Creación de aplicación de escritorio con Electron Js

En este documento vamos a hacer una breve introducción en el uso del módulo **`Electron Js`** el cual nos permite empaquetar proyectos creados en JavaScript y convertirlos en una aplicación de escritorio la cual puede ser ejecutada en diferentes sistemas operativos.

En este ejemplo reutilizaremos gran parte del código presentado en el documento anterior (Creación **de proyecto de `NodeJs` con `SocketIo` en Visual Studio Code**) para crear una aplicación en donde iremos registrando distintas solicitudes de inscripción a un catedra de la facultad.

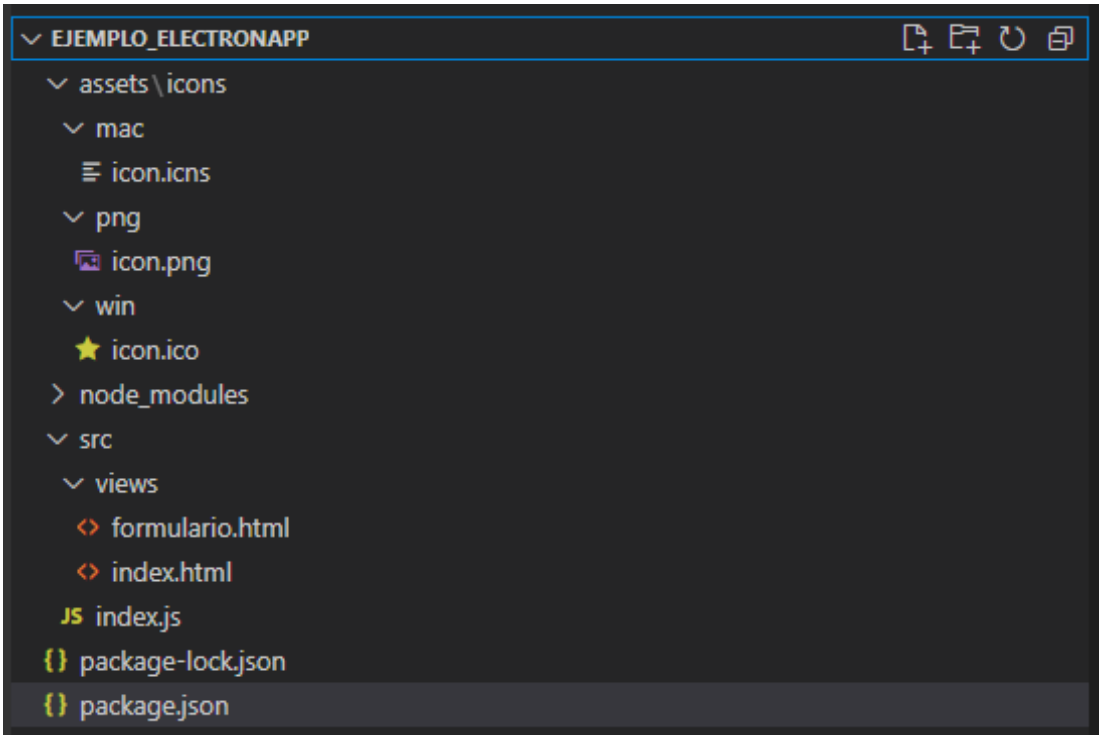
Para ello debemos iniciar un nuevo proyecto `NodeJs` en Visual Studio Code exactamente como ya se ha explicado. Ejecutamos el comando:

```
npm init --yes
```

Luego instalaremos el módulo `Electron` ejecutando:

```
npm install electron
```

Lo siguiente será crear las carpetas y archivos tal cual se muestra en la siguiente imagen:



Recordar que la carpeta `node_modules` y el archivo `package-lock.json` son generados automáticamente luego de instalar el módulo **`electron`**.

Dentro del directorio `assets/icons` encontraremos los respectivos iconos para cada sistema operativo. Estos archivos serán usados a la hora de empaquetar la aplicación.

Lo que debemos hacer ahora es configurar el archivo `package.json`, el cual hasta el momento se ve de la siguiente manera:

```
{
  "name": "Ejemplo_ElectronApp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "electron": "^13.1.8"
  }
}
```

Agregaremos el siguiente atributo debajo de versión:

```
"productName": "App Inscipcion electron"
```

Este será el nombre que tendrá nuestra app luego de haber sido empaquetada. También editaremos el atributo `"description"` con una breve descripción sobre la funcionalidad de la app. En el atributo `"main"` indicaremos el directorio del archivo principal de nuestra aplicación, es decir en `src/index.js`.

