Node y el gestor de paquetes NPM (Parte I)

Conociendo NPM

Competencia

- Describir qué es un gestor de dependencias, qué es NPM y para qué utilizarlo

Introducción

Uno de los conceptos más importantes dentro del flujo de trabajo de un desarrollador, sea cual sea su área de especialización, es la gestión de paquetes o módulos usados en el proyecto. Estos módulos o paquetes pueden ser representados como librerías, frameworks, plugins o cualquier apartado de código compatible con nuestro desarrollo.

En el caso particular de Node, tenemos a NPM, una potente herramienta que se encargará de gestionar todas las dependencias que necesitemos para desarrollar nuestros sistemas. En este capítulo aprenderás qué es y cómo utilizarlo, para posteriormente implementarlo y escalar tus aplicaciones con el uso de librerías y paquetes de terceros.

¿Qué es NPM?

NPM (del inglés Node Package Manager) es el sistema gestor de dependencias, paquetes o librerías predeterminado para proyectos que utilizan como base a Node y JavaScript, ¿Y qué es un gestor de dependencias? Es una herramienta que gestiona nuestros proyectos, los paquetes o librerías que estemos utilizando, si quieres saber más al respeto revisa el **Material Apoyo Lectura - ¿Qué es un gestor de dependencia?**, ubicado en "Material Complementario".

Por defecto, este gestor viene integrado al momento de instalar Node en el sistema operativo. Así como Node, NPM también se va actualizando eventualmente, podemos revisar qué versión de NPM tenemos instalado ejecutando la siguiente instrucción en un terminal de comandos (como Bash, CMD, Powershell, etc):

```
npm -v
```

Presionando enter, la instrucción anterior arrojará el número de versión instalada, como muestra la siguiente imagen:

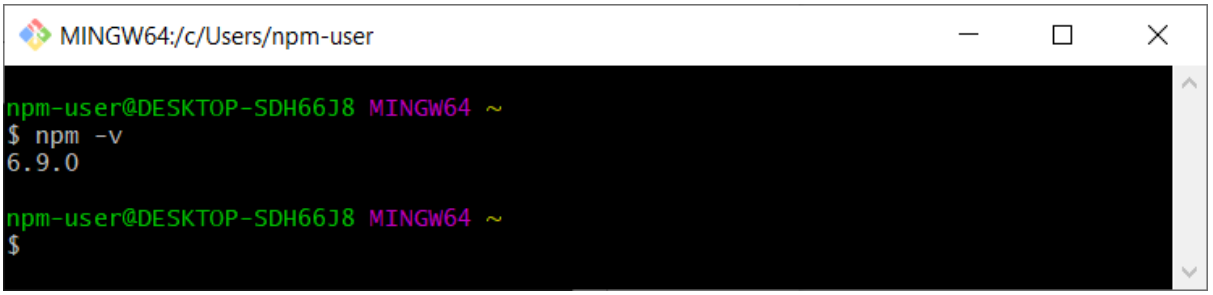

Imagen 1. Comando npm -v para revisar la versión actual de NPM.
Fuente: Desafío Latam

Así como con el comando ejecutado anteriormente, todas las tareas que se realicen con NPM están pensadas para ser ejecutadas en una línea de comandos. A medida que se avance con esta lectura, veremos que los comandos utilizados con NPM suelen ser sencillos de utilizar, sumado a que existe una cantidad importante de recursos y documentación en Internet sobre el uso de esta herramienta.

Por otro lado, a NPM también se le conoce como una plataforma de alojamiento en línea de paquetes, aplicaciones y/o proyectos Node de carácter open source y privado. Este repositorio de código se puede encontrar bajo la dirección https://www.npmjs.com/ y alberga una cantidad importante de paquetes que aumentan día a día.
Cada uno de los proyectos publicados en npmjs.com puede ser descargado e instalado en un proyecto local mediante la interfaz de comandos con la utilidad de NPM local.

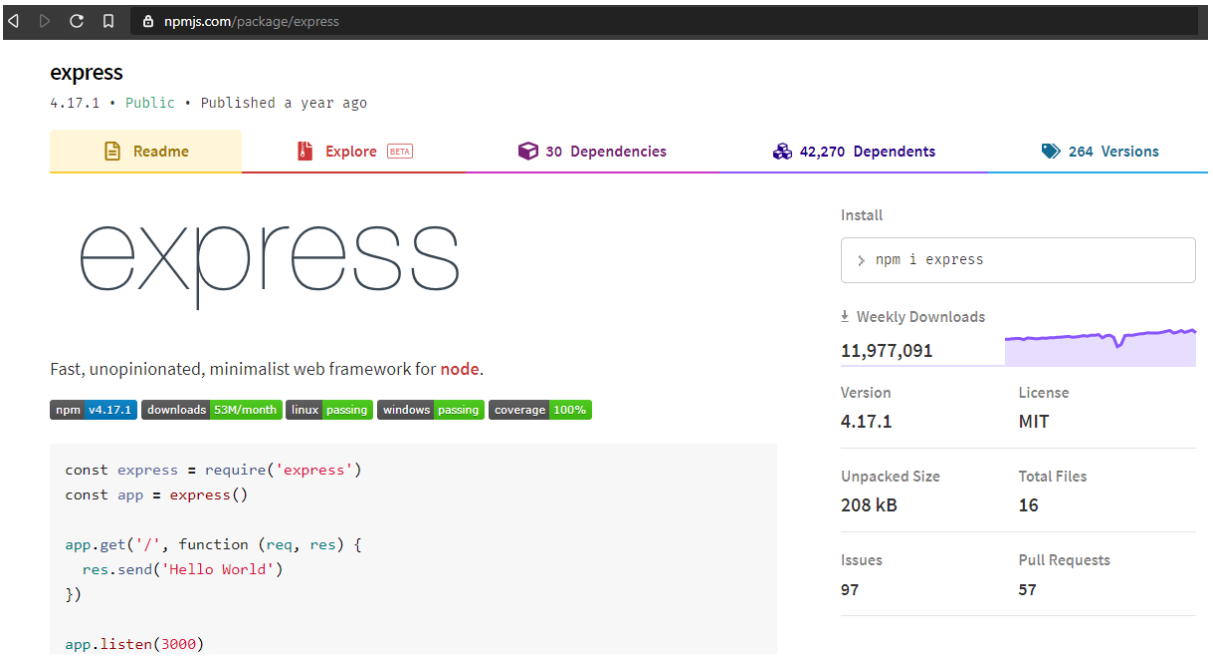

Imagen 2. Paquete express para el desarrollo en Node publicado en NPM.
Fuente: npmjs.com

En el sitio web de NPM, como podemos ver en la imagen anterior, se dispone de la documentación necesaria y datos de relevancia para una dependencia, como instrucciones de uso, códigos de ejemplo, versiones, descargas semanales, entre otros. Si se desea buscar un paquete, librería o aplicación en particular, npmjs.com es el sitio desde donde se debe empezar.

¿Para qué sirve NPM?

