Universidad Politécnica Salesiana

Ingeniería en Ciencias de la Computación

BASES DE NODE JS

Informe 03

Autor:
Ricardo Romo

April 30, 2020

1. Realizar un programa con node que nos permita crear tareas recibiendo como parámetros: el nombre de la tarea, su estado(**true** cumplida, **false** no realizada), además dependiendo de la opción que elijamos podremos crear una tarea, actualizar, borrar y listar. Además, al crearse las tareas, deberán guardarse en un archivo **json** con las tareas ya antes creadas.

Para la realización de este problema, lo primero que debemos hacer es **iniciar npm** en la carpeta local en donde vamos a trabajar. El inicializar **npm** nos ayuda a crear el archivo **packege.json** el cual es la raíz de nuestro programa y nos ayuda con la manipulación de módulos que vamos a usar como con la lista de dependencias que necesitara el programa

```
Ricardo@DESKTOP-0LQ688U MINGW64 ~/Desktop/clases web (master)
$ npm init
```

este comando nos pedirá información adicional que podemos llenar a decisión de cada uno

```
package name: (clases-web)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author: Ricardo Romo
license: (ISC)
```

Al ejecutar el comando, nos pedirá cierta información adicional dependiendo del programa que deseemos realizar, generando así nuestro archivo **package.json** con la lista de dependencias , módulos que queramos usar y la información introducida al inicializar npm .

```
{
  "name": "codigo",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Ricardo Romo",
  "license": "ISC",
  "description": "",
  "dependencies": {
```

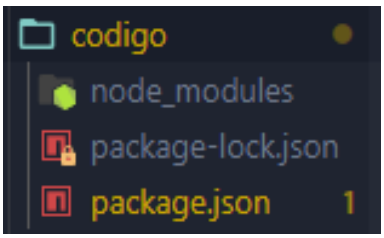
Ahora necesitaremos el paquete **yargs** el cual te ayuda a crear herramientas interactivas de línea de comandos, analizando argumentos y generando una elegante interfaz de usuario. y para instalar como dependencia en nuestro package.json solo necesitaremos ingresar la siguiente línea de comandos en la terminal ubicada en nuestra carpeta:

```
Ricardo@DESKTOP-0LQ688U MINGW64 ~/Desktop/clases web/clases-web/informes_latex/informe_03/codigo (master)
$ npm install yargs
```

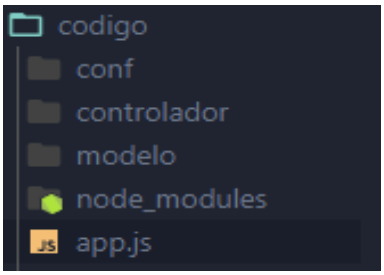
Para confirmar la instalación podemos dirigirnos a nuestro **packege.json** y podemos observar que **yargs** ya se encuentra como una dependencia necesaria para nuestro programa

```
1    },
2    "author": "Ricardo Romo",
3    "license": "ISC",
4    "description": "",
5    "dependencies": {
6      "yargs": "^15.3.1"
7    }
```

Además de esto, podemos visualizar una nueva carpeta creada donde tendremos el módulo yargs.



Ahora crearemos 3 carpetas:
conf : Almacenara las configuraciones dentro del yargs. **controlador :** Realizara las actividades de crear, actualizar, borrar y listar las tareas. **modelo :** En el cual se almacenara el .json con las tareas creadas. **app.js :** que va ser el índex de nuestra aplicación.



Dentro de la carpeta **conf** crearemos un el documento yargs.js el cual tendrá las siguientes configuraciones.

```
const description = {
    demand: true,
    alias: 'd',
    desc: 'description to task'
}

const complete = {
    default: true,
    alias: 'c',
    desc: 'make complete or pending'
}
```

1. declararemos dos objetos, uno llamado **description** y **complete** que almacenaran los siguientes atributos.
   (a) **demand :** Especifica que al recibir datos, este debe ser ingresado obligatoriamente.
   (b) **alias :** Es una forma rápida de referirnos a este atributo para no llamarlo de forma completa.
   (c) **desc:** Descripción de lo que realiza este objeto
   (d) **default:** El valor que toma automáticamente si no es ingresado ningún dato.

