



MANUAL DE PROGRAMADOR

PÁGINA WEB DEL CENTRO COMERCIAL LA VICTORIA

DESCRIPCIÓN BREVE

El presente documento explica el funcionamiento del código de la página web del Centro Comercial La Victoria para que algún programador pueda continuar su mejoramiento.

Administrador

Equipo de Sistemas del Museo Amparo

INDICE

1. INTRODUCCIÓN.....	2
2. MODELO MVC	2
a. Modelo	3
b. Controlador.....	3
c. Vistas	3
3. CODEIGNITER.....	3
4. UBICACIÓN DE LAS CARPETAS DE LA PÁGINA	6
5. ARCHIVOS DE CONFIGURACIÓN	6
6. CARPETA MODELS.....	7
7. CARPETA CONTROLLERS.....	7
8. CARPETA VIEWS.....	8
9. CONTENIDO DE LOS ARCHIVOS DE LAS VISTAS	8
10. FUNCIONES DEL CONTROLADOR	9
11. FUNCIONES DEL MODELO.....	10
12. LIBRERIAS, STYLES Y JAVASCRIPT	11
13. INSERTAR DATOS.....	12
14. ACTUALIZAR DATOS.....	14
15. ELIMINAR DATOS.....	16
16. CONSULTA DE DATOS.....	18
17. MAPA INTERACTIVO	20
18. ALMACENAR IMÁGENES EN LAS CARPETAS DEL PROYECTO	23
19. RUTAS	24
20. Referencias.....	25

1. INTRODUCCIÓN

En este documento se presentara la manera de cómo fue elaborado el proyecto de la página web del Centro Comercial La Victoria (CCV).

Dicho documento explica las funciones que contiene y la manera en que se fue programando la página, para ello es necesario tener conocimientos en lenguaje php, bases de datos SQL o MySQL, también conocer el framework CodeIgniter, además de conocer el modelo MVC (Model View Controller) ya que este último fue el modelo que se ocupa para realizar este proyecto.

Este proyecto fue realizado con la ayuda de las aplicaciones XAMPP y Sublime Text, los cuales usted tendrá que descargar y configurar de la manera adecuada para su equipo.

2. MODELO MVC

El MVC o Modelo-Vista-Controlador es un patrón de arquitectura de software que, utilizando 3 componentes (Vistas, Models y Controladores) separa la lógica de la aplicación de la lógica de la vista en una aplicación. Es una arquitectura importante puesto que se utiliza tanto en componentes gráficos básicos hasta sistemas empresariales; la mayoría de los frameworks modernos utilizan MVC (o alguna adaptación del MVC) para la arquitectura, entre ellos podemos mencionar a Ruby on Rails, Django, AngularJS y muchos otros más. En este pequeño artículo intentamos introducirte a los conceptos del MVC.

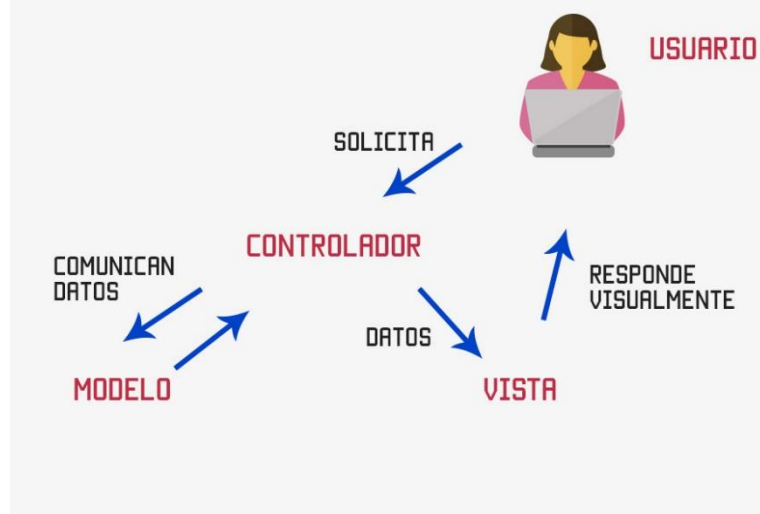


Imagen 2.1 – Diagrama del MVC

a. Modelo

Es la capa donde se trabaja con los datos, por lo tanto, contendrá mecanismos para acceder a la información y también para actualizar su estado. Los datos los tendremos habitualmente en una base de datos, por lo que en los modelos tendremos todas las funciones que accederán a las tablas y harán los correspondientes selects, updates, inserts, etc.

No obstante, cabe mencionar que cuando se trabaja con MCV lo habitual también es utilizar otras librerías como PDO o algún ORM como Doctrine, que nos permiten trabajar con abstracción de bases de datos y persistencia en objetos. Por ello, en vez de usar directamente sentencias SQL, que suelen depender del motor de base de datos con el que se esté trabajando, se utiliza un dialecto de acceso a datos basado en clases y objetos.

b. Controlador

Contiene el código necesario para responder a las acciones que se solicitan en la aplicación, como visualizar un elemento, realizar una compra, una búsqueda de información, etc.

En realidad es una capa que sirve de enlace entre las vistas y los modelos, respondiendo a los mecanismos que puedan requerirse para implementar las necesidades de nuestra aplicación. Sin embargo, su responsabilidad no es manipular directamente datos, ni mostrar ningún tipo de salida, sino servir de enlace entre los modelos y las vistas para implementar las diversas necesidades del desarrollo.

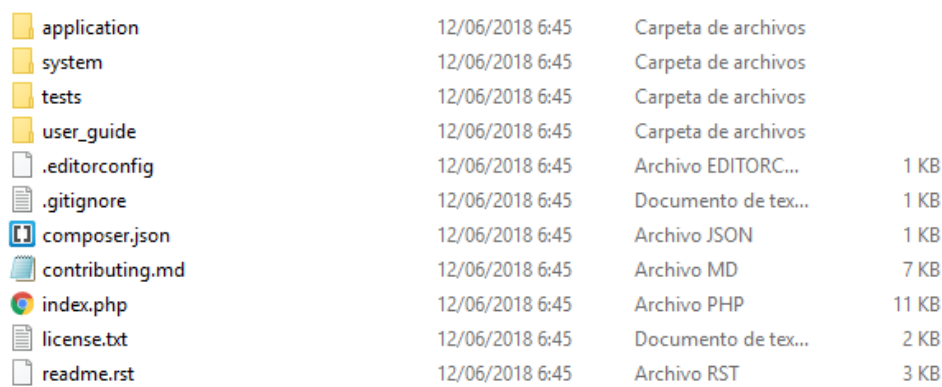
c. Vistas

Las vistas, como su nombre nos hacen entender, contienen el código de nuestra aplicación que va a producir la visualización de las interfaces de usuario, o sea, el código que nos permitirá renderizar los estados de nuestra aplicación en HTML. En las vistas nada más tenemos los códigos HTML y PHP que nos permite mostrar la salida.

En la vista generalmente trabajamos con los datos, sin embargo, no se realiza un acceso directo a éstos. Las vistas requerirán los datos a los modelos y ellas se generarán la salida, tal como nuestra aplicación requiera.

