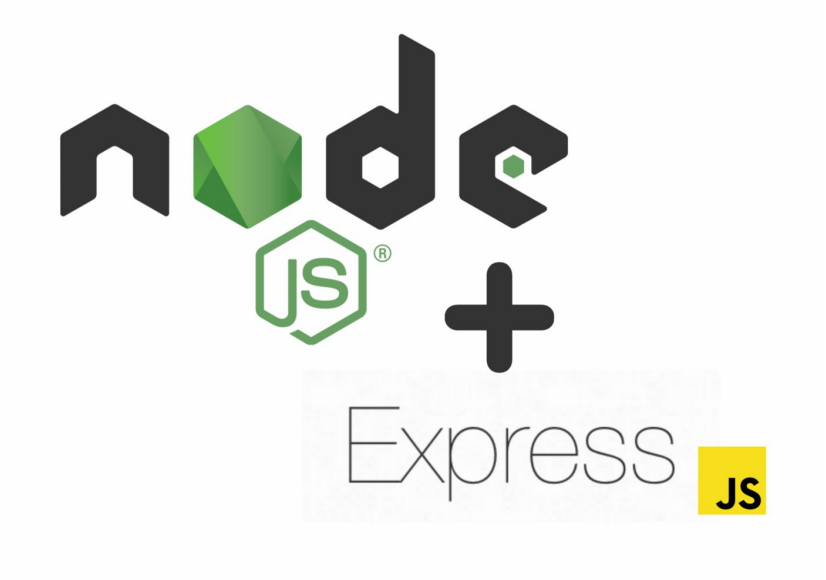
releevant.



linkedin.com/in/maortizolid

maortizolid@gmail.com

releevant.

Node Package Manager (NPM)

Node Package Manager (NPM) es el administrador de paquetes para Node.js. Su función principal es facilitar la instalación, gestión y actualización de paquetes de software de Node.js.

Algunas funciones clave de NPM incluyen:

- **Instalación de Paquetes**: Permite instalar paquetes de Node.js de manera sencilla. Por ejemplo, puedes ejecutar npm install nombre-del-paquete para descargar e instalar un paquete específico.
- **Gestión de Dependencias**: Administra las dependencias de tu proyecto, asegurando que todas las bibliotecas necesarias estén disponibles y actualizadas.
- **Versionado**: Permite especificar versiones de paquetes para garantizar la consistencia en entornos de desarrollo y producción.
- **Scripts**: Permite definir scripts personalizados en el archivo package.json, lo que facilita la ejecución de tareas como pruebas, compilaciones y más.
- **Publicación de Paquetes**: Facilita la publicación de tus propios paquetes para que otros desarrolladores puedan utilizarlos.

En resumen, NPM simplifica la administración de dependencias y la distribución de software en el ecosistema de Node.js.

releevant.

Línea de comandos de NPM

La línea de comandos de npm es la interfaz de línea de comandos utilizada con el administrador de paquetes npm en Node.js. Permite instalar, gestionar y publicar paquetes de software.

En su web podemos buscar cualquier dependencia, ofreciendo diversa información como la documentación, versiones, descargas semanales, etc

Alternativas populares como Yarn y PNPM también ofrecen funcionalidades similares, pero con enfoques ligeramente diferentes y mejoras de rendimiento. Estas herramientas alternativas buscan abordar ciertos problemas, como la velocidad de instalación y la gestión eficiente de dependencias en proyectos Node.js.

releevant.

Inicializar un proyecto con npm init.

- El comando `init` se utiliza para inicializar un proyecto. Cuando ejecutas este comando, crea un archivo **`package.json`**.
- Al ejecutar `npm init`, se te pedirá que proporciones cierta información sobre el proyecto que estás inicializando. Esta información incluye el nombre del proyecto, el tipo de licencia, la versión, etc.
- Para saltarte el proceso de proporcionar la información tú mismo, puedes simplemente ejecutar el comando `npm init -y`.

El archivo `package.json`

- Es el archivo que se crea cuando se inicializa un proyecto o se instala una dependencia por primera vez.
- Se ubica en la carpeta raíz del proyecto, donde coloca toda la **información** que se conoce sobre el mismo.
- Es es un simple fichero de texto, en **formato JSON** que incorpora a través de varios campos información muy variada.

releevant.

Vamos a ver estos conceptos inicializando un proyecto e instalando nuestra primera dependencia. Para ello vamos a copiar el último ejercicio `7.ls-advance` en otro archivo (por ejemplo `1.ls-advance2`).

