www.phparch.com

January 2022
Volume 21 - Issue 1

php[architect]

The Magazine For PHP Professionals

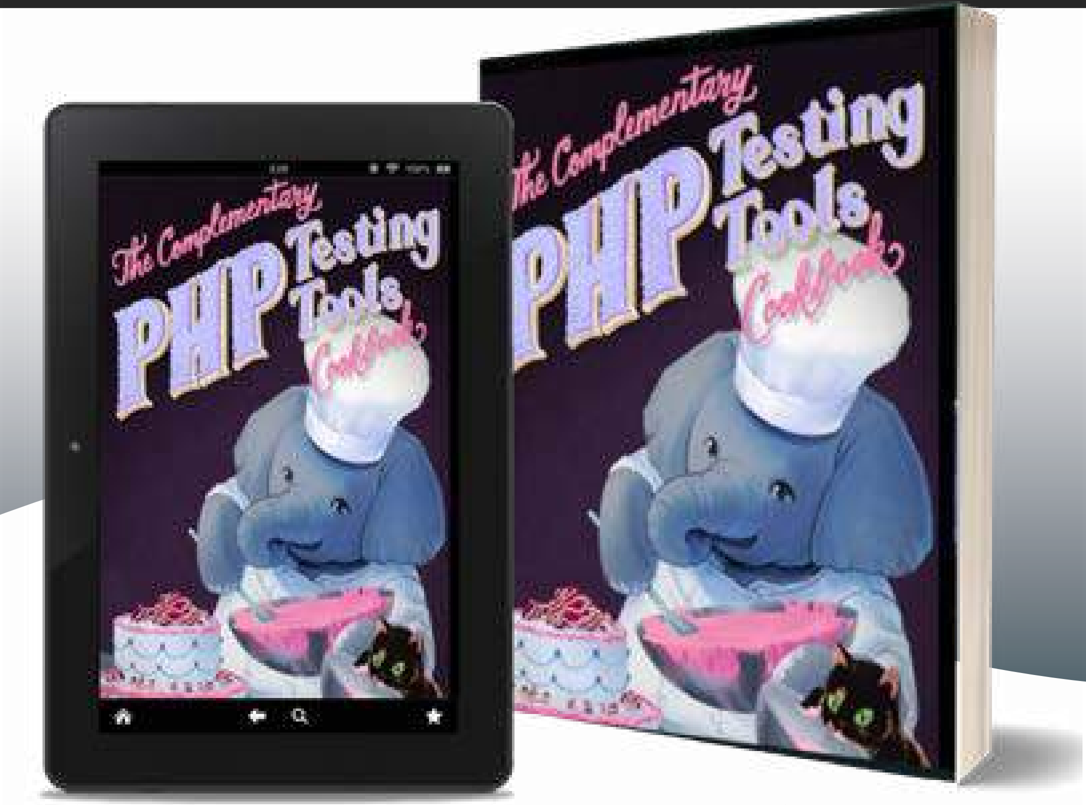
Domain-Driven Resolutions

Introducing FilterIterators

Hack Your Home With a Pi

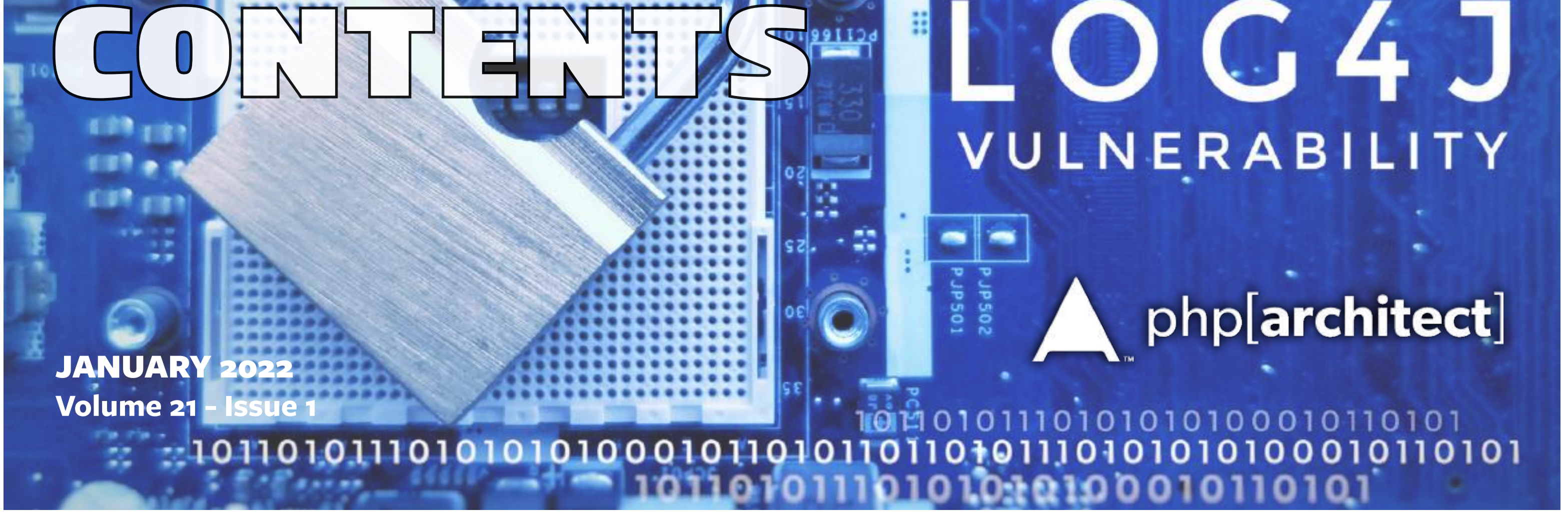
Effective Mentoring

ALSO INSIDE



The Complementary
PHP Testing Tools
Cookbook

CONTENTS



JANUARY 2022
Volume 21 - Issue 1

Edited with new resolutions

php[**architect**] is published twelve times a year by:

PHP Architect, LLC
9245 Twin Trails Dr #720503
San Diego, CA 92129, USA

Subscriptions
Print, digital, and corporate subscriptions are available. Visit https://www.phparch.com/magazine to subscribe or email contact@phparch.com for more information.

Contact Information:
General mailbox: contact@phparch.com
Editorial: editors@phparch.com

Print ISSN 1709-7169
Digital ISSN 2375-3544

Although all possible care has been placed in assuring the accuracy of the contents of this magazine, including all associated source code, listings and figures, the publisher assumes no responsibilities with regards of use of the information contained herein or in all associated material.

Happy New Year

John Congdon

Happy New Year from everyone at PHP Architect. We genuinely hope you had a wonderful holiday season and are ready to do great things in 2022. We also hope that we have a small part in making 2022 great for you.

As 2021 fades from our memories, we hope that a few of you received fantastic geeky gifts during the holidays. Hopefully, some of those geeky gifts included Raspberry Pis. Over the next few months, we will build a project with one and hopefully inspire you to create something useful for yourself.

This is also a great time to establish some resolutions to aid in your coding career. What do you want to accomplish this year? Continuing education is always a great goal and something that we here at PHP Architect strive to provide. We are constantly learning how to improve our process, and one of our New Year's Resolutions is to be more prompt in our monthly delivery. We know this will take another few months, but it is our goal. Now I want to take a moment to ask you for a favor. Have you ever wanted to be a "published writer"? Maybe you still need a New Years Resolution. We'll help you keep it. As a magazine for our community, we rely on community members to write articles for us. And better yet, we pay for your content to be published. So take a moment and ask yourself a few questions. "Have I learned anything new recently that helped me?" "Do I want to learn a new skill to share with other people? Sometimes having an article deadline will help you focus and learn that skill. Honestly, this point has helped me write all of my articles and conference talks. If you are interested, please send an email to write@phparch.com[1] with your ideas, and we will help you turn your idea into a publishable article.

This month's issue gets started with "Introducing FilterIterators" by Mauro Chojrin. This feature article will get you looking at your projects in a whole new way. Our second feature is the first in a series called "How to Hack your Home with a Raspberry Pi" by Kenneth Marks. I've always been fascinated by the Raspberry Pi but have always been intimidated by them. Ken is opening my eyes, and now I want to use them everywhere.

Our columnists have been hard at work too. Joe Ferguson's, The Workshop, is giving newer developers a course on "Apache and PHP - Back to Basics." Every developer should have more of an understanding of what happens even before their PHP code fires up and starts handling a request. In Education Station, Chris Tankersly is talking about "Background Queues" to improve performance for our applications. Eric Mann brings us "The Terrifying Scale of a Security Bug" in this month's Security Corner. If you're out on the hunt for a job, make sure you understand the basics of the "Infamous Fizz Buzz" in this month's PHP Puzzles by Oscar Merida. Then Eric Van Johnson does a great interview with Patrick Allaert, one of your PHP 8.1 Release Managers.

And finally{}, Beth Tucker-Long talks "Experts or Out-of-touch?" where she shares some of her past with PHP Internals.

As always, thank you for subscribing to PHP Architect. We have big plans for 2022, and we hope you do too.

[1] *write@phparch.com: mailto:write@phparch.com*

Write For Us

If you would like to contribute, contact us, and one of our editors will be happy to help you hone your idea and turn it into a beautiful article for our magazine.

Visit https://phpa.me/write or contact our editorial team at write@phparch.com and get started!

Stay in Touch

Don't miss out on conference, book, and special announcments. Make sure you're connected with us.

- Subscribe to our list: http://phpa.me/sub-to-updates
- Twitter: @phparch
- Facebook: http://facebook.com/phparch

Download the Code Archive:

http://phpa.me/January2022_code

Introducing FilterIterators

Mauro Chojrin

Iterators over collections provide a clean separation of concerns. Filter Iterators are a great way to abstract the logic of your selecting sub-sets of Collections elements.

Introduction

This article will provide an overview of what Collections are and why you should prefer them to plain PHP arrays. Then I'll focus on one particular aspect of working with Collections: filtering elements. You'll have the opportunity to see an implementation of the concepts discussed in the context of a coding challenge issued as part of an actual technical interview. By reading this article, you'll learn how to structure your code to better prepare for eventual requirement changes.

Setting the Ground

Before jumping into the details, it makes sense to take a moment to explore some basic concepts, namely Collections and Iterators.

What is a Collection?

A Collection is basically a set of objects stored together in memory. The simplest form of a Collection is an Array or a LinkedList. These are simple in the sense that elements are stored in a linear sequence, thus making the iteration over them trivial: when you're processing an element there's no doubt about which one comes next.

Usually, PHP arrays will do the job, no argument about that, but if you want to make your code easily reflect your ideas, relying on specialized structures will be extremely useful.

This article will show you my own Collection implementation for simplicity. Alternatively, since version 7, PHP offers a nice implementation through its DS extension[1]. I recommend you check it out if you're not familiar with it yet.

What is an Iterator?

An Iterator is an Object whose sole responsibility is to loop through the elements contained within a Collection using a particular algorithm. An Iterator should not modify the underlying Collection but act on it as a mere observer. In the case of PHP, Iterators are implemented as part of the SPL extension[2], which I'll be using in the code examples below.

What is the Need for Iterators?

We use Iterators is to separate the storage from the iteration over the elements. In a way, it's an attempt to respect the Single Responsibility Principle (The **S** in SOLID[3]) .

Think of a complex Collection, such as a tree or a graph. In this kind of structure, there are many ways to loop through the elements. You could have, for instance, a method that would return the next element of the Collection in a breadth-first-search, then another in a depth-first-search. And then, later, perhaps you'll need to iterate over the elements in ascending numerical order and, since you're at it, why not implement another one that will traverse the set by creation order.

If you follow this path, you'll soon find yourself changing the Collection class on a regular basis. Definitely not a good idea if you want to keep the bug count low.

That's what Iterators come to solve: they provide a clean way to create new looping mechanisms without changing the code that deals with memory organization and such.

When a new iteration criteria is needed all you have to do is put together a new Iterator class and the rest of your application can stay untouched.

If you want to go deeper into these concepts I recommend reading this great article on Iterators[4].

Different Kinds of Iterators

There are many kinds of Iterators[5] that can be created for a given Collection. Let's take a quick look at some of them:

- AppendIterator[6]. This class allows the combination of Iterators in a sequential style (When one Iterator reaches the end of the underlying collection, the loop continues over the next one until all of them have been consumed).
- InfiniteIterator[7]. This class is used to produce a *circular* iteration: when the end of the loop is reached, the Iterator is automatically rewound, so there's always a next element.
- LimitIterator[8]. This class is used to loop over a fixed sub-set of the Collection's elements.

All of these are examples of Iterators that can be composed with others, allowing for the construction of complex iteration logic. Specific Iterators are designed to work exclusively

[1] *DS extension: https://www.php.net/book.ds*

[2] *SPL extension: https://www.php.net/spl.iterators*

[3] *SOLID: https://www.thinktocode.com/?p=47*

[4] *Iterators: https://phpa.me/refactoring-guru-iterator*

[5] *many kinds of Iterators: https://www.php.net/spl.iterators*

[6] *AppendIterator: https://www.php.net/class.appenditerator*

[7] *InfiniteIterator: https://www.php.net/class.infiniteiterator*

[8] *LimitIterator: https://www.php.net/class.limititerator*

with the file system, such as the DirectoryIterator[9] and its family.

Apart from these, you can create your own if you find the need to put together a more specific iteration algorithm.

Filteriterators

One particularly interesting type of Iterator is the FilterIterator[10].

This class is designed to loop through a subset of the elements of a Collection. Yet, unlike the LimitIterator, the subset is not fixed but dynamic. LimitIterator elements are selected via applying a filtering function to each one and retaining those for which the result is true.

A FilterIterator won't work *directly* on the collection. Instead, it leverages a pre-existing Iterator to access the elements and is another composite Iterator.

Going back to our graph example, say you have a breadth-first-search Iterator implemented, and now you want to process a particular set of those nodes. A nice way to accomplish that would be to design a FilterIterator and then use it together with the graph.

A Somewhat Real Use Case

Recently I got this email from a programmer I've been coaching:

> *Hi Mauro,*
>
> *It's been a while, hope you are doing great?*
>
> *I have a PHP coding challenge that needs to be solved using solid principles, and also I think it involves using the PHP Iterator class, which I'm not familiar with. Can you help me out with it?*
>
> *I hope you can help me out. I really need to understand the best way to solve this.*

Listing 1.

```php
<?php

spl_autoload_register(function (string $className) {
    require_once $className . '.php';
});

$httpReader = new HTTPReader();
$jsonData = file_get_contents($_ENV['OFFERS_ENDPOINT']);

$offerCollection = $httpReader->read($jsonData);

$subcommand = $argv[1];

$productIterator = $offerCollection->getIterator();
$subcommand2FilterMapping = [
    'count_by_price_range' => function (Iterator $productIterator, array $argv) {
        return new PriceFilterIterator($productIterator, $argv[2], $argv[3]);
    },
    'count_by_vendor_id' => function(Iterator $productIterator, array $argv) {
        return new VendorIdFilterIterator($productIterator, intval($argv[2]));
    },
];

// Please note that validation is left out to simplify the example.

echo iterator_count($subcommand2FilterMapping[$argv[1]]($productIterator, $argv));
```

The problem to be solved was expressed like this:

> *Your task is to create a CLI script that will read JSON-based data from a specific endpoint via HTTP. The script will contain several sub-commands to filter and output the loaded data. The commands should be:*
>
> *Find objects by price range (given price_from and price_to as arguments). Find objects by a certain sub-object definition. All given sub-commands should only output the number of objects in stock.*
>
> *Technical Requirements:*
>
> *Implement the ReaderInterface for fetching the JSON HTTP endpoint and thus work with the OfferCollectionInterface and OfferInterface on the loaded data.*

Listing 1 shows a little script[11] I put together to solve the problem and help my student understand the underlying concepts.

I'll go over its main parts to show how the FilterIterator can be implemented in PHP. Let's start with the main script, the file `run.php` shown in Listing 1.

Here you can see how the script:

1. Incorporates a very simple autoloading function by calling `spl_autoload_register`. Doing so is meant to prevent the explicit requirement of supporting classes over the rest of the code.
2. Creates a Product Collection through an `HTTPReader` instance. The HTTPReader's goal is to create a new Collection from the results of applying `json_decode` to the text gathered from the external endpoint.
3. Outputs the number of elements in the Collection that match the criteria specified by a command-line argument by calling the function iterator_count[12] and feeding it with the appropriate Iterator.

[9] *DirectoryIterator:* https://www.php.net/class.directoryiterator

[10] *FilterIterator:* https://www.php.net/class.filteriterator

[11] *a little script:* https://github.com/mchojrin/HTTPJsonFilter

[12] *iterator_count:* https://www.php.net/function.iterator-count

The last point is where I want you to focus your attention. Note how, depending on the specified sub-command, a particular `FilterIterator` sub-class is used on top of the original Iterator.

Listing 2 shows the code for `PriceFilterIterator`.

This class has two methods:

- The constructor
- `accept`

`accept` is the most important part of the class, as it's where the actual filtering logic is stored. In this particular case, the way to determine whether a product should be part of the iteration is checking that its price is within established boundaries.

These boundaries are defined when creating the Iterator's instance. They are the arguments to the constructor method.

Every FilterIterator works this way: the `accept` method is called upon each element yielded by the underlying Iterator and, if the evaluation results in a boolean `true`, the element will be considered part of the loop; otherwise, it will be skipped.

Now look at the `VendorIdFilterIterator` class shown in Listing 3.

The structure is pretty much the same. The only significant difference is the definition of the `accept` method, which, in this case, checks the vendor Id against a fixed value established at construction time.

Since this is just a simulation, this Proof Of Concept project includes a script named output_json.php (Listing 4) that will produce the JSON output expected at the other end.

To test this code — first, spin up a server that can act as a remote endpoint. A simple way to do that is by running

```
php -S localhost:8000 output_json.php.
```

Then, if you issue the command

```
OFFERS_ENDPOINT=http://localhost:8000 php run.php count_by_price_range 200 500
```

you'll get `2` as a response and, if you use the following, you'll get 1.

```
OFFERS_ENDPOINT=http://localhost:8000 php run.php count_by_vendor_id 84
```

Listing 2.

```php
<?php

class PriceFilterIterator extends FilterIterator
{
    private float $priceFrom, $priceTo;

    public function __construct
     Iterator $iterator, float $priceFrom, float $priceTo)
    {
        parent::__construct($iterator);
        $this->priceFrom = $priceFrom;
        $this->priceTo = $priceTo;
    }

    public function accept() : bool
    {
        $currentPrice = parent::current()->getPrice();

        return $currentPrice >= $this->priceFrom &&
            $currentPrice <= $this->priceTo;
    }
}
```

Listing 3.

```php
<?php

class VendorIdFilterIterator extends FilterIterator
{
    private int $vendorId;

    public function __construct
         (Iterator $iterator, int $vendorId)
    {
        parent::__construct($iterator);
        $this->vendorId = $vendorId;
    }

    public function accept() : bool
    {
        $vendor_id = current()->getVendorId();
        return $vendor_id === $this->vendorId;
    }
}
```

Listing 4.

```php
<?php

header('Content-Type: application/json');

echo json_encode([
    [
        'offerId' => 123,
        'productTitle' => 'Coffee machine',
        'vendorId' => 35,
        'price' => 390.4,
    ],
    [
        'offerId' => 124,
        'productTitle' => 'Napkins',
        'vendorId' => 35,
        'price' => 15.5,
    ],
    [
        'offerId' => 125,
        'productTitle' => 'Chair',
        'vendorId' => 84,
        'price' => 230.0,
    ],
]);
```

Listing 5.

```php
<?php

class ProductTitlePrefixFilterIterator extends FilterIterator
{
    private string $prefix;

    public function __construct(Iterator $iterator, string $prefix)
    {
        parent::__construct($iterator);
        $this->prefix = $prefix;
    }

    public function accept() : bool
    {
        return str_starts_with(
            parent::current()->getProductTitle(),
           $this->prefix);
    }
}
```

Listing 6.

```php
$subcommand2FilterMapping = [
    'count_by_price_range' => function (Iterator $productIterator, array $argv) {
        return new PriceFilterIterator($productIterator, $argv[2], $argv[3]);
    },
    'count_by_vendor_id' => function (Iterator $productIterator, array $argv) {
        return new VendorIdFilterIterator($productIterator, intval($argv[2]));
    },

    'count_by_title_prefix' => function (Iterator $productIterator, array $argv) {
        return new ProductTitlePrefixFilterIterator($productIterator, $argv[2]);
    },
];
```

To filter results by any other criteria, such as Product Title, you would have to:

1. Create a new `FilterIterator` class
2. Add a new entry to `$subcommand2FilterMapping`

The first point will end with you having a class such as Listing 5.

