EBook Gratuito

APPRENDIMENTO

electron

Free unaffiliated eBook created from
Stack Overflow contributors.

#electron

Sommario

Di

Capitolo 1: Iniziare con l'elettrone

Osservazioni

Cos'è l'elettrone?

Electron è un **framework open-source** , che viene utilizzato per creare applicazioni desktop utilizzando **HTML** , **CSS** e **JavaScript** . All'interno, funziona grazie a **Chromium** e **Node.js.**

Il suo creatore originale, GitHub , collabora con una vasta comunità di sviluppatori per mantenere il progetto, che può essere trovato qui .

Uno dei vantaggi principali dell'utilizzo di Electron è che, poiché si basa su tecnologie web, è **multipiattaforma** , consentendo di distribuire applicazioni per Linux, MacOS e Windows, con lo stesso codice.

Dispone inoltre di elementi nativi come menu e notifiche, oltre a utili strumenti di sviluppo per il debug e la segnalazione degli arresti anomali.

App create su Electron

Alcuni esempi di applicazioni che utilizzano questo framework sono:

- Atomo
- Slack per desktop
- Codice di Visual Studio
- GitBook
- Maledizione
- Wordpress per desktop

... e molti altri .

Versioni

Versione	Osservazioni	Data di rilascio
1.0.0		2016/05/09
1.0.1		2016/05/11
1.0.2		2016/05/13
1.1.0		2016/05/13

Versione	Osservazioni	Data di rilascio
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	Ci sono più tra questo e 1.4.7, ma ce n'erano troppi per elencare	2016/06/16
1.4.7	Ultima versione dal 19 novembre 2016	2016/11/19
1.6.11		2017/05/25
1.7.3	Ultima versione dal 19 giugno 2017	2017/06/19

Examples

Installazione di elettroni

dipendenze

Per installare l'elettrone devi prima installare Node.js , che viene fornito con npm .

Come installarlo?

Usa npm :

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

# OR

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

Ciao mondo!

Impostare

Una struttura di progetto di Electron di solito assomiglia a questo:

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

Ora creiamo i file e inizializziamo il nostro `package.json` .

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

Nota: se il parametro `main` non è specificato in `package.json` , Electron utilizzerà `index.js` come punto di ingresso predefinito.

Il processo principale

In Electron, il processo che esegue lo script principale di `package.json` è chiamato il **processo principale** . Qui possiamo visualizzare una GUI creando istanze `BrowserWindow` .

Aggiungi quanto segue 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()
})
```

Modello HTML e processo di rendering

Successivamente creiamo la GUI per l'app. Electron utilizza le pagine Web come GUI, ciascuna gestita nel proprio processo chiamato **processo di rendering** .

Aggiungi il seguente codice a `index.html` :

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

Esecuzione dell'applicazione

Esistono diversi modi per eseguire un'app Electron.

Con `electron-prebuilt` installato globalmente

Innanzitutto, assicurati di aver installato un `electron-prebuilt` preinstallato .

Ora possiamo testare l'app usando questo comando:

```
$ electron .
```

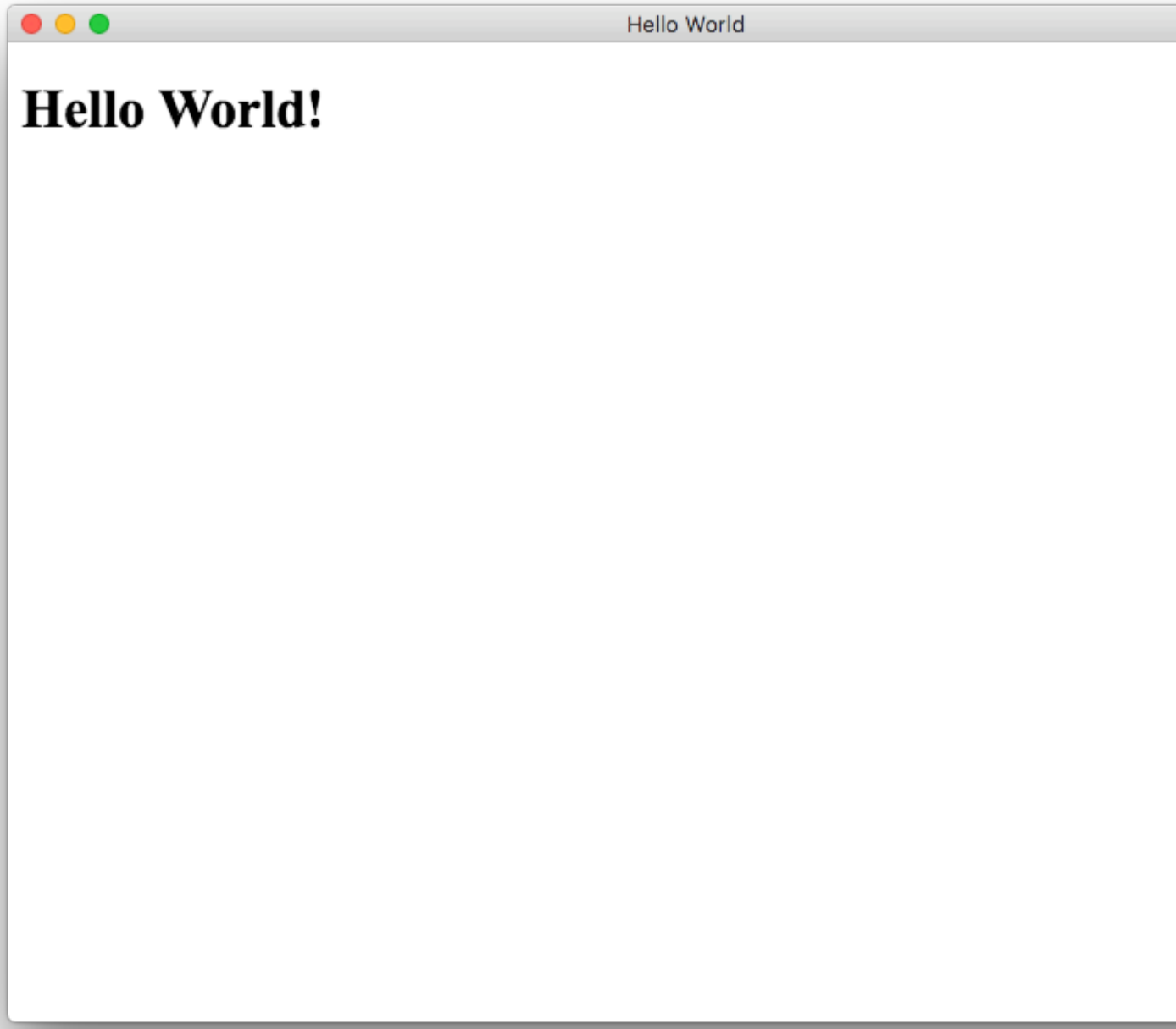
Metodo 2: senza `electron-prebuilt` installato globalmente

Innanzitutto, dovremo inserire la cartella dell'app (la cartella in cui package.json è).

Lì, apri una finestra del prompt dei comandi / terminale e digita `npm install` per installare il necessario nella cartella dell'app.

Successivamente, digitare `npm start` per avviare l'app. Ricorda che il tuo `package.json` deve ancora specificare uno script 'start'.

Se tutto ha funzionato correttamente, dovresti vedere qualcosa di simile a questo:



Congratulazioni! Hai creato con successo la tua prima app Electron.

Leggi Iniziare con l'elettrone online: https://riptutorial.com/it/electron/topic/4934/iniziare-con-l-elettrone

Capitolo 2: Electron-tray-app

Examples

App Electron Tray

Aggiunta di un'icona alla barra del vassoio

```
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();
    });
});
```

Leggi Electron-tray-app online: https://riptutorial.com/it/electron/topic/8160/electron-tray-app

Capitolo 3: elettrone-winstaller

introduzione

Modulo NPM che crea programmi di installazione di Windows per le app Electron. Aiuterà a creare un singolo EXE per l'applicazione di finestre Electron

Sintassi

- **Installa globalmente**
- npm install -g electron-winstaller
- **Installa localmente**
- installazione npm --save-dev electron-winstaller

Parametri

Nome configurazione	Descrizione
AppDirectory	Il valore dell'autore per i metadati del pacchetto nuget. Per impostazione predefinita, il campo dell'autore dal file package.json della tua app non è specificato.
proprietari	Il valore proprietario per i metadati del pacchetto nuget. Il valore predefinito è il campo degli autori quando non specificato.
EXE	Il nome del file .exe principale della tua app. Utilizza il campo del nome nel file package.json dell'app con un'estensione .exe aggiunta quando non specificato.
descrizione	Il valore di descrizione per i metadati del pacchetto nuget. Fa il default al campo della descrizione dal file package.json della tua app quando non specificato.
versione	Il valore della versione per i metadati del pacchetto nuget. Il valore predefinito è il campo della versione dal file package.json della tua app quando non specificato.
titolo	Il valore del titolo per i metadati del pacchetto nuget. Il valore predefinito è il campo productName e quindi il campo del nome dal file package.json dell'app quando non specificato.
nome	ID modello applicazione Windows (appId). Predefinito al campo del nome nel file package.json dell'app.

Nome configurazione	Descrizione
certificateFile	Il percorso di un certificato di firma del codice Authenticode
CertificatePassword	La password per decrittografare il certificato fornito in certificateFile
signWithParams	Params per passare a signtool. Esegue l'override del certificatoFile e del certificatoPassword.
iconUrl	Un URL per un file ICO da utilizzare come icona dell'applicazione (visualizzato in Pannello di controllo> Programmi e funzionalità). Predefinito all'icona Atom.
setupIcon	Il file ICO da utilizzare come icona per Setup.exe generato
SetupExe	Il nome da utilizzare per il file Setup.exe generato
setupMsi	Il nome da utilizzare per il file Setup.msi generato
nomsi	Squirrel.Windows dovrebbe creare un programma di installazione MSI?
remoteReleases	Un URL per i tuoi aggiornamenti esistenti. Se forniti, questi verranno scaricati per creare aggiornamenti delta
remoteToken	Token di autenticazione per aggiornamenti remoti

Examples

Costruisci JS

Qui è il file di build di base per costruire eseguibile dall'app di windows electron.

