

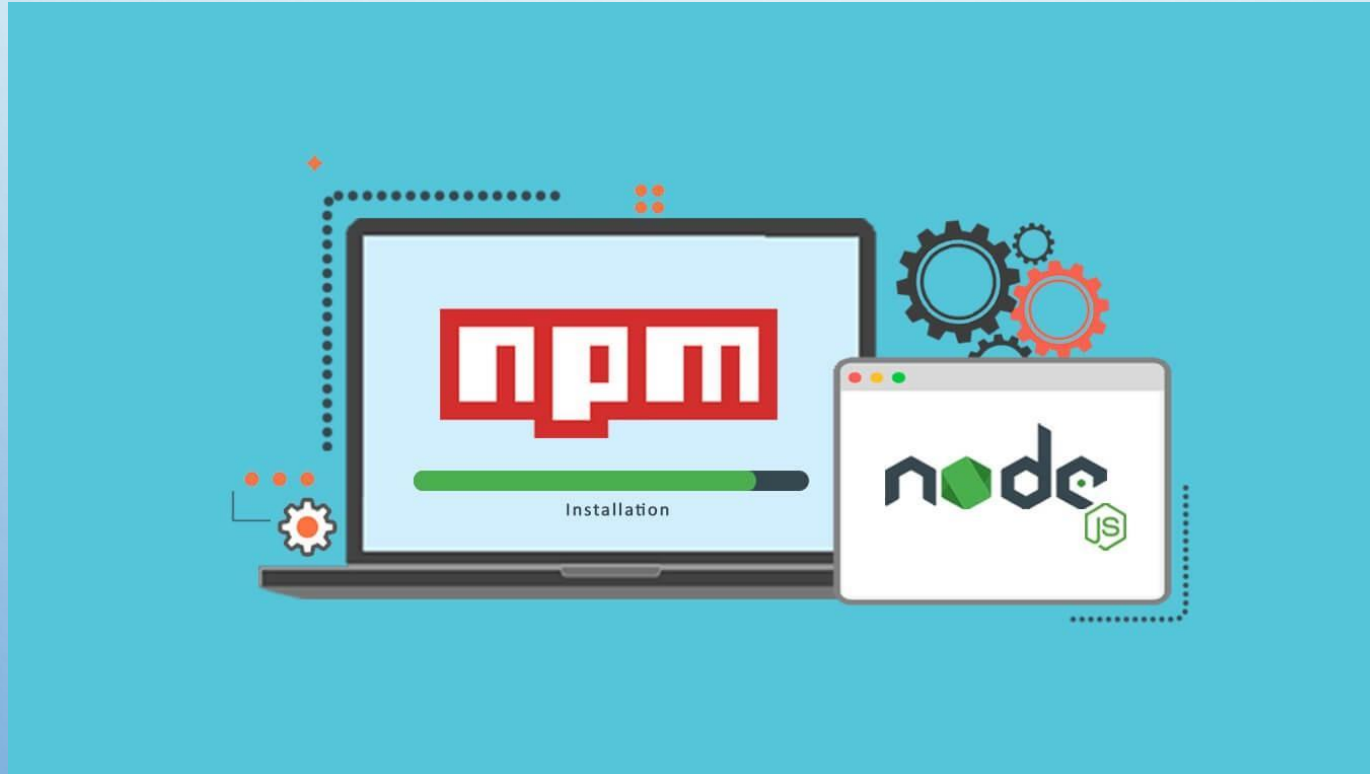
NPM - EXPRESS

Concepto

- Los **Package Managers** (o Administradores de paquetes) sirven para no tener que descargar, instalar y mantener las dependencias de un proyecto a mano.
- Estas aplicaciones facilitan la **descarga e instalación de las librerías** que utiliza el proyecto.
- Se requiere que conozcamos el nombre exacto de la **librería** (y versión deseada si es necesario) y contar con conexión a Internet.
- Mediante un comando se descargará de un **repositorio centralizado** la versión correspondiente de la dependencia especificada y se agregará al proyecto.

NPM

- NodeJS cuenta con su propio Administrador de Paquetes: **NPM** (*NodeJS Package Manager*).



