EBook Gratis

APRENDIZAJE electron

Free unaffiliated eBook created from
Stack Overflow contributors.

#electron

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: electron

It is an unofficial and free electron ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official electron.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con electron

Observaciones

¿Qué es el electrón?

Electron es un **marco de código abierto** , que se utiliza para crear aplicaciones de escritorio utilizando **HTML** , **CSS** y **JavaScript** . En el interior, funciona gracias a **Chromium** y **Node.js.**

Su creador original, GitHub , trabaja con una amplia comunidad de desarrolladores para mantener el proyecto, que se puede encontrar aquí .

Una de las ventajas principales del uso de Electron es que, dado que se basa en tecnologías web, es **multiplataforma** , lo que permite implementar aplicaciones para Linux, MacOS y Windows, con el mismo código.

También cuenta con elementos nativos, como menús y notificaciones, así como herramientas de desarrollo útiles para la depuración y el informe de errores.

Aplicaciones construidas en Electron

Algunos ejemplos de aplicaciones que utilizan este framework, son:

- Átomo
- Slack para escritorio
- Código de Visual Studio
- GitBook
- Maldición
- Wordpress para escritorio

... y muchos otros .

Versiones

Versión	Observaciones	Fecha de lanzamiento
1.0.0		2016-05-09
1.0.1		2016-05-11
1.0.2		2016-05-13
1.1.0		2016-05-13

Versión	Observaciones	Fecha de lanzamiento
1.1.1		2016-05-20
1.1.2		2016-05-24
1.1.3		2016-05-25
1.2.0		2016-05-26
1.2.1		2016-06-01
1.2.2		2016-06-08
1.2.3	Hay más entre esto y 1.4.7, pero había demasiados para enumerar	2016-06-16
1.4.7	Última versión a partir del 19 de noviembre de 2016	2016-11-19
1.6.11		2017-05-25
1.7.3	Última versión a partir del 19 de junio de 2017	2017-06-19

Examples

Instalación de electron

Dependencias

Para instalar electron primero debes instalar Node.js , que viene con npm .

¿Cómo instalarlo?

Utilice npm :

```
# Install the `electron` command globally in your $PATH
npm install electron -g

# OR

# Install as a development dependency
npm install electron --save-dev
```

Hola Mundo!

Preparar

Una estructura de proyecto de Electron usualmente se ve así:

```
hello-world-app/
├── package.json
├── index.js
└── index.html
```

Ahora vamos a crear los archivos e inicializar nuestro `package.json` .

```
$ mkdir hello-world-app && cd hello-world-app
$ touch index.js
$ touch index.html
$ npm init
```

Nota: Si el parámetro `main` no se especifica en `package.json` , Electron usará `index.js` como el punto de entrada predeterminado.

El proceso principal

En Electron, el proceso que ejecuta el script principal de `package.json` se denomina **proceso principal** . Aquí podemos mostrar una GUI creando instancias de `BrowserWindow` .

Agregue lo siguiente a `index.js` :

```
const { app, BrowserWindow } = require('electron')

// Global reference to the window object
let win

// This method will be called when Electron has finished
// initialization and is ready to create browser windows
app.on('ready', function(){
    // Create the window
    win = new BrowserWindow({width: 800, height: 600})

    // Open and load index.html to the window
    win.loadURL('file://' + __dirname + '/index.html')

    // Emitted when the window is closed.
    win.on('closed', () => {
        // Dereference the window object
        win = null
    });
})

// Quit the app if all windows are closed
app.on('window-all-closed', () => {
    app.quit()
})
```

Plantilla HTML y proceso de renderizado

A continuación creamos la GUI para la aplicación. Electron utiliza páginas web como su GUI, cada una de las cuales se ejecuta en su propio proceso denominado **proceso de representación** .

Agregue el siguiente código a `index.html` :

```
<!DOCTYPE html>
<html>
<head>
    <title>Hello World</title>
</head>
<body>
    <h1>Hello World!</h1>
</body>
</html>
```

Ejecutando la aplicación

Hay varias formas de ejecutar una aplicación electrónica.

Con `electron-prebuilt` instalado `electron-prebuilt` instalado globalmente

Primero, asegúrate de que tienes instalado `electron-prebuilt` .

