Técnico en DESARROLLO DE SOFTWARE

Programación Avanzada

Programación Avanzada

Semana II

1. Formularios

¿Qué son los formularios?

Un formulario es un grupo de elementos que son usados para obtener información de los usuarios, consisten en botones, campos de texto, cajas de opciones y más. Generalmente van acompañados de etiquetas que indican qué tipo de información va asociada a cada campo. A pesar de esto, los formularios no son estrictamente necesarios para obtener información, nos ayudan a organizar los elementos, pero podemos, por ejemplo, colocar un input y un botón sin necesidad de que estén contenidos en un formulario.

Los formularios tienen, generalmente, un botón que tiene un evento especial llamado submit qué nos sirve para enviar los datos del formulario a nuestras aplicaciones.

Un formulario se ve, similar a esto

```
<form id ="formulario">
    <label for="campo1">
        NombreCampo1
    </label>
    <input type="text" id="campo1">

    <input type="submit" value="Haz click" >
</form>
```

Donde tiene varios campos de tipo input, que nos sirven para obtener datos, pueden ser de tipo: text, password, button, entre otros. Puede encontrar una lista más completa aquí.

Los input van acompañados de una label, que nos indica el nombre del campo, por ejemplo, si en un input queremos obtener el nombre, el label tendrá de valor "Nombre"

Al final del formulario tendremos un input de tipo "submit", este funciona como un botón y envía un evento de envió de formulario, para ser usado por servidores, o en nuestro caso la aplicación.

Podemos obtener el evento del formulario de esta forma:

```
var form = document.getElementById('formulario')
form.addEventListener('submit', (event)=>{
    event.preventDefault()
    //más líneas de código
})
```

Este bloque de código hace lo siguiente: Primero obtiene el formulario y lo guarda en una variable, luego coloca un listener al formulario para esperar el evento submit, qué es el evento de envío de formulario.
Luego tenemos una línea que dice event.preventDefault() ¿para qué sirve esto? El comportamiento por defecto del evento submit, envía el evento y luego limpia los campos de un formulario, queremos evitar esto y solamente enviar los eventos cuando estemos preparados.
Nótese que también podríamos colocar un botón con el evento 'click' sin embargo es mejor usar los eventos adecuados, en caso de que el tipo de evento sea relevante para la situación que queremos realizar.

2. Hojas de estilos

Las hojas de estilos CSS tienen una peculiaridad, podemos usar varias hojas de estilos en una sola aplicación o página web.

El nombre CSS significa Cascading Style Sheet, la parte de cascading es la que nos interesa, significa que se aplican las hojas de estilo una después de la otra, entonces podemos usar una hoja de estilo como base, y luego aplicar clases específicas en otra hoja de estilos.

Por ejemplo, podemos ir a páginas web como https://www.free-css.com, y bajar alguna hoja ya prediseñada para no empezar desde 0. Luego crear nuestra propia hoja de estilo y colocarla después de la que descargamos.

De esta forma podemos utilizar varias hojas para una misma aplicación, por ejemplo teniendo una hoja con el estilo general, y otra en la cual usamos campos específicos para la ventana que estamos trabajando.

3. Validaciones de datos

Tan importante como el uso correcto de las funciones y de los elementos son los datos que estamos enviando.
Tenemos que ser capaces de controlar la información que se está enviando e ingresando, por ejemplo, si queremos que un usuario envíe solamente números, entonces tenemos que validar que no esté enviando letras, si queremos que escriba su nombre, tenemos que ver que no contenga números.

Para eso podemos hacer validaciones de datos, estas son muy importantes y la idea es que podamos evitar ingresos de datos no deseados.
Pueden ser tan simples como verificar que un campo no esté vacío o complejas cómo verificar que lo que el usuario ingresó sea un correo electrónico.