3. CODEIGNITER

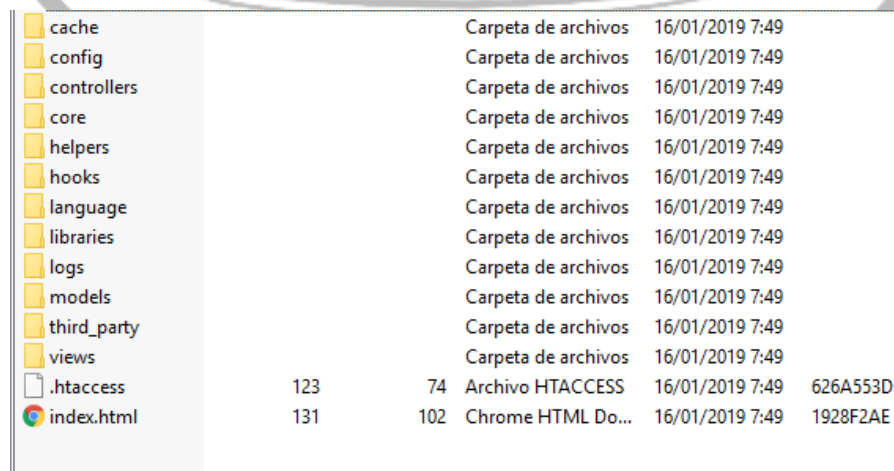
Para trabajar de esta manera también contamos con la ayuda de un framework llamado Codelgniter el cual al descargarlo de su página oficial nos da las carpetas y archivos necesarios para comenzar a realizar el diseño de nuestra web. Dichas carpetas y archivos se pueden ver en la Imagen 3.1 y las cuales las moveremos o cambiaremos el nombre de la carpeta raíz con el nombre de nuestro proyecto que en nuestro caso sería CCVictoria.



application	12/06/2018 6:45	Carpeta de archivos	
system	12/06/2018 6:45	Carpeta de archivos	
tests	12/06/2018 6:45	Carpeta de archivos	
user_guide	12/06/2018 6:45	Carpeta de archivos	
.editorconfig	12/06/2018 6:45	Archivo EDITORC...	1 KB
.gitignore	12/06/2018 6:45	Documento de tex...	1 KB
composer.json	12/06/2018 6:45	Archivo JSON	1 KB
contributing.md	12/06/2018 6:45	Archivo MD	7 KB
index.php	12/06/2018 6:45	Archivo PHP	11 KB
license.txt	12/06/2018 6:45	Documento de tex...	2 KB
readme.rst	12/06/2018 6:45	Archivo RST	3 KB

Imagen 3.1 - Archivos y Carpetas de Codelgniter.

Para encontrar las carpetas “Models”, “Views” y “Controllers” nos meteremos a la carpeta llamada “Application” y ahí encontraremos las carpetas antes mencionadas y la carpeta “Config”. Como podemos ver en la Imagen 3.2.



cache		Carpeta de archivos	16/01/2019 7:49	
config		Carpeta de archivos	16/01/2019 7:49	
controllers		Carpeta de archivos	16/01/2019 7:49	
core		Carpeta de archivos	16/01/2019 7:49	
helpers		Carpeta de archivos	16/01/2019 7:49	
hooks		Carpeta de archivos	16/01/2019 7:49	
language		Carpeta de archivos	16/01/2019 7:49	
libraries		Carpeta de archivos	16/01/2019 7:49	
logs		Carpeta de archivos	16/01/2019 7:49	
models		Carpeta de archivos	16/01/2019 7:49	
third_party		Carpeta de archivos	16/01/2019 7:49	
views		Carpeta de archivos	16/01/2019 7:49	
.htaccess	123	74	Archivo HTACCESS	16/01/2019 7:49 626A553D
index.html	131	102	Chrome HTML Do...	16/01/2019 7:49 1928F2AE

Imagen 3.2 - Ubicación de carpetas models, views, controllers y config.

Dentro de la carpeta de “Config” encontraremos archivos de configuración como se muestra en la Imagen 3.3, de los cuales solo nos enfocaremos en los archivos “config.php” y “database.php”.

autoload.php	4.025	1.120	Archivo PHP	16/01/2019 7:49	EEEE1FEC
config.php	18.436	5.468	Archivo PHP	16/01/2019 7:49	803939C3
constants.php	4.322	1.453	Archivo PHP	16/01/2019 7:49	CC847424
database.php	4.499	1.873	Archivo PHP	16/01/2019 7:49	F800091C
doctypes.php	2.441	549	Archivo PHP	16/01/2019 7:49	0DD3C552
foreign_chars.php	3.046	1.496	Archivo PHP	16/01/2019 7:49	FA4AB93F
hooks.php	417	213	Archivo PHP	16/01/2019 7:49	E73F2F67
index.html	131	102	Chrome HTML Do...	16/01/2019 7:49	1928F2AE
memcached.php	498	260	Archivo PHP	16/01/2019 7:49	65B7C268
migration.php	3.032	943	Archivo PHP	16/01/2019 7:49	2EC65C83
mimes.php	10.233	2.120	Archivo PHP	16/01/2019 7:49	E2165A4B
profiler.php	477	240	Archivo PHP	16/01/2019 7:49	A21DE4A2
routes.php	1.970	837	Archivo PHP	16/01/2019 7:49	E72E8828
smileys.php	3.181	898	Archivo PHP	16/01/2019 7:49	8A81E01A
user_agents.php	6.152	2.200	Archivo PHP	16/01/2019 7:49	02EE45BF

Imagen 3.3 - Archivos de configuración.

En la Imagen 3.4 y 3.5 podemos ver el contenido de los archivos “config.php” y “database.php” respectivamente.

```

1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3
4 /*
5 |-----
6 | Base Site URL
7 |-----
8 |
9 | URL to your CodeIgniter root. Typically this will be your base URL,
10 | WITH a trailing slash:
11 |
12 | http://example.com/
13 |
14 | WARNING: You MUST set this value!
15 |
16 | If it is not set, then CodeIgniter will try guess the protocol and path
17 | your installation, but due to security concerns the hostname will be set
18 | to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.
19 | The auto-detection mechanism exists only for convenience during
20 | development and MUST NOT be used in production!
21 |
22 | If you need to allow multiple domains, remember that this file is still
23 | a PHP script and you can easily do that on your own.
24 |
25 |*/
26 $config['base_url'] = '';
27
28 /*
29 |-----
30 | Index File
31 |-----
32 |
33 | Typically this will be your index.php file, unless you've renamed it to
34 | something else. If you are using mod_rewrite to remove the page set this
35 | variable so that it is blank.
36 |
37 |*/
38 $config['index_page'] = 'index.php';
39
40 /*
41 |-----
42 | URI PROTOCOL
43 |-----
44 |
45 | This item determines which server global should be used to retrieve the
46 | URI string. The default setting of 'REQUEST_URI' works for most servers.
47 | If your links do not seem to work, try one of the other delicious flavors:
48 |
49 | 'REQUEST_URI' Uses $_SERVER['REQUEST_URI']
50 | 'QUERY_STRING' Uses $_SERVER['QUERY_STRING']
51 | 'PATH_INFO' Uses $_SERVER['PATH_INFO']

```

Imagen 3.4 - Contenido del archivo Config.php

```

62 CodeIgniter will store the SQL statement for debugging purposes.
63 However, this may cause high memory usage, especially if you run
64 a lot of SQL queries ... disable this to avoid that problem.
65
66
67 The $active_group variable lets you choose which connection group to
68 make active. By default there is only one group (the 'default' group).
69
70 The $query_builder variable lets you determine whether or not to load
71 the query builder class.
72 */
73 $active_group = 'default';
74 $query_builder = TRUE;
75
76
77 $db['default'] = array(
78     'dsn' => '',
79     'hostname' => 'localhost',
80     'username' => '',
81     'password' => '',
82     'database' => '',
83     'dbdriver' => 'mysqli',
84     'dbprefix' => '',
85     'pconnect' => FALSE,
86     'db_debug' => (ENVIRONMENT !== 'production'),
87     'cache_on' => FALSE,
88     'cachedir' => '',
89     'char_set' => 'utf8',
90     'dbcollat' => 'utf8_general_ci',
91     'swap_pre' => '',
92     'encrypt' => FALSE,
93     'compress' => FALSE,
94     'stricton' => FALSE,
95     'failover' => array(),
96     'save_queries' => TRUE
97 );

```

Imagen 3.5 - Contenido del archivo database.php

Ahora dentro de las carpetas models, views y controllers almacenaremos todos nuestros archivos que ocuparemos y crearemos para nuestro sitio web.

4. UBICACIÓN DE LAS CARPETAS DE LA PÁGINA

Es importante tener un orden en los archivos que se ocupan en el diseño de una página web, esto con el fin de que alguna persona en un futuro requiera realizar nuevos cambios en la página y pueda conocer donde se almacena los archivos que requiera cambiar o modificar.

