**Corso ElectronJs**



# **Indice:**

1. [Lezione 1 - Introduzione](#_Lezione_1_-)
2. [Lezione 2 – Struttura di un’applicazione base](#_Lezione_2_–)
3. [Lezione 3 – Primo Setup](#_Lezione_3_–)
4. [Lezione 4 – IPC Comunication](#_Lezione_4_–)
5. [Lezione 5 – IPC, un’applicazione pratica](#_Lezione_5_–)
6. [Lezione 6 – Custom window](#_Lezione_6_–)
7. [Lezione 7 – Custom topbar](#_Lezione_7_–)
8. [Lezione 8 – Barra del menu](#_Lezione_8_–)
9. [Lezione 9 – Gestire più finestre](#_Lezione_9_–)
10. [Lezione 10 – Icona Tray](#_Lezione_10_–)
11. [Lezione 11 – Context Menu](#_Lezione_11_–)
12. [Lezione 12 – Notifiche](#_Lezione_12_–)
13. [Lezione 13 – Scorciatoie da tastiera](#_Lezione_13_–)
14. [Lezione 14 – Creare file per installazione con Electron Forge](#_Lezione_14_–)
15. [Lezione 15 – Aggiornare e firmare l’app](#_Lezione_15_–)
16. [Appendice – Costruire un’app con Angular ed ElectronJs](#_Appendice_–_Costruire)

## **Lezione 1 - Introduzione**

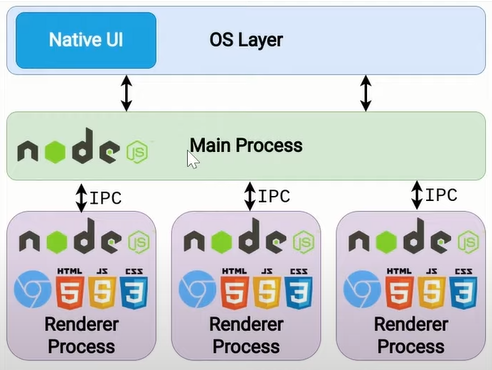
È un framework JavaScript che permette di scrivere applicazioni desktop cross-platform.

Permette di accedere alle funzionalità del SO (es. Rinominazione di file), usando tecnologie web.

**Applicazioni** che **usano** ElectronJS:

Notion, Figma, Skype, Trello, Microsoft Team, VSCode, ecc...

**Struttura:**



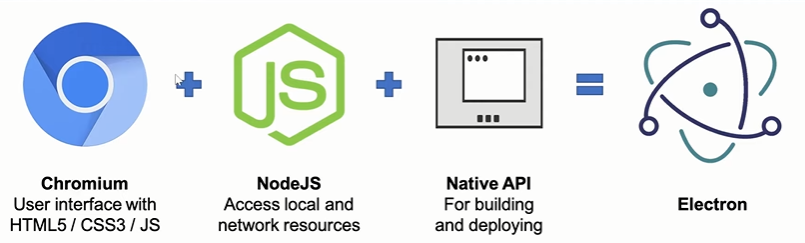
**Chromium**: Finestra che ci permette di renderizzare HTML, CSS e JS

**Node.Js**: tramite con il quale è possibile interagire cone le API Native

**API Native**: accesso a funzinalità native SO**\***

**IPC**: InterProcessComunication che permette di comunicare il processo principale (**main process** Node) con I processi di rendering lato chromium, così che I render processi singoli possano comunicare tra loro.

Es. **Un “Browser”, un’applicazione**.



\*

**Api di sistema di electron**

1. Finestre e Schermi:

* **BrowserWindow**: Consente la creazione e il controllo delle finestre dell'applicazione, inclusi la dimensione, la posizione, l'aspetto e il comportamento.
* **Screen**: Fornisce informazioni sulle dimensioni, la disposizione e le caratteristiche dei monitor collegati.

1. File System:

* **fs**: Fornisce accesso al file system locale per la lettura, la scrittura e la gestione dei file e delle directory.
* **dialog**: Permette di aprire finestre di dialogo per selezionare file o cartelle.

