

ACTIVIDAD DE PROYECTO FORMATIVO

INSTRUCTOR: LUIS FERNANDO SANCHEZ

APRENDIZ: ALEJANDRO GOMEZ GARCIA

FICHA: 3223875

CTMA

## ACTIVIDAD #1-NO ES EVIDENCIA

### Título:

Análisis de la Experiencia de Usuario y Estándares de Diseño en Aplicaciones Digitales

### Introducción

Una interfaz gráfica de usuario (GUI) es el medio mediante el cual una persona interactúa con un sistema o aplicación. Su diseño debe ser intuitivo, limpio y accesible, ya que influye directamente en la facilidad de uso y en la satisfacción del usuario. Para lograrlo, se aplican estándares de diseño como Material Design (Google) y Human Interface Guidelines (Apple), que orientan sobre la estructura, los colores y la coherencia visual de las aplicaciones.

### Desarrollo

#### Aplicaciones con buena experiencia

- **Spotify:**

presenta una interfaz ordenada con menús visibles, colores contrastantes y navegación sencilla. Los botones de reproducción son grandes y accesibles, lo que facilita el uso incluso en dispositivos móviles.

Estándares aplicados: Material Design — consistencia visual, jerarquía clara y accesibilidad.

- **Google Maps:**

ofrece un diseño funcional y limpio. Los iconos y rutas están bien diferenciados, y la aplicación responde rápidamente a las acciones del usuario.

Estándares aplicados: simplicidad, diseño responsivo y retroalimentación visual.

## **Aplicaciones con mala experiencia**

- **Sitios de noticias con exceso de anuncios:**

muestran ventanas emergentes y banners que distraen al usuario, generando desorden y lentitud.

Estándares ignorados: claridad, jerarquía visual y equilibrio de contenido.

- **Aplicaciones de transporte mal diseñadas:**

incluyen colores similares, botones poco visibles y menús ocultos, dificultando la reserva o el seguimiento de viajes.

Estándares ignorados: consistencia, accesibilidad y navegación intuitiva.

## **Conclusión**

Seguir estándares como Material Design y HIG permite crear interfaces más agradables, coherentes y fáciles de usar, mejorando la experiencia del usuario. Ignorarlos genera confusión y frustración, afectando la percepción de calidad y la eficiencia del software.

## ACTIVIDAD #2

### ¿Qué es un gestor de dependencias de código?

Es una herramienta que automatiza la instalación, actualización y administración de librerías o paquetes que necesita un proyecto para funcionar.

En JavaScript, permite agregar fácilmente código de terceros (como frameworks o utilidades) sin tener que hacerlo manualmente.

### ¿Qué es npm?

npm (Node Package Manager) es el gestor de dependencias oficial de Node.js.

Permite descargar, instalar, actualizar y compartir paquetes de JavaScript publicados por otros desarrolladores en el registro de npm.

### ¿Para qué se utiliza principalmente npm?

Se utiliza principalmente para:

- Instalar dependencias (librerías, frameworks, herramientas).
- Ejecutar scripts definidos en el proyecto.
- Gestionar versiones de las dependencias.
- Publicar y compartir paquetes propios en el registro npm.

### ¿Qué es el versionado semántico?

El versionado semántico (SemVer) es un sistema de numeración que indica los cambios realizados en una librería o proyecto mediante tres números:

MAJOR.MINOR.PATCH

Ejemplo: 2.4.1

## **¿Cómo está especificado el versionado semántico?**

- MAJOR: Cambia cuando se hacen modificaciones incompatibles con versiones anteriores.  
Ejemplo: 1.x.x → 2.0.0
- MINOR: Cambia cuando se agregan nuevas funcionalidades compatibles.  
Ejemplo: 1.2.0 → 1.3.0
- PATCH: Cambia cuando se corrigen errores o se hacen pequeñas mejoras.  
Ejemplo: 1.3.1 → 1.3.2

## **¿Qué son las dependencias locales?**

Son las librerías instaladas solo para un proyecto específico, dentro de su carpeta. Se almacenan en la carpeta node\_modules y se declaran en el archivo package.json.

Ejemplo: npm install express

## **¿Qué son las dependencias de desarrollo?**

Son las dependencias necesarias solo durante la fase de desarrollo, no en producción.

