Venturi Congress Manager

Manual Técnico

Porras Peralta, Manuel
2ºDAI

Índice

Manual Técnico de la aplicación Venturi Congress Man

Aplicación Web

- **index.php**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
	<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
	<title>Registrarse en el evento</title>
	<script type="text/javascript" src="./lib/js/funciones.js"></script>
	<script type="text/javascript" src="./lib/js/sha1.js"></script>
</head>
<style type="text/css">
  @import './lib/style/estilos.css';
</style>
<body>
<?php
	require_once('./lib/db/db_connect.php.inc');
	include_once('registro.php');
?>
</body>
</html>
```

Este archivo realiza la inclusión de las funciones generales de funciones.js y de las de sha1.js (cifrado). También define el estilo de la aplicación web cargando el css de estilos.css. Incluímos el archivo externo de conexión a la base de datos db_connect.php.inc y finalmente cargamos la página principal de la aplicación (registro.php)

- **registro.php**

```
require_once('./lib/php/recaptchalib.php');
require_once('./lib/php/sendmail.php');
$publickey = "6Lezo84SAAAAAK9O-cgYF9sh-Gc4adwj_Mj2o86e";
$privatekey = "6Lezo84SAAAAAJrxmlCqpKlgypEjt7aMNQuJtiZE";
```

Estas líneas se encargan de incluir dos bibliotecas necesarias para la correcta ejecución y tratamiento del formulario de registro en el evento.

- La primera, recaptchalib.php, se encarga de gestionar el sistema de seguridad Recaptcha de Google para no dejar pasar registros no deseados mediante bots.
- La segunda, sendmail.php, se encarga de gestionar el envío del correo de validación a la dirección de correo electrónico facilitada por el futuro

visitante al evento.

- Las líneas siguientes simplemente son para registrar las claves privada y pública que necesita Recaptcha para conectar con el servidor de Google.

Este archivo se bifurca en la ejecución principal y el tratamiento de los datos enviados por el usuario de la aplicación, que desea registrarse en algún evento ofrecido por el sistema.
Así, mediante

```
if(!isset($_POST['nombre']) || !isset($_POST['apellido1']) || !isset($_POST['apellido2']) || !
isset($_POST['dni']) || !isset($_POST['mail']) || !isset($_POST['evento'])) {
```

si no nos han pasado ningún campo por medio de AJAX, quiere decir que es un registro nuevo, por lo que presentamos al usuario el formulario en blanco:

```
<div id="registro" align="center">
<h2>Reserva de plaza en el evento</h2>
<table align="center">
	<tr>
		<td>Nombre:</td>
		<td><input type="text" id="regNombre" name="regNombre" size="30"
maxlength=40 onkeypress="return filtroTeclado(event,2);"
onblur="putOK('imgNombre','regNombre');"/><img id="imgNombre" class="icono"
src="images/Blank.png" /></td>
	</tr>
	<tr>
		<td>Primer apellido:</td>
		<td><input type="text" id="regApellido1" name="regApellido1" size="30"
maxlength=40 onkeypress="return filtroTeclado(event,2);"
onblur="putOK('imgApellido1','regApellido1');"/><img id="imgApellido1" class="icono"
src="images/Blank.png" /></td>
	</tr>
	<tr>
		<td>Segundo apellido:</td>
		<td><input type="text" id="regApellido2" name="regApellido2" size="30"
maxlength=40 onkeypress="return filtroTeclado(event,2);"
onblur="putOK('imgApellido2','regApellido2');"/><img id="imgApellido2" class="icono"
src="images/Blank.png" /></td>
	</tr>
	<tr>
		<td>Correo electrónico:</td>
		<td><input type="text" id="regMail" name="regMail" size="50"
maxlength=100 onkeypress="return
filtroTeclado(event,5);"onblur="putOK('imgMail','regMail'); "/><img id="imgMail"
class="icono" src="images/Blank.png" /></td>
	</tr>
	<tr>
		<td>DNI:</td>
		<td><input type="text" id="regDNI" name="regDNI" size="10"
maxlength=8 onkeypress="return filtroTeclado(event,7);"
onblur="putOK('imgDNI','regDNI');"/>
```

```
                <input type="text" id="regLetraDNI" name="regLetraDNI" size="2"
maxlength=1 onkeypress="return filtroTeclado(event,6);"
onblur="putOK('imgDNI','regLetraDNI');"/><img id="imgDNI" class="icono"
src="images/Blank.png" /></td>
        </tr>
        <?php if(!isset($_POST['source_event'])) { ?>
        <tr>
                <td>Evento:</td>
                <td><select id="lstEvento" name="lstEvento">
        <?php
                $db=db_connect();
                $select="select IDEvento,Nombre from Evento";
                if($doQuery=mysqli_query($db,$select)) {
                        while($listEventos = mysqli_fetch_assoc($doQuery)) {
                                echo '<option value="'.
$listEventos['IDEvento'].'">'.utf8_encode($listEventos['Nombre']).'</option>';
                        };
                        mysqli_free_result($listEventos);
                        mysqli_close();
                }
                else echo '<option>No se encontraron eventos</option>';
        ?>
        </select></td>
        </tr>
        <?php } else { ?>
        <h3> <?php echo $_POST['source_event']; }?></h3>
</table>
<br>
<div id="captcha"> <?php echo recaptcha_get_html($publickey);?></div>
<br>
<div class="lopd" align="justify">
En conformidad con lo previsto en la <a
href="https://www.agpd.es/portalwebAGPD/canaldocumentacion/legislacion/estatal/common/pdfs/Ley-15_99.pdf">Ley Orgánica 15/1999, del 13 de diciembre, de Protección de Datos de Carácter Personal (LOPD)</a>,
se le informa de la existencia de un registro en la base de datos de su titularidad en el cual se incluirán los datos necesarios para proporcionar los servicios que los usuarios soliciten.<br>
El titular podrá ejercitar los derechos reconocidos en la LOPD sobre este fichero y, en particular, los de acceso, rectificación o cancelación de datos y oposición, si resultara pertinente, así como el de revocación del consentimiento para la cesión de sus datos en los términos previstos en la LOPD. Los usuarios pueden realizar estas acciones enviando una solicitud al
administrador de este sitio web.
</div>
<input id="btnLopd" name="btnLopd" type="checkbox">He leído y acepto la LOPD</input>
<br><div id="mensaje"></div><br>
<button id="btnSend" name="btnSend" type="submit" onmouseover="checkForm();"
onclick="enviaForm();">Enviar</button>
<button name="btnClose" type="button"
```

```
onclick="cerrar('registro','main');limpiaRegistro();">Cancelar</button>
</div>
```

De la sucesión anterior de código HTML mezclado con PHP, cabe destacar la carga de datos de la base de datos en el componente <select> para mostrar los eventos disponibles en los que el usuario puede registrarse. Definimos dos botones para controlar lo que se hace con el formulario, “Enviar”, que se dedica a, como su nombre indica, enviar los datos del formulario para su tratamiento y comprobación posterior, y “Cancelar”, que termina la aplicación y cierra la ventana.
También, si pasamos por encima del botón “Enviar” pero sin pulsarlo, se realiza una comprobación preliminar de los campos del formulario, salvo el Recaptcha, y se marca con una X aquél campo incorrecto o con un símbolo verde de aprobación en caso contrario.
El NIF introducido se comprueba según la fórmula de correspondencia de la letra con la serie de números del mismo.

