

TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Operaciones CRUD.
- Conceptos utilizados en CLI.
- Utilización de módulo yargs.

Cuando nos referimos al desarrollo de aplicaciones web, muchas veces aparece el concepto de CRUD. Éste es un acrónimo para los siguientes conceptos:

- **Create:** Crear.
- **Read:** Leer.
- **Update:** Modificar.
- **Delete:** Eliminar.

Al hablar de objetos de datos y persistencia, necesitamos, mínimo, de estas cuatro acciones, para poder tener un modelo de datos efectivo. Por lo tanto, una aplicación que maneje cierta capa de persistencia de datos debería cumplir con cada una de éstas para los datos que están involucrados en la aplicación.

El patrón **CRUD** es muy común a la hora de construir aplicaciones, pues provee un gran espacio de trabajo, que permite recordar a los desarrolladores como construir modelos de datos completos y escalables.

Imaginemos que tenemos que desarrollar una aplicación de una tienda de comics. Para ésta, sabemos que el modelo de datos principal es el de un comic con sus características. Consideremos que un objeto "comic", tiene la siguiente estructura:

```
1  "comic": {  
2    "id": <Numero>,  
3    "titulo": <String>,  
4    "autor": <String>,  
5    "issn": <Numero>  
6    "cantidad": <Numero>  
7  }
```

Para poder hacer viable el uso de esta aplicación, deberíamos asegurarnos de que existan las funciones necesarias para completar todas las operaciones dentro del concepto **CRUD**.

- **Create - Crear:** para esta opción, debemos tener una función que sea capaz de crear o ingresar cada vez que un nuevo comic llegue a la tienda. Ésta debe entregar los valores del título, autor, issn y cantidad. Luego de que sea llamada, tiene que estar disponible este nuevo comic dentro de nuestra base de datos, idealmente con un id único para su posterior identificación.
- **Read - Leer:** aquí necesitamos una función que sea capaz de hacer una llamada a la base de datos para obtener todos los libros, o bien, un solo libro identificándolo por: id, titulo, autor, o ISSN. Ésta solo debe traer de vuelta la información, y no realizar ningún tipo de modificación.
- **Update - Modificar:** debe existir también una función encargada de modificar la información de un comic, en caso de ser necesario. Ésta obtendrá los datos nuevos, y realizará los cambios al comic ya existente en la base de datos.
- **Delete - Eliminar:** por último, debe existir también una función encargada de eliminar un comic del catálogo de productos. Ésta recibirá uno o más campos para identificar el comic en cuestión, y luego eliminarlo de la base de datos.

Con estas cuatro acciones, seremos capaces de obtener una aplicación funcional para la gestión del modelo de datos de una tienda de comics. Si falta alguna de ellas, la funcionalidad real de la aplicación se ve en peligro. Por ejemplo: si es que un comic se llega a acabar, y ya no hay más unidades para vender, no tiene sentido que éste siga existiendo en la base de datos. Si no pudiésemos eliminarlo, ocasionaría un problema para la tienda, al seguir ofreciendo un producto que ya no existe. Problemáticas similares ocurren al imaginar casos en donde no podamos llevar a cabo alguna de las acciones mencionadas.

CONSTRUYENDO CLI

Como ya lo hemos mencionado anteriormente, **CLI** significa "Command Line Interface", y como definición, es cualquier programa con el cual se interactúa por medio de la línea de comandos de un sistema operativo, encargándose de interpretar los comandos ingresados, y actuar acorde a ellos. En el CUE anterior construiste un CLI que permitía acceder y cambiar información en el sistema de archivos de tu computador. Observamos como **node** tiene la capacidad de recibir argumentos de entrada en el momento de su ejecución, y de acceder a los argumentos desde el objeto **process** en la propiedad **argv**

```
JS index.js X
JS index.js
1 console.log(process.argv);
2
3
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

at internal/main/run_main_module.js:17:47

C:\Users\nodeporcomandos>node index.js uno 2 tres 4 cinco

```
[
  'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\nodeporcomandos\\index.js',
  'uno',
  '2',
  'tres',
  '4',
  'cinco'
]
```

Si bien no hay nada incorrecto con esta forma de acceder a los argumentos, se vuelve un poco más complejo al momento de ingresar más parámetros, y de definirlos dentro de **node**.