Validaciones en los formularios

La validación principal que se puede realizar son las validaciones de campos, en HTML se pueden colocar restricciones sobre los campos que estamos ingresando: cantidad de caracteres, si es obligatorio o no, y finalmente un patrón de datos.

Validación	Descripción
required	*Indica que el campo es obligatorio*
maxlength	*Indica el largo máximo de caracteres que se puede ingresar*
minlength	*Indica el largo mínimo de caracteres que se puede ingresar*
max	*Indica la cantidad máxima que puede valer el campo (usado con números o fechas)*
min	*Indica la cantidad mínima que puede valer el campo (usado con números o fechas)*
pattern	*Indica qué tipos de ingresos acepta este campo*

Estas validaciones se activan automáticamente al tratar de enviar un evento submit y se les coloca una ventana de alerta al usuario indicando qué campos tienen errores y por qué.
Se pueden colocar en un campo de input de la siguiente manera:

```
<input type = "text" id="nombre" required>
<input type = "text" id="usuario" maxlength=10>
```

Expresiones regulares

Las expresiones regulares son patrones, es decir describen una secuencia de caracteres que deben cumplir ciertas condiciones, como que un carácter no puede aparecer más de 1 vez consecutiva. También podemos hacer validaciones más específicas como poder validar un correo electrónico, que tenga varios caracteres, que tenga una @, que esté seguido de otros caracteres, luego un punto (.) y finalmente de dos a tres caracteres.

Expresión	*Significado*
a	*Exactamente 1 vez la letra a*

ab	*Exactamente 1 vez la letra a o la letra b.*
a-z	*Cualquier letra minúscula de la a a la z*
A-Z	*Cualquier letra mayúscula de la A a la Z*
a-zA-Z	*Cualquier letra minúscula o mayúscula desde la a hasta la z*
0-9	*Cualquier número entre el 0 y el 9*
. (punto)	*Cualquier carácter*
\. (diagonal inversa y punto)	*El carácter punto, la diagonal inversa se usa para colocar caracteres especiales que nos interesan, por ejemplo el signo de suma y el asterisco*
a+ (signo de más)	*una expresión seguida de un signo de más indica que la expresión se puede repetir una o más veces*
a (asterisco)*	*una expresión seguida de un asterisco indica que la expresión anterior se puede encontrar 0 o más veces*
a[1] a[1,2] a[1,]	*indica que la expresión se tiene que encontrar exactamente la cantidad de veces indicada.* *Pueden especificarse rangos cerrados o abiertos.* *Un rango cerrado indica la cantidad mínima y máxima que puede aparecer un carácter o grupo de caracteres, estos rangos se separan por coma.* *Un rango abierto indica un mínimo de ocurrencias, más no un máximo.*
?=	*Indica una búsqueda hacia adelante, sin consumir el carácter, es decir vemos los caracteres que están delante de nuestra posición actual*

Una expresión regular funciona en términos de hacer coincidir el patrón con la palabra o la cadena de caracteres, las posiciones y cantidades deben de ser exactas, no podemos fácilmente cambiar el orden o la cantidad de los caracteres que tenemos.
Por ejemplo:

expresión	aceptado	no aceptado
hola|HOLA	*hola, HOLA*	*Hola, hOLA, hOlA*
La expresión busca la palabra hola o la palabra HOLA, notemos el separador | que sirve para indicar la operación lógica 'o'	los únicos dos valores aceptados son la palabra 'hola' todo en minúsculas, o la palabra 'HOLA' todo en mayúsculas	Es la palabra correcta, pero no en la combinación de minúsculas y mayúsculas correcta

expresión	aceptado	no aceptado
a[3]	*aaa*	*aba, aa, AAA,*
La expresión nos dice que tenemos que tener exactamente 3 letras 'a' en secuencia	Son letras 'a', no 'A' y existen 3 exactamente	en 'aba' hay una letra que no encaja, la 'b', en el patrón, si bien hay 3 caracteres, la cantidad de 'a' no es la correcta 'aa' usa las letras requeridas pero no la cantidad adecuada 'AAA' usa la cantidad requerida, y la letra correcta, pero esáa en