```
<?php }
else {
if(isset($_POST["recaptcha_challenge_field"]) &&
isset($_POST["recaptcha_response_field"])) {
        $resp=recaptcha_check_answer($privatekey,
                                                $_SERVER["REMOTE_ADDR"],
                                                $_POST["recaptcha_challenge_field"],
                                                $_POST["recaptcha_response_field"]);
```

Este fragmento muestra cómo comprobamos que el Recaptcha recibido era correcto y también se comienza la segunda parte del script registro.php, dedicada a la comprobación y registro efectivo en la base de datos del sistema de los datos del usuario. Si el Recaptcha es inválido, no se continúa y se debe realizar el proceso de registro de nuevo.

```
if ($resp->is_valid)
        {
                include_once('./lib/db/db_connect.php.inc');

                $nombre=utf8_decode($_POST['nombre']);
                $apellido1=utf8_decode($_POST['apellido1']);
                $apellido2=utf8_decode($_POST['apellido2']);
                $mail=$_POST['mail'];
                $dni=$_POST['dni'];
                $evento=$_POST['evento'];
                $md5=md5($dni.$mail.$evento);
```

Una vez que comprobamos que el Recaptcha es válido, declaramos unas variables que albergarán los datos recibidos por AJAX del formulario enviado en la primera parte de registro.php. Las cadenas de caracteres que lo necesitan,

se decodifican a UTF8 con utf8_decode($cadena). Creamos un MD5 con los datos $dni, $mail e ID del evento ($evento).

```
$db=db_connect();
```

Conectamos con la base de datos.

```
$aforo=mysqli_fetch_array(mysqli_query($db,"select count(*) from ListaVisitantes where IDEvento='".$evento."' and Activo=1;"));
$maxaforo=mysqli_fetch_array(mysqli_query($db,"select Aforo from Evento where IDEvento='".$evento."';"));
```

En la primera variable guardamos los visitantes que hay ya registrados y activados en el evento. La segunda recibe el máximo aforo disponible en la ubicación del lugar donde se celebrará el evento al que intenta inscribirse el usuario.

El siguiente fragmento se autodescribe a sí mismo, tras corroborar que hay plazas libres en el evento:

```
if($aforo<$maxaforo){
//Comprobamos si se ha registrado en otro evento previamente, para no volverlo a crear en la tabla de Visitante
$select="select * from Visitante inner join ListaVisitantes on ListaVisitantes.IDEvento=".$evento." where (DNI like '".$dni."' or Email like '".$mail."') and Visitante.IDVisitante=ListaVisitantes.IDVisitante;)";
if(mysqli_num_rows(mysqli_query($db,$select))<1) {
//Comprobamos que no existe el DNI en Visitante ni en ese evento
	$select="select DNI from Visitante inner join ListaVisitantes on ListaVisitantes.IDEvento=".$evento." where DNI like '".$dni."' and Visitante.IDVisitante=ListaVisitantes.IDVisitante;";
	if(mysqli_num_rows(mysqli_query($db,$select))<1) {
//Comprobamos que no existe la dirección de correo en Visitante ni en ese evento
		$select="select Email from Visitante inner join ListaVisitantes on ListaVisitantes.IDEvento=".$evento." where Email like '".$mail."' and Visitante.IDVisitante=ListaVisitantes.IDVisitante;";
		if(mysqli_num_rows(mysqli_query($db,$select))<1) {
//Comprobamos que el visitante no está ya registrado en el evento seleccionado, para ello guardamos el IDVisitante de la tabla Visitante
			$select="select IDVisitante from Visitante where DNI like '".$dni."' or Email like '".$mail."'";
			$id=mysqli_fetch_row(mysqli_query($db,$select));

			$id=$id[0];
//Buscamos el evento que coincida con el Visitante cuya IDVisitante es $id
			$select="select IDEvento,IDVisitante from ListaVisitantes where
```

```
IDEvento like '".$evento."' and IDVisitante like '".$id."'";
                        if(mysqli_num_rows(mysqli_query($db,$select))<1) {
                                if(sendmail($mail,$md5)) {
                                        echo "Se ha registrado correctamente en el evento.
Revise su correo y confirme su asistencia antes de que pasen 24 horas.";
                                        $select="select * from Visitante where DNI like '".
$dni."' or Email like '".$mail."';";
                                        if(mysqli_num_rows(mysqli_query($db,$select))<1) {
//Insertamos al Visitante en la tabla ya que ni su dni ni su correo existen, y
puede haber individuos con el mismo nombre pero son distintas personas
                                        $insertar="insert into
Visitante(Nombre,Apellido1,Apellido2,DNI,Email)          values ('".$nombre."','".
$apellido1."','".$apellido2."','".$dni."','".$mail."')";
                                        mysqli_query($db,$insertar);
                                }
                                $select="select IDVisitante from Visitante where DNI like '".
$dni."'";
                                $id=mysqli_fetch_array(mysqli_query($db,$select));
//Registramos efectivamente al Visitante en el Evento, es decir, lo insertamos
en ListaVisitantes
                                $insertar="insert into ListaVisitantes(IDEvento,IDVisitante)
values ('".$evento."','".$id[0]."')";
                                mysqli_query($db,$insertar);
                                $insertar="insert into
ValidaReserva(IDVisitante,IDEvento,MD5,Fin) values ('".$id[0]."','".$evento."','".
$md5."',NOW()+INTERVAL 1 DAY)";
                                mysqli_query($db,$insertar);
                        }
                        else
                                echo "No se pudo enviar el correo de validación, por lo que no
se ha realizado la reserva correctamente. Inténtelo de nuevo.";
                }
                else echo "Usted está ya registrado en este evento, revise su correo para más
información.";
        }
        else echo "La dirección de correo electrónico proporcionada está ya registrada.
Compruebe sus datos.";
}
else echo "El DNI proporcionado está ya registrado. Compruebe sus datos.";
}
else echo "Está registrado en nuestra base de datos con otro nombre y apellidos. Compruebe
sus datos.";
}
else echo "Lo sentimos pero no puede reservar plaza en el evento, ya está completo.";

mysqli_close($db);
}
else echo "Captcha inválido. Vuelva a intentar el registro.";
}
}
```