Ahora editaremos el atributo `"scripts"` borrando el comando `"test"` y añadiremos lo siguiente:

```
"start": "(set NODE_ENV=development)&&(electron src/index.js)",
"package-mac": "electron-packager . --overwrite --platform=darwin --
arch=x64 --icon=assets/icons/mac/icon.icns --prune=true --out=release-
builds",
"package-win": "electron-packager . electron-tutorial-app --overwrite --
asar=true --platform=win32 --arch=ia32 --icon=assets/icons/win/icon.ico -
-prune=true --out=release-builds --version-string.CompanyName=CE --
version-string.FileDescription=CE --version-
string.ProductName=\"Electron Tutorial App\"",
"package-linux": "electron-packager . electron-tutorial-app --overwrite -
-asar=true --platform=linux --arch=x64 --icon=assets/icons/png/icon.png -
-prune=true --out=release-builds"
```

Con el comando `npm start` iniciaremos la aplicación en modo desarrollador, es decir, tendremos a disposición las DevTools para debugguear posibles errores en el código.

Por otro lado los comandos `npm run package-<OperativeSystem>` nos permitirán empaquetar la aplicación en para el sistema operativo elegido. La carpeta de la aplicación empaquetada se creara dentro de nuestro proyecto en el directorio `release-builds.`

Ya habiendo hecho todos estos cambios en el `package.json` nos resta instalar el modulo empaquetador de `Electron`. Lo haremos mediante el comando:

```
npm install electron-packager
```

Por ultimo completaremos el atributo `"author"` con nuestro nombre y a todo el atributo `"Dependencies"` lo remplazaremos por lo siguiente:

```
  "Dependencies": {

  },
  "devDependencies": {
    "electron": "^13.1.8",
    "electron-packager": "^15.3.0"
  }
```

De esta forma se indicará que `"electron"` y `"electron-packager"` solo serán dependencias usadas durante el desarrollo la app. El archivo `package.json` deberá lucir de la siguiente forma:

```
{
  "name": "Ejemplo_ElectronApp",
  "productName": "App Inscipcion electron",
  "version": "1.0.0",
  "description": "ejemplo de uso de electron",
  "main": "src/index.js",
  "scripts": {
    "start": "(set NODE_ENV=development)&&(electron src/index.js)",
    "package-mac": "electron-packager . --overwrite --platform=darwin --
arch=x64 --icon=assets/icons/mac/icon.icns --prune=true --out=release-
builds",
    "package-win": "electron-packager . electron-tutorial-app --
overwrite --asar=true --platform=win32 --arch=ia32 --
icon=assets/icons/win/icon.ico --prune=true --out=release-builds --
version-string.CompanyName=CE --version-string.FileDescription=CE --
version-string.ProductName=\"Electron Tutorial App\"",
    "package-linux": "electron-packager . electron-tutorial-app --
overwrite --asar=true --platform=linux --arch=x64 --
icon=assets/icons/png/1024x1024.png --prune=true --out=release-builds"
  },
  "keywords": [],
  "author": "Jonh Doe",
  "license": "ISC",
  "Dependencies": {

  },
  "devDependencies": {
    "electron": "^13.1.8",
    "electron-packager": "^15.3.0"
  }
}
```

Ahora si ya estamos en condiciones de comenzar a programar la aplicación:

El archivo `index.js` será el código que ejecute el proceso principal de la app. Por lo cual comenzaremos por ahí. En dicho archivo escribiremos:

```
const {app, BrowserWindow, Menu, ipcMain} = require('electron');
const url = require('url');
const path = require('path');
```

De esta forma requerimos los módulos que usaremos para la ejecución del código.

Del modulo `'electron'` obtendremos las siguientes propiedades:

`app:` manejaremos las funcionalidades del proceso principal;

`BrowserWindow:` gestionaremos las ventanas que tendrá la app;

`Menu:` nos permitirá personalizar el menú superior de la app;

`ipcMain:` se encargará de la comunicación mediante escucha y emisiones de eventos (muy parecido a los WebSockets).

