



# Descubre CodeIgniter y todas sus ventajas

PHP



Qualentum Lab

# Descubre CodeIgniter y todas sus ventajas

En el mundo del desarrollo de aplicaciones web, las bases de datos son fundamentales para almacenar y gestionar información de manera eficaz. En este contexto, CodeIgniter, un framework en PHP muy versátil, destaca por su capacidad para facilitar la **interacción entre aplicaciones web y las bases de datos**.

¡Empezamos!

*Autor: Jesús Donoso*

☰ Introducción

☰ Creación de proyectos

☰ Estructura de proyectos

☰ Controladores

☰ Modelos

☰ Conexión a BBDD

☰ Enrutamiento

☰ Vistas

☰ Validación de formularios

☰ Seguridad

☰ Conclusiones y recursos de interés

# Introducción



Qualentum Lab

CodeIgniter es un framework de desarrollo de aplicaciones web de código abierto que ha ganado importancia en el mundo del desarrollo web debido a sus características y ventajas notables.

CodeIgniter es una herramienta que simplifica y agiliza la creación de aplicaciones web en PHP.

Aunque su desarrollo original estuvo a cargo de la compañía EllisLab y posteriormente fue adquirido por el British Columbia Institute of Technology (BCIT), la comunidad de desarrolladores ha continuado su desarrollo y evolución.

## Sencillez y facilidad de uso

Una de las razones de su importancia radica en su enfoque, así como en la sencillez y facilidad de uso. Y es que CodeIgniter destaca por su estructura ligera y su simplicidad, convirtiéndola en una valiosísima opción para aquellos desarrolladores que desean crear aplicaciones web de forma eficiente y sin enfrentarse a una curva de aprendizaje muy pronunciada.

## Rendimiento

El rendimiento es otro factor muy valorable de CodeIgniter, puesto que está diseñado para ofrecer un rendimiento óptimo, con componentes eficientes que garantizan tiempos de respuesta rápidos en las aplicaciones web.

## Documentación detallada

La documentación detallada de CodeIgniter es otro punto positivo. Completa y fácil de entender, se ha convertido en una magnífica guía para que los desarrolladores aprendan su manejo en poco tiempo y a resolver cualquier problema con el que se tropiecen de manera eficaz.

## Flexibilidad

La flexibilidad es otra de sus características distintivas, ya que nos permite a los desarrolladores utilizar nuestras propias convenciones y configuraciones en lugar de imponernos una estructura rígida, lo que resulta muy beneficioso en proyectos personalizados.

## Seguridad

Como ya sabemos, la seguridad es un aspecto crítico en el desarrollo web. Pues bien, CodeIgniter aborda este tema con características integradas de seguridad, como la protección contra ataques [CSRF](#) y medidas para prevenir la inyección SQL.

---

**A pesar de no ser tan popular como otros frameworks, por ejemplo, Laravel o Django, CodeIgniter cuenta con una comunidad activa de desarrolladores que brindan soporte, crean extensiones y mantienen el framework actualizado.**

---

Por último, hay que decir que, a pesar de los cambios en la propiedad y de la dirección del proyecto a lo largo de los años, CodeIgniter ha continuado recibiendo **actualizaciones y mejoras**, por lo tanto, todo apunta a su viabilidad a largo plazo.

# Creación de proyectos



La creación de un proyecto en CodeIgniter implica varios pasos que se deben seguir para establecer una base sólida a la hora de desarrollar una aplicación web. Por lo tanto, vamos a describir el proceso para crear un proyecto tanto de forma manual como a través de Composer.

## Creación de forma manual

En primer lugar, debemos **descargar la última versión** de CodeIgniter desde el sitio web oficial de [CodeIgniter](#). Es importante asegurarse de obtener la versión más reciente para aprovechar las últimas características y las correcciones de errores.

Una vez descargado, el archivo ZIP de CodeIgniter debe **descomprimirse** en el directorio de trabajo deseado, en el servidor local o en el servidor de producción. Si se prefiere, se puede renombrar la carpeta principal que contiene los archivos de CodeIgniter. Esto puede hacerse para reflejar el nombre del proyecto o la aplicación que vamos a desarrollar.

## Creación a través de Composer

Ahora sí, ahora estudiaremos cómo crear un proyecto CodeIgniter utilizando Composer, veremos que es muy sencillo.