Ahora podemos probar la aplicación usando este comando:

```
$ electron .
```

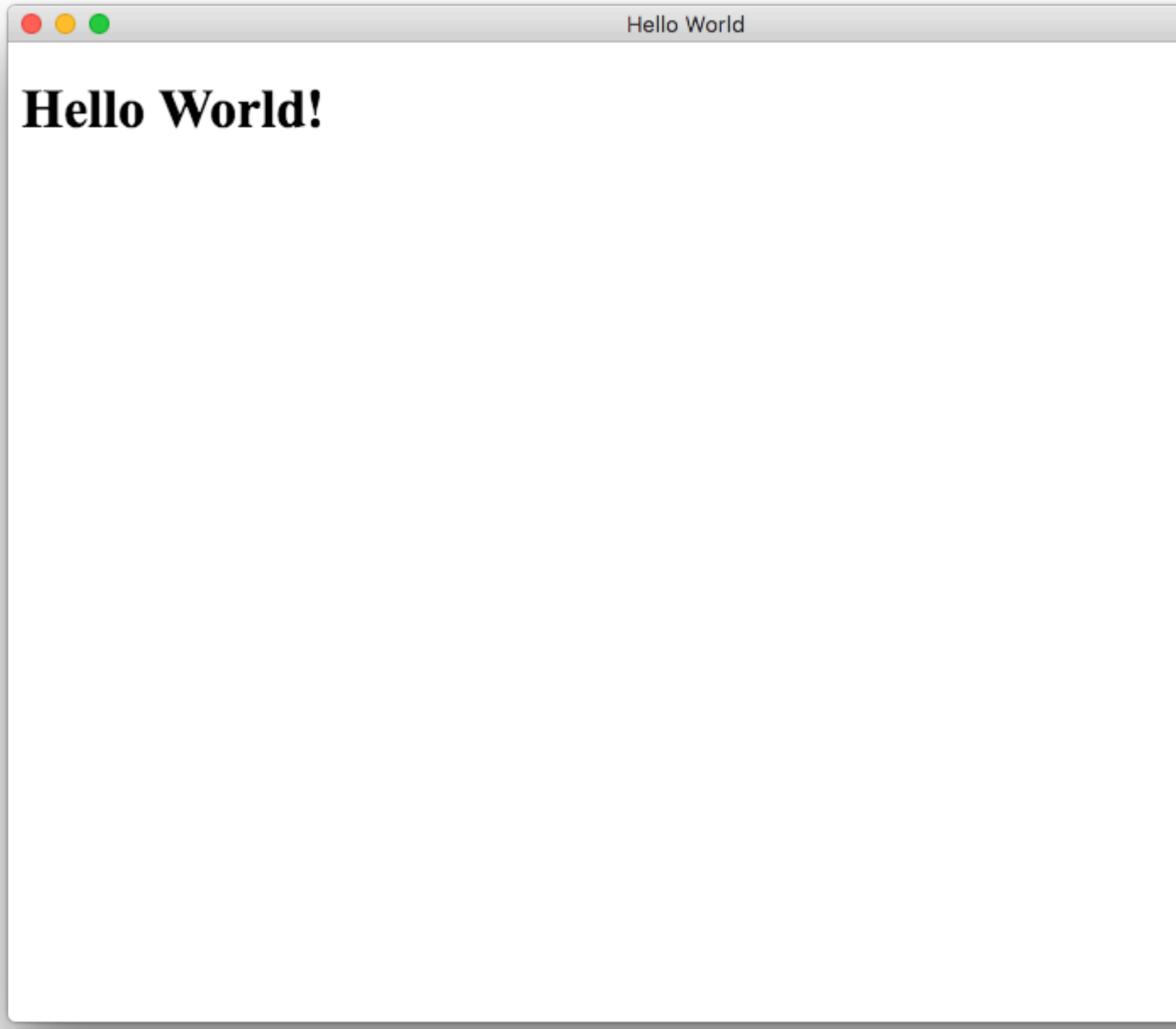
Método 2 - Sin `electron-prebuilt` instalado globalmente

Primero, tendremos que ingresar la carpeta de su aplicación (la carpeta donde está package.json).

Allí, abra una ventana de Terminal / Símbolo del sistema y escriba `npm install` para instalar lo necesario en la carpeta de esa aplicación.