El principal objetivo de utilizar NPM como una herramienta de apoyo al desarrollo de software, es proveer un mecanismo automatizado para gestionar dependencias y tener un fácil control sobre éstas, mediante la utilización de comandos en una terminal del sistema.

En la siguiente imagen te muestro un diagrama donde podrás ver los diferentes usos o aplicaciones que nos ofrece NPM.

Imagen 3. Diagrama de acciones a realizar con el gestor NPM.
Fuente: Desafío Latam

En la imagen anterior, podemos ver algunas acciones que son posible de realizar mediante el gestor NPM. Para cada una de estas acciones, NPM nos provee comandos sencillos para lograr administrar paquetes de código; desde instalar una o más dependencias hasta actualizar, eliminar o listar éstas. Inclusive hacer público un repositorio que se desee compartir en Internet.

Manejo de dependencias

Competencia

- Aplicar procedimiento de instalación, desinstalación y actualización de paquetes utilizando el gestor NPM

Introducción

En esta capítulo aprenderás a ocupar diferentes comandos de NPM para la gestión de dependencias, entre ellos estarán los comandos para instalar, actualizar, desinstalar, listar y eliminar dependencias de terceros en un proyecto. Ésto te entregará las herramientas suficientes para empezar a gestionar de manera rápida y eficiente todas las dependencias o paquetes de código que estimemos convenientes.

Notarás al culminar el capítulo que la gestión de dependencias son sólo líneas de comandos que ejecutamos en la terminal, no obstante, saber cómo manejar las dependencias de tus proyectos puede ser mucho más importante que eso.

Día a día las tecnologías van evolucionando y actualizándose, en muchos casos las versiones de nuestros paquetes podrían quedar descontinuadas y por consecuencia generar problemas de compatibilidad con otras dependencias, la versión más reciente de un paquete ofrece una funcionalidad o rendimiento mucho más óptimo al que se tiene. Es de suma relevancia hacerle persecusión a los comportamientos para ofrecer y tener un software estable y escalable.

Manejo de dependencias

A continuación aprenderás los comando básicos y cotidianos de NPM en el desarrollo de aplicaciones, me refiero a los comandos que te ayudarán a instalar, actualizar, listar, revisar si la versión actual es la más actual y desinstalar dependencias.

Instalación

Las instalaciones de los paquetes pueden realizarse de forma manual copiando y pegando el código, no obstante contamos con un comando para comunicarnos con todos los repositorios desde nuestra terminal.

Para comenzar a instalar nuevas dependencias dentro del proyecto, podemos ejecutar el siguiente comando:

```
npm install <nombre_del_paquete>
```

Donde **<nombre_del_paquete>** será el nombre de la dependencia o librería que se desea instalar. Al ejecutar este comando, se registrará una dependencia dentro del archivo package.json como obligatoria para que el proyecto pueda funcionar.

Veámoslo directamente con un ejemplo, prosigue con las siguientes indicaciones:

1. Crea una nueva carpeta y abrela con tu editor de código preferido (De preferencia VSC)
2. Posteriormente crea un archivo index.js.
3. Realiza la iniciación de un proyecto NPM con el comando:

```
npm init
```

Perfecto, para este punto deberás tener solo el archivo index.js y el archivo package.json, ahora en la terminal procede a instalar la conocida librería llamada jQuery mediante la ejecución del siguiente comando:

```
npm install jquery
```

De manera alternativa, podemos instalar jQuery con los siguientes comandos:

```
npm install jquery --save
npm i jquery --save
npm i jquery
```

Desde la versión 5 de NPM en adelante, la opción --**save** es opcional y tendrá el mismo efecto si se especifica o no. Al ejecutar alguno de estos comandos deberás recibir algo parecido a lo que te muestro en la siguiente imagen.

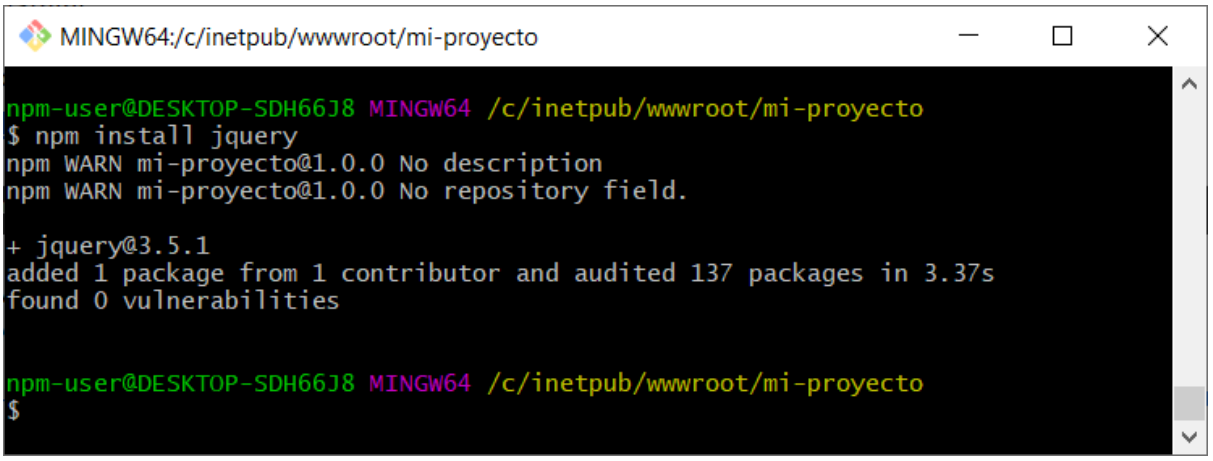

Imagen 4. Instalación de jQuery mediante npm install.
Fuente: Desafío Latam

Con esto, se instala la versión más reciente de la librería jQuery, que en el momento de realizada esta acción, corresponde a la 3.5.1.

Una vez que termine el proceso anterior, podemos ver que se creó la carpeta **/node_modules,** que contiene dentro de su estructura el código fuente de jQuery. Junto con lo anterior, también se creó un archivo llamado **package-lock.json** que contiene información detallada de los paquetes que se instalan dentro del proyecto, este último no se tocará en ningún momento. Si quieres saber más sobre de qué se trata la carpeta node_modules y el package.json revisa el **Material Apoyo Lectura - La carpeta node_modules y el package.json**, ubicado en "Material Complementario"

¿Cómo verifico que efectivamente fue instalada la dependencia? La forma rápida de comprobarlo es leyendo el package.json porque deberá haberse agregado automáticamente el objeto "dependencias" declarando únicamente el paquete jQuery y su versión, así como te muestro con el siguiente código.

```
{
 "name": "ejercicio-lectura",
 "version": "1.0.0",
 "description": "",
 "main": "index.js",
 "scripts": {
   "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "",
 "license": "ISC",
 "dependencies": {
   "jquery": "^3.5.1"
 }
}
```

Otra forma de comprobar que una dependencia fue instalada es buscando su carpeta correspondiente dentro de la carpeta node_modules, no obstante esto puede ser engorroso y debería ser innecesario pues el package.json deberá referenciar las dependencias instaladas.

