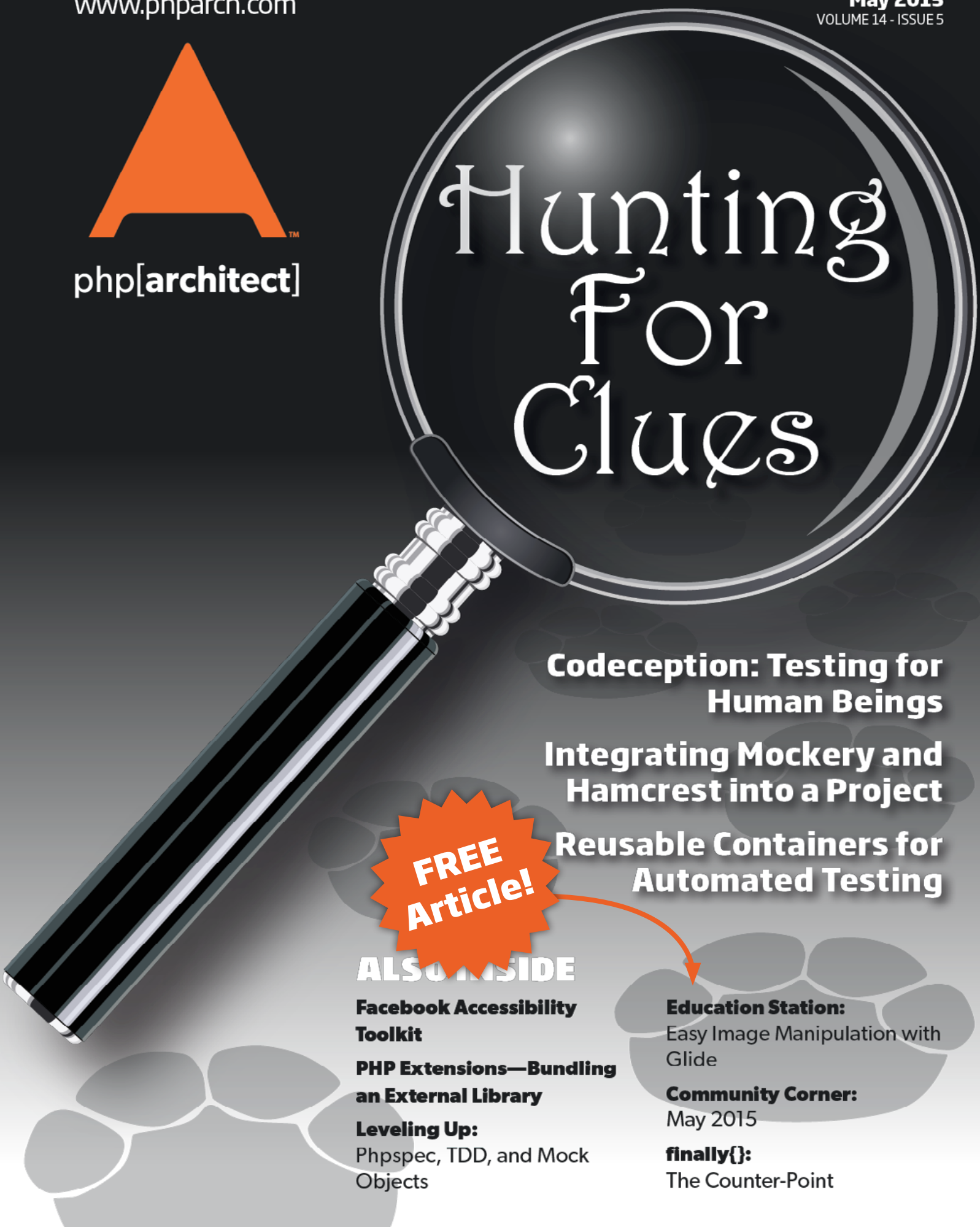
www.phparch.com

May 2015
VOLUME 14 - ISSUE 5



php[architect]

Hunting For Clues

Codeception: Testing for Human Beings

Integrating Mockery and Hamcrest into a Project

Reusable Containers for Automated Testing

FREE Article!