1. Comunicazione di rete:

* **net**: Consente di effettuare richieste HTTP/HTTPS personalizzate e di creare server TCP o IPC.
* **ipcMain** e **ipcRenderer**: Consentono la comunicazione asincrona tra il processo principale (main process) e i processi di rendering (renderer process).

1. Sistema operativo e informazioni sulla macchina:

* **app**: Fornisce informazioni e controlli specifici dell'applicazione, ad esempio la gestione degli eventi del ciclo di vita dell'applicazione.
* **shell**: Permette di eseguire operazioni sul file system, ad esempio aprire file o URL con l'applicazione predefinita del sistema.
* **systemPreferences**: Consente l'accesso alle preferenze di sistema, come le impostazioni di aspetto e comportamento dell'utente.

1. Notifiche di sistema:

* **Notification**: Permette di mostrare notifiche di sistema personalizzate.

1. **Menu** e **MenuItem**: Consentono la creazione di menu personalizzati per l'applicazione, inclusi menu di applicazione, menu contestuali e menu di barra.
2. **GlobalShortcut**: Permette la registrazione di scorciatoie da tastiera globali per l'applicazione.
3. **Clipboard**: Fornisce funzionalità per leggere e scrivere dati negli appunti del sistema.
4. **PowerMonitor**: Permette di rilevare eventi relativi all'alimentazione come la sospensione o la ripresa del sistema.
5. **Tray**: Permette la creazione di icone nella barra delle applicazioni o nell'area di notifica.
6. **AutoUpdater**: Fornisce funzionalità per la gestione degli aggiornamenti automatici dell'applicazione.
7. **Webview**: Consente di incorporare pagine web all'interno delle finestre dell'applicazione.
8. **TouchBar**: Permette la creazione di controlli personalizzati nella Touch Bar dei MacBook Pro.
9. **Protocol**: Consente di registrare protocolli personalizzati per gestire l'apertura di URL specifici all'interno dell'applicazione.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 2 – Struttura di un’applicazione base**

Un’applicazione ElectronJs è basilarmente costituita da:

**Package.json:** Setta le varie dipendenze e setta il file node **main.js**

**Main.js**:file node che fa da Main Process e che fa comunicare con tramite gli IPC tutti I Render Process, come l’index.html

**Index.html**: parte puramente html,css e js che renderizza tramite chromium le pagine.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 3 – Primo Setup**

* **Inizializziamo progetto**

npm init

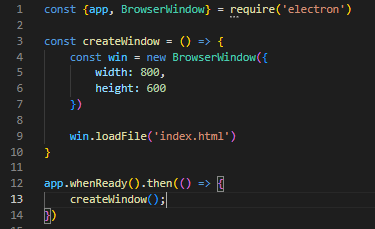
* **Installiamo electronjs**

npm i electron --save-dev

* **Creiamo la prima pagina index.html**
* **Creazione del file main.js**
* **Import del package electron**

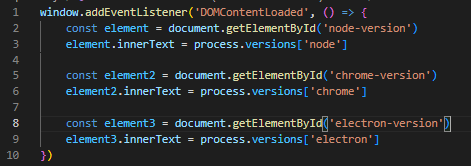
Inizializzazione di una nuova finestra assegnando ad essa una nuova “finestra del browser”

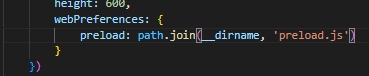
Caricamento come finestra del browser del file index.html



* Carichiamo dei dati prima del caricamento della pagina

Creiamo lo script preload.js, qui inseriremo una funzione in preload che effettuerà delle operazioni, in questo caso inserirà negli elementi con id specifico la versione degli oggetti di riferimento.

Definendo il caricamento dello script nel main.js



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 4 – IPC Comunication**



L'espressione **comunicazione tra processi** (in inglese **inter-process communication** o **IPC**) si riferisce a tutte quelle tecnologie software il cui scopo è consentire a diversi processi di comunicare tra loro scambiandosi dati e informazioni. I processi possono risiedere sullo stesso computer o essere distribuiti su una rete.