```
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}`));
```

Leggi elettrone-winstaller online: https://riptutorial.com/it/electron/topic/9492/elettrone-winstaller

Capitolo 4: Funzione remota: usa le funzioni elettroniche in JavaScript

introduzione

Se devi modificare alcune cose in renderer.js o main.js ma vuoi fare le modifiche in index.html, puoi usare la funzione remota. Ti consente di accedere a tutte le funzioni di elettroni di cui hai bisogno!

Sintassi

- usa il telecomando come `require("electron")` :
    - main.js: `const electron = require("electron");`

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

Examples

Utilizzo del telecomando impostando la barra di avanzamento

```
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);
}
```

Utilizzo del telecomando impostando la finestra a schermo intero

```
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();
}
```

Leggi Funzione remota: usa le funzioni elettroniche in JavaScript online: https://riptutorial.com/it/electron/topic/8719/funzione-remota--usa-le-funzioni-elettroniche-in-javascript

Capitolo 5: Imballaggio di un'app di elettroni

introduzione

Una volta pronto per la distribuzione, la tua app di elettroni può essere impacchettata in un file eseguibile.

Le applicazioni Electron possono essere impacchettate per funzionare su Windows (32/64 bit), OSX (macOS) e Linux (x86 / x86_64).

Per impacchettare il tuo codice, usa il pacchetto npm 'electron-packager \

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

Sintassi

- $ electron-packager
- SourceDir
- nome dell'applicazione
- --platform = piattaforma
- --arch = arco
- [bandiere opzionali ...]

Parametri

Parametro	Dettagli
SourceDir	La directory dei tuoi file di applicazione di elettroni
nome dell'applicazione	Il nome della tua domanda
piattaforma	La piattaforma per cui vuoi compilare il tuo codice. Omettere questo verrà compilato per il sistema operativo host
arco	L'architettura di sistema per cui vuoi compilare il tuo codice. Omettere questo verrà compilato per l'arco host

Examples

Installazione di electron-packager

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

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

Confezionamento da CLI

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

Imballaggio dalla sceneggiatura

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

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

Realizzare script npm per automatizzare l'imballaggio di Electron

Un modo conveniente per pacchettizzare l'applicazione è scrivere gli script nel file `packages.json` ed eseguirli con il 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"
    }
}
```

E per eseguirli basta scrivere:

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

Una ripartizione dei flag di comando è:

```
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
```

Prima, l'esecuzione degli script modifica devDependencies in dipendenze poiché electron-packager non è in grado di raggruppare i pacchetti nelle devDependencies nell'app. In packager.json, cambia la parola (se è lì o se i pacchetti sono installati usando --save-dev in npm install) devDependencies alle sole dipendenze.

Leggi Imballaggio di un'app di elettroni online:
https://riptutorial.com/it/electron/topic/8945/imballaggio-di-un-app-di-elettroni

Capitolo 6: Processo principale e di rendering.

Osservazioni

Il processo che esegue lo script principale di `package.json` è chiamato il **processo principale** . Il processo principale crea pagine Web creando istanze `BrowserWindow` . Ogni pagina Web di Electron viene eseguita nel proprio processo, che viene chiamato il **processo di rendering** . Il processo principale gestisce tutte le pagine Web e i relativi processi di rendering. Ogni processo di rendering è isolato e si preoccupa solo della pagina web in esecuzione al suo interno.