Ejemplo: herramientas de pruebas o compilación como eslint, jest, o webpack.  
Se instalan con:

npm install --save-dev nombre\_paquete

## **¿Qué son las dependencias globales?**

Son paquetes instalados en el sistema, no en un proyecto específico, y pueden usarse desde cualquier ubicación en la terminal.

Ejemplo: instalar nodemon o npm globalmente:

npm install -g nodemon

## **¿Qué es el archivo package.json, qué apartados tiene y cuál es su utilidad?**

El archivo package.json es el archivo principal de configuración de un proyecto Node.js.

Define la información del proyecto y sus dependencias.

Apartados comunes:

- "name" → nombre del proyecto
- "version" → versión del proyecto
- "description" → descripción breve
- "scripts" → comandos automatizados (por ejemplo, npm start)
- "dependencies" → dependencias necesarias
- "devDependencies" → dependencias de desarrollo
- "author", "license", "repository" → información adicional

### **Utilidad:**

Permite reproducir y compartir el proyecto fácilmente, ya que contiene todo lo necesario para instalar sus dependencias con un solo comando:

`npm install`

## **¿Qué es el archivo package-lock.json y qué utilidad tiene?**

package-lock.json registra las versiones exactas de las dependencias instaladas (y sus subdependencias).

Su función es garantizar que todos los desarrolladores usen las mismas versiones del proyecto, evitando errores por diferencias de versiones.

## **¿Qué es la carpeta node\_modules en un proyecto de npm?**

La carpeta node\_modules contiene todos los paquetes y librerías instaladas mediante npm.

Cada dependencia puede tener sus propias dependencias internas, formando una estructura de carpetas jerárquica.

No se recomienda modificar nada dentro de ella manualmente.

## Cheat sheet

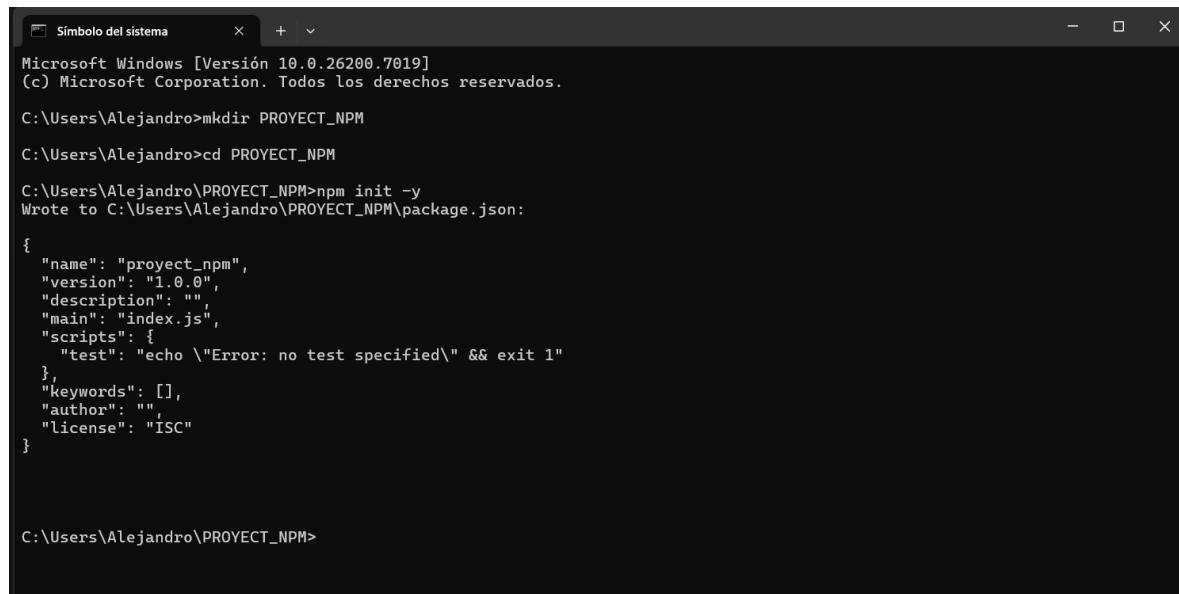
Proceso	Comando	Descripción / Uso
<b>Inicializar un proyecto npm</b>	npm init npm init -y	Crea el archivo package.json con la configuración del proyecto. -y usa valores por defecto.
<b>Instalar dependencias locales</b>	npm install nombre_paquete npm i nombre_paquete	Instala un paquete solo para el proyecto actual (guardado en node_modules y package.json).
<b>Instalar dependencias de desarrollo</b>	npm install --save-dev nombre_paquete npm i -D nombre_paquete	Instala librerías necesarias solo en desarrollo (por ejemplo: eslint, jest).
<b>Instalar dependencias globales</b>	npm install -g nombre_paquete	Instala el paquete globalmente, disponible desde cualquier proyecto o terminal.
<b>Visualizar dependencias instaladas</b>	npm list --depth=0 npm list -g --depth=0	Muestra las dependencias locales o globales instaladas (sin subdependencias).
<b>Instalar una versión específica</b>	npm install nombre_paquete@1.2.3	Instala una versión exacta del paquete. Usa @latest para la más reciente.
<b>Crear un comando (scripts en package.json)</b>	En package.json: "scripts": { "start": "node app.js", "dev": "nodemon app.js" } Ejecutar → npm run dev	Define comandos personalizados para automatizar tareas (ej. iniciar servidor, testear, compilar).
<b>Actualizar dependencias</b>	npm update npm pm update nombre_paquete	Actualiza todas las dependencias o una específica a su versión más reciente compatible.
<b>Eliminar paquetes</b>	npm uninstall nombre_paquete npm remove nombre_paquete	Elimina el paquete del proyecto y lo quita del package.json.
<b>Actualizar/Reinstalar node_modules</b>	rm -rf node_modules npm install npm rebuild	Elimina y reinstala todas las dependencias del proyecto para resolver errores o limpiar versiones.