Importar paquete yargs:
Para importar nuestro paquete **yargs** previamente instalado declararemos una variable constante llamada argv, la cual almacenara las líneas de comando insertadas desde la terminal al llamar a nuestro índex.

```
const argv = require('yargs')
    .command('create', 'Create a task', {
        description,
        complete
    })
    .command('update', 'Update a task', {
        description
    })
    .command('clean', 'Delete a task', {
        description
    })
    .argv;
```

1. **require() :** Es una función que nos permite incluir módulos ya instalados, y llamándolos solo con su nombre
2. **command.() :** Define los comandos por los cuales se va a recibir la información, los atributos que recibe son:
3. (a) Nombre por el cual se puede ingresar
   (b) Descripción de las actividades que realiza

(c) Atributos propios de estos, en este caso sus atributos se encuentran en los objetos **description** y **complete** antes ya declarada.

4. **.argv :** Obtenga los argumentos como un simple objeto.

Al instanciar todo esto lo hacemos de manera local para el documento, es decir solo este documento podrá manejar nuestro **argv** por lo tanto tenemos que exportarlo para que otros documentos también la puedan utilizar.

```
1  module.exports = {
2      argv
3  }
```

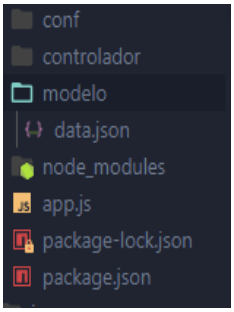
1. **module.exports** especifica que objetos, variables, funciones van hacer exportadas para ser usadas.

Ahora dentro de controlador crearemos un archivo llamado **task-for-do** el cual nos ayudara a crear, actualizar, borrar y listar las tareas.

Primero importamos el módulo **fs** que nos ayudara con el manejo de archivos (module de node js que por default ya está instalado) y creamos un array **taskfordo** el cual almacenara las tareas.

```
1  const fs = require('fs');
2
3  let taskfordo = [];
```

Crearemos la carpeta modelo y dentro de este el archivo data.json en la cual se va almacenar.



Dentro de **taskfordo.js** crearemos los métodos **load_data()** (cargar data.json) y **save_data**(Guardar en data.json)

```
1  const load_data = () => {
2      taskfordo = require('../modelo/data.json')
3  }
4
5  const save_data = () => {
6      //load_data()
7      let data = JSON.stringify(taskfordo)
8      fs.writeFile('modelo/data.json', data, (err) => {
9          if (err) throw new Error('Dont save data ', err);
10     })
11 }
```

1. **load_data()**

   (a) almacena e nuestro vector **taskfordo** el archivo **data.json** ingresando su ubicación.

2. **sabe_data()**

   (a) creamos una variable **data** el cual transforma los datos almacenados en **taskfordo** a un formato **JSON**

   (b) **fs.writefile()** es una función de **fs** que nos permite crear un archivo enviando los parámetros de:

      i. Dirección donde se almacena y se encuentra el archivo

      ii. Los datos que se almacenaran

      iii. Funcion de error en caso de fallar con la operación.

Funcion Crear "create()"

```
1  const create = (description) => {
2      load_data();
3      let task = {
4          description,
5          complete: false
6      }
7      taskfordo.push(task);
8      save_data();
9      return taskfordo;
10 }
```

---

1. Definimos que esta función va a recibir un atributo llamado **description**

2. Cargamos nuestro archivo data en el cual vamos almacenar con la función ya creada **load_data()**

3. Creamos una variable **task** que almacenara **description** y un atributo boleano llamado complete que identifica si la tarea esta completa.

