UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

"CREACIÓN DE UN SITIO WEB INSTITUCIONAL CON EL FRAMEWORK PHALCON"

TRABAJO ESCRITO

EN LA MODALIDAD DE: MEMORIA DE DESEMPEÑO DE SERVICIO SOCIAL

QUE PARA OBTENER EL TÍTULO

DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

BLAS DE JESUS ANGEL HERIBERTO

ASESOR: M. EN C. MARCELO PÉREZ MEDEL

Ciudad Nezahualcóyotl, Estado de México, 2018.

Agradecimientos

Primero agradecer a Dios por acompañarme en mi trayectoria escolar, por darme salud y cuidarme en todo momento. Por darme el entendimiento para afrontar exámenes, tareas. Por ayudarme económicamente a que no me faltara nada y contar con lo necesario para terminar la carrera.

A mi mamita hermosa, porque siempre estuvo ahí, todos los días se levantaba a la misma hora que yo para darme algo de desayunar o para llevarlo a la escuela, tomaba parte de su gasto para mi pasaje cuando no completaba, sin lugar a dudas es la persona que más admiro.

A todos mis hermanos que siempre me apoyaron, me dieron un consejo o una enseñanza. En especial, quiero agradecer a Adrián por ser un todo poderoso, casi como un papá, siempre me ayudó económicamente en todo lo que pudo, sin esperar nada a cambio. Gracias a ti pude terminar una carrera, ahora me toca a mí ayudarles.

Al amor de mi vida y mejor amiga Alejandra Arévalo Morales, por todos estos años que llevamos de conocernos y ser novios, porque siempre estuviste ahí echándome porras, por apapacharme y darme energías, por las risas y los buenos momentos juntos. ¡Que vengan muchos años más!

Al M. en C. Marcelo Pérez Medel por todas sus enseñanzas dentro y fuera de clases, aprendí mucho de usted en la carrera y en el diplomado. Gracias por hacerme formar parte del Lab Com y ser como un amigo, por ser mi asesor de tesis y darme siempre muy buenos consejos.

Al M. en C. Jesús Hernández Cabrera que hizo todo esto posible, por confiar en mí a apoyarlo en los cursos intersemestrales. Gracias por portarse como un amigo y preocuparse siempre por sus alumnos. Sin duda alguna es una máquina de conocimientos.

Al Dr. Edgar Morales Palafox por todas sus enseñanzas y confiar en mí en los proyectos que se realizaban en Lab Com. Gracias por ser como un amigo y Dios quiera que pronto se recupere.

A mis amigos de la carrera Eduardo B., Eliot S., Ramón G., Dayane H., por hacer que la carrera fuera divertida, por las pláticas que terminaban en risas y por el equipo que formamos para salir adelante en las asignaturas. En especial gracias a Lalo porque siempre estuvo apoyándome como un hermano.

A mis compañeros que conocí en Lab Com por todos los buenos momentos. En especial Omar G., por convertirse en mi amigo y apoyarme cuando lo necesitaba, por darme consejos al realizar la tesina y los momentos divertidos jugando LoL.

A Pedro F., Martin F. y Carlos F., y demás amigos del Porto por hacer mis días de risa y hacer el estrés de lado.

Contenido

Introducción

En la actualidad contar con un sitio web es indispensable para una institución, empresa o gobierno, debido a que las recientes tecnologías nos permiten acceder a internet desde cualquier parte del mundo y poder consultar información con un dispositivo móvil o de escritorio.

Por ejemplo, un sitio web creado para una empresa serviría para mostrar al público los servicios que ofrece, permitiría también realizar transacciones en línea o permitir a sus clientes acceder a su información de los movimientos y compras que ha realizado.

Otro ejemplo, son los sitios web creado para las instituciones que proporción información a las personas que estén interesados en tomar algún curso. Las finalidades de estos sitios son puramente publicitarios e informativos, aunque en algunos se pueden realizar cursos en línea y así permitir a los alumnos subir tareas o trabajos desde sus casas.

Para crear un sitio web existen diversas tecnologías, desde lo básico que es realizando páginas estáticas con HTML y algo de CSS. La desventaja de hacerlo de esta manera es que el sitio no puede interactuar con el usuario ni mostrar información que solo sea requerida ya que no se cuenta con acceso a una base de datos.

La mejor forma de crear un sitio web utilizando un lenguaje de programación que nos permita interactuar con una base de datos y los datos introducidos por el usuario. La mayoría de lenguajes ya están orientados al desarrollo web: ASP, JAVA, Phyton, Ruby, PHP, etc.

Crear un sitio web que implique una elaboración alta, necesita de grandes recursos como talento humano y equipo. También se necesitan más horas de trabajo para desarrollar la infraestructura, definir un lenguaje de programación que más se adapte a las necesidades, la seguridad, la maquetación (diseño del sitio), la conexión con la base de datos, los módulos que conforman el sitio, así como las rutas de acceso.

Para facilitar todas estas tareas existen diferentes *frameworks* que son marcos de trabajo orientados a resolver problemas específicos y que nos ayudan a optimizar procesos de mediante estructuras de código prediseñadas para cierto lenguaje y que están listas para ser usadas cuando se necesiten. Por un lado, el desarrollador sólo se preocupa por la funcionalidad y el diseño del sitio, y por el otro, el *framework* se encarga de administrar las tareas que resultan ser monótonas cuando se inicia un proyecto.

Este trabajo se basa en la experiencia y conocimientos adquiridos como parte del servicio social que realice en el Laboratorio de cómputo, dentro de la Facultad de Estudios Superiores Aragón, en

la cual participe en la creación de un sitio web para el Centro Tecnológico Aragón, utilizando como lenguaje de programación PHP y de *framework,* **Phalcon**.

En el capítulo 1 veremos una introducción al *framework* **Phalcon** donde se detalla:

- Definición y características
- Ventajas de uso y ventajas con otros *frameworks*
- Qué es el modelo MVC y cómo se estructuran sus carpetas
- Creación de un proyecto en **Phalcon** desde cero
- Definición y uso del motor de plantillas VOLT
- Definición y ejemplos de enrutamiento
- Conexión a base de datos y consultas PHQL al modelo
- Qué son las peticiones GET y POST
- Cómo cargar recursos al proyecto

Durante el capítulo 2, se hace enfoque al diseño del sitio web (conocido como *Front- end*) y a la forma de mostrar información dinámicamente, sin recargar la página que el usuario está visualizando. Se explica a detalle como agregar estilos a los elementos HTML que conforman la página haciendo uso de CSS y los *frameworks Bootstrap* y *jQuery*:

CSS:

- Qué es y cómo usar una hoja de estilo
- Herencia de estilos
- Asignar propiedades por medio de clases, id y demás selectores
- Agregar estilos al texto
- Uso de márgenes
- Fondos de pantalla
- Posicionamiento

Bootstrap:

- Qué es y como importarlo
- Aprender a maquetar utilizando el sistema de grillas
- Agregar estilos al texto
- Crear tablas y hacerlas responsivas
- Manejar imágenes
- Agregar botones y formularios
- Que son y como utilizar ventanas modal

jQuery:

- Que es y como importarlo al proyecto
- Manipulación de los elementos del DOM
- Manejar eventos
- Que es AJAX y ejemplo de uso
- Seleccionar nodos padres, nodo hijo y nodo hermano

Para el capítulo 3 se hace uso de la mayoría de herramientas vistas en los capítulos anteriores para explicar la creación del sitio web del Centro Tecnológico Aragón en los módulos:

- Galería
- Directorio
- Eventos

Para este apartado también definimos que es un *plugin* de jQuery y como adaptarlo al proyecto.

Capítulo 1. Introducción al framework Phalcon

1.1 Introducción

En este capítulo se presentan algunos conceptos que son importantes para conocer y utilizar un marco de trabajo (*framework* de PHP*).

Un proyecto va pasando por una serie de fases a lo largo de su desarrollo. Dichas etapas son conocidas como el ciclo de vida de un sistema de cómputo:

- Planificación. Estudios de viabilidad, análisis de riesgos, costos, tiempos y recursos.
- Análisis. Se refiere a las características que debe tener el sistema.
- Diseño. Son los módulos que debe tener el sistema para que resuelva las características suscitadas en la fase de análisis.
- Implementación . Desarrollo del proyecto.
- Pruebas. Ejecución de pruebas unitarias y pruebas integrales con la finalidad de detectar errores.
- Instalación o despliegue. Poner en marcha el proyecto en un entorno que permita el correcto funcionamiento.
- Uso y mantenimiento. Añadir nuevas funcionalidades, hacer actualizaciones, corregir errores.

Enfocándonos al diseño e implementación, en esta etapa se define la estructura de archivos y carpetas del proyecto:

- Carpetas con archivos para la conexión y consultas a la base de datos
- Carpetas con archivos que definen la interfaz gráfica
- Carpetas donde se incluyen fuentes
- Carpetas que contengan *scripts***
- Archivos con información en formato *JSON****
- Archivos PHP con la lógica del proyecto
- Carpetas que contienen imágenes

*PHP es un lenguaje de programación para el desarrollo web.

**Los scripts son un conjunto de instrucciones programadas para ser ejecutadas cuando ocurre un evento o cuando pasa un tiempo determinado.

***JSON (*JavaScript Object Notation*) es un formato de texto ligero para el intercambio de datos en la red. Popular en la mayoría de los lenguajes de programación.

- Carpetas con hojas de estilo

Ahora bien ¿Cómo organizamos todo esto? Algunos programadores tienen su propia forma de estructurar un proyecto, una propuesta de ello es la que se muestra en la figura 1.1.

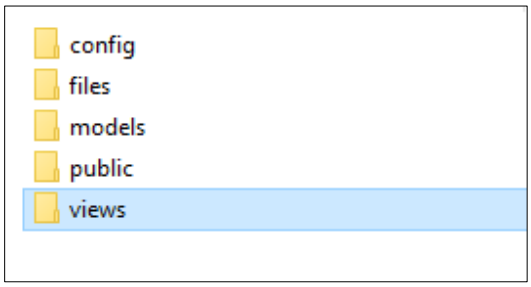


Figura 1.1 Estructura de carpetas.

La carpeta *config* contendrá archivos de configuración general (*back-end**), así como de conexión a la base de datos (en PHP), dentro de la carpeta *files* tenemos archivos con datos JSON, *models* contará con todas las *queries*** para interactuar con la base de datos. La carpeta *public* posee archivos o imágenes que se utilizan para darle estilos a nuestras páginas (librerías JS, fuentes, archivos CSS) y en *views* se colocan los archivos que conforman las interfaces gráficas (HTML).

Las diferentes formas de estructurar un proyecto son debido a que no se está utilizando un patrón de diseño.

Un **patrón de diseño** es una solución reutilizable para resolver un problema de diseño que sea común en el contexto en que se utilice. Se trata de plantillas que definen una solución recurrente al momento de programar y además no son exclusivos de un lenguaje de programación, sino que pueden utilizarse en la mayoría de estos.

Los patrones de diseño están formalizados como buenas prácticas en la programación y se dividen en:

- Creacionales: Definen la mejor forma instanciar un objeto de una forma abstracta.
- Estructurales: Son aquellos que especifican la forma de crear nuevos objetos nuevas funcionalidades por medio de la herencia de otros.
- De comportamiento: Tratan la comunicación entre objetos.
- De arquitectura: Fija como se debe estructurar un proyecto.

*El *back-end* es la parte encargada de procesar los datos de entrada que manda el usuario desde el entorno grafico (*front-end*). Esta parte se encuentra oculta al usuario final.

**Una *query* es una consulta que se realiza a la base de datos para mostrar un resultado.

En este caso solo nos enfocaremos en el patrón de diseño de arquitectura de software, ya que son utilizados en la mayoría de *frameworks*. Algunos de estos patrones son:

- *Front controller*. Define una forma de controlar los *requests* (peticiones).
- MVC. Model-View-Controller. Divide la aplicación en tres partes interconectadas, separando el *back-end* del *front-end*.
- ADR. Action-Domain-Responder. Se plantea como un patrón MVC más refinado y orientado al desarrollo web.
- *Service locator*. Emplea un registro central que devuelve información necesaria para tareas cuando se hacen peticiones.
- *Active record.* Acceso a una base de datos mediante un objeto.
- *Publish-subscribe*. Forma de notificar los cambios en un número de clases.
- *Inversion of control*. Diseño que invierte el funcionamiento tradicional en el que el código personalizado llama a librerías reusables.

En PHP (y cualquier otro lenguaje) el Modelo-Vista-Controlador es utilizado por los *frameworks* más populares ya que separa la lógica de negocio del interfaz utilizado y tiene como finalidad crear sistemas robustos y fáciles de mantener. Por esta razón, en este documento nos centraremos a hablar de dicho patrón**.**

El *Modelo-Vista-Controlador* o MVC es un patrón de arquitectura que divide la lógica de negocio de la lógica de la vista en tres componentes:

- Los modelos. Se encarga de la administración de datos de la lógica de negocio, guardando y recuperando información de la base de datos. Las modificaciones realizadas son comunicadas a la vista.
- Las vistas. Presentan toda la información visual al usuario, todo lo relacionado a la interfaz gráfica debe estar en las vistas. Estos datos pueden ser solicitados por el usuario y devueltos por el modelo.
- Los controladores. Recibe peticiones hechas por el usuario y las atiende, si es requerido solicita datos al modelo y después despacha el resultado a la vista.

El ciclo de vida del MVC es el siguiente:

1. El usuario hace una petición.
2. El controlador recibe la petición.
3. El controlador solicita los datos al modelo correspondiente (si se requiere).
4. El modelo hace las consultas necesarias a la base de datos.
5. El controlador recibe la información proporcionada por el modelo y la envía a la vista.
6. La vista muestra al usuario la información que responda a su solicitud.

Anteriormente habíamos creado una estructura de carpetas cuando no se utilizaba un patrón de diseño (Figura 1.1), ahora que sabemos que es el MVC, se propone una estructura diferente pero básica utilizando este patrón (Figura 1.2).

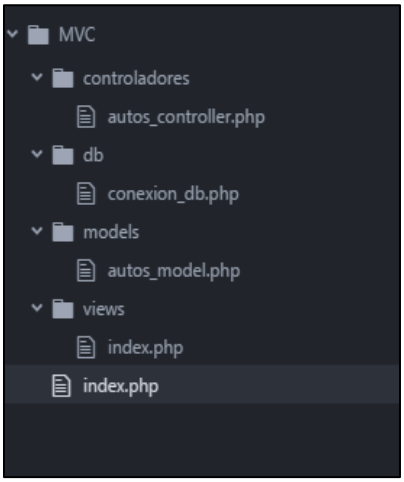


Figura 1.2 Estructura de carpetas con MVC.

Con base a la estructura anterior crearemos un ejemplo básico de uso del MVC. Para esto es necesario saber un poco de PHP y Programación orientada a objetos*.

- “index.php”. Se carga la conexión a la base de datos y el controlador inicial.

```php
<?php
  //cargamos el archivo con la configuracion a la base de datos
  require_once("db/conexion_db.php");
  //cargamos nuestro controlador
  require_once("controladores/autos_controller.php");
?>
```

Figura 1.3 carga de la conexión con base de datos y controlador principal.

*La programación orientada a objetos o simplemente POO es un paradigma que asemeja la programación a lo que una persona puede percibir en la vida real, haciendo uso de objetos y cada objeto contiene propiedades y acciones (métodos).

Un ejemplo sencillo utilizando este paradigma: un coche puede acelerar. Donde el coche es el objeto y acelerar corresponde al método.

- “conexion_db.php”. creamos un método estático donde retornamos una conexión a partir de las credenciales de acceso a la base de datos (**Figura 1.4**).

```php
<?php
  class Conexion{
    public static function conectar(){
      //conexion con nuestra base de datos
      $conexion=new mysqli("localhost", "root", "", "mvc");
      $conexion->query("SET NAMES 'utf8'"); //codificación de simbolos y caracteres
      return $conexion;//retornamos la conexión establecida
    }
  }
?>
```

Figura 1.4 Archivo “conexion_db.php”.

- “autos_model.php”. Este archivo contiene la clase “ModeloAutos” que incluye dos métodos: un constructor para inicializar nuestras variables y “getAutos” que retorna una lista con los resultados obtenidos de realizar la consulta.

```php
<?php
  class ModeloAutos{
    private $conexion;
    private $autos;
    //metodos constructor donde inicializamos nuestra conexión
    // y el arreglo autos
    public function __construct(){
        $this->conexion=Conexion::conectar();
        $this->autos=array();
    }
    //metodos donde obteenmos los autos con base a nuestra consulta
    public function getAutos(){
        $consulta=$this->conexion->query("select * from autos;");
        while($filas=$consulta->fetch_assoc()){
            $this->autos[]=$filas;
        }
        //retornamos el arreglo con los autos obtenidos de la consulta
        return $this->autos;
    }
  }
?>
```

Figura 1.5 Archivo *autos_model.php*.

- “autos_controller.php”. Este controlador importa el modelo creado anteriormente. Después, se declaran dos variables: la variable $autos instancía un objeto de la clase ModeloAutos y en la variable $coches se asignan los datos retornados por el método “getAutos()”.

Al final se incluye el código de la vista.

```php
<?php
// se realiza una Llamada al modelo
require_once("models/autos_model.php");
$autos=new ModeloAutos();
$coches=$autos->getAutos();

//Llamada a la vista
require_once("views/autos_view.php");
?>
```

Figura 1.6 Archivo “autos_controller.php”.

- “autos_view.php”. Contiene código con las etiquetas básicas para crear una página HTML. En esta página se coloca a través de un *foreach* (sentencia de control semejante a un ciclo *for*) el arreglo con los coches retornados de la consulta.

```php
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <title>Autos 2018</title>
    </head>
    <body>
      /se imprime el arreglo dada la marca del coche
        <?php
            foreach ($coches as $coche) {
                echo $coche["marca"]."<br/>";
            }
        ?>
    </body>
</html>
```

Figura 1.7 Archivo “autos_view.php”.

Para el MVC es recomendable usar estándares de nomenclatura en los archivos y clases creadas, aunque algunos *frameworks* por ejemplo **Phalcon** sugieren llamar al controlador como “AutosController”, a los modelos como “Autos” y a la vista “autos.volt”. La extensión VOLT la veremos más adelante.

Algunas ventajas de utilizar el MVC son:

- Se divide la lógica de negocio del diseño.
- Es utilizado por muchos *frameworks* en PHP y otros lenguajes de programación.
- Reutilizar código. Para las vistas te permite duplicar estilos o contenidos. El manejo de los datos se hace mediante los modelos así que si en algún momento modificas la base de datos solo tienes que modificar el modelo correspondiente.

- Fácil de mantener. Los cambios que se realicen en una parte producirán menor impacto a otras, ya que se encuentran aisladas unas de otras.
- Permite la escalabilidad ya que cuando necesitemos agregar nuevas funcionalidades a nuestra aplicación sabremos en qué parte añadir código y cómo organizarlo.

Ahora que vimos que es un patrón de diseño y el MVC, es momento de saber que es un *framework**.***

Un ***framework*** o marco de trabajo es un conjunto estandarizado de componentes (clases, funciones, bibliotecas, archivos de configuración, patrones de diseño) y buenas practicas que sirven como punto de partida para resolver problemas similares a los que fue diseñado. En pocas palabras es una servicio genérico incompleto donde nosotros agregamos y configuramos lo que haga falta.

Algunas ventajas al utilizarlo son:

- Acelerar el desarrollo de la aplicación. De esta manera solo te enfocas en resolver el problema para el que fue requerido el proyecto y no perder tiempo en estructurar y configurar una aplicación desde cero.
- Reutilizar código.
- Utilizan buenas prácticas y uso de patrones de diseño arquitectónicos.
- Contienen herramientas que ayudan con el cache, paginación, etc.
- Brindan cierta seguridad para proteger tu sitio de ataques.
- Organización de archivos y carpetas utilizando estándares.
- Comunicación fácil y automatizada con la base de datos.

La siguiente tabla muestra los *frameworks* más utilizados en PHP con el MVC.

Nombre	Versiones PHP	BD soportadas	Conocimientos previos	Documentación	Comunidad	Versión actual
codeIgniter 	5.6 o superior	-MySQL (5.1+) -Oracle -PostgreSQL -MS SQL -SQLite -CUBRID -Firebird	-PHP -Programación Orientada a Objetos	-Actualizada constantemente -Organizada	-Foros -Blogs -Wiki	3.1.8

		-ODBC				
Zend	5.6 o superior	-IBM DB2 -MariaDB -MySQL -Oracle -Microsoft SQL Server -SQLite -Firebird	-PHP avanzado -Programación Orientada a Objetos -Conocimiento amplio de MVC.	-Actualizada -Amplia -Organizada	-Foros -Blogs -Newsletter -Convenciones	3
Yii	5.4.0 o superior	-MySQL -MariaDB -SQLite -PostgreSQL -CUBRID -Oracle -MSSQL	-PHP -Programación Orientada a Objetos	-Excelente -Detallada -Actualizada	- Foros -Blogs -Wiki -Redes sociales	2.0
Symfony	7.1.3	-MySQL -PostgreSQL -SQLite -Oracle -MSSQL	-PHP -Programación Orientada a Objetos -Uso de consola	-Completa -Comprensiva	-Red propia: SensioLabs Connect	4.1
CakePHP	5.6 o superior	-MySQL -PostgreSQL -Microsoft SQL Server -SQLite	-PHP -Programación orientada a objetos	-Deficiente -Detallada -Actualizada -Estructurada	- Foros -Chats -Wiki -Comunidad	3.5
Phalcon	5.5 o superior	-MySQL -PostgreSQL -SQLite	-PHP	-Excelente -Comprensiva -Organizada	-Poca comunidad, pero activa	3.2

						
Laravel 	PHP 7.1.3	-MySQL -PostgreSQL -SQLite -SQL Server	-PHP	-Excelente -Detallada -Con ejemplos -Fácil de entender	-Creciente y activa -Foros -Github -Bolsa de trabajo	5.6

Tabla 1.1 Características de los *frameworks* para PHP.

Estos son algunos puntos que se recomiendan considerar antes elegir un *framework*:

- Soporte para base de datos
- Comunidad y soporte
- Documentación
- Arquitectura

En este escrito nos enfocaremos solamente al uso de **Phalcon,** ya que como parte de mi servicio social fue requerido para realizar un sitio web institucional del cual hablaré en el capítulo 3.

1.2 Phalcon: Definición y características

Phalcon es un *framework* reciente de tipo *Open Source** para PHP basado en el patrón de diseño MVC. Esta desarrollado como una extensión del lenguaje C, esto hace que se mejore el rendimiento y se reduzca el consumo de recursos para cada petición que se realice.

- Fue desarrollado en 2012 por Andrés Gutiérrez y un grupo de colaboradores
- Su nombre se debe a la unión de PHP y *Falcon* (halcón en inglés)
- En su versión 0.3.5 se incluye un mapeo objeto-relacional (ORM**) escrito en C
- En su versión 0.5.0 se introdujo PHQL (lenguaje de consulta de **Phalcon**)
- En su versión 0.6.0 se introdujo VOLT (motor de plantillas para las vistas)

Entre sus características internas se encuentran:

*Los sistemas *Open Source* significan que son de código abierto y de dominio público.

**ORM (*Object-Relational mapping*) es una técnica para trabajar con una base de datos relacional utilizando la programación orientada a objetos.

- Menor costo de memoria y CPU en comparación con otros *frameworks* (figura 1.8).
- Sus componentes son independientes, con esto se puede trabajar solo con las partes que necesites del *framework*.
- El código es compilado y no interpretado. Como sabemos PHP es un lenguaje interpretado, es decir, que el código fuente es traducido a lenguaje maquina por el intérprete instrucción

  El código en **Phalcon** esta compilado previamente por una plataforma y procesador especifico.
- Utiliza la estructura de archivos MVC.
- Hace uso de inyección de dependencias (patrón de diseño donde se suministran objetos a una clase).
- Maneja servicios para escribir aplicaciones REST (*Representational State Transfer*) con la finalidad de que los recursos creados puedan ser consumidos.
- Carga clases PHP de forma automática utilizando el estándar PSR-4 (Estándar de autocarga de clases).
- Enrutamiento fácil y sencillo.

Entre sus características para el manejo de datos se encuentran:

- Las consultas a la base de datos se hacen de manera más eficiente, esto se debe a su ORM escrito en C.
- Su lenguaje PHQL (*PhalconQL*) es orientado a objetos y permite realizar consultas utilizando el lenguaje SQL (lenguaje de consulta estructurada).
- Mantiene un manejo de caché para acceder de manera más rápida a datos utilizados de forma frecuente.

Entre sus características para el manejo de vistas y *Front-end* se encuentran:

- Utiliza el motor de plantillas VOLT escrito en *Zephir/C**, permite crear vistas de forma más fácil.
- Permite traducir aplicaciones a diferentes idiomas.
- Utiliza *Tags*** para crear formularios que permiten ahorrar código HTML.

* *Zephir* es un lenguaje de alto nivel para crear extensiones de PHP. Dichas extensiones son exportadas a código C y después se compilan.

**Los *Tags* son etiquetas que encapsulan un comportamiento o una porción de código generalmente HTML.

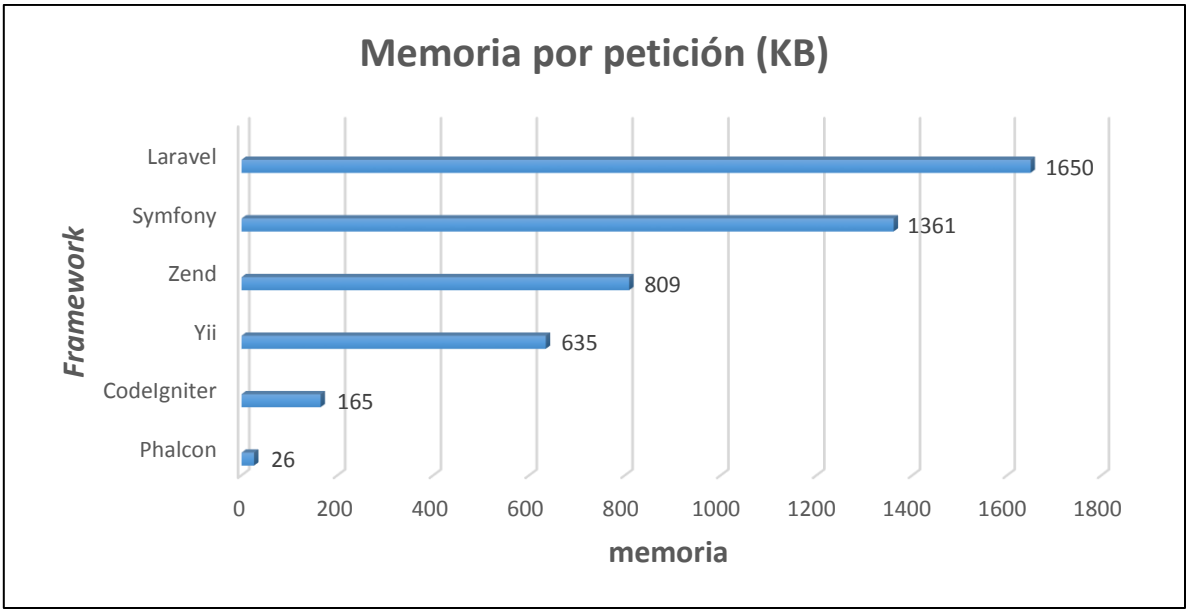


Figura 1.8 El uso de memoria por cada solicitud, es menos a comparación con otros marcos de trabajo.

1.2.1 Comparación de Phalcon con otros frameworks

La siguiente tabla hace una comparación de **Phalcon** con los *frameworks* más populares para el desarrollo en PHP.

Framework	Tipo de proyecto	Lenguaje	Escalabilidad	Rendimiento
Yii	Proyectos de gran escala como foros, portales, RESTful web services	Escrito en PHP	Pequeños y medianos proyectos	Comparativamente bajo
Laravel	Aplicaciones web debido a que es sofisticado	Escrito en PHP siguiendo el patrón MVC	Alta para todo tipo de proyectos	Alto, pero lento a comparación a otros
Phalcon	Usado para una gran variedad de proyectos	Incluye PHP y C	Bueno para proyectos medianos	Alto rendimiento

Tabla 1.2 Phalcon vs otros *frameworks.*

1.3 Instalación en Windows utilizando XAMPP

Para poder utilizar **Phalcon** en nuestro equipo necesitamos de un servidor que ejecute código en PHP, en este caso utilizaremos XAMPP, el entorno en que se realiza esta guía básica de instalación es en el sistema operativo Windows 7, pero también es compatible con Windows 8 y 10.

XAMPP es un programa *todo en uno* que contiene *Apache, **MariaDB y ***Perl, además instala phpMyAdmin que es una aplicación web que sirve para administrar las bases de datos de MySQL (figura 1.9).[6]

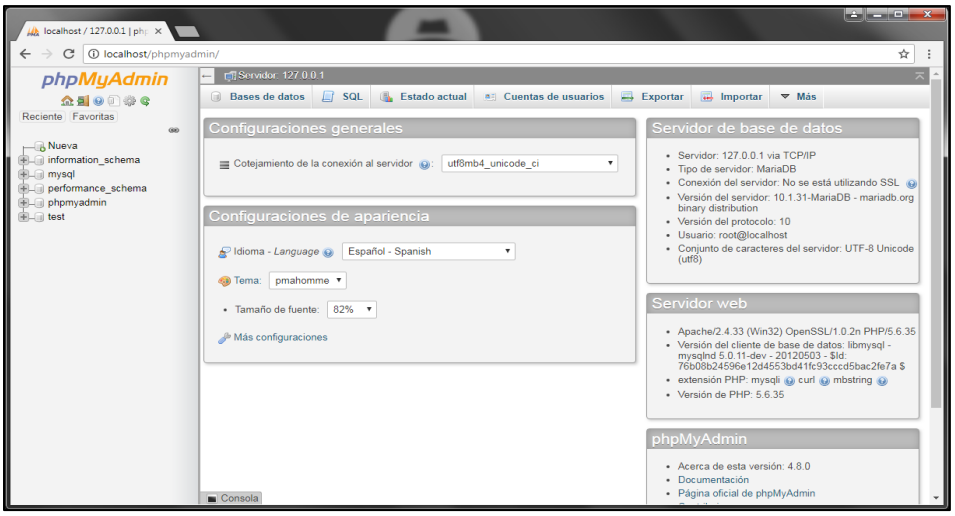


Figura 1.9 phpMyAdmin. Una vez iniciado XAMPP se puede acceder a esta consola desde el navegador: *localhost/phpmyadmin.*

Para poder instalar XAMPP simplemente necesitamos ir con nuestro navegador a la dirección *https://www.apachefriends.org/es/index.html*. En este caso se descargó la versión 5.6.35 que contiene:

- Apache 2.4.33
- MariaDB 10.1.31
- PHP 5.6.35
- phpMyAdmin 4.8.0
- OpenSSL 1.0.2
- Filezilla FTP Server 0.9.41
- Tomcat 7.0.56

* Apache es un servidor web HTTP de código abierto.

**MariaDB es un sistema de gestión de bases de datos que surge a partir de MySQL.

***Perl es un lenguaje de programación similar a C.

1.3.1 Instalación de XAMPP

1. Dirigirse a la carpeta donde se descargó el archivo de instalación y ejecutarlo. Este tiene un nombre similar a "*xampp-win32-5.6.35-0-VC11-installer".*
2. En la siguiente ventana se selecciona la dirección donde se instalará XAMPP. Es recomendable dejarla por defecto.

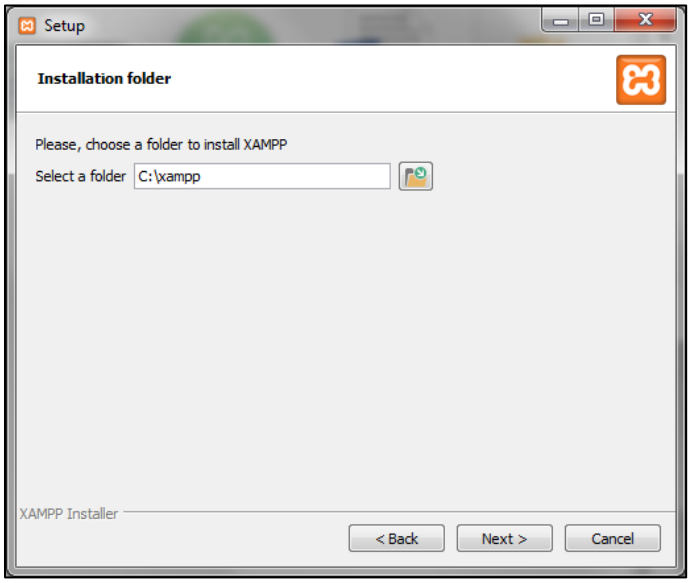


Figura 1.10 Selección del folder de instalación de XAMPP.

3. Una vez dada la ruta, el programa instalará todos los archivos necesarios, cuando termine mostrará una ventana informando que la instalación se ha hecho de manera exitosa.

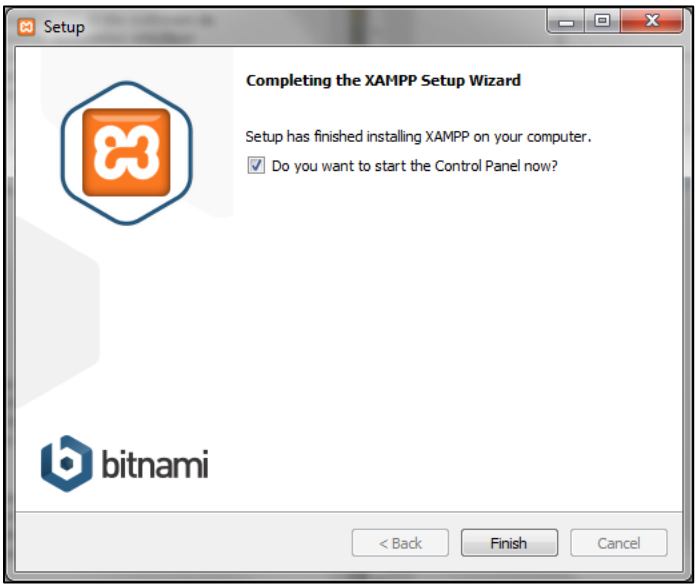


Figura 1.11 Finalización de la instalación de XAMPP.

4. Si XAMPP no se abre automáticamente al cerrar el instalador nos dirigiremos a Windows -> todos los programas, ya ubicados ahí buscamos la carpeta XAMPP, después procederemos a seleccionar “XAMPP Control Panel”.

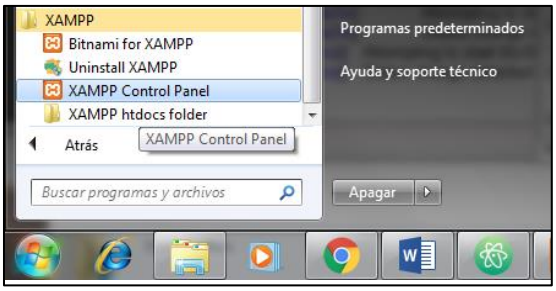


Figura 1.12 Selección de XAMPP Control Panel.

5. En la sección *Actions,* damos clic al botón *Start* del módulo Apache y esperar a que cambie a color verde. ¡Listo! ya tenemos corriendo Apache ¿fácil no?

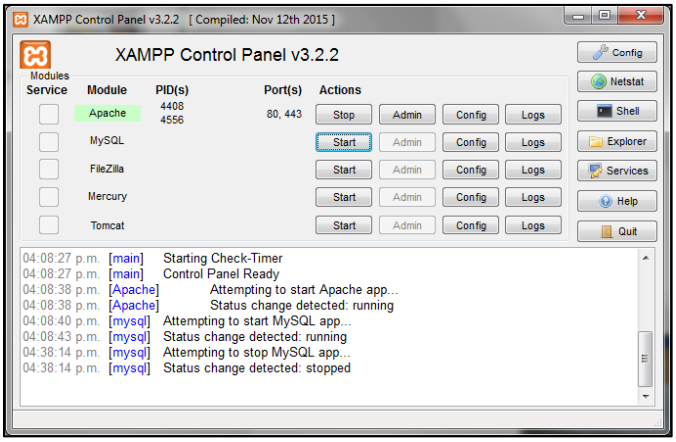


Figura 1.13 Corriendo Apache desde XAMPP Control Panel.

6. Para verificar que no hubo errores no dirigimos con el navegador a la dirección: *localhost*. Debemos ver la página inicial de XAMPP (**Figura 1.14**).

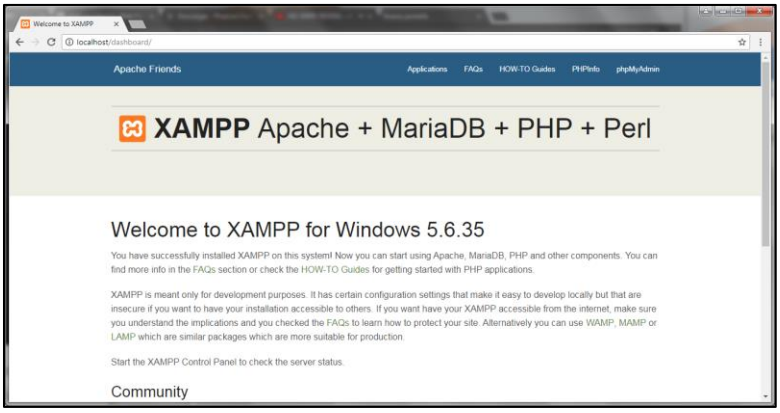


Figura 1.14 Página principal de XAMPP.

- La carpeta donde crearemos todos nuestros proyectos se encuentra en *C:\xampp\htdocs*

1.3.2 Instalación de Phalcon

1. Debemos dirigirnos a la página oficial de **Phalcon:** *https://phalconphp.com/es/* .Una vez ahí, seleccionamos “Descargar” en el menú de navegación.

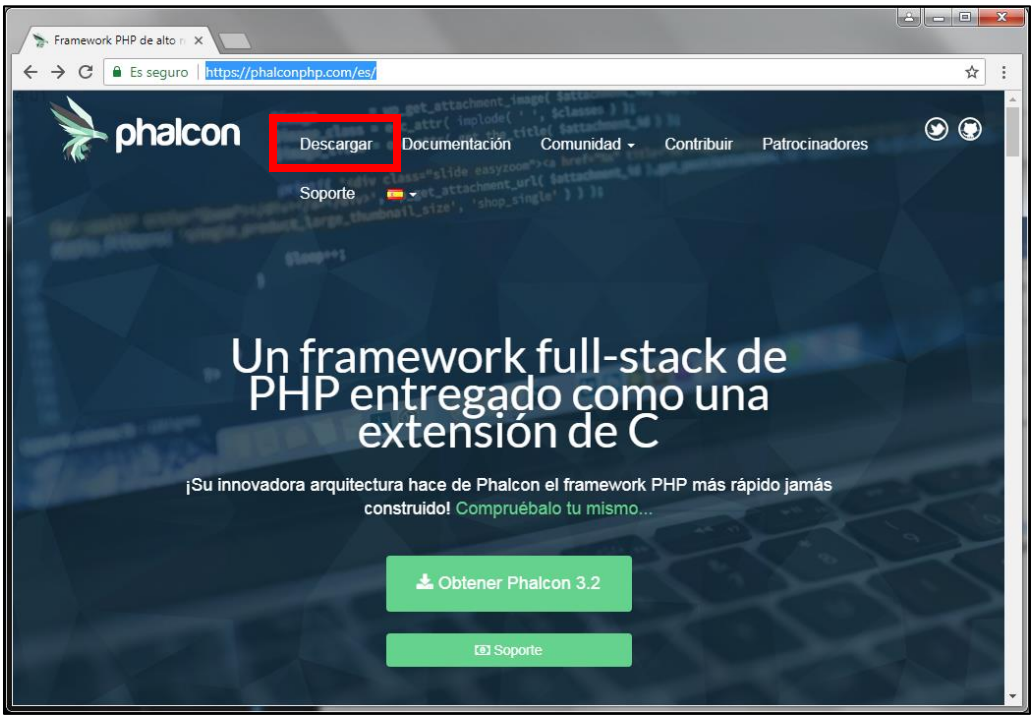


Figura 1.15 Página oficial de **Phalcon.**

2. Nos cambiamos a la pestaña **Windows DLLs**. Una vez ahí aparecerá un enlace a **Github* donde descargaremos el archivo ***dll* que se adapte a XAMPP y a tu versión de PHP. En este caso, descargamos el Zip llamado *phalcon_x86_vc11_php5.6.0_3.3.2*.

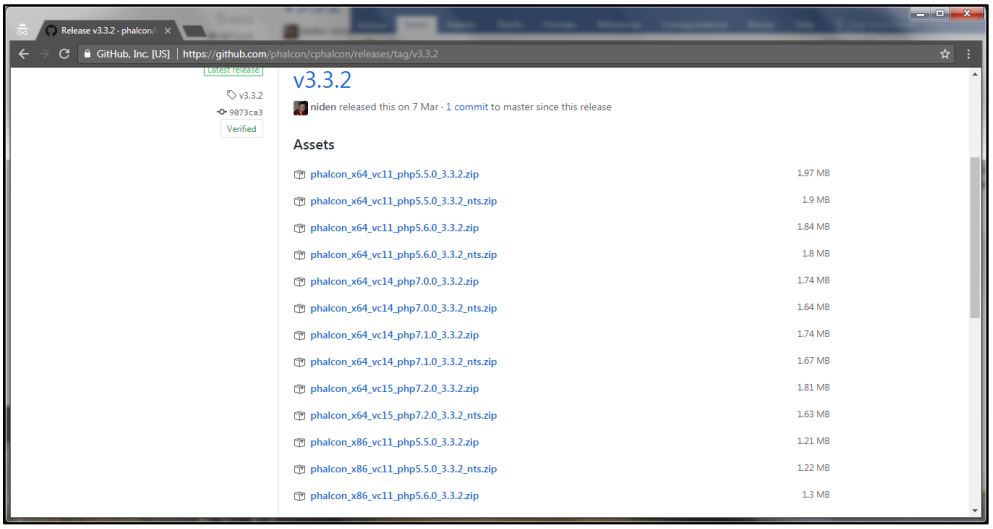


Figura 1.16 Repositorio del archivo *dll* para Windows en sus arquitecturas x64 y x86

3. Debemos mover el *dll* a las extensiones de PHP, para esto descomprimimos el Zip que descargamos y copiamos *php_phalcon.dll* en la siguiente ruta: *C:\xampp\php\ext*.

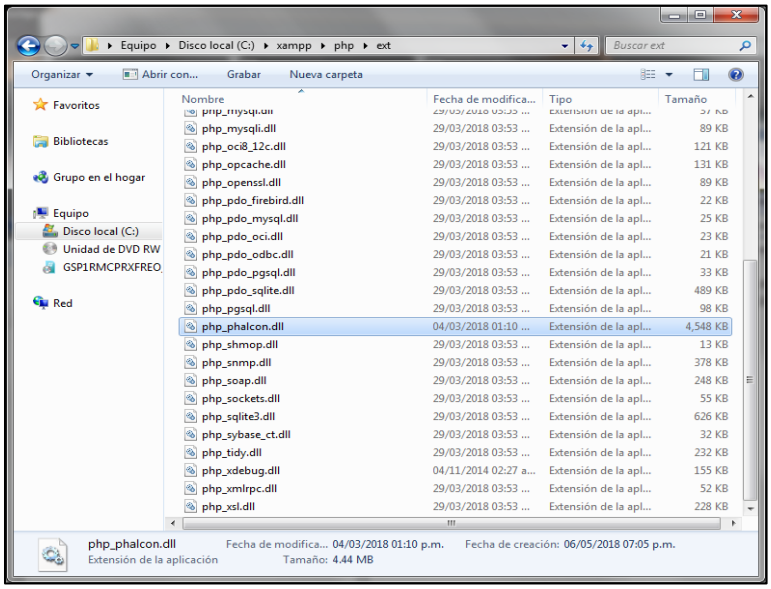


Figura 1.17 Ruta donde debe colocarse el archivo *dll.*

**Github* es un repositorio en la nube para proyectos y que usa el control de versiones *Git.*

** Un *dll* (*Dinamic Link Library*) son un conjunto de instrucciones que se ejecutan a petición de una aplicación.

4. Por ultimo modificamos el archivo *php.ini* ubicado en C:*xampp**php*. Buscamos la sección donde están las extensiones y agregamos lo siguiente:

```
extension=php_phalcon.dll
```

- Puedes utilizar WordPad o cualquier otro editor de texto que desees.

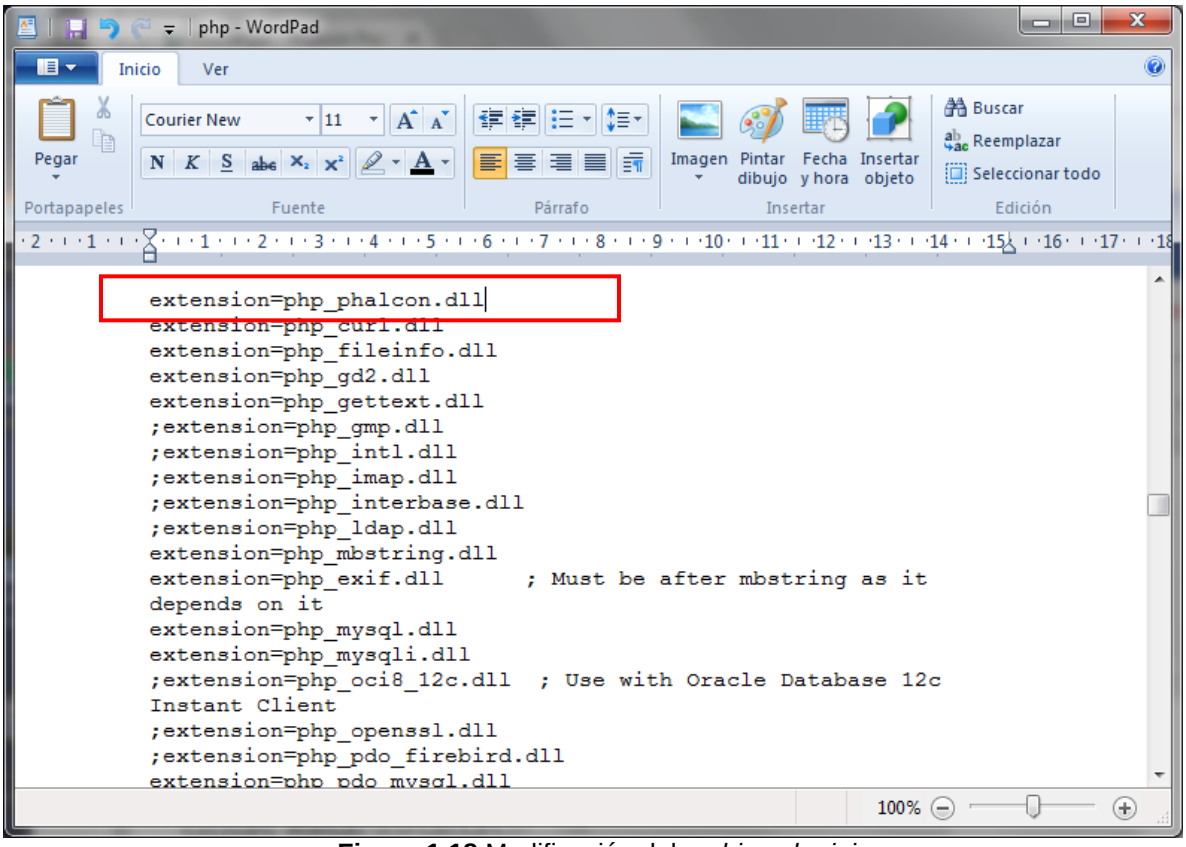


Figura 1.18 Modificación del *archivo php.ini*

5. Solo falta reiniciar XAMPP o detener el Servicio de **Apache** y volverlo a iniciar. Después debemos dirigimos con el navegador a *http://localhost/dashboard/phpinfo.php*. Una vez dentro buscamos que **Phalcon** sea reconocido como una extensión para PHP, como se muestra en la **Figura 1.19**.

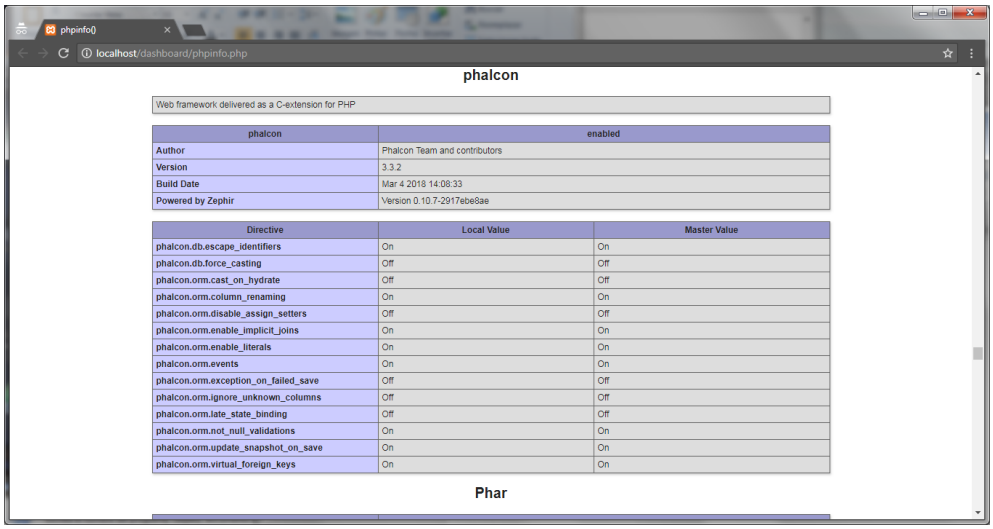


Figura 1.19 Phalcon es reconocido como una extensión.

1.3.3 Instalación de herramientas de desarrollo o Developer Tools

Las *herramientas de desarrollo* son *scripts* que permiten agregar componentes básicos (como vistas, controladores, modelos, etc.) en una aplicación con un simple comando en consola, de esta manera ahorraremos tiempo para crear componentes.

Para poder instalarlo primero debemos descargar la carpeta que contiene estas herramientas. En el navegador nos dirigimos a *https://github.com/phalcon/phalcon-devtools* y descargamos el archivo Zip.

1. Descomprimimos el Zip en cualquier ruta de del disco local por ejemplo *C:\phalcon-devtools-master.*

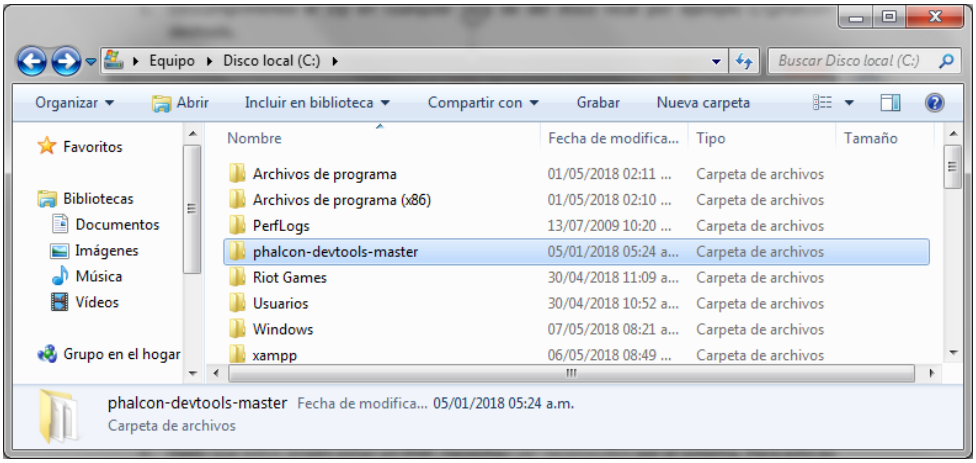


Figura 1.20 Se descomprime el Zip en la raíz del disco local C .

2. Dado que estos *scripts* están en PHP, necesitan ser reconocidos por el sistema. Para esto es necesario copiar la ruta donde se encuentra el archivo *php.exe* y agregarla como variable de entorno PATH. En XAMPP, *php.exe* está ubicado en *C:\xampp\php*.
3. Para agregar una variable de entorno en Windows nos vamos a Panel de control -> Sistema y seguridad -> Sistema. Abrimos Configuración avanzada del sistema.