And a mapping defined like Listing 6.

Finally, to test it, run `OFFERS_ENDPOINT=http://localhost:8000 php run.php count_by_title_prefix Ch` and you should get `1` as a result.

Of course, the use of these filters is by no means limited to counting elements. Once the sub-set is computed, you can do whatever you need with them. In fact, you can use these filters as a basis for the application of others if that makes sense for the problem you're looking to solve.

Another approach to getting these results would be to rely on array filtering functions instead of FilterIterators, like Listing 7.

The main "problem" with this is that it doesn't comply with the specific requirement of the challenge, so it doesn't really constitute a valid answer for the case under analysis.

Listing 7.

```php
<?php

spl_autoload_register(function (string $className) {
    require_once $className . '.php';
});

$httpReader = new HTTPReader();
$jsonData = file_get_contents($_ENV['OFFERS_ENDPOINT']);

$offerCollection = $httpReader->read($jsonData);

$subcommand = $argv[1];

$subcommand2FilterMapping = [
    'count_by_price_range' => function (OfferCollection $collection, array $argv) {
        return count(array_filter($collection->toArray(), function (Offer $offer) use ($argv) {

            return $offer->getPrice() >= $argv[2] && $offer->getPrice() <= $argv[3];
        }));
    },
];

echo $subcommand2FilterMapping[$argv[1]]($offerCollection, $argv);
```

In a more realistic scenario, this could be a way to get the answers you're looking for. You can see how it is even a less verbose implementation, which could be preferred in some situations.

I have two main concerns about this:

1. The code is not as well organized as the former.
2. It requires the introduction of an *ad hoc* `toArray` method to match the `array_filter` function signature, which makes using a Collection pointless.

The fact that the `array_filter` function forces the first argument to be an array instead of something that can be iterated over is, in my opinion, a language shortcoming. Who knows, perhaps this will be changed in the future, but for the time being, this is the reality.

A third way to tackle this issue would be to provide a way for the Collection to filter itself. That could be interesting, especially if this filtering method got a callback as an argument, thus giving it a lot of flexibility:

```
public function filter(Callable $filter)
{
    return new OfferCollection(
        array_filter(
          $this->offers,
          $filter
        )
    );
}
```

I'd still be a bit skeptical about such an implementation in terms of how that would play out in the long run, though. It seems like we might be putting a little too much responsibility in the Collection class.

Conclusion

In this article, you learned about Collections, Iterators in general, and FilterIterators in particular.

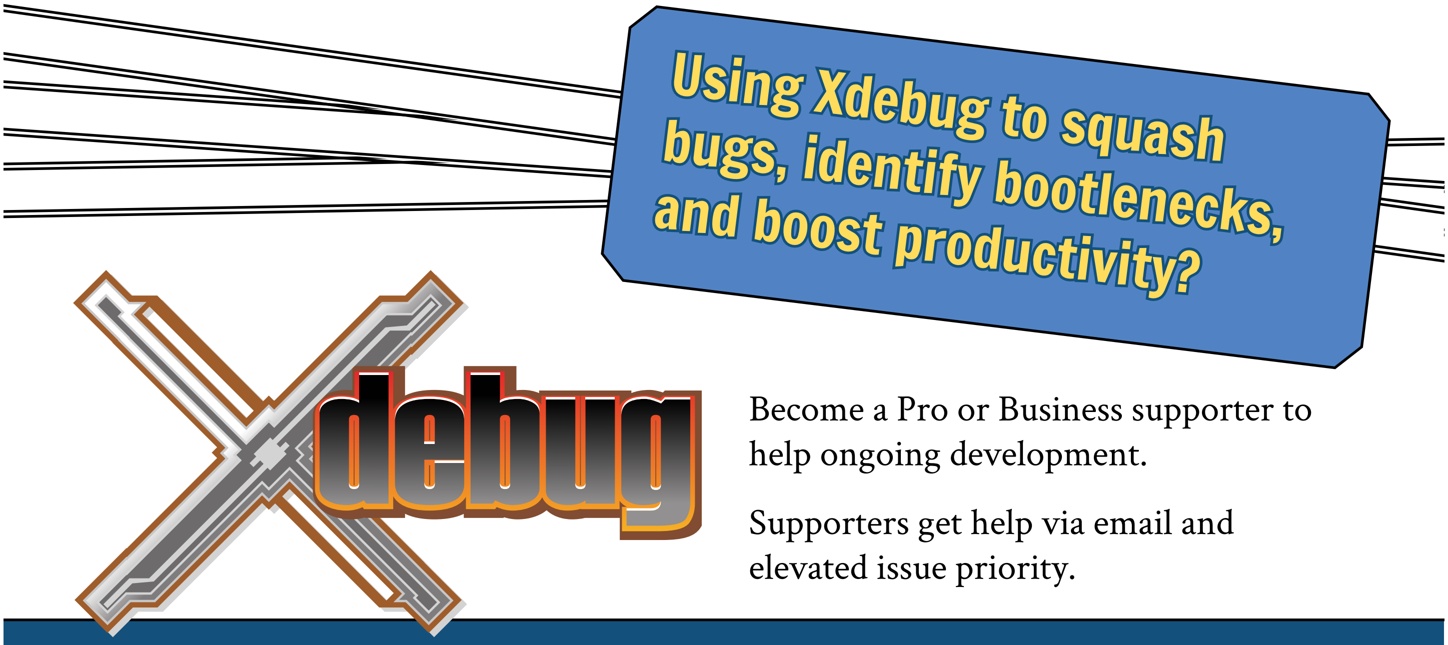
While the usage of Collections and, in general, advanced data structures is not a very common practice when it comes to PHP, many new and interesting tools are available to use to your advantage, especially if you're up-to-date with your PHP version.

I hope I was able to poke your curiosity and show you a bit of how you can structure your code to make it easier to read and maintain over time.

Mauro Chojrin is a PHP Trainer and Consultant. He has been involved in the IT Industry since 1997 in a wide array of positions, including technical support, development, team leadership, IT management, and teaching. He also maintains his blog[13] and YouTube channel[14] where he shares his knowledge with the world.

[13] *his blog:* https://academy.leewayweb.com

[14] *YouTube channel:* https://phpa.me/youtube-leeway



hookrelay

“I wish we had webhooks as awesome as Stripe...”

Get reliable webhook delivery, with debugging and logging, in minutes.

Stripe-quality webhooks for everyone

So you want to add webhooks to your app, and after having worked with the webhooks that Stripe provides, you want yours to be as great as theirs. Well, that's more than just sending a JSON payload to your customer's URL and calling it a day, right?

Show what was sent in a webhook

Deal with slow receiving servers, timeouts, and other errors

Record delivery results so your customers can diagnose what happened when a webhook didn't show up as expected

Guard against SSRF

Add a background job so you don't hold up your web requests (and add extra costs for a background worker if you're deploying to Heroku)

Allow a webhook to be re-sent

Retry webhook deliveries with backoffs (up to days later) to work around errors without overwhelming the receiving server

Deploy webhooks in minutes, not days.

Sign up now, and you can start receiving webhooks from integrations like GitHub, etc., even before you have an app to receive them.

https://phpa.me/hookrelay

How to Hack your Home with a Raspberry Pi - Part 1 - Installing the OS and Configuring the Pi

Ken Marks

In September of 2017, I gave a talk called: 'Hack your Home with a Raspberry Pi' at the Madison PHP Conference. It was a half-day tutorial, walking participants through the process of configuring a Raspberry Pi, installing a LAMP stack on it, connecting a sensor, and ultimately having it send a text message to their phone.

Since then, I have been running this as a lab for my Advanced PHP class I teach at our local community college, Madison College.

One of the main reasons I gave this tutorial was to dispel the myth of the Internet of Things (IoT) being difficult. Sometimes people get scared away from writing code that talks to small embedded devices (i.e., sensors connected to Raspberry Pis or Arduino boards). My goal is to show you how easy it is to create a device that reports sensor data that you can get on a network. Along the way, you will see that deploying an application to a Raspberry Pi is just as simple as deploying to any Linux-based LAMP stack! [should I say, it's as easy as Pi :-)]

This article is the first of several in a series. I know php[architect] Magazine is a publication dedicated to PHP development, but I hope it doesn't scare you that not all the code you will be deploying/using in this series will be PHP 😱. In addition to PHP (and SQL), we will be using C/C++ and JavaScript. We will be creating a couple of UNIX services as well. So come along on this journey with me as I show you how to hack your home with a Raspberry Pi!

The goals for the whole series (can we call it a project?) are:

- Installing the Raspberry Pi operating system
- Configuring your Pi
- Installing a LAMP Stack on your Pi
- Connecting an accelerometer sensor to your Pi
- Logging data from an accelerometer to a database
- Creating and installing a Unix Service on your Pi
- Building a Web Service for querying the accelerometer data
- Building a plotting application to display the accelerometer data
- Sending a text message to your phone using SendMail

Wow! That's a pretty big list which is why it will take several articles to cover.

In this article, I will cover:

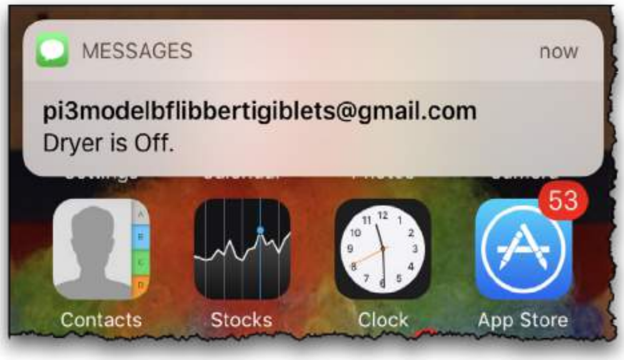
- Installing the Raspberry Pi operating system
- Configuring your Pi

It starts with a story...

My initial idea and motivation for this series (project?) starts with a story about our old clothes dryer, which quit working one day. I was tasked with replacing it. Everything was great, and I got kudos for getting a decent dryer. However, there was one annoying flaw with this new dryer. We do our laundry in the basement, and this new dryer chimes at us — very quietly — when the clothes are done. Our old dryer buzzed its bloody head off when the dry cycle was complete, and I swear you could hear it from the street. Of course, being the forward-thinking Rube Goldberg kind of person I am, I said, "I can create an App for that!" So, since I was

New Samsung Dryer with Pi

Text Message from Pi that Dryer is Off


MESSAGES now

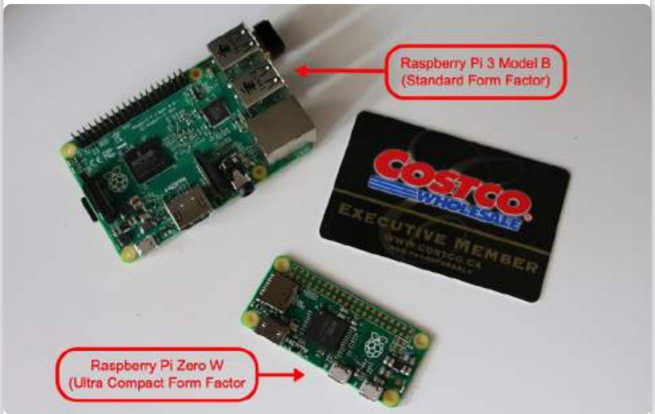
pi3modelbflibbertigiblets@gmail.com
Dryer is Off.

fairly familiar with Raspberry Pis and am always looking for an excuse to buy new geek toys, I set my sights on using the smallest Raspberry Pi on the market for my project.

List of essential components

The smallest Raspberry Pi I can use for this project is the *Raspberry Pi Zero W*:

Raspberry Pi Form Factors



If you use a *Raspberry Pi Zero W*, keep in mind that it only supports 2.4GHz wireless. So, if your wireless router only supports 5GHz, you might want to use another model of the Pi. The Pi 3 B+ and 4 B support 2.4 and 5GHz wireless ethernet. The upside of using the Pi 3 B+ or 4 B is that you won't have to do any soldering.

Do I really need to learn how to solder?

> *What did the soldering iron say to the capacitor? Go flux yourself!*

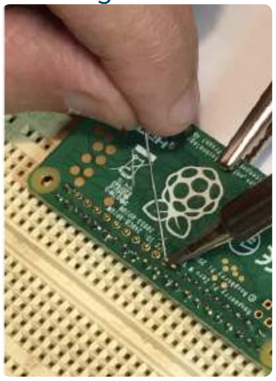
All Raspberry Pis have a General Purpose Input Output (GPIO) bus which is a duel-row male 40 pin header used to connect sensors and other devices to the Pi:

Raspberry Pi 40pin header

Later on, we'll be connecting an accelerometer sensor to the GPIO bus of the Pi. Because I'm using the Pi Zero W, I'll have to solder on a duel-row male 40 pin header onto the board to expose the GPIO bus pins since the Pi Zero W board does not come with it pre-populated.

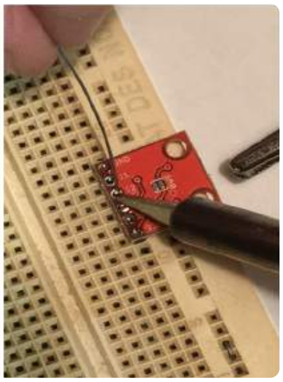
I've included a few photos of me soldering the header onto the Pi and 6 pins onto one version of the accelerometer sensor.

Soldering Pins



Head Soldered

Soldering Pins on Accelerometer



Pi Zero W in Case

Again, you can use any Raspberry Pi, but I thought it would be really cool to use a Raspberry Pi half the size of a credit card. I recommend you use a Raspberry Pi with wireless ethernet built-in since that will be the configuration I will document here.

Okay, let's start a numbered list of the five items we will need:

1. **Your computer**

   You will need to have a computer connected to the internet to download the Raspberry Pi Operating System. Ideally, you will want to connect to your Pi via SSH from your computer. I'm going to assume you are running some flavor of Unix (Ubuntu, MacOS, etc.). However, there is plenty of information on the internet for communicating with your Raspberry Pi if you have a Windows PC.

2. **Your home wireless router**

   I'm not going to go into how you configure this, as everyone is different. I will assume you have one and go through a generic method for connecting to it.

3. **Raspberry Pi**

   I recommend getting a kit that includes a case, a power supply, and an SD card. I get mine from CanaKit[1], but you can get one from Amazon or any other Raspberry Pi reseller).

[1] *CanaKit: https://www.canakit.com*

Duel row 40 pin Male Header

Here is a list of things associated with your Pi you might want to have

- duel row 40 pin header (if you buy a Pi Zero W). You can get this from Amazon[2].
- soldering iron and solder (if you buy a Pi Zero W)
- case
- power supply for Pi
- SD card at least 8GB. (It's probably not worth getting one less than 16GB. I would recommend getting a card rated at UHS-1 or better. Most are).
- SD card reader. Many Raspberry Pi kits come with one that plugs into the USB port of your computer.
- HDMI cable (if you want to configure your Pi with a keyboard and monitor)
- keyboard (if you want to configure your Pi with a keyboard and monitor)
- monitor (if you want to configure your Pi with a keyboard and monitor)

4. **Accelerometer**

There are a lot of different accelerometer sensors available. I wanted one that uses the I2C bus for this project — more on that later. There is a small C program we will be using that communicates with an *MMA8452Q* triple-axis accelerometer. There are two slightly different board types:

Figure 1. MMA8452 from Sparkfun

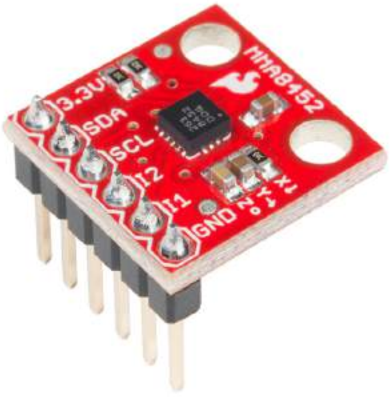
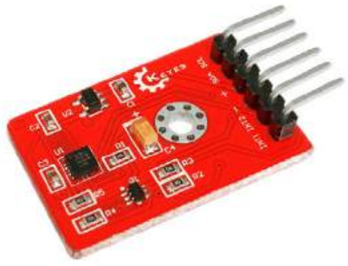


Figure 2. Keyes MMA8452Q from eBay



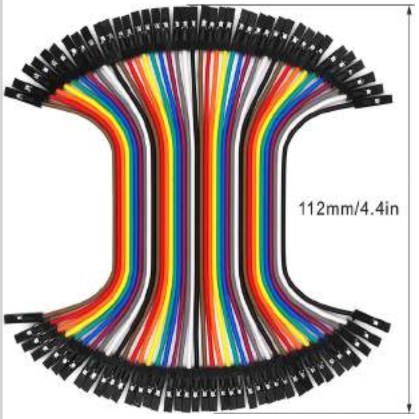
- The MMA8452 shown in Figure 1 can be purchased from Sparkfun[3].
- And the Keyes MMA8452Q, which can be purchased from eBay[4]. See Figure 2.

The nice thing about the Keyes version is that it comes pre-populated with the 6 pins already soldered in

5. **female to female 4 pin ribbon cable to connect the accelerometer to Pi**

These typically come as a 40 pin ribbon cable, and you peel off 4 wires to connect the accelerometer to the GPIO bus of the Pi. You can purchase this from Amazon[5].

40 Pin Ribbon Cable



Configuring your Raspberry Pi

Now that we have all of our essential equipment and components, we need to install an operating system and configure our Raspberry Pi.

Downloading the Raspberry Pi OS image

First, we need to get an image of the Raspberry Pi Operating System (OS) and get it onto our SD card. The best way to do that is to download the *Raspberry Pi Imager*[6]. There are links to download the imager for MacOS, Windows, or Ubuntu on the page. You can even use the *apt* package manager within a UNIX terminal window to install the *Raspberry Pi Imager*. After downloading the Imager package for your OS, follow the installation instructions.

Plug your SD card reader into your computer and insert the SD card you will be using for your Raspberry Pi.