En primer lugar, nos tenemos que asegurar de que tenemos instalado Composer en el sistema. Si no es así, podemos hacerlo desde su [página oficial](#). A continuación, deberemos abrir un terminal, navegar al directorio deseado para crear el proyecto y ejecutar el siguiente comando para crear ese proyecto nuevo.

```
composer create-project codeigniter4/appstarter nombre_proyecto
```

---

**Recuerda: este comando descarga la última versión de CodeIgniter 4 y configura la estructura básica del proyecto en el directorio *nombre\_proyecto*.**

Como ves, crear un proyecto CodeIgniter utilizando Composer es muy simple. Pero además este método tiene una **ventaja**: simplifica la gestión de dependencias y actualizaciones, ya que Composer se encarga de la instalación y actualización de los componentes necesarios para el proyecto.

## Configuración de la base de datos

Ya hemos creado un proyecto de forma manual o a través de Composer, pues bien, ahora toca algo esencial, la configuración de la base de datos.

La creación de una base de datos se debe realizar en el servidor de bases de datos y luego tendremos que configurar los detalles de la conexión en el archivo "database.php", ubicado en la carpeta "*application /config*" de CodeIgniter. Esto incluye detalles como el tipo de base de datos, el nombre de usuario, la contraseña y el nombre de la base de datos.

Veamos ahora un ejemplo de la configuración del archivo.

```
$db['default'] = array(
    'dsn'      => '',
    'hostname' => 'localhost',
    'username' => 'tu_usuario',
    'password' => 'tu_contraseña',
    'database' => 'nombre_de_tu_base_de_datos',
    'dbdriver'  => 'mysqli',
    'dbprefix'  => '',
    'pconnect'  => FALSE,
    'db_debug'  => (ENVIRONMENT !== 'production'),
    'cache_on'  => FALSE,
    'cachedir'  => '',
    'char_set'  => 'utf8',
    'dbcollat'  => 'utf8_general_ci',
    'swap_pre'  => '',
    'encrypt'   => FALSE,
    'compress'  => FALSE,
    'stricton'  => FALSE,
    'failover'  => array(),
    'save_queries' => TRUE
);
```

En el archivo *"config.php"*, ubicado en la misma carpeta *"application/config"*, debemos configurar la URL base de la aplicación. ¿Por qué? Por un motivo fundamental: así nos aseguramos de que **Codelgniter genere las URL correctas en la aplicación**.

```
$config['base_url'] = 'http://localhost/mi_proyecto/';
```

Pues bien, una vez realizada la configuración de la base de datos, podemos decir que el proyecto Codelgniter está preparado para el inicio del desarrollo.

# Estructura de proyectos



La estructura de directorios en un proyecto de CodeIgniter sigue una organización estándar que ayuda a mantener el código y los recursos de la aplicación organizados y accesibles.

A continuación, pasaremos a describir algunos de los directorios o archivos más importantes en un proyecto con CodeIgniter:

## application

En este directorio principal de la aplicación se encuentran subdirectorios importantes como *controllers*, *models*, *views*, *config* y *helpers*, para las bibliotecas de funciones útiles, y *libraries* para las bibliotecas personalizadas. En cuanto al directorio *ThirdParty*, este se utiliza para agregar bibliotecas de terceros.

**public** —

En este directorio se almacenan los recursos públicos que se sirven directamente al navegador, como archivos CSS, JavaScript e imágenes. Es común configurar un archivo `.htaccess` en este directorio para mejorar la seguridad y la gestión de URL.

**writable** —

En esta carpeta se almacenan archivos generados dinámicamente, como los registros de errores, los archivos de sesión y las cachés. El servidor web debe tener permisos de escritura en este directorio.

**system** —

Contiene el núcleo de CodeIgniter y no debe modificarse directamente. Incluye las clases y bibliotecas esenciales para el funcionamiento del framework.

**tests** —

Si se realizan pruebas unitarias o de integración, este directorio puede utilizarse para almacenar dichas pruebas y los archivos relacionados con estas. Es una buena práctica tener pruebas automatizadas para garantizar la calidad del código.

**vendor** —

Este directorio guarda las dependencias de *Composer*, un administrador de paquetes para PHP. Si la aplicación utiliza paquetes o bibliotecas de terceros, se almacenan también aquí.