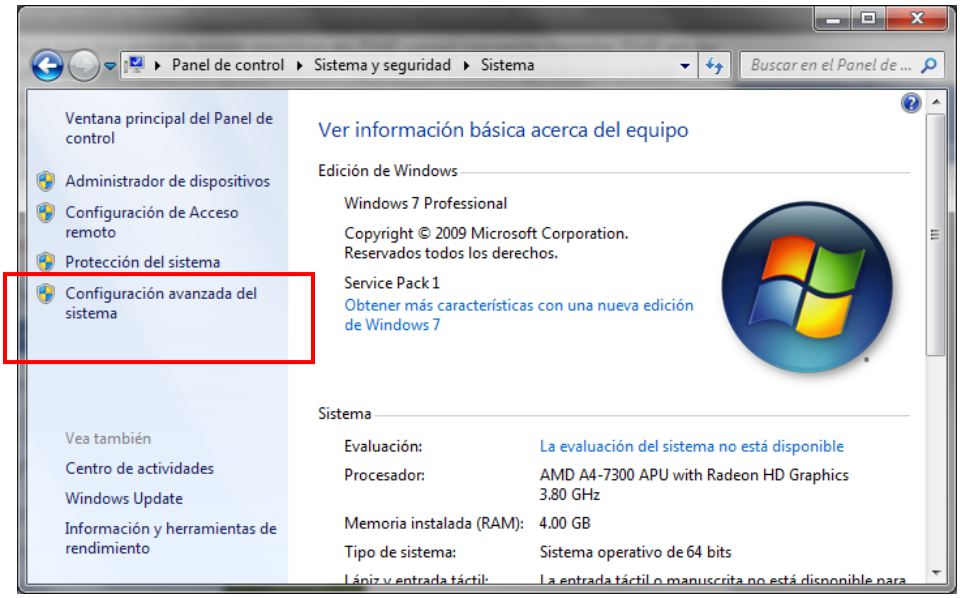


Figura 1.21 Ubicación de “Configuración avanzada del sistema”.

4. Damos clic en variables de entorno

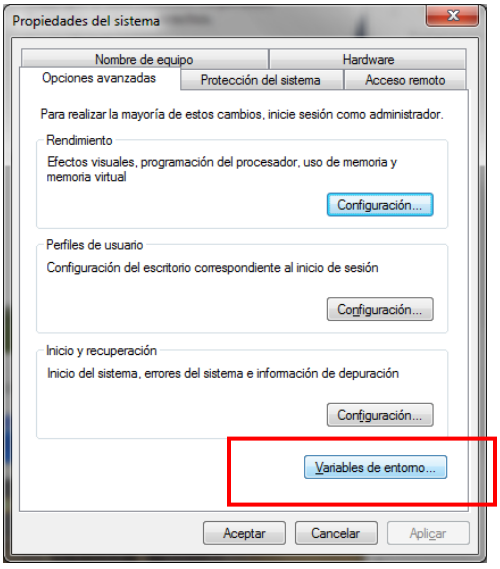


Figura 1.22 Ubicación de las “variables de entorno”.

5. Colocamos la ruta donde se encuentra PHP (*C:\xampp\php*) y la ruta donde se descomprimieron las herramientas de desarrollo (*C:\phalcon-devtools-master*) en la variable *Path* ubicada en variables del sistema. Antes de pegar la ruta ponemos ";" Aceptamos y listo.

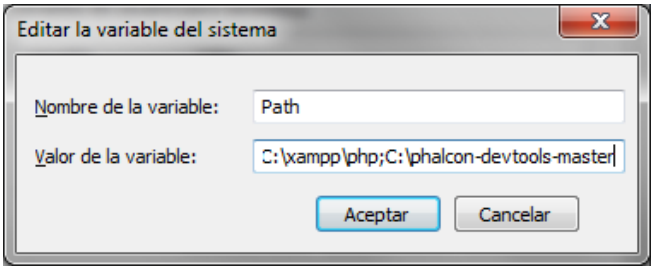


Figura 1.23 Edición de la variable *Path*.

6. Para saber si las herramientas de desarrollo fueron instaladas correctamente ingresamos a símbolo del sistema presionando las teclas Windows (Win) + R y escribimos "cmd".

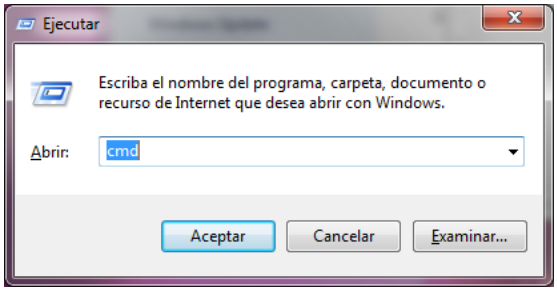


Figura 1.24 Apertura de la herramienta ejecutar de Windows.

7. Una vez dentro de la consola ejecutamos el comando "**Phalcon**" y deberá aparecer la versión instalada de las *herramientas de desarrollo,* así como sus comandos disponibles.

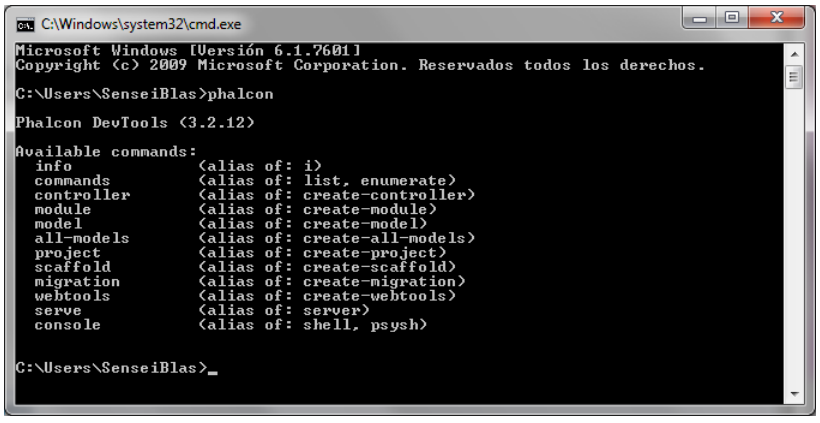


Figura 1.25 Lista de comandos en **Phalcon.**

1.4 Estructura de una aplicación en Phalcon

Como hemos estado viendo, el patrón de arquitectura utilizado por **Phalcon** es el MVC. pero ¿de qué carpetas y archivos están compuesto un proyecto?

Cuando creamos un proyecto en **Phalcon** debemos generar una estructura de carpetas, subcarpetas y ficheros iguales a los mostrados en la siguiente figura :

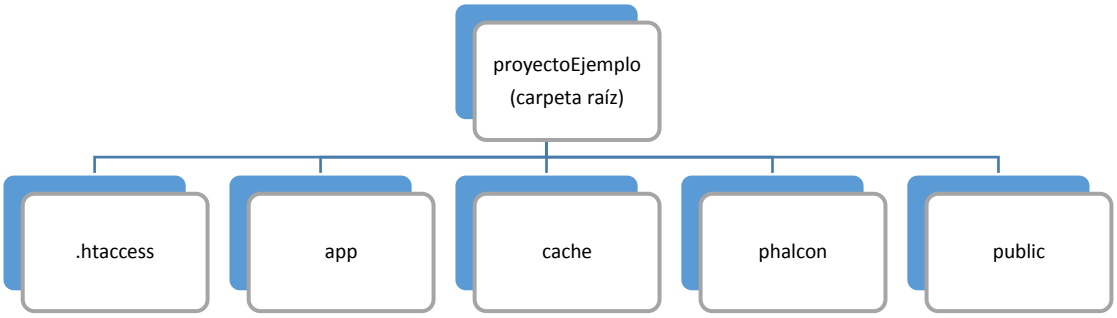


Figura 1.26 Estructura de un proyecto en **Phalcon.**

Una vez creado un proyecto, su estructura de carpetas, subcarpetas y archivos se verá de la siguiente forma:

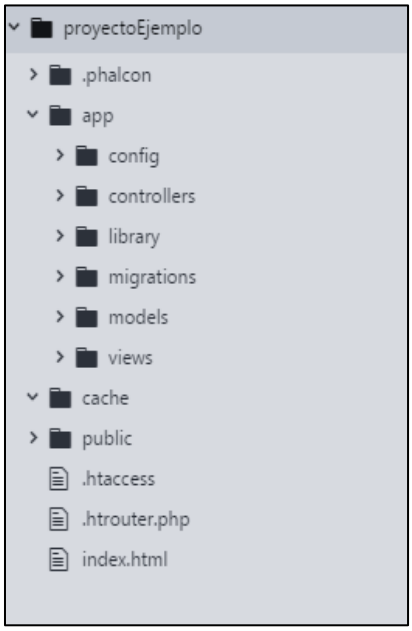


Figura 1.27 Carpetas, subcarpetas y archivos de proyecto en **Phalcon.**

- Carpeta *app*. Contiene los archivos y carpetas fundamentales para el correcto funcionamiento de nuestra aplicación. A su vez cuenta con subcarpetas como son:

  - *config*. Incluye archivos que manejan la configuración de la aplicación: configuraciones de conectividad a la base de datos, configuración de la URL raíz, bibliotecas de terceros, servicios que se pueden incluir, etc.
  - *controllers*. En esta carpeta se colocarán los controladores que agreguemos.
  - *library*. Aquí se importan las bibliotecas creadas por terceros o las que contiene **Phalcon**.
  - *migrations*. Incluimos archivos para la migración de datos de esta manera podremos cambiar o modificar nuestra base de datos de manera estructurada y ordenada.
  - *models*. Carpeta donde se colocan los modelos creados, estos gestionan y manipulan las tablas de nuestra base de datos.
  - *views.* Sirve para insertar las vistas que proporcionan al usuario final información de forma gráfica.
- Carpeta cache. Almacena datos en la cache de **Phalcon** con el fin de mejorar el rendimiento.
- Carpeta public. Incluye hojas de estilo, archivos JavaScript*, imágenes, metadatos, etc.
- ".htaccess". Es un archivo de configuración que se carga tan pronto como se inicia el servidor. Por ejemplo, se puede configurar un sitio web para que esté disponible solo para direcciones IP específicas.

1.5 Creación de una aplicación

A continuación, crearemos una aplicación desde cero que contiene un formulario para poder registrar alumnos. Esto lo haremos con el fin de aprender a crear aplicaciones con **Phalcon** y conocer el papel que juegan los archivos y carpetas que lo conforman.

1.5.1 Creando una base de datos y una tabla para guardar alumnos registrados

Para poder crear una base de datos nos dirigimos desde el navegador a la dirección *http://localhost/phpmyadmin/* una vez dentro en el damos clic en "Nueva" del menú lateral izquierdo (**Figura 1.28**).

* JavaScript es un lenguaje de programación interpretado utilizado en el lado del cliente.

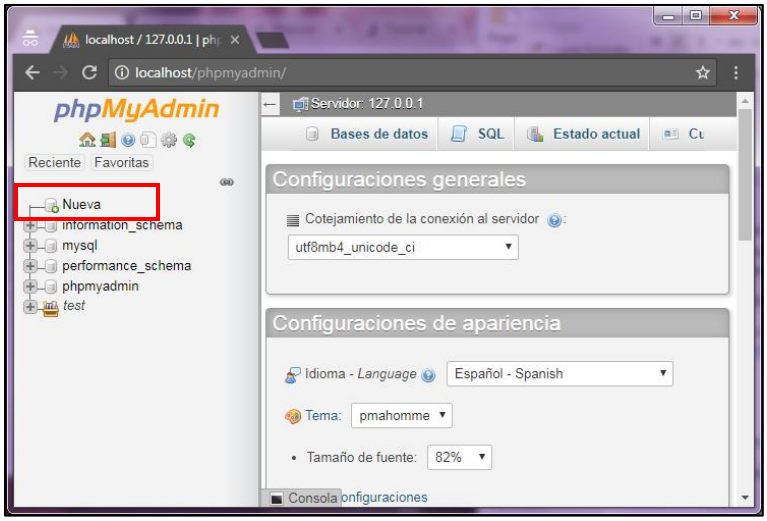


Figura 1.28 Creación de una nueva base de datos**.**

En la siguiente ventana debemos llenar los campos pertenecientes al nombre de la base de datos (en este caso la llamaremos “curso-db”) y el cotejamiento (codificación) será *utf8_spanish_ci*.

Figura 1.29 Nombre y cotejamiento de la nueva base de datos.

Una vez creada la base de datos, se mostrará una nueva ventana con un formulario para crear una tabla sencilla, en este ejemplo estamos haciendo el registro de alumnos a un curso, por lo tanto, crearemos una tabla “alumno” que cuente con las siguientes columnas**:**

Nombre	Tipo	Longitud de caracteres	Llave primaria, autoincrementable
id	Int	11	☑
nombre	Varchar	20	☒
matricula	Varchar	13	☒
edad	Int	11	☒

Tabla 1.3 Campo de la tabla “alumno”.

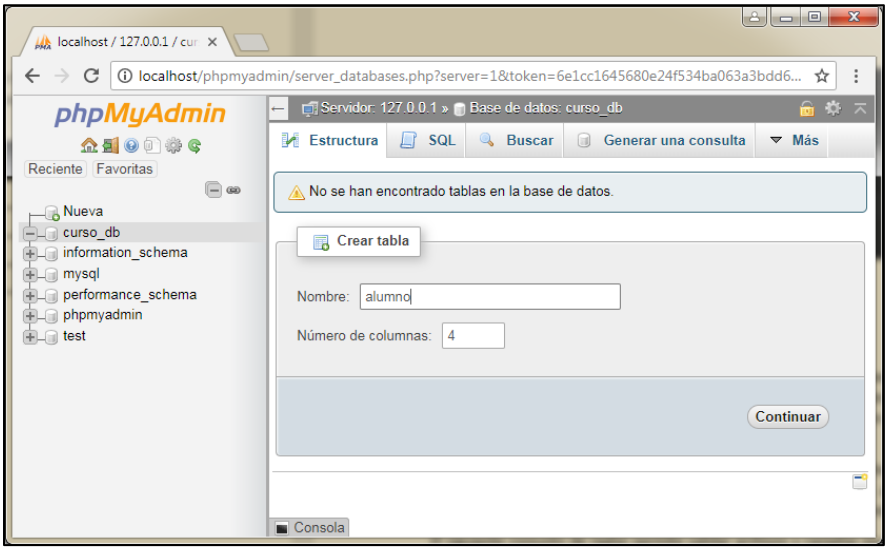


Figura 1.30 Creación de la tabla “alumno”.

En la siguiente ventana debemos indicar en cada columna el nombre, tipo de dato, longitud de caracteres o si es autoincrementable con base a la **tabla 1.3**. El llenado de datos queda de la siguiente manera:

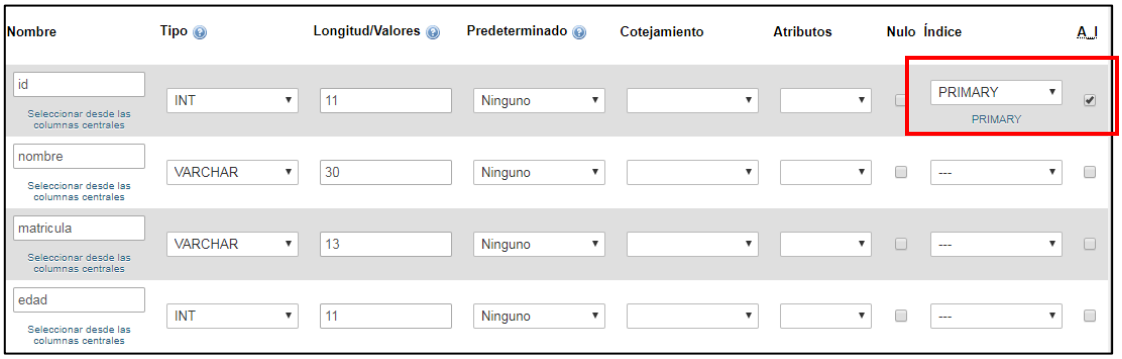


Figura 1.31 asignando *Primary Key* al campo id.

Listo ya tenemos nuestra base de datos “curso_db” creada, así como la tabla “alumno”.

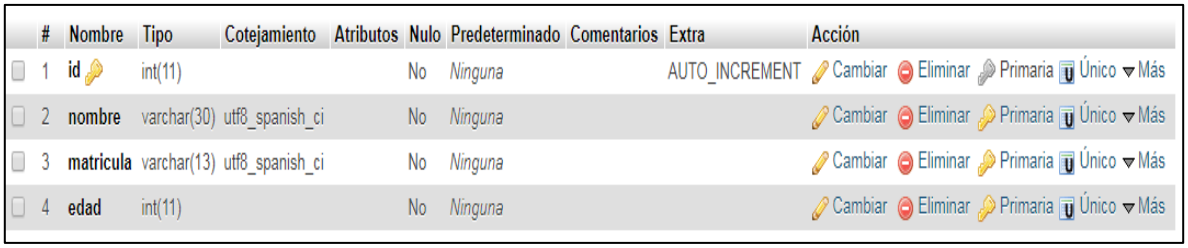


Figura 1.32 Campos creados para la tabla “alumnos”.

1.5.2 Creando la estructura de carpetas

La estructura de carpetas y archivos propuestos son los siguientes:

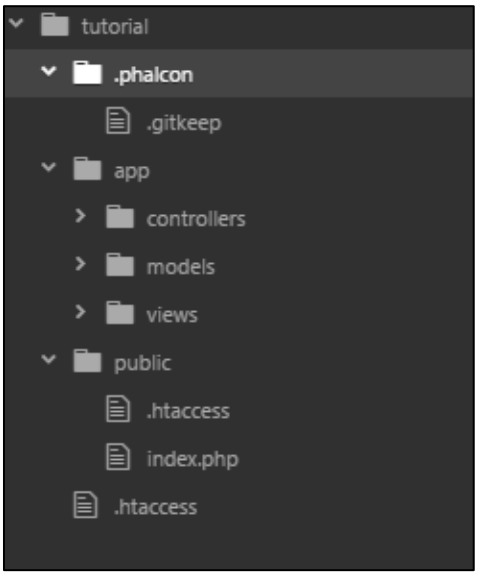


Figura 1.33 Carpetas, subcarpetas y archivos para crear un proyecto en **Phalcon** desde cero.

- Tutorial es la carpeta raíz del proyecto.

1.5.3 Creando URL´s amigables

Una URL amigable describe el contenido de la página web a la que se está visitando, de tal manera sea más fácil de usar y recordar por el usuario permitiendo también comunicar una página en concreto a otras personas.

https://www.dominio.mx./buscar.php&id=6/ Esta no es una URL amigable ☒

https://www.dominio.mx./alumno/angel Esta si es una URL amigable ☑

Las URL´s amigables son mejores para SEO (*Search Engine Optimization*) el cual nos sirve para optimizar un sitio web y mejorar la posición en los resultados obtenidos por los motores de búsqueda. Es un buen consejo para ver tu sitio en las primeras páginas de un navegador.

En esta ocasión utilizaremos el módulo *rewrite* para Apache. Esta serie de reglas debemos escribirlas en un archivo:

- Ubicación: ***tutorial/***
- Nombre: ".htaccess"

```
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule  ^$ public/    [L]
    RewriteRule  ((?s).*) public/$1 [L]
</IfModule>
```

Figura 1.34 Contenido del archivo ".htaccess".

mod_rewrite permite crear direcciones URL de manera dinámica de modo que sean más legibles y fáciles de recordar. Cuenta con funciones para evitar el acceso a nuestro sitio web a una determinada dirección IP, permite redireccionar nuestro dominio a otro sitio y también evitar que otros sitios utilicen nuestras imágenes (*hotlinking*).

Todas las peticiones al proyecto serán dirigidas a la carpeta *public/* de forma que este haga el papel de directorio raíz. Esto asegura que los archivos pertenecientes al *back-end* del proyecto permanezcan ocultos a los usuarios finales y evitar problemas de seguridad.

El siguiente conjunto de reglas permite validar archivos y carpetas existentes, todo los demás lo redirecciona al Bootstrap de la aplicación. Estas reglas debemos escribirlas en un archivo con:

- Ublicacion: *tutorlal/public*
- Nombre: ".htaccess"

```
AddDefaultCharset UTF-8
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^((?s).*)$ index.php?_url=/$1 [QSA,L]
</IfModule>
```

Figura 1.35 Contenido completo del archivo ".htaccess".

1.5.4 Creando el fichero Bootstrap

Este archivo (llamado así en la documentación de **Phalcon,** no confundirlo con el *framework*) sirve como base para poder arrancar nuestra aplicación ya que con ella podemos inicializar y configurar componentes, también podemos definir y cambiar el comportamiento de nuestra aplicación.

- Ubicación: *tutorial/public*
- Nombre: "index.php"

```php
<?php

use Phalcon\Db\Adapter\Pdo\Mysql as DbAdapter;
use Phalcon\Di\FactoryDefault;
use Phalcon\Loader;
use Phalcon\Mvc\Application;
use Phalcon\Mvc\Url as UrlProvider;
use Phalcon\Mvc\View;

//definición de rutas absolutas en constantes para solicitar recursos
define('BASE_PATH', dirname(__DIR__));
define('APP_PATH', BASE_PATH . '/app');

// se define un autocargador
$loader = new Loader();
$loader->registerDirs(
    array(
        APP_PATH . '/controllers/',
        APP_PATH . '/models/'
    )
)->register();
```

```php
// Se crea un inyector de dependencias
$di = new FactoryDefault();

// Se establece el componente de las vistas
$di['view'] = function() {
    $view = new View();
    $view->setViewsDir(APP_PATH . '/views/');
    return $view;
};

// Se establece una url base
$di['url'] = function() {
    $url = new UrlProvider();
    $url->setBaseUri('/');
    return $url;
};

// Se establece la conexión con la base de datos
$di['db'] = function() {
    return new DbAdapter(array(
        "host"     => "127.0.0.1",
        "username" => "root",
        "password" => "",
        "dbname"   => "curso_db"
    ));
};

// se atienden las peticiones
try {
    $application = new Application($di);
    echo $application->handle()->getContent();
} catch (Exception $e) {
     echo "Exepción: ", $e->getMessage();
}
```

Figura 1.36 Contenido del archivo “index.php”.

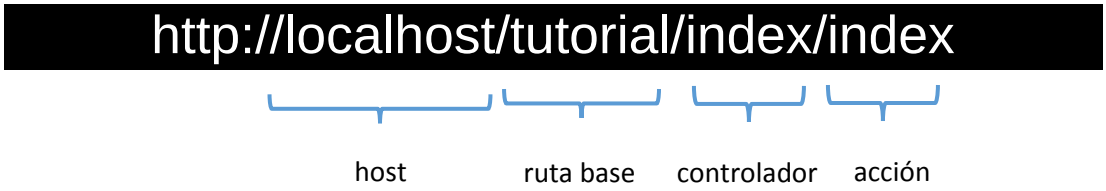
- En las líneas 11 y 12 se definen las constantes, como la ruta base (*tutorial/*) y el directorio app (*tutorial/app*).
- *Phalcon\Di\FactoryDefault* nos permite crear un inyector de dependencias, es decir, carga automáticamente todos los servicios que nos proporciona **Phalcon.** En la línea 23 creamos un objeto ($id) de esta clase**.**
- *Phalcon\Db\Adapter\Pdo\Mysql as DbAdapter* nos permite especificar la configuración para la conexión a la base de datos MySQL. En las líneas 40-47 asignamos a $id lo que retorna la clase *DbAdapter*, el cual necesita como parámetros de inicialización los datos que usamos para establecer una conexión a base de datos como son: *host, username, password* y *dbname*.
- *use Phalcon\Loader* esta clase nos va a servir para cargar clases de nuestro proyecto de manera automática. En las líneas 15-21 creamos el objeto de la clase *Loader* y después hacemos uso de dos métodos:
    1. *registerDirs.* Registra la ruta donde están los controladores (*tutorial/app/controllers*) y modelos (*tutorial/app/models*).
    2. *Register.* Guarda los directorios registrados.
- *Phalcon\Mvc\Url as UrlProvider* se encarga la gestión de todas las URL's de nuestro proyecto. En las líneas 33-37 se asigna a $di nuestra URL base, es decir, *tutorial/.*
- *Phalcon\Mvc\View* clase que nos permite trabajar con las vistas. En las líneas 27-31 a nuestro objeto $id se le asigna $view el cual es retornado por una función anónima.
  $view contiene la ruta donde se encuentran almacenadas las vistas (tutorial/app/views).
- *Phalcon\Mvc\Application* inicializa el entorno de las solicitudes, enruta la solicitud entrante y luego despacha a cualquier acción nueva, agrega todas las respuestas y las devuelve cuando el proceso se completa. En las líneas 50-55 se despachan todas las peticiones provenientes de nuestro proyecto. Notemos que la clase *Application* recibe como parámetro $di (objeto que atiende todos nuestros servicios).

1.5.5 Creando un controlador para la página principal (index)

De manera predeterminada **Phalcon** busca un controlador con el nombre *IndexController* que atiende a la página inicial de nuestro proyecto (*localhost/tutorial*).

- Cuando ingresamos a nuestro proyecto, **Phalcon** divide la URL buscando el controlador y su método correspondiente.
- Si no se ingresa un controlador por defecto se cargará *indexController.*
- Si se ingresa el controlador, pero no la acción, cargara la acción *indexAction.*

Por ejemplo, supongamos que queremos ir a la página principal de nuestro proyecto *localhost/tutorial* esto es equivalente a *localhost/tutorial/index/index*, donde el primer *index* es el controlador y el segundo es la acción.



Debemos considerar lo siguiente a la hora de crear un controlador:

- Los controladores deben de llevar el sufijo *Controller*
- La primera letra en el nombre del controlador es mayúscula
- Un ejemplo de nombre de un controlador es *WelcomeController*
- Las acciones deben llevar el sufijo *Action*

Una vez explicado esto, vamos a crear un controlador y una acción:

- Ubicación: *tutorial/app/controllers*
- Nombre: “IndexController.php”

```php
<?php

use Phalcon\Mvc\Controller;

class IndexController extends Controller
{

  public function indexAction()
  {

  }

}
```

Figura 1.37 Contenido del archivo “IndexController.php”.

Hemos creado un controlador (clase “IndexController”) y una acción (método “indexAction”) haciendo uso de la clase *Phalcon\Mvc\Controller.*

1.5.6 Creando una vista

Las vistas son la carta de presentación al usuario final. Son páginas web con código HTML, CSS y en ocasiones PHP si es que se muestra información proveniente del modelo asociado.

Phalcon por defecto buscará una vista con el mismo nombre de la última acción solicitada dentro de un directorio llamado igual al controlador ejecutado. Si la URL es *localhost/tutorial/index/index*, **Phalcon** mostrará la vista (si existe) llamada "index" (ubicada en *app/views/index*) donde el primer "index" de la URL es el controlador que se solicitó.

Sabiendo esto crearemos una vista para la página principal:

Dentro de *views* (*app/views*) crearemos una carpeta con el nombre "index". En esta carpeta que añadimos crearemos la vista:

- Ublicacion: *tutorial/app/views/index*
- Nombre: "index.phtml"

```
<?php

echo "<h1>Curso de Phalcon</h1>";

echo $this->tag->linkTo("tutorial/signup","Haz clic
 aquí para registrarte en el curso!");
```

Figura 1.38 Contenido del archivo "index.phtml".

$this->tag->linkTo("tutorial/signup", "Haz clic aquí para registrarte en el curso!") tiene su equivalente en HTML:

Haz clic aquí para registrarte en el curso!

Si vamos con el navegador a *localhost/tutorial*, veremos lo siguiente:

Figura 1.39 Página principal del proyecto "tutorial" visualizada desde el navegador.

Hasta este momento ya hemos creado un controlador, una acción y una vista correspondientes a la página principal.

1.5.7 Creando un formulario para el registro de alumnos

Primero crearemos la vista donde se mostrará el formulario de registro, para ello añadimos una carpeta llamada “signup” dentro de *views*. Una vez creada colocamos el código correspondiente (**Figura 1.40**).

- Ubicación: *tutorial/views/signup/*
- Nombre: “index.phtml”

```
1  <h1>Formulario de registro</h1>
2
3  <?php echo $this->tag->form('tutorial/signup/register') ?>
4
5    <p>
6      <label for="nombre">Nombre</label>
7      <?php echo $this->tag->textField("nombre") ?>
8    </p>
9
10   <p>
11     <label for="matricula">Matricula</label>
12     <?php echo $this->tag->textField("matricula") ?>
13   </p>
14   <p>
15     <label for="edad">Edad</label>
16     <?php echo $this->tag->textField("edad") ?>
17   </p>
18
19   <p>
20     <?php echo $this->tag->submitButton("Registrar") ?>
21   </p>
22
23 </form>
```

Figura 1.40 Contenido del archivo “index.phtml”.

Haremos uso de *Tags*, que son una forma “abreviada” de utilizar HTML:

- El método *Phalcon\Tag::form()* (Línea 3) recibe sólo como parámetro la URL relativa a un controlador/acción en la aplicación. En pocas palabras, la URL invoca al controlador *SignupController* y la acción *registerAction* que crearemos más adelante.
- En las líneas 7, 12, 16 creamos campos de texto donde almacenamos los datos ingresados por el usuario. Estos campos son: nombre, matrícula y edad.
- En la línea 20 insertamos un botón de tipo *submit* con el nombre “registrar”.

Vayamos con el navegador a la URL a *http://localhost/tutorial/signup* y veremos lo siguiente*:*

Figura 1.41 Formulario creado usando *Tags* visto desde el navegador.

1.5.8 Creando un modelo

Recordemos que un modelo es el encargado de interactuar con nuestra base de datos. Aquí es donde vamos a crear todas las consultas necesarias para atender todas las peticiones del usuario final.

Ahora es momento de hacer el mapeo de nuestra tabla “alumnos”. Para crear un modelo debemos considerar lo siguiente:

- Todos los modelos deben crearse en el directorio *app/models*
- Deben llamarse igual que la tabla que va a ser mapeada
- La primera letra de su nombre debe empezar en mayúscula

Sabiendo estas reglas, vamos a crear el modelo “Alumnos”:

- Ubicación: *app/models*
- Nombre: “Alumno.php”

```php
<?php

use Phalcon\Mvc\Model;

class Alumno extends Model
{

    public $id;
    public $nombre;
    public $matricula;
    public $edad;

}
```

Figura 1.42 Contenido del modelo “Alumno.php”.

- En la línea 3 Importamos la clase *Phalcon\Mvc\Model*
- En la línea 5 creamos la clase *Alumno* (mismo nombre del modelo) que extiende de *Model*. En esta clase definimos los atributos con el mismo nombre (considerando mayúsculas y minúsculas) que las columnas de la tabla *“*alumno” en base de datos*.*

1.5.9 Creando un controlador de registro

Vamos a crear un nuevo controlador con la acción *“*index” y “register*”.* La última acción guardará los datos obtenidos del formulario en la tabla “alumnos” de la base de datos*.*

- Ubicación: *tutorial/app/controllers/*
- Nombre: “SignupController.php”

```php
<?php

use Phalcon\Mvc\Controller;

class SignupController extends Controller
{

    public function indexAction()
    {

    }

```

```
13    public function registerAction()
14    {
15
16      $alumno = new Alumno();
17
18      // almacenando y verificando errores
19      $success = $alumno->save(
20        $this->request->getPost(),
21        array('nombre', 'matricula','edad')
22      );
23
24      if ($success) {
25        echo "Gracias por registrarte al curso!";
26      } else {
27        echo "Ups, creo que hubo un problema al registrarte: "
28        foreach ($user->getMessages() as $message) {
29          echo $message->getMessage(), "<br/>";
30        }
31      }
32
33      $this->view->disable();
34    }
35
36  }
```

Figura 1.43 Contenido del controlador “SignupController.php”.

- En las líneas 3-11 importamos *Phalcon\Mvc\Controller* para después crear la clase *SignupController* que extiende de *Controller*. También creamos la acción “indexAction” que atiende a la vista *views/signup/index.phtml*.
- En las líneas 13-36 creamos la acción “registerAction”, el cual realiza tres funciones:
    1. Crea un objeto de la clase (modelo) “Alumno” que guarda en la tabla correspondiente de la base de datos la información obtenida por el formulario a través de *request->getPost().*
    2. Una vez realizado lo anterior se verifica que no existan errores, de ser así los mostrara en pantalla al usuario.
    3. *$this->view->disable()* sirve para avisar a **Phalcon** que la acción no contiene una vista.

¡Listo ya hemos terminado! Solo nos queda ir a *http://localhost/tutorial/* y probar.

Figura 1.44 Página principal del proyecto.

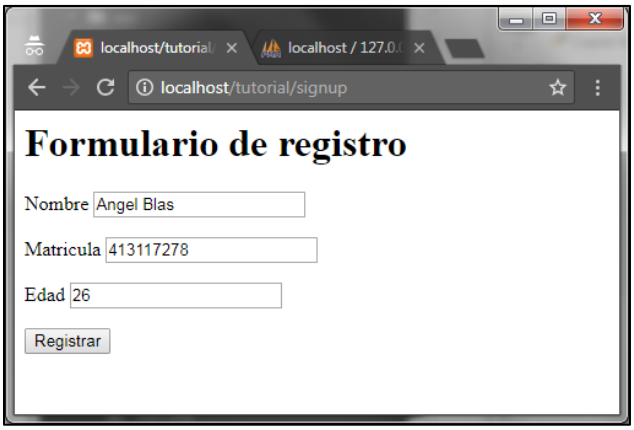


Figura 1.45 Formulario de registro.

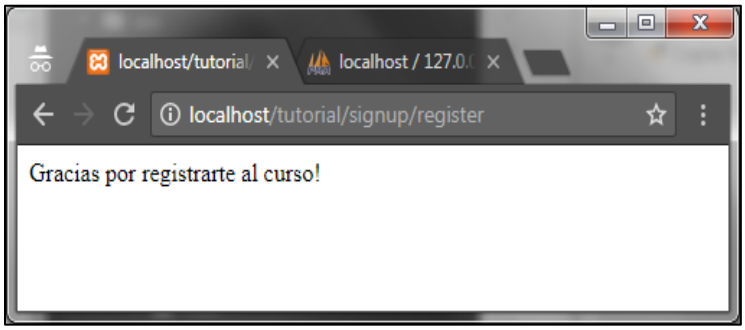


Figura 1.46 Alumno registrado satisfactoriamente.

Verificamos en nuestra base de datos que el registro se haya hecho satisfactoriamente.

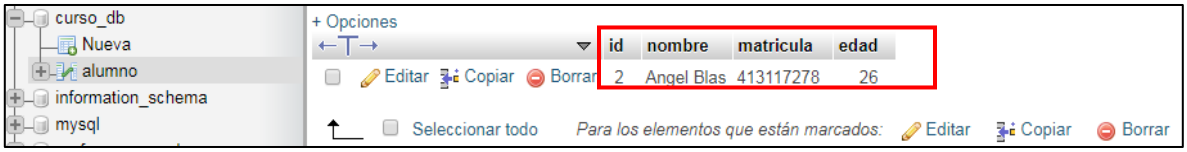


Figura 1.47 Se comprueba que el usuario se haya insertado en la tabla “alumno”.

1.6 Creación de una aplicación utilizando las herramientas de desarrollo

Una vez que vimos como crear un proyecto desde cero, ahora es momento de agilizar aún más su desarrollo. En el tema anterior aprendimos a crear la estructura de carpetas, los archivos de configuración, modelos, controladores y vistas. Ya es momento de simplificar estas tareas, para ello crearemos un nuevo proyecto llamado "proyectoEjemplo" utilizando las *herramientas de desarrollo*.

En XAMPP por defecto todos los proyectos que creamos deben estar almacenados en la ruta *C:\xampp\htdocs*. Desde el símbolo del sistema nos dirigimos a esta ubicación y ejecutamos el siguiente comando:

```
Phalcon create-project <nombre-del-proyecto>
```

- Esta línea nos permite crear un proyecto en **Phalcon**
- Para moverse entre carpetas en el símbolo del sistema utilizamos el comando *dir*

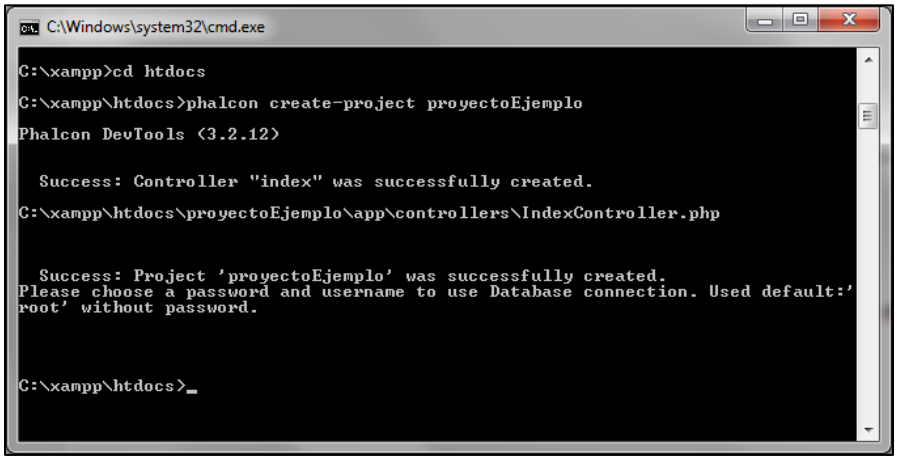


Figura 1.48 Creación de un proyecto con las *herramientas de desarrollo*.

Si el comando se ejecutó sin errores podremos ver que **Phalcon** ha creado un conjunto de archivos y carpetas necesarios para un proyecto (**Figura 1.49**).

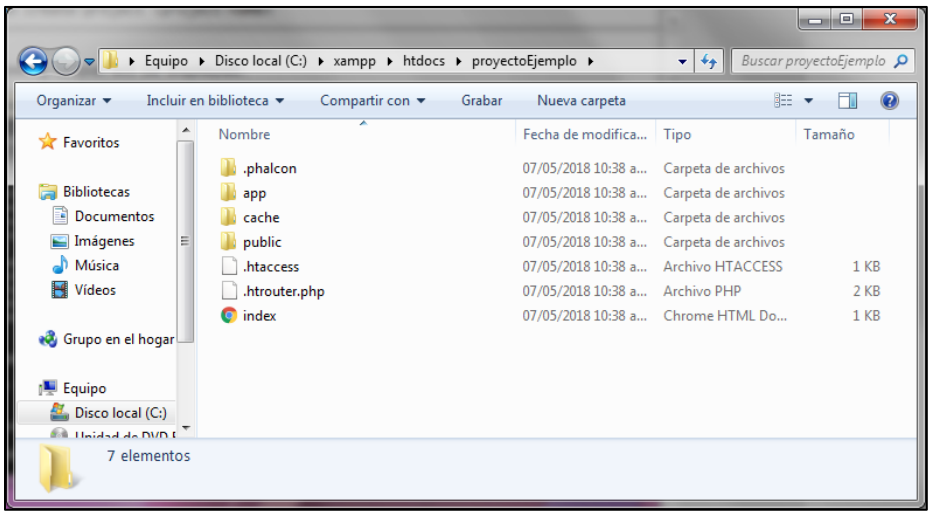


Figura 1.49 Estructura de carpetas creadas por las *herramientas de desarrollo*.

Ahora con el navegador nos dirigimos a la dirección: *localhost/nombredelproyecto*. En este caso el nombre del proyecto es “proyectoEjemplo”, lo cual la dirección seria *localhost/proyectoEjemplo*.

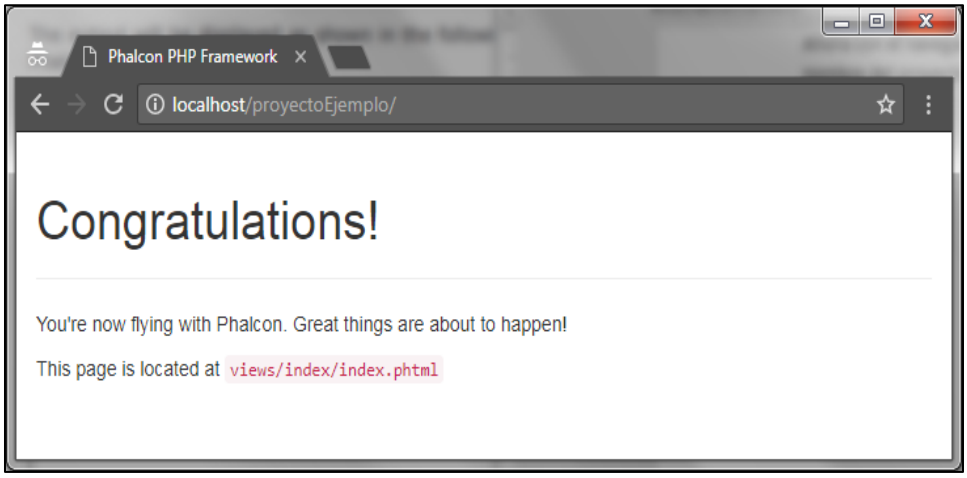


Figura 1.50 Página principal del proyecto.

Revisemos a fondo el proyecto que se generó con las *herramientas de desarrollo*. Nota que es muy similar al proyecto del tema anterior cuando creamos un formulario para registrar alumnos. También debes notar que en *app* se creó la carpeta *config* que contiene archivos de configuración e inicialización de componentes y servicios (*loader.php, router.php, services.php*), así como la conexión a la base de datos (*config.php*). De esta manera se separa aún más el código que escribimos en el fichero Bootstrap.

1.6.1 Creando un controlador

Vamos a añadir un controlador, por ejemplo, crearemos uno llamado "WelcomeController". Para ello nos dirigimos desde el símbolo del sistema a *C:\xampp\htdocs\proyectoEjemplo y* ejecutamos el siguiente comando:

```
phalcon create-controller <nombre-del-controlador>
```

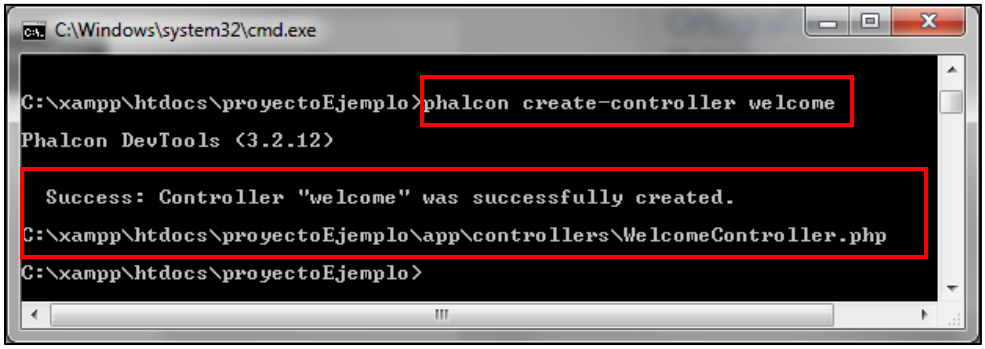


Figura 1.51 Creación de un proyecto con las *herramientas de desarrollo*.

¡Qué fácil es crear un controlador! Recuerda que cuando se ejecuta el comando, este debe ir sin el sufijo *Controller* ya que **Phalcon** lo agregará por sí mismo.

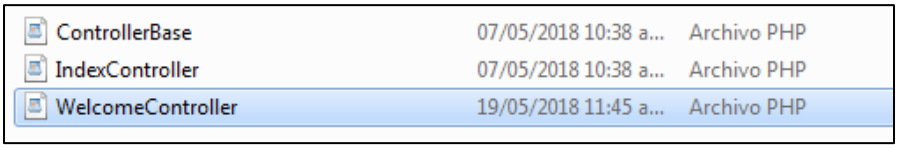


Figura 1.52 Controlador creado con las *herramientas de desarrollo*.

Y si revisamos el código que se generó en "WelcomeController.php*"* veremos que es similar a los que hemos creado con anterioridad.

```php
<?php

class WelcomeController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

}
```

Figura 1.53 Código del controlador "WelcomeController".

1.6.2 Creando un modelo

Utilizando la base de datos del tema anterior, vamos a crear un modelo de la tabla “alumnos”, pero antes, debemos configurar la conexión a la base de datos, para ello nos dirigimos a *app/config* y editamos el archivo “config.php” añadiendo nuestras credenciales de acceso.

```
9   return new \Phalcon\Config([
10      'database' => [
11          'adapter'     => 'Mysql',
12          'host'        => 'localhost',
13          'username'    => 'root',
14          'password'    => '',
15          'dbname'      => 'curso_db',
16          'charset'     => 'utf8',
17      ],
```

Figura 1.54 Configuración para la conexión a base de datos.

Ahora crearemos el modelo, así que nos dirigimos desde el símbolo del sistema a *C:\xampp\htdocs\proyectoEjemplo y* ejecutamos el siguiente comando.

```
phalcon create-model<nombre-del-modelo>
```

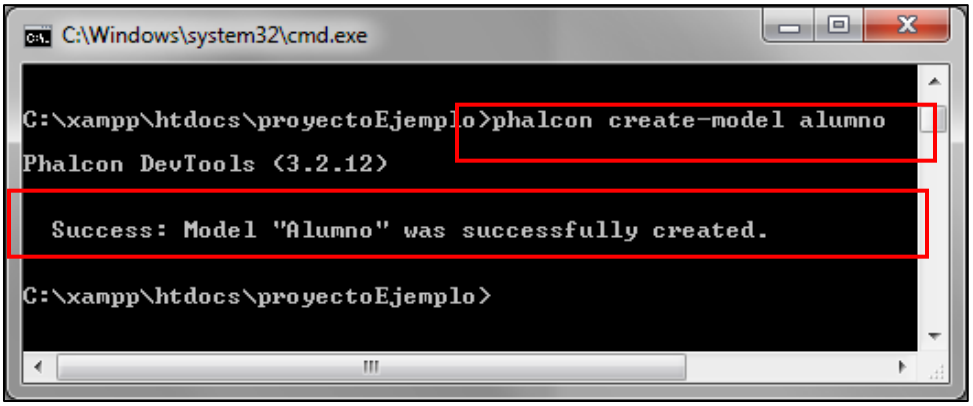


Figura 1.55 Creación de un modelo utilizando las *herramientas de desarrollo*.

- Lo modelos que agreguemos deben de llamarse igual a las tablas de la base de datos, de lo contrario las herramientas de desarrollo nos arrojaran un error (**Figura 1.56**).

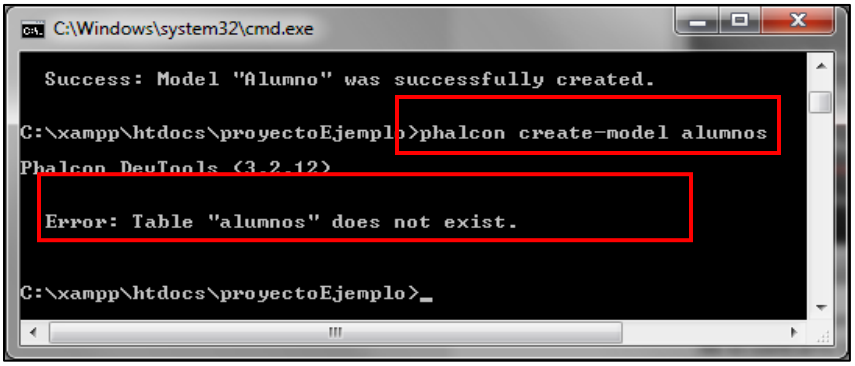


Figura 1.56 Error al crear un modelo de una tabla que no existe.

Si abrimos el modelo creado veremos, que se definieron variables en la clase “Alumno” con el mismo nombre que las columnas de la tabla en base de datos.

```
3   class Alumno extends \Phalcon\Mvc\Model
4   {
5
6       /**
7        *
8        * @var integer
9        * @Primary
10       * @Identity
11       * @Column(column="id", type="integer", length=11, nullable=false)
12       */
13      public $id;
14
15      /**
16       *
17       * @var string
18       * @Column(column="nombre", type="string", length=20, nullable=false)
19       */
20      public $nombre;
21
22      /**
23       *
24       * @var string
25       * @Column(column="matricula", type="string", length=13, nullable=false)
26       */
27      public $matricula;
28
29      /**
30       *
31       * @var integer
32       * @Column(column="edad", type="integer", length=11, nullable=false)
33       */
34      public $edad;
35
```

```php
    /**
     * Initialize method for model.
     */
    public function initialize()
    {
        $this->setSchema("curso_db");
        $this->setSource("alumno");
    }

    /**
     * Returns table name mapped in the model.
     *
     * @return string
     */
    public function getSource()
    {
        return 'alumno';
    }
```

Figura 1.57 Contenido del modelo “Alumno” (parte 1).

Se ha creado el método *initialize()* para inicializar el modelo (de esta manera nosotros le podemos dar un nombre diferente a la base de datos o a la tabla).

```php
    /**
     * Allows to query a set of records that match the specified conditions
     *
     * @param mixed $parameters
     * @return Alumno[]|Alumno|\Phalcon\Mvc\Model\ResultSetInterface
     */
    public static function find($parameters = null)
    {
        return parent::find($parameters);
    }

    /**
     * Allows to query the first record that match the specified conditions
     *
     * @param mixed $parameters
     * @return Alumno|\Phalcon\Mvc\Model\ResultInterface
     */
    public static function findFirst($parameters = null)
    {
        return parent::findFirst($parameters);
    }

}
```

Figura 1.58 Contenido del modelo “Alumno” (parte 2).

Por ultimo vemos dos métodos estáticos, que son *queries* para realizar consultas a los registros de la tabla. *find()* y *findFirst()* los veremos más adelante.

Crear un proyecto utilizando las *herramientas de desarrollo* es muy fácil y sencillo, además que se realiza todo de manera más eficiente, de esta manera te enfocas más en aspectos del *front-end.*

Si queremos conocer todos los comandos y descripción de las tareas que realiza las *herramientas de desarrollo* únicamente debemos escribir en el símbolo del sistema el siguiente comando:

```
phalcon
```

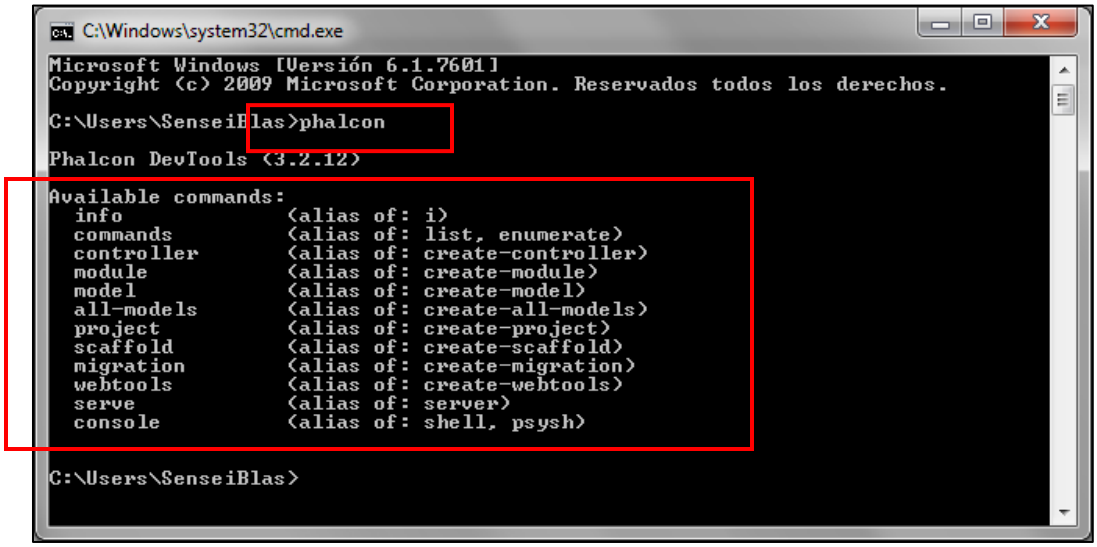


Figura 1.58 Lista de comandos de las *herramientas de desarrollo*.

1.6.3 Creando vistas

Desde el inicio de estos tutoriales hemos trabajado vistas con la extensión *.phtml* que combina HTML y PHP. Pensaras que un archivo PHP también puede incluir HTML, pero esta no es una buena práctica ya que esta extensión debe de contener código relacionado a la lógica de negocio y no tanto a la presentación.

1.6.3.1 Motor de plantillas VOLT

VOLT es un motor de plantillas escrito en C para PHP, lo cual hace que la carga de vistas sea más rápida. Nos permite escribir vistas de manera rápida y amigable, de esta manera mejora las buenas practicas. Además nos ayuda a ahorrar código (por ejemplo apertura y cierre <*?php ¿*>, declaración de variables, bucles, condicionales, etc.).