## ACTIVIDAD #3

### Manejo de dependencias en npm

#### 1. Inicialización de un Proyecto npm

##### Crear una Carpeta para el Proyecto y Inicializar un Proyecto npm



```
Microsoft Windows [Versión 10.0.26200.7019]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Alejandro>mkdir PROYECT_NPM
C:\Users\Alejandro>cd PROYECT_NPM

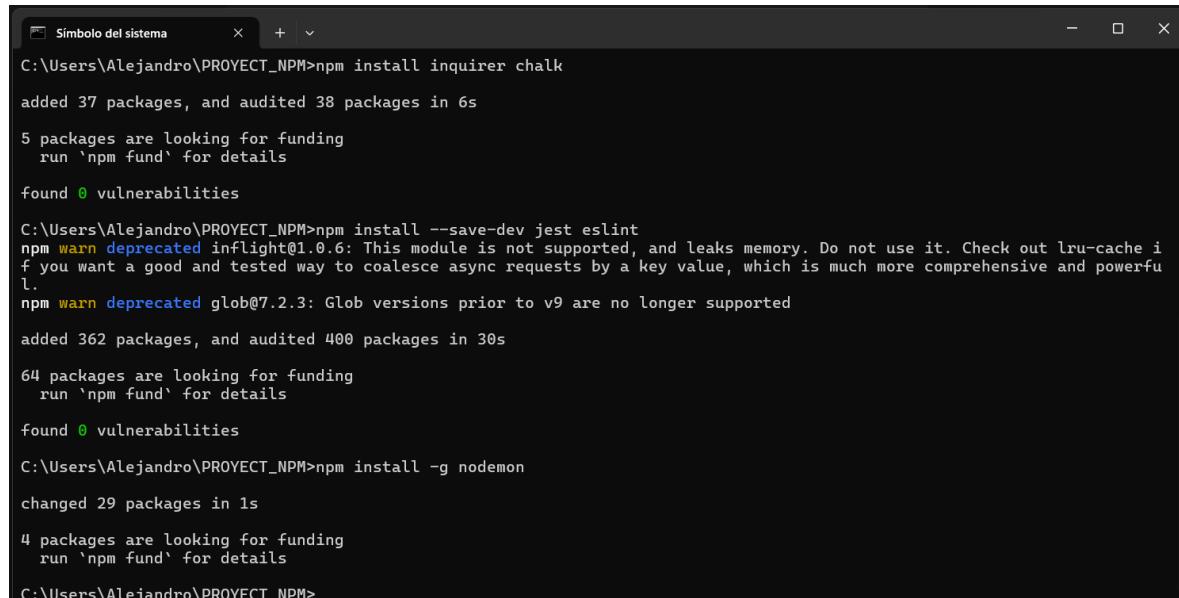
C:\Users\Alejandro\PROYECT_NPM>npm init -y
Wrote to C:\Users\Alejandro\PROYECT_NPM\package.json:

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

C:\Users\Alejandro\PROYECT_NPM>
```