ALSO INSIDE

Facebook Accessibility Toolkit

PHP Extensions—Bundling an External Library

Leveling Up:
Phpspec, TDD, and Mock Objects

Education Station:
Easy Image Manipulation with Glide

Community Corner:
May 2015

finally{}:
The Counter-Point

Easy Image Manipulation with Glide

Matthew Setter

Welcome back to another edition of Education Station. In this month's edition, we're going to have fun with images; specifically, we're going to look at a library that makes it easy to manage just about everything to do with images in a web application. What's more, it's a library brought to you by The PHP League, otherwise known as *The League of Extraordinary Packages*—it's called *Glide*.

DisplayInfo()

Requirements:

- PHP 5.4 or above
- Composer
- Git
- Glide - http://glide.thephpleague.com

Related URLs:

- Silex Documentation - http://silex.sensiolabs.org/documentation

Easy Image Manipulation with Glide

But before we get into it, let's set the scene appropriately. You're building a web application (or a website for a client), and you need to make images available, with no more effort than a standard HTML `img` link. But via that link, you need to be able to pass different parameters, allowing for the image to be rendered with a range of different effects and transformations, such as the following:

- Image manipulation (including cropping, stretching, and resizing)
- Image adjustment (including setting the brightness, contrast, and gamma level)
- Image effects (including blurring, pixelation, and filtering)
- Image quality
- Securing image URLs against user abuse

If you wanted all of this, it's understandable that to code it in-house would take quite a bit of time and effort. Therefore, it is much better to find an external library—one developed, maintained, and tested to do all of this—which you can take and use.

Fortunately, that's just what Glide is. Built atop two well-tested libraries, Intervention Image and Flysystem, which we looked at in February this year, Glide makes it virtually painless to set up an image server and integrate it with your site or application. Now you have one less consideration to manage as you build your application—one less component to build, manage, and test.

In this month's column, we're going to look at how to get it set up in coordination with Silex and how to perform a range of the effects, which I've already mentioned above. Just briefly, the reason for Silex is that Silex handles a range of functionality, which Glide doesn't natively include, such as routing; in addition, it is tightly integrated with the secure URLs feature.

So first, let's make both Glide and Silex available to our application.

Installing Glide & Silex

Assuming that you already have a new project directory set up for testing out Glide and that you have Composer in your system path, in the project directory, you can create a new `composer.json` file and add in the configuration below. You should also ensure the `exif` extension is enabled in your php installation.

```
{
    "require": {
        "league/glide": "0.3.*"
    }
}
```

Alternatively, you can run the following commands, which will do it for you:

```
composer require league/glide:"0.3.*";
composer require silex/silex:"~1.2";
```

If you've created the file manually, then don't forget to run `composer install`.

Directory Setup

Next, in your project directory, create a new directory called `data`; then, inside data, create two further directories, called `source` and `cache`. These two sub-directories will contain the images and the cached copies of them, respectively.

A Basic Glide Server

With the software available, the Glide source will be available in `/vendor`. So now, create a new file called `index.php`, and add in the code from Listing 1.

This will set up a basic Glide server, which looks for the original images in `data/source` and creates cached copies after the first request in `data/cache`. The third option, `base_url`, sets up the base URL for the img tag's `src` attribute. For example, let's say that I have an image with the name of `1.jpg`.

It's rather uncreative, but it works for the purposes of a demonstration. With the server set up and configured, we've then created a basic Silex application with debugging enabled (just in case something goes wrong) and initialized a request object. Of course, when you use this in production, you'd want to flip `debug` to `false`.