Para poder hacer esto posible, hacemos uso de dos delimitadores:

Delimitador	Función
{% ... %}	- Sirve para ejecutar sentencias o declarar variables - remplaza las etiquetas de apertura y cierre en PHP
{{ ... }}	- Imprime el resultado de una expresión o una variable - Remplaza a *echo* en PHP

Tabla 1.4 Delimitadores del motor de plantillas VOLT

1.6.3.2 VOLT como un servicio

VOLT viene cargado por defecto en un proyecto creado con las *herramientas de desarrollo*, así que no es necesario cargarlo, este servicio es solicitado en el archivo “services.php” en *app/config.* Las clases que hacen esto posibles son:

- *Phalcon\Mvc\View*
- *Phalcon\Mvc\View\Engine\Volt as VoltEngine*

El código del archivo “services.php” se encentra en la **Figura 1.59**.

```php
/**
 * Setting up the view component
 */
$di->setShared('view', function () {
    $config = $this->getConfig();

    $view = new View();
    $view->setDI($this);
    $view->setViewsDir($config->application->viewsDir);

    $view->registerEngines([
        '.volt' => function ($view) {
            $config = $this->getConfig();

            $volt = new VoltEngine($view, $this);

            $volt->setOptions([
                'compiledPath' => $config->application->cacheDir,
                'compiledSeparator' => '_'
            ]);

            return $volt;
        },
        '.phtml' => PhpEngine::class

    ]);

    return $view;
});
```

Figura 1.59 Archivo “services.php”.

1.6.3.3 Ejemplo de uso de una vista con extensión .volt

Sobre este mismo proyecto (“proyectoEjemplo”) crearemos un controlador llamado “EjemploController”.

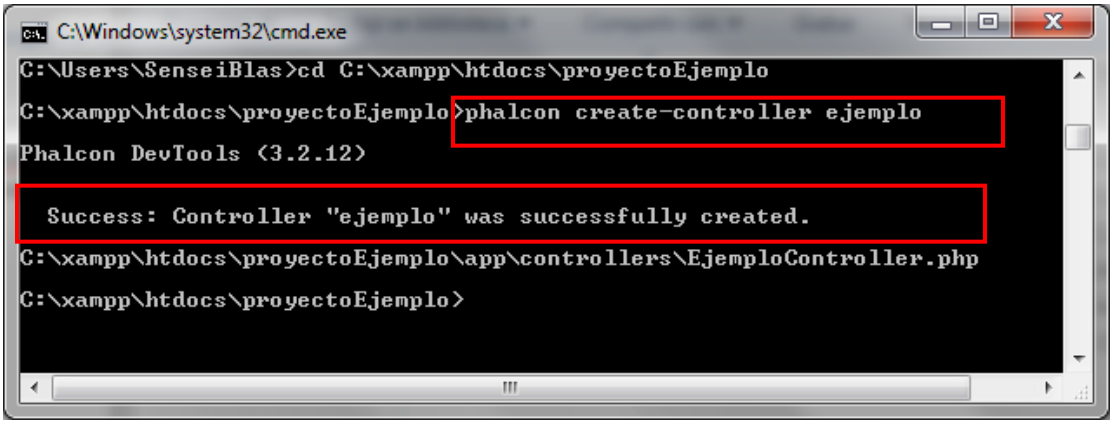


Figura 1.60 Creación del controlador “EjemploController” con las *herramientas de desarrollo*.

Después creamos en *app/views* una carpeta llamada “ejemplo” y dentro de esta el archivo “index.volt”.

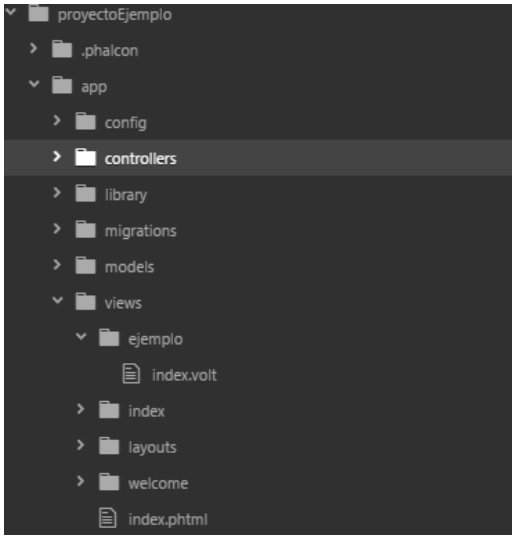


Figura 1.61 Creación de la carpeta “ejemplo” y dentro se crea el archivo “index.volt”.

Dentro de *“*index.volt*”* escribimos algún mensaje, por ejemplo escribiré lo siguiente:

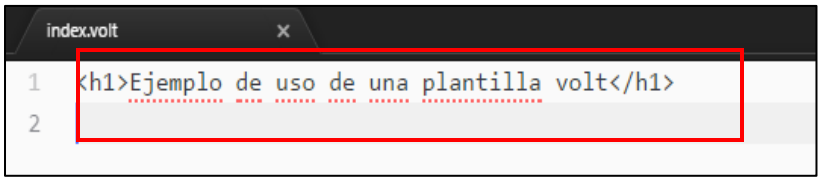


Figura 1.62 Contenido de “index.volt”.

Ahora vamos con el navegador a *http://localhost/proyectoEjemplo/ejemplo* y ya tenemos una vista!

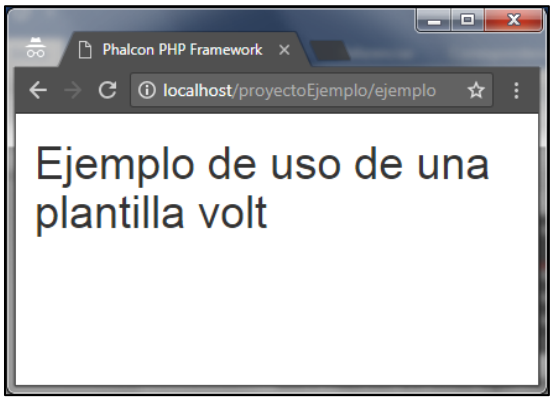


Figura 1.63 Página principal del proyecto vista desde el navegador.

Aprendamos con ejemplos lo que se puede hacer con VOLT.

1.6.3.3.1 Variables

Manejo de cadenas	
Ejemplo de uso	***Visualización en el navegador***
`{% set nombre = 'Angel Blas' %}` `<h2>Hola mi nombre es {{nombre}}</h2>`	Hola mi nombre es Angel Blas
Manejo de enteros	
Ejemplo de uso	***Visualización en el navegador***
`{% set edad = 25 %}` `<h2>Hola mi edad es {{edad}}</h2>`	Hola mi edad es 25
Manejo de arreglos	
Ejemplo de uso	***Visualización en el navegador***

`{% set frutas = ['manzana', 'platano', 'naranja'] %}`
`<h3>Lista de frutas disponibles</h3>`
`<ul>`
`{% for fruta in frutas %}`
`<li>{{ fruta }}</li>`
`{% endfor %}`
`</ul>`
	Lista de frutas disponibles
	• manzana
	• platano
	• naranja

1.6.3.3.2 Comentarios

Utilizan los delimitadores **{# … #}**

Manejo de cadenas	
Ejemplo de uso	***Visualización en el navegador***

`{# agragamos a precio un comentario #}`
`{% set gasolina = 20 %}`
`{{gasolina}}`
	20

La mayoría de editores de texto no reconocen por defecto archivos volt. En los ejemplos anteriores y los próximos se muestran con el editor de texto Atom instalado por defecto, para que pueda reconocer la extensión, es necesario instalar el paquete *language-volt 0.2.2* desde el editor.

1.6.3.3.3 Filtros

Son métodos que utilizan para para trabajar con cadenas o números. Los más útiles son:

Filtro	Descripción
escape	Aplica *Phalcon\Escaper->escapeHtml()* Esto nos sirve cuando queremos imprimir etiquetas HTML como texto plano a la página
trim	Quita espacios en blanco extras tanto a la izquierda como a la derecha
striptags	Retira etiquetas HTML
capitalize	Pone en mayúscula la primer letra de una cadena
lower	Convierte toda la cadena a minúsculas
upper	Convierte toda la cadena a mayúsculas
length	Devuelve la longitud de una variable
sort	Ordena una cadena de menor a mayor

Tabla 1.5 Filtros del motor de plantillas VOLT

Los ejemplos de uso de los filtros se muestran en la **Figura 1.64**.

```
{# filtro e o escape #}
{{ '<h1>Hola<h1>'|e }}
{{ '<h1>Hola<h1>'|escape }}

{# filtro trim #}
{{ '   hola aquí hay espacios en blanco   '|trim }}

{# filtro striptags #}
{{ '<h1>Hola amigos<h1>'|striptags }}

{# filtro capitalize #}
{{ 'angel Blas'|capitalize }}

{# filtro lower #}
{{ 'LIONEL MESSI'|lower }}

{# filtro upper #}
{{ 'messi'|upper }}
```

```
{# filtro length #}
{{ 'numeros'|length }}
{{ [1, 10, 5]|length }}

{# filtro sort #}
{% set ordena = [10, 12, 2,60]|sort %}
<ul>
   {% for num in ordena %}
   <li>{{ num }}</li>
   {% endfor %}
</ul>
```

Figura 1.64 Uso y sintaxis de los filtros.

1.5.3.3.4 Estructuras de control

Ciclo For	
Ejemplo de uso	***Visualización en el navegador***
{# for #} {% set tabla = [1, 2, 3, 4, 5]%} {% for num in tabla %} 9 x {{ num }} = {{9*num}} {% endfor %} 	• 9 x 1 = 9 • 9 x 2 = 18 • 9 x 3 = 27 • 9 x 4 = 36 • 9 x 5 = 45
For con clave-valor	

Ejemplo de uso	*Visualización en el navegador*
{% set paises = ['España': 1, 'México': 5, 'Argentina': 3] %} {% for pais, habitantes in paises %} País: {{ pais }} Habitantes: {{ habitantes }} </br> {% endfor %}	País: España Habitantes: 1 País: México Habitantes: 5 País: Argentina Habitantes: 3

Condicionales If, else

Ejemplo de uso	*Visualización en el navegador*
{% set votantes = ['Alejandra': 17, 'Cecilia': 18, 'Susana': 30] %} {% for nombre, edad in votantes %} {% if edad >= 18 %} {{nombre}} ya puede votar {% elseif edad < 18 %} {{nombre}} no puede votar {% endif %}</br> {% endfor %}	Alejandra no puede votar Cecilia ya puede votar Susana ya puede votar

Switch

Ejemplo de uso	*Visualización en el navegador*
{% set puntos_ganados = 3 %} {% switch puntos_ganados %} {% case 0 %} Obtuviste 0 puntos por lo tanto perdiste el partido {% break %} {% case 1 %} Obtuviste 1 punto por lo tanto empataste el partido {% break %} {% case 3 %} Obtuviste los 3 puntos, has ganado {% break %} {% default %} Número incorrecto {% endswitch %}	Obtuviste los 3 puntos, has ganado

1.7 Enrutamiento

El enrutamiento consiste en descomponer una petición URL para ser analizada y posteriormente atendida por un controlador asignado. Anteriormente vimos como es que **Phalcon** atiende una URL válida a través de controladores y acciones. La siguiente figura muestra el proceso que realiza **el** patrón MVC con las solicitudes entrantes.

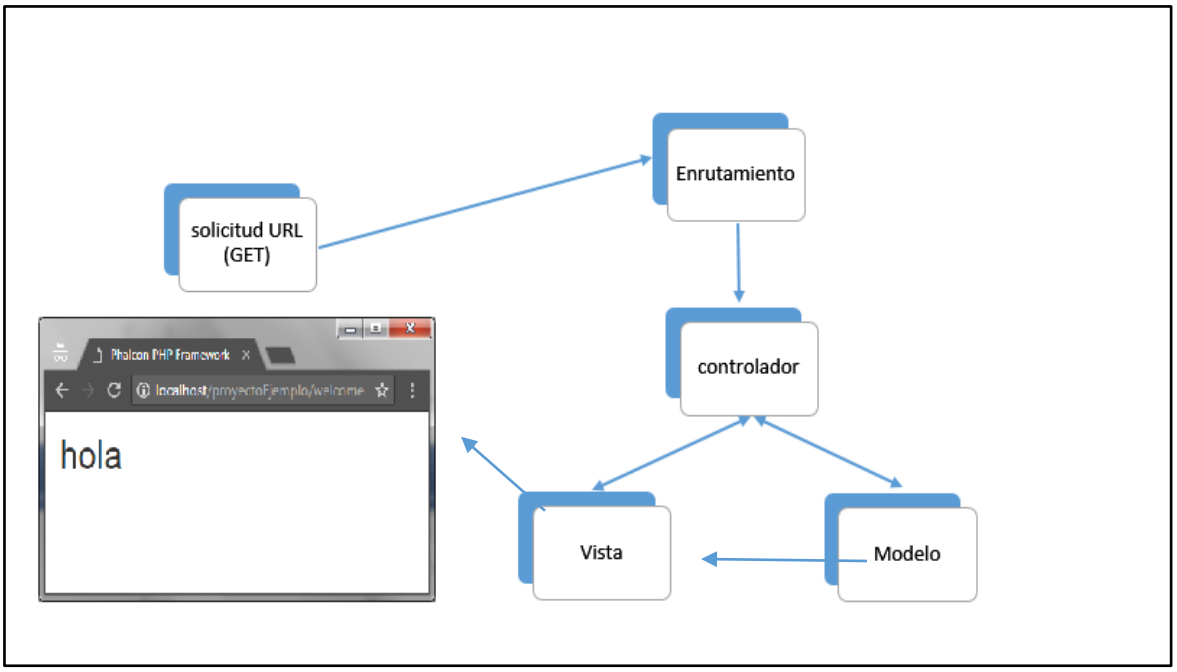


Figura 1.65 Forma en que el patrón MVC atiende una petición.

Cuando ingresamos una URL en el navegador, por ejemplo *http://localhost/proyectoEjemplo/ejemplo* quiere decir que estamos accediendo mediante una petición HTTP* de tipo GET**, esta petición es analizada por **Phalcon** mediante el archivo "router.php" (**Figura 1.66**), en caso de no existir un controlador que atienda esta petición mandara un error, de lo contrario será atendida, si es necesario se interactúa con el modelo asociado para obtener información de la base de datos. Esta información la recibe el controlador que invocara una vista donde se presentarán los resultados.

Hypertext Transfer Protocol o HTTP es el protocolo de comunicación que permite las transferencias de información en internet.

**GET es un método HTTP para solicitar recursos en internet.

router.php

```php
<?php

$router = $di->getRouter();

// Define your routes here

$router->handle();
```

Figura 1.66 Archivo "router.php" en ubicado en *app/config*.

1.7.1 Creación de Rutas

Al momento de crear un proyecto utilizando las *herramientas de desarrollo* se genera en *app/config* un archivo llamado "router.php" con las siguientes líneas de código de la **Figura 1.66**:

- $router es un objeto que se usa para traer el servicio *router* a partir del método *getRouter()* de nuestro contenedor encargado de inyectar las dependencias.

Supóngase que queremos ingresar a la dirección *http:///localhost/proyectoEjemplo/home/prueba* donde asignaremos como controlador a "IndexController" y la acción "rutaAction".

en "router.php" debemos definirla de la siguiente forma:

```php
<?php

$router = $di->getRouter();

// Define your routes here
$router->add(
    '/home/prueba',
    [
        'controller' => 'index',
        'action'     => 'ruta',
    ]
 );

$router->handle();
```

Figura 1.67 Agregando rutas en el archivo "router.php".

Con el método *add* agregamos la ruta entre comillas simples seguidas de una coma, ingresamos como parámetros un arreglo donde definimos el controlador y la acción que despacharán a la URL ingresada desde el navegador.

Para poder visualizarlo necesitamos crear el controlador “IndexController” y dentro colocar la acción “rutaAction”:

```php
<?php

class IndexController extends ControllerBase
{

    public function indexAction()
    {

    }
    public function rutaAction()
    {
        echo "Esta es la ruta de prueba";
    }

}
```

Figura 1.68 Contenido de la acción “RutaAction”.

Ahora si nos vamos con el navegador a la dirección *http:///localhost/proyectoEjemplo/home/prueba* veremos lo siguiente:

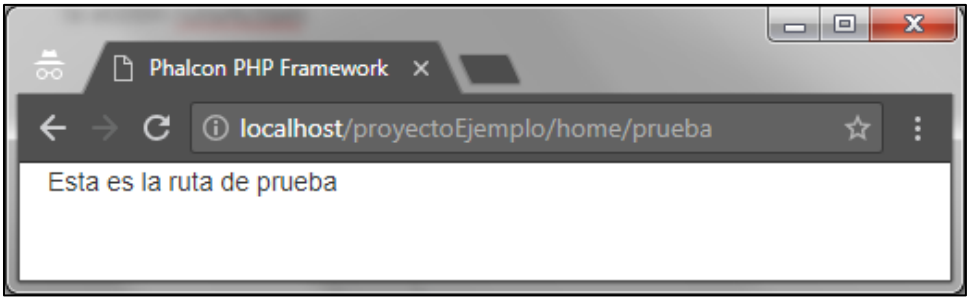


Figura 1.69 Vista que atiende a la ruta *http:///localhost/proyectoEjemplo/home/prueba.*

Imaginemos ahora que nuestro proyecto es un sistema bancario donde un usuario previamente registrado podrá visualizar su estado de cuenta, pero antes de acceder necesita ingresar su nombre de usuario y contraseña. Cuando el usuario ingresa correctamente, el sistema mostrará la página principal con su nombre y su estado de cuenta.

Podríamos definir una ruta para mostrar dicha información (**Figura 1.70**):

```
14  $router->add(
15      '/home/:action/:params',
16      [
17          'controller' => 'index',
18          'action'     => 1,
19          'params'     => 2,
20      ]
21  );
```

Figura 1.70 Configuración de parámetros a la ruta.

- :*action* llama a cualquier acción que ingresemos, siempre y cuando exista.
- :*params* significa que podemos ingresar cualquier parámetro separados por una /.
- *IndexController* es el controlador que atenderá la petición.
- Si quisiéramos llamar a cualquier controlador se coloca *:controller*.
- Cuando ingresamos ":" debemos numerarlos al momento de pasar los parámetros en el *array*.

Ahora en "IndexController" crearemos la acción "bienvenidaAction" que recibe como parámetros los datos del usuario vía URL.

```
15      public function bienvenidaAction($nombre,$saldo)
16      {
17        echo "Hola " . $nombre . " tienes: " . $saldo .
18        " pesos en tu cuenta";
19      }
20
```

Figura 1.71 Código de la acción "bienvenidaAction". Recibe como parámetros $nombre y $saldo.

Si ingresamos a la dirección *http://localhost/proyectoEjemplo/home/bienvenida/angel/5000/* veremos el siguiente resultado:

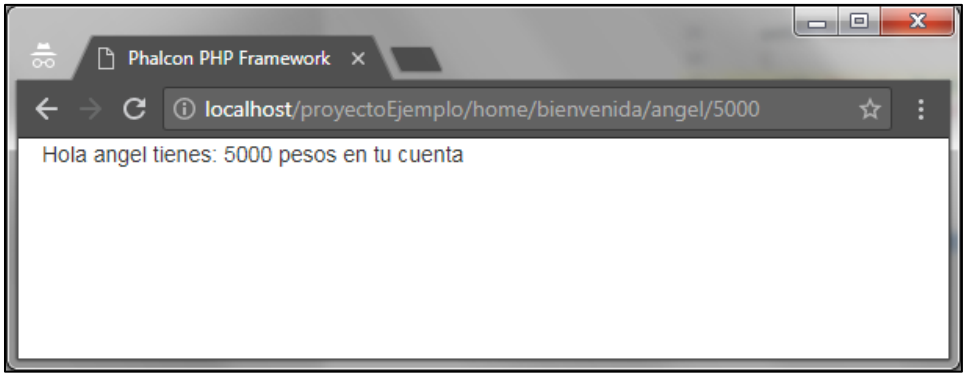


Figura 1.72 Vista que atiende a la ruta *http://localhost/proyectoEjemplo/home/prueba.*

1.8 Conexión a una base de datos

Anteriormente habíamos visto una forma de configurar una conexión a la base de datos, pero desde el archivo Bootstrap ubicado en *public/index.php.* Ahora veremos que cuando creamos un nuevo proyecto con las *herramientas de desarrollo*, la configuración cambia de documento a *app/config/config.php* (**Figura 1.73**).

```
9   return new \Phalcon\Config([
10      'database' => [
11          'adapter'     => 'Mysql',
12          'host'        => 'localhost',
13          'username'    => 'root',
14          'password'    => '',
15          'dbname'      => 'curso_db',
16          'charset'     => 'utf8',
17      ],
```

Figura 1.73 Configuración a base de datos *app/config/config.php*.

En la figura anterior tenemos los siguientes parámetros:

- adapter. Nos va a permitir cambiar entre los diferentes motores de bases de datos soportados por **Phalcon**.
- host. Dominio donde se encuentra almacenada la base de datos.
- username. Nombre de usuario de acceso.
- password. Contraseña de acceso.
- dbname. Nombre de la base de datos.
- charset. Tipo de codificación.

1.8.1 Adaptadores de base de datos

Se utiliza para especificar el motor de base de datos de deseamos, así de esta manera se importarán únicamente las herramientas específicas del sistema seleccionado. **Phalcon** utiliza PHP Data Objects (PDO) en pocas palabras hace uso de clases y objetos para realizar consultas y conexiones.

Estas son las bases de datos soportadas:

Clase	Descripción
Phalcon\Db\Adapter\Pdo\Mysql	Adaptador MySQL
Phalcon\Db\Adapter\Pdo\Postgresql	Adaptador para PostgreSQL

Phalcon\Db\Adapter\Pdo\Sqlite	Adaptador SQLite

Tabla 1.5 Adaptadores de base de datos.

1.9 Consultas PHQL

Phalcon Query Language (PHQL) es un lenguaje de alto nivel para realizar consultas SQL con programación orientada a objetos y que además contiene un analizador escrito en lenguaje C que se encarga de traducir las sentencias en alto nivel a el lenguaje del sistema de gestión de base de datos con el que estemos trabajando.

Vamos a crear consultas que se utilizan de forma frecuente. Anteriormente ya habíamos creado una base de datos (“curso_db”) y una tabla (“alumno”), también en el proyecto anterior (“proyectoEjemplo”) creamos un modelo de “alumno” utilizando las *herramientas de desarrollo*.

Sobre este mismo proyecto vamos a crear un controlador llamado “AlumnoController” y ahí escribiremos nuestras consultas.

Antes de empezar vayamos a nuestra tabla alumno y agreguemos más registros.

Figura 1.74 Registros en la tabla “alumno”.

Ahora en “AlumnoController” para poder hacer consultas debemos importar la siguiente clase:

```
Phalcon\Mvc\Model\Query
```

```
1  <?php
2  use Phalcon\Mvc\Model\Query;
3  class AlumnoController extends \Phalcon\Mvc\Controller
4  {
```

Con el siguiente código vamos a traer todos los registros de nuestra tabla con la consulta SELECT.

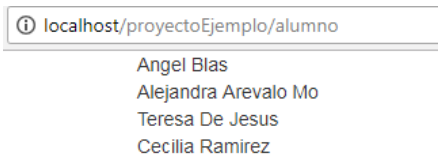
Ejemplo de uso	*Resultado en el navegador*

```
public function indexAction()
{
    // Instanciar el Query
    $query = new Query(
        'SELECT * FROM Alumno',
        $this->getDI()
    );

    // Ejecutar una consulta retornando
    //los resultados
    $alumnos = $query->execute();
    //imprimimos los resultados
    foreach ($alumnos as $alumno) {
      echo $alumno->nombre . "<br>";
    }
}
```

También podemos hacerlo utilizando *modelsManager*:

Ejemplo de uso	*Resultado en el navegador*

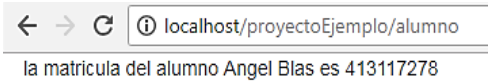
```
public function indexAction()
{
    // Ejecutando una consulta simple
    $query = $this->modelsManager->createQuery('SELECT * FROM Alumno');
    $alumnos  = $query->execute();
    //imprimimos los resultados
    foreach ($alumnos as $alumno) {
      echo $alumno->nombre . "<br>";
    }

}
```



Ahora vamos a realizar una consulta para obtener la matricula del alumno “Angel Blas”. Para esto necesitamos definir un parámetro utilizando la sentencia *WHERE*.

Ejemplo de uso	*Resultado en el navegador*

```
public function indexAction()
{
  $alumno = $this->modelsManager->executeQuery(
      'SELECT * FROM Alumno WHERE nombre = :nombre:',
      [
        'nombre' => 'Angel Blas',
      ]
  );
  //imprimimos el número de matrícula
  echo "la matricula del alumno ".$alumno[0]->nombre." es ".
  $alumno[0]->matricula;

}
```

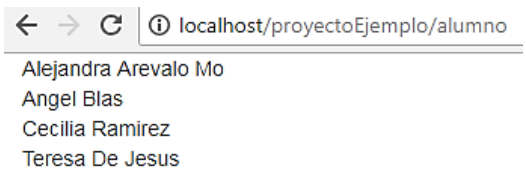


- Los métodos anteriores utilizan consultas casi idénticas al lenguaje SQL, pero en vez de hacer referencia a la tabla en base de datos se hace sobre el modelo “Alumno”.

Haremos uso de la cláusula *ORDER BY* para ordenar los resultados por nombre.

Ejemplo de uso	***Resultado en el navegador***

```
$query = $this->modelsManager->createQuery(
    'SELECT * FROM Alumno ORDER BY Alumno.nombre'
);
$alumnos = $query->execute();

foreach ($alumnos as $alumno) {
 echo $alumno->nombre . "<br>";
}
```



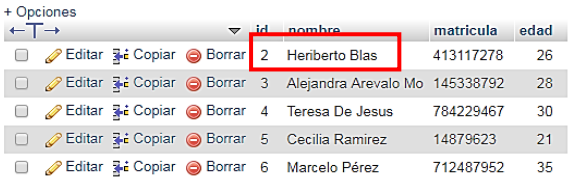
- PHQL escapa automáticamente los datos de entrada para que de esta manera se eviten las inyecciones SQL.

Si queremos insertar datos a nuestra tabla, debemos usar la instrucción *INSERT*.

Otra de las consultas más comunes es la actualización de algún registro en la tabla. Para ello haremos uso de la instrucción *UPDATE* que es muy similar a *INSERT* donde tenemos que definir las columnas a actualizar. En este ejemplo cambiaremos el nombre del registro con el id = 2.

Ejemplo de uso	***Validación en la tabla “alumno”:***

```
public function indexAction()
{
  // Actualizando el nombre
    $phql = 'UPDATE Alumno SET nombre = "Heriberto Blas"
    WHERE id = 2';
    $this->modelsManager->executeQuery($phql);
}
```



Por último haremos la eliminación de un registro, para ello haremos uso de la instrucción *DELETE*. En el siguiente ejemplo eliminaremos al alumno con el id = 5.

Ejemplo de uso	***Validación en la tabla “alumno”:***

```
public function indexAction()
{
    // Borrando una fila
    $phql = 'DELETE FROM Alumno WHERE id = 5';
    $this->modelsManager->executeQuery($phql);
}
```

1.10 Creación de un CRUD

Un CRUD es el acrónimo que se le da a un subsistema que realiza las operaciones de:

- Crear (*Create*)
- Leer (*Read*)
- Actualizar (*Update*)
- Borrar (*Delete*)

Son las 4 instrucciones básicas que se realizan en una base de datos y que en el tema anterior realizamos, si quisiéramos crear un proyecto sencillo que realice estas tareas sobre los datos de una tabla en particular podríamos realizar un CRUD. Las *herramientas de desarrollo* nos facilitan esta tarea.

En el siguiente ejemplo crearemos un *Scaffolding* (andamio en español) que es la forma en que las *herramientas de desarrollo* llaman a un CRUD.

Para realizar un *Scaffolding* de la tabla "alumno" crearemos un nuevo proyecto habilitando las *webtools* que es lo mismo a las *herramientas de desarrollo*, pero ya no en modo consola de comandos sino web y con interfaz gráfica.

Ahora nos dirigimos a *C:\xampp\htdocs* desde el símbolo del sistema y ejecutamos el comando siguiente:

```
phalcon create-project <nombre-proyecto> --enable-webtools
```

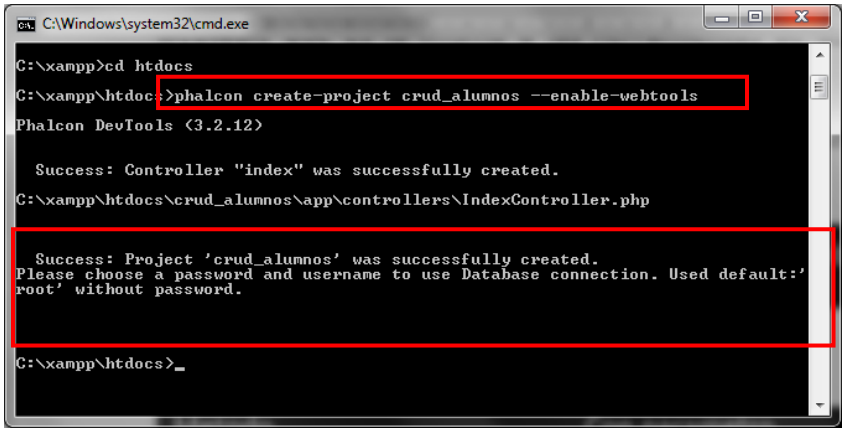


Figura 1.75 Registros en la tabla "alumno".

Si revisamos la estructura de carpetas y archivos del proyecto creado, notaremos que no cambia en nada, solo se añaden nuevos archivos en la carpeta *public/*

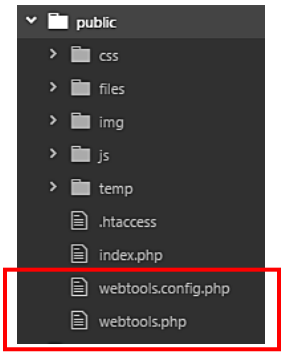


Figura 1.76 Registros en la tabla “alumno”.

Debemos modificar archivo *config.php* ubicado en *app/config/* para conectar nuestro proyecto a la base de datos y también añadiremos la siguiente línea al final del *array* “*application*”:

```
18      'application' => [
19          'appDir'         => APP_PATH . '/',
20          'controllersDir' => APP_PATH . '/controllers/',
21          'modelsDir'      => APP_PATH . '/models/',
22          'migrationsDir'  => APP_PATH . '/migrations/',
23          'viewsDir'       => APP_PATH . '/views/',
24          'pluginsDir'     => APP_PATH . '/plugins/',
25          'libraryDir'     => APP_PATH . '/library/',
26          'cacheDir'       => BASE_PATH . '/cache/',
27          'staticUri'      => '/crud_alumnos/',
```

Figura 1.77 Se define la variable “staticUri” con la URL estática del proyecto.

- En la **Figura 1.77**, la línea 28 permite cargar los estilos CSS para la página de las *webtools*.

Si nos dirigimos a la URL *http://localhost/crud_alumnos/webtools.php* veremos la siguiente pantalla:

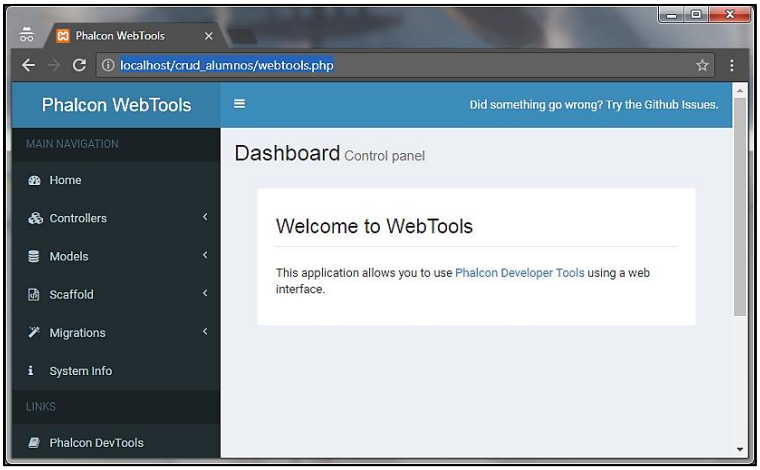


Figura 1.78 Página principal de las *webtools*.

Elegimos *"Scaffold"* del menú de navegación *y* después *"Generate".* Dentro de la siguiente ventana notaras que el campo *"Schema"* contiene el nombre de la base de datos y *"Table name*" muestra una lista desplegable con las tablas disponibles.

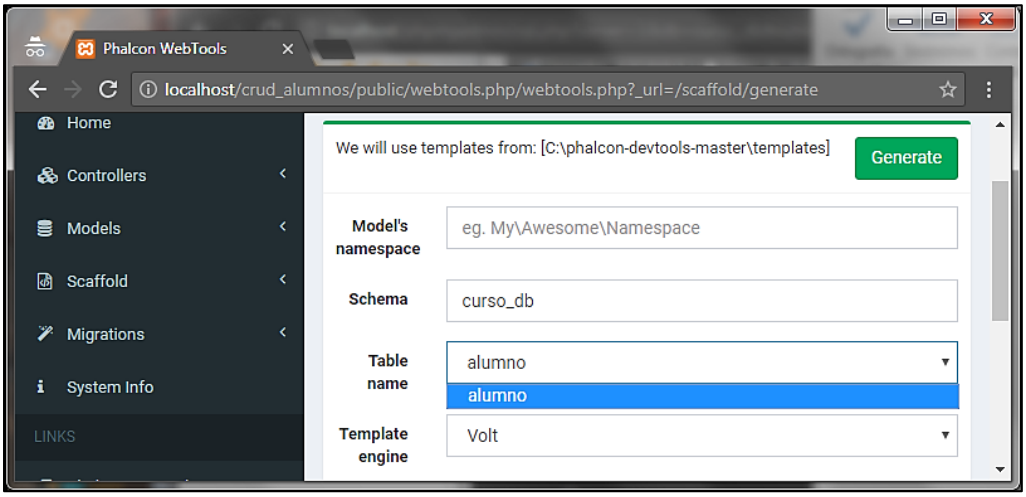


Figura 1.79 Selección de la base de datos "curso_db" y la tabla "alumno".

Solo nos queda seleccionar la tabla y dar *clic* en el botón "Generate" ubicado en la parte superior derecha.

Una vez creado nos dirigimos a *http://localhost/crud_alumnos/alumno* y ¡veremos nuestro CRUD generado! Puedes probar todas las acciones disponibles, intenta buscar un alumno o agregar uno nuevo. También puedes actualizar o eliminar.

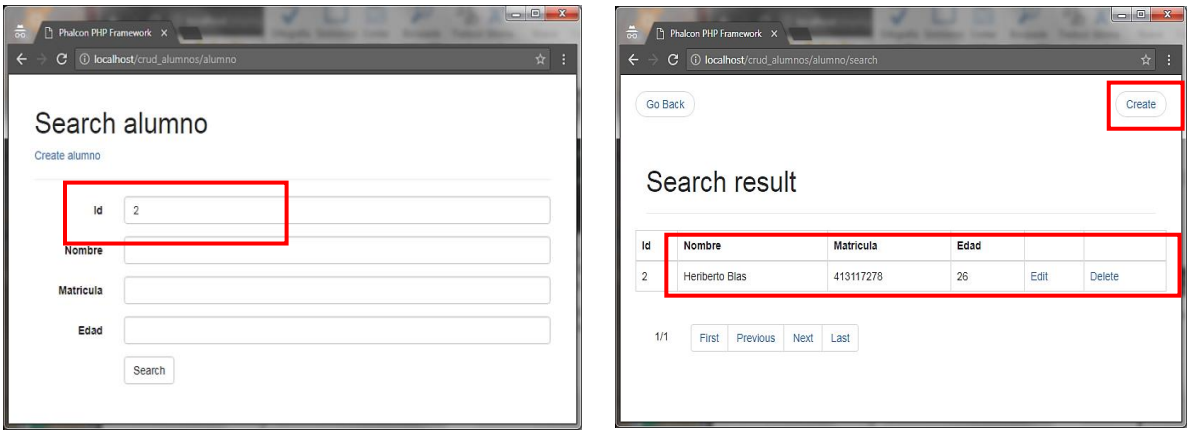


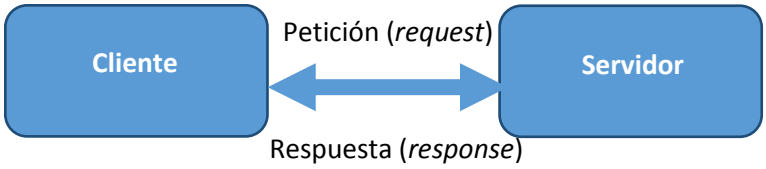
Figura 1.80 Visualización del CRUD generado con las *herramientas de desarrollo.*

1.11 Peticiones HTTP

El protocolo de comunicación HTTP (Hypertext Transfer Protocol) es el encargado de la transferencia de información (en particular HTML), permitiendo la interacción entre la aplicación al usuario. Por medio de este protocolo podemos realizar diversas funciones, por citar algunas:

- Solicitar archivos o programas.
- Consultar información a una base de datos a partir de datos ingresados.
- Obtener direcciones IP.
- Obtener el lenguaje del navegador utilizado por el usuario a fin de conocer la región de visita.

Su modo de operación es mediante la URL:



Existen diferentes métodos con los que trabaja este protocolo:

- GET
- HEAD
- POST
- PUT
- DELETE

Nosotros nos vamos a centrar en GET y POST:

- GET. Su objetivo es obtener información por parte del servidor puede ser de un archivo, un recurso, etc. Este método funciona independientemente de que tengamos que solicitar datos (*request*) para que el servidor pueda respondernos (*response*) como esperamos.
- POST. Su objetivo es enviar información por parte del cliente hacia el servidor. Generalmente utilizando formularios donde el usuario envía datos (*request*) para que puedan ser tratados por el servidor y regresar una respuesta (*response*) sobre información del usuario.

Para obtener información de un usuario por medio de un formulario en **Phalcon** tenemos que crear un objeto de la clase *Phalcon\Http\Request* que nos permitirá acceder a los valores enviados por GET o POST.

1.11.1 Utilizando GET

En nuestro proyecto creado anteriormente (“proyectoEjemplo”) vamos a agregar un formulario utilizando únicamente HTML sobre la vista ubicada en *views/index/index.volt:*

```
1  <form action="/proyectoEjemplo/registro/guardar">
2    Nombre:<br>
3    <input type="text" name="nombre"><br>
4    Apellido:<br>
5    <input type="text" name="apellido"><br><br>
6    <input type="submit" value="Enviar">
7  </form>
```

Figura 1.81 Dentro del formulario se define la ruta que atiende los datos a mandar.

- Para poder ingresar la URL que atenderá los datos guardados debemos hacerlo utilizando el atributo *action*. El controlador que recibirá la información es “RegistroController” y su acción correspondiente es “guardarAction”.
- Por defecto el tipo de método de envió de información es GET, si quiéranos cambiarla debemos escribir el atributo *method.*

Debemos crear el controlador registro y dentro la acción guardar. Aquí recibiremos los datos enviados:

```
11      public function guardarAction()
12      {
13        $nombre = $this->request->get('nombre');
14        $apellido = $this->request->get('apellido');
15        echo "Tu nombre completo es: " . $nombre .
16        $apellido;
17
18      }
```

Figura 1.82 Acción que atiende a la petición del formulario.

Si nos dirigimos con el navegador a la URL *http://localhost/proyectoEjemplo/ejemplo* veremos el formulario, ingresamos nombre y apellido para después visualizar un mensaje con los datos escritos.

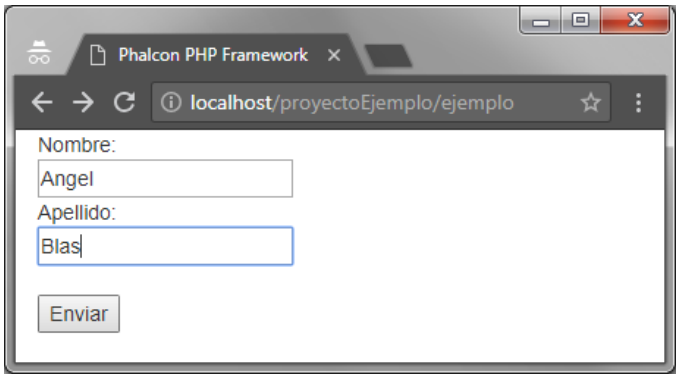


Figura 1.83 Visualización del formulario en el navegador.

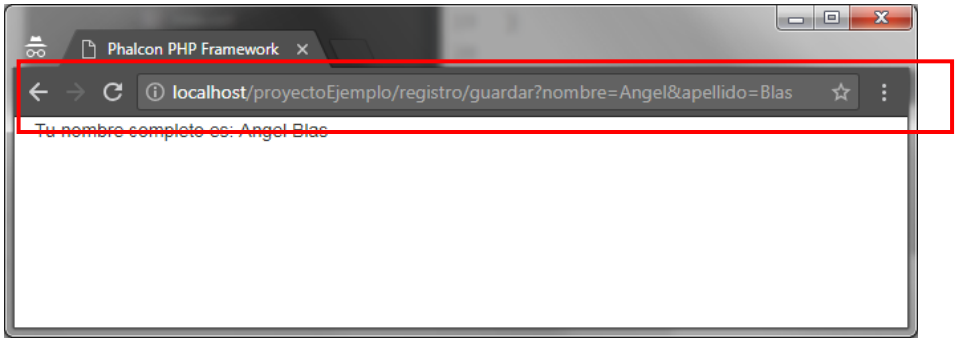


Figura 1.84 Pase de parámetros por URL.

Nota que en la URL después de la acción se encuentran las dos variables “nombre” y “apellido”, así como sus valores, cada variable está separada por el carácter “&” principal característica del método GET (**Figura 1.84**).

1.11.2 Utilizando POST

En la vista donde creamos el formulario, solo debemos agregar el siguiente atributo:

```
1  <form action="/proyectoEjemplo/registro/guardar" method="POST">
2    Nombre:<br>
3    <input type="text" name="nombre"><br>
4    Apellido:<br>
5    <input type="text" name="apellido"><br><br>
6    <input type="submit" value="Enviar">
7  </form>
```

Figura 1.85 dentro del formulario se define el tipo de método.

Nos dirigimos nuevamente con el navegador a la URL *http://localhost/proyectoEjemplo/ejemplo* y veremos el formulario, ingresamos nombre y apellido para después visualizar un mensaje con los datos escritos.

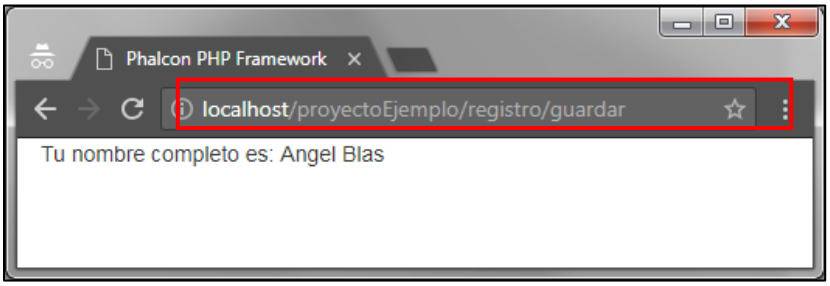


Figura 1.86 Los datos enviados no se visualizan en la URL.

La diferencia entre GET y POST se muestra en la forma de enviar los datos a la página cuando se pulsa el botón “Enviar”. Mientras que el GET utiliza la URL para enviar los datos, el método POST los envía de forma que no sean visibles para el usuario haciéndolo más seguro.

1.12 Agregar recursos con Assets

En este apartado veremos la forma en que debemos importar recursos para darle diseño a los componentes de nuestras páginas a fin de hacerlas más atractivas. No solo aprenderemos a importar archivos CSS sino también archivos JavaScript para darles un efecto dinámico e inclusive permitir interactuar con el usuario.

Assets es un componente que nos va a permitir importar recursos como hojas de estilo o librerías JavaScript a nuestro proyecto.

Para poder ver a detalle cómo trabaja **Phalcon** con el manejo de recursos vamos a crear un nuevo proyecto llamado “recursosAssets”. Una vez creado vamos a visualizarlo con el navegador, deberás ver algo como esto:

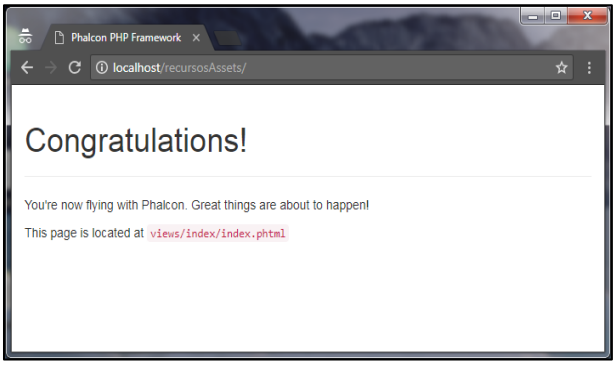


Figura 1.87 Página principal del proyecto “recursosAssets”.

El mensaje que se muestra, dice que la página visualizada se encuentra ubicado en el archivo “index.phtml” en *views/index/.* Si nos dirigimos a ese archivo veremos lo siguiente:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="utf-8">
5          <meta http-equiv="X-UA-Compatible" content="IE=edge">
6          <meta name="viewport" content="width=device-width, initial-scale=1">
7          <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
8          <title>Phalcon PHP Framework</title>
9          <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
10         <link rel="shortcut icon" type="image/x-icon" href="<?php echo $this->url->get('img/favicon.ico'); ?>" />
11     </head>
12     <body>
13         <div class="container">
14             <?php echo $this->getContent(); ?>
15         </div>
16         <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
17         <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>
18         <!-- Latest compiled and minified JavaScript -->
19         <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
20     </body>
21 </html>
```

Figura 1.88 Contenido del archivo “index.phtml”.

Analizando los puntos de la figura anterior, encontramos lo siguiente:

1. Esta página es una plantilla que contiene la estructura de etiquetas básica de un HTML (<html>, <head>, <body>).
2. El contenido de la etiqueta <head> contiene el elemento *link* que hace relación a una hoja de estilo (*StyleSheet*) externa o CDN (*Content Delivery Network*). La etiqueta necesita como atributo a *href* que es donde colocamos la ruta donde se encuentra la hoja de estilos.

Las herramientas de desarrollo cargan por defecto carga el *framework* Bootstrap así que podemos utilizarlo como ejemplo para darle estilos a una imagen.

3. Esta página se utiliza como plantilla para todas las demás vistas, en la línea 14 se hace usos del método *getContent()* que indica a *Phalcon\Mvc\View* donde debe inyectar el contenido de la vista que el usuario está visualizando.
4. Se importan los CDN´s en JavaScript para que Bootstrap funcione correctamente**.**

Ahora modificaremos el nombre de este archivo para que sea una vista con la extensión VOLT y no *phtml*.

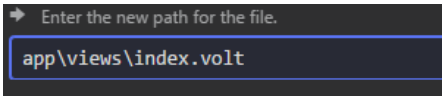


Figura 1.89 Cambio de extensión del archivo a “index.volt”.

También debes realizar el mismo procedimiento para la vista ubicada en *views/index/index.phtml*.

Ahora tenemos que cambar el método *getContent() (ubicado en views/index.volt)* por su equivalente en volt con los delimitadores **{{ … }}**.

Figura 1.90 Usamos los delimitadores del motor de plantillas VOLT.

Si recargamos la página de nuestro proyecto veremos que no ha cambiado nada.

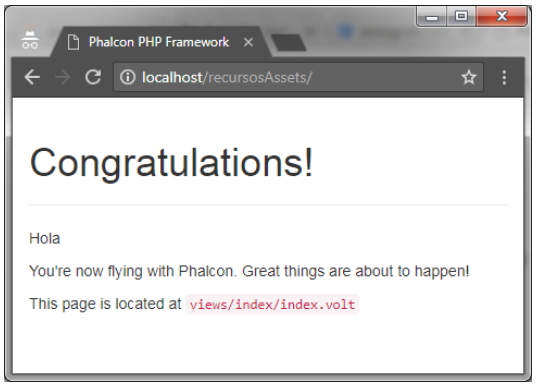


Figura 1.91 Página principal del proyecto.

Para poder agregar recursos necesitamos crear un archivo llamado “assets.php” y dentro importar *Phalcon\Assets\Manager* para después crear un objeto que se encargará de añadir las colecciones de CSS y JavaScript.

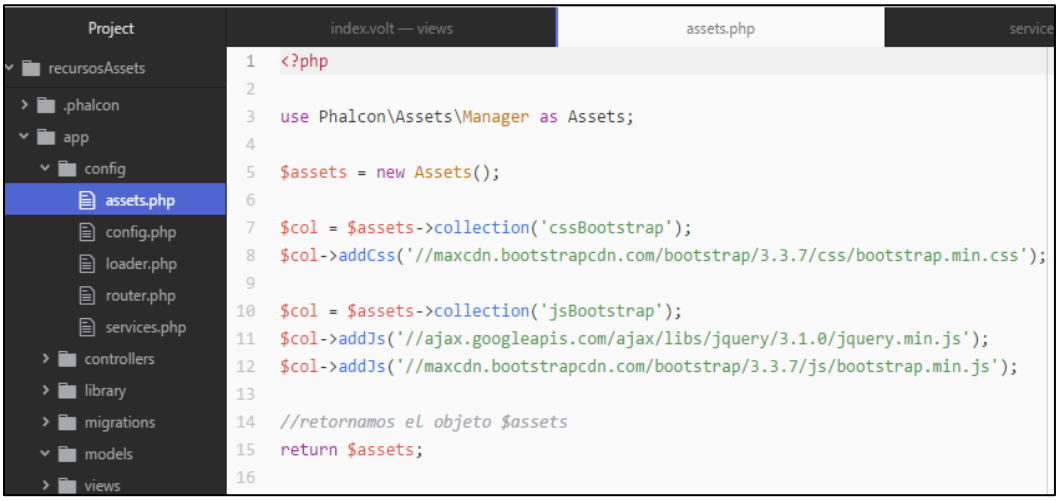


Figura 1.92 Contenido del archivo “assets.php”.

Como vemos en la **Figura 1.92**, se crea un objeto ($assets) de la clase “Assets” para acceder a su método *collection* que nos sirve para darle un nombre a nuestra colección en este caso “cssBootstrap*”* y asignarlo a la variable $col.

- En las líneas 8,11 y 13 se ingresa como parámetro la ruta donde se encuentra ubicado nuestro CSS (*addCss*) o bien el JavaScript (*addJs*). Por último retornamos el objeto $assets para que sea reconocido por el contenedor.

Registramos el servicio en el archivo “services.php” ubicado en *app/config/* agregando el siguiente código al final del archivo:

```
114 $di->setShared('assets', function () {
115     return include APP_PATH . '/config/assets.php';
116 });
117
```

Figura 1.93 Registro del servicio Assets.

Ahora en el archivo “index.php” ubicado en *app/views/* quitamos la hoja de estilos de Bootstrap así como sus correspondientes archivos JavaScript y ahora las cargamos desde el componente Assets utilizando los delimitadores **{{ … }} (Figura 1.94)**.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -
        <title>Phalcon PHP Framework</title>
        {{ assets.outputCss('cssBootstrap') }}
        <link rel="shortcut icon" type="image/x-icon" href="<?php echo $this->url->get('img/favicon.ico')?>"/>
    </head>
    <body>
        <div class="container">
            {{ content() }}
        </div>
        {{ assets.outputJs('jsBootstrap') }}
    </body>
</html>
```

Figura 1.94 Importación de los recursos utilizando delimitadores.

Si recargamos la página debemos ver que no ha cambiado, eso quiere decir que hemos hecho todo correctamente.

Probemos que el *framework* Bootstrap fue cargado correctamente, para ello modificaremos la vista (archivo “index.php” ubicado en *app/views/index/*) añadiendo lo siguiente:

```
1  <div class="page-header">
2      <strong>
3          <h1 class="text-center text-danger">
4          ¡Felicidades hemos incluido nuestros recursos con Assets!
5          </h1>
6      </strong>
7  </div>
8
9  <div class="container">
10     <img src="img/imagen.jpg" class="img-circle img-responsive col-md-12">
11 </div>
12
```

Figura 1.95 Contenido del archivo “index.php”.

Las clases *text-center*, *text-danger*, *container*, *img-circle*, *img-responsive*, *col-md-12* pertenecen al *framework* Bootstrap**,** ya tendremos tiempo de explicarlas en el siguiente capítulo, por ahora solo las usaremos para verificar la importación de los recursos.

En la línea 10 agregamos una imagen con la etiqueta <img> y pasamos mediante el atributo *src* la ruta donde se encuentra almacenada en este caso es *public/img* con el nombre de “imagen.jpg”.

Muy bien recarguemos nuevamente la página y veremos algo parecido a esto:

Figura 1.96 Estilos añadidos utilizando el *framework* Bootstrap que fue añadido como un recurso utilizando Assets.

