Node.js

- Open-source
- cross-platform
- mas de 1 M de paquetes open source
- Mismo principio 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, cookies
    - No están las variables del navegador como document o window
    - No tiene restricciones de acceso a ficheros
    - No tienes problemas de versiones del navegador
        - Incompatibilidad con versiones antiguas de JS

LTS - long term support

- **pares duran mas**
- impares menos

Qué significan los números de la version X.Y.Z?

- Estándar de numeración de versiones para software
- Semantic Versioning (SemVer)
- MAJOR.MINOR.PATCH
    - Major
        - cambios incompatibles
    - Minor
        - Se añade funcionalidad retrocompatible
    - Patch
        - Correción de errores retrocompatibles

NVM

- Node Version Manager
- Trabajar con diferentes versiones de Node.js

npm

- Node Package Manager
- Gestor de paquetes de Node.js
- Instalado junto con Node.js
- También se usa para JS en el frontend

npm init
- Crear el archivo de configuración package.json

npm install y npm i
- Instalación de las dependencias
- Requiere archivo de configuración package.json

npm install nombre_paquete
npm install <nombre_paquete>@<version>
npm list o npm ls
npm update
npm update nombre_paquete
npm uninstall nombre_paquete
npm view nombre_paquete

REPL
- Read Evaluate Print Loop
- Entorno de node.js en forma de consola
- Se puede usar el tabilador para autocompletar
- comandos especiales
  - .help
  - .exit

npx
- Permite ejecutar código
- No hace falta tener instalado el paquete
- npx cowsay "Hello"

Ejecutar usando node = node index.js

Require sirve para importar modulos

Variables de entorno
- Accesibles con el módulo process
  - Disponibles sin necesidad de requiere

    - Podemos definirlas el ejecutar el programa
        - USER_ID = 239482 USER_KEY=foobar node app.js

Argumentos

- Se pueden añadir al ejecutar el programa
- node index.js joe smith

JSON

- JavaScript Object Notation
- Formato ligero para el intercambio de datos
- Soporta objetos, arrays y valores
- No permiten comentarios

```
{
"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
}
```

- Objeto
    - Set desordenado de pares clave/ valor:
- Rodeado por llaves { }
- La clave es un string y tiene que tener comillas dobles
        - "clave" : "valor"

- la clave deberia usar nomenclatura lowe camel case
- Los datos se separan por comas

No se pone coma en la ultima clave valor

Array

- Colección ordenada de valores
- Rodeado por corchetes [ ]

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

- No se garantiza el orden del contenido de los objetos
- Los elementos de un array si que estan ordenados

Convertir un objeto JSON a texto

- let text = JSON.stringify(obj)

Convertir un texto en formato JSON a texto

- let obj = JSON.parse(text);

package.json

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

```
{
"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 Sánchez",
"license": "ISC",
"dependencies": {
    "minimist": "^1.2.5"
  }
}
```

- "name": "prueba_node"
  - Nombre de paquete
  - hasta 214 caracteres
  - Solo minusculas
- "version": "1.0.0"
  - Version del paqeute
  - Formato semver
- "description": "Ejercicio de prueba de Node.js"
  - Descripción del paquete
- "main": "index.js"
  - Punto de entrada del paquete
  - Donde se buscará para exportarlo como módulo
  - Si no existe el valor por defeco es index.js
  - Se ejecuta el archivo con node .

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

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

"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

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

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

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

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

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

- Dependencias solo para el desarrollo
- No necesarias en producción
- Se añaden automáticamente al hacer
  - npm install --save-dev nombre_paquete

Hay muchas mas opciones

- "bugs": "https://github.com/whatever/package/issues"
- "homepage": "https://whatever.com/package"
- "license": "MIT"
- "repository":"github:whatever/testing"
- "private": true

Hay un problema con package.json

- El proyecto no es reproducible al 100%
- Alguien podría usar la version 2.5.2 otro la 2.5.3 y otro la 2.6.0
  - Puede dar lugar a inconssitencias en el proyecto
- Alternativa
  - Subir la carpeta node_modules al repositorio

        - No recomendable porque puede llegar a ocupar mucho espacio

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
    - npm update
- npm list

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

Peticiones HTTP

- GET
- POST
- PUT
- DELETE

Ficheros

- fs
    - File system
    - Acceso e interracción con el sistema de ficheros
    - core de node.js
    - los métodos son async por defecto
- fs - open
    - fs.open

Express

- Web framework
- Permite definir métodos http
- permite definir las rutas
- unopinionated

middleware

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

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

template engine

- 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
- ejemplos
  - Pug
  - EJS
    - Embedded Javascript templates

Express - express-generator

- npx express-generator
- Hay que hacer npm install
- carpeta route define las rutas
- carpeta views define los templates

Sockets.IO

- Comunicación servidor y cliente
- Bidireccional
- WebSockets
- Reconexión en caso de fallo
- escalable
- Enviar y recibir eventos
- Enviar mensaje
  - socket.emit(msj)
- Enviar mensaje a todos los clientes conectados
  - io.emit(msj)
- hacer broadcast
  - Enviar a todos menos a uno mismo
  - socket.broadcast.emit(msj)

DEBUG=* node index.js

Logging

- Logging es guardar información relevante de nuestro programa para su posterior análisis
- Objetivos
  - comprobar que el programa funciona correctamente
  - solucionar bugs
  - análisis de parámetros
- Que 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
  - expeciones
- Que información NO queremos loggear
  - Información sensible
    - Información persona (DNI, Nombres y apellidos, email)
    - Información médica
    - Información financiera
    - Contraseñas
    - Direcciones IP
- Cuidado con las URL porque podrian tener informacion sensible
  - /user/<email>
- Cuidado con el tamaño del 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 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
  - informacion de diagnóstico
- Trace
  - todos los posibles detalles del comportamiento de la aplicación

Console

- Simple
- Similar a las versiones del navegador
- objeto global
- Varias versiones
  - Console.error()
  - console.warn()
  - console.log()
  - console.debug

Conviene usar una libreria para facilitar el logging

- ligero
- dar formato
- distribuir los logs

    - terminal
    - fichero
    - base de datos
    - http

Librerias de logging en Node.js

- Winston
    - Simple
    - soporte para multiples transportes
    - Desacopla partes del proceso de logging
    - flexibilidad en el formato
    - permite definir tus propios niveles
- Pino

Base de datos

- node.js puede trabajar con cualquier DB
- SQL
    - MySQL
    - Oracle
    - SQLite
    - PostgreSQL
- NoSQL
    - MongoDB
    - Cassandra
    - Regis
- funciona mejor con NoSQL

Passport

- Authentication middleware
- Fácilmente integrable con Expres
- Soporta muchas estrategias:
    - username/password
    - 0Auth
    - OpenID

APM

- Applicaiton Performance Monitoring/Management
- Gestión de
    - rendimiento

    - disponibilidad
    - logs
    - tráfico
    - uso de recursos
    - tasa de errores
    - latencia
- Varias opciones en Node.js
    - app metrics
    - retrace
    - PM2
    - clinic
    - Express-satus-monitor

Testing

- Buscar bugs
- evitar futuros errores
- ejecución automática
- Librerías
    - selenium
    - mocha
    - jest
    - tape
- Continuous integration
    - cada merge del codigo en la rama main implica
        - code build
            - descarga de dependencias
            - instalación de herramientas
            - compilación de código
            - linting errores de estilo
            - Generacion de la version final
        - test
            - unit tests
            - integration tests
            - ent to end tests
            - UI tests
    - Si el CI falla, se genera un informa
    - Si el CI tiene éxito se publicara la version en producción