



Clase 37. Programación Backend

# ***Versiones y paquetes***



## ***OBJETIVOS DE LA CLASE***

- Conocer y aprender sobre nvm.
- Conocer acerca de los administradores de paquetes en Node.

# ***CRONOGRAMA DEL CURSO***

Clase 36



**Twilio & OWASP**

Clase 37



**Creación de proyectos**

Clase 38



**Arquitectura de capas**

# ***ADMINISTRADORES DE VERSIONES***



# *¿Qué es?*



- **NVM (Node Version Manager)** es un script bash simple para administrar múltiples versiones activas de *Node* en nuestro sistema.
- Nos permite instalar múltiples versiones de *Node*, ver todas las versiones disponibles para la instalación y todas las versiones instaladas en nuestro sistema.
- También admite la ejecución de una versión específica de *Node*, puede mostrar la ruta al ejecutable donde se instaló, y mucho más.



# ***Instalar NVM en Windows***

- 1 Para instalar NVM, debemos primero descargarlo desde [la página de versiones de NVM](#).
- 2 Luego, debemos asegurarnos de eliminar la versión de Node y *npm* que tengamos instaladas en nuestro sistema, previo a instalar *nvm*.
- 3 Ejecutar el instalador y seguir los pasos que indica. Aceptar los términos, luego elegir la ruta de instalación.
- 4 Además, tendremos que definir el directorio de instalación de Node que funcionará como un enlace simbólico que apunta a la versión actualmente utilizada de Node.



# Usar NVM en Windows



Para empezar a usarlo, en una terminal usamos el comando

```
nvm
```

Obtendremos la siguiente salida.

```
Running version 1.1.7.

Usage:

nvm arch                : Show if node is running in 32 or 64 bit mode.
nvm install <version> [arch] : The version can be a node.js version or "latest" for the latest stable version.
                                Optionally specify whether to install the 32 or 64 bit version (defaults to system arch).
                                Set [arch] to "all" to install 32 AND 64 bit versions.
                                Add --insecure to the end of this command to bypass SSL validation of the remote download server.
nvm list [available]      : List the node.js installations. Type "available" at the end to see what can be installed. Aliased as ls.
nvm on                   : Enable node.js version management.
nvm off                  : Disable node.js version management.
nvm proxy [url]          : Set a proxy to use for downloads. Leave [url] blank to see the current proxy.
                                Set [url] to "none" to remove the proxy.
nvm node_mirror [url]    : Set the node mirror. Defaults to https://nodejs.org/dist/. Leave [url] blank to use default url.
nvm npm_mirror [url]     : Set the npm mirror. Defaults to https://github.com/npm/cli/archive/. Leave [url] blank to default url.
nvm uninstall <version>  : The version must be a specific version.
nvm use [version] [arch] : Switch to use the specified version. Optionally specify 32/64bit architecture.
                                nvm use <arch> will continue using the selected version, but switch to 32/64 bit mode.
nvm root [path]          : Set the directory where nvm should store different versions of node.js.
                                If <path> is not set, the current root will be displayed.
nvm version              : Displays the current running version of nvm for Windows. Aliased as v.
```



# *Usar NVM en Windows*

Para visualizar todas las versiones de Node disponibles para instalar usamos el comando:

```
nvm list available
```

CURRENT	LTS	OLD STABLE	OLD UNSTABLE
15.6.0	14.15.4	0.12.18	0.11.16
15.5.1	14.15.3	0.12.17	0.11.15
15.5.0	14.15.2	0.12.16	0.11.14
15.4.0	14.15.1	0.12.15	0.11.13
15.3.0	14.15.0	0.12.14	0.11.12
15.2.1	12.20.1	0.12.13	0.11.11
15.2.0	12.20.0	0.12.12	0.11.10
15.1.0	12.19.1	0.12.11	0.11.9
15.0.1	12.19.0	0.12.10	0.11.8
15.0.0	12.18.4	0.12.9	0.11.7
14.14.0	12.18.3	0.12.8	0.11.6
14.13.1	12.18.2	0.12.7	0.11.5
14.13.0	12.18.1	0.12.6	0.11.4
14.12.0	12.18.0	0.12.5	0.11.3
14.11.0	12.17.0	0.12.4	0.11.2
14.10.1	12.16.3	0.12.3	0.11.1
14.10.0	12.16.2	0.12.2	0.11.0
14.9.0	12.16.1	0.12.1	0.9.12
14.8.0	12.16.0	0.12.0	0.9.11
14.7.0	12.15.0	0.10.48	0.9.10