Como en este caso de la página del CCV se trabajó bajo el modelo MVC, se manejó 3 carpetas llamadas “Models”, “Views” y “Controllers”, otra carpeta más que se maneja es la llamada “Config” donde se encontraran nuestros archivos de configuración como por el ejemplo las variables de la conexión a la base de datos.

Las ubicaciones de nuestras carpetas son las siguientes:

- **Models:** CCVictoria/application/models
- **Views:** CCVictoria/application/views
- **Controllers:** CCVictoria/application/controllers
- **Config:** CCVictoria/application/config
- **Administrador Models:** CCVictoria/administrador/application/models
- **Administrador Views:** CCVictoria/administrador/application/views
- **Administrador** **Controllers:**
CCVictoria/administrador/application/controllers
- **Administrador Config:** CCVictoria/administrador/application/config

5. ARCHIVOS DE CONFIGURACIÓN

Antes de comenzar a realizar cambios en la página web es necesario conocer los archivos de configuración que se manejan como ya se había hablado anteriormente los archivos de configuración son: config.php y database.php los cuales el contenido de dichos archivos de la página se muestran en las Imágenes 5.1 y 5.2. Cabe mencionar que en el archivo config.php se modificó una línea de código de la variable \$config['base_url'] que esta variable la ocuparemos en nuestra página esto para que sea una mejor manera de usar el direccionamiento de los archivos de nuestro sitio web. Mientras que en el archivo database.php colocaremos el valor a nuestras variables para acceder a la base de datos que manejamos dentro de la página en este caso nuestra base de datos se llama “ccvictoria”.

```

1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3
4 /*
5  * Base Site URL
6  * -----
7  *
8  * URL to your CodeIgniter root. Typically this will be your base URL,
9  * WITH a trailing slash:
10  *
11  * http://example.com/
12  *
13  * WARNING: You MUST set this value!
14  *
15  * If it is not set, then CodeIgniter will try guess the protocol and path
16  * your installation, but due to security concerns the hostname will be set
17  * to $_SERVER['SERVER_ADDR'] if available, or localhost otherwise.
18  * The auto-detection mechanism exists only for convenience during
19  * development and MUST NOT be used in production!
20  *
21  * If you need to allow multiple domains, remember that this file is still
22  * a PHP script and you can easily do that on your own.
23  */
24
25 $config['base_url'] = 'http://192.168.1.7/CCVictoria/';
26
27 /*
28  * Index File
29  * -----
30  *
31  * Typically this will be your index.php file, unless you've renamed it to
32  * something else. If you are using mod_rewrite to remove the page set this
33  * variable so that it is blank.
34  */
35 $config['index_page'] = 'index.php';
36
37 /*
38  * URI PROTOCOL
39  * -----
40  *
41  * This item determines which server global should be used to retrieve the
42  * URI string. The default setting of 'REQUEST_URI' works for most servers.
43  * If your links do not seem to work, try one of the other delicious flavors:
44  *
45  * 'REQUEST_URI' - For most servers.
46  * 'query_string' - If your server has a 'query_string' global set, use it.
47  * 'path_info' - For newer servers that have the 'path_info' global set.
48  */

```

Imagen 5.1 - Contenido del archivo config.php de la página del CCV.

```

58 ['failover'] array - A array with 0 or more data for connections if the main should
59 ['save_queries'] TRUE/FALSE - Whether to "save" all executed queries.
60 NOTE: Disabling this will also effectively disable both
61 $this->db->last_query() and profiling of DB queries.
62 When you run a query, with this setting set to TRUE (default),
63 CodeIgniter will store the SQL statement for debugging purposes.
64 However, this may cause high memory usage, especially if you run
65 a lot of SQL queries ... disable this to avoid that problem.
66
67 The $active_group variable lets you choose which connection group to
68 make active. By default there is only one group (the 'default' group).
69
70 The $query_builder variable lets you determine whether or not to load
71 the query builder class.
72
73 $active_group = 'default';
74 $query_builder = TRUE;
75
76 $db['default'] = array(
77     'dsn' => '',
78     'hostname' => 'localhost',
79     'username' => 'root',
80     'password' => '',
81     'database' => 'ccvictoria',
82     'dbdriver' => 'mysql',
83     'dbprefix' => '',
84     'pconnect' => FALSE,
85     'db_debug' => (ENVIRONMENT !== 'production'),
86     'cache_on' => FALSE,
87     'cachedir' => '',
88     'char_set' => 'utf8',
89     'dbcollat' => 'utf8_general_ci',
90     'swap_pre' => '',
91     'encrypt' => FALSE,
92     'compress' => FALSE,
93     'stricton' => FALSE,
94     'failover' => array(),
95     'save_queries' => TRUE
96 );
97

```

Imagen 5.2 - Contenido del archivo database.php de la página del CCV

Una vez ya modificados nuestros archivos de configuración podemos pasar a las comenzar a programar nuestro sitio web.

6. CARPETA MODELS

Dentro de nuestra carpeta de models colocaremos nuestro archivo donde contendrá las funciones que se ejecutaran en la base de datos tales como INSERT, SELECT, UPDATE, DELETE. Dicho archivo lo llamaremos “bases.php” como se muestra en la imagen 6.1, también podemos encontrar el archivo index.html el cual no ocuparemos.

PREHISPANICO-PC > CCVictoria > application > models



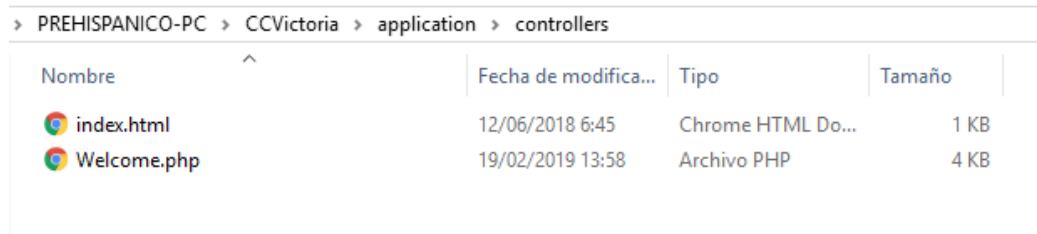
Nombre	Fecha de modifica...	Tipo	Tamaño
 bases.php	18/02/2019 12:58	Archivo PHP	3 KB
 index.html	12/06/2018 6:45	Chrome HTML Do...	1 KB

Imagen 6.1 – Contenido de la carpeta models.

7. CARPETA CONTROLLERS

Dentro de nuestra carpeta de controllers podemos encontrar un archivo llamado Welcome.php como se muestra en la imagen 7.1 donde se encontrara todas nuestras funciones principales y las cuales realizara la comunicación entre el modelo y las vistas.

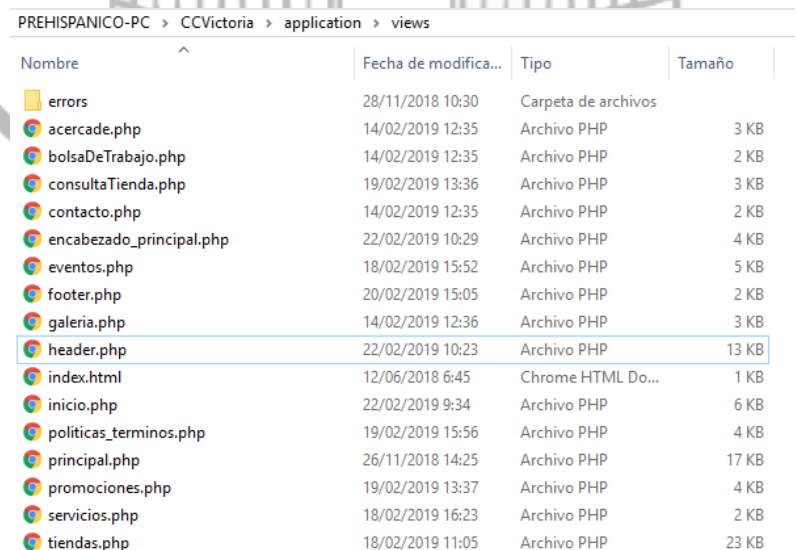


PREHISPANICO-PC > CCVictoria > application > controllers			
Nombre	Fecha de modifica...	Tipo	Tamaño
index.html	12/06/2018 6:45	Chrome HTML Do...	1 KB
Welcome.php	19/02/2019 13:58	Archivo PHP	4 KB

Imagen 7.1 – Contenido de la carpeta Controllers.