1.13 Resumen

En este capítulo vimos que el *framework* **Phalcon** es un marco de desarrollo para PHP que se basa en el patrón de arquitectura Modelo – Vista - Controlador. También realizamos una comparativa de **Phalcon** con otros *frameworks* y conocimos sus ventajas, viendo que mejora el rendimiento y reduce el consumo de recursos.

Abarcamos desde la forma de programar sin el uso de algún marco de trabajo hasta la utilización de un patrón de diseño que mejora la práctica de programación y da ventajas de uso a corto y largo plazo.

Todo lo anterior fue meramente teórico, para la práctica aprendimos muchos conceptos y formas de uso mediante ejemplos:

- XAMPP es un programa que nos permite instalar un servidor Apache con PHP y MySQL.
- Aprendimos a instalar **Phalcon** en XAMPP**.**
- **Las** herramientas de desarrollo son *scripts* que nos sirven para crear proyectos, modelos, controladores y *Scaffolding* con un solo comando.
- Conocimos la estructura de archivos, carpetas y subcarpetas del MVC.

- Creamos una aplicación o proyecto desde cero donde agregamos alumnos a una tabla en base de datos por medio de un formulario. En esta aplicación se agregaron y conocieron los siguientes conceptos:
    - URL´s amigables.
    - El Boostrap que es el archivo principal donde inicializamos el inyector de dependencias para configurar componentes y servicios.
    - Aprendimos a crear controladores, la forma de nombrarlos y las acciones o métodos que agreguemos.
    - Creamos vistas volt y phtml.
    - Creamos un modelo que se encarga de interactuar con las tablas de la base de datos.
- Después de la creación de un proyecto desde cero hicimos uso de las herramientas de desarrollo para poder crear:
    - Proyectos
    - Modelos
    - Controladores
- El motor de plantillas volt se encarga de mostrar vistas de manera más eficiente debido a que es una extensión de C. De este motor de plantillas aprendimos a:
    - Agregar delimitadores
    - Agregar volt como un servicio
    - Trabajar con variables, arreglos, filtros y estructuras de control
- El enrutamiento se utiliza para definir controladores, y acciones (con o sin paramentos) que responderán a una URL ingresada por el usuario.
- Configuramos una conexión a base de datos y también hablamos acerca de los adaptadores con los que **Phalcon** cuenta.
- Creamos consultas con *Phalcon Query Language* (PHQL) que es un lenguaje de alto nivel para realizar consultas SQL.
- HTTP (*Hypertext Transfer Protocol* o HTTP) es un protocolo de transferencia de información y que permite la interacción entre el cliente y el usuario. De este tema aprendimos a recibir peticiones por los métodos GET y POST. A grandes rasgos GET manda por URL las variables mientras que POST las oculta.
- Assets es un componente que nos permite agregar recursos para mejorar la presentación de una página.

Capítulo 2: Vistas: hojas de estilo, Bootstrap y jQuery

2.1 Introducción

Este capítulo está enfocado al desarrollo *front-end*, es decir, la parte encargada de darle un estilo personalizado acorde al diseño de quien solicita el proyecto. El *front-end* no debe quedar de lado ya que es la parte que complementa a la lógica de negocio.

Aquí hablaremos de las tecnologías más comunes que se utilizan por el lado del cliente como es HTML, CSS (lenguaje de programación de diseño) y JavaScript (lenguaje de programación interpretado), aunque existen *frameworks* para estos dos tipos de lenguajes *y* que nos ahorran mucho tiempo al momento de diseñar.

Estos *frameworks* son:

- Bootstrap. Permite dar un diseño a los elementos HTML para páginas web de escritorio o móvil.
- jQuery. Permite manipular elementos HTML para que realicen algún evento, también permiten interacción con la técnica AJAX.

Hasta hoy en día son usados en el trabajo, por ello considero que es importante conocer.

2.2 Hojas de estilo CSS

Las Hojas de estilo en cascada (del inglés *Cascading Stylesheets*) o CSS es un lenguaje de diseño para dar formato a los elementos escritos en HTML de una página web.

La forma en que trabajan las hojas de estilo es la siguiente:

- En la vista se escriben todos los elementos correspondientes en HTML.
- Las hojas de estilo son documentos que deben ser incluidos para dar diseño a los elementos HTML (color, tamaño, fuente, posición, margen, etc.).

En el tema Assets del capítulo anterior vimos como agregar recursos a nuestro proyecto en **Phalcon**, que resumido en un diagrama sería el de la **Figura 2.1**.

Figura 2.1 Proceso de importación y definición de estilos en **Phalcon**. Primero se carga el archivo CSS como un recurso (“assets.php”) y después importamos el recurso en el la plantilla principal “index.volt”.

2.2.1 Sintaxis de las hojas de estilo

Una hoja de estilo contiene selectores donde se declaran las reglas de diseño, cada declaración a un selector debe ir encerrado por llaves.

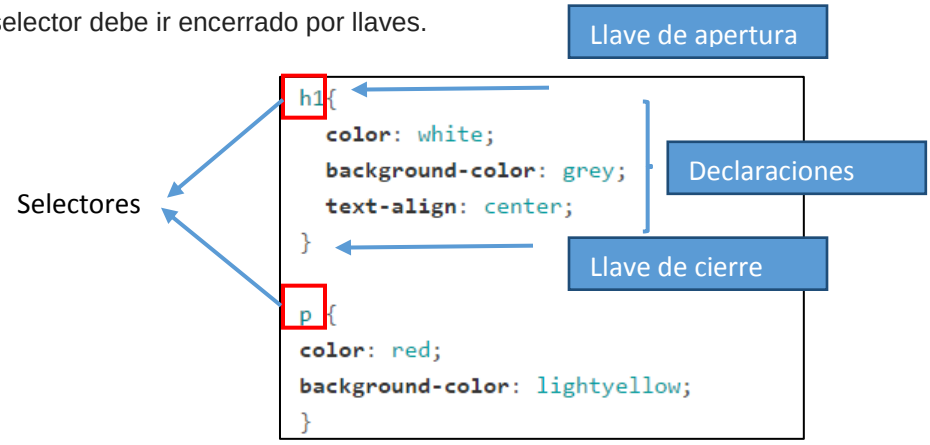


Figura 2.2 Sintaxis de las hojas de estilo.

- Cada declaración va acompañada por su propiedad-valor correspondiente separadas por “:”.
- Las declaraciones deben estar separadas por “;”.

- Un conjunto de declaraciones hacia un mismo selector va separado por llaves.
- Los selectores pueden ser elementos HTML, clases, id´s, etc.

Si queremos definir más de un selector debemos separarlos por comas.

```
h1,p{
   color: white;
   background-color: grey;
   text-align: center;
}
```

Figura 2.3 Selección de elementos.

Para agregar comentarios a las hojas de estilo debemos utilizar los delimitadores /* … */

```
/* agregamos estilos a las etiquetas
   <h1> y <p> */
h1,p{
   color: white;
   background-color: grey;
   text-align: center;
}
```

Figura 2.3 Uso de comentarios.

2.2.2 Propiedades por medio de clases y por id

Cuando necesitemos agregar un estilo a varios elementos podemos asignarles un nombre genérico a todos los elementos que lo necesiten, esto puede darse por medio de clases. Si de lo contrario solo queremos aplicar el estilo a un único elemento podemos usar id, cada id debe tener un nombre diferente.

Tipo de definición	Sintaxis	Ejemplo

Clase	.nombre_clase	.texto{
		
		}

Id	#nombre_id	#imagen{
		
		}

Tabla 2.1 Selección de elementos por clase o id.

```css
.resaltado{
  color:#000000;
  background-color:#ffff00;
  font-style:italic;
}

#padre{
  background-color: black;
  color: white;
}
```

```html
<p class="resaltado">Texto resaltado</p>
<p>
  así es como se utliza una
  <span class=resaltado>clase</span>
</p>

<div id="padre">
    <h4>utilizando id</h4>
</div>
```

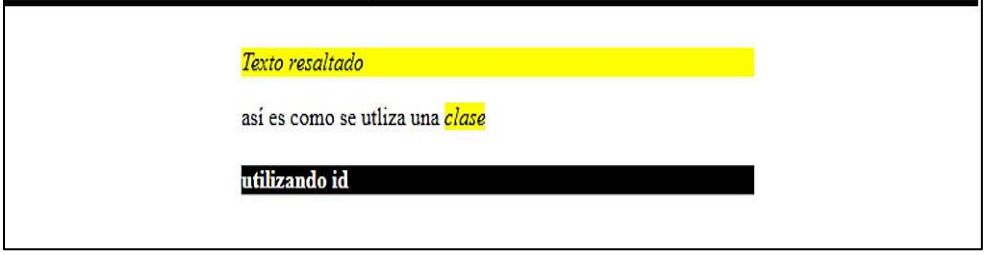


Figura 2.4 Ejemplos de los selectores.

2.2.3 Herencia de estilos

La mayoría de estilo se heredan del elemento padre a los elementos hijos, por ejemplo, <body> es el elemento padre de todos los elementos que agregamos (etiquetas <h1>, <p>, , etc). De la misma forma si tenemos un elemento donde le definimos un color al texto y dentro agregamos una etiqueta <h1>, el color de esta etiqueta será del mismo definido en el .

Por lo tanto, si agregamos estilos al elemento <body> o cualquier otro elemento, este los heredará a los elementos que están dentro.

```css
body{
  color:blue;
  text-align: center;
}
#padre{
  color:red;
}
```

```html
<h1> herencia de estilos </h1>

<div id="padre">
    <h4>h4 es hijo del elemento div</h4>
</div>
```

Figura 2.5 Ejemplos de herencia de estilos.

- Nota que <h1> es hijo de <body> por lo tanto hereda las propiedades color azul al texto y alineación centrado, mientras que el con id=”padre” define rojo para el texto por lo que ya no lo hereda de <body>, pero si hereda la alineación ya que no fue redefinida.

2.2.4 Bordes a los elementos

Todos los elementos que agregamos dentro de la estructura HTML generan “cajas” que encierran al elemento, no se ven porque por defecto se encuentran ocultas, pero si agregamos a cualquier etiqueta la propiedad *background-color* notaremos un rectángulo relleno con el color introducido.

Figura 2.6 Borde de los elementos.

Ahora que ya sabemos esto, es posible darles estilo a estos bordes. Las siguientes tablas muestran algunas propiedades de bordes, así como los valores que pueden tomar.

Propiedad	Descripción	Valores
border	Genera los 4 bordes de la caja así como su color, grosor y estilo	-border color -border-width -border-style
border-top	Genera el borde superior	-border color -border-width -border-style
border-right	Genera el borde derecho	
border-bottom	Genera el borde inferior	
border-left	Genera el borde izquierdo	
border-color	Aplica un color a los bordes	-Colores RGB, HTML, hexadecimal

border-width	Define un grosor de bordes	Medidas en pixels (px)
border-style	Define un estilo a los bordes	none, hidden, dashed, dotted, double, groove, inset, outset ridge, solid

Tabla 2.2 Estilos de borde (parte 1).

Figura 2.7 Ejemplos de uso de los estilos de borde.

Propiedad	Descripción	Valores
border-radius	Genera un borde circular	Valores en px
border-top-right-radius	Genera radio de borde para cada esquina señalada	Valores en px
border-bottom-right-radius		
border-bottom-left-radius		
border-top-left-radius		

Tabla 2.3 Estilos de borde (parte 2).

Los ejemplos de uso se muestran en la siguiente Figura:

```css
.frase1 {
  font-family: cursive;
  text-align: right;
  padding: 3px 10px;
  border: #7fffd4 5px solid;
  border-radius: 20px;
}

.frase2 {
  font-family: cursive;
  text-align: right;
  padding: 3px 50px;
  border-top-left-radius: 20px;
  border-bottom-right-radius: 20px;
  border: #7fffd4 5px solid;

}
```

```html
<div class="frase1">
  <p>La vida es muy simple pero insistimos en hacerla complicada. Confucio.</p>
</div>
<br>
<div class="frase2">
  <p>
    La mayor declaración de amor es la que no se hace; el hombre que siente mucho,
    habla poco. Platón.
  </p>
</div>
```

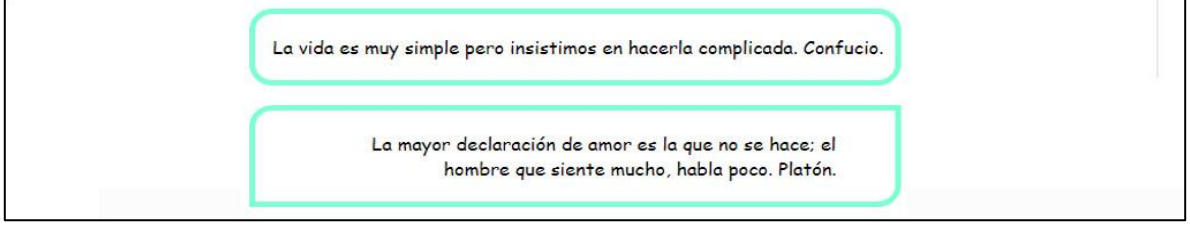


Figura 2.8 Ejemplos de uso de los estilos de borde.

2.2.5 Propiedades al texto

En la tabla siguiente veremos algunas propiedades que determinaran la apariencia del texto.

Propiedad	Descripción	Valores
Text-align	Determina la alineación de los elementos (no solo texto)	-left (izquierda) -right (derecha) -center (centro) -justify (justificado)
text-decoration	-decora el texto	-none -underline (subrayado)
text-transform	Convierte el texto de mayúsculas a minúsculas y viceversa	-uppercase (mayúscula) -lowercase (minúscula)
word-spacing	Controla la separación de letras en	- medida en px

	un texto	
color	Determina el color del texto	-Colores RGB, HTML

Tabla 2.4 Propiedades del texto.

Los ejemplos de uso se muestran en la siguiente figura:

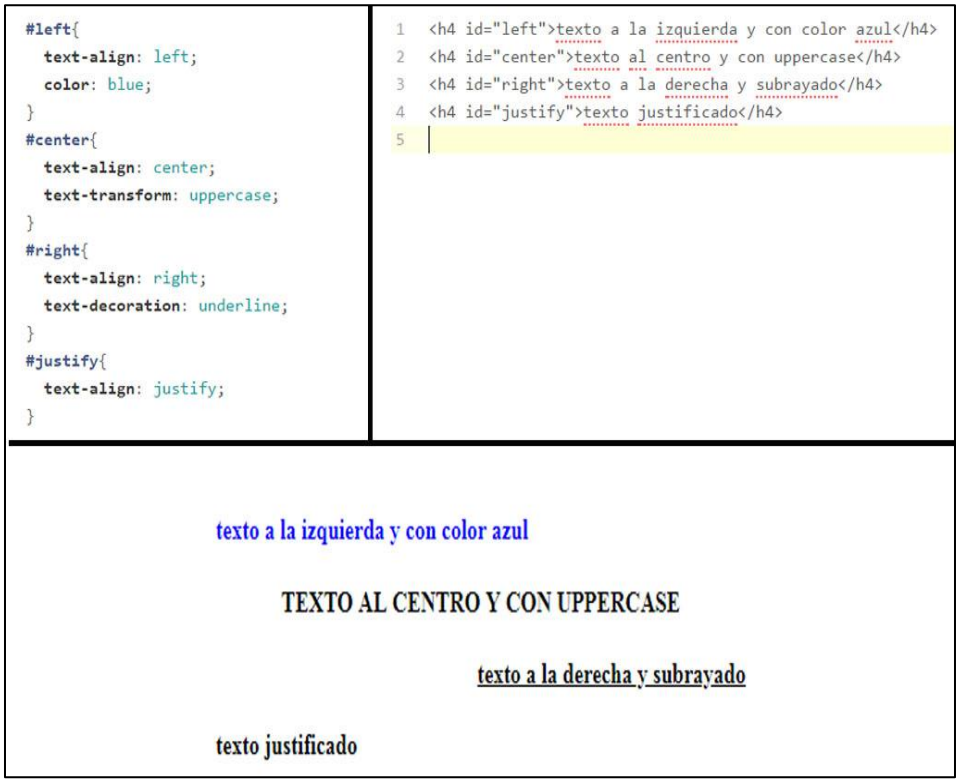


Figura 2.9 Ejemplos de uso de los estilos de texto.

2.2.6 Fuentes

En el tema anterior vimos propiedades relacionadas a la alineación y color del texto, ahora veremos las propiedades a las fuentes como el tipo de letra, tamaño de letra, grosor, interlineado, itálicas, etc. La siguiente tabla un breve resumen de las propiedades más comunes de uso.

Propiedad	Descripción	Valores
font	Determina todas las propiedades relacionadas con el tipo de letra	-font-style, font-variant, font-weight, font-size, line-height, font-family
font-family	Determina el tipo de letra	-Tipos de letra

font-size	Determina el tamaño de letra	-Tamaño absoluto, tamaño relativo, distancia o porcentaje
font-style	Convierte el texto en itálica	-normal -italic -oblique
font-weight	Convierte el texto a negrita	- normal -bold Valores numéricos
line-height	Coloca un interlineado en el texto	-porcentajes o em

Tabla 2.5 Tipos de fuente.

```css
#fontfamily1{
  font-family: cursive;
}
#fontfamily2{
  font-family: sans-serif;
}
#italica-negrita{
  font-family: sans-serif;
  font-size: 110%;
  font-style: italic;
  font-weight: bold;
}
#font1{
  font: bold 90% monospace;
  color: red;
}
#font2{
  font: oblique bold 90% cursive;
}
```

```html
<p id="fontfamily1">
  “La vida es como una caja de bombones,
  nunca sabes lo que te va a tocar.”
</p>
<p id="fontfamily2">
  “La vida es como una caja de bombones,
  nunca sabes lo que te va a tocar.”
</p>
<p id="italica-negrita">
  “La vida es como una caja de bombones,
  nunca sabes lo que te va a tocar.”
</p>
<p id="font1">
  “La vida es como una caja de bombones,
  nunca sabes lo que te va a tocar.”
</p>
<p id="font2">
  “La vida es como una caja de bombones,
  nunca sabes lo que te va a tocar.”
</p>
```

“La vida es como una caja de bombones, nunca sabes lo que te va a tocar.”

“La vida es como una caja de bombones, nunca sabes lo que te va a tocar.”

“La vida es como una caja de bombones, nunca sabes lo que te va a tocar.”

“La vida es como una caja de bombones, nunca sabes lo que te va a tocar.”

“La vida es como una caja de bombones, nunca sabes lo que te va a tocar.”

Figura 2.10 Ejemplos de uso de los tipos de fuente.

2.2.7 Margen interior

El margen interior se refiere al espacio que existen entre el borde y el contenido de un elemento. Como sabemos, el borde existe, aunque no esté definido, solo que se encuentra transparente.

Para poder aplicar un margen interior debemos utilizar la propiedad *padding.* Por medio de esta propiedad podemos definir márgenes a los cuatro bordes, uno o algunos dependiendo del número valores que ingresamos:

1. Aplica margen a los cuatro lados.
2. El primer valor se aplica a los lados superior e inferior y el segundo a los lados izquierdo y derecho.
3. El primer valor aplica al lado superior, el segundo a los lados izquierdo y derecho y, el tercer lado al inferior.
4. Se aplican en el siguiente orden: superior, derecho, inferior, izquierdo.

Los ejemplos de uso de la propiedad *padding* se muestran en la siguiente figura:

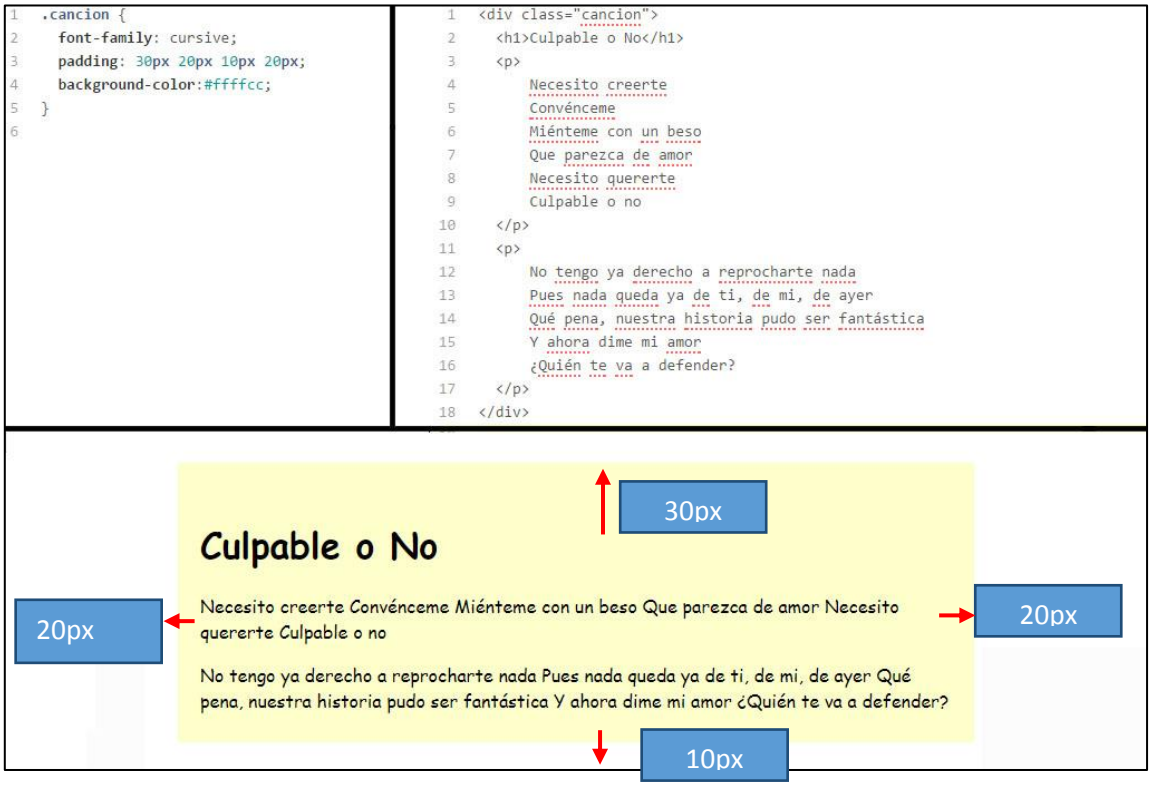


Figura 2.11 Ejemplos de uso del margen interior (*padding*).

2.2.8 Margen exterior

El margen hace referencia entre el espacio que existe entre el elemento padre del elemento hijo independientemente de que el borde este visible o no. Es posible definir si queremos agregar un margen a un lado, a dos, a tres o a los cuatro si seguimos la siguiente regla:

1. Aplica margen a los cuatro lados.
2. El primer valor se aplica a los lados superior e inferior y el segundo a los lados izquierdo y derecho.
3. El primer valor aplica al lado superior, el segundo a los lados izquierdo y derecho y el tercer lado al inferior.
4. Se aplican en el siguiente orden: superior, derecho, inferior, izquierdo.

Ejemplos de uso de márgenes podemos visualizarlo en la figura siguiente:

Figura 2.11 Ejemplos de uso del margen exterior (*margin*).

2.2.9 Fondos

Podemos agregar un fondo a casi todos los elementos HTML, puede ser una imagen o algún color existente de la gran cantidad de gama de colores llamándolos por nombre, su hexadecimal, RGB, etc. La siguiente tabla (**2.6**) muestra algunas propiedades referentes a el fondo.

Propiedad	Descripción	Valores
background	Define las propiedades de cualquier elemento	-background-color, background-image, background-position, background-size, background-repeat, background-attachment, background-origin y background-clip
background-color	Define el color e fondo de un elemento	-Valores de color
Background-imagen	Define una imagen como fondo de un elemento	-URL de la imagen
background-size	Define el tamaño de la imagen de fondo	-porcentajes o distancias -auto -contain -cover
background-repeat	Determina la posición en que se repite una imagen de fondo	-no-repeat -repeat-x -repeat-y -repeat

Tabla 2.6 Propiedades relacionadas al fondo.

Ejemplos de uso de las propiedades de fondo de la página las podemos encontrar en la **Figura 2.12**.

```css
body{
  background-color: MintCream;
}
.bloque1,.bloque2,.bloque3 {
  text-align: center;
  font-size: 20px;
  width: 300px;;
  height: 200px;
  margin-left: auto;
  margin-right: auto;
  display: inline-block;
}
.bloque1{
  background-image: url('../img/suarez.png');
  background-size: 75px 50px;
  background-repeat: repeat;
}
.bloque2{
   background: #FFDF85 url("../img/fuego.png");
   background-size: contain;
   background-repeat: no-repeat;

}
.bloque3{
  background-image: url('../img/fondo.jpg');
  background-repeat: space;
  background-size: cover;
}
```

```html
<div class="bloque1">
  <p>bloque 1</p>
</div>

<div class="bloque2">
  <p>bloque 2</p>
</div>

<div class="bloque3">
  <p>bloque 3</p>
</div>
```

Figura 2.12 Ejemplos de uso de la propiedad *background*.

2.2.10 Posicionamiento relativo

Para colocar un elemento en determinada posición (en pixeles) podemos utilizar la propiedad *position*. Por defecto el valor de esta propiedad es *static* donde los elemento agregados se ordenan de izquierda a derecha y de arriba hacia abajo. El valor *relative* hace que modifiquemos estos valores para indicar el punto de partida donde se colocara el elemento.

```css
body{
  background-image: url("../img/estadio.jpeg");
}
.posicion-relativa{
  position:relative;
  left:100px;
  top:180px;
}
```

```html
<img class="posicion-relativa" src="img/suarez.png">
```

Figura 2.13 Ejemplos de uso del posicionamiento relativo.

2.2.11 Posicionamiento absoluto y propiedad z-index

La propiedad posición absoluta se refiere a cuando un elemento se superpone a otro. Cuando colocamos imágenes, texto, etc. en la misma posición se colocarán una encima de otra ocultando las anteriores, si nosotros quisiéramos controlar el orden podemos utilizar la propiedad z-index indicando en sus valores la prioridad de visualización.

Veamos la siguiente imagen:

Notamos dos coches donde el de color rojo está por encima del coche azul, si tuviéramos que hacerlo utilizando CSS tendríamos algo como esto:

```css
body{
  background-image: url("../img/carretera.jpg");
}
#caja1{
  position:absolute;
  left:650px;
  top:500px;
  z-index:1;
}
#caja2{
  position:absolute;
  left:600px;
  top:550px;
  z-index:2;
}
.azul{
  width: 300px; height: 120px;
}
.rojo{
  width: 320px; height: 140px;
}
```

```html
<div class ="contenedor">
    <div id="caja1">
      <img class="azul" src="img/coche-azul.png">
    </div>
    <div id="caja2">
      <img class="rojo" src="img/coche-rojo.png">
    </div>
</div>
```

Con z-index indicamos que elemento se mostrara primero, en este caso primero se muestra el coche azul (1) y después el rojo (2). El elemento con mayor valor quedara por encima de los demás

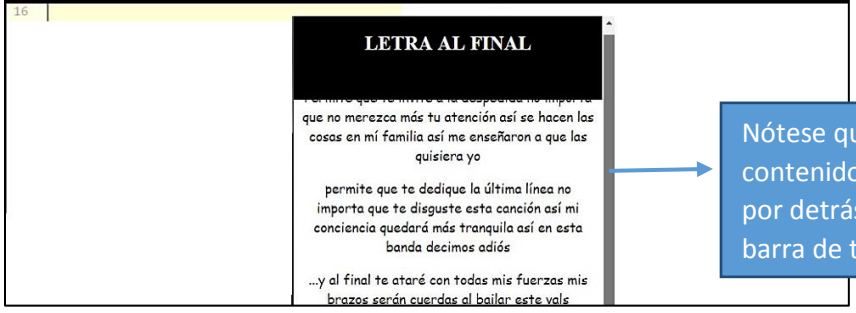
Figura 2.14 Ejemplos de uso del posicionamiento absoluto y la propiedad *z-index*.

2.2.12 Posicionamiento fijo (fixed)

El posicionamiento fijo ancla cualquier elemento en una coordenada determinada en el navegador, de esta manera siempre será visible en su posición a pesar de que se esté navegando entre el cuerpo de la página. Este tipo de posicionamiento es muy común en los menús de navegación haciendo que se encuentren siempre visibles a pesar de que se haga *scroll*, el contenido pasara por detrás.

```css
#barra-titulo{
  position:fixed;
  left:0px;
  top: 0px;
  width:100%;
  height:100px;
  background-color:black;
  color:white;
  text-align: center;
}
#cancion{
  padding-top:70px;
  font-family: cursive;
  text-align: center;
}
```

```html
<div id="barra-titulo">
    <h2>LETRA AL FINAL</h2>
</div>
<div id="cancion">
    <p> Permite que te invite a la despedida
    no importa que no merezca más tu atención
    así se hacen las cosas en mí familia
    así me enseñaron a que las quisiera yo
    </p>
    <p>
      permite que te dedique la última línea
      no importa que te disguste esta canción
      así mi conciencia quedará más tranquila
      así en esta banda decimos adiós
    </p>
```



Nótese que el contenido pasa por detrás de la barra de titulo

Figura 2.15 Ejemplos de uso del posicionamiento fijo.

2.2.13 Propiedad float

Esta propiedad permite colocar un elemento a la izquierda o derecha de la pantalla, y los elementos ingresados después fluirán alrededor del elemento flotante.

A continuación, se presentan algunos ejemplos de uso:

```css
.primer-letra {
  float:left;
  font-family: sans-serif;
  color: red;
  font-size:30px;
}
.flotante {
      float: left;
      width: 200px;
      height: 120px;
      margin: 3px;
      padding: 3px;
    }
.izquierda {
      float: left;
      width: 200px;
      height: 120px;
      margin: 3px;
      padding: 3px;
    }
.derecha {
      float: right;
      width: 200px;
      height: 120px;
      margin: 3px;
      padding: 3px;
    }
.versus{
  font-family: sans-serif;
  color: red;
  font-size:50px;
  position: relative;
  left: 90px;
}
```

```html
<p>
  <span class="primer-letra">Y</span>
  o sé que soy de tu agrado<br>
  no niegues en darme el sí <br>
  que yo te he ofrecido a ti<br>
  un matrimonio sagrado<br>
</p>

<img src="img/coche-rojo.png" class="flotante">
<img src="img/coche-rojo.png" class="flotante">
<img src="img/coche-rojo.png" class="flotante">
<img src="img/coche-rojo.png" class="flotante">
<img src="img/coche-rojo.png" class="flotante">
<img src="img/coche-rojo.png" class="flotante">

<p>
  <img class="izquierda" src="img/equipo1.png"/>
  <img src="img/equipo2.png" class="derecha"/>
  <span class ="versus">VS</span>
</p>
```

Figura 2.16 Ejemplos de uso de la propiedad *float*.

2.2.14 Selectores

Anteriormente vimos que podemos seleccionar elementos por medio de su clase o id, también por su etiqueta, pero existe otra manera de seleccionar elementos, la siguiente tabla muestra algunas formas importantes que es necesario conocer:

Tipo de selector	Descripción
Selector de hijos “>”	Selecciona todos los elementos hijos del elemento deseado
Selector de hermano adyacente “+”	Selecciona el primer elemento que viene después del elemento deseado
Selector de atributo “[]”	Selecciona el elemento que contenga el atributo deseado.
first-child “:first-child”	Selecciona el primer elemento hijo del elemento padre
last-child “:last-child”	Selecciona el último elemento hijo del elemento padre

Tabla 2.7 Propiedades relacionadas al fondo.

Algunos ejemplos de uso se muestran la siguiente figura:

```css
/*selección de hijos de ul*/
ul>p{
  color:red;
}
/*selección de hermano adyacente de h1*/
h1+p{
  color:blue;
}
/*selección por el atributo target*/
a[target]{
  color:red;
  text-decoration: none;
}
/*selección por first-child  y last-child*/
ol li:first-child {
  color:red;
}
ol li:last-child {
  color:red;
}
```

```html
<ul>
  <p>parrafo color rojo</p>
  <li>Opción 1</li>
  <li>Opción 2</li>
  <li>Opción 2</li>
</ul>

<h1>Soy una etiqueta H1</h1>
<p>Este es el hermano adyacente de H1</p>

<a href="http://www.youtube.com" target="_blank">Ir a youtube</a>

<p>Los mese marcados de rojo no estan disponibles</p>
<ol>
    <li>Enero</li>
    <li>Febrero</li>
    <li>Marzo</li>
    <li>Abril</li>
</ol>
```

Figura 2.17 Ejemplos de uso de los selectores.

2.3 Bootstrap

Bootstrap es un *framework* que nos va a permitir dar un diseño a la mayoría de elementos HTML, y no solo eso, también permite darle al sitio web un diseño responsivo, es decir, hace que el diseño se ajuste al tamaño de pantalla del dispositivo donde se visualice. Lo mejor de todo es que es código abierto (*Open Source*).

Estas son algunas ventajas que nos da utilizar el *framework*:

- Nos va a permitir una mejor maquetación. La maquetación se refiere a la forma en que se organizan todos los elementos de una página.
- Contiene iconos, colores, tablas, sliders y demás herramientas CSS y HTML.
- Es responsivo y se adapta a cualquier tamaño de pantalla. Esto hace que dejemos a un lado las *media queries* de CSS3.
- Se integra con otras librerías de JavaScript, por ejemplo, jQuery.
- Contiene una versión *minificada* que hace más liviano el código y por lo tanto la página carga más rápido.

2.3.1 Como obtener Bootstrap

Existen dos formas de incluir Bootstrap en nuestro sitio web:

1. Incluir el CDN para no descargarlo.
2. Descargarlo desde su página oficial.

2.3.1.1 Bootstrap CDN

Como mencionamos con anterioridad una CDN (*Content Delivery Network* en inglés) son un conjunto de servidores distribuidos en diferentes partes del mundo y que almacenan copias con contenido estático. Existen desventajas de su uso, por ejemplo: la posibilidad de que el servidor se caiga, no se pueden hacer pruebas a nuestro sitio de forma local ya que debemos estar forzosamente conectados a internet, problemas de compatibilidad si el CDN es modificado.

Bootstrap contiene un CDN para agregarlo de forma casi inmediata y poder utilizarlo, simplemente tenemos a su página oficial: *http://getbootstrap.com/* y dentro de esta buscar la sección "BootstrapCDN" que como vemos está en la versión 4.

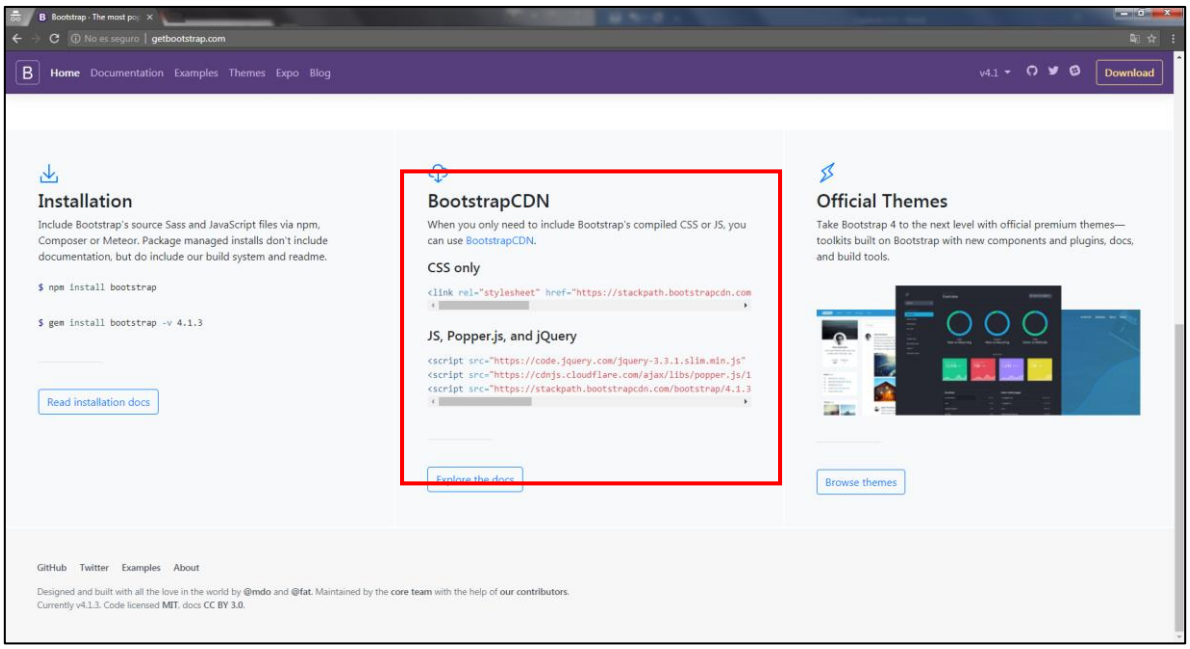


Figura 2.18 BoostrapCDN desde la página oficial.

Una vez ahí debemos copiar los CSS y JS correspondientes y colocarlos dentro de nuestro archivo "index" en **Phalcon** ubicado en *app/views.* Debe verse parecido a la **Figura 2.19**.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
        <title>Phalcon PHP Framework</title>
        <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPk
        <link rel="shortcut icon" type="image/x-icon" href="<?php echo $this->url->get('img/favicon.ico')?>"/>
    </head>
    <body>
        <div class="container">
            {{ content() }}
        </div>
        <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js" integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js" integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy"
    </body>
</html>
```

Figura 2.19 Como incluir el CDN a un proyecto en **Phalcon**.

Cuando creamos un proyecto nuevo utilizando las *herramientas de desarrollo* en **Phalcon**, que por defecto carga Bootstrap pero en la versión 3, así que solo tenemos que remplazar lo que está escrito o podemos utilizar la versión anterior aunque existen mucha diferencia entre estas versiones (3 y 4).

2.3.1.2 Descargando Bootstrap 4

Esta es la manera más recomendable de importar Bootstrap a nuestro proyecto, así evitamos los posibles fallos que conlleva utilizar un CDN.

Dentro de su página oficial http://getbootstrap.com/ nos dirigimos a "*Download*" y de ahí descargamos el Zip que incluye el CSS y JS necesarios.

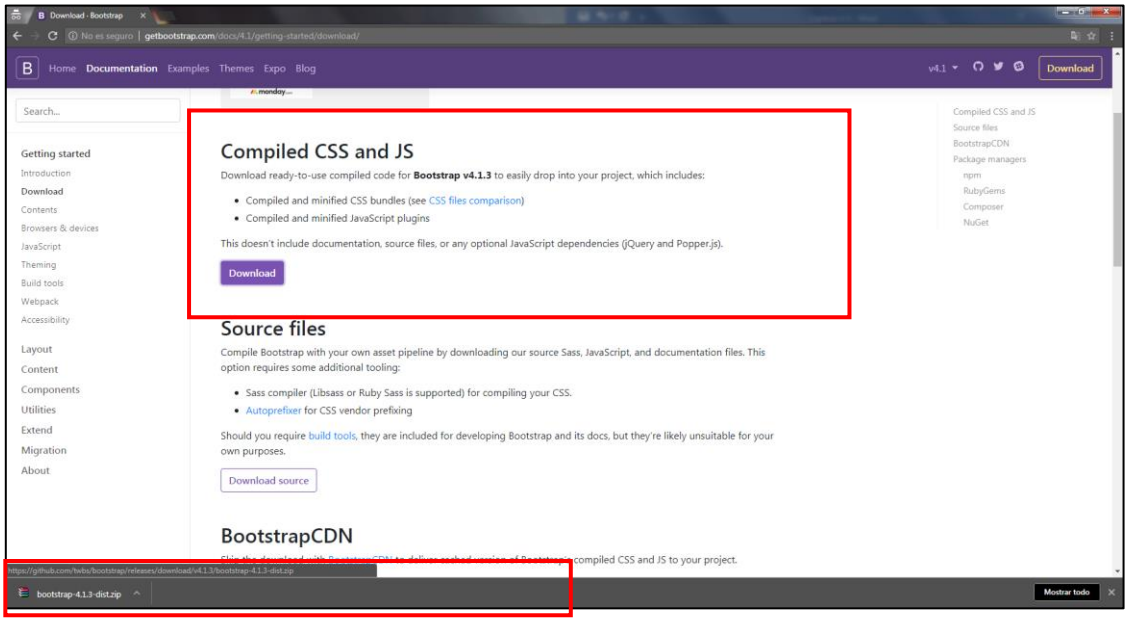


Figura 2.20 como descargar Bootstrap desde su página oficial.

Una vez descomprimido el Zip vamos a encontrar dos carpetas: css y js. Dentro de css tomamos el archivo *boostrap.css* o su versión minificada *bootstrap.min*.

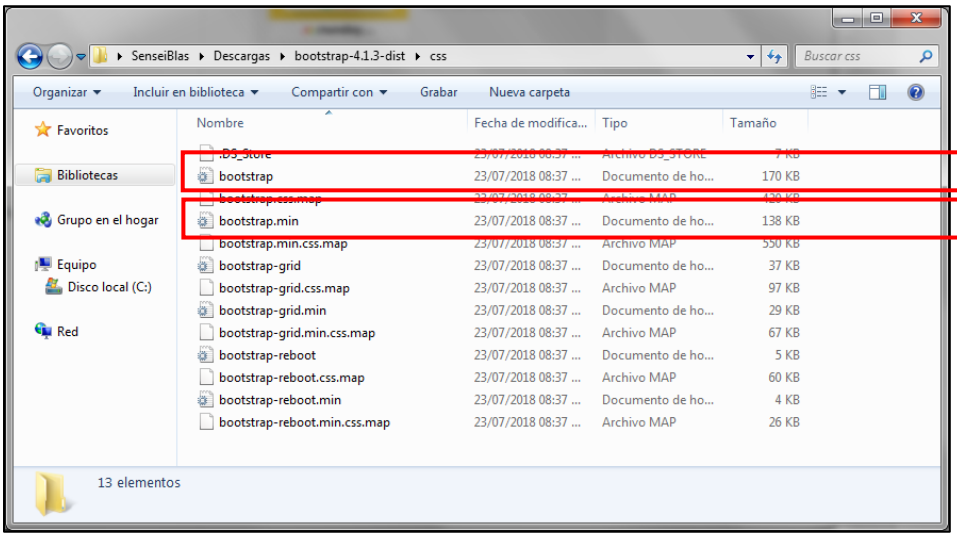


Figura 2.21 Archivos CSS de Bootstrap en su versión normal y minificada.

Lo mismo para la carpeta JS podemos tomar *bootstrap.js* o su versión minificada *bootstrap.min.*

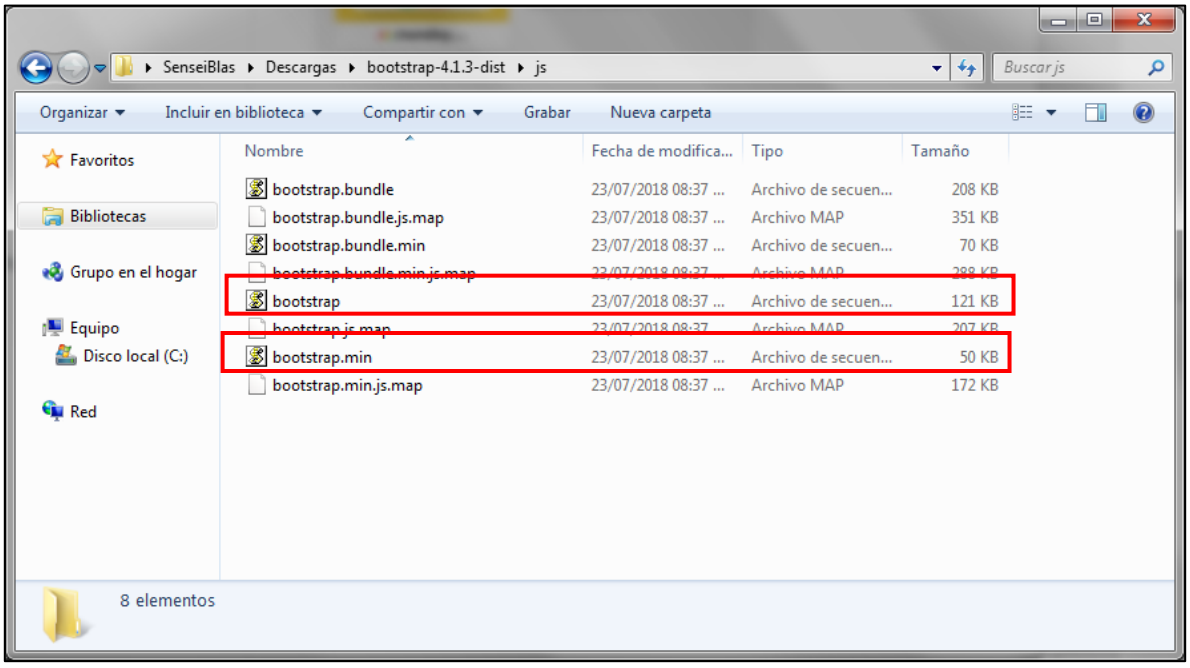


Figura 2.22 Archivos js de Bootstrap en su versión normal y minificada.

El directorio donde debemos copiar estos archivos es en “assets” ubicada en la carpeta *public*, Una vez dentro de sus carpetas correspondientes.

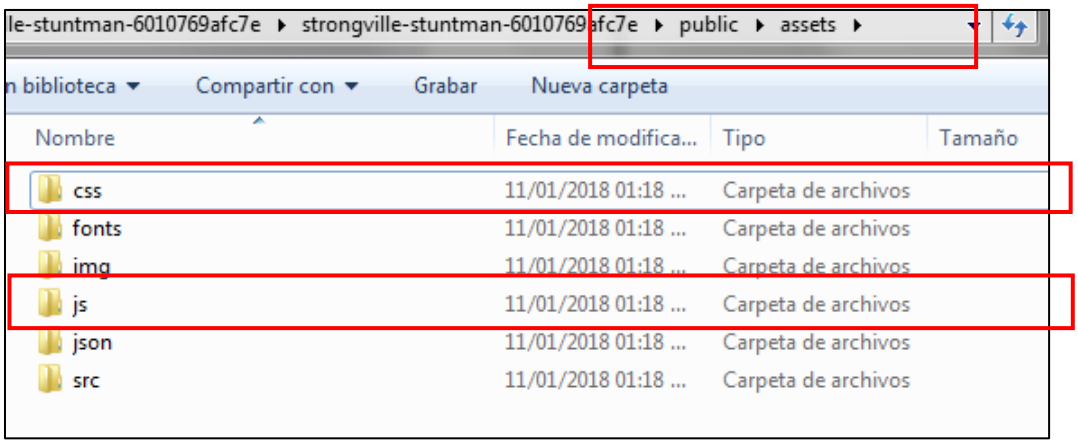


Figura 2.23 Carpetas en **Phalcon** donde deben colocarse los archivos descargados de Bootstrap.

Para utilizarlos simplemente debemos agregarlos como recursos con Assets como lo vimos en el capítulo anterior (tema **1.12**) solo que ahora colocando el directorio donde se encuentra cada archivo.

- CSS: '/assets/css/bootstrap.min.css'
- JS: '/assets/js/bootstrap.min.js'

2.3.2 Sistema de rejillas

Como habíamos mencionado, Bootstrap es responsivo y puede adaptarse a diferentes tamaños de pantalla, desde dispositivos móviles hasta computadoras de escritorio. El sistema de rejillas o de grillas nos va a permitir maquetar un sitio utilizando filas y columnas. Se pueden manejar "n" número de filas, pero solo hasta 12 columnas, en donde nosotros podemos definir el tamaño que ocuparán, también podemos indicar como van a ordenarse cuando el tamaño de la pantalla aumente o disminuya.

2.3.2.1 Clases de cuadriculas

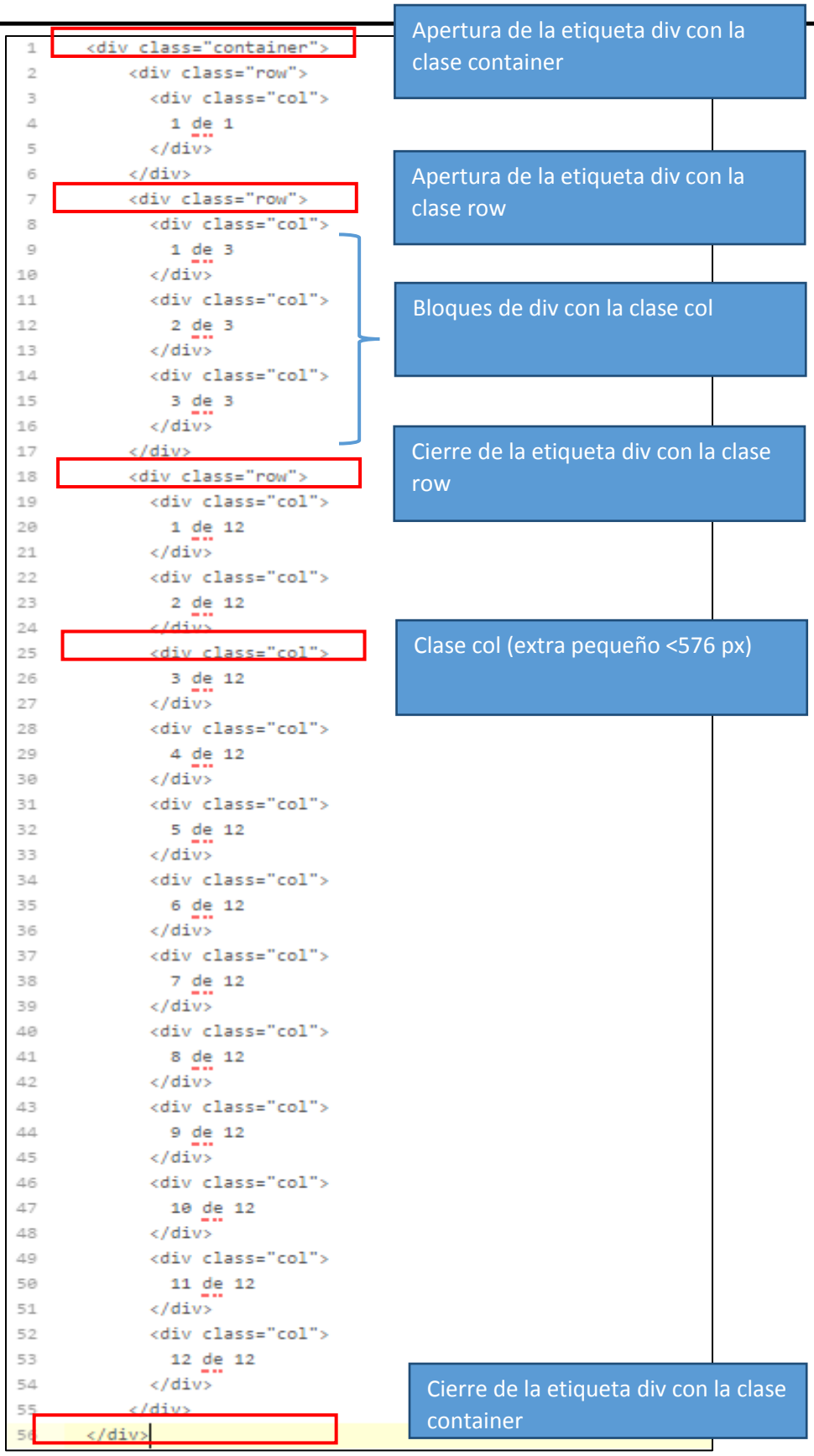
Bootstrap cuenta con cinco diferentes clases que se “activan” dependiente el tamaño de la pantalla.

Prefijo	Tamaño	Numero de columnas	Orden de las columnas
.col-	Extra pequeño <576 px	12	automático
.col-sm-	Pequeño ≥576 px	12	automático
.col-md-	Medio ≥768 px	12	automático
.col-lg-	Grande ≥992 px	12	automático
.col-xl-	Extra grande ≥1200 px	12	automático

Tabla 2.8 Clases de cuadricula

Para poder entender la tabla anterior, primero veremos un ejemplo sencillo utilizando la clase *.col-*

El HTML se muestra en la **Figura 2.24**.