**appstarter** —

Este directorio solo se incluye en versiones anteriores a la 4.0. de CodeIgniter. Contiene un instalador para crear la estructura inicial de directorios de una aplicación.

**.env** —

Este archivo permite configurar variables de entorno para diferentes tipos como de desarrollo, producción y pruebas.

Es importante destacar que, aunque esta estructura de directorios es la típica en CodeIgniter, se pueden realizar modificaciones según las necesidades del proyecto. Recuerda que este framework que estamos estudiando es reconocido por su **flexibilidad**, en cuanto a la organización de archivos y carpetas, lo que permite adaptar la estructura a las preferencias específicas de desarrollo.

# Controladores



Qualentum Lab

---

Los controladores en CodeIgniter son una parte clave de la **arquitectura MVC** (*Model-View-Controller*) que se utiliza en el desarrollo de aplicaciones web.

---

Estos controladores son los componentes encargados de recibir las solicitudes de los navegadores web, de procesarlas y determinar qué respuesta o vista se debe mostrar a los usuarios.

¿Y cómo funcionan estos controladores? ¡Vamos a verlo!

Para crear un controlador en CodeIgniter, se debe generar un archivo PHP en el directorio "app/Controllers". El nombre del archivo debe coincidir con el nombre del controlador y seguir una convención de capitalización de estilo *CamelCase*. Por ejemplo, si se desea crear un controlador llamado "*Productos*", el archivo se denominaría "*Productos.php*". Seguro que con un ejemplo lo entendemos mejor.

```
// Archivo: app\Controllers\Productos.php
namespace App\Controllers;

class Productos extends BaseController
{
    public function index()
    {
        // Lógica del controlador
    }
}
```

---

**Apunte importante: en CodeIgniter 4, se recomienda que los controladores extiendan de la clase *BaseController*. Esta clase proporciona funcionalidades y métodos útiles que se pueden emplear en los controladores.**

Cada método dentro del controlador representa una acción que se puede llevar a cabo en una página web, por ejemplo, el método *index* en el controlador "Productos" podría mostrar una lista de productos. Además, cada método puede realizar lógica y cargar vistas. Los controladores suelen cargar vistas para presentar la interfaz de usuario. Para hacerlo se utiliza el método "*\$this->load->view()*", tal y como vemos en el siguiente fragmento de código:

```
public function index()
{
    $data = []; // Datos que se pasan a la vista
    return view('productos/index', $data);
}
```

Con frecuencia, los controladores interactúan con modelos para obtener o actualizar datos en la base de datos. Para hacerlo, se cargan modelos en el controlador y se utilizan para realizar operaciones en la base de datos.

```
$productoModel = new ProductoModel();
$productos = $productoModel->findAll();
```

Los controladores también pueden encargarse de la lógica de negocio y del flujo de la aplicación, como la validación de formularios, la autenticación de usuarios y la administración de sesiones.

---

**En definitiva, los controladores en CodeIgniter se encargan de gestionar las solicitudes web, de determinar las acciones que se deben realizar y, además, cargan vistas y controlan la lógica de la aplicación. Contribuyen significativamente a la separación de la lógica de presentación y la lógica de negocios en aplicaciones web, esto sin duda facilita el desarrollo y el mantenimiento de aplicaciones escalables.**

# Modelos



Qualentum Lab

Los modelos en CodeIgniter son componentes esenciales en el patrón de diseño **modelo-vista-controlador (MVC)** y se utilizan para interactuar con la base de datos y realizar operaciones relacionadas con los datos en una aplicación web.

Estos modelos además se encargan de la lógica de negocio y proporcionan una interfaz para acceder y manipular datos de la base de datos de manera estructurada.

¿Cómo podemos crear un modelo en CodeIgniter? El primer paso es **generar un archivo PHP en el directorio "app/Models"** de la aplicación. Los nombres de estos archivos de modelo deben coincidir con el nombre del modelo y deben extender la clase `CodeIgniter\Model`.

```
// En app/Models/ProductModel.php
namespace App\Models;

use CodeIgniter\Model;

class ProductModel extends Model
{
    protected $table = 'products';
    protected $primaryKey = 'id';
    protected $allowedFields = ['name', 'price', 'quantity'];

    // Puedes agregar funciones adicionales según sea necesario
}
```

Después, dentro del modelo, se definen **atributos** como:

- `$table` donde debe ser el valor de la tabla a la que representa el modelo.
- `$primaryKey` donde se indica el valor del ID de la tabla a la que representa.
- `$allowedFields` que representa el resto de los atributos de la tabla.