Comunicazione tra il codice lato pagina web e il codice lato node, il main process.

**Web Page --> Main**:

inviamo dei dati dalla pagina web al main process

**Web page:**

<script>

const { ipcRenderer } = require('electron');

const button = document.getElementById('button');

button.addEventListener('click', () => {

console.log('log di test');

// Emissione dato ipc verso il Main Process

ipcRenderer.send('datoIPC', {data: 'data'})

})

</script>

1. Importiamo un modulo di comunicazione IPC nello script
2. Definiamo un evento di send (send(nomeEvento, dati), verso il main process usando il metodo
3. Agganciamo tale evento ad un bottone del body della web page

**Main Process:**

webPreferences: {

// Integra node lato web

nodeIntegration: true,

// Non isola il contesto

contextIsolation: false,

// Preload di questo script precaricamento applicazione

preload: path.join(\_\_dirname, 'preload.js')

}

1. Configuriamo la web page per ricevere moduli e non isolare il context

ipcMain.on('datoIPC', (event, data) => {

console.log('Dato IPC Arrivato: ', data);

})

1. Riceviamo a stampiamo i dati arrivati

Main Process --> Web Page:

(il processo è presso chè speculare a quello precedente)

mainWindow.webContents.send('datoMainProcess', {data: 'dataDaMainProcess'})

1. Inviamo dei dati dal Main Process alla web Page

ipcRenderer.on('datoMainProcess', (event, data) => {

console.log('Dato Main Process Arrivato: ', data);

})

1. Riceviamoli nella web page

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 5 – IPC, un’applicazione pratica**



Proviamo a creare un’applicazione che permetta la conversione di immagini.

1. Installiamo la libreria webp-converter

npm i webp-converter

1. Definiamo un input lato web page tramite il quale inviamo una richiesta al main process main.js

button.addEventListener('click', imgConverter)

function imgConverter() {

ipcRenderer.send('imgWebp')

}

1. Nel main.js importiamo la libreria webp-converter

const webp = require('webp-converter');

1. Mettiamo in ascolto l’ipcMain e usiamo la funzione atta al processo di conversione, dando path del file e restituendo una risposta, la quale comunicherà alla web page l’esito del processo di conversione

ipcMain.on('imgWebp', (event, data) => {

console.log('Dato IPC Arrivato: ', data);

const result = webp.cwebp("./steve.jpg","./steve.webp","-q 100",logging="-v");

result.then((response) => {

console.log(response);

mainWindow.webContents.send('datoMainProcess', {data: 'Processo di conversione andato a buon fine'})

});

})

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 6 – Custom window**

È possibile effettuare tutta una serie di personalizzazione della finestra dell’applicazione:

Alcuni esempi:

**Width**: larghezza di finestra all’avvio

**MinWidth**: dimensione minima alla quale può essere contratta la finestra

**MaxWidth**: dimensione massima alla quale può essere espansa la finestra

**Resizable**: possibilità di scalare la finestra

**Minimizable**: possibilità di minimizzare la finestra

**Fullscreen**: l’applicazione parte a tutto schermo

**Fullscreenable**: possibiltà di rendere a tutto schermo la finestra

**Frame**: finestra senza drag e drop e bloccata

È possibile definire un drag & drop usando, lato stile, il webkit -webkit-app-region: drag;

**Transparent**: rendere la finestra trasparente



*Esempio di finestra senza frame ma d&d tramite stile*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 7 – Custom topbar**

È possibile sostituire il frame con la barra delle applicazioni predefinita creandone una custom aggiungendo le funzionalità di **minimizzazione, full screen e chiusura**.



Utilizzando dei bottoni che inviino verso il main process l’input, è possibile attivare tali funzionalità

* **Minimizzazione:**

ipcMain.on('window:minimize', (event, data) => {

mainWindow.minimize();

})

La funzione minimize() poerterà alla riduzione ad icona della finestra.

* **Maximizzazione:**

ipcMain.on('window:square', (event, data) => {

if (mainWindow.isMaximized()) {

mainWindow.restore()

} else {

mainWindow.maximize();

}

})

Il controllo permette di verificare se la finestra è già a tutto schermo, nel qual caso sia così viene usato restore per portare allo stato precedente la finestra, differente questa verrà massimizzata in fullscreen.