Después, `npm start` a ejecutar la aplicación. Tenga en cuenta que su `package.json` todavía tiene que especificar un script de "inicio".

Si todo funcionó correctamente, deberías ver algo como esto:



¡Felicidades! Has creado con éxito tu primera aplicación electrónica.

Lea Empezando con electron en línea: https://riptutorial.com/es/electron/topic/4934/empezando-con-electron

Capítulo 2: Aplicación de bandeja de electrones

Examples

Aplicación de bandeja electrónica

Añadiendo un icono a tu barra de bandeja

```
let tray = null;
let mainWindow = null;
let user = null;

app.on('ready', () => {
    /**
     * Tray related code.
     */
    const iconName = 'icon.png';
    const iconPath = path.join(__dirname, iconName);
    tray = new Tray(iconPath);
    tray.setToolTip('AMP Notifier App');
    const contextMenu = Menu.buildFromTemplate([{
        label: 'Quit',
        click: destroyApp
    }]);
    tray.setContextMenu(contextMenu);

    tray.on('click', () => {
        app.quit();
    });
});
```

Lea Aplicación de bandeja de electrones en línea:
https://riptutorial.com/es/electron/topic/8160/aplicacion-de-bandeja-de-electrones

Capítulo 3: Empaquetando una aplicación electrónica

Introducción

Cuando esté listo para su distribución, su aplicación electrónica se puede empaquetar en un archivo ejecutable.

Las aplicaciones electrónicas se pueden empaquetar para ejecutarse en Windows (32/64 bit), OSX (macOS) y Linux (x86 / x86_64).

Para empaquetar su código, use el paquete npm 'electron-packager \

https://github.com/electron-userland/electron-packager

Sintaxis

- $ paquete de electrones
- Sourcedir
- nombre de la aplicación
- --plataforma = plataforma
- --arch = arco
- [banderas opcionales ...]

Parámetros

Parámetro	Detalles
Sourcedir	El directorio de tus archivos de aplicación electrónica.
nombre de la aplicación	El nombre de tu aplicación.
plataforma	La plataforma para la que quieres compilar tu código. Omitiendo esto se compilará para el sistema operativo host.
arco	La arquitectura del sistema para la que desea compilar su código. Omitiendo esto se compilará para el arco host.

Examples

Instalar el empaquetador de electrones

```
# for use in npm scripts
npm install electron-packager --save-dev

# for use from cli
npm install electron-packager -g
```

Embalaje de CLI

```
electron-packager C:/my-app MyApp
```

Embalaje desde el guión

```
var packager = require('electron-packager');

packager({
    dir: '/',
}, function(err, path){
    if(err) throw err;
    // Application has been packaged
});
```

Realización de scripts npm para automatizar el empaquetado electrónico.

Una forma conveniente de empaquetar su aplicación es escribir los scripts en su archivo `packages.json` y ejecutarlos con el comando `npm run`

```
{
    "name": "AppName",
    "productName": "AppName",
    "version": "0.1.1",
    "main": "main.js",
    "devDependencies": {
        "electron": "^1.6.6",
        "electron-packager": "^8.7.0"
    },
    "scripts": {
        "package-mac": "electron-packager . --overwrite --platform=darwin --arch=x64 --
icon=images/icon.png --prune=true --out=release-builds",
        "package-win": "electron-packager . --overwrite --platform=win32 --arch=ia32 --
icon=images/icon.png --prune=true --out=release-builds",
        "package-linux" : "electron-packager . --overwrite --platform=linux --arch=x64 --
icon=images/icon.png --prune=true --out=release-builds"
    }
}
```

Y para ejecutarlos solo escribes:

```
npm run package-mac
npm run package-win
npm run package-linux
```

Un desglose de las banderas de comando es:

```
electron-packager .     // this runs the packager in the current folder
--overwrite             // overwrite any previous build
--platform=darwin       // platform for which the binaries should be created
--arch=x64              // the OS architecture
--icon=images/icon.png  // the icon for the app executable
--prune=true            // this does not copy your dev-dependencies that appear in your
packages.json
--out=release-builds    // the name of the folder were the binaries will be outputed
```

Antes, la ejecución de los scripts cambia las Dependencias de Dependencias a dependencias ya que el empaquetador de electrones no puede agrupar los paquetes en las Dependencias de Dependencias en la aplicación. En packager.json, cambie la palabra (si está ahí o si los paquetes se instalan usando --save-dev en npm install) devDependencies a solo dependencias.

