Pin

Hola, soy Nelson Rojas y en este documento quiero contarte qué es Pin y cómo puedes usarlo en tus proyectos.

Qué es Pin

Pin es un framework PHP minimalista que te permite crear aplicaciones web de manera sencilla y eficiente.

Tiene una arquitectura basada en el patrón Front Controller, lo que significa que cada solicitud será capturada por el archivo index.php. El Front Controller traduce la solicitud en una **página** y una **acción**. La **página** es solo un archivo php, no una clase, solo un conjunto de funciones. La **acción** corresponde a una función dentro de este archivo.

Minimalista

Pin es minimalista, no tiene muchas características, pero tiene lo necesario para crear aplicaciones web. La lista de características es la siguiente:

- Front Controller
- Vistas con plantillas o sin ellas
- Variables de sesión
- Redirección de URLs
- Manejo de errores
- Acceso a la base de datos usando PDO
- Acceso a elementos de formulario de forma segura
- Helpers para generar elementos HTML y de formulario de forma sencilla
- Autoload de clases
- Control de acceso a páginas

PROF

Y eso es todo.

Extensible

Pin es extensible, puedes crear tus propias funciones y usar las que ya existen.
Como el autoload de clases es una función de PHP sencilla, es cosa que la intervengas como desees. De hecho, hasta puedes usar librerías cargadas con composer si agregas el autoload de composer a tu proyecto.

```
//autoload de clases en Pin (en el archivo load.php)
spl_autoload_register(function($className){
    $file = PIN_PATH . 'libs' . DS . strtolower($className) . '.php';
    if (file_exists($file)) {
        require_once $file;
        return;
```

```
    } else {
        throw new Exception("$className no existe en $file", 1);
    }
});
```

Cómo trabaja Pin

Al usar el Front Controller, cada solicitud se traduce en una página y una acción.

Por ejemplo, si la URL es http://localhost/pin/pages/show/contactenos, el Front Controller traduce esto en la página *pages* y la acción *show*, y le pasa el parámetro *contactenos* a la acción.

Luego, el Front Controller carga la página, lo que significa que carga el archivo *pages.php* y busca una función llamada *show* dentro de ese archivo.

Si la función *show* existe, el Front Controller la ejecuta, pasando los parámetros que necesite, y le pasa el resultado a la vista.

```
//archivo pin/pages/pages.php

function show($slug)
{
    load_view("pages/show");
    //espera que exista una vista en pin/views/pages/show.phtml
}
```

En el ejemplo anterior, la vista cargada es una vista sin plantilla, es decir, carga lo que esté definido en el archivo *pages/show.phtml*.

Si quisieras usar una plantilla, puedes hacerlo de la siguiente manera:

PROF

```
//archivo pin/pages/page.php
function show($slug)
{
    //crear una instancia de Template
    //de forma predeterminada, Pin usa como directorio de plantillas
pin/templates
    //y particularmente la plantilla por defecto es *default.phtml*
    $template = new Template();
    //crea una variable $saludo que será enviada luego a la vista
    $template->saludo = "Hola Mundo";
    //de acuerdo al ejemplo de la url, $slug contendrá *contactenos*
    $template->slug = $slug;
    $template->render("pages/show");
    //carga la vista pages/show.phtml, enviando las variables $saludo y
```

```
$slug
    //y envolviendo el resultado en la plantilla default.phtml
}
```

Nótese que a causa de lo que se explicó anteriormente, Pin usa un sistema de directorios predefinido para alojar varias de sus características.

- *pin/pages*: para las páginas (los archivos php que contienen las funciones)
- *pin/views*: para las vistas (los archivos phtml que contienen el código html)
- *pin/templates*: para las plantillas (los archivos phtml que contienen el diseño de las vistas)
- *pin/helpers*: para las helpers (los archivos php que contienen funciones útiles)

El archivo *load.php* que está definido en la raíz del proyecto tiene una lista de funciones útiles principalmente para *cargar* cosas como vistas, plantillas, helpers, etc.

La lista de funciones disponibles es la siguiente:

- *load_view($view, $vars = [])*: carga una vista y eventualmente le pasa variables en un arreglo asociativo.
- *load_helper($helper)*: carga un helper.
- *load_config($config)*: carga una configuración.
- *load_partial($partial, $vars = [])*: carga un parcial (una vista que se puede usar dentro de otra vista) y eventualmente le pasa variables en un arreglo asociativo.
- *load_file($file, $ext, $path, $type, $vars = [])*: carga un archivo (una vista, una plantilla o un helper) y eventualmente le pasa variables en un arreglo asociativo. Est útil para cargar cualquier tipo de archivo que sea necesario.
- *redirect_to($url)*: redirige a una url.