* **Chiusura:**

ipcMain.on('window:close', (event, data) => {

mainWindow.close()

})

Chiude l’applicazione.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 8 – Barra del menu**



È possibile costruire una barra del menu da zero, definendo il template, applicandolo alla webpage e **inizializzandola**.

Nel template è possibile, ad esempio, definire:

* **Id**: **identifica** univocamente una voce del menu
* **Label**: aggiunge un’etichettaal menu
* **Role**: definisce un **evento preconfezionato** a quella voce di menu
* **Eventi**: aggiunge una funzione ad una voce di menu dopo un evento (es. **click**)
* **Enable**: abilità/disabilità una voce di un **sottomenu**
* **Visible**: rende visibile/invisibile una voce di un **sottomenu**
* **Accelerator**: aggiunge una shortkey per quella voce di menu

Ecc...

* **Creazione Template:**

// Definizione Menu barra

const template = [

{label: 'prova'},

{label: 'prova1'},

{label: 'prova2'},

{label: 'prova3'}

]

* **Applicazione del Template al Menu:**

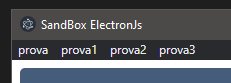
// Costruzione del menu applicando il template

const menu = Menu.buildFromTemplate(template)

* **Inizializzazione del nuovo Menu:**

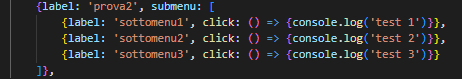
// Inizializzare il menu

Menu.setApplicationMenu(menu)



*Risultato*

*Nel template è possibile definire un sottomenu...*



...*, aggiungere delle funzioni e...*



*... impostarne di preconfezionate.*



* **Aggiungere dinamicamente una voce al Menu**

Tramite l’uso del MenuItem è possibile aggiungere dinamicamente una voce al menu



*Pulsante menu con funzione di aggiunta*

Funzione che **aggiunge dinamicamente** un nuovo pulsante ad ogni click

function addDynamicMenu() {

menu.append(

new MenuItem({

label: 'aggiunto'

})

)

Menu.setApplicationMenu(menu)

}

**Nota**: ogni aggiunta richiede una **reinizzializzaione** del menu.

È possibile, oltre che aggiungere alla fine della lista, anche inserire in una posizione specifica un nuovo pulsante

function insertDynamicMenu() {

menu.insert(2,

new MenuItem({

label: 'aggiunto2'

})

)

Menu.setApplicationMenu(menu)

}

* **Richiamare una voce di menu specifica usando la funzione**

menu.getMenuItemById('btn').enabled = false;

*Es. di disabilitazione di una voce di menu con id ‘****btn’****.*

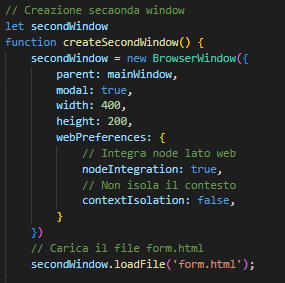
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 9 – Gestire più finestre**



La creazione di ulteriori finestre in Electron è assimilabile ala creazione della principale

Tramite l’inizializzazione di un’altra **BrowserWindow** è possibile descrivere una seconda finestra.



La finestra può essere **indipendente** dalla principale o **relazionarsi** ad essa.

Nell’esempio sopra definiamo la **finestra principale** come finestra padre tramite l’attributo **parent.**

Da ciò possiamo trattare la seconda finetra come un **modale**, usando l’attributo sopra scritto.

Da ciò deriva la possibilità di usare la funzione **quit()** che, alla chiusura della finestra principale, permette la chiusura delle altre, oltre che l’uscita dall’intera applicazione

// Alla chiusura di questa finestra principale verrà chiusa l'intera Applicazione (con tutte le finstre secondarie a seguito)

mainWindow.on('closed', () => {app.quit()})

* **Comunicazione tra finestre:**

E’ possibile far comunicare più finestre tramite l’**Arco IPC**, un sistema che utilizza ipcRenderer (lato web page) e ipcMain (lato Main process) in invio (**send**) e ascolto (**on**) che utililizzano come nodo il file **main.js**.

