**NPM**

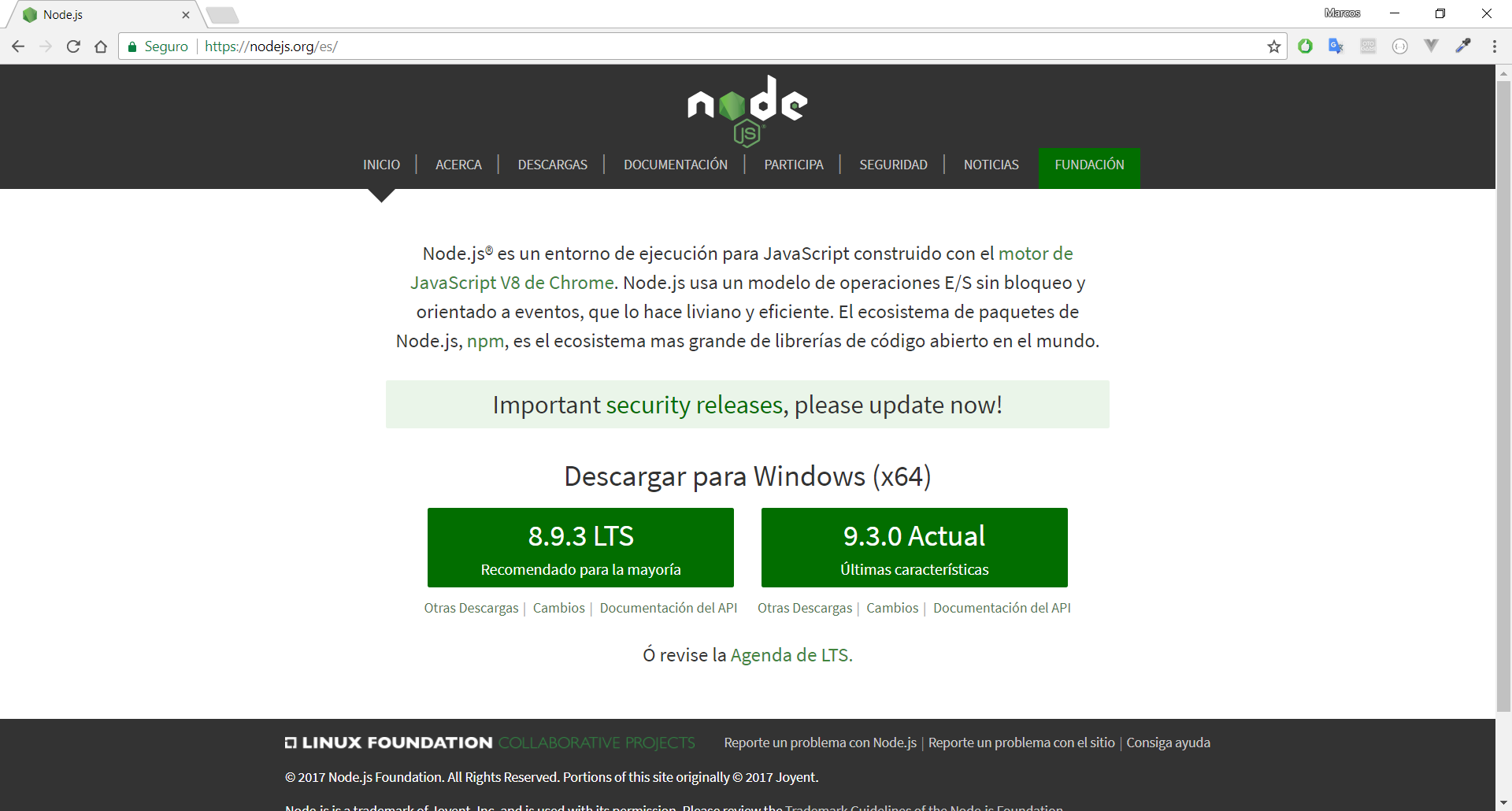
NPM o Node Package Manager es un gestor de paquetes, el cual facilita enormemente el manejo de paquetes en JavaScript. Con NPM es realmente fácil instalar paquetes simplemente escribiendo unos cuantos comandos básicos, los cuales más adelante explicaremos un poco más a detalle. NPM es el manejador de paquetes por defecto para Node.js. ¿Pero qué es Node.js?