```
1   <div class="container">
2       <div class="row">
3         <div class="col">
4           1 de 1
5         </div>
6       </div>
7       <div class="row">
8         <div class="col">
9           1 de 3
10        </div>
11        <div class="col">
12          2 de 3
13        </div>
14        <div class="col">
15          3 de 3
16        </div>
17      </div>
18      <div class="row">
19        <div class="col">
20          1 de 12
21        </div>
22        <div class="col">
23          2 de 12
24        </div>
25        <div class="col">
26          3 de 12
27        </div>
28        <div class="col">
29          4 de 12
30        </div>
31        <div class="col">
32          5 de 12
33        </div>
34        <div class="col">
35          6 de 12
36        </div>
37        <div class="col">
38          7 de 12
39        </div>
40        <div class="col">
41          8 de 12
42        </div>
43        <div class="col">
44          9 de 12
45        </div>
46        <div class="col">
47          10 de 12
48        </div>
49        <div class="col">
50          11 de 12
51        </div>
52        <div class="col">
53          12 de 12
54        </div>
55      </div>
56  </div>
```

Figura 2.24 Ejemplo de usos de la clase *.col-*.

- La clase *container* coloca un margen fijo y centrado al contenido que se encuentre dentro de la etiqueta con la clase. El margen se mantiene intacto a pesar que se cambie de tamaño el navegador.

```
<div class="container">
    ....
</div>
```

- La clase *row* sirve para colocar las filas y dentro poder colocar las columnas. De esta manera las columnas son elementos hijos de las filas y a su vez estas del contenedor (*container*).

```
<div class="container">
  <div class="row">
    <div class="col">
      ....
    </div>
  ....
  </div>
</div>
```

- Al utilizar la clase *col* significa que el ancho se aplica a dispositivos extra pequeños y a todos los dispositivos en general por lo que la distribución de las columnas se mantiene igual en los diferentes tamaños de la pantalla.

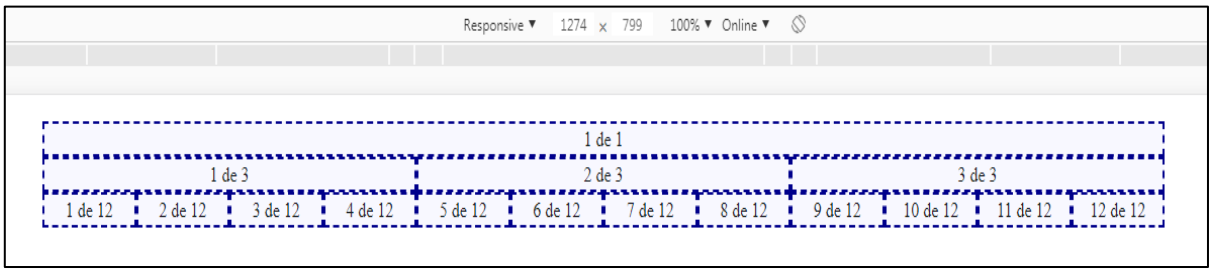


Figura 2.25 Visualización en el navegador a 1274 px.

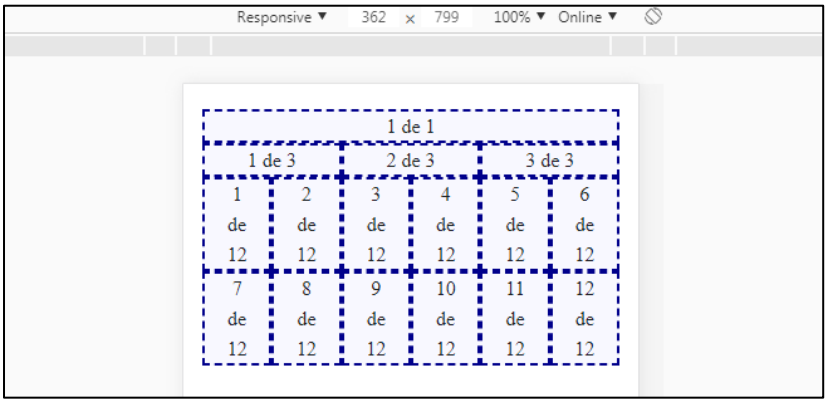


Figura 2.26 Visualización en el navegador a 362 px.

Bien ahora vamos a ver otro ejemplo donde utilizaremos las demás clases de cuadriculas y posteriormente haremos una explicación:

```
<div class="container">
      <div class="row">
            <div class="col-md-1">.col-md-1</div>
            <div class="col-md-1">.col-md-1</div>
            <div class="col-md-2">.col-md-2</div>
            <div class="col-md-2">.col-md-2</div>
            <div class="col-md-3">.col-md-3</div>
            <div class="col-md-2">.col-md-2</div>
            <div class="col-md-1">.col-md-1</div>
        </div>
        <div class="row">
          <div class="col-md-2">.col-md-2</div>
            <div class="col-md-3">.col-md-3</div>
            <div class="col-md-7">.col-md-7</div>
        </div>
        <div class="row">
          <div class="col-md-4">.col-md-4</div>
            <div class="col-md-4">.col-md-4</div>
            <div class="col-md-4">.col-md-4</div>
        </div>
        <div class="row">
          <div class="col-md-5">.col-md-5</div>
            <div class="col-md-7">.col-md-7</div>
        </div>
        <div class="row">
          <div class="col-md-6">.col-md-6</div>
            <div class="col-md-6">.col-md-6</div>
        </div>
        <div class="row">
          <div class="col-md-12">.col-md-12</div>
        </div>
    </div>
</div>
```

Figura 2.27 Ejemplo de usos de la clase *.col-md.*

- Dentro de la primera fila hemos colocado 7 columnas con la clase *col-md-*, esto quiere decir que podemos dar los tamaños que se desean a modo que den **12** columnas en total:

  1+1+2+2+3+2+1 = 12 columnas

- Realizamos la misma distribución de columnas para las demás filas.

Si lo visualizamos en el navegador observamos lo siguiente:

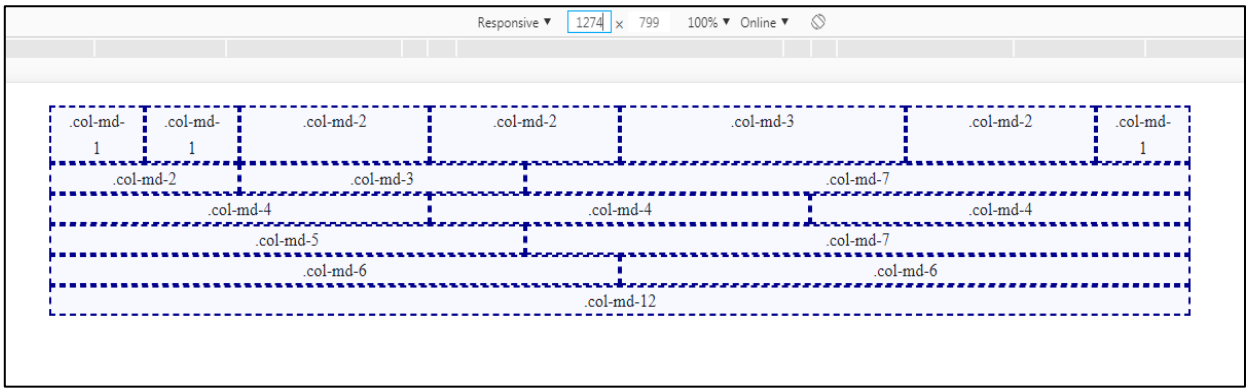


Figura 2.29 Visualización en el navegador a 1274 px.

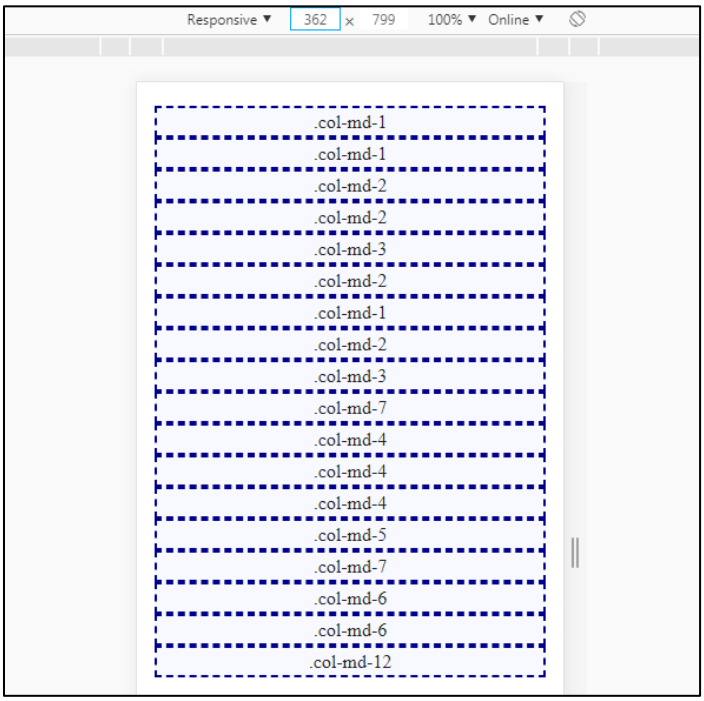


Figura 2.29 Visualización en el navegador a 362 px.

¿Notaste la diferencia? Revisemos a detalle: como el tamaño de las comunas se definió con la clase de cuadricula *col-md-*, es decir, tamaño medio (≥768 px) por lo tanto cuando sea mayor o igual a esos pixeles se vera de la forma a como fue maquetada, de lo contrario, cuando sea menor a este tamaño todas las columnas se colocarán en fila una debajo de la otra.

Ahora supongamos que quisiéramos mantener el mismo orden pero para tamaños de pantalla pequeños, para esto podemos combinar clases de cuadricula.

```
<div class="row">
  <div class="col-6 col-md-6">.col-md-6</div>
    <div class="col-6 col-md-6">.col-md-6</div>
</div>
```

Se utilizan dos tipos de clases para fijar el orden para cada tamaño de pantalla (tamaño extra pequeño y medio).

Si actualizamos el navegador veremos que cambian las últimas columnas y mantienen su orden de dos en una fila:

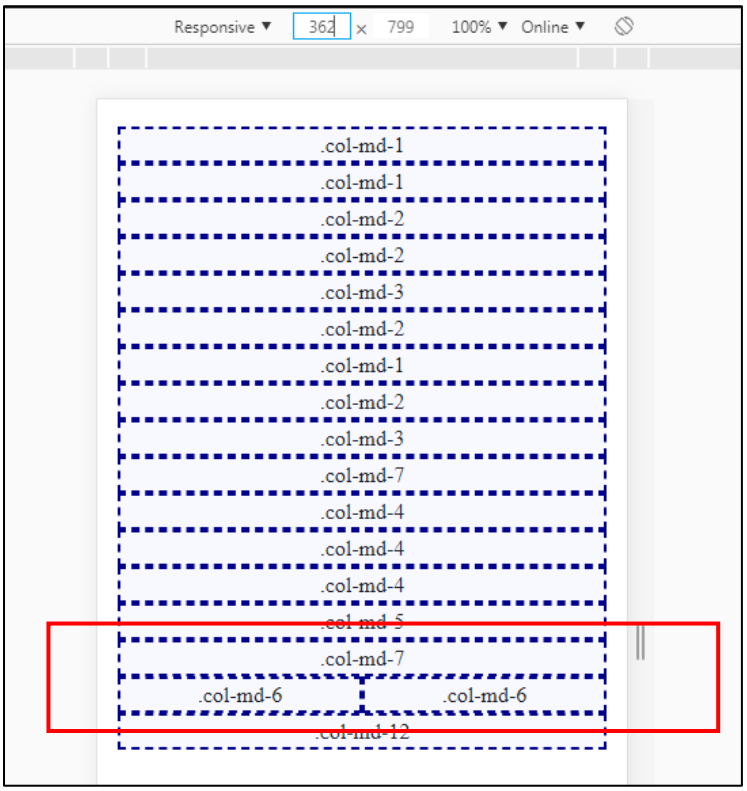


Figura 2.30 Se mantiene las dos columnas en una fila a pesar de que la pantalla se encuentra a 362 px.

2.3.3 Estilos rápidos al texto

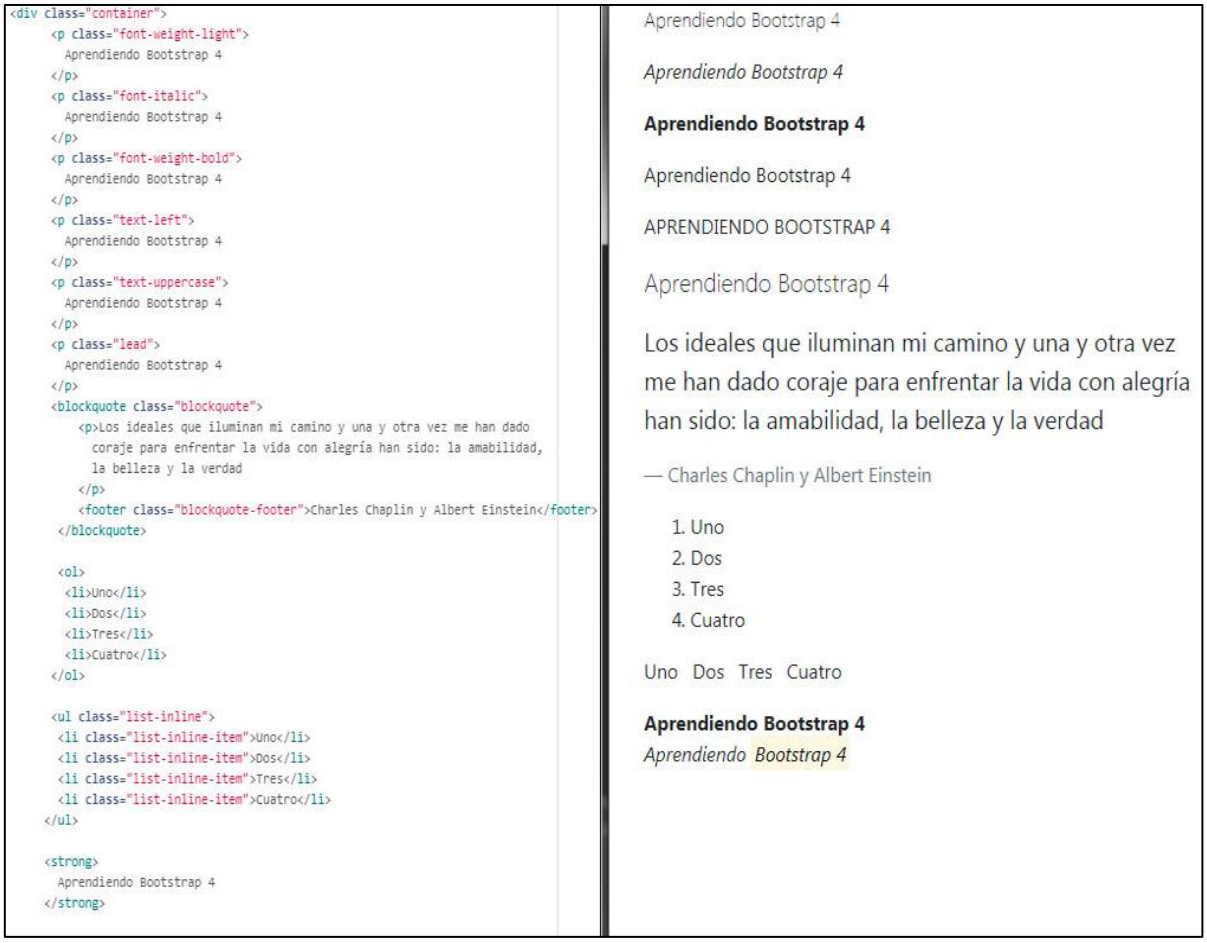
Bootstrap cuenta con clases y etiquetas que solo requieren ser incluirlas para darle un estilo rápido al texto sin necesidad de tener que crear un archivo CSS para definirlo nosotros mismos. A continuación, se muestra una tabla con las clases y etiquetas básicas comunes, que nos pueden sacar de apuros cuando se requiera.

Clase	Descripción
.font-weight-light	Letra fina
.font-italic	Texto en cursiva
.font-weight-bold	Texto en negrita
.text-left	Alineado a la izquierda
.text-center	Alineado al centro
.text-right	Alineado a la derecha
.text-justify	Texto justificado
.text-lowercase	Texto en minúsculas
.text-uppercase	Texto en mayúsculas
.lead	Destaca un párrafo
.list-inline	Lista los elementos en horizontal. Estos elementos deben ir seguidos de la clase *list-inline-item*

Tabla 2.8 Clases disponibles para el texto.

Etiqueta	Descripción
blockquote	Cita frases o párrafos. Se puede acompañar con la etiqueta *footer*. Para esto es necesario las clases *blockquote* y *blockquote-footer*
ol, li	Permite mostrar una lista ordenada
ul	Permite mostrar listas desordenadas
strong	Muestra el texto en negrita
em	Muestra el texto en cursiva

Tabla 2.9 Etiquetas utilizadas para el texto.



```
<div class="container">
    <p class="font-weight-light">
      Aprendiendo Bootstrap 4
    </p>
    <p class="font-italic">
      Aprendiendo Bootstrap 4
    </p>
    <p class="font-weight-bold">
      Aprendiendo Bootstrap 4
    </p>
    <p class="text-left">
      Aprendiendo Bootstrap 4
    </p>
    <p class="text-uppercase">
      Aprendiendo Bootstrap 4
    </p>
    <p class="lead">
      Aprendiendo Bootstrap 4
    </p>
    <blockquote class="blockquote">
        <p>Los ideales que iluminan mi camino y una y otra vez me han dado
          coraje para enfrentar la vida con alegría han sido: la amabilidad,
          la belleza y la verdad
        </p>
        <footer class="blockquote-footer">Charles Chaplin y Albert Einstein</footer>
    </blockquote>

    <ol>
      <li>Uno</li>
      <li>Dos</li>
      <li>Tres</li>
      <li>Cuatro</li>
    </ol>

    <ul class="list-inline">
      <li class="list-inline-item">Uno</li>
      <li class="list-inline-item">Dos</li>
      <li class="list-inline-item">Tres</li>
      <li class="list-inline-item">Cuatro</li>
    </ul>

    <strong>
      Aprendiendo Bootstrap 4
    </strong>
```

Aprendiendo Bootstrap 4

Aprendiendo Bootstrap 4

Aprendiendo Bootstrap 4

Aprendiendo Bootstrap 4

APRENDIENDO BOOTSTRAP 4

Aprendiendo Bootstrap 4

Los ideales que iluminan mi camino y una y otra vez me han dado coraje para enfrentar la vida con alegría han sido: la amabilidad, la belleza y la verdad

— Charles Chaplin y Albert Einstein

1. Uno
2. Dos
3. Tres
4. Cuatro

Uno Dos Tres Cuatro

Aprendiendo Bootstrap 4

Aprendiendo Bootstrap 4

Figura 2.31 Ejemplo de usos de estilos al texto mostrados en la tabla **2.8** y **2.9**.

2.3.4 Colores de texto y de fondo

También, podemos ingresar clases a las etiquetas HTML a modo que podamos darle un color al texto o agregar un color para el fondo, estas clases son las siguientes:

Clase	Descripción
text-muted	Texto en color gris claro
text-primary	Texto en color azul
text-success	Texto en color verde
text-info	Texto en color turquesa
text-warning	Texto en color amarillo

text-danger	Texto en color rojo
text-secondary	Texto secundario
text-dark	Texto en color negro

Tabla 2.10 Colores utilizando Bootstrap.

Las etiquetas anteriores podemos usarlas para indicar alertas según sea el contexto, estos podrían ser algunos ejemplos de uso:

```
<div class="container">
  <h2>Colores de contexto: </h2>
  <p class="text-muted">Esta enlace se encuentra desactivada hasta
    que te registres!
  </p>
  <p class="text-primary">Es importante que guardes cambios</p>
  <p class="text-success">Contraseña correcta</p>
  <p class="text-white bg-info">puedes combiar color de texto y fondo de texto</p>
  <p class="text-warning">¿Seguro que deseas continuar?</p>
  <p class="text-danger">El nombre de usuario o contraseña no son válidos</p>
  <p class="bg-danger text-white">El nombre de usuario o contraseña no son válidos</p>
  <p class="text-dark">texto en color negro</p>
</div>
```

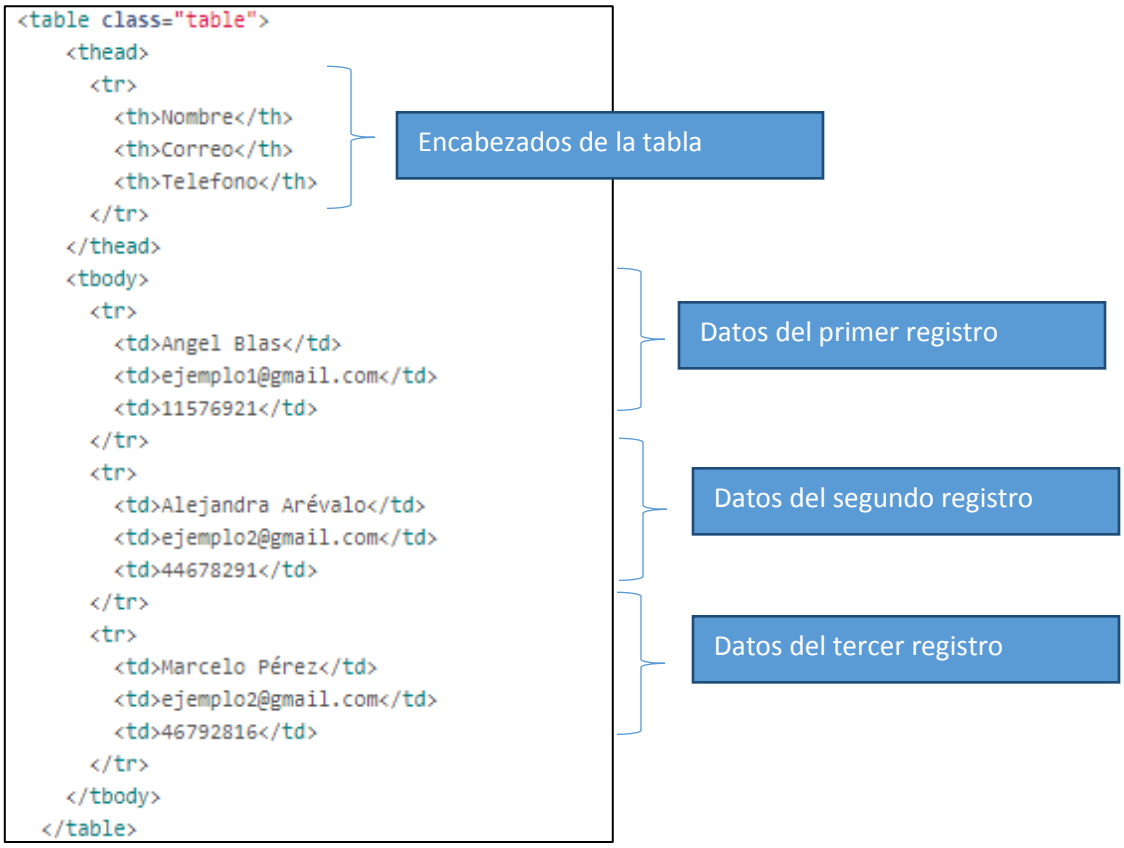
Figura 2.32 Ejemplo de usos de los colores de texto y fondo de los elementos.

Para cambiar el color de fondo solo tenemos que utilizar la clase *bg-* y en nombre del color, que es el mismo al que se usa para el texto. Por ejemplo, *bg-danger, bg-success, bg-info*, etc.

2.3.5 Tablas

Las tablas son un recurso importante entre los elementos HTML ya que por medio de filas y columnas agrupamos información que necesitemos presentar al usuario. Bootstrap posee clases para el manejo de tablas, a fin de darles un diseño agradable y con algunos efectos. Tiene estilos para todas las partes que conforman una tabla, desde los encabezados como a las filas y

columnas. También nos permitirá hacerlas responsivas, así que se adaptarán al tamaño de la pantalla en la que se visualiza.




```
<table class="table">
    <thead>
      <tr>
        <th>Nombre</th>
        <th>Correo</th>
        <th>Telefono</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>Angel Blas</td>
        <td>ejemplo1@gmail.com</td>
        <td>11576921</td>
      </tr>
      <tr>
        <td>Alejandra Arévalo</td>
        <td>ejemplo2@gmail.com</td>
        <td>44678291</td>
      </tr>
      <tr>
        <td>Marcelo Pérez</td>
        <td>ejemplo2@gmail.com</td>
        <td>46792816</td>
      </tr>
    </tbody>
  </table>
```

Figura 2.33 Estructura HTML para insertar una tabla utilizando Boootstrap.

Analizando la figura anterior tenemos lo siguiente:

- <table> y </table> corresponden a la apertura de la tabla. Se debe usar la clase *table*.
- <thead> y </thead> corresponden a la apertura y cierre de los encabezados, cada elemento debe ir entre etiquetas <th>.
- <tbody> y </tbody> corresponden a la apertura y cierre del cuerpo de la tabla. Marcamos las filas con <tr> y </tr>.

La **Figura 2.34** es el resultado del código anterior.

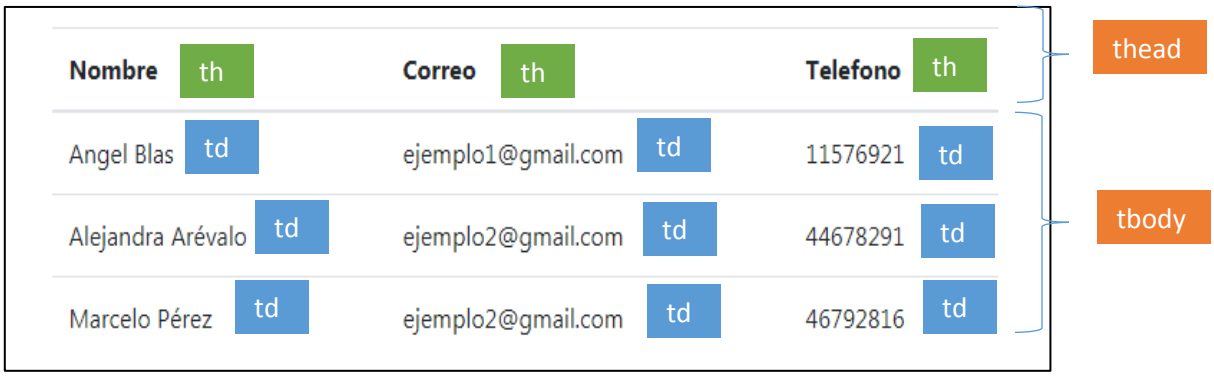


Figura 2.34 Visualización de la tabla en el navegador.

Lo que hicimos fue insertar una tabla básica, para darle algunos estilos debemos ingresar alguna o algunas de las clases que se presentan a continuación:

Clase	Descripción
. table-striped	Agrega el estilo "cebra" donde un registro en el cuerpo de la tabla tiene un color de fondo gris claro y el siguiente registro un fondo blanco. Este estilo se repite en todos los registros de la tabla
.table-bordered	Agrega todos los bordes de la tabla
.table-hover	Agrega el efecto *Hover* a los registros de la tabla cuando pasa el cursor por encima de ellos
.table-dark	Agrega como estilo un color de fondo oscuro a toda la tabla
.table-borderless	Elimina todos los bordes de la tabla
.thead-dark	Coloca un color oscuro de fondo al encabezado de la tabla y su texto color blanco
.thead-light	Coloca un color gris claro de fondo al encabezado de la tabla y su texto color negro

Tabla 2.11 Clases para dar formato a los elementos de una tabla.

Estas clases deben de colocarse dentro de la etiqueta de apertura <table> y después de la clase *table:*

```
<table class="table ">
    <thead>
      <tr>
```

Aquí se colocan

Por ejemplo:

```
<table class="table table-hover">
    <thead>
      <tr>
```

Para los estilos a los encabezados (*thead-dark* y *thead-light*), las clases deben de colocarse de la misma manera que la anterior, solo que ahora dentro de la etiqueta de apertura <thead>.

```
<table class="table">
    <thead class="thead-dark">
      <tr>
```

La siguiente figura, muestra el resultado de utilizar clases:

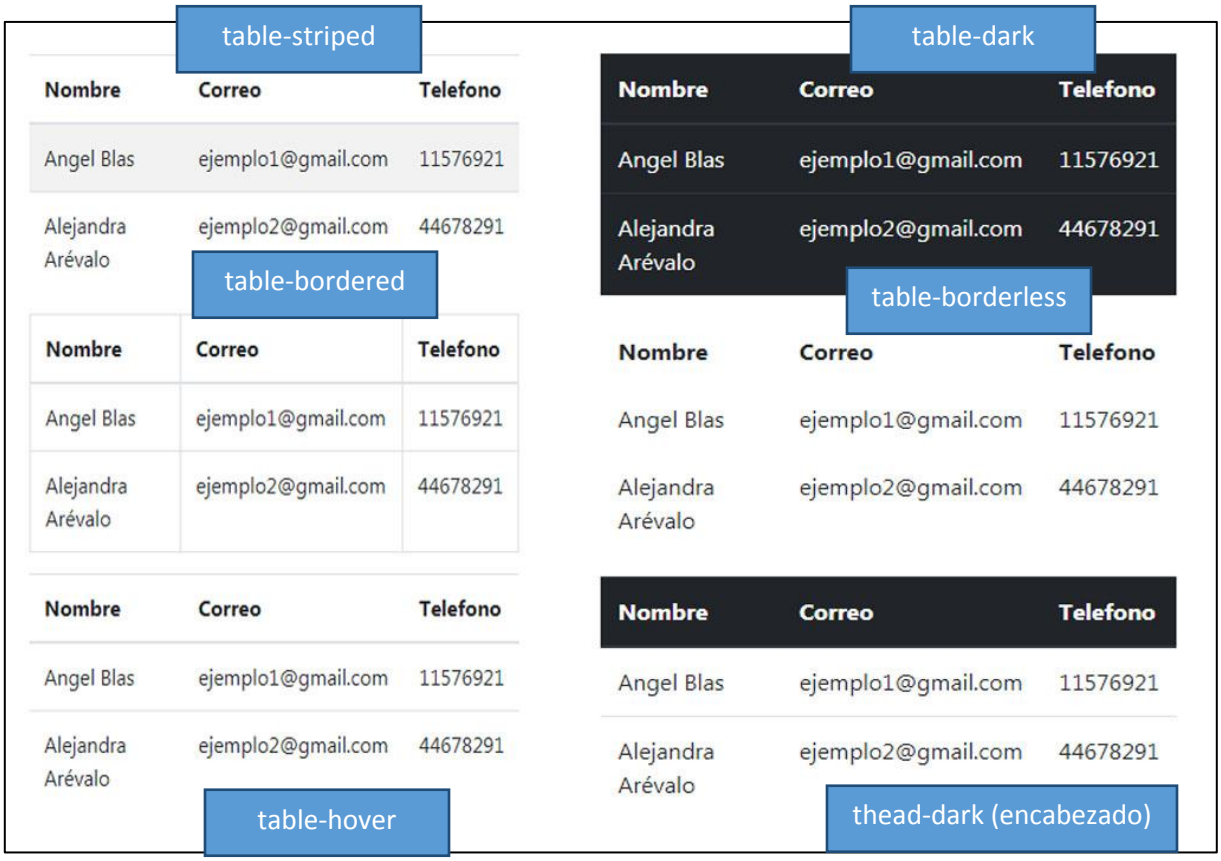


Figura 2.35 Estilos en las tablas utilizando las clases: *table-striped, table-dark, table-bordered, table-borderless, table-hover y thead-dark.*

2.3.5.1 Tablas responsivas

Las tablas que agregamos en el ejemplo anterior solo se adaptaban a un mínimo de tamaño de pantalla, a tamaños más pequeños, dejan de verse y pierden el estilo responsivo. A pesar de usar Bootstrap, las tablas no son responsivas por defecto, para ello necesitamos hacer uso de la clase *table-responsive.*

Para hacer uso de *table-responsive* clase debemos colocarla dentro de una etiqueta , que a su vez contendrá la tabla.

```
<div class="table-responsive">
  <table class="table">
    ...
  </table>
</div>
```

Nosotros podemos establecer los puntos de corte para el tamaño máximo y mínimo para que así se adapte a los diferentes dispositivos utilizando algo muy similar a las clases de cuadricula. La siguiente tabla muestra los 4 tipos de clases, para las diferentes resoluciones:

Clase	Resolución de pantalla
table-responsive-sm	< 576px
table-responsive-md	< 768px
table-responsive-lg	< 992px
table-responsive-xl	< 1200px

Tabla 2.12 Clases para los diferentes tamaños de tabla.

La siguiente figura, muestra un ejemplo usando la clase *table-responsive-sm* en donde se muestra una barra de desplazamiento para la tabla cuando el tamaño de la ventana es inferior a 576 px.

Figura 2.36 Uso de la clase *table-responsive-sm*. En la visualización se observa una barra de desplazamiento horizontal.

2.3.6 Manejo de imágenes

Bootstrap contiene una serie de clases dedicadas al estilo, alineación y tamaño de las imágenes.

Clase	Descripción
rounded	Redondea los bordes de una imagen
rounded-circle	Hace que la imagen tome una forma circular
float-left	Flota una imagen a la izquierda
float-right	Flota una imagen a la derecha
mx-auto .d-block	Centra la imagen, sin importar el tamaño de la pantalla
img-fluid	Hace que la imagen se ajuste automáticamente al tamaño de la pantalla

Tabla 2.13 Clases para el manejo de imágenes.

A continuación, se muestra el uso de estas clases que funcionan dentro de la etiqueta <img>

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
        <img src="img/fondo.jpg" class="float-left">
        <img src="img/fondo.jpg" class="float-right">
    </div>
  </div>
  <div class="row">
    <div class="col-md-6">
        <img src="img/fondo.jpg" class="rounded mx-auto d-block">
    </div>
    <div class="col-md-6">
        <img src="img/fondo.jpg" class="rounded-circle mx-auto d-block">
    </div>
  </div>
  <div class="row">
    <div class="col-md-12">
        <img src="img/fondo.jpg" class="mx-auto d-block">
    </div>
  </div>
  <div class="row">
    <div class="col-md-12">
        <img src="img/fondo.jpg" class="img-fluid mx-auto d-block">
    </div>
  </div>
</div>
```

Figura 2.37 Uso de las clases para las imágenes. La visualización en navegador se muestra en la **Figura 2.38**.

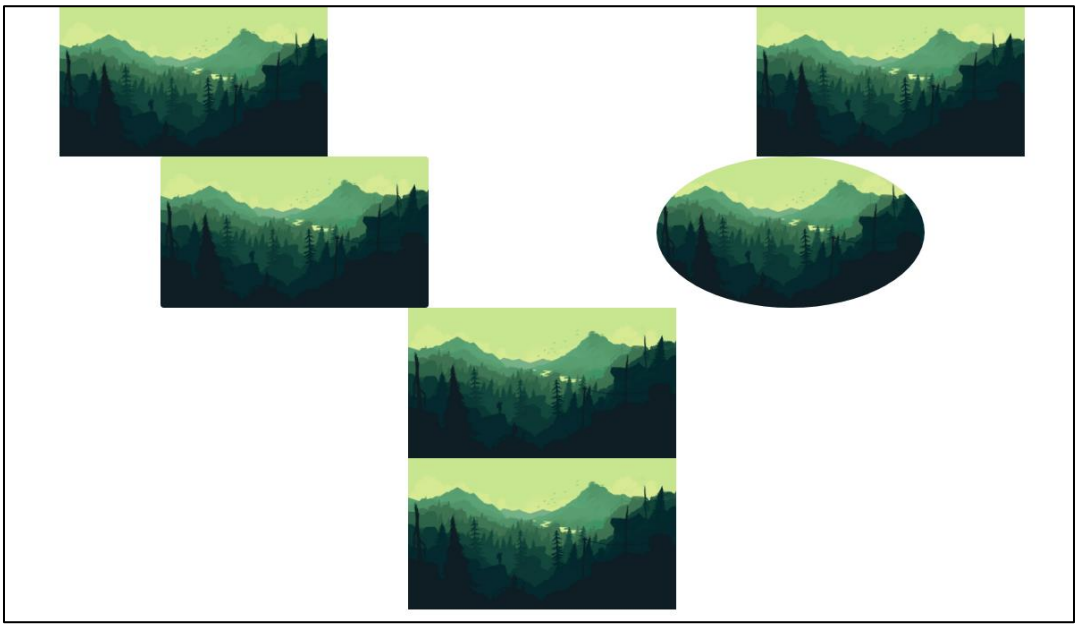


Figura 2.38 Visualización en navegador de las imágenes con las clases de Bootstrap.

La última imagen (**Figura 2.38**) es la única que contiene la clase *img-fluid,* de tal manera que se va a ajustar al tamaño de la pantalla en la que se visualice, mientras que las demás se perderán y no se verán completas.

2.3.7 Botones

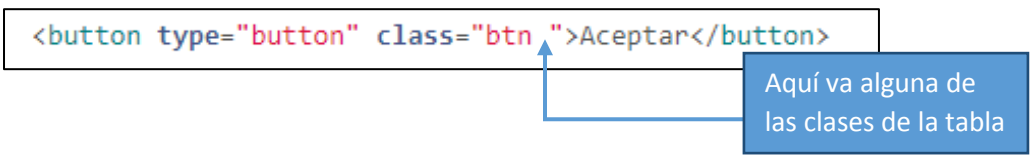
La mayoría de sitios web contienen páginas con uno o varios botones que realizan determinada acción, por ejemplo, hacer una consulta si se encuentra dentro de un formulario, ir a otros sitios externos, ordenar elementos o ejecutar ciertos eventos, entre muchos usos más. Bootstrap contiene botones que estéticamente son muy vistosos y que tienen el efecto *hover* que hace que cambien de color cuando el puntero pasa por encima y cuando se le da *clic*.

Para poder insertar un botón utilizando clases, debemos añadir la clase *btn* dentro de la etiqueta <button>. Esto creara un botón predeterminado, si queremos darle algún color con los que cuenta Bootstrap, debemos agregar alguna de las siguientes clases después de la clase *btn*:

Clase	Descripción
btn-primary	Genera un botón color azul y texto blanco
btn-secondary	Genera un botón color gris oscuro y texto blanco
btn-success	Genera un botón color verde y texto blanco
btn-info	Genera un botón color turquesa y texto blanco
btn-warning	Genera un botón color amarillo y texto negro
btn-danger	Genera un botón color rojo y texto blanco
btn-dark	Genera un botón color negro y texto blanco
btn-light	Genera un botón color gris claro y texto negro
btn-link	Cambia el botón por un enlace

Tabla 2.14 Clases para agregar botones.

La sintaxis en HTML seria la siguiente:



```
<div class="container">
  <h2 class="text-center">Botones en Bootstrap</h2>
  <button type="button" class="btn">Aceptar</button>
  <button type="button" class="btn btn-primary">btn-primary</button>
  <button type="button" class="btn btn-secondary">btn-secondary</button>
  <button type="button" class="btn btn-success">btn-success</button>
  <button type="button" class="btn btn-info">btn-info</button>
  <button type="button" class="btn btn-warning">btn-warning</button>
  <button type="button" class="btn btn-danger">btn-danger</button>
  <button type="button" class="btn btn-dark">btn-dark</button>
  <button type="button" class="btn btn-light">btn-light</button>
  <button type="button" class="btn btn-link">btn-link</button>
</div>
```

Figura 2.39 Ejemplos de uso de las clases mostradas en la **Tabla 2.14**.

2.3.8 Formularios

Uno de los elementos importantes de un sitio web son los formularios, que tienen diversos usos dentro de una página web dinámica, ya sea para iniciar sesión, solicitar datos personales o para realizar búsquedas de algún elemento, entre otros. Bootstrap maneja dos formas de organizar un formulario; puede ser con lelementos apilados o en línea, para ambas necesitamos hacer uso de una etiqueta con la clase *form-group* para cada elemento del formulario que agreguemos. Dicha clase nos proporciona un margen adecuado para separar los elementos. También podemos tener un cierto control con la entrada de datos mediante el atributo *type,* así aseguramos que los datos introducidos sean válidos (por ejemplo, números y correos), definir entradas tipo *password* para ocultar la contraseña y no dejarla visible.

La siguiente figura presenta un formulario de inicio de sesión utilizando Bootstrap:

```
<div class="container">
  <div class="row">
    <div class="col-md-6 offset-md-3">
      <h2 class="center">Inicio de sesión</h2>
      <form action="/login.php">                                    1
        <div class="form-group">                                    2
          <label for="email">Correo:</label>
          <input type="email" class="form-control" id="email">
        </div>
        <div class="form-group">                                    3
          <label for="pwd">Contraseña:</label>
          <input type="password" class="form-control" id="pwd">
        </div>
        <div class="form-group form-check">                         4
          <label class="form-check-label">
            <input class="form-check-input" type="checkbox"> Recordarme
          </label>
        </div>
        <button type="submit" class="btn btn-primary">Iniciar</button>
      </form>
    </div>
  </div>
</div>
```

Figura 2.40 Formulario de inicio de sesión utilizando clases de Bootstrap.

De acuerdo al número de bloque (1-4) de la figura anterior, tenemos que:

1. Se inicia el formulario con la etiqueta <form>. El atributo *action* corresponde al archivo o ruta que recibirá los datos obtenidos en el formulario.
2. Se añade un con la clase *form-group* para agregar el margen a los elementos que se encuentren dentro. Se agrega una entrada de tipo *email* para asegurar que se reciban correos válidos.
3. Se coloca de nuevo un con la clase *form-group* para agregar el margen a los elementos que se encuentren dentro.
   Se agrega una entrada de tipo *password* para ocultar la contraseña.
4. Se crea un con la clase *form-group* y *form-check* para agregar el margen a los elementos que se encuentren dentro y poder ingresar un *checkbox* con la clase *form-check-input*.

La visualización en el navegador es la siguiente (**Figura 2.41**):

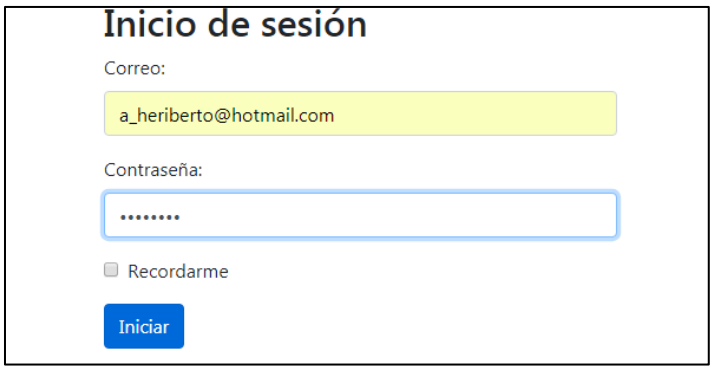


Figura 2.41 Formulario de inicio de sesión visto en el navegador.

Se pueden alinear todos los elementos horizontalmente, para ello utilizamos la clase *form-inline* y el atributo *role* con el valor *form*, este tipo de alineación suele servir cuando se genera un buscador por ejemplo:

```
<div class="container">
  <h2>Ingresa los datos de búsqueda:</h2>
  <form class="form-inline" role="form">
    <label for="cuenta" class="mb-2 mr-sm-2">Número de cuenta:</label>
    <input type="text" class="form-control mb-2 mr-sm-2" id="cuenta" placeholder="Número de cuenta">
    <button type="button" class="btn btn-primary mb-2">Buscar</button>
  </form>
</div>
```

Figura 2.41 Formulario horizontal utilizando la clase *form-inline*.

De la figura anterior tenemos que:

- La clase *mb-* nos ayuda a dar un margen inferior al elemento
- La clase *mr-sm* nos permite a dar un margen a la derecha al elemento

2.3.9 Entrada de datos

En los formularios vistos anteriormente solo eran para entrada de datos utilizando cajas de tipo texto y un botón de tipo *submit*, pero existen más elementos que son recomendables saber y utilizarlos cuando se requieran. La siguiente tabla enlista dichos controles:

Elemento	Descripción
Área de texto (textarea)	Es similar a la caja de texto, solo que este es más grande y acepta un mayor número de caracteres
Casillas de verificación (checklist)	Sirven para presentar una o más opciones donde el usuario puede elegir entre una o varias opciones
Botones de radio (radio button)	Sirven para presentar una o más opciones donde el usuario solo puede elegir una opción
Lista de selección	Muestra una lista desplegable con elementos

Tabla 2.15 Elementos para la entrada de datos.

A continuación, veremos la manera de estructurarlos, así como ejemplos de uso:

Área de texto

```
<div class="form-group">
  <label for="comentario">Escribe tu comentarios:</label>
  <textarea class="form-control" rows="5" id="comentario"></textarea>
</div>
```

Figura 2.42 Ejemplo de uso de un <textarea>.

Casilla de verificación

```
<h5>Elige el color que te gusta:</h5>
<div class="form-check">
  <label class="form-check-label">
    <input type="checkbox" class="form-check-input" value="1">Rojo
  </label>
</div>
```

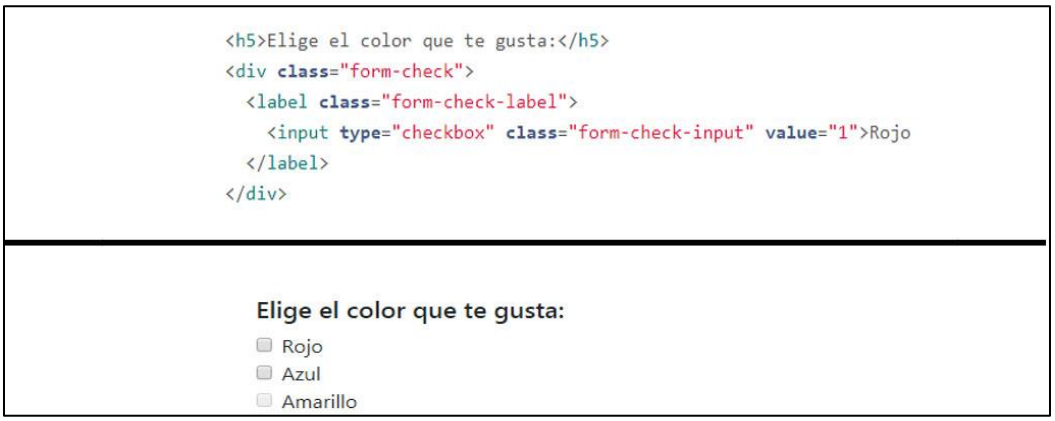


Figura 2.43 Ejemplo de uso de un *checkbox*.

Botones de radio

```
<h5>Sexo:</h5>
<div class="form-check">
  <label class="form-check-label">
    <input type="radio" class="form-check-input" name="radio">Hombre
  </label>
</div>
<div class="form-check">
  <label class="form-check-label">
    <input type="radio" class="form-check-input" name="radio">Mujer
  </label>
</div>
```

Figura 2.44 Ejemplo de uso de un botón de radio.

Lista de selección

Figura 2.44 Ejemplo de uso de una lista de selección.

2.3.10 Ventana modal

El último elemento que veremos son las ventanas modales, en estos tiempos cada vez es más utilizada ya que le da un estilo moderno a la navegación entre páginas web. Una ventana modal es un elemento que aparece por encima de las demás cuando se selecciona un elemento, puede ser un botón, un enlace o una imagen. Esto nos sirve para mostrar información al usuario del elemento que seleccione.

- Para que funcione debemos cargar la librería jQuery, ya que se necesitan componentes de JavaScript. Hasta ahora no hemos visto como descargar e importar jQuery a nuestro proyecto, pero podemos usar los CDN´s de Google y cargarlo como un recurso utilizando Assets desde la siguiente página:

  https://developers.google.com/speed/libraries/#jquery

Una vez importado jQuery, veremos un ejemplo de una ventana modal:

- Creamos dos controles que desplegaran la ventana modal cuando se haga clic sobre ellos: uno es un botón y el otro es una imagen; aunque puede ser también un texto o un enlace, etc. (**Figura 2.45**).

```
<!-- controles para abrir la ventana modal -->
<div class="row">
  <div class="col-md-6">
    <button type="button" class="btn btn-primary float-right" data-toggle="modal" data-target="#miModal">
      Abrir ventana
    </button>
  </div>
  <div class="col-md-6">
    <img src="img/bmw.png" class="img-thumbnail" data-toggle="modal" data-target="#miModal">
  </div>
</div>
```

Figura 2.45 Creación de los controles que mostraran la ventana modal.

- Las etiquetas del botón y la imagen, contienen el atributo *data-toggle* con el valor *modal* y *data-target* con el valor del id de la modal que definiremos más abajo, en este caso es *#miModal*, pero puede ser el que quieras.
- El padre que le da estructura la modal debe ir con la clase *modal* y con el id que se definió anteriormente (*#mymodal*). Dentro del , se crea un hijo con la clase *modal-dialog,* dentro agregamos otro con la clase *modal-content,* que se utiliza para darle encabezado, cuerpo y pie a la modal.

```
<!-- contenido de la ventana modal -->
<div class="modal" id="miModal">
  <div class="modal-dialog">
    <div class="modal-content">

      <!-- encabezado de la ventana -->
      <div class="modal-header">
        <h4 class="modal-title">Auto BMW</h4>
        <button type="button" class="close" data-dismiss="modal">&times;</button>
      </div>

      <!-- cuerpo de la ventana -->
      <div class="modal-body">
        <img src="img/bmw.png" class="mx-auto d-block">
        <p>Vestibulum a justo sed mauris luctus ultrices. Nulla cursus mattis turpis at scelerisque.
        Nam elit dui, hendrerit vitae consectetur in, cursus sed tellus. Duis ipsum erat,
        consectetur in enim non, bibendum tristique erat. Mauris ac massa velit.
        Nulla malesuada tortor nec mauris pharetra, ac convallis urna efficitur.
        Pellentesque commodo varius elit sed fermentum.</p>
      </div>

      <!-- pie de la ventana -->
      <div class="modal-footer">
        <button type="button" class="btn btn-danger mx-auto d-block" data-dismiss="modal">Cerrar</button>
      </div>

    </div>
  </div>
</div>
```

Figura 2.46 Código que genera el encabezado, cuerpo y pie de la ventana.

El resultado en el navegador del código anterior es el siguiente:

- Podemos cerrar esta ventana presionando el botón “cerrar” o con la tecla *esc*.

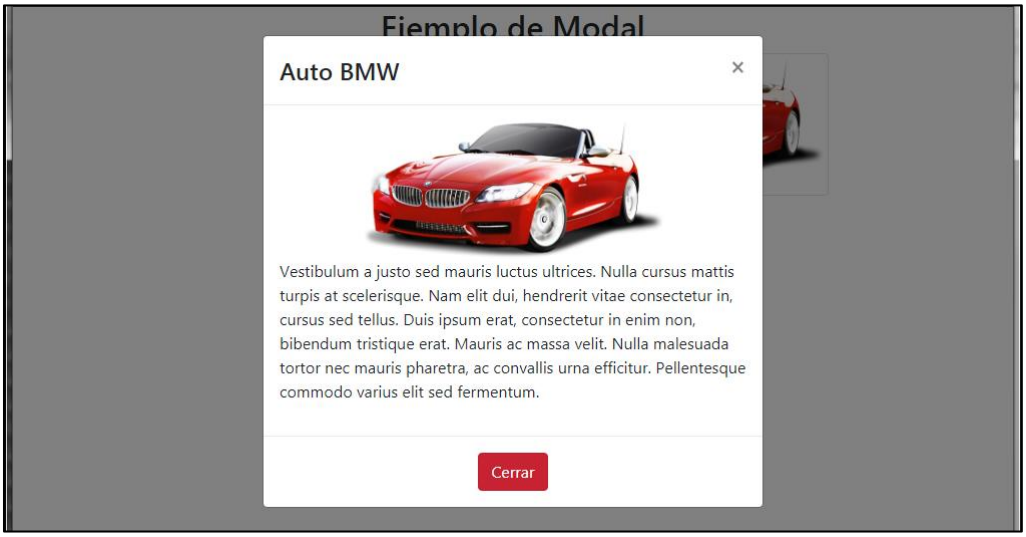


Figura 2.46 Al dar *clic* sobre el botón o el enlace se muestra la ventana modal.

2.4 jQuery

jQuery es una librería o *framework* JavaScript para crear páginas web dinámicas, con igual o más efectos que se pueden conseguir con CSS. No solo podemos obtener animaciones sino también podemos realizar consultas a la base de datos y mostrarlas al cliente sin recargar la página o direccionarla a otra, esto es posible haciendo uso de AJAX (que definiremos más adelante).

Las ventajas del uso de esta librería, son las siguiente:

- Podemos manipular elementos HTML y DOM (Modelo de Objetos del Documento). De este modo se puede hacer interacción con los elementos que estructuran el documento y también obtener sus valores.
- Podemos añadir o quitar estilos CSS o elementos HTML.
- Se pueden crear efectos y animaciones.
- Cuenta con AJAX.
- Contiene mucha comunidad y es muy popular.
- Otros *frameworks* trabajan en conjunto, por ejemplo, Bootstrap.
- Compatibilidad con la mayoría de navegadores.