```
?>
```

Con cada mysqli_query realizamos consultas del tipo select, insert y update de forma efectiva en PHP. Al finalizar todas las consultas a la base de datos, la cerramos con un mysqli_close($db). Mostramos los mensajes con el resultado de las operaciones ya haya sido éxito o fracaso por medio de echo.
Con mysqli_fetch_array y _assoc, recogemos valores de los resultados de las consultas realizadas, así podemos recoger IDs que nos convengan para realizar búsquedas y hacer comprobaciones necesarias.

- **Validacion.php**

```
$db=db_Connect();
$md5=$_GET['validacion'];
//Busca el md5 recibido en la base de datos
$md5_valido=mysqli_query($db,"select * from ValidaReserva where MD5 like '".
$md5."'");
//Recoge el resultado de la búsqueda y lo guarda en una array
if(!$resultado=mysqli_fetch_array($md5_valido))
        echo "No se ha encontrado la reserva en la base de datos";
```

Conectamos a la base de datos, guardamos el dato recibido, en este caso la cadena MD5, lo buscamos en la base de datos en la tabla ValidaReserva y comparamos el valor de MD5 recibido con el valor almacenado previamente en la base de datos. Si coinciden, se continúa, si no, se muestra un mensaje de error. El siguiente fragmento se explica por sí solo, gracias a los comentarios en el código:

```
//Almacena los campos IDVisitante e IDEvento de la tabla ListaVisitantes en
sendas variables
$visitanteID=$resultado[0];
$eventoID=$resultado[1];
//Busca el ID en la tabla ListaVisitantes que coincide con el IDVisitante y el
IDEvento de la tabla ListaVisitantes
$savedID=mysqli_query($db,"select IDVisitante,IDEvento from ListaVisitantes where
IDVisitante='".$visitanteID."' and IDEvento='".$eventoID."'");
        if(!$resultado=mysqli_fetch_array($savedID))
                echo "No se encuentra la reserva registrada en esta base de datos";
        else {
//Elimina el registro de la tabla ValidarReserva, ya que se ha encontrado y se
procede la validación
                if(mysqli_query($db,"delete from ValidaReserva where MD5 like '".
$md5."';")){
                if(mysqli_query($db,"update ListaVisitantes set Activo='1' where
IDVisitante='".$visitanteID."' and IDEvento='".$eventoID."'"))
                        echo "¡Reserva validada correctamente!";
                else
                        echo "No se pudo activar la reserva";
```

```
                }
                else
                        echo "Ya se realizó la reserva con este código de activación: ".$md5;
        }
    }
    mysqli_close($db);
```

Y cerramos la base de datos.

- **db_connect.php.inc**

```
function db_connect() {
        $res=mysqli_connect('localhost', 'vcmuser', 'vcmuser','vcongressman') or die('Error al conectar con la base de datos');
        return $res;
}
```

Mediante la función mysqli_connect, obtenemos una conexión con la base de datos o bien muestra un error si no lo consigue.

- **estilos.css**

```
.capanegra {
    display: none;
    position: absolute;
    text-align: center;
    top: 0%;
    left: 0%;
    width: 100%;
    height: 100%;
    background-color: black;
    z-index: 1001;
    opacity: .80;
    }
.capablanca {
    display: none;
    position: absolute;
    top: 10%;
    left: 5%;
    width: 90%;
    height: 80%;
    border:         12px solid cadetblue;
    background-color: white;
    z-index: 1002;
    overflow: auto;
}
.lopd {
    overflow: auto;
    border: 2px solid red;
    top: 2%;
    left: 2%;
    width: 80%;
```

```
        height: 20%;
        margin: 10px auto 15px auto;
        background-color: grey;
    }

    .icono {
        border-style: none;
        border: 0;
        width: 16px;
        height: 16px;
    }

    body {
        background-color: brown;
        color: wheat;
    }

    TD {
        text-align: left;
    }
```

Básicamente se diseña la interfaz web para que tenga un mejor look and feel de cara al usuario que desea registrarse en el evento. Se determinan factores como el color de fondo, los márgenes y el tamaño de los iconos, además de un par de capas que pueden tener sentido si se implementa alguna vez la aplicación como ventana emergente dentro del contexto de la página que la invoca, no como un popup normal, sino por capas.

- **funciones.js**

```
//AJAX
function ajax(){
        if (window.XMLHttpRequest)
                return new XMLHttpRequest();
        else if(window.ActiveXObject)
                return new ActiveXObject("Microsoft.XMLHTTP");
        else return false;
}
```

Función clásica de conexión AJAX que crea el objeto que usaremos para las transferencias de datos entre PHP y Javascript.

```
//Caracteres válidos
var letrasMayus = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZÁÉÍÓÚÜ ";
var letrasMinus = letrasMayus.toLowerCase();
var letrasDNI = "TRWAGMYFPDXBNJZSQVHLCKE";
var letrasDNIminus = letrasDNI.toLowerCase();
var numbers = "0123456789";
var guiones = "-_"
```

```
var raros = "'@ªº";
var email =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789@._";
var tildesMayus = "ÀÈÙÒÙÂÊÎÔÛÄËÏÖÇ";
var tildesMinus = tildesMayus.toLowerCase();
var signos = ",.;:¡!¿?";
var beta= "β";
```

Como dice el comentario, se definen unas cadenas que utilizaremos para comprobar que los campos del formulario son válidos. Las utilizaremos en la siguiente función para filtrar lo que el usuario teclea, así evitamos que pueda haber algún campo inválido. Lo que el usuario escriba tendrá un formato válido seguro, pero podrá existir ya en la base de datos por lo que no sería válido en el sentido de válido para su registro en el evento.