***Node.js:*** En palabras resumidas, Node.js es una librería y entorno de ejecución de Entrada/Salida (o I/O), dirigida para eventos. Node es asíncrono y a diferencia de otros lenguajes de programación, se basa en un solo hilo para ejecutar todas sus operaciones. Node.js está basado en el motor V8 de Google, en el cual también se basa el navegador Google Chrome.

Ahora mismo no profundizaremos en Node, ya que hay tiene su propio apartado en esta guía, pero para poder tener NPM instalado en nuestra computadora, primero tendremos que instalar Node.js y automáticamente NPM será instalado.

**¿Cómo instalar Node.js?**

Para instalar Node.js en nuestra computadora, simplemente entramos a la página oficial, la cual es: <https://nodejs.org/es/>

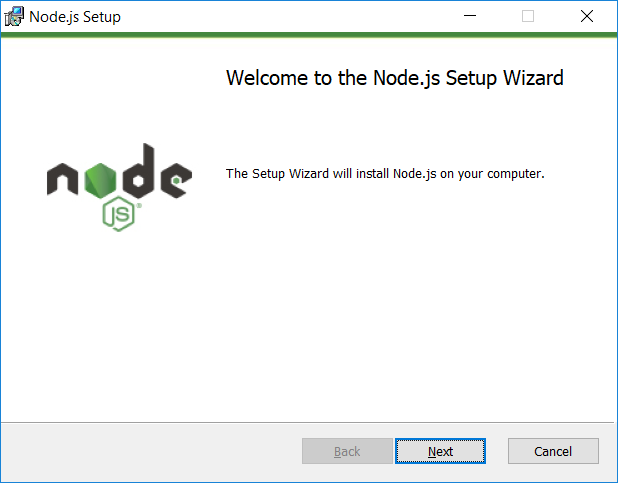


Se nos mostrará la siguiente página. Y aquí nos aparecerán las distintas versiones recomendadas de Node que podemos instalar. Ahora mismo, las versiones que hay son la *8.9.3 LTS* y *9.3.0.* La que recomendamos instalar es la versión LTS.

LTS (Long Term Support), es la versión que contiene todas las funciones de Node que se consideran estables, lo que significa que la versión LTS ha sido “testeada” rigurosamente. Además, esta versión tendrá varios años de soporte y no será descontinuada rápidamente. Esta la recomendada para la mayoría de los usuarios.

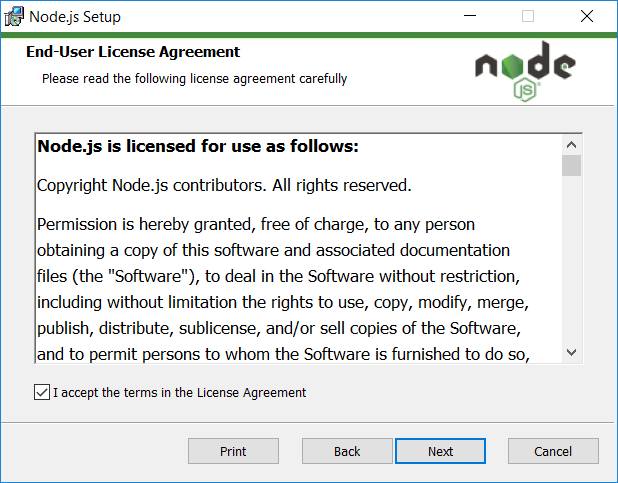
En el caso de la versión 9.3.0 Actual, tendremos acceso a las últimas características añadidas a Node.js, es decir tendremos acceso a las nuevas estandarizaciones que se hayan añadido a JavaScript en esta version, pero hay probabilidad de que alguna característica en particular pueda cambiar ligeramente en la siguiente versión o que contenga algún bug en su código fuente. De cualquiera manera, es improbable que esta versión tenga bug de gran calibre. Así que en caso de que se deseen usar las últimas características, esta es una buena opción.

En nuestro caso, elegiremos la versión LTS. Así que, simplemente la descargaremos e instalaremos según nuestro sistema operativo lo requiera.

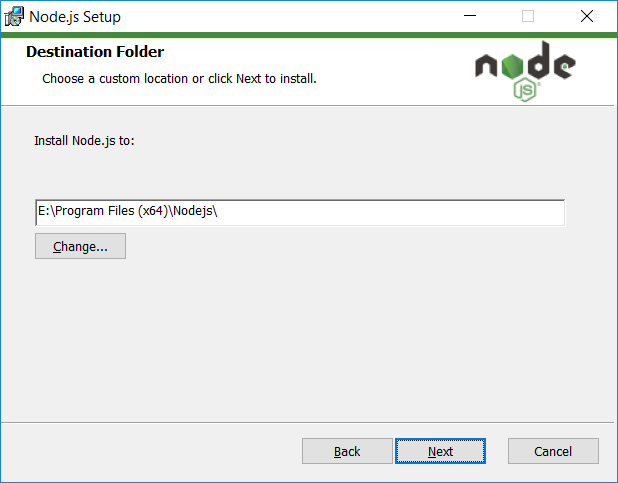
En nuestro caso, lo instalaremos en Windows.