En los temas anteriores creamos elementos HTML y agregamos estilos mediante CSS y Boostrap, todos estos elementos y estilos se cargaban cuando introducíamos la URL de la página o cuando se recarga la página. Pero ¿Qué pasa si nosotros queremos interactuar con el usuario?

Imaginemos que mediante Bootstrap agregamos un botón y una imagen, cuando dicho botón sea presionado por el usuario la imagen deba ocultarse, esto lo podemos hacer utilizando jQuery, ¿Recuerdas que para utilizar una ventana modal necesitábamos cargar esta librería a nuestro proyecto? lo que pasaba es que jQuery estaba a la espera de que se presionara el botón con el id de la ventana para desplegarla y lo mismo se repite cuando se presionaba el botón cerrar.

En este tema aprenderemos a manipular el DOM y los elementos HTML mediante los métodos ya existentes en jQuery, también veremos que es AJAX y como utilizarlo.

Por ser una librería JavaScript es recomendable saber programar lo básico en este lenguaje, de lo contrario veras que no es difícil, ya que la programación es algo parecida a PHP (o cualquier otro lenguaje) y un poco de CSS.

2.4.1 Como obtener jQuery

Al igual que Bootstrap, existen dos formas de importar jQuery en nuestro proyecto:

1. Descargando la librería desde su página oficial
2. Incluyéndolo desde un CDN

2.4.1.1 Descargando jQuery

Lo único que debemos hacer es ir a la página de Jquery: *http://jquery.com/* y dar clic sobre el botón que dice “Download jQuery” (**Figura 2.47**).

- En estos momentos se encuentra disponible la versión 3.3.1.

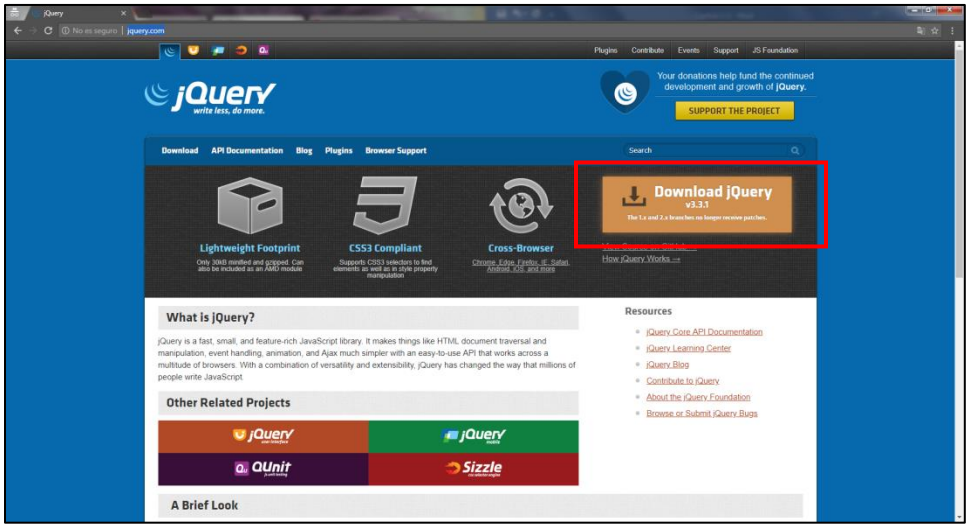


Figura 2.47 Botón para descargar la última versión de jQuery.

Después de dar *clic*, el botón nos redirigirá a una nueva página donde podemos escoger la forma de descarga: jQuery comprimido para producción o descomprimido para desarrollo. Podemos elegir cualquiera, aunque es recomendable la versión comprimida ya que pesa menos y la carga en tu proyecto es más rápida.

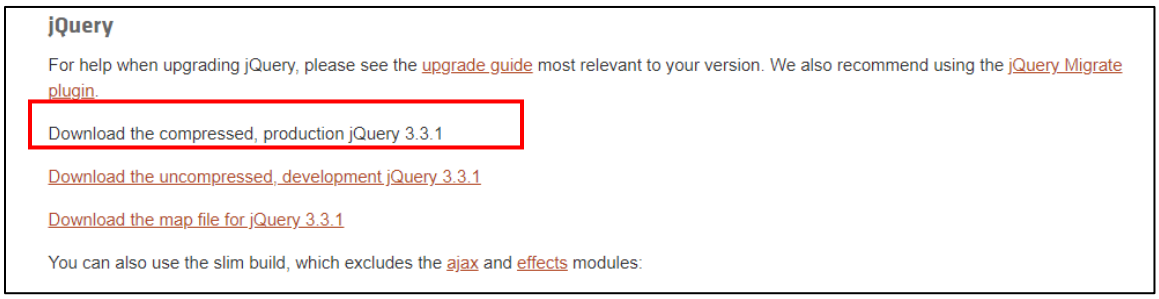


Figura 2.48 Selección de la versión comprimida de jQuery.

Cualquiera que elijas abrirá una ventana con el código del *framework*, lo que tienes que hacer es seleccionar todo, copiarlo y guardarlo con la extensión js utilizando cualquier editor de texto (**Figura 2.49**).

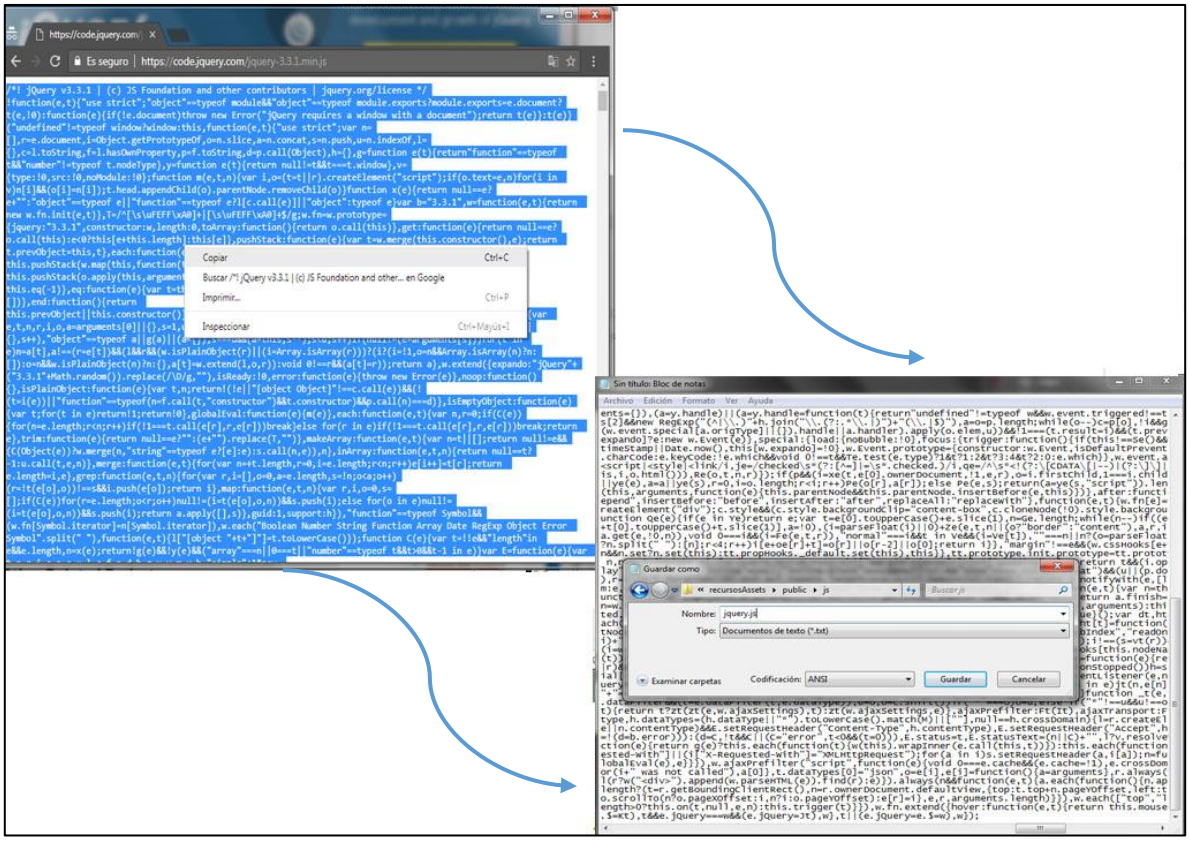


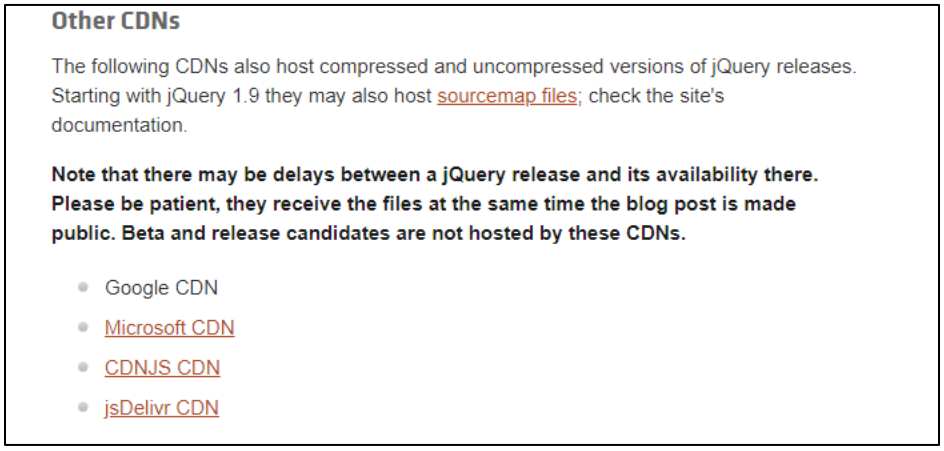
Figura 2.49 Guardamos el código con el nombre “jquery.js” utilizando bloc de notas.

- Recuerda que la ruta en **Phalcon** donde debe ir este archivo es en *public/js* y lo debes importar como un recurso utilizando Assets.

2.4.1.2 jQuery CDN

Para obtenerlo, al igual que los pasos anteriores necesitamos ir la página de jQuery, dar *clic* en descargar y después buscar la opción “Using jQuery with a CDN”.

Una vez dentro, navegamos un poco más abajo hasta donde dice “Other CDNs”, ahí encontraras una lista de servidores que contienen el *framework*.



Other CDNs

The following CDNs also host compressed and uncompressed versions of jQuery releases. Starting with jQuery 1.9 they may also host sourcemap files; check the site's documentation.

Note that there may be delays between a jQuery release and its availability there. Please be patient, they receive the files at the same time the blog post is made public. Beta and release candidates are not hosted by these CDNs.

- Google CDN
- Microsoft CDN
- CDNJS CDN
- jsDelivr CDN

Figura 2.50 Lista con todos los CDN disponibles.

Recomiendo seleccionar *Google CDN*, puedes insértalo como un recurso de Assets o colocarlo dentro de index.volt ubicado en *app/views/.*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
        <title>Phalcon PHP Framework</title>
        {{ assets.outputCss('cssBootstrap') }}

        <link rel="shortcut icon" type="image/x-icon" href="<?php echo $this->url->get('img/favicon.ico')?>"/>
    </head>
    <body>
        <div class="container">
            {{ content() }}
        </div>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
        {{ assets.outputJs('jsBootstrap') }}
    </body>
</html>
```

Figura 2.51 Importación del CDN de jQuery dentro de la etiqueta <script>.

2.4.2 DOM

El DOM (***Document Object Model*** o Modelo de Objetos del Documento en español) es un estándar que se utiliza para la manipulación de los elementos HTML de un sitio web. Como hemos visto en CSS podemos acceder a un elemento por medio de algún atributo, su clase, su id o por elementos padre acceder a sus hijos y viceversa. Con jQuery se accede de la misma manera.

Una forma de ser vista la estructura de un documento HTML, es precisamente como un árbol jerárquico: por ejemplo <html> es el padre de todos los elementos, <head> y <body> son

hermanos e hijos de <html>, de la misma manera todo lo que agreguemos en <body> serán hermanos entre si y lo mismo aplica para todos los elementos.

Un documento HTML sencillo en un esquema jerárquico sería como el siguiente:

```
<html>
  <head>
    <title>
      página de ejemplo
    </title>
  </head>
  <body>
    <p>Párrafo 1</p>
    <p>Párrafo 2</p>
  </body>
</html>
```

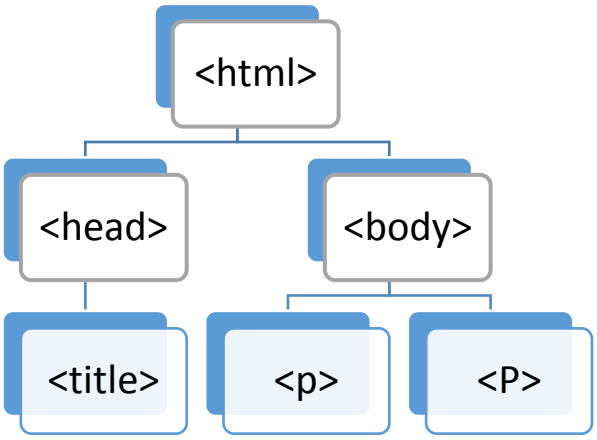


Figura 2.52 Maquetación de un documento HTML.

Los nodos más importantes son:

- Document: es el nodo principal del documento DOM
- Element: son todos los elementos HTML

2.4.3 Sintaxis

La sintaxis de jQuery se asemeja un poco a la de CSS, por tal motivo considere importante ver la parte de los selectores.

Recordemos como accedíamos a un elemento mediante CSS:

```
.clase{
  ....
}
```

Para una clase

```
p{
  ...
}
```

Para un elemento

```
#id{
  ...
}
```

Para un id

Cuando seleccionamos un elemento, declaramos las reglas de estilo dentro de llaves. En jQuery es algo similar solo que la selección en vez de ir acompañada de reglas de estilo, ahora va seguida por una acción y la selección de los elementos se hace con el símbolo "$".

```
$("elementoSeleccionado").accion(
  bloque de instrucciones
  ...
);
```

Figura 2.53 Sintaxis en jQuery.

De la figura anterior encontramos lo siguiente:

- El símbolo $ es realiza el llamando a jQuery
- elementoSeleccionado puede ser cualquier etiqueta HTML
- .accion es la acción o función que va a realizar el elemento

Por ejemplo para ocultar un elemento con jQuery se utiliza el método *hide(),* por ejemplo para ocultar todas las imágenes de nuestro sitio, se haría de la siguiente manera:

```
$("img").hide();
```

- Con la selección de <img> ocultamos todos los elementos que contengan esta etiqueta.

Si queremos ocultar solo los elementos que contengan la clase "miClase", seria de la siguiente forma:

```
$(".miClase").hide();
```

Ahora si deseamos ocultar el elemento con id = "miModal", quedaría así:

```
$("#miModal").hide();
```

Otro selector importante que hasta el momento no hemos visto es cuando se hace referencia al elemento seleccionado, por ejemplo, si tenemos un botón azul y al darle *clic* queremos que cambie de color, podríamos hacerle referencia por un id o una clase o bien por la palabra reservada *this:*

```
$(this).hide();
```

¿Verdad que la selección es similar a CSS?

Ahora, todo el código que hagamos debe ir en un archivo con la extensión js dentro de *public/js* y con la siguiente estructura:

```
$(document).ready(function(){
  $("img").hide();
  $(".miClase").hide();
  $("#miModal").hide();
  $(this).hide();
  ....
  ....
});
```

Figura 2.54 Estructura de un documento JavaScript.

Entendamos el código de la figura anterior:

- $ llama a jQuery y selecciona al elemento *document*
- La acción es *ready*, que significaría algo como "*cuando el documento esté listo, ejecuta la función*". Es importante colocar el método así se ejecuta el código cuando la página haya terminado de cargarse y no antes pudiendo generar un funcionamiento inesperado.
- Dentro de las llaves van las funciones que necesitemos.

2.4.4 Otro tipo de selectores

Aparte de los selectores anteriores existen muchísimos, aunque son poco frecuentes, podrían servir de ayuda el algún momento. La siguiente tabla contiene selectores de gran utilidad:

Selector	Descripción	Ejemplo
*	Selecciona todos los elementos	$("*")
.clase1, .clase2	Selecciona más de una clase	$("triangulo","circulo")
elemento1, elemento2	Selecciona más de un elemento	$("p","h6")
:first	Si existen n número de elementos con la misma etiqueta, selecciona la primera que encuentre	$("div:first")

:last	Si existen n número de elementos con la misma etiqueta, selecciona el ultimo que encuentre	$(“:last”)
:first-child	Selecciona el primer elemento que es hijo de su padre	$("div p:first-child")
:last-child	Selecciona el último elemento que es hijo de su padre	$("div p:last-child")
:nth-child(n)	Selecciona el elemento hijo con la posición introducida	$("div:nth-child(5)")
:nth-last-child(n)	Selecciona el elemento hijo con la posición introducida, pero contando desde el ultimo	$("div:nth-child(2)")
:contains(text)	Selecciona todos los elementos que contengan el texto introducido	$(":contains('Angel')")
:parent	Selecciona al elemento que es padre del elemento seleccionado	$(":parent")
[attribute]	Selecciona todos los elementos con el atributo href	$("[href]")
[attribute$=value]	Selecciona todos los elementos con el atributo href y que su valor termine en el introducido	$("[href$='.jpg']")

Tabla 2.16 Tipo de selectores en jQuery.

Ejemplos de uso de algunos de estos métodos serán utilizados en el capítulo 3, cuando se tenga que seleccionar elementos hijos.

2.4.5 Eventos

Los eventos son instrucciones que están a la espera de ejecutarse cuando un usuario realiza determinada acción. Algunos ejemplos, es cuando pasas el *mouse* sobre un elemento, cuando se da *clic* en una imagen, cuando quitas el puntero del elemento seleccionado, etc.

Pueden ser programados para que se ejecuten por si mismas después de cierto tiempo, por ejemplo, en los videos de *YouTube*, algunos contienen un *script* para que muestre publicidad cada cierto tiempo durante la reproducción del video (**Figura 2.55**).

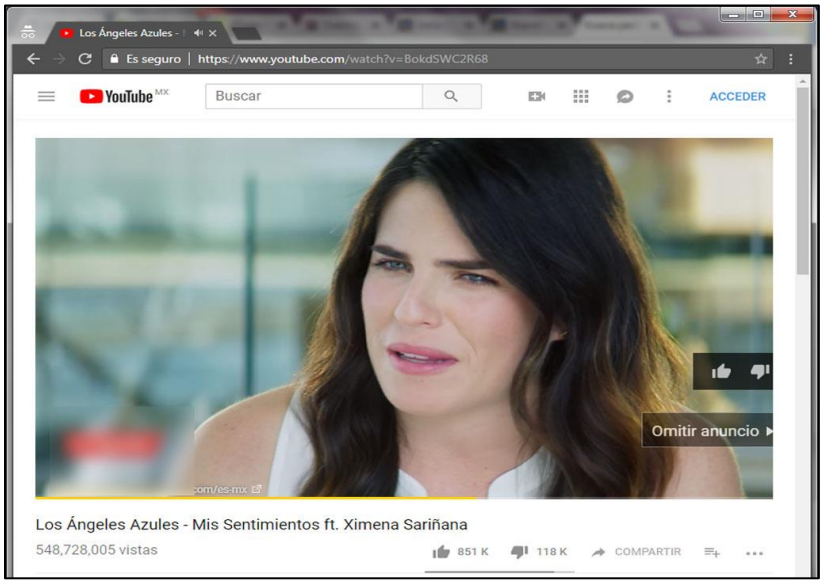


Figura 2.55 Script que se ejecuta cada cierto tiempo mostrando publicidad.

La siguiente tabla muestra los eventos más comunes en uso que se ejecutan cuando el mouse interactúa con los elementos HTML:

Evento	Descripción
ready()	Esta función se ejecuta cuando la página cargue por completo. $(document).ready()
click()	Esta función se ejecuta cuando se da *clic* sobre el elemento HTML.
dblclick ()	Esta función se ejecuta cuando se da doble *clic* sobre el elemento HTML.

mouseenter()	Esta función se ejecuta cuando pasa el *mouse* sobre el elemento HTML.
mouseleave()	Lo contrario a la función anterior, se va a ejecutar cuando el mouse se quita del elemento
hover	Esta función debe ser programada con dos funciones: una se ejecuta cuando el mouse pasa por encima del documento, y la otra se ejecuta cuando el mouse sale del elemento
focus()	Es utilizada para el llenado de campos en un formulario y se ejecuta cuando se selecciona un campo de texto

Tabla 2.16 Eventos en jQuery.

Ahora vamos un ejemplo de la sintaxis para implementar eventos:

```
$(document).ready(function(){
  $("p").click(function(){
    $(this).hide();
  });
});
```

Figura 2.56 Ejemplo de uso de evento *clic()*.

En la figura anterior utilizamos el evento *clic()* sobre el elemento <p>, es decir, cuando en el navegador demos *clic* sobre cualquier etiqueta <p> se oculten por medio de la función *hide()*.

Veamos un ejemplo que implique más funciones y eventos. Para esto usaremos dos imágenes iguales, pero una ha sido retocada.

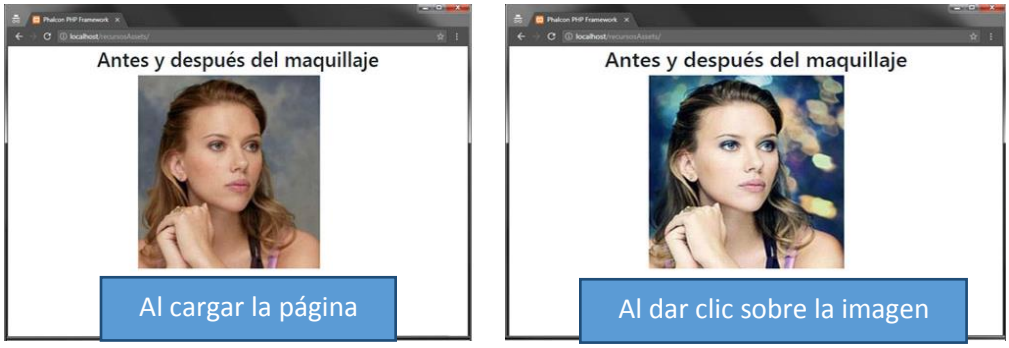


Figura 2.56 Ejemplo de uso de eventos para cambiar de una imagen a otra dando *clic*.

En el HTML utilizamos Bootstrap:

```
<div class="container">
  <div class="row">
    <div class="col-md-12">
      <h1 class="text-center">
        Antes y después del maquillaje
      </h1>
    </div>
    <div class="row justify-content-center">
      <div class="col-md-6 col-lg-1 offset-md-3 offset-lg-11" id="contenedor">
        <img src="img/antes.jpg" id="antes">
      </div>
    </div>
</div>
```

Figura 2.57 HTML que compone la vista de la **Figura 2.56**.

Analizando el código de la figura anterior tenemos que:

- Tenemos un con el id = “contenedor” que nos va a servir para mostrar y agregar las imágenes.
- De inicio tenemos cargado en el DOM una imagen llamada “antes” y con id = “antes”.

El código en el archivo js utilizando jQuery es el siguiente:

```
$(document).ready(function(){
  $("#antes").click(function(){
    $(this).hide();
    $("#contenedor").append("<img src='img/despues.jpg' id='despues'>");
  });
    $("#contenedor").on('click', '#despues',function(){
    $(this).hide();
    $("#antes").show();
  });
});
```

Figura 2.58 Código en jQuery que interactúa con los elementos HTML de la **Figura 2.57**.

- Dentro de $(document).ready tenemos dos selectores con sus eventos.
- El primer selector se realiza sobre el elemento con el id = “antes” y le asignamos el evento *clic()*, es decir, que cuando se seleccione a la imagen *antes.jpg* se oculte por medio de la función *hide()*.
- Una vez que se ocultó la imagen seleccionamos el elemento con el id = “contenedor” y con la función *append()* le insertamos código HTML con la ruta de la imagen *después.jpg* y el id = “despues”

hasta aquí lo que hacemos es cambiar de la imagen “antes.jpg” a “después.jpg”, pero regresar de la segunda a la primera ya no se hace de la misma manera. Al usar la función *append()* agregamos HTML después de que cargara la página, por lo tanto el evento *clic()* no funciona, pues no es posible asociar eventos a elementos cargados dinámicamente.

Esto se soluciona utilizando el evento *on()* que sirve para asociar eventos a los elementos del DOM. El evento recibe ciertos parámetros:

```
$(selector).on(evento, selectorHijo, parametros, funcion, mapeo)
```

- Selector. Elemento HTML.
- Evento: Evento que se va a realizar cuando se seleccione el elemento
- SelectorHijo (opcional): Selección del nodo hijo dentro del elemento
- Parámetros (opcional): Parámetros que recibe la función
- Función(obligatorio): Función a ejecutar cuando se realice el evento
- Mapeo (opcional): Mapeos para cada evento

En nuestro ejemplo, el selector es el con id = “contenedor”, el evento es *click(),* el selector hijo es la etiqueta <img> con id =”despues” y al final tenemos la función.

- Se oculta la imagen con id = “despues” con *hide()*
- Se muestra de nuevo la imagen con id = “antes” con *show()*

Como vimos utilizar eventos no es nada difícil simplemente es conocer cuál es su función. En el ejemplo anterior utilizamos *clic(),* pero podría sustituirse por cualquier otro de la **Tabla 2.16** .

Si quisiéramos cambiar el evento por *dblclick()* solo tenemos que colocarlo donde sea necesario:

```
$(document).ready(function(){
  $("#antes").dblclick(function(){
    $(this).hide();
    $("#contenedor").append("<img src='img/despues.jpg' id='despues'>");
  });
    $("#contenedor").on('dblclick', '#despues',function(){
    $(this).hide();
    $("#antes").show();
  });
});
```

Figura 2.59 Se cambia el evento *click()* por *dbclick().*

2.4.6 AJAX

AJAX (Asynchronous Javascript and XML) es una tecnología que nos permite interactuar con el *back-end* para obtener datos y mostrarlos al usuario sin necesidad de actualizar la página o redireccionar a una nueva, de ahí recibe el nombre asíncrono. Esto permite un sitio web dinámico haciendo la comunicación entre cliente y servidor de manera ágil, mejorando la experiencia del usuario.

- Asíncrono se refiere a que las consultas que la página realiza al servidor se ejecutan en segundo plano, sin afectar la visualización del contenido.
- jQuery contiene esta tecnología, permitiendo realizar peticiones GET y POST de forma sencilla.
- Ejemplos de uso de AJAX son los buscadores de *Google* y *Youtube*, ambos tienen un buscador que despliegan sugerencias que coincidan con el texto que el usuario ingresa.

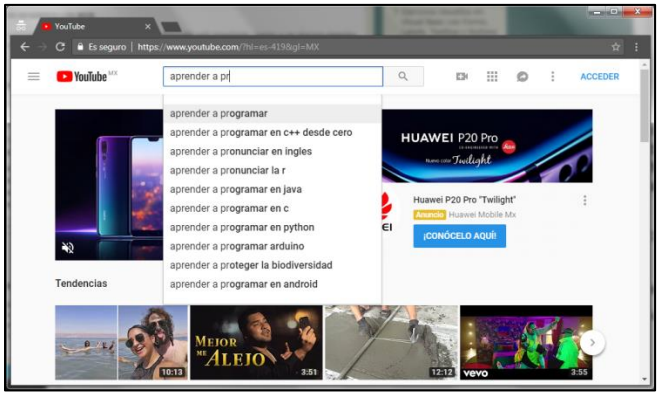


Figura 2.60 Buscador en tiempo real de *youtube*.

Vamos a desarrollar un ejemplo utilizando AJAX con jQuery y **Phalcon**: Se trata de un buscador que muestre en una tabla (hecha con Bootstrap) información de o de los alumnos que coincidan a lo que se está ingresando. Cuando se borre lo que está escrito en el buscador, la tabla mostrara todos los alumnos. El SGBD con el que crearemos la base de datos será MySQL.

Primero creamos la base de datos, en este caso se llama “curso_db” y agregamos la tabla “alumno” que contiene los siguientes campos:

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	id	int(11)			No	*Ninguna*		AUTO_INCREMENT	Cambiar Eliminar Más
☐	2	nombre	varchar(20)	utf8_spanish_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	matricula	varchar(13)	utf8_spanish_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	edad	int(11)			No	*Ninguna*			Cambiar Eliminar Más

Figura 2.61 Campos de la tabla “alumno”.

- Puedes agregar a la tabla los registros que quieras, en este caso solo agregaremos cinco:

	id	nombre	matricula	edad
Editar Copiar Borrar	2	Heriberto Blas	413117278	26
Editar Copiar Borrar	3	Alejandra Arévalo M	145338792	28
Editar Copiar Borrar	4	Teresa De Jesus	784229467	30
Editar Copiar Borrar	6	Marcelo Pérez	712487952	35
Editar Copiar Borrar	7	Lourdes Sánchez	876449276	30

Figura 2.62 Campos de la tabla “alumno”.

Ahora en **Phalcon** creamos un proyecto con el nombre que queramos y dentro debemos crear un modelo para la tabla “alumno” y un controlador que llamaremos “BuscarController”, para esto utilizamos las *herramientas de desarrollo* (vistas en el capítulo 1).

- Recuerda que antes de crear el modelo, primero debemos configurar el archivo *config.php* con las credenciales de acceso a la base de datos. El archivo se encuentra ubicado en *app/config*.

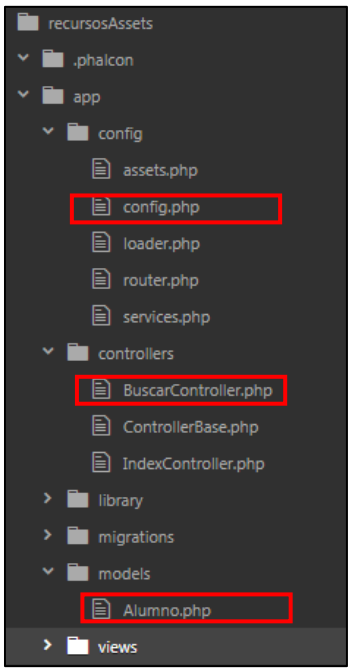


Figura 2.63 Se debe crear un controlador, configurar la conexión con la base de datos para después crear un modelo de “alumno”.

➢ Debes cargar Bootstrap y jQuery.

El HTML de la vista es el siguiente:

```
<div class="container">
  <div class="text-center">
    <h1>Buscador en tiempo real <small>con jQuery(AJAX) y Phalcon</small></h1>
  </div>
  <div class="row">
    <div class="col-md-3">
      <div class="form-group">
        <label for="search">Buscar alumno:</label>
        <input type="text" class="form-control" id="search" placeholder="Buscar">
      </div>
    </div>
    <div class="col-md-3 offset-md-3" id="result">
    </div>
  </div>
</div>
```

Figura 2.64 El código contiene un formulario y un para insertar la tabla con los alumnos buscados.

Analizando el código de la figura anterior, tenemos:

- Contiene una etiqueta <input> tipo “text” con el id = “search”.
- El con id = “result” está vacío porque aquí vamos a mostrar la tabla con el o los alumnos.

El archivo js contiene lo siguiente:

```
$(document).ready(function(){
  $('#search').focus();

  $('#search').on('keyup', function(){
    var search = $('#search').val();
    $.ajax({
      type: 'POST',
      url: 'http://localhost/recursosAssets/buscar',
      data: {'search': search},
    })
    .done(function(resultado){
      $('#result').html(resultado);
    })
    .fail(function(){
      alert('Hubo un error');
    })
  })
})
```

Figura 2.65 Contenido del archivo js.

- El método *focus()* hace que se coloque el cursor automáticamente en el buscador con el id = “search”
- El evento *keyup()* recibirá lo que se ingrese por teclado.

- Creamos la variable “search”, que le asignaremos lo que escriba el usuario por teclado. El valor lo obtenemos con el método *val()*.
- Se utiliza el método AJAX ($.ajax) donde ingresamos como parámetros:
    1. El tipo de petición
    2. La URL a donde realizamos la petición (en este caso es al controlador “BuscarController”)
    3. La variable que contiene los datos ingresados por teclado.
- *.done* es una función que se ejecuta cuando la petición se haya realizado correctamente. En este caso se busca el elemento con id = “result” y se coloca el código HTML que forma la tabla con los resultados obtenidos. El método html() pondrá en el DOM lo que retorne el controlador.
- *.fail* solo se ejecuta si la petición registró algún error, por ejemplo, que la ruta a donde se hace referencia no exista. En este caso la función solo mostrara una ventana emergente.

Por último, “BuscarController” contiene el siguiente código:

```
6       public function indexAction()
7       {
8         $search = $this->request->getPost("search");
9         $query = $this->modelsManager->createQuery("SELECT * FROM Alumno
10          WHERE nombre LIKE '%$search%' ");
11        $alumnos = $query->execute();
12
13        $html = "<table class='table table-striped table-responsive'>
14                  <thead class='thead-dark'>
15                    <tr>
16                      <th scope='col'>Nombre</th>
17                      <th scope='col'>Matrícula</th>
18                      <th scope='col'>Edad</th>
19                    </tr>
20                  </thead>
21                  <tbody>";
22        foreach ($alumnos as $alumno) {
23          $html .= "
24                    <tr>
25                      <td>$alumno->nombre</td>
26                      <td>$alumno->matricula</td>
27                      <td>$alumno->edad</td>
28                    </tr>
29              ";
30        }
31        $html.= "  </tbody>
32                </table>";
33        return $html;
34      }
```

Figura 2.66 Contenido del controlador “BuscarController”.

Si analizamos el código de la figura anterior, destacamos lo siguiente:

- En la línea 8 recibimos lo que fue enviado por POST al controlador y lo asignamos a la variable $search.
- En las líneas 9-11 creamos el *query* con la consulta y el resultado de la ejecución se asigna a la variable $alumnos.
- En las líneas 13-21 creamos la variable $html donde concatenaremos el resultado de la consulta con HTML y Bootstrap, para esto creamos una tabla y definimos los encabezados.
- En las líneas 22-30 por medio de un *foreach* concatenamos los datos del o los alumnos y se asignan a la variable $html.
- En las líneas 31 y 32 cerramos la tabla con sus etiquetas.
- En la línea 33 retornamos la variable $html que procesara jQuery con la función *.done*.

Teniendo todo listo, así se visualiza en el navegador:

Figura 2.67 La página carga la tabla con todos los alumnos.

Figura 2.68 Si introducimos datos, la tabla filtrará los resultados que contengan los caracteres ingresados.

2.4.7 Manipulación de elementos HTML

En el ejemplo anterior agregamos de manera dinámica una tabla con el método *html()*, pero existen más formas de agregar elementos con jQuery. La siguiente tabla muestra los métodos más frecuentes de uso, así como su descripción:

Método	Descripción
append ()	Agrega el contenido al final del elemento seleccionado
prepend ()	Agrega el contenido al principio del elemento seleccionado
after ()	Agrega el contenido después del elemento seleccionado
before ()	Agrega el contenido antes del elemento seleccionado

Tabla 2.16 Métodos para agregar código HTML de forma dinámica.

Ejemplos sencillos de la tabla anterior, son los siguiente:

La vista

```
<p id="uno">con</p>
<img src="img/irelia.png" id="irelia">
```

js

```
$(document).ready(function(){
  $("#uno").append(" <strong>jQuery</strong>.", "<small> (este es otro parámetro)");
  $("#uno").prepend(" Aprendiendo "," <small> (este es otro parámetro) ");

  $("#irelia").after("<img src='img/brand.png' class='img-fluid img-thumbnail'>",
      "<img src='img/MonkeyKing.png'>");
  $("#irelia").before("<img src='img/ahri.png' class='img-fluid img-thumbnail'>");
})
```

Figura 2.69 Uso de los métodos jQuery para agregar HTML.

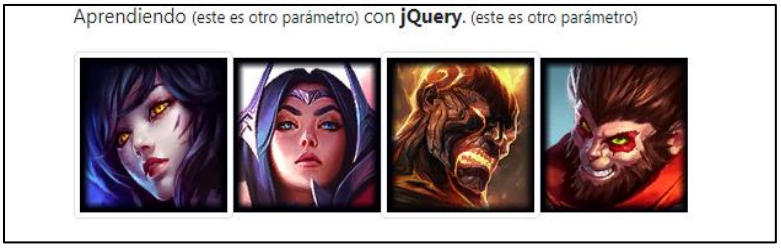


Figura 2.70 Visualización en el navegador.

- Como se puede ver, los métodos de la **Figura 2.69** recibieron dos parámetros, pero pueden recibir n parámetros.

2.4.8 Nodo padre, nodo hijo y nodo hermano

Dentro del DOM de un documento podemos también acceder a los nodos padre, hijos y hermanos de cualquier elemento HTML.

2.4.8.1 Nodo padre

El nodo padre es el elemento que se encuentra por encima del elemento seleccionado.

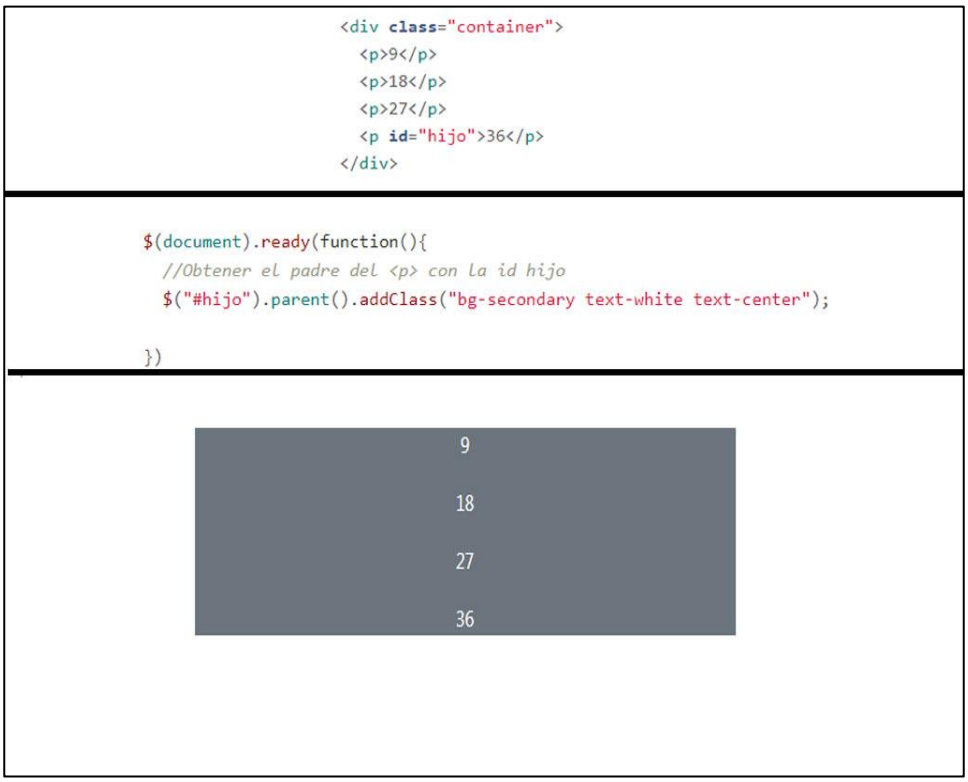


Figura 2.71 Ejemplo de selección de un nodo padre.

2.4.8.2 Nodo hijo

El nodo hijo es el elemento o elementos que se encuentran por debajo del elemento seleccionado.

Figura 2.72 Ejemplo de selección de un nodo hijo.

2.4.8.3 Nodo hermano

El nodo hermano es el elemento o elementos que se encuentran al mismo nivel del elemento seleccionado.

La figura 2.72 muestra un ejemplo de selección de un nodo hermano.

Figura 2.69 Ejemplo de selección de un nodo hermano.

2.4.9 Resumen

En este capítulo se estudió a grandes rasgos tecnologías que nos permiten dar a un sitio web un mejor diseño, como se importan y trabajan en conjunto con el *framework* **Phalcon**.

Una página web debe tener una buena presentación y que además el usuario pueda interactuar con los elementos que se visualizan, es por ello que se considera importante saber algo de hojas de estilo y el uso de estos *frameworks* que son utilizados por sus funciones y el ahorro de tiempo que nos brindan. Aunque no sean un lenguaje de programación es algo que todo programador debe conocer.

Para CSS, los temas abordados fueron:

- La forma de declarar reglas de estilo a uno o varios elementos HTML.
- Acceder a los elementos por su clase o id.
- Las propiedades básicas para dar formato al texto, así como las fuentes.
- Márgenes.
- Fondos de pantalla usando colores o imágenes.
- Posicionamiento de los elementos.
- Selección de elementos padre, hijo, hermanos, etc..

Para Bootstrap se mencionó que es un *framework* para dar diseño y animaciones a los elementos HTML, además de que permite hacer un sitio web responsivo. Los temas tratados en esta sección fueron los siguientes:

- Instalación
- Sistema de grillas para filas y columnas
- Estilos al texto y colores
- Uso de tablas
- Manejo de imágenes
- Botones y formularios
- Uso de ventanas modales

Por último y no menos importante, se dedicó una sección al uso básico de las funciones y métodos con los que cuenta jQuery, que es una librería JavaScript muy popular para realizar páginas web dinámicas. Los temas tratados en esta sección fueron los siguientes:

- Instalación
- Que es el DOM y como se manipulan sus nodos
- La sintaxis de jQuery
- Selección de elementos HTML
- Uso de eventos
- Trabajar de manera asíncrona con *el back-end* por medio de AJAX
- Insertar HTML dinámicamente

Cabe señalar que existen muchísimo más funciones para estos dos *frameworks* que podrán revisar en la documentación de su página oficial.

Capítulo 3. Participación en el desarrollo del sitio web para el Centro Tecnológico Aragón

3.1 Introducción

El centro Tecnológico Aragón es un área de investigación y desarrollo de la ciencia y la tecnología realizando proyectos para las instituciones, la industria o las diferentes instancias gubernamentales.

Además, impulsa a los alumnos a tener una mejor formación antes o después del término de sus estudios, preparándolos para el trabajo, ya que te permite realizar el servicio social a través del ejercicio teórico – práctico.

Cuenta con investigadores y profesores muy bien preparados, alumnos y diversas áreas de estudio y laboratorios como son: arquitectura, diseño industrial, ahorro de energía, incubadora InnovaUNAM, Invernadero, materiales, medición e instrumentación, mecánica aplicada a materiales, cómputo para la enseñanza, seguridad informática y computo, entre otras.

Como estudiante de ingeniería en computación realice el servicio social en el laboratorio de cómputo a cargo del M. en C**.** Marcelo Pérez Medel y del M. en C**.** Jesús Hernández cabrera. Dentro del periodo del servicio se nos planteó la propuesta de realizar un sitio web para la página del centro tecnológico que contara con las siguientes características:

- Diseño minimalista y adaptable a móviles
- Distinción de categorías mediante colores
- Con identidad institucional
- Diseño cohesivo (léase integral) en fuentes y estilos
- Actualización de contenidos mediante información de BD

El entorno de trabajo propuesto fue el siguiente:

- Apache Web Server 2.4+
- PHP 5.6+
- **Phalcon** Framework 3+
- Para el diseño web se propuso el *framework* Bootstrap en su versión 4.
- Se utilizó jQuery en su versión 3.2.1 para trabajar en conjunto a Bootstrap y dar un diseño dinámico a la página y también para utilizar la técnica AJAX.

- El control de versiones utilizado fue Bitbucket, que es una plataforma para el control de versiones y respaldos, permitiendo administrar proyectos usando *git* en la nube.
- Se asignaron actividades con *trello* para trabajar de forma coordinada.

El sitio web del centro tecnológico Aragón fue realizado con un menú principal con enlace a los siguientes módulos:

- Laboratorios y áreas
- Directorio
- Aula virtual
- Correo CTA
- Coordinación
- Eventos
- Acerca de (contiene historia de CTA, galería con un slider o carrusel de imágenes) y un FAQ (lista de preguntas frecuentes con su respuesta)

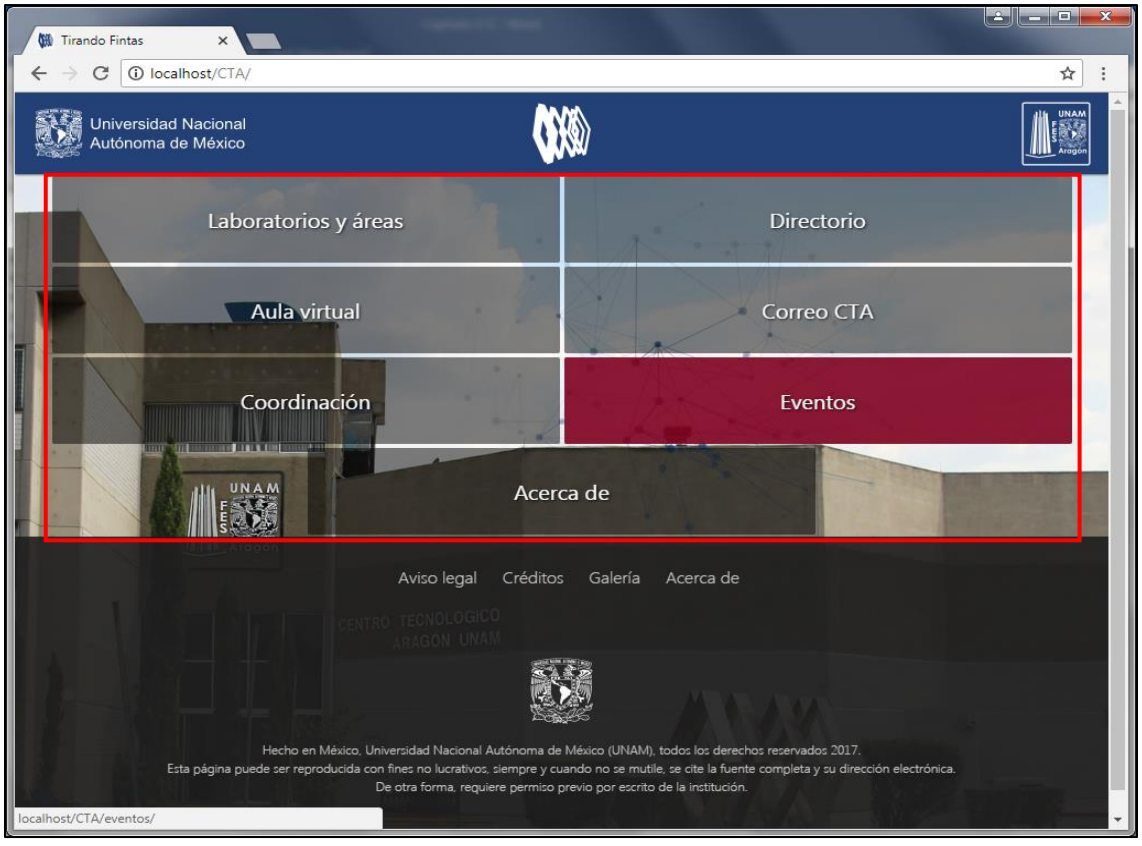


Figura 3.1 Página principal del Centro Tecnológico Aragon.

En este capítulo se mostrará la forma en como fue realizado la sección galería y los módulos evento y directorio utilizando las herramientas de los dos capítulos anteriores.

3.2 Sección galería

Para esta sección se encuentra un carrusel o *slider* de imágenes haciendo uso de un *plugin* en JavaScript, las imágenes se cargan dinámicamente y la ruta como descripción se definen en un archivo JSON y no de base de datos con la finalidad de que la página cargue más rápido. Todos estos conceptos los explicaremos más adelante:

Figura 3.2 *Slider* de imágenes de la sección “Galería”.

3.2.1 jQuery Simple Slider

los *plugins* son herramientas desarrolladas por terceros y que nos permiten resolver alguna necesidad en específico. La mayoría son *Open Source* para que los programadores puedan modificarlos y adaptarlos a las necesidades de su sitio.

El *slider* incluido en la galería es un *plugin* que está hecho utilizando HTML, CSS, JavaScript y jQuery. No es necesario entender a fondo las líneas de código que lo conforman, sino conocer su funcionamiento.

El nombre del autor* y ejemplos de uso lo puedes encontrar en la página: *https://codepen.io/CaptainSmollett/pen/epGWJa* . Inclusive la página contiene editores de texto para HTML, CSS y JavaScript para que puedas ir probando el código.

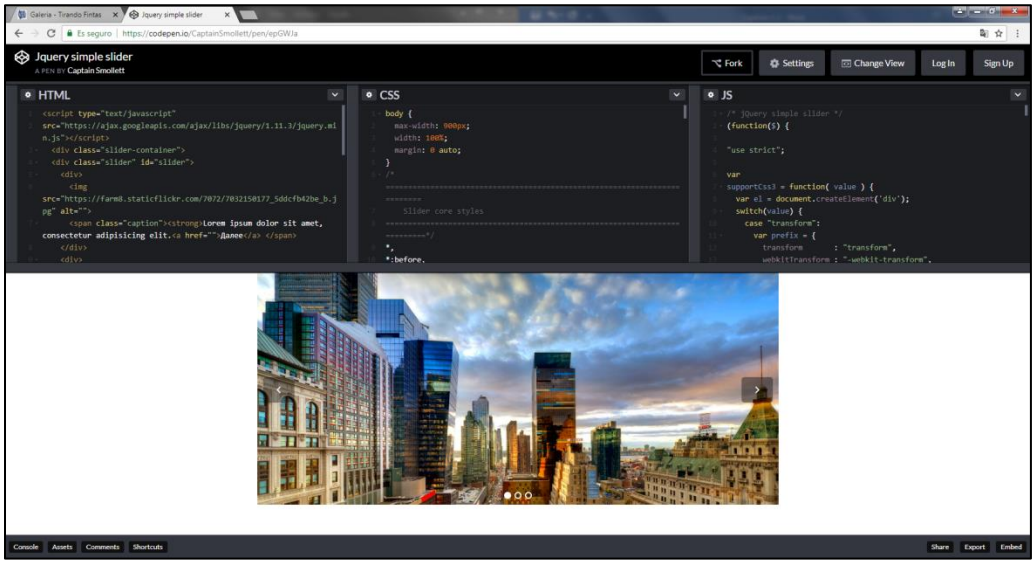


Figura 3.3 Código que compone el *plugin jQuery Simple Slider.*

3.2.2 Importando jQuery Simple Slider al sitio web del Centro Tecnológico Aragón

El código se copió de la página del *plugin* y fue puesto en archivos CSS y JS con su código correspondiente, posteriormente fueron colocados en **Phalcon** en *public/css* y *public/js*. Por último, se cargaron como recursos utilizando Assets.

```
// == RECURSOS UTILIZADOS EN TODOS LOS ARCHIVOS DEL SITIO ==
$col = $assets->collection('cssCore');
$col->addCss('/assets/css/bootstrap.min.css');
$col->addCss('/assets/css/style.css');
$col->addCss('/assets/css/footer.css');

$col = $assets->collection('jsCore');
$col->addJs('/assets/js/jquery-3.2.1.min.js');
$col->addJs('/assets/js/popper.min.js');
$col->addJs('/assets/js/bootstrap.min.js');
```

Carga de bootstrap.css

Carga de jquery.min.js

Carga de Popper.min.js

Carga de boostrap.min.js

Figura 3.4 Carga de los recursos CSS y js utilizados para la vista. jQuery es necesario para que el *plugin* funcione.

*Smollett, C. (s.f.). Jquery simple slider. Recuperado de https://codepen.io/CaptainSmollett/pen/epGWJa

```
// == GALERIA ==
$col = $assets->collection('cssGaleria');
$col->addCss('/assets/css/slider.css');

$col = $assets->collection('jsGaleria');
$col->addJs('/assets/js/slider.js');
$col->addJs('/assets/js/transicion.js');
```

Carga del código CSS del slider

Carga de los js para el funcionamiento del slider

Figura 3.5 Carga de los recursos js con el código del *plugin.*

- CSS -> slider.css. Este archivo queda tal cual se copió y no se hacen modificaciones.
- JS -> slider.js y transicion.js. En la siguiente imagen se explica cómo separar el código.

