**Electron.io**

José Abraham Torres Juárez

Desarrollo de aplicaciones web,

Tecnológico de Monterrey, Campus Cuernavaca

A01420804@itesm.mx

ABSTRACT

Building desktop applications requires a lot of effort to get it working on the three principal platforms Linux, MacOS and Windows. However, these three OS can run a web browser, without problems. Electron is a framework that takes advantage of this and allow you to create cross-platform applications using HTML, CSS and JavaScript.

# 1. INTRODUCCIÓN

Electron facilita el desarrollo de las aplicaciones de escritorio multiplataforma, es un framework que está respaldado por Node.js y el navegador open-source Chromium. Lo que permite que cada aplicación sin importar la plataforma para la cual fue generada (Windos, MacOS, Linux) se ejecutará de manera perfecta.

Además de esto Electron se puede impulsar empleando algunos paquetes (npm), frameworks o librerías ya desarrolladas para el desarrollo web. Como lo son jQuery, Bootstrap, Angular, React entre muchas otras tecnologías que facilitan y eficientizan el desarrollo web.

**2. CONTENIDO**

Como demostración del funcionamiento principal de Electron, crearemos un editor de código propio. Este editor nos permitirá generar y editar código en distintos lenguajes y además configurar los colores tanto del editor como del código, tamaño y tipo de letra.

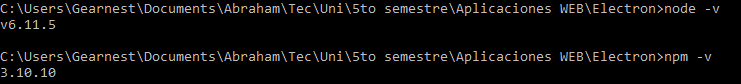
Para esto es necesario tener conocimientos básicos de HTML, CSS, JavaScript, así como saber manejar la consola o terminal.

**2.1 PREPARACIÓN**

Antes que nada, es necesario contar con Node.js y npm, si no contamos con Node instalado, es necesario descargarlo de la siguiente liga:

<https://nodejs.org/es/>

Podemos verificar que la instalación se realizó con éxito a través de la terminal, desde ella ingresamos los siguientes comandos:



El output de los comandos puede variar dependiendo de las versiones que se tengan instaladas, con el primer comando obtenemos la versión de Node instalada y con el segundo la versión de npm.

Npm se incluye dentro de Node y nos permite utilizar código desarrollado por otras personas con el cual podremos agilizar nuestro proceso de desarrollo. Es con esta herramienta con la que instalaremos todas nuestras dependencias en nuestro entorno de desarrollo, entre ellas Electron.

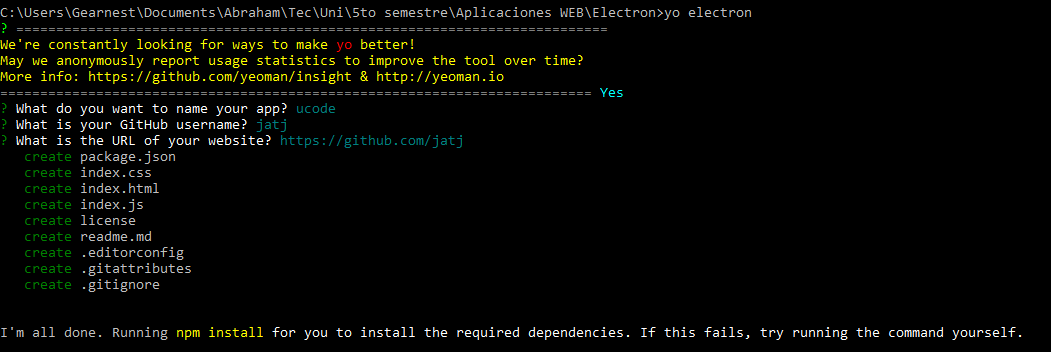
Para facilitar la preparación de Electron utilizaremos el siguiente npm *generator-electron.* Lo instalamos con el siguiente comando:

$ npm install --global yo generator-electron

Después de instalarse, es necesario crear una carpeta vacía en la cual crearemos nuestro proyecto de Electron, una vez dentro de ella desde la terminal ejecutamos el siguiente comando:

$ yo electron

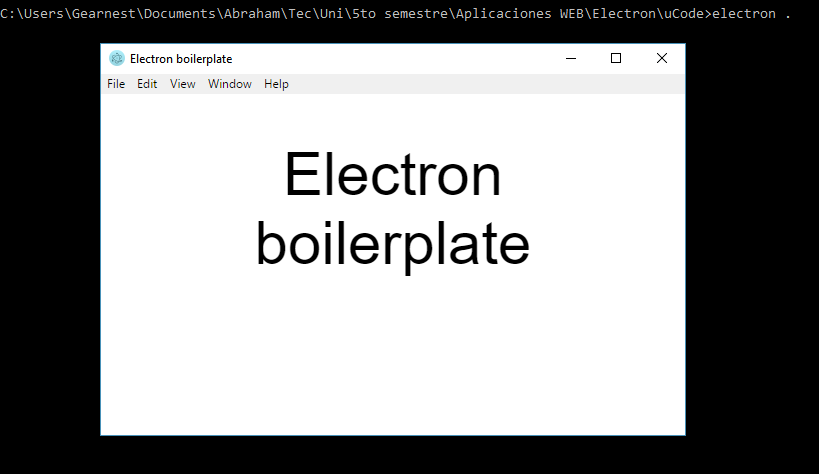
El comando preguntará información relevante sobre el proyecto, generará y configurará los archivos necesarios para correr la aplicación, lo que dará un resultado como el siguiente.



