**NODE.JS**

**Módulos.**

Uno de los aspectos más poderosos de Node.js es su sistema de módulos. Los módulos son unidades independientes de funcionalidad encapsulada y reutilizables en distintos proyectos, a través de npm.

**Sistemas de módulos dentro de Node.**

Estos son la forma en la que node organiza y gestiona los paquetes de código reutilizable. Utiliza **npm** determinadamente, es una colección de bibliotecas de código abierto disponibles para todos aquellos que quieran instalar y usar en proyectos.

Conceptos clave para el sistema de módulos de node:

**require()**: Se utiliza para importar módulos en un archivo de JavaScript.

**Npm**: Gestor de paquetes que permite instalar, compartir y administrar dependencias en proyectos de node.

**Package.json:** Archivo de metadatos de un proyecto node que incluye información de el mismo, como el nombre, versión, dependencia, etc. Administra las dependencias y scripts del proyecto.

**Npm Registry**: Almacena paquetes npm, publicados por desarrolladores para que otros los instalen y utilicen en proyectos.

**Scoped package:** Paquetes npm asociados con su alcance especifico, ayuda a evitar el conflicto de nombres y facilita la operación del paquete.

**Síncrono y Asíncrono (bloqueante y no bloqueante)**

Son el modo en que se ejecutan ciertas operaciones dentro del flujo de un programa.

**Síncrono**: donde las operaciones son ejecutadas en secuencia una tras otra de forma ordenada. Se completa una operación para continuar con la siguiente, es decir, el hilo de ejecución se detiene hasta que la actual esté finalizada.

**Asíncrono**: donde las operaciones pueden iniciarse y completarse en momentos diferentes, sin tener que esperar que una operación termine para poder continuar, permitiendo que el programa continue en ejecución mientras ciertas operaciones esperan a terminar su ejecución.

**FS** (File System): Es uno de los módulos nativos de Node que **permite interactuar en el sistema de archivos del servidor**. Puedes leer, escribir y manipular los archivos con dicho modulo.

Código bloqueante.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media En la línea 5 se declara la variable data como el lector del archivo de forma síncrona, ya que **por default node es asíncrono**.

Código no bloqueante.

Captura de pantalla de un celular

Descripción generada automáticamente

**Single thread y Multi threading**

Threading: Es la capacidad de un programa de realizar múltiples tareas simultáneamente. Es una secuencia de ejecución dentro de un procesos.

Multithreading: Es el uso de múltiples threads, entendiendo que un programa puede dividir su trabajo en varias secuencias de ejecución independientes ejecutadas en el mismo espacio del proceso. **Node no soporta el multithreading**

**Event Loop (Single thread).**

Es el hilo único que maneja Node al momento de ejecutar código de forma simultánea. Se realiza un registro de **callbacks** al momento de realizar operaciones que requieren de un tiempo de espera (operaciones bloqueantes) (Lectura de archivos, conexiones a BD, etc.).

Diagrama

Descripción generada automáticamente

Las callbacks se encuentran en forma de pila de operaciones en espera, conforme finalicen se incorporan nuevamente al hilo de node y devuelven sus respuestas solicitadas.

Texto

Descripción generada automáticamente En este ejemplo vemos a los console que fluyen normalmente en el hilo de node, y en las líneas 3,7 y 11 encontramos los bloqueantes de setTimeout, estos ingresan al callback de peticiones, aunque tarden 0 segundos.

Dando el siguiente resultado Texto

Descripción generada automáticamente

**Etapa de los módulos.**

**NPM.**

Es el gestor de paquetes oficial para Node.js. Es una herramienta fundamental para cualquier desarrollador que trabaje con Node.js, ya que permite gestionar dependencias, compartir código y configurar scripts de desarrollo.

Instala, actualiza y desinstala paquetes (bibliotecas o módulos).

Comando npm init = Inicializar un proyecto Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

O con “npm init -y” para crear el package.json de forma predeterminada.

Ahora ejecutamos la instalación del paquete “responsimple” en consola. Al igual que cualquier otro paquete que queramos instalar dentro de la carpeta de módulos, podríamos hacerlo de la siguiente forma:

Texto

Descripción generada automáticamente

Esto haciendo la instalación más fácil y dándonos siempre a entender que dependencias tenemos instaladas, listados en el “package.json”, así como mostrándonos las distintas carpetas que ahora tenemos instaladas dentro de node\_modules

**Node\_modules**

Esta carpeta en un proyecto de Node.js es esencial para su funcionamiento, contiene las dependencias necesarias para ejecución del código: como el **almacenamiento de dependencias, estructura jerárquica, ejecución del proyecto y compatibilidad y versionado.**

**Archivos package.json y package-lock.json**

* **package.json**: Este archivo lista todas las dependencias de tu proyecto junto con sus versiones. Al ejecutar npm install, npm usa este archivo para determinar qué descargar y colocar en node\_modules.
* **package-lock.json o yarn.lock**: bloquea las versiones exactas de las dependencias instaladas en tu proyecto. Esto garantiza que cualquier persona que instale el proyecto obtenga las mismas versiones de las dependencias, evitando así problemas de compatibilidad y errores inesperados debidos a actualizaciones de dependencias.

Texto

Descripción generada automáticamente

Es importante, si se trabaja con algún controlador de versiones, **colocar en el archivo “git ignore” la carpeta node\_modules**.

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente con confianza media

Para instalar una versión en particular de lo que sea (ejemplo; react) utilizamos la siguiente sentencia y versión. Primero removemos “uninstall” o “un” para eliminar las dependencias actuales de React y ejecutamos las de la versión deseada.