**Nota**: per motivi di **performance** è consigliabile, oltre alla chiusura, nullificare la finestra secondaria.

secondWindow.close();

secondWindow = null;

**Nota 2**: Il problema della comunicazione tra finestre non si pone in classiche applicazioni, il passaggio dei dati avviene in modo congruo alla tecnologia usata, si pone se si vuole passare il dato nella modalità di Electron.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 10 – Icona Tray**



Sono icone dei programmi sempre attivi o dei processi in esecuzione in background.

Queste icone permettono di minimizzare, effettuare operazioni veloci e mantenere in background il programma, senza che questo occupi la barra delle applicazioni in modo massivo.

Le proprietà fondamentali che permettono la gestione dell’icona tray sono Tray e nativeImage.

icon = nativeImage.createFromPath(path.join(\_\_dirname,'trayIcon.png'));

tray = new Tray(icon)



**Inizializzazione e utilizzi principali:**

1. Istanziamento della variabile di appoggio

let tray = null

let icon = null

1. Definizione del path e inizzializzazione dell’icona

icon = nativeImage.createFromPath(path.join(\_\_dirname,'trayIcon.png'));

tray = new Tray(icon)

1. Applicazione di un menu attivabile con il tasto dx

const contextMenu = Menu.buildFromTemplate([

{label: 'chiudi', role: 'quit'}

])

tray.setContextMenu(contextMenu)

1. Aggiunta di un titolo

tray.setTitle('Title Applicazione di Giacomo')

1. Aggiunta di un tooltip

tray.setToolTip('Tooltip Applicazione di Giacomo')

1. Minimizzazione della finestra principale nell’icona

mainWindow.on('minimize', (event) => {

mainWindow.setSkipTaskbar(true)

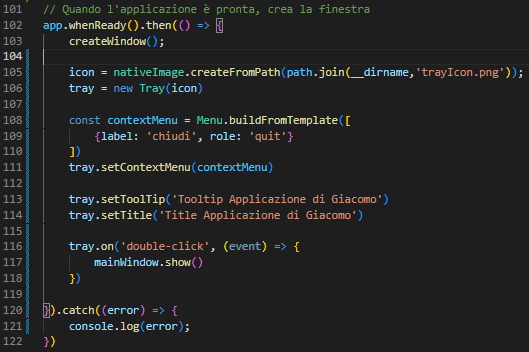
})

1. Set, all’evento ‘double.click’, dell’apertura della finestra principale dell’applicazione

tray.on('double-click', (event) => {

mainWindow.show()

})



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 11 – Context Menu**

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

Il **menu contestuale** (o **context-menu**) è un elenco di possibili interazioni che è possibile effettuare con l'oggetto con cui si vuole interagire. L'attributo "contestuale" fa riferimento al fatto che le voci del menu dipendono dal tipo di oggetto (o oggetti) con cui si vuole interagire e dall'applicazione che gestisce l'oggetto, nel senso che le voci dipendono dal contesto in cui si sta lavorando e cambiano in relazione ad esso.

Ci sono **3 possibilità** che permettono di creare/gestire un context-menu:

1. **Soluzione web app (Non Nativa)**

Ovvero la creazione di una modale che tramite eventi elencati permetta di gestire delle operazioni.

1. **Soluzione diretta Electron (Nativa)**

Uso delle direttive Electron che permettono di creare direttamente un menu contestuale.

1. **Soluzione tramite libreria (Semi Nativa)**

Uso di una libreria che astrae e permette di creare un menù contestuale in modo più semplice, ma meno diretto.

Adesso vedremo la 2 e la 3.