Funcion Listar "getlist()"

```
1  let getlist = () => {
2      load_data();
3      return taskfordo;
4  }
```

---

1. Cargamos nuestras tareas con **load_data()** (esta carga **data.json** en la variable **taskfordo**)

2. Retornemos **taskfordo**

Función actualizar "update()"

```
let update = (des, completado = true) => {
    load_data();
    let index = taskfordo.findIndex(task => task.description
        === des)
        //console.log(index);
    if (index >= 0) {
        taskfordo[index].complete = completado;
        save_data();
        return true;
    }
    return false;
}
```

1. Definimos la función **update** la cual recibe los parámetros de la descripción, y el estado de completado.
2. Cargamos en **taskfordo** nuestra **data.json** con la función **load_data()**
3. Declaramos una variable la cual almacena la posición dentro del vector **taskfordo** si este es igual a la descripción que recibe la función.
4. Si el índice es encontrado remplazaremos el atributo complete por **true**
5. Guardaremos el cambio que hemos realizado con la funcion **save_data()**
6. y retornamos el valor **true** en caso de ser completado con éxito y si no el valor **false**

Función Eliminar "deleted()"

```
let deleted = (descripcion) => {
    load_data();
    let index = taskfordo.findIndex(task => task.description
        === descripcion)
    if (index >= 0) {
        let obj = taskfordo.filter(task => task.description !==
             descripcion)
        taskfordo = obj;
        save_data();
        return true;
    }
    return false;
}
```

1. Esta función recibirá el atributo de la descripción del atributo a eliminar.
2. Cargamos nuestra data con la función **load_data()**
3. Identificamos la posición en la que se encuentra el elemento a eliminar
4. Si se encuentra el índice utilizamos la función **-filter()** el cual almacena todo los datos que no contengan esa información

5. Actualizamos nuestra variable **taskfordo** por el nuevo vector que se encuentra ya sin la tarea deseada.

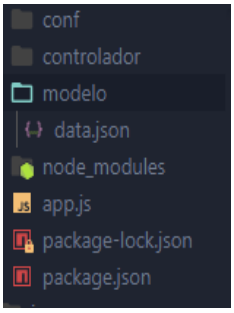
6. Utilizamos la función **save_data()** para guardar los cambios.

7. Retornamos un valor **true** si se completó con éxito, caso contrario un **false**.

Exportación de funciones

```
1  module.exports = {
2      create,
3      getlist,
4      update,
5      deleted
```

Al igual que yargs necesitamos exportar estas funciones para que nuestra **app.js** pueda usarlos para esto utilizaremos **module.exports**
Ahora crearemos en la carpeta raíz nuestro **app.js** el cual va ser nuestro menú para usar cada una de nuestras funciones y utilizando **yargs** para poder ingresar los parámetros



Dentro de **app.js :** Importaremos nuestra configuración de **yargs.js** y nuestras funciones dentro de **taskfordo.js**.
Además instalaremos **colors** con el comando **npm install colors –save** dentro de nuestro **app.js**

```
1  const argv = require('./config/yars').argv;
2  const tasks = require('./controlador/task-for-do')
3  const color = require('colors')
```

Iniciaremos una variable **command** que almacenara la lectura e ingreso de datos desde la terminal

```
1  let command = argv._[0];
```

Para el menú utilizaremos un **switch/case** que nos permitirá ejecutar cualquiera de las opciones dependiendo el caso que se ingrese

```
switch (command) {
    case 'create':
        let task = tasks.create(argv.description);
        console.log(task);
        break;
    case 'list':
        let list = tasks.getlist();
        for (let task of list) {
            console.log(color.green(`Descripcion: ${
                task.description}`));
            console.log(color.green(`Completado: ${
                task.complete}`));
        }
        break;
    case 'update':
        let resp = tasks.update(argv.description, argv.complete
            );
        console.log(resp);
        break;
    case 'clean':
        let del = tasks.deleted(argv.description)
        console.log(del);
        break;
    default:
        console.log('command not found ');
        break;
}
```