[2] *Amazon:* *https://phpa.me/amazon-40pin-duel*

[3] *Sparkfun:* *https://www.sparkfun.com/products/13926*

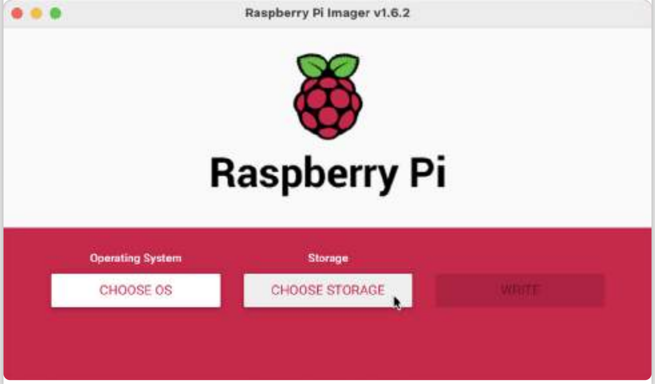
[4] *eBay: https://phpa.me/ebay-accelerometer*

[5] *Amazon:* *https://phpa.me/amazon-fem-jumper-40*

[6] *Raspberry Pi Imager:* *https://www.raspberrypi.com/software/*

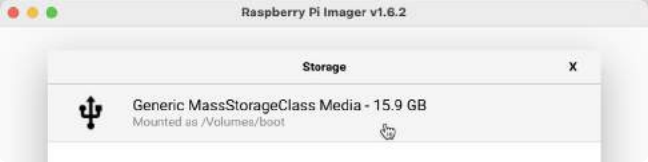
Launch the Raspberry Pi Imager, and you should see the main screen of the *Raspberry Pi Imager*:

Main Screen of Raspberry Pi Imager



Select *CHOOSE STORAGE,* and a *Storage* dialog should be displayed:

Raspberry Pi Imager Storage Dialog

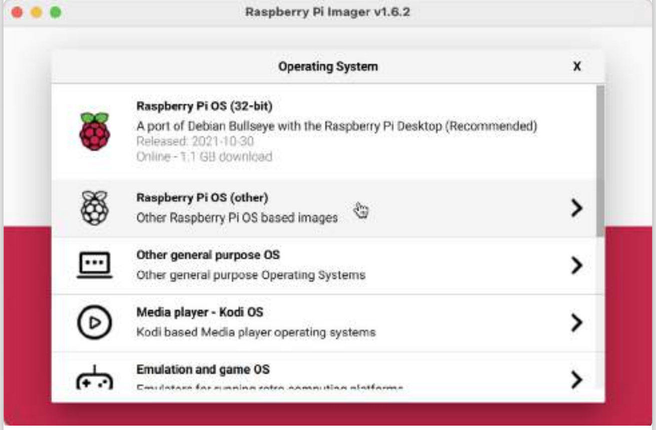


Generic MassStorageClass Media - 15.9 GB
Mounted as /Volumes/boot

Select your SD card - typically listed as *Generic MassStorageClass Media* and its size.

Now select *CHOOSE OS,* and the *Operating System* dialog should be displayed:

Pi Imager OS Dialog: Choose Raspberry Pi OS (other)



Raspberry Pi OS (32-bit)
A port of Debian Bullseye with the Raspberry Pi Desktop (Recommended)
Released: 2021-10-30
Online - 1.1 GB download

Raspberry Pi OS (other)
Other Raspberry Pi OS based images

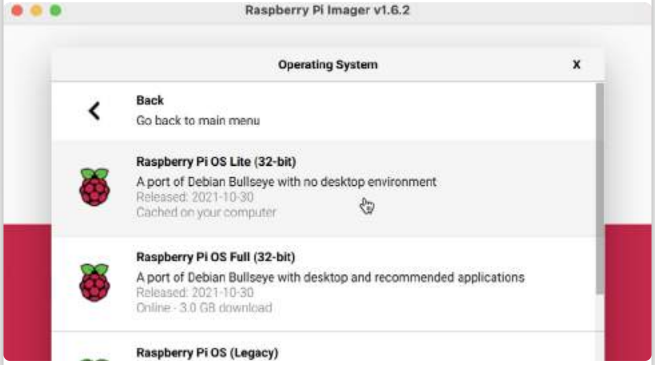
Other general purpose OS
Other general purpose Operating Systems

Media player - Kodi OS
Kodi based Media player operating systems

Emulation and game OS
Emulators for running retro-computing platforms

Select *Raspberry Pi OS (other),* and the *Operating System* dialog will display a list of Raspberry Pi operating systems to choose from:

Pi Imager OS Dialog: Choose Raspberry Pi OS Lite (32-bit)



Back
Go back to main menu

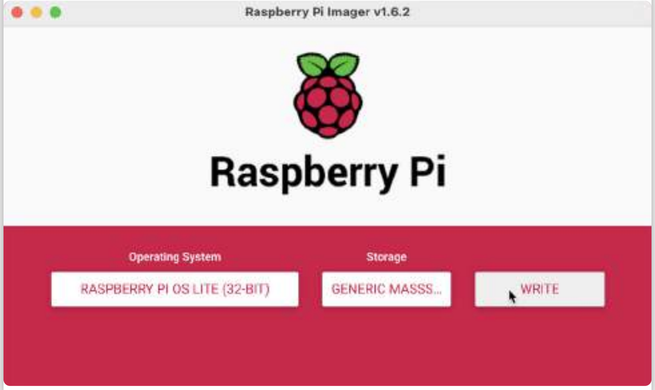
Raspberry Pi OS Lite (32-bit)
A port of Debian Bullseye with no desktop environment
Released: 2021-10-30
Cached on your computer

Raspberry Pi OS Full (32-bit)
A port of Debian Bullseye with desktop and recommended applications
Released: 2021-10-30
Online - 3.0 GB download

Raspberry Pi OS (Legacy)

Select *Raspberry Pi OS Lite (32-bit)* - a Linux server installation that will minimize the amount of CPU utilization since we're not doing a full desktop installation. You will now be returned to the main screen of the *Raspberry Pi Imager*:

Main Screen of Raspberry Pi Imager: WRITE



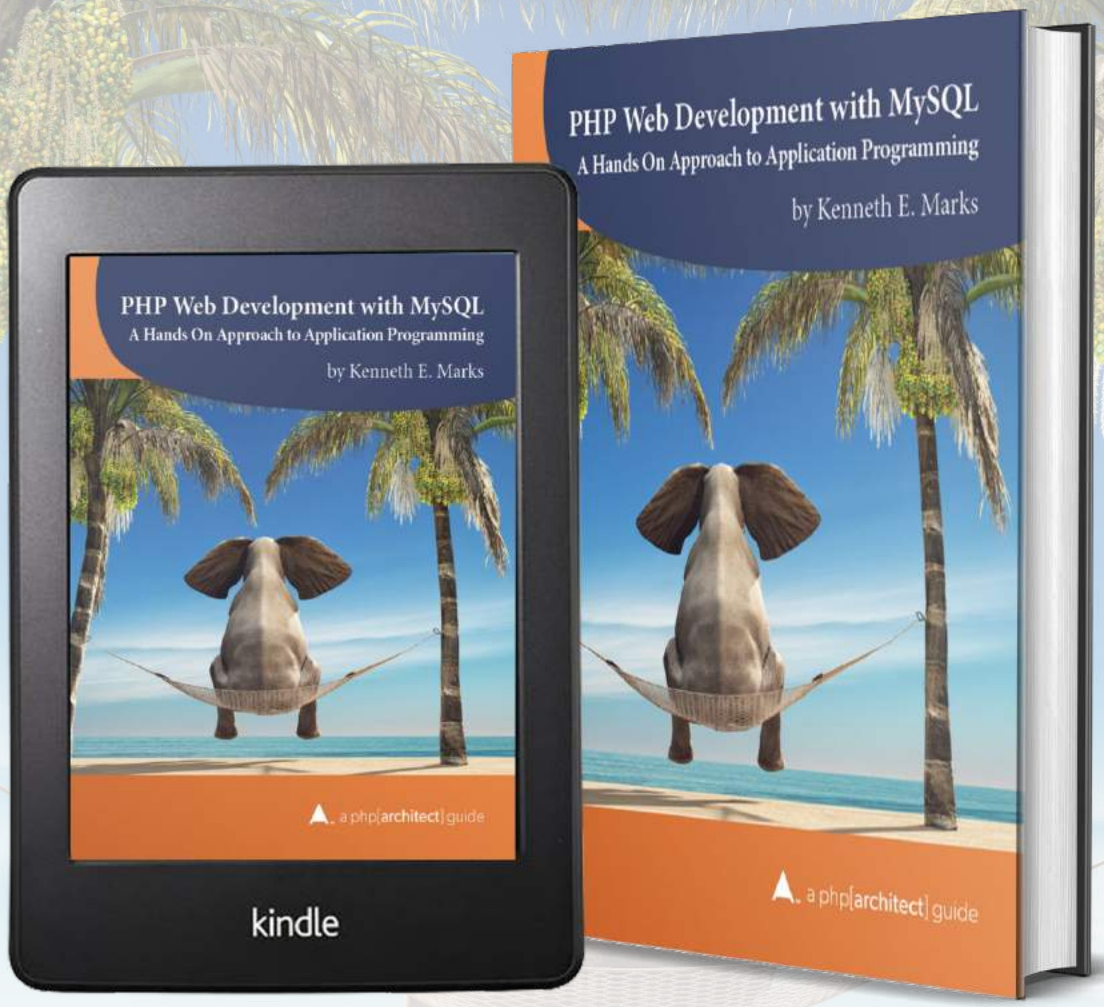
Now select *WRITE* to write the image to the SD card. You will now see a *Warning* dialog, letting you know all existing data on your SD card will be erased and ensure you want to continue. Select *YES.*

You will usually see a *Permissions* dialog pop up asking you to enter your admin password for your computer before you can write to your SD card.

After entering your credentials, the Raspberry Pi Imager will start writing the OS to your SD card. When the write operation is complete, a *Write Successful* dialog should be displayed. Select *CONTINUE,* and quit the *Raspberry Pi Imager* application. You can now remove your SD card from your SD card reader.

Now we need to configure our Raspberry Pi. There are two ways you can do this:

- Headless configuration
- Direct configuration


PHP Web Development with MySQL
A Hands On Approach to Application Programming
by Kenneth E. Marks
a php[architect] guide
kindle

I will first show you the headless configuration as I find it's the simplest. Then I will show you the direct configuration.

Headless configuration

So, let's say you have your Raspberry Pi, and you don't have a keyboard or extra monitor laying about, or like me, you forgot all of that (and the magic HDMI dongle that came with your Raspberry Pi Zero W kit)! Your only option at this point is to perform a headless configuration on your Pi. That's a good thing because it's usually the easiest option.

To create a headless configuration, we need to modify the contents of the SD card so that when the Raspberry Pi boots, it will:

- Allow us to SSH into the Raspberry Pi
- Connect the Pi to our Wi-Fi router

Re-insert your SD card back into your SD card reader.

Open up a terminal window and navigate to the root folder of your SD card. I'm on a Mac and will go to `/Volumes/boot`:

To enable SSH, we need to create an empty file called `ssh` in the `boot` folder of the SD card. Type the following command in the terminal window:

```
touch /Volumes/boot/ssh
```

Next, we need to add our Wi-Fi network information. For that, we need to create a file called `wpa_supplicant.conf` and put it in the `boot` folder of the SD card. Type the following command in the terminal window to create the `wpa_supplicant.conf` file:

```
touch /Volumes/boot/wpa_supplicant.conf
```

Next, we need to edit the `wpa_supplicant.conf` file and add our Wi-Fi network SSID and password. Paste the following into the editor, and update the country code as needed, and your network SSID and password:

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="NETWORK-NAME"
    psk="NETWORK-PASSWORD"
}
```

Save the file and exit your editor.

Eject the SD card and insert it into the Raspberry Pi

Mitch Allen has a complete set of instructions[7] for how to create a headless configuration for your Raspberry Pi that covers all the major operating systems of your host computer.

Direct configuration

Direct configuration requires connecting a USB keyboard and HDMI monitor up to your Pi. Some Raspberry Pis require dongles to connect a standard HDMI cable and USB keyboard. The Pi Zero even comes with a USB dongle.

Insert the SD card into your Raspberry Pi

Connect your Pi to a monitor and a keyboard.

Connect your power supply to your Pi and plug the power supply into a mains A/C wall socket.

After the Raspberry Pi boots up, log in with the user `pi` and the default password of `raspberry`.

Next, we need to add our Wi-Fi network information. For that, we need to create a file called `wpa_supplicant.conf` and put it into the `/etc/wpa_supplicant/` folder. We will be using the built-in `nano` editor to create the `wpa_supplicant.conf` file. Since this file requires `root` access, we have to create this file as *superuser*. Type the following command in the command-line terminal:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Now let's edit the `wpa_supplicant.conf` file and add our Wi-Fi network SSID and password. Paste the following into the editor, and update the country code as needed, and your network SSID and password:

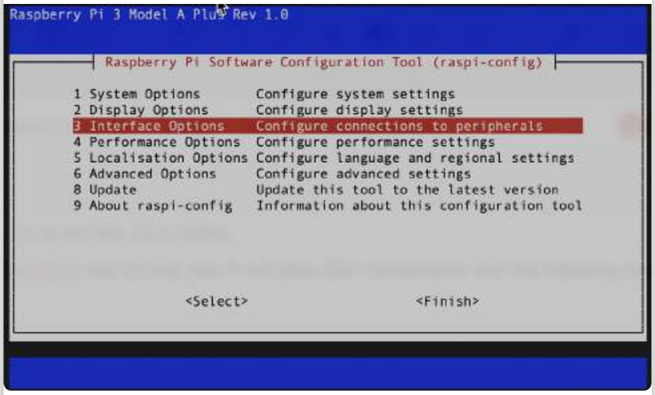
```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="NETWORK-NAME"
    psk="NETWORK-PASSWORD"
}
```

Type `^O` to write out the file and `^X` to exit the `nano` editor.

Next, we're going to run the `raspi-config` tool so that your Pi will allow SSH connections. Run the following command as *superuser* to display the main screen of the `raspi-config` tool.

```
sudo raspi-config
```

Main Screen: Select Interface Options

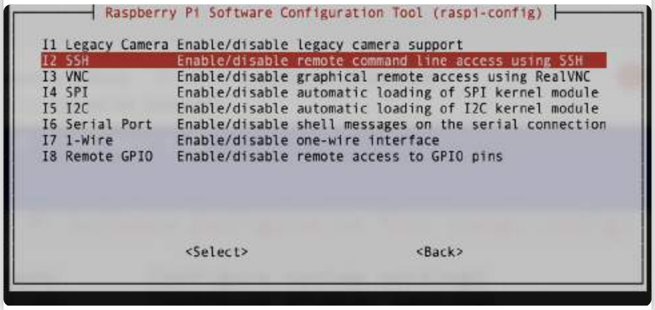


```
Raspberry Pi 3 Model A Plus Rev 1.0

        Raspberry Pi Software Configuration Tool (raspi-config)

   1 System Options       Configure system settings
   2 Display Options      Configure display settings
   3 Interface Options    Configure connections to peripherals
   4 Performance Options  Configure performance settings
   5 Localisation Options Configure language and regional settings
   6 Advanced Options     Configure advanced settings
   8 Update               Update this tool to the latest version
   9 About raspi-config   Information about this configuration tool

              <Select>                    <Finish>
```

[7] *complete set of instructions: https://phpa.me/desertbot-pi-wifi-setup*

Using the *arrow* keys, highlight `3 Interface Options` and press `Enter`, and the *Interface Options* screen will be displayed:

Interface Options: Select SSH



```
┤ Raspberry Pi Software Configuration Tool (raspi-config) ├

 I1 Legacy Camera Enable/disable legacy camera support
 I2 SSH           Enable/disable remote command line access using SSH
 I3 VNC           Enable/disable graphical remote access using RealVNC
 I4 SPI           Enable/disable automatic loading of SPI kernel module
 I5 I2C           Enable/disable automatic loading of I2C kernel module
 I6 Serial Port   Enable/disable shell messages on the serial connection
 I7 1-Wire        Enable/disable one-wire interface
 I8 Remote GPIO   Enable/disable remote access to GPIO pins

             <Select>                          <Back>
```

Using the *arrow* keys, highlight `I2 SSH` and press `Enter`, and a dialog asking, “Would you like the SSH server to be enabled?” will pop up. Select `Yes` and press `Enter`.

Press `Enter` after the dialog, “The SSH server is enabled” pops up, and you will be returned to the *main* screen of the `raspi-config` tool. Using the tab key, select `<Finish>` and press `Enter`.

Type the following into the command-line terminal to shut down the Raspberry Pi:

```
sudo shutdown -h now
```

SSH into your Pi

Now connect the power supply to your Raspberry Pi and plug it into A/C mains.

After a few minutes, type the following into a terminal to remove any previous SSH keys generated for the host `raspberrypi.local`:

```
ssh-keygen -R raspberrypi.local
```

If you receive a *host not found* error, that’s okay. That just means you didn’t have any previous SSH keys generated for `raspberrypi.local`.

Next, SSH into your Pi with the user `pi` by typing the following command into the terminal:

```
ssh pi@raspberrypi.local
```

Type in the default password of `raspberry`. If all goes well, you should be logged in to your Raspberry Pi.

Run `raspi-config`

At this point, you want to configure your Raspberry Pi for your locale settings, along with changing the password. To do this, we will run the `raspi-config` program as *superuser*.

Assuming you are logged into your Pi using SSH, type the following command into the terminal:

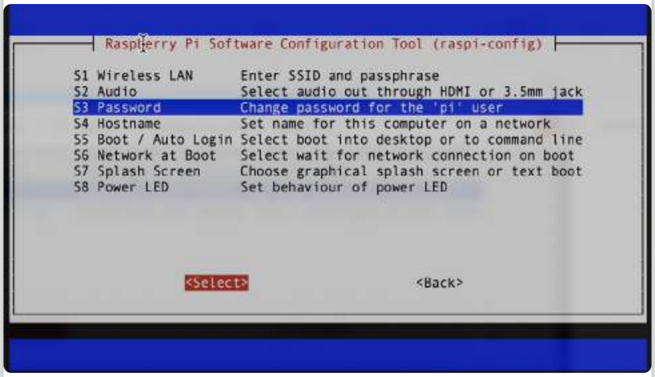
```
sudo raspi-config
```

You should see the *raspi-config* main screen displayed again.

We first need to make sure we have the latest version of the `raspi-config` tool. Use the *arrow* keys to highlight `8 Update` and tab over to the `<Select>` option and press `Enter`.

`raspi-config` will be updated to the latest version, and the *raspi-config* main screen will be redisplayed with the *System Options* highlighted. Tab over to the `<Select>` option and press `Enter`. The following *System Options* will be displayed, allowing you to change your password:

System Options: Select Password



```
┤ Raspberry Pi Software Configuration Tool (raspi-config) ├

 S1 Wireless LAN     Enter SSID and passphrase
 S2 Audio            Select audio out through HDMI or 3.5mm jack
 S3 Password         Change password for the 'pi' user
 S4 Hostname         Set name for this computer on a network
 S5 Boot / Auto Login Select boot into desktop or to command line
 S6 Network at Boot  Select wait for network connection on boot
 S7 Splash Screen    Choose graphical splash screen or text boot
 S8 Power LED        Set behaviour of power LED

             <Select>                          <Back>
```

Use the *arrow* keys to highlight `S3 Password` and tab over to the `<Select>` option and press `Enter`. Follow the instructions to enter your new password. For purposes of this series, I’ll set mine to `pizerow`.

After changing your password, you will be taken back to the main screen of the `raspi-config` tool. Next, we need to modify the *Interface Options* to allow our Pi to use the GPIO bus and talk to our accelerometer sensor we will be connecting to it later on. Use the *arrow* keys to highlight `3 Interface Options` and tab over to the `<Select>` option and press `Enter`.

We specifically want to enable the *I2C* bus, as that is the protocol our accelerometer will use to communicate with the Pi. Use the *arrow* keys to highlight `I5 I2C` and tab over to the `<Select>` option and press `Enter`.

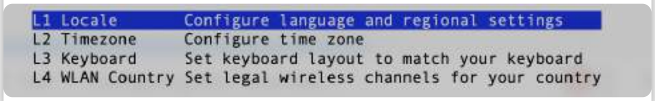
When you see this screen asking, “Would you like the ARM I2C interface to be enabled?”, tab over to `<Yes>` and press `Enter`.

Then select `<Ok>`, and you will be taken back to the main menu of the `raspi-config` tool.

Finally, we will change our localization options. Select `5 Localisation Options` in the Main Screen.

We need to configure our *Locale*, *Timezone*, and *WLAN Country*:

Raspi-config Localization Options: Select Locale

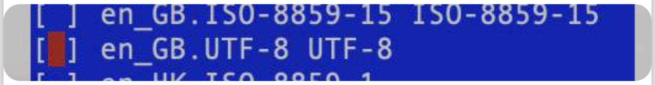


```
L1 Locale       Configure language and regional settings
L2 Timezone     Configure time zone
L3 Keyboard     Set keyboard layout to match your keyboard
L4 WLAN Country Set legal wireless channels for your country
```

First the *Locale*.

Using the *arrow* keys, I will scroll down until I find the selection of `en_GB.UTF-8 UTF-8`. Since I am not in the United Kingdom, I will disable `en_GB.UTF-8 UTF-8` by pressing the *spacebar*:

Configuring Locales: Disable en_GB.UTF-8

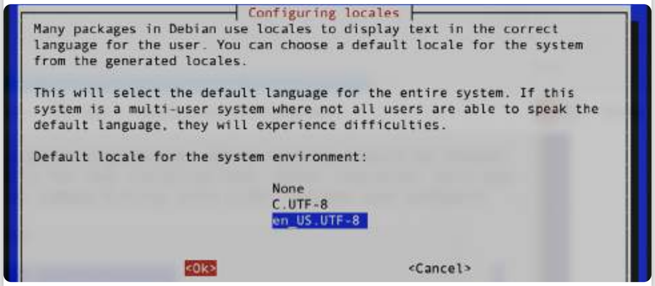


```
[ ] en_GB.ISO-8859-15 ISO-8859-15
[ ] en_GB.UTF-8 UTF-8
```

Which will make the selection (denoted by the asterisk) disappear. Then I will continue to use the *arrow* keys until I find my locale. I'll again use the *spacebar* to select my locale, which is `en_US.UTF-8 UTF-8` since I live in the US.

Then I'll tab over, and select `<Ok>` and I should see a screen asking me to pick the default locale for the system:

Configuring Locales: Select en_US.UTF-8 as Default



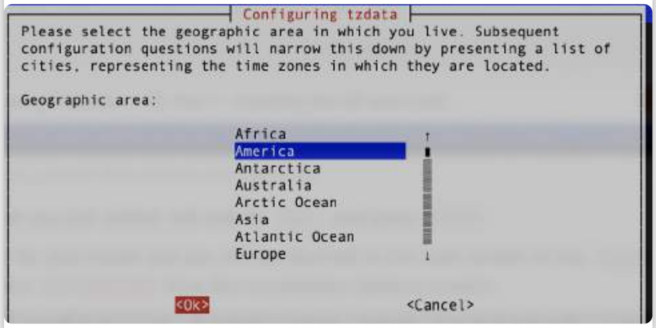
```
                     Configuring locales
Many packages in Debian use locales to display text in the correct
language for the user. You can choose a default locale for the system
from the generated locales.

