ACTIVIDAD DE PROYECTO FORMATIVO

INSTRUCTOR: LUIS FERNANDO SANCHEZ

APRENDIZ: SAMUEL RENDON LOAIZA

FICHA: 3223875

CTMA

ACTIVIDAD 1: NO ES EVIDENCIA

TITULO

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

INTRODUCCION

Una interfaz gráfica es la parte visual de una aplicación o sitio web que permite al usuario interactuar con el sistema mediante elementos como botones, íconos, menús y ventanas, en lugar de usar solo texto o comandos.

Es importante que sea clara y fácil de usar porque una buena interfaz permite que el usuario entienda rápidamente cómo realizar tareas, ahorre tiempo y disfrute la experiencia, evitando confusión o errores.

DESARROLLO

Aplicaciones con buena experiencia:

1. YouTube:

- **Elementos que facilitaron el uso**: menú claro, colores coherentes, búsqueda rápida
- **Estándares aplicados:** material design (claridad, consistencia)

2. WhatsApp:

- **Elementos que facilitaron el uso:** botones grandes, chat intuitivo, iconos conocidos.
- **Estándares aplicados:** simplicidad, accesibilidad.

Aplicaciones con mala experiencia:

1. sitio con mucha publicidad:

- **Dificultades:** saturación visual, lentitud, confusión.
- **Estándares ignorados:** claridad, jerarquía visual.

2.app de compras confusas:

- **Dificultades:** colores similares, navegación poco intuitiva.
- **Estándares ignorados:** consistencia, diseño responsivo.

CONCLUSION

Adoptar estándares de diseño mejora la experiencia del usuario, facilita la navegación y hace que las aplicaciones sean más accesibles y agradables. Ignorarlos genera confusión y frustración.

ACTIVIDAD2

1. Gestor de dependencias de código:
Un gestor de dependencias de código es una herramienta que permite instalar, actualizar y administrar las librerías o paquetes que un proyecto necesita para funcionar correctamente. Gracias a estos gestores, los desarrolladores pueden mantener su código organizado y asegurarse de que todas las partes del proyecto utilicen las versiones correctas de las librerías. Ejemplos comunes de gestores de dependencias son npm, Yarn y pnpm.

2. que es npm:
npm (Node Package Manager) es el gestor de dependencias oficial de Node.js. Su función principal es permitir la instalación, actualización y administración de paquetes o librerías de JavaScript que facilitan el desarrollo de aplicaciones web, móviles o de servidor. Además, npm cuenta con un repositorio en línea donde los desarrolladores pueden compartir sus propios paquetes con la comunidad.

3. Para qué se utiliza principalmente npm:
npm se utiliza principalmente para instalar y gestionar paquetes de JavaScript dentro de un proyecto. También permite ejecutar scripts personalizados, automatizar tareas como iniciar un servidor o compilar código, y manejar las dependencias del proyecto. En general, npm simplifica el proceso de desarrollo y mantiene un control claro de las herramientas utilizadas en cada proyecto.

4. Versionado semántico:
El versionado semántico es un sistema que permite identificar los cambios realizados en una aplicación o librería mediante un formato de tres números: Mayor.Menor.Patch (por ejemplo, 2.5.1). Este método facilita a los desarrolladores saber si una actualización introduce cambios importantes, nuevas funciones o solo corrige errores, manteniendo la compatibilidad entre versiones.

5. Cómo está especificado el versionado semántico:
El versionado semántico se especifica en tres partes: el número Mayor (X) cambia cuando se realizan modificaciones que rompen la compatibilidad con versiones anteriores; el número Menor (Y) aumenta cuando se agregan nuevas funciones sin afectar la compatibilidad; y el número Patch (Z) se incrementa cuando se corrigen errores o se hacen mejoras menores. Por ejemplo, la versión 2.3.4 indica versión mayor 2, menor 3 y patch 4.

6. Dependencias locales:
Las dependencias locales son los paquetes que se instalan directamente dentro de un proyecto, en la carpeta llamada node_modules. Estas dependencias solo afectan al proyecto en el que se

instalaron y se registran en el archivo package.json, lo que permite que cualquier persona que clone el proyecto pueda instalar las mismas dependencias con un solo comando.

7. Dependencias de desarrollo:
Las dependencias de desarrollo son aquellas que se utilizan únicamente durante la etapa de creación del proyecto, pero no son necesarias cuando la aplicación se ejecuta en producción. Ejemplos de este tipo de dependencias son las herramientas de pruebas, compilación o verificación de código, como Jest, Vite o ESLint. Estas dependencias se instalan normalmente con el comando npm install --save-dev.