Lea Empaquetando una aplicación electrónica en línea:
https://riptutorial.com/es/electron/topic/8945/empaquetando-una-aplicacion-electronica

Capítulo 4: Función remota - usa funciones electrónicas en JavaScript

Introducción

Si tiene que cambiar algunas cosas en renderer.js o main.js pero desea hacer los cambios en index.html, puede usar la función remota. ¡Te permite acceder a todas las funciones electrónicas que necesitas!

Sintaxis

- uso remoto como `require("electron")` :
    - main.js: `const electron = require("electron");`

      index.html: `const electron = require("electron").remote;`

Examples

Usando el control remoto configurando la barra de progreso

```
const { remote } = require("electron"); // <- The Node.js require() function is
                                        // added to JavaScript by electron

function setProgress(p) { // p = number from 0 to 1
  const currentWindow = remote.getCurrentWindow();
  currentWindow.setProgressBar(p);
}
```

Usando el control remoto configurando la ventana a pantalla completa

```
const { remote } = require("electron"); // <- The Node.js require() function is
                                        // added to JavaScript by electron

function fullscreen(f) { // p = false or true
  const currentWindow = remote.getCurrentWindow();
  currentWindow.maximize();
}
```

Lea Función remota - usa funciones electrónicas en JavaScript en línea: https://riptutorial.com/es/electron/topic/8719/funcion-remota---usa-funciones-electronicas-en-javascript

Capítulo 5: ganador de electrones

Introducción

Módulo NPM que construye instaladores de Windows para aplicaciones Electron. Ayudará a crear un solo EXE para la aplicación de ventanas electrón.

Sintaxis

- **Instalar globalmente**
- npm install -g electron-winstaller
- **Instalar localmente**
- npm install --save-dev electron-winstaller

Parámetros

Nombre de configuración	Descripción
directorio de aplicaciones	El valor de los autores para los metadatos del paquete nuget. El valor predeterminado es el campo de autor del archivo package.json de su aplicación cuando no se especifica.
propietarios	El valor de los propietarios para los metadatos del paquete nuget. El valor predeterminado es el campo de autores cuando no se especifica.
exe	El nombre del archivo .exe principal de su aplicación. Esto utiliza el campo de nombre en el archivo package.json de su aplicación con una extensión .exe agregada cuando no se especifica.
descripción	El valor de descripción para los metadatos del paquete nuget. El valor predeterminado es el campo de descripción del archivo package.json de su aplicación cuando no se especifica.
versión	El valor de la versión para los metadatos del paquete nuget. El valor predeterminado es el campo de versión del archivo package.json de su aplicación cuando no se especifica.
título	El valor del título para los metadatos del paquete nuget. El valor predeterminado es el campo productName y luego el campo de nombre del archivo package.json de su aplicación cuando no se especifica.
nombre	ID de modelo de aplicación de Windows (appId). El valor predeterminado es el campo de nombre en el archivo package.json de

Nombre de configuración	Descripción
	su aplicación.
CertificateFile	La ruta a un certificado de firma de código Authenticode
certificado contraseña	La contraseña para descifrar el certificado dado en el archivo de certificado
firmarconpamas	Parámetros para pasar a signtool. Invalida a CertificateFile y certificatePassword.
iconUrl	Una URL a un archivo ICO para usar como icono de la aplicación (que se muestra en Panel de control> Programas y características). De forma predeterminada, el icono de Atom.
setupIcon	El archivo ICO que se usará como icono para el Setup.exe generado
setupExe	El nombre a usar para el archivo Setup.exe generado
setupMsi	El nombre a usar para el archivo Setup.msi generado.
noMsi	¿Debería Squirrel.Windows crear un instalador de MSI?
lanzamientos remotos	Una URL para sus actualizaciones existentes. Si se dan, estos serán descargados para crear actualizaciones delta
remoto hablado	Token de autenticación para actualizaciones remotas

Examples

Construir js

Aquí está el archivo de compilación básica para compilar el ejecutable desde la aplicación electron windows.

```
var electronInstaller = require('electron-winstaller');
var resultPromise = electronInstaller.createWindowsInstaller({
    appDirectory: 'Your_electron_application_path',
    authors: 'Author Name',
    description: "Description"
});

resultPromise.then(() => console.log("Build Success!"), (e) => console.log(`No dice:
${e.message}`));
```

Lea ganador de electrones en línea: https://riptutorial.com/es/electron/topic/9492/ganador-de-electrones

Capítulo 6: Proceso principal y renderizador.