8. CARPETA VIEWS

Dentro de nuestra carpeta de views encontraremos todos los archivos de las vistas de la aplicación como se muestra en la imagen 8.1 que son todo el contenido que se muestra de la aplicación.



PREHISPANICO-PC > CCVictoria > application > views			
Nombre	Fecha de modifica...	Tipo	Tamaño
errors	28/11/2018 10:30	Carpeta de archivos	
acercade.php	14/02/2019 12:35	Archivo PHP	3 KB
bolsaDeTrabajo.php	14/02/2019 12:35	Archivo PHP	2 KB
consultaTienda.php	19/02/2019 13:36	Archivo PHP	3 KB
contacto.php	14/02/2019 12:35	Archivo PHP	2 KB
encabezado_principal.php	22/02/2019 10:29	Archivo PHP	4 KB
eventos.php	18/02/2019 15:52	Archivo PHP	5 KB
footer.php	20/02/2019 15:05	Archivo PHP	2 KB
galeria.php	14/02/2019 12:36	Archivo PHP	3 KB
header.php	22/02/2019 10:23	Archivo PHP	13 KB
index.html	12/06/2018 6:45	Chrome HTML Do...	1 KB
inicio.php	22/02/2019 9:34	Archivo PHP	6 KB
politicas_terminos.php	19/02/2019 15:56	Archivo PHP	4 KB
principal.php	26/11/2018 14:25	Archivo PHP	17 KB
promociones.php	19/02/2019 13:37	Archivo PHP	4 KB
servicios.php	18/02/2019 16:23	Archivo PHP	2 KB
tiendas.php	18/02/2019 11:05	Archivo PHP	23 KB

Imagen 8.1 – Contenido de la carpeta Views.

9. CONTENIDO DE LOS ARCHIVOS DE LAS VISTAS

Dentro de los archivos de las vistas encontraras toda la parte del diseño de nuestra aplicación, es decir, como el usuario podrá ver la aplicación desde un navegador web. Como podemos ver en la imagen 9.1 en el archivo se puede ver todo el código principalmente HTML.

```
1 <head>
2 <title>CENTRO COMERCIAL LA VICTORIA</title>
3 </head>
4
5 <div id="myCarousel" class="carousel slide" data-ride="carousel">
6   <!-- Indicators -->
7   <ol class="carousel-indicators">
8     <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
9     <li data-target="#myCarousel" data-slide-to="1"></li>
10  </ol>
11
12  <!-- Wrapper for slides -->
13  <div class="carousel-inner" role="listbox">
14    <div class="item active">
15      
17      <div class="carousel-caption">
18        <h3><a href="<?php echo base_url();?>index.php/Welcom/evento">Más Eventos</a></h3>
19      </div>
20    </div>
21
22    <div class="item">
23      
25      <div class="carousel-caption">
26        <h3><a href="<?php echo base_url();?>index.php/Welcom/evento">Más Eventos</a></h3>
27      </div>
28    </div>
29  </div>
```

Imagen 9.1 – Contenido del archivo inicio.php de las vistas.

10. FUNCIONES DEL CONTROLADOR

En el controlador se encontraran funciones que harán un llamado a las vistas como se muestra en la imagen 10.1, para que el usuario pueda verlas desde un navegador web.

```
10 /* FUNCIONES PARA ABRIR LAS PAGINAS */
11 public function index()
12 {
13     if (isset($_SESSION['useradmi'])) {
14         redirect('/Welcom/panel_administrador', 'location');
15     }
16     else{
17         $this->load->view('header');
18         $this->load->view('encabezado_principal');
19         $this->load->view('login');
20         $this->load->view('footer');
21     }
22 }
23
24
```

Imagen 10.1 – Ejemplo de la función index.

También se encuentran funciones donde reciben los datos de los formularios de la aplicación como se muestra en la imagen 10.2 y en esas mismas funciones se encuentra el llamado al modelo.

```
161     public function validaUsuario(){
162         $user = $this->input->post('user',TRUE);
163         $pass = $this->input->post('clave',TRUE);
164         $data = array('user' => $user, 'clave' => $pass);
165         $data['user'] = $this->bases->validaUsuario($data);
166
167         if ($data['user'] == FALSE) {
168             redirect('/Welcome/index','location');
169         }
170         else{
171             session_start();
172             $_SESSION['useradmi'] = $user;
173             $_SESSION['pass'] = $pass;
174             redirect('/Welcome/panel_administrador','location');
175         }
176     }
177 }
178
```

Imagen 10.2 – Ejemplo de una función recibiendo valores de un formulario de las vistas.

11. FUNCIONES DEL MODELO

Como ya se mencionó anteriormente el archivo del modelo contiene funciones que se refieren a las bases de datos como lo son insertar, actualizar, eliminar y consultar y el contenido se puede observar en la imagen 11.1, todas estas funciones son llamadas en el controlador.

```
bases.php
1 <?php
2 defined('BASEPATH') OR exit('No direct script access allowed');
3 class bases extends CI_Model {
4
5     function __construct()
6     {parent::__construct();}
7
8
9     /* FUNCIÓN PARA VALIDAR EL USUARIO */
10    public function validaUsuario($data){
11        $sql = 'SELECT User, pass FROM usuario WHERE User="'.$data['user'].'" AND pass="'.$data['clave'].'"';
12        $query = $this->db->query($sql);
13
14        if($query->num_rows() > 0)
15            return $query;
16        else
17            return FALSE;
18    }
19
20    /* FUNCIONES PARA INGRESAR DATOS A LA BASE DE DATOS (PARTE DEL MODELO) */
21    public function ingresaTienda($data){
22        $sql = 'INSERT INTO tienda (idTienda, Nombre, Descripcion, Telefono, Horario, Imagen, No_Local, Pagina,
23        data['tel'], "'.$data['horario'].'", "'.$data['vimagen'].'", "'.$data['local'].'", "'.$data['Page
24        $query = $this->db->query($sql);
25    }
26
27    public function ingresaPromo($data){
28        $sql = 'INSERT INTO promocion (idPromocion, NombreP, Descripcion, Imagen, Vigencia, Tienda_idTienda) V
29        vimagen'], "'.$data['vigencia'].'", "'.$data['Tname'].'"';
30        $query = $this->db->query($sql);
31    }
32
33    public function ingresaEvento($data){
```

Imagen 11.1 – Contenido del archivo bases.php

Dentro del archivo bases.php, se encuentran nuestras funciones para las bases como se muestra en la imagen 11.2, que realiza una consulta de los registros de una de nuestras tablas de la base de datos.

```
176
177 public function getEventos(){
178     $sql = 'SELECT * FROM evento ORDER BY Nombre';
179     $query = $this->db->query($sql);
180
181     if($query->num_rows() > 0)
182         return $query;
183     else
184         return FALSE;
185 }
186
```

Imagen 11.2 – Función para consultar eventos.