En caso de que descargues el código de un proyecto o estés haciendo respaldo del tuyo, puedes exportar todos tus archivos exceptuando la carpeta node_modules y aplicar el siguiente comando en la terminal:

```
npm install
```

¿Qué hace esto? NPM con la ayuda de Node y de forma automatizada leerá el package.json e instalará todas las dependencias que se encuentren declaradas en él. De esta manera se reducirá el tamaño del proyecto y en próximas descargas sólo bastará con este comando para descargar las dependencias de dicho proyecto.

Desinstalación

Para eliminar o desinstalar algún paquete que ya no se necesite, podemos ejecutar el siguiente comando en la consola:

```
npm uninstall <nombre_del_paquete>
```

Esta acción quitará la dependencia indicada, tanto del archivo **package.json** como de la carpeta **/node_modules**.

Ahora ejecutemos el siguiente comando para eliminar jQuery de nuestro proyecto:

```
npm uninstall jquery
```

Deberás recibir una respuesta como la que verás en la siguiente imagen.

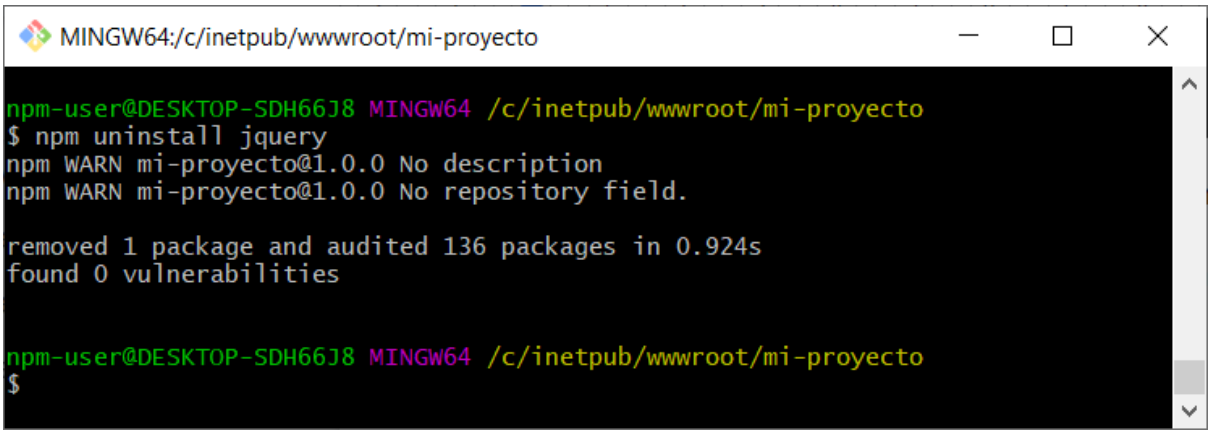

Imagen 5. Desinstalación de jQuery.
Fuente: Desafío Latam

Versionamiento y actualización

Es posible decidir qué versión de la dependencia se instalará, muy probablemente esto sea necesario por posibles problemas de compatibilidad con nuestro software o con otras dependencias. La forma de especificar el número de la versión del paquete es escribiendo junto a su nombre y el símbolo "@", así como te muestro en la siguiente sintaxis.

```
npm install <nombre_del_paquete>@<numero_version>
```

Hagamos la prueba, usa el siguiente comando para volver a instalar jQuery pero en esta ocasión en la versión 3.2

```
npm install jquery@3.2
```

Verás que esto al igual que en la imagen 4 instalará la librería pero en este caso en la versión indicada.

Lo siguiente será actualizar alguna dependencia, para esto ocuparemos la siguiente forma:

```
npm update <nombre_del_paquete>
```

De forma predeterminada NPM al actualizar o instalar una dependencia (sin especificar su versión) buscará en el repositorio la versión más reciente.

Para actualizar jQuery entonces usamos el siguiente comando.

```
npm update jquery
```

Y el resultado será el mismo que el de la instalación pero en este caso se trata de una actualización, tal como te lo muestro en la imagen.

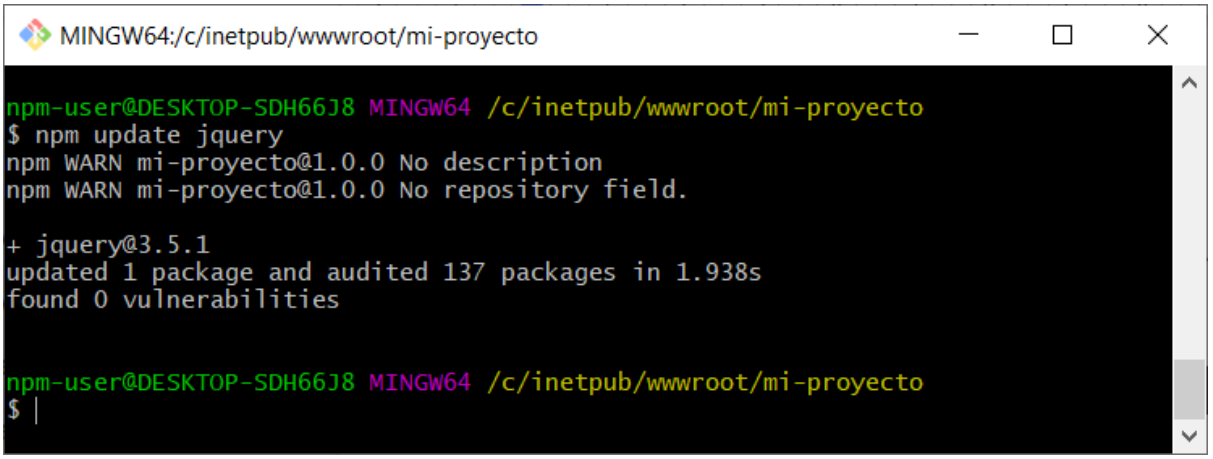

Imagen 6. Actualización de la dependencia jQuery.
Fuente: Desafío Latam

Es posible además ejecutar **npm update** sin indicar un nombre de alguna dependencia, lo cual actualizará todos los paquetes posibles que se encuentren listados en el archivo **package.json**. Si indicamos la bandera **-g**, podemos actualizar los paquetes instalados a nivel global o de sistema.

Ahora, ¿Qué sucede si deseo actualizar una dependencia para solo aplicar parches o quizá solo actualizar a una versión menor? Para esto, podemos modificar el registro de la dependencia en el archivo **package.json**, anteponiendo un carácter especial que NPM lo interpretará para saber a qué nivel deseamos actualizar. Por ejemplo:

- **~3.5.1**: El carácter "tilde" nos asegurará que solo se corregirán errores, es decir, se aplicará el parche más reciente disponible.

- **^3.5.1**: El carácter intercalación nos garantizará que se corregirán errores y se agregarán funcionalidades nuevas, es decir, se cambiará a la versión menor más reciente disponible.

- ***3.5.1**: El carácter asterisco nos asegurará que se actualizará a la versión mayor más reciente disponible. Aplicar este tipo de configuración suele ser de alto impacto, por lo que se requiere cuidado al proceder.