En el modelo también se definen funciones que realicen diversas operaciones en la base de datos. Estas operaciones pueden incluir **inserción, actualización, eliminación y recuperación de datos**. Las funciones van a encapsular la lógica empresarial relacionada con la manipulación de datos, por ejemplo, en un modelo de usuarios, es posible definir funciones como `insertarUsuario()`, `actualizarUsuario()`, `eliminarUsuario()`, `obtenerUsuarioPorID()`..., entre otras.

Los modelos en CodeIgniter utilizan el sistema de base de datos del propio framework para interactuar con la base de datos. Esto se logra mediante el uso del objeto de base de datos `$db`. Los modelos pueden realizar consultas a la base de datos de manera segura y estructurada. Por ejemplo, para obtener un usuario por su ID, un modelo puede contener una función como la que se puede observar a continuación:

```
public function obtenerUsuarioPorID($id) {
    return $this->db->table('usuarios')
        ->where('id', $id)
        ->get()
        ->getRow();
}
```

Los controladores utilizan los modelos para acceder y manipular datos. ¿Y cómo se cargan los modelos en los controladores? Ni más ni menos que utilizando el método `$this->load-model()`. ¡Ah!, y un dato que debemos tener en cuenta es que las funciones del modelo se llaman según sea necesario.

Explicado este proceso, vamos a ver un ejemplo de un fragmento de código para cargar un modelo en un controlador y darle uso:

```
public function mostrarUsuario($id) {  
    $this->load->model('UsuarioModel');  
    $usuario = $this->UsuarioModel->obtenerUsuarioPorID($id);  
    // Se realizan acciones con el usuario recuperado  
}
```

Por último, cabe señalar que los modelos en CodeIgniter también pueden incluir **reglas de validación** y, así, garantizar que los datos ingresados cumplen con ciertos criterios antes de ser insertados en la base de datos. Esto, sin duda, es una gran contribución a la hora de mantener la integridad de los datos y prevenir errores indeseados.

# Conexión a BBDD



En CodeIgniter, las operaciones de base de datos se realizan a través del modelo de datos integrado. Para comprender esta funcionalidad estudiaremos las operaciones básicas de la base de datos en CodeIgniter, incluyendo la conexión, la recuperación, la inserción, la actualización y la eliminación de datos.

No obstante, debemos saber antes que CodeIgniter utiliza un archivo de configuración para gestionar la conexión a la base de datos. Este archivo se encuentra en **application/config/database.php**.

```
$db['default'] = array(
    'dsn'      => '',
    'hostname' => 'localhost',
    'username' => 'nombre_usuario',
    'password' => 'contraseña',
    'database' => 'nombre_base_de_datos',
    // ...
);
```

## Consultas a la base de datos

Para interactuar con la base de datos es necesario **cargar la clase de base de datos** en el controlador o el modelo correspondiente.

```
// Se carga la clase de base de datos utilizando:
$this->load->database();
```

Ya a la hora de realizar consultas, se puede hacer uso de **QueryBuilder**, ya que facilita la creación de consultas de manera programática. Igualmente podemos utilizar SQL como veremos en algunos ejemplos.

## Consultas SELECT (seleccionar datos)

```
$query = $this->db->get('tabla'); // SELECT * FROM tabla  
$result = $query->result(); // Obtiene los resultados como un array de  
objetos
```

## Consultas INSERT (insertar datos)

```
$data = array(  
    'campo1' => 'valor1',  
    'campo2' => 'valor2'  
)  
$this->db->insert('tabla', $data);
```

## Consultas UPDATE (actualizar datos)

```
$data = array(  
    'campo1' => 'nuevo_valor1',  
    'campo2' => 'nuevo_valor2'  
)  
$this->db->where('id', 1);  
$this->db->update('tabla', $data);
```

## Consultas DELETE (eliminar datos)

```
$this->db->where('id', 1);
$this->db->delete('tabla');
```

## Consultas personalizadas (SQL directo)

```
$query = $this->db->query("SELECT * FROM tabla WHERE id = ?", array($id));
$result = $query->result();
```

