Node

Lo primero que vamos a hacer es crear una carpeta de nombre “Proyecto-Node” (donde trabajaremos todo el proyecto en Node). Una vez creada, la abriremos desde el Visual Studio Code, y debemos acceder a su ubicación desde la consola de comandos. Para esto, desde el editor de código, vamos a poner la opción “Terminal” -> “Nuevo Terminal”. Esto nos abrirá la consola con la ruta de nuestro proyecto.

Ahora vamos a iniciar nuestro proyecto. Para esto, en la consola, pondremos el siguiente comando:

```
npm init
```

¿QUÉ DATOS ME PIDEN?

Nombre del paquete / módulo: es el nombre que le vamos a poner a nuestro proyecto. Podemos dejar el que nos muestra por defecto (nombre de carpeta).

Versión: nos pregunta cuál va a ser la versión inicial del proyecto.

Descripción: de acá se extrae la información relevante del paquete que estamos creando.

Entry point: es el archivo desde el cual se ejecuta nuestro proyecto una vez que se importa / requiere el paquete.

Test command: es el comando de ejecución de pruebas, ya que los paquetes de node podemos compartirlos con la comunidad y estas pruebas se utilizan para verificar que no haya ningún error en el código que estamos subiendo. Por ahora podemos dejarlo con el valor por defecto (vacío).

Git repository: una URL de git, un repositorio donde se guardaría nuestro código si es que tenemos.

Keywords: son términos clave de nuestro proyecto, para facilitar la búsqueda dentro de la comunidad de módulos de NPM.

Author: la persona u organización que está creando el paquete.

Licencia: son los permisos de uso que le doy a mi paquete.

Una vez que terminemos, nos va a mostrar la información que ingresamos en un formato JSON y nos va a preguntar si esa información está bien. Si está todo bien, podemos escribir yes y apretar enter.

Ahora, si revisamos la carpeta del proyecto, vamos a ver que tenemos un archivo llamado package.json. Este archivo es muy importante para los proyectos de Node.js, ya que guarda mucha información sobre la configuración del proyecto (qué dependencias son las necesarias, quién desarrolló el proyecto, a quien comunicarse cuando algo falle, que scripts se utilizan para ejecutar el proyecto, etc).

Teniendo este mencionado archivo, podemos asumir que tenemos un proyecto de Node inicial, pero vacío. De acá en adelante podemos empezar a crear nuestros propios archivos del proyecto, instalar otros paquetes con npm, etc.

Instalando módulos en nuestro proyecto

Utilizando NPM podemos instalar módulos y utilizarlos en nuestros proyectos. Para lograr esto, tenemos que ejecutar desde la consola el comando “npm install” y el nombre del módulo que queremos instalar.

Por ejemplo vamos a instalar un módulo muy conocido llamado cowsay. Este paquete es muy popular para personas que están aprendiendo Node.js y NPM. Simplemente muestra un mensaje (que podemos cambiar) y una imagen por consola. Este paquete no va a tener ninguna importancia en nuestro proyecto, simplemente es para que veamos cómo descargar un paquete y utilizarlo. En la consola, ponemos lo siguiente:

```
npm install -g cowsay
```

Después de un tiempo que tarda la descarga, ya contamos con ese paquete en nuestro proyecto. Para verificar esto, podemos ir al archivo “package.json” y dentro del apartado “dependencies”, nos debe aparecer el paquete recién descargado con su versión.

Y como seguramente se habrán dado cuenta, todos los paquetes que instalemos en nuestro proyecto aparecerán ahí, en la sección de dependencias.

Cuando instalamos un paquete en nuestro proyecto, también se crea una carpeta de nombre ‘’node_modules’’. Esta carpeta es muy importante, ya que es donde se almacenan todas las dependencias y librerías que utilizamos para que funcione correctamente nuestro proyecto.

Suele ser una carpeta de un tamaño muy grande, por lo cual no es recomendable subirla, ya sea acá en la plataforma cuando entreguemos un desafío, o si subimos nuestro proyecto a un repositorio como github. Si hacemos esto último y otra persona descarga nuestro proyecto en su propia computadora, con sólo poner el comando npm install se le creará también la carpeta node_modules con todas las dependencias para que funcione.