La interfaz gráfica del instalador es sencilla y es básicamente la misma que podemos encontrar en los instaladores de programas para Windows.

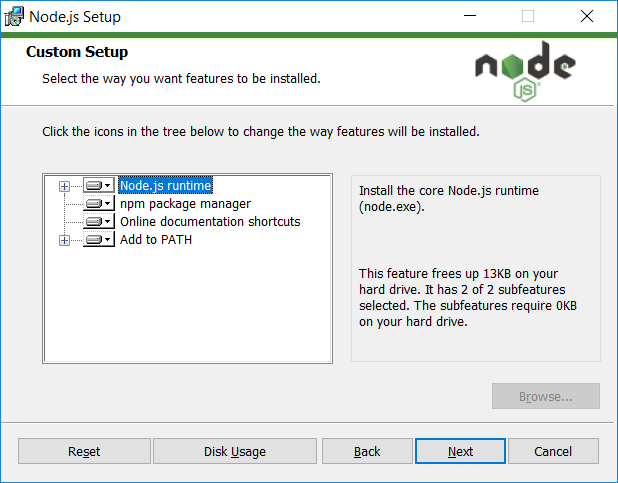
Entonces, sólo tenemos que dar siguiente.



Aceptamos los términos y condiciones del software.

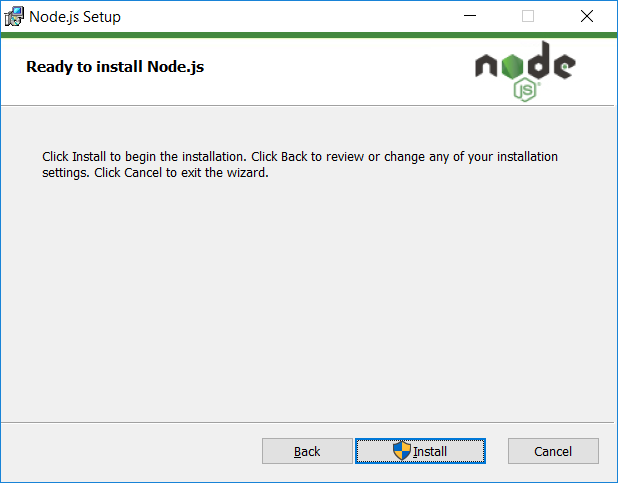


Elegimos el directorio principal en el que se instalará Node.js



Nos aparecerá esta ventana en la cual se nos preguntará sobre qué componentes de Node queremos instalar. Si nos fijamos con detenimiento, uno de estos componentes en NPM.

Simplemente daremos siguiente.



Finalmente nos aparecerá la opción de “Instalar”, damos clic sobre ella, si es necesario, damos privilegios de administrador y esperamos a que termine la instalación y damos clic en “Finalizar”.

La forma de instalación varía, dependiendo de nuestro sistema operativo, pero no supondría un problema instalarlo en ninguno de los sistemas operativos principales ya sea, Windows, MacOS o Ubuntu.