```
//Función que filtra el contenido de una cadena
function filtroTeclado( evento, campo ) {
	var keyNum = ( window.event ? evento.keyCode : evento.which );
	if(navigator.userAgent.indexOf("Firefox") != -1)
		keyNum = evento.keyCode;
	var keyChar;
	var chkValid = '';
	switch (campo)
	{
		//Llamada desde login.php
		case 0:
chkValid=letrasMayus+letrasMinus+tildesMayus+tildesMinus+signos+guiones;
					break;
		case 1:
chkValid=letrasMayus+letrasMinus+tildesMayus+tildesMinus+signos+numbers;
					break;
		//Llamada desde registro.php
		case 2:
chkValid=letrasMayus+letrasMinus+tildesMayus+tildesMinus+signos+guiones;
					break;
		case 3:
chkValid=letrasMayus+letrasMinus+tildesMayus+tildesMinus+signos+numbers;
					break;
		case 4:
chkValid=letrasMayus+letrasMinus+tildesMayus+tildesMinus+signos+numbers+guiones+raros+be
ta;
					break;
		case 5: chkValid=email;
					break;
		case 6: chkValid=letrasDNI+letrasDNIminus;
					break;
		case 7: chkValid=numbers;
					break;
	};
```

```
	//chkvalid = chkvalid+"\t";
	if ( keyNum == 13 ) {
		var boton = document.getElementById("btnSend");
		boton.click();
	}

	if ( keyNum == 9 || keyNum == 8 || keyNum == 46 ||
		( keyNum >= 16 && keyNum <= 18 ) ||
		( keyNum >= 37 && keyNum <= 40 ) ) return true;
	else {
		if(navigator.userAgent.indexOf("Firefox") != -1)
			keyNum = evento.which;
		keyChar = String.fromCharCode(keyNum);
		return chkValid.indexOf(keyChar) != -1;
	}
}
```

Con la última comprobación de si el navegador utilizado es Firefox, arreglamos el problema conocido de no poder pulsar la tecla “Tab”.

```
//Abre ventana modal
function abrir(claro,oscuro) {
	document.getElementById(claro).style.display='block';
	document.getElementById(oscuro).style.display='block';
}

//Cierra ventana modal
function cerrar(claro,oscuro) {
	document.getElementById(claro).style.display='none';
	document.getElementById(oscuro).style.display='none';
}
```

Estas dos funciones modifican visualmente la presentación de las capas, pero no las utilizamos en la implementación actual de la aplicación web.

```
//Devuelve el valor en texto del campo
function getValue(campo) {
	return document.getElementById(campo).value;
}
```

Esta función aclara un poco el código y su razón de ser es plenamente ahorrar tecleo a la hora de leer un valor de un campo del formulario. Invocando getValue con el ID del campo en cuestión, bastará de ahora en adelante para obtener dicho valor.

```
//Envía registro
function enviaForm() {
	if(checkForm()) {
		formulario=new ajax();
		challenge=Recaptcha.get_challenge();
```

```
		response=Recaptcha.get_response();
		envio="nombre="+getValue('regNombre')+
					"&apellido1="+getValue('regApellido1')+
					"&apellido2="+getValue('regApellido2')+
					"&mail="+getValue('regMail')+
					"&dni="+getValue('regDNI')+getValue('regLetraDNI')+
					"&evento="+getValue('lstEvento')+
					"&recaptcha_challenge_field="+challenge+
					"&recaptcha_response_field="+response;
		formulario.open("POST","registro.php",true);
		formulario.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
		formulario.send(envio);
		formulario.onreadystatechange = function() {
			var item=document.getElementById('registro');
			item.innerHTML=formulario.responseText+"<br>";

		}
		limpiaRegistro();
		//cerrar('registro','main');
	}
	else alert("Asegúrese de rellenar correctamente todos los campos y sobre todo... ¡Debe leer y
aceptar la Ley Orgánica de Protección de Datos!");
}
```

Esta función se utiliza para enviar los datos del formulario a través de AJAX para ser tratados de nuevo por el script en PHP correspondiente. Es una función clásica de AJAX adaptada a nuestras necesidades, que hace uso de la previamente definida función getValue, dejando el código algo más amigable y menos farragoso. Tras el correcto envío, se limpia el formulario y se espera la respuesta del servidor. Evidentemente si no se tuvo éxito y algún campo quedó sin rellenar, se pide que se rellene antes de realizar el envío efectivo de los datos.

```
//Devuelve si la cadena introducida cumple el tamaño mínimo en cada caso
function esCorrecto(cadena) {
	var numChar=1;
	if(cadena=="regMail") {
		cadena=document.getElementById(cadena).value;
		return checkMail(cadena);
	}
	else if (cadena=="regDNI" || cadena=="regLetraDNI") {
			var dni=document.getElementById("regDNI").value;
			var letra=document.getElementById("regLetraDNI").value.toUpperCase();
			return checkDNI(dni,letra);
	}
	cadena=document.getElementById(cadena).value;
	if(cadena.length<numChar) return false;
	return true;
}
```

El fragmento anterior se autodescribe y no hace falta señalar nada. Si detecta que es una dirección de correo, la comprueba aparte invocando la función checkMail() y si es un DNI/NIF, hace lo propio invocando a checkDNI(dni,letra) y se efectúa un return con el valor devuelto en cada caso.

```
//Comprueba si el DNI coincide con la letra proporcionada y por tanto es un DNI válido
function checkDNI(dni,letra) {
	var letraReal = letrasDNI.charAt(dni % 23);
	if (letra == letraReal) return true;
	else return false;
}

//Comprueba la dirección de correo
function checkMail(mail) {
	expresion=/^[_a-z0-9-]+(.[_a-z0-9-]+)*@[a-z0-9-]+(.[a-z0-9-]+)*(.[a-z]{2,3})$/;
	if(!expresion.exec(mail))	return false;
	else	return true;
}
```

Las funciones anteriores devuelven true o false si las cadenas comprobadas son válidas. La comprobación se realiza en cada caso con el resultado de comparar la cadena proporcionada con la expresión regular que define la validez de una dirección de correo electrónico o si la letra del DNI se corresponde con el carácter extraído de realizar el módulo a la cifra del dni entre 23.

```
//Cambia imagen ok o cross si los datos del campo activo son correctos
function putOK(imagen,texto) {
	imagen=document.getElementById(imagen);
	if(esCorrecto(texto)) {
		imagen.src="./images/ok_icon.png";
		return true;
	}
	else {
		imagen.src="./images/cross-icon.png";
		return false;
	}
}
```

Cambia el icono a OK o X según el campo comprobado esté en un formato correcto o no.

El fragmento siguiente comprueba campo a campo el formulario además del Recaptcha enviado y de si se ha validado el control checkbox del mismo para aceptar la Ley Orgánica de Protección de Datos (LOPD).