This is a partial list. For a complete list, visit <https://nodejs.org/download/release>





# *Usar NVM en Windows*

Luego, para instalar la versión específica que queramos, podemos usar el comando

```
nvm install 14.15.4
```

```
Downloading node.js version 14.15.4 (64-bit)...  
Complete  
Creating C:\Users\sdkca\AppData\Roaming\nvm\temp  
  
Downloading npm version 6.14.10... Complete  
Installing npm v6.14.10...  
  
Installation complete. If you want to use this version, type  
  
nvm use 14.15.4
```



# *Usar NVM en Windows*

- Finalmente, después de instalar una versión específica de Node, podemos cambiar *nvm* para usar esa versión con `nvm use 14.15.4`
- Deberíamos obtener en la salida: `Now using node v14.15.4 (64-bit)`
- Además, podemos verificar la versión de Node que estemos utilizando actualmente con: `node --version`
- Y para verificar la versión de *npm* `npm --version`



# *Instalar NVM en Linux*



Para instalar o actualizar *nvm* en nuestra distribución de Linux, podemos descargar el *script* de instalación automática usando las herramientas de línea de comando *wget* como se muestra.

```
# curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash  
OR  
# wget -qO- https://raw.githubusercontent.com/nvm-sh/nvm/v0.34.0/install.sh | bash
```



# ***Instalar NVM en Linux***



- Esta secuencia de comandos de instalación automática clona el repositorio nvm en ~/.nvm en su directorio de inicio y agrega los comandos de origen necesarios a las secuencias de comandos de inicio de nuestra shell, es decir, ~/.bash\_profile, ~/.zshrc, ~/.profile o ~/.bashrc, según el programa de shell que estemos utilizando.
- Verificar la instalación de nvm con el comando.

```
# command -v nvm
```



# *Usar NVM en Linux*



- Para descargar, compilar e instalar la última versión de Node, ejecutar el siguiente comando `# nvm install node`
- Para instalar una versión específica de Node, primero listar todas las versiones disponibles con el comando `# nvm ls-remote`
- Luego si instalar la versión deseada con el comando `# nvm install 10.15.3`



# *Usar NVM en Linux*



Es posible verificar todas las versiones de Node instaladas con el comando: `# nvm ls`

```
[root@tecmint ~]# nvm ls
->      v10.15.3
        v12.2.0
        system
default -> node (-> v12.2.0)
node    -> stable (-> v12.2.0) (default)
stable  -> 12.2 (-> v12.2.0) (default)
iojs    -> N/A (default)
unstable -> N/A (default)
lts/*   -> lts/dubnium (-> v10.15.3)
lts/argon -> v4.9.1 (-> N/A)
lts/boron -> v6.17.1 (-> N/A)
lts/carbon -> v8.16.0 (-> N/A)
lts/dubnium -> v10.15.3
[root@tecmint ~]#
```



# *Usar NVM en Linux*



Podemos usar la versión de Node que queramos dentro de las instaladas con el comando:

```
# nvm use node #use default
OR
# nvm use 10.15.3
```

Alternativamente, podemos usar la versión de Node directamente ejecutándola:

```
# nvm run node #run default
OR
# nvm run 10.15.3
```



# *Usar NVM en Linux*



Es importante destacar que podemos ver la ruta al ejecutable donde se instaló una versión de Node específica de la siguiente manera:

```
# nvm which 10.15.3
# nvm which 12.2.0
# nvm which system #check system-installed version of a node using "system" alias
```

```
[root@tecmint ~]# nvm which 10.15.3
/root/.nvm/versions/node/v10.15.3/bin/node
[root@tecmint ~]#
[root@tecmint ~]# nvm which 12.2.0
/root/.nvm/versions/node/v12.2.0/bin/node
[root@tecmint ~]#
[root@tecmint ~]# nvm which system
/usr/bin/node
[root@tecmint ~]#
```





# *Usar NVM en Linux*



Además, para configurar manualmente una versión de Node predeterminada que se utilizará en cualquier shell nuevo, utilizar el alias "predeterminado" como se muestra.

```
# nvm alias default 10.15.3  
# nvm alias default system  
# nvm alias default 12.2.0
```

```
[root@tecmint ~]# nvm alias default 10.15.3  
default -> 10.15.3 (-> v10.15.3)  
[root@tecmint ~]#  
[root@tecmint ~]# nvm alias default system  
default -> system  
[root@tecmint ~]# nvm alias default 12.2.0  
default -> 12.2.0 (-> v12.2.0)  
[root@tecmint ~]#
```