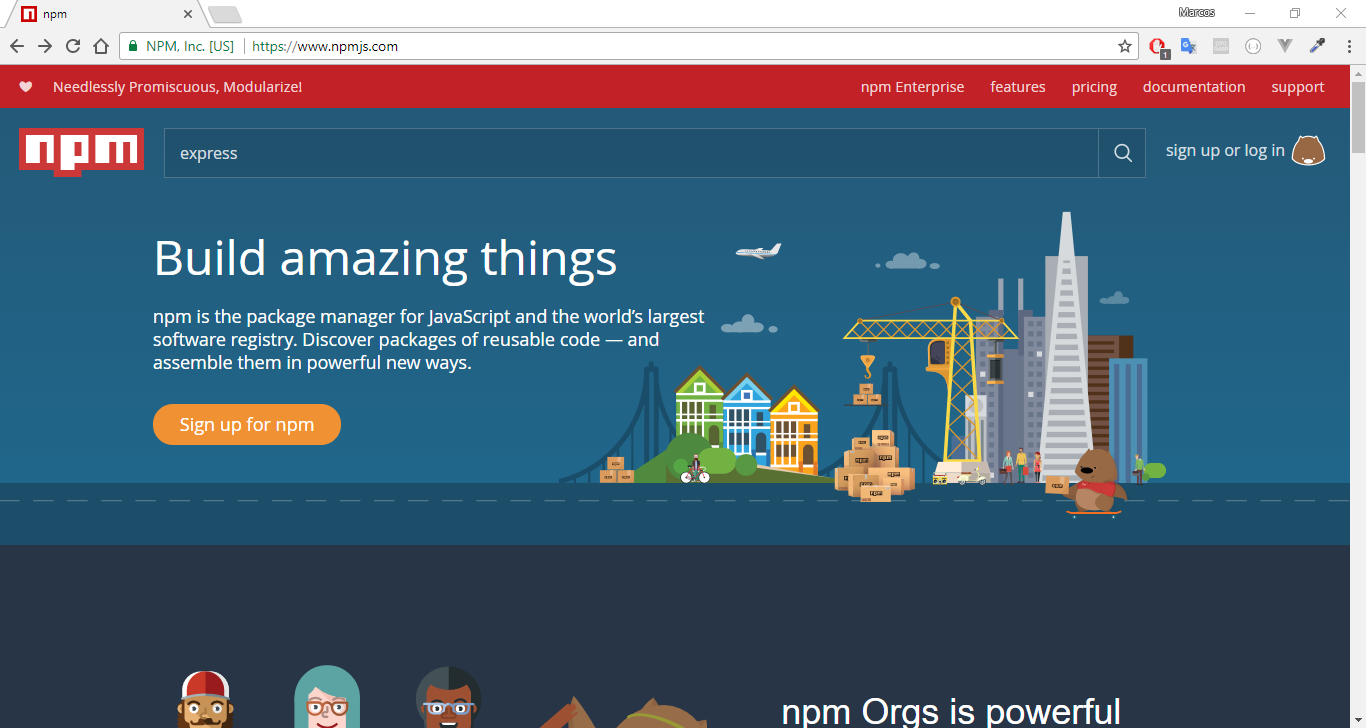
**¿Qué es y cómo funciona NPM?**

Muy bien, después de haber instalado Node, automáticamente se habrá instalado NPM, lo cual significa que ya podremos utilizarlo, pero… ¿qué es y cómo funciona NPM?

NPM nos permite descargar e instalar paquetes y módulos de manera sencilla en nuestro sistema operativo. Básicamente, NPM nos facilita enormemente compartir y reusar código a través de repositorios.

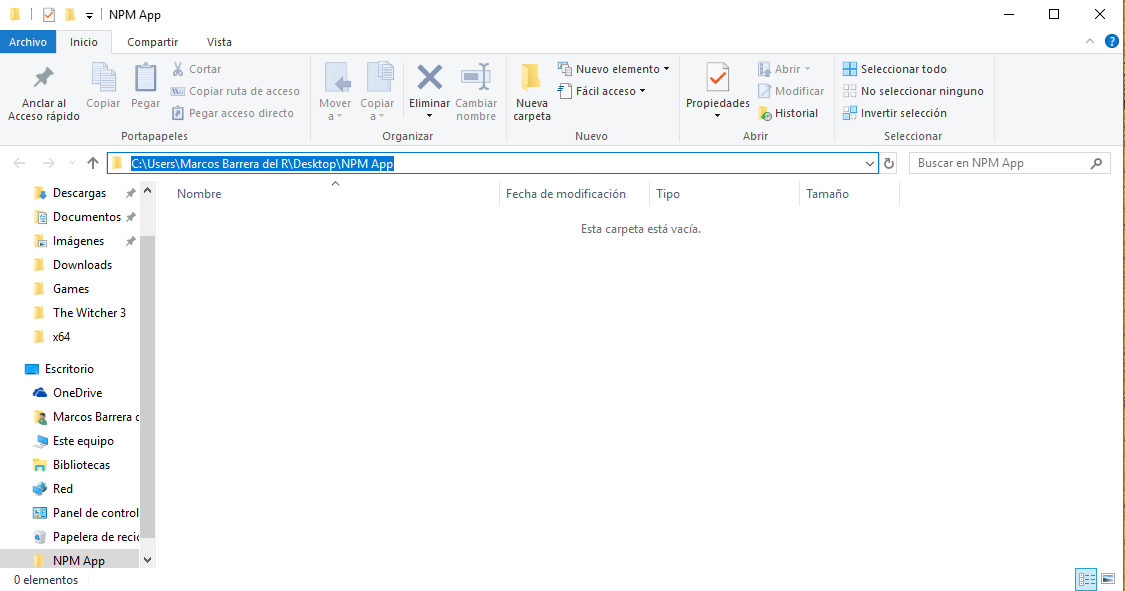
Para poder ver el código que se encuentra disponible en NPM, tendremos que ingresar a la siguiente página web <https://www.npmjs.com/> o simplemente buscar NPM en Google y entrar al primer enlace.