Dependencias con NPM

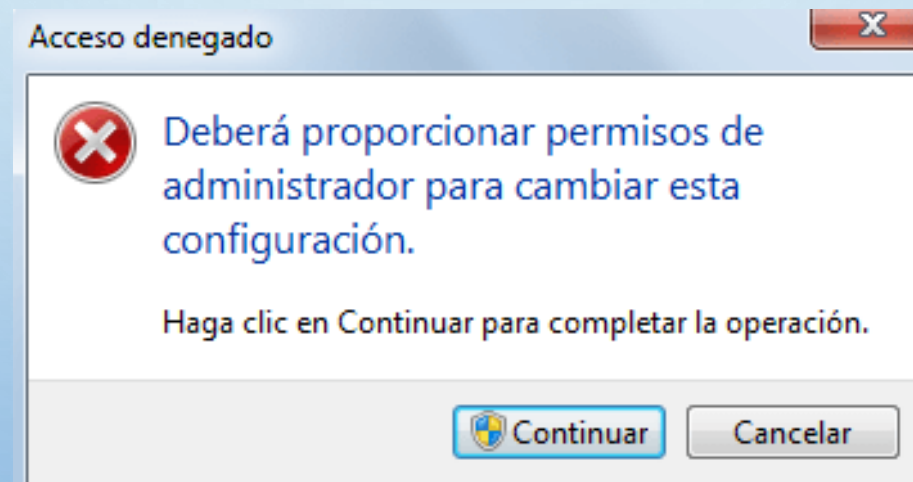
```
//Global:  
npm install -g nombre-de-la-librería  
//Local:  
npm install nombre-de-la-librería
```

- Las dependencias pueden instalarse en **forma global o local**.
- Si instalamos una dependencia en **forma global**, **todos** nuestros **programas** desarrollados en NodeJS contarán con esa **librería**, y con la versión que haya sido instalada.
- En cambio, si instalamos en **forma local**, podremos **elegir** exactamente qué **librería** y con qué **versión** contará **cada proyecto** que desarrollemos.

- La **instalación local** de dependencias es la opción es la más **recomendable**, para poder tener múltiples proyectos usando distintas versiones de una misma librería sin generar problemas de **compatibilidad** al actualizar a una nueva versión que no sea retrocompatible con las anteriores.
- Sin embargo, muchas veces es útil instalar en forma **global** **librerías utilitarias** (por ejemplo librerías de testing) que son usadas para facilitar las **tareas** de **programación** y **revisión** durante la etapa de **desarrollo** pero que no son necesarias para el uso de la aplicación.

- El programa A usa la librería “fecha” en su versión 1.0, que cuenta con el método “*dameFecha()*”.
- El programa B usa la librería “fecha” pero en su versión 2.0, que ya no cuenta con el método “*dameFecha()*” ya que éste fue reemplazado por el nuevo método “*dameFechaLocal()*”.
- Si instalamos “fecha” en **forma global**, al actualizar la librería fecha, romperemos el programa A, ya que intentará usar un método que ya no existe.
- Si realizamos dos instalaciones **locales** (fecha 1.0 para A, fecha 2.0 para B) al actualizar cada una por separado, cada programa seguirá contando con la versión correspondiente).

Es posible que al instalar dependencias en forma global se nos solicite tener permisos de administrador, ya que estaremos editando archivos de configuración y agregando contenidos en carpetas del sistema.



Package.json

Package.json

```
{
  "type": "module",
  "name": "mi-proyecto",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4",
    "joi": "~14.3.1",
    "sleep": "*6.0.0"
  },
  "devDependencies": {
    "jest": "latest"
  }
}
```

- Es un archivo de configuración en formato **JSON** que es parte de un proyecto Node.js.
- Podemos especificar en este archivo la lista de dependencias, que son las librerías que usa el proyecto para funcionar o para realizar distintos tipos de testing.

Dependencias

- ❑ Siempre que hayamos especificado nuestras dependencias en el archivo de configuración (package.json) podremos actualizar y mantener de forma fácil y segura las dependencias del proyecto con el comando ***npm install***
- ❑ Además podemos hacer que npm agregue como dependencia al package.json un módulo que estamos instalando. Si lo queremos como **dependencia del proyecto**, al comando *'install'* le agregamos el nombre del módulo: ***npm install <algún-módulo>***
- ❑ Si sólo es una **dependencia del entorno de desarrollo**, le agregamos ***--save-dev*** ó ***-D***: ***npm install --save-dev <algún-módulo-dev>*** ó ***npm install -D <algún-módulo-dev>***

Calculadora de edad

Vamos a practicar lo aprendido hasta ahora

Tiempo: 5/10 minutos

Realizar un proyecto en node.js que permita calcular **cuantos años y días totales transcurrieron desde la fecha de tu nacimiento**. Para ello utilizar la dependencia *moment* instalándola en forma local desde npm. Imprimir los resultados por consola.