# *Usar NVM en Linux*



- Podemos crear un archivo de inicialización `.nvmrc` en el directorio raíz de nuestro proyecto (o en cualquier directorio principal) y agregar un número de versión de Node o cualquier otro indicador u opción de uso que `nvm` comprenda. Luego utilizamos algunos de los comandos que acabamos de ver para operar con la versión especificada en el archivo.
- Para obtener más información, podemos consultar el comando `nvm --help` o entrar al repositorio de Github de Node Version Manager: <https://github.com/nvm-sh/nvm>.

# ***ADMINISTRADORES DE PAQUETES***



# *¿De qué se trata?*



- Al desarrollar y usar aplicaciones Node, un software común en el que los desarrolladores y los usuarios generales siempre confían es un administrador de paquetes.
- Un administrador de paquetes de Node interactúa con los repositorios de paquetes en línea (que contienen bibliotecas, aplicaciones y paquetes) y ayuda de muchas maneras, inclusive con la instalación de paquetes y la administración de dependencias.
- Algunos administradores de paquetes también cuentan con componentes de administración de proyectos.



# *¿De qué se trata?*



Ayuda a especificar la biblioteca como una dependencia para su aplicación, de modo que, en cualquier sistema donde esté instalada la aplicación también se instalará la biblioteca, para que la aplicación funcione correctamente.



# ***NPM***



- **Npm** (Node Package Manager) es un administrador de paquetes de Node multiplataforma que fue desarrollado para ayudar a los desarrolladores de JavaScript a compartir fácilmente su código en forma de paquetes.
- **Npm** puede descargar paquetes y buscar actualizaciones de los paquetes que ya has instalado.



# ***NPM***



- Estos son componentes principales:
  - Interfaz de línea de comandos CLI.
  - Repositorio.
  - Website.



# ***YARN***



- **Yarn** no solo es un administrador de paquetes rápido, seguro, confiable y de código abierto, sino que también es un administrador de proyectos para proyectos estables y reproducibles.
- Tiene un hilo, que permite que la velocidad de instalación de paquetes sea mayor que en npm. Esto además, soluciona los problemas de npm de que se puedan ejecutar programas mientras se está instalando módulos y de que la seguridad no es alta.
- Cuando algunos módulos se instalan a través del hilo, pueden instalarse nuevamente sin conectarse a la red. El hilo también puede controlar la versión de la que depende el módulo, y es seguro y confiable.





# ***YARN: instalación***



Primero se instala en forma global, directamente desde npm:

```
$ npm install -g yarn
```

Podemos verificar su correcta instalación solicitando ver la versión actual:

```
$ yarn --version
```

Para inicializar un proyecto, desde el directorio raíz del mismo ejecutamos:

```
$ yarn init -y
```

Para instalar o desinstalar dependencias usamos los comando *add* y *remove*:

```
$ yarn add <package-name> [--dev]
```

```
$ yarn remove <package-name>
```



# ***PNPM***



- **Pnpm** es un administrador de paquetes de código abierto, multiplataforma, rápido y eficiente en el espacio en disco.
- A diferencia de npm y yarn, que crean un directorio plano *node\_modules*, pnpm funciona de forma un poco diferente: crea un diseño no plano *node\_modules* que utiliza enlaces simbólicos para crear una estructura anidada de dependencias.



# ***PNPM***



- Los archivos dentro de *node\_modules* están vinculados desde un único almacenamiento direccionable por contenido. Este enfoque es eficaz porque le permite ahorrar gigabytes de espacio en disco.
- **pnpm** también admite alias que le permiten instalar paquetes con nombres personalizados, completar la pestaña de la línea de comandos y usa un archivo de bloqueo llamado *pnpm-lock.yaml*.



# ***CREAR PROYECTO CON YARN***

*Tiempo: 5 minutos*

# Crear proyecto con Yarn

Desafío  
generico



Tiempo: 5 minutos

Instalar el gestor de paquetes **yarn** con **npm** en forma global.

Verificar la versión instalada de yarn.

Crear un proyecto de node.js utilizando yarn.

Instalar express con yarn.

Crear un servidor express que devuelva en su ruta raíz el mensaje: "Hola Yarn".