¿Cómo sucede esto? La respuesta está en el archivo “package.json”, que contiene las dependencias. Por lo cual este archivo sí es conveniente subirlo siempre que compartamos nuestro proyecto.

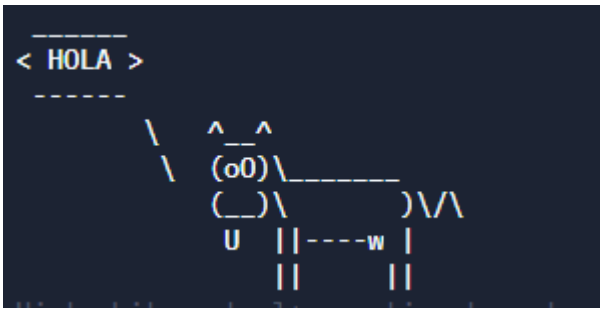
Ahora vamos a utilizar el paquete. Si nos fijamos en la documentación oficial de cowsay, podemos utilizarlo como un módulo. Lo primero que haremos será crear un archivo de nombre “index.js”. Luego, en la primera línea de código de este archivo, debemos requerir el paquete que vayamos a utilizar, de la siguiente manera:

const cowsay = require("cowsay");

De esa manera, mediante la función require, ponemos el nombre del paquete que acabamos de descargar, y lo guardamos en una constante (también podría ser una variable) del mismo nombre que tiene el paquete. Lo único que queda es copiar desde la documentación oficial cómo utilizarlo, y pegarlo en nuestro código.

```
const cowsay = require("cowsay");

console.log(cowsay.say({
    text : "HOLA",
    e : "oO",
    T : "U "
}));
```



Módulos internos

Ya vimos cómo instalar un módulo externo a través del gestor de paquetes de Node. Por supuesto que existen muchos más, ya que NPM es una enorme comunidad.

Pero también podemos crear nosotros mismos un módulo, que después requeriremos desde algún archivo de nuestro proyecto. Y si ese módulo que creamos ayuda a resolver muchos problemas, ¡lo podemos subir a NPM para que se lo descargue la comunidad!

Entonces… ¿Cómo hacemos para crear un módulo de manera interna?

Vamos a crear un módulo sencillo, que nos permita pasarle a una función un precio, y que esta me lo devuelva con el IVA incluido.

Entonces, vamos a crear un archivo (que será nuestro módulo) de nombre **“calcular.js”**. Aquí dentro, haremos la lógica para lograr lo dicho anteriormente.

```
// guardamos en una constante el % de IVA
const IVA = 0.21;

// la función recibe por parámetro un precio y le sumamos el IVA
const calcularPrecio = (precio) => precio + (precio * IVA);

// finalmente, exportamos la función
module.exports = calcularPrecio;
```

En el módulo que acabamos de escribir, lo que hicimos fue guardar el porcentaje del IVA en una constante, hacer una función flecha para calcular la lógica del precio con el iva, y finalmente, exportamos la

función mediante una función reservada del lenguaje, llamada module.exports

Como se estarán dando cuenta, así como exportamos esta función, debemos ahora importarla en el archivo principal. Vamos a hacerlo.

```
// importamos nuestro módulo y lo guardamos en una variable
const obtenerPrecio = require('./calcular');

// lo que importamos es la función que exportamos en el anterior archivo
// la ejecutamos y obtenemos el precio final con IVA
// mostrando en la consola el precio
console.log(obtenerPrecio(200));
```

El ejemplo anterior funciona perfectamente. Ahora, imaginemos la siguiente situación: Queremos exportar otra función, que, además de calcular el precio con IVA, nos descuente un 10% para clientes especiales. ¿Cómo hacemos para exportar más de una función?

Podemos exportarlas como un objeto, en la que cada función actúe como una propiedad del mismo.

```
const IVA = 0.21;
const descuento = 0.10

module.exports = {
calcularPrecioIva: (precio) => precio + (precio * IVA), //No olvidar la coma
```

```
calcularPrecioDesc: (precio) => precio + (precio * IVA) - (precio * descuento)
};
```

Ahora en el index, para mostrarla en consola, llamamos a la constante obtenerPrecio y luego accedemos al objeto (que es una función) deseado.

```
const obtenerPrecio = require('./calcular');

console.log(obtenerPrecio.calcularPrecioIva(200));
console.log(obtenerPrecio.calcularPrecioDesc(400));
```