Ejecutamos `npm init`, o si queremos saltarnos las preguntas, `npm init -y`

Se creará el archivo package.json y tendrá un formato como este:

```
{
  "name": "releevant",
  "version": "1.0.0",
  "description": "Ejemplo inicializando un proyecto",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Miguel Angel Ortiz",
  "license": "ISC"
}
```

releevant.

Dependencias

En Node.js, una dependencia se refiere a un módulo o paquete de software externo que tu aplicación utiliza para funcionar correctamente.

Estas dependencias son especificadas en el archivo `package.json` de tu proyecto y son administradas por el administrador de paquetes npm.

Las dependencias pueden incluir bibliotecas, frameworks o cualquier otro componente necesario para que tu aplicación se ejecute correctamente.

El uso de dependencias facilita la reutilización de código y la gestión eficiente de recursos en el desarrollo de aplicaciones Node.js.

releevant.

Tipos de dependencias

- Las **dependencias de desarrollo** son aquellos paquetes que necesitamos en un proyecto mientras estamos desarrollando, pero una vez tenemos el código generado del proyecto, no vuelven a hacer falta. Los paquetes instalados con el flag `--save-dev` o `-D` se instalan en esta modalidad, guardándose en la sección `devDependences` del fichero `package.json`.

- Las **dependencias de producción** son aquellos paquetes que necesitamos tener en la web final generada, como librerías Javascript necesarias para su funcionamiento o paquetes similares. Los paquetes instalados con el flag `--save-prod`, `-P` o directamente sin ningún flag se instalan en esta modalidad, guardándose en la sección `dependences` del fichero `package.json`.

- Para pasar una **dependencia de desarrollo a producción** no es necesario desinstalar y volver a instalar el paquete. Simplemente, escribimos el comando de instalación de la modalidad a la que queremos pasar: `npm install --save-prod`. Internamente, `npm` se encargará de modificar el `package.json` y cambiar el paquete ya instalado de la sección `devDependences` a `dependences`.

releevant.

node_modules

Cuando instalas una dependencia por primera vez en un proyecto Node.js utilizando npm, se crea la carpeta `node_modules/` en el directorio de tu proyecto.

Esta carpeta almacena todas las dependencias y módulos necesarios para tu aplicación.

Cada paquete instalado tiene su propia carpeta dentro de `node_modules/`, lo que ayuda a organizar y gestionar las dependencias de manera estructurada.

La existencia de esta carpeta permite a Node.js acceder y cargar los módulos necesarios durante la ejecución de tu aplicación.

releevant.

El archivo package-lock.json.

- Es un archivo **generado automáticamente** cuando se **instalan paquetes o dependencias** en el proyecto. Su finalidad es mantener un historial de los paquetes instalados y optimizar la forma en que se generan las dependencias del proyecto y los contenidos de la carpeta `node_modules/`.

- Este archivo debe conservarse e incluso versionearse para añadirlo al repositorio de control de versiones, puesto que es algo favorable para el trabajo con `npm`. Es por ello que debe añadirse al fichero `.gitignore`.

releevant.

Instalamos la dependencia picocolors

`npm install picocolors` o `npm i picocolors`

Con esta dependencia vamos a darle color al formato de salida de nuestro ejercicio. Por defecto, si no indicamos nada, se instalará como **dependencia de producción.**

Se creará la carpeta `node_modules/` y nuestro archivo `package.json` quedará de la siguiente forma:

```
{
  "name": "releevant",
  "version": "1.0.0",
  "description": "Ejemplo inicializando un proyecto",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Miguel Angel Ortiz",
  "license": "ISC",
  "dependencies": {
    "picocolors": "^1.0.0"
  }
}
```

releevant.

Actualizaciones automáticas:

El símbolo ^ antes de la versión de una dependencia indica que se permiten actualizaciones automáticas para ciertas versiones del paquete.

Especifica que la dependencia puede actualizarse a nuevas versiones que no incluyan cambios incompatibles con la versión especificada, de acuerdo con las reglas de versionado semántico (SemVer).

Por ejemplo, si tienes "dependencia": "^1.2.3", significa que se permiten actualizaciones automáticas para versiones compatibles hasta, pero no incluyendo, la versión 2.0.0.

Esto ayuda a mantener tu proyecto actualizado con las últimas correcciones de errores y mejoras sin introducir cambios que podrían romper la compatibilidad.

releevant.

Desventajas de permitir actualizaciones automáticas:

- *Inconsistencia* : Las actualizaciones automáticas pueden introducir cambios que afectan la compatibilidad con tu código existente, lo que podría romper tu aplicación.

- *Control* : Perder un cierto grado de control sobre las actualizaciones puede resultar problemático si necesitas mantener versiones específicas de dependencias por razones de compatibilidad.