Crear un script start que ejecute nodemon con el punto de entrada del servidor.

Ejecutar el script start con yarn (pista: funciona igual que con **npm**).



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

***NPM***



# *Características*



- NPM ya viene incluido con la instalación de Node.
- Podemos utilizar el website para descubrir paquetes, configurar perfiles y administrar otros aspectos de nuestra experiencia npm.
- El CLI se ejecuta desde una terminal y es la forma en que la mayoría de los desarrolladores interactuamos con npm (por ejemplo: *npm install*).
- Este es un repositorio en línea seguro para la publicación de proyectos Node de código abierto, como bibliotecas y aplicaciones.





# *Características*



- Para instalar y publicar paquetes, los desarrolladores utilizan un cliente de línea de comandos llamado npm, que también se utiliza para la gestión de versiones y la gestión de dependencias.
- Podemos usar de forma gratuita, una opción que nos permite crear paquetes públicos, auditar dependencias, publicar actualizaciones, entre otros.



# ***Características***



- **npm** se usa para instalar todos los demás administradores de paquetes de Node que vimos.
- **npm** también es compatible con la seguridad de JavaScript, y puede integrarse con herramientas de terceros.



# Usos



Npm tiene muchos usos, algunos de ellos son:

- Adaptar paquetes de código para aplicaciones o incorporarlos tal como están.
- Descargar herramientas independientes que podemos usar de inmediato.
- Ejecutar paquetes sin descargarlos usando el comando *npx*.



# *Usos*



Además:

- Actualizar las aplicaciones fácilmente cuando se actualice el código.
- Compartir código con cualquier usuario de npm, de cualquier lugar.
- Restringir el código a desarrolladores específicos.
- Administrar múltiples versiones de código y de dependencias.



# ***package.json***



Cada proyecto en JavaScript – ya sea Node o una aplicación de navegador – puede ser enfocado como un paquete *npm* con su propia información de paquete y su archivo *package.json* para describir el proyecto.



# ***package.json***



- *package.json* se generará cuando se ejecute *npm init* para inicializar un proyecto JavaScript/Node, con estos metadatos básicos proporcionados por los desarrolladores:
  - ***name***: el nombre de tu librería/proyecto JavaScript
  - ***version***: la versión de tu proyecto.
  - ***description***: la descripción del proyecto
  - ***license***: la licencia del proyecto



# ***npm scripts***



- *package.json* también soporta la propiedad *scripts* que puede definirse para ejecutar herramientas de línea de comandos que se instalan en el contexto local del proyecto. Por ejemplo, la porción de scripts de un proyecto npm puede tener un aspecto similar a este:

```
{
  "scripts": {
    "build": "tsc",
    "format": "prettier --write **/*.ts",
    "format-check": "prettier --check **/*.ts",
    "lint": "eslint src/**/*.ts",
    "pack": "ncc build",
    "test": "jest",
    "all": "npm run build && npm run format && npm run lint && npm run pack && npm test"
  }
}
```

# ***Dependencias vs devDependencias***



- Ambos vienen en forma de objetos clave-valor (key-value) con los nombre de las librerías npm como clave y sus versiones en formato semántico como valor.
- Las dependencias se instalan mediante el comando `npm install` con las banderas `--save` y `--save-dev`.
- Están pensadas para ser usadas en entornos de producción y desarrollo/prueba respectivamente.





# ***package-lock.json***



Este archivo describe las versiones exactas de las dependencias utilizadas en un proyecto de JavaScript *npm*. Si *package.json* es una etiqueta descriptiva genérica, *package-lock.json* es una tabla de ingredientes.



# ***package-lock.json***



- Así como no solemos leer la tabla de ingredientes de un producto, *package-lock.json* no está pensado para ser leído línea por línea por los desarrolladores (a menos que estemos desesperados por resolver problemas de "funciona en mi máquina").
- *package-lock.json* es usualmente generado por el comando *npm install*, y también es leído por nuestra herramienta npm CLI para asegurar la reproducción de los entornos de construcción para el proyecto con *npm ci* (*clean install*) en lugar de *npm i* (*regular install*).