Existen muchos otros comandos como el "list" para recibir una lista de todas las dependencias instaladas con sus versiones, o el outdated que nos indicará qué versiones se encuentran desactualizadas, sin embargo estos son los más comunes.

Puedes conseguir todos los comandos en la siguiente dirección oficial de NPM

Ejercicio propuesto (1)

Completa las siguientes frases:

- Para desinstalar una dependencia se debe usar el comando _____
- Para instalar una versión en específico, por ejemplo la versión 4.5 de bootstrap se usaría el comando _____
- Para actualizar una dependencia, por ejemplo canvasjs se usaría el comando _____
- Para iniciar un proyecto NPM usamos el comando _____

Conociendo paquetes en Node

Competencias

- Identificar los paquetes más utilizados en las aplicaciones hechas con Node para solucionar un problema planteado
- Implementar el paquete nodemon para la automatización del levantamiento de aplicaciones Node

Introducción

El desarrollo puro puede ser beneficioso en cuanto a rendimiento se refiere, no obstante, instalar dependencias y módulos convierten nuestro proceso de desarrollo en un proceso más ágil y óptimo, por todo el tiempo que nos ahorramos al no intentar reinventar la rueda. Muchos de estos paquetes que usamos en el desarrollo son creados por la misma comunidad de desarrolladores a nivel mundial, personas u organizaciones que han dedicado una importante cantidad de tiempo en desarrollar una solución a un problema planteado y que disponen su código fuente de forma gratuita en NPM.

En este capítulo, identificarás varios paquetes conocidos de NPM y sus utilidades para conseguir resultados ideales en menos tiempo de desarrollo.

Una breve mirada a los paquetes más populares.

A continuación se presenta una lista con una breve descripción de los paquetes más utilizados en los proyectos para Node y que podemos gestionarlos a través de NPM.

- **Express:** Es un framework para aplicaciones web de servidor, famoso por la creación de infraestructuras web rápidas, minimalista y flexibles. Su popularidad crece y se ha convertido en un estándar para el desarrollo Backend en Node.

- **Nodemon:** Es una librería de gran utilidad para el desarrollo, ya que permite ir reflejando los cambios que vamos haciendo en nuestro código a medida que vamos guardando nuestro archivo. Cada vez que guardamos nuestro código, el mismo se compila inmediatamente, lo cual evita que por cada cambio realizado tengas que volver a ejecutar las líneas de código correspondientes en la consola.

- **Yargs:** En nuestros desarrollos, hay ocasiones en que necesitaremos crear algún script que ejecute alguna tarea en específico, como por ejemplo leer un archivo, automatizar una tarea, etc. Normalmente estos script recibirán parámetros y se comportaron de acuerdo a estos valores enviados. Yargs nos ayudará a formatear a objetos los parámetros que se envíen al ejecutar un script por consola.

- **Moment:** Es una librería que nos permitirá la manipulación de fechas y horas en nuestros desarrollos, trabajarlas en distintos formatos, acceder a la fecha y hora actual, realizar sumas, restas, comparaciones entre fechas, etc. Una gran gama de funcionalidades que todo desarrollo requiere en poca o gran medida.

- **Socket.io:** Libreria basada en WebSocket, que permite la comunicación bidireccional entre Cliente y Servidor. Con esto podrás realizar tareas en el servidor, por ejemplo, avisar a una aplicación de correos la llegada de un correo nuevo, o a una aplicación de chat indicarle que hay un mensaje nuevo por revisar.

- **Mocha:** Mocha es un framework de pruebas o test que nos entrega muchas características para la evaluación de código. Mocha puede ser implementado en Node o por medio de un navegador. Realiza sus pruebas en serie generando reportes flexibles y exactos. Es muy simple, flexible y divertido en su uso.

-

- **Underscore:** Es una librería de JavaScript que nos provee de un conjunto de métodos y funcionalidades que se pueden clasificar en manejo de colecciones, arreglos, funciones y objetos. Permite trabajar con funcionalidades avanzadas de búsqueda y filtros en arreglos.

- **Lodash:** Este paquete nació como una bifurcación de Underscore por lo que está compuesto en gran parte por las mismas propiedades y métodos, no obstante, es demandado hoy en día mucho en el mercado laboral por sus optimizaciones de rendimiento.

- **Morgan:** Es un middleware de registro de sucesos o conocidas como logger, que nos permite escribir en la consola peticiones, errores o lo que necesitemos mostrar o monitorear. Es una gran herramienta para utilizarla en el registro de errores, tanto para el desarrollo como para su funcionamiento productivo.

- **Jimp:** Es una biblioteca de procesamiento de imágenes para Node escrita completamente en JavaScript sin dependencias nativas. Su API es muy cómoda de ocupar.

- **Nodemailer:** Nodemailer es un módulo para aplicaciones Node que permite enviar correos electrónicos de forma sencilla. El proyecto comenzó en 2010 cuando no había una opción sensata para enviar mensajes de correo electrónico, hoy es la solución a la que recurren la mayoría de los usuarios de Node de forma predeterminada.

- **Axios:** Preferida por la mayoría de los programadores, es una librería para consultas de API REST que te permite hacer peticiones bajo el protocolo HTTP especificando los verbos en forma de métodos, obteniendo en su ejecución una promesa que devuelve como parámetro la respuestas de la consulta.

- **Chalk:** Es un paquete de NPM que ofrece funciones para colorear y personalizar la estética de los mensajes por consola. Contiene una amplia gama de colores y formas de estilizar los mensajes y de esta manera hacer más intuitiva y cómoda la lectura de las reacciones que puedan tener nuestras aplicaciones.

- **UUID:** Por sus siglas en inglés Universally Unique Identifier (Identificador único universal), este paquete es muy usado cuando necesitamos generar un campo identificador de un producto o recurso, su objetivo es generar identificadores para reconocer el objeto dentro de un sistema.

Todas estas dependencias puedes usarlas dentro de tus desarrollos y lo interesante sucede cuando mezclas sus poderes con los módulos integrados de node.

Ejercicio propuesto (2)

De los paquetes descritos previamente, ¿Cuáles crees que serían los ideales para solucionar los siguientes de planteamientos?

- Se requiere personalizar la estética de los mensajes impresos por consola: _____
- Se requiere desarrollar un servidor minimalista y flexible : _____
- Se requiere implementar en una aplicación la generación de identificadores universales: _____
- Se necesita desarrollar un servidor que envíe correos electrónicos masivos: _____

Nodemon

Este paquete no puede faltar en el proceso de desarrollo de servidores con Node. Su objetivo es automatizar el levantamiento del servidor cada vez que salvas cambios en el código, de esta manera no tendrás que preocuparte por cancelar la terminal dando de baja el servidor por cada nueva instrucción o cambio escrito.

Para instalar nodemon deberás ocupar el siguiente comando:

```
npm i nodemon
```

Este paquete tiene la peculiaridad de que no necesita ser importado, simplemente con el hecho de ser instalado podrás ocuparlo como un comando en la terminal. ¿Es posible usar una dependencia como un comando? Claro que si, te lo muestro con el siguiente ejercicio.