A Basic Route

Now, believe it or not, that's a lot of the work already done. However, we still need to configure a route so that we can access the images from an application. I'm going to keep it simple and use the following: `/img/{id}`. When dispatched, this will request from the Glide server the file `/data/source/1.jpg`. So now, to define this route in `index.php`, add the following code from Listing 2 at the end (above `$app->run()`).

If you're not familiar with Silex applications, this sets up a so-called named route, which can contain named parameters, such as `{id}`. So to complete the earlier example, we could request routes such as `/img/1`, `img/200`, and so on.

You'll notice that the first parameter to the closure is `$id`, which matched the named argument. Silex takes care of automatically initializing `$id` for us by parsing the dispatched URL and extracting a value for `$id` from it. Finally, we pass to the closure both our Silex `$app` and the Glide `$server` variable.

index.php **LISTING 1**

```
01. <?php
02.
03. require_once('vendor/autoload.php');
04.
05. use League\Flysystem\Adapter\Local;
06. use League\Flysystem\Filesystem;
07. use League\Glide\ServerFactory;
08. use Symfony\Component\HttpFoundation\Request;
09.
10. // Setup Glide server
11. $server = ServerFactory::create([
12.     'source' => new Filesystem(new Local(__DIR__
13.                                          . '/data/source')),
14.     'cache' => new Filesystem(new Local(__DIR__
15.                                         . '/data/cache')),
16.     'base_url' => '/img',
17.     'max_image_size' => 1000 * 1000,
18. ]);
19.
20. $app = new Silex\Application([
21.     'debug' => true
22. ]);
23.
24. // Create request object
25. $request = Request::createFromGlobals();
26.
27. $app->run();
```

For further information on Silex, check out the online documentation, which you can find at http://silex.sensiolabs.org/documentation. You will note that it is quite extensive and very well written.

A Basic Image Request

Inside the closure, the code makes use of the `outputImage()` function. This both generates and outputs a manipulated image, based on the image requested. The second parameter, a configuration array, is the most interesting part. Here, we've specified two parameters, `w` and `h`, which I'm sure you've guessed are width and height, respectively. Here's the full list of available options:

Adding our image route **LISTING 2**

```
01. // define image route
02. $app->get(
03.     '/img/{id}',
04.     function($id, Request $request) use($app, $server) {
05.         $server->outputImage(
06.             $app->escape($id) . '.jpg',
07.             [
08.                 'w' => 400,
09.                 'h' => 400,
10.             ]
11.         );
12.     }
13. );
14.
15. $app->run();
```

Parameter	Description
w	width of the image
h	height of the image
fit	Sets how the image is fitted to its target dimensions (May be one of the following: contain, max, stretch, crop)
crop	Sets where the image is cropped when the fit parameter is set to crop
rect	Crops the image to specific dimensions prior to any other resize operations
or	Angle of orientation
bri	Brightness
con	Contrast
gam	Image gamma
sharp	Sharpness
blur	Amount of blurring
pixel	Amount of pixelation
filt	Apply a greyscale or sepia filter
q	Image quality
fm	Image format (can be Jpeg, PNG, or Gif)

We're not going to add all of the effects, but we're going to play with the width, height, orientation, pixelation, image quality, and application of a filter. To do that, we need a good image. And as I'm a huge Marvel fan, and Avengers 2—Age of Ultron's just about out, what better image to use than one of the Avengers, which I found on Flickr (Creative Commons, of course), at http://phpa.me/avengers-mural.

Original image at 2048x1536 **FIGURE 1**

You can see the original in Figure 1. Download it and save it in `data/source` as `avengers.jpg`. Next, update the Silex route to be `/img/{name}`, instead of `/img/{id}/small`, and in the closure, change `$id` to `$name`. With that done, use PHP's built-in web server to launch the application. If you're not familiar with using it, you can start it up with the following command:

```
php -S localhost:8080 -t .
```

Assuming all went well, open up `http://localhost:8080/img/avengers2` in your browser, and you'll see the image scaled to 400 pixels wide and 400 pixels high. If you now look in `/data/cache`, you'll also see a cached copy of the file.