Observaciones

El proceso que ejecuta el script principal de `package.json` se denomina **proceso principal** . El proceso principal crea páginas web creando instancias de `BrowserWindow` . Cada página web en Electron se ejecuta en su propio proceso, que se denomina **proceso de representación** . El proceso principal gestiona todas las páginas web y sus procesos de renderización correspondientes. Cada proceso de renderizado está aislado y solo le importa la página web que se ejecuta en él.

Examples

Comunicación asíncrona IPC

Código fuente del proceso principal `index.js` :

```
const {app, BrowserWindow, ipcMain} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.webContents.openDevTools()
  win.on('closed', () => {
    win = null
  })
  win.webContents.on('did-finish-load', () => {
    win.webContents.send('asyncChannelToRenderer', 'hello')
  })
})

ipcMain.on('asyncChannelToMain', (event, arg) => {
  console.log(arg + ' from renderer')
  if (arg === 'hello') {
    event.sender.send('asyncChannelToRenderer', 'world')
  }
})
```

Proceso de renderizado en `index.html` :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World IPC</title>
    <script>
      require('electron').ipcRenderer.on('asyncChannelToRenderer', (event, arg) => {
        console.log(arg + ' from main')
        if (arg === 'hello') {
          event.sender.send('asyncChannelToMain', 'world')
        }
```

```
      })
    </script>
  </head>
  <body>
    <button onclick="require('electron').ipcRenderer.send('asyncChannelToMain',
'hello')">click me</button>
  </body>
</html>
```

Módulo remoto RMI

El módulo `remote` permite RMI simple (invocación de método remoto) de los objetos del proceso principal del proceso del renderizador. Primero crea el proceso principal en `index.js`

```
const {app, BrowserWindow} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.on('closed', () => {
    win = null
  })
})
```

y luego el proceso remoto `index.html`

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      const {BrowserWindow, app} = require('electron').remote
    </script>
  </head>
  <body>
    <button onclick= "let win = new BrowserWindow();
win.loadURL(`file://${__dirname}/index.html`)">new window</button>
    <button onclick= "app.quit()">quit</button>
  </body>
</html>
```

Comunicación IPC síncrona

Crear `index.js` como

```
const {app, BrowserWindow, ipcMain} = require('electron')
let win = null

app.on('ready', () => {
  win = new BrowserWindow()
  win.loadURL(`file://${__dirname}/index.html`)
  win.webContents.openDevTools()
  win.on('closed', () => {
    win = null
  })
```

```
})

ipcMain.on('syncChannelToMain', (event, arg) => {
  console.log(arg + ' from renderer')
  event.returnValue = 'world'
})
```

y el proceso de renderizador `index.html` como

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World IPC</title>
  </head>
  <body>
    <button onclick="console.log(require('electron').ipcRenderer.sendSync('syncChannelToMain',
'world') + ' from main')">click me</button>
  </body>
</html>
```

Lea Proceso principal y renderizador. en línea:
https://riptutorial.com/es/electron/topic/5432/proceso-principal-y-renderizador-

Capítulo 7: Usando bootstrap en electron

Introducción

Uno de los mejores frameworks front-end en el mundo web en twitter bootstrap. Como electron se basa en el navegador web, podemos usar fácilmente bootstrap con electron para usar el poder de bootstrap en nuestro marco de electron. La última versión de bootstrap a partir de hoy es 3.3.7 y bootstrap 4 aún está en fase alfa.

Examples

Enlace de Electron con Bootstrap

Para utilizar bootstrap, hay 2 casos.

1. La aplicación electrónica está conectada a internet.
2. La aplicación electrónica no está conectada a internet.

Para las aplicaciones electrónicas que están conectadas a internet, solo podemos hacer uso de los enlaces CDN para bootstrap e incluirlos en nuestros archivos html.

El problema surge cuando tenemos que llevarlo a la versión sin conexión donde la aplicación no está conectada a la red. En ese caso,

1. Descargar bootstrap desde Bootstrap
2. Descomprime la carpeta en la aplicación electrónica
3. En el directorio bootstrap, hay archivos css y javascript.
4. Para una mejor comprensión, mueva los archivos bootstrap css a la carpeta CSS (todos los archivos de estilo estarán en esta carpeta) y los archivos bootstrap js a la carpeta JS (todos los archivos Javascript estarán en esta carpeta)
5. En sus archivos html, vincule los archivos html usando el siguiente código