Ejercicio guiado: Ups, ¡me equivoqué!

Desarrollar una aplicación que imprima por consola un mensaje. La idea será voluntariamente escribir el mensaje que queremos enviar a la consola, al corregirlo y guardar cambios, notar que el servidor fue levantado de nuevo automáticamente arrojando por consola el mensaje corregido

En este ejercicio no serán necesario los pasos porque solo debes tener la siguiente línea de código escrita en tu index.js

```
console.log('Desafio Laram')
```

Ahora abre la terminal y escribe entonces el siguiente comando

```
nodemon index.js
```

Deberás obtener en la misma terminal lo que te muestro en la siguiente imagen.

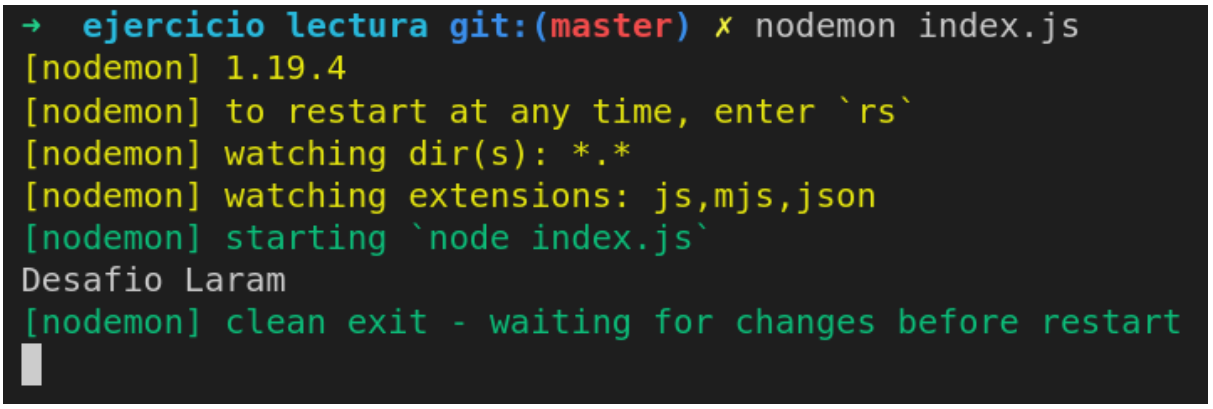

Imagen 7. Respuesta de nodemon al levantar la aplicación.
Fuente: Desafío Latam

Ahora la aplicación está preparada para volver a compilarse cada vez que se guarde un cambio en el código. Para esto cambia la letra "r" por una "t" en la palabra Laram y guarda el cambio. Deberás recibir lo que te muestro en la siguiente imagen por la terminal.

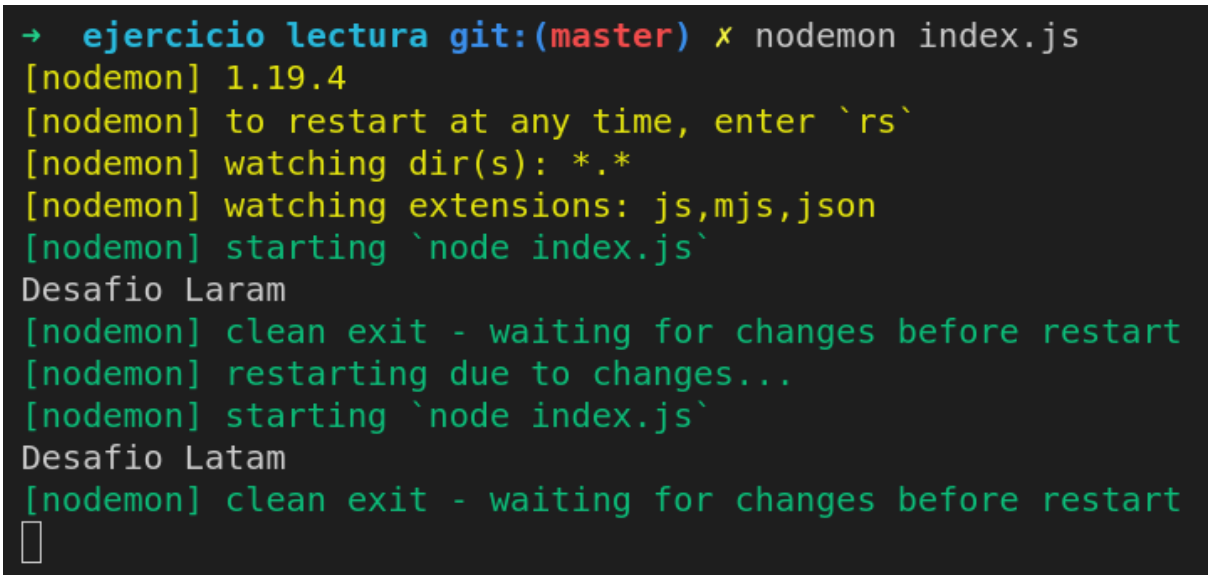

Imagen 8. Respuesta de nodemon al salvar el documento editado.
Fuente: Desafío Latam

Como puedes notar, la aplicación se reseteo inmediatamente y ahora estamos recibiendo por consola el mensaje corregido.

En caso que recibas errores por la terminal al intentar levantar una aplicación, te sugiero que revises el **Material Apoyo Lectura - Errores comunes en Node** ubicado en "Material Complementario", en donde encontrarás una lista de diferentes errores que podrían aparecer durante el desarrollo de una aplicación.

Paquetes NPM que no procesan datos

Competencias

- Implementar el paquete chalk en una aplicación node para la personalización estética de los mensajes enviados a consola
- Implementar el paquete UUID en una aplicación node para la generación de identificadores únicos universales

Introducción

Al desarrollar aplicaciones en Node, constantemente estaremos haciendo uso de paquetes, no obstante, no todos los paquetes o dependencia son creados con el objetivo de procesar algún dato, sino simplemente ofrecernos una herramienta que nos sea de utilidad para algún caso dado. Tenemos por ejemplo los paquetes Chalk y UUID, los cuales no se consideran herramientas para procesar variables ya que pueden ser utilizados independientemente y no requieren de argumentos o algún paso de información para ser ocupados.

En este capítulo, aprenderás a personalizar el estilo de los mensajes de tu consola con el paquete Chalk para leerlos con más comodidad e intuición, además veremos cómo generar identificadores únicos universales con el paquete UUID que representan instancias u objetos. Ambos paquetes te serán de mucha utilidad en un futuro cuando debas crear aplicaciones complejas de muchas entidades y necesites identificarlas de una manera segura, cuando te topes con la necesidad de emitir varios mensajes por consola y quieras identificar las descripciones de una forma más intuitiva.