Un ejemplo de salida:

Hoy es 11/01/2021

Nací el 29/11/1968

Desde mi nacimiento han pasado 52 años.

Desde mi nacimiento han pasado 19036 días.

Aclaración:

Importar moment con require (CommonJS)

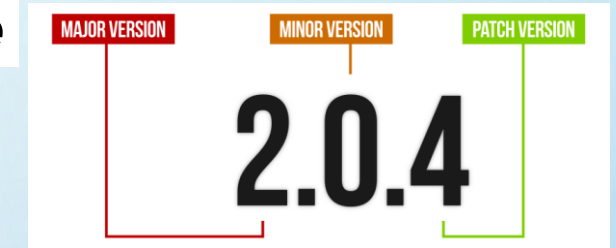
Ayuda:

Utilizar los métodos diff y format de moment

Versionado

Versionado

Las librerías de NPM siguen un estándar de versionado de tres números, separados entre sí por un punto:



- **Major Release:** El primer número corresponde a actualizaciones grandes/significativas que incluyen muchas nuevas características, o que cambian de manera significativa el funcionamiento de las existentes.
- **Minor Release:** El segundo número corresponde a actualizaciones pequeñas que agregan pocas cosas nuevas o actualizan algún detalle del funcionamiento de la librería.
- **Patches:** El tercer número corresponde a arreglos o parches que corrigen defectos en las funcionalidades de la librería.

Versionado

Cada una de las versiones de las dependencias está precedida por un símbolo (\sim $^$ $*$) que indica la forma en la que deseamos que se actualice ese módulo cada vez que ejecutemos npm install

~ (solo patches)

Si escribimos en nuestro package.json: ~0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch.
- Cuando salga la versión 0.14.0 no se actualizará, ya que es una Minor Release.
- Cuando salga la versión 1.1.0 no se actualizará, ya que es una Major Release.

^ (patches y actualizaciones menores)

Si escribimos en nuestro package.json: ^0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch.
- Cuando salga la versión 0.14.0 se actualizará, ya que es una Minor Release.
- Cuando salga la versión 1.1.0 no se actualizará, ya que es una Major Release

* (todas las actualizaciones)

Si escribimos en nuestro package.json: *0.13.0

- Cuando salga la versión 0.13.1 se actualizará en nuestro proyecto, ya que es un Patch
- Cuando salga la versión 0.14.0 se actualizará, ya que es una Minor Release
- Cuando salga la versión 1.1.0 se actualizará, ya que es una Major Release

Más símbolos

> descargar/actualizar a cualquier **versión posterior** a la dada

>= descargar/actualizar a cualquier **versión igual o posterior** a la dada

<= descargar/actualizar a cualquier **versión anterior** a la dada

< descargar/actualizar a cualquier **versión igual o anterior** a la dada

Más opciones

- Finalmente, si no se pone **ningún símbolo**, se acepta únicamente la **versión especificada**.
- Si en lugar de escribir una versión, se escribe '**latest**', se descargará o actualizará siempre a la última versión disponible.
- Adicionalmente se pueden crear combinaciones con los criterios anteriores. Por ejemplo: `1.0.0 || >=1.1.0 <1.2.0` usará la versión 1.0.0 (si la encuentra) o alguna a partir de 1.1.0, pero anteriores a 1.2.0

Calculadora de edad II

Modificar el proyecto anterior para importar moment con import
(ES Modules)

Tiempo: 5 minutos

Modificar el proyecto anterior para **importar moment con import** (ES Modules)

Representar la fecha de nacimiento y la actual en formato americano
MM/DD/YY

Ejemplo de salida:

Hoy es 01/11/21

Nací el 11/29/68

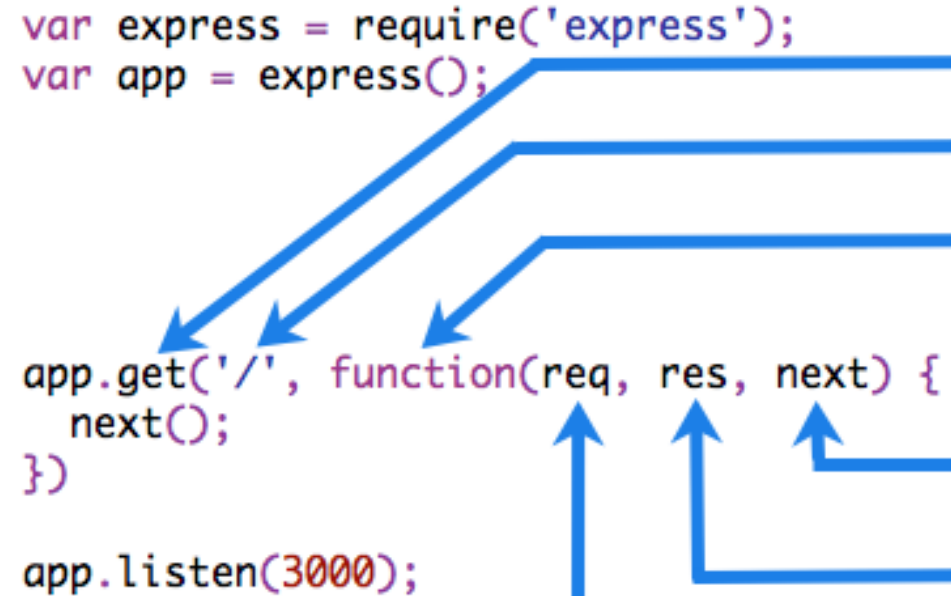
Desde mi nacimiento han pasado 52 años.

Desde mi nacimiento han pasado 19036 días.

Además: el proyecto deberá admitir sólo actualizaciones patches de su dependencia moment

Express

```
var express = require('express');  
var app = express();  
  
app.get('/', function(req, res, next) {  
  next();  
})  
  
app.listen(3000);
```

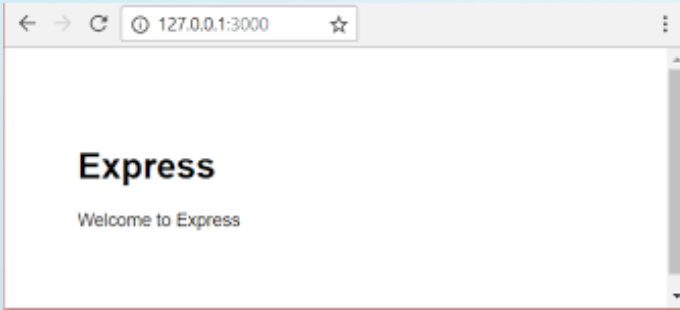


Introducción

NodeJS cuenta con módulos nativos para manejar el envío y recepción de peticiones de tipo http/s, sin embargo, usaremos para nuestra aplicación un módulo externo llamado **express**

Algunas de sus principales características son:

- ★ Es muy popular y fácil de usar.
- ★ Nos facilitará la tarea de **crear** los distintos **puntos de entrada** de nuestro **servidor**.
- ★ También permite personalizar la manera en que se maneja cada petición en forma más simple y rápida.



Express.js

Express es un **framework web** minimalista, con posibilidad de ser utilizado tanto para aplicaciones/páginas web como para aplicaciones de servicios. Como todo módulo, lo primero que debemos realizar es agregarlo como dependencia en nuestro proyecto

Instalación desde la consola:
npm install express



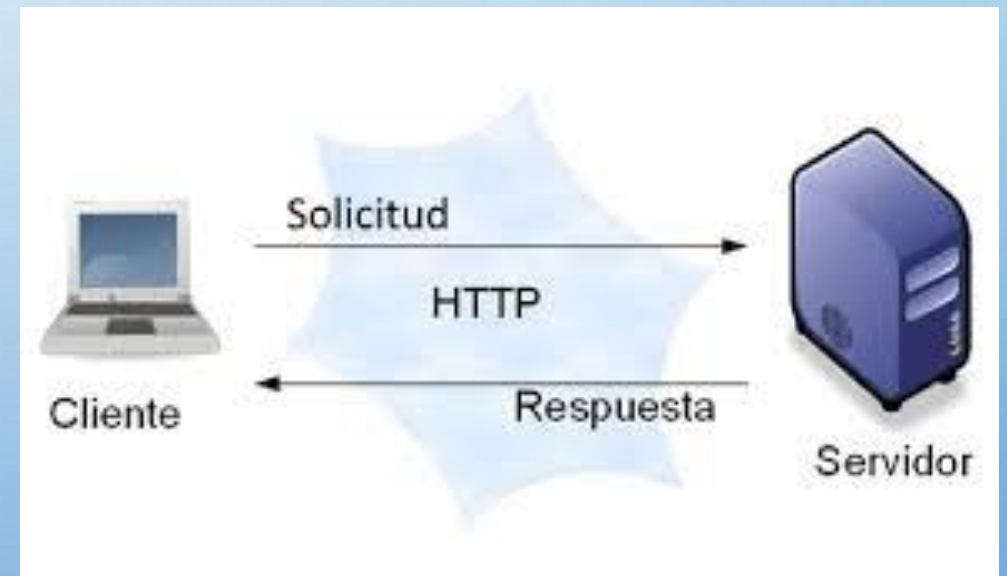
Express como framework soporte para servidores REST



{ API REST }

Introducción

Express nos permite definir, para cada tipo de petición HTTP que llegue a una determinada URL, qué acciones debe tomar, mediante la definición de un callback para cada caso que consideremos necesario incluir en nuestra API.