Creamos dos constantes las cuales serán usadas para crear las ventanas que tendrá nuestra app

```
let mainWindow;
let nuevaInscripcionWindow;
```

Con el evento `'ready'` damos comienzo a la ejecución de la app.

```
app.on('ready', () => {

});
```

Dentro de la función flecha ejecutaremos los siguiente:

```
        mainWindow = new BrowserWindow({
        width: 800,
        height: 500,
        webPreferences: {
            nodeIntegration: true,
            contextIsolation: false
        }
    });
```

De esta forma estamos iniciando una nueva ventana que en este caso sera la ventana principal. Los atributos `width` y `height` establecen el tamaño por defecto de la ventana y dentro de `webPreferences` pondremos el siguiente atributo `nodeIntegration: true` el cual nos garantiza que las ventanas abiertas tengan la integración de NodeJs.

Ahora le diremos a la ventana cual será el archivo `.html` que deberá ejecutar al abrirse, esto lo haremos mediante

```
    mainWindow.loadURL(url.format({
        pathname: path.join(__dirname, 'views/index.html'),
        protocol: 'file',
```

```
        slashes: true
    }));
```

Ahora lo que haremos es crear un menú para la ventana desde una plantilla de declararemos por fuera de del evento `'ready'`. Esto lo hacemos con las siguientes líneas:

```
    const mainMenu = Menu.buildFromTemplate(templateMenu);
    Menu.setApplicationMenu(mainMenu);
```

Primero construimos el `mainMenu` desde el template y luego se lo asignamos a la aplicación haciendo uso de `Menu`.

Por ultimo escucharemos el evento `'closed'` de la ventana principal y cerraremos la aplicación si esto ocurre. Si hubieran sub-ventanas abiertas las mismas serán cerradas también.

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

Para generar la plantilla del menú principal escribiremos lo siguiente

```
const templateMenu = [

{
    label: 'Archivo',
    submenu:   [   {
                    label: 'Nueva Inscripción',
                    accelerator: 'Ctrl+N',
                    click(){
                        nuevaInscripcion();
                    }
                },

                {
                    label: 'Remover inscrpciones',
                    click(){
                      mainWindow.webContents.send('borrarInscripciones');
                    }
                },

                {
                    label: 'Salir',
                    accelerator: 'Ctrl+Q',
                    click(){
                    app.quit();
                    }
                }
            ]
}
];
```

Notar que el menú principal es un arreglo de objetos en donde cada objeto representa una pestaña distinta. Cada pestaña es un arreglo de objetos también en donde cada objeto conforma un elemento de ese submenú. Esta estructura puede ser repetida cuantas veces quieras para así obtener el template que se desee.

Para darle una acción a algún elemento del menú se utiliza la función `click(){}` en la cual se llevaran a cabo las actividades que se deseen al hacer click sobre él.

Así por ejemplo al darle click en el sub-elemento `'Nueva Inscripción'` del elemento `'Archivo'` en el `Menu` será ejecutada la siguiente función. La cual creara una nueva de forma similar a la ventana principal.

```
function nuevaInscripcion(){
    nuevaInscripcionWindow = new BrowserWindow({
        width: 400,
        height: 500,
        title: 'Formulario de Incripción',
        webPreferences: {
            nodeIntegration: true,
            contextIsolation: false
            }
    });
    nuevaInscripcionWindow.setMenu(null);
    nuevaInscripcionWindow.loadURL(url.format({
        pathname: path.join(__dirname, 'views/formulario.html'),
        protocol: 'file',
        slashes: true
    }));
    nuevaInscripcionWindow.on('closed', () => {
        nuevaInscripcionWindow = null;
    })


}
```

Por ultimo veremos cómo se utiliza `ipcMain` para comunicar las diferentes ventanas de la app. En este caso el proceso principal estará escuchando el evento `'nuevaIscripcion'` el cual será emitido por la ventana secundaria de nuestra app al darle clic en Enviar formulario.

```
ipcMain.on('nuevaIscripcion', (e, datos) => {
    mainWindow.webContents.send('nuevaInscripcion', datos);
    nuevaInscripcionWindow.close();
});
```

Dentro del evento `'nuevaIscripcion'` lo que aremos es reenviar el objeto `datos` datos recibido a la ventana principal `mainWindow` con el método `webContents.send();`. Por ultimo cerraremos la ventana secundaria `nuevaInscripcionWindow` con `close();`

Ya sea desde la ventana `mainWindow` o `nuevaInscripcionWindow` deberemos importar el parámetro `ipcRenderer` mediante la siguiente línea de codigo.

