

Es un framework for web developing, permite:

- definir métodos HTTP
- definir las rutas
- trabajar con middleware

Unopinionated → no impone un modelo estricto para tu aplicación
– No define el template engine
– No define la base de datos

Instalar → `npm install express`

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

//definir ruta para manejar solicitudes HTTP GET
app.get('/', (req, res) => {
    res.send('Hello World!');
});

//abrimos servidor en el puerto especificado
app.listen(port, () => {
    console.log(`Example app listening at http://localhost:${port}`);
});
```

Para definir la ruta podemos usar cuanquier otro método → app.post, app.delete...

Middleware

Codigo que ejecutamos en medio de otra ejecución. Necesario invocar el método `next()` para continuar la cadena.

Es una función que tiene acceso al objeto de solicitud (req), el objeto de respuesta (res) y a la siguiente función en la pila de middleware.

```
var myLogger = function (req, res, next) {
console.log('LOGGED');
next(); // pasa al siguiente middleware o ruta
}
app.use(myLogger);
```

Rutas

Express te permite definir rutas para diferentes métodos HTTP (GET, POST, PUT, DELETE, etc.) y diferentes URLs.

Por ejemplo:

```
app.get('/saludo', (req, res) => {
  res.send('¡Hola, cómo estás?');
});

app.post('/datos', (req, res) => {
  res.send('¡Datos recibidos!');
});

app.put('/actualizar', (req, res) => {
  res.send('¡Datos actualizados!');
});

app.delete('/eliminar', (req, res) => {
  res.send('¡Datos eliminados!');
});
```

Templates

Son motores de plantillas, se utilizan para generar páginas HTML dinámicas. que el servidor envía al cliente. Esto es útil cuando necesitas insertar datos dinámicos en vistas, como el nombre de un usuario o info de una base de datos.

En tiempo de ejecución se reemplazan las variables por sus valores

Hay varios engines para esto:

PUG

Antiguamente JADE, basado en Haml

En nuestro Backend:

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

// Configurar el motor de plantillas
app.set('view engine', 'pug'); // Cambia 'pug' por 'ejs' o el motor que elijas
app.set('views', './views');   // Carpeta donde estarán las plantillas

// Ruta que usa la plantilla
app.get('/', (req, res) => {
  res.render('index', { title: 'Hola, Express!', message: '¡Bienvenido al mundo de los templates!' });
});

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

Y crearíamos nuestro archivo idex.pug para poner el contenido (No HTML)

```
html
  head
    title= title
  body
    h1= message
    p Esto es un ejemplo de una página generada dinámicamente.
```

EJS

Embedded JavaScript templates, USA HTML

simplemente añadir las etiquetas

`<% … %>`
`<%= … %>`

para el flujo de control y variables

en un archivo index.ejs

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <h1><%= message %></h1>
  <p>Esto es un ejemplo de una página generada dinámicamente.</p>
</body>
</html>
```

Express-generator

`npx express-generator -v ejs express_test`

INICIAR PROYECTO

`express nombre-proyecto`

Crea la siguiente estructura de archivos:

- 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
```

1. app.js**:** Archivo principal que configura la aplicación Express.
2. bin/www**:** Script para iniciar el servidor.
3. public/**:** Carpeta para recursos estáticos como CSS, JavaScript e imágenes.
4. routes/**:** Carpeta para definir rutas de la aplicación.
5. views/**:** Carpeta para las vistas, si se usa un motor de plantillas.
6. package.json**:** Archivo con las dependencias y scripts del proyecto.

con las siguéntes dependencias en el pacjage.json

```
{
    "name": "express-test",
    "version": "0.0.0",
    "private": true,
    "scripts": {ha
        "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"
    }
}
```

Un ejemplo clásico de /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) { /*..*/ }
```