# *Npm install*



- Este es el comando más utilizado en el desarrollo de aplicaciones JavaScript/Node hoy en día.
- Por defecto, `npm install <nombre del paquete>` instalará la última versión con el signo `^`. Un `npm install` dentro del contexto de un proyecto `npm` descarga los paquetes en la carpeta `node_modules` del proyecto según las especificaciones de `package.json`, actualizando la versión del paquete (y a su vez regenerando `package-lock.json`) donde pueda basándose en la coincidencia de las versiones `^` y `~`.
- Podemos especificar una bandera global `-g` si deseamos instalar un paquete en el contexto global que podremos utilizar en cualquier lugar de nuestra computadora.



# *Npm ci*



- Al igual que si *package-lock.json* no existe ya en el proyecto se genera cada vez que se llama a *npm install*, ***npm ci*** consume este archivo para descargar la versión exacta de cada paquete individual del que depende el proyecto.
- Así es cómo podemos asegurarnos de que el contexto de nuestro proyecto se mantiene exactamente igual en las diferentes máquinas, ya sean nuestras laptops utilizadas para el desarrollo o CI (Integración Continua).



# *Npm Audit*



- Con el enorme número de paquetes que se han publicado y que pueden ser fácilmente instalados, los paquetes npm son susceptibles a los malos autores con intenciones maliciosas.
- Al darse cuenta de que había un problema en el ecosistema, la organización *npm.js* tuvo la idea de ***npm audit***. Mantienen una lista de lagunas de seguridad para que los desarrolladores puedan auditar sus dependencias con el uso del comando *npm audit*.



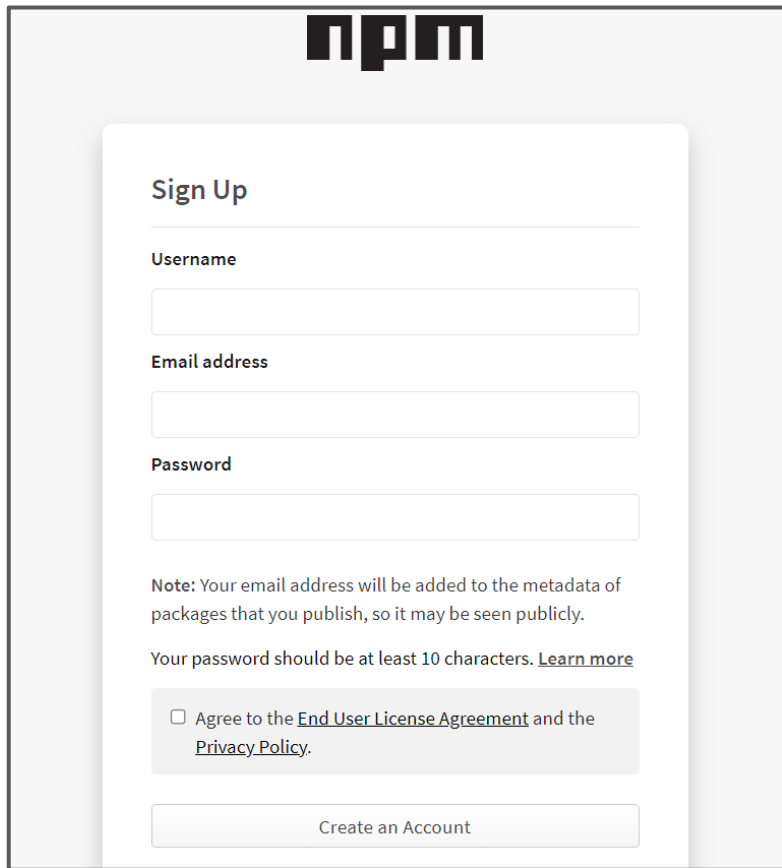
# ***Npm Audit***



- ***npm audit*** da a los desarrolladores información sobre las vulnerabilidades y si hay versiones con correcciones a las que actualizar.
- Si las correcciones están disponibles en las próximas actualizaciones de versiones que no sean de última hora, se puede utilizar ***npm audit*** para actualizar automáticamente las versiones de las dependencias afectadas.



# *Empezando a usar npm*



The screenshot shows the npm website's sign-up interface. At the top is the npm logo. Below it is a 'Sign Up' heading. The form contains three input fields: 'Username', 'Email address', and 'Password'. Below the 'Email address' field is a note: 'Note: Your email address will be added to the metadata of packages that you publish, so it may be seen publicly.' Below the 'Password' field is a note: 'Your password should be at least 10 characters. [Learn more](#)'. At the bottom of the form is a checkbox labeled 'Agree to the [End User License Agreement](#) and the [Privacy Policy](#)'. Below the checkbox is a 'Create an Account' button.