```
// este código es lo unico que necesitamos para transicion.js,  todo lo de arriba lo colocamos en slider.js
$(".slider-container").sliderUi({
  speed: 700,
  cssEasing: "cubic-bezier(0.285, 1.015, 0.165, 1.000)"
});
```

Figura 3.6 Porción de código que se copiara en “transición.js”. El código restante se coloca en “slider.js”.

3.2.3 Creación de la vista

El código en la vista (HTML) cambia a como lo ejemplifica el *plugin* ya que ahí se cargan las imágenes con su respectiva descripción de forma estática y así no es como se requirió. Entonces, para cargar las imágenes dinámicamente utilizamos jQuery, es por ello que se requiere de un con id = “imagenes”.

Lo primero que se realizó fue crear la vista, que como sabemos, en **Phalcon** se ubican en *app/views/<nombre del controlador (en este caso es galería)>/* con el nombre “index.volt”. Dentro colocamos el siguiente código:

```
<div class="slider-container" id="caption-slide" >
    <div class="slider" id="imagenes">
    </div>
    <div class="switch" id="prev"><span></span></div>
    <div class="switch" id="next"><span></span></div>
</div>
```

Figura 3.7 El con id=”imagenes” que se usa para insertar las imágenes.

- Los con las clases “switch” son controles para avanzar a la siguiente imagen o regresar a la anterior.
- El con el id=”caption-slide” sirven de referencia para habilitar la descripción de la imagen.

3.2.4 Creación del archivo JSON

JSON (acrónimo de *JavaScript Object Notation*), es un formato de texto ligero para el intercambio de datos en la red, muy popular, incluso la mayoría de lenguajes de programación contienen funciones para crear, analizar y procesar este tipo de formatos.

Dentro de estos archivos podemos procesar cadenas, números, booleanos y nulos, también objetos y arreglos.

La sintaxis debe seguir las siguientes reglas:

- Los datos se separan por comas.
- los datos se escriban en pares donde el primero es la clave y el segundo el valor.
- Los objetos JSON contienen llaves {} de apertura y cierre.
- Los corchetes [] guardan arreglos, inclusive otros objetos JSON.

Un ejemplo de un objeto JSON que guarda un artículo con su precio sería:

```
var objetoJSON = {"articulo":"refresco","precio":"8"};
```

Ya sabiendo esto explicaremos como se crea el JSON para la carga de las imágenes:

```
[{"imagenes":[
    {"descripcion":"Conferencia dentro del Centro Técnologico Aragón",
      "ruta":"assets/img/galeria/conferencia_cta.jpg"},
    {"descripcion":"Centro Técnologico Aragón",
      "ruta":"assets/img/galeria/cta_aragon.jpg"},
    {"descripcion":"Laboratorio de seguridad informática",
      "ruta":"assets/img/galeria/seguridad_informatica.jpg"}
    ]
 }]
```

Figura 3.8 Contenido del archivo “imágenes.json”.

De la **Figura 3.8** debemos conocer lo siguiente:

- Tenemos corchetes de apertura y cierre y dentro definimos un arreglo “imagenes” donde colocamos los objetos con las claves “descripcion” y “ruta”.
- Cada objeto está encerrado con llaves y separado por comas
- Si queremos agregar más objetos, tenemos que colocar al final del último elemento una coma y después colocar llaves de apertura y cierre. Adentro definimos su descripción y ruta.
- Las imágenes en **Phalcon** se colocarán en *public/assets/img.*
- Este archivo se guardó en *public/assets/json*. Primero se creó la carpeta *json* y se colocó el archivo con el nombre “imágenes.json”.

3.2.4 Carga dinámica de imágenes

El archivo “transicion.js” queda de la siguiente manera:

```
1  $.ajax({
2              url:  base_url+'/assets/json/imagenes.json',
3              dataType: 'json',
4              success: function(data){
5
6                  $.each(data, function(index){
7
8                      var imagenes = data[index].imagenes;
9
10                     $.each(imagenes, function(_index){
11                         $("#imagenes").append("<div><img src='"+base_url+imagenes[_index].ruta+"'>"+
12              "<span class='caption'>"+imagenes[_index].descripcion+"</span></div>");
13                     })
14                 })
15             $("#slider-container").sliderUi({
16                 speed: 700,
17                 cssEasing: "cubic-bezier(0.285, 1.015, 0.165, 1.000)"
18             });
19                 $("#caption-slide").sliderUi({
20                     caption: true
21                 });
22             }
23 });
```

Figura 3.9 Contenido del archivo “transicion.js”.

De la **Figura 3.9** analizamos las siguientes líneas de código:

- En las líneas 15-18 pegamos la parte de código que separamos de “slider.js”.
  Por medio de *sliderUI* podemos configurar los efectos y la velocidad de transición.

- En las líneas 19-21 seleccionamos el elemento con id=”caption-slide” y hacemos que muestre las descripción de la imagen con *caption* y el valor *true.* Esta función se encuentra en la documentación del slider, por lo tanto no es necesario crearla.

Nota que el código está dentro de una función AJAX

- En la línea 1 se define la URL a donde se hace la petición.
- La URL se define por la variable “base_url” declarada en el archivo “index.volt” ubicado en *app/views/* antes de cargar los recursos js:

```
<body>
  <div class="container">

  {# Mostrar cabecera institucional en la página principal #}
  {% include 'shared/header.volt' %}
  {{ content() }}
  {# Mostrar pie de página institucional en la página principal #}

  </div>
  <script>var base_url = '//' + location.host + "{{ static_url('/') }}";</script>
  {{ assets.outputJs('jsCore') }}
  {{ assets.outputJs('jsNavbar') }}
  {% if assets.exists('js' ~ idVista) %}
  {{ assets.outputJs('js' ~idVista) }}
  {% endif %}
</body>
```

Aquí se declara la variable con nombre del host concatenado con la URL estática y una diagonal

Figura 3.10 Contenido del archivo “index.volt”. Dentro definimos la variable “base_url”.

- *Location.host* es una función JavaScript que devuelve el nombre de host actual y el puerto de la URL.
- *static_url()* es un método del motor de plantillas VOLT que concatena la ruta estática del proyecto con lo que se ingrese de parámetro.
  Por ejemplo, si nos encontramos en un servidor local (*localhost*) y el nombre del proyecto se llama CTA, el valor de la variable “base_url” es*:*

http://localhost/CTA/

location.host

static_url()

Continuando el análisis de la **Figura 3.9**, tenemos que:

- En la línea 3 se define el tipo de datos que se solicitan por medio de *dataType* con el valor "json".
- La función en la línea 4 se va a ejecutar cuando la petición se realice satisfactoriamente:

```
  $.each(data, function(index){

      var imagenes = data[index].imagenes;

      $.each(imagenes, function(_index){
          $("#imagenes").append("<div><img src='"+base_url+imagenes[_index].ruta+"'>"+
"<span class='caption'>"+imagenes[_index].descripcion+"</span></div>");
      })
  })
```

  - Tenemos dos sentencias de control *each* anidados.
  - El primer *each* itera (con "index") cada objeto del arreglo "imagenes" de "imágenes.json" y lo asigna a la variable "imágenes".
  - El segundo *each* itera la variable (con "_index"), después se selecciona el elemento HTML con id = "#imagenes" e insertamos el código HTML con jQuery y el método *append().*
  - Para definir la ruta completa donde se guardan las imágenes utilizamos de nuevo la variable "base_url".
  - El ciclo termina de ejecutarse cuando ya no haya objetos por recorrer en el JSON.

3.3 Módulo directorio

Este módulo contiene una lista con información de contacto de todos los trabajadores y académicos que conforman el Centro Tecnológico Aragón, además de dos entradas de datos tipo radio y un buscador en tiempo real para poder filtrar los contactos por medio de los datos que introduzca el usuario.

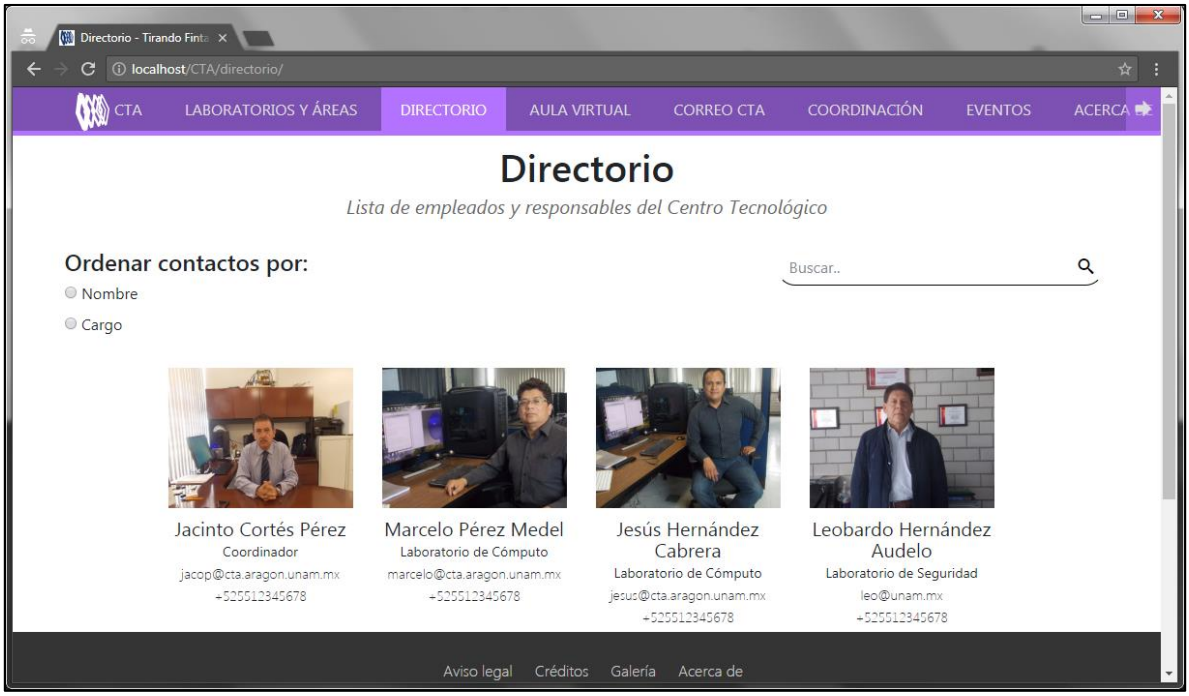


Figura 3.11 Página del módulo “directorio”.

El módulo funciona de la siguiente manera:

- Cuando un usuario accede a directorio por medio del menú principal o de navegación, el módulo muestra toda la lista con datos de contacto y foto de cada persona que labora en el Centro Tecnológico.
- Cuando se selecciona el radio botón “Nombre”, la lista se ordenará alfabéticamente por nombre de contacto.
- Cuando se selecciona el radio botón “Cargo”, la lista se ordenará alfabéticamente por cargo.
- Cuando se ingresa un dato en el buscador con la etiqueta “buscar”, la lista muestra en los contactos que contengan el valor introducido.
- Si esta seleccionado el radio botón “Nombre” y además se introduce un dato, la lista muestra los contactos que contengan el valor introducido ordenados alfabéticamente por nombre
- Si esta seleccionado el radio botón “Cargo” y además se introduce un dato, la lista muestra los contactos que contengan el valor introducido ordenados alfabéticamente por cargo.
- Si el buscador tiene un dato y el usuario selecciona cualquier radio botón, la lista muestra los contactos que contengan el valor introducido ordenados alfabéticamente por el radio botón seleccionado.

El funcionamiento de este módulo implica la ejecución de varias condicionales para filtrar los contactos de manera correcta, es necesario hacerlo de forma automática utilizando AJAX, que se va a encargar de realizar las peticiones en segundo plano al modelo de la tabla en la base de datos. Cuando la página se carga por primera vez no necesitamos de AJAX, se pueden cargar los datos por medio del controlador realizando la consulta "*SELECT* *" y posteriormente mandarlos a la vista.

La tabla en base de datos se llama "empleado" y presenta la siguiente estructura:

- Id_empleado. Identificador (*primary key*)
- cargo
- mensaje
- activo. Cambia de valor cuando el empleado deje de trabajar en el Centro Tecnológico
- correo
- nombre
- paterno
- materno
- ruta_foto. Ubicación de la foto de contacto (*public/assets/img…..*)

+ Opciones

	id_empleado	cargo	mensaje	activo	correo	nombre	paterno	materno	telefono	ruta_foto
Editar Copiar Borrar	1	Coordinador	Lorem ipsum dolor sit amet, consectetur adipiscing...	1	jacop@cta.aragon.unam.mx	Jacinto	Cortés	Pérez	+525512345678	/assets/img/directorio/jcp.jpg
Editar Copiar Borrar	4	Laboratorio de Cómputo		1	marcelo@cta.aragon.unam.mx	Marcelo	Pérez	Medel	+525512345678	/assets/img/directorio/mpm.jpg
Editar Copiar Borrar	5	Laboratorio de Cómputo		1	jesus@cta.aragon.unam.mx	Jesús	Hernández	Cabrera	+525512345678	/assets/img/directorio/jhc.jpg
Editar Copiar Borrar	6	Laboratorio de Seguridad	NULL	1	leo@unam.mx	Leobardo	Hernández	Audelo	+525512345678	/assets/img/directorio/lha.jpg

Figura 3.11 Campos de la tabla empleado. Los registros son solo ejemplos.

Para este módulo solo nos interesan mostrar los campos con el cargo, correo, nombre completo teléfono y la ruta para la foto.

3.3.1 Creación de la vista

Lo primero que se hace es crear la vista, que como sabemos, en **Phalcon** se ubican en *app/views/<nombre del controlador (en este caso es directorio)>/* con el nombre "index.volt". El código que contiene es el siguiente:

```
<div>
  <h1>Directorio</h1>
  <h3>Lista de empleados y responsables del Centro Tecnológico</h3>
</div>
<!-- aquí se agregan los controles de búsqueda -->
<div class="container">
  <div class="row">
    <div class="col order-12">
        <h4>Ordenar contactos por:</h4>
                <div class="btn-group-vertical">
                    <div class="radio">
                        <label><input type="radio" value="1" name="radio-boton" class="radio-style" id="nombre">
                          Nombre</label><br>
                        <label><input type="radio" value="2" name="radio-boton" class="radio-style"> Cargo</label>
                    </div>
                </div>
    </div>
    <div class="col-md-4 order-md-12">
      <div class="caja-texto inner-addon right-addon">
          <i class="glyphicon icon-search"></i>
          <input type="text" class="form-control" id="buscar" placeholder="Buscar..">
      </div>
    </div>
  </div>
</div>
<!-- seccion donde se muestra la lista de contactos -->
 <section class="team">
  <div class="container">
    <div class="row">
      <div class="col-md-10 col-md-offset-1">
          <div class="row pt-md" id="result">
              <!-- con esta variable agregamos los contactos desde el controlador  -->
              {{html}}
          </div>
      </div>
    </div>
  </div>
</section>
```

Figura 3.12 Contenido del archivo “index.volt”.

Analizando el código de la figura anterior nos encontramos con clases de Bootstrap:

- la clase *col order-12* hace que el buscador se coloque por encima de los botones tipo radio y no al revés como lo hace Bootstrap, cuando el tamaño de la pantalla es pequeño (**Figura 3.13**).

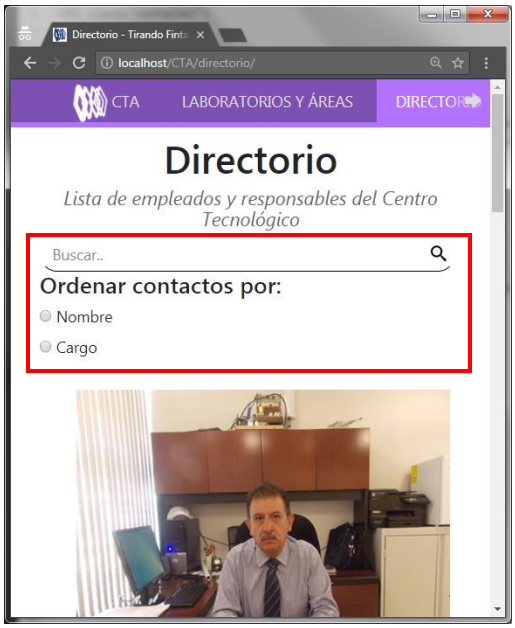


Figura 3.13 Modulo directorio en versión para móvil (tamaño pequeño).

- Las clases *col-m4 order-md-12* sirven para ordenar el buscador hasta la parte derecha de la columna para tamaños de pantalla medianos o grandes.

Figura 3.14 Orden de los elementos en versiones para escritorio.

- *col-md-1, col-md-offset-1* la primera clase es una columna de tamaño 10 y la segunda recorre el una columna a la derecha cuando el tamaño de la pantalla es mediano, para evitar que se desfase la lista con los contactos.
- Por último, tenemos un con id = “result” y dentro los delimitadores del motor de plantillas VOLT con la variable “html” que va a contener el código para imprimir toda la lista de contactos provenientes del controlador.

3.3.1 Creación del modelo

Esta parte es sencilla solo tuvimos que añadir el modelo utilizando las *herramientas de desarrollo* explicadas en el capítulo 1.

- Recuerda que el nombre del modelo debe ser igual al de la tabla de la base de datos.

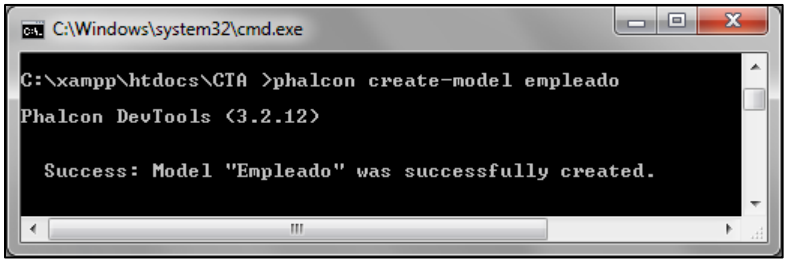


Figura 3.15 Creación del modelo “Empleado” utilizando las *herramientas de desarrollo*.

3.3.2 Creación del controlador

El nombre del controlador es “DirectorioController” fue creado con las herramientas de desarrollo y dentro de su acción por defecto (*indexAction()*) se realiza la consulta para traer todos los contactos, tratarlos y mandarlos a la vista.

```
public function indexAction()
{
    $empleados = Empleado::find();
    $html =  $this->generaHtml($empleados);
    $this->view->setVar('html', $html);
}
```

Figura 3.16 Acción “index” del controlador “DirectorioController”.

- Se crea una variable $empleados y se le asigna lo que retorna el método *find()*.

Método	Con parámetro	Sin parámetro
find()	- Si es una condición devuelve todos los registros que cumplan con la condición ingresada. También se puede combinar con cláusulas SQL	retorna todos los registros de la tabla

Tabla 3.1 Definición del método *find()*.

Continuando con el análisis de la **Figura 3.16**, se define lo siguiente:

- Se crea la variable $html y se le asigna lo que retorna la función *generaHtml()*, recibe por parámetro la variable $empleado (la que tiene el resultado de la consulta).

- la última línea es un método de **Phalcon** para mandar a la vista la variable $html con el nombre “html” (en rojo)

```
$this->view->setVar('html', $html);
```

- La función *generaHtml()* se declara antes de la acción “indexAction()”, y contiene lo siguiente:

```
class DirectorioController extends \Phalcon\Mvc\Controller
{
    public function generaHtml($empleados){
        $html="";
        foreach($empleados as $empleado)
        {
        $html .= '<div class="col-lg-3 col-md-3 col-sm-4 col-xs-12 profile">
              <div>
                <img src="'.$this->url->getStatic($empleado->ruta_foto).'" class="img-fluid">
              </div>
              <h1 class="text-center">'.$empleado->nombre.' '.$empleado->paterno.' '.$empleado->materno.'</h1>
              <strong> <h2 class="text-center">'.$empleado->cargo.'</h2></strong>
              <h2 class="text-center">'.$empleado->correo.'</h2>
              <h2 class="text-center">'.$empleado->telefono.'</h2>
            </div>';
        }
        return $html;
    }
```

Figura 3.17 Contenido de la función *generaHtml()*.

Revisemos que es lo que hace esta función:

- Creamos la variable $html de tipo *string.*
- Concatenamos los datos y el HTML con todos los contactos.
- *foreach* recorre la variable $empleados que fue pasada por parámetro.
- Por último retornamos la variable $html.

3.3.3 Estilos

Los estilos agregados fueron sencillos utilizando las reglas de CSS vistas en el capítulo 2: entre ellos se puede ver el tamaño de letra márgenes y bordes para los elementos y también para el diseño del buscador.

```
.parrafos{
    font-size: 12px;
    margin-top: 0px;
}
.row {
    margin-right: -9px;
    margin-left: -15px;
}
.col-md-offset-1{
    margin-left:8.33333333%;
}

h6.description{
  font-weight: bold;
  letter-spacing: 2px;
  color: #999;
  border-bottom: 1px solid rgba(0, 0, 0,0.1);
  padding-bottom: 5px;
}
.profile{
  margin-top: 25px;
}
.profile h1{
  font-weight: normal;
  font-size: 20px;
  margin:10px 0 0 0;
}
.profile h2{
  font-size: 14px;
  font-weight: lighter;
  margin-top: 5px;
}
```

```
#buscar{
    border-top-width: none;
    border-top-color: white;
    border-bottom-color: black;
    border-left-color: white;
    border-right-color: white;
    border-width: 1px;
    border-radius: 20px;
}

#buscar:focus{
    border-top-width: 0px;
    border-bottom-color: black;
    border-left-color: white;
    border-right-color: white;
    border-width: 1px;
    border-radius: 20px;
}
.radio-style{
    background-color: blueviolet;
    width: 230px:
}
```

Estilos al buscador con id = “buscar”

Estilos al radio botón

Figura 3.18 Estilos a los elementos de la vista “directorio”.

3.3.4 Búsqueda y filtrado de los contactos

Hasta aquí la página mostraría todos los contactos y con estilo CSS, pero el buscador y los radio botón aun no funcionan y deben trabajar en conjunto.

Al realizar la búsqueda se deben considerar los posibles escenarios, cuando:

- Se elige ordenar por nombre o cargo sin introducir nada en el buscador.
- Se elige ordenar y el buscador contiene datos.
- Se introducen datos en el buscador, pero no está selecciona ninguna forma de ordenamiento.
- El buscador contiene datos y también se selecciona una forma de ordenamiento.

Sabiendo esto, en la vista donde se agregaron los radios botón, se les asigno un valor para cada uno:

```
<label><input type="radio" value="1" name="radio-boton" class="radio-style" id="nombre">
  Nombre</label><br>
<label><input type="radio" value="2" name="radio-boton" class="radio-style"> Cargo</label>
```

- 1 para nombre
- 2 para cargo

El archivo js es muy extenso, es mejor explicarlo por partes.

```
1  $(document).ready(function(){
2      $('#buscar').attr('value','');
3      $('input[name=radio-boton]').attr('checked',false);
4      $("input[name=radio-boton]").click(function () {
```

Figura 3.19 Inicio del documento js.

- En la línea 1 se indica el inicio del documento.
- En la línea 2 se selecciona el elemento con id = “buscar”, en este caso es el buscador y le asigna el valor vacío. Esto nos sirve para borrar lo que se encuentre escrito al recargar la página.
- En la línea 3 se tiene función similar a la anterior, pero ahora es para los botones de radio.
- En la línea 4 se selección ambos radio botón y se les asigna el evento *click()*, así, cuando se seleccione alguno de estos se ejecute la función de la **Figura 3.20**.

```
4       $("input[name=radio-boton]").click(function () {
5       var radio_valor = $('input:radio[name=radio-boton]:checked').val();
6       var valor_input = $('#buscar').val();
7
8           if(radio_valor == 1 && valor_input==""){
9               $.ajax({
10              type: 'POST',
11              url: base_url+'/directorio/porNombreVacio',
12              data:{'radio_valor': radio_valor,'valor_input':valor_input},
13              })
14              .done(function(resultado){
15              $('#result').html(resultado)
16              })
17              .fail(function(){
18              alert('Hubo un error')
19              })
20          }else if(radio_valor==1 && valor_input != ""){
21              $.ajax({
22              type: 'POST',
23              url: base_url+'/directorio/porNombreDatos',
24              data:{'valor_input':valor_input},
25              })
26              .done(function(resultado){
27              $('#result').html(resultado)
28              })
29              .fail(function(){
30              alert('Hubo un error')
31              })
32          }
33
34          if(radio_valor == 2 && valor_input==""){
35              $.ajax({
36              type: 'POST',
37              url: base_url+'/directorio/porCargoVacio',
38              })
39              .done(function(resultado){
40              $('#result').html(resultado)
41              })
42              .fail(function(){
43              alert('Hubo un error')
44              })
45          }else if(radio_valor==2 && valor_input != ""){
46              $.ajax({
47              type: 'POST',
48              url: base_url+'/directorio/porCargoDatos',
49              data:{'valor_input':valor_input},
50              })
51              .done(function(resultado){
52              $('#result').html(resultado)
53              })
54              .fail(function(){
55              alert('Hubo un error')
56              })
57          }
58      });
```

Bloque 1

Bloque 2

Bloque 3

Bloque 4

Bloque 5

Figura 3.20 Esta función está dividida en cinco bloques que se explican a más adelante.

Para el bloque 1 tenemos lo siguiente:

- Se toma el valor del radio botón (1 o 2) y se asigna a la variable “radio_valor”
- Se toma el valor del buscador con id=”buscar” y se asigna a “valor_input”

Teniendo los valores de ambos controles, deben analizarse mediante el condicional *if anidado* con las combinaciones resultantes. Los bloques 2-5 tiene una estructura similar, solo cambia la condición a evaluar, los datos enviados y la URL donde se realiza la petición AJAX.

El bloque 2 se ejecuta cuando:

- “radio-valor” = 1 y “valor_input” es vacío
- La URL a donde se hace la “valor_input” *http:///localhost/CTA/directorio/porNombreVacio*

El bloque 3 se ejecuta cuando:

- “radio-valor” = 1 y “valor_input” es diferente de vacío
- La URL a donde se hace la petición es: *http://localhost/CTA/directorio/porNombreDatos*

El bloque 4 se ejecuta cuando:

- “radio-valor” = 2 y “valor_input” es vacío
- La URL a donde se hace la petición es: *http://localhost/CTA/directorio/porCargoVacio*

El bloque 5 ejecuta cuando:

- “radio-valor” = 3 y “valor_input” es diferente de vacío
- La URL a donde se hace la petición es: *http://localhost/CTA/directorio/porCargoDatos*

Cuando la petición POST de cada bloque se hace correctamente (.*done*) se ejecuta la función donde *resultado* se inserta con el método *html()* al elemento del DOM con id = “result”. Si la petición falla (.*fail*) se ejecuta la función que muestra una alerta.

El controlador es el mismo para todas las peticiones (“directorio”), lo que cambia es la acción:

- *porNombreVacioAction()*
- *porNombreDatosAction()*
- *porCargoVacioAction()*
- *porCargoDatosAction()*

Revisemos el código que contiene cada acción:

```
32      public function porNombreVacioAction()
33      {
34
35          $empleados = Empleado::find(array(
36          "order" => "nombre"
37          ));
38          $html =  $this->generaHtml($empleados);
39          return $html;
40
41      }
42
```

Figura 3.21 Acción “porNombreVacio”.

- la variable $empleados recibe los datos del método *find()* que espera por parámetro un arreglo de tipo clave – valor para ordenar por nombre.
- creamos una variable $html que recibe lo que retorna el método *generaHtml()*. Este método se explicó anteriormente, solo recorre la variable $empleados y concatena los datos con etiquetas HTML.
- Al final retornamos $html a la petición AJAX.

```
43      public function porNombreDatosAction()
44      {
45          $buscar=$this->request->getPost("valor_input");
46
47          $empleados = $this->modelsManager->executeQuery(
48            "SELECT * FROM Empleado WHERE CONCAT(nombre,' ',paterno,' ',materno) LIKE '%$buscar%' ORDER BY nombre");
49           $html =  $this->generaHtml($empleados);
50           return $html;
51
52      }
```

Figura 3.22 Acción “porNombreDatos”.

- $buscar recibe con el método *request->getPost()* la variable “valor_input” enviada por POST.
- Se ejecuta la consulta que ordena los datos por nombre con los resultados que contengan el valor de la variable $buscar. Esto se logra con las cláusulas *LIKE y ORDER BY.*
- *CONCAT* concatena el nombre completo del contacto separado por un espacio en blanco.
- Se procesa $empleados con el método *generaHtml()* y se retorna la variable $html.

```
54      public function porCargoVacioAction()
55      {
56
57          $empleados = Empleado::find(array(
58          "order" => "cargo"
59          ));
60          $html =  $this->generaHtml($empleados);
61          return $html;
62
63      }
```

Figura 3.23 Acción “porCargoVacio”.

- Es similar a *porNombreVacioAction()*, solo cambia la consulta al modelo: trae todos los contactos ordenados alfabéticamente por cargo.

```
65      public function porCargoDatosAction()
66      {
67          $buscar=$this->request->getPost("valor_input");
68
69          $empleados = $this->modelsManager->executeQuery(
70      "SELECT * FROM Empleado WHERE CONCAT(nombre,' ',paterno,' ',materno) LIKE '%$buscar%' ORDER BY cargo");
71          $html =  $this->generaHtml($empleados);
72          return $html;
73
74      }
```

Figura 3.24 Acción “porCargoDatos”.

- Similar a *porNombreDatosAction()*, solo cambia la consulta, aquí se ordena por cargo.

Hasta aquí terminamos la primera parte del archivo js. De las acciones anteriores se ejecutan cuando un radio botón se selecciona y previamente el buscador puede estar vacío o con datos, pero ¿qué pasa cuando entramos por primera vez a la página y no seleccionamos ningún radio botón y directamente escribimos en el buscador? Es por ello que nos hacen falta más opciones de búsqueda.

Continuando con el análisis del archivo js, encontramos una serie más de *if anidados* (**Figura 3.25**).

```
60      $('#buscar').on('keyup',function(){                                    Bloque 6
61          var radio_valor = $('input:radio[name=radio-boton]:checked').val();
62          var valor_input = $('#buscar').val()
63          if(radio_valor == 1){                                              Bloque 7
64             $.ajax({
65              type: 'POST',
66              url: base_url+'/directorio/buscarConNombre',
67              data:{'valor_input':valor_input},
68              })
69              .done(function(resultado){
70              $('#result').html(resultado)
71              })
72              .fail(function(){
73              alert('Hubo un error')
74              })
75          }else if(radio_valor==2){                                          Bloque 8
76              $.ajax({
77              type: 'POST',
78              url: base_url+'/directorio/buscarConCargo',
79              data:{'valor_input':valor_input},
80              })
81              .done(function(resultado){
82              $('#result').html(resultado)
83              })
84              .fail(function(){
85              alert('Hubo un error')
86              })
87          }else{                                                             Bloque 9
88              $.ajax({
89              type: 'POST',
90              url: base_url+'/directorio/buscar',
91              data:{'valor_input':valor_input},
92              })
93              .done(function(resultado){
94              $('#result').html(resultado)
95              })
96              .fail(function(){
97              alert('Hubo un error')
98              })
99          }
100
101     })
102 });
```

Figura 3.25 Segunda parte del archivo js está dividido en cuatro bloques más para que pueda ser estudiado.

El bloque 6 se ejecuta cuando el usuario ingresa datos al buscador (con el id = ”buscar”), esto se realiza con *keyup*.

- Se obtiene el valor del radio seleccionado (si es que hay alguno) y se asigna a la variable “radio_valor”.
- Se obtiene los datos introducidos con el teclado y se le asigna a la variable “valor_input”.

Con estos datos se ejecutará el bloque que cumpla con la condición:

- El bloque 7 se ejecuta si “radio_valor” = 1 (ordenar por nombre)

- El bloque 8 se ejecuta si “radio_valor” = 2 (ordenar por cargo)
- El bloque 9 se ejecuta si “radio_valor” no tienen ningún valor

La variable que se envía por POST mediante AJAX es “valor_input”, Cuando la petición POST de cada bloque se hace correctamente (*.done*) se ejecuta la función donde *resultado* se inserta con el método *html()* al elemento del DOM con id = “result”. Si la petición falla (*.fail*) se ejecuta la función que muestra una alerta.

El controlador es el mismo para todas las peticiones (directorio) en las URL´s, lo que cambia es la acción:

- *buscarConNombreAction()*
- *buscarConCargoAction()*
- *buscarAction()*

Analizando las acciones, se observa que son similares a las que revisamos con anterioridad:

```
76      public function buscarAction()
77      {
78          $buscar=$this->request->getPost("valor_input");
79
80          $empleados = $this->modelsManager->executeQuery(
81      "SELECT * FROM Empleado WHERE CONCAT(nombre,' ',paterno,' ',materno) LIKE '%$buscar%'");
82          $html =  $this->generaHtml($empleados);
83          return $html;
84
85
86      }
87
88      public function buscarConNombreAction()
89      {
90          $buscar=$this->request->getPost("valor_input");
91
92          $empleados = $this->modelsManager->executeQuery(
93      "SELECT * FROM Empleado WHERE CONCAT(nombre,' ',paterno,' ',materno) LIKE '%$buscar%' ORDER BY nombre");
94          $html =  $this->generaHtml($empleados);
95          return $html;
96
97
98      }
99
100      public function buscarConCargoAction()
101     {
102         $buscar=$this->request->getPost("valor_input");
103
104         $empleados = $this->modelsManager->executeQuery(
105     "SELECT * FROM Empleado WHERE CONCAT(nombre,' ',paterno,' ',materno) LIKE '%$buscar%' ORDER BY cargo");
106         $html =  $this->generaHtml($empleados);
107         return $html;
108
109     }
```

Figura 3.26 Contenido de las acciones “buscar”, “buscarConNombre”, “buscarConCargo”.

- $buscar recibe con el método *request->getPost()* la variable “valor_input” enviada por POST
- La consulta al modelo cambia para cada acción:
    - *buscarConNombreAction()* arroja los resultados ordenados por nombre y que además contengan el valor de la variable $buscar.
    - *buscarConCargoAction()* manda los resultados ordenados por cargo y que además contengan el valor de la variable $buscar.
    - *buscarAction()* retorna los resultados que contengan el valor de la variable $buscar.
- CONCAT concatena el nombre completo del contacto separado por un espacio en blanco
- Se procesa $empleados con el método *generaHtml(),* se asigna a $html y se retorna a la petición AJAX.

Ya abarcamos la estructura de la vista, estilos, el modelo, el controlador y las acciones y el archivo JavaScript. Con todo esto hacemos posible que nuestro buscador funcione perfectamente:

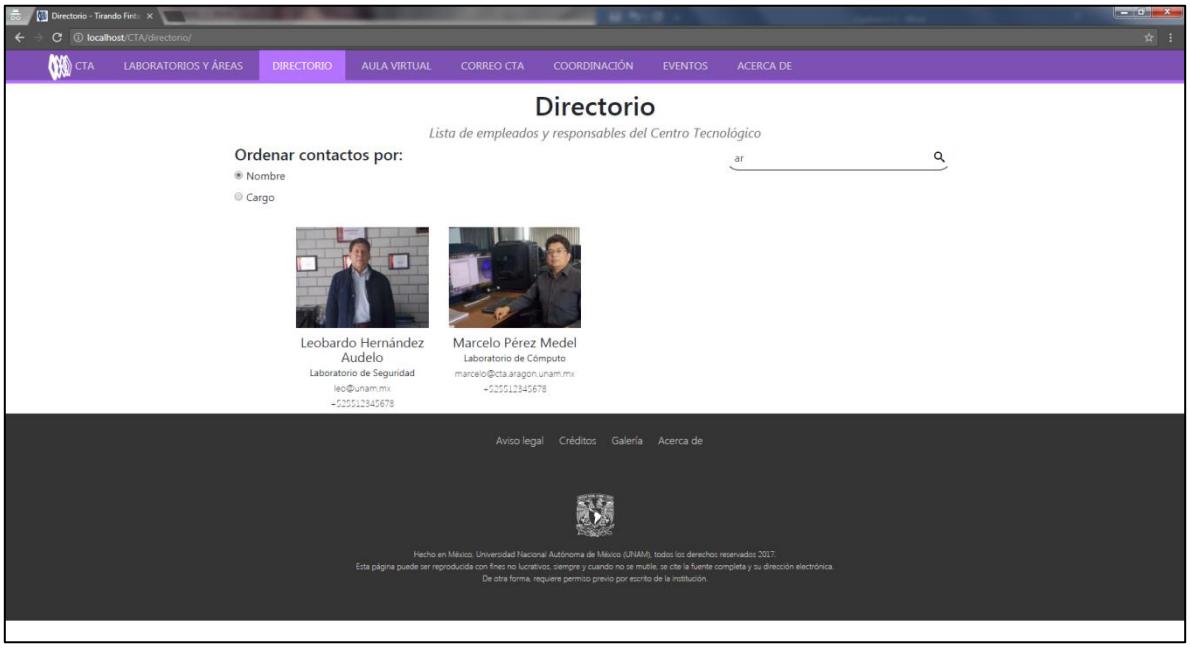


Figura 3.27 Página del módulo “directorio” donde se puede buscar un contacto por medio de los controles de búsqueda.

3.4 Módulo eventos

Cuando se selecciona desde el menú principal el apartado “Eventos”, la página carga un calendario en el mes y año actual, cuenta con un color distintivo para el día presente y otro color para los días con algún evento marcado.

En la parte inferior del calendario, se cuenta con una agenda que marca los eventos próximos a ocurrir, si el usuario elige un día marcado como evento, la agenda se actualizará con el evento seleccionado y aparecerá un botón “Detalles del evento” para que al dar clic se despliegue una ventana modal con información de las actividades a realizar durante el evento:

- Nombre de la actividad
- Descripción
- Ponente
- Inicio de la actividad
- Fin de la actividad

Un evento puede tener muchas actividades dependiendo el periodo de tiempo que dure, también es posible ver eventos pasados.

Si no existe un evento próximo, la agenda muestra un mensaje informando que no existen eventos.

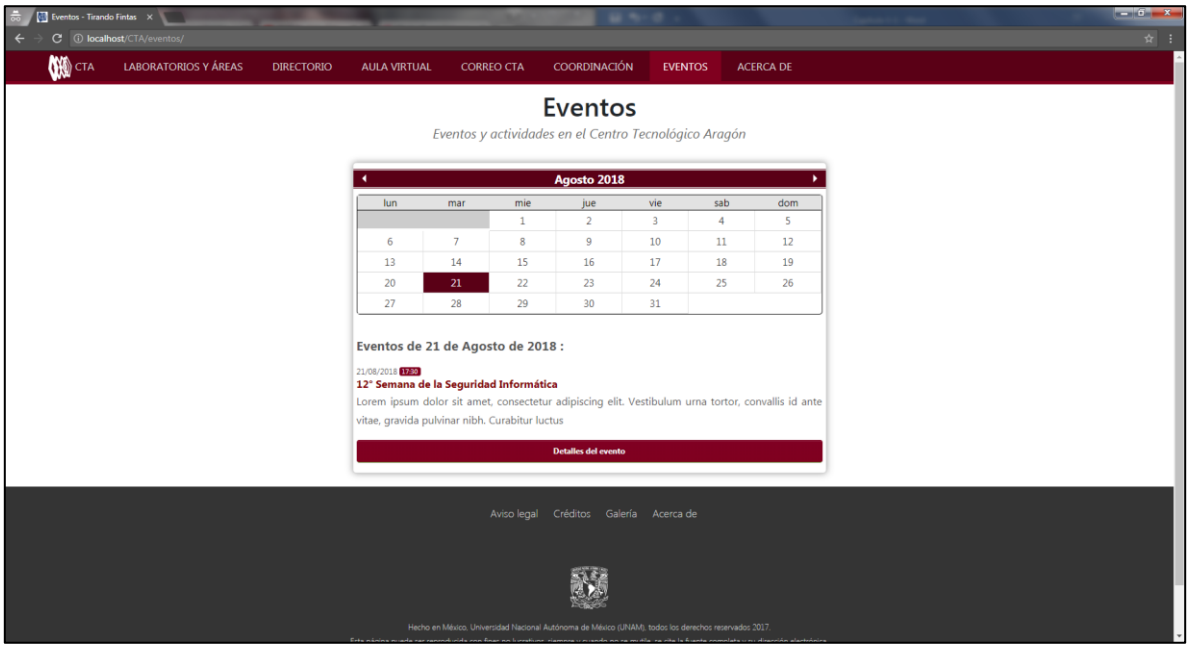


Figura 3.28 Página del módulo “eventos” con un calendario donde se marcan los eventos a ocurrir dentro del Centro Tecnológico.

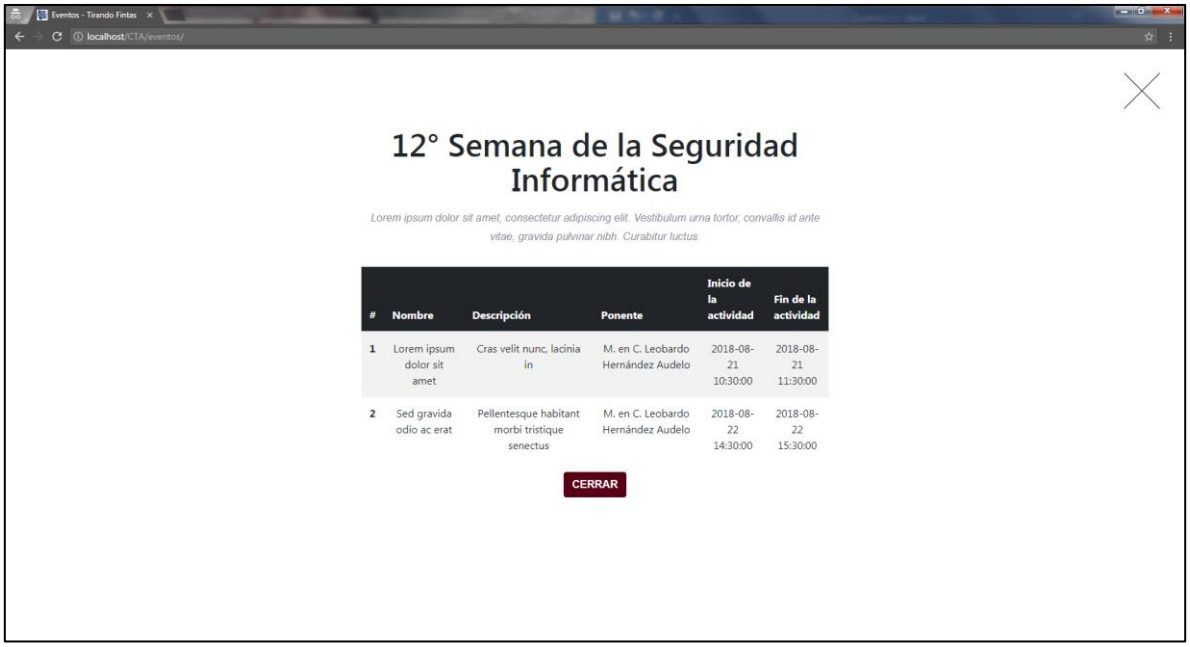
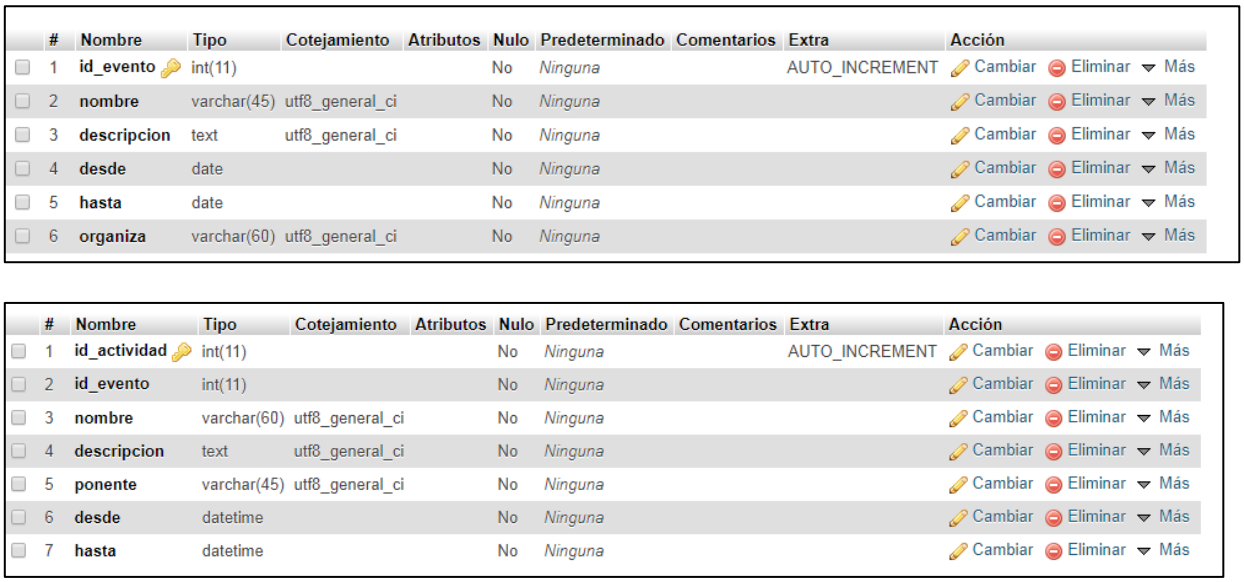


Figura 3.29 Ventana modal que se despliega al seleccionar el botón “Detalles del evento”.

La realización de este módulo implica el uso de dos tablas en la base de datos, una se encarga de mostrar los eventos y la otra en mostrar las actividades.



	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**id_evento**	int(11)			No	*Ninguna*		AUTO_INCREMENT	Cambiar Eliminar Más
☐	2	**nombre**	varchar(45)	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	3	**descripcion**	text	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**desde**	date			No	*Ninguna*			Cambiar Eliminar Más
☐	5	**hasta**	date			No	*Ninguna*			Cambiar Eliminar Más
☐	6	**organiza**	varchar(60)	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más

	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
☐	1	**id_actividad**	int(11)			No	*Ninguna*		AUTO_INCREMENT	Cambiar Eliminar Más
☐	2	**id_evento**	int(11)			No	*Ninguna*			Cambiar Eliminar Más
☐	3	**nombre**	varchar(60)	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	4	**descripcion**	text	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	5	**ponente**	varchar(45)	utf8_general_ci		No	*Ninguna*			Cambiar Eliminar Más
☐	6	**desde**	datetime			No	*Ninguna*			Cambiar Eliminar Más
☐	7	**hasta**	datetime			No	*Ninguna*			Cambiar Eliminar Más

Figura 3.30 Campos de la tabla “evento” y “actividad”.

los datos traídos desde la base de datos son guardados en un archivo JSON que podrá procesar el *plugin* con el calendario.

3.4.1 jQuery Event Calendar Plugin

El *plugin* del calendario se encuentra en la página *http://www.vissit.com/jquery-event-calendar-plugin-english-version/*. Una vez ahí encontraremos encontrar información acerca del autor*, ver una demo del calendario o un enlace a GitHub para descargarlo.

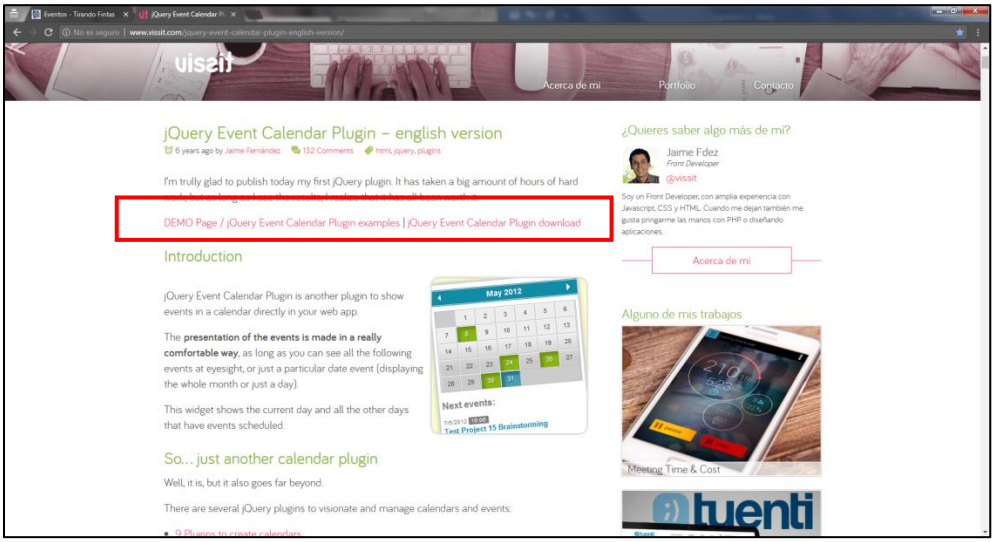


Figura 3.31 Descarga del *plugin* desde su página oficial**.**

Al dar *clic* sobre el enlace para ver la demo, veremos una explicación de todas las configuraciones y formatos de fecha y hora con las que cuenta el calendario.

- Para este caso se trabajó con la configuración *Human Date Format*

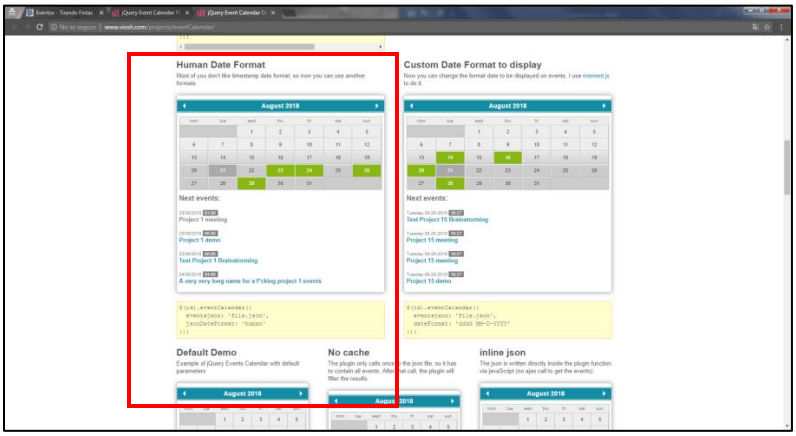


Figura 3.32 Configuración *Human Date Format***.**

*Fernández, J. (2012). jQuery Event Calendar Plugin – english version. Recuperado 20 mayo, 2018, de http://www.vissit.com/jquery-event-calendar-plugin-english-version/

Para descargarlo solo tenemos que dar clic en el enlace de descarga en página principal y una vez dentro de GitHub podemos ver todas las carpetas y ficheros, así como el código. Para tener el ZIP del *plugin* tenemos que dar clic al botón “Download ZIP” de la lista desplegable “Clone or download”

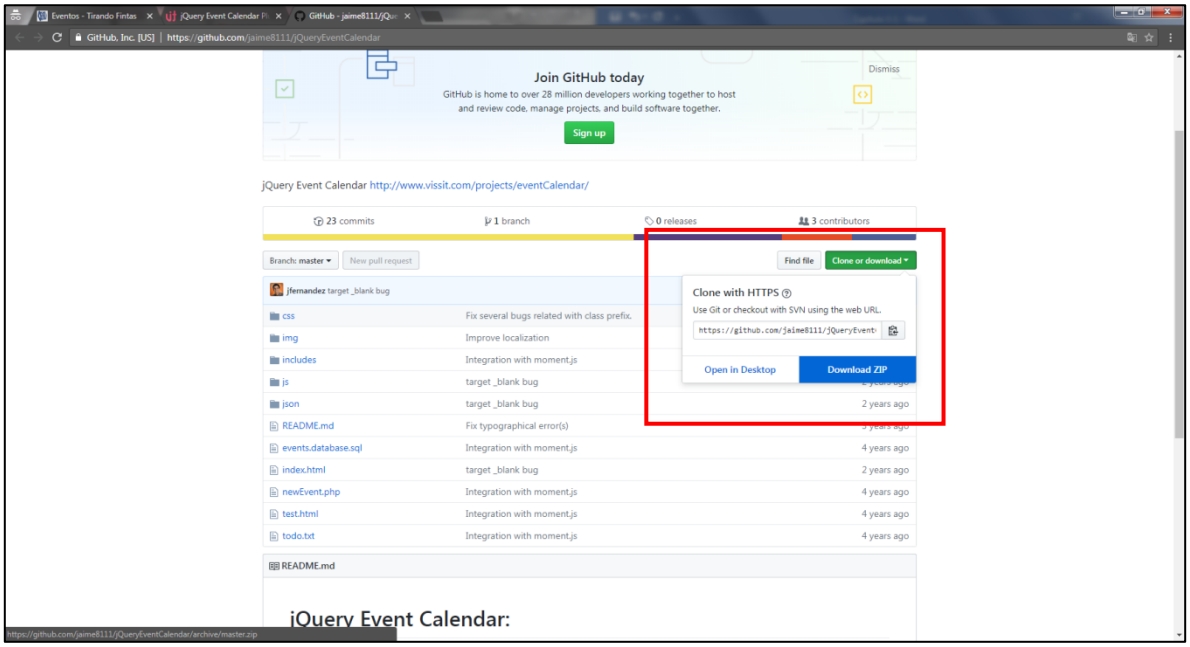


Figura 3.33 Descarga del Zip con el *plugin***.**

3.4.1 jQuery Migrate Plugin

Al integrar el *plugin* del calendario al sitio web del Centro Tecnológico, se encontraron incompatibilidades en las versiones de jQuery, el sitio fue planteado para trabajar con la versión 3.2.1 mientras que el calendario lo hace con la versión 1.9.1