Otros comandos útiles de Query Builder que debemos recordar son:

- **\$query->result()**: se usa para obtener un array de objetos resultantes de una consulta SELECT.
- **\$query->row()**: se utiliza para obtener un solo objeto resultante.
- **num\_rows()**: se utiliza para obtener el número de filas afectadas por una consulta.

```
$query = $this->db->get('tabla');
$num_rows = $query->num_rows();
```

Tras este inciso, proseguimos para cerrar este apartado sobre las operaciones en CodeIgniter. Cabe destacar en este punto que este framework también admite transacciones para **asegurar la integridad de los datos**. Podemos verlo a través de este ejemplo:

```
$this->db->trans_start();
// Operaciones de base de datos aquí
$this->db->trans_complete();
if ($this->db->trans_status() === FALSE) {
    // La transacción falló
}
```

Y aquí damos por finalizada la revisión de las operaciones básicas que podemos realizar con una base de datos en CodeIgniter.

---

**Recuerda: para interactuar con la base de datos se puede utilizar Query Builder o con la ejecución de consultas SQL directamente.**

# Enrutamiento



Definimos el enrutamiento en CodeIgniter como el proceso de mapear las URL de solicitud a los controladores y métodos específicos en una aplicación web.

Como ya podemos suponer, CodeIgniter proporciona un sistema de enrutamiento flexible que permite a los desarrolladores definir rutas personalizadas para dirigir las solicitudes entrantes a funciones de controladores específicas.

## Definición de rutas

Las rutas personalizadas se definen en el archivo "app/Config/Routes.php". Y es en este archivo donde se especifican las rutas personalizadas que determinan cómo se manejan las URL. Cada ruta personalizada se define utilizando el método apropiado, por ejemplo, **\$routes->get()**, **\$routes->post()**, **\$routes->add()**, y se asocia a un controlador y un método específicos.

Si se desea que la URL "/productos" se dirija al método "index" del controlador "Productos", la ruta podría definirse de la siguiente manera:

```
$routes->get('productos', 'Productos::index');
```

En este ejemplo, el **método get()** se emplea para establecer una ruta que maneja las solicitudes GET a la URL "/productos". Cuando un usuario acceda a esta URL, CodeIgniter dirige automáticamente la solicitud al controlador "Productos" y ejecuta su método "index".

## Rutas con parámetros

También CodeIgniter nos da la posibilidad de definir rutas que incluyan **parámetros dinámicos** en la URL. Por ejemplo, si se necesita que la URL "/producto/123" dirija a un producto específico, podemos definir la ruta de esta manera:

```
$routes->get('producto/(:num)', 'Productos::ver/$1');
```

En este ejemplo, "(:num)" es una expresión regular que captura un número y lo pasa como un parámetro al método "ver" del controlador "Productos". De este modo, el número 123 se pasa al método "ver" como argumento.

## Rutas con restricciones

CodeIgniter permite establecer una **ruta predeterminada** que se utiliza si no se proporciona una URL específica. Esto se logra mediante la definición de una ruta predeterminada en el archivo de rutas.

Es posible establecer restricciones en las rutas para limitar los tipos de valores capturados en las URL, como números, letras o patrones personalizados. Supongamos que se desea que una URL solo coincida si contiene un número de dos dígitos en una parte específica de la URL. Se puede lograr esto mediante una restricción con una expresión regular. Aquí hay un ejemplo:

```
$routes->get('producto/(\d{2})', 'Productos::detalle/$1');
```

En este caso, la expresión regular (`|d{2}|`) requiere que la parte de la URL después de `"/producto/"` sea un número de exactamente dos dígitos. Si la URL cumple con esta restricción, se dirige al método `"detalle"` del controlador `"Productos"` y el número se pasa como parámetro.

---

**Sin duda, el enrutamiento en CodeIgniter es muy potente y práctico para que los desarrolladores podamos definir rutas personalizadas y dirigir las solicitudes a controladores y métodos específicos. ¿Ventajas? Nos permite crear URL amigables y gestionar la lógica del enrutamiento en nuestra aplicación web.**

# Vistas



Las vistas en CodeIgniter son componentes esenciales en el patrón de diseño modelo-vista-controlador (MVC) y se utilizan para representar la interfaz de usuario y mostrar datos a los usuarios en una aplicación web.