It's worth mentioning here that the original image is only used, by default, if a cached copy has not been generated based on the configuration options supplied. What's great about Glide is that you don't have to do anything other than configure the cache directory and ensure that it has the correct permissions to set up caching.

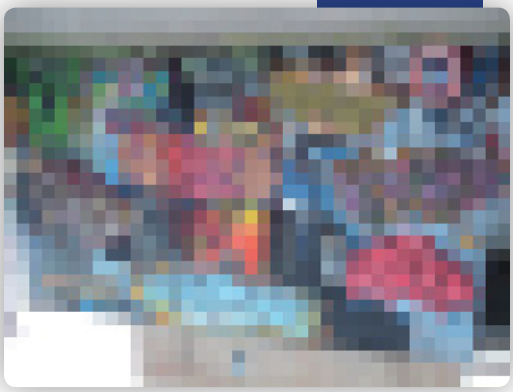
It's all done for us. Now let's flip the image upside down and pixelate it. To do that, we're going to set the orientation (or) to 180 degrees and pixelation (pixel) to 10. Orientation accepts the following: auto, 0, 90, 180, and 270, while Pixel accepts a number between 0 and 1000.

So update the configuration array to the following:

```
[
    'w' => 400,
    'h' => 400,
    'or' => 180,
    'pixel' => 10
]
```

If you reload the page in your browser, you'll see the image is now upside down and barely recognizable, as in Figure 2.

Upside down and pixelized **FIGURE 2**



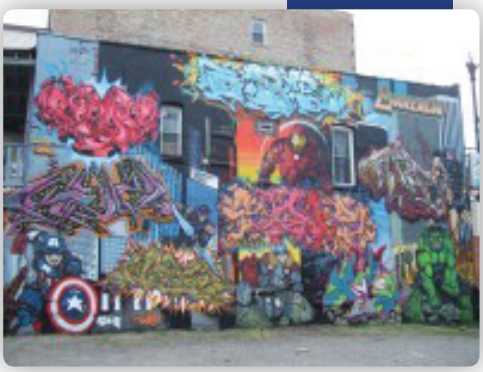
Now that that's done, let's make it recognizable again by removing the pixelation, resetting it to its original orientation, removing 'or,' and having it automatically scale the height, proportional to a width of 1000 pixels. In addition to that, we'll apply the last effect, a Sepia filter. To do all of this, update the configuration array again to the following:

```
[
    'w' => 1000, 'filt' => 'sepia'
]
```

Reloading the page again, you'll see the final version of the image, as in Figure 3. I don't know about you, but I think that a Sepia effect always looks amazing. There's something about a lack of color that makes an image look glorious.

Classic sepia tone **FIGURE 3**

Scaled to 200 pixels wide **FIGURE 4**



Protecting Images

Now that we've looked at the effects that you can create with Glide, it's important to cover one final key aspect of the library, one essential in a production environment—securing images. If you've not thought of it before now, securing images is essential so that malicious users can't abuse your server by attempting to generate any number of variations of one or more images to exhaust server memory and CPU.

In essence, it works by restricting image parameters when a unique token is signed with your secret key, which isn't present in the request. If it's missing or if it doesn't match the expected hash of attributes+secret key, like a salt in a password, the new combination won't be generated.

So what we're going to do is adjust the original route to only render an image if a valid URL is used, which we'll generate in a second route. In the second route, we'll generate a URL, which contains the signing key, for the Avengers 2 image scaled to a width of 200 pixels. You can see this in Figure 4.

First, let's look at the route that generates the URL, which you can see in Listing 3.

Now let's look at Listing 4 with the revised, original route, which accounts for image protection. Notice that we had to add some `use` statments for some helper classes.