```
//Comprueba el formulario completo antes de mandarlo
function checkForm() {
	var valido=true;
	if(!putOK('imgApellido1','regApellido1')) valido=false;
	if(!putOK('imgApellido2','regApellido2')) valido=false;
	if(!putOK('imgNombre','regNombre')) valido=false;
	if(!putOK('imgDNI','regDNI')) valido=false;
	if(!putOK('imgDNI','regLetraDNI')) valido=false;
	if(!putOK('imgMail','regMail')) valido=false;
	if(!document.getElementById("btnLopd").checked) valido=false;
	if(!Recaptcha.get_response()) valido=false;
	return valido;
}
```

Los siguientes fragmentos son las funciones encargadas de limpiar el registro completo e incluso los iconos utilizados. Inicializa completamente el formulario.

```
//Inicializa hoja de registro
function limpiaRegistro() {
	limpiar('regApellido1');
	limpiar('regApellido2');
	limpiar('regNombre');
	limpiar('regDNI');
	limpiar('regLetraDNI');
	limpiar('regMail');
	limpiarImg('imgApellido1');
	limpiarImg('imgNombre');
	limpiarImg('imgApellido2');
	limpiarImg('imgMail');
	limpiarImg('imgDNI');
	Recaptcha.reload();
	document.getElementById('btnLopd').checked = false;
}

//Inicializa los campos del registro
function limpiar(campo) {
	return document.getElementById(campo).value='';
}

//Inicializa las imágenes del registro
function limpiarImg(campo) {
	return document.getElementById(campo).src='images/Blank.png';
}
```

El fragmento restante es una prueba por si se invoca a la aplicación externamente y se le pasa como parámetro el nombre del evento en el que se quiere registrar el usuario. Esto ocurriría en el supuesto de que algún día se integre la aplicación con el Proyecto Integrado de mis compañeros de ASIR. En caso de recibir un nombre de evento, simplemente se elimina el componente <select> (combobox) y se sustituye por el nombre del evento en cuestión.

```
function enviaEvento() {
	formulario=new ajax();
	envio="source_event=Prueba";
	formulario.open("POST","registro.php",true);
	formulario.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
	formulario.send(envio);
	formulario.onreadystatechange = function() {
		var item=document.getElementById('mensaje');
		item.innerHTML=formulario.responseText+"<br>";
	}
}
```

Aplicación de Escritorio

Módulo “vcmd”

- **main.cpp**

```
#include "mysql_connection.h"
#include <cppconn/driver.h>
#include <cppconn/statement.h>
#include <cppconn/exception.h>

#define DBHOST "tcp://127.0.0.1:3306"
#define USER "vcmuser"
#define PASSWORD "vcmuser"
#define DATABASE "vcongressman"

int main()
{
    sql::Driver *driver;
    sql::Connection *connection;
    sql::Statement *statement;

    //Realizamos la conexión al servidor con vcmuser
    driver = get_driver_instance();
    connection = driver->connect(DBHOST, USER, PASSWORD);

    //Seleccionamos la base de datos
    connection->setSchema(DATABASE);

    //DELETE de la reserva de plaza en ListaVisitantes si no se ha
activado en 24h el visitante mediante el enlace enviado a la dirección de
correo electrónico registrada.
    statement = connection->createStatement();
    try {
      statement->executeQuery("DELETE FROM ListaVisitantes WHERE
(IDEvento, IDVisitante) IN (SELECT IDEvento,IDVisitante FROM ValidaReserva
WHERE Fin < NOW());");
    } catch (sql::SQLException &e) {}
    //DELETE del Visitante si no tiene plazas activas reservadas en ningún
Evento.
    statement = connection->createStatement();
    try {
      statement->executeQuery("DELETE FROM Visitante WHERE IDVisitante NOT
IN (SELECT IDVisitante FROM ListaVisitantes);");
    } catch (sql::SQLException &e) {}
    //DELETE del MD5 caducado que posibilitaba la activación del Visitante
en el Evento en ListaVisitantes.
    statement = connection->createStatement();
    try {
      statement->executeQuery("DELETE FROM ValidaReserva WHERE Fin <
NOW();");
    } catch (sql::SQLException &e) {}
    //Borramos los objectos usados
    delete statement;
    delete connection;
    return 0;
}
```

Este archivo al disponer de tanto comentario, hace que sólo deba señalar que las únicas bibliotecas añadidas y utilizadas son las de la biblioteca MySQL Connector/C++ (paquete libmysqlcppconn7-dev en Debian GNU/Linux). Se debe crear una sentencia sql cada vez que se desee realizar una consulta a la base de datos, y esto se logra con statement = connection->*createStatement*();
Se envuelven correctamente con try/catch a la ejecución de la consulta y directamente se pasa como parámetro la consulta en lenguaje SQL que se quiera efectuar.

```
statement->executeQuery("DELETE FROM ValidaReserva WHERE Fin < NOW();");
```

Se reserva memoria para los objectos driver, connection y statement. En el caso de driver, servirá para obtener una instancia del controlador de MySQL utilizado por nuestra aplicación. Con connection se establecerá la conexión y se guardará dicha conexión en el objeto. Así con setSchema, especificamos la tabla que vamos a utilizar y comenzamos con las operaciones de manteminiento a las que se dedicará este diminuto script o programa en forma de servicio residente en /etc/cron.hourly (se ejecutará cada hora). Cuando no tengamos nada más que hacer con los objetos reservados en memoria, los eliminamos o liberamos de la misma con delete *objeto*; donde objeto será el nombre del que deseemos eliminar de memoria. Si todo ha ido bien, devolvemos el tradicional 0 con un return 0;

Huelga mencionar que para una mayor claridad de código se definieron previamente unas constantes preprocesadas con #define, que incluyen los datos de conexión a la base de datos del sistema y en concreto a la de nuestra aplicación.

Módulo “VenturiCongressMan”

- **main.cpp**

```
#include <QApplication>
#include "maincongressman.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainCongressMan w;
    w.show();
    return a.exec();
}
```

De aquí señalar que se incluye QApplication debido a que vamos a definir nuestra aplicación del tipo Qt y vamos a utilizar muchas funciones y componentes de dicha biblioteca gráfica.
Decir también que todos los componentes gráficos se utilizan y configuran en el Qt Designer integrado en el QtCreator. En esta parte independiente del IDE podremos diseñar completamente nuestro interfaz gráfico para posteriormente tratarlo y controlarlo a través de código C++.
En el main.cpp simplemente definimos w como nuestro “formulario” principal, lo mostramos con w.show() y dejamos en ejecución a la aplicación propiamente dicha mediante return a.exec(). La aplicación solo finalizará si se termina la ejecución desde dentro de w bien por pulsar el botón diseñado para cerrar la aplicación, bien por un error en tiempo de ejecución o bien si forzamos el cierre de alguna forma por medio de código C++.

- **dbhelper.h**

