GUIDA ALL'USO DI ELECTRON

1. TEORIA DI BASE SU ELECTRON

Cos'è Electron?

Electron è un framework per costruire **applicazioni desktop multipiattaforma** (Windows, macOS e Linux) utilizzando **tecnologie web**: JavaScript, HTML e CSS.

Electron unisce due tecnologie principali:

- **Chromium** – il motore di rendering di Google Chrome, usato per l'interfaccia utente
- **Node.js** – ambiente per JavaScript lato server, che permette di interagire con il filesystem, processi, rete, ecc.

> Quindi: con Electron puoi costruire app che sembrano native, ma sono scritte come una web app.

Architettura di base di un'app Electron

- **Main Process**

  Il cuore dell'applicazione. Gestisce le finestre, comunica con il sistema operativo e coordina tutto. Usa Node.js.

- **Renderer Process**

  È ogni finestra dell'app. Mostra l'interfaccia utente e funziona come una pagina HTML. Può eseguire codice JavaScript e comunicare col Main Process.

---

2. STRUMENTI NECESSARI

Requisiti

- **Node.js e npm** (npm viene installato con Node): https://nodejs.org
- Un editor di testo: ti consiglio **Visual Studio Code**

Verifica installazione

Apri il terminale e digita:

```
node -v
npm -v
```

Se vedi le versioni, sei pronto.

---

3. CREAZIONE DI UN NUOVO PROGETTO

Passo 1: Inizializza il progetto

Apri il terminale:

```
mkdir mia-app-electron
cd mia-app-electron
npm init -y
```

Questo crea un file `package.json` con le informazioni di base del progetto.

Passo 2: Installa Electron

```
npm install electron --save-dev
```

Installa Electron come dipendenza di sviluppo.

Passo 3: Struttura iniziale dei file

Crea questi file nella root del progetto:

```
/mia-app-electron
│
├── package.json
```

```
├── main.js        ← entry point: processo principale
├── preload.js     ← (opzionale) ponte sicuro tra Main e Renderer
└── index.html     ← interfaccia utente
```

---

4. SCRIVERE IL CODICE

`package.json`

Aggiorna il file `package.json` così:

```
{
  "name": "mia-app-electron",
  "version": "1.0.0",
  "main": "main.js",
  "scripts": {
    "start": "electron ."
  },
  "devDependencies": {
    "electron": "^latest_version"
  }
}
```

La chiave `"main"` dice a Electron da dove partire: `main.js`.

`main.js` – Processo principale

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

function creaFinestra () {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, 'preload.js')
    }
  })

  win.loadFile('index.html')
}

app.whenReady().then(() => {
  creaFinestra()

  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) creaFinestra()
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') app.quit()
})
```

Cosa fa questo codice:

- Quando l'app è pronta (`app.whenReady()`), crea una finestra (`BrowserWindow`)
- Carica `index.html`
- Su Mac, ricrea la finestra se viene chiusa (comportamento standard)
- Chiude l'app se tutte le finestre vengono chiuse

`preload.js` – Ponte tra Main e Renderer

```
// Preload – sicuro, separa logica di sistema da interfaccia
window.addEventListener('DOMContentLoaded', () => {
  const sostituzioni = {
    versioneElectron: process.versions.electron,
    versioneNode: process.versions.node,
    versioneChrome: process.versions.chrome
  }

  for (const chiave in sostituzioni) {
    const elemento = document.getElementById(chiave)
    if (elemento) {
      elemento.innerText = sostituzioni[chiave]
    }
  }
})
```

`index.html` – Interfaccia utente

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>La mia app Electron</title>
</head>
<body>
  <h1>Benvenuto!</h1>
  <p>Electron: <span id="versioneElectron"></span></p>
  <p>Node.js: <span id="versioneNode"></span></p>
  <p>Chrome: <span id="versioneChrome"></span></p>
</body>
</html>
```

Questo HTML visualizza le versioni di Electron, Node e Chrome usando il preload.

---

5. AVVIARE L'APPLICAZIONE

Nel terminale:

```
npm start
```

Si aprirà una finestra desktop con le informazioni sulle versioni.

---

6. COSA PUOI FARE DOPO?

Ora che la struttura è pronta, puoi iniziare a **costruire la tua app vera e propria**:

- Usare un framework frontend (React, Vue, ecc.)
- Comunicare tra main e renderer con IPC
- Salvare dati in un file o database locale (es. SQLite, NeDB, LowDB)
- Aggiungere funzionalità come:
  - Menu personalizzati
  - Notifiche di sistema
  - Drag & drop
  - Interazione col filesystem

AGGIUNGERE UN MENÙ PERSONALIZZATO

Obiettivo:

Aggiungere un menù come quello classico delle app desktop ("File", "Modifica", "Visualizza", ecc.), con voci personalizzate e scorciatoie da tastiera.