Una vez generada la estructura del proyecto e instalado los npms necesarios para correr la aplicación, podemos verificar el funcionamiento de Electron con el siguiente comando:

$ electron .

Con este comando iniciaremos la aplicación la cual hasta este momento es una página en blanco con un título, se vera de esta manera:

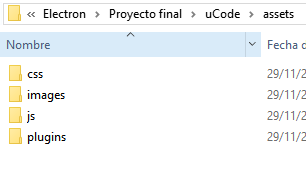


Para facilitar y hacer más rápido el desarrollo de este editor de código, utilizaremos el npm *jquery,* así como el *bootstrap*

$ npm install jquery --save

$ npm install bootstrap --save

Una vez instalados los paquetes necesarios podemos necesitaremos crear un folder para contener los recursos de nuestro proyecto, al cual nombraremos assets, y dentro del creamos las carpetas con los recursos por utilizar en nuestro proyecto:



Para poder enfocarnos en electron utilizaremos un API de un editor de código, el cual ya cuenta con temas y sintaxis de lenguajes, lo que le dará una buena funcionalidad e imagen a nuestro editor, utilizaremos la API Ace (<https://ace.c9.io/>) de editor de código.

Descargaremos la versión pre empaquetada del siguiente link: <https://github.com/ajaxorg/ace-builds/> Una vez descargada agregamos la carpeta *src-noconflict* a nuestro proyecto en la carpeta *plugins* dentro de *assets*, y le cambiamos el nombre a *ace* para evitar el nombre confuso.

**2.2 COMPORTAMIENTO DE ELECTRON**

Cuando se inicia una aplicación de Electron se corre el proceso principal conocido como *main process*, para nuestro proyecto este código es el *index.js* este código lo generamos automáticamente y se encuentra en la raíz del proyecto, su contenido es el siguiente:

*// Es una directiva de ECMAScript 5, la cual indica que todo el código*

*// se ejecutará en modo estricto. Limitando a no poder usar variables*

*// sin declarar, entre otras restricciones.*

*// Mas información: https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Modo\_estricto*

'use strict'**;**

*// Inicializa la aplicacion de electron*

**const** electron **=** require**(**'electron'**);**

**const** app **=** electron**.**app**;**

*// Agrega utilidades como shorcuts para abrir la consola o para recargar la aplicación*

require**(**'electron-debug'**)();**

*// Almacenara la ventana principal del proyecto*

**let** mainWindow**;**

*// Función que se ejecutara cuando la ventana es cerrada*

**function** onClosed**()** **{**

*// Elimina la referencia de la ventana principal, si se tuvieran más ventanas se*

*// necesitaria utilizar un arreglo para almacenarlas*

mainWindow **=** **null;**

**}**

*// Función de retorno que se ejecuta cuando la aplicación esta lista*

**function** createMainWindow**()** **{**

*// Inicializa una nueva ventana con un tamaño de 600 x 400*

**const** win **=** **new** electron**.**BrowserWindow**({**

width**:** **600,**

height**:** **400**

**});**

*// Carga el archivo index.html que se encuentra en la raiz del proyecto dentro de la ventana*

win**.**loadURL**(**`file**:***//${\_\_dirname}/index.html`);*

*// Se enlaza el evento de que la ventana se cerro con la funcion de retorno onClosed*

win**.**on**(**'closed'**,** onClosed**);**

*// Regresa el valor de la ventana*

**return** win**;**

**}**

*// Evento que se ejecuta cuando todas las ventanas de la aplicación son cerradas*

app**.**on**(**'window-all-closed'**,** **()** **=>** **{**

*// En el caso de estar corriendo en MacOS se cierra la aplicación, ya que de lo contrario*

*// la aplicación permaneceria corriendo en background*

**if** **(**process**.**platform **!==** 'darwin'**)** **{**

app**.**quit**();**

**}**

**});**

*// Evento que se llama cuando la aplicación se encuentra activa*

app**.**on**(**'activate'**,** **()** **=>** **{**

*// Si no se ha generado la ventana principal, se crea una nueva*

**if** **(!**mainWindow**)** **{**

mainWindow **=** createMainWindow**();**

**}**

**});**

*// Evento que se llama cuando la aplicación se encuentra lista*

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

*// Se crea una nueva ventana, la ventana principal*

mainWindow **=** createMainWindow**();**

**});**

Este código genera una nueva ventana de 600 x 400 y carga el contenido del archivo *index.html*

**2.3 CONFIGURACIÓN DE APLICIÓN MULTIVENTANA**

Deseamos que nuestro editor de código cuente con múltiples ventanas, para poder estar editando en diferentes escritorios o tener diferentes proyectos abiertos. Para esto es necesario cambiar la estructura de nuestro *index.js.*

Como se puede observar el código anterior solamente cuenta con una variable que almacena la ventana abierta, de tal manera que solo podemos controlar una ventana a la vez, reemplazaremos esta variable por un arreglo en el cual almacenaremos todas las ventanas de la aplicación.

