Debbuging & Easier Development

Aprenderemos debugging, herramientas y tips and tricks que nos ayudarán a escribir aplicaciones de NodeJs de manera eficiente.

More on debugging Node.js: https://nodejs.org/en/docs/guides/debugging-getting-started/

Debugging Node in Visual Studio Code: https://code.visualstudio.com/docs/nodejs/nodejs-debugging

1. Entendiendo los NPM Scripts

https://docs.npmjs.com/cli/v8/commands/npm-init/

Siempre que levantamos el servidor teníamos que utilizar "node app.js", si bien no conlleva mucho trabajo existe la posibilidad de definir algunos scripts en un proyecto de NodeJs que nos pueden ayudar con este tipo de tareas y muchas otras.

Para habilitar esta opción tenemos que user NPM (Node Package Manager), es instalado junto con NodeJs. Utilizamos NPM para instalar paquetes de terceros, es decir, paquetes que no están en incluidos en los Core Modules, ya sean pequeños features, funcionalidades complejas, etc.

Con npm init podemos setear un paquete nuevo o uno existente. Creará un archivo de configuración package.json que será el encargado de indicarle al NPM que tipo de paquetes tiene el proyecto, la versión, descripción, etc. También tiene una sección de scripts que colocando el nombre del script en el proyecto donde se hizo el npm init podemos ejecutar instrucciones simplemente con el nombre.

Ejecutamos el script poniendo npm "NOMBRE" si el nombre es especial para npm. Si el nombre no es uno especial que reconoce NPM tengo que utiliza la sintaxis "npm run NOMBRE"

2. Instalando paquetes de terceros

https://github.com/apex/up/issues/603

https://stackoverflow.com/questions/9268259/how-do-you-prevent-install-of-devdependencies-npm-modules-for-node-js-package

NPM nos permite instalar paquetes de terceros desde la bodega de NPM repository. La idea de los paquetes es tener un set de configuraciones y no tener que reinventar la rueda. Estos paquetes además de instalarse son manejados por NPM.

Para el ejemplo de esta sección instalaremos un paquete que nos permite realizar cambios en el código y no tener que tirar y levantar el servidor para que tenga efecto.

Los paquetes los podemos encontrar googleando, en foros, buscando en la página oficial de npm, etc. Lo importante es que estos paquetes se instalan con la nomenclatura que se menciona a continuación y, muy importante, generalmente tiene la documentación en el repositorio para saber cómo se usa y sus funcionalidades.

"npm install NOMBRE_DEL_PAQUETE". Por default se instala la versión más reciente.

Los paquetes de desarrollo son paquetes que ayudarán durante el desarrollo y la producción de dependencias. Nodemon es un paquete de desarrollo

Podemos aclarar que cuando qué tipo de dependencia es. Esto se logra con la flag "--save-dev". Con la flag "--save" inidicamos en que es un paquete de producción y con la "-g" lo instala de manera global en la máquina para que se use en cualquier lugar.

Es posible que un paquete instale otras dependencias necesarias para su funcionamiento.

npm install nodemon --save-dev -> Instala el paquete de manera local y en el package.json inidica que es una dependencia de desarrollo.

```
"devDependencies": {
    "nodemon": "^2.0.16"
}
```

package-lock.json guarda las versiones exactas que instalamos.

npm install ira al package.json y hará un update/instalación a todos los paquetes que encuentre.

3. Global Features vs Core Modules vs Third-Party Modules

The last lectures contained important concepts about available Node.js features and how to unlock them.

You can basically differentiate between:

Global features: Keywords like const or function but also some global objects like process

Core Node.js Modules: Examples would be the file-system module ("fs"), the path module ("path") or the Http module ("http")

Third-party Modules: Installed via npm install - you can add any kind of feature to your app via this way

Global features are always available, you don't need to import them into the files where you want to use them.

Core Node.js Modules don't need to be installed (NO npm install is required) but you need to import them when you want to use features exposed by them.

Example:

```
const fs = require('fs');
```

You can now use the fs object exported by the "fs" module.

Third-party Modules need to be installed (via npm install in the project folder) AND imported.

Example (which you don't need to understand yet - we'll cover this later in the course):

```
// In terminal/ command prompt

npm install --save express-session

// In code file (e.g. app.js)

const sessions = require('express-session');
```

4. Global & Local npm Packages

In the last lecture, we added nodemon as a local dependency to our project.

The good thing about local dependencies is that you can share projects without the node_modules folder (where they are stored) and you can run npm install in a project to then re-create that node_modules folder. This allows you to share only your source code, hence reducing the size of the shared project vastly.

The attached course code snippets also are shared in that way, hence you need to run npm install in the extracted packages to be able to run my code!

I showed that nodemon app.js would not work in the terminal or command line because we don't use local dependencies there but global packages.

You could install nodemon globally if you wanted (this is NOT required though - because we can just run it locally): npm install -g nodemon would do the trick. Specifically the -g flag ensures that the package gets added as a global package which you now can use anywhere on your machine, directly from inside the terminal or command prompt.

Si instalamos un paquete de manera local deberíamos generar un script en el package.json para correrlo.

5. Understanding different Error Types

-Errors de Sintaxis

-Errores de Runtime

-Errores de Lógica

6. Debugging Node.Js in Visual Studio Code

https://code.visualstudio.com/docs/nodejs/nodejs-debugging

El debugger es muy parecido al de Visual Studio. Establecemos breakpoints y vamos a poder movernos con F11 y F10 a través del código donde se ejecute. Hay que tener en cuenta que hay que ejecutar el debugger (F5) en el archivo que llame al breakpoint. En el caso de este ejemplo la app llama al routes. Pero puede suceder que desde la misma app debugeemos.

VSC tiene una consola de depuración que nos da muchísima información, como lo que contiene un objeto, el stack de llamadas, etc.

En la solapa watch podemos escribir alguna variable para ver su comportamiento.

Podemos cambiar la configuración del comportamiento del debugger para que, por ejemplo, cuando tenemos el nodemon poder reiniciar el debugger automáticamente. En Ejecutar->Agregar Configuración podemos cambiar el "restart"=true, "runTimeExecutable" = "nodemon" y otras varias configuraciones dentro del array de configuraciones.

7. Resumen del módulo

npm

- -npm es acrónimo de "Node Pacakage Manager" y permite administrar proyectos de NodeJs y sus dependencias
- -Podemos inicializar un proyecto con "npm init"
- -npm scripts pueden ser definidas en el package.json para generar shortcuts para tareas comunes/comandos.

Instalamos paquetes de terceros con npm.

- Proyectos de Node usualmente no utilizan solamente los core modules. También es necesario utilizar paquetes de terceros.
- Se instalan a través del NPM y su bodega.
- Podemos diferenciarlos entre producción (--save), desarrollo (--save-dev) y globales (-g)

Tipos de errores.

- Sintaxis, Runtime y de Lógica pueden romper la app.
- Errores de sintaxis y runtime lanzan errores o mensajes.
- Errores de logica pueden resolverse con testing y la ayuda del debugger.

Debbuging

- Utilizamos el debugger de VSC para ir paso a paso en el codigo.
- Analizamos variables en el runtime.
- Mirar y manipular variables.
- Poner breakpoints para debuggear.