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

Desde la ventana `mainWindow` ecucharemos el evento `'nuevaInscripcion'` y llevamos a cabo las tareas necesarias para mostrar en la pantalla las inscripciones que fueron emitidas.

```
        ipcRenderer.on('nuevaInscripcion', (e, datos)=> {

        });
```

Mientras que en la ventana `nuevaInscripcionWindow` emitiremos dicho evento enviando la objeto `datos` mediante la siguiente línea:

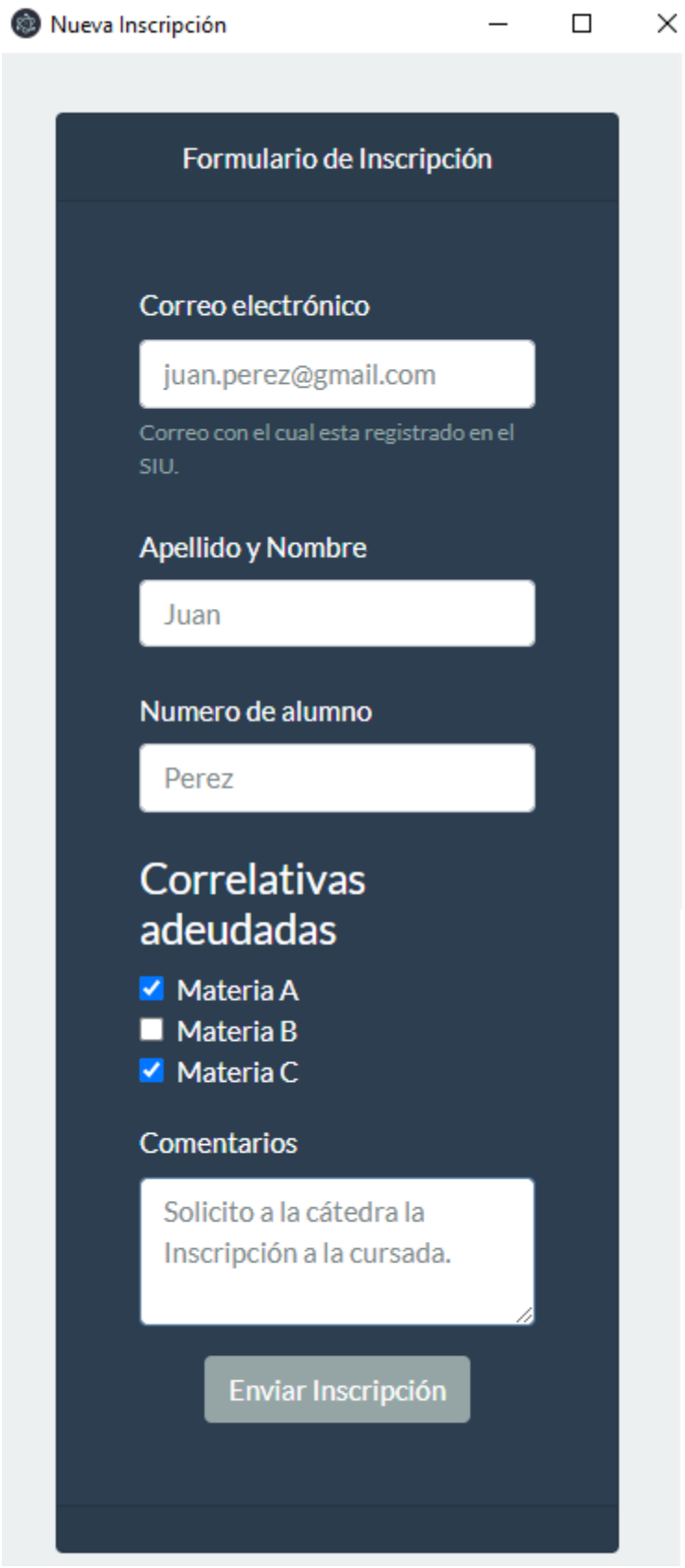
```
ipcRenderer.send('nuevaIscripcion',datos);
```

Con todo esto explicado ya estamos en condiciones de ejecutar nuestro nuestra app con el comando:

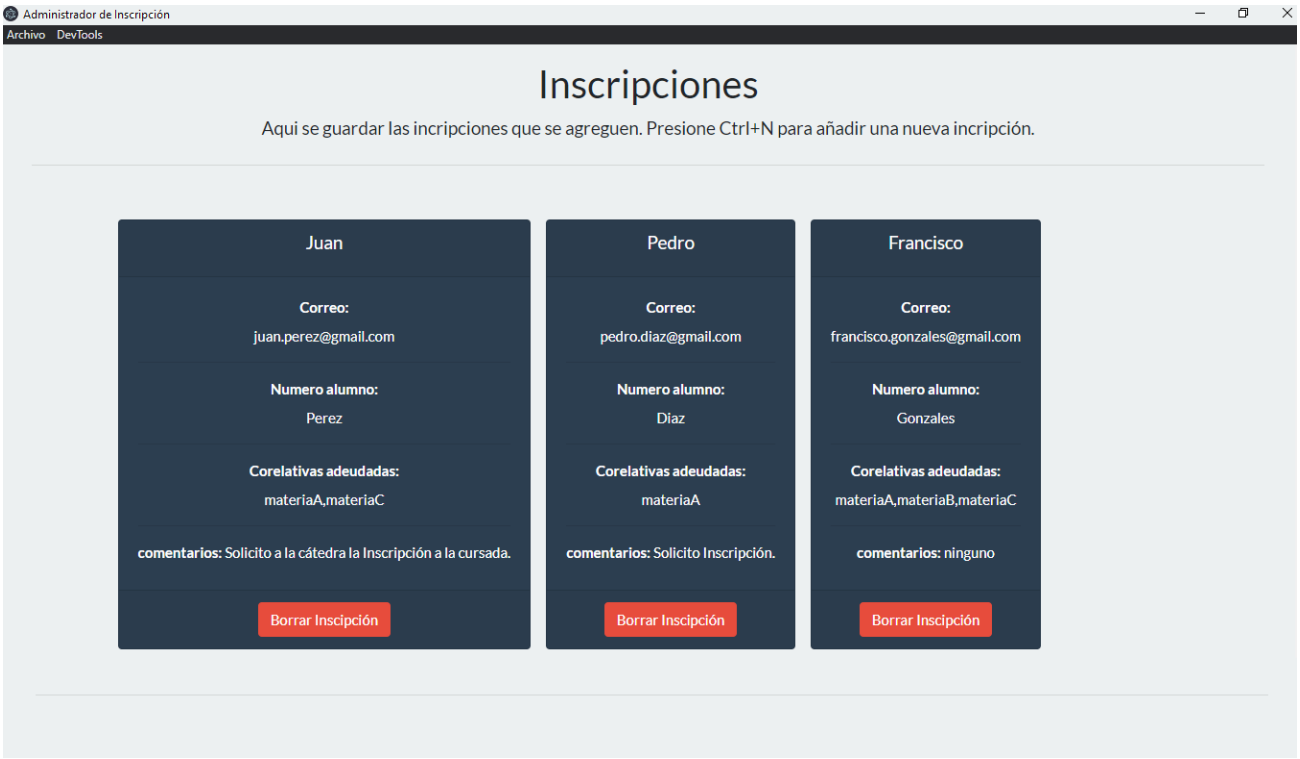
```
npm start
```

Se nos abrirá la siguiente ventana.

Con `Ctrl+N` podremos crear una nueva inscripción o si no yendo a Archivo>Nueva inscripción.



Una vez enviada la inscripción se agregará en la ventana principal de la app desde la cual se podrán eliminar de a una o todas juntas.


Administrador de Inscripción
Archivo DevTools
Inscripciones
Aqui se guardar las incripciones que se agreguen. Presione Ctrl+N para añadir una nueva incripción.
Juan
Correo:
juan.perez@gmail.com
Numero alumno:
Perez
Corelativas adeudadas:
materiaA,materiaC
comentarios: Solicito a la cátedra la Inscripción a la cursada.
Borrar Inscipción
Pedro
Correo:
pedro.diaz@gmail.com
Numero alumno:
Diaz
Corelativas adeudadas:
materiaA
comentarios: Solicito Inscripción.
Borrar Inscipción
Francisco
Correo:
francisco.gonzales@gmail.com
Numero alumno:
Gonzales
Corelativas adeudadas:
materiaA,materiaB,materiaC
comentarios: ninguno
Borrar Inscipción