*// Almacenara las ventanas abiertas de la aplicación*

**let** windows **=** **[];**

En el evento *app on ready* en el cual llamamos el método de createWindow, en lugar de darle valor a la ventana, tendremos que anexar la nueva ventana al arreglo de la siguiente manera:

*// Se crea una nueva ventana, la ventana principal*

windows**.**push**(**createWindow**(**windows**.**length**));**

Esto es necesario incluirlo de igual forma en el evento *app on actívate* de la siguiente manera:

app**.**on**(**'activate'**,** **()** **=>** **{**

*// Si no se ha generado la ventana principal, se crea una nueva*

**for(let** i **=** **0;** i **<** windows**.**length**;** i**++){**

**if** **(!**windows**[**i**])** **{**

windows**.**push**(**createWindow**(**windows**.**length**));**

**}**

**}**

**});**

Además, modificaremos la función *onClosed* por lo siguiente:

**function** onClosed**(**index**)** **{**

*// Elimina la referencia de la ventana principal, si se tuvieran más ventanas se*

*// necesitaria utilizar un arreglo para almacenarlas*

windows**[**index**]** **=** **null;**

**}**

Con estos cambios estará lista nuestra aplicación para crear múltiples ventanas y poder controlarlas desde nuestro proceso principal.

**2.3 CONFIGURACIÓN DE VENTANA**

El proyecto generado, crea una ventana de 600 x 400, la cual es muy pequeña para un editor de texto, así que obtendremos el tamaño de la pantalla actual y generaremos un monitor con esas medidas.

Para esto es necesario almacenar las propiedades de la pantalla utilizada, esta información puede obtenerse en cuanto la aplicación esta lista. El cual es un evento que el *index.js* ya maneja. Solo es cuestión de modificarlo de la siguiente manera:

*// Almacenan las propiedades de la pantalla*

**let** screen**;**

**let** display**;**

**let** area**;**

*// Evento que se llama cuando la aplicación se encuentra lista*

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

*// Se obtienen las dimensiones de la pantalla en uso*

screen **=** electron**.**screen**;**

display **=** screen**.**getPrimaryDisplay**();**

area **=** display**.**workArea**;**

*// Se crea una nueva ventana, la ventana principal*

mainWindow **=** createMainWindow**();**

**});**

Al mismo tiempo configuraremos un color de fondo y el icono de nuestra aplicación, para el cual es necesario incluir el módulo *path,* lo hacemos de la siguiente manera:

**var** path **=** require**(**'path'**)**

Con este módulo y con las variables necesarias almacenadas, podemos configurar la creación de nuestras nuevas ventanas en la función *createWindow.*

**const** win **=** **new** electron**.**BrowserWindow**({**

width**:** area**.**width**,**

height**:** area**.**height**,**

backgroundColor**:** '#3EB2BF'**,**

icon**:** path**.**join**(**\_\_dirname**,** 'assets/images/uCode.png'**)**

**});**

De esta manera obtenemos una ventana buen tamaño en la que ya podremos editar. Incluimos el icono personalizado de nuestra aplicación proporcionando la ruta a él, uniendo el directorio raíz de la aplicación con la ubicación de nuestro icono en nuestra carpeta *images.* Además, incluimos un color de fondo para nuestra aplicación

**2.3 PROCESO RENDERER**

Como ya lo mencionamos el archivo *index.js* es nuestro proceso principal o *main.* Este proceso crea nuevas ventanas, estas ventanas estarán ligadas a un archivo que crearemos llamado en este caso *editor.js,* lo colocaremos en el directorio *assets/js*. Este proceso se denomina como *renderer.*

Desde el *renderer* tendremos la interacción con el usuario dentro de cada ventana creada. Es necesario incluir nuestro código editor.js dentro del index.html ya que es el archivo que será cargado por cada ventana, podemos incluirlo agregando el siguiente elemento antes de cerrar la etiqueta de body:

**<script>** require**(**"./assets/js/editor.js"**)</script>**

Una vez incluido podremos tener la interacción con el usuario desde esta ventana.

**2.3 EDITOR**

Para poder utilizar el editor en nuestra ventana, necesitamos añadirla en el *index.html*. De igual forma que incluimos nuestro *editor.js*. Al mismo tiempo añadiremos *Bootstrap* para facilitarnos el desarrollo del editor.

Dentro de la etiqueta head incluimos el css de *bootstrap*

**<link** rel**=**"stylesheet" href**=**"node\_modules/bootstrap/dist/css/bootstrap.min.css"**>**

Después antes de cerrar el body nuestro index.html quedaría de la siguiente manera:

**<script** src**=**"assets/plugins/ace/ace.js" type**=**"text/javascript" charset**=**"utf-8"**></script>**

**<script** src**=**"assets/plugins/ace/ext-modelist.js" type**=**"text/javascript" charset**=**"utf-8"**></script>**

**<script** src**=**"assets/plugins/ace/ext-themelist.js" type**=**"text/javascript" charset**=**"utf-8"**></script>**

**<script>** require**(**"./assets/js/editor.js"**)</script>**

**<script** type**=**"text/javascript" src**=**"node\_modules/bootstrap/dist/js/bootstrap.min.js"**></script>**

Se incluye ace.js que incluye el editor y las extensiones de *ace ext-modelist.js* y *ext-themelist.js*, las cuales contienen la información de las sintaxis de lenguajes y temas de colores que se encuentran en el editor, los usaremos más adelante.