This will select the default language for the entire system. If this
system is a multi-user system where not all users are able to speak the
default language, they will experience difficulties.

Default locale for the system environment:

                              None
                              C.UTF-8
                              en_US.UTF-8

          <Ok>                                  <Cancel>
```

Use the *arrow* keys to select the *locale* you just added, tab over to `<Ok>`, and press `Enter`.

The necessary files will be generated for your *locale,* and you will be returned to the *main* screen of the `raspi-config` tool. Select the *Localization* option again and select `L2 Time-zone` from the *Localization Options* screen.

I'm in *America*, so that's what I'm selecting for my *Geographic area* from the *Configuring tzdata* screen:

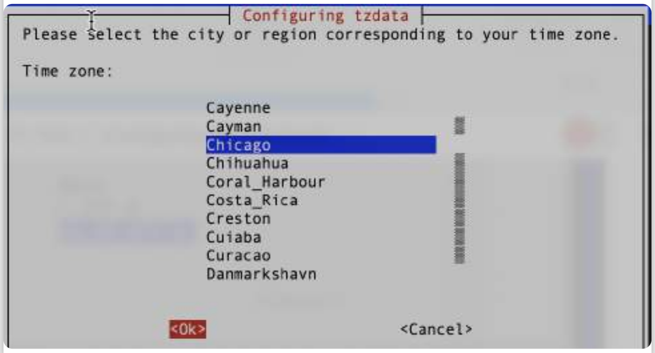
Configuring tzdata - Geographic area : Select America



```
                      Configuring tzdata
Please select the geographic area in which you live. Subsequent
configuration questions will narrow this down by presenting a list of
cities, representing the time zones in which they are located.

Geographic area:

                    Africa
                    America
                    Antarctica
                    Australia
                    Arctic Ocean
                    Asia
                    Atlantic Ocean
                    Europe

          <Ok>                                  <Cancel>
```

Doing so brings me to my *Time zone* selection. I am in Madison, Wisconsin, so I'll pick *Chicago* on the *Configuring tzdata* screen:

Configuring tzdata - Time zone: Select Chicago



```
                       Configuring tzdata
Please select the city or region corresponding to your time zone.

Time zone:

                          Cayenne
                          Cayman
                          Chicago
                          Chihuahua
                          Coral_Harbour
                          Costa_Rica
                          Creston
                          Cuiaba
                          Curacao
                          Danmarkshavn

             <Ok>                                 <Cancel>
```

The necessary settings will be changed for your time zone, and you will be returned to the main screen of the `raspi-config` tool. Select the *Localization* option again and select `L4 WLAN Country`. Again, I'm in the United States, so that is what I'll select.

The necessary settings will be changed for your WLAN Country, and you will be returned to the main screen of the `raspi-config` tool. Tab over to `<Finish>` and press `Enter`.

You will be asked if you would like to reboot the Pi. Select `<Yes>`. You will be logged out, and your SSH connection closed out of your terminal while the Raspberry Pi reboots.

Wait about a minute and log back into your Pi from your terminal window on your host computer. You will need to use your new password now to log in.

Conclusion

You now have a Raspberry Pi that is fully configured! This concludes our first installment of *How to Hack your Home with a Raspberry Pi*. In the next installment, we will install a full LAMP stack. Until next month, I hope you have fun playing around with your Pi!

Ken Marks has been working in his dream job as a Programming Instructor at Madison College in Madison, Wisconsin, teaching PHP web development using MySQL since 2012. Prior to teaching, Ken worked as a software engineer for more than 20 years, mainly developing medical device software. Ken is actively involved in the PHP community, speaking and teaching at conferences. @FlibertiGiblets

Five Strategies for Becoming an Effective Mentor

Olivia Liddell

One of the most pivotal moments in my career happened about ten years ago. I was a few weeks away from finishing the teacher residency program that I had been a part of for the past year. As part of the final phase of the program, the director gave me a call to inform me of my permanent classroom placement for the following year.

I was hoping to get placed at a school with all of my friends who had been in my residency cohort so that we could continue to support each other in our first steps as full-time Chicago Public Schools teachers. However, I soon realized that the director had different plans for me. As we spoke on the phone, he explained that while the rest of my friends would be together in a cluster of schools, I was being assigned to a different school where I didn't yet know anyone. He decided to place me there because he and the other residency program leaders saw a lot of potential in me to succeed not only as a teacher but also as a mentor. They assigned me to that school so that they could fast-track me on the path to start mentoring new teacher trainees within a couple of years.

At that point in my career, I had had several mentors (both formal and informal), but I had never really mentored anyone myself. As the program director continued to describe how excited he was for me to go down this path, all I could do was nod my head, try not to panic, and begin figuring a way to get out of it.

I'd love to be able to say that what happened next was that I faced my fears head-on, went to my assigned school with an open mind, and fully embraced the challenge of becoming a mentor. But that didn't happen. Over the next month, I found an opening at a different school in the network that was actually a better fit for my teaching interests (middle school Arabic) and got permission to transfer there.

However, the topic of mentoring continued to linger in my mind for years to come. As I looked back on that conversation with my program director, I eventually came to realize that part of my reluctance to step into the role of a mentor came from the fact that I didn't know how to mentor anyone. I didn't want to take on the task of mentoring someone else and trying to help them shape their career when I actually had no idea how to do that.

But then, as I reflected back on the mentors that I've had over the years, I started taking small steps to identify specific strategies that were most helpful for me as a mentee and also approaches that I felt were ineffective and could have been better. I used this as the basis of learning about mentoring and continued to take training and do more research to improve my skills. Mentoring has become something that I am incredibly passionate about because of how much potential it has to help others discover new ideas and make advancements in their careers.

So, how do you become an effective mentor? This article explores five key strategies to help you get started with mentoring or improve your already existing mentoring practice. As you read this article, I encourage you to reflect on your past experiences with mentoring (both as a mentor and as a mentee) and think about which strategies most resonate with you.

Strategy #1: Identify your strengths.

One of the most common misconceptions that I have heard from people who are interested in mentoring but afraid to get started is that they feel they don't have anything to offer to anyone else. To me, one of the best things about mentoring is the opportunity for you to choose exactly which parts of yourself you want to highlight when you are choosing to help someone else. In other words, you don't have to be an expert in everything to be able to provide value to others.

Take a moment to think about some of the places where you might have seen mentoring in pop culture. Two of the best examples of this that come to mind are the reality TV shows *Shark Tank* and *The Voice*. On both of these shows, contestants get to choose which panelists they want as mentors—for investing in their business or coaching them to win the singing competition. This choice is typically based on some level of compatibility between the contestants' interests and panelists' areas of expertise. For instance, someone launching a sports-themed business on *Shark Tank* would most likely choose to be mentored by Mark Cuban (owner of the Dallas Mavericks basketball team) over other panelists whose business interests are in other areas.

Which leads to the question: What are some of the reasons why someone might choose *you* to be their mentor? Uncovering your strengths can help you determine which of your strengths and skills might be the most useful for you to put forward and offer to potential mentees.

I have found a lot of value throughout my career in focusing on specific areas that I can help mentees with, rather than casting a much broader net. To give you an idea of how this might look for you, some of the areas that I've focused on recently with mentees include public speaking, transitioning from classroom teaching into other careers, and developing strategies to help prepare for certification exams.

If you find yourself getting stuck with identifying your strengths, consider taking some time to complete the Reflected Best Self Exercise. This exercise is a tool to help you uncover and reflect on the times when you were at your best, based on the feedback that you receive from people who know you best.

1. **Select 3-5 respondents and ask them for feedback.** Specifically, these respondents will be providing insights on when they have seen you at your best. Ideally, you'll want to choose people who have known you throughout different stages of your career to increase the likelihood of receiving diverse responses.

2. **Identify patterns in their responses.** When I first completed this exercise, I saw a common theme—people believed I was excellent at public speaking. At the time, the idea of public speaking terrified me very much but knowing that other people saw it as a strength helped encourage me to take intentional steps to begin focusing more on it.

3. **Compose your self-portrait.** This step is intended to be a written self-portrait about how you view yourself and your potential, but if you are artistic, feel free to do a visual self-portrait as well! In this step of the exercise, you are taking what others have shared about you and using the feedback to create a vision of where you see yourself going next.

Strategy #2: Determine the type of mentoring that you want to do.

Earlier in my career, when I was afraid to become a mentor, part of my resistance was due to the fact that I was unaware of the different types of mentoring that I could potentially get involved with. From my perspective, I saw a mentor as someone who was completely responsible for guiding their mentee throughout their career journey. Mentoring seemed like an incredibly daunting task, and not wanting to possibly ruin someone else's career, I decided that it wasn't for me.

I wish I had known that mentoring could occur in a wide variety of forms. Today in my experience as a mentor, there are some mentees that I have worked with, in a formal capacity over several months, and others that I have partnered with, more informally, on an as-needed basis.

After you have identified your strengths, and before you jump in to start mentoring someone, it's essential to identify which type of mentoring might be the best fit for you. The two types that I'll focus on in this article are informal and formal mentoring.

With informal mentoring, goals are typically unspecified. Mentors and mentees usually self-select each other, and outcomes are generally not tracked or measured. You might find an informal mentee through a tech meetup group, a department social hour, or even just by chatting with colleagues in the break room.

In an informal mentoring relationship, it is essential for you and your mentee to be proactive when seeking opportunities to connect. Although you might not be officially establishing goals with your mentee or tracking their progress, finding opportunities to connect and extend the relationship can help you ensure that they can still benefit from the time they spend interacting with you.

As you work with mentees in an informal context, it can be helpful to advise them on the best ways to leverage their relationship with you. For example, I typically advise my informal mentees to reach out to me with specific questions and an overview of the steps they have already taken to solve a problem. This way, we can make the most of the time that we spend together, whether through a live meeting or through email exchanges.

With formal mentoring, mentors and mentees are often paired based on some type of compatibility. Goals are established, and outcomes are tracked and measured. Formal mentoring relationships might be developed through company-wide initiatives led by Human Resources. For example, a company might sponsor a program for their interns to formally be paired with staff members for a mentoring relationship throughout the duration of the internship.

If you are considering getting involved with formal mentoring, a key first step is defining the purpose and timeline. Work backward from the question, "By the end of this mentoring relationship, what should the mentees be able to do?" For example, will you focus on helping your mentee improve their technical skills or prepare for a new leadership position? Will you be working with them for a period of 6 months or an entire year? Your answers to these questions can help you determine how to best structure the mentoring relationship and also establish clear expectations for everyone involved.

Next, you should determine exactly how the effectiveness of the mentoring program will be measured. Metrics for this might include competency assessments, employee retention rates, improved job performance rates, etc. Including metrics in a formal mentoring program can help you obtain data to show which parts of the program were most effective, which can be particularly useful to know if you need to grow and scale the program over time, potentially.

Determining your mentoring approach's parameters can help you become a more effective mentor because it allows you to more clearly understand what you are attempting to accomplish with your mentee. Rather than trying to do anything and everything for your mentee, setting a clear focus from the outset is a great way to help set yourself up for success.

Strategy #3: When communicating with your mentee, focus on active listening.

As a mentee, there have been many times when I felt that my mentor wasn't truly listening to me. It sometimes seemed as though they had come into the conversation, emphasizing what they wanted to tell me rather than hearing what I wanted to share first. I used these experiences to help remind myself that as a mentor, one of my most important responsibilities is to listen to my mentee.

One aspect of active listening involves being fully present with your mentee. Although it can be easy to want to multitask and divide your attention across several areas at once, it's important to consider the impact that doing this could potentially have on your mentee. For many mentees who are new to the workforce and working with a mentor for the first time, it can already seem overwhelming to think about taking up someone else's time for their own personal benefit. Thus, being fully present and engaged with your mentee helps to remind them that you have chosen to spend your time working with them and coaching them along their career journey.

Another active listening strategy that you can use with your mentee is paraphrasing. Sometimes when I'm in conversations with my mentees, I might not understand exactly what they're trying to tell me the first time. For example, they might be describing a problem in a very broad or vague manner but are still seeking a specific outcome. Rather than assuming that I fully understand what was said, I pause to paraphrase the information that I heard them say. Paraphrasing involves restating someone else's message in your own words. I often use handy sentence starters to help me launch into paraphrasing, such as:

- "What I hear you saying is… ."
- "As I understand it, you're feeling… ."
- "It sounds like you're saying that… ."

Paraphrasing gives you the opportunity to get on the same page as your mentee before you move into the next step of offering suggestions and feedback.

Even with paraphrasing, however, there might also be times when you are unclear about a few details and need to clarify them before moving forward. As a mentor, asking clarifying questions is actually a good thing. Although it may seem daunting to ask questions at the risk of coming across as uninterested or unengaged, the process of clarifying information can demonstrate to your mentee that you are actively listening to them and need to know more about some of the specific areas that they mentioned.

Here's an example of why this matters: I once went into a conversation with a mentor on a day when we were scheduled to discuss an upcoming project. My mentee started by saying, "I'm sure you're excited about the new project!" I was actually terrified about it, and then I felt that because my mentee assumed that I should have been excited, I was hesitant to share my true feelings about the project with her. On the other hand, if the conversation had started by giving me the opportunity to share my feelings first, I most likely would have felt comfortable enough to tell her about my anxiety around the project.

Sentence starters to use in the clarifying phase of communication include:

- "What do you mean by…?"
- "Can you tell me more about…?"
- "How do you feel about…?"

Strategy #4: Give your mentee feedback that is specific and actionable.

When I was completing my teacher residency program, I had to participate in daily debrief sessions with my mentor and coach. We would review the lessons that I had taught, discuss what went well and what could have gone better, and determine areas to focus on for the future. Over the course of the school year, I built a great rapport with my mentor and coach and truly came to value the feedback that they shared with me.

However, during one phase of the program, I was temporarily assigned to another school, where I had to debrief with a different coach. Her role in mentoring me was intended to be the same as the others, but unfortunately, her feedback was nowhere near as helpful. I'll never forget the feedback she gave to me one afternoon, which was the most unhelpful and unactionable feedback that I have ever received in my career: "Olivia, you need to fix your face."

After I got past my initial shock and asked the coach to explain what she meant by that, she told me that in her opinion, I wasn't looking happy while teaching my lesson that day, and so she felt that I needed to smile more. The real takeaway that I got from this had nothing to do with my facial expressions and everything to do with actually how to give helpful advice to a mentee.

First, you want to ensure that your feedback is specific. The "fix your face" advice that I described above is an example of vague advice that was overwhelmingly vague and discouraging. However, even positive and encouraging feedback can still be vague and unhelpful. For instance, if you have ever been given feedback along the lines of "Good job!" or "Keep up the great work!", you might have felt good about yourself at the moment, but then later wondered what exactly it was that you did in the past that you should apparently still be doing. Thus, making your feedback specific can help your mentee precisely understand what they should be doing.

Next, providing actionable feedback can help your mentee comprehend why they should do something in a certain way. Doing so provides reinforcement for them to reflect back on the rationale for taking the actions that you are suggesting.

Suppose that you and your mentee are working on a code review, and you need to give them feedback. An example of unspecific and unactionable feedback might be: "Good job, this looks really great." By comparison, an example of specific and actionable feedback could be:

- "You did well at ____. This is great to keep doing because…"
- "In the future, consider doing _____ because…"

By receiving feedback in this format, you empower your mentee to improve in the areas that you have suggested and continue coming back to you for additional recommendations.

Strategy #5: Teach your mentee how to take ownership of the mentoring relationship.

As a mentor, it's a natural feeling to want to go above and beyond for your mentees and to do all that you can to help them succeed. Knowing when to step back and allow my mentees to take more of the reins was actually one of the areas that I most struggled with in my earliest years of mentoring. I didn't realize at the time that as much as I was working with my mentees to improve their technical, leadership, and overall career skills, I also had the valuable opportunity to teach them how to become a mentor someday themselves.

Determining exactly how to give your mentee more chances for ownership can vary, depending on the particular setup of your mentoring relationship and their individual work experience. For example, when I have mentored employees who were at the beginning of their careers working in their first full-time job, part of giving them more ownership involved tasking them with creating the calendar invitations for our meetings, setting the meeting agendas, etc. For more seasoned employees, I challenged them to identify the metrics that they wanted to associate with their short-term and long-term mentoring goals.

As part of this strategy, you should also make sure to provide specific and actionable feedback as your mentee learns how to take on more responsibilities, even for the smallest tasks. Leading by example and providing ongoing feedback can help enable you to become a more effective mentor and also show your mentee how to do the same thing themselves.

When I reflect back on the phone call in which the residency program director invited me to pursue a path towards mentoring, I wonder how differently things might have gone if I had been given the tools to help me feel equipped to take on such a momentous task. Although I was afraid to become a mentor at the time, I eventually went on to become the chairperson for my teacher residency program's alumni network. In this role, I've advocated for opportunities for my fellow alumni to give back and share their knowledge, particularly through mentoring.

Perhaps the most important thing that I've learned about becoming an effective mentor is that this sort of transformation doesn't happen overnight. It's a lifelong journey, filled with endless chances to learn more, improve over time, and help make a positive impact on others.

Olivia Liddell is a Technical Curriculum Developer at Amazon Web Services. Over the course of her career, she has created innovative teaching and technical training solutions for learners from diverse backgrounds and skill levels. A Certified Ethical Hacker, Olivia frequently speaks on topics such as social engineering and security awareness. In her spare time, she enjoys distance running, studying linguistic anthropology, and developing web applications for Arabic language learners. @oliravi

The Terrifying Scale of a Security Bug

Eric Mann

A remote code execution vulnerability discovered in the widely used Log4J library exposed billions of machines to malicious actors in December. Unfortunately, fixing this bug was not straightforward and left much of the Internet exposed to bad actors for over a week.

In mid-November, a security researcher with Alibaba's Cloud Security team[1] identified a flaw in the popular Log4J library, a tool used by most Java developers to implement logging in their application. The bug itself allowed an attacker to inject a string of their own choosing into the Java Naming and Directory Interface (JNDI) component of the Java runtime. This interface will attempt to resolve the string passed in, in some cases looking up additional information over DNS or by executing code provided as part of the input string.

Said more simply—any application logging web requests with a vulnerable version of Log4J allowed a remote, unauthenticated attacker to execute whatever commands they wanted on the application server. The worst possible outcome of any bug. Or, in the words of Amit Yoran, CEO of cybersecurity company Tenable,

> *It is by far the single biggest, most critical vulnerability ever.*

> *Injection attacks are very common in modern software. They're so common in fact that the Open Web Application Security Project (OWASP) considers them the third most frequently seen[2] category of application security risks in the market. An injection vulnerability occurs any time untrusted user input is passed directly to a critical component of an application—like to the data layer (SQL) or to business logic that attempts to resolve a system name (JNDI).*

The Scope of the Incident

The initial security report[3] included all versions of Log4J through version 2.14.1. Given the trivial nature of the exploit itself—attackers were able to remotely execute code on Minecraft servers by pasting a string in the in-game chat[4]—the community moved rapidly to patch the issue.

Perhaps too rapidly.

Log4J 2.15.0 rolled out[5] the same day the original exploit was documented publicly. It was a Friday afternoon, but many engineers worked late into the evening (or through the weekend) to patch their applications.

Unfortunately, the fix was incomplete. Just days later, researchers discovered another (related) vulnerability[6]. Many non-default configurations would allow for further exploitation of the underlying bug. This could still allow for information leaks (the exfiltration of environment secrets), remote code execution, or even denial of service attacks. The engineers, just as quickly, launched Log4J 2.16.0 only for yet another related vulnerability[7] to be uncovered just three days later. The team has released version 2.17.0[8], containing all the previous fixes, and is thought to be secure as of this writing.

In all, the original issue required not just one, but three separate passes to fully remediate the flawed code and keep our systems patched. Given how rapidly the team had to move to correct the issue, there's always the potential that this latest version is still incomplete and will require more work down the road.

The Scale of the Incident

The open-source Log4J library is distributed by the Apache Foundation and used by most Java-based software you are probably familiar with. Minecraft uses it for logging, as does Jenkins. Even iCloud, Spotify and Oracle use it. Tesla vehicles run an embedded version of Java as part of their control interface that uses Log4J. Amazon uses it to power the control planes behind most of AWS' infrastructure[9] as well.

[1] *Alibaba's Cloud Security team:* https://www.mcafee.com/blogs/?p=133000

[2] *considers them the third most frequently seen:* https://owasp.org/Top10/A03_2021-Injection/

[3] *The initial security report:* https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[4] *Minecraft servers by pasting a string in the in-game chat:* https://twitter.com/MalwareTechBlog/status/1469290238702874625

[5] *Log4J 2.15.0 rolled out:* https://us-cert.cisa.gov/node/17179

[6] *discovered another (related) vulnerability:* https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45046

[7] *yet another related vulnerability:* https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-45105

[8] *released version 2.17.0:* https://phpa.me/techtarget-log4J

[9] *control planes behind most of AWS' infrastructure:* https://aws.amazon.com/security/security-bulletins/AWS-2021-006/

While this particular issue impacted Java and not PHP, there are several ways in which Java could impact our lives or our applications. Knowing that much of the AWS stack runs on Java means any application could have been impacted prior to Amazon patching their libraries. Anyone who uses Jenkins as a task runner or CI/CD platform could potentially be exposed. Elasticsearch, Couchbase, Okta, and even Debian were all impacted by this vulnerability[10]. Even if your code is 100% secure, there is a significant chance another component of your stack is not, and it's up to you to keep track of all of those moving pieces to keep your application stable.

In fact, over the last four months alone, Log4J has been downloaded over 28 million times[11], placing it on the top 0.003% of Java modules in terms of popularity and usage. To put things in perspective, the PHP Monyog logging library was downloaded just 1 million times[12] over the same four-month period.

As of December 20th, just ten days after public disclosure, the community was finally able to identify the sheer scale of the incident. An estimated 93% of cloud environments[13] were vulnerable to the issues in Log4J—while not all of those environments are exposed to the public Internet, enough of them are that this is indeed the most critical vulnerability our community has faced to date.

Where We Go From Here

Some in the development community often criticize PHP for its reputation of being insecure. In reality, any language is only as secure as the coding practices leveraged by the developers who wield that language. It's more often that insecure code slips through due to innocent mistakes only to be discovered when a package, library, or program receives unexpected popularity and attention.

I wrote about my own experience on the receiving end of a similar apocalyptic security notice with a WordPress plugin in the November issue[14] of this magazine. I have no doubt some of the Apache team will be writing about their experience with Log4J for quite a while.

In the meantime, the best we can do is ensure we always follow best practices and keep our projects secure.

- Track the OWASP Top Ten.
- Conduct rigorous code reviews.
- Perform both internal and external penetration tests against your applications.
- Update every library or component you use (whether they're PHP-related or otherwise) to ensure a critical vulnerability like that found in Log4J doesn't slip by.

In other words, we keep doing what we do best with the sober understanding that this kind of bug could happen to any of us, at any time, in any of our applications.

Related Reading

- *Security Corner: Updating the OWASP Top Ten* by Eric Mann, October 2021. https://phpa.me/update-OWASP-top-ten
- *Security Corner: No Bug Too Small* by Eric Mann, November 2021. http://phpa.me/no-bug-too-small
- *Debugging with Purpose* by Joseph Maxwell, June 2021. http://phpa.me/debug-with-purpose

Eric is a seasoned web developer experienced with multiple languages and platforms. He's been working with PHP for more than a decade and focuses his time on helping developers get started and learn new skills with their tech of choice. You can reach out to him directly via Twitter: @EricMann

[10] https://heimdalsecurity.com/blog/?p=39256
[11] http://phpa.me/log4j-sets-internet-on-fire
[12] http://phpa.me/monolog-dl-stats
[13] https://phpa.me/darkreading-log4j-flaw
[14] https://phpa.me/no-bug-too-small

Apache and PHP - Back to Basics

Joe Ferguson

This month we're diving into Apache[1] and PHP configuration to better understand the relationship between the web (HTTP) server and our application. When getting started with PHP, it's quite common for tutorials and guides to skip over the webserver and focus more on the language aspects. PHP developers need to have a strong understanding of how the webserver executes their code. As requests and responses being handled by a server are the primary purpose of web applications. We're going to install Apache, PHP, and review configuration for ensuring our application not only functions well but leverages Virtual Hosts to serve multiple applications.

Apache is an open-source HTTP server launched in 1995 and has been the most popular web server on the internet since 1996. My earliest experience with Apache goes back to 1998 with PHP and Perl Common Gateway Interface[2] applications on Debian and Red Hat Linux distributions. The ability to build your web server on the internet in 1998 felt incredibly empowering. These days of static sites being hosted in S3 storage buckets or being hosted by the hot new startup, it's easy for developers to forget about the webserver.

We'll be using Ubuntu 20.04 for our examples, but Apache runs on just about every Linux distribution you'd like to use. These examples will hold in Ubuntu-like distributions but always refer to your distribution documentation. In Red Hat, Centos, and similar the Apache package is named `httpd` instead of `apache2`; however, the configuration options will be the same.

Installing Apache and PHP using `apt` on Ubuntu will install and configure nearly all of the required parts to wire PHP to apache, which is done via the `libapache2-mod-php` package giving Apache the ability to run our PHP applications.