```
18        <!-- Theme CSS file: it makes eventCalendar nicer -->
19        <link rel="stylesheet" href="css/eventCalendar_theme_responsive.css">
20
21        <!--<script src="js/jquery.js" type="text/javascript"></script>-->
22        <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js" type="text/javascript"></script>
```

Figura 3.34 Versión con la que trabaja el *plugin* (1.9.1).

En versiones reciente de jQuery se han cambiado características o comportamientos de versiones anteriores por lo tanto algunas funciones cambiaron su nombre, fueron actualizadas o eliminadas.

No todo estaba perdido ya que navegando por internet se encontró que jQuery contiene un *plugin* para simplificar la transacción de versiones anteriores a las actuales.

Existe dos versiones del *plugin*:

- Sin comprimir: Muestra en consola advertencias con los errores que se generan y así poder darles solución.
- Comprimida: Soluciona los errores de compatibilidad sin mostrar errores en consola.

En este caso se trabajó con la versión comprimida que podemos descargarla desde la página oficial de jQuery. Solo tenemos que descargarlo e importarlo como un recurso con Assets

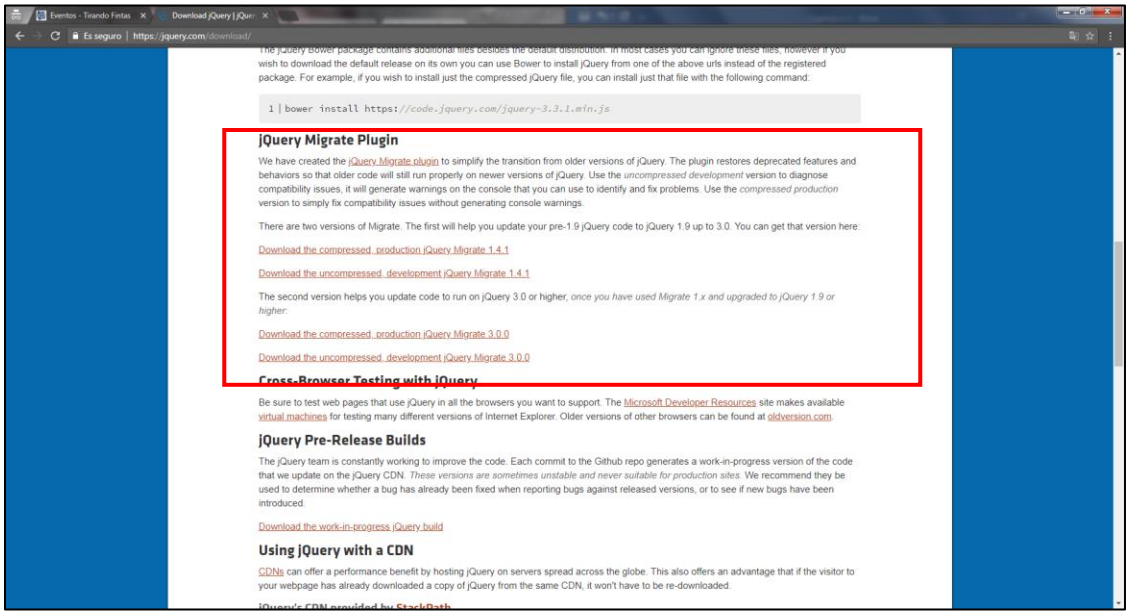


Figura 3.35 Descarga del *plugin jQuery Migrate.*

3.4.2 Importando archivos CSS y JS necesarios

Una vez descargado *jQuery Migrate Plugin* y el ZIP con el *plugin* del calendario, es momento de importarlos al sitio web del centro tecnológico. Esta es la lista de todos los archivos CSS y JS que conforman el módulo:

```
// == EVENTOS ==
$col = $assets->collection('cssEventos');
$col->addCss('/assets/css/eventCalendar.css');
$col->addCss('/assets/css/eventCalendar_theme_responsive.css');
$col->addCss('/assets/css/coordina.css');

$col = $assets->collection('jsEventos');
$col->addJs('/assets/js/jquery-migrate-3.0.0.min.js');
$col->addJs('/assets/js/moment.js');
$col->addJs('/assets/js/jquery.eventCalendar.js');
$col->addJs('/assets/js/calendario.js');
```

Figura 3.36 Importación de los elementos que componen el *plugin*.

- Se incluye *jQuery migrate* en la versión 3.0.0.
- “calendario.js” es un archivo que nosotros crearemos para agregar dinámicamente las funciones que hagan falta.
- Coordina.css contiene los estilos para la ventana modal.
- Todos los demás archivos pertenecen al *plugin* del calendario y se pueden encontrar dentro del ZIP descargado.

3.4.3 Creación de la vista

La vista fue editada para separar el código código JS y dejando únicamente el HTML necesario para mostrar el calendario y la ventana modal.

Figura 3.37 Elementos HTML que componen la vista.

- El con id = “eventCalendarHumanDate” nos va a servir para insertar el calendario y los eventos.
- La modal tiene el id=”portfolioModal1” y es por este que vamos a poder acceder a la ventana.

3.4.4 Creación del controlador y el archivo JSON

Hasta este momento ya sabemos que antes debemos de crear los modelos para las tablas “evento” y “actividad”, utilizando las *herramientas de desarrollo* es más que sencillo, así que solo nos enfocamos a revisar el código del controlador.

Una vez creado el controlador y teniendo la acción *indexAction()* debemos colocar ahí la parte encargada de traer todos los eventos desde la base de datos y guardarlos en un archivo JSON de forma que pueda ser reconocido por el calendario cuando se carga la página.

- El JSON debe guardarse en *public/src/json* con el nombre “eventos.json”

El código para la creación de JSON es el siguiente:

```
public function indexAction() {

    $evento_json = array();
    $separador=" ";
    $html_span = "<span class=\"d-none\">";
    $eventos = Evento::find();
    foreach($eventos as $evento)
    {
        $evento_json[] = array('date'=>$evento->desde.$separador."17:30:00",
                               'type'=> 'meeting',
                               'title'=>$evento->nombre,
                               'description'=>$evento->descripcion.$separador.$html_span.$evento->id_evento.'</span>',
                               'url'=>'#portfolioModal1');
    }
    //se crea el json
    $evento_json = json_encode($evento_json);
    $ruta = str_replace("app", "",APP_PATH).'public/assets/json/eventos.json';
    //se guarda el json el el archivo
    file_put_contents($ruta,$evento_json);
}
```

Figura 3.38 Código para generar el JSON.

- Primero se crea el arreglo $evento_json y se inicializa vacío.
- $separador solo nos sirve para colocar espacios en blanco.
- $html_span contiene la etiqueta con la clase de Bootstrap *d-none*, esto con el fin de saber que evento fue seleccionado por el usuario y así poder desplegar la ventana modal con las actividades correspondientes.

- $eventos contiene el resultado de la consulta al modelo Evento. El método *find()* es similar a *SELECT *.*
- Recorremos la variable evento y vamos llenando el arreglo $evento_json con la clave – valor siguiente:
    - 'date' su valor será la fecha y hora del evento, en este caso la hora se muestra sin variable para que se pueda ver el formato que lleva.
    - El valor 'type' es el que viene por defecto en el *plugin*.
    - ´title' tiene como valor el nombre del evento.
    - ´descripcion´ tiene como la descripción del evento, después un espacio en blanco y concatenado a la etiqueta con el id del evento. Por ejemplo, si el id fuera 1, el quedaría así: *1*
    - El valor de ´url´ es el id de la ventana modal.
- Una vez llenado el arreglo $evento_json tenemos que convertirlo a JSON con la función *json_encode().*
- La variable $ruta contiene el nombre y la URL donde son guardados.
- Con *file_put_contents()* guardamos el JSON en la ruta ingresada por parámetro.

Si abrimos el JSON generado veremos algo parecido a esto:

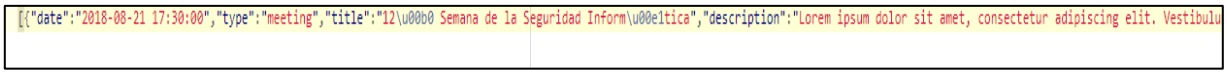


Figura 3.39 Contenido de "eventos.json".

3.4.4 Cargar el JSON al plugin del calendario

Al entrar a la página eventos el controlador genera el JSON con los datos de la tabla evento, una vez que se hace esto necesitamos cargar el archivo con el *plugin*, para esto utilizamos jQuery y algunos parámetros de configuración que viene incluidos en la documentación del calendario.

La primera parte del archivo "calendario.js" contiene el siguiente código:

```
1  $(document).ready(function() {
2      $("#eventCalendarHumanDate").eventCalendar({
3          eventsjson: base_url+'/assets/json/eventos.json',
4          locales: base_url+'/assets/json/locale.es.json',
5          jsonDateFormat: 'human'  // 'YYYY-MM-DD HH:MM:SS'+
6
7      });
8
```

Figura 3.40 Carga de "eventos.json" y configuración del *plugin*.

- Cuando el documento se carga completamente se selecciona el elemento con id = “eventCalendarHumanDate” y se ejecuta la función *eventCalendar*. Tenemos que pasar como parámetro:
    - la ruta donde se encuentra el JSON
    - El idioma del calendario
    - El formato ‘human’ para trabajar mejor con la fecha y hora

3.4.5 Acceder al id del evento por medio del botón “Detalles del evento”

Hasta aquí ya tenemos el calendario con los eventos, pero al seleccionar alguno, el botón “Detalles del evento” aun no funciona.

Tenemos que conocer algún atributo del botón para poder seleccionarlo utilizando jQuery, así que vamos al navegador (en este caso utilizamos Chrome) y presionamos F12 para abrir las *herramientas para desarrolladores.*

Una vez abierta la ventana, seleccionamos la herramienta ubicada en la parte superior izquierda o Ctrl + Shift + C y damos clic en el botón “Detalles del evento” (**Figura 3.41**).

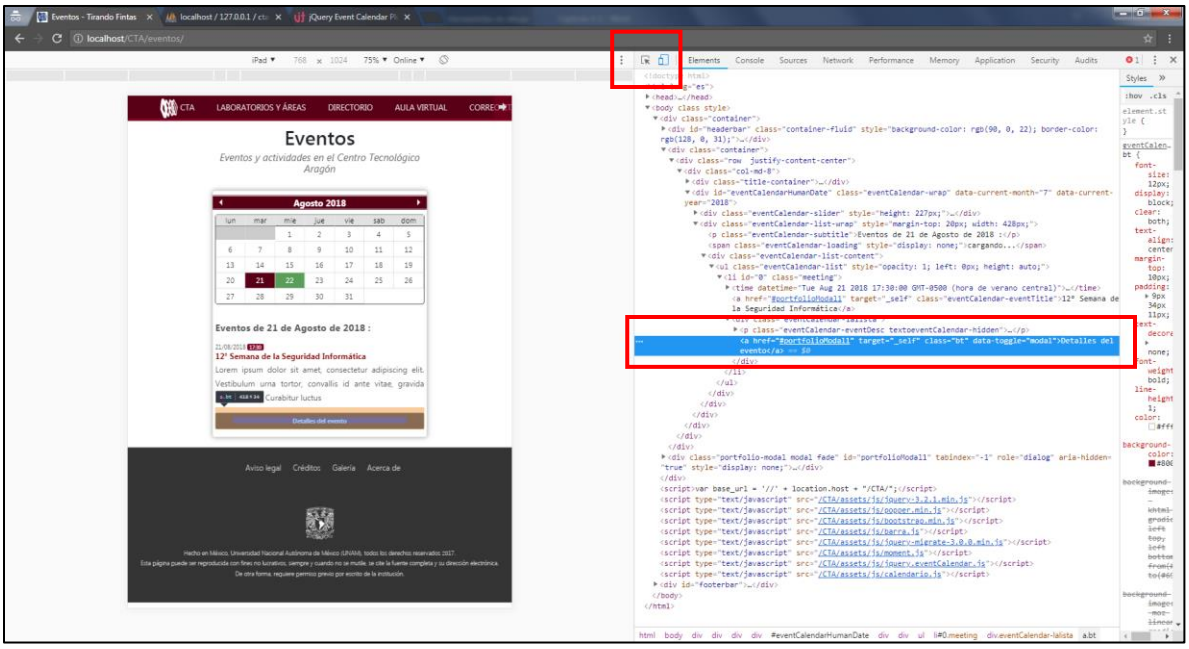
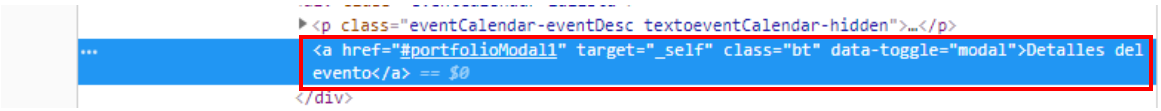


Figura 3.41 Selección del botón “Detalles del evento” utilizando las *herramientas para desarrolladores*.

Cuando seleccionamos el botón, en la parte derecha de la ventana se marca con color azul el HTML que le da estructura, así podemos conocer sus atributos.



Al revisar notamos que no es un elemento <button> sino un enlace <a> con el atributo *href* apuntando al id de la ventana modal (esto lo hicimos cuando creamos el JSON). Ahora sabemos también que la clase CSS para darle estilos se llama “bt”.

- Para añadir el atributo *target y data-toggle* que son los que permiten la apertura de la ventana modal, tenemos que abrir el archivo “jquery.eventCalendar.js” y buscar las siguientes líneas de código para incluir los atributos.

```
    // create a button to go to event url
    boton_mas_detalles = '<a href="#portfolioModal1" target="_self" class="bt" data-toggle="modal">'+eventsOpts.locales.txt_GoToEventUrl+'</a>';
}
```

Ya con estos valores volvemos a “calendario.js” y colocamos el código restante:

```
$(document).on("click","a.bt",function(){                    1
        enlace = $(this);                                    2
        enlace = enlace.parent();
        descripcion_evento = enlace.text();
        descripcion_evento = descripcion_evento.replace("Detalles del evento","");
        descripcion_evento = descripcion_evento.split(" ");
        //buscar el ultimo elemento de la cadena(id del evento)
        longitud = descripcion_evento.length;
        id_evento = descripcion_evento[longitud-1];

        $.ajax({                                             3
            type: 'POST',
            url: base_url+'/eventos/actividades',
            data:{'id_evento':id_evento},
            })
            .done(function(resultado){
            $('#modal').html(resultado)
            })
            .fail(function(){
            alert('Hubo un error')
            })
    });
});
```

Figura 3.42 Contenido de “calendario.js”. Se divide en varias partes, aquí se muestran los últimos tres bloques.

Para explicarlo mejor lo que se encuentra en la **Figura 3.42** mostraremos la imagen con el calendario y algún evento seleccionado:

Figura 3.43 Contenido de “calendario.js”. Se divide en varias partes, aquí se muestran los últimos tres bloques.

En el bloque 1 utilizamos el evento *on()* sobre el documento, esto se debe a que el calendario se agregó al DOM después de que la página fue cargada, por lo tanto, nos sirve para trabajar con elementos cargados dinámicamente.

Por parámetro pasamos:

- Evento. En este caso es “click”.
- Elemento (a.bt) es decir la etiqueta <a> con la clase “bt”.

Una vez que se da clic sobre el botón se ejecuta lo que está en el bloque 2

- En la línea 10 creamos la variable enlace y se le asigna el elemento seleccionado con $(this).
- En la línea 11 seleccionamos el elemento padre del botón.

Si visualizamos esto con las herramientas de desarrollador notaremos al elemento padre:

```
<div class="eventCalendar-lalista"> == $0
  <p class="eventCalendar-eventDesc textoeventCalendar-hidden">
    "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum urna tortor,
    convallis id ante vitae, gravida pulvinar nibh. Curabitur luctus "
    <span class="d-none">1</span>
  </p>
  <a href="#portfolioModal1" target="_self" class="bt" data-toggle="modal">Detalles
  del evento</a>
</div>
```

Figura 3.43 El con la clase “”eventCalendar-lalista” es el elemento padre de la etiqueta <p> y <a>.

- <p> y <a> son hijos del elemento
- es hijo del elemento <p>

Continuando con la **Figura 3.42** con el bloque 2 tenemos que:

- En la línea 12 se obtiene el texto. Si mostráramos la variable “enlace” con una alerta tendríamos:

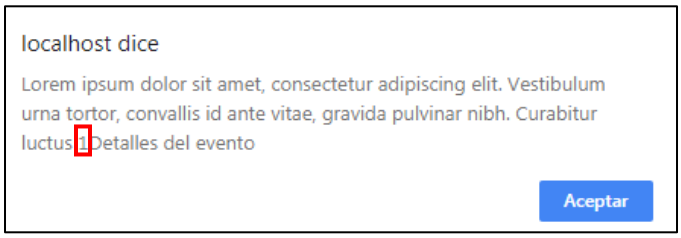


Figura 3.44 Contenido de la variable “enlace”.

- No nos interesa tener la descripción del evento y tampoco el texto del botón, solo el número que se encuentra entre estos que corresponde al id.
- Para eliminar el texto del botón utilizamos la función *replace()* e ingresamos por parámetro el texto que deseamos remplazar y como segundo parámetro el texto que lo remplaza. Si mostramos la variable “descripcion_evento” con una alerta tendríamos:

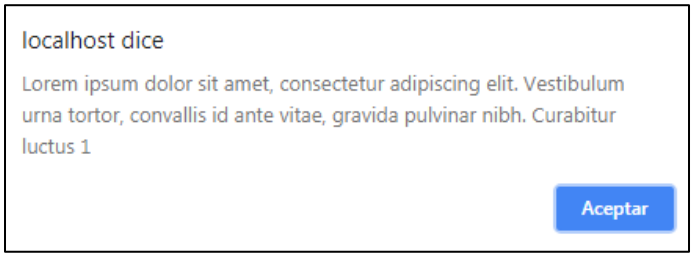


Figura 3.45 Contenido de la variable “descripción_evento”.

- En la línea 14 aplicamos el método *split()* para separar la variable “descripcion_evento*”* en un arreglo de palabras. Cada palabra se forma cuando el método encuentra un espacio en blanco.

Tenemos un arreglo de longitud n pero el único que nos interesa es el último elemento que contiene el id del evento.

- En la línea 16 creamos la variable longitud y le asignamos el tamaño del arreglo.
- En la línea 17 se asigna a la variable “id_evento” el valor del último elemento del arreglo “descripcion_evento”.

Si mostramos la variable “id_evento” con una alerta tendríamos:

Figura 3.45 Contenido de la variable “id_evento”.

Ya con el id del evento se ejecuta el bloque 3 que no es más que una petición AJAX mandando la clave del evento por POST a la URL *http://localhost/CTA/eventos/actividades*, donde el controlador es “eventosController” y la acción es *actividadesAction()*.

- Cuando la petición se realiza correctamente (*.done*) se ejecuta la función que agrega al elemento con id = “modal” el resultado de la petición mediante el método *html()*.
- Si la petición falla se manda una alerta notificando el error.

Antes de ver el código de *actividadesAction()* tenemos que entender lo que contiene la ventana modal:

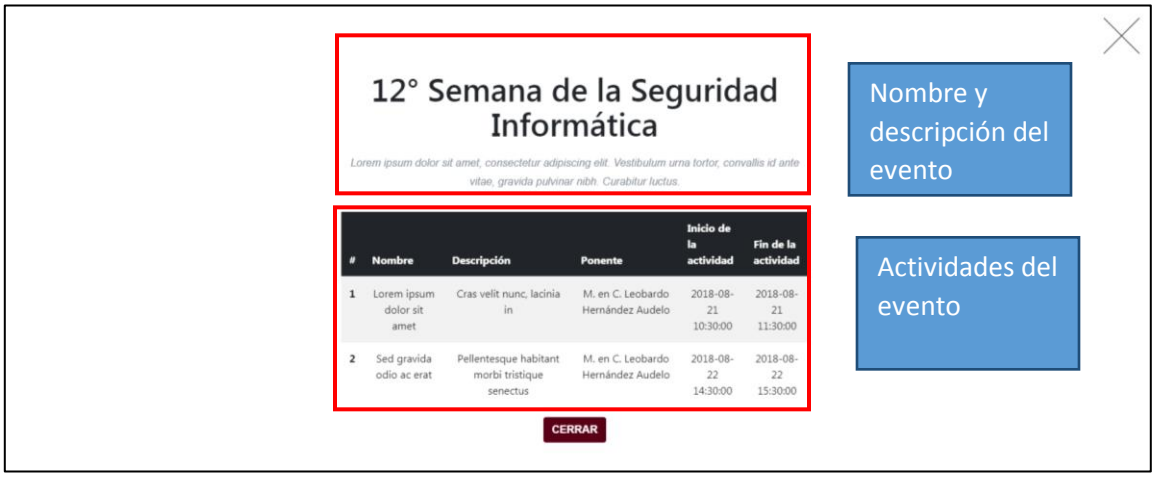
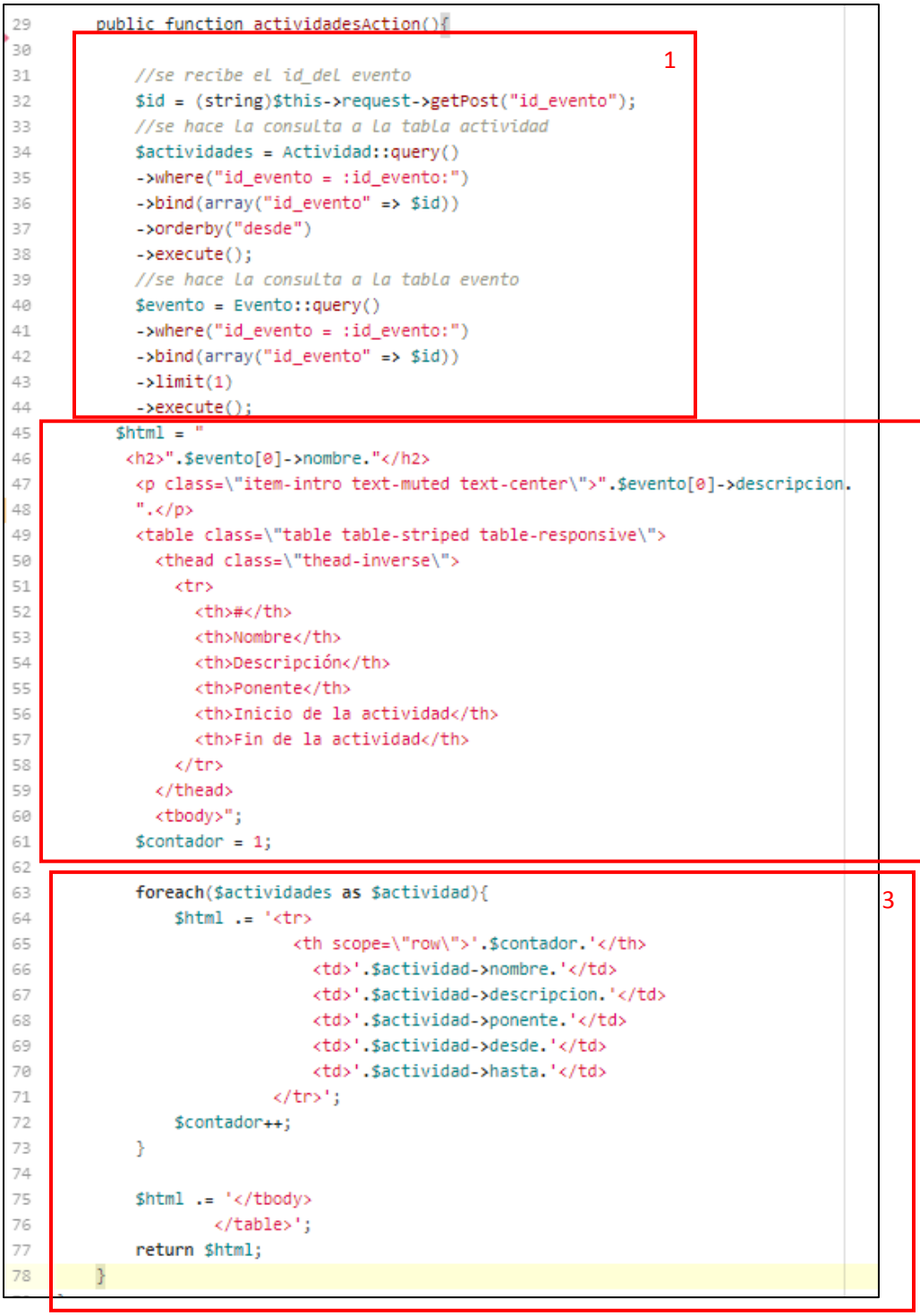


Figura 3.46 Contenido que compone la ventana modal.

Para la acción trabajaremos con los modelos “Evento” y “Actividad” y los imprimiremos en el cuerpo de la modal concatenando también la tabla, todo el código se almacena en una variable y después se retorna a la petición AJAX.

Ahora sí, el código en *actividadesAction()* se muestra en la **Figura 3.47**.




```
29      public function actividadesAction(){
30
31          //se recibe el id_del evento
32          $id = (string)$this->request->getPost("id_evento");
33          //se hace la consulta a la tabla actividad
34          $actividades = Actividad::query()
35          ->where("id_evento = :id_evento:")
36          ->bind(array("id_evento" => $id))
37          ->orderby("desde")
38          ->execute();
39          //se hace la consulta a la tabla evento
40          $evento = Evento::query()
41          ->where("id_evento = :id_evento:")
42          ->bind(array("id_evento" => $id))
43          ->limit(1)
44          ->execute();
45         $html = "
46          <h2>".$evento[0]->nombre."</h2>
47           <p class=\"item-intro text-muted text-center\">".$evento[0]->descripcion.
48           ".</p>
49           <table class=\"table table-striped table-responsive\">
50             <thead class=\"thead-inverse\">
51               <tr>
52                 <th>#</th>
53                 <th>Nombre</th>
54                 <th>Descripción</th>
55                 <th>Ponente</th>
56                 <th>Inicio de la actividad</th>
57                 <th>Fin de la actividad</th>
58               </tr>
59             </thead>
60             <tbody>";
61          $contador = 1;
62
63          foreach($actividades as $actividad){
64              $html .= '<tr>
65                          <th scope=\"row\">'.$contador.'</th>
66                            <td>'.$actividad->nombre.'</td>
67                            <td>'.$actividad->descripcion.'</td>
68                            <td>'.$actividad->ponente.'</td>
69                            <td>'.$actividad->desde.'</td>
70                            <td>'.$actividad->hasta.'</td>
71                        </tr>';
72              $contador++;
73          }
74
75          $html .= '</tbody>
76                  </table>';
77          return $html;
78      }
```

Figura 3.47 Código de la acción “actividades”.

En el bloque 1 recibimos el id del evento pasado por POST y después realizamos la consulta al modelo “Actividad” pasando como paramento el id del evento, el resultado de esta consulta se almacena en la variable $actividad.

Para poder darle un título a nuestra ventana modal también tenemos que hacer una consulta al modelo “Evento” para traer el nombre y la descripción, la consulta recibe el id del evento como parámetro (líneas 40-44).

- Esta forma de consultar datos es diferente, pero similar a un SELECT * con la cláusula *WHERE.*
- Puede ser sustituida por una consulta más parecida a SQL como se estudiaron en el capítulo 1.

En el bloque 2 primero se crea la variable $html que se encarga de formar el cuerpo de la ventana, concatenando el HTML con los datos que vienen del arreglo $actividad y $evento de esta manera formamos la tabla.

- La variable contadora solo nos sirve para enumerar las actividades.
- La consulta hecha al modelo Evento solo retorna un resultado ya que no puede haber más de dos eventos con el mismo id.

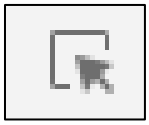
El bloque 3 recorre el arreglo con las actividades ya que un evento puede tener más de una actividad.

- Al final del ciclo *foreach* cerramos el cuerpo de la tabla y la tabla con </tbody> y </table> respectivamente.
- Por último retornamos $html.

3.4.5 Modificando los estilos originales del plugin

Una vez que tenemos el calendario funcionando es hora de modificar los estilos que tiene por defecto el *plugin*, para esto hacemos uso de las *herramientas para desarrollador* (F12 en Chrome) del navegador. Al seleccionar el elemento que deseamos modificar podemos conocer sus estilos que contiene y hasta en que archivo se encuentra, para que así desde nuestro editor de texto vayamos a la línea con la regla de estilo y modificarla.

Recuerda que para seleccionar un elemento primero debemos seleccionar la herramienta ubicada en la parte superior izquierda:



En el siguiente ejemplo, si seleccionamos en el calendario el día actual (con fondo color verde) veremos en la parte derecha de la ventana los estilos CSS que contiene, navegando un poco encontraremos el color (con la regla *background*).

Los estilos mostrados por las *herramientas para el desarrollador* podemos encontrar el color de fondo del elemento y el color de texto. inclusive podemos modificarla.

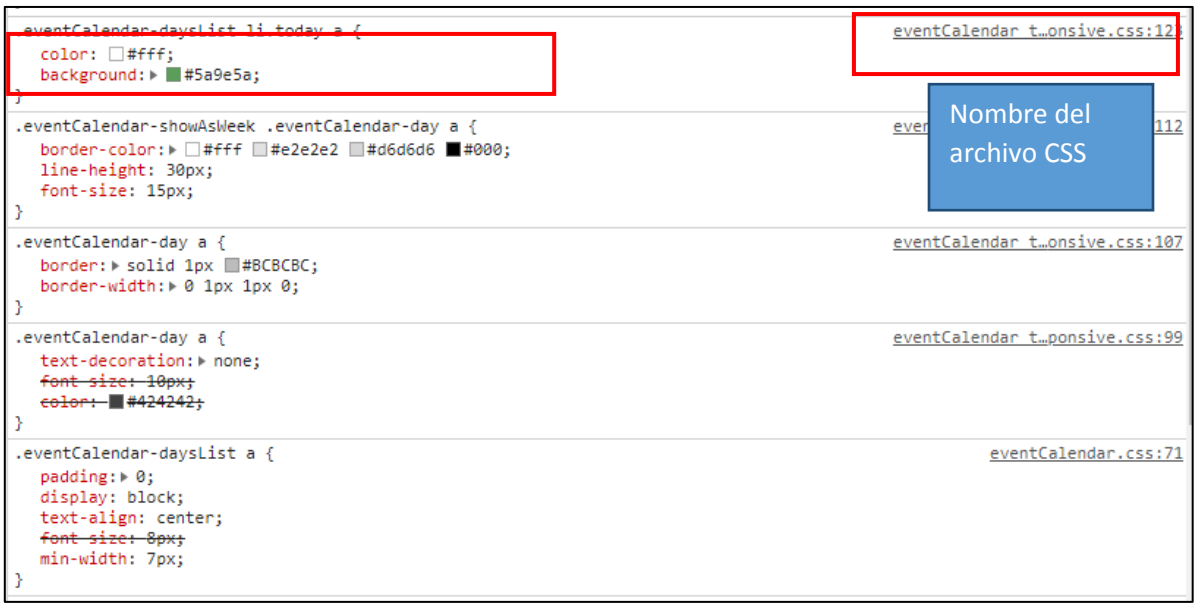


Figura 3.48 en la parte izquierda encontramos las propiedades y clases, del lado derecho encontramos el nombre del archivo al que pertenecen.

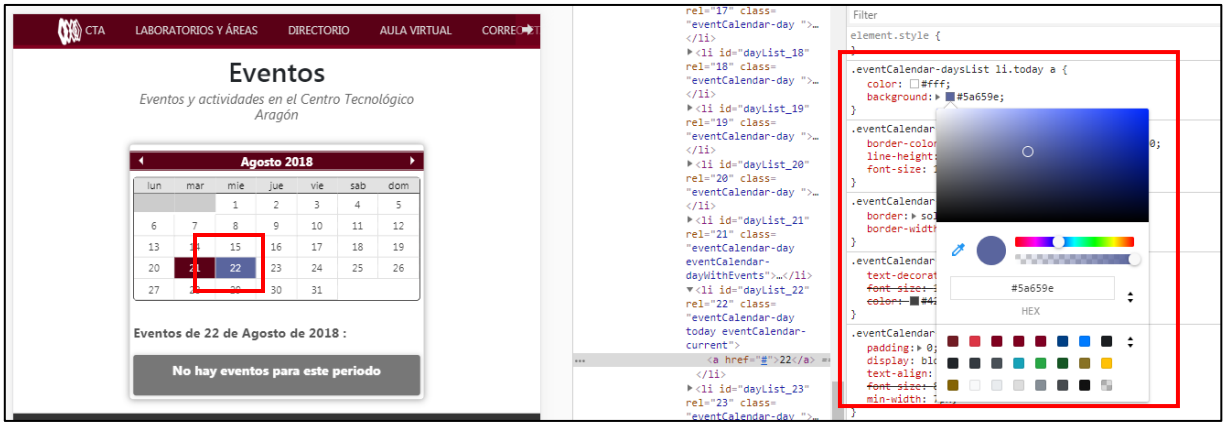


Figura 3.49 Podemos editar el color de la propiedad y saber su equivalente a hexadecimal.

- Esto podemos hacer para la mayoría de los elementos

Aunque en el navegador podemos cambiar de color, al recargar la página el color regresa al valor original ya que tiene que modificarse desde el archivo. Lo que se hace en este caso es copiar el nombre de la clase del elemento que modificaremos e ir al archivo CSS correspondiente y buscarlo para cambiar las reglas de estilo.

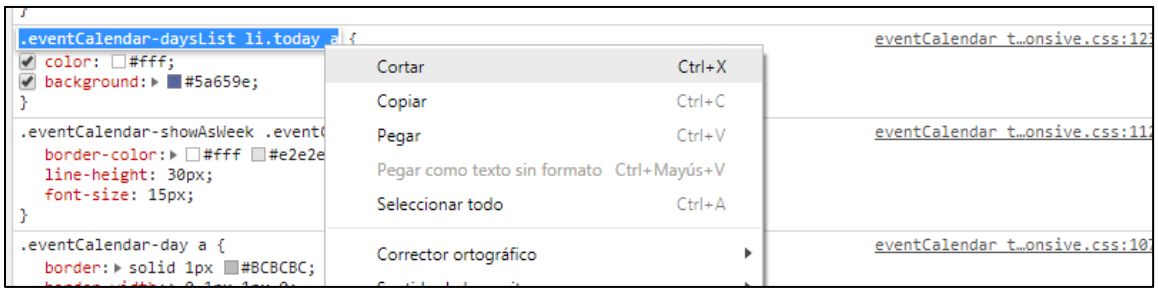


Figura 3.50 Podemos copiar el nombre de la clase o propiedad dando doble *clic*.

En la mayoría de editores de texto (Atom, Sublime Text y Brackets) contienen un buscador de palabras. En este caso usaremos Atom.

- Para acceder a su buscador se presionan las teclas *Ctrl + f*

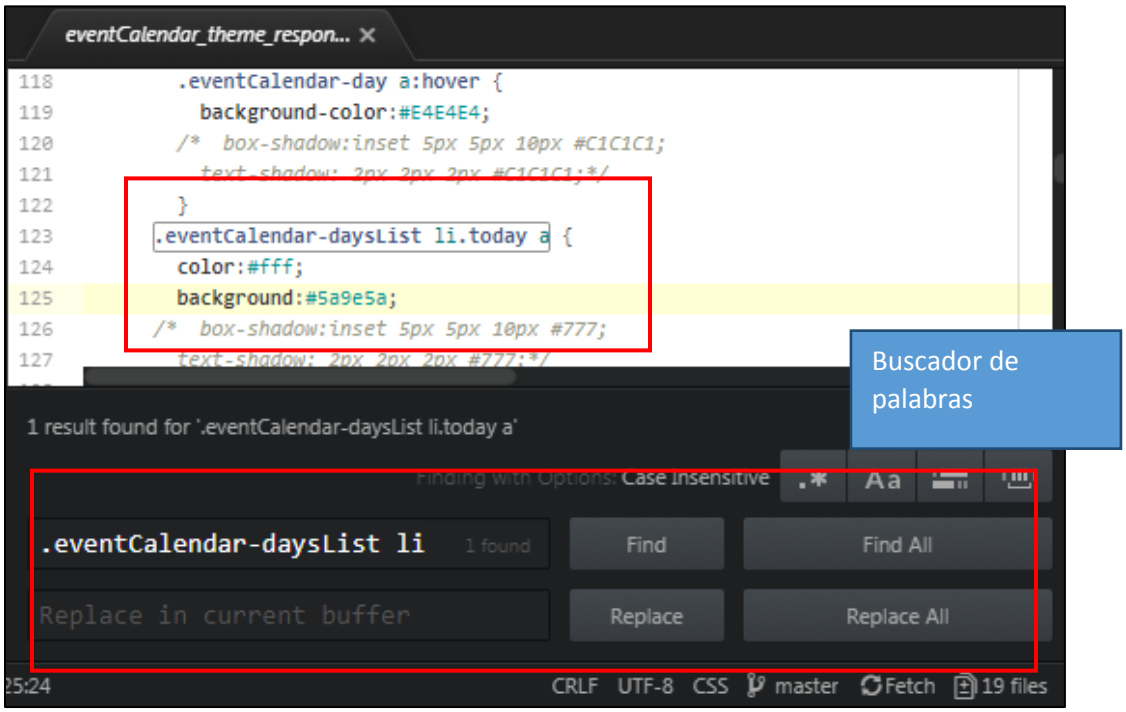


Figura 3.51 Dentro del buscador ingresamos el nombre de la clase para encontrar la propiedad a editar.

3.5 Resumen

Durante el capítulo se hizo uso de las herramientas vistas en los capítulos 1 y 2 para crear los módulos que componen el sitio web de Centro Tecnológico Aragón.

Para la sección “galería” se definió y explicó lo siguiente:

- Un *plugin* en es una herramienta desarrollada por terceros que resuelven una necesidad en específico.
- jQuery Simple Slider es un carrusel de imágenes o *slider* que contiene botones para avanzar o adelantar lo que se muestra.
- La imagen y la descripción son cargadas desde un archivo JSON.
- Un archivo JSON (*JavaScript Object Notation*) es un formato de texto ligero para el intercambio de datos en la red. Popular en la mayoría de los lenguajes de programación.
- Se mostró a detalle como compone la vista y el código jQuery para insertar las etiquetas <img> de forma dinámica.

En el módulo “directorio” establecimos conexión con la tabla “empleado” en base de datos MySQL, esto con la finalidad de hacer consultas y traer los contactos que el usuario ingresa en el buscador.

- Se mostró el código que compone la vista y se definieron sus estilos CSS.
- Pudimos crear un modelo y el controlador utilizando las *herramientas de desarrollo* de **Phalcon.**
- Explicamos como funcionan botones de radio y el buscador, mostrando del código js y jQuery.
- Creamos *if anidados* en JavaScript para filtrar las consultas.
- Cada petición realizada con AJAX se hacía con el mismo controlador, pero acción diferente.
- Las consultas deferían solo en lo que se deseaba buscar, pero en la mayoría se hizo uso de las cláusulas *WHERE, LIKE y ORDER BY*.
- Vimos que se puede incluir código HTML dentro de una variable en PHP.
- Aprendimos a insertar código HTML dinámicamente con el método *.html()*.

Por último, el módulo “Eventos”, se hizo uso del *plugin* “*jQuery Event Calendar*”:

- Aprendimos a descargarlo e importarlo a nuestro proyecto.
- Utilizamos *jQuery Migrate Plugin* para solucionar problemas de compatibilidad.
- Explicamos el código que contiene la vista.
- Creamos un controlador y generamos el JSON que contiene los eventos que fueron cargados desde la tabla “evento” en la base de datos.
- Dimos función al botón “Detalles del evento” para que desplegara una ventana modal con las actividades asociadas al evento seleccionado.
- Para consultar las actividades de un evento, primero accedimos a su id utilizando jQuery y JavaScript.

Conclusiones

En este trabajo se presentó el uso del *framework* **Phalcon** como parte del *back-end* y las técnicas que se utilizan para dar formato y estilo al *front-end* todo con la finalidad de poder crear proyectos web sencillos o robustos.

Phalcon es sin duda un gran apoyo para comenzar un proyecto porque proporciona muchos servicios que como programadores podemos llegar a utilizar. Como vimos, existen muchos *frameworks* que son populares e inclusive cuentan con más documentación. Pero lo que lo hace realmente potente al resto de los demás es la velocidad con la trabaja, debido a que este hecho como una extensión del lenguaje de programación C.

Siempre es bueno tener alternativas de uso de los *frameworks*, cuando empecé el capítulo 1 definimos el patrón de arquitectura Modelo-Vista-Controlador, ya que es un patrón de diseño muy popular, así que si entendemos el uso del MVC podemos adaptarlo a cualquier *framework* e inclusive cualquier lenguaje.

Como trabajo derivado de mi participación del servicio social me ayudo bastante a enriquecer mis conocimientos en la programación en PHP, lenguaje que considero con una complejidad intermedia y una buena demanda laboral. Al ser un lenguaje enfocado al desarrollo, es completamente necesario conocer CSS, JavaScript y *frameworks* como Boostrap y Jquery. Que a pesar de los años sigue siendo utilizados por las ventajas que ofrece.

Cuando comencé a desarrollar los módulos que eran requeridos para el sitio, desconocía el funcionamiento de **Phalcon**, pero poco a poco descubrí las ventajas con las que cuenta. Pese a no ser un framework que tenga demasiada información en la web o en libros, Con la documentación de su página oficial es más que suficiente ya que cuenta con ejemplos y explicación hasta en diferentes idiomas. Entender **Phalcon** cuando se es nuevo en el uso de *frameworks* resulta difícil, pero no imposible.

El sitio del Centro Tecnológico Aragón combinó lo que aprendimos en el capítulo 1 y 2 entendimos cómo funciona un *plugin* y como poder integrarlo al sitio, para después cambiar su estilo a las necesidades del cliente, en este caso el Laboratorio de Cómputo. Vimos que es un JSON y como utilizar este tipo de archivos en la programación.

En el desarrollo dedique horas de trabajo que valieron la pena, gracias a la ayuda que te ofrece el Laboratorio de Cómputo uno puede aprender más de lo que se ve en clase y te ayuda a prepararte para la vida laboral.

Bibliografía

Rada, C. (2015). Learning Phalcon PHP. Livery Place, Reino Unido: Packt Publishing Ltd.

Kane, C. (2017). Mastering Phalcon Php. California, Estados Unidos: CreateSpace Independent Publishing Platform.

Iakovlev, S., & Schissler, D. (2016). Phalcon Cookbook. Birmingham, Reino Unido: Packt Publishing Limited.

Castillo, A. A. (2017). Curso de Programación Web: JavaScript, Ajax y jQuery (2ª ed.). Plaza América, Vigo, España: IT Campus Academy.

Popel, D. (2007). Learning PHP Data Objects. Birmingham, Reino Unido: Packt Publishing Ltd.

Durango, A. (2015). Diseño Web con CSS (2ª ed.). Plaza América, Vigo, España: IT Campus Academy.

Lengstorf, J. (2010). Pro PHP and jQuery. Nueva York, Estados Unidos: Apress.

Schulz, R. G. (2008). Diseño web con CSS. Cataluña, España: Marcombo.

Lett, J. (2018). Bootstrap 4 Quick Start: Responsive Web Design and Development Basics for Beginners. Sterling Heights , Michigan, Estados Unidos: Bootstrap Creative.

Miller, S. A. (2014a). Getting Started with Phalcon. Recuperado de https://books.google.com.mx/books?id=gxqxAgAAQBAJ&dq=phalcon&hl=es&source=gbs_navlinks_s

Selección de framework PHP para el desarrollo de un sistema de administración de contenido básico. (2017, 8 septiembre). Recuperado 10 abril, 2018, de *https://dev.peacefulrain.net/*

Bustos, G. (2018, 15 febrero). Tutorial completo de Phalcon para principiantes. Recuperado 11 abril, 2018, de *https://www.hostinger.mx/tutoriales/tutorial-phalcon-para-principiantes/*

Herramientas para desarrolladores de Phalcon. (s.f.). Recuperado 11 abril, 2018, de https://docs.phalconphp.com/es/3.3/devtools-usage

¿Qué es el módulo mod_rewrite? (s.f.). Recuperado 11 abril, 2018, de *https://www.hostinet.com/formacion/hosting-alojamiento/que-es-el-modulo-mod_rewrite-y-para-que-sirve/*

Parra, I. (2013, 22 diciembre). Volt, el motor de plantillas de Phalcon. Recuperado 15 abril, 2018, de *https://www.uno-de-piera.com/volt-el-motor-de-plantillas-de-phalcon/*

GitBook. (s.f.). Enrutamiento básico. Recuperado 20 abril, 2018, de *https://richos.gitbooks.io/laravel-5/content/capitulos/chapter9.html*

Lenguaje de consulta de Phalcon (PHQL). (s.f.). Recuperado 21 abril, 2018, de *https://docs.phalconphp.com/es/3.2/db-phql*

Phalcon - Scaffolding Application. (s.f.). Recuperado 23 abril, 2018, de *https://www.tutorialspoint.com/phalcon/phalcon_scaffolding_application.htm*

Parra, I. (2013, 22 diciembre). Rutas en Phalcon, personaliza tus urls. Recuperado 23 abril, 2018, de *https://www.uno-de-piera.com/rutas-en-phalcon-personaliza-tus-urls/*

Enrutamiento. (s.f.). Recuperado 25 abril, 2018, de *https://docs.phalconphp.com/es/3.2/routing*

DefiniciónABC. (s.f.). Definición de HTTP. Recuperado 26 abril, 2018, de *https://www.definicionabc.com/tecnologia/http.php*

El protocolo HTTP. (2018, mayo). Recuperado 26 abril, 2018, de *https://es.ccm.net/contents/264-el-protocolo-http*

El protocolo HTTP. (2018, mayo). Recuperado 26 abril, 2018, de *https://es.ccm.net/contents/264-el-protocolo-http*

Aprenderaprogramar.com. (s.f.). get y post HTML: method. Formas de envío de datos en formulario. Diferencias y ventajas. Recuperado 27 abril, 2018, de *https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=527:get-y-post-html-method-formas-de-envio-de-datos-en-formulario-diferencias-y-ventajas-ejemplos-cu00721b&catid=69&Itemid=192*

Mclibre. (2017, 10 octubre). Qué es una hoja de estilo. Recuperado 2 mayo, 2018, de *http://www.mclibre.org/consultar/amaya/css/css-que-es.html*

Mclibre. (2018, 5 enero). Lista de propiedades CSS estables. Recuperado 3 mayo, 2018, de *http://www.mclibre.org/consultar/amaya/css/css-propiedades.html#css-bordes*

Pedraza, A. (2014, 24 noviembre). Div con imagen de fondo responsive CSS3. Recuperado 3 mayo, 2018, de *https://desarrollofrontend.com/div-con-imagen-de-fondo-responsive/*

W3schools. (s.f.-b). CSS Colors. Recuperado 5 mayo, 2018, de *https://www.w3schools.com/CSSref/css_colors.asp*

Tutorialesprogramacionya. (s.f.). Posicionamiento relativo (position: relative). Recuperado 10 mayo, 2018, de *https://www.tutorialesprogramacionya.com/cssya/temarios/descripcion.php?inicio=30&cod=37&punto=36*

Mclibre. (2017, 24 septiembre). CSS 2: Posicionamiento flotante. Recuperado 10 mayo, 2018, de *http://www.mclibre.org/consultar/amaya/css/css-posicionamiento-flotante.html*

Puntoabierto. (s.f.). Qué es Bootstrap y cuáles son sus ventajas. Recuperado 11 mayo, 2018, de *https://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas*

ARWEB. (2014, 26 septiembre). ¿Qué es Bootstrap y cómo funciona en el diseño web? Recuperado 13 mayo, 2018, de *https://www.arweb.com/chucherias/%C2%BFque-es-bootstrap-y-como-funciona-en-el-diseno-web/*

Sanso, R. (2013, 12 octubre). 8 herramientas para maquetar webs de forma sencilla. Recuperado 14 mayo, 2018, de *http://www.rafelsanso.com/8-herramientas-para-maquetar-webs-de-forma-sencilla/*

Corchado, B. (s.f.). CDN: ¿qué es y qué ventajas aporta? Recuperado 15 mayo, 2018, de *https://www.hosteurope.es/blog/cdn-que-es-y-que-ventajas-aporta*

Bootstrap 4 sistema de rejilla. (s.f.). Recuperado 15 mayo, 2018, de *https://www.apuntes-web.es/sistema-de-rejilla-bootstrap-4/*

Quackit. (s.f.). Bootstrap 4 Containers. Recuperado 17 mayo, 2018, de *https://www.quackit.com/bootstrap/bootstrap_4/tutorial/bootstrap_contain.cmf*

Robledano, A. (2015, 3 junio). Tutorial de Bootstrap 3: El texto. Recuperado 19 mayo, 2018, de *https://openwebinars.net/blog/tutorial-de-bootstrap-3-el-texto/*

Tablas HTML. (s.f.). Recuperado 21 mayo, 2018, de *https://www.hazunaweb.com/curso-de-html/tablas/*

TutorialHTML. (s.f.). Tablas HTML. Recuperado 23 mayo, 2018, de *http://www.tutorialhtml.net/manualHTML/tablas-cabeceras.php*

Referencia HTML. (s.f.). Elemento textarea. Recuperado 25 mayo, 2018, de *http://www.htmlquick.com/es/reference/tags/textarea.html*

Duarte, E. (2013, 16 marzo). jQuery: Qué es, Orígenes, Ventajas y Desventajas. Recuperado 26 mayo, 2018, de *http://blog.capacityacademy.com/2013/03/16/jquery-que-es-origenes-ventajas-desventajas/*

NorfiPC. (s.f.). JQuery, todos los selectores, eventos, métodos y funciones. Recuperado 26 mayo, 2018, de *https://norfipc.com/codigos/jquery-todos-selectores-eventos-metodos-funciones.php*

Gómez, J. M. (2018, 7 enero). ¿Cómo acortar URL?, los 3 acortadores de url mas útiles. Recuperado 28 mayo, 2018*,* de *https://laculturadelmarketing.com/como-acortar-url/*

Ejemplos para asociar eventos a elementos creados dinámicamente con jquery. (2017, 3 junio). Recuperado 28 mayo, 2018, de *http://www.videotutoriales.es/video-tutoriales-gratis-espanol-online/asociar-eventos-elementos-creados-dinamicamente-jquery/*

Méndez, M. (2012, 24 octubre). jQuery Blues: 1.DOM. Recuperado 30 mayo, 2018, de *https://www.mmfilesi.com/blog/jquery-blues-1-dom/*

Barrena, A. (2015, 7 enero). jQuery on() y off(): Asignar de eventos a elementos del DOM. Recuperado 1 junio, 2018, de *https://www.anerbarrena.com/jquery-on-off-4767/*

Salas, I. (2018, 19 junio). Control de eventos con jquery para elementos html generados dinámicamente. Recuperado 3 junio, 2018, de *https://programandoointentandolo.com/2017/06/control-eventos-jquery-elementos-dinamicos.html*

Reina Cárdenas, C. E. (s.f.). Ventajas y desventajas de Ajax y cómo usarlo con jQuery. Recuperado 4 junio, 2018, de *https://platzi.com/blog/ajax-con-jquery/*

¿Cómo usar y configurar un ajax con jQuery? (2014, 28 agosto). Recuperado 12 junio, 2018, de *https://platzi.com/blog/ajax-con-jquery/*

JQuery. (s.f.). Category: Ajax. Recuperado 15 junio, 2018, de *https://api.jquery.com/category/ajax/*

Rodríguez, A. (s.f.). Ajax: ejemplos efectos y uso en desarrollos web con HTML, CSS… Ventajas e inconvenientes. Recuperado 20 junio, 2018, de *https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=920:ajax-ejemplos-efectos-y-uso-en-desarrollos-web-con-html-css-ventajas-e-inconvenientes-cu01205f&catid=83&Itemid=212*

Librosweb. (s.f.). La librería jQuery. Recuperado 22 junio, 2018, de *https://librosweb.es/libro/ajax/capitulo_10/la_libreria_jquery.html*

Villalobos, J. (2013, 2 octubre). Introducción a JSON, sintaxis y ejemplos. Recuperado 24 junio, 2018, de *http://codigoprogramacion.com/cursos/javascript/introduccion-a-json-sintaxis-y-ejemplos.html#.W5bVF85KjIV*

W3schools. (s.f.-c). Location host Property. Recuperado 27 junio, 2018, de *http://codigoprogramacion.com/cursos/javascript/introduccion-a-json-sintaxis-y-ejemplos.html#.W5bVF85KjIV*