8. Dependencias globales:
Las dependencias globales son paquetes que se instalan en todo el sistema operativo y no en un proyecto específico. Esto permite utilizarlas desde cualquier carpeta o terminal sin necesidad de instalarlas nuevamente en cada proyecto. Se suelen instalar con el comando npm install -g nombre, y son útiles para herramientas que se ejecutan en la línea de comandos, como nodemon o typescript.

9. Archivo package.json:
El archivo package.json es un documento esencial que describe la información básica del proyecto y las dependencias que utiliza. Incluye apartados como el nombre del proyecto, su versión, descripción, scripts, dependencias y dependencias de desarrollo. Su principal utilidad es permitir que otros desarrolladores puedan entender el proyecto y reinstalar fácilmente todas las librerías necesarias ejecutando el comando npm install.

10. Archivo package-lock.json:
El archivo package-lock.json se crea automáticamente cuando se instalan dependencias con npm. Su función es registrar las versiones exactas de todos los paquetes y sus subdependencias utilizadas en el proyecto. De esta forma, se garantiza que cualquier persona que instale el proyecto obtenga las mismas versiones y evite errores de compatibilidad entre diferentes entornos.

11. Carpeta node_modules:
La carpeta node_modules es donde npm almacena todas las dependencias instaladas de un proyecto, junto con las dependencias que cada una de ellas necesita. Esta carpeta contiene el código fuente de todas las librerías necesarias para que la aplicación funcione correctamente. Aunque puede ocupar mucho espacio, es fundamental para el funcionamiento del proyecto durante el desarrollo.

CHEAT SHEET:

Proceso	Comando / Acción	Descripción
Inicializar un proyecto de npm	npm init (o npm init -y para hacerlo rápido)	Crea el archivo package.json, donde se registran los datos del proyecto y sus dependencias.
Instalar dependencias locales	npm install nombre_paquete	Instala una librería dentro del proyecto (se guarda en node_modules y se agrega a dependencies del package.json).
Instalar dependencias de desarrollo	npm install nombre_paquete --save-dev	Instala herramientas necesarias solo durante el desarrollo, como linters o compiladores. Se guardan en devDependencies.
Instalar dependencias globales	npm install -g nombre_paquete	Instala un paquete disponible en todo el sistema. Se puede usar desde cualquier terminal.
Visualizar dependencias instaladas	npm list (o npm list --depth=0 para solo ver el nivel principal)	Muestra una lista de todos los paquetes instalados y sus versiones.
Instalar una versión específica de un paquete	npm install nombre_paquete@versión Ej: npm install express@4.18.2	Instala una versión concreta de una dependencia. Útil para mantener compatibilidad.
Crear un comando en tu proyecto (scripts)	En el archivo package.json agrega dentro de "scripts": "start": "node index.js"	Permite ejecutar comandos personalizados con npm run nombre_script (ej: npm run start).
Actualizar dependencias	npm update	Actualiza todos los paquetes instalados a sus versiones más recientes compatibles según el package.json.
Eliminar paquetes	npm uninstall nombre_paquete	Elimina un paquete del proyecto y lo borra del package.json.
Actualizar carpeta node_modules	rm -rf node_modules && npm install (en Windows: rmdir /s /q node_modules && npm install)	Borra todas las dependencias instaladas y las vuelve a instalar según el package.json.