		mayúsculas no en minúsculas

expresión	aceptado	no aceptado
[a-z][4,6]	*hola*	*holamundo, hey, HOLA*
La expresión nos dice que podemos colocar cualquier letra minúscula, un mínimo de 4 veces y un máximo de 6	La palabra 'hola' está conformada por 4 letras minúsculas	La frase 'holamundo', si bien son solamente letras minúsculas, tiene 9 caracteres La palabra 'hey' tiene menos caracteres que el mínimo propuesto La palabra 'HOLA' usa letras, pero son mayúsculas

expresión	aceptado	no aceptado
([a-z][A-Z])+	*aZ, aZaZ*	*Za, aaZ, aZZa*
La expresión nos dice que tenemos que tener una letra minúscula, seguida de una letra mayúscula, una o más veces	Una letra minúscula seguida de otra pero mayúscula. Como el patrón nos permite repetir, siempre que se cumpla, una mayúscula y una minúscula, es aceptado	el orden en 'Za' no es el correcto 'aaZ' encaja parcialmente, la primera letra rompe el patrón, pero podemos ver que el patrón se cumple en los caracteres 2 y 3, sin embargo, como el patrón nos pide, una minúscula y

		una mayúscula, no es aceptado 'aZZa' no cumple el orden correcto, ya que la segunda parte está invertida

Existen muchas otras expresiones y reglas que podemos usar, puede visitar páginas como https://regexr.com para descubrirlas

Veamos unas expresiones más complejas, hacer coincidir palabras o cantidad de letras o números es fácil, pero ¿qué pasa si queremos validar patrones más complejos?

Por ejemplo:

¿Cómo validamos un correo electrónico?

Vamos a ver un poco simplificado, un correo electrónico tendrá 3 partes:

- nombre de usuario @ sitio o proveedor. dominio
- Correo simplificado
- ([a-zA-Z0-9]+)\@([a-zA-Z0-9]+)\.([a-zA-Z0-9]){3}

¿Cómo podemos validar una contraseña?

La contraseña tendrá las siguientes características:

- 8 caracteres
- al menos 1 letra minúscula
- al menos 1 letra mayúscula
- al menos 1 número
- al menos 1 carácter especial

Resolvemos esto haciendo uso de la siguiente expresión ?=.* (Signo de interrogación, signo de igual, punto, asterisco) seguida de los grupos de caracteres de los que queremos al menos 1. Esta expresión nos dice que el grupo de caracteres que

especificamos puede aparecer luego de cualquier cantidad de símbolos, y al hacer esto para todos los grupos por separado, estamos diciendo que puede haber una cantidad de símbolos antes de cualquiera de los caracteres que nos interesa.

Podemos usar paréntesis para agrupar o separar pequeños pedazos de las expresiones regulares que nos interesa evaluar por separado.

Luego ponemos la restricción de cantidad, eso lo hacemos colocando los rangos dentro de llaves { } puede ser {8,15} por ejemplo, para tener un máximo de 15 caracteres o algo como {8, } para indicar que puede ser cualquier cantidad de ocho o más caracteres.

contraseña simplificada
(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[^a-zA-Z0-9]).{8,}

Validaciones en JS

Además de poder validar la información en los formularios, también podemos capturar la información en los archivos de JS y validarlos ahí antes de realizar alguna operación.