Cada ventana de uCode contará con la posibilidad de tener múltiples pestañas abiertas, con diferentes tipos de sintaxis en cada ventana. Para simular las pestañas dentro de nuestra aplicación utilizaremos los tabs de *Bootstrap.* Por lo que el index.html incluirá el markup necesario dentro de body:

**<div** class**=**"container-fluid"**>**

**<div** class**=**"row"**>**

**<div** class**=**"col-xs-12" style**=**"padding: 0px;"**>**

**<div** class**=**"card"**>**

**<ul** class**=**"nav nav-tabs" role**=**"tablist"**>**

**</ul>**

**<div** class**=**"tab-content"**>**

**</div>**

**</div>**

**</div>**

**</div>**

**</div>**

Además, incluimos algunos estilos en el index.css:

**html,**

**body** **{**

**padding:** **0;**

**margin:** **0;**

**}**

**body** **{**

**font-family:** **-**apple**-**system**,** 'Helvetica Neue'**,** Helvetica**,** **sans-serif;**

**}**

**a{**

**color:**white**;**

**}**

.container-fluid**{**

**min-height:** **100**vh**;**

**background-color:** **#3EB2BF;**

**}**

.editor **{**

**position:** **fixed** *!important***;**

**right:** **0;**

**left:** **0;**

**width:** **100%;**

**height:** calc**(100%** **-** **42px);**

**}**

.delete **{**

**cursor:pointer;**

**}**

.delete**:hover{**

**color:**grey**;**

**}**

Ahora en el *editor.js* añadimos *jQuery* de la siguiente manera:

window**.**$ **=** window**.**jQuery **=** require**(**'jquery'**);**

También añadimos unas constantes en las cuales pondremos la configuración default del editor:

*// Configuración default del editor en lenguaje y tema*

**const** DEFAULT\_MODE **=** "ace/mode/javascript"

**const** DEFAULT\_THEME **=** "ace/theme/monokai"

Después con *jQuery* creamos una función que se ejecutará cuando la página haya cargado, dentro de esta función añadimos toda la configuración del editor, la cual incluye 4 variables globales y 6 funciones que realizarán lo siguiente:

* modeOptions: almacenará la lista de opciones de sintaxis del editor, obtenidas del archivo *ext-modelist.js*
* themeOptions: almacenará la lista de opciones de temas, obtenidas del archivo *ext-themelist.js*
* editors: almacenará los editores o tabs con contenido dentro de la ventana.
* currTheme: almacenará el tema actual de la ventana, el cual tendrán todas las tabs.
* CodeEditorSetup: función que inicializa los modeOptions y themeOptions, además de ejecutar la función addEditor
* AddEditor: función que añade el markup HTML necesario para crear una nueva tab en el *index.html,* dentro de este tab inicializa un editor con el tema actual (currTheme) y con el lenguaje por default (DEFAULT\_MODE);
* GetActiveEditor: función que regresa el valor del editor activo en la ventana, si no hay ningún editor activo regresa nulo
* SetActiveEditor: función que obtiene un índice y pone como activo al editor que se encuentra en ese índice del arreglo editors
* RemoveEditor: función que elimina el editor que está en el índice proporcionado y pone activo el primer editor que se encuentre
* CloseActiveEditor: función que elimina el editor que se encuentra activo

*// Evento cuando la página esta lista*

$**(()=>{**

*// Lista de Opciones de sintaxis*

**var** modeOptions **=** **[];**

*// Lista de opciones de temas*

**var** themeOptions **=** **[];**

*// Lista de editores o tabs en la ventana*

**var** editors **=** **[];**

*// Tema actual*

**var** currTheme **=** DEFAULT\_THEME**;**

*// función que inicializa los modeOptions y themeOptions, además de ejecutar la función addEditor*

**function** codeEditorSetup**(){**

*// ################### Lista de lenguajes ###################*

*// Obtiene todos los modos o lenguajes de sintaxis disponibles*

**var** modelist **=** ace**.**require**(**"ace/ext/modelist"**);**

*// Genera un arreglo de opciones de menu para cambiar la sintaxis*

**for** **(let** i **=** **0;** i **<** modelist**.**modes**.**length**;** i**++)** **{**

**let** modeOption **=** **{**

label**:** modelist**.**modes**[**i**].**caption**,**

click**(){**

**var** editor **=** getActiveEditor**();**

editor**.**editor**.**getSession**().**setMode**(**modelist**.**modes**[**i**].**mode**);**

**}**

**}**

*// Chequea el lenguaje por default*

**if(**modelist**.**modes**[**i**].**mode **==** DEFAULT\_MODE**)**

modeOption**.**checked **=** **true;**

modeOptions**.**push**(**modeOption**);**

**}**

*// ################### Lista de temas ###################*

*// Obtiene todos los temas disponibles*

**var** themelist **=** ace**.**require**(**"ace/ext/themelist"**);**

*// Genera un arreglo de opciones de menu para cambiar el tema*

**for** **(let** i **=** **0;** i **<** themelist**.**themes**.**length**;** i**++)** **{**

**let** themeOption **=** **{**

label**:** themelist**.**themes**[**i**].**caption**,**

type**:** 'radio'**,**

click**:** **function(){**

currTheme **=** themelist**.**themes**[**i**].**theme**;**

**for** **(var** j **=** editors**.**length **-** **1;** j **>=** **0;** j**--)** **{**

editors**[**j**].**editor**.**setTheme**(**themelist**.**themes**[**i**].**theme**);**

**}**

**}**

**}**

*// Chequea el tema por default*

**if(**themelist**.**themes**[**i**].**theme **==** DEFAULT\_THEME**)**

themeOption**.**checked **=** **true;**

themeOptions**.**push**(**themeOption**);**

**}**

addEditor**();**

$**(**"body"**).**on**(**"click"**,** "span.delete"**,function(){**

console**.**log**(**"click en delete"**);**

removeEditor**(**$**(this).**attr**(**"data-index"**));**

**})**

**}**

*// función que añade el markup HTML necesario para crear una nueva tab en el index.html, dentro de este tab inicializa un editor con el tema actual*

