Título del documento:
Gestion de dependencias de código según lenguaje de programación.

Nombre del estudiante:
Liliana Panesso

Institución:
SENA

Curso:
Análisis y Desarrollo de Software

Profesor:
Luis Sanchez

Ficha:
3223873

Introducción

En el desarrollo de proyectos de software, especialmente con JavaScript y Node.js, es fundamental comprender cómo funcionan los gestores de dependencias. Estas herramientas permiten instalar, actualizar y organizar los paquetes que necesita un proyecto para ejecutarse correctamente. En esta actividad se presentan definiciones básicas sobre npm, el versionado semántico, los tipos de dependencias y archivos clave como package.json y package-lock.json. Además, se incluye una cheat sheet con los comandos más utilizados, lo que facilita el aprendizaje práctico y el manejo eficiente de proyectos en JavaScript.

Descripción de la Actividad:

El aprendiz de forma individual el aprendiz, realizara la definicion de los siguientes términos sobre

los procesos de gestion de dependencias en Javascript según los siguientes interrogantes:

- ***¿Que es un gestor dependencias de codigo?***

  R/:son aquellas herramientas que nos permiten instalar, actualizar y eliminar estas dependencias de terceros que tiene nuestro proyecto para su funcionamiento mediante una línea de comandos, los más usados son NPM y Yarn.

- ***¿Qué es npm?***

  R/:NPM, cuyas siglas significan *Node Package Manager*, es un administrador de paquetes que permite a los desarrolladores de JavaScript gestionar fácilmente las dependencias de sus proyectos. Se encarga de rastrear qué módulos dependen de otros, instalar automáticamente los necesarios y mantener actualizadas las bibliotecas utilizadas.

- ***¿Para qué se utiliza principalmente npm?***

R/:NPM (*Node Package Manager*) se utiliza principalmente para gestionar las dependencias de proyectos desarrollados con Node.js o JavaScript en general. Al ejecutar el comando npm init, se crea un archivo llamado package.json, que contiene los metadatos del proyecto, como el nombre, versión, autor, dependencias y scripts. Este comando lanza un asistente en la consola que solicita información básica para componer dicho archivo.

- ***¿Qué es el versionado semántico?***

  R/:El versionado semántico, también llamado *Semantic Versioning* o *semver*, es una forma ordenada de ponerle número a cada versión de un software. Sirve para indicar si se han hecho cambios pequeños, mejoras importantes o ajustes que podrían afectar el funcionamiento del programa.

  Además, usar este sistema ayuda a tener un control claro de los avances y modificaciones del proyecto. Así, cualquier persona que trabaje con ese software puede saber fácilmente qué ha cambiado y cómo afecta a su trabajo.

- ***Como está especificado el Versionado Semántico.***

  R/: El Versionado Semántico (SemVer) se especifica mediante tres números en el formato **X.Y.Z**, donde cada dígito representa un tipo específico de cambio: la versión **Mayor (X)** para cambios que rompen la compatibilidad, la versión **Menor (Y)** para nuevas funcionalidades compatibles con versiones anteriores y la versión **Parche (Z)** para corregir errores compatibles con versiones anteriores.

- ***Qué son las dependencias locales***

  R/: Las dependencias locales en Node.js son paquetes que se instalan directamente dentro del proyecto en el que se esta trabajando, en lugar de estar disponibles para todos los proyectos del sistema. Esto significa que se guardan en una carpeta llamada node_modules dentro del proyecto, se registran automáticamente en el archivo package.json bajo las secciones "dependencies" o "devDependencies", y solo afectan ese proyecto específico, sin interferir con otros estén en el computador. Este sistema permite mantener cada proyecto bien organizado, con sus propias herramientas y configuraciones.

- ***Qué son las dependencias de desarrollo***

  R/: Las **dependencias de desarrollo** (también llamadas **devDependencies**) son los **paquetes que solo se necesitan durante la etapa de desarrollo** de un proyecto, **no cuando la aplicación ya está en producción o funcionando para los usuarios.**

- ***Qué son las dependencias globales***

  R/: Las **dependencias globales** son paquetes o módulos que se **instalan en todo el sistema operativo**, no solo dentro de un proyecto específico.
  Esto significa que se pueden usar **desde cualquier carpeta o proyecto**, sin tener que volver a instalarlos cada vez.

- ***¿Que es el archivo package.json y que apartados tiene y cual es su utilidad?***

R/:Es como la **“tarjeta de identidad” del proyecto**: contiene toda la información necesaria para que otros (y Node.js) sepan **qué hace el proyecto, qué necesita para funcionar y cómo ejecutarlo**.