Recordemos esta función:

```
var form = document.getElementById('formulario')
form.addEventListener('submit', (event)=>{
   event.preventDefault()
   //más líneas de código
})
```

Para un formulario, vamos a obtener el evento de submit, y podemos evitar que envié la información, luego podemos validar nosotros mismos el contenido de los campos. Por ejemplo, si tenemos un formulario así, un simple formulario de usuario y contraseña

```
    <form id="formulario">
        <label for="nombre">Nombre</label>
```

```
        <input type="text" id="nombre" required>
        <label for="pass">Contraseña</label>
        <input type="password" id="pass"  required>
        <div id="error_pass"></div>
        <input type="submit" id="enviar" value="Enviar Datos">
    </form>
```

Podemos usar la función para validar la contraseña.

```
document.getElementById('formulario').addEventListener('submit', function(event){
    event.preventDefault()
    var pass = document.getElementById('pass')

    if(pass.value.length < 8){
        alert('La contraseña debe tener 8 o más carácteres')
        pass.classList.add('invalid')
    }
}
```

O realizar validaciones para mostrar los requerimientos de la expresión regular. Para eso, primero vamos a crear las expresiones regulares que contienen cada uno de los requerimientos. Por ejemplo, que necesitamos una mayúscula y una minúscula.

```
var exprMin = RegExp("[a-z]")
var exprMay = RegExp("[A-Z]")
```

Luego podemos evaluar si el contenido del campo contraseña contiene alguno de estos caracteres.

```
if(!pass.value.match(exprMin)){
        alert('debe contener una minúscula')
    }
```

4. Comunicación entre ventanas y el proceso principal

Las ventanas con las que interactuamos y el proceso o la aplicación principal, son dos cosas aparte, podemos pensar en un navegador, la página web actual es parte del navegador, pero no es el navegador en sí, por ejemplo podemos encontrar las configuraciones y el historial de navegación que no son páginas web.

Aún con los navegadores, pensemos ¿cómo puede un sitio obtener nuestra información? como el usuario de inicio de sesión o las preferencias de idioma. Esto es porque se comunica con el navegador y este le da esa información.
Entonces vamos a ver cómo podemos comunicar las páginas de una aplicación con la aplicación.

Primero tenemos que agregar un archivo, que va a manejar la comunicación. Este archivo lo usamos para evitar colocar todos los módulos en la página, como medida de seguridad. Este archivo se conoce como preload.js.

El archivo va a funcionar como un API, es decir, como una colección de funciones que nos va a permitir interactuar con node.

Vamos a colocar lo siguiente:

```
const [ ipcRenderer, contextBridge ] = require('electron')

contextBridge.exposeInMainWorld(
        'comunicacion',
        {
        registroValido: (datos) => ipcRenderer.send('registroValido', datos)
        }
)
```

ipcRenderer es el objeto que utilizaremos para enviar eventos a la aplicación, puede encontrar la documentación de este en la página de electron. Podemos pensar en él

como un mensajero, que va a llevar mensajes desde la interfaz del usuario, hacia nuestra aplicación. El equivalente que realiza el proceso inverso se llama ipcMain.

contextBridge es el objeto que nos va a permitir realizar la conexión al archivo js sin necesidad de darle acceso a los paquetes de node.

Ahora, el método exposeInMainWorld va a recibir varios parámetros, primero va el nombre con el que se podrá acceder a los métodos que queremos realizar. En este caso se llama 'comunicación' pero puede tener cualquier nombre.

Luego de eso va a recibir una lista de funciones, estas serán todas las funciones que queremos tener en el archivo.

Podemos leer más al respecto en la página de electron sobre contextBridge.

Pero por el momento, solo necesitamos saber lo siguiente:

- la primera palabra es el nombre de la función que vamos a usar
- luego de los dos puntos (:), se encuentra entre paréntesis los argumentos que le vamos a dar a la función
- => nos indica que lo que vamos a hacer es mandar a llamar una función
- ipcRenderer es el objeto que va a enviar mensajes, tiene dos métodos que nos interesan: send es para enviar del renderer al main, y on es para recibir un mensaje del main en el renderer.
- Estas funciones van a tener dos parámetros: el canal de comunicación, y los argumentos. El canal de comunicación, podemos pensar como una dirección, estamos diciendo, vamos a colocar nuestro mensaje en el edificio 1-1 de la zona 1, entonces los que estén atentos a este edificio, pueden recibir el mensaje. Los argumentos son una lista de valores que vamos a enviar a la dirección que colocamos.

