



## Node js 2. Npm y el archivo Package.json

Nombre		Curso	
Apellidos		Fecha	

### EL ARCHIVO PACKAGE.JSON

### INSTALAR PAQUETES CON NPM EL ARCHIVO

### PACKAGE.JSON El índice con la información de nuestro proyecto

Al crear un nuevo proyecto con **npm init**, se lanzará un asistente que tras algunas preguntas, crea un archivo llamado **package.json** en la carpeta raíz del proyecto, donde coloca toda la información que se conoce sobre el mismo. Este archivo es un simple fichero de texto, en formato JSON que incorpora a través de varios campos información muy variada.

Este fichero tiene una estructura definida muy concreta, y a través de sus campos se puede guardar y recuperar información muy útil.

### Campos del package.json

Si creamos un nuevo proyecto con los valores por defecto (**omitiendo el asistente**) con el comando **npm init -y**, se generará el siguiente archivo **package.json**, el cuál podemos abrir con nuestro editor de texto preferido para modificarlo:

```
{
  "name": "frontend-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Por defecto, obtendrá el nombre de la carpeta del proyecto (**recuerda que es importante utilizar minúsculas y guiones en lugar de espacio**) y nos colocará la versión **1.0.0** por defecto.

Recuerda que el campo **name** no debe empezar por punto, guión bajo ni debe tener el mismo nombre que otro paquete que uses como dependencia. Una buena recomendación es que se utilice un prefijo/scope como namespace, del tipo **@author/nombre-paquete**.

Veamos algunos de los campos más habituales de **package.json** que debemos conocer:

Campo	Descripción
<b>name</b>	Nombre del proyecto, librería o paquete. Se recomienda que coincida con el repositorio.
<b>version</b>	Versión del paquete. Generalmente se utiliza <b>semver</b> (lo veremos más adelante).
<b>description</b>	Descripción breve del paquete o proyecto.
<b>type</b>	Tipo de <u>sistema de módulos</u> a utilizar. Con <b>module</b> usa ESM, en caso contrario usa CommonJS.
<b>main</b>	<b>Punto de entrada</b> del proyecto. Suele ser <b>index.js</b> (node) o <b>index.html</b> (browser).
<b>module</b>	Idem al anterior, pero respecto a <u>ES Modules</u> en lugar de <b>CommonJS</b> .
<b>scripts</b>	Colección de scripts del proyecto (lo veremos más adelante).
<b>keywords</b>	de con palabras clave relacionadas con el proyecto. Util en búsquedas.
<b>author</b>	Nombre del autor del paquete o un con <b>name</b> , <b>email</b> y/o <b>url</b> .
<b>license</b>	Tipo de licencia del paquete o proyecto. Por defecto, ISC.
<b>dependencies</b>	Colección de paquetes para producción y la versión instalada.
<b>devDependencies</b>	Colección de paquetes para desarrollo y la versión instalada.
<b>homepage</b>	URL de la página principal del paquete.
<b>repository</b>	URL del repositorio. Se debe indicar <b>type</b> (git, svn...) y <b>url</b> (ruta).
<b>bugs</b>	Objeto con campo <b>url</b> con la URL de la página de issues del proyecto.

## Sintaxis de npm

npm <command>

## Usage:

npm install install all the dependencies in your project  
npm install <foo> add the <foo> dependency to your project  
npm test run this project's tests  
npm run <foo> run the script named <foo>  
npm <command> -h quick help on <command>  
npm -l display usage info for all commands  
npm help <term> search for help on <term> (in a browser)  
npm help npm more involved overview (in a browser)  
All commands:

access, adduser, audit, bin, bugs, cache, ci, completion,  
config, dedupe, deprecate, diff, dist-tag, docs, doctor,  
edit, exec, explain, explore, find-dupes, fund, get, help,  
hook, init, install, install-ci-test, install-test, link,  
ll, login, logout, ls, org, outdated, owner, pack, ping,  
pkg, prefix, profile, prune, publish, rebuild, repo,  
restart, root, run-script, search, set, set-script,  
shrinkwrap, star, stars, start, stop, team, test, token,  
uninstall, unpublish, unstar, update, version, view, whoami

```
# instalamos los paquetes howler y react  
$ npm install howler  
$ npm install react
```

Si nuestro paquete (**o alguna dependencia**) tiene definidos estos últimos 3 campos en su **package.json** podemos utilizar los siguientes comandos, incluso si no tenemos instalada la dependencia en nuestro proyecto:

```
# Accede a la homepage de documentación del paquete "howler"  
$ npm docs howler  
$ npm home howler  
  