1. **Create**
   (a) Creamos una variable **task**, la cual llama a la variable **tasks** que almacena nuestras funciones creadas en **taskfordo.js**
   (b) llamamos a la función **create** y enviamos nuestro **argv.description** que está vinculado con la configuración de nuestro **yargs**
   (c) Mandamos a imprimir en consola las tareas que tenemos
2. **List**
   (a) Creamos una variable llamada **list** que almacena nuestro **taskfordo** que nos lanza la función **getlist()** de nuestro archivo **taskfordo.js** almacenado en **tasks**
   (b) Mandamos a imprimir con **color.** especificando el color que queremos que aparezca en pantalla.
3. **Update**
   (a) Creamos una variable **resp** la cual llamara al método **update** y enviamos el nombre de la tarea a actualizar
4. **Clean**
   (a) Creamos la variable **del** la cual almacena la respuesta del método **deleted** enviando el nombre de la tarea a eliminar

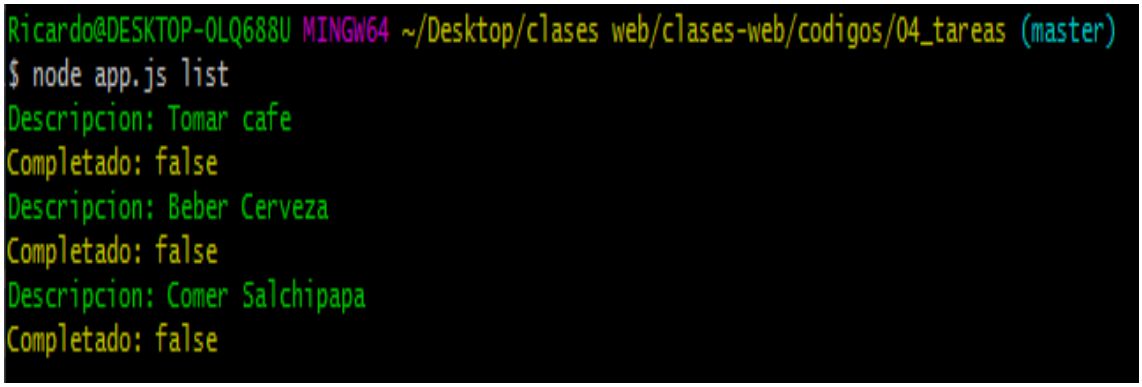
5. **Default:** simplemente imprime en la consola en caso de que cualquier caso ingresado no se encuentre.

EJECUCIÓN

Dirígete a terminal dentro de la carpeta raíz y ejecuta el comando **npm init** para iniciar y **npm install** para instalar las dependencias agregadas en **package.json** :

1. **Crear Tareas**

2. **Listar tareas**



3. **Actualizar tareas**

```
Ricardo@DESKTOP-OLQ688U MINGW64 ~/Desktop/clases web/clases-web/codigos/04_tareas (master)
$ node app update -d "Tomar cafe"
0
true

Ricardo@DESKTOP-OLQ688U MINGW64 ~/Desktop/clases web/clases-web/codigos/04_tareas (master)
$ node app.js list
Descripcion: Tomar cafe
Completado: true
Descripcion: Beber Cerveza
Completado: false
Descripcion: Comer Salchipapa
Completado: false
```

4. **Eliminar tareas**

```
Ricardo@DESKTOP-OLQ688U MINGW64 ~/Desktop/clases web/clases-web/codigos/04_tareas (master)
$ node app.js clean -d "Tomar cafe"
true

Ricardo@DESKTOP-OLQ688U MINGW64 ~/Desktop/clases web/clases-web/codigos/04_tareas (master)
$ node app.js list
Descripcion: Beber Cerveza
Completado: false
Descripcion: Comer Salchipapa
Completado: false
```

Este informe y código lo pudes encontrar aqui: https://github.com/rromom/clases-web.git