Uso del módulo

Para poder usar el módulo, lo primero que debemos hacer es **importarlo** al comienzo de nuestro archivo. El objeto obtenido luego del import es una **función**. Al ejecutarla, nos devolverá el **servidor** que configuraremos posteriormente con los detalles de nuestra aplicación.

Ejemplo de inicialización

```
import express from 'express'  
const app = express()
```

Conexión del servidor

Debemos indicar en qué puerto de nuestra computadora queremos que nuestra aplicación comience a escuchar peticiones. Este puerto será de uso exclusivo de nuestro servidor, y no podrá ser compartido con otras aplicaciones.

Ejemplo de conexión

```
const puerto = 8080

const server = app.listen(puerto, () => {
  console.log(`servidor inicializado en puerto ${server.address().port}`)
})
```

Si el puerto elegido es el cero (0), express elegirá un puerto al azar entre los disponibles del sistema operativo en ese momento.

Manejo de errores de conexión

Para indicar una situación de error en la puesta en marcha del servidor, podemos configurar el evento 'error' a través del método 'on' sobre la salida de 'listen'

Ejemplo de conexión (con evento de error)

```
const PORT = 8080

const server = app.listen(PORT, () => {
  console.log(`Servidor http escuchando en el puerto ${server.address().port}`)
})
server.on("error", error => console.log(`Error en servidor ${error}`))
```

*El **argumento error** del callback configurado para el **evento error**, nos da la descripción del error ocurrido.*

Configuración petición Get

Algunas peticiones no requieren el envío de **ningún dato extra** en particular para obtener el recurso buscado. Este es el caso de la ***petición GET***. Como respuesta a la petición, devolveré el **resultado** deseado en **forma de objeto**.

Ejemplo de petición GET

```
app.get('/api/mensajes', (req, res) => {  
  console.log('request recibido')  
  
  // acá debería obtener todos los recursos de tipo 'mensaje'  
  
  res.json({ msg: 'Hola mundo!' })  
})
```

Servidor con express

Crear un proyecto de servidor http en node.js que utilice la
dependencia express

Tiempo: 10/15 minutos

Crear un proyecto de servidor http en node.js que utilice la dependencia express, escuche en el puerto 8080 y tenga tres rutas get configuradas:

A) '/' en esta ruta raíz, el servidor enviará un elemento de título nivel 1 (un h1 en formato HTML) que contenga el mensaje: 'Bienvenidos al servidor express' en color azul.

B) '/visitas' donde con cada request, el servidor devolverá un mensaje con la cantidad de visitas que se hayan realizado a este endpoint. Por ej. 'La cantidad de visitas es 10'

C) '/fyh' donde se devolverá la fecha y hora actual en formato objeto:
{fyh: '11/1/2021 11:36:04'}

Mostrar por consola el puerto de escucha del servidor al momento de realizar el listen. En caso de error, representar el detalle.

Aclaración:

importar express con require (CommonJS)

SERVIDOR CON EXPRESS

Realizar un proyecto de servidor basado en node.js que utilice el middleware express e implemente tres endpoints en el puerto 8080:

- 1) Ruta get '/items' que responda un objeto con todos los productos y su cantidad total en el siguiente formato: { items: [productos], cantidad: (cantidad productos) }
- 2) Ruta get '/item-random' que devuelva un producto elegido al azar desde un array de productos que se encuentran en el archivo 'productos.txt'. El formato de respuesta será: { item: {producto} }
- 3) La ruta get '/visitas' devuelve un objeto que indica cuantas veces se visitó la ruta del punto 1 y cuantas la ruta del punto 2. Contestar con el formato: { visitas : { items: cantidad, item: cantidad } }

Usar 'productos.txt' del ejercicio anterior.

- Utilizar import para importar las dependencias necesarias.
- Representar por consola el puerto de conexión del servidor y mensajes de error si los hubiese.