Al ingresar a la página, se mostrará el siguiente contenido.

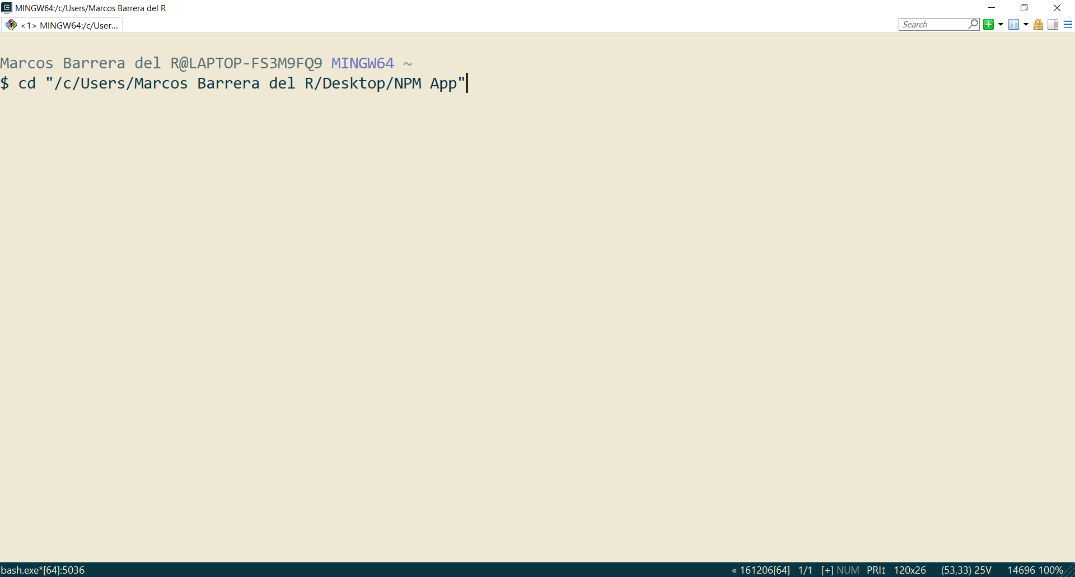


En esta página podremos buscar cualquier módulo que se encuentre disponible en NPM. Para conocer a más detalle la funcionalidad de NPM, se recomienda leer la documentación de NPM.

Como ya comentamos, NPM es una manejador de paquetes, el cual nos permite gestionarlos de una manera sencilla, para evitarnos muchos dolores de cabeza. Entonces, para usar NPM, tendremos que utilizar la terminal de nuestro sistema operativo. CMD, PowerShell o cualquier terminal puede hacer el trabajo. En nuestro caso, utilizaremos la una llamada ConEmu. Además, tendremos que utilizar un editor de texto, para poder configurar algunos archivos necesarios en NPM. Nosotros recomendamos usar Visual Studio Code, el cual se puede descargar desde el siguiente link https://code.visualstudio.com/download

Lo primero que haremos es crear una carpeta llamada en el directorio que queramos NPM App. Después tendremos que ingresar a ella a través de la consola de comandos.

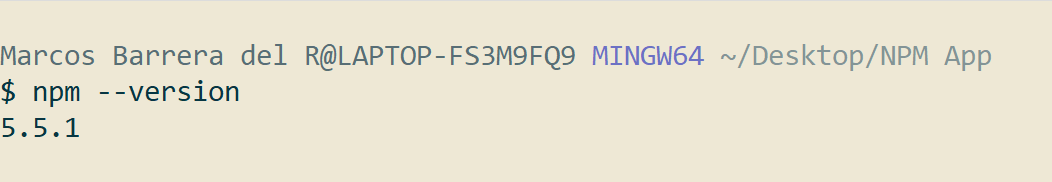
En Windows, podemos copiar simplemente el directorio que nos proporciona el explorador:



Y lo pegamos en nuestra terminal junto con el comando “cd”. O también podemos ir navegando entre carpetas hasta llegar a la carpeta deseada con el mismo comando.