```
#ifndef DBHELPER_H
#define DBHELPER_H

#include <QDebug>
#include <tablatextaligncenter.h>

class dbHelper
{
public:
    dbHelper();
    bool connect();
    TablaTextAlignCenter* query(int);
    TablaTextAlignCenter* query(int, int);
    TablaTextAlignCenter* query(int, QStringList);
    TablaTextAlignCenter* query(int, QString, QString);
    TablaTextAlignCenter* doQuery(QString queryString, int op);
    void close();
};

#endif // DBHELPER_H
```

Se observa que se definen las funciones de tipo TablaTextAlignCenter, esto se hace por necesidad ya que las tablas definidas normalmente no dejan cambiar la alineación del contenido ni otros valores, y esto lo solucionamos mediante este método. TablaTextAlignCenter no es más que una clase que hereda directamente de la clase QSqlQueryModel.
Hacemos un override del miembro data de la clase QSqlQueryModel:

```
QVariant TablaTextAlignCenter::data ( const QModelIndex & index, int role
) const
{
    if (role == Qt::TextAlignmentRole)
        return Qt::AlignCenter;
    return QSqlQueryModel::data(index, role);
}
```

Y le decimos que cuando reciba un rol de alineación de texto, devuelva un AlignCenter, con lo que obtenemos la alineación deseada de los valores de la base de datos mostrados en la tabla o el componente que realiza la consulta.

- **dbhelper.cpp**

```
QSqlDatabase db;
bool dbHelper::connect()
{
    QStringList cData;
    QFile config(":/db/dbconf.ini");
    if (config.open(QIODevice::ReadOnly))
    {
       QTextStream content(&config);
       while (!content.atEnd())
       {
           content.readLine(); //saltamos la línea de sección
           cData.append(content.readLine());
       }
       config.close();
       db = QSqlDatabase::addDatabase(cData.at(0));
       db.setHostName(cData.at(1));
       db.setDatabaseName(cData.at(2));
       db.setUserName(cData.at(3));
       db.setPassword(cData.at(4));
       if(db.open())
        return true;
    }
    return false;
}
```

Aquí definimos db de forma global para poder utilizarla al conectar a la base de datos y al cerrarla, ya que son funciones distintas e independientes. La función connect establece la conexión gracias a la lectura de los parámetros de la misma, que se leen del archivo externo configurable dbconf.ini, situado en el directorio de recursos “db” del proyecto. Si se lee el archivo correctamente, se cierra el archivo almacenado en la variable “config” y se procede al intento de conexión. La forma de leer el archivo es mediante la definición de una variable de tipo

QtextStream llamada “content” que no es más que una clase que almacena un flujo de datos del archivo abierto en formato texto, como su nombre indica. Mientras no se llega al final, se sigue leyendo línea a línea y se guarda todo lo leído en una QstringList “cData”, un array de strings clásico pero personalizado de la biblioteca Qt.
Configuramos la base de datos con cada miembro del array cData y ahora sí, procedemos al intento de apertura de la base de datos para utilizarla durante todo el tiempo de ejecución de nuestra aplicación VenturiCongressMan. Devuelve true si lo consigue y false si pasa cualquier otra cosa.
Los fragmentos de código siguientes se encargan de definir las funciones o bueno, la función query con sus subsiguientes sobrecargas para cumplir todas nuestras necesidades. Así pues tendremos hasta 4 tipos de query según necesitemos evaluar más o menos campos en las tablas consultadas.

```
TablaTextAlignCenter* query(int); Para select simples en SQL. Ej:
case 1: queryString = "select IDOrganizador,Nombre from Organizador;";

TablaTextAlignCenter* query(int, int);

Búsquedas más complejas o delete
case 7: queryString = "delete from Evento where
IDEvento="+QString::number(fk)+";";

TablaTextAlignCenter* query(int, QstringList);

Inserts y Updates, además de Selects más complejos que los anteriores, con
búsquedas por más campos, por eso se recibe un array de cadenas.
case 11: queryString = "INSERT INTO Evento
(Nombre,IDOrganizador,IDLugar,IDUbicacion,FechaInicio,FechaFin,Aforo)
VALUES('"+
          what.at(0)+
          "','"+what.at(1)+
          "','"+what.at(2)+
          "','"+what.at(3)+
          "','"+what.at(4)+
          "','"+what.at(5)+
          "','"+what.at(6)+"');";

TablaTextAlignCenter* query(int, QString, Qstring);

TablaTextAlignCenter* dbHelper::query(int op, QString what, QString where)
{
  QString queryString;
  switch (op) {
      case 15: queryString = "select * from Visitante, ListaVisitantes
where DNI like '"+what+
               "' and Visitante.IDVisitante=ListaVisitantes.IDVisitante and
Activo='1' and ListaVisitantes.IDEvento='"+where+
               "';";
               break;
  }
  return doQuery(queryString,op);
}
```

Aquí dejamos la estructura de control switch con vistas a que en un futuro haga falta añadir más consultas del tipo como la que se realiza con el join entre tablas.

```
TablaTextAlignCenter* dbHelper::doQuery(QString queryString, int op){
    QSqlQuery query;
    if(query.exec(queryString)){
        TablaTextAlignCenter *model = new TablaTextAlignCenter;
        model->setQuery(query);
        switch (op){
        case 3:
            model->setHeaderData(0, Qt::Horizontal,
QObject::trUtf8("ID"));
            model->setHeaderData(1, Qt::Horizontal,
QObject::trUtf8("Nombre del Evento"));
            model->setHeaderData(2, Qt::Horizontal, QObject::trUtf8("ID
Organizador"));
            model->setHeaderData(3, Qt::Horizontal, QObject::trUtf8("ID
Lugar"));
            model->setHeaderData(4, Qt::Horizontal,
QObject::trUtf8("Ubicación"));
            model->setHeaderData(5, Qt::Horizontal, QObject::trUtf8("Fecha
Inicio"));
            model->setHeaderData(6, Qt::Horizontal, QObject::trUtf8("Fecha
Fin"));
            model->setHeaderData(7, Qt::Horizontal,
QObject::trUtf8("Aforo"));
            break;
        case 14:
        case 15:
            model->setHeaderData(0, Qt::Horizontal,
QObject::trUtf8("ID"));
            model->setHeaderData(1, Qt::Horizontal,
QObject::trUtf8("Nombre"));
            model->setHeaderData(2, Qt::Horizontal,
QObject::trUtf8("Primer Apellido"));
            model->setHeaderData(3, Qt::Horizontal,
QObject::trUtf8("Segundo Apellido"));
            model->setHeaderData(4, Qt::Horizontal,
QObject::trUtf8("DNI"));
            model->setHeaderData(5, Qt::Horizontal,
QObject::trUtf8("E-Mail"));
            break;
        case 16:
            model->setHeaderData(1,Qt::Horizontal,
QObject::trUtf8("CIF/NIF"));
            model->setHeaderData(2, Qt::Horizontal,
QObject::trUtf8("Organizador"));
            model->setHeaderData(3, Qt::Horizontal,
QObject::trUtf8("Contacto"));
            break;
        case 23:
            model->setHeaderData(1,Qt::Horizontal,
QObject::trUtf8("Lugar"));
            model->setHeaderData(2, Qt::Horizontal,
QObject::trUtf8("Dirección"));
            model->setHeaderData(3, Qt::Horizontal,
QObject::trUtf8("Ciudad"));
```