Route for a secure image **LISTING 3**

```
01. $app->get(
02.     '/print-secure-url',
03.     function() use($app, $server, $signingKey) {
04.         $urlBuilder = UrlBuilderFactory::create(
05.                         'http://localhost:8080', $signingKey);
06.         return $urlBuilder->getUrl('/img/avengers2',
07.                                    ['w' => 200]);
08. });
```

Secure image validation **LISTING 4**

```
01. <?php
02. require_once('vendor/autoload.php');
03.
04. use League\Flysystem\Adapter\Local;
05. use League\Flysystem\Filesystem;
06. use League\Glide\ServerFactory;
07. use League\Glide\Http\UrlBuilderFactory;
08. use League\Glide\Http\SignatureException;
09. use League\Glide\Http\SignatureFactory;
10. use Symfony\Component\HttpFoundation\Request;
11.
12. // Setup Glide server
13. $server = ServerFactory::create([
14.     'source' => new Filesystem(new Local(__DIR__
15.                                          . '/data/source')),
16.     'cache' => new Filesystem(new Local(__DIR__
17.                                         . '/data/cache')),
18.     'base_url' => '/img',
19.     'max_image_size' => 1000 * 1000,
20. ]);
21.
22. $app = new Silex\Application([
23.     'debug' => true
24. ]);
25.
26. // Create request object
27. $request = Request::createFromGlobals();
28.
29. // a secret key only we know
30. $signingKey = "Ultron Goes Down";
31.
```

Continued Next Page

If you now open up http://localhost:8080/print-secure-url, you'll see that it generates a URL which includes the following:

```
/img/avengers?w=200&s=4c740579415917b96cb22ed3adcd5a8e
```

If you open that up, you'll see the image render correctly. If you change the secret key(s) or any of the image options, you'll be redirected to a 404 page, indicating that the image isn't available.

Wrapping Up

And that's how to rapidly set up a flexible yet secure image server for your web applications and websites. Sure, we've not looked at every function that the library has to offer, but I'm confident that if you're keen, there's more than enough for you to still look at. Explore and enjoy. I hope that this library helps you out—and saves you lots of developmental effort and overhead.

'Til next time, happy coding.

Secure image validation **LISTING 4 (CONT'D)**

```
32. // route to display URL for our secure image
33. $app->get(
34.     '/print-secure-url',
35.     function() use($app, $server, $signingKey) {
36.         $urlBuilder = UrlBuilderFactory::create(
37.                         'http://localhost:8080', $signingKey);
38.         return $urlBuilder->getUrl('/img/avengers',
39.                                    ['w' => 200]);
40. });
41.
42. // route to display secure image, if token is correct
43. $app->get(
44.     '/img/{name}',
45.     function($name, Request $request)
46.         use($app, $server, $signingKey) {
47.         // Validate HTTP signature
48.         try {
49.             SignatureFactory::create($signingKey)
50.                 ->validateRequest($request);
51.         } catch (SignatureException $e) {
52.             // Handle error
53.             $app->abort(404, "Illegal Image Request");
54.         }
55.
56.         $server->outputImage(
57.             $app->escape($name) . '.jpg',
58.             ['w' => 200]
59.         );
60.     }
61. );
62.
63. $app->run();
```

MATTHEW SETTER is a software developer, freelance technical writer http://www.matthewsetter.com/services/ and editor of Master Zend Framework, dedicated to helping you become a Zend Framework master? Want to be a master? Come find out more http://www.masterzendframework.com/welcome-from-phparch.

Twitter: @settermjd

Want more articles like this one?

Keep your skills current and stay on top of the latest PHP news and best practices by reading each new issue of php[architect], jam-packed with articles.

Learn more every month about frameworks, security, ecommerce, databases, scalability, migration, API integration, devops, cloud services, business development, content management systems, and the PHP community.

php[architect]

magazine

books

conferences

training

phparch.com

Get the complete issue for only $6!

We also offer digital and print+digital subscriptions starting at $49/year.