*// (currTheme) y con el lenguaje por default (DEFAULT\_MODE)*

**function** addEditor**(){**

*// ################### Desactiva el editor actual ###################*

**var** activeEditor **=** getActiveEditor**();**

**if(**activeEditor **!=** **null){**

$**(**"#liEditor\_"**+**activeEditor**.**index**).**removeClass**(**"active"**);**

$**(**"#tabEditor\_"**+**activeEditor**.**index**).**removeClass**(**"active"**);**

**}**

*// ################### Agrega HTML ###################*

**let** active **=** **(**editors**.**length **==** **0)** **?** ' active' **:** ''**;**

**let** tab **=** `

**<**li id**=**"liEditor\_${editors.length}" role**=**"presentation" **class=**"${active}"**><**a href**=**"#tabEditor\_${editors.length}" aria**-**controls**=**"tabEditor\_${editors.length}" role**=**"tab" data**-**toggle**=**"tab"**><**span **class=**"nombreCodigo"**>**Código #$**{**editors**.**length**}<**/span> <span class="glyphicon glyphicon-remove delete" data-index="${editors.length}"></span></a></li>

`**;**

**let** tabContent **=** `

**<**div role**=**"tabpanel" **class=**"tab-pane ${active}" id**=**"tabEditor\_${editors.length}"**>**

**<**section **class=**"editor" id**=**"editor\_${editors.length}"**><**/section>

**<**/div>

`**;**

$**(**"ul[role='tablist']"**).**append**(**tab**);**

$**(**".tab-content"**).**append**(**tabContent**);**

*// ################### Configuracion de editor ###################*

*// Genera el area para editar el texto en el elemento con el id="main"*

**let** editor **=** ace**.**edit**(**"editor\_"**+**editors**.**length**);**

*// Da el tema por default monokai*

editor**.**setTheme**(**currTheme**);**

*// Da el lenguaje por default JavaScript*

editor**.**getSession**().**setMode**(**DEFAULT\_MODE**);**

*// Oculta el margen de impresion*

editor**.**setShowPrintMargin**(false);**

*// Manda resizar el editor cuando se resizea la ventanana*

$**(**window**).**resize**(()=>{**

editor**.**resize**();**

**});**

editors**.**push**({**

editor**:**editor**,**

index**:**editors**.**length**,**

id**:** "editor\_"**+**editors**.**length

**});**

setActiveEditor**(**editors**[**editors**.**length **-** **1]);**

**}**

*// función que regresa el valor del editor activo en la ventana, si no hay ningún editor activo regresa nulo*

**function** getActiveEditor**(){**

**for** **(var** i **=** editors**.**length **-** **1;** i **>=** **0;** i**--)** **{**

**if(**$**(**"#"**+**editors**[**i**].**id**).**parent**().**hasClass**(**"active"**)){**

**return** editors**[**i**];**

**}**

**}**

**return** **null;**

**}**

*// función que obtiene un índice y pone como activo al editor que se encuentra en ese índice del arreglo editors*

**function** setActiveEditor**(**editor**){**

**if(**editor **==** **null)** **return;**

$**(**"#liEditor\_"**+**editor**.**index**).**addClass**(**"active"**);**

$**(**"#tabEditor\_"**+**editor**.**index**).**addClass**(**"active"**);**

**}**

*// función que elimina el editor que esta en el indice proporcionado y pone activo el primer editor que se encuentre*

**function** removeEditor**(**index**){**

dialog**.**showMessageBox**({**

type**:**"warning"**,**

title**:**"Cerrar archivo"**,**

message**:** "¿Estás seguro que deseas cerrar el archivo?"**,**

buttons**:[**"Aceptar"**,** "Cancelar"**],**

cancelId**:1**

**},(**res**)** **=>** **{**

**if(**res **==** **0){**

$**(** "#liEditor\_"**+**index **).**remove**();**

$**(** "#tabEditor\_"**+**index **).**remove**();**

*// Regresa al primer codigo*

setActiveEditor**(**editors**[0]);**

**}**

**});**

**}**

*// función que elimina el editor que se encuentra activo*

**function** closeActiveEditor**(){**

**var** editor **=** getActiveEditor**();**

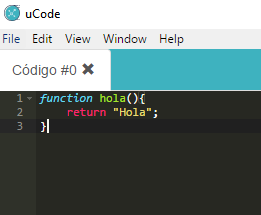
removeEditor**(**editor**.**index**);**

**}**

codeEditorSetup**();**

});

En este punto si corremos nuestra aplicación de electron, veremos lo siguiente:



**2.3 MENU DE HERRAMIENTAS**

Para cambiar el menú que electron trae por default y añadirle más funcionalidad a nuestra aplicación utilizaremos varios módulos los cuales necesitamos incluir al inicio de *editor.js*

*// Carga los modules a utilizar*

**const** **{**remote**}** **=** require**(**'electron'**)**

*// Permiten cambiar el menu*

**const** **{**Menu**,** MenuItem**}** **=** remote

**const** Window **=** require**(**'electron'**).**remote**.**getCurrentWindow**()**

*// Mostrar dialog box*

**const** dialog **=** require**(**'electron'**).**remote**.**dialog

*// Gestor de archivos*

**const** fs **=** require**(**'fs'**);**

Además, hay que crear un template para el menú, el cual contiene todas las opciones y submenús que necesitemos en nuestra aplicación. El template cuenta de objetos con las siguientes posibles propiedades:

* Label: Es el texto que vera el usuario en el menú
* Submenú: Si la opción tiene múltiples ociones se añade este arreglo que contendrá más objetos del menú
* Role: son roles predefinidos por electron, lista de roles: <https://github.com/electron/electron/blob/master/docs/api/menu-item.md#roles>
* Accelerator: son shorcuts declarados en el módulo globalShortcut, se puede sobreescribir su comportamiento, más información: https://github.com/electron/electron/blob/master/docs/api/menu-item.md

Las opciones del menú cuentan con muchas más propiedades, aquí hay más información sobre esas propiedades: <https://github.com/electron/electron/blob/master/docs/api/menu-item.md>. Mencionamos solo las que utilizaremos en nuestra aplicación.

Para agregar el menú customizado, creamos una función menuSetup dentro del mismo listener de jQuery en la cual declaramos el template del menú y se lo asignamos a la ventana.

**function** menuSetup**(){**

*// Template para el menu de la aplicación*

**const** menuTemplate **=** **[**

**{**

label**:** 'Archivo'**,**

id**:** 'archivo'**,**

submenu**:** **[**

**{**

label**:** 'Nuevo archivo'**,**

click**:** addEditor**,**

accelerator**:** 'CommandOrControl+N'

**},**

**{**

label**:** 'Abrir archivo'**,**

click**:** openFile**,**

accelerator**:** 'CommandOrControl+O'

**},**

**{**

label**:** 'Guardar'**,**

click**:** saveFile**,**

accelerator**:** 'CommandOrControl+S'

**},**

**{**

label**:** 'Guardar como'**,**

click**:** saveAsFile**,**

accelerator**:** 'CommandOrControl+Shift+S'

**},**

**{**

type**:** 'separator'

**},**

**{**

label**:** 'Cerrar archivo'**,**

click**:** closeActiveEditor**,**

accelerator**:** 'CommandOrControl+W'

**}**

**]**

**},**

**{**

label**:** 'Editar'**,**

id**:** 'editar'**,**

submenu**:** **[**

**{** label**:** 'Deshacer'**,** role**:** 'undo' **},**

**{** label**:** 'Rehacer'**,** role**:** 'redo' **},**

**{**type**:** 'separator'**},**

**{** label**:** 'Cortar'**,** role**:** 'cut' **},**

**{** label**:** 'Copiar'**,** role**:** 'copy' **},**

**{** label**:** 'Pegar'**,** role**:** 'paste' **},**

**{** label**:** 'Seleccionar todo'**,** role**:** 'selectall' **}**

**]**

**},**

**{**

label**:** 'Ventana'**,**

id**:** "ventana"**,**

submenu**:** **[**

**{**

label**:** 'Nueva ventana'**,**

click**:** **function(){**

ipcRenderer**.**send**(**'newWindow'**,** **1);**

**},**

accelerator**:** 'CommandOrControl+Shift+N'

**},**

**{** label**:** 'Minimizar'**,** role**:** 'minimize' **},**

**{** label**:** 'Cerrar'**,** role**:** 'close'**,** accelerator**:** 'CommandOrControl+Shift+W' **}**

**]**

**},**

**{**

label**:** 'Vista'**,**

id**:** 'vista'**,**

submenu**:** **[**

**{** label**:** 'Pantalla completa'**,** role**:** 'togglefullscreen' **}**

**]**

**},**

**{**

label**:** 'Configuración'**,**

id**:** 'configuracion'**,**

submenu**:** **[**

**{**

label**:** 'Lenguaje'**,**

submenu**:** modeOptions

**},**

**{**

label**:** 'Temas'**,**

submenu**:** themeOptions

**}**

**]**

**},**

**{**

label**:** 'Ayuda'**,**

id**:** 'ayuda'**,**

submenu**:** **[**

**{**

label**:** 'Más información'**,**

click**(){**

remote**.**shell**.**openExternal**(**'https://github.com/jatj/uCode'**);**

**}**

**}**

**]**

**}**

**]**

menu **=** Menu**.**buildFromTemplate**(**menuTemplate**);**

Window**.**setMenu**(**menu**);**

**}**

**2.4 GESTOR DE ARCHIVOS**

Es importante mencionar que el template tiene opciones del menú que hacen referencia a funciones que no hemos incluido aún.

La aplicación necesita abrir, generar y guardar archivos. Para esto en el menú ligamos las opciones de nuevo, abrir y guardar con los siguientes métodos creados:

* OpenFile: Crea un nuevo editor, abre un archivo y pone el contenido del archivo en el editor
* SaveAsFile: Guarda un archivo como
* SaveFile: Guarda archivo si el archivo es nuevo o si el parámetro saveAs es verdadero busca un nuevo path para guardarlo, si el archivo fue abierto y tiene registrado el path

*// Crea un nuevo editor, abre un archivo y pone el contenido del archivo en el editor*

**function** openFile**(){**

dialog**.**showOpenDialog**((**fileNames**)** **=>** **{**

*// fileNames is an array that contains all the selected*

**if(**fileNames **===** **undefined){**

console**.**log**(**"No se selecciono el archivo"**);**

**return;**

**}**

fs**.**readFile**(**fileNames**[0],** 'utf-8'**,** **(**err**,** data**)** **=>** **{**