## Creación de la vista

Las vistas en CodeIgniter son archivos de plantilla escritos en HTML y pueden incluir código PHP para mostrar dinámicamente datos en la interfaz de usuario. Estos archivos se almacenan en el directorio "app/Views" de la aplicación.

## Carga de vistas desde controladores

Para mostrar una vista en una página web, se debe cargar desde un método de un controlador utilizando el método `$this->load->view()` en CodeIgniter. El método `view()` toma el nombre del archivo de vista como argumento, por ejemplo:

```
public function mostrar_pagina() {
    $this->load->view('mi_vista');
}
```

## Cómo pasar datos a las vistas

Los controladores pueden pasar datos a las vistas a través de un **array asociativo**. Estos datos se utilizan para mostrar información dinámica en la interfaz de usuario, por ejemplo:

```
$datos['nombre'] = 'Juan';
$this->load->view('saludo', $datos);
```

En las vistas, se pueden acceder a las variables pasadas desde el controlador mediante las etiquetas PHP. Aquí vemos un ejemplo para mostrar el nombre en la vista:

```
<p>Bienvenido, <?php echo $nombre; ?></p>
```

Por último, hay que señalar que CodeIgniter permite **anidar vistas** dentro de otras vistas y reutilizar partes de una vista en diferentes lugares de la aplicación. Esto facilita la organización y la reutilización del código de interfaz de usuario, como veremos en el siguiente apartado.

## Vistas anidadas

La anidación de vistas en CodeIgniter es una técnica que nos permite **crear vistas compuestas por fragmentos reutilizables**. Esto es especialmente útil para evitar la duplicación de código en las vistas y para organizar mejor la estructura de tu interfaz de usuario. Vamos a desgranar el proceso para entenderlo mejor.

En primer lugar, se crean vistas parciales que representan fragmentos de la interfaz de usuario. Estos fragmentos pueden ser encabezados, pies de página, menús, formularios u otras partes de la estructura que se utilizarán en varias páginas. Por ejemplo, se pueden crear vistas llamadas "*encabezado.php*" y "*pie.php*" en el directorio "*app/Views*" de la aplicación.

```
<!DOCTYPE html>
<html>
<head>
    <title>Mi Página</title>
</head>
<body>
    <?php $this->load->view('encabezado'); ?>
    <!-- Contenido de la página principal -->
    <h1>Bienvenido a mi página</h1>
    <p>Contenido principal de la página.</p>
    <?php $this->load->view('pie'); ?>
</body>
</html>
```

Un último apunte, en este ejemplo, las vistas "*encabezado.php*" y "*pie.php*" se cargan dentro de la vista principal "*mi\_vista.php*".

# Validación de formularios



Qualentum Lab

La validación de datos y formularios en CodeIgniter es otra parte esencial del desarrollo de aplicaciones web, que permite además **garantizar la integridad y seguridad** de los datos registrados por los usuarios. ¿Cómo se realiza esta validación de datos y formularios? ¡Vamos a verlo rápidamente!

## Se carga la biblioteca de validación

En el controlador, primero se carga la biblioteca de validación utilizando el siguiente código:

```
Se carga la biblioteca de validación en el controlador mediante:  
$this->load->library('form_validation');
```

## Se definen reglas de validación

Luego, se definen reglas de validación para los campos del formulario utilizando el método **set\_rules()** de la biblioteca de validación. Por ejemplo, para validar un campo de nombre como requerido y con una longitud mínima, se utiliza el código que se muestra a continuación:

```
// Se establecen reglas de validación para los campos del formulario, por  
ejemplo:  
$this->form_validation->set_rules('nombre', 'Nombre',  
'required|min_length[3]');
```

## Se ejecuta el proceso de validación

Las reglas de validación se definen de manera similar para otros campos. Una vez que se han definido estas reglas, se ejecuta el proceso de validación en el controlador, generalmente después de cargar la biblioteca y establecer las reglas. Observemos este ejemplo:

```
if ($this->form_validation->run() == false) {  
    // Mostrar errores de validación o recargar el formulario  
} else {  
    // Procesar los datos enviados correctamente  
}
```