```
            model->setHeaderData(4, Qt::Horizontal,
QObject::trUtf8("Provincia"));
            break;
        case 24:
            model->setHeaderData(1,Qt::Horizontal,
QObject::trUtf8("Nombre"));
            model->setHeaderData(2, Qt::Horizontal, QObject::trUtf8("Aforo
máximo"));
            break;
        }
        return model;
    }
    else
        qDebug() << "No se pudo ejecutar la consulta en la base de
datos...";
    return NULL;
}
```

Definimos una variable de tipo QSqlQuery, que es la que va a realizar la consulta deseada proporcionada por los query previamente definidos y especificados. Así con simplemente escribir query.exec(queryString), siendo queryString la cadena SQL devuelta por hacer algún query previo y siendo esta query de query.exec una variable del tipo QsqlQuery, definimos un modelo SQL en base a la consulta realizada. Esto se lleva a cabo en la línea “model->setQuery(query);”. El switch se emplea simplemente para definir la estructura visual de la tabla que vamos a mostrar en la pestaña correspondiente desde donde se haya solicitado la consulta a la base de datos. Se devuelve el modelo ya configurado y cargado si todo fue bien, o se muestra en tiempo de compilación un mensaje de error enviado al debugger de QtCreator.

- **maincongressman.cpp**
  Debido a la amplia y suficiente cantidad de comentarios en el código, hay que aclarar "pocos" puntos para facilitar la continuidad de mejoras o expansión de la aplicación.
  Se puede resumir en los siguientes puntos clave:
  - **setupUi():** se encarga de reservar memoria para el interfaz de usuario, para que posteriormente se pueda alterar su configuración en su forma visual y lógica.
  - **InitUI():** es una función creada para establecer el interfaz inicial, modificando visualmente los componentes gráficos previamente definidos con setupUi().
  - **Variable "ui":** se define al crear MainCongressMan y enlaza la forma lógica del componente gráfico del ui definido en la plantilla .ui con su forma visual, para poder alterar sus características y poder reaccionar a los eventos que sucedan en tiempo de ejecución. La forma de hacerlo será generalmente ui->*nombredelcomponente*->*método*(), ya que estamos programando con orientación a objetos, y "ui" es precisamente un objeto de la clase definida MainCongressMan del espacio de nombres Ui (Ui::MainCongressMan).
  - Para controlar los eventos hace falta definir unos "slots" en el archivo de cabecera que conecten el componente gráfico con la acción lógica a realizar cuando efectuemos una acción sobre dicho componente gráfico. Así trabaja Qt y la verdad resulta muy cómodo una vez que te acostumbras, ya que siempre se sigue la misma metodología: Creamos el componente en el QtDesigner, vamos a QtCreator y en el .h definimos el slot correspondiente, por ejemplo void on_cmdCheckin_clicked(); para más tarde ir al archivo .cpp e implementar dicha función de la que ya conocemos su cabecera.

```
//Registra o desregistra la asistencia del visitante que realiza el
check-in
   void MainCongressMan::on_cmdCheckin_clicked()
   {
       QFont f = ui->cmdCheckin->font();
       QSqlQueryModel* model =
   dynamic_cast<QSqlQueryModel*>(ui->tblAsistente->model());
       QSqlQueryModel* eventoModel =
   dynamic_cast<QSqlQueryModel*>(ui->cmbEventos->model());
       QString idA = model->record(0).value(0).toString();
       QString idE =
   eventoModel->record(ui->cmbEventos->currentIndex()).value(0).toStrin
   g();
       QString checked;
       QStringList cambioCheckin,actualizaAsistente;
       QString nombre = ui->txtNombreAsistente->text();
       QString apellido1 = ui->txtApellido1->text();
       QString apellido2 = ui->txtApellido2->text();
       QString dni = ui->txtDNIAsistente->text();

       if(ui->cmdCheckin->isChecked()){
           f.setBold(true);
```

```
            ui->cmdCheckin->setIcon(QIcon(":/icons/dialog-ok-5.png"));
            checked = "1";
        }
        else {
            f.setBold(false);

    ui->cmdCheckin->setIcon(QIcon(":/icons/document-close-2.png"));
            checked = "0";
        }
        cambioCheckin << idE << idA << checked;
        actualizaAsistente << nombre << apellido1 << apellido2 << idE <<
    dni;
        dbConnection->query(18,cambioCheckin);

    ui->tblAsistente->setModel(dbConnection->query(14,actualizaAsistente
    ));
        ui->cmdCheckin->setFont(f);
    }
```

Con esto, cada vez que pulsemos en cmdCheckin, obtendremos una respuesta al evento satisfactoria según nuestras necesidades. Así operamos con todos los elementos gráficos del interfaz diseñado en el archivo maincongressman.ui, que por cierto es una especie de .xml que QtDesigner y QtCreator interpretan correctamente para definir y conformar nuestro interfaz de usuario.

- Aclarar también que con

```
QSqlQueryModel* modelEvento =
dynamic_cast<QSqlQueryModel*>(ui->cmbEventos->model());
   QString evento = modelEvento->record(index).value(0).toString();
```

seleccionamos siempre el índice real del ítem sobre el que se ha pulsado en un combobox, nombrados siempre con las tres letras cmb seguidas del nombre del componente, en este caso cmbEventos (combobox que muestra los eventos almacenados en la base de datos y disponibles para la búsqueda del asistente en dicho evento).

- dynamic_cast nos hace un casting de tipos de objetos, bastante útil a la hora de hacer compatibles objetos que casi lo son, como en este caso, ui->cmbEventos->model(), del tipo QAbstractItemModel y modelEvento, que es un QSqlQueryModel del que podemos obtener el valor en formato QString del evento seleccionado en la lista cmbEventos, en este caso el índice, que trataremos posteriormente como un int (entero), ya que se trata del índice IDEvento de nuestras tablas de la base de datos.
- Los componentes de tablas tienen propiedades y como tales, podemos modificarlas gracias a los métodos inherentes a ellas.