Chalk

¿Alguna vez has visto una terminal con colores? Pues chalk probablemente fue el paquete usado para eso. Su API es bastante clara y directa, además de permitir concatenaciones de métodos y propiedades para construir toda la estética en una sola línea

Para instalar chalk deberás ocupar el siguiente comando:

```
npm i chalk
```

Ejercicio guiado: Aplicando el paquete chalk

Se requiere implementar el paquete Chalk en una aplicación que imprima un mensaje por consola de color azul. Realiza los siguientes pasos para la creación de una mini aplicación que imprima un "Hola Mundo!" con el paquete chalk:

- ***Paso 1:*** Importa el paquete chalk en una constante
- ***Paso 2:*** Envía por consola un "Hola Mundo!" pero este texto deberá ser el argumento del método "blue" del objeto importado.

```
// Paso 1
const chalk = require('chalk')
// Paso 2
console.log(chalk.blue('Hola Mundo!'))
```

- ***Paso 3:*** Correr la aplicación por la terminal y deberás recibir lo que se muestra en la siguiente imagen.

```
→  ejercicio lectura git:(master) ✗ node index.js
Hola Mundo!
```

Imagen 9. Impresión por consola de un mensaje en color azul.
Fuente: Desafío Latam

¿Genial cierto? Aún hay más, ese metodo "blue" también puede ser usado como propiedad y ser concatenada con otras propiedades, sustituye el console.log escrito anteriormente por la siguiente línea de código.

```
console.log(chalk.blue.bgRed.bold('Hola Mundo!'))
```

Ahora si vuelves a correr el archivo index.js obtendrás lo que te muestro en la siguiente imagen en donde el mensaje ahora estará en color azul, con fondo rojo y en negritas.

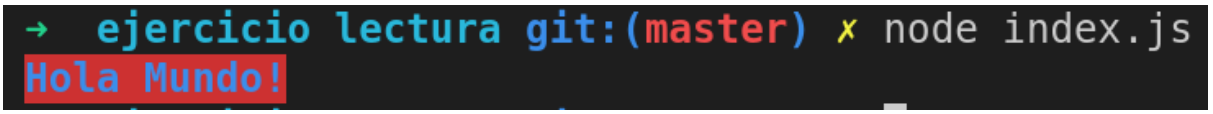

Imagen 10. Impresión por consola de un mensaje en color azul, con fondo rojo y en negritas.
Fuente: Desafío Latam

Si quieres aprender más sobre este paquete te dejo el link directo al repositorio en NPM.

Ejercicio propuesto (3)

Crear una aplicación que devuelva por consola un mensaje de fondo amarillo y color de texto verde.

UUID

El UUID es un paquete de NPM que nos permite crear identificadores únicos universales, ¿Y como se ven estos identificadores? Son cadenas de texto conformadas por 32 dígitos divididos en cinco grupos que obedecen a la siguiente forma: 8-4-4-4-12. Es decir, 36 caracteres y cuatro guiones que los separan.

Por ejemplo, un identificador UUID se ve así: “670b9562-b30d-52d5-b827-655787665500”.

Para instalar UUID deberás ocupar el siguiente comando:

```
npm i uuid
```

Ejercicio guiado: Aplicando el paquete UUID

Realizar una pequeña aplicación que utilice UUID para enviar por consola un identificador único universal. Para esto sigue los siguientes pasos:

- ***Paso 1:*** Importar el paquete UUID en una constante especificando que usaremos la versión 4.

- ***Paso 2:*** Enviar por consola la llamada al método uuidv4.

```
// Paso 1
const { v4: uuidv4 } = require('uuid')
// Paso 2
console.log(uuidv4())
```

Ahora corre la aplicación 3 veces por la terminal y deberás recibir lo que se muestra en la siguiente imagen.

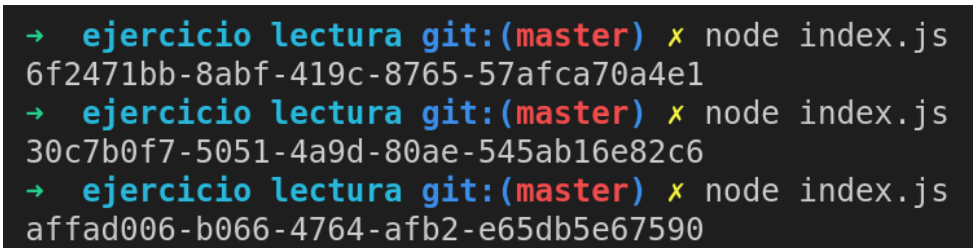

Imagen 11. Impresión por consola de 3 identificadores únicos.
Fuente: Desafío Latam

Como bien puedes apreciar, ninguno de los identificador se repite, esto lo pudieras hacer las veces que quieras y seguirás recibiendo un id distinto.

Recordemos que estos identificadores los ocupamos para representar una entidad, por ejemplo, en una aplicación real que administre productos, cada producto podría ser identificado por estas cadenas de caracteres, no obstante no estamos obligados a usar los 36 dígitos, podríamos ocupar el método splice o el slice para tomar solo una parte de estos. En el siguiente código te muestro cómo usar solo una parte de estos identificadores.

```
"670b9562-b30d-52d5-b827-655787665500".slice(0,6)
```

Esto devolvería el String que te muestro a continuación.

```
"670b95"
```

Y ahora esta cadena de texto podría ser finalmente el identificador que necesitamos. Si necesitas repasar los métodos para recortar cadenas de texto y arreglos te dejo la documentaciones de splice y slice.

Ejercicio propuesto (4)

Crear una aplicación que devuelva por consola los últimos 6 caracteres de un UUID.

Paquetes NPM para el procesamiento de datos

Competencias

- Implementar la librería Moment y Lodash para el procesamiento de fechas y arreglos.
- Implementar la librería Axios para el consumo de APIs.

Introducción

A diferencia de los paquetes que aplicaste en el capítulo anterior, existen librerías que nos ofrecen métodos o funciones para el procesamiento de datos, esto quiere decir, que su objetivo es trabajar con variables o funciones, para facilitar algún proceso que de otra manera sería engorroso o más tedioso de trabajar con código propio.

En este capítulo, aprenderás a usar tres de los paquetes más populares para el procesamiento de datos, estos serán Moment, Lodash y Axios, los cuales facilitan el trabajo al momento de querer procesar datos de tipo fecha, arreglos y objetos, y el consumo de apis respectivamente.

Usar estas librerías te ahorrarán un tiempo considerable al momento de desarrollar tus aplicaciones gracias a sus métodos minimalistas y la optimización de procesos que ofrecen.

Moment

Tal vez uno de los más populares paquetes/librería en el mundo de JavaScript es Moment, nos ha sacado de aprietos a miles de desarrolladores por su alta y óptima capacidad de procesar el objeto Date().

El uso de esta librería radica en el tratamiento de fechas de una forma muy cómoda y su instalación se da con el siguiente comando

```
npm i moment
```

En el [sitio oficial de Moment.js] encontrarás un montón de propiedades y métodos con los que podrías estar todo el día jugando. Entre los métodos que más destacan están el "subtract" y el "add", ambos para restar o agregar días respectivamente.