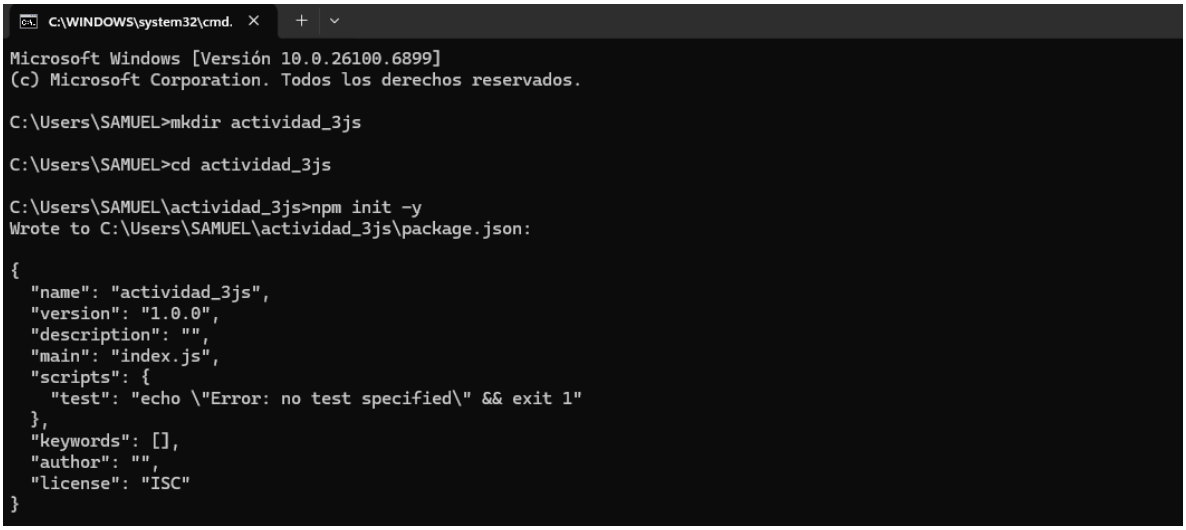
ACTIVIDAD 3

Paso a paso

1. Inicialización de un proyecto nmp

Primero, crea y accede al directorio de tu proyecto:

Inicializa el proyecto con npm:

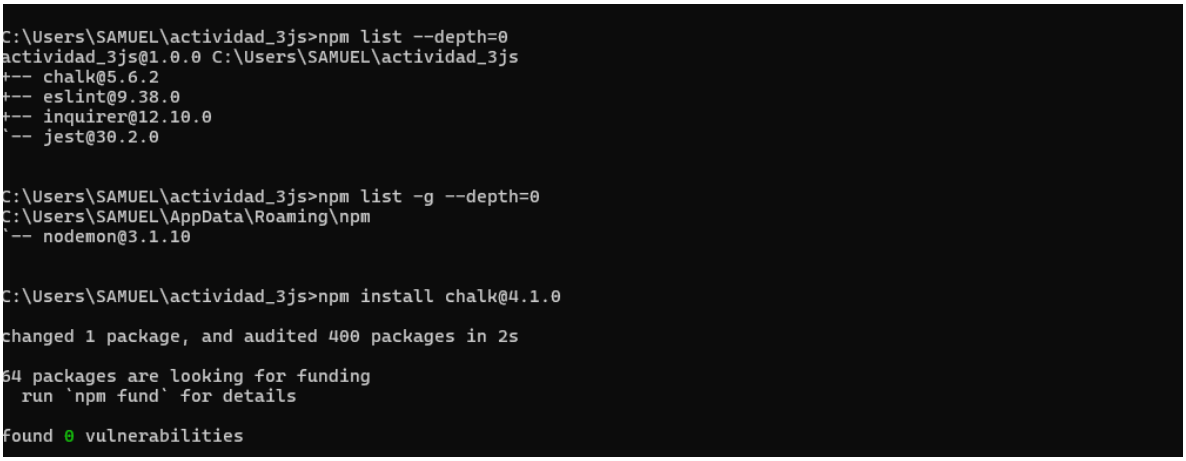


2. instalación de dependencias

Instalar Dependencias Locales, Instalar Dependencias de Desarrollo y Instalar Dependencias Globales