* **Metodo nativo (2)**

Aggiungiamo un event listener alla nostra finestra principale, la quale aspetterà un click del tasto dx per aprire un menu

const template = [

{label: 'bottone1', click: () => {console.log('test click su pulsante menu')}},

{label: 'bottone2', submenu: [

{label: 'sottobottone1', click: () => {console.log('test 1')}},

{type: 'separator'},

{label: 'sottobottone2', click: () => {console.log('test 2')}, enabled: false},

{label: 'sottobottone3', click: () => {console.log('test 3')}}

]},

{label: 'OpenSecondWindow', click: createSecondWindow},

]

*Creazione del template del menu*

const menu = Menu.buildFromTemplate(template)

*Costruzione del menu*

Menu.setApplicationMenu(menu)

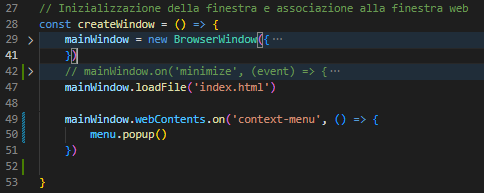
*Inizializzazione del menu*

mainWindow.webContents.on('context-menu', () => {

menu.popup()

})

*Aggiunta event listener sulla finestra principale*



* **Metodo semi-nativo (3)**

Verrà usata la libreria **electron-context-menu**

1. Basta importare a libreria

const contextMenu = require('electron-context-menu');

1. E inizializzarla

contextMenu({

})

Dentro l’oggetto sarà possibile inserire varie proprietà che saranno le azioni che la libreria automatizza



*Mostra il “copia immagine”*

e sarà possibile dare un’etichetta personalizzata ad ognuna

contextMenu({

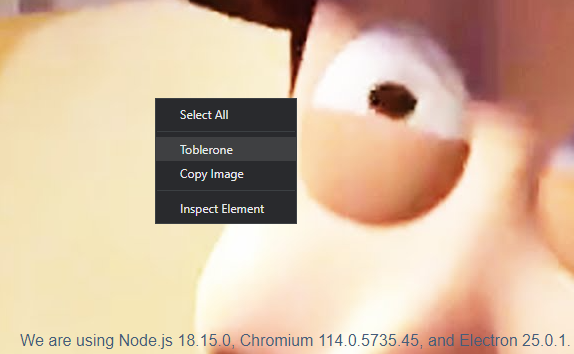
showSaveImageAs: true,

labels: {

saveImageAs: 'Toblerone'

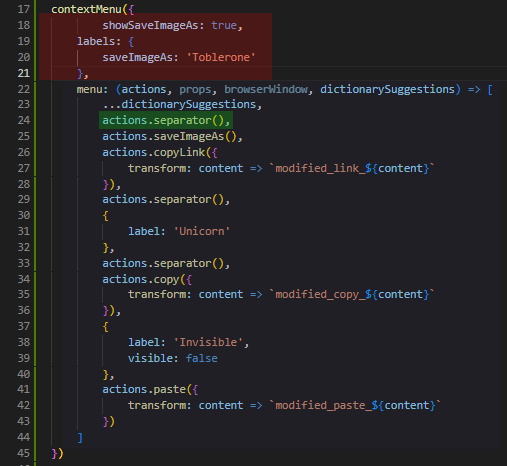
}

})



E' possibile settare in modo automatico il menu **importando dei set** di default dalla libreria.

Questo permette comunque di effettuare **personalizzazioni**.



* **Label personalizzato**
* **Menu importato**
* **Init della funzionalità che porterà la nostra customizzazione**

È possibile anche aggiungere label personalizzate all’interno del menu:

menu: (actions, props, browserWindow, dictionarySuggestions) => [

actions.separator(),

{

label: 'Unicorn', click: () => {console.log('Wooody')}

},

actions.separator(),

]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 12 – Notifiche**

**Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente**

L’attivazione di notifiche di Sistema è relativamente semplice e usa l’oggetto **Notification** presente nel set di proprietà di ElectronJs.

Dopo l’import, bisognerà **istanziare** un che verrà definito da una serie di proprietà, quali title, subtitle, icon, ecc…

…usare il metodo **show()** per visualizzarlo.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteÈ possibile definire degli eventi attraverso i quali triggerare delle operazioni.

