

# BASES DE NODE.JS

Wilson Aguilar  
PLATAFORMAS WEB

30 de abril de 2020

## 1. Documentación

### 1.1. Requerir paquetes

NodeJs tiene varias librerías o módulos que podemos utilizar que nos otorgan varias funcionalidades que pueden ser de ayuda en nuestro programa, además podemos instalar módulos adicionales por si necesitamos de alguna funcionalidad extra.

Para ello utilizamos la función `require(modulo)` donde el modulo es el nombre del paquete que vamos a utilizar.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code is as follows:

```
1  const { writeFile, promises: fs } =  
    require('fs');  
2  const colors = require('colors');
```

Figura 1: require en NodeJs

### 1.2. Importar archivos al proyecto

A medida de que la aplicación que desarrollamos se va haciendo mas grande, es necesario dividir nuestro código en módulos para poder reutilizar

partes del código que ya hayamos hecho. Por lo general un módulo esta conformado por varias funciones, para que estas funciones puedan ser accedidas desde otro archivo primero hay que exportarlas.

Utilizamos el objeto `module.exports` que es un objeto de NodeJs que nos indica que funciones van a poder ser accedidas desde otro archivo.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code shown is a single line: `1 module.exports = { createFile, listarTabla }`.

```
1 module.exports = { createFile, listarTabla }
```

Figura 2: export modules

### 1.3. Recibir información de la línea de comandos

La lógica principal de la aplicación se encuentra en el archivo `app.js` que es el encargado de recibir el comando que enviamos y en base a ese comando ejecutar alguna acción.

A screenshot of a code editor with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The code shows the logic for receiving and processing command-line arguments using `argv`. It includes variable declarations for `comando`, `base`, and `limite`, followed by a `switch` statement that handles 'listar' and 'crear' commands, logging results or errors. Line numbers 1 through 16 are visible on the left side of the code block.

```
1 let comando = argv._[0];
2 let base = argv.base
3 let limite = argv.limite
4
5 switch (comando) {
6   case 'listar':
7     console.log(listarTabla(base, limite))
8     break;
9   case 'crear':
10    createFile(base, limite)
11      .then(msj => console.log(msj))
12      .catch(err => console.log(err.message))
13    break;
14   default:
15     break;
16 }
```

Figura 3: `app.js` - Encargado de recibir los comandos

## 1.4. Manejo de paquetes

npm es una herramienta de NodeJs que nos ayuda a administrar nuestro proyecto y los módulos que vamos necesitando.

Para iniciar un nuevo proyecto de npm ejecutamos `npm init`, después nos preguntará algunas opciones de nuestro proyecto y al final se creará un archivo llamado `package.json` con la descripción de nuestro proyecto. También podemos ejecutar `npm init -y` si queremos saltarnos las preguntas.

Con el comando `npm install --save <nombre-paquete>`, podremos agregar nuevos módulos que a nuestro proyecto.

Con el comando `npm install --save-dev <nombre-paquete>` instalaremos el módulo como dependencia de desarrollo, esto nos indica que ese módulo no es necesario para que la aplicación funcione, sino que ayuda al desarrollo de la misma.

Para eliminar un paquete usamos `npm uninstall <nombre-paquete>`.

Cabe resaltar que cualquier modificación que hagamos con npm afectará al archivo `package.json` ya que ahí es donde se guarda toda la información de nuestro proyecto.

## 1.5. Yargs

Yargs es un módulo de npm que nos ayuda al desarrollo de aplicaciones de consola, nos ayuda a manejar de una forma mucho más fácil los argumentos, opciones, comandos, entre otras opciones.

Para instalar yargs utilizamos el comando `npm install --save yargs`

## 1.6. Configuración YARGS

Yargs tiene varias funciones que nos permiten agregar comandos, opciones y entre otras. Esto nos permite desarrollar aplicaciones de consola mucho más fácil, sin tener que preocuparnos por las validaciones.

Para usar yargs usamos `require('yargs')` y justo después de eso podemos empezar a agregarle opciones. El resultado nos devuelve el objeto ya configurado donde solo debemos aplicar la lógica para cada comando.

La función `command(comando, descripcion, opciones)` nos permite crear un nuevo comando y recibe como argumentos el nombre del comando, su descripción y un objeto con las opciones que tendrá el comando.



Figura 4: Configuración básica de yargs

## 1.7. Optimización de Yargs

Para optimizar el código se movió toda la configuración de yargs a un archivo separado, creacion un objeto con las configuraciones para no tener que volver a escribir lo mismo y lo exportamos para que podamos usarlo en el archivo principal app.js




Figura 5: Optimización en yargs

## 1.8. Colores en la Consola

Para poder agregar colores a nuestra aplicacion instalamos el módulo colors de npm con `npm install --save colors`.

Ahora para utilizarlo hacemos un `require('colors')` en nuestro archivo mutiplicar.js, despues para agregar colores es tan facil como ejecutar un metodo en cualquier string como `"hola mundo".green`



```
1  const colors = require('colors');
2
3  const listarTabla = (base, limite) => {
4    if (isNaN(base) || isNaN(limite)) {
5      throw new Error(
6        `Los argumentos enviados deben ser un numero.`.red)
7    }
8    let data = `
9      =====
10     TABLA DEL ${base}
11     =====\n`
12    .green
13    for (let i = 1; i <= limite; i++) {
14      data += `${base} * ${i} = ${base * i}\n`;
15    }
16
17    return data
18  }
```

Figura 6: Uso colors

## 2. Github

### 2.1. Publicar proyecto en GitHub

Para subir nuestro proyecto a github debemos crear uno vacío. Después lo agregamos como repositorio remoto con el comando

```
git remote add <nombre> <url>
```

Una vez configurado el repositorio agregamos todos nuestros archivos con

```
git add -A
```

Después hacemos un commit con:

```
git commit -m "primer commit"
```

Por último subimos los cambios al repositorio remoto con el comando:

```
git push -u origin master
```

### 2.2. Repositorio

El repositorio donde se encuentra el código es el siguiente:

<https://github.com/WilsonAG/plataformas-web.git>