**Corso ElectronJs**



Lezione 1 - Introduzione

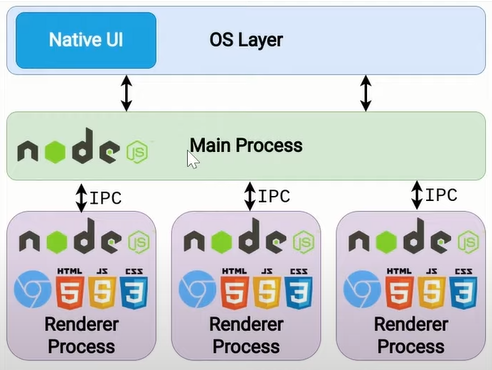
E’ un framework js che permette di scrivere applicazioni desktop cross-platform.

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

Applicazioni che usano ElectronJS:

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

**Struttura:**



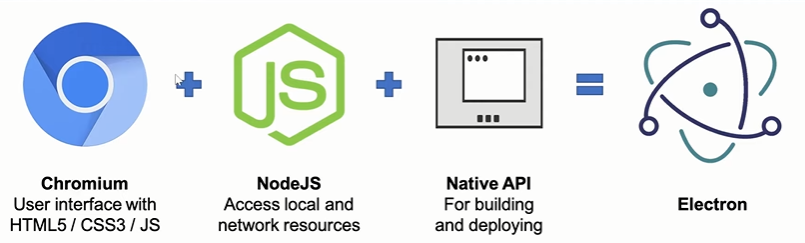
Chromium : Finestra che ci permette di renderizzare HTML,CSS e JS

NodeJs : 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 thread Node) con I processi di rendering lato chromium, così che I render processsingoli possano comunicare tra loro.

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



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

Lezione 2 – Struttura

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**

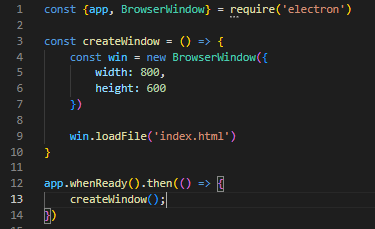
npm i electron --save-dev

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

Import del pacchetto electron

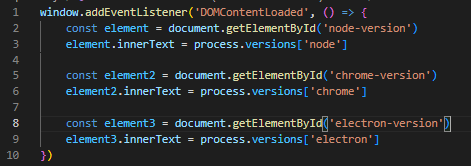
Inizializzazione di una nuova finestra assegnando ad essa una nuova “finestra del browser”

Caricamento come finestra del browser del file index.html

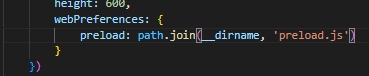


* Carichiamo dei dati prima del caricamento della pagina

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



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 thread 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

E’ 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

E’ possibile sostituire il frame con la barra delle applicazioni predefinità creandone una custom aggiungendo le funzionalità di **minimizzazione, fullscreen e chiusura**.



Utilizzando dei bottoni che inviino verso il main thread 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 finstra 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 7 – Barra del menu

E’ 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 **etichetta** al 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:**

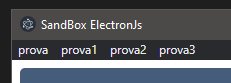
// Costruzione del menu applicando il template

const menu = Menu.buildFromTemplate(template)

* **Inizializzazione del nuovo Menu:**

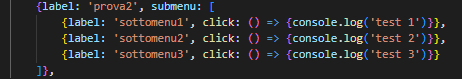
// Inizializzare il menu

Menu.setApplicationMenu(menu)



*Risultato*

*Nel template è possibile definire un sotto menu...*



...*, 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.

E’ 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’.*