12. LIBRERIAS, STYLES Y JAVASCRIPT

En nuestra aplicación como en cualquier aplicación manejamos librerías las cuales se encuentran especificadas en el código de la aplicación (véase Imagen 12.1) dentro del archivo header.php que está localizado en la carpeta de views. En ese mismo archivo se encuentra el código de los estilos para la página (véase Imagen 12.2) y también los javascripts (véase Imagen 12.3).

```
<meta charset="utf-8">
<link rel="icon" href="<?php echo base_url();>img/Logo CCV.png">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
<link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet" type="text/css">
<link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet" type="text/css">
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.0/css/all.css" integrity="sha384-1Zn37f5QgtY3VHgisS14W3ExzMWZxybE15J5SEsOp95+oqd12jhcu+A56Ebc1zFSJ"
crossorigin="anonymous">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.0/js/bootstrap.min.js"></script>
<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FggCb/K3Q1LNF0u91ta32o/1MZX1twRo80tmkMRdAu8="crossorigin="anonymous"></script>
```

Imagen 12.1 – Librerías de la página.

```
<style>
body {
    font: 400 15px Lato, sans-serif;
    line-height: 1.8;
    color: #818181;
}
h2 {
    font-size: 24px;
    text-transform: uppercase;
    color: #303030;
    font-weight: 600;
    margin-bottom: 30px;
}
h4 {
    font-size: 19px;
    line-height: 1.375em;
    color: #303030;
    font-weight: 400;
    margin-bottom: 30px;
}
h6 {
    font-size: 15px;
    line-height: 1.175em;
    color: #303030;
    font-weight: 400;
    margin-bottom: 30px;
}
```

Imagen 12.2 – Estilos que maneja la página web.

```
<script>
$(document).ready(function(){
    // Add smooth scrolling to all links in navbar + footer link
    $(".navbar a, footer a[href='#myPage']").on('click', function(event) {
        // Make sure this.hash has a value before overriding default behavior
        if (this.hash !== "") {
            // Prevent default anchor click behavior
            event.preventDefault();

            // Store hash
            var hash = this.hash;

            // Using jQuery's animate() method to add smooth page scroll
            // The optional number (900) specifies the number of milliseconds it takes to scroll to the specified area
            $('html, body').animate({
                scrollTop: $(hash).offset().top
            }, 900, function(){

                // Add hash (#) to URL when done scrolling (default click behavior)
                window.location.hash = hash;
            });
        } // End if
    });
});
```

Imagen 12.3 – JavaScript que maneja la página web.

13. INSERTAR DATOS

En esta sección hablaremos de cómo trabaja la parte de insertar datos de la aplicación a la base de datos y cuál es el procedimiento que se realiza en el código.

El procedimiento que se lleva a cabo es el siguiente:

Primero se debe llenar el formulario que se le muestra al usuario por lo que el código se puede ver en la imagen 13.1 que es el formulario para los comentarios, este formulario lo podemos encontrar en el archivo contacto.php dentro de la carpeta de las vistas.

```
15 <form method="POST" action="<?php echo base_url(); ?>index.php/Welcome/InsertComment">
16 <div class="col-sm-7 slideanim">
17 <div class="row">
18 <div class="col-sm-6 form-group">
19 <input class="form-control" id="name" name="Nombre" placeholder="Nombre" type="text" required>
20 </div>
21 <div class="col-sm-6 form-group">
22 <input class="form-control" id="email" name="email" placeholder="Email" type="email" required>
23 </div>
24 </div>
25 <!--<input class="form-control" id="comments" name="comments" placeholder="Comentarios" style="width:
26 800px; height: 150px;" required><br>-->
27 <textarea class="form-control" id="comments" name="comments" placeholder="Comentarios" rows="5"
28 required></textarea><br>
29 <div class="row">
30 <div class="col-sm-12 form-group">
31 <button class="btn btn-success pull-right" type="submit">Enviar</button>
32 </div>
33 </div>
</form>
```

Imagen 13.1 – Formulario para enviar un comentario.

Una vez llenado el formulario y darle clic al botón “Enviar” nuestro código realizara una acción que hace el llamado a nuestra función InsertComment() como se muestra en la imagen 13.2

```
<form method="POST" action="<?php echo base_url(); ?>index.php/Welcome/InsertComment">
```

Imagen 13.2 – Llamado a la función InsertComment.

La función InsertComment() la podemos encontrar en nuestro archivo del controlador, en el cual obtendrán los valores que se le dieron en el formulario como lo podemos ver en la imagen 13.3 que son guardadas en un arreglo y ese mismo arreglo se manda al archivo del modelo

```
public function InsertComment(){
    $Nombre = $this->input->post('Nombre',TRUE);
    $email = $this->input->post('email',TRUE);
    //$descripcion = $this->input->post('comments',TRUE);
    $descripcion = $_POST['comments'];

    $data = array('Nombre' => $Nombre, 'email' => $email, 'comments' => $descripcion);
    $this->bases->AgregarComentarios($data);
    //echo $data['descripcion'];
    redirect('/Welcome/contacto','location');
}
```

Imagen 13.3 – Función InsertComment() del archivo del controlador.

Como podemos ver la función InsertComment() hace el llamado a una función llamada AgregarComentarios(\$data) como se muestra en la imagen 13.4 que se encuentra en el archivo bases.php de la carpeta de modelos.

```
$this->bases->AgregarComentarios($data);
```

Imagen 13.4 – Llamado a la función AgregarComentarios(\$data) del modelo.

En la función AgregarComentarios(\$data) lo que realiza es tomar los datos que se le pasaron en el formulario y que el controlador la almaceno en un arreglo y lo almacena en la base de datos a través del comando INSERT como se muestra en la imagen 13.5


```

public function AgregarComentarios($data){
    $sql = 'INSERT INTO comentarios (idComentarios, Nombre, Email, Descripcion) VALUES (NULL, "'. $data['Nombre'].',
    "'. $data['email'].', "'. $data['comments'].')';
    $query = $this->db->query($sql);
}

```

Imagen 13.5 – Función AgregarComentarios(\$data) del modelo.

De este modo se realizan la inserción de datos de la página web del CCV y son almacenados en la base de datos.

14. ACTUALIZAR DATOS

Para actualizar los datos de algún registro ya existente en la base de datos, en la página se le coloco la opción de modificar algún registro por lo que en la parte de la vista se le coloco un SELECT para que seleccionara el registro tal como se puede observar en el código de la imagen 14.1 y donde principalmente hace una consulta de los datos de una tabla de la base de datos el cual se explicara en otra sección.

```

tiendas.php
159 <!-- Modal For Edit -->
160 <div id="modalEdit" class="modal">
161
162 <form class="modal-content animate add" method="POST" action="<?php echo base_url();>index.php/Welcome/modificaTienda">
163 <div class="imgcontainer">
164 <span onclick="document.getElementById('modalEdit').style.display='none'" class="close" title="Close Modal">&times;</span>
165 </div>
166
167 <div class="container">
168 <label for="editTienda"><b>Buscar Tienda para Modificar</b></label>
169 <select name="editTienda" id="editTienda">
170 <?php
171 if($tienda != FALSE){
172     foreach ($tienda->result() as $row) {
173     ?>
174     <option value="<?php echo $row->idTienda?>">
175     <?php
176     echo $row->Nombre;
177     ?>
178     </option>
179     <?php
180     }
181     }
182     ?>
183 </select>
184
185 <button type="submit">Buscar</button>
186 </div>
187
188 <div class="container" style="background-color:#f1f1f1">
189 <button type="button" onclick="document.getElementById('modalEdit').style.display='none'" class="cancelbtn">Cancel</button>
190 </div>
191 </form>
192 </div>

```

Imagen 14.1 – Código para seleccionar algún registro a modificar.

Una vez seleccionado el registro a modificar, la vista hará un llamado a la función modificaTienda() como se ve en la imagen 14.2 encontrada en el controlador que nos dirigirá a una nueva vista donde se encuentra el formulario donde se editaran los valores del registro seleccionado como se muestra el código en la imagen

14.3 y que de la misma hace una consulta a todos los valores del registro seleccionado.

```
public function modificaTienda()
{
    if (isset($_SESSION['useradmi'])) {
        $id_tienda = $this->input->post('editTienda', TRUE);
        $data['tienda'] = $this->bases->getTiendaID($id_tienda);
        $this->load->view('header');
        $this->load->view('encabezado_admi');
        $this->load->view('modificaTienda', $data);
        $this->load->view('footer');
    }
    else{
        redirect('/Welcome/index', 'location');
    }
}
```