Como *load.php* contiene una lista de funciones útiles cargada cada vez que se hace una solicitud, también aloja un interceptor para *manejar errores*.

Una función no mencionada antes es *load_page_from_url($url)*: esta función es la que se encarga de traducir la url en una página y una acción y cargar la página correspondiente.

PROF

No es algo que necesites cargar manualmente, esta llamada se hace automáticamente.

Acceso a la base de datos

Pin usa PDO para acceder a la base de datos. Para ello utiliza el archivo *config.php* donde aloja la configuración de la base de datos.
Luego, usando la clase *Db*, puedes acceder a la base de datos de la siguiente manera:

```
//consultar todos los registros de la tabla users
$users = Db::findAll("SELECT * FROM users");

//consultar un registro de la tabla users donde el id sea 1
$user = Db::findFirst("SELECT * FROM users WHERE id = :id", [":id" =>
1]);
```

```php
//insertar un registro en la tabla users
Db::insert("users", ["name" => "John Doe", "email" => "john@doe.com"]);

//actualizar un registro en la tabla users donde el id sea 1
Db::update("users", ["name" => "Jane Doe"], "WHERE id = 1");

//eliminar un registro en la tabla users donde el id sea 1
Db::delete("users", "WHERE id = :id", [":id" => 1]);

//obtener la fecha y hora desde el servidor de base de datos
$fecha_hora = Db::getScalar("SELECT CURRENT_TIMESTAMP");
```

La clase *Db* también implementa métodos para usar transacciones, como *beginTransaction()*, *commit()* y *rollBack()*.

Principalmente, la idea de usar la clase *Db* es para evitar escribir código sql directamente en las páginas, y en su lugar usar métodos que facilitan la escritura y ejecución de sentencias sql. Aunque esto es más bien válido para operaciones CUD (Crear, Actualizar y Eliminar). Para las operaciones de lectura, sigue siendo necesario escribir el código sql directamente, aunque puedes asegurarlo usando los parámetros de PDO.

En otro de mis proyectos, IsaliaPHP, existe una clase llamada SqlBuilder que permite construir sentencias sql de forma más sencilla y segura sin tener que escribir tanto código.

Otras clases útiles de Pin

Template

La clase *Template* es la encargada de cargar las plantillas y pasarles variables para su uso.
En ella se pueden cargar variables que estarán disponibles en las vistas. Para ello se basa de los métodos mágicos de PHP __get y __set. Así, para pasar una variable *saludo* a la vista, se debe usar

PROF

```php
$template->saludo = "Hola Mundo";
```

La vista esperará encontrar una variable *$saludo* y usar su valor.

También es posible cambiar la vista por defecto que se usará en el template usando el método *setTemplate($path)*:

```php
$template->setTemplate("otra_vista");
```

Se espera que exista un archivo llamado *otra_vista.phtml* en el directorio *pin/templates*.

Para renderizar la vista, se debe usar el método *render($view)*:

```
$template->render("pages/show");
```

Se espera que exista un archivo llamado *show.phtml* en el directorio *pin/views/pages*.

Para que el contenido de la vista pueda ser cargado en la plantilla, la plantilla tendrá una variable especial llamada *$yield*.

El ejemplo siguiente presenta el contenido de la plantilla *default.phtml* y la vista *page/show.phtml*:

```
//plantilla pin/templates/default.phtml
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Pin Default Template</title>
  <?= stylesheet_link('bootstrap'); ?>
  <style>
      body {
        padding-top: 60px; /* 60px to make the container go all the way
to the bottom of the topbar */
      }
  </style>
  <?= stylesheet_link('bootstrap-responsive'); ?>
  <!-- HTML5 shim, for IE6-8 support of HTML5 elements -->
  <!--[if lt IE 9]>
      <?= javascript_include_tag('html5shiv'); ?>
  <![endif]-->
</head>
<body>
  <?= load_partial('navbar'); ?>
  <div class="container">
    <?php echo $yield; ?>
  </div><!-- end container -->
  <?= javascript_include_tag('jquery'); ?>
  <?= javascript_include_tag('bootstrap'); ?>
</body>
</html>
```

Como se aprecia, es una plantilla básica que incluye el uso de *partials* para cargar el navbar y las funciones helpers para cargar los scripts de javascript y las hojas de estilo css.

```
//vista pin/views/pages/show.phtml
<h1><?= $saludo;?></h1>
```

Session

La clase *Session* es la encargada de manejar las variables de sesión.
Tiene una lista de métodos para manejar las variables de sesión de forma sencilla:

- *set($key, $value)*: establece una variable de sesión.
- *get($key)*: obtiene una variable de sesión.
- *delete($key)*: elimina una variable de sesión.
- *destroy()*: elimina todas las variables de sesión.