Después de abrir el directorio con nuestra terminal, también lo abriremos con nuestro editor de texto.

El primer comando que utilizaremos es el siguiente: **npm --version**



Con este comando podremos ver la versión que tenemos actualmente instalada en nuestro sistema operativo y para verificar que tenemos NPM instalado correctamente en nuestro sistema.

**Package.json**

Todos los paquetes de NPM contienen un archivo llamado “package.json” el cual usualmente está situado en la carpeta raíz de nuestro proyecto. Este archivo contiene información relevante que nuestro proyecto necesita para funcionar. Este archivo almacena el nombre de nuestro proyecto, su versión, autor, aunque esos datos pueden ser opcionales. La parte más relevante que maneja este archivo son las **dependencias** de nuestro proyecto. Las **dependencias** son todas aquellas librerías de las cuales nuestra aplicación depende, es decir, sin ellas no funcionaría.

Una funcionalidad realmente útil que nos otorga NPM junto con “package.json” es crear script personalizados para nuestra aplicación.

A continuación, presentaremos los distintos comandos básicos que nos ofrece NPM y los que utilizaremos en secciones más avanzadas de la guía.

Ejemplo de un archivo package.json:

{

"name": "npmapp",

"version": "1.0.0",

"description": "Mi descripción",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Marcos Barrera",

"license": "MIT"

}

**npm init:** Este comando nos permite crear desde cero un archivo “package.json”, donde nos preguntará qué nombre de proyecto queremos, la versión de dicho proyecto, descripción, el punto de arranque de nuestro programa, entre otros. Aunque este comando no será necesario al momento de la aplicación de esta guía, es bueno saber que existe para futuras referencias. Si deseamos dejar los valores por defecto sin tener que responder a ninguna de las preguntas que nos aparecen en consola, tendremos que correr el mismo comando, pero con la bandera “-y” como parámetro, es decir, **npm init -y** .

***NOTA:*** En algunos sistemas operativos, puede causar conflictos tener un directorio con espacios al usar **npm init -y**. Es decir, si tenemos un directorio como el siguiente:

*C:/Ejemplos Node/Mi ejemplo con espacios/*

Esto puede causar que NPM no pueda crear el “package.json” adecuadamente. Así que, la manera ideal de nombras nuestros directorios y archivos, es sin espacios en sus nombres. Por ejemplo:

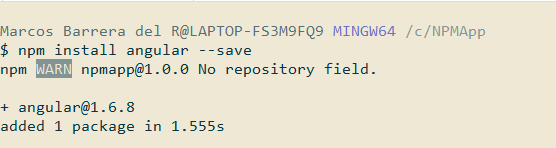
*C:/EjemplosNode/MiEjemploSinEspacios/*

**npm install:** Cada vez que deseemos instalar una nueva librería o módulo para poder utilizarlo en nuestra aplicación, el comando que nos ayudará a conseguirlo es “npm install”. Por ejemplo, para instalar el módulo llamado **angular** localmente en el directorio de nuestro proyecto, el comando sería el siguiente:

npm install angular --save

La bandera “--save” sirve para indicarle a npm que guarde a *angular*como dependencia en el “package.json” de nuestro proyecto.

Una vez que hayamos esperado a que se instalase nuestro módulo/paquete, se mostrará algo similar a lo siguiente:



Abrimos nuestro proyecto con nuestro editor:



Una vez en nuestro editor, podemos ver que ha habido varios cambios al ejecutar nuestro comando *npm install*.

**1.-** El primer cambio es que nuestro *package.json* ha sido modificado y se le agregó una nueva *key* llamada *dependencies.* En las dependencias se encontrarán todas aquellas librerías y módulos que nuestra aplicación requiere para funcionar. En nuestro caso sólo se muestra *angular,* ya que es el único módulo que hemos instalado. Además, podemos apreciar la versión de nuestro package que se instaló.

**2.-** El segundo cambio es que se creó una nueva carpeta en nuestro directorio del proyecto, la cual se llama *node\_modules.* Esta carpeta contiene cada uno de los módulos externos necesarios para que nuestro proyecto funcione adecuadamente. En nuestro caso, es *angular*.