Ahora el archivo main vamos a pasar este archivo como parámetro en webPreferences así:

```
function createWindow(){
        const ventana = new BrowserWindow({
        width : 550,
        height : 230,
        webPreferences: {
        preload: path.join(app.getAppPath(),'preload.js')
        },

        })
        ventana.loadFile('index.html')
}
```

Vamos a hacer un pequeño cambio, vamos a obtener una referencia a ventana fuera de la función para que podamos realizar la comunicación más adelante, la función debería quedar así

```
let ventana;
function createWindow(){
        ventana = new BrowserWindow({
        width : 550,
        height : 230,
        webPreferences: {
            preload: path.join(app.getAppPath(),'preload.js')
        },

        })
        ventana.loadFile('index.html')
}
```

También vamos a necesitar agregar los objetos que utilizamos ahí, colocamos estas líneas en la parte de arriba:

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

Nótese que agregamos ipcMain al require que ya teníamos:

Eso nos dice, el archivo index.html puede usar las funciones del archivo preload.js Este preload puede ser reutilizado para otras páginas o podemos hacer uno particular para cada una. Lo veremos más adelante.

Entonces, ahora con un botón, por ejemplo, el de submit de un formulario podemos hacer lo siguiente

```
form.addEventListener('submit', (event)=>{
        event.preventDefault()
        window.comunicacion.registroValido([nombre.value, pass.value]);
})
```

Donde window es el objeto que contiene el preload que acabamos de crear y nombre y pass son los elementos de input del formulario. como recordatorio, se pueden obtener de la siguiente forma

```
var  pass = document.getElementById('pass')
var nombre = document.getElementById('nombre')
```

Esto envía un mensaje, ahora, ¿cómo lo recibimos en la aplicación?
Primero colocamos la siguiente línea al principio del archivo de la aplicación.

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

El ipcMain es el objeto que usaremos en el proceso main, al igual que ipcRenderer, podemos encontrar su documentación en la página de electron

Luego colocamos una función de esta manera:

```
ipcMain.on('evento', (event, args)=>{
    //aquí podemos obtener el evento y los argumentos
    console.log(args)
})
```

La función está escuchando el canal 'evento', si seguimos con la analogía de la dirección, es como si ya estuviéramos esperando en el edificio 1-1 de la Zona 1 y cuando llega alguien con un mensaje, lo podemos recibir.

Esta función va a tener una función como segundo parámetro, esto es para indicar que vamos a hacer cuando recibimos un mensaje.

La segunda función, tiene 2 parámetros: El evento que vino con el mensaje, y los argumentos que nos enviaron, esta segunda función es un listener.

Los argumentos son obvio, los queremos para saber que enviaron, pero ¿para qué queremos el evento? Para saber si fue, por ejemplo, el envió del formulario, si fue un timeout (que se acabó el tiempo para responder) o también para saber qué ventana fue la que nos envió el mensaje si varias ventanas tuvieran el mismo preload.

Esa es una parte de la comunicación, la siguiente sería el proceso inverso, enviar desde la aplicación a una página web. Con el ipcRenderer y el ipcMain ya colocados, vamos a ver, primero en el archivo de la aplicación.

Ahora, podemos hacer algo como esto,

```
ventana.webContents.send('inicio',['param1', 'param2'])
```

webContents, es el objeto que nos permitirá enviar mensajes a un BrowserWindow, luego tiene el método send que funciona igual que el que vimos anteriormente. Un canal de comunicación y argumentos. Si queremos pasar múltiples argumentos, los podemos enviar como un arreglo, es decir, colocarlos dentro de corchetes [] y separados por coma.