Ejercicio guiado: Aplicando Moment.js

Crear una mini aplicación que imprima por consola la ejecución del método moment, esto devolverá un formato de fecha correspondiente al momento en el que es ejecutada la función, de esta manera podrás tener un primer encuentro con este paquete.

- ***Paso 1:*** Importar el paquete moment en una constante.
- ***Paso 2:*** La importación del paquete en sí es un método, así que envía su llamado directamente por consola.

```
// Paso 1
const moment = require('moment')
// Paso 2
console.log(moment())
```

Ahora corre la aplicación y deberás recibir algo como lo que se muestra en la siguiente imagen.

```
→  ejercicio lectura git:(master) ✗ node index.js
Moment<2020-09-15T01:43:12-03:00>
```

Imagen 12. Impresión por consola del método moment().
Fuente: Desafío Latam

Nota la forma en la que recibimos la fecha, es bastante parecida a lo que nos devuelve una nueva instancia del objeto Date de JavaScript. Moment también nos ofrece diferentes formatos y métodos para generar o procesar fechas.

Por ejemplo, tenemos el método "subtract" y el método "add" que nos permiten restar y agregar días, meses, años, horas, etc. También tenemos el método "format" con el que podemos especificar el formato de fecha que queremos ocupar.

Vayamos manos al código para probar un poco más este interesante paquete. Sustituye el console.log que hiciste anteriormente con la siguiente línea de código para que veas un ejemplo de estos métodos en donde restaremos 10 días a la fecha actual y le daremos un formato específico.

```
console.log(moment().subtract(10, 'days').format('MMM Do YY'))
```

Ahora si vuelves a correr el archivo index.js obtendrás lo que se muestra en la siguiente imagen.

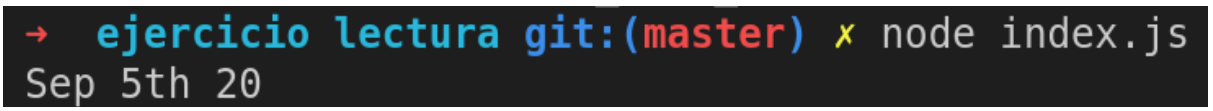

Imagen 13. Impresión por consola del método moment() restado 10 días.
Fuente: Desafío Latam

Como puedes ver, con pocos cambios logramos procesar una fecha y obtener un formato diferente al que acostumbramos a hacer en programación básica con el "new Date()".

Acá te dejo el sitio oficial de esta librería donde encontrarás toda la documentación referente a su API.

Ejercicio propuesto (5)

Crear una aplicación que devuelve la fecha actual sumando 10 días y en el siguiente formato "dddd"

Muy bien, hasta ahora has aprendido a imprimir mensajes personalizados con colores por consola, a generar identificadores únicos universales y a tratar fechas con el paquete moment. Es momento de intentar hacer una mezcla de paquetes, para esto desarrollaremos un ejercicio.

Ejercicio guiado: ¿Qué día será en el futuro?

Desarrollar una mini aplicación que devuelva por consola la fecha actual sumandole 10.000 días, además supondremos que estas son consultas que serán registradas como un historial por lo que deberán tener un identificador único.

Sigue los siguientes pasos para el desarrollo de este ejercicio:

- ***Paso 1:*** Importar los paquetes chalk, uuid y moment.
- ***Paso 2:*** Crear un objeto "consulta"
- ***Paso 3:*** Almacenar en la propiedad "fecha" la fecha actual sumada 10.000 días
- ***Paso 4:*** Generar un identificador único universal
- ***Paso 5:*** Imprimir por consola el objeto "consulta"

```
// Paso 1
const chalk = require('chalk')
const { v4: uuidv4 } = require('uuid')
const moment = require('moment')
// Paso 2
const consulta = {
 // Paso 3
 fecha: moment().add(10000, 'days').format('MMM Do YY'),
 // Paso 4
 ID: uuidv4(),
}
// Paso 5
console.log(consulta)
```

- ***Paso 6:*** Ahora intenta correr la aplicación y obtendrás algo como lo que se muestra en la imagen.

```
→  ejercicio lectura git:(master) ✗ node index.js
{ fecha: 'Feb 1st 48', ID: '6605edc9-1d62-46fe-9d14-f0f669adf0af' }
```

Imagen 14. Impresión por consola del ejercicio guiado "¿Qué día será en el futuro?".
Fuente: Desafío Latam

Excelente! que bien lo has hecho, podrás darte cuenta que no ha sido nada complicado sino todo lo contrario y es que aunque no todos, así funcionan la mayoría de los paquetes NPM que te puedas encontrar.

Lodash

Ahora conocerás uno de los paquetes que en varias ocasiones ha liderado el ranking de dependencias más usadas en aplicaciones con JavaScript, se trata de lodash, la librería que facilitará tus días cuando te encuentres con problemas que aparentan exigir horas de concentración e iteraciones. Siendo más específico, este paquete te ayudará a eliminar la molestia de trabajar con arreglos, números, objetos, cadenas de texto, etc.

Para instalar lodash deberás ocupar el siguiente comando:

```
npm i lodash
```

Ejercicio guiado: Pares o nones

Desarrollar una aplicación que divida un arreglo de números en 2 arreglos, separando los números pares e impares. Sigue los siguientes pasos para el desarrollo de este ejercicio:

- ***Paso 1:*** Importar el paquete lodash en una constante. Esta librería tiene una peculiaridad que es ser usada como un guión bajo, es decir la constante que crees declararla con un "_"

- ***Paso 2:*** Crear un arreglo de números en secuencia del 1 al 6.

- ***Paso 3:*** Usar el método llamado "partition" que recibe como primer parámetro el arreglo de números y como segundo parámetro tendrás el callback de este método, entonces se debe escribir una función de flecha que recibe un parámetro identificando cada uno de los elementos del arreglo y retorne los que cumplan con la siguiente condición "n % 2"

```
// Paso 1
const _ = require('lodash')
// Paso 2
const numeros = [1, 2, 3, 4, 5, 6]
// Paso 3
console.log(_.partition(numeros, (n) => n % 2))
```

- ***Paso 4:*** Ejecutar la aplicación por la terminal y deberás recibir lo que se muestra en la siguiente imagen.

```
→  ejercicio lectura git:(master) ✗ node index.js
[ [ 1, 3, 5 ], [ 2, 4, 6 ] ]
```

Imagen 15. Impresión por consola de la partición de un arreglo.
Fuente: Desafío Latam

Para hacerte una idea del tiempo que te estás ahorrando te presento a continuación una solución al problema planteado que lodash ha logrado resolver en una línea de código.

```
const numeros = [1, 2, 3, 4, 5, 6]
let pares = []
let impares = []
for (let index = 0; index < numeros.length; index++) {
 if (numeros[index] % 2) {
   impares.push(numeros[index])
 } else {
   pares.push(numeros[index])
 }
}
let arregloFinal = [impares, pares]
console.log(arregloFinal)
```

¿Sorprendente la diferencia cierto? Ahora en adelante cada vez que tengas un problema, primero pregúntate si la solución ya existe y probablemente te consigas un paquete de NPM que te resuelva el problema con muy poco esfuerzo.