Imagen 14.2 – Función modificaTienda() que se encuentra en el controlador.

```
<div id="modifica" class="container-fluid text-center">
<?php
    if($tienda != FALSE){
        foreach ($tienda->result() as $row) {
            >
            <form class="login text-center" method="POST" action="<?php echo base_url();>index.php/Welcom/updateTienda">
                <div class="container">
                    <h2>ACTUALIZAR DATOS DE TIENDA <?php echo $row->Nombre;?></h2>
                    <hr>
                    <input type="hidden" value="<?php echo $row->idTienda;?>" name="idT" required>
                    <p for="Tname"><b>Nombre de la Tienda</b></p>
                    <input type="text" value="<?php echo $row->Nombre;?>" name="Tname" required>
                    <p for="descripcion"><b>Descripción</b></p>
                    <input id="descripcion" type="text" value="<?php echo $row->Descripcion;?>" name="descripcion" required>
                    <p for="tel"><b>Telefono</b></p>
                    <input id="tel" type="text" value="<?php echo $row->Telefono;?>" name="tel" required>
                    <p for="horario"><b>Horario</b></p>
                    <input id="horario" type="text" value="<?php echo $row->Horario;?>" name="horario" required>
                    <p for="local"><b>No. Local</b></p>
                    <input id="local" type="text" value="<?php echo $row->No_Local;?>" name="local" required>
                    <p for="Page"><b>Página Web</b></p>
                    <input id="Page" type="text" value="<?php echo $row->Pagina_Web;?>" name="Page" required>
                    <button type="submit">Actualizar</button><br>
                </div>
            </form>
            <?php
        }
    }
```

Imagen 14.3 – Formulario para editar los valores de un registro.

El procedimiento para actualizar datos de la base de datos desde la aplicación es el mismo que el de agregar datos ya que hace el llamado a una función del controlador (imagen 14.4) que este almacena todos los valores en un arreglo como se muestra en la imagen 14.5 y dicha función del controlador hace un llamado a una función del modelo que editara los valores con el comando UPDATE como se ve en la imagen 14.6 y de ese modo se podrá visualizar al instante ya que la función del controlador te recarga la página nuevamente para que veas la tabla con tus registros almacenados.

```

<div id="modifica" class="container-fluid text-center">
<?php
    if($tienda != FALSE){
        foreach ($tienda->result() as $row) {
    }
    <form class="login text-center" method="POST" action="<?php echo base_url();>index.php/welcome/updateTienda">

        <div class="container">

            <h2>ACTUALIZAR DATOS DE TIENDA <?php echo $row->Nombre;?></h2>
            <hr>

            <input type="hidden" value="<?php echo $row->idTienda;?>" name="idT" required>

            <p for="Tname"><b>Nombre de la Tienda</b></p>
            <input type="text" value="<?php echo $row->Nombre;?>" name="Tname" required>

            <p for="descripcion"><b>Descripción</b></p>
            <input id="descripcion" type="text" value="<?php echo $row->Descripcion;?>" name="descripcion" required>

            <p for="tel"><b>Telefono</b></p>
            <input id="tel" type="text" value="<?php echo $row->Telefono;?>" name="tel" required>

            <p for="horario"><b>Horario</b></p>
            <input id="horario" type="text" value="<?php echo $row->Horario;?>" name="horario" required>

            <p for="local"><b>No. Local</b></p>
            <input id="local" type="text" value="<?php echo $row->No_Local;?>" name="local" required>

            <p for="Page"><b>Página Web</b></p>
            <input id="Page" type="text" value="<?php echo $row->Pagina_Web;?>" name="Page" required>

            <button type="submit">Actualizar</button><br>
        </div>
    </form>
<?php
}
}

```

Imagen 14.4 – Llamado a la función updateTienda().

```

public function updateTienda(){
    $id_tienda = $this->input->post('idT', TRUE);
    $Nombre = $this->input->post('Tname', TRUE);
    $Descripcion = $this->input->post('descripcion', TRUE);
    $Telefono = $this->input->post('tel', TRUE);
    $Horario = $this->input->post('horario', TRUE);
    // $Imagen = $this->input->post('vimagen', TRUE);
    $NoLocal = $this->input->post('local', TRUE);
    $Web = $this->input->post('Page', TRUE);

    // $data = array('Tname' => $Nombre, 'descripcion' => $Descripcion, 'tel' => $Telefono, 'horario' => $Horario, 'vimagen' => $Imagen, 'local' =>
    $data = array('idT' => $id_tienda, 'Tname' => $Nombre, 'descripcion' => $Descripcion, 'tel' => $Telefono, 'horario' => $Horario, 'local' => $NoLocal
    , 'Page' => $Web);
    $this->bases->updateTienda($data);
    //echo $data['descripcion'];
    redirect('/welcome/tienda/#tiendas', 'location');
}

```

Imagen 14.5 – Función updateTienda() del controlador.

```

public function updateTienda($data){
    $sql = 'UPDATE tienda SET Nombre = "'.$data['Tname'].'", Descripcion = "'.$data['descripcion'].'", Telefono = "'.$data['tel'].'", Horario = "'.$data[
    'horario'].'", No_Local = "'.$data['local'].'", Pagina_Web = "'.$data['Page'].'"' WHERE idTienda = "'.$data['idT'].'"';
    $query = $this->db->query($sql);
}

```

Imagen 14.6 – función updateTienda(\$data) del modelo

De este modo es como se trabaja la actualización de datos desde el código de la aplicación del CCV.

15. ELIMINAR DATOS

Para eliminar datos de la base de datos desde la aplicación es una forma más sencilla ya que realizamos un procedimiento parecido con el de actualizar datos

ya que a través de un SELECT como se ve en la imagen 15.1 seleccionaremos un registro ya existente en la base de datos y daremos clic en el botón de “Eliminar” donde lo que hará el código es hacer un llamado a la función de borrado en el controlador (imagen 15.2) y el cual esa función almacenara el valor del registro seleccionado y la enviara a la función del modelo (imagen 15.3) y esta realizará la eliminación del registro por medio del comando DELETE (imagen 15.4)

```
<div id="modalDelete" class="modal">
  <form class="modal-content animate add" method="POST" action="<?php echo base_url();>index.php/welcome/eliminaTienda">
    <div class="imgcontainer">
      <span onclick="document.getElementById('modalDelete').style.display='none'" class="close" title="Close Modal">&times;</span>
    </div>
    <div class="container">
      <label for="deleteTienda"><b>Buscar Tienda a Eliminar</b></label>
      <select name="deleteTienda" id="deleteTienda">
        <?php
          if($tienda != FALSE){
            foreach ($tienda->result() as $row) {
              <?php
                <option value="<?php echo $row->idTienda?>">
                  <?php
                    echo $row->Nombre;
                  </?php>
                </option>
              <?php
            }
          <?php
        </select>
        <button type="submit" class="btn btn-danger">Eliminar</button>
      </div>
      <div class="container" style="background-color:#f1f1f1">
        <button type="button" onclick="document.getElementById('modalDelete').style.display='none'" class="cancelbtn">Cancel</button>
      </div>
    </form>
  </div>
```

Imagen 15.1 – Código para seleccionar un registro a eliminar.

```
<form class="modal-content animate add" method="POST" action="<?php echo base_url();>index.php/welcome/eliminaTienda">
```

Imagen 15.2 – Llamado a la función eliminaTienda().

```
public function eliminaTienda(){
    $Nombre = $this->input->post('deleteTienda',TRUE);

    $data = array('deleteTienda' => $Nombre);
    $fila = $this->bases->eliminaTienda($data);
    //echo $fila;
    redirect('/Welcome/tienda/#tiendas','location');
}
```

15.3 – Función eliminaTienda() del controlador.

```
public function eliminaTienda($data){
    $sql = 'DELETE FROM tienda WHERE idTienda = "'.$data['deleteTienda'].'"';
    $query = $this->db->query($sql);
}
```

Imagen 15.4 – Función eliminaTienda() del modelo.

Cabe mencionar que de esta manera es como se trabaja la parte de eliminar datos de nuestra base de datos, también algo importante es poner el comando WHERE al momento de realizar la eliminación ya que sin el WHERE corremos el riesgo de eliminar todos los valores de una tabla de nuestra base de datos.

16. CONSULTA DE DATOS

En la parte de la consulta de datos se realizan dos formas, consultar toda la tabla general o mostrar un solo registro que para este caso lo realiza en la parte de actualización y eliminación de datos y que explicaremos en esta sección.