**if(**err**){**

console**.**log**(**"Error :" **+** err**.**message**);**

**return;**

**}**

addEditor**();**

editors**[**editors**.**length **-** **1].**editor**.**insert**(**data**);**

editors**[**editors**.**length **-** **1].**path **=** fileNames**[0];**

**let** tokens **=** fileNames**[0].**split**(**"\\"**);**

$**(**"#liEditor\_"**+**editors**[**editors**.**length **-** **1].**index **+** " .nombreCodigo"**).**text**(**tokens**[**tokens**.**length **-** **1])**

**});**

**});**

**}**

*// Guarda un archivo como*

**function** saveAsFile**(){**

saveFile**(true);**

**}**

*// Guarda archivo si el archivo es nuevo o si el parametro saveAs es verdadero busca un nuevo path para guardarlo,*

*// si el archivo fue abierto y tiene registrado el path*

**function** saveFile**(**saveAs**){**

**var** editor **=** getActiveEditor**();**

**var** filepath **=** editor**.**path**;**

**var** content **=** editor**.**editor**.**getValue**();**

**if(**filepath **!=** "" **&&** filepath **!=** **null** **&&** filepath **!=** **undefined** **&&** saveAs **!=** **true){**

fs**.**writeFile**(**filepath**,** content**,** **(**err**)** **=>** **{**

**if** **(**err**)** **{**

alert**(**"Error al guardar el archivo" **+** err**.**message**);**

console**.**log**(**err**);**

**return;**

**}**

**});**

**}else{**

dialog**.**showSaveDialog**({**

properties**:** **[**'createDirectory'**,** 'promptToCreate '**]**

**},(**dirPath**)=>{**

**if(**dirPath **==** "" **||** dirPath **==** **null** **||** dirPath **==** **undefined)** **return;**

fs**.**writeFile**(**dirPath**,** content**,** **(**err**)** **=>** **{**

**if** **(**err**)** **{**

alert**(**"Error al guardar el archivo" **+** err**.**message**);**

console**.**log**(**err**);**

**return;**

**}else{**

editor**.**path **=** dirPath**;**

**let** tokens **=** dirPath**.**split**(**"\\"**);**

$**(**"#liEditor\_"**+**editor**.**index **+** " .nombreCodigo"**).**text**(**tokens**[**tokens**.**length **-** **1]);**

**}**

**});**

**})**

**}**

**}**

En estas funciones se utiliza el módulo *fs*  para poder abrir, leer y guardar archivos, si se necesita más información del uso ir a: <https://nodejs.org/api/fs.html>

Además, se incluye la llamada de dialog box, esta funcionalidad se obtiene del modulo *dialog*. Nos permite mostrar mensajes nativos del sistema operativo en el que se corre la aplicación, para más información: <https://github.com/electron/electron/blob/master/docs/api/dialog.md>

**2.3 COMUNICACIÓN ENTRE PROCESOS**

Si corremos nuestra aplicación en este momento podemos crear nuevos editores o tabs y dentro de ellos abrir archivos y guardarlos, el único detalle es que si queremos crear una nueva ventana ya sea desde el menú *ventana* o con el shorcut CommandOrControl+Shift+N, no crea la nueva ventana.

Esto es porque las ventanas se crean en el proceso principal, y necesitamos que de alguna forma el proceso renderer le diga al principal que genere una nueva ventana. Para esto se ocupa IPC, desde el renderer es necesario incluir el siguiente require junto con los demás del *editor.js*

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

Después de agregarlo no es necesario hacer mas ya que la funcionalidad ya fue incluida dentro del template del menú, la cual se encuentra dentro de la declaración del menu *ventana* en su función click:

click**:** **function(){**

ipcRenderer**.**send**(**'newWindow'**,** **1);**

**},**

Esta función envía a través del modulo de ipc al evento *newWindow* un 1, desde el proceso principal nos suscribiremos a este evento de la siguiente manera.

Dentro de *index.js* es necesario incluir al inicio del código lo siguiente:

*// IPC module*

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

Con esto estamos listos para suscribirnos al evento *newWindow*, lo hacemos de la siguiente manera:

*// // Evento que escucha los eventos de los editores de crear nueva ventana*

ipcMain**.**on**(**'newWindow'**,** **(**event**,** msg**)** **=>** **{**

createWindow**(**windows**.**length**);**

**});**

De esta manera comunicamos al proceso main con el renderer, creando una nueva ventana desde el main cada vez que reciba un mensaje al evento *newWindow*. Si se necesita más información sobre IPC:

<https://github.com/mozilla/positron-electron/blob/master/docs/api/ipc-main.md>

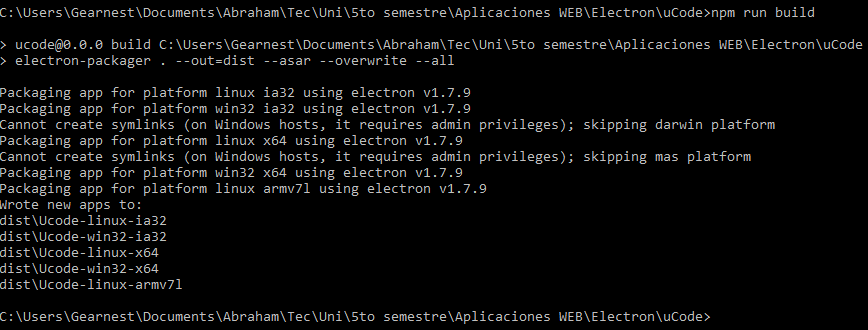
<https://github.com/electron/electron/blob/master/docs/api/ipc-renderer.md>