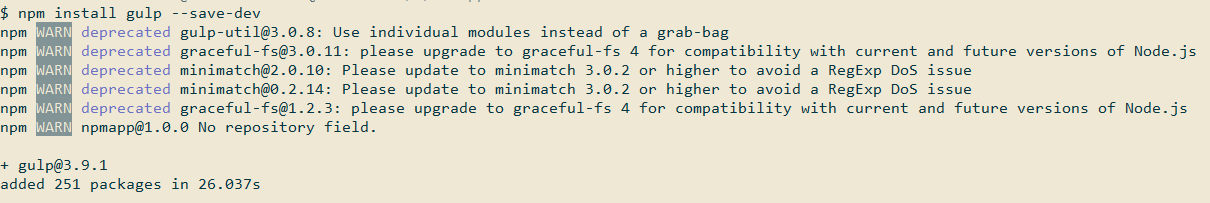
También se creó un archivo llamada “package-lock.json”, pero no profundizaremos en él, ya que no es relevante en esta guía.

Como mencionamos anteriormente, al hacer “*npm install angular --save”, angular* fue agregado como dependencia a nuestro package.json. Esto nos permite poder compartir nuestro proyecto sin incluir nuestra carpeta *node\_modules (la cual contiene muchos paquetes que pueden llegar a ser muy pesados).* Esta manera de compartir es realmente útil a la hora de subir proyectos a Github o Bitbucket, ya que reduce significativamente el internet requerido para subirlo.

Simplemente cuando hayamos transferido nuestro proyecto a otra computadora, deberemos entrar al directorio principal y ejecutar el comando “*npm install*”, lo cual instalaría automáticamente todas nuestras dependencias faltantes en una nueva carpeta *node\_modules.* Por esta razón fue necesario que NPM agregara nuestro paquete instalado al archivo “package.json”, para así tener una referencia de dónde poder descargarlo en caso de que no se encuentre instalado en el proyecto.

Otra característica del comando “npm install” es que nos permite instalar paquetes que sólo son necesarios al estar desarrollando nuestro proyecto, pero que no son necesarios en producción cuando la aplicación esté corriendo en un ambiente de producción real.

Por ejemplo, instalaremos gulp, un paquete muy útil en ambiente de desarrollo, pero que no es necesario en ambiente de producción. En casos como este, utilizaremos el comando “npm install” de la siguiente manera:



Si nos fijamos en nuestro archivo “package.json”, notaremos que se le agregó una nueva *key* llamada *devDependencies:*

{

"name": "npmapp",

"version": "1.0.0",

"description": "Mi descripción",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Marcos Barrera",

"license": "MIT",

"dependencies": {

"angular": "^1.6.8"

},

**"devDependencies": {**

**"gulp": "^3.9.1"**

**}**

}

Esto nos permite poder diferenciar cuáles paquetes necesitamos en producción y cuáles no. En el caso de que estemos en un ambiente de producción y que no queramos instalar los paquetes de *devDependencies* en nuestra carpeta *node\_modules*, simplemente tendremos que correr el comando “*npm install --production*”. Con nuestro “package.json”, si corriésemos el anterior comando, sólo se instalaría el paquete de *angular* y el paquete de *gulp* sería ignorado.

Sin embargo, si queremos que se instalen todos nuestros paquetes sin importar que tipo de dependencia sea, simplemente tendríamos que ejecutar el comando “*npm install*”, lo cual en nuestro ejemplo instalaría tanto a *angular,* como a *gulp.*

El comando “*npm install*” también nos permite instalar versiones específicas de un paquete. Por ejemplo, si quisiéramos instalar el *lodash (*una utilería de Javascript*),* en su versión “4.17.3”, el comando que utilizaríamos sería el siguiente:

npm install lodash@4.17.3 --save

Y en nuestras dependencias se agregaría exactamente esa versión de paquete:

"dependencies": {

"angular": "^1.6.8",

"gulp": "^3.9.1",

"lodash": "^4.17.3"

},

**Semantic Versioning en NPM**

La manera en la que NPM maneja las versiones de sus paquetes en la siguiente



* PATCH: Este número hace referencia a los parches que recibe el paquete, es decir, si hay algún bug en el paquete que fue arreglado, el número PATCH incrementará, pero no afectará al funcionamiento ni agregará cambios que rompan el funcionamiento actual del paquete.
* MINOR: Este número hace referencia a las nuevas mejoras y características que recibirá un determinado paquete. Es decir, cuando el paquete actualice su versión MINOR, significara que se agregó nueva funcionalidad al paquete, pero sin romper la funcionalidad anterior. Esto garantiza que las aplicaciones de dependan de dicho paquete sigan funcionando.
* MAJOR: Cuando este número cambia, significa que el paquete a sufrido un gran cambio en su funcionalidad y, por lo tanto, sea muy probable que agregue cambios que rompan la funcionalidad de versiones anteriores. Es decir, si tenemos “*lodash* 3.17.3” corriendo en nuestra aplicación y actualizamos a la versión 4.0.0, es muy probable que nuestra aplicación deje de funcionar.