## Errores de validación

En el caso de que la validación falle, existe la posibilidad de mostrar mensajes de error al usuario para indicar qué campos no cumplen con las reglas de validación. Estos errores de validación se obtienen mediante el siguiente código:

```
// Para mostrar los errores de validación, se utiliza:  
echo validation_errors();
```

## Personalizar los mensajes de error

**¡Importante!** es posible personalizar los mensajes de error para cada regla de validación utilizando el **método `set_message()`** de la biblioteca de validación, por ejemplo:

```
// Se personalizan los mensajes de error para reglas específicas, por  
ejemplo:  
$this->form_validation->set_message('required', 'El campo {field} es  
obligatorio.');
```

Y como no podría ser de otra manera, CodeIgniter también ofrece funciones de saneamiento de datos para limpiar y filtrar estos antes de guardarlos en la base de datos. Esto ayuda a prevenir ataques y a mantener la seguridad de los datos, ¿lo vemos a continuación? ¡Vamos allá!

# Seguridad



Como anunciamos al inicio del fastbook, la **prevención** de ataques comunes es esencial en el desarrollo de aplicaciones web basadas en CodeIgniter para garantizar la seguridad y proteger la integridad de los datos de los usuarios.

Algunas **medidas** importantes que se pueden tomar para prevenir ataques comunes son las siguientes, ¡toma buena nota de ellas!



## Inyección de SQL

- Se deben utilizar consultas preparadas o el sistema de Query Builder de CodeIgniter para construir consultas SQL de manera segura.
- Hay que evitar también la concatenación directa de variables en las consultas SQL.
- Es importante utilizar funciones como `$this->db->escape()` para sanear los datos antes de incluirlos en las consultas SQL.
- Y muy recomendable es limitar los privilegios de la cuenta de base de datos utilizada por la aplicación.



## Cross-Site Scripting (xss)

- Al obtener datos del formulario, se debe utilizar la función `$this->input->post()`, ya que aplica automáticamente la función `xss_clean()`.
- Se debe utilizar la biblioteca de formularios de CodeIgniter, que aplica la limpieza XSS de manera predeterminada.
- Es esencial validar y filtrar cualquier entrada de usuario antes de mostrarla en las vistas.
- Aquí mostramos cómo configurar el archivo `config.php` para activar el filtro XSS globalmente.

```
$config['global_xss_filtering'] = true;
```



## Cross-Site Request Forgery (CSRF)

- Es importante habilitar la protección CSRF en la aplicación mediante la configuración del archivo `config.php`.
- Se deben utilizar las funciones `form_open()` y `form_close()` para generar formularios, ya que estas generan automáticamente los campos de CSRF.
- Se debe verificar la presencia del token CSRF en las solicitudes POST utilizando `$this->security->csrf_verify()`.

```
$config['csrf_protection'] = true;
```

# Conclusiones y recursos de interés



Qualentum Lab

A lo largo del fastbook, hemos visitado los conceptos más importantes para el uso del framework CodeIgniter en el desarrollo de aplicaciones web. Como has estudiado, este framework PHP no solo **simplifica el proceso de construcción de aplicaciones**, sino que también destaca por su integración eficiente con bases de datos, brindándonos a los desarrolladores las herramientas necesarias para gestionar datos de manera efectiva.

En este contexto, hemos descubierto la versatilidad de CodeIgniter al proporcionar una estructura organizada que agiliza el desarrollo y fomenta las mejores prácticas. Además, hemos explorado cómo CodeIgniter facilita la interacción con bases de datos mediante **su soporte nativo para el lenguaje SQL**, permitiendo la ejecución eficiente de consultas y garantizando la integridad de los datos almacenados.

En fin, CodeIgniter emerge como un aliado sólido y confiable en el desarrollo de aplicaciones web modernas. Su capacidad para simplificar la gestión de bases de datos, junto con su estructura modular y escalabilidad, lo convierten en una opción muy valorable para aquellos profesionales que buscamos construir aplicaciones eficientes y robustas en un entorno dinámico de la web.

---

**Antes de dar por finalizado el fastbook, te recuerdo la documentación oficial que hemos compartido y que puede resultar muy relevante para tu formación, ¡no la pierdas de vista!**

- Documentación oficial de [CodeIgniter](#).
- Documentación sobre [Composer](#).

**¡Enhorabuena! Fastbook superado**



[Qualentum.com](http://Qualentum.com)