```
sudo apt install apache2 php libapache2-mod-php php-xml
```

Configuration and Directives

Apache is configured by plain text directives placed in a specific folder structure. The primary configuration file is `apache2.conf`. The rest of the configuration files `ports.conf`, and files in `conf-enabled`, `mods-enabled`, and `sites-enabled` are all combined together as the Apache configuration. Combining what ports to listen on, generic statements or snippets, then what server mods have been enabled. Finally including what HTTP services are running and listening on specific devices, or virtual hosts. Virtual hosts are how you tell Apache how to run our application.

PHP configuration is where you would expect it on a Debian derivative located at `/etc/php/`. The PHP configuration for Apache will be in the folder `/etc/php/7.4/apache2/php.ini`. This folder structure follows a similar pattern for each version of PHP we install.

On Ubuntu systems, we'll see the contents of `/etc/apache2` where our virtual hosts will be placed in `sites-available`. We can selectively enable specific sites using the `a2ensite` command, which will create a symbolic link from `/etc/apache/sites-available/our-virtualhost.conf` to `/etc/apache/sites-enabled/our-virutalhost.conf`, allowing us to easily turn sites on or off by enabling or disabling them. The ability to toggle sites on and off is immensely helpful when experimenting with configuration options for a specific site. It is easier to isolate the running configuration by turning off sites or other configuration options you may not need. See Listing 1.

Listing 1.

```
/etc/apache2# tree -L 1
.
├── apache2.conf├── conf-available
├── conf-enabled
├── envvars
├── magic
├── mods-available
├── mods-enabled
├── ports.conf
├── sites-available
└── sites-enabled
```

[1] Apache: https://httpd.apache.org

[2] Common Gateway Interface: https://w.wiki/3QZ7

Listing 2.

```
DefaultRuntimeDir ${APACHE_RUN_DIR}
PidFile ${APACHE_PID_FILE}
Timeout 300
KeepAlive On
MaxKeepAliveRequests 100
KeepAliveTimeout 5
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}
HostnameLookups Off
ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel warn
IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf
Include ports.conf
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>

```

Listing 2 continued.

```
<Directory /usr/share>
    AllowOverride None
    Require all granted
</Directory>

<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

AccessFileName .htaccess
<FilesMatch "^\.ht">
    Require all denied
</FilesMatch>
LogFormat "%v:%p %h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" vhost_combined
LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
IncludeOptional conf-enabled/*.conf
IncludeOptional sites-enabled/*.conf
```

We're going to leave most of `apache2.conf` as the defaults, although it is the file to edit to change or adjust:

- the log formatting of Apaches log, you can find the `LogFormat` definitions
- where the logs are stored
- the global values for `KeepAliveTimeout` (Seconds to wait for another request from the same client on the same connection)
- the Linux user and group Apache runs as (`www-data` by default)

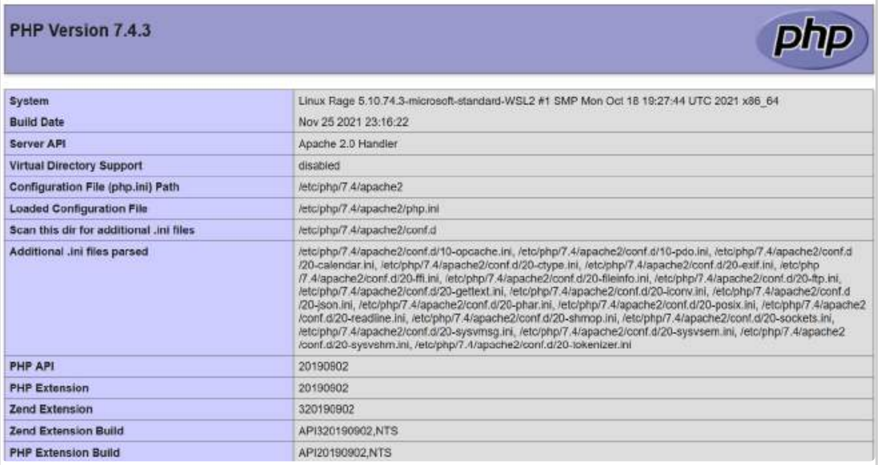
While `apache2.conf` looks extensive, it appears a bit less daunting when you look at the default configuration without the large comment blocks. See Listing 2.

Default Host Configuration

We can see there are two sites enabled by default `000-default.conf` and `default-ssl.conf`. Together these files contain the directives to serve a basic site from the path `/var/www/html`. The virtual host configurations will inherit the entire Apache configuration by default. Virtual hosts are specific configurations for websites and applications being served from Apache. We can use `sudo vim /var/www/html/info.php` to create a new PHP file and paste `<? phpinfo();`. Saving the file (`:wq` or `:x`) and visiting `http://localhost/info.php` shows you something similar to the PHP Version 7.4.3 info page.

While PHP 7.4 is a fine release, in 2021 it's somewhat old. We should use Ondřej Surý's[3] Private Package Archive (PPA) to install more recent versions of PHP. To add the PPA, we'll run `sudo add-apt-repository ppa:ondrej/php`, which adds Ondřej's repository on our system. To install PHP 8.1 in our current configuration we can run `sudo apt install -y libapache2-mod-php8.1 php8.1-xml` . With our new PHP version enabled, we can use `sudo a2dismod php7.4` to disable PHP 7.4 from Apache and use `sudo a2enmod php8.1` to enable PHP 8.1 for use with Apache. To apply our PHP version changes, run `sudo service apache2 restart`, and we can refresh our `http://localhost/info.php` browser window to see we're

PHP Version 7.4.3 info page



PHP Version 7.4.3

System	Linux Rage 5.10.74.3-microsoft-standard-WSL2 #1 SMP Mon Oct 18 19:27:44 UTC 2021 x86_64
Build Date	Nov 25 2021 23:16:22
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-ffi.ini, /etc/php/7.4/apache2/conf.d/20-fileinfo.ini, /etc/php/7.4/apache2/conf.d/20-ftp.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-json.ini, /etc/php/7.4/apache2/conf.d/20-phar.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvsem.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini
PHP API	20190902
PHP Extension	20190902
Zend Extension	320190902
Zend Extension Build	API320190902,NTS
PHP Extension Build	API20190902,NTS

[3] *Ondřej Surý's: https://deb.sury.org*

Listing 3.