Todos son métodos estáticos, así que para usarlos se debe llamar a la clase y luego el método:

```
Session::set("saludo", "Hola Mundo");

echo Session::get("saludo");

Session::delete("saludo");

Session::destroy();
```

Request

La clase *Request* es la encargada de manejar los elementos enviados usando formularios o por la url.

Para ello tiene una lista de métodos para acceder a los elementos enviados:

- *get($key)*: obtiene un elemento enviado usando el método get.
- *post($key)*: obtiene un elemento enviado usando el método post.
- *hasPost($key)*: verifica si un elemento fue enviado usando el método post.
- *hasGet($key)*: verifica si un elemento fue enviado usando el método get.

Todos estos métodos son estáticos, así que para usarlos se debe llamar a la clase y luego el método:

PROF

```
echo Request::get("nombre");
```

Helper

De forma predeterminada, Pin tiene una lista de helpers predefinidos que puedes usar en tus páginas.
La lista de helpers es la siguiente:

- *html_tags*: para generar código html y es cargado automáticamente.

Dentro de *html_tags* se encuentran funciones para generar código html como las que se indican a continuación:

- *stylesheet_link($src)*: genera un link para una hoja de estilo css.
- *javascript_include_tag($src)*: genera un link para un script de javascript.

- *img_tag($src, $alt, $attributes)*: genera una etiqueta img.
- *form_tag($action, $attributes)*: genera una etiqueta form.
- *end_form_tag()*: cierra una etiqueta form.
- *submit_tag($caption, $attributes)*: genera un botón submit.
- *button_tag($caption, $type, $attributes)*: genera un botón.
- *text_field_tag($name, $value, $attributes)*: genera un campo de texto.
- *password_field_tag($name, $value, $attributes)*: genera un campo de contraseña.
- *text_area_tag($name, $value, $attributes)*: genera un área de texto.
- *hidden_field_tag($name, $value, $attributes)*: genera un campo oculto.
- *check_box_tag($name, $value, $checked, $attributes)*: genera una casilla de verificación.
- *radio_button_tag($name, $value, $checked, $attributes)*: genera un botón de radio.
- *label_tag($field, $caption, $attributes)*: genera una etiqueta para un campo.
- *select_tag($name, $options, $include_blank, $attributes)*: genera una lista desplegable.
- *options_for_dbselect($data, $show, $value, $selected)*: genera opciones para una lista desplegable a partir de un conjunto de datos.
- *options_for_select($data, $selected)*: genera opciones para una lista desplegable a partir de un conjunto de datos.
- *js_redirect_to($action, $seconds)*: genera un script para redirigir a una acción después de un tiempo.
- *button_to_action($caption, $action, $attributes)*: genera un botón que redirige a una acción.

Control de acceso a páginas

Pin permite controlar el acceso a páginas usando sesiones.

Para ello, se debe crear una función llamada *page_initializer* en la página que se desea proteger. Esta función será ejecutada automáticamente antes de que se ejecute la acción de la página.

```
//archivo pin/pages/page.php
function page_initializer()
{
    if (Session::get("is_logged_in") !== true) {
        //ir a la pagina inicial
        Session::set("flash", "Debe iniciar sesión para acceder al
recurso *<i>page</i>*");
        return redirect_to("");
    }
}
```


El ejemplo anterior protege la página *page* para que no pueda ser accedida si no se ha iniciado sesión. Si no se ha iniciado sesión, se redirigirá a la página inicial y se mostrará un mensaje de error usando la variable de sesión *flash*.

Conclusión

Pin es un framework PHP minimalista que te permite crear aplicaciones web de manera sencilla y eficiente.

Tiene una arquitectura basada en el patrón Front Controller, lo que permite una fácil comprensión y uso.
Cuenta con una lista de helpers predefinidos para generar código html y formularios de forma sencilla.
Permite controlar el acceso a páginas usando sesiones.
Te ofrece una clase para acceder a la base de datos usando PDO y otra para manejar los datos de los formularios de forma segura.
Y, si quieres usar plantillas y ahorrar tiempo escribiendo sólo lo necesario en las vistas, también puedes hacerlo.

Te deseo éxito si usas Pin!
Saludos cordiales desde Talca, Chile!
Nelson Rojas
Creador de Pin