3. Gestión de Paquetes

Visualizar Paquetes Instalados e Instalar una Versión Específica de un Paquete



ACTIVIDAD 4

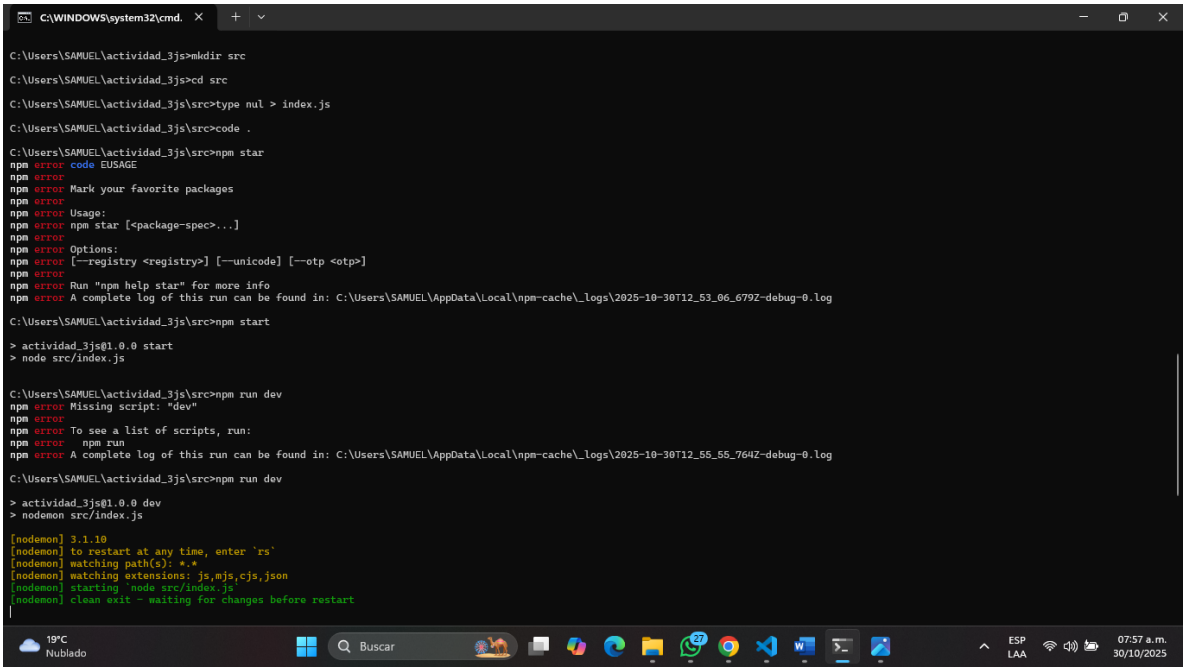
4. Scripts y Actualizaciones

Crear la Estructura de Directorios del Proyecto, Organiza tu proyecto creando una carpeta

Src para el código fuente:

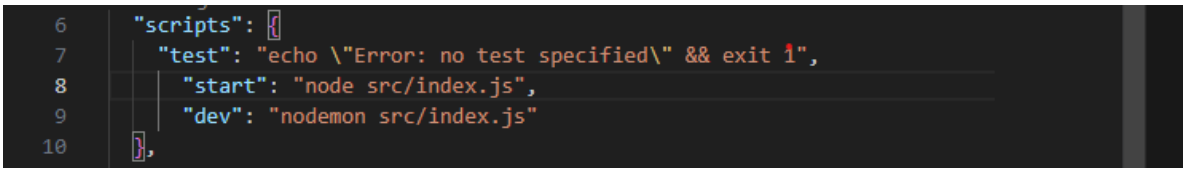
Ahora, puedes ejecutar tu aplicación con: npm star

Se puede ejecutar la aplcacion con npm run: npm run dev



Crear un Comando (Script) en el Proyecto

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



Actualizar independencias

- Verificar qué dependencias están desactualizadas: npm outdated
- Actualizar todas las dependencias locales: npm update
-
- Actualizar una dependencia específica a la última versión compatible: npm install chalk@latest
- Para actualizar a la última versión absoluta (rompiendo los rangos), puedes usar la herramienta [npm-check-updates (ncu)]:

```
C:\Users\SAMUEL\actividad_3js\src>cd C:\Users\SAMUEL\actividad_3js

C:\Users\SAMUEL\actividad_3js>npm outdated
Package  Current  Wanted  Latest  Location            Depended by
chalk      4.1.0   4.1.2   5.6.2  node_modules/chalk  actividad_3js

C:\Users\SAMUEL\actividad_3js>npm update

added 22 packages, removed 26 packages, changed 5 packages, and audited 396 packages in 22s

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

found 0 vulnerabilities

C:\Users\SAMUEL\actividad_3js>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\SAMUEL\actividad_3js>
```

```
C:\Users\SAMUEL\actividad_3js>npm install -g npm-check-updates

added 1 package in 3s

C:\Users\SAMUEL\actividad_3js>ncu -u
Upgrading C:\Users\SAMUEL\actividad_3js\package.json
[====================] 4/4 100%

All dependencies match the latest package versions :)

C:\Users\SAMUEL\actividad_3js>npm install

up to date, audited 418 packages in 3s

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

found 0 vulnerabilities
```

Actualizar Dependencias Globales

Ver qué paquetes globales están desactualizados: npm outdated -g --depth=0

Actualizar todos los paquetes globales: npm update -g

Actualizar un paquete global específico a la última versión: npm install -g nodemon@latest

```
C:\Users\SAMUEL\actividad_3js>npm outdated -g --depth=0

C:\Users\SAMUEL\actividad_3js>npm update -g

changed 30 packages in 3s

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

C:\Users\SAMUEL\actividad_3js>npm install -g nodemon@latest

changed 29 packages in 2s

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

C:\Users\SAMUEL\actividad_3js>
```

Limpieza y Mantenimiento

Para desinstalar un paquete y eliminarlo del `package.json`: npm uninstall nombre-del-paquete

Si deseas reinstalar todas las dependencias desde cero: rm -rf node_modules npm install

```
C:\Users\SAMUEL\actividad_3js>npm uninstall chalk

removed 1 package, and audited 417 packages in 3s

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

found 0 vulnerabilities

C:\Users\SAMUEL\actividad_3js>rmdir  /s /q node_modules

C:\Users\SAMUEL\actividad_3js>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 a
sync 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 19s

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

found 0 vulnerabilities
```