```
<VirtualHost *:80>
    ServerName fresh.test
    DocumentRoot /var/www/fresh/public
    ErrorLog ${APACHE_LOG_DIR}/fresh-error.log
    CustomLog ${APACHE_LOG_DIR}/fresh-access.log combined
</VirtualHost>
<VirtualHost *:443>
    ServerName fresh.test
    DocumentRoot /var/www/fresh/public
    SSLEngine on
        SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
</VirtualHost>
```

now running PHP 8.1.0. An important note: we're leveraging mod_php[4], an Apache module, to run PHP; this means we can only have one PHP version running at a time. To run multiple PHP versions with Apache, we'd want to install and configure PHP-FPM[5], which we'll cover in a future installment. Using `mod_php`, we're wrapping PHP inside Apache. FPM, for example, is an external (to Apache) service typically connected via sockets or TCP.

To create a self signed SSL certificate for our local development site we can use `sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt`. For usage across networks (i.e., once you're not on your local machine), you'll want to use Let's Encrypt[6] to create functional SSL certificates that browsers will trust because they're signed by a proper Certificate Authority, unlike our self-signed certificate.

Now with a fresh PHP 8.1 working with Apache, we're ready to create our first virtual host which will serve as our development environment for our project. We'll create a fresh application located at `/var/www/fresh` and use `sudo chown -R www-data:www-data /var/www/fresh` to ensure our `www-data` user owns the files and will be able to run them via Apache. Create a new file at `/etc/apache2/sites-available/fresh.test.conf` containing the Virtual host directives as shown in Listing 3.

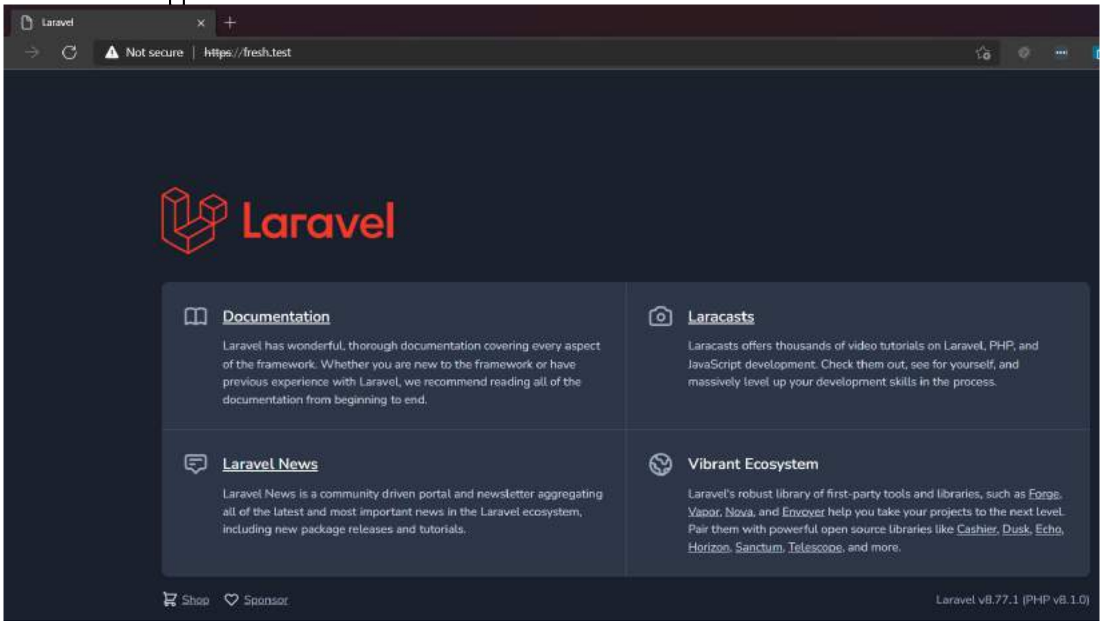
Our `fresh.test` Virtual host will listen on any network address to port 80 for incoming requests: `<VirtualHost *:80>`.

[4] *mod_php: https://stackoverflow.com/questions/2712825*

[5] *PHP-FPM: https://www.php.net/install.fpm*

[6] *Let's Encrypt: https://letsencrypt.org*

Fresh Laravel application.



Documentation

Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

Laracasts

Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News

Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.

Vibrant Ecosystem

Laravel's robust library of first-party tools and libraries, such as Forge, Vapor, Nova, and Envoyer help you take your projects to the next level. Pair them with powerful open source libraries like Cashier, Dusk, Echo, Horizon, Sanctum, Telescope, and more.

Shop   Sponsor   Laravel v8.77.1 (PHP v8.1.0)

We'll use `ServerName fresh.test` to instruct Apache to serve requests at our development URL. The `DocumentRoot` directive instructs Apache to use the path as the webroot to serve `fresh.test` from `/var/www/fresh/public/` path. The next two log directives configure the `ErrorLog` to contain all site errors and a `CustomLog` to log all the requests accessed of this server. Once the server has sent the request to the PHP process the application's logging configuration may utilize these same log locations. Although, they will often use their own such as `storage/logs/laravel.log` is the default for a Laravel application log location.

With our Virtual host configuration complete, we need to enable our site with `sudo a2ensite fresh.test.conf` then reload Apache with `sudo service apache2 reload`. Before we test our configuration in the browser, we need to ensure our system's host file contains a mapping for our `fresh.test` domain since it's not a real registered domain name. We'll want to add `127.0.0.1 fresh.test` to `/etc/hosts` on Linux, Unix, macOS systems while Windows users will find this file at `C:\Windows\system32\drivers\etc\hosts`.

Because we're using a self-signed certificate, web browsers may show warnings when we visit `https://fresh.test` such as Firefox's "Firefox detected a potential security threat and did not continue to fresh.test" We know this is our development site, so we can click "Advanced" and "Accept the Risk and Continue" to proceed to our site, which is now serving our fresh Laravel application.

Ultimately there is not much to Apache. When software has been around this long, it's reassuring to know the investment spent learning it is still useful today and carries over to all aspects of web development, not just PHP. The fundamentals Apache teaches us will apply to every back-end programming language, whether PHP, Ruby, or Python. The biggest thing I appreciate about the Ubuntu ecosystem is how numerous mundane tasks such as: creating paths, shortcuts to configuration files, and auto wiring up PHP & Apache via `mod_php`; are automatic. Alternatively, on Debian or Red Hat systems, you will be responsible for creating and copying a lot of these files and configuration paths manually. Another benefit of Apache is that so much content exists on every level; from complete new users to grizzled veterans, there's always something new to learn.

Thanks for joining a trip down memory lane and learning about the Apache and PHP basics. I hope you're empowered to start hacking on your instances of Apache and exploring what the most popular web server on the internet has to offer. Next month we'll be diving into FPM configuration and demonstrating how to use Nginx[7] to provide more secure and isolated applications.

Joe Ferguson is a PHP developer and community organizer. He is involved with many different technology related initiatives in Memphis including the Memphis PHP User group. He's been married to his extremely supportive and amazing wife for a really long time and she turned him into a crazy cat man. They live in the Memphis suburbs with their two cats. @JoePFerguson

[7] *Nginx: https://www.nginx.com*

Interview with PHP 8.1 Release Manager Patrick Allaert

Eric Van Johnson

PHP 8.1 is now the current version of PHP, and there is a new team behind managing the releases of PHP 8.1. For this release of PHP, the Internals groups decide to do something different. There is still a veteran release manager for PHP 8.1, namely, Joe Watkins (@krakjoe). But instead of only having one new release manager, "rookie," they elected two additional people to manage the release. This month we speak to one of those rookie release managers, Patrick Allaert. When Patrick isn't coding PHP or C, you might find him dancing Forró/Salsa/Bachata or kitesurfing on the ocean.

How did you get started with coding, and what got you interested in PHP?

In the late '90s, I ran a website about song lyrics. PHP 3 helped me build a fast-evolving website. Later on, in 2002, as part of my thesis, I had to develop an ultra-fast custom TCP protocol communication between a website (made in PHP 4) and a proprietary 3D engine: this was the first time I discovered PHP's source code.

What do you do professionally?

I am a Freelancer providing services mainly around PHP, its core, and ecosystem. I am currently working on Blackfire's PHP extension and as an IT architect for a Belgian company in the public sector. Previously I was a core developer of eZ Publish/eZ Platform CMS (now Ibexa).

How much PHP coding do you do?

It is pretty even between PHP and C, about 50% of each.

How long have you been involved with PHPInternals?

Probably something like 15 years when I started discussing on the #internals PHP mailing list. My first real practical involvement was in 2009, and this is when I got access (called internally "karma") to PHP's source code to contribute to PHPT tests that are part of PHP's unit test suite. Later, I contributed with a patch to have "Array to string conversion" notices (now warnings) before PHP 5.4: that was silently converting it to string "array," pretty useless and hiding a lot of possible bugs.

What is it like being a release manager for PHP? What are the responsibilities?

I would summarize it as watching the CI, preparing tag/tarballs/versions, and then announcing them.

How has it been working with Joe Watkins through this process?

Joe helped us with the questions that Ben or myself had, but another great resource that previous managers have contributed to is the various wiki pages about release management. This is the first time there have been 2 "new" release managers.

What challenges do you see for PHP in the coming years?

The desire, by some, of PHP being a stricter language, by some others, of being enriched with new features, and by yet others of keeping backward compatibility as much as possible is quite tough and often leads to passionate discussions. On the other side, the engine keeps getting more and more complex as new features get in. We are very lucky to have Nikita Popov and Dmitry Stogov aboard. Unfortunately, the number of people able to understand all the parts of the engine does not scale as much as its complexity.

What tools do you use in your daily development workflow today?

- IDE: Vim and JetBrains (PHPStorm & CLion)
- OS: Gentoo Linux
- Shell: bash
- Debugging tools: xdebug, gdb, strace/ltrace, Wireshark, inotify tools
- Static analyzer: Phan
- VCS: git

What are some of your proudest accomplishments or achievements?

Getting the trust of PHP internals to be one of the Release Managers is quite a big thing for me. I am quite proud of it.

Who, in any development community, do you look up to or inspire you, and why?

Rasmus Lerdorf, Nikita Popov, Derick Rethans & Sara Golemon for their brilliant minds.

Do you have any opinions about PHP Frameworks, and are there any you are interested in using or do use?

PHP Frameworks are a bit incomparable to other languages "Web

Frameworks." With general-purpose languages such as C#, C/C++, Java, Python, Ruby, … there is no native "web" support at the level of the language itself. It usually relies on Web Frameworks for that. While some will advocate that PHP can be considered general-purpose, it is most famous and tailored towards web development.

When dealing with the "web," there are many different aspects:

First, the HTTP protocol, when a **request** comes, it is a message that contains an HTTP method, a target, an HTTP version, many possible headers (including cookies/session info), and possibly a body.

Whatever webserver you are working with, PHP abstracts those concepts and makes the information available in a standardized way (`$_GET`, `$_POST`, `$_COOKIE`, `$_FILES`, `$_SESSION`). All of those superglobals are lazy-loaded until you have to deal with one of them; PHP won't spend the time to interpret that part of the request.

When it comes to answering a request, PHP abstracts the complexity of generating the HTTP **response**. That response will be composed of a protocol version, a status code/text, headers, and, most of the time, a body.

In order to deal with the complexity of combining all those ingredients, PHP provides a full suite of functions to control the outcome: `header()`, `set_cookie()`, `sessions_start()`, `http_response_code()`, … and writing in the body is as simple as using echo or having content outside of "".

While the body has to appear after all the headers, PHP provides output buffering so that you may not have to deal with the order of operations.

Secondly, the web is also about formats, while HTML can be just considered as simple text, PHP natively provides ways to interact (both reading and writing) with the most popular formats on the web: XML (with SimpleXML, DOM, XMLReader, XMLWriter), XSL, JSON, images (with GD, ImageMagick, exif),…

Finally, it is also common to interact with 3rd parties HTTP-enabled services. This is also natively available and made easy thanks to the awesome bundled curl library or through SOAPClient/SOAPServer.

This makes PHP already closer to many other languages' Web Framework in terms of features.

For a quality project, this one should respect and use many best practices/principles: coding standard, version control, clean routing mechanism, design/architectural patterns, separation of concerns (MVC), inversion of control (dependency injection), SOLID, YAGNI, KISS, DRY, security, static analysis, testing (unit, functional, BDD)…

Many (PHP) frameworks will promote and advocate those concepts and often guide their users to achieve them. PHP.net[1] and its documentation does not really guide newcomers much and, I wish it could. However, as there are many different ways and opinions to reach those best practices, it may be hard to keep neutrality.

As a veteran PHP developer, I do not feel the need to use a PHP Framework as all of the patterns or best practices I mentioned above can be achieved without, nor would it help me. That does not mean I am reinventing the wheel, nor that I do not include libraries. My personal projects use the following skeleton: "vanilla" PHP + composer + phan. Not using another framework other than PHP, my projects are usually simpler to keep up-to-date as PHP evolves with a strong backward compatible policy. Over more than ten years, very few things had to be changed while upgrading PHP.

When I work on a project that includes a major PHP Framework, it is Symfony.

Next month, we speak with the other first-time release manager Ben Ramsey.

Eric Van Johnson is the CTO of [DiegoDev Group](), LLC. An organizer of San Diego PHP (SDPHP) and podcaster with php[podcast], PHPUgly, and PHPRoundtable. A husband, father, and enjoyer of scotch and baseball. @shocm

[1] PHP.net: http://php.net

Turning to Domain-Driven Design

Edward Barnard

We begin our implementation of Domain-Driven Design with simple refactoring. We'll take a close look at what complexity might be hidden within the expected implementation. That observation will guide us in refactoring toward Domain-Driven Design.

Implementing the Dragon Wrangling Pattern

Why turn to Domain-Driven Design in your PHP project? With an existing codebase, it's time to ask that question when the code has become too fragile to touch and too costly to change. I call this "hitting the wall" because it has become so difficult to move forward. At this point, we, as PHP developers, need to solve two problems simultaneously:

1. We are reluctant to touch the code at all because of the difficulty and danger of adding or changing features. The risk of breaking something along the way is simply too high.
2. We need to onboard newer developers or persons new to the project are unaware of the pitfalls buried in the complex codebase.

I suggested a solution, the Dragon Wrangling Pattern, in the October 2021 php[architect]. It's time to begin implementing that pattern in an actual project. This article assumes you're able to go back and review the reasoning and motivations behind this pattern.

Athlete Registration

Our project will be implementing two specific workflows involving high school athletes:

- Create an online account
- Register to compete during a particular season

My company, the USA Clay Target League, is a non-profit corporation that facilitates sports related to shooting at clay targets with a shotgun (Trap, Skeet, 5-Stand, and Sporting Clays). We have created a means for teams to compete against each other without ever coming face-to-face during the season.

Think about it this way. Consider a bowling league that competes at their home Bowling Lanes competition. A second team competes at a different location, and they both put up their scores. We assume any particular bowling lane is identical to any other Bowling Lanes setup. Someone scoring 90, for example, should be identical to someone else scoring 90 at the other location. In comparing or combining scores, we are "comparing apples to apples," so to speak.

So it is with clay target shooting. Each gun range has the same setup and layout. We assume the score at one location is equivalent to that at another location. We ignore the fact that wind and weather will have a significant impact from one day to the next. That's part of the fun!

Scoring is quite simple—just like with bowling. In bowling, each pin is either still standing or is knocked over. There's no "degree of difficulty" or "artistic impression" to consider. With trap shooting, a shot either breaks the clay target or does not.

The mechanics are simple. That should make them easy to implement, right? Unfortunately, it's not that simple!

It turns out we can't actually ignore the weather. A late winter storm could mean the gun ranges still need to be dug out and made operational. Further south, a tropical storm could prevent a team from competing that week. We do, in fact, have business rules for delaying the season's start for one state while allowing a make-up week for an individual team affected by something outside their control (such as a hurricane).

In other words, even the operational scoring system carries enough complexity that we should be considering Domain-Driven Design.

Before a competition, however, we first need to assemble the team. We'll focus on the above two workflows (create an account and season registration) with that end in mind.

I'll be showing you a pattern that's so trivial that it may appear as pointless and wasted effort. But we'll also be seeing its power. It works!

Let's take the time to understand the problem we're trying to solve. I'm reminded of the old joke that runs, "We just got a new project! You two start coding while I go find out the requirements." Don't take that approach! As we learn about the problem, we'll be taking careful note of where complexity may lay hidden.

Create Online Account

With the USA Clay Target League, every athlete must create an online account. Furthermore, every athlete account must be tied to a team. But, the athlete must already have an account for the team to invite the athlete.

We have a “which came first, the chicken or the egg?” problem. Our solution is this sequence:

1. The team must exist.
2. The team invites the athlete to create an account using that team’s “team key.” The “team key” is like a coupon code. It’s permanent and specific to that team. It can be handed out on a piece of paper, sent via email, etc.
3. The athlete creates an account.
4. The team (i.e., the team’s coach) then invites the athlete to compete in one or more specific disciplines (Trap, Skeet, etc.) for the upcoming season.
5. The athlete accepts the invitation(s) registering for (and optionally paying the fees for) the competition season.

The workflow for Step 3 above, creating an account, is deceptively simple.

1. Enter the team key, first and last name, and date of birth. Many of our business rules are based on age, and thus we need the date of birth right away.
2. Set up the login, including username and password. Depending on age (different requirements apply when under age 18, under age 13, etc.), we may or may not require an email address.
3. Collect guardian information (optional if the athlete is age 18 or older) and emergency contact.

That’s a simple workflow, but the question of whether something is required or optional depends on age and certain other considerations. Thus, we have a bit of complexity that must be handled somewhere. Thus far, however, this doesn’t look like anything complex enough to push us in the direction of Domain-Driven Design.

Season Registration

Most of the season registration is a simple workflow. Enter demographics such as what school grade you’re in, firearm safety certificate number and completion date, and so on.

Then, suddenly, implementation becomes complex. Oops! What’s the problem? Can we simplify it?

Here are the variables.

- We are a non-profit corporation, and we allow the athlete to include a donation during season registration.
- Season fees can be either “Athlete Pay” or “Team Pay.” With Team Pay, once registration closes, the school team pays a single invoice based on the actual registrations. With Athlete Pay, the athlete pays the fee during season registration (i.e., right now).
- The athlete could be invited to one, or up to all four, of the disciplines. Each discipline has the same fee. The fee, as noted above, is payable by either the team or the athlete.
- The athlete need not select all invitations at once. For example, if invited to both Trap and Skeet, the athlete could register for Skeet now, optionally coming back to register for Trap later.
- All registrations with Athlete Pay, and all registrations with a donation, proceed to the payment screen.

Thus our number of use cases is (Athlete Pay or Team Pay) times (with or without donation) times (accepting zero through up to four invites) times, possibly multiple trips through season registration.

Can We Simplify?

In fact, what I described *is* the simplification. We tossed out a feature that added another complex variable to the mix. This brought me down to 43 distinct “happy path” scenarios for the single web page where our athlete chooses what disciplines to compete in and whether or not to include a donation to the League.

More importantly, this exercise allowed me to lay out a short set of business rules to be discussed with our stakeholders. Rather than having a bunch of “magic” under the covers, we have well-structured business rules written out that any relevant person can easily understand. We also have those 43 scenarios running via `behat`, showing in detail how we fulfill each of those different paths through the code.

It took me the better part of a day to boil that “simple” invitation page down to a coherent, small set of business rules that make sense. That was time well spent. Getting that unexpected complexity under control was crucial.

Let’s write some production code.

The First Workflow

The first workflow, creating a new account, is not difficult. We have many years of experience doing exactly this at the USA Clay Target League. We’re spreading the workflow across several web pages. (The reasons are outside the scope of this article, but they’re based on previous years’ experience with this functionality.) This is a data-collection application with “Next” buttons at the bottom of each screen.

It made the most sense to us to put the entire workflow in a single Controller class. Each web page becomes a method in that Controller. We created what Fowler in *Patterns of Enterprise Application Architecture* (PoEAA), page 110, calls a “Transaction Script:

> *Organizes business logic by procedures where each procedure handles a single request from the presentation”.*

As it happens, that is the very first pattern cataloged in PoEAA, so it seems a good starting point. Fowler explains when to use it:

> *The glory of Transaction Script is its simplicity. Organizing logic this way is natural for applications with only a small amount of logic, and it involves very little overhead either in performance or understanding.*

That fits our situation exactly!

Except that it doesn't. There's a problem. That one Controller class became far longer than 1,000 lines long. Business logic became deeply nested if blocks. The code works, which is good. Yet Fowler notes in *Refactoring: Improving the Design of Existing Code, 2nd Edition*, page 73 ("Long Function"):

> *Since the early days of programming, people have realized that the longer a function is, the more difficult it is to understand.*

True, our Transaction Script (the Controller class) is not complex, but we greatly added to the *mental* complexity, often called the cognitive load.

Furthermore, Fowler (PoEAA, pp. 111-112) notes that a Transaction Script pattern quickly reaches its limit:

> *As the business logic gets more complicated, however, it gets progressively harder to keep it in a well-designed state… more complex business domains need to build a Domain Model… It's hard to quantify the cutover level, especially when you're more familiar with one pattern than the other. You can refactor a Transaction Script design to a Domain Model design, but it's a harder change than it otherwise needs to be. Therefore, an early shot is often the best way to move forward.*

A funny thing happened! Looking at that 1,500-line Controller class, I decided I was going to refactor it right now, even though it wasn't a good time. I had already built comprehensive test coverage, so I felt my risk of breaking something was pretty low. Only just now, as I type up this walkthrough, did I pull Fowler off the shelf and discover he included the exact same advice with his first pattern in the book.

Fowler completes his advice thus:

> *However much of an object bigot you become, don't rule out Transaction Script. There are a lot of simple problems out there, and a simple solution will get you up and running much faster.*

Within our PHP ecosystem, that remains *extremely* good advice!

Refactor

At this point, the first workflow, in its entirety, sits within one large Controller class. Each controller method handles one specific webpage within that workflow. At this point, the code is absolutely typical.

Each method represents a single-use case, feature, or ability. I had created ten feature files covering both "happy paths" and edge cases, including various kinds of invalid input.

Why am I making a point about the feature files? Because they guide the refactoring. They are part of the Dragon Wrangling Pattern for this reason.

We'll be aligning our software according to the use case. At this point, "controller method," "use case," and "feature file" mean pretty much the same thing. That makes it easy to refactor and restructure. We have a comprehensive set of tests in place, ensuring we don't break any existing feature code.

For each controller method, then, we'll be creating two new classes:

- The use case handler
- The use case handler's Repository

Instead of thinking of the controller method as a web page handler, think of it as a use case handler. It's basically the same thing, but now we're thinking in terms of functionality or business need rather than visual presentation.

Our first step will be to move any business logic out of the controller method and into the use case handler. If we have six controller methods handling the six steps in the workflow, that means we'll now have six separate use case handlers—that is, six new classes, with each use case handler being its own PHP class.

It's oft been said that the two most difficult problems in Computer Science are cache invalidation, naming things, and off-by-one errors. Our problem of the moment is naming things.

"Use case handler," forsooth. Scott Millett, in *Patterns, Principles, and Practices of Domain-Driven Design" (p. 108), calls the use case handler an Application Service:

> *The application service layer represents the use cases and behavior of the application. Use cases are implemented as application services that contain application logic to coordinate the fulfillment of a use case by delegating to the domain and infrastructural layers.*

Scott got it right. We'll call this "thing" an Application Service from here on out.

What's the Repository? At this point, it's simply a wrapper for reaching the application's MySQL database. However, it is very specific and not generalized in any way. It fulfills the Application Service's specific needs *and no more*. It's that restriction that makes the Repository design so powerful, but the reasons why are outside the scope for now.

Application Service

Here is one of the Application Services from that first workflow. You will immediately note that there's basically nothing there! That's powerful, it turns out, and matches Millet's statement that the application service layer coordinates the use case by delegating.

The Application Service contains three methods and each of those methods delegates to the Repository method of the same name. This Application Service class is simply a pass-through.

The corresponding Repository looks like Listing2.

The final method, with the `try / catch` surrounding a manual database transaction, matches the pattern explained with Listing 3 of "Designing for MySQL Transaction Failures" in December 2021 php[architect].

You might have noticed that we reloaded the database row to be updated. We load it once, and then, inside the MySQL transaction, we load it again. That's to ensure we don't accidentally store stale data by "stepping on" some other process that

Listing 1.