**Immagine che contiene testo, schermata, Software multimediale, software

Descrizione generata automaticamente**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 13 – Scorciatoie da tastiera**



Esecuzioni di operazioni tramite shortcut di tastiera

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Questo sistema, posto a livello globale, agirà in ogni componente dell’applicazione.

**Ma se dovessimo specificare la shortcut dentro una specifica finestra?**

In quel caso dovremo agire sui webContents della finestra di riferimento

  mainWindow.webContents.on("before-input-event", (event, input) => {

    if (input.control && input.alt && input.key.toLowerCase() === "i") {

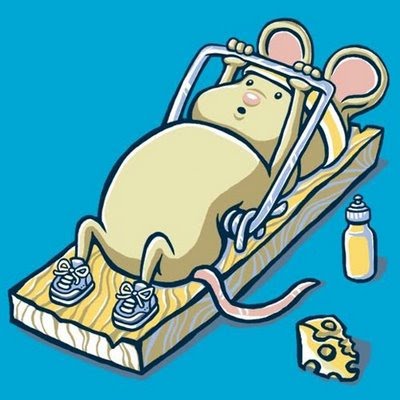
      event.preventDefault();

      console.log("Scorciatoia specifica");

    }

  });

Agendo sulla finestra specifica (in questo caso la principale), potremo eseguire, all’ascolto dell’evento di input, un’azione, previo controllo che gli input siano i tasti **Ctrl**, **Alt** e **i**.

Un’alternativa è usare la libreria **mousetrap**.

Questa velocizzerà l’uso delle shortcut da tastiera, permettendone l’uso direttamente nello script.js della web page.

var Mousetrap = require("mousetrap");

Mousetrap.bind("ctrl", function () {

  console.log("ctrl shift k");

});

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 14 – Creare file per installazione con Electron Forge**



Di default ElectronJs non dispone di un sistema che permetta la creazione di un file eseguibile.

Per questo ci viene in aiuto Electron Forge, una cli che creare un package eseguibile della nostra applicazione.

Protocollo d’uso:

1. Installazione della cli di electron-forge

**npm i -D @electron-forge/cli**

1. Installazione di tutti i pacchetti utili a forge per pacchettizzare l’applicazione

**npx electron-forge import**

**A questo punto i comandi del nostro package.json saranno stati cambiati in quelli supportati da electron-forge!**

1. Aggiungiamo uno script di creazione dell’eseguibile nel package.json



Nota: potrebbero essere richieste delle info da inserire nel **package.json**, nel qual caso capitasse, basta aggiungerle affinché il processo vada avanti.

1. Quando il processo sarà andato a buon fine, basterà cliccare sul file di setup per eseguire l’applicazione!

Immagine che contiene testo, schermata, software, Software multimediale

Descrizione generata automaticamente

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Lezione 15 – Aggiornare e firmare l’app**

Il processo di **aggiornamento** dell’applicazione può essere eseguito tramite package installabili, questi **notificheranno l’utente** che esistono degli aggiornamenti dell’applicazione, ma c’è un **ma**.