#### 2. Instalación de Dependencias

##### Instalar Dependencias Locales y Instalar Dependencias de Desarrollo y Instalar Dependencias Globales



```
C:\Users\Alejandro\PROYECT_NPM>npm install inquirer chalk
added 37 packages, and audited 38 packages in 6s

5 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>npm install --save-dev jest eslint
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported

added 362 packages, and audited 400 packages in 30s

64 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

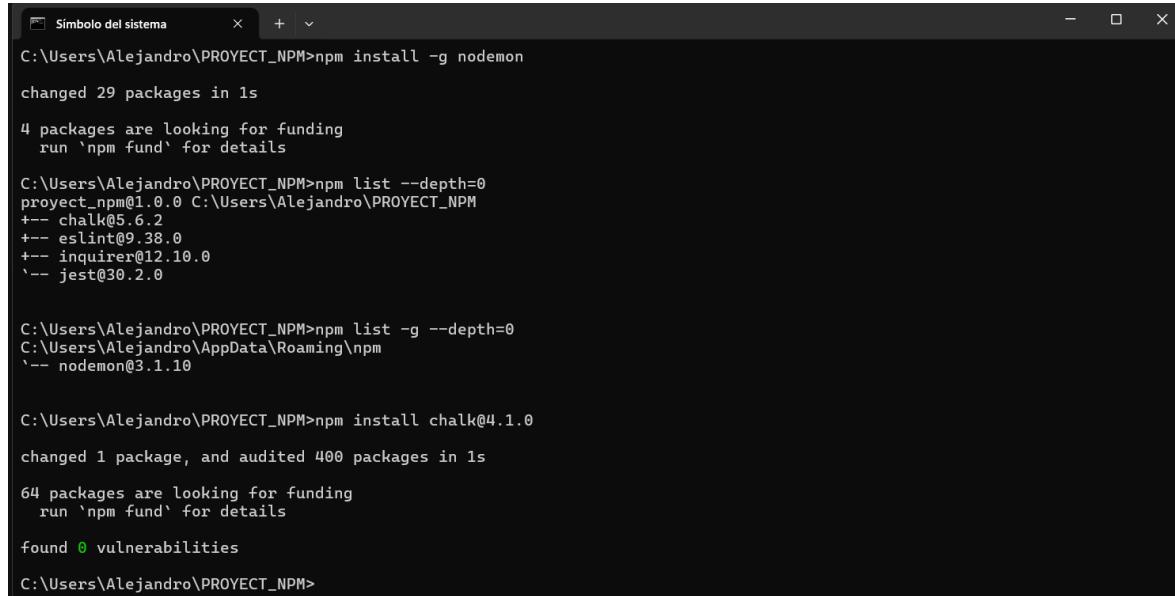
C:\Users\Alejandro\PROYECT_NPM>npm install -g nodemon
changed 29 packages in 1s

4 packages are looking for funding
  run `npm fund` for details

C:\Users\Alejandro\PROYECT_NPM>
```

### 3. Gestión de Paquetes

Para visualizar Paquetes Instalados, Instalar una Versión Específica de un Paquete



```
Símbolo del sistema x + v
C:\Users\Alejandro\PROYECT_NPM>npm install -g nodemon
changed 29 packages in 1s
4 packages are looking for funding
  run 'npm fund' for details

C:\Users\Alejandro\PROYECT_NPM>npm list --depth=0
proyecto_npm@1.0.0 C:\Users\Alejandro\PROYECT_NPM
+-- chalk@5.6.2
++-- eslint@9.38.0
+-- inquirer@12.10.0
`-- jest@30.2.0

C:\Users\Alejandro\PROYECT_NPM>npm list -g --depth=0
C:\Users\Alejandro\AppData\Roaming\npm
`-- nodemon@3.1.10

C:\Users\Alejandro\PROYECT_NPM>npm install chalk@4.1.0
changed 1 package, and audited 400 packages in 1s
64 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

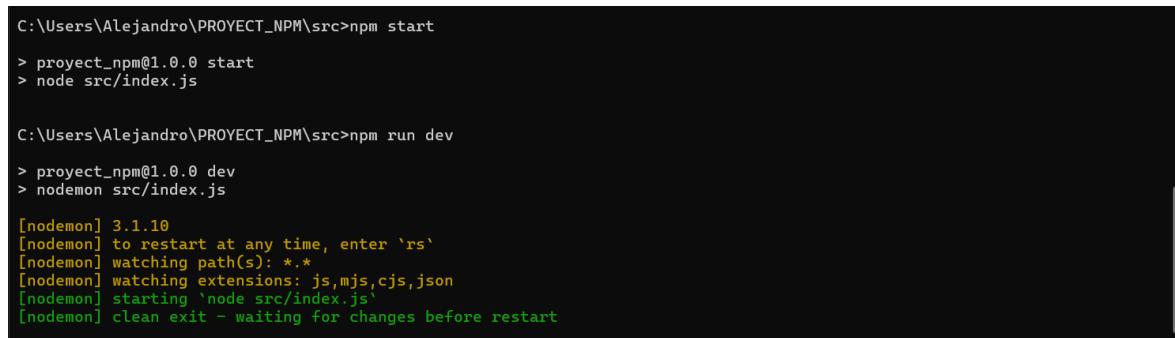
C:\Users\Alejandro\PROYECT_NPM>
```