```php
<?php

declare(strict_types=1);

namespace App\BoundedContexts\AMS\ApplicationServices\CreateAccount;

use App\BoundedContexts\AMS\Repository\CreateAccount\RAddMedicalInfo;
use App\BoundedContexts\Infrastructure\ReportError\ReportError as Reporter;
use App\Controller\RegistrationController AS Controller;
use App\Model\Entity\Participant;
use App\Model\Entity\User;
use Cake\Http\ServerRequest as Request;
use JetBrains\PhpStorm\Immutable;
use JetBrains\PhpStorm\Pure;

class AddMedicalInfo extends BaseCreateAccount
{
    #[Immutable(Immutable::CONSTRUCTOR_WRITE_SCOPE)]
    private RAddMedicalInfo $repository;

    #[Pure]
    public function __construct(Reporter $reporter, Controller $controller, Request $request, RAddMedicalInfo $repository)
    {
        parent::__construct($reporter, $controller, $request);
        $this->repository = $repository;
    }

    public function newParticipantEntity(): Participant
    {
        return $this->repository->newParticipantEntity();
    }

    public function loadParticipant(int $id): Participant
    {
        return $this->repository->loadParticipant($id);
    }

    public function processMedicalInfo(Participant $participant, User $user): bool {
        return $this->repository->updateParticipantWithMedicalInfo($participant, $user);
    }
}
```

Listing 2.

```php
<?php
namespace App\BoundedContexts\AMS\Repository\CreateAccount;

use App\BoundedContexts\AMS\DomainModel\Interfaces\Constants\CProfile;
use App\BoundedContexts\Infrastructure\LoadTableModels\ParticipantTrait;
use App\BoundedContexts\Infrastructure\ReportError\CReportError;
use App\BoundedContexts\Infrastructure\ReportError\ReportError;
use App\BoundedContexts\Infrastructure\ThirdNormal\CCreatedBy;
use App\Model\Entity\Participant;
use App\Model\Entity\User;
use Cake\Database\Exception\DatabaseException;
use Cake\Http\ServerRequest;
use Cake\I18n\FrozenDate;
use Exception;

class RAddMedicalInfo implements CCreatedBy, CProfile, CReportError
{
    use ParticipantTrait;

    private ReportError $reporter;
    private ServerRequest $request;

    public function __construct(ReportError $reporter, ServerRequest $request)
    {
        $this->reporter = $reporter;
        $this->loadModels();
        $this->request = $request;
    }
    private function loadModels(): void { $this->loadParticipantsTable(); }
    public function newParticipantEntity(): Participant { return $this->participantsTable->newEmptyEntity(); }
    public function updateParticipantWithMedicalInfo(Participant $participant, User $user): bool
    {
        $connection = $this->participantsTable->getConnection();
        try {
            $connection->transactional(function () use ($participant, $user) {
                $entity = $this->participantsTable->get($participant->id);
                $entity->is_active = true;
                $entity->modified_by = $user->id;
                $data = $this->request->getData();
                $consent = $data[Participant::FIELD_MEDICAL_TREATMENT_CONSENT] ?? false;
                if ($consent) {
                    $entity->medical_consent_date = new FrozenDate();
                }
                $entity = $this->participantsTable->patchEntity($entity, $data);
                if ($entity->hasErrors()) {
                    $this->reporter->processPatchErrors($entity->getErrors());
                    throw new DatabaseException('Participant patch errors');
                }
                $this->participantsTable->saveOrFail($entity);
            });
        } catch (Exception $e) {
            $message = 'Could not update participant with medical info';
            $this->reporter->flashError($message);
            $message .= ': ' . $e->getMessage();
            $detail = ['participant' => $participant->toArray(), 'user' => $user->toArray(),'backtrace' => $e->getTrace()];
            $this->reporter->logError($message, self::ERROR_TYPE_LOGGABLE, $detail);
            $this->reporter->flush();
            return false;
        }
        return true;
    }
}
```

was simultaneously making the same mistake. There are other solutions to this problem, such as locking the row for update when reading the row, but either way, it's an important practice to follow.

These sorts of subtle timing problems can be rare and nasty. It's easier to avoid them in the first place.

Listing 3.

```php
<?php

declare(strict_types=1);

namespace App\BoundedContexts\AMS\Factory\CreateAccount;

class CreateAccountFactory
{
    public static function addMedicalInfo(
        RegistrationController $controller,
        ServerRequest $request
    ): AddMedicalInfo {
        $reporter = new ReportError($controller);
        $repository = new RAddMedicalInfo($reporter, $request);
        return new AddMedicalInfo($reporter, $controller, $request, $repository);
    }

    public static function addGuardianContacts(
        RegistrationController $controller,
        ServerRequest $request
    ): AddGuardianContacts {
        $reporter = new ReportError($controller);
        $repository = new RAddGuardianContacts($reporter, $request);
        return new AddGuardianContacts($reporter, $controller, $request, $repository);
    }
}
```

The next piece, again part of the Dragon Wrangling Pattern, is a separate Factory to connect each Application Service to its Repository. Listing 3 is a partial listing, condensed for space:

Each Application Service gets its own class. As we saw, the "add medical information" page gets class `AddMedicalInfo`. I adopted the convention of prefixing the companion Repository with "R" to create class `RAddMedicalInfo` as the Repository.

Since our workflow processes webpage form data, I pass the controller and the request into the factory. The first line of the "add medical information" controller method is:

```php
$appService = CreateAccountFactory::addMedicalInfo($this, $this->request);
```

Finally, I discovered that I needed to be able to pass error messages back to the user from anywhere, whether it be the Application Service Layer, the Repository, or whatever. I created the `ReportError` class to handle passing messages back to the Controller.

The Factory ensures that the error reporter has a link to the Controller object. Everything else (Application Service and Repository) contains a link to the Error Reporter. The error reporter has two basic pipelines. One is error notifications going back to the user, and the other reports more comprehensive failure information to the database. The repository listing shows both pipelines in use within the `catch()` block (methods `flashError()` and `logError()`).

Benefits

After completing that first refactoring, with all tests passing, I found that implementing the next workflow went a lot faster, complexity and all. The first workflow's application services were mostly empty. But that provided me a pattern I could follow far more rapidly with the more-complex second workflow.

This turned out to be a useful "divide and conquer" strategy. I could more easily focus on each use case one at a time. That made it easier to extract extra logic, or extra concepts, into separate classes. Those separate classes, with few or no dependencies, became far easier to unit test.

Separating object creation into a factory method proved useful as well, even though it was barely necessary. The centralized logging and error reporting, with a hot path back to the Controller (and therefore the user), made error handling far easier to write.

In fact, it turned out; it became quite easy to include error handling for every decision or interaction. The net result will likely be a more resilient application while keeping the complexity lower.

Finally, since implementing the first two workflows, we've realized the need for adding notifications to a dashboard. There might be an error condition requiring resolution (such as missing certification), a pending invitation, a tournament qualification, or whatever. We might realize we need to add the notification while in the middle of a workflow.

But look! The infrastructure already exists. Thanks to the Factory approach, all objects have access to the centralized error reporter. That error reporter can be easily expanded to include posting dashboard notifications. The code would be similar to Listings 4, 5, 6 in "Designing for MySQL Transaction Failures."

Summary

We did a lot more explaining than coding. That's a good thing because we were able to simplify our codebase. It's now far easier to understand because we organized it to be seen one use case at a time. We introduced several aspects of the Dragon Wrangling Pattern:

- The `behat` test suite (the feature files) guided us in restructuring the code according to the use case.
- With one separate Application Service class for each use case or feature, we have a useful way to "divide and conquer."
- We didn't say much about the Repository, but it's generally small and limited to directly fulfilling the Application Service's needs.
- The Factory provided us a consistent way of connecting up the pieces for each use case, encouraging us to follow similar practices from use case to use case.

Ed Barnard had a front-row seat when the Morris Worm took down the Internet, November 1988. He was teaching CRAY-1 supercomputer operating system internals to analysts as they were being directly hit by the Worm. It was a busy week! Ed continues to indulge his interests in computer security and teaching software concepts to others. [@ewbarnard](https://twitter.com/ewbarnard)

New and Noteworthy

PHP Releases

PHP 8.1.1 (Bug Fix Release):

https://www.php.net/archive/2021.php#2021-12-17-1

PHP 8.0.14 (Bug Fix Release):

https://www.php.net/archive/2021.php#2021-12-17-1

PHP 7.4.27 (Bug Fix Release):

https://www.php.net/archive/2021.php#2021-12-16-1

PHP 7.3: Has reached end of life, time to upgrade.

https://www.php.net/supported-versions.php

News

PHP Foundation Update

The PHP Foundation has nearly 1200 contributors and over $300,000USD.

https://opencollective.com/phpfoundation

PHP version stats: January, 2022

Brent shares his biyearly summary of which PHP versions are used across the community.

https://stitcher.io/blog/php-version-stats-january-2022

PHP in 2022

Brent shares his thoughts on the year ahead for his 4th year in a row. Exciting times are ahead; let's take a look at modern-day PHP!

https://stitcher.io/blog/php-in-2022

Laravel real-time Code Execution Monitoring

The idea behind Inspector is to create a monitoring environment specifically designed for software developers avoiding any server or infrastructure configuration that many developers hate to deal with.

http://phpa.me/laravel-code-monitor

Capsule DI and Argument Inheritance

Paul M Jones releases version 3.4.0 of Capsule, an autowiring dependency injection system for PHP 8.0+.

http://phpa.me/capsule-di

Qiq Templates

Paul M Jones announces the first release of Qiq, a template system for developers who prefer native PHP templates—but with a light dusting of syntax sugar when you want it.

http://phpa.me/qiq-templates

PhpStorm 2021.3.1 is released

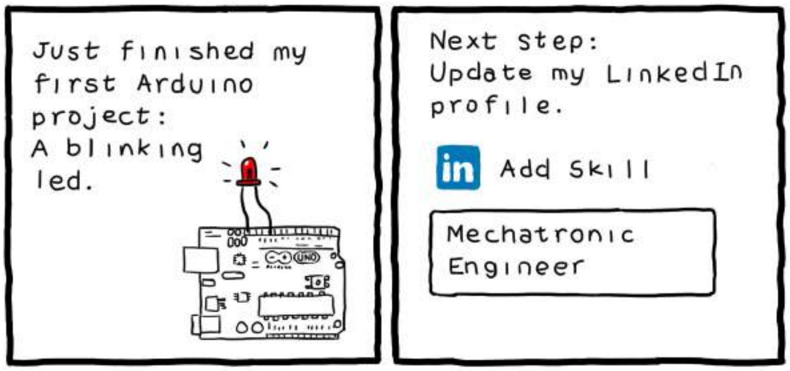
The first bug-fix update for PhpStorm 2021.3 is now available.

http://phpa.me/phpstorm-2021-3-1

A look at what is coming to Laravel 9

Laravel v9 will be the next LTS version of Laravel, and it will be coming out sometime in early 2022. In this post, Laravel News wanted to outline all the new features and changes announced so far.

http://phpa.me/laravel-9-preview


Just finished my first Arduino project: A blinking led.
Next step: Update my LinkedIn profile.
in
Add Skill
Mechatronic Engineer

turnoff.us | Daniel Stori
Shared with permission from the artist

Infamous Fizz Buzz

Oscar Merida

Let's start the year looking at a classic interview question—Fizz Buzz. It's misused to screen applicants and may not be testing for what it thinks it is. Nonetheless, it doesn't hurt to be prepared if you encounter it. We'll look at various approaches to solve it.

Recap

Write a PHP script that prints integers from 1 to some `$maximum`.

1. For multiples of 3, output “Fizz” instead of the number
2. For multiples of 5, print “Buzz” instead of the number
3. For multiples of both 3 and 5, print “FizzBuzz” instead of the number

Why?

In Why Can't Programmers Program[1], Jeff Atwood popularized using Fizz Buzz and coding challenges like it. Hiring programmers is challenging. Education and certifications can get you only so far. Tests like these are designed to assess if a candidate thinks like a “real programmer” in a reasonably short amount of time. But our daily work involves discovery, investigation, and trial-and-error before we know what code we need to write.

Real-world problems are rarely as well defined as Fizz Buzz. In a poor interview, you may not be given more than a couple of minutes to show a solution or even iterate on your first one. It also excludes people who don't test well or don't have esoteric programming knowledge memorized. The idea that “most programmers can't program” may not be true in the first place, as explained by Justin Falcone in *FizzBuzz and Folklore*[2].

Coding challenges are not going away from the interview process. If asked to do one, ask yourself if the organization is a good fit. Why are they still using it, and would you be happy at that job? If you do want that position, it can't hurt to be prepared.

> *For a deep-dive into the issue with screening applicants this way, see Ed Barnards' book The Fizz Buzz Fix: Secrets to Thinking Like an Experienced Software Developer*[3].

The Secret Sauce

The modulus operator is at the heart of any solution. PHP uses `%` as the modulus operator. It's a handy but seldom used operator. Many programmers, particularly novices, may forget it exists. It complements the division operator for integers by giving us the “remainder,” or what's leftover from long division.

Six goes into 32 five times with a remainder of two.

```
echo  32 / 6; // 5.3333333333333
echo  32 % 6; // 2
```

The Loop Solution

A straightforward initial solution is to loop through the given range to produce our output.

Listing 1.

```
$maximum = 50;
for ($i = 1; $i <= 50; $i++) {
    if ($i % 5 === 0 && $i % 3 === 0) {
        echo "FizzBuzz";
    } else if ($i % 5 === 0) {
        echo "Buzz";
    } else if ($i % 3 === 0) {
        echo "Fizz";
    } else {
        echo $i;
    }
    echo " ";
}
```

That loop produces the following output:

```
1 2 Fizz 4 Buzz Fizz 7 8 Fizz Buzz 11
Fizz 13 14 FizzBuzz 16 17 Fizz 19 Buzz
Fizz 22 23 Fizz Buzz 26 Fizz 28 29
FizzBuzz 31 32 Fizz 34 Buzz Fizz 37 38
Fizz Buzz 41 Fizz 43 44 FizzBuzz 46 47
Fizz 49 Buzz
```

Another frequent mistake here is in getting the `if` expressions in the wrong order. Make sure you include a case for integers that are multiples of both 3 and 5.

[1] *Why Can't Programmers Program:* https://phpa.me/codinghorror-why-cant

[2] *FizzBuzz and Folklore:* https://phpa.me/medium-fizzbuzz

[3] *Secrets to Thinking Like an Experienced Software Developer:* https://www.phparch.com/books/the-fizz-buzz-fix/

Improving the Loop

One improvement we can make to our solution is to put the `if` logic for what to print out into a function. I'm not a fan of functions that echo values. Doing so is an unintended side effect and assumes you want to display the result to `STDOUT`. Instead, I prefer to return the string and let the calling code decide how to handle it.

Listing 2.

```php
function FizzBuzzResult(int $num) : string {
    if ($num % 15 === 0) {
        return "FizzBuzz";
    }
    if ($num % 5 === 0) {
        return "Buzz";
    }
    if ($num % 3 === 0) {
        return "Fizz";
    }
    return $num;
}

$maximum = 50;
for ($i = 1; $i <= 50; $i++) {
    echo FizzBuzzResult($i) . " ";
}
```

The output is identical in both cases. However, we've taken a big step in maintainability. The loop code isn't mingled with the logic for figuring out what string to output. More so, we've documented our function by using type hints for arguments and our function's return value. Another developer who reads the signature knows to pass an integer value and can expect to get some a string value. I try to avoid long chains of `if-else` statements in my code. Reading code with nested `if`s is challenging. Here, I opted to return a value as soon as possible, so I don't need to chain a bunch of `elseif` clauses. One minor optimization I added was to check if a number is divisible by 15 instead of doing two checks for divisibility by 3 and 5.

No Loop

Can we do this without a loop? Of course! We can use our `FizzBuzzResult()` function and PHP's native array functions for a more, well, functional approach.

```php
$maximum = 50;
$numbers = range(1, $maximum);
$numbers = array_map('FizzBuzzResult', $numbers);
echo join(' ', $numbers);
```

Look, Ma! No loop. I use `range()`[4] to initialize an array with the numbers 1 to 50. Then, `array_map()`[5] does the hard work. It passes every element in our `$numbers` array to `FizzBuzzResult()` and returns an array with the result for each. If you peek at this array before we echo it, it looks like the `var_dump()` below. Keep in mind that arrays are zero-indexed. The first element in our original array was the integer value `1` at index `0`:

Listing 3.

```
php > var_dump($numbers);
array(50) {
  [0]=>
  string(1) "1"
  [1]=>
  string(1) "2"
  [2]=>
  string(4) "Fizz"
  [3]=>
  string(1) "4"
  [4]=>
  string(4) "Buzz"
  [5]=>
  string(4) "Fizz"
  [6]=>
  string(1) "7"
  [7]=>
  string(1) "8"
  [8]=>
  string(4) "Fizz"
  [9]=>
  string(4) "Buzz"
  [10]=>
  string(2) "11"
  [11]=>
  string(4) "Fizz"
  [12]=>
  string(2) "13"
  [13]=>
  string(2) "14"
  [14]=>
  string(8) "FizzBuzz"
  [15]=>
```

Otherwise, the output from `join()`[6] is identical to our earlier solutions. For very large number ranges, this solution uses more memory since we're using arrays. But the functional approach means there are no unintended side effects from each step. Once you're familiar with PHP's specialized array functions, you can use them to operate on arrays without constantly writing loops. This approach would be well suited to doing something else with `$numbers` after it's run through `array_map()`, like if we'd been asked to count the occurrences of `Fizz`, `Buzz`, and `FizzBuzz` in a given range.

[4] *range(): https://php.net/range*

[5] *array_map(): https://php.net/array_map*

[6] *join(): https://php.net/join*

Idiomatic Php

Type hints and functions are all well and good, but what would a solution that leverages PHP's type coercion and built-in operators look like? It might venture close into the realm of too-clever code that you'd hate to maintain. It should look something like this:

```
for ($i = 1; $i <= 50; $i++) {
   $result = ($i % 3 == 0 ? 'Fizz' : '') .
                         ($i % 5 == 0 ? 'Buzz' : '');
   echo ($result ? $result : $i) . ' ';
}
```

Short and … sweet? I use the ternary operator to append either FizzBuzz, Fizz, Buzz, or nothing to `$result`. When I go to output the value, if `$result` is empty PHP displays the initial number. It works, but it takes brainpower to understand what's going on and it needs effort to extend.

Integer Factors

For next month's puzzle, given a positive integer, write a PHP script that finds and outputs the pairs of positive integers that can be multiplied, in any order, to equal that integer.

For example, given the integer 24, your script should output:

```
1,24
2,12
3,8
4,6
```

Bonus: For any positive or negative integer, output the pairs of integers that can be multiplied to equal that integer.

Some Guidelines And Tips

The puzzles can be solved with pure PHP. No frameworks or libraries are required.

- *Each solution is encapsulated in a function or a class, and I'll give sample output to test your solution against.*
- *You're encouraged to make your first attempt at solving the problem without using the web for clues.*
- *Refactoring is encouraged.*
- *I'm not looking for speed, cleverness, or elegance in the solutions. I'm looking for solutions that work.*
- *Go ahead and try many solutions if you like.*
- *PHP's interactive shell (`php -a` at the command line) or 3rd party tools like PsySH[7] can be helpful when working on your solution.*
- *To keep solutions brief, we'll omit the handling of out-of-range inputs or exception checking.*

Oscar Merida has been working with PHP since version 4 was released and is constantly learning new things about it and still remembers installing an early version on Apache. When not coding or writing, he enjoys RPGs, soccer, and drawing. @omerida

[7] PsySH: https://psysh.org

The Fizz Buzz Fix
Secrets to Thinking Like an Experienced Software Developer
by Edward Barnard
a php[architect] anthology
kindle

Background Queues

Chris Tankersley

Web developers, especially PHP developers, have it easy. We write applications that take a web request, do some work, and return some sort of response. When we do our job properly, we return that request as fast as possible. According to HubSpot[1], the first four seconds of a page load are the most important. After that, conversion rates start to drop off rapidly. Older stats show that users would begin to leave at the three-second mark.

While much of this goes into the full page render, nothing happens until we return that base response. These days there is even increased pressure as the web turns increasingly toward delivering APIs. Humans may wait up to three seconds for a page to render, but many applications would never wait more than a few seconds on an API. Our applications need to return right away.

What happens when you need to do a large amount of work, though? If a user uploads a video, they still expect as immediate a response as the website can give. If a page or process is a multi-second, or even a multi-minute or multi-hour process, how do we handle that?

PHP is traditionally a single-threaded, procedural language. Meaning that when a request comes in, the engine starts at the beginning of a script and processes it line-by-line. The engine does not move on to the next line until the current line is finished. For most workloads, like rendering a JSON or HTML response, this is fine.