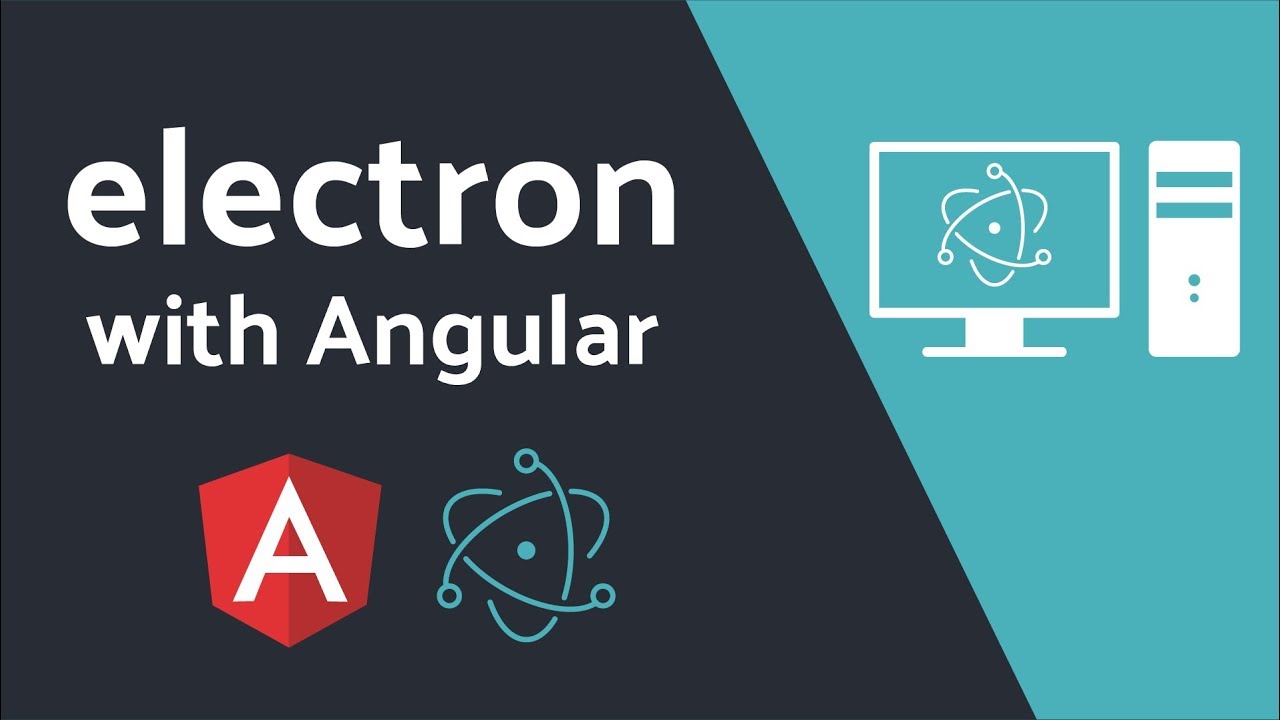
Una **applicazione desktop**, a differenza di una web, richiede una **firma**, che abiliti l’applicazione ad essere installata ed usata sui vari sistemi operativi. In assenza di questa, l’applicazione, in fase di installazione potrebbe avere dei problemi nell’esecuzione o addirittura essere **non installabile**.

La firma, per essere ottenuta, deve essere comprata in degli **store online** che verifichino la stabilità e non malevolità del programma.

I costi si aggirano tra i **150 $** e i **400 $**.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Appendice – Costruire un’app con Angular ed ElectronJs**



Per questa sezione vedi **repository**: <https://github.com/GiacomoBorsellino/electronjs-course>

…pezzi appunti altro…

Nota: Ricorda di **inizializzare** la **mainWindow** nell’**appWhenReady()**

* **Modifica lo script**

**"start": "ng build --base-href ./ && electron .",**

*ovvero, crea la build, prendi l’app.js del path ./ e poi avvia electron che avvierà la finestra.*

* Aggiungiamo il sistema di comunicazione tra angular ed electron

**npm i ngx-electron-fresh**

* Importiamo il NgxElectronModule nell’app.module dell’applicazione

Immagine che contiene testo, schermata, software, schermo

Descrizione generata automaticamente

**Nota**: [in fase di import potrebbe essere necessario riavviare il server Angular](https://stackoverflow.com/questions/60290309/error-ng6002-appears-in-the-ngmodule-imports-of-appmodule-but-could-not-be-res)

* Import del service electron nel nostro controller angular



* Istanziamento nel costruttore



* E tramite una funzione inviamo, tramite il metodo ipcRender, verso il main process di electron (app.js o main.js), dopo aver controllato che questa si una applicazione electron

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

* Nel main.js (o app.js) ascoltiamo e riceviamo la comunicazione dal controller angular e inviamo verso la main web page (mainWindow) una nuova comunicazione

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

* Nel controller del nostro componente mettiamo in ascolto di questa comunicazione definendo nel costruttore il **listener** dell’**ipcRender**.

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Questo è lo sviluppo base di una comunicazione IPC tra Electron e Angular, per vedere le restanti possibilità e operatività, vedere il resto degli appunti.