Para mostrar todos los registros de la tabla el usuario no tiene que realizar ninguna acción ya que la misma aplicación se lo mostrará automáticamente y que en algunas ocasiones como lo son en la parte del administrador le mostrará los registros en una tabla, para ello en alguna función del controlador (por ejemplo la función “evento()” que se ve en la imagen 16.1) le solicitará al modelo una consulta a través de una función y la almacenará en un arreglo como se ve en la imagen 16.2 y el cual dicho arreglo lo mandará al archivo de la vista (imagen 16.3) para que dentro de la vista la podamos manipular e ir colocando los datos dentro de una tabla como se ve en la imagen 16.4

```
public function evento()
{
    if (isset($_SESSION['useradmi'])) {
        $data['evento'] = $this->bases->getEventos();
        $this->load->view('header');
        $this->load->view('encabezado_admi');
        $this->load->view('eventos', $data);
        $this->load->view('footer');
    }
    else{
        redirect('/Welcome/index', 'location');
    }
}
```

Imagen 16.1 – Función evento() del controlador.

```
$data['evento'] = $this->bases->getEventos();
```

Imagen 16.2 – Llamado a la función getEventos() y almacenados en un arreglo en la posición “evento”.

```
$this->load->view('eventos', $data);
```

Imagen 16.3 – Enviamos la variable data a la vista para su manipulación

```

<table border="2px">
<thead>
<tr>
<th align="center" bgcolor="#000000" style="color:white">NOMBRE</th>
<th align="center" bgcolor="#000000" style="color:white">LUGAR</th>
<th align="center" bgcolor="#000000" style="color:white">DESCRIPCIÓN</th>
<th align="center" bgcolor="#000000" style="color:white">FECHA</th>
<th align="center" bgcolor="#000000" style="color:white">HORARIO</th>
</tr>
</thead>
<?php
if($evento != FALSE){
    foreach ($evento->result() as $row) {
    <tr>
<td><?php echo $row->Nombre; ?></td>
<td><?php echo $row->Lugar; ?></td>
<td><?php echo $row->Descripcion; ?></td>
<td><?php echo $row->FechaEvento; ?></td>
<td><?php echo $row->Horario; ?></td>
</tr>
<?php
    }
}
?>
</table>

```

Imagen 16.4 – colocamos los datos en una tabla para que el usuario lo pueda ver.

Para realizar la consulta de datos hacemos uso de la función `getEventos()` en este caso y el cual podemos ver el código en la imagen 16.5 donde usamos el comando `SELECT` y que dicho archivo se encuentra en la carpeta del modelo.

```

public function getEventos(){
    $sql = 'SELECT * FROM evento ORDER BY Nombre';
    $query = $this->db->query($sql);

    if($query->num_rows() > 0)
        return $query;
    else
        return FALSE;
}

```

Imagen 16.5 – Función `getEventos()` del modelo.

Ahora bien, para realizar la consulta a un solo registro el procedimiento es el mismo solo que hacemos uso de una línea de código más y el cual la mandamos como parámetro en una función del modelo como se ve en la imagen 16.6 y la función del archivo del modelo aplica el comando `WHERE` como se ve en la imagen 16.7

```

$id_tienda = $this->input->post('editTienda', TRUE);
$data['tienda'] = $this->bases->getTiendaID($id_tienda);

```

Imagen 16.6 – Consulta a un solo registro de la base de datos


```

public function getTiendaID($id_tienda){
    $sql = 'SELECT * FROM tienda WHERE idTienda = "'.$id_tienda.'";
    $query = $this->db->query($sql);

    if($query->num_rows() > 0)
        return $query;
    else
        return FALSE;
}

```

Imagen 16.7 – Archivo de consulta de un solo registro

Cabe mencionar que existe otro modo de consultar un registro y el cual se ocupa en la parte del mapa interactivo y es a través del comando “href” (imagen 16.8) en el cual el controlador lo almacena en una variable (imagen 16.9) y lo manda como parámetro en la función del modelo (imagen 16.10)

```

self" />
href="<?php echo base_url();?>index.php/Welcome/tienda/26/#InfoTienda" s
self" />
href="<?php echo base_url();?>index.php/Welcome/tienda/48/#InfoTienda" s
self" />

```

Imagen 16.8 – Comando href donde envía los valores 26 y 48 en este ejemplo.

```

$data['id'] = $this->uri->segment(3);

```

Imagen 16.9 – Se almacena el valor enviado a una variable.

```

$data['tienda'] = $this->bases->GetTienda($data);

```

Imagen 16.10 – Se envía el valor como parámetro de una función y se almacena el resultado en otra variable para que pueda ser usada en la vista.

17. MAPA INTERACTIVO

Se ocupó la librería jquery.maphilight.js. Esta librería se encuentra en la ruta CCVictoria/Librerias/maphilight-master/jquery.maphilight.js

Para indicar el resaltado de tiendas y locales se creó una función extra con jquery el cual llamara al método .maphilight () de la librería. Véase la imagen 17.1

```
<script type="text/javascript" src="<?php echo base_url();>Librerias/maphilight-master/jquery.maphilight.js"></script>

<script type="text/javascript">$(function() {
    $('.map').maphilight();
    $('.s').css('background-color', 'green');
});</script>
```

Imagen 17.1 – Función maphilight

Para un cambio o actualización del plano se tendrá que rediseñar desde la página <https://www.image-maps.com/>. Véase la Imagen 17.2 y 17.3.

Con user: **sistemas** pass: **@Mampar0t1**

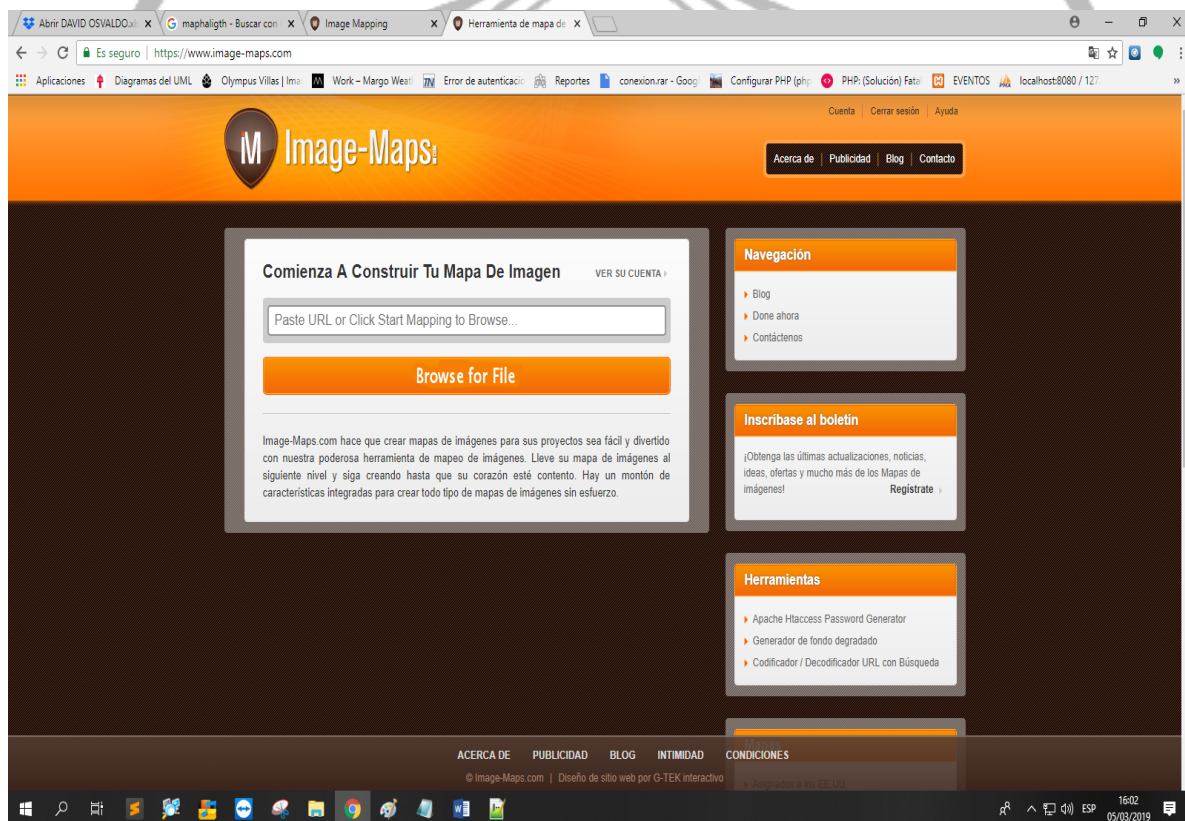


Imagen 17.2 – Pagina web



Imagen 17.3 – Pagina web

Nota: Es necesario dejar la carpeta en la ruta CCVictoria/Librerias de lo contrario no reconocerá su funcionamiento.