Apartados principales del package.json

Apartado	***Descripción / Utilidad***
name	Nombre del proyecto. No puede tener espacios ni caracteres especiales. Ejemplo: "name": "mi-proyecto".
version	Versión actual del proyecto siguiendo el formato semántico (1.0.0).
description	Breve descripción del proyecto o su propósito.
main	Archivo principal que se ejecuta (por defecto suele ser index.js).
scripts	Contiene comandos que se pueden ejecutar con npm run. Ejemplo: "start": "node index.js".
author	Nombre del creador o responsable del proyecto.
license	Tipo de licencia (por ejemplo, "MIT", "ISC", "GPL-3.0").
dependencies	Lista de paquetes necesarios para que el proyecto funcione (se instalan con npm install).

Apartado	*Descripción / Utilidad*
devDependencies	Paquetes usados solo durante el desarrollo (se instalan con npm install --save-dev).
repository	Indica el repositorio del proyecto (por ejemplo, en GitHub).
keywords	Palabras clave relacionadas con el proyecto (sirven para identificarlo).
bugs	Información de contacto o enlace para reportar errores.
homepage	Página web oficial o documentación del proyecto.

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

  R/:El archivo **package-lock.json** es un documento que **se genera automáticamente** cuando se instalan dependencias en un proyecto de **Node.js** usando **npm**.

  Su función principal es **registrar las versiones exactas** de todos los paquetes y sus subdependencias (las dependencias de las dependencias).

- ***que es la carpeta node_modules en un proyecto de npm***

R/: La carpeta **node_modules** es una carpeta que **se crea automáticamente** en un proyecto de **Node.js** cuando instalas dependencias usando **npm** (Node Package Manager). Es la **carpeta donde se guardan todos los paquetes y librerías** que tu proyecto necesita para funcionar.

- ***Realiza una Cheat sheet de los siguientes procesos (ver ejemplo:https://pin.it/2TauqeogV):***
    - como inicializar un proyecto de npm.
    - Cómo instalar dependencias locales
    - Cómo instalar dependencias de desarrollo
    - Cómo instalar dependencias globales
    - Cómo visualizar los dependencias instaladas
    - Cómo instalar una versión específica de un paquete
    - Cómo crear un comando en tu proyecto(scripts en el package.json)
    - Cómo actualizar dependencias
    - Cómo eliminar paquetes
    - Cómo actualizar **node_modules**

      R/:

CHEAT SHEET

como inicializar un proyecto de npm.

Crea el archivo package.json que define el proyecto, copiando en la consola el comando correspondiente.

```bash
npm init -y
```

Cómo instalar dependencias locales

Se instalan solo para el proyecto actual (se guardan en node_modules y package.json). Ejecutando el comando npm install mas el nombre del paquete.

```bash
npm install nombre-paquete
```

Cómo instalar dependencias de desarrollo

Solo se usan durante el desarrollo (no en producción).Se ejecuta en la consola el comando npm install seguido del nombre de la carpeta + espacio + --save-dev.

```bash
npm install nodemon --save-dev
```

Cómo instalar dependencias globales

Se instalan en todo el sistema y pueden usarse en cualquier proyecto. Se ejecuta en la consola el comando npm install + -g + espacio + nombre de la carpeta.

```bash
npm install -g nombre-paquete
```

Cómo visualizar los dependencias instaladas

Para ver solo los paquetes principales (sin subdependencias):npm list --depth=0
Para ver los globales: npm list -g --depth=0

Para listar las dependencias locales instaladas:

```
npm list --depth=0
```

Para listar las dependencias globales:

```
npm list -g --depth=0
```

Cómo instalar una versión específica de un paquete

Puedes elegir una versión concreta del paquete. se ejecuta en la consola el comando npm install + nombre del paquete + @ + version del paquete.

```bash
npm install nombre-paquete@versión
```

CHEAT SHEET

Cómo crear un comando en tu proyecto(scripts en el package.json)

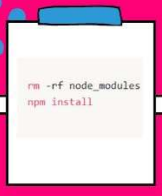
Se agrega dentro del "scripts" del package.json. y se ejecuta el comando npm + run + start o npm + run + dev.

Cómo actualizar dependencias

Se utiliza el comando npm + update para Actualizar todos los paquetes a sus últimas versiones compatibles o npm + update nombre-paquete para Actualizar un paquete específico.

Cómo eliminar paquetes

se ejecuta el comando npm + uninstall + nombre-paquete en consola. Quita un paquete instalado y lo elimina del package.json.



Cómo actualizar node_modules

Se ejecuta el siguiente comando en consola: rm -rf node_modules npm install

Conclusión

El manejo de dependencias en JavaScript es un proceso esencial para garantizar que los proyectos funcionen de manera ordenada y eficiente. Herramientas como npm simplifican la instalación y actualización de paquetes, mientras que el versionado semántico ayuda a mantener un control claro sobre los cambios realizados en el software. Asimismo, la correcta gestión de dependencias locales, globales y de desarrollo permite que cada proyecto tenga sus propias configuraciones sin afectar otros. Finalmente, los archivos *package.json* y *package-lock.json*, junto con la carpeta *node_modules*, son piezas clave para la organización del proyecto. Con la cheat sheet elaborada, se cuenta con una guía práctica que facilita la aplicación de estos conceptos en el día a día del desarrollo.