```
<link rel="stylesheet" href="path_to_the_offline_bootstrap_css_file">
<script scr="path_to_the_offline_bootstrap_js_file"></script>
```

De esta manera, puedes comenzar a usar el bootstrap de Twitter en el marco electrónico.

Lea Usando bootstrap en electron en línea: https://riptutorial.com/es/electron/topic/10897/usando-bootstrap-en-electron

Capítulo 8: Usando nedb en electron

Examples

Instalacion de nedb

Es muy fácil instalar nedb.

```
npm install nedb --save    # Put latest version in your package.json
```

Para los amantes de las glorietas,

```
bower install nedb
```

Conexión de la aplicación electrónica con Nedb

Al crear aplicaciones electrónicas, por lo general, el backend está en una carpeta separada (archivos js) y el front-end está en una carpeta separada (archivos html). En el backend, para poder usar la base de datos, debemos incluir el paquete nedb con la declaración requerida de la siguiente manera.

```
var Datastore = require('nedb'),db = new Datastore({ filename: 'data.db', autoload: true });
```

Tenga en cuenta que la carga del archivo de base de datos es una tarea asíncrona.

Insertar datos en nedb

Básicamente, para insertar registros en nedb, los datos se almacenan en forma de json, siendo la clave los nombres de las columnas y el valor de esos nombres serán los valores de ese registro.

```
var rec = { name: 'bigbounty',age:16};

db.insert(rec, function (err, newrec) {   // Callback is optional
  // newrec is the newly inserted document, including its _id
  // newrec has no key called notToBeSaved since its value was undefined
});
```

Tenga cuidado con todas las operaciones de la base de datos, ya que son asíncronas.

Nota **: Si _id no está allí en los datos json que está insertando, entonces automáticamente será creado por nedb.

Buscar en nedb

Para buscar registros en nedb, nuevamente necesitamos simplemente pasar el json que contiene los criterios de búsqueda como un parámetro a la función de búsqueda del objeto db.

```
db.find({ name: 'bigbounty' }, function (err, docs) {
  // docs is an array containing documents that have name as bigbounty
  // If no document is found, docs is equal to []
});
```

Para encontrar solo un documento, como usamos limit en mysql, es fácil en nedb.

```
db.findOne({ name: 'bigbounty' }, function (err, doc) {
  // doc is only one document that has name as bigbounty
  // If no document is found, docs is equal to []
});
```

Eliminar en nedb

Para eliminar documentos en nedb, es muy fácil. Sólo tenemos que utilizar la función de eliminación de objeto db.

db.remove ({name: 'bigbounty'}, function (err, numremoved) {// numremoved es el número de documentos que se eliminan.});

Lea Usando nedb en electron en línea: https://riptutorial.com/es/electron/topic/10906/usando-nedb-en-electron

Creditos

S. No	Capítulos	Contributors
1	Empezando con electron	Alphonsus, Community, Eslam Mahmoud, Florian Hämmerle, Hewbot, J F, Piero Divasto, SimplyCodin, Theo, vintproykt, Vishal
2	Aplicación de bandeja de electrones	Anavar Bharmal, nmnsud
3	Empaquetando una aplicación electrónica	bigbounty, Dan Johnson, VladNeacsu
4	Función remota - usa funciones electrónicas en JavaScript	B. Colin Tim, Florian Hämmerle
5	ganador de electrones	Krupesh Kotecha
6	Proceso principal y renderizador.	mrkovec
7	Usando bootstrap en electron	bigbounty
8	Usando nedb en electron	bigbounty