```
JS index.js X
JS index.js > ...
1 const argumentos = process.argv.slice(2);
2
3 const argumentoUno = argumentos[0];
4 const argumentoDos = argumentos[1];
5 const argumentoTres = argumentos[2];
6 const argumentoCuatro = argumentos[3];
7 const argumentoCinco = argumentos[4];
8
9 console.log(argumentoUno, argumentoDos, argumentoTres, argumentoCuatro, argumentoCinco);
10
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js uno 2 tres 4 cinco

uno 2 tres 4 cinco

C:\Users\nodeporcomandos>

Además, es muy común que un CLI dé opciones de ejecución con las llamadas **“flags”**, como cuando instalaste **nodemon** con la opción **“-g”**.

```
C:\Users\nodeporcomandos>npm install nodemon -g  
[.....] / rollbackFailedOptional: verb npm-session dc0d5978141eac82
```

Todo lo anterior se puede construir sin problemas utilizando **node**, pero tomaría más tiempo del necesario. Para esto, existe un módulo llamado **yargs**, el cual incorpora todos estos conceptos, y te permite utilizarlos de forma rápida y efectiva.

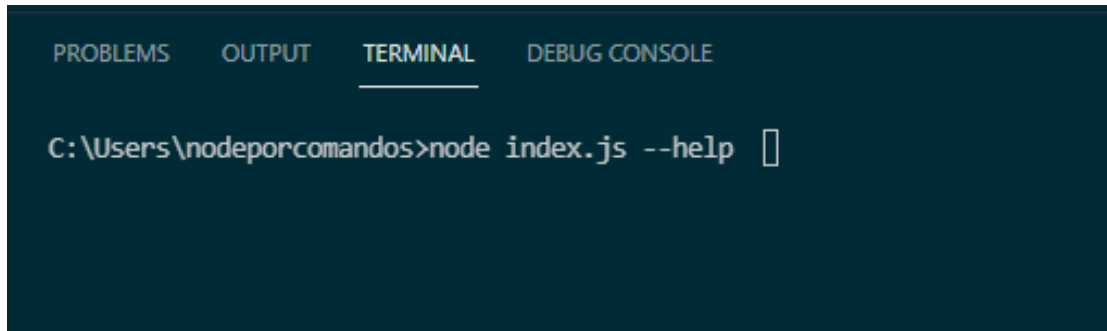
Instalaremos **yargs** en un nuevo proyecto, y conoceremos sus métodos principales.

```
C:\Users\nodeporcomandos>npm install yargs  
[.....] / rollbackFailedOptional: verb npm-session 20b87d839ed5ab6b
```

Yargs ofrece varios métodos para hacer más fácil el manejo de opciones, revisaremos una estructura básica para añadir un comando. Lo primero que se debe mencionar, es que todos los métodos que utiliza **yargs** van encadenados. En el siguiente código, podemos ver el método **command()**, que recibe como primer argumento el comando que será utilizado en la línea de comandos; y como segundo parámetro, recibe una descripción respecto a la utilidad del comando. Luego, existe un método **help**, el cual nos da automáticamente la opción **“--help”**, con la cual podemos obtener ayuda al momento de ejecutar nuestro programa.

```
JS index.js  X  
JS index.js > ...  
1  const yargs = require('yargs')  
2  const args = yargs  
3    .command('hola', 'Saludar al programador')  
4    .help()  
5    .argv  
6
```

Para ejecutar nuestro programa, y ver la opción de ayuda, utilizamos el siguiente formato de ejecución por la consola.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js --help
```

Lo cual nos devuelve lo siguiente.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js --help
index.js [command]

Comandos:
  index.js hola  Saludar al programador

Opciones:
  --version  Muestra número de versión  [booleano]
  --help    Muestra ayuda                [booleano]

C:\Users\nodeporcomandos>
```

Como puedes ver, tenemos el comando “hola” que hemos definido, y además dos opciones por defecto. Agregaremos un argumento, más al método `command()`, el cual es un objeto que nos permite definir el nombre de las opciones que ese comando tendrá. En el siguiente código, estamos definiendo la opción “amigable”, con la descripción “Realizar saludo amigable”.

```
JS index.js x
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4    →  .amigable: {
5    →    describe: "Realizar un saludo amigable"
6    →  }
7  })
8    .help()
9    .argv
10
```

Y si esta vez corremos nuestro programa utilizando el comando hola, junto a la opción "--amigable", obtenemos el siguiente resultado.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
C:\Users\nodeporcomandos>node index.js hola --help ←
index.js hola

Saludar al programador

Opciones:
--version  Muestra número de versión          [booleano]
--help     Muestra ayuda                      [booleano]
--amigable Realizar un saludo amigable ←

C:\Users\nodeporcomandos>
```

También podemos definir un alias para la opción amigable, lo que nos permitirá utilizar una "opción corta", empleando un solo guion.