### 4. Scripts y Actualizaciones

Crear la Estructura de Directorios del Proyecto

Organiza tu proyecto creando una carpeta src para el código fuente:

Ahora, puedes ejecutar tu aplicación con: npm run dev



```
C:\Users\Alejandro\PROYECT_NPM\src>npm start
> proyecto_npm@1.0.0 start
> node src/index.js

C:\Users\Alejandro\PROYECT_NPM\src>npm run dev
> proyecto_npm@1.0.0 dev
> nodemon src/index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching path(s): ***!
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting 'node src/index.js'
[nodemon] clean exit - waiting for changes before restart
```

### Crear un Comando (Script) en el Proyecto

Puedes definir scripts personalizados en el package.json. Por ejemplo, para ejecutar tu aplicación:

```
"scripts": [
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node src/index.js",
  "dev": "nodemon src/index.js"
],
```

## Actualizar Dependencias

### 1. Actualizar Dependencias Locales

```
npm outdated
```

Este comando te mostrará una tabla con las dependencias desactualizadas: versión instalada, versión deseada y última versión disponible.

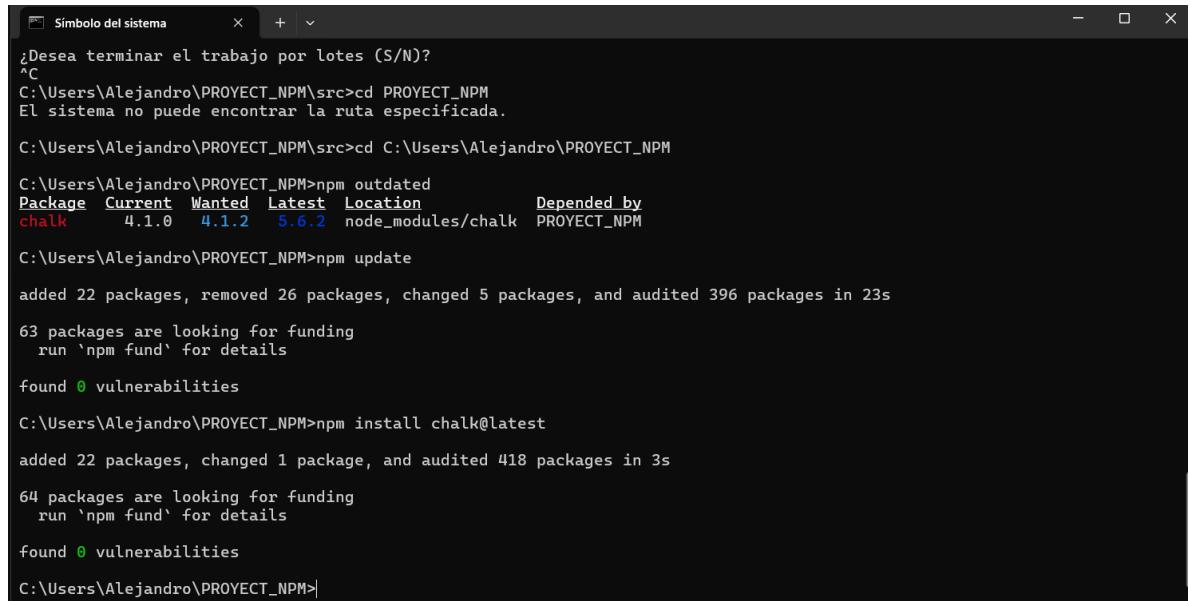
Actualizar todas las dependencias locales

```
npm update
```