Se recomienda instalar la dependencia con la última versión estable. Podemos hacerlo a la hora de instalar la dependencia: `npm i picocolors -E`

También podemos llevar un control de las versiones con la extensión para Visual Studio Code, Version Lens

releevant.

Ahora modificamos el código del ejercicio `1.ls-advance2` para utilizar la dependencia picocolors.

```
const fs = require('node:fs/promises')
const path = require('node:path')
const pc = require('picocolors')

const folder = process.argv[2] ?? '.'

async function ls (directory) {
  let files
  try {
    files = await fs.readdir(folder)
  } catch {
    console.error(pc.red`❌ error al leer el directorio ${folder}`)
    process.exit(1)
  }

  const filesPromises = files.map(async file => {
    const filePath = path.join(folder, file)
    let stats
```

releevant.

```
    try {
      stats = await fs.stat(filePath) // información del archivo
    } catch (err) {
      console.error(`error al leer el archivo ${filePath}`, err)
      process.exit(1)
    }

    const isDirectory = stats.isDirectory()
    const fileType = isDirectory ? 'd' : 'f'
    const fileSize = stats.size.toString()
    const fileModified = stats.mtime.toLocaleString()

    return `${pc.magenta(fileType)} ${pc.cyan(file.padEnd(20))} ${pc.green(fileSize.padStart(10))} ${pc.yellow(fileModified)}`
  })

  const filesInfo = await Promise.all(filesPromises)

  filesInfo.forEach(file => console.log(file))
}

ls(folder)
// node 1.ls-advance2.js ../01_modulos_nodejs
```

releevant.

Dependencias de desarrollo

Vamos a ver como se instala una dependencia de desarrollo. Para ello vamos a instalar `StandardJS`. aprovechando para ver cómo funciona un linter para JavaScript.

StandardJS es un linter para JavaScript. Funciona revisando tu código en busca de posibles problemas, errores de estilo y violaciones de las reglas de codificación establecidas por la convención de StandardJS. El propósito principal de un linter como *StandardJS* es mejorar la consistencia y calidad del código fuente, así como detectar posibles errores antes de la ejecución de la aplicación.

Cuando instalas *StandardJS* como una dependencia de desarrollo en tu proyecto y ejecutas el linter, proporciona información sobre cualquier problema que encuentre en tu código según las reglas predefinidas.

Los linters como *StandardJS* son herramientas valiosas para mantener un código limpio, fácil de leer y libre de errores en proyectos JavaScript.

releevant.

Para indicar a dependencia que es de desarrollo, escribimos lo siguiente:
`npm i standard -D` o `npm i standard --dev`

Si queremos instalarla solo en el proyecto: `npm i standard --save--dev`

Nuestro package.json quedaría:

```
{
  "name": "releevant",
  "version": "1.0.0",
  "description": "Ejemplo inicializando un proyecto",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Miguel Angel Ortiz",
  "license": "ISC",
  "dependencies": {
    "picocolors": "^1.0.0"
  }
  "devDependencies": {
    "standard": "^17.1.0"
  }
}
```

releevant.

Ahora añadimos la configuración de ESLint en el package.json. Al agregar esta configuración, estamos integrando ESLint en el proyecto y configurandolo para que utilice las reglas de estilo predefinidas por el paquete "standard".

Esto te permite mantener un estilo de código consistente y seguir las convenciones de codificación establecidas por StandardJS en tu proyecto.

```
{
  "name": "releevant",
  "version": "1.0.0",
  "description": "Ejemplo inicializando un proyecto",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Miguel Angel Ortiz",
  "license": "ISC",
  "dependencies": {
    "picocolors": "^1.0.0"
  }
  "devDependencies": {
    "standard": "^17.1.0"
  },
  "eslintConfig": {
    "extends": "standard"
  }
}
```

releevant.

Necesitaremos instalar la extensión ESLint en el Visual Code Studio y hacer un par de configuraciones para que al guardar los cambios en el fichero se apliquen las reglas.

En Visual Code Studio accedemos a settings.json:

- Ir al Menú File (Archivo),
- luego a Preferences (Preferencias),
- y seleccionar Settings (Configuraciones),
- ahí sólo da clic en el botón Open Settings (JSON) que está en la esquina superior derecha

Una vez en la configuración del archivo verificamos/añadimos estas opciones en la sección "`[javascript]`":

```
"[javascript]":
 {
        "editor.defaultFormatter": "vscode.typescript-language-features",
        "editor.formatOnSave": true
 },
"editor.codeActionsOnSave": {
        "source.fixAll.eslint": true
 },
```