Para realizar la parte de consultar la tienda a la que se le da clic en el mapa se colocó la función “href” donde en el link que se coloca (véase imagen 17.4) se le envía un dato en este caso se envía el número de local.

```
4" href="<?php echo base_url();?>index.php/Welcome/tienda/24/#InfoTienda" s
5" href="<?php echo base_url();?>index.php/Welcome/tienda/25/#InfoTienda" s
5" href="<?php echo base_url();?>index.php/Welcome/tienda/26/#InfoTienda" s
8" href="<?php echo base_url();?>index.php/Welcome/tienda/48/#InfoTienda" s
```

Imagen 17.4 – Función href donde en el link se le envía el número de local.

Para capturar el valor dentro de nuestro controlador hacemos uso de la función uri y ese valor lo almacenamos en una variable que la pasaremos como parámetro en una función del modelo (véase imagen 17.5) para consultar la tienda seleccionada buscándola por el número de local (véase imagen 17.6).

```
$data['id'] = $this->uri->segment(3);
$id = $data['id'];
if ($id != NULL) {
    $data['tienda'] = $this->bases->GetTienda($data);
    $this->load->view('header');
```

Imagen 17.5 – Se almacena el valor y se manda como parámetro.

```

public function GetTienda($data){
    $sql = 'SELECT * FROM tienda WHERE No_Local = "'.$data['id'].'"';
    $query = $this->db->query($sql);

    if($query->num_rows() > 0)
        return $query;
    else
        return FALSE;
}

```

Imagen 17.6 – Función para consultar la tienda por número de local.

18. ALMACENAR IMÁGENES EN LAS CARPETAS DEL PROYECTO

Dentro de esta aplicación se realizó también una parte en la que desde el código se almacena las imágenes de los logos, eventos y promociones en sus respectivas carpetas del proyecto y así se puede hacer el llamado a esa misma imagen.

Para realizar esta acción primero en nuestro formulario donde se selecciona la imagen colocaremos el texto “enctype=’multipart/form-data” (véase imagen 18.1) y también se colocará el tipo dato en nuestra etiqueta input, que en nuestro caso es tipo file (véase imagen 18.2).

```

<form class="modal-content animate add" enctype="multipart/form-data" method
<div class="imgcontainer">
    <span onclick="document.getElementById('modalAdd').style.display='none'"
</div>

```

Imagen 18.1 – Formulario con la función enctype.

```

<label for="vimagen"><b>Imagen Logo</b></label>
<!--<input id="vimagen" type="text" name="vimagen" required>-->
<input id="vimagen" type="file" name="vimagen" onChange="ver(form.file.value)">

```

Imagen 18.2 – Tipo de la etiqueta Input.

Ahora en nuestro controlador debemos realizar la acción de almacenar la imagen de su carpeta origen a la carpeta destino con la ayuda de la función move_uploaded_file() que nos proporciona php (véase imagen 18.3) y dentro de nuestra base de datos solo almacenamos el nombre de la imagen.

```

//Imagen = $this->input->post('vimagen',TRUE);
$NoLocal = $this->input->post('local',TRUE);
$Web = $this->input->post('Page',TRUE);
$Imagen = $_FILES['vimagen']['name'];

$data = array('Nombre' => $Nombre, 'descripcion' => $Descripcion, 'tel' => $Telefono, 'horario' => $Horario, 'vimagen' => $Imagen,
$this->bases->ingresaTienda($data);

$dir_imagen = "C:/xampp/htdocs/CCVictoria/logo_tiendas/";
$dir_subida = $dir_imagen.basename($_FILES['vimagen']['name']);
move_uploaded_file($_FILES['vimagen']['tmp_name'], $dir_subida);

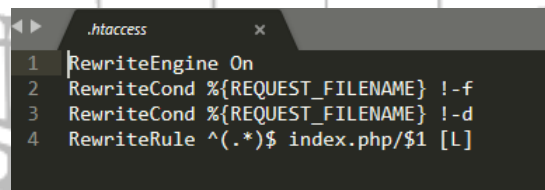
```

Imagen 18.3 – Uso de la función move_uploaded_file().

19. RUTAS

La página del CCV maneja las llamadas URL Amigables o Rutas, por ejemplo, que aparezca en el link solamente ccvictoria/promoción. Sin estas rutas amigables tenemos el link como el siguiente: ccvictoria/index.php/Welcome/promoción.

Primero debemos quitar del link index.php para ello se debe agregar un nuevo archivo llamado .htaccess y lo colocaremos en la raíz de la carpeta de nuestro proyecto y dentro de este archivo se colocará unas líneas de código como se muestra en la imagen 19.1, esto para omitir ver el archivo index.php en nuestro link.



```

1 RewriteEngine On
2 RewriteCond %{REQUEST_FILENAME} !-f
3 RewriteCond %{REQUEST_FILENAME} !-d
4 RewriteRule ^(.*)$ index.php/$1 [L]

```

Imagen 19.1 – Contenido del archivo .htaccess

Ahora para omitir la palabra “Welcome” en nuestro link debemos hacer uso del archivo routes.php que se encuentra en la carpeta config (véase Imagen 19.2) en este archivo colocaremos nuestras rutas tal como se observa en la imagen 19.3 y de esta manera ya podremos hacer uso de unas rutas mas accesibles para el usuario.

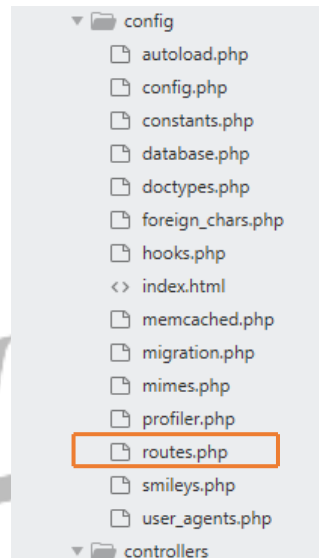


Imagen 19.2 – Archivo routes.php

A screenshot of a code editor showing the contents of the 'routes.php' file. The code defines several routes using the 'route' function. It starts with examples for 'my-controller/index' and 'my-controller/my-method'. Then, it sets default values for 'default_controller' to 'welcome', '404_override' to an empty string, and 'translate_uri_dashes' to 'FALSE'. A comment '/* Mis Rutas */' precedes a list of custom routes: 'index' points to 'welcome/index', 'promocion' to 'welcome/promocion', 'servicio' to 'welcome/servicio', 'tienda' to 'welcome/tienda', 'galeria' to 'welcome/galeria', 'evento' to 'welcome/evento', 'contacto' to 'welcome/contacto', 'politicas' to 'welcome/politicas', and 'consulta_tienda' to 'welcome/tienda/?/#InfoTienda'.

Imagen 19.3 – Rutas personalizadas para la aplicación dentro del archivo routes.php

De esta manera ya podemos hacer uso de rutas mas accesibles para el usuario.

20. Referencias

- [1]. Código Facilito (2019). MVC (Model, View, Controller) Explicado.
<https://codigofacilito.com/articulos/mvc-model-view-controller-explicado>
- [2]. Desarrollo Web (2014). Qué es MVC.
<https://desarrolloweb.com/articulos/que-es-mvc.html>
- [3]. CodeIgniter (2019). CodeIgniter Web Framework.
<https://www.codeigniter.com/>