**2.3 COMPILACIÓN DE ELECTRON**

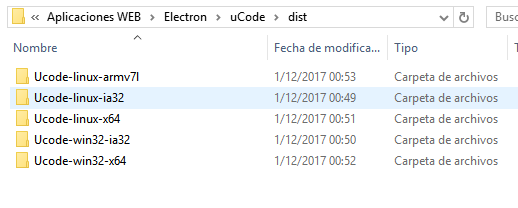
En este punto está terminada nuestra aplicación y podemos generar los ejecutables con nuestra aplicación lista para usar. Para esto ocupamos un módulo llamado *electron-packager*, al usar *yo electron* para generar nuestro proyecto, se auto configuró el comando para compilar nuestra aplicación para Linux y Windows de diferentes arquitecturas, por lo que solo es necesario correr el siguiente comando:

$ npm run build

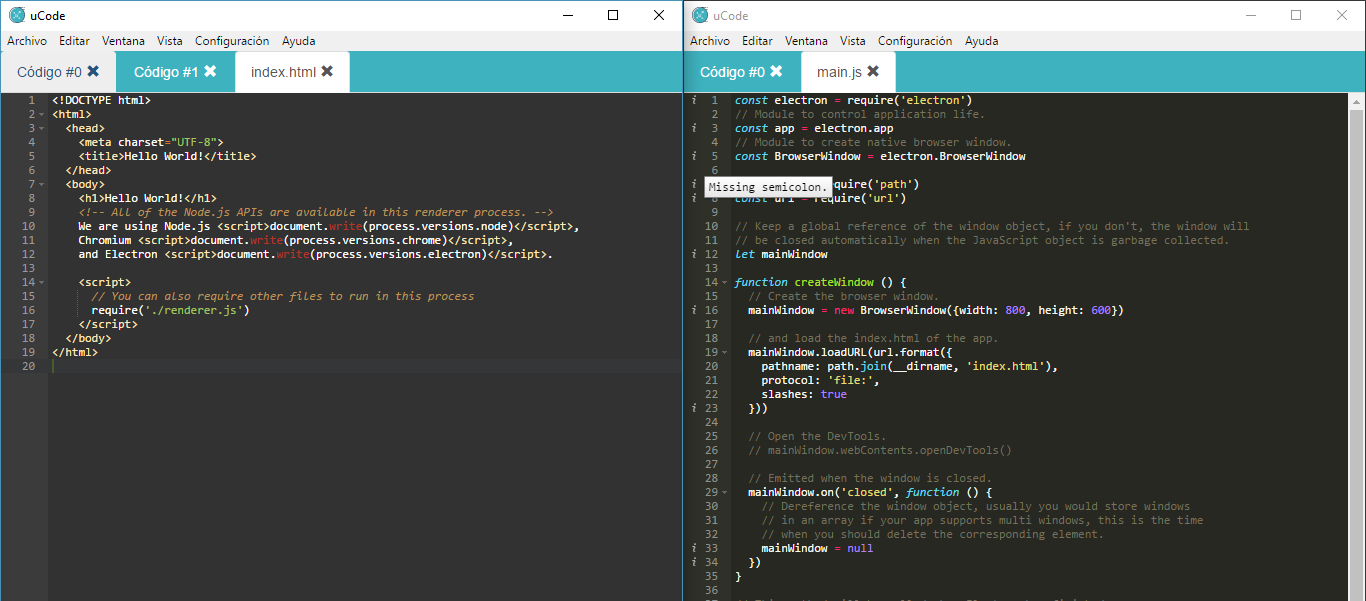
Una vez terminado tendremos un resultado parecido en la consola



Y en nuestro proyecto se habrá generado una nueva carpeta llamada *dist,* la cual contendrá los archivos de nuestras compilaciones de la aplicación, como se muestra en la siguiente imagen:



Una vez compilados podemos ejecutar nuestra aplicación buscando el archivo con extensión .exe y disfrutar de nuestro propio editor de código multilenguaje y multitema.



**3. CONCLUSIÓN**

Como el primer proyecto de Electron que he desarrollado me ha parecido una muy buena tecnología, sobre todo muy bien documentada y activa en los sitios de desarrolladores. Lo que hace más corta la curva de aprendizaje de esta tecnología si ya se tiene un base de programación web (HTML, CSS, JS, Node).

Electron es una tecnología que le abre un segmento enorme al desarrollo web, ya que no debemos limitarnos a verlas aplicaciones desde un navegador, con electron podemos verlas desde nuestros escritorios.

**3. REFERENCIAS**

* *ace.c9.io.* (s.f.). Obtenido de https://ace.c9.io/#nav=api&api=editor
* *electronjs.org.* (s.f.). Obtenido de https://electronjs.org/docs
* Engvall, C. (18 de 10 de 2016). *christianengvall.se.* Obtenido de https://www.christianengvall.se/electron-app-icons/
* *ourcodeworld.com.* (15 de 08 de 2017). Obtenido de https://ourcodeworld.com/articles/read/537/how-to-execute-a-function-of-the-main-process-inside-the-renderer-process-in-electron-framework