Examples

Comunicazione IPC asincrona

Codice di origine del processo principale `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')
  }
})
```

Processo di rendering in `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>
```

Modulo remoto RMI

Il modulo `remote` consente il semplice RMI (richiamo del metodo remoto) degli oggetti processo principali dal processo di rendering. Prima crea il processo principale in `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
  })
})
```

e quindi il processo 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>
```

Comunicazione IPC sincrona

Crea `index.js` come

```
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'
})
```

e il processo di rendering `index.html` come

```
<!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>
```

Leggi Processo principale e di rendering. online:
https://riptutorial.com/it/electron/topic/5432/processo-principale-e-di-rendering-

Capitolo 7: Usando il bootstrap in elettrone

introduzione

Uno dei migliori framework front-end nel mondo web in twitter bootstrap. Poiché l'elettrone si basa sul browser web, possiamo facilmente utilizzare il bootstrap con l'elettrone per utilizzare la potenza del bootstrap nel nostro framework di elettroni. L'ultima versione di bootstrap ad oggi è la 3.3.7 e bootstrap 4 è ancora in fase alpha.

Examples

Collegamento di elettroni con Bootstrap

Per utilizzare il bootstrap, ci sono 2 casi.

1. L'app electron è connessa ad internet
2. L'app electron non è connessa ad internet

Per le app di elettroni connesse a Internet, possiamo semplicemente utilizzare i collegamenti CDN per il bootstrap e includerlo nei nostri file html.

Il problema arriva quando dobbiamo portarlo alla versione offline dove l'app non è connessa alla rete. In quel caso,

1. Scarica Bootstrap da Bootstrap
2. Decomprimere la cartella nell'app electron
3. Nella directory bootstrap ci sono file css e javascript.
4. Per una migliore comprensione, sposta i file cst bootstrap nella cartella CSS (Tutti i file di stile saranno in questa cartella) e avvia i file js nella cartella JS (Tutti i file Javascript saranno in questa cartella)
5. Nei tuoi file html, collega i file html usando il seguente codice

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

In questo modo puoi iniziare a utilizzare il bootstrap di Twitter nel framework degli elettroni.

Leggi Usando il bootstrap in elettrone online: https://riptutorial.com/it/electron/topic/10897/usando-il-bootstrap-in-elettrone

Capitolo 8: Usando nedb in elettrone

Examples

Installazione di nedb

È molto facile installare nedb.

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

Per le persone che amano i bower,

```
bower install nedb
```

Connessione dell'elettronica con Nedb

Durante la creazione di app elettroniche, in genere il back-end si trova in una cartella separata (file js) e il front-end si trova in una cartella separata (file html). Nel back-end, per utilizzare il database, dobbiamo includere il pacchetto nedb con la dichiarazione require come segue.

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

Tenere presente che il caricamento del file di database è un'attività asincrona.

Inserisci dati in nedb

Fondamentalmente, per inserire record su nedb, i dati sono archiviati sotto forma di json con la chiave come nomi di colonne e il valore per quei nomi sarà il valore per quel record.

```
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
});
```

Fai attenzione a tutte le operazioni del database, poiché sono asincrone.

Nota **: Se _id non è presente nei dati di JSON che stai inserendo automaticamente, sarà creato per te da nedb.

Cerca in nedb

Per cercare i record in nedb, di nuovo dobbiamo solo passare il json contenente i criteri di ricerca come parametro per la funzione di ricerca dell'oggetto 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 []
});
```

Per trovare solo un documento, dato che usiamo il limite in mysql, è facile in 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 []
});
```

Elimina in nedb

Per rimuovere documenti in nedb, è molto semplice. Dobbiamo solo usare la funzione di rimozione dell'oggetto db.

db.remove ({name: 'bigbounty'}, function (err, numremoved) {// numremoved è il numero di documenti rimossi.});

Leggi Usando nedb in elettrone online: https://riptutorial.com/it/electron/topic/10906/usando-nedb-in-elettrone

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con l'elettrone	Alphonsus, Community, Eslam Mahmoud, Florian Hämmerle, Hewbot, J F, Piero Divasto, SimplyCodin, Theo, vintproykt, Vishal
2	Electron-tray-app	Anavar Bharmal, nmnsud
3	elettrone-winstaller	Krupesh Kotecha
4	Funzione remota: usa le funzioni elettroniche in JavaScript	B. Colin Tim, Florian Hämmerle
5	Imballaggio di un'app di elettroni	bigbounty, Dan Johnson, VladNeacsu
6	Processo principale e di rendering.	mrkovec
7	Usando il bootstrap in elettrone	bigbounty
8	Usando nedb in elettrone	bigbounty