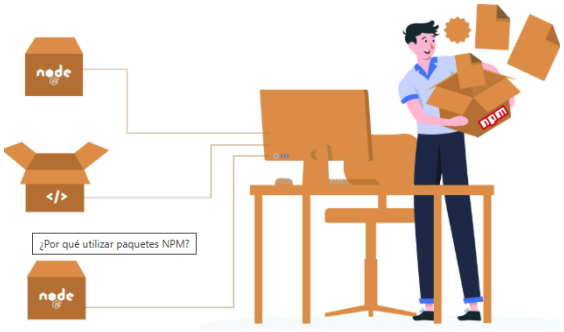
DESARROLLO M6_AE3_ABPRO-EJERCICIO GRUPAL SALA 4

Manejo de paquetes en Node.js con npm

1. **¿Qué es npm y para qué sirve?**

   NPM (Node Package Manager) es el gestor de paquetes por defecto de Node.js. Nos permite instalar, compartir y controlar las dependencias que necesita una aplicación. Es decir, facilita la incorporación de bibliotecas externas al proyecto en desarrollo, sin tener que escribir todo desde cero.



2. **Instalación y desinstalación de paquetes básicos con ejemplos (npm install, npm uninstall)**

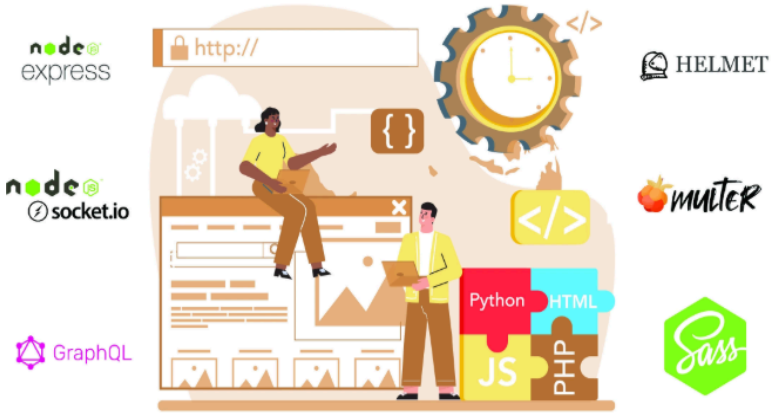
   Se pueden instalar paquetes según las necesidades del desarrollo de la aplicación. Sin embargo, es recomendable mantener únicamente los paquetes indispensables, ya que esto hace que el proyecto sea más liviano y reduce el riesgo de conflictos.

   ✔ Para instalar un paquete:
   npm install lodash

   Esto agrega la librería Lodash a nuestro proyecto para trabajar con arrays, objetos y cadenas.

   ✔ Desinstalar un paquete:
   npm uninstall lodash

   Esto elimina el paquete si ya no lo necesitamos.



3. **Manejo de versiones de paquetes: diferencias entre ^, ~, o instalación específica (@1.2.3).**

El manejo de versiones es útil para controlar que nuestro proyecto no se rompa por cambios inesperados en nuevas versiones de los paquetes.

✔ Símbolos importantes:

^1.2.3: Permite actualizar a cualquier versión mayor compatible (1.x.x).
Ejemplo: se actualizará a 1.3.0 o 1.9.9, pero no a 2.0.0.

~1.2.3: Permite actualizar solo el parche (1.2.x).
Ejemplo: se actualizará a 1.2.4 o 1.2.9, pero no a 1.3.0.

@1.2.3: Instala exactamente esa versión específica.

✔ Ejemplo práctico:
npm install express@4.18.2

- Instalar con versión exacta: npm install express@4.18.2
- instalar con ^ (caret, por defecto): npm install express@^4.18.2
- Instalar con ~ (tilde): npm install express@~4.18.2

4. **Comandos básicos de npm: init, install, uninstall, update, list, outdated.**

Comando	**Descripción**
npm init	Inicializa un proyecto y crea package.json
npm init -y	Crea un package.json con valores por defecto.
npm install	Instala todas las dependencias listadas en package. json.

npm install <paquete>	Desinstala un paquete
npm uninstall <paquete>	Elimina un paquete instalado
npm update	Actualiza los paquetes instalados
npm list	Lista todos los paquetes instalados en el proyecto
npm outdated	Muestra los paquetes desactualizados

Ejemplo: ***npm list***
Permite revisar qué paquetes ya están en nuestro proyecto y sus versiones.

5. ¿Qué es la carpeta node_modules y por qué no se sube al repositorio?

La carpeta **node_modules** se genera al instalar un paquete con **NPM** y contiene todos los archivos necesarios de las dependencias. Esta carpeta no debe modificarse manualmente ni subirse al repositorio GitHub, ya que ocupa demasiado espacio. En su lugar se utiliza el archivo **package.json**, el cual permite que otros desarrolladores instalen las mismas dependencias mediante el comando **npm install**.

6. ¿Qué contiene el archivo package.json? Mencionar propiedades clave

El archivo **package.json** es el corazón del proyecto Node.js. Contiene la configuración y metadatos para un proyecto, como su nombre, versión y autor, además de las dependencias de software (librerías) necesarias para que funcione.

Contiene propiedades clave:

- ✔ name: nombre del proyecto.
- ✔ version: versión del proyecto.
- ✔ description: breve descripción del proyecto.
- ✔ main: ruta al archivo principal de la aplicación.
- ✔ dependencies: paquetes necesarios para ejecutar la app.
- ✔ devDependencies: paquetes necesarios solo para desarrollo.
- ✔ scripts": comandos personalizados (por ejemplo, start, test).
- ✔ Keywords: una matriz de palabras clave que ayudan a describir tu proyecto y a encontrarlo en los registros de npm.
- ✔ Autor: información sobre el autor del proyecto.
- ✔ License: indica bajo qué licencia se distribuye el proyecto.

Ejemplo:

{

```
        "name": "mi-proyecto",
        "version": "1.0.0",
        "description": "Comprender qué es NPM y para qué se utiliza",
        "main": "index.js",
        "scripts": {
        "start": "node index.js",
        "dev": "nodemon index.js"
},
"dependencies": {
          "express": "^4.18.2"
},
"devDependencies": {
          "nodemon": "^3.0.1"
},
"author": "M6_AE3_ABPRO-Ejercicio grupal – SALA 4",
"license": "MIT"
}
```

7. **¿Cómo se requiere un paquete dentro de un archivo .js en Node? (con ejemplos).**

Para utilizar un paquete dentro de una aplicación y/o proyecto se debe importar:

✔ Ejemplo con un paquete instalado:
```
const express = require('express');
const app = express();
```

✔ Ejemplo con archivo propio:
```
const funciones = require('./misFunciones.js');
funciones.saludar();
```

8. **Diferencia entre importar archivos propios y paquetes externos**

✔ Importar paquetes externos:
- Son librerías o dependencias instaladas con npm install o que ya vienen incluidas en Node.js.
- No necesitan ruta relativa (./ o ../).

✔ Importar archivos propios:
- Son archivos o módulos creados dentro del proyecto.

- Siempre requieren una ruta relativa (./ para el mismo directorio, ../ para subir un nivel).

9. ¿Cómo se exportan funciones o módulos desde un archivo en Node.js?

Para compartir funciones, objetos o clases entre archivos, se usa **module.exports.**

✔ Exportar una función desde un archivo propio:

```
// misFunciones.js
function saludar() {
   console.log("Hola equipo SALA 4!");
}

module.exports = { saludar };

o export function saludar();
```

✔ Importar en otro archivo.

```
const funciones = require('./misFunciones.js');
funciones.saludar(); // Imprime: Hola equipo!

o import {saludar} from './misFunciones.js';
```

✔ También puedes exportar varios elementos:

```
module.exports = {
  saludar,
  despedir
};
```

Conclusión del desarrollo

✔ NPM facilita el manejo de librerías y dependencias en Node.js.

✔ Conocer comandos básicos, la estructura de package.json y la diferencia entre paquetes externos y archivos propios permite un desarrollo más organizado y colaborativo.

✔ La práctica en equipo y los ejemplos concretos ayudan a comprender estos conceptos de forma más clara y aplicable.