```
echo 'This line prints first' . PHP_EOL;
sleep(1);
echo 'This line prints after one second' . PHP_EOL;
echo 'Then this line prints last' . PHP_EOL;
```

In other scenarios, this may present a problem. Since the engine processes the scripts line-by-line, this leads to a situation known as "blocking code," which means that the engine can get into situations where long-running code, like a `sleep()` call, will "block" execution of further code until that line has finished. The system is not hung; it is just busy doing one thing at a time.

Queues and Workers

One of my earliest jobs was working for an insurance company. There were a lot of reports that needed to be built and, while we did have a back-office system, sometimes we needed a better way to make those reports available via the website. That often meant running the reports directly and waiting for the output.

When an agent requested a larger report, the website would regularly time out. The issue was just due to the amount of data gathered from various sources, and it took longer than the server would wait. The agents would then e-mail in and say things were broken.

What we decided to do was implement a queueing system. When the agent would request a report, we would create a request for a report and put it into a queue system or a program that stores messages. A second program, called a worker, running on the servers that generated the reports, would take that request and run it. The output of the report would then be saved to a place on disk, and the agent alerted that the report was ready.

As this was one of my first times doing this, I used MySQL as the "queue." I would insert a row to the database to a table named `report_queue`, and the worker program would just constantly "`SELECT * FROM report_queue LIMIT 1`", do the work, then run the query again. The worker itself, shown in Listing 1, was just a small script that ran forever.

Listing 1.

```
$pdo = new \PDO(...);
while (true) {
    foreach ($pdo->query('SELECT * FROM report_queue LIMIT 1') as $row) {
        // Read the message and do the work
        // Remove the message from the queue
        $pdo
            ->prepare('DELETE FROM report_queue WHERE id = :id')
            ->execute(['id' => $row['id']]);
    }
    sleep(30);
}
```

This simple solution worked, but only because we had a single worker. Demand was never high enough for us to ever really queue up more than a few reports at any one time. Agents were happy because the website would instantly confirm their request for the report, and the report would be ready

[1] *HubSpot: https://blog.hubspot.com/marketing/page-load-time-conversion-rates*

shortly after. No more support requests that the site was broken.

There are a few shortcomings with this setup. The first is its inability to run multiple workers. There is no efficient way to know when a queue item has been picked up. You could add a `locked` column to the queue table, but there is always a chance two workers could grab the same message as there is some time between doing the `SELECT` to get the row and the `UPDATE` to flag the message as locked.

The second is stuck messages. If a row gets flagged as locked but the worker crashes, there is no clean way to unlock the row without manual intervention. Automating that a queue item is stuck can be tricky since some external system would have to know when something is stuck versus just taking a long time.

You could add a TTL (Time to Live) on the message and a timestamp for the locking. If the timestamp expires, unlock the row! Except now you have another program that just watches for rows that need to be unlocked. At the end of the day, there are better ways to handle queueing.

Fast forward to a later job. We had multiple reports that needed to be run and reports that depended on other reports. It was not just a simple "request a report" anymore—we kicked off nightly processing and would generate hundreds of reports a night.

The simple database-backed queue I had used before would not cut it. We would need to run dozens of workers at the same time. Thankfully there are queueing systems out there that work very nicely with PHP. One of the simplest is beanstalkd[2].

beanstalkd is a lightweight job queue that handles all the deficiencies of the previous database system. Messages are automatically locked when handed out to workers and unlocked if a worker takes longer than expected. It can flush its messages to the disk to help avoid issues with the queue itself crashing. If there are problems, someone can just telnet into the beanstalkd instance and view or flush messages.

Setting up beanstalkd is simple. It should exist in most package managers[3] (though Redhat and CentOS-based systems will need to enable the EPEL library). Once beanstalkd is started, it is all set to start receiving messages. There is almost nothing to configure. By default, beanstalkd listens on port 11300.

While the protocol is a simple text protocol, I recommend using `pda/pheanstalk`. It is a pure userland implementation of the beanstalkd protocol and provides you with both the ability to add messages to queues and create workers that use those queues. You can install it using composer.

```
$ composer require pda/pheanstalk:^4.0
```

Pheanstalk 4, which I will be using for the sample code, requires the `mbstring` extension to be installed and PHP 7.3 or higher. If you do not have `mbstring` installed or are on an older PHP version, you will have to use Pheanstalk 3. Overall the ideas will be the same only the syntax will be different. I am happy to say it works fine with PHP 8.1.

The way beanstalkd works is by your application submitting a message, or a simple block of text, to a "tube," which is a queue of those messages. A worker process will "reserve" or ask for a message from the queue. beanstalkd will take the oldest message, lock it, and hand it off to the worker. At the end of the job, the worker will request that either the message be deleted or "release" it back to the queue to be processed by someone else (for example, if the job hits an error and cannot be completed).

Adding a message to a queue is fairly simple. We need to create a connection to a beanstalkd instance. We then select a tube to use and submit a message to it. See Listing 2.

Listing 2.

```php
use Pheanstalk\Pheanstalk;

$pheanstalk = Pheanstalk::create('127.0.0.1');

// Queue a Job
$tube = $pheanstalk->useTube('reports');
$message = ['type' => 'analytics_summary', 'user' => 5];
$tube->put(json_encode($message));
```

Here we connect to a beanstalkd instance on the local machine. You can connect to a remote machine, but be careful to encrypt the transfer using something like an SSH connection. beanstalkd is a plain-text protocol, so you need to be very careful about inter-machine communication.

Once we have the connection created, we select a tube with `->useTube('reports')`. beanstalkd will automatically create tubes when requested, so we do not have to initialize or configure the tube beforehand. The first time we request a tube, it will be created. If you want to see a list of existing tubes, you can use `->listTubes()`.

Once we have the tube, we can add a message to the queue with `->put()`. In this case, we are using JSON, but any text format can work. You can submit XML, YAML, a normal string of text, or anything text-based you want. All that matters is that the worker understands how to decode the message. beanstalkd does not care what the message was nor enforces any sort of structure.

This code can be a part of a normal web application backend. Putting a message into a queue is a quick process and will not add much time to the actual execution of the web request. We leave the longer-running work to the worker, which will be a second script running on the server.

Consuming messages is just as easy as shown in Listing 3. We connect to beanstalkd, watch for a message to come across a tube, and wait to reserve a message.

[2] *beanstalkd: https://beanstalkd.github.io/*

[3] *beanstalkd installation: https://beanstalkd.github.io/download.html*

Here we connect to the local beanstalkd instance just like with the message sender. This time we use ->watch('reports') to use a tube as we will watch that tube for any new messages that come in. Once we are watching a tube, we immediately call ->reserve() to try and get a message to use. The worker will hang on this command until a message is available to process. If there are no messages, the worker will hang until there is one.

Once a message is reserved, we can get the message. Since we expect messages in this tube to be a JSON string, we decode it. We do whatever is needed to run the report and then use ->delete($job) to delete the job from the queue. If we encounter any errors, we instead use ->release($job) to release the job back to the queue so another worker can try again in the future.

Keeping the worker running

This logic is wrapped in a while (true) {} loop, and once our worker is finished with a job, it will attempt to grab another message. Doing so turns our script into a daemon or long-running process. It will try to reserve messages until it crashes (if it ever does) or is otherwise interrupted.

Using a while() loop is easy but not the best when it comes to memory usage. PHP's garbage collector, an internal process the engine uses to clean up memory from unused variables and data, has gotten vastly better since PHP 5.3. Still, depending on how your worker script is structured, there may be situations where the garbage collector cannot clean up everything you expect, and this can cause memory usage to rise over time.

The easiest way to stop this is to allow your worker to run on only so many tasks before it dies. For example, we may swap out the while() loop for a for() loop and only process 100 jobs before we stop (Listing 4).

What can we do to keep the process running? One option, if you have the pcntl extension[4] installed, is to register a shutdown function that restarts the script. We can call-pcntl_exec()` with the current process information to spawn a second process right before our script ends. See Listing 5.

$_SERVER['_'] contains the program that was executed, which is normally the path to PHP itself. $argv contains all of the arguments supplied to the command, like the script name and any additional arguments. Since we are restarting the job, we can just re-use all the supplied information to restart the process.

The other alternative is to use an external program like supervisord[5] to restart the process when it finishes. supervisord is a process monitor and makes sure that certain processes are always running. While we want our worker to always be running, we also need it to restart. Using a for() loop to stop after so many executions and letting the script end will force supervisord to restart the script, much like our pcntl option above.

Listing 3.

```php
use Pheanstalk\Pheanstalk;

$pheanstalk = Pheanstalk::create('127.0.0.1');
$pheanstalk->watch('reports');
while (true) {
    $job = $pheanstalk->reserve();

    try {
        $jobPayload = json_decode($job->getData(), true);
        // Do some work
        $pheanstalk->delete($job);
    } catch(\Exception $e) {
        $pheanstalk->release($job);
    }
}
```

Listing 4.

```php
use Pheanstalk\Pheanstalk;

$pheanstalk = Pheanstalk::create('127.0.0.1');
$pheanstalk->watch('reports');
for ($i = 0; $i < 100; $i++) {
    $job = $pheanstalk->reserve();
    try {
        $jobPayload = json_decode($job->getData(), true);
        // Do some work
        $pheanstalk->delete($job);
    } catch(\Exception $e) {
        $pheanstalk->release($job);
    }
}
```

Listing 5.

```php
register_shutdown_function(function () {
    global $argv;
    echo 'Restarting the worker' . PHP_EOL;
    pcntl_exec($_SERVER['_'], $argv);
});

use Pheanstalk\Pheanstalk;

$pheanstalk = Pheanstalk::create('127.0.0.1');
$pheanstalk->watch('reports');
for ($i = 0; $i < 100; $i++) {
    $job = $pheanstalk->reserve();
    try {
        $jobPayload = json_decode($job->getData(), true);
        // Do some work
        $pheanstalk->delete($job);
    } catch(\Exception $e) {
        $pheanstalk->release($job);
    }
}
```

[4] pcntl extension: https://phpa.me/pcntl

[5] supervisord: http://supervisord.org

I use the `pcntl` option most of the time. Using `pcntl` requires an extension not normally installed on most hosts, but using supervisord requires the ability to install your own apps. Both options require some manual setup, so choose whichever works for you.

Why Queues over Async/Multi-threading?

I would argue the traditional multi-threaded paradigm would be better suited for background tasks compared to what is now called "async" programming. I would still recommend a queue-based system overall, though. You get the benefits of multi-threaded scaling without the headaches of writing a multi-threaded application.

The main draw to async programming is that it avoids blocking processes and speeds up execution. In theory, using async programming will allow our report to process in the background while other things continue to run, like server web requests. In the real world, I find that is not what happens.

Async programming does not resolve the blocking issue fully, no matter the language. It is very easy to write blocking code accidentally, and the additional effort it takes to write correct async code for very long tasks can be quite involved. Breaking down a procedural process into small async blocks can require refactoring quite a bit of code. That time could be better spent just writing workers, which will be easier for more developers to understand and maintain.

That is on top of the fact that most PHP code is written to be blocking, and no amount of refactoring will change that. For example, you cannot use any of the normal PDO drivers for databases in async programming since the drivers are blocking. You have to use special database drivers that are async aware, which means your code is now coupled to the async driver. You also lose out on libraries like Doctrine, which, while they technically would work fine, the drivers they use (i.e., the PDO ones) are all blocking.

Multi-threaded applications can get around the blocking issue to an extent. One thread will only block itself, so any other spawned threads will continue to work normally. The issue with multi-threading applications is that they can be harder to build and maintain, and there is a lot of boilerplate code just to spawn, watch, and make sure child threads are working.

Writing a general worker usually requires very little change to how most PHP developers think about programming. Workers take a job off the queue, work on it procedurally, and then do it all over again in a simple loop. Scaling workers is as simple as just starting more workers.

Queue up that work

This article should give you a good base to start adding background workers to your application. The traditional worker-queue setup is robust enough for many workloads. There are many ways to scale, like more feature-rich queues or expanding your workers to be containerized or functions.

So take a look and see if adding a queue system to your app can help speed things up or make things better for your users.

Related Reading

- *Boosting User Perceived Performance with Laravel Horizon* by Scott Keck-Warren, August 2021. http://phpa.me/boost-performance
- *Education Station: Producer-Consumer Programming* by Edward Barnard, September 2018. http://phpa.me/education-sept-18

Chris Tankersley is a husband, father, author, speaker, podcast host, and PHP developer. Chris has worked with many different frameworks and languages throughout his twelve years of programming but spends most of his day working in PHP and Python. He is the author of Docker for Developers and works with companies and developers for integrating containers into their workflows. @dragonmantank

Experts or Out-of-touch?

Beth Tucker Long

After talking to someone about ideas for new security education, I popped over to check out the latest OWASP Top Ten list. A quote on their homepage stood out to me:

> *This category represents the scenario where the security community members are telling us this is important, even though it's not illustrated in the data at this time. https://owasp.org/www-project-top-ten/*[1]

The experts in their community were telling them that a specific issue was critical and widespread enough to warrant a place in the top ten, but the data they collected from codebases and users didn't reflect this at all. Is this because the issue is too up-and-coming to be reflected in the current boots-on-the-ground numbers, but we need to act now because it will soon be a huge issue? Or is this a situation where the experts work on a level so different from the standard developer that the security risk is only applicable to them and not in everyday circumstances?

This difference got me thinking about how PHP has evolved over the last ten years. I used to closely follow PHP Internals and even got involved in a few of the discussions. I felt like it was important for me to be involved as much as I could. After all, these decisions were affecting my livelihood and my community. After about a year, I started feeling like I didn't have much to contribute to the discussion. The people making the decisions worked in such a drastically different environment than me, and neither of us could truly relate to the other's point of view. Not to mention, my point of view was not the exciting, academic, cutting edge side of things, so that didn't help either.

I stopped making comments and just watched the discussions—keeping informed while lurking. Slowly, though, I faded further and further from the discussions until I found I was months behind on reading the posts and just unsubscribed. I felt a bit bad about giving up on being involved in internals, but I didn't have anything to say about closures or scalar type hints or inheritance by anonymous classes. Honestly, most of the projects that I was working on were companies just starting to upgrade to 5.4. PHP 5.6 was still out of reach for many of my customers when 7 was being prepped for release. Many of my projects were short-term or light enough usage that a full-blown, properly abstracted system was unnecessary and would create a painful complexity without providing any needed benefits.

The thing I most needed the core internals team to do was to avoid any breaking changes so I could get my clients to upgrade. Needless to say, that was not their top priority, and rightfully so. I needed consistency to decrease costs for my customers. The core internals team needed to modernize the language and add features and constructs to match what other popular languages offer. I was concerned about ease of debugging while they were concerned about ensuring that PHP enforced strict standards to remove ambiguity in coding and encourage best practices.

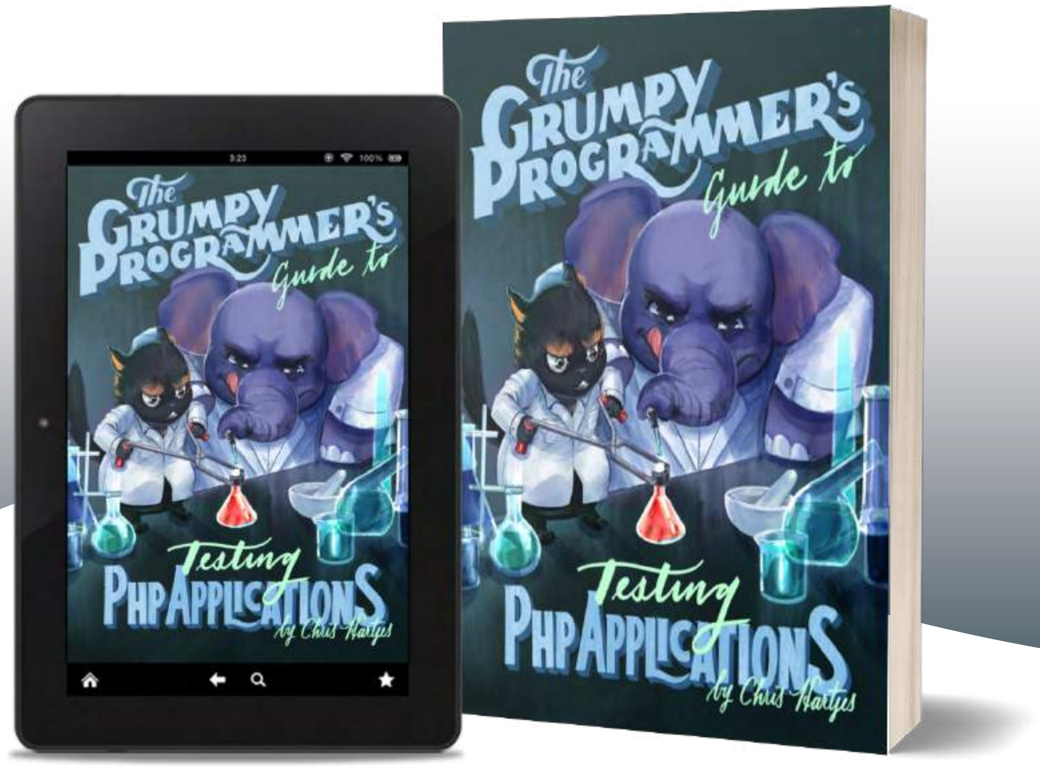
Who was wrong? Who was out of touch? Was I too entrenched in old code and projects on a tight budget to realize the importance of these features which I couldn't imagine ever using? Were the core devs too focused on academic theory and too isolated within huge corporate infrastructures to understand what devs like me needed?

The changes made to PHP from version 7 and beyond have skyrocketed the language into maturity, sped things up considerably, and helped PHP gain some of the professional respect it truly deserves. These are all great things. While they were streamlining things and enforcing best practices, they made things more difficult for developers like me who work with small businesses, ancient codebases, and hosting companies unwilling to upgrade. In making things more advanced to appeal to senior programmers and large organizations, they upped the difficulty for anyone trying to learn PHP or debug unfamiliar code.

There is always give and take, a bit of pain with each step forward. Change is hard. However, the beauty of our community is that we are so diverse. We have a strong group with people who push us forward and other people that keep us grounded. Thanks to this, PHP supports a vast range of organization sizes and project complexities. From small shops that just need a contact form to companies supplying content and services to millions of users, PHP is out there solving problems of all shapes and sizes. When we encounter differences, remember: we are experts in our own experiences, and because those experiences are so different, there's a lot we can learn from each other.

Beth Tucker Long is a developer and owner at Treeline Design, a web development company, and runs Exploricon, a gaming convention, along with her husband, Chris. She leads the Madison Web Design & Development and Full Stack Madison user groups. You can find her on her blog (http://www.alittleofboth.com) or on Twitter @e3BethT

[1] *https://owasp.org/www-project-top-ten/: https://owasp.org/www-project-top-ten/*


The Grumpy Programmer's Guide to
Testing
PHP Applications
by Chris Hartjes

ATMOSPHERE

Build the future of TV streaming with Atmosphere.

Help us become the global leader in entertainment for business with our modern tech stack.

Laravel 8 | PHP 8 | Vue.js | SQL | tvOS and iOS

Revolutionizing the TV Experience for Businesses

Employee-First Culture | Creative Office Space | Open Vacation Policy | Full Benefits | Daily Lunches | Company Outings

See our job openings at
atmosphere.tv/careers