Esto lo podemos colocar, por ejemplo, en un receptor de evento, así:

```
ipcMain.on('registroValido',(event, args)=>{
        ventana.webContents.send('error','recibido')
})
```

Podemos notar que este es el evento que envía el formulario cuando el evento es correcto. Lo que estamos haciendo es responder al mensaje recibido.
Vamos a agregar una función al preload que nos permitirá recibir el mensaje, se vería así:

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

contextBridge.exposeInMainWorld(
        'comunicacion',
        {
        registroValido: (datos) => ipcRenderer.send('registroValido', datos)
        ,
        inicioCorrecto: (callback) => ipcRenderer.on('inicioCorrecto', callback)
        }
)
```

Donde callback es una función de callback en la que podemos recibir el mensaje y procesarlo.
Ahora en la página web.Es decir en el archivo renderer o index.js vamos a colocar una función sencilla

```
window.comunicacion.inicioCorrecto(function(event, args){
        alert(args)
```

```
})
```

Ahí vemos la respuesta que nos envió el main.

5. Múltiples ventanas

En general la mayoría de las aplicaciones serán lo que se conoce como aplicaciones de una sola página, pero a veces es importante colocar más de una ventana, si queremos tener por ejemplo una ventana de ingreso de datos o una ventana de inicio de sesión.

Para crear una ventana nueva, basta con realizar otra función igual que la que usamos para crear la ventana principal. En un programa que tenemos dos ventanas, acabaríamos con algo así:

```
let ventana
function crearPrincipal(){
        ventana = new BrowserWindow({
        width : 500,
        height : 500,
        webPreferences: {
           preload: path.join(app.getAppPath(),'preload.js')
        },
    })
    ventana.loadFile('primero.html')
}

let ventana2
function crearSecundario(){
        ventana2 = new BrowserWindow({
        width : 500,
        height : 500,
        webPreferences: {
```

```
        preload: path.join(app.getAppPath(),'preload.js')
      ],
    })
    ventana2.loadFile('segundo.html')
}
```

Cada ventana tendría asociado su propio archivo HTML y su archivo JS.
Con la segunda ventana podemos hacer lo mismo que con la primera, incluso podríamos enviar mensajes desde la primera ventana hacia la segunda ventana.

En el archivo JS de la primera ventana:

```
window.comunicacion.enviarMensaje('hola segunda ventana')
```

En el main:

```
ipcMain.on('mensaje', function(event, args){
      crearSecundario()
      ventana2.webContents.on('did-finish-load', ()=>{
          ventana2.webContents.send('mensaje',args)
       })
})
```

Para enviar un mensaje a la ventana 2 tenemos que: Primero crear la ventana, y luego esperar al evento 'did-finish-load' que nos va a indicar cuando es que terminó de cargar la ventana y está lista para recibir eventos. La función que escucha el evento 'did-finish-load' no requiere argumentos porque no los vamos a usar.
Luego en el archivo JS de la ventana 2:

```
window.comunicacion.recibirMensaje(function(event, args){
    alert(args)
})
```

Entonces, el mensaje lo envía la ventana 1 hacia el main, el main lo recibe y lo retransmite a la ventana 2 y en la ventana 2 mostramos el mensaje que nos envió la ventana 1.

Descargo de responsabilidad

La información contenida en este documento descargable en formato PDF o PPT es un reflejo del material virtual presentado en la versión online del curso. Por lo tanto, su contenido, gráficos, links de consulta, acotaciones y comentarios son responsabilidad exclusiva de su(s) respectivo(s) autor(es) por lo que su contenido no compromete al área de e-Learning del Departamento GES o al programa académico al que pertenece.

El área de e-Learning no asume ninguna responsabilidad por la actualidad, exactitud, obligaciones de derechos de autor, integridad o calidad de los contenidos proporcionados y se aclara que la utilización de este descargable se encuentra limitada de manera expresa para los propósitos educacionales del curso.

Galileo UNIVERSIDAD La Revolución en La Educación

GES Galileo Educational System