Actualizar una dependencia a la última versión absoluta

A veces quieras forzar la actualización a la última versión disponible, incluso si es una nueva versión mayor (puede incluir breaking changes): `npm install chalk@latest`

Ejemplo:



```
Símbolo del sistema x + v
:Desea terminar el trabajo por lotes (S/N)?
`C
C:\Users\Alejandro\PROYECT_NPM\src>cd PROYECT_NPM
El sistema no puede encontrar la ruta especificada.

C:\Users\Alejandro\PROYECT_NPM\src>cd C:\Users\Alejandro\PROYECT_NPM

C:\Users\Alejandro\PROYECT_NPM>npm outdated
Package Current Wanted Latest Location Depended by
chalk 4.1.0 4.1.2 5.6.2 node_modules(chalk) PROYECT_NPM

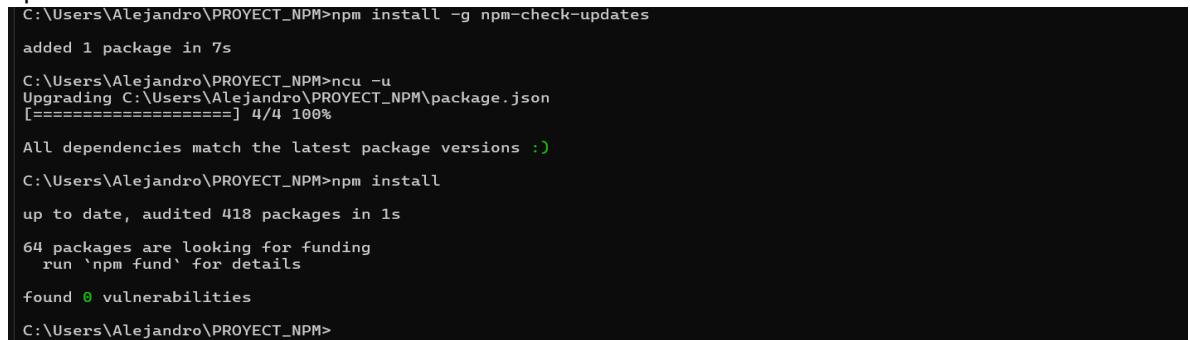
C:\Users\Alejandro\PROYECT_NPM>npm update
added 22 packages, removed 26 packages, changed 5 packages, and audited 396 packages in 23s
63 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>npm install chalk@latest
added 22 packages, changed 1 package, and audited 418 packages in 3s
64 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>
```

Para actualizar a la última versión absoluta (rompiendo los rangos), puedes usar la herramienta [npm-check-updates \(ncu\)](#):