Texto

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora ponemos la versión 18.1.0

Texto

Descripción generada automáticamente

Utilizando el comando “npm update react” se actualizará a las nuevas versiones de dependencias en react. **Es importarte ver este cambio reflejado en el package.json de la dependencia de React, ya que en esa carpeta marca una versión nueva, y en package.json de la carpeta “npm” continua con la versión instalada anteriormente, se necesita cambiar el numero de versión ahí también**.

Podemos crear un npm install de forma global (-g).

**El comando serve** se utiliza para lanzar un servidor estático que sirve archivos HTML, CSS y JavaScript en aplicaciones web. Útil para el desarrollo y pruebas locales.

Se puede instalar de forma **global (npm install -g serve)** y estar disponible en cualquier proyecto, o bien de manera **local (npm install serve --save-dev)** para un proyecto en específico.

Una vez instalado de forma global se puede ejecutar el serve desde cualquier punto de las dependencias de node y desde cualquier carpeta.

Texto

Descripción generada automáticamente

Dentro de los end points se pueden mostrar en una vista de navegador.

**Configuración de ejecución de scripts.**

Dentro de powershell, ejecutado como administrador ingresamos el comando siguiente para el listado de directivas de ejecución: Get-ExecutionPolicy -List. Una vez aareciendo esto ejecutamos el siguiente código para des inhabilitar LocalMachine: Set-ExecutionPolicy Unrestricted y en seguida seguir las instrucciones de las flags.

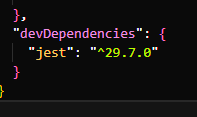
**Tipos de banderas de dependencias**

Las dependencias son módulos externos que hacen funcionar un proyecto, se gestionan en el package.json. La sección “Dependencies” del package.json son **paquetes de dependencias de producción para las aplicaciones**.

Texto

Descripción generada automáticamente

Las “devDependences” o **dependencias de desarrollo** son necesarias durante el desarrollo del proyecto, como herramientas de prueba, testeo, de construcción, entre otras. Utilizando las sentencia **npm install xpaquete –save-dev** y especificado en el archivo package.json en una sección de devdependecies.



Así mismo, la bandera llamada **“--production”** simplemente se encarga de instalar las dependencias especificadas en “dependences” de package.json, ignorando las “devDependences”.

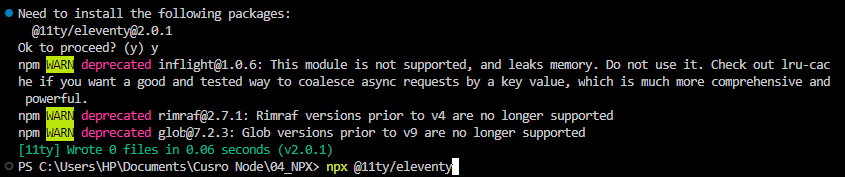
Para ver mas a detalle: <https://jonmircha.com/nodejs#aprende-m%C3%A1s>

**Comando NPX.**

Empleada a partir de la versión 5.2.0 de npm. Se utiliza para ejecutar paquetes de Node.js sin necesidad de instalarlos globalmente. Eficiente para ejecutar los scripts o comandos proporcionados por paquetes de Node.js sin agregar a la lista de dependencias.

Al ejecutar **npx** con un paquete especificado, al no estar instalado local o globalmente, se instalará, lo ejecutará y lo borrará.

**Ejemplos con librería 11ty**. Iniciando por crear la carpeta e instalar npm para la creación del archivo package.json. Después se tiene que hacer la **ejecución** de 11ty en tiempo de desarrollo dentro de la carpeta creada.



El concepto de **Eleventy** es generar archivos estáticos para JavaScript. Dichos generadores crean archivos HTML estáticos desde las plantillas de datos ára hacer sitios más rápidos y seguros, al no haber BD activa o procesos de servidor en tiempo real. Más en: <https://chatgpt.com/share/278ecc45-fa37-41e5-b886-587e7635b63a>

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media Ejecutamos el comando anterior en la misma carpeta. Ahora nos da una carpeta con el index.html con el siguiente resultado.

Texto

Descripción generada automáticamente

El comando en si va al registro de Eleventy en npm y ejecuta el comando sin tenerlo instalado como dependencia ni como global en el proyecto.

**Scripts Personalizados.**

Son relevantes en diferentes herramientas (ejemplo Eleventy). Son comandos específicos definidos en el package del proyecto, permiten automatizar tareas comunes como ejecución de pruebas, compilación de código, inicio de un servidor, entre otras.

Se definen en la sección de scripts del archivo package, aquí vemos ejemplos con 11ty y su archivo markdown.

Texto

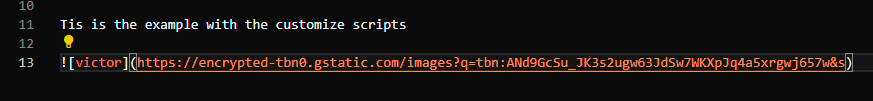
Descripción generada automáticamente Texto

Descripción generada automáticamente Ahora podemos ver la ejecución del script build (para este se tiene que ocupar la palabra run) que se encarga de iniciar 11ty y crear la actualización del markdown.

Texto

Descripción generada automáticamente Ahora estamos corriendo al script start el cual nos levanta un servidor con 11ty, de igual forma solamente vuelve a utilizar el paquete 11ty de forma temporal, y si ingresamos al localhost nos da la página del index.html.

Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente Si agregamos una imagen como esto…  y damos guardar en **md** podemos ver que el server sigue corriendo en tiempo real y aparece en la página.