- En primer lugar, vamos a crearnos una cuenta en npm. Luego veremos cómo subir nuestros propios paquetes.
- Entramos entonces en la página de npm con el link <https://www.npmjs.com/signup> y llenamos el formulario de registro.



# *Empezando a usar npm*



Vemos entonces, una vez que tenemos nuestro usuario de **npm**, todo lo que podemos hacer desde allí.

The screenshot shows the npm website interface. At the top, there's a navigation bar with links for Products, Pricing, Documentation, and Community. Below this is a search bar with the text "Search packages" and a "Search" button. The main content area is divided into three sections:

- Popular libraries:** A list of popular libraries including lodash, react, chalk, tslib, axios, express, commander, request, moment, and react-dom.
- Discover packages:** A grid of categories for discovering packages, including Front-end, Back-end, CLI, Documentation, CSS, Testing, IoT, Coverage, Mobile, Frameworks, Robotics, and Math.
- By the numbers:** A section showing statistics for packages, including the total number of packages (1,620,550), downloads in the last week (30,146,893,608), and downloads in the last month (132,135,232,734).





# *Creando nuestro paquete npm*



Como usuarios de npm, podemos crear paquetes sin ámbito para usar en nuestro propios proyectos y publicarlos en el repositorio público de npm para que otros los usen en sus proyectos.

- En primer lugar, debemos chequear si tenemos instalada la versión actualizada de npm, sino, usamos este comando para instalarla:

```
npm install npm@latest -g
```



# ***Creando nuestro paquete npm***




- Para comenzar a crear un paquete para npm, primero debemos crearnos el proyecto de Node en nuestra computadora.
- Recordar siempre, crear el archivo README con información acerca del paquete, qué funcionalidades tiene, y cómo lo podemos utilizar.



# ***Creando nuestro paquete npm***



- Es recomendable subir nuestro paquete a un repositorio remoto por ejemplo en *Github*  .
- Antes de publicar nuestro paquete en npm, se recomienda testearlo.



# ***Creando nuestro paquete npm***



- Una vez que lo hayamos probado y funcione todo correctamente, en la consola al directorio de nuestro paquete e iniciamos sesión en npm con el comando ***npm login*** y nuestros datos de usuario y contraseña de la cuenta que nos creamos en npm.
- Luego, lo publicamos en npm con el comando ***npm publish***.
- Para removerlo, utilizamos el comando:

***npm unpublish <nombre-paquete>***



# *Creando nuestro paquete npm*



- Ahora que ya publicamos nuestro paquete, podemos visualizarlo desde la página de npm entrando a [https://npmjs.com/package/\\*package-name](https://npmjs.com/package/*package-name).
- Hay que tener en cuenta que no puede haber dos paquetes de NPM con el mismo nombre, por lo que, si el que pusimos ya existía vamos a tener que cambiarlo.
- Además, si actualizamos el paquete, cuando lo hagamos debemos volver a ejecutar el comando de `npm publish`. Hay que asegurarse de actualizar el `package.json` a la última versión para que npm no lo rechace al querer publicarlo.



# ***CREAR DEPENDENCIA CON NPM***

*Tiempo: 10 minutos*

# Crear dependencia con npm

Desafío  
generico



Tiempo: 10 minutos

Realizar un proyecto que subiremos a npm como dependencia propia.

Este paquete debe exportar cuatro funciones para realizar estas operaciones básicas entre dos números: suma, resta, multiplicación y división. Subir el proyecto a npm.

Verificar que la dependencia se encuentre en npm

(<https://www.npmjs.com/package/nombre-del-paquete>)

Abrir el proyecto servidor del desafio anterior e instalar con npm la dependencia recién creada.

Importar la dependencia en este proyecto y realizar un endpoint get por cada operación ingresando los valores por query params. Probar el correcto funcionamiento de esas rutas.

***¿PREGUNTAS?***

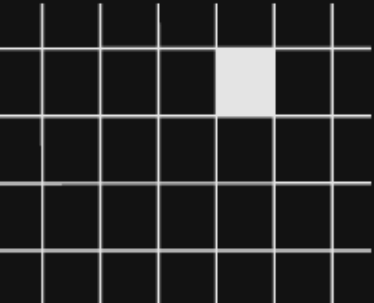






***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Administradores de paquetes de Node.
  - Nvm, qué es y cómo usarlo.
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDO LA EDUCACIÓN***