Si miramos la manera en la que NPM instaló lodash anteriormente en nuestro “package.json”, podremos ver que tiene el símbolo “^” al lado de la versión instalada.

"lodash": "^4.17.3"

El símbolo “^” significa que NPM instalará la versiones MINOR y PATCH más actuales que tenga disponibles al momento de ejecutar el comando “*npm install*” o el comando “*npm update*” (explicado más adelante), sin actualizar la versión MAJOR para evitar posibles problemas de compatibilidad.

Si solo quisiéramos recibir las actualizaciones PATCH, tendríamos que añadir el símbolo “~” al lado de nuestro paquete de la siguiente manera:

"lodash": "~4.17.3"

Si quisiésemos mantener una versión exacta de nuestro paquete, sin actualizarlo, se establecería la versión de la siguiente manera:

"lodash": "4.17.3"

En el caso **no recomendado** de que quisiésemos instalar la última versión MAJOR de nuestro paquete, lo haríamos de la siguiente manera:

"lodash": "\*"

**npm update:** Este comando actualizará todos nuestros paquetes dependiendo de qué criterio de “semantic versioning” hayamos establecido en nuestro *package.json* para nuestros paquetes, es decir, haciendo uso de “^”, “~” o “\*”.

Por ejemplo, si tuviéramos el siguiente paquete:

"lodash": "^4.15.0"

Y aplicásemos el comando “*npm update*” en el directorio raíz de nuestro proyecto, NPM actualizaría dicho paquete de la siguiente manera:

"lodash": "^4.17.4"

Como podemos ver, NPM actualiza hasta la última versión disponible MINOR y PATCH, gracias al uso del símbolo “^”.

**npm uninstall:** Este comando nos permite desinstalar de nuestro proyecto cualquier paquete que queramos.

"dependencies": {

"angular": "^1.6.8",

"gulp": "^3.9.1",

"lodash": "^4.17.3"

},

"devDependencies": {

"gulp": "^3.9.1"

}

Por ejemplo, dadas las anteriores dependencias, para desinstalar el paquete de *angular,* tendríamos que usar el siguiente comando:

npm unistall angular –save

El comando anterior removería *angular* de la carpeta *node\_modules* y del archivo *package.json,* ya que al agregar la bandera “--save”, indicamos a NPM que lo remueva de package.json*.*

Si quisiéramos remover una dependencia de tipo *devDependencies,* la bandera tendría que ser cambiada a “*--save-dev*”.

npm unistall gulp --save-dev

**Módulos globales**

Por último, en NPM, lo que haremos será instalar módulos/paquetes globales. Este tipo de paquetes son útiles cuando necesitamos tener acceso a un módulo/paquete desde cualquier parte de nuestra computadora, es decir, si instalamos un paquete global, podremos acceder desde “C:\”, desde “C:\Users\Usuario\Documents” o desde cualquier directorio de nuestra computadora sin necesidad de tener que instalarlo nuevamente.

Por ejemplo, si quisiéramos utilizar el *package* llamado *live-server*, el cual nos permite correr un servidor web automáticamente en el directorio que deseemos; lo que nos se puede ocurrir hacer es instalarlo de la siguiente manera:

npm install live-server

Suponiendo que estuviésemos instalándolo desde el directorio “C:\NPMApp”, el anterior comando instalaría el package localmente en ese directorio, es decir, sólo tendríamos acceso a la funcionalidad del package “live-server” desde el directorio “C:\NPMApp”. Desde ningún otro directorio podríamos acceder a “live-server”, a no ser que instalásemos el paquete nuevamente en cada uno de los otros directorios, lo cual no sería óptimo. Así que, para solucionar el anterior problema, lo que se necesita hacer es instalar nuestro package (live-server) de manera global. Para instalarlo globalmente, el comando sería el siguiente:

npm install -g live-server

Como podemos ver, el comando es prácticamente idéntico al de una instalación normal, a excepción de la bandera “-g”. Esta bandera le indica a NPM que tiene que instalar a live-server de manera global. Al hacer esto, nuestro package ya será accesible desde cualquier carpeta/directorio de nuestra computadora.