```
ui->tblAsistente->setColumnWidth(1,150);
ui->tblAsistente->setColumnWidth(2,150);
ui->tblAsistente->setColumnWidth(3,150);
ui->tblAsistente->setColumnWidth(4,100);
ui->tblAsistente->setColumnWidth(5,500);
ui->tblAsistente->hideColumn(0);
ui->tblAsistente->hideColumn(6);
ui->tblAsistente->hideColumn(7);
```

setColumnWidth se encarga de establecer el ancho de columna de forma

visual, mientras que hideColumn oculta las columnas que no nos convenga poder ver en tiempo de ejecución, simplemente se ocultarán de la vista del usuario de la aplicación, pero obviamente podremos seguir utilizando los valores de las celdas de esas columnas.

- Para movernos por los listados de los combobox definidos y sincronizar los datos reflejados en ellos con lo que se está pulsando, empleamos

```
realIndex =
busca(temp->record(index).value(2).toInt(),ui->cmbOrganizador->count(),
organizador);
ui->cmbOrganizador->setCurrentIndex(realIndex);
```

Se comprende mejor si declaramos:

```
  QSqlQueryModel* organizador =
dynamic_cast<QSqlQueryModel*>(ui->cmbOrganizador->model());
```

y

```
   QSqlQueryModel* temp =
   dynamic_cast<QSqlQueryModel*>(ui->tblEventos->model());
```

Por tanto, podemos decir que estamos buscando el organizador con ID record(index).value(2).toInt(), mientras que no se llegue al límite de organizadores (ui->cmbOrganizador->count()) en “organizador”, que no es más que un puntero de tipo QSqlQueryModel que apunta al modelo SQL que está definido en cmbOrganizador. Es decir, obtenemos el IDOrganizador real del Organizador correspondiente al Evento seleccionado en la tabla tblEventos

- Con setFocus() establecemos el foco de la aplicación en el componente que invoque setFocus.
- Queda claro con el comentario, pero no sobra si se comenta:

```
//Inicializamos los validadores para los distintos campos de los
formularios
    QString strNIF ="\\d{8,8}[a-zA-Z]{1,1}";
    QString strCIFNIF=strNIF+"|[a-zA-Z]{1,1}\\d{8,8}";
    QString strEmail="^[_a-z0-9-]+(.[_a-z0-9-]+)*@[a-z0-9-]+(.[a-z0-9-]
+)*(.[a-z]{2,3})$";
    QString strNames=QString::fromUtf8("^([À-Ÿà-ÿa-zA-Z-]
{3,40})|([À-Ÿà-ÿa-zA-Z-]{3,20}\\s{1,1}[À-Ÿà-ÿa-zA-Z-]{3,20})$");
    QString strString=QString::fromUtf8("^[À-Ÿà-ÿa-zA-Z0-9 ºª#&()]
{3,100}$");
    QString strOtherNames = QString::fromUtf8("^[À-Ÿà-ÿa-zA-Z ]{3,100}$");
    QString strAddress=QString::fromUtf8("^[À-Ÿà-ÿa-zA-Z0-9
ºª#\\-\\/&()\\\\\,]{3,200}$");
    QString strTelefono="^([+]{1,1}[0-9]{11,11})|([0-9]{9,9})";
```

Se hace precisamente lo que dice el comentario, como se observa, son todo expresiones regulares que nos servirán para controlar lo que se introduce por teclado en los formularios de las pestañas de nuestra aplicación.

- Las funciones doInsert y doUpdate son análogas, pero cada una hace una función determinada, a pesar de que sus definiciones sean prácticamente idénticas en esencia. Una se encarga de hacer Insert de SQL y otra hace Update, ya estemos insertando un nuevo elemento o bien actualizado o editando uno ya existente.

- Se ha intentado en todo momento reutilizar y aprovechar la mayor cantidad de código posible, esto se ve reflejado en el empleo de cláusulas switch para distinguir si se está en una pestaña u otra, ya que hay varias funciones comunes que varían internamente dependiendo de dónde estemos. Como muestra, un botón:

```
//Creamos un MessageBox personalizado para cada información o
confirmación que se necesite dar al usuario
   bool MainCongressMan::customBox(const char* message, int op)
   {
       QMessageBox box;
       QAbstractButton *myYesButton;
       box.setText(trUtf8(message));
       switch (op){
           case 1: //Confirmación
               box.setWindowTitle(trUtf8("Confirmación"));
               myYesButton = box.addButton(trUtf8("Sí"),
   QMessageBox::YesRole);
               box.addButton(trUtf8("No"), QMessageBox::NoRole);
               box.setIcon(QMessageBox::Critical);
               box.exec();
               if(box.clickedButton()==myYesButton)
                   return true;
               break;
           case 2: //Información
               box.setWindowTitle(trUtf8("Información"));
               box.addButton("OK",QMessageBox::AcceptRole);
               box.setIcon(QMessageBox::Information);
               box.exec();
               return true;
               break;
           case 3: //Alerta
               box.setWindowTitle(trUtf8("¡Error!"));
               box.addButton("OK",QMessageBox::AcceptRole);
               box.setIcon(QMessageBox::Critical);
               box.exec();
               return true;
               break;
       }
       return false;
   }
```

  Aquí se ve reflejada la idea, tenemos una función que nos crea una ventana emergente según necesitemos una de Confirmación de la acción, de simple Información o de avisar de algún error crítico. Al hacer click en “Sí” u “OK”, devolvemos true, si no, devolvemos false y no se hace nada.
- Como nota final, decir que QtCreator permite añadir recursos a nuestra aplicación de forma muy sencilla. En el caso que nos ocupa, pudimos disponer de un recurso utilizado por la conexión a la base de datos, esto es, el archivo de configuración dbconfig.ini (no se ve en la compilación final ni ya empaquetado), que contiene todos los valores personalizados para trabajar con nuestra base de datos. También disponemos de unos recursos gráficos, en este caso, los iconos, almacenados e indexados en el recurso images.qrc. Todas las imágenes están bajo licencias libres.