

Diagnóstico Node, npm y APIs

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Diagnóstico

- Crearse una cuenta en [Quizizz](#).
- Realizar el quizz.



Npm

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Node Package Manager es un manejador de paquetes de node (el más popular de JavaScript). Permite para compartir e instalar paquetes.

Se compone de 2 partes:

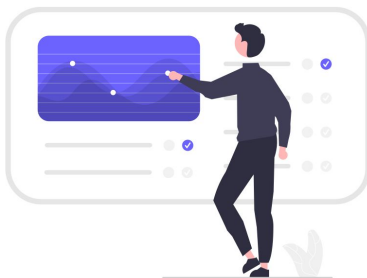
- Un **repositorio online** para **publicar y descargar paquetes** de software libre.
- Una **CLI** para la terminal **que permite interactuar** instalar, gestionar y publicar paquetes.

NOTA: Se puede considerar un gestor de dependencias de proyectos de tipo npm.

Gestor de dependencias

Son herramientas que facilita la instalación, actualización y eliminación de las dependencias de un proyecto de software. Las dependencias son bibliotecas o paquetes de software externos que un proyecto utiliza para su funcionamiento, como bibliotecas de utilidades, frameworks, módulos, etc.

En resumen, los gestores de dependencias son herramientas fundamentales en el desarrollo de software moderno ya que garantizan la consistencia y compatibilidad del entorno de desarrollo. Es importante que los desarrolladores sepan utilizarlos para maximizar la eficiencia y la fiabilidad de sus proyectos.



¿Por qué usar un gestor de dependencias?

1. **Gestión Eficiente de dependencias:** Ahorra tiempo y esfuerzo al desarrollador. En lugar de descargar e instalar manualmente cada biblioteca que se necesita.
2. **Control de versiones:** Garantiza que todas las versiones de las dependencias sean compatibles entre sí y que el proyecto funcione correctamente.
3. **Gestión de conflictos:** En proyectos grandes con muchas dependencias, es común que haya conflictos entre las versiones de las dependencias. Los gestores de dependencias pueden resolver estos conflictos automáticamente.
4. **Mejora la reproducibilidad:** Al especificar las versiones de las dependencias en un archivo de configuración (package.json) es más sencillo que otros desarrolladores puedan clonar el repositorio y replicar contar con el mismo entorno de desarrollo sin problemas.

Comandos de Npm

Inicialización de un proyecto npm

- **npm init:** Inicializa una carpeta como un proyecto de npm.

Levantar un proyecto npm

- **npm start:** Permite “iniciar” un proyecto de node. Es el único comando de tipo script que no requiere la palabra run.
 - **Sintaxis:** `npm run other-command`.

Comandos de Npm

Instalar/desinstalar dependencias

- **npm i**: Instala todas las dependencias del package.json
- **npm install -g <package-name>**: Instala un paquete globalmente.
- **npm install <package-name>**: Instala un paquete y lo agrega al package.json
- **npm install --save <package-name>**: Instala un paquete y lo agrega al package.json
- **npm install -D <package-name>**: Instala un paquete y lo agrega al package.json en la parte de devDependencies.
- **npm install --save-dev <package-name>**: Instala un paquete y lo agrega al package.json en la parte de devDependencies.

Comandos de Npm

Gestión de dependencias

- `npm uninstall <package-name>`
- `npm -g uninstall <package-name>`
- `npm search <package-name>`
- `npm ls`
- `npm update -save`
- `npm list`
- `npm list -g --depth 0`
- `npm outdated`

Paquetes

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Paquetes

Son módulos distribuidos en forma de librerías que resuelven alguna necesidad de desarrollo.

A continuación se listan algunos de los módulos:

- npm.
- expressjs.
- socket.
- mongoose.
- mocha.
- create-react-app.
- vite.

NOTA: Tu también puedes desarrollar tus propios paquetes y publicarlos.

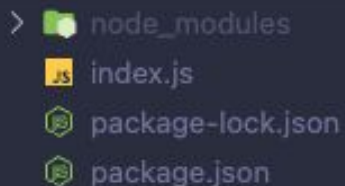
Scaffold Npm

DEV.FX
DESARROLLAMOS(PERSONAS);

dev

Estructura de proyecto npm

- **node_modules:** Carpeta donde se instalan las dependencias de un proyecto npm, normalmente esta carpeta se agrega al .gitignore.
- **package.json:** Guardan las dependencias y los comandos de node.
- **package-lock.json:** Guarda un snapshot de las dependencias que se instalaron en un determinado momento.



A screenshot of a file explorer window with a dark background. It shows a directory structure for an npm project. At the top, there is a folder icon followed by the text 'node_modules'. Below it is a file icon followed by 'index.js'. Then there are two more file icons followed by 'package-lock.json' and 'package.json' respectively. The icons are small and colored: a green folder icon, a yellow file icon with a JS extension, and green file icons with a lock and a package icon.

```
> node_modules  
index.js  
package-lock.json  
package.json
```

package.json

Este archivo guarda las dependencias y los comandos de node.

- **name:** Nombre que se le asigna al proyecto, generalmente es el mismo nombre de la carpeta y el repositorio.
- **versión:** Se utiliza para conocer la versión del proyecto, el estándar para versionar es SEMVER.
- **description:** Meta data sobre lo que trata el proyecto.
- **license:** Tipo de licencia de software del proyecto.

package.json

Este archivo guarda las dependencias y los comandos de node.

- **scripts:** Para declarar comandos largos que después se pueda ejecutar con `npm run <commande>`.
- **devDependencies:** Son dependencias que sólo se instalan en el entorno local.
- **dependencies:** Son dependencias que se instalan en cualquier entorno (local, test, qa, producción y pro).

package-lock.json

- Da visibilidad de los cambios en el árbol de dependencias.
- Garantiza la conciliación de dependencias.
- Se autogenera cada que un comando modifica el `node_modules` o el `package.json`.
- Es usualmente generado por el comando `npm install`.
- Este archivo tiene las versiones exactas de las dependencias utilizadas por un proyecto npm.
- Debe subirse al repositorio.

Scripts

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Scripts

Son comandos propios que se pueden agregar al package.json para poderlos ejecutar con **npm run <my-comand>**.

```
"scripts": {  
  "test": "echo \"Error: no test specified\" && exit 1",  
  "start": "node index.js",  
  "dev": "nodemon index.js"  
},
```