Passo 1 – Importare `Menu` in `main.js`

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

Passo 2 – Creare la struttura del menu

Inserisci questa funzione in `main.js` **prima** di `app.whenReady()`:

```
function creaMenuPersonalizzato() {
  const modelloMenu = [
    {
      label: 'File',
      submenu: [
        {
          label: 'Salva',
          accelerator: 'CmdOrCtrl+S',
          click: () => {
            console.log('Hai cliccato su Salva')
          }
        },
        {
          label: 'Esci',
          accelerator: 'CmdOrCtrl+Q',
          role: 'quit'
        }
      ]
    },
    {
      label: 'Visualizza',
      submenu: [
```

```
      {
        label: 'Ricarica',
        role: 'reload'
      },
      {
        label: 'Apri DevTools',
        role: 'toggleDevTools'
      }
    ]
  }
]

const menu = Menu.buildFromTemplate(modelloMenu)
Menu.setApplicationMenu(menu)
}
```

Passo 3 – Chiamare la funzione nel punto giusto

All'interno di `app.whenReady()`, **dopo** aver creato la finestra:

```
app.whenReady().then(() => {
  creaFinestra()
  creaMenuPersonalizzato()
})
```

Risultato:

Hai un menù con:

- Scorciatoie (Ctrl+S per salvare, Ctrl+Q per uscire)
- Accesso a strumenti di sviluppo
- Possibilità di estendere facilmente il menu

Puoi ovviamente aggiungere funzioni specifiche nel `click()` di ogni voce.

GESTIRE LA COMUNICAZIONE TRA MAIN E RENDERER (IPC)

Obiettivo:

Permettere al processo Renderer (la tua finestra) di **inviare messaggi** al processo Main e **riceverne risposte**.

Concetti base:

- `ipcMain`: usato nel `main.js`, riceve i messaggi
- `ipcRenderer`: usato nel frontend, per inviare/ricevere

Passo 1 – Prepara il Preload (`preload.js`)

Nel file `preload.js`, esponi un'API sicura al frontend:

```
const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('elettrone', {
  inviaMessaggio: (canale, dati) => ipcRenderer.send(canale, dati),
  riceviMessaggio: (canale, callback) => ipcRenderer.on(canale, (event, ...args) => callba
ck(...args))
})
```

Passo 2 – Main process: riceve il messaggio (`main.js`)

Importa `ipcMain` e aggiungi un listener:

```
const { ipcMain } = require('electron')

ipcMain.on('messaggio-dal-renderer', (event, arg) => {
  console.log('Ricevuto dal Renderer:', arg)

  // invia una risposta al Renderer
  event.reply('risposta-dal-main', 'Ciao dal Main process!')
})
```

Passo 3 – Renderer: invia e riceve (`index.html`)

Nel file HTML, aggiungi un bottone:

```
<button onclick="invia()">Invia Messaggio</button>
<p id="risposta"></p>

<script>
  function invia() {
    window.elettrone.inviaMessaggio('messaggio-dal-renderer', 'Ciao Main!')
  }

  window.elettrone.riceviMessaggio('risposta-dal-main', (dati) => {
    document.getElementById('risposta').innerText = dati
  })
</script>
```

Risultato:

- Cliccando il bottone, il Renderer invia un messaggio al Main
- Il Main risponde con un altro messaggio
- Il Renderer aggiorna l'interfaccia con la risposta ricevuta

---

CREARE UN ESEGUIBILE CON ELECTRON FORGE O ELECTRON BUILDER

Opzione 1 – Electron Forge (più semplice)

Passo 1 – Installazione

Nel tuo progetto:

```
npx electron-forge import
```

Forge aggiornerà `package.json` e creerà una struttura di base per il packaging.

Passo 2 – Avviare l'app con Forge

```
npm start
```

Passo 3 – Generare il pacchetto

```
npm run make
```

Troverai l'eseguibile nella cartella `out/`.

- Su **Windows**: un file `.exe`
- Su **macOS**: un file `.dmg`
- Su **Linux**: un file `.deb` o `.AppImage`

Opzione 2 – Electron Builder (più avanzato e flessibile)

Passo 1 – Installazione

```
npm install electron-builder --save-dev
```

Nel `package.json`, aggiungi:

```
"scripts": {
  "start": "electron .",
  "build": "electron-builder"
}
```

Aggiungi anche una sezione `build`:

```
"build": {
  "appId": "com.miaapp.electron",
  "productName": "MiaApp",
  "files": [
    "**/*"
  ],
  "directories": {
    "output": "dist"
  }
}
```

Passo 2 – Eseguire la build

```
npm run build
```

Troverai l’eseguibile nella cartella `dist/`.

> Electron Builder supporta anche auto-update, icona personalizzata, firma digitale, ecc.

Quale scegliere?

Scopo	Usa Forge se...	Usa Builder se...
Inizi a esplorare Electron	Vuoi semplicità	Vuoi opzioni avanzate
Distribuzione semplice	Non ti serve firma o aggiornamenti automatici	Vuoi installatori professionali
Multi-piattaforma	Sì	Sì (meglio supportato)