La documentación completa y oficial de lodash contiene la amplia documentación de todos sus métodos y propiedades. Acá te dejo el link para que vayas a visitarla,

Ejercicio propuesto (6)

Basado en el ejercicio “Pares o nones”, crea una aplicación que particione el siguiente arreglo

```
const numeros = [true, 0, null, undefined, '', 22, false]
```

El objetivo será dividir el arreglo en dos grupos en donde el primero contenga los elementos evaluados lógicamente como true y el segundo los elementos considerados lógicamente como false.

Axios

Axios es la librería para el consumo de API REST más popular en el desarrollo con JavaScript, querido y defendido por la gran mayoría de los desarrolladores, teniendo a la fecha, en el momento del desarrollo de esta lectura, más de 13 millones de descargas semanales en su repositorio de NPM.

Su instalación se consigue a través del siguiente comando

```
npm install axios
```

El uso de este paquete es bastante intuitivo por su forma declarativa de llamar a los métodos, dejándonos una sintaxis cómoda y directa para desarrollar. A continuación, te muestro la sintaxis básica para usar esta librería, en donde encontrarás un uso parecido al método fetch que ocupamos en el frontend, para el consumo de API REST.

```
axios
.<verbo http>( <url a consultar>)
.then( [callback con la data de la consulta] )
.catch( [callback con el error de la consulta] )
```

Esto se podrá ver con claridad en un ejemplo de aplicación, pero antes es importante que sepas que a diferencia del método fetch, axios te entrega como respuesta de cualquier consulta, un objeto con el detalle de la pura petición. Dentro de ésta se encontrará una propiedad “data” que será la que contendrá la información solicitada. Así como te muestro en la siguiente imagen.

Imagen 16. Impresión por consola del objeto que recibimos en una consulta con axios.
Fuente: Desafío Latam

Como bien puedes observar, tenemos diferentes propiedades que representan la consulta que se está realizando, entre estas propiedades tenemos la configuración utilizada, la cabecera, el estatus obtenido en la consulta y entre otras cosas la propiedad "data", que es la que contiene finalmente la data que necesitamos.

Ejercicio guiado: Aplicando axios

Consumir la API de Rick and Morty en el siguiente endpoint https://rickandmortyapi.com/api/character/1, el objetivo será imprimir en consola el nombre de este personaje. Instala el paquete NPM en un proyecto nuevo y sigue los siguientes pasos para el desarrollo de este ejercicio.

- ***Paso 1:*** Importar el paquete "axios" en una constante

- ***Paso 2:*** Ocupar la instancia de "axios" y su método "get" pasando como argumento la URL del endpoint a consultar

- ***Paso 3:*** El método usado en el paso anterior devuelve una promesa, así que en el callback del método "then" recibirás el objeto "data" con la información que se mostró en la imagen 16. En esta API el nombre del personaje se encuentra en el primer nivel del objeto de la data.

  Guardar en una constante el nombre del personaje usando la siguiente especificidad "data.data.name", siendo la primera "data" el objeto de axios, la segunda "data" el objeto correspondiente al personaje y por supuesto la propiedad "name" que devuelve el String del personaje que estamos consultando

- ***Paso 4:*** En el catch capturar e imprimir un error de consulta en caso de existir.

```
// Paso 1
const axios = require("axios");

// Paso 2
axios
  .get("https://rickandmortyapi.com/api/character/1")
  .then((data) => {
    // Paso 3
    const name = data.data.name;
    console.log(name);
  })
  // Paso 4
  .catch((e) => {
    console.log(e);
  });
```

Ahora levanta tu aplicación en la terminal y deberás obtener lo que te muestro en la siguiente imagen.

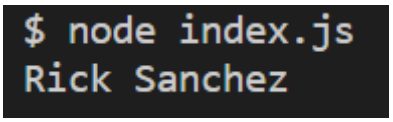

Imagen 17. Impresión por consola del nombre de un personaje de Rick and Morty.
Fuente: Desafío Latam

Ahí está, el nombre del personaje es “Rick Sanchez” obtenido con la librería axios

Ejercicio propuesto (7)

Basado en el ejercicio de Rick and Morty, desarrolla una aplicación en Node que utilice Axios para consultar el valor del dólar en Chile utilizando la Siguiente API de indicadores económicos.

Resumen

A lo largo de esta lectura cubrimos:

- Qué es NPM, utilidad y uso de comandos básicos.
- Una breve mirada de los paquetes más populares de NPM en el desarrollo de aplicaciones Node.
- El paquete Chalk para impresión de mensajes por consola con estética personalizada.
- El paquete UUID para la generación de identificadores universales únicos
- El paquete/librería Moment para el tratamiento de fechas de una forma cómoda en JavaScript.
- El paquete/librería Lodash para el procesamiento óptimo y minimalista de matrices y objetos.
- Implementar la librería Axios para el consumo de APIs.
- El paquete Nodemon para el relevantamiento automatizado del servidor o aplicaciones al guardar cambios en el código.

Solución de los ejercicios propuestos

1. Completa las siguientes frases:

    - Para desinstalar una dependencia se debe usar el comando npm uninstall <nombre de la dependencia>
    - Para instalar una versión en específico, por ejemplo la versión 4.5 de bootstrap se usaría el comando npm install bootstrap@4.5
    - Para actualizar una dependencia, por ejemplo canvasjs se usaría el comando npm update canvasjs
    - Para iniciar un proyecto npm usamos el comando npm init

2. De los paquetes descritos previamente, ¿Cuáles crees que serían los ideales para solucionar los siguientes de planteamientos?

    - Se requiere personalizar la estética de los mensajes impresos por consola: Chalk
    - Se requiere desarrollar un servidor minimalista y flexible : Express
    - Se requiere implementar en una aplicación la generación de identificadores universales: UUID
    - Se necesita desarrollar un servidor que envíe correos electrónicos masivos: Nodemailer

3. Crear una aplicación que devuelva por consola un mensaje de fondo amarillo y color de texto verde

```
console.log(chalk.green.bgYellow('Hola Mundo!'))
```

4. Crear una aplicación que devuelva por consola los últimos 6 caracteres de un uuid

```
console.log(uuidv4().slice(30))
```

5. Crear una aplicación que devuelva la fecha actual sumando 10 días y en el siguiente formato “dddd”

```
console.log(moment().add(10, 'days').format('dddd'))
```

6. Basado en el ejercicio “Pares o nones”, crea una aplicación que particione el siguiente arreglo

```
const numeros = [true, 0, null, undefined, '', 22, false]
console.log(_.partition(numeros, (n) => n))
```

7. Basado en el ejercicio de Rick and Morty, desarrolla una aplicación en Node que utilice Axios para consultar el valor del dólar en Chile utilizando la api de https://mindicador.cl/api.

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

axios
  .get("https://mindicador.cl/api")
  .then((data) => {
    const dolar = data.data.dolar.valor;
    console.log("El valor del dolar en Chile es: " + dolar);
  })
  .catch((e) => {
    console.log(e);
  });
```