# Accede al repositorio del paquete "howler"  
$ npm repo howler  
  
# Accede a la página de issues del paquete "howler"
```

```
$ npm issues howler
```

```
$ npm bugs howler
```

## Dependencias del proyecto

En nuestro fichero **package.json** recién creado probablemente no existan los campos **dependencies** y **devDependencies**, pero si estamos ante un proyecto en el que ya han sido instalados paquetes, si existirán. El apartado **dependencies** contiene el nombre y versión de los paquetes de producción, mientras que el campo **devDependencies** contiene el nombre y versión de los paquetes de desarrollo:

```
# Mostramos los paquetes de producción del package.json
```

```
$ cat package.json | jq .dependencies
```

```
{
  "react": "^16.13.1",
  "howler": "^2.2.0"
}
```

Lo realmente interesante de estos campos es que funcionan a modo de índice o guía de paquetes necesarios para el proyecto, de modo que sólo con el fichero **package.json** en una carpeta vacía, podemos indicarle a **npm** que descargue todas las dependencias necesarias para el proyecto en la carpeta **node\_modules/** escribiendo lo siguiente:

```
# Pedimos a NPM que descargue las dependencias necesarias en node_modules/
```

```
$ npm install
```

Probablemente, este proceso tardará un rato, dependiendo del número de dependencias que sea necesario instalar, de su tamaño, de la conexión a Internet de la que dispongamos e incluso de si disponemos de un disco duro lento o un disco SSD rápido. Una vez instalados los paquetes, tendremos nuestro proyecto listo para realizar bare imports, es decir, importaciones sin ruta (**solo con el nombre del paquete**), que **node** buscará en la carpeta **node\_modules/**:

```
// Importamos la librería Howler de la carpeta node_modules/
```

```
import { Howler, Howl } from "howler";

const audio = new Howl({
  src: ["/assets/audio.mp3"]
});
```

Nota: Si quieres más información sobre las importaciones nativas de Javascript, echa un vistazo a las sentencias **import** y **export** en el artículo [Módulos ECMAScript](#).

## El archivo package-lock.json

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 versionarse para añadirlo al repositorio de control de versiones, puesto que es algo favorable para el trabajo con **npm**. Es por ello que no debe añadirse al fichero **.gitignore**.

# Preguntas de NODE Js y npm

1. Instala node JS
2. Ejecuta npm npm init -y,

```
PS C:\Users\2DAW\Documents\DWEC\Tema 8\ejemploNode> npm init -y
Wrote to C:\Users\2DAW\Documents\DWEC\Tema 8\ejemploNode\package.json:

{
  "name": "ejemplonode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs"
}
```

3. Ejecuta npm install howler

```
PS C:\Users\2DAW\Documents\DWEC\Tema 8\ejemploNode> npm install howler
added 1 package, and audited 2 packages in 1s

found 0 vulnerabilities
```

4. Ejecuta npm install react

```
PS C:\Users\2DAW\Documents\DNEC\Tema 8\ejemploNode> npm install react
added 1 package, and audited 3 packages in 1s
found 0 vulnerabilities
```

##### 5. Copia a qui tu package.json

```
{
  "name": "ejemplonode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "howler": "^2.2.4",
    "react": "^19.2.4"
  }
}
```

##### 6. Rellena que son cada uno de estos campos

<b>name</b>	El nombre de tu proyecto o librería. Debe ser único si decides publicarlo en npm.
<b>version</b>	La versión actual (normalmente siguiendo el estándar SemVer, como 1.0.0).
<b>description</b>	Una breve explicación de qué hace el proyecto. Ayuda a que otros lo encuentren en npm.

<b>type</b>	Define si el proyecto usa módulos de Node.js tradicionales (commonjs) o el estándar moderno de JavaScript (module).
<b>main</b>	El punto de entrada principal del programa (donde reside la lógica de inicio, ej.
<b>module</b>	(Opcional) Indica el punto de entrada para herramientas que utilizan módulos ES (como herramientas de empaquetado).
<b>scripts</b>	Un diccionario donde defines comandos personalizados (ej. "start": "node app.js", "test": "jest").
<b>keywords</b>	Un array de palabras clave que ayudan a indexar tu proyecto en las búsquedas.
<b>author</b>	El nombre del creador (puedes incluir email y sitio web).
<b>license</b>	La licencia del software (ej. MIT, ISC, Apache-2.0).
<b>dependencies</b>	Librerías necesarias para que la aplicación funcione en producción (ej. express).
<b>devDependencies</b>	Librerías necesarias solo para el desarrollo y pruebas (ej. nodemon, jest, eslint).
<b>homepage</b>	La URL de la página principal del proyecto.
<b>repository</b>	Indica dónde se encuentra el código fuente (ej. una URL de GitHub).
<b>bugs</b>	La URL o el correo electrónico donde los usuarios pueden reportar problemas.

# Hola soy Ismael



Edit `src/App.js` and save to reload.

[Learn React](#)