```
JS index.js x
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4      amigable: {
5        describe: "Realizar un saludo amigable",
6        alias: "a"
7      }
8    })
9    .help()
10   .argv
11
```

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

```
C:\Users\nodeporcomandos>node index.js hola --help
index.js hola

Saludar al programador

Opciones:
  --version  Muestra número de versión      [booleano]
  --help     Muestra ayuda                  [booleano]
  -a, --amigable Realizar un saludo amigable

C:\Users\nodeporcomandos>
```

Hasta ahora, nuestro programa no está realizando ninguna acción, más que darnos ayuda con las opciones. Veamos cómo implementar código, haciendo uso del comando y sus opciones. El método `command()`, acepta un último argumento, el cual es una función que se será utilizada al momento de ejecutar el programa con el comando.

```
JS index.js x
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4      amigable: {
5        describe: "Realizar un saludo amigable",
6        alias: "a"
7      }
8    }, () => {
9      console.log("Hola programador")
10    })
11   .help()
12   .argv
13
```

Y al ejecutar el programa junto con la opción "hola", recibimos.

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js hola
Hola programador

C:\Users\nodeporcomandos>

```

La función que hemos agregado al método `command()` recibe el objeto `"argv"`, el cual contiene la lista de las opciones y comandos ingresados por la línea de comandos. Cuando pasamos una opción, en caso de ser ingresada, esta puede evaluarse como verdadero o falso.

```

JS index.js  x
JS index.js > [?] args > yargs.command('hola', 'Saludar al programador') callback
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4      amigable: {
5        describe: "Realizar un saludo amigable",
6        alias: "a"
7      }
8    }, (argv) => {
9      if(argv.amigable){
10        return console.log("Hola amigo programador !");
11      }
12      console.log("Hola programador");
13    })
14    .help()
15    .argv
16

```

Esto nos permite ejecutar nuestro programa de la siguiente forma, y obtener el resultado a continuación.



```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js hola --amigable
Hola amigo programador !

C:\Users\nodeporcomandos>node index.js hola
Hola programador

C:\Users\nodeporcomandos>
```

Las opciones también pueden pasarse con un valor relacionado a ellas.

```
JS index.js  X
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4      |   amigable: {
5      |     |   describe: "Realizar un saludo amigable",
6      |     |   alias: "a"
7      |   }
8    }, (argv) => {
9      |   console.log(argv.amigable);
10   })
11   .help()
12   .argv
13
```

Y ejecutando nuestro programa con la siguiente sintaxis, podemos ver el resultado.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js hola --amigable="Este es un saludo amigable"
Este es un saludo amigable

C:\Users\nodeporcomandos>
```

En este último ejemplo, es importante notar que, si quieres enviar un valor relacionado a una opción, donde la cadena de texto contenga espacios, debes usar comillas dobles para que la línea de comandos entienda que eso es un solo valor.

Podemos agregar la característica a una opción para que esta sea obligatoria, utilizando la propiedad **"demandOption"**, igual a **"true"**.

```
JS index.js x
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3  .command('hola', 'Saludar al programador', {
4    amigable: {
5      describe: "Realizar un saludo amigable",
6      alias: "a",
7      demandOption: true
8    }
9  }, (argv) => {
10    console.log(argv.amigable);
11  })
12  .help()
13  .argv
14
```

Esta opción permite realizar una validación, y en caso de que la opción no sea añadida, recibiremos el siguiente mensaje.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js hola
index.js hola

Saludar al programador

Opciones:
  --version  Muestra número de versión      [booleano]
  --help     Muestra ayuda                  [booleano]
  -a, --amigable Realizar un saludo amigable [requerido]

Falta argumento requerido: amigable
C:\Users\nodeporcomandos>
```

Agregaremos un comando más a nuestro programa.

```
JS index.js x
JS index.js > ...
1  const yargs = require('yargs')
2  const args = yargs
3    .command('hola', 'Saludar al programador', {
4      amigable: {
5        describe: "Realizar un saludo amigable",
6        alias: "a",
7        demandOption: true
8      }
9    }, (argv) => {
10     console.log(argv.amigable);
11   })
12   .command('despedir', 'Despide al programador', {
13     gracioso: {
14       describe: "Realizar un despido gracioso",
15       alias: "g"
16     }
17   }, (argv) => {
18     if(argv.gracioso){
19       return console.log("Hasta la vista, programababy");
20     }
21     console.log("Hasta pronto !");
22   })
23   .help()
24   .argv
25
```

Y si ejecutamos el nuevo comando, veremos lo siguiente.

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
C:\Users\nodeporcomandos>node index.js despedir
Hasta pronto !
C:\Users\nodeporcomandos>
```

Ahora, utilizando la opción "--gracioso", pero esta vez con el alias.