`npm install -g npm-check-updates ncu -u` # Actualiza los rangos de versión en package.json  
`npm install` # Instala las nuevas versiones:



```
C:\Users\Alejandro\PROYECT_NPM>npm install -g npm-check-updates
added 1 package in 7s

C:\Users\Alejandro\PROYECT_NPM>ncu -u
Upgrading C:\Users\Alejandro\PROYECT_NPM\package.json
[=====] 4/4 100%
All dependencies match the latest package versions :)

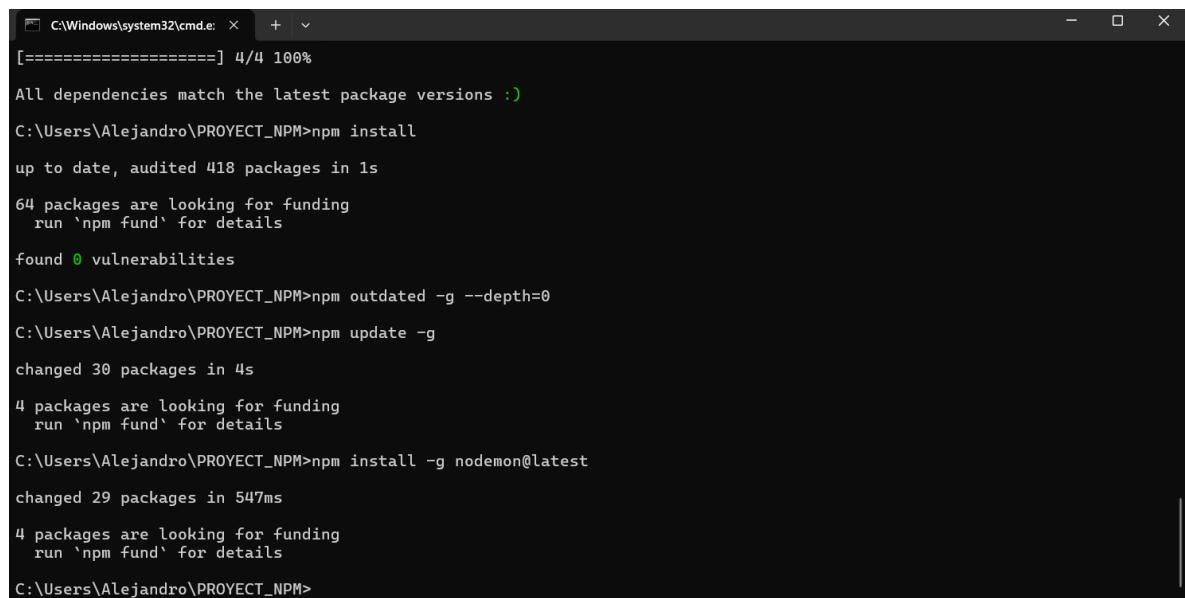
C:\Users\Alejandro\PROYECT_NPM>npm install
up to date, audited 418 packages in 1s
64 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>
```

## 2. Actualizar Dependencias Globales

Las dependencias globales son herramientas que instalaste con el flag -g, como nodemon, npm-check-updates, o herramientas CLI.

Para mantener actualizados los paquetes globales en npm, primero se puede verificar cuáles están desactualizados con el comando npm outdated -g --depth=0, que muestra una lista de los paquetes instalados globalmente junto con sus versiones actuales y disponibles. Luego, para actualizar todos los paquetes globales a sus versiones más recientes compatibles, se utiliza npm update -g. Si se desea actualizar un paquete específico, se puede ejecutar npm install -g nombre-paquete, y para instalar la última versión absoluta del mismo, se añade @latest, por ejemplo: npm install -g nodemon@latest



```
C:\Windows\system32\cmd.exe: X + V
[=====] 4/4 100%
All dependencies match the latest package versions :D
C:\Users\Alejandro\PROYECT_NPM>npm install
up to date, audited 418 packages in 1s
64 packages are looking for funding
  run 'npm fund' for details
found 0 vulnerabilities
C:\Users\Alejandro\PROYECT_NPM>npm outdated -g --depth=0
C:\Users\Alejandro\PROYECT_NPM>npm update -g
changed 30 packages in 4s
4 packages are looking for funding
  run 'npm fund' for details
C:\Users\Alejandro\PROYECT_NPM>npm install -g nodemon@latest
changed 29 packages in 547ms
4 packages are looking for funding
  run 'npm fund' for details
C:\Users\Alejandro\PROYECT_NPM>
```

## 5. Limpieza y Mantenimiento

Para eliminar un paquete del proyecto y actualizar las dependencias, se utiliza el comando npm uninstall nombre-del-paquete, que también lo elimina del archivo package.json. Por ejemplo, npm uninstall chalk desinstala ese paquete específico. Si se desea reinstalar todas las dependencias desde cero —por ejemplo, para corregir errores o conflictos en los módulos— se debe eliminar la carpeta node\_modules y luego ejecutar npm install, lo que descarga nuevamente todas las dependencias definidas en el package.json.

```
C:\Users\Alejandro\PROYECT_NPM>npm uninstall chalk
removed 1 package, and audited 417 packages in 2s
63 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>rm -rf node_modules
"rm" no se reconoce como un comando interno o externo,
programa o archivo por lotes ejecutable.

C:\Users\Alejandro\PROYECT_NPM>rmdir /s /q node_modules

C:\Users\Alejandro\PROYECT_NPM>npm install
npm warn deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async requests by a key value, which is much more comprehensive and powerful.
npm warn deprecated glob@7.2.3: Glob versions prior to v9 are no longer supported
added 416 packages, and audited 417 packages in 5s
63 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

C:\Users\Alejandro\PROYECT_NPM>
```