```

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js despedir -g
Hasta la vista, programababy

C:\Users\nodeporcomandos>

```

Hasta este momento, nuestro programa ha empezado a usar varias líneas de código, implementando el uso de solo dos comandos. Podemos tratar de hacerlo más legible, definiendo los objetos de configuración de los comandos y sus funciones aparte.

Primero podemos definir los objetos en sus propias variables.

```

JS index.js  X
JS index.js > ...
1  const yargs = require('yargs')
2
3  const holaConfig = {
4    amigable: {
5      describe: "Realizar un saludo amigable",
6      alias: "a",
7      demandOption: true
8    }
9  }
10 const despedirConfig = {
11   gracioso: {
12     describe: "Realizar un despido gracioso",
13     alias: "g"
14   }
15 }
16
17 const args = yargs.command('hola', 'Saludar al programador', holaConfig, (argv) => {
18   console.log(argv.amigable);
19 })
20 .command('despedir', 'Despide al programador', despedirConfig, (argv) => {
21   if (argv.gracioso) {
22     return console.log("Hasta la vista, programababy");
23   }
24   console.log("Hasta pronto !");
25 })
26 .help()
27 .argvs
28

```

Y luego, las funciones.

```

JS indexjs x
JS indexjs > ...
1  const yargs = require('yargs')
2
3  const holaConfig = {
4    amigable: {
5      describe: "Realizar un saludo amigable",
6      alias: "a",
7      demandOption: true
8    }
9  }
10 const despedirConfig = {
11   gracioso: {
12     describe: "Realizar un despido gracioso",
13     alias: "g"
14   }
15 }
16 const funcionHola = (argv) => {
17   console.log(argv.amigable);
18 }
19 const funcionDespedir = (argv) => {
20   if (argv.gracioso) {
21     return console.log("Hasta la vista, programababy");
22   }
23   console.log("Hasta pronto !");
24 }
25
26 const args = yargs
27   .command('hola', 'Saludar al programador', holaConfig, (argv) => funcionHola(argv))
28   .command('despedir', 'Despide al programador', despedirConfig, (argv) => funcionDespedir(argv))
29   .help()
30   .argv
31
  
```

Si bien hemos aumentado las líneas de código, se ha logrado una gran legibilidad en él, sobre todo en la porción en donde se definen los comandos.

```

const args = yargs
  .command('hola', 'Saludar al programador', holaConfig, (argv) => funcionHola(argv))
  .command('despedir', 'Despide al programador', despedirConfig, (argv) => funcionDespedir(argv))
  .help()
  .argv
  
```

El lenguaje de **yargs** depende de tu sistema operativo. En el caso de que en tu ambiente local el mensaje del método **demandOption** aparezca en inglés, u otro idioma, puedes añadir el método **locale()** para cambiarlo.



```

JS indexjs x
JS indexjs > | args
26 const args = yargs
27 .command('hola', 'Saludar al programador', holaConfig, (argv) => funcionHola(argv))
28 .command('despedir', 'Despide al programador', despedirConfig, (argv) => funcionDespedir(argv))
29 .help()
30 .locale('es') // Español
31 .argv
32

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js --help
index.js [command]

Comandos:
  index.js hola      Saludar al programador
  index.js despedir  Despide al programador

Opciones:
  --version  Muestra número de versión      [boolean]
  --help     Muestra ayuda                  [boolean]

C:\Users\nodeporcomandos>
  
```

```

JS indexjs x
JS indexjs > | args
26 const args = yargs
27 .command('hola', 'Saludar al programador', holaConfig, (argv) => funcionHola(argv))
28 .command('despedir', 'Despide al programador', despedirConfig, (argv) => funcionDespedir(argv))
29 .help()
30 .locale('en') // Ingles
31 .argv
32

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js --help
index.js [command]

Commands:
  index.js hola      Saludar al programador
  index.js despedir  Despide al programador

Options:
  --version  Show version number      [boolean]
  --help     Show help                [boolean]

C:\Users\nodeporcomandos>
  
```

```

JS indexjs x
JS indexjs > ...
26 const args = yargs
27 .command('hola', 'Saludar al programador', holaConfig, (argv) => funcionHola(argv))
28 .command('despedir', 'Despide al programador', despedirConfig, (argv) => funcionDespedir(argv))
29 .help()
30 .locale('pirate') // Pirata en ingles ??
31 .argv
32

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

C:\Users\nodeporcomandos>node index.js --help
index.js [command]

Choose yer command:
  index.js hola      Saludar al programador
  index.js despedir  Despide al programador

Options for me hearties!
  --version  'Tis the version ye be askin' fer      [boolean]
  --help     Parlay this here code of